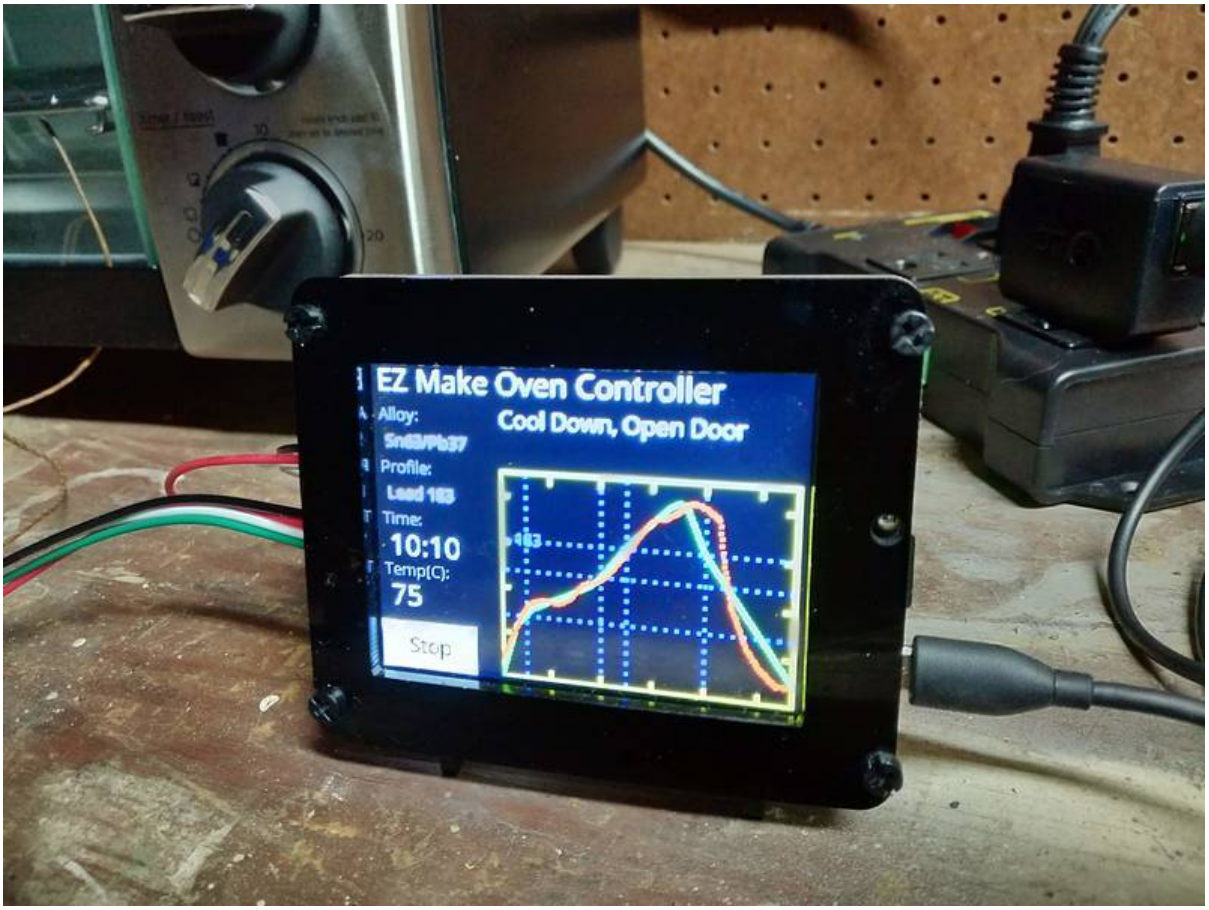
EZ Make Oven

Created by Dan Cogliano



https://learn.adafruit.com/ez-make-oven

Last updated on 2024-06-03 02:56:32 PM EDT

Table of Contents

Overview

Soldering through hole components onto printed circuit boards (PCBs) is a very useful skill to have and needed for many DIY projects. However, there are some components that do not have through hole versions and are only available as surface mount versions. Or, a project may need a smaller footprint using smaller components. In these cases, you will need to put aside your solder iron and use a reflow oven.

A reflow oven is used for soldering of surface mount electronic components onto PCBs. In the past, reflow ovens were large, expensive and limited to commercial units. With the popularity of the DIY movement, reflow ovens have come down in size and price to make them affordable to hobbyists. Some of these are available in kit form. Unfortunately, these kits require modifying a toaster oven's high voltage wiring and can take time to assemble.

The EZ Make Oven is just that: it's an easy to make reflow oven that requires no high voltage wiring and does not require any disassembly of the oven. The reflow oven can be put together in just a few hours with just some minimal soldering required. Other features of the EZ Make Oven include:

- Graphical display of solder profile graph
- Display of current stage and overlay of oven temperature with solder profile
- Beep notifications for each reflow stage
- Beep notification when reflow is complete and oven door can be opened.

Since the EZ Make Oven is using CircuitPython, these features are also available:

- **Install new code easily with drag and drop updates**
- **JSON based solder profile files**

- **New solder profiles can be created using a text editor**
- **Drag and drop new solder profiles**
- **On-the-fly code customization (no compiling needed)**

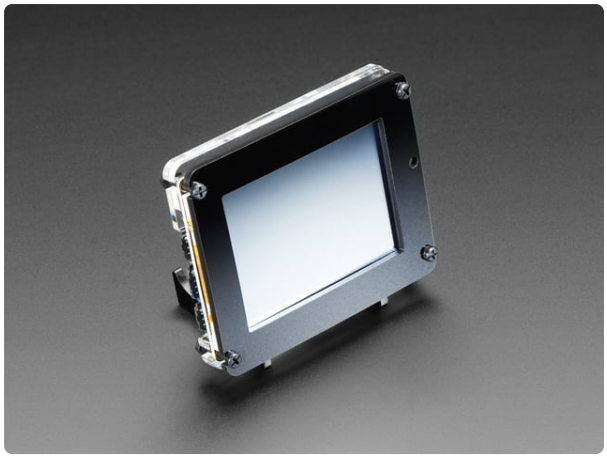
Parts

With the exception of the toaster oven itself, all the parts for this project are available at the Adafruit store. The PyPortal is used as the controller for the EZ Make Oven. It is an excellent choice for this project, as it provides a nice display with touch screen, ports for I2C and digital pins, and sound for notifications. The optional desktop stand provides a nice professional look for the PyPortal.

This project uses the American standard power outlet and 120 volts.

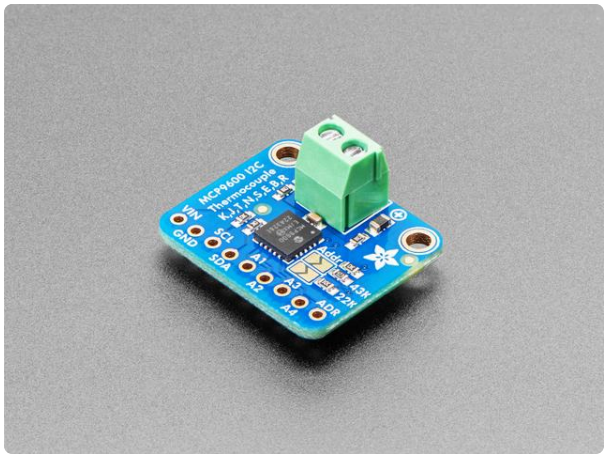
Never Ever Ever leave a Toaster Oven unattended, especially with a project like this! Burnt PCBs smell REALLY BAD and contain nasty stuff, so you'll want to be around to quickly shut it off if something goes awry!

Adafruit PyPortal - CircuitPython Powered Internet Display
PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116



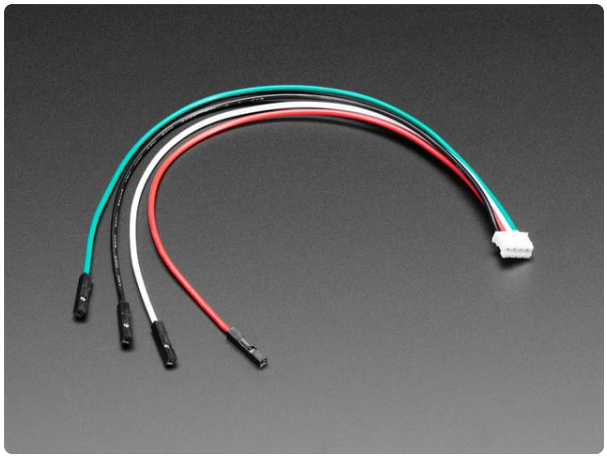
[Adafruit PyPortal Desktop Stand Enclosure Kit]
PyPortal is our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Create little pocket...
https://www.adafruit.com/product/4146

[Controllable Four Outlet Power Relay Module version 2]
Say goodbye to hazardous high voltage wiring and create the Internet of Things with safe, reliable power control....
https://www.adafruit.com/product/2935



[Adafruit MCP9600 I2C Thermocouple Amplifier]
Thermocouples are very sensitive, requiring a good amplifier with a cold-compensation reference. The Adafruit MCP9600 does all that for you, and can be easily...
https://www.adafruit.com/product/4101

[Thermocouple Type-K Glass Braid Insulated]
Thermocouples are best used for measuring temperatures that can go above 100 °C. This is a bare wires bead-probe which can measure air or surface temperatures. Most inexpensive...
https://www.adafruit.com/product/270



[JST PH 2mm 4-Pin to Female Socket Cable - I2C STEMMA Cable]
This cable will let you turn a JST PH 4-pin cable port into 4 individual wires with high-quality 0.1" female sockets on the end. We're carrying these to match up with any of...
https://www.adafruit.com/product/3950

[STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm]
This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" male header plugs on the end. We're carrying these to match up with our...
https://www.adafruit.com/product/3893

[High Temperature Polyimide Tape - 1cm wide x 33 meter roll]
Polyimide Tape (sometimes referred to by the brand name Kapton Tape) is an interesting addition to your toolbox! Polyimide Tape remains stable across...
https://www.adafruit.com/product/3057

The Toaster Oven

Danger: Do not use your reflow toaster oven for cooking food! Solder used in reflow ovens include lead and other dangerous metals and must not be used in the same oven for cooking food.

The EZ Make Oven uses an unmodified toaster oven. This means you don't need to take it apart and void the warranty like some other reflow oven projects. This can be purchased online or at your nearest small appliance store.

The toaster oven needed for this project is a simple, manual toaster oven, since it is controlled at the power outlet, not by the oven's front panel controls. These are fairly economical in price since no fancy electronics are needed. It should be small (4 slice capacity), at least 1100 watts and a maximum temperature of 450F / 230C or better.

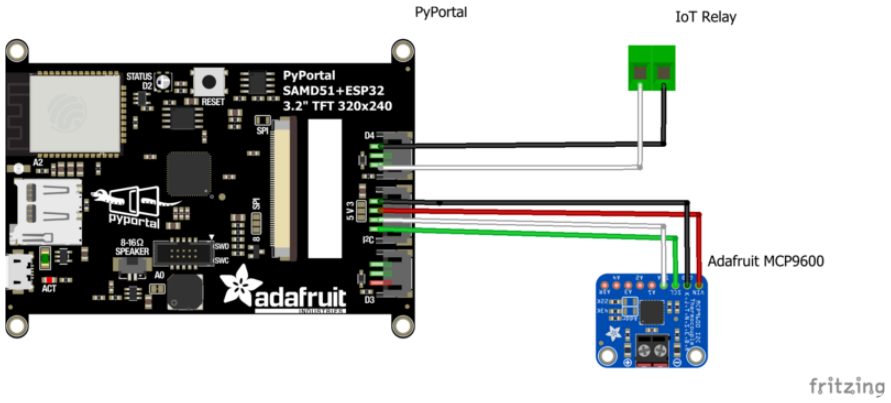
You probably should not choose a convection toaster oven if given the choice. Although they may create a more even air temperature in the oven, they are typically much larger than what is needed for a reflow oven. Also, a convection oven fan could, in theory, create a wind force that moves small, light-weight components off their pads on the PCB. The exception to this is a toaster oven marketed as using "natural convection", which is what this project is using. This type of oven does not use a fan, and the only visible difference between this and a regular toaster oven is the use of rounded corners in the interior. Presumably this enhances the air flow to make the temperature more even throughout the oven.

There are many toaster ovens to choose from that fit these requirements. For this project, we used the Black & Decker Natural Convection Toaster Oven, model TO1755SB, which can fit 4 slices of bread (as in toast, not breadboards :-)).

Before using your toaster oven, you will need to set the oven temperature to it's maximum setting. Each toaster oven may have different ways to accomplish this. Some may have a maximum temperature, some may call it "toast" mode. Make sure the toast mode is not using a timer. If it does, check to see if you can override it with a

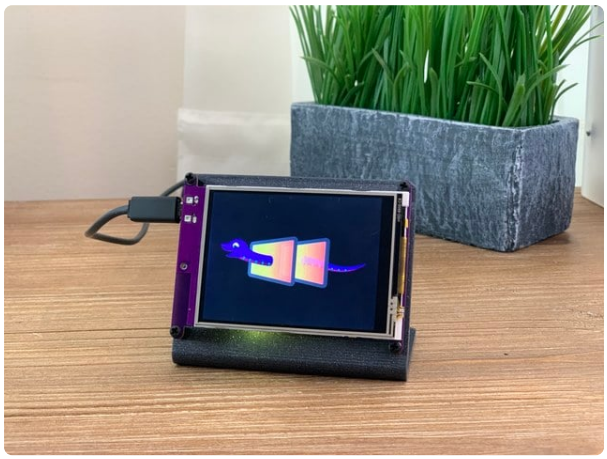
"stay on" timer option. Otherwise, you will need to set the timer to its maximum and reset it each time you use the oven. Also, do not use the "broil" toaster option. This will only use the top heating element, and we want both top and bottom heating elements in use for our reflow oven.

Putting It All Together



In addition to the toaster oven, there are three main components that comprise the EZ Make Oven:

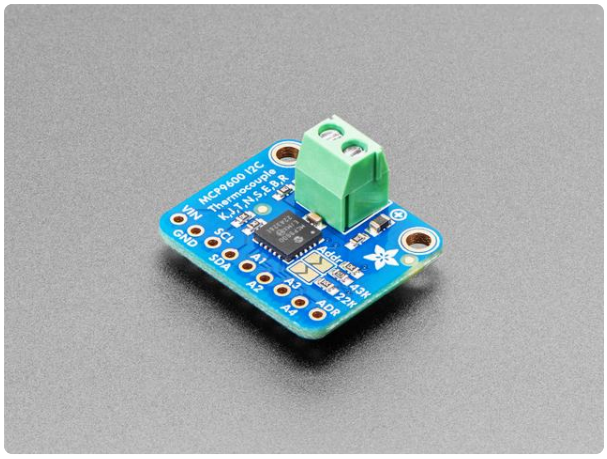
- **Adafruit PyPortal** - This is the main controller for the EZ Make Oven. It makes an excellent controller for this project since it features a nice graphics touch screen and a speaker to generate beep notifications. It also includes the needed ports to connect the next two items.



Adafruit PyPortal - CircuitPython Powered Internet Display

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116

- **Adafruit MCP9600 Thermocouple Amplifier and "K" Type Thermocouple** - The MCP9600 connects to the PyPortal via the I2C port located on the PyPortal. The EZ Make Oven code uses the MCP9600 to monitor the temperature inside the oven. The "K" type thermocouple connects to the MCP9600 and the end is placed inside the oven in the center near the PCB.



Adafruit MCP9600 I2C Thermocouple Amplifier
Thermocouples are very sensitive, requiring a good amplifier with a cold-compensation reference. The Adafruit MCP9600 does all that for you, and can be easily...
https://www.adafruit.com/product/4101

- **Controllable Four Outlet Power Relay Module** - This appears to be a normal power strip but it can actually control the toaster oven temperature using the PyPortal to turn the power on and off as needed to match the solder temperature profile. It uses 1 of the 2 digital pin ports on the PyPortal.

Controllable Four Outlet Power Relay Module version 2
Say goodbye to hazardous high voltage wiring and create the Internet of Things with safe, reliable power control....
https://www.adafruit.com/product/2935

More about how these components work together is described below.

The Adafruit MCP9600 Thermocouple Amplifier and "K" Type Thermocouple

The MCP9600 is one of the few (if not only) I2C compatible thermocouple amplifier (at this writing at least). With recent CircuitPython support it is a key component to this project.

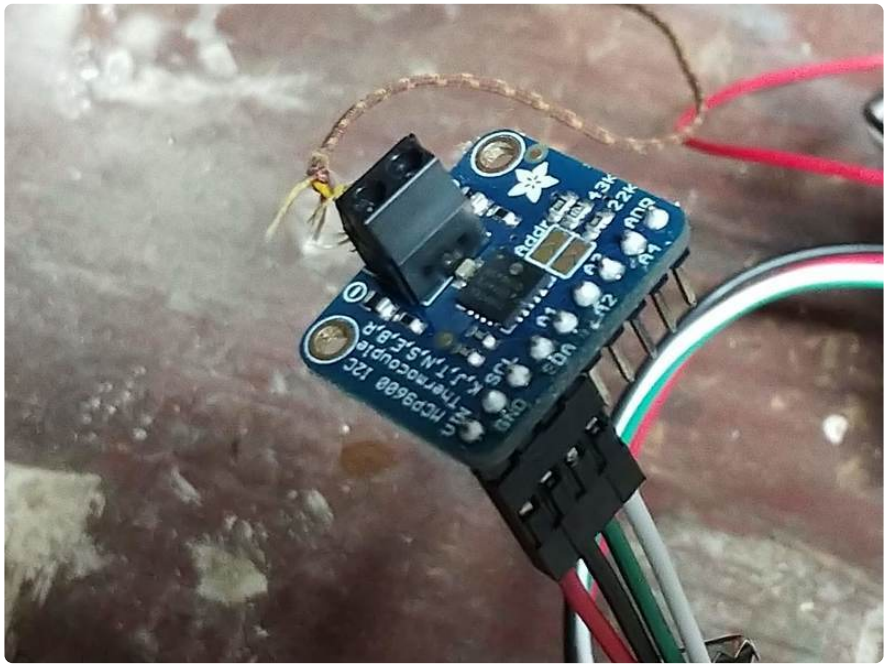
Light soldering is needed to add header pins to the MCP9600 module. There is a learning guide available for it (https://adafru.it/Gfq). After soldering the header pins, the module can be connected to the PyPortal via a breadboard using a male I2C STEMMA cable (http://adafru.it/3955), or it can be connected directly with no breadboard needed using a female I2C STEMMA cable (http://adafru.it/3950). Both

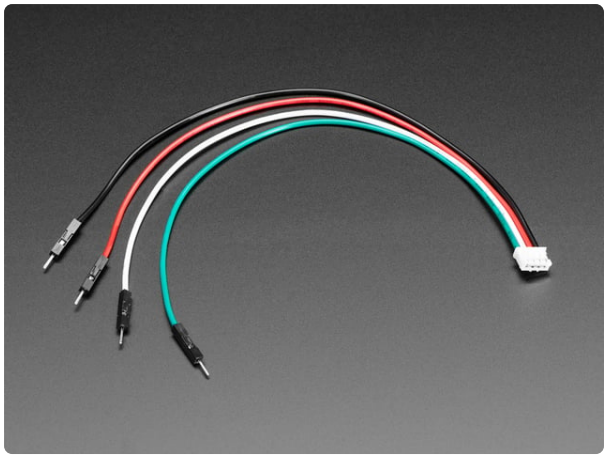
cable types are available in the Adafruit store. The cables should be connected as follows:

- **Black - Ground**
- **Red - VIN**
- **White - SDA**
- **Green - SCL**

A thermocouple needs to be attached to the MCP9600. This is a cable with a temperature sensor at one end and two wires at the other end. The MCP9600 supports several thermocouple types. The default type is "K", which we use for this project. Adafruit sells two types of "K" thermocouples that are 1 meter in length, one with a stainless steel tip (http://adafru.it/3245) and one without (http://adafru.it/270). Either will work fine for this project. The two wire end of the thermocouple is connected to the MCP9600. The wire with the yellow casing is connected to the "+" connector. If you can't tell and wire it wrong, don't worry. You can easily tell if it is wired wrong when you test the oven when it is all connected. If the temperature reading goes down when the oven turns on, then the wires are backwards and need to be swapped. The thermocouple should be placed in the center of the oven at or near where the PCB will be located. You can fish the thermocouple wire through the door hinge to prevent crimping the wire when the door is closed.

You may need to change the power of the I2C port from 5 volts to 3.3 volts. This is done by cutting and soldering a trace in the back of the PyPortal near the I2C port. More in the "using-the-oven" section of this guide.

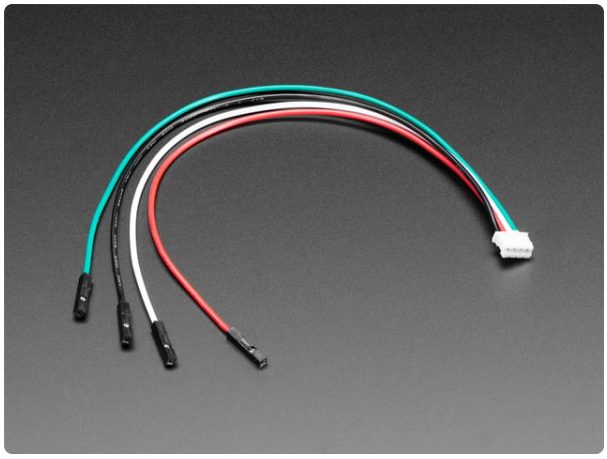




JST PH 2mm 4-Pin to Male Header Cable - I2C STEMMA Cable - 200mm

This cable will let you turn a JST PH 4-pin cable port into 4 individual wires with high-quality 0.1" male header plugs on the end. We're carrying these to match up with any...

https://www.adafruit.com/product/3955



JST PH 2mm 4-Pin to Female Socket Cable - I2C STEMMA Cable

This cable will let you turn a JST PH 4-pin cable port into 4 individual wires with high-quality 0.1" female sockets on the end. We're carrying these to match up with any of...

https://www.adafruit.com/product/3950

Controllable Four Outlet Power Relay Module

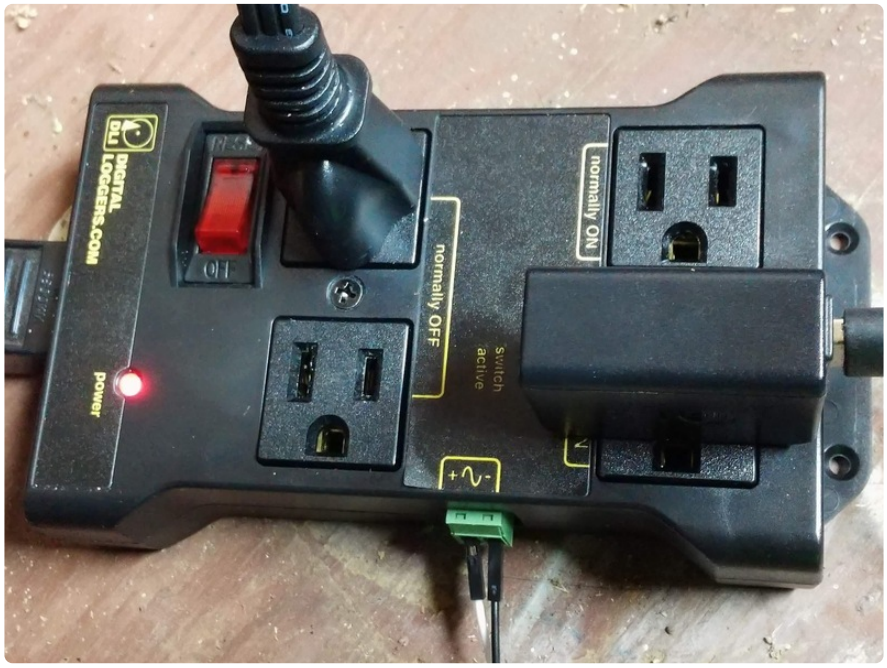
The Controllable Four Outlet Power Relay Module (http://adafru.it/2935) from Data Loggers puts the "EZ" in the EZ Make Oven. It allows the PyPortal to turn the power on and off for an outlet using one of the PyPortal's digital pins. Other reflow oven kits require dismantling the toaster oven and rewiring it to the controller, requiring additional parts and some high voltage wiring. With the EZ Make Oven, you simply plug the toaster oven into the appropriate outlet and then plug the device into a wall outlet. It is that easy! It also has an "always on" outlet where the PyPortal can be plugged into it as well.

Controllable Four Outlet Power Relay Module version 2

Say goodbye to hazardous high voltage wiring and create the Internet of Things with safe, reliable power control....

https://www.adafruit.com/product/2935

There are two digital pin connectors on the PyPortal: **D3** and **D4**. The project code uses **D4** (the top connector) and is connected to the power relay module via a JST 3-pin to male header cable (http://adafru.it/3893). Only two of the wires are needed, the middle red cable stays disconnected. The white cable is connected to the left and the black cable is connected to the right of the digital pin connector of the module.



STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" male header plugs on the end. We're carrying these to match up with our...

https://www.adafruit.com/product/3893

Installing the Code

Install Libraries

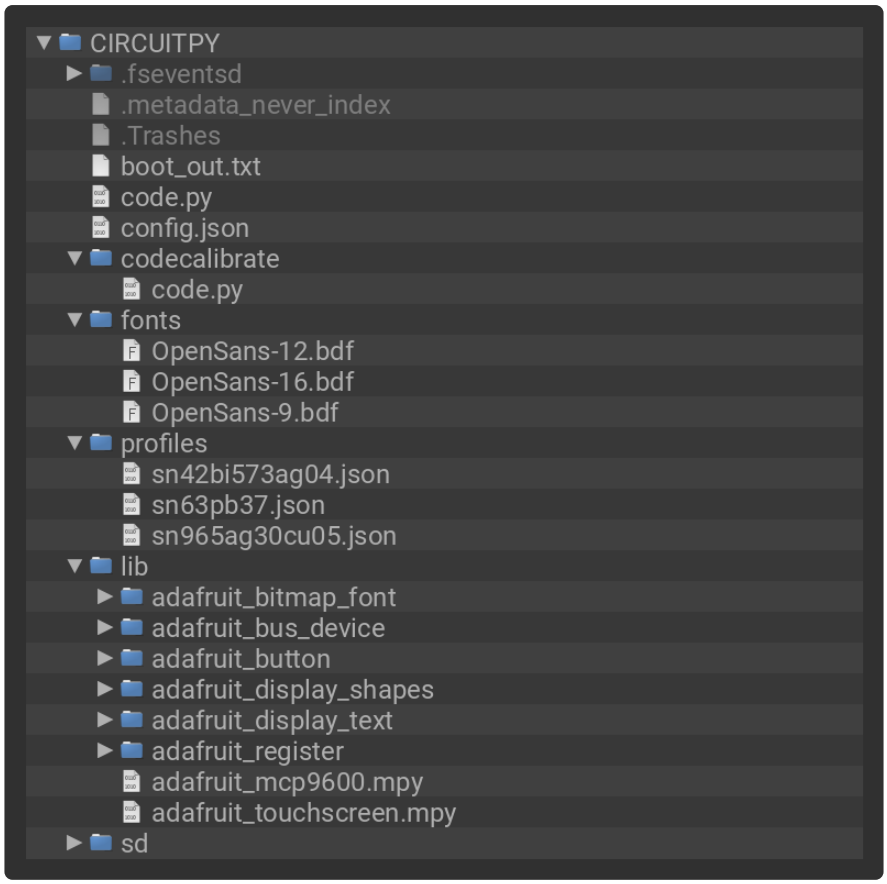
Make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Egk) for your board. This project requires at least CircuitPython version 6.1.0. You will need to install the appropriate libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/ENC) matching your version of CircuitPython.

You can find these files in the CircuitPython Library bundle here (https://adafru.it/ENC).

- **adafruit_bitmap_font/**
- **adafruit_bus_device/**
- **adafruit_button.mpy**
- **adafruit_display_shapes/**
- **adafruit_display_text/**
- **adafruit_esp32spi/**
- **adafruit_io/**
- **adafruit_mcp9600.mpy**
- **adafruit_portalbase/**
- **adafruit_pyportal/**
- **adafruit_register/**
- **adafruit_requests.mpy**
- **adafruit_touchscreen.mpy**
- **neopixel.mpy**

Install Code

Download the project files from the GitHub repo (https://adafru.it/19EN).



In any of the code boxes below, you can get all the files in one step by clicking the "Download Project Bundle" button.

In addition to the **code.py** file, this project also requires several supporting folders and files. These folders are required for this project:

- **fonts -** Contains fonts used for displaying text on the PyPortal
- **profiles -** Contains profiles of solder pastes

These additional files are also needed:

- **codecalibrate.py -** A program used to calibrate the toaster oven with the EZ Make Oven code. Run this when installing new versions or replacing the toaster oven.
- **config.json -** Contains configuration settings, including the values created from the **codecalibrate.py** program.

Calibrating the Oven

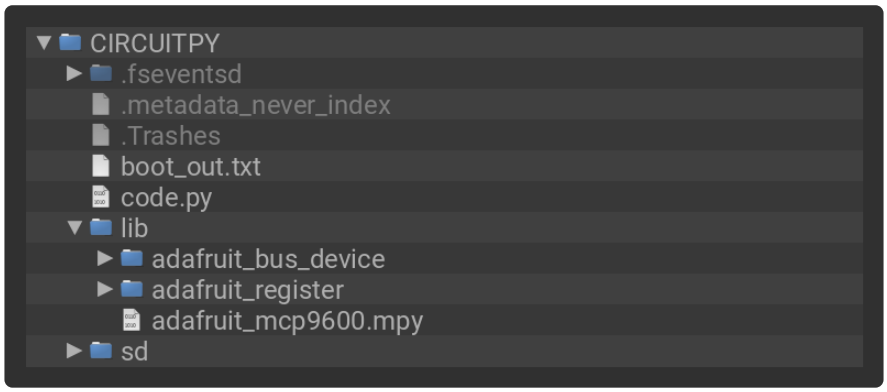
Not all toaster ovens are created equally. The EZ Make Oven takes into account various factors in a toaster oven, including size, wattage, and rate of temperature rise.

This is done using the calibration sketch, which is used gather information about how the oven reacts to temperature changes.

You will need to temporarily rename the **codecalibrate.py** program to **code.py** (and rename **code.py** to another name temporarily) in order to run the calibration program.

This program will turn on the oven, let the oven temperature ramp up to 100 degrees C, turn it off, and measure the time and temperature difference when the temperature levels off. These values will be printed on the display, which will need to be written down and then entered in the **config.json** file.

This calibration program should be run when installing new versions of the EZ Make Oven or when replacing the toaster oven for another toaster oven.




```
# SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import sys
```

```
import board
import busio
import digitalio
from adafruit_mcp9600 import MCP9600

SENSOR_ADDR = 0X67

i2c = busio.I2C(board.SCL, board.SDA,frequency=200000)
try:
    sensor = MCP9600(i2c,SENSOR_ADDR,"K")
except ValueError as e:
    print(e)
    print("Unable to connect to the thermocouple sensor.")
    sys.exit(1)

oven = digitalio.DigitalInOut(board.D4)
oven.direction = digitalio.Direction.OUTPUT

def oven_control(enable=False):
    #board.D4
    oven.value = enable

check_temp = 100
print("This program will determine calibration settings ")
print("for your oven to use with the EZ Make Oven.\n\n")
for i in range(10):
    print("Calibration will start in %d seconds..." % (10-i))
    time.sleep(1)
print("Starting...")
print("Calibrating oven temperature to %d C" % check_temp)
finish = False
oven_control(True)
maxloop=300
counter = 0
while not finish:
    time.sleep(1)
    counter += 1
    current_temp = sensor.temperature
    print("%.02f C" % current_temp)
    if current_temp >= check_temp:
        finish = True
        oven_control(False)
    if counter >= maxloop:
        raise Exception("Oven not working or bad sensor")

print("checking oven lag time and temperature")
finish = False
start_time = time.monotonic()
start_temp = sensor.temperature
last_temp = start_temp

while not finish:
    time.sleep(1)
    current_temp = sensor.temperature
    print(current_temp)
    if current_temp <= last_temp:
        finish = True
    last_temp = current_temp

lag_temp = last_temp - check_temp
lag_time = int(time.monotonic() - start_time)

print("** Calibration Results **")
print("Modify config.json with these values for your oven:")
print("calibrate_temp:", lag_temp)
print("calibrate_seconds:",lag_time)
```

Solder Paste Profiles

The EZ Make Oven contains a few solder paste profiles in the profiles folder. You can view these profiles with a text editor to see if one of them closely matches the solder paste you are using.

If it does, great! Update the **config.json** file with the appropriate solder paste file name (without the "**.json**" extension).

If not, you can easily create a new profile in the profiles folder. Use one of the existing profiles as a guide. It would not hurt to add a few additional data points to smooth out the curve, as the EZ Make Oven uses straight lines to connect the profile points.

Here are some of the items in the profile file explained:

- `temp_range`: temperature range of the profile, used to define the minimum and maximum y axis of the graph
- `time_range`: time range of the profile, used to define the minimum and maximum x axis of the graph
- `stages`: start points for the stages preheat, soak, reflow and cool. The format of the point is [x, y], where x is the time value and y is the temperature value of the starting point of the stage
- `profile`: data points that make up the solder profile. The format of each point is [x, y], where x is the time value and y is the temperature value value of a point on the solder profile graph

The Code for the EZ Make Oven

The code is available from GitHub. You will need to copy and modify the **config.json** file, which contains the solder profile, the MCP9600 I2C address of the sensor board and the calibration settings for the oven.

```
{
    "sensor_address": 103,
    "profile": "sn63pb37",
    "calibrate_temp": 29.5625,
    "calibrate_seconds": 37
}
```

The EZ Make Oven code can be downloaded from the link below or copied and paste to the file **code.py** on the PyPortal **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2019 Dan Cogliano for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import json
import array
import math
import gc
import board
import busio
import audioio
import audiocore
import displayio
import digitalio
import os
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import bitmap_label as label
from adafruit_display_shapes.circle import Circle
from adafruit_button import Button
import adafruit_touchscreen
from adafruit_mcp9600 import MCP9600

TITLE = "EZ Make Oven Controller"
VERSION = "1.3.3"

print(TITLE, "version ", VERSION)
time.sleep(2)

PROFILE_SIZE = 2  # plot thickness
GRID_SIZE = 2
GRID_STYLE = 3
TEMP_SIZE = 2
AXIS_SIZE = 2

BLACK = 0x0
BLUE = 0x2020FF
GREEN = 0x00FF55
RED = 0xFF0000
YELLOW = 0xFFFF00

WIDTH = board.DISPLAY.width
HEIGHT = board.DISPLAY.height

palette = displayio.Palette(5)
palette[0] = BLACK
palette[1] = GREEN
palette[2] = BLUE
palette[3] = RED
palette[4] = YELLOW

palette.make_transparent(0)

BACKGROUND_COLOR = 0
PROFILE_COLOR = 1
GRID_COLOR = 2
TEMP_COLOR = 3
AXIS_COLOR = 2

GXSTART = 100
GYSTART = 80
GWIDTH = WIDTH - GXSTART
GHEIGHT = HEIGHT - GYSTART

ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
```

```
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(WIDTH, HEIGHT),
)


class Beep(object):
    def __init__(self):
        self.duration = 0
        self.start = 0
        tone_volume = 1  # volume is from 0.0 to 1.0
        frequency = 440  # Set this to the Hz of the tone you want to generate.
        length = 4000 // frequency
        sine_wave = array.array("H", [0] * length)
        for i in range(length):
            sine_wave[i] = int(
                (1 + math.sin(math.pi * 2 * i / length)) * tone_volume * (2 ** 15 -
1)
            )
        self.sine_wave_sample = audiocore.RawSample(sine_wave)

        self._speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
        self._speaker_enable.switch_to_output(False)

        if hasattr(board, "AUDIO_OUT"):
            self.audio = audioio.AudioOut(board.AUDIO_OUT)
        elif hasattr(board, "SPEAKER"):
            self.audio = audioio.AudioOut(board.SPEAKER)
        else:
            raise AttributeError("Board does not have a builtin speaker!")

    # pylint: disable=protected-access
    def play(self, duration=0.1):
        if not self._speaker_enable.value:
            self._speaker_enable.value = True
            self.audio.play(self.sine_wave_sample, loop=True)
            self.start = time.monotonic()
            self.duration = duration
            if duration <= 0.5:
                # for beeps less than .5 sec, sleep here,
                # otherwise, use refresh() in loop to turn off long beep
                time.sleep(duration)
                self.stop()

    def stop(self):
        if self._speaker_enable.value:
            self.duration = 0
            self.audio.stop()
            self._speaker_enable.value = False

    def refresh(self):
        if time.monotonic() - self.start >= self.duration:
            self.stop()


class ReflowOvenControl(object):
    global message, timediff, sgraph, timer_data

    states = ("wait", "ready", "start", "preheat", "soak", "reflow", "cool")

    def __init__(self, pin):
        self.oven = digitalio.DigitalInOut(pin)
        self.oven.direction = digitalio.Direction.OUTPUT
        with open("/config.json", mode="r") as fpr:
            self.config = json.load(fpr)
            fpr.close()
        self.sensor_status = False
        with open("/profiles/" + self.config["profile"] + ".json", mode="r") as fpr:
            self.sprofile = json.load(fpr)
```

```
            fpr.close()
        i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)
        try:
            self.sensor = MCP9600(i2c, self.config["sensor_address"], "K")
            self.ontemp = self.sensor.temperature
            self.offtemp = self.ontemp
            self.sensor_status = True
        except ValueError:
            print("temperature sensor not available")
        self.control = False
        self.reset()
        self.beep = Beep()
        self.set_state("ready")
        if self.sensor_status:
            if self.sensor.temperature >= 50:
                self.last_state = "wait"
                self.set_state("wait")

    def reset(self):
        self.ontime = 0
        self.offtime = 0
        self.enable(False)
        self.reflow_start = 0

    def get_profile_temp(self, seconds):
        x1 = self.sprofile["profile"][0][0]
        y1 = self.sprofile["profile"][0][1]
        for point in self.sprofile["profile"]:
            x2 = point[0]
            y2 = point[1]
            if x1 <= seconds < x2:
                temp = y1 + (y2 - y1) * (seconds - x1) // (x2 - x1)
                return temp
            x1 = x2
            y1 = y2
        return 0

    def set_state(self, state):
        self.state = state
        self.check_state()
        self.last_state = state

    # pylint: disable=too-many-branches, too-many-statements
    def check_state(self):
        try:
            temp = self.sensor.temperature
        except AttributeError:
            temp = 32  # sensor not available, use 32 for testing
            self.sensor_status = False
            # message.text = "Temperature sensor missing"
        self.beep.refresh()
        if self.state == "wait":
            self.enable(False)
            if self.state != self.last_state:
                # change in status, time for a beep!
                self.beep.play(0.1)
            if temp < 35:
                self.set_state("ready")
                oven.reset()
                draw_profile(sgraph, oven.sprofile)
                timer_data.text = format_time(0)

        if self.state == "ready":
            self.enable(False)
        if self.state == "start" and temp >= 50:
            self.set_state("preheat")
        if self.state == "start":
            message.text = "Starting"
            self.enable(True)
```

```
        if self.state == "preheat" and temp >= self.sprofile["stages"]["soak"][1]:
            self.set_state("soak")
        if self.state == "preheat":
            message.text = "Preheat"
        if self.state == "soak" and temp >= self.sprofile["stages"]["reflow"][1]:
            self.set_state("reflow")
        if self.state == "soak":
            message.text = "Soak"
        if (
            self.state == "reflow"
            and temp >= self.sprofile["stages"]["cool"][1]
            and self.reflow_start > 0
            and (
                time.monotonic() - self.reflow_start
                >= self.sprofile["stages"]["cool"][0]
                - self.sprofile["stages"]["reflow"][0]
            )
        ):
            self.set_state("cool")
            self.beep.play(5)
        if self.state == "reflow":
            message.text = "Reflow"
            if self.last_state != "reflow":
                self.reflow_start = time.monotonic()
        if self.state == "cool":
            self.enable(False)
            message.text = "Cool Down, Open Door"

        if self.state in ("start", "preheat", "soak", "reflow"):
            if self.state != self.last_state:
                # change in status, time for a beep!
                self.beep.play(0.1)
            # oven temp control here
            # check range of calibration to catch any humps in the graph
            checktime = 0
            checktimemax = self.config["calibrate_seconds"]
            checkoven = False
            if not self.control:
                checktimemax = max(
                    0,
                    self.config["calibrate_seconds"]
                    - (time.monotonic() - self.offtime),
                )
            while checktime <= checktimemax:
                check_temp = self.get_profile_temp(int(timediff + checktime))
                if (
                    temp + self.config["calibrate_temp"] * checktime / checktimemax
                    < check_temp
                ):
                    checkoven = True
                    break
                checktime += 5
            if not checkoven:
                # hold oven temperature
                if (
                    self.state in ("start", "preheat", "soak")
                    and self.offtemp > self.sensor.temperature
                ):
                    checkoven = True
            self.enable(checkoven)

    # turn oven on or off
    def enable(self, enable):
        try:
            self.oven.value = enable
            self.control = enable
            if enable:
                self.offtime = 0
                self.ontime = time.monotonic()
```

```
                self.ontemp = self.sensor.temperature
                print("oven on")
            else:
                self.offtime = time.monotonic()
                self.ontime = 0
                self.offtemp = self.sensor.temperature
                print("oven off")
        except AttributeError:
            # bad sensor
            pass


class Graph(object):
    def __init__(self):
        self.xmin = 0
        self.xmax = 720  # graph up to 12 minutes
        self.ymin = 0
        self.ymax = 240
        self.xstart = 0
        self.ystart = 0
        self.width = GWIDTH
        self.height = GHEIGHT

    # pylint: disable=too-many-branches
    def draw_line(self, x1, y1, x2, y2, size=PROFILE_SIZE, color=1, style=1):
        # print("draw_line:", x1, y1, x2, y2)
        # convert graph coords to screen coords
        x1p = self.xstart + self.width * (x1 - self.xmin) // (self.xmax - self.xmin)
        y1p = self.ystart + int(
            self.height * (y1 - self.ymin) / (self.ymax - self.ymin)
        )
        x2p = self.xstart + self.width * (x2 - self.xmin) // (self.xmax - self.xmin)
        y2p = self.ystart + int(
            self.height * (y2 - self.ymin) / (self.ymax - self.ymin)
        )
        # print("screen coords:", x1p, y1p, x2p, y2p)

        if (max(x1p, x2p) - min(x1p, x2p)) > (max(y1p, y2p) - min(y1p, y2p)):
            for xx in range(min(x1p, x2p), max(x1p, x2p)):
                if x2p != x1p:
                    yy = y1p + (y2p - y1p) * (xx - x1p) // (x2p - x1p)
                    if style == 2:
                        if xx % 2 == 0:
                            self.draw_point(xx, yy, size, color)
                    elif style == 3:
                        if xx % 8 == 0:
                            self.draw_point(xx, yy, size, color)
                    elif style == 4:
                        if xx % 12 == 0:
                            self.draw_point(xx, yy, size, color)
                    else:
                        self.draw_point(xx, yy, size, color)
        else:
            for yy in range(min(y1p, y2p), max(y1p, y2p)):
                if y2p != y1p:
                    xx = x1p + (x2p - x1p) * (yy - y1p) // (y2p - y1p)
                    if style == 2:
                        if yy % 2 == 0:
                            self.draw_point(xx, yy, size, color)
                    elif style == 3:
                        if yy % 8 == 0:
                            self.draw_point(xx, yy, size, color)
                    elif style == 4:
                        if yy % 12 == 0:
                            self.draw_point(xx, yy, size, color)
                    else:
                        self.draw_point(xx, yy, size, color)

    def draw_graph_point(self, x, y, size=PROFILE_SIZE, color=1):
```

```
        """ draw point using graph coordinates """

        # wrap around graph point when x goes out of bounds
        x = (x - self.xmin) % (self.xmax - self.xmin) + self.xmin
        xx = self.xstart + self.width * (x - self.xmin) // (self.xmax - self.xmin)
        yy = self.ystart + int(self.height * (y - self.ymin) / (self.ymax -
self.ymin))
        print("graph point:", x, y, xx, yy)
        self.draw_point(xx, max(0 + size, yy), size, color)

    def draw_point(self, x, y, size=PROFILE_SIZE, color=1):
        """Draw data point on to the plot bitmap at (x,y)."""
        if y is None:
            return
        offset = size // 2
        for xx in range(x - offset, x + offset + 1):
            if xx in range(self.xstart, self.xstart + self.width):
                for yy in range(y - offset, y + offset + 1):
                    if yy in range(self.ystart, self.ystart + self.height):
                        try:
                            yy = GHEIGHT - yy
                            plot[xx, yy] = color
                        except IndexError:
                            pass


def draw_profile(graph, profile):
    global label_reflow

    """Update the display with current info."""
    for i in range(GWIDTH * GHEIGHT):
        plot[i] = 0

    # draw stage lines
    # preheat
    graph.draw_line(
        profile["stages"]["preheat"][0],
        profile["temp_range"][0],
        profile["stages"]["preheat"][0],
        profile["temp_range"][1] * 1.1,
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )
    graph.draw_line(
        profile["time_range"][0],
        profile["stages"]["preheat"][1],
        profile["time_range"][1],
        profile["stages"]["preheat"][1],
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )
    # soak
    graph.draw_line(
        profile["stages"]["soak"][0],
        profile["temp_range"][0],
        profile["stages"]["soak"][0],
        profile["temp_range"][1] * 1.1,
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )
    graph.draw_line(
        profile["time_range"][0],
        profile["stages"]["soak"][1],
        profile["time_range"][1],
        profile["stages"]["soak"][1],
        GRID_SIZE,
```

```
        GRID_COLOR,
        GRID_STYLE,
    )
    # reflow
    graph.draw_line(
        profile["stages"]["reflow"][0],
        profile["temp_range"][0],
        profile["stages"]["reflow"][0],
        profile["temp_range"][1] * 1.1,
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )
    graph.draw_line(
        profile["time_range"][0],
        profile["stages"]["reflow"][1],
        profile["time_range"][1],
        profile["stages"]["reflow"][1],
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )
    # cool
    graph.draw_line(
        profile["stages"]["cool"][0],
        profile["temp_range"][0],
        profile["stages"]["cool"][0],
        profile["temp_range"][1] * 1.1,
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )
    graph.draw_line(
        profile["time_range"][0],
        profile["stages"]["cool"][1],
        profile["time_range"][1],
        profile["stages"]["cool"][1],
        GRID_SIZE,
        GRID_COLOR,
        GRID_STYLE,
    )

    # draw labels
    x = profile["time_range"][0]
    y = profile["stages"]["reflow"][1]
    xp = int(GXSTART + graph.width * (x - graph.xmin) // (graph.xmax - graph.xmin))
    yp = int(GHEIGHT * (y - graph.ymin) // (graph.ymax - graph.ymin))

    label_reflow.x = xp + 10
    label_reflow.y = HEIGHT - yp
    label_reflow.text = str(profile["stages"]["reflow"][1])
    print("reflow temp:", str(profile["stages"]["reflow"][1]))
    print("graph point: ", x, y, "->", xp, yp)

    x = profile["stages"]["reflow"][0]
    y = profile["stages"]["reflow"][1]

    # draw time line (horizontal)
    graph.draw_line(
        graph.xmin, graph.ymin + 1, graph.xmax, graph.ymin + 1, AXIS_SIZE,
AXIS_COLOR, 1
    )
    graph.draw_line(
        graph.xmin, graph.ymax, graph.xmax, graph.ymax, AXIS_SIZE, AXIS_COLOR, 1
    )
    # draw time ticks
    tick = graph.xmin
    while tick < (graph.xmax - graph.xmin):
        graph.draw_line(
```

```
            tick, graph.ymin, tick, graph.ymin + 10, AXIS_SIZE, AXIS_COLOR, 1
        )
        graph.draw_line(
            tick,
            graph.ymax,
            tick,
            graph.ymax - 10 - AXIS_SIZE,
            AXIS_SIZE,
            AXIS_COLOR,
            1,
        )
        tick += 60

    # draw temperature line (vertical)
    graph.draw_line(
        graph.xmin, graph.ymin, graph.xmin, graph.ymax, AXIS_SIZE, AXIS_COLOR, 1
    )
    graph.draw_line(
        graph.xmax - AXIS_SIZE + 1,
        graph.ymin,
        graph.xmax - AXIS_SIZE + 1,
        graph.ymax,
        AXIS_SIZE,
        AXIS_COLOR,
        1,
    )
    # draw temperature ticks
    tick = graph.ymin
    while tick < (graph.ymax - graph.ymin) * 1.1:
        graph.draw_line(
            graph.xmin, tick, graph.xmin + 10, tick, AXIS_SIZE, AXIS_COLOR, 1
        )
        graph.draw_line(
            graph.xmax,
            tick,
            graph.xmax - 10 - AXIS_SIZE,
            tick,
            AXIS_SIZE,
            AXIS_COLOR,
            1,
        )
        tick += 50

    # draw profile
    x1 = profile["profile"][0][0]
    y1 = profile["profile"][0][1]
    for point in profile["profile"]:
        x2 = point[0]
        y2 = point[1]
        graph.draw_line(x1, y1, x2, y2, PROFILE_SIZE, PROFILE_COLOR, 1)
        # print(point)
        x1 = x2
        y1 = y2


def format_time(seconds):
    minutes = seconds // 60
    seconds = int(seconds) % 60
    return "{:02d}:{:02d}".format(minutes, seconds, width=2)


def check_buttons_press_location(p, button_details):
    """
    Function to easily check a button press within a list of Buttons
    """
    for button in button_details:
        if button.contains(p):
            return button
    return None
```

```
def change_profile(oven):
    """
    Function added to render the available profile selections to screen and then
load into memory

    Limitations: Only the first 6 profiles will be displayed to honor to the size
format of the screen
    """
    selected_file = None

    display_group = displayio.Group()
    gc.collect()

    title_label = label.Label(font3, text=TITLE)
    title_label.x = 5
    title_label.y = 14
    display_group.append(title_label)
    profile_label = label.Label(font2, text="Profile Change")
    profile_label.x = 5
    profile_label.y = 45
    display_group.append(profile_label)

    selected_label_default_text = "Selected Profile: "
    selected_label = label.Label(font1, text=selected_label_default_text)
    selected_label.x = 5
    selected_label.y = HEIGHT - 20
    display_group.append(selected_label)

    buttons = []
    button_details = {}
    button_x = 20
    button_y = 60
    button_y_start = 60
    button_height = 30
    button_width = 120
    spacing = 10
    profile_limit = 6
    count = 0
    dir_list = os.listdir("/profiles/")
    for f in dir_list:
        f = f.split('.')[0]
        button = Button(
            x=button_x, y=button_y, width=button_width, height=button_height,
label=f, label_font=font1
        )
        button_details[f] = [button_x, button_y, button_width, button_height]
        buttons.append(button)
        button_y += button_height + spacing

        count+=1

        if count == 3:
            button_x += button_width + spacing
            button_y = button_y_start

        if count >= 6:
            break

    save_button = Button(x=WIDTH-70, y=HEIGHT-50, width=60, height=40, label="Save",
label_font=font2)
    button_details["Save"] = [WIDTH-70, HEIGHT-50, 60, 40]
    buttons.append(save_button)

    for b in buttons:
        display_group.append(b)

    board.DISPLAY.root_group = display_group
```

```
    try:
        board.DISPLAY.refresh(target_frames_per_second=60)
    except AttributeError:
        board.DISPLAY.refresh_soon()
    print("Profile change display complete")

    while True:
        gc.collect()
        try:
            board.DISPLAY.refresh(target_frames_per_second=60)
        except AttributeError:
            board.DISPLAY.refresh_soon()

        p = ts.touch_point

        if p:
            button_pressed = check_buttons_press_location(p, buttons)
            if button_pressed:
                print(f"{button_pressed.label} button pressed")
                if button_pressed.label == "Save":
                    with open("/profiles/" + selected_file + ".json", mode="r") as
fpr:
                        oven.sprofile = json.load(fpr)
                        fpr.close()
                        oven.reset()
                    return
                else:
                    selected_file = button_pressed.label
                    print(f"Profile selected: {selected_file}")
                    selected_label.text = selected_label_default_text + " " +
button_pressed.label

                time.sleep(1)  # for debounce


def default_view():
    """
    The below code was wrapped into this fucntion to give execution back and forth
between this the default view
    of the EZ Make Oven and the alternative profile selection view.

    As such there were numerous global variables that have been declared as needed
in the various other functions
    that use them.
    """

    global label_reflow, oven, message, timediff, plot, sgraph, timer_data

    display_group = displayio.Group()
    board.DISPLAY.root_group = display_group

    plot = displayio.Bitmap(GWIDTH, GHEIGHT, 4)

    display_group.append(
        displayio.TileGrid(plot, pixel_shader=palette, x=GXSTART, y=GYSTART)
    )

    timediff = 0

    print("melting point: ", oven.sprofile["melting_point"])

    label_reflow = label.Label(font1, text="", color=0xFFFFFF, line_spacing=0)
    label_reflow.x = 0
    label_reflow.y = -20
    display_group.append(label_reflow)
    title_label = label.Label(font3, text=TITLE)
    title_label.x = 5
    title_label.y = 14
```

```
display_group.append(title_label)
# version_label = label.Label(font1, text=VERSION, color=0xAAAAAA)
# version_label.x = 300
# version_label.y = 40
# display_group.append(version_label)
message = label.Label(font2, text="Wait")
message.x = 100
message.y = 40
display_group.append(message)
alloy_label = label.Label(font1, text="Alloy:", color=0xAAAAAA)
alloy_label.x = 5
alloy_label.y = 40
display_group.append(alloy_label)
alloy_data = label.Label(font1, text=str(oven.sprofile["alloy"]))
alloy_data.x = 10
alloy_data.y = 60
display_group.append(alloy_data)
profile_label = label.Label(font1, text="Profile:", color=0xAAAAAA)
profile_label.x = 5
profile_label.y = 80
display_group.append(profile_label)
profile_data = label.Label(font1, text=oven.sprofile["title"])
profile_data.x = 10
profile_data.y = 100
display_group.append(profile_data)
timer_label = label.Label(font1, text="Time:", color=0xAAAAAA)
timer_label.x = 5
timer_label.y = 120
display_group.append(timer_label)
timer_data = label.Label(font3, text=format_time(timediff))
timer_data.x = 10
timer_data.y = 140
display_group.append(timer_data)
temp_label = label.Label(font1, text="Temp(C):", color=0xAAAAAA)
temp_label.x = 5
temp_label.y = 160
display_group.append(temp_label)
temp_data = label.Label(font3, text="--")
temp_data.x = 10
temp_data.y = 180
display_group.append(temp_data)
circle = Circle(308, 12, 8, fill=0)
display_group.append(circle)

sgraph = Graph()

# sgraph.xstart = 100
# sgraph.ystart = 4
sgraph.xstart = 0
sgraph.ystart = 0
# sgraph.width = WIDTH - sgraph.xstart - 4  # 216 for standard PyPortal
# sgraph.height = HEIGHT - 80  # 160 for standard PyPortal
sgraph.width = GWIDTH  # 216 for standard PyPortal
sgraph.height = GHEIGHT  # 160 for standard PyPortal
sgraph.xmin = oven.sprofile["time_range"][0]
sgraph.xmax = oven.sprofile["time_range"][1]
sgraph.ymin = oven.sprofile["temp_range"][0]
sgraph.ymax = oven.sprofile["temp_range"][1] * 1.1
print("x range:", sgraph.xmin, sgraph.xmax)
print("y range:", sgraph.ymin, sgraph.ymax)
draw_profile(sgraph, oven.sprofile)
buttons = []
if oven.sensor_status:
    button = Button(
        x=0, y=HEIGHT - 40, width=80, height=40, label="Start", label_font=font2
    )
    buttons.append(button)
    profile_button = Button(
        x=WIDTH - 100, y=40, width=100, height=30, label="Profile Change",
```

```
label_font=font1
        )
        buttons.append(profile_button)

    for b in buttons:
        display_group.append(b)

    try:
        board.DISPLAY.refresh(target_frames_per_second=60)
    except AttributeError:
        board.DISPLAY.refresh_soon()
    print("display complete")
    last_temp = 0
    last_state = "ready"
    last_control = False
    second_timer = time.monotonic()
    timer = time.monotonic()

    display_profile_button = True
    is_Running = True
    while is_Running:
        gc.collect()
        try:
            board.DISPLAY.refresh(target_frames_per_second=60)
        except AttributeError:
            board.DISPLAY.refresh_soon()
        oven.beep.refresh()  # this allows beeps less than one second in length
        try:
            oven_temp = int(oven.sensor.temperature)
        except AttributeError:
            oven_temp = 32  # testing
            oven.sensor_status = False
            message.text = "Bad/missing temp sensor"
        if oven.control != last_control:
            last_control = oven.control
            if oven.control:
                circle.fill = 0xFF0000
            else:
                circle.fill = 0x0
        p = ts.touch_point
        status = ""
        last_status = ""

        if p:
            if button.contains(p):
                print("touch!")
                if oven.state == "ready":
                    button.label = "Stop"
                    oven.set_state("start")
                else:
                    # cancel operation
                    message.text = "Wait"
                    button.label = "Wait"
                    oven.set_state("wait")
                time.sleep(1)  # for debounce

            elif profile_button.contains(p):
                if message.text == "Ready":  # only allow profile change when NOT
running
                    print("Profile change button pressed")
                    is_Running = False
                    change_profile(oven)

        if oven.sensor_status:
            if oven.state == "ready":
                status = "Ready"
                display_profile_button = True
                if last_state != "ready":
                    oven.beep.refresh()
```

```
                    oven.reset()
                    draw_profile(sgraph, oven.sprofile)
                    timer_data.text = format_time(0)
                if button.label != "Start":
                    button.label = "Start"
            if oven.state == "start":
                status = "Starting"
                display_profile_button = False
                if last_state != "start":
                    timer = time.monotonic()
            if oven.state == "preheat":
                if last_state != "preheat":
                    timer = time.monotonic()  # reset timer when preheat starts
                status = "Preheat"
                display_profile_button = False
            if oven.state == "soak":
                status = "Soak"
                display_profile_button = False
            if oven.state == "reflow":
                status = "Reflow"
                display_profile_button = False
            if oven.state == "cool" or oven.state == "wait":
                status = "Cool Down, Open Door"
            if last_status != status:
                message.text = status
                last_status = status

            if oven_temp != last_temp and oven.sensor_status:
                last_temp = oven_temp
                temp_data.text = str(oven_temp)
            # update once per second when oven is active
            if oven.state != "ready" and time.monotonic() - second_timer >= 1.0:
                second_timer = time.monotonic()
                oven.check_state()
                if oven.state == "preheat" and last_state != "preheat":
                    timer = time.monotonic()  # reset timer at start of preheat
                timediff = int(time.monotonic() - timer)
                timer_data.text = format_time(timediff)
                print(oven.state)
                if oven_temp >= 50:
                    sgraph.draw_graph_point(
                        int(timediff), oven_temp, size=TEMP_SIZE, color=TEMP_COLOR
                    )

            last_state = oven.state

        # Manage whether the Profile Button should be displayed or not
        if display_profile_button == True:
            try:
                display_group.append(profile_button)
            except ValueError:
                # profile_button already in the display_group
                pass
        else:
            try:
                display_group.remove(profile_button)
            except ValueError:
                # profile_button is missing, handle gracefully
                pass

# Global variables that are used in numerous of the supporting functions
font1 = bitmap_font.load_font("/fonts/OpenSans-9.bdf")
font2 = bitmap_font.load_font("/fonts/OpenSans-12.bdf")
font3 = bitmap_font.load_font("/fonts/OpenSans-16.bdf")
label_reflow = None
oven = ReflowOvenControl(board.D4)
message = None
timediff = 0
plot = None
```

```
sgraph = None
timer_data = None

# Essentially the main function of the entire codebase
while True:
    default_view()
```

Using the Oven

Danger: Do not use your reflow toaster oven for cooking food! Solder used in reflow ovens include lead and other dangerous metals and must not be used in the same oven for cooking food.

Warning: Do not leave your reflow oven unattended. Before leaving, turn off the oven power using the switch on the IoT relay, which shuts down both the oven and the PyPortal.

With all the hardware in place, the **config.json** file updated and the code installed, you are now ready to use the EZ Make Oven! Before we start reflowing boards, however, we need to do a dry run test to ensure the equipment is functioning and the components are communicating properly.

Verify the thermocouple is in the center of the oven at or near the rack where the PCB board will be placed. The close proximity to the PCB will ensure an accurate temperature reading. For a more accurate reading, taping the thermocouple to a PCB will measure the temperature at the PCB surface. This can be done using a high temperature tape such as Polyimide Tape (http://adafru.it/3057) (sometimes referred to by the brand name Kapton Tape). If the PCB board is too small or has no room, you can use a spare empty PCB and place it next to the PCB to be reflowed.

High Temperature Polyimide Tape - 1cm wide x 33 meter roll
Polyimide Tape (sometimes referred to by the brand name Kapton Tape) is an interesting addition to your toolbox! Polyimide Tape remains stable across...
https://www.adafruit.com/product/3057

Once you have verified the connections and the placement of the thermocouple, press the "Start" button to begin the dry run. This will take a few minutes to run. You will see the oven go through several stages, including start, preheat, soak, reflow and cool down. There will be a short beep for each stage, ending with a long 5 second beep for the cool down stage and you can open the oven door.

You are now ready to use your EZ Make Oven. Need some help designing a PCB to use in your reflow oven? Look at this Adafruit Learning Guide for Eagle (https://adafru.it/GfL).

Troubleshooting Tips

Is your oven not working as expected? Here are a couple of troubleshooting tips. If you still have a problem, you can ask a question in the Adafruit Forums.

The oven turns on, but the temperature goes down instead of up.

Your thermocouple wiring is backwards, swap the two wires on the MCP9600

The red circle oven indicator displays on the PyPortal screen, but my oven does not turn on.

Check that the relay light is on in the relay module. If not, check the cabling between the PyPortal and the relay. If the connection appears to be OK, you may have the wires backwards. Verify at the relay module that the white cable is on the left and the black cable is on the right.

Verify the oven controls are set properly. They should be set to maximum temperature, toast mode, and timer set to "always on". If the timer does not have that option, set it to the maximum time each time you use the oven.

The display shows "Bad/missing temp sensor"

This means the EZ Make Oven can not find the MCP9600 sensor used to measure the oven temperature. Verify the four wires of the I2C cable connecting the PyPortal to the sensors are connected properly. Also, verify the **config.json** file contains the correct I2C address for the sensor. This must be a decimal (not hexadecimal) value.

If these options do not help, you may need to change the power of the I2C port from 5 volts to 3.3 volts. This is done by cutting and soldering a trace in the back of the PyPortal near the I2C port. You can find more discussion on this at this link (https://adafru.it/GfM).

The PyPortal hangs on boot up

Temporarily unplug the MCP9600 from the PyPortal and reboot it. If it boots up properly, then you may need to change the power of the I2C port from 5 volts to 3.3 volts. This is done by cutting and soldering a trace in the back of the PyPortal near the I2C port. You can find more discussion on this at this link (https://adafru.it/GfM).