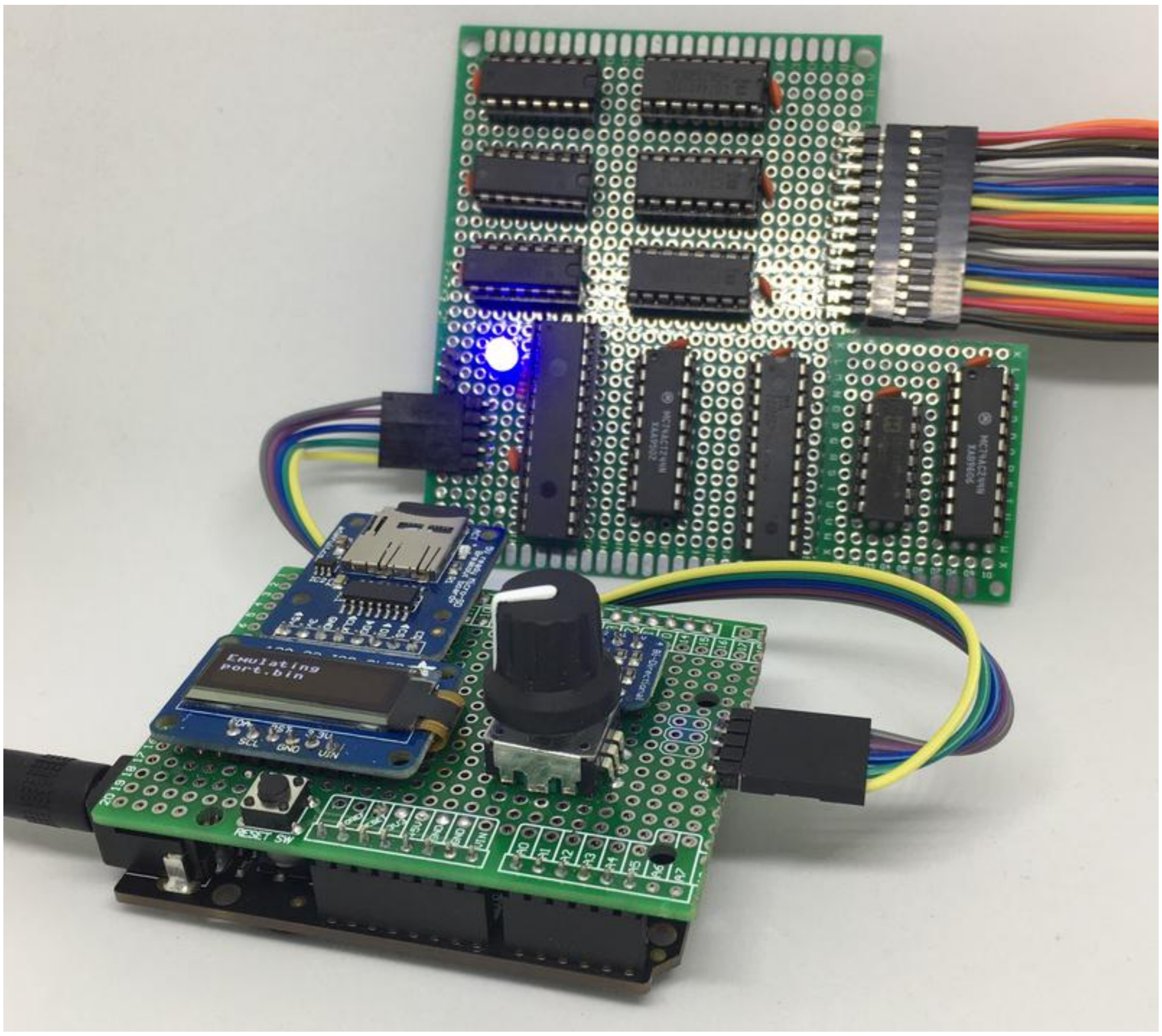
Digital Circuits 6: An EPROM Emulator

Created by Dave Astels



https://learn.adafruit.com/digital-circuits-6-eprom-emulator

Last updated on 2022-12-01 03:12:03 PM EST

Table of Contents

Overview 3

- Parts Used

Interface Hardware 5

Emulator Hardware 7

- A Tale of Two Memories
- Dual porting the 6116
- Controlling the Emulator
- Loading the 6116

An aside: The Octal 3-state Buffers 11

- 3-state
- Octal 3-state buffer chips
- The 74245 Octal Bus Transceiver

CircuitPython Code 15

- Main
- Directories
- Emulator

Series Index 25

Overview

I recently designed and breadboarded a simple Z80 based computer. This was the sort of thing I did for fun and profit years ago. This time it was, well, for fun and profit. But it was a little different. Back in the day I had a drawer full of Erasable Programmable Read-Only memory chips (EPROMs), as well as an EPROM eraser and programmer*. This time I had none of that. So I'd have to get inventive.

When I designed the system, I put in 2K of ROM/EPROM and 2K of static ram. Naturally when I bought parts, I got extras. So, that meant I had an extra 2K static ram chip sitting in a drawer. That's a powerful thing: having extra parts sitting in drawers. What I decided to do was build an EPROM emulator using that ram chip, some transistor/transistor (TTL) logic and a microcontroller board.

I was using a Z80 assembler (`z80asm` from z80pack ()) on Linux to write code for the Z80, so using a microSD card seemed a reasonable way to get code from Linux to the emulator. Once I had decided to do that, I needed a way to navigate the file system on the card, select a binary file, and load it onto the RAM in the emulator. See how things snowball!

To make the project a bit simpler and more modular, I chose to divide it into two separate parts: the controller/user interface (UI), and the emulator. To maximize the flexibility of that separation, I decided to connect them using an I^{2}C bus. That proved to be a very good choice as I went through several iterations using different microcontroller boards: from an Arduino Mega2620 to an ItsyBisty M0 Express and finally to an Metro M4 Express.

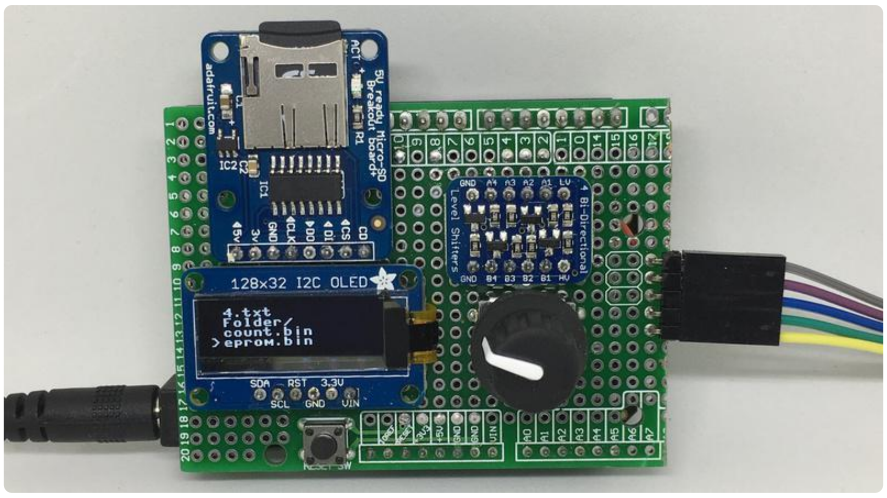
While I designed this project to support my work on a Z80 system, it can be used to replace a 2716 EPROM in any system. In this guide, I'll call that system the host.

* I'm planning to build a new take on an EPROM programmer sometime in the near future which will share some features of this project, and in some ways could be considered as Part 2. It will most likely use the same controller/ui design.

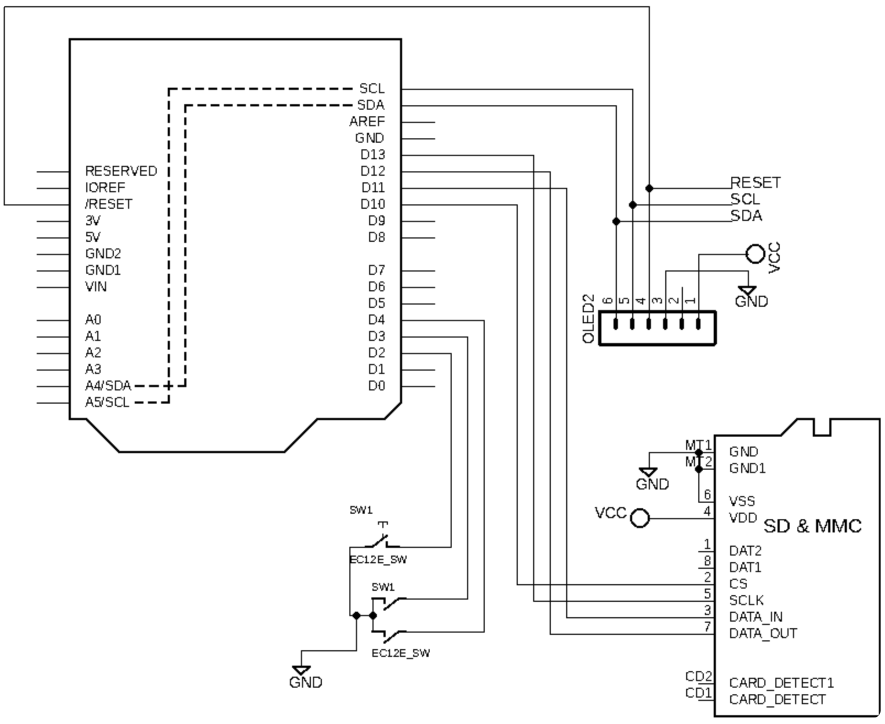
Parts Used

1 x MicroSD card breakout board+ MicroSD card breakout board+	https://www.adafruit.com/product/254
1 x Monochrome 128x32 I2C OLED graphic display Monochrome 128x32 I2C OLED graphic display	https://www.adafruit.com/product/931
1 x Rotary Encoder + Extras Rotary Encoder + Extras	https://www.adafruit.com/product/377
1 x Adafruit Metro M4 Microchip ATSAMD51	https://www.adafruit.com/product/3382
1 x 4-channel I2C-safe Bi-directional Logic Level Converter 4-channel I2C-safe Bi-directional Logic Level Converter - BSS138	https://www.adafruit.com/product/757
1 x MCP23017 - i2c 16 input/output port expander chip MCP23017 - i2c 16 input/output port expander	https://www.adafruit.com/product/732
1 x Silicone Cover Stranded-Core Wire 30AWG Silicone Cover Stranded-Core Wire - 50ft 30AWG Black	https://www.adafruit.com/product/3164

Interface Hardware



The interface hardware is pretty simple. My end goal is to build it around an ItsyBitsy M4 Express board. Alas there aren't any yet, so I did the next best thing: I build it as a Metro M4 Express shield. Not the compact form factor I wanted, but all the same capabilities.



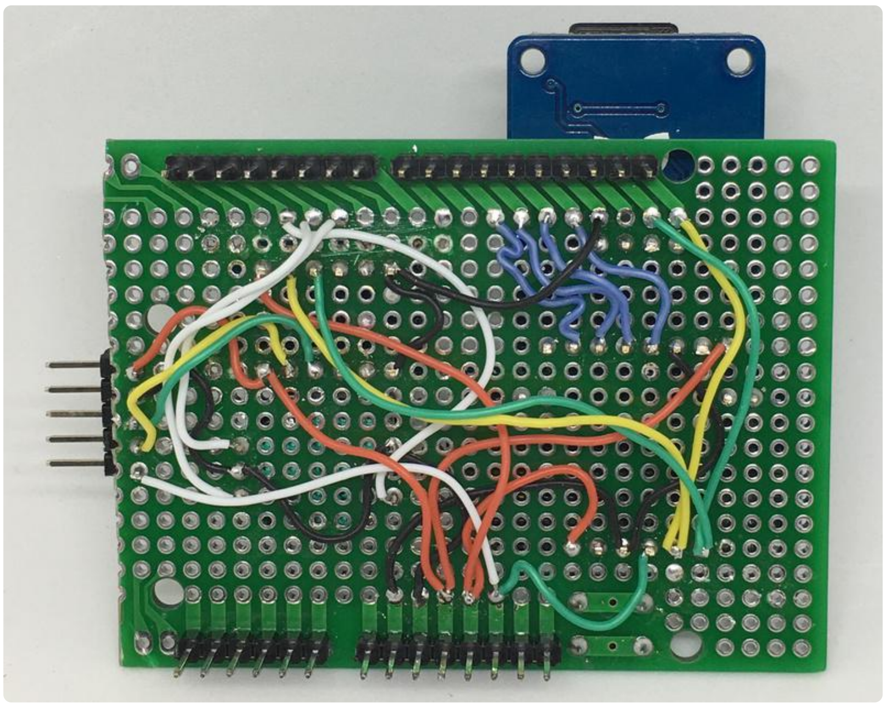
There's a microSD breakout hooked to the SPI lines, a 128x32 OLED on the I^{2}C lines, and a rotary encoder (with push switch) using 3 digital inputs.

See the related guides for more details on each of these:

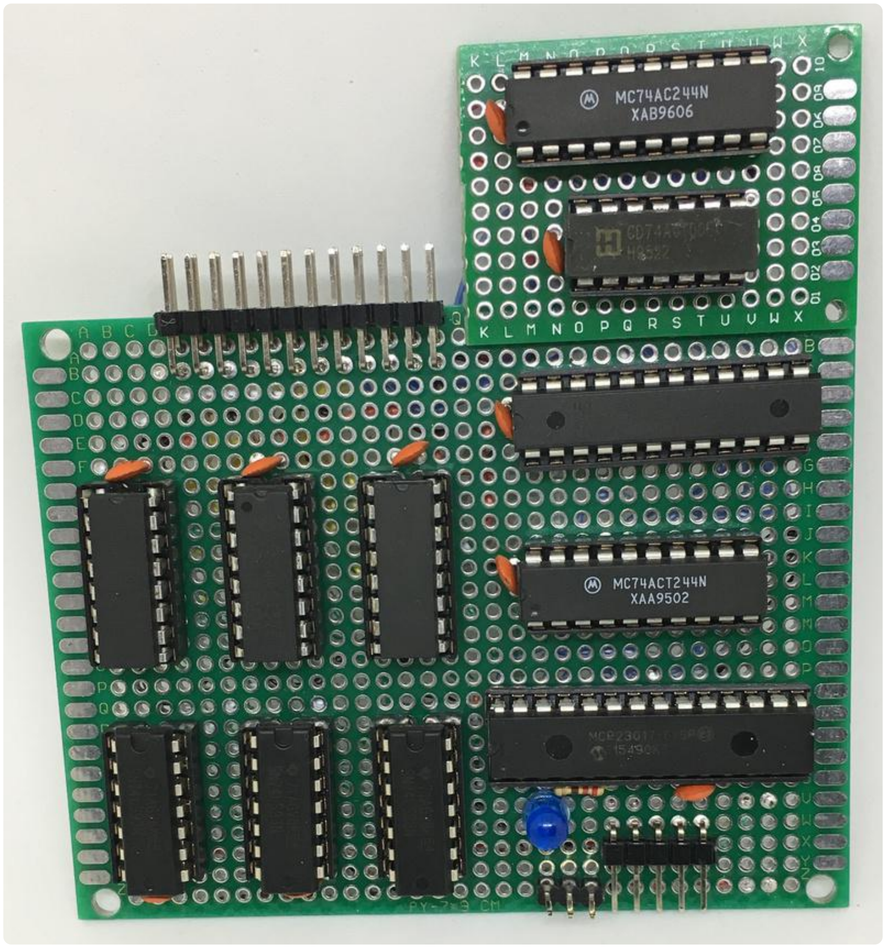
- Metro M4 ()
- microSD Breakout Tutorial ()
- OLED Displays ()
- Rotary Encoder ()

Because I made the decision to use I^2C between the interface circuit and the emulator circuit, any MCU with I^2C capabilities would work.

One small wrinkle: the Metro M4 is a 3.3v system while the emulator is 5v. Adafruit to the rescue with the BSS138 based 4-channel I^2C-safe Bi-directional Logic Level Converter. We just need to run the I^2C lines through this between the two circuits.



Emulator Hardware



A Tale of Two Memories

For my host design I used what I knew: the 2716 EPROM and 6116 static RAM. The nice thing about this pair of chips is that they are pin for pin compatible: you can swap them in a design.

We can take advantage of this in the emulator to replace the 2716 on the host board with a 6116 in the emulator. That way we can put code into the 6116 (since it's a RAM) and the host treats it like an EPROM.

If we just plugged the 6116 into the host there would be no way to get code into it. So we need to have it accessible from a controller. That would allow us read binary files from an SD card and write them to the 6116, then let the host use it in place of the 2716 it thinks is there.

The question is how to make that happen.

Dual porting the 6116

Basically we want to dual port the 6116: let two different systems use it. The controller would write into it and the host would read from it.

History note: the Apple][design dual ported the system RAM to allow both the CPU and the video system to have interleaved access (the video hardware had access while the CPU was in it's "working internally" phase of each cycle. This allowed for very simple, efficient display hardware, with the bonus that video access took care of the dynamic RAM's refresh requirements (see Digital Circuits 5: Memories () for more on dynamic RAM).

Below is the schematic of the emulator. At the bottom-left and bottom right is the address and data related signals, respectively, from the host.

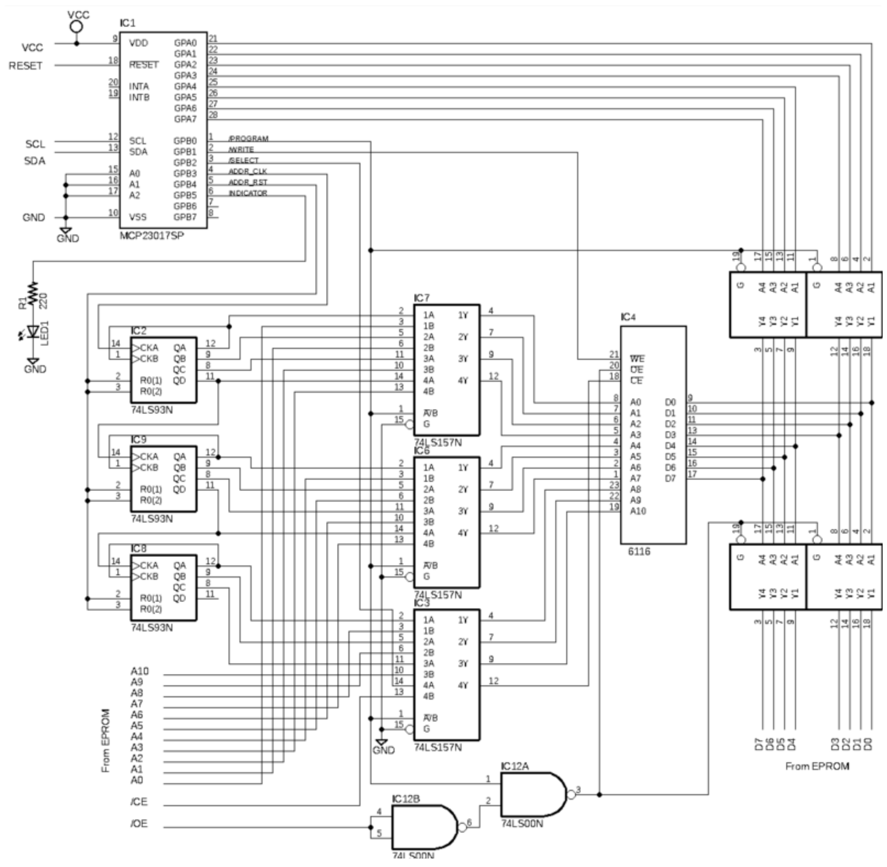
The data bus from the host (on the right) is connected to the 6116 through a tristate buffer (the '244) that isolates the RAM from the host while the emulator is loading data into the RAM. The output enable of the RAM is controlled by the same signal that controls that '244. That's what the NAND gates at the bottom are for. The output of the RAM and the buffer to connect the RAM's data pins to the host are controlled by the output enable from the host EPROM connection, gated by the emulator's mode signal (low when the 6116 is being loaded, high when the host has access). Using NANDs and inverting the host's output enable means that data flows from the 6116 to the host when the host asks for it (the /OE from the host is low) and the emulator is in host access mode.

There is another '244 connecting the controller to the data bus of the 6116 while the emulator is in program mode. More about that later.

Now consider the address bus from the host (bottom left). It doesn't connect to the 6116 directly or though a buffer. Instead it goes connects to the B sides of 11 dual input multiplexers (the 74LS157s in the center of the schematic). 2K of memory needs an 11-bit binary number to cover all locations. The chip enable signal from the host goes through a multiplexer as well. The outputs of the multiplexers connect to the corresponding pins of the 6116.

The other inputs to the multiplexers? They come from a bank of 4-bit binary counters (on the left in the schematic) that are chained together to make a single 12-bit counter. Since we only need 11 address lines, the highest valued output of the counter is not used. The A side of the final multiplexer connects to a chip select control signal described later.

The outputs of the multiplexers go to the corresponding input of the 6116.



Controlling the Emulator

So that explains the dual-porting, the host connection, and the emulator address generation. Let's turn our attention to the top part of the schematic. Core to this is a MCP23017 that is controlled via I^{2}C by the control board.

Port A is simple: it's just the 8-bit data to be placed into the 6116. When the emulator is in programming mode the second '244 is enabled, connecting the output of port A to the 6116's data pins.

Port B is a collection of single-bit control signals, starting with bit 0:

- /PROGRAM is low when the emulator is in program mode: i.e. the 6116 address is provided by the counters, and it's data by port A of the 23017; the host is cut off from the 6116 in this mode. When /PROGRAM is high, the 6116 is connected to the host.
- /WRITE connects to the write enable input of the 6116. When it goes low the values on its data pins are written into the location specified by the values on its address pins.

- /SELECT connects to the 6116's chip enable pin when in program mode. This serves to enable the chip for subsequent reading writing. The 6116 does nothing unless its chip enable pin is low. When in EMULATE mode, the chip enable comes from the host.
- ADDR_CLK increments the address counter when it is pulsed low briefly.
- ADDR_RST is normally low, and making it high briefly resets the address counter to 0.
- INDICATOR is connected to an LED simply as an indication of when the emulator is in EMULATE mode. I used a separate port B output for this due to the current draw of an LED. Just to be safe, and since I had port B outputs to spare.

Loading the 6116

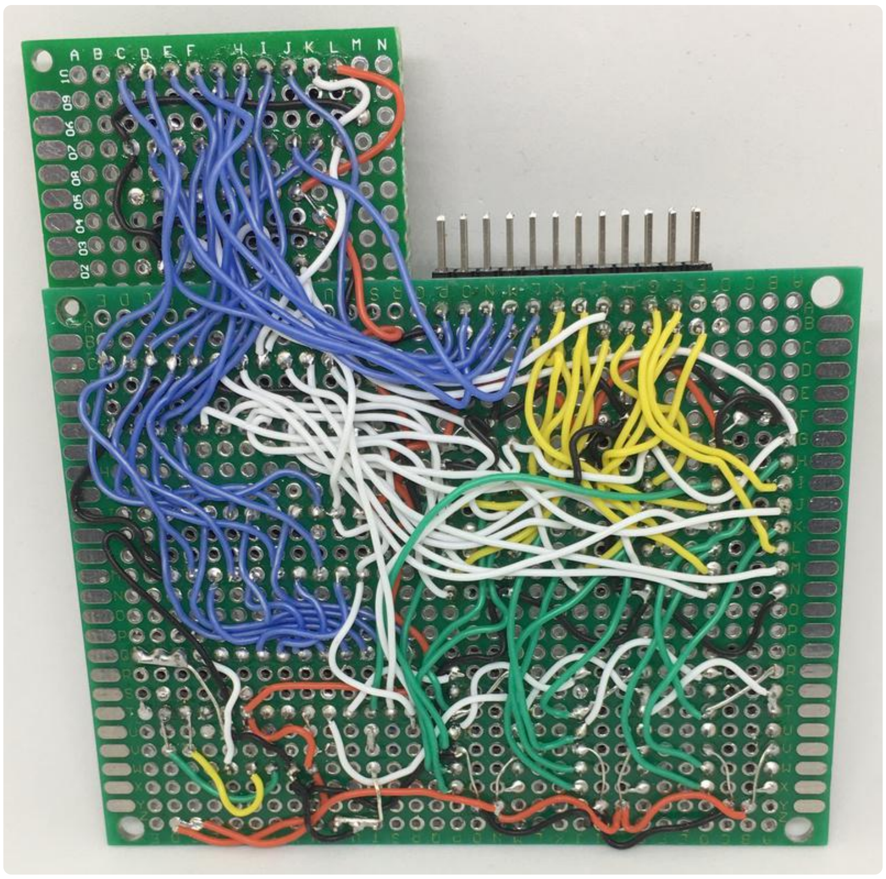
Loading data into the 6116 is simple, but the sequencing has to be right:

- Place the emulator into PROGRAM mode (take /PROGRAM low)
- Reset the address counter (pulse ADDR_RST high)
- For each byte to be programmed:
    - Output the next byte on port A of the 23017
    - Activate the 6116 (take /SELECT low)
    - Write the data (pulse /WRITE low)
    - Deactivate the 6116 (take /SELECT high)
    - Advance the address counter (pulse ADDR_CLK low)
- Place the emulator into EMULATE mode (take /PROGRAM high)

One limitation incurred by using an address counter (rather than an additional 23017 to provide the address) is that arbitrary locations cannot be written. That's not a problem since the use case is to load the 6116 with the contents of a 2K EPROM; writing arbitrary locations isn't required.

One advantage of using an address counter is that it provides a better example of the circuits we've been looking at in previous parts of this guide series.

This circuit has a little of everything: logic gates, counters, multiplexers, and memory. It also includes the 74244 chip that we saw back in part 2. Let's have a closer look this time.



An aside: The Octal 3-state Buffers

3-state

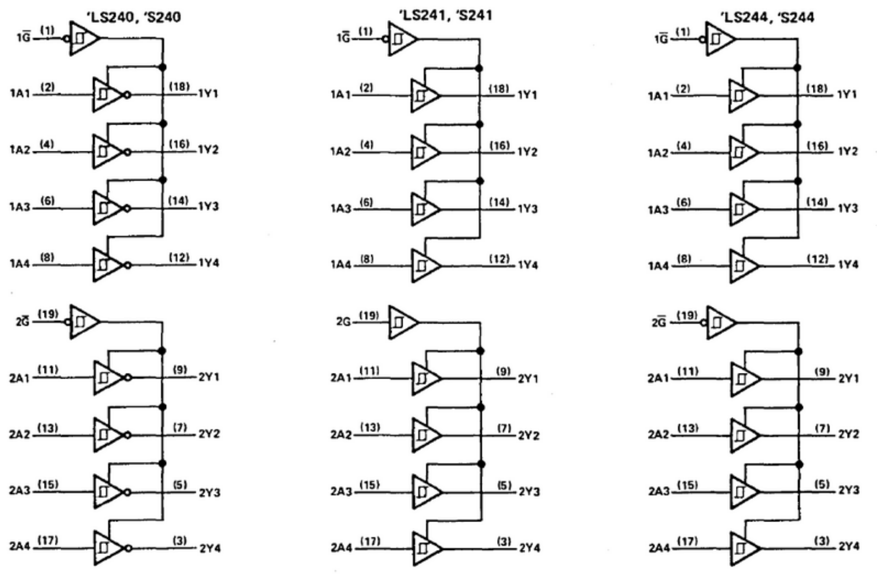
Three state? But, you say, "I thought binary had 2 states". Well it does: high and low. But what about "disconnected", i.e. no signal? That's the third state we're talking about. A signal line in this third state, aka hi-impedance aka open-collector, doesn't provide a logic level, or anything; it's electrically disconnected. If you have multiple signal sources all running through 3-state buffers, you can enable 1 of them at a time and its logic value will propagate onward.

It's a handy way to basically turn off a signal. If you remember back to part 1, I said that an logic output could connect to some number (limited by its fan-out) of logic inputs, but that multiple outputs could not connect to a single input? Well, 3-state buffers are the way around that. They let you connect one output at a time to an input, controlled by other logic signals.

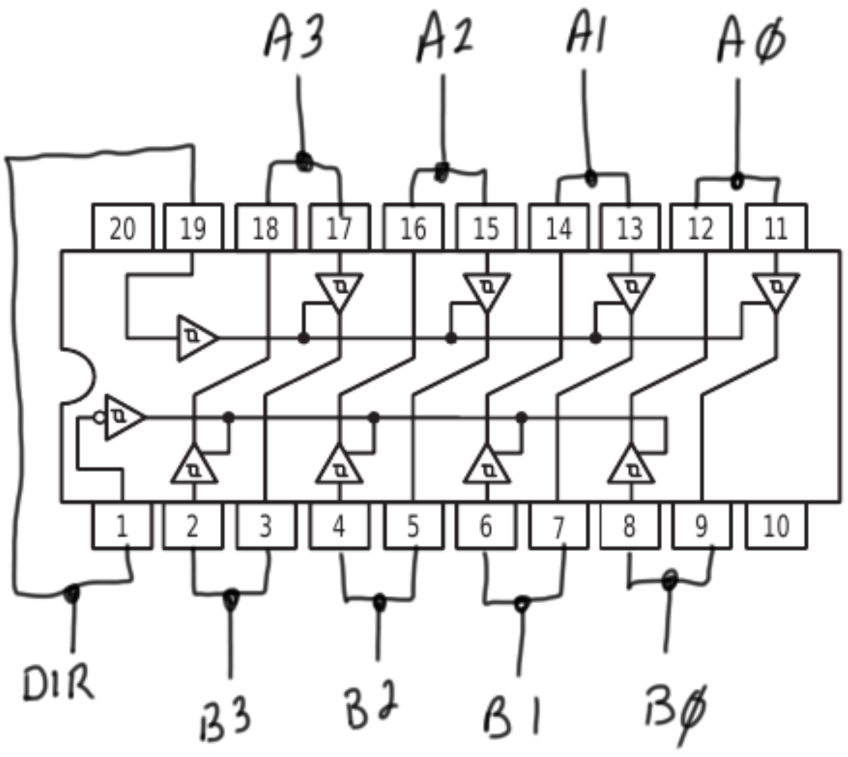
Octal 3-state buffer chips

Consider this family of octal tri-state buffer chips: 74240, 74241, and 74244. They are slightly different from each other, and all incredibly useful. All three include two sets of four tri-state buffers, each set with it's own enable input. They differ along two axis: whether the the buffers are inverting or non-inverting, and whether the two enable inputs are both active low or one active low and one active high.

The '240 is the only one of the three with inverting buffers; the '241 and '244 are both non-inverting. The '241 is the only one of the three with one active low enable, and one active high enable; the '240 and '241 have two active low enable inputs.

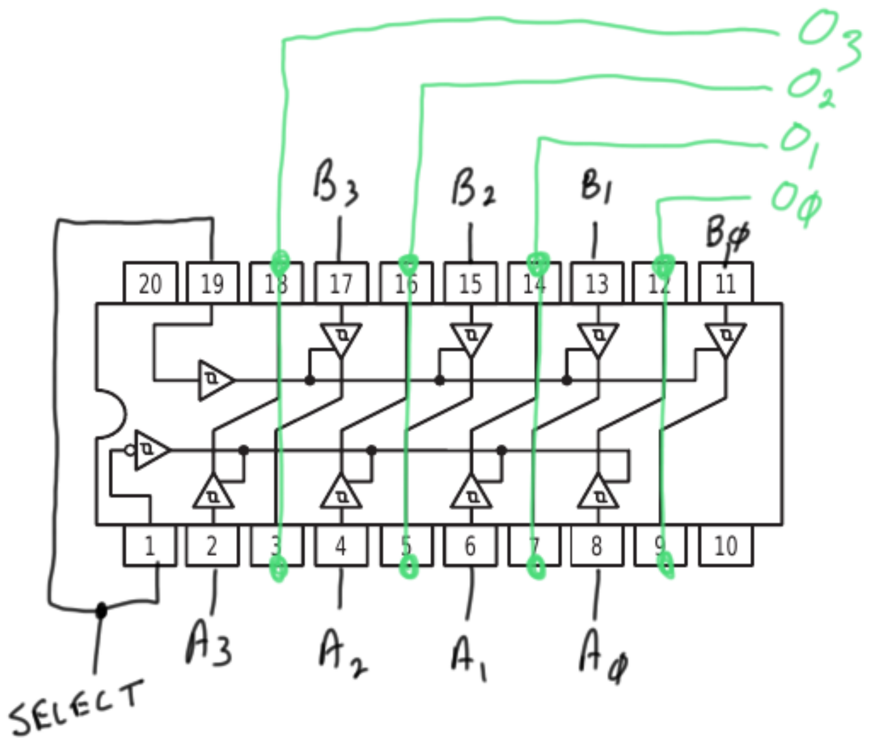


Why would you want the two enable inputs to be different? Notice how the two sets of buffers go in the opposite direction. Consider if you connect pairs together as well as connecting the enable inputs together on a '241:



What we have now is a bidirectional 4-bit buffer. The DIR signal controls which buffer of each pair is enabled. That determines whether data flows A->B or B->A because only one buffer in each pair will be enabled, the buffer going the opposite direction will be disabled. This is much like the 74245 buffer chip we'll look at shortly.

Another potential use is to make a 4 2-input multiplexers:

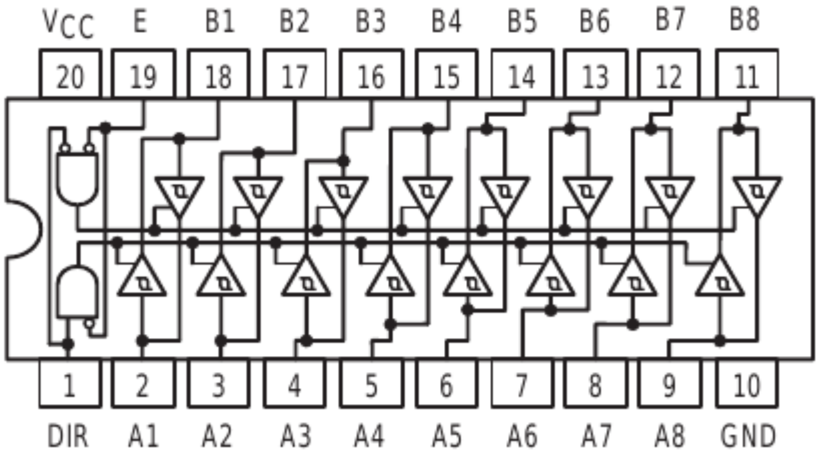


Here, the enable pins are tied together as a select input that enables (as before) one buffer from each pair. In this case, though, the outputs of the pair of buffers are tied together and each input is part of an A or B set of signals. Now the select input choose either A or B to be connected to the O outputs.

The 74245 Octal Bus Transceiver

The '244 and crew are great if you need data to flow (or not) in a single direction. They can even be convinced to let you control the direction of bidirectional data flow. We did that in the above circuit. However, while it lets us control the direction of flow, we lost the ability to disable it entirely. Also, it left us with only 4 bits being controlled. Computers, especially since 8-bit CPUs became a thing, like to have groups of signals that are multiples of 8 in size.

Add to this that bidirectional, and enable/disable control over a set of related signals (generally referred to as a bus) is incredibly useful. Enter the 74245.



This chip has it all. Eight bits wide, direction control, and an overall enable/disable. You'll be hardpressed to find an 8-bit computer that didn't use one of these to buffer its data bus (and likely a couple of '244s buffering it's address bus). The 8-bit CPUs did not have adequate fan-out to handle the amount of circuitry and peripheral chips in a typical computer, so their buses had to be buffered and '244s and '245s were the chips for the job.

Note that we could build an equivalent circuit with a couple '241s and some gates. But that would take 2 20-pin chips and a couple 14-pin ones. The '245 does it all in a single 20-pin package.

One notable thing about the '245 is that all the As are on one side and all the Bs are on the other. This isn't that big of a deal on a PCB, but on a breadboard it's really handy to have a bus go in one side of the chip and out the other. So much so that I'll use this in place of a '244 for one-direction buffering of a bus. All that needs to be done is to hardwire the DIR input to have signals flow in the desired direction.

CircuitPython Code

Like the hardware, the software is very modular:

- Overall setup, and control, including handling the rotary encoder
- A class representing a directory on the SD card with the ability to navigate up and down the tree
- A class to manage the emulator
- A helper class to properly debounce the encoder push switch (this is usable for any use of input switches)

Main

Main is simple enough:

- Initialize things
- Helper functions
- Loop, handling the rotary encoder and its switch and using the results to manipulate the current directory node and the emulator

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
The MIT License (MIT)

Copyright (c) 2018 Dave Astels

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
```

```
THE SOFTWARE.

--------------------------------------------------------------------------------

EPROM emulator UI in CircuitPython.
Targeted for the SAMD51 boards.

by Dave Astels
"""

import sdcardio
import adafruit_ssd1306
import board
import busio
import digitalio
import storage
from adafruit_debouncer import Debouncer
from directory_node import DirectoryNode
from emulator import Emulator

# pylint: disable=global-statement
# ------------------------------------------------------------------------------
# Initialize Rotary encoder

# Encoder button is a digital input with pullup on D2
encoder_switch = digitalio.DigitalInOut(board.D2)
encoder_switch.direction = digitalio.Direction.INPUT
encoder_switch.pull = digitalio.Pull.UP
button = Debouncer(encoder_switch)

# Rotary encoder inputs with pullup on D3 & D4
rot_a = digitalio.DigitalInOut(board.D4)
rot_a.direction = digitalio.Direction.INPUT
rot_a.pull = digitalio.Pull.UP

rot_b = digitalio.DigitalInOut(board.D3)
rot_b.direction = digitalio.Direction.INPUT
rot_b.pull = digitalio.Pull.UP

# ------------------------------------------------------------------------------
# Initialize I2C and OLED

i2c = busio.I2C(board.SCL, board.SDA)

oled = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
oled.fill(0)
oled.text("Initializing SD", 0, 10)
oled.show()

# ------------------------------------------------------------------------------
# Initialize SD card

# SD_CS = board.D10
# Connect to the card and mount the filesystem.
spi = busio.SPI(board.D13, board.D11, board.D12)  # SCK, MOSI, MISO
sdcard = sdcardio.SDCard(spi, board.D10)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")

oled.fill(0)
oled.text("Done", 0, 10)
oled.show()

# ------------------------------------------------------------------------------
# Initialize globals

encoder_counter = 0
encoder_direction = 0
```

```
# constants to help us track what edge is what
A_POSITION = 0
B_POSITION = 1
UNKNOWN_POSITION = -1  # initial state so we know if something went wrong

rising_edge = falling_edge = UNKNOWN_POSITION

PROGRAM_MODE = 0
EMULATE_MODE = 1

current_mode = PROGRAM_MODE
emulator = Emulator(i2c)


# ----------------------------------------------------------------------------------
# Helper functions

def is_binary_name(filename):
    return filename[-4:] == ".bin"


def load_file(filename):
    data = []
    with open(filename, "rb") as f:
        data = f.read()
    return data


def display_emulating_screen():
    oled.fill(0)
    oled.text("Emulating", 0, 0)
    oled.text(current_dir.selected_filename, 0, 10)
    oled.show()


# pylint: disable=global-statement
def emulate():
    global current_mode
    data = load_file(current_dir.selected_filepath)
    emulator.load_ram(data)
    emulator.enter_emulate_mode()
    current_mode = EMULATE_MODE
    display_emulating_screen()


# pylint: disable=global-statement
def program():
    global current_mode
    emulator.enter_program_mode()
    current_mode = PROGRAM_MODE
    current_dir.force_update()


# ----------------------------------------------------------------------------------
# Main loop

current_dir = DirectoryNode(oled, name="/sd")
current_dir.force_update()
rising_edge = falling_edge = UNKNOWN_POSITION
rotary_prev_state = [rot_a.value, rot_b.value]

while True:
    # reset encoder and wait for the next turn
    encoder_direction = 0

    # take a 'snapshot' of the rotary encoder state at this time
    rotary_curr_state = [rot_a.value, rot_b.value]

    # See https://learn.adafruit.com/media-dial/code
    if rotary_curr_state != rotary_prev_state:
        print("Was: {}".format(rotary_prev_state))
        print("Now: {}".format(rotary_curr_state))
        if rotary_prev_state == [True, True]:
            if not rotary_curr_state[A_POSITION]:
                print("Falling A")
                falling_edge = A_POSITION
            elif not rotary_curr_state[B_POSITION]:
                print("Falling B")
                falling_edge = B_POSITION
            else:
                continue

        if rotary_curr_state == [True, True]:
            if not rotary_prev_state[B_POSITION]:
                rising_edge = B_POSITION
                print("Rising B")
            elif not rotary_prev_state[A_POSITION]:
                rising_edge = A_POSITION
                print("Rising A")
            else:
                continue

            # check first and last edge
            if (rising_edge == A_POSITION) and (falling_edge == B_POSITION):
                encoder_counter -= 1
                encoder_direction = -1
                print("%d dec" % encoder_counter)
            elif (rising_edge == B_POSITION) and (falling_edge == A_POSITION):
                encoder_counter += 1
                encoder_direction = 1
                print("%d inc" % encoder_counter)
            else:
                # (shrug) something didn't work out, oh well!
                encoder_direction = 0

            # reset our edge tracking
            rising_edge = falling_edge = UNKNOWN_POSITION

        rotary_prev_state = rotary_curr_state

        # Handle encoder rotation
    if current_mode == PROGRAM_MODE:  # Ignore rotation if in EMULATE mode
        if encoder_direction == -1:
            current_dir.up()
        elif encoder_direction == 1:
            current_dir.down()

    # look for a press of the rotary encoder switch press, with debouncing
    button.update()
    if button.fell:
        if current_mode == EMULATE_MODE:
            program()
        elif is_binary_name(current_dir.selected_filename):
            emulate()
        else:
            current_dir = current_dir.click()
```

Directories

The DirectoryNode class provides the ability to navigate a file system. It was written with an SD filesystem in mind, but just relies on the `os` module so should work with any storage that `os` uses.

The class provides public properties to get the name and full path of the selected file, as well as methods to force a screen update, move up and down in the displayed list, and do the appropriate thing in response to a selection event (in this case pushing the rotary encoder's switch).

When you create the root level instance, you provide an oled instance to display on, as well as the name of the root (which defaults to "/").

```
i2c = busio.I2C(board.SCL, board.SDA)
oled = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
...
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")
...
current_dir = DirectoryNode(oled, name="/sd")
current_dir.force_update()
```

Use of the `DirectoryNode` is straightforward as well, We can call `up()`, `down()`, and `click()` in response to activity on the rotary encoder, the `DirectoryNode` instance takes care of maintaining the display:

```
if current_mode == PROGRAM_MODE:      # Ignore rotation if in EMULATE mode
        if encoder_direction == -1:
            current_dir.up()
        elif encoder_direction == 1:
            current_dir.down()

    # look for a press of the rotary encoder switch press, with debouncing
    button.update()
    if button.fell:
        if current_mode == EMULATE_MODE:
            program()
        elif is_binary_name(current_dir.selected_filename):
            emulate()
        else:
            current_dir = current_dir.click()
```

Notice that the `click()` method returns a `DirectoryNode` which is assigned to the current node. That's because clicking could result in moving to a child or parent directory.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
The MIT License (MIT)

Copyright (c) 2018 Dave Astels

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
```

```
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

--------------------------------------------------------------------------------

Manage a directory in the file system.
"""

import os


class DirectoryNode:
    """Display and navigate the SD card contents"""

    def __init__(self, display, parent=None, name="/"):
        """Initialize a new instance.
           :param adafruit_ssd1306.SSD1306 on: the OLED instance to display on
           :param DirectoryNode below: optional parent directory node
           :param string named: the optional name of the new node
        """
        self.display = display
        self.parent = parent
        self.name = name
        self.files = []
        self.top_offset = 0
        self.old_top_offset = -1
        self.selected_offset = 0
        self.old_selected_offset = -1

    def __cleanup(self):
        """Dereference things for speedy gc."""
        self.display = None
        self.parent = None
        self.name = None
        self.files = None
        return self

    @staticmethod
    def __is_dir(path):
        """Determine whether a path identifies a machine code bin file.
           :param string path: path of the file to check
        """
        if path[-2:] == "..":
            return False
        try:
            os.listdir(path)
            return True
        except OSError:
            return False

    @staticmethod
    def __sanitize(name):
        """Nondestructively strip off a trailing slash, if any, and return the
result.
           :param string name: the filename
        """
        if name[-1] == "/":
            return name[:-1]
        return name
    # pylint: disable=protected-access
    def __path(self):
        """Return the result of recursively follow the parent links, building a full
           path to this directory."""
        if self.parent:
            return self.parent.__path() + os.sep + self.__sanitize(self.name)
        return self.__sanitize(self.name)

    def __make_path(self, filename):
        """Return a full path to the specified file in this directory.
           :param string filename: the name of the file in this directory
         """
        return self.__path() + os.sep + filename

    def __number_of_files(self):
        """The number of files in this directory, including the ".." for the parent
            directory if this isn't the top directory on the SD card."""
        self.__get_files()
        return len(self.files)

    def __get_files(self):
        """Return a list of the files in this directory.
           If this is not the top directory on the SD card, a
           ".." entry is the first element.
           Any directories have a slash appended to their name."""
        if len(self.files) == 0:
            self.files = os.listdir(self.__path())
            self.files.sort()
            if self.parent:
                self.files.insert(0, "..")
            for index, name in enumerate(self.files, start=1):
                if self.__is_dir(self.__make_path(name)):
                    self.files[index] = name + "/"

    def __update_display(self):
        """Update the displayed list of files if required."""
        if self.top_offset != self.old_top_offset:
            self.__get_files()
            self.display.fill(0)
            min_offset = min(self.top_offset + 4, self.__number_of_files())

            for i in range(self.top_offset, min_offset):
                self.display.text(self.files[i], 10, (i - self.top_offset) * 8)
            self.display.show()
            self.old_top_offset = self.top_offset

    def __update_selection(self):
        """Update the selected file lighlight if required."""
        if self.selected_offset != self.old_selected_offset:
            if self.old_selected_offset > -1:
                old_offset = (self.old_selected_offset - self.top_offset) * 8

                self.display.text(">", 0, old_offset, 0)

            new_offset = (self.selected_offset - self.top_offset) * 8
            self.display.text(">", 0, new_offset, 1)
            self.display.show()
            self.old_selected_offset = self.selected_offset

    @staticmethod
    def __is_directory_name(filename):
        """Is a filename the name of a directory.
           :param string filename: the name of the file
        """
        return filename[-1] == '/'

    @property
    def selected_filename(self):
        """The name of the currently selected file in this directory."""
        self.__get_files()
        return self.files[self.selected_offset]

    @property
    def selected_filepath(self):
        """The full path of the currently selected file in this directory."""
        return self.__make_path(self.selected_filename)

    def force_update(self):
        """Force an update of the file list and selected file highlight."""
        self.old_selected_offset = -1
        self.old_top_offset = -1
        self.__update_display()
        self.__update_selection()

    def down(self):
        """Move down in the file list if possible, adjusting the selected file
        indicator and scrolling the display as required."""
        if self.selected_offset < self.__number_of_files() - 1:
            self.selected_offset += 1
            if self.selected_offset == self.top_offset + 4:
                self.top_offset += 1
                self.__update_display()
        self.__update_selection()

    def up(self):
        """Move up in the file list if possible, adjusting the selected file
indicator
           and scrolling the display as required."""
        if self.selected_offset > 0:
            self.selected_offset -= 1
            if self.selected_offset < self.top_offset:
                self.top_offset -= 1
                self.__update_display()
        self.__update_selection()

    def click(self):
        """Handle a selection and return the new current directory.
           If the selected file is the parent, i.e. "..", return to the parent
           directory.
           If the selected file is a directory, go into it."""
        if self.selected_filename == "..":
            if self.parent:
                p = self.parent
                p.force_update()
                self.__cleanup()
                return p
        elif self.__is_directory_name(self.selected_filename):
            new_node = DirectoryNode(
                self.display, self, self.selected_filename)
            new_node.force_update()
            return new_node
        return self
```

Emulator

The emulator class manages the emulator circuit. Its public interface is simple enough:

- enter program mode, to allow the RAM to be loaded
- enter emulation mode, giving control of the emulator RAM to the host

- load the emulator RAM with data

The `load_ram` method takes care of the controlling the the address counters and multiplexers. By decomposing into a handful of private methods, the class is quite straight forward:

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
The MIT License (MIT)

Copyright (c) 2018 Dave Astels

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

--------------------------------------------------------------------------------
Manage the emulator hardware.
"""

import adafruit_mcp230xx
import digitalio

# control pin values

PROGRAMMER_USE = False
EMULATE_USE = True

WRITE_ENABLED = False
WRITE_DISABLED = True

CHIP_ENABLED = False
CHIP_DISABLED = True

CLOCK_ACTIVE = False
CLOCK_INACTIVE = True

RESET_INACTIVE = False
RESET_ACTIVE = True

LED_OFF = False
LED_ON = True

ENABLE_HOST_ACCESS = False
DISABLE_HOST_ACCESS = True
```

```
class Emulator:
    """Handle all interaction with the emulator circuit."""

    def __init__(self, i2c):
        self.mcp = adafruit_mcp230xx.MCP23017(i2c)
        self.mcp.iodir = 0x0000  # Make all pins outputs

        # Configure the individual control pins

        self.mode_pin = self.mcp.get_pin(8)
        self.mode_pin.direction = digitalio.Direction.OUTPUT
        self.mode_pin.value = PROGRAMMER_USE

        self.write_pin = self.mcp.get_pin(9)
        self.write_pin.direction = digitalio.Direction.OUTPUT
        self.write_pin.value = WRITE_DISABLED

        self.chip_select_pin = self.mcp.get_pin(10)
        self.chip_select_pin.direction = digitalio.Direction.OUTPUT
        self.chip_select_pin.value = CHIP_DISABLED

        self.address_clock_pin = self.mcp.get_pin(11)
        self.address_clock_pin.direction = digitalio.Direction.OUTPUT
        self.address_clock_pin.value = CLOCK_INACTIVE

        self.clock_reset_pin = self.mcp.get_pin(12)
        self.clock_reset_pin.direction = digitalio.Direction.OUTPUT
        self.clock_reset_pin.value = RESET_INACTIVE

        self.led_pin = self.mcp.get_pin(13)
        self.led_pin.direction = digitalio.Direction.OUTPUT
        self.led_pin.value = False

    def __pulse_write(self):
        self.write_pin.value = WRITE_ENABLED
        self.write_pin.value = WRITE_DISABLED

    def __deactivate_ram(self):
        self.chip_select_pin.value = CHIP_DISABLED

    def __activate_ram(self):
        self.chip_select_pin.value = CHIP_ENABLED

    def __reset_address_counter(self):
        self.clock_reset_pin.value = RESET_ACTIVE
        self.clock_reset_pin.value = RESET_INACTIVE

    def __advance_address_counter(self):
        self.address_clock_pin.value = CLOCK_ACTIVE
        self.address_clock_pin.value = CLOCK_INACTIVE

    def __output_on_port_a(self, data_byte):
        """A hack to get around the limitation of the 23017
        library to use 8-bit ports"""
        self.mcp.gpio = (self.mcp.gpio & 0xFF00) | (data_byte & 0x00FF)

    def enter_program_mode(self):
        """Enter program mode, allowing loading of the emulator RAM."""
        self.mode_pin.value = PROGRAMMER_USE
        self.led_pin.value = LED_OFF

    def enter_emulate_mode(self):
        """Enter emulate mode, giving control of the emulator
        ram to the host."""
        self.mode_pin.value = EMULATE_USE
        self.led_pin.value = LED_ON

    def load_ram(self, code):
        """Load the emulator RAM. Automatically switched to program mode.
        :param [byte] code: the list of bytes to load into the emulator RAM
        """
        self.enter_program_mode()
        self.__reset_address_counter()
        for data_byte in code:
            self.__output_on_port_a(data_byte)
            self.__activate_ram()
            self.__pulse_write()
            self.__deactivate_ram()
            self.__advance_address_counter()
```

Series Index

1. Binary, Boolean, and Logic ()
2. Some Tools ()
3. Combinational Circuits ()
4. Sequential Circuits ()
5. Memories ()
6. An EPROM Emulator ()
7. MCUs... how do they work? ()