

Explore Hardware

(old and new)

w/CircuitPython





Hi

Scott Shawcroft aka *tannewt*

Freelance software engineer

Project lead on CircuitPython for Adafruit

2009 UW Grad in Computer Engineering

Worked on Google Maps 2009 - 2015

Plan

- CircuitPython Demo
- Vertical slice of CircuitPython
 - Software
 - Hardware
- GameBoy Demo



CircuitPython Demo



CircuitPython

Code + Community



Code + Community

- Python is the easiest way to iterate on software
- CircuitPython code and toolchain travels with the device for ultimate hackability
- <https://github.com/adafruit/circuitpython>
- Built on MicroPython



Code + Community

- Code of Conduct
- Active community on Discord and GitHub
- 190+ CircuitPython-compatible libraries
- 80+ Supported boards



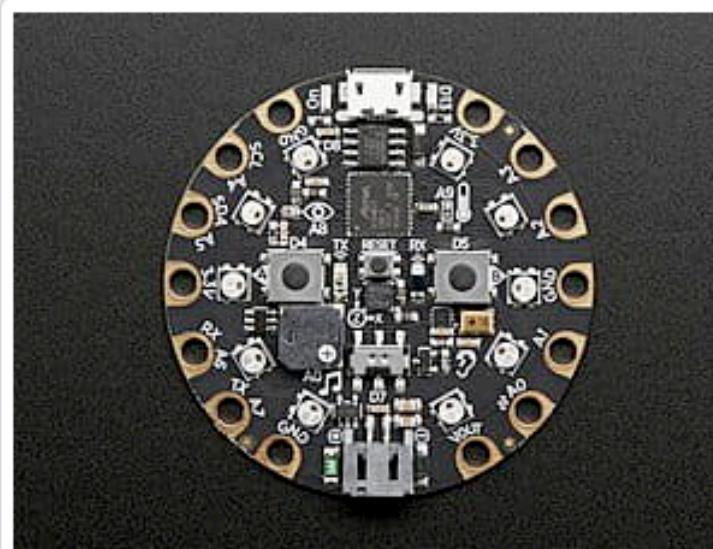
Downloads

Search for CircuitPython boards



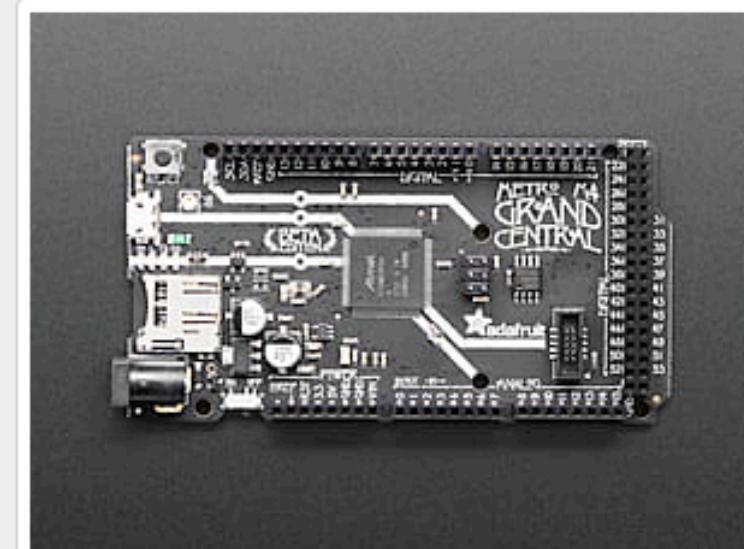
PyPortal

By Adafruit



Circuit Playground Express

By Adafruit



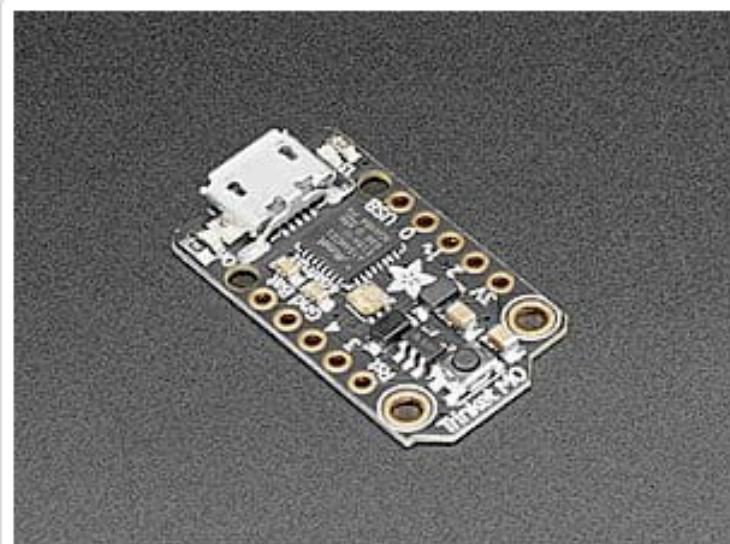
Grand Central M4 Express

By Adafruit

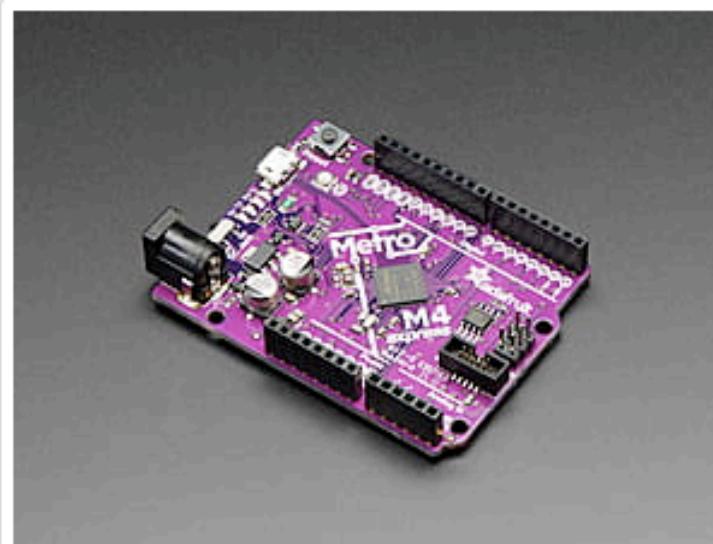


NeoTrellis M4

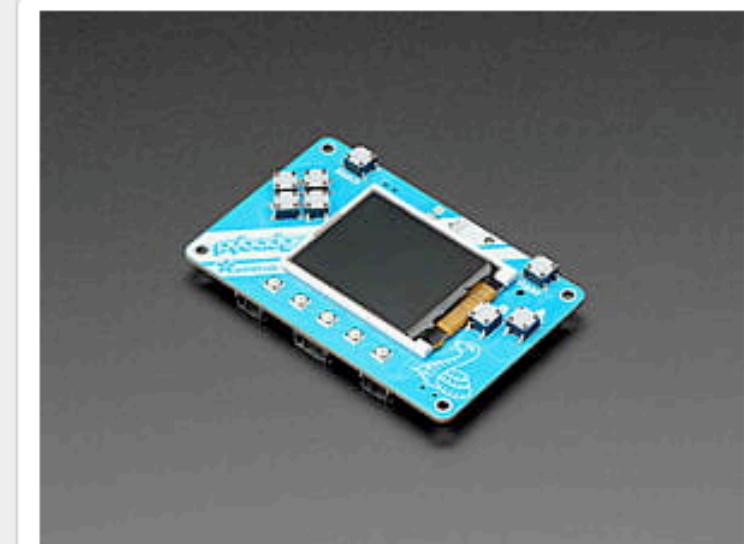
By Adafruit



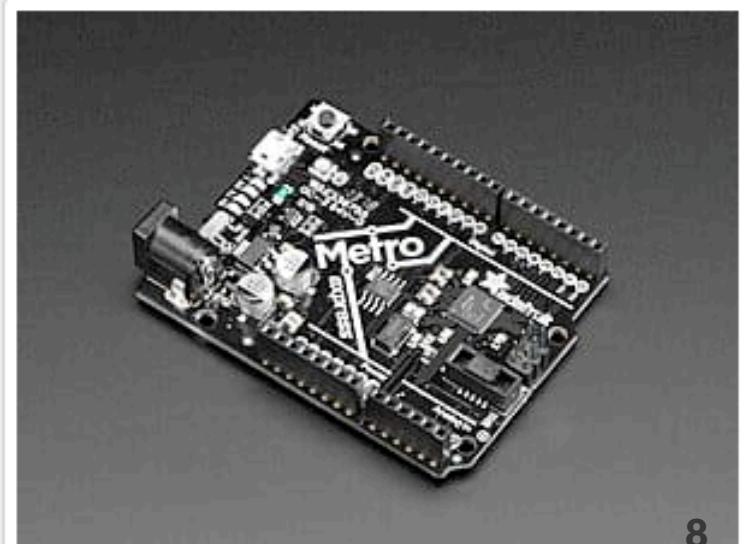
Trinket M0



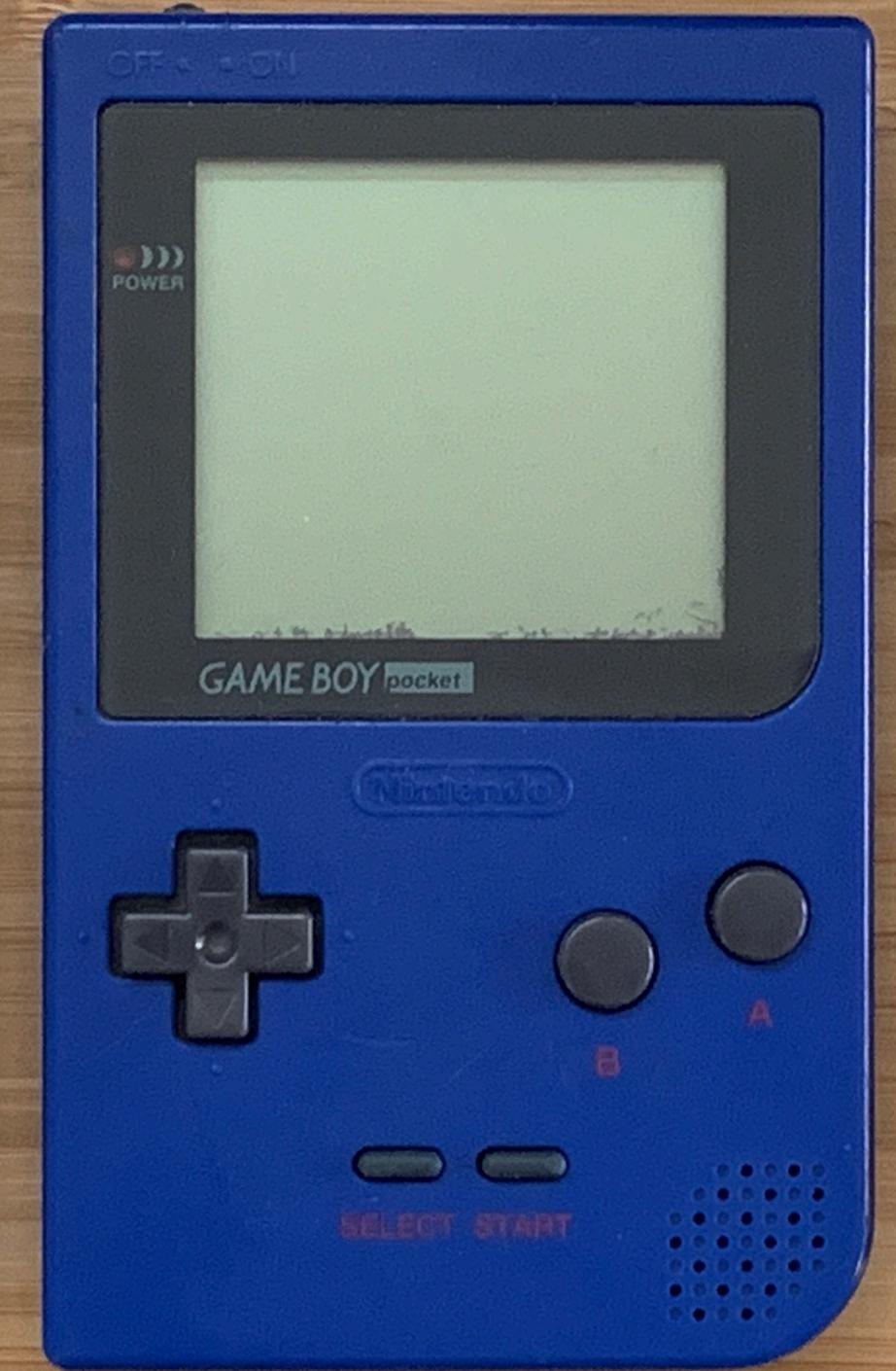
Metro M4 Express

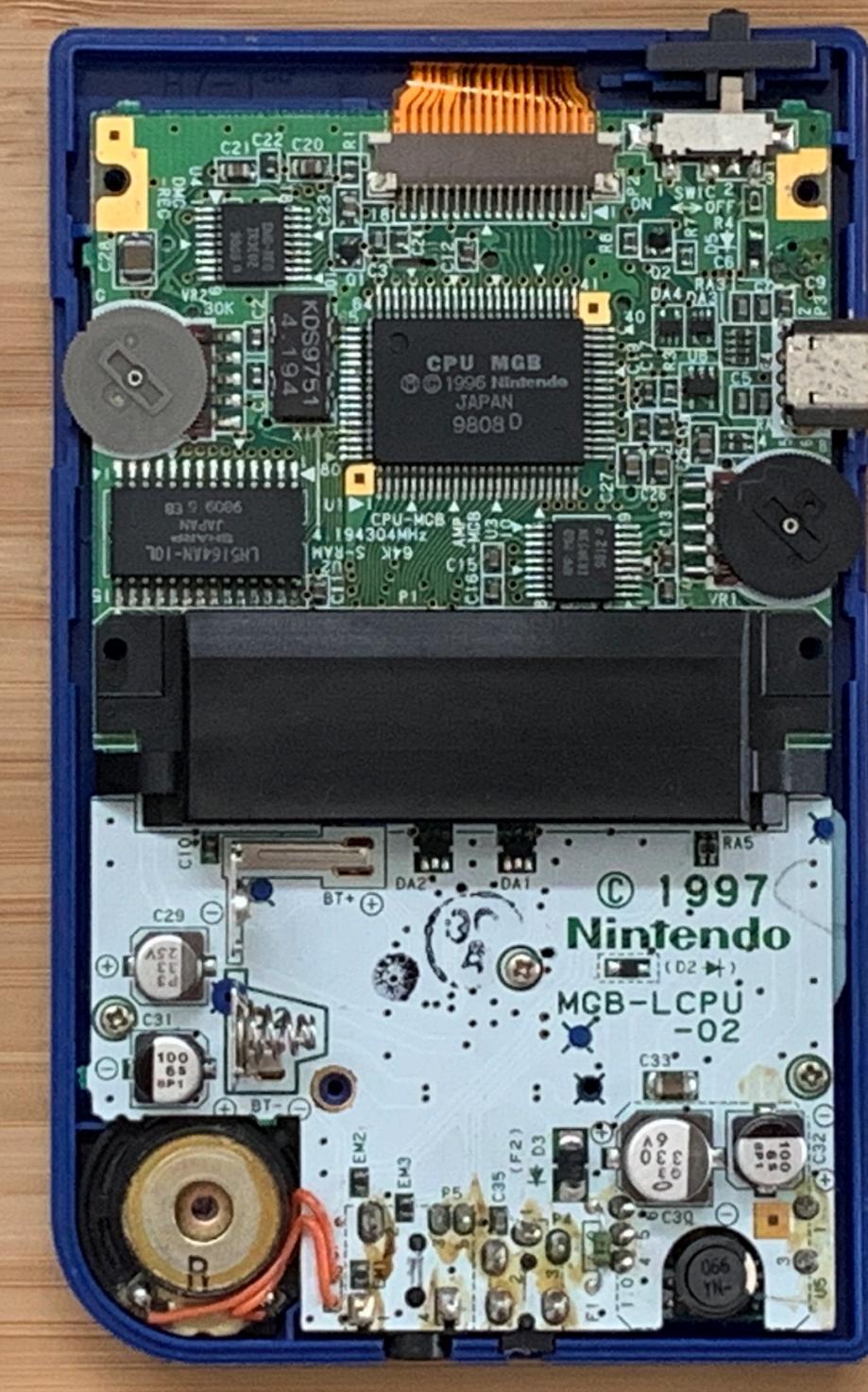
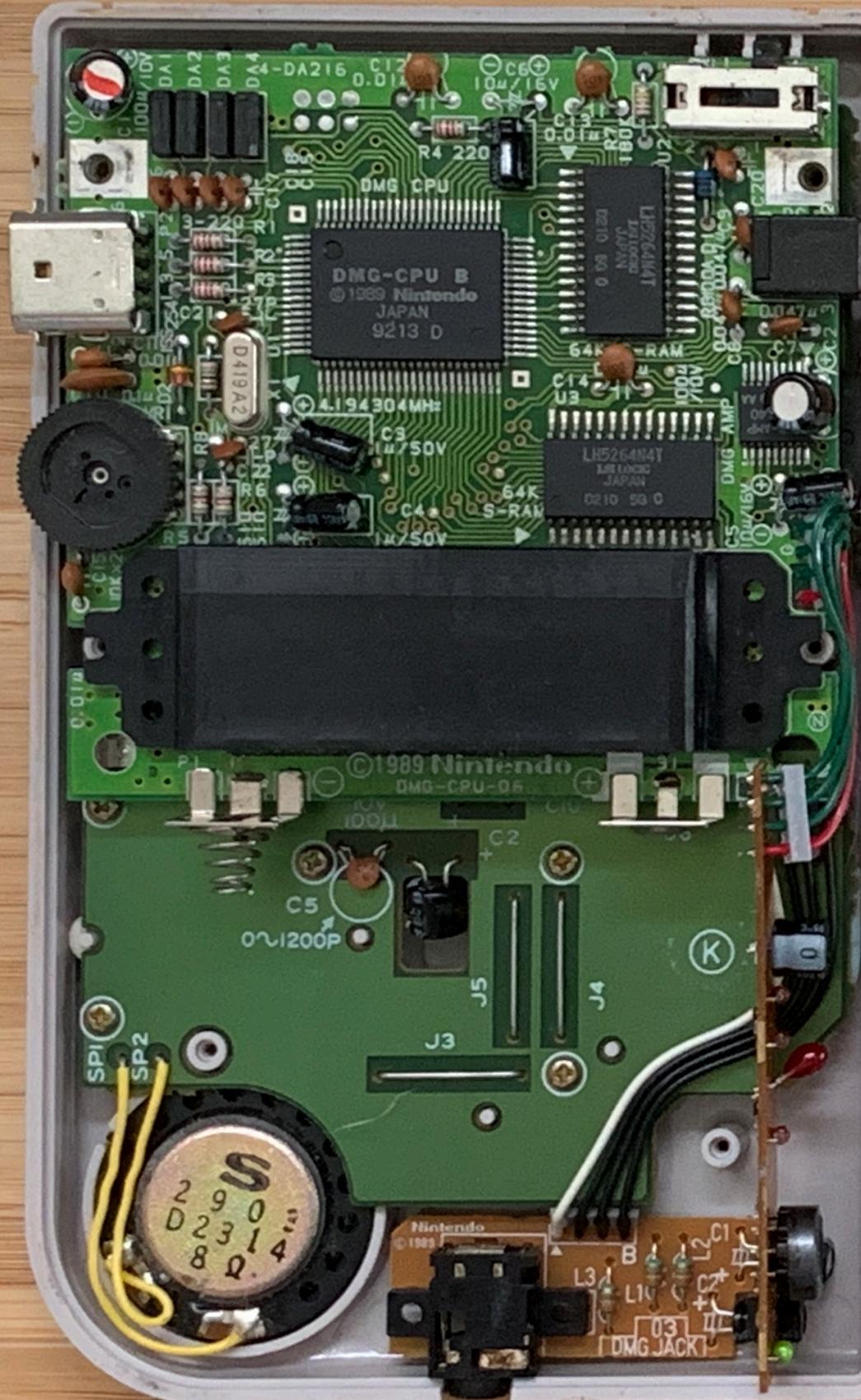


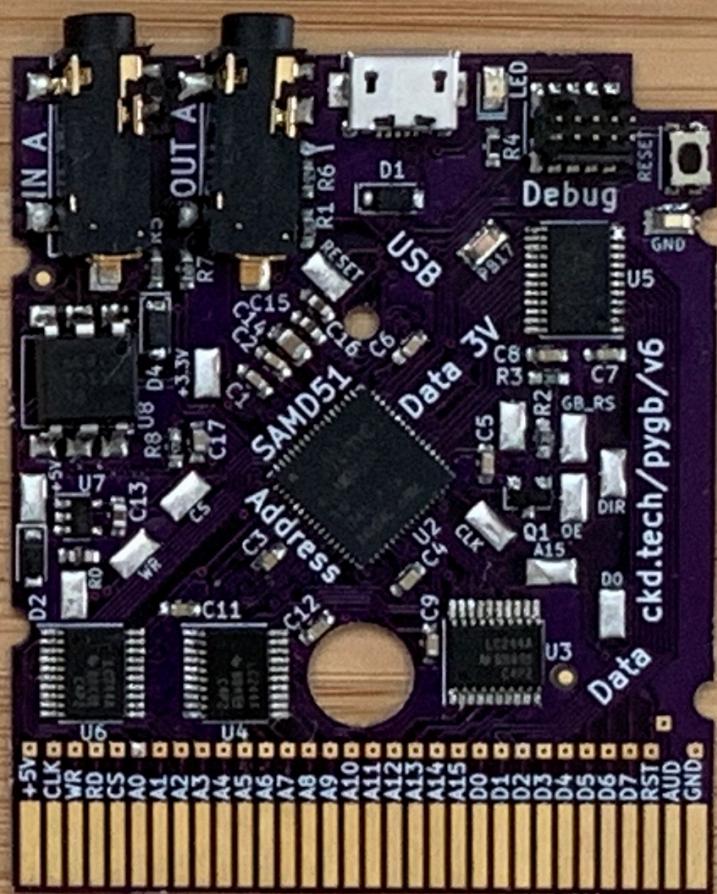
PyBadge



Metro M0 Express



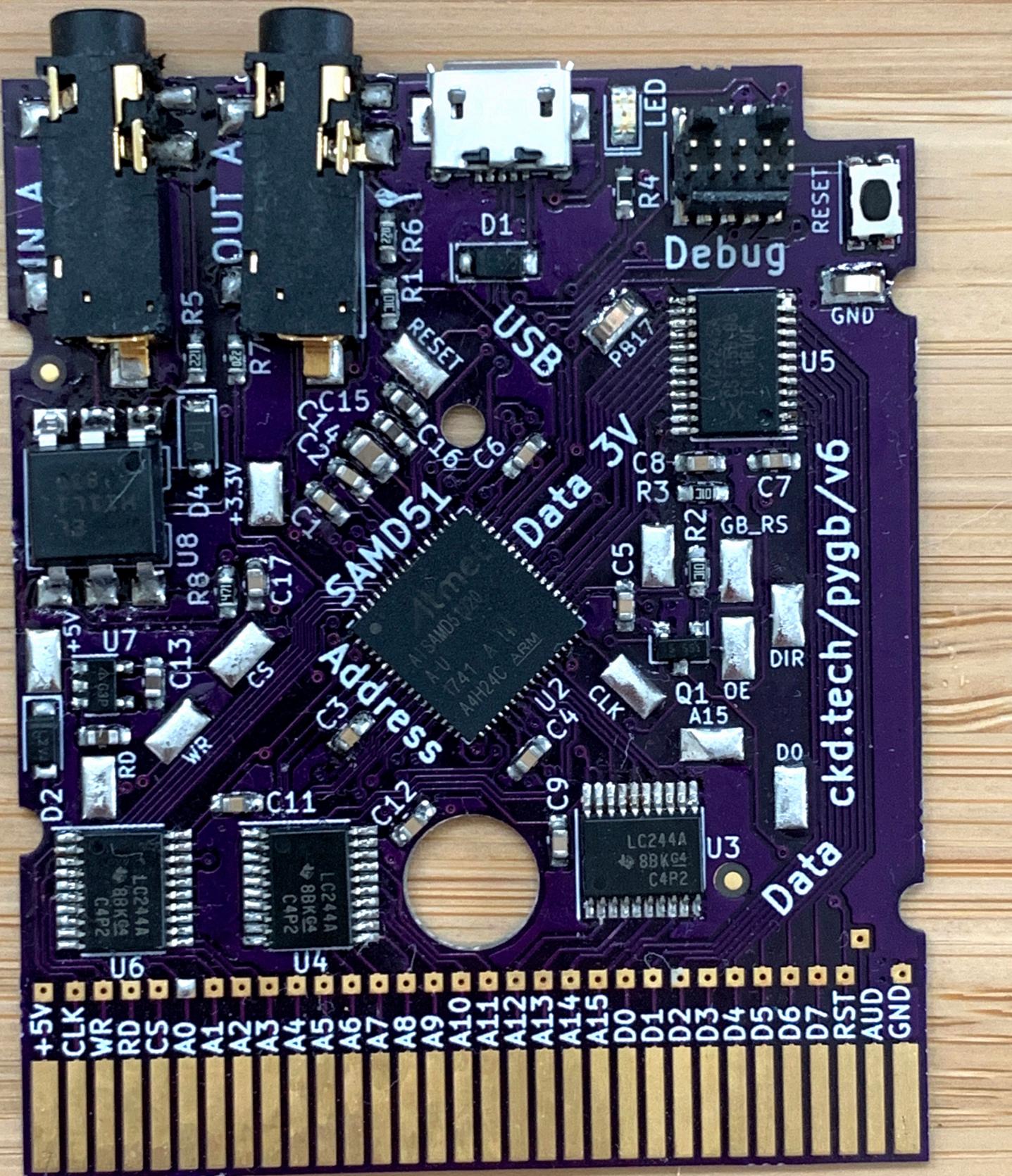




MCU as Cart

- DHole in late 2014 used an STM32F4 as a GameBoy cart.¹
- Respond with a byte on the data bus for every 1 MHz clock where the address is in cart range (0x0000-0x7fff)
- Chose to use the SAMD51 because it is 120 MHz and already had CircuitPython support

¹https://dhole.github.io/post/gameboy_cartridge_emu_1/



code.py

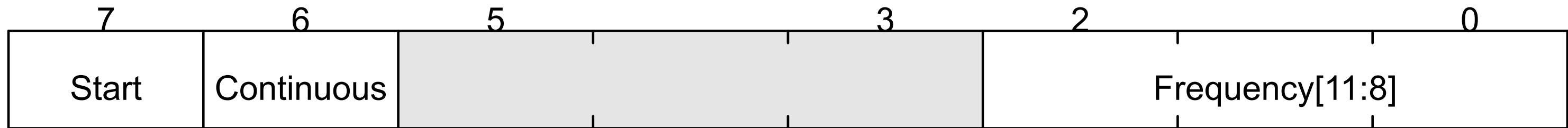
```
from adafruit_gameboy import gb

# Register's are documented here:
# http://marc.rawer.de/Gameboy/Docs/GBCPUMan.pdf

# Voice 1
#     - Bit 7 - Start
#     - Bit 6 - Counter/consecutive
#     - Bit 2-0 - Top 3 frequency bits
gb[0xff14] = 0b1000111 # 0x87
```



The magic bit in `0xff14`



User code - code.py

```
from adafruit_gameboy import gb

# Register's are documented here:
# http://marc.rawer.de/Gameboy/Docs/GBCPUMan.pdf

offset = 0xff10 # Voice 1

# Voice 1
#   - Bit 7 - Start
#   - Bit 6 - Counter/consecutive
#   - Bit 2-0 - Top 3 frequency bits
gb[offset + 4] = 0b10000111 # 0x87
```



User code - code.py

```
from adafruit_gameboy import gb

# Register's are documented here:
# http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf

offset = 0xff10 # Voice 1

# Voice 1
#   - Bit 7 - Start
#   - Bit 6 - Counter/consecutive
#   - Bit 2-0 - Top 3 frequency bits
gb[offset + 4] = 0b10000111 # 0x87
```



Python library - adafruit_gameboy.py

```
import gbio

class GameBoy:
    def __init__(self):
        self._byte_buf = bytearray(6)
        self._byte_buf[1] = 0x0e # Load next value into C
        self._byte_buf[3] = 0x3e # Load next value into A
        self._byte_buf[5] = 0xe2 # Load A into 0xff00 + C

    def __setitem__(self, index, value):
        if index > 0xff00:
            self._byte_buf[2] = index - 0xff00
            self._byte_buf[4] = value
            gbio.queue_commands(self._byte_buf)
```



Python library - adafruit_gameboy.py

```
import gbio

class GameBoy:
    def __init__(self):
        self._byte_buf = bytearray(6)
        self._byte_buf[1] = 0x0e # Load next value into C
        self._byte_buf[3] = 0x3e # Load next value into A
        self._byte_buf[5] = 0xe2 # Load A into 0xff00 + C

    def __setitem__(self, index, value):
        if index > 0xff00:
            self._byte_buf[2] = index - 0xff00
            self._byte_buf[4] = value
            gbio.queue_commands(self._byte_buf)
```



Python library - adafruit_gameboy.py

```
import gbio

class GameBoy:
    def __init__(self):
        self._byte_buf = bytearray(6)
        self._byte_buf[1] = 0x0e # Load next value into C
        self._byte_buf[3] = 0x3e # Load next value into A
        self._byte_buf[5] = 0xe2 # Load A into 0xff00 + C

    def __setitem__(self, index, value):
        if index > 0xff00:
            self._byte_buf[2] = index - 0xff00
            self._byte_buf[4] = value
        gbio.queue_commands(self._byte_buf)
```



Shared C API - module

```
// shared-bindings/gbio/__init__.c

STATIC const mp_rom_map_elem_t gbio_module_globals_table[] = {
    { MP_ROM_QSTR(MP_QSTR___name__), MP_ROM_QSTR(MP_QSTR_gbio) },
    { MP_ROM_QSTR(MP_QSTR_queue_commands), MP_ROM_PTR(&gbio_queue_commands_obj) },
    { MP_ROM_QSTR(MP_QSTR_queue_vblank_commands), MP_ROM_PTR(&gbio_queue_vblank_commands_obj) },
};

STATIC MP_DEFINE_CONST_DICT(gbio_module_globals, gbio_module_globals_table);

const mp_obj_module_t gbio_module = {
    .base = { &mp_type_module },
    .globals = (mp_obj_dict_t*)&gbio_module_globals,
};
```



Shared C API - module

```
// shared-bindings/gbio/__init__.c

STATIC const mp_rom_map_elem_t gbio_module_globals_table[] = {
    { MP_ROM_QSTR(MP_QSTR___name__), MP_ROM_QSTR(MP_QSTR_gbio) },
    { MP_ROM_QSTR(MP_QSTR_queue_commands), MP_ROM_PTR(&gbio_queue_commands_obj) },
    { MP_ROM_QSTR(MP_QSTR_queue_vblank_commands), MP_ROM_PTR(&gbio_queue_vblank_commands_obj) },
};

STATIC MP_DEFINE_CONST_DICT(gbio_module_globals, gbio_module_globals_table);

const mp_obj_module_t gbio_module = {
    .base = { &mp_type_module },
    .globals = (mp_obj_dict_t*)&gbio_module_globals,
};
```



Shared C API - module

```
// shared-bindings/gbio/__init__.c

STATIC const mp_rom_map_elem_t gbio_module_globals_table[] = {
    { MP_ROM_QSTR(MP_QSTR___name__), MP_ROM_QSTR(MP_QSTR_gbio) },
    { MP_ROM_QSTR(MP_QSTR_queue_commands), MP_ROM_PTR(&gbio_queue_commands_obj) },
    { MP_ROM_QSTR(MP_QSTR_queue_vblank_commands), MP_ROM_PTR(&gbio_queue_vblank_commands_obj) },
};

STATIC MP_DEFINE_CONST_DICT(gbio_module_globals, gbio_module_globals_table);

const mp_obj_module_t gbio_module = {
    .base = { &mp_type_module },
    .globals = (mp_obj_dict_t*)&gbio_module_globals,
};
```



Shared C API - function

```
// shared-bindings/gbio/__init__.c

//| .. method:: queue_commands(instructions)
//|
//| These instructions are run immediately and this function will block until they finish.
//|
STATIC mp_obj_t gbio_queue_commands(mp_obj_t instructions){
    mp_buffer_info_t bufinfo;
    if (!mp_get_buffer(instructions, &bufinfo, MP_BUFFER_READ)) {
        mp_raise_TypeError(translate("buffer must be a bytes-like object"));
    } else if (bufinfo.typecode != 'B' && bufinfo.typecode != BYTEARRAY_TYPECODE) {
        mp_raise_ValueError(translate("instruction buffer must be a bytearray or array of type 'B'"));
    }
    common_hal_gbio_queue_commands(bufinfo.buf, bufinfo.len);
    return mp_const_none;
}
MP_DEFINE_CONST_FUN_OBJ_1(gbio_queue_commands_obj, gbio_queue_commands);
```



Shared C API - function

```
// shared-bindings/gbio/__init__.c

//| .. method:: queue_commands(instructions)
//|
//| These instructions are run immediately and this function will block until they finish.
//|
STATIC mp_obj_t gbio_queue_commands(mp_obj_t instructions){
    mp_buffer_info_t bufinfo;
    if (!mp_get_buffer(instructions, &bufinfo, MP_BUFFER_READ)) {
        mp_raise_TypeError(translate("buffer must be a bytes-like object"));
    } else if (bufinfo.typecode != 'B' && bufinfo.typecode != BYTEARRAY_TYPECODE) {
        mp_raise_ValueError(translate("instruction buffer must be a bytearray or array of type 'B'"));
    }
    common_hal_gbio_queue_commands(bufinfo.buf, bufinfo.len);
    return mp_const_none;
}
MP_DEFINE_CONST_FUN_OBJ_1(gbio_queue_commands_obj, gbio_queue_commands);
```



Shared C API - function

```
// shared-bindings/gbio/__init__.c

//| .. method:: queue_commands(instructions)
//|
//| These instructions are run immediately and this function will block until they finish.
//|
STATIC mp_obj_t gbio_queue_commands(mp_obj_t instructions){
    mp_buffer_info_t bufinfo;
    if (!mp_get_buffer(instructions, &bufinfo, MP_BUFFER_READ)) {
        mp_raise_TypeError(translate("buffer must be a bytes-like object"));
    } else if (bufinfo.typecode != 'B' && bufinfo.typecode != BYTEARRAY_TYPECODE) {
        mp_raise_ValueError(translate("instruction buffer must be a bytearray or array of type 'B'"));
    }
    common_hal_gbio_queue_commands(bufinfo.buf, bufinfo.len);
    return mp_const_none;
}
MP_DEFINE_CONST_FUN_OBJ_1(gbio_queue_commands_obj, gbio_queue_commands);
```



Shared C API - function

```
// shared-bindings/gbio/__init__.c

//| .. method:: queue_commands(instructions)
//|
//| These instructions are run immediately and this function will block until they finish.
//|
STATIC mp_obj_t gbio_queue_commands(mp_obj_t instructions){
    mp_buffer_info_t bufinfo;
    if (!mp_get_buffer(instructions, &bufinfo, MP_BUFFER_READ)) {
        mp_raise_TypeError(translate("buffer must be a bytes-like object"));
    } else if (bufinfo.typecode != 'B' && bufinfo.typecode != BYTEARRAY_TYPECODE) {
        mp_raise_ValueError(translate("instruction buffer must be a bytearray or array of type 'B'"));
    }
    common_hal_gbio_queue_commands(bufinfo.buf, bufinfo.len);
    return mp_const_none;
}
MP_DEFINE_CONST_FUN_OBJ_1(gbio_queue_commands_obj, gbio_queue_commands);
```



Shared C API - function

```
// shared-bindings/gbio/__init__.c

//| .. method:: queue_commands(instructions)
//|
//| These instructions are run immediately and this function will block until they finish.
//|
STATIC mp_obj_t gbio_queue_commands(mp_obj_t instructions){
    mp_buffer_info_t bufinfo;
    if (!mp_get_buffer(instructions, &bufinfo, MP_BUFFER_READ)) {
        mp_raise_TypeError(translate("buffer must be a bytes-like object"));
    } else if (bufinfo.typecode != 'B' && bufinfo.typecode != BYTEARRAY_TYPECODE) {
        mp_raise_ValueError(translate("instruction buffer must be a bytearray or array of type 'B'"));
    }
    common_hal_gbio_queue_commands(bufinfo.buf, bufinfo.len);
    return mp_const_none;
}
MP_DEFINE_CONST_FUN_OBJ_1(gbio_queue_commands_obj, gbio_queue_commands);
```



Lowest level C

```
// ports/atmel-samd/common-hal/gbio/__init__.c

void common_hal_gbio_queue_commands(const uint8_t* buf, uint32_t len) {
    // Wait for a previous sequence to finish.

    uint32_t total_len = 0;

    memcpy(command_cache + 2, buf, len);
    total_len += len;

    // Start DMA and wait for it.
    DmacDescriptor* descriptor_out = dma_descriptor(dma_out_channel);
    descriptor_out->BTCTRL.reg |= DMAC_BTCTRL_VALID;
    descriptor_out->BTCNT.reg = total_len;
    descriptor_out->SRCADDR.reg = ((uint32_t) command_cache) + total_len;
    descriptor_out->DSTADDR.reg = (uint32_t)&PORT->Group[0].OUT.reg + 2;

    dma_enable_channel(dma_out_channel);

    // Wait for DMA
}
```



Lowest level C

```
// ports/atmel-samd/common-hal/gbio/__init__.c

void common_hal_gbio_queue_commands(const uint8_t* buf, uint32_t len) {
    // Wait for a previous sequence to finish.

    uint32_t total_len = 0;

    memcpy(command_cache + 2, buf, len);
    total_len += len;

    // Start DMA and wait for it.
    DmacDescriptor* descriptor_out = dma_descriptor(dma_out_channel);
    descriptor_out->BTCTRL.reg |= DMAC_BTCTRL_VALID;
    descriptor_out->BTCNT.reg = total_len;
    descriptor_out->SRCADDR.reg = ((uint32_t) command_cache) + total_len;
    descriptor_out->DSTADDR.reg = (uint32_t)&PORT->Group[0].OUT.reg + 2;

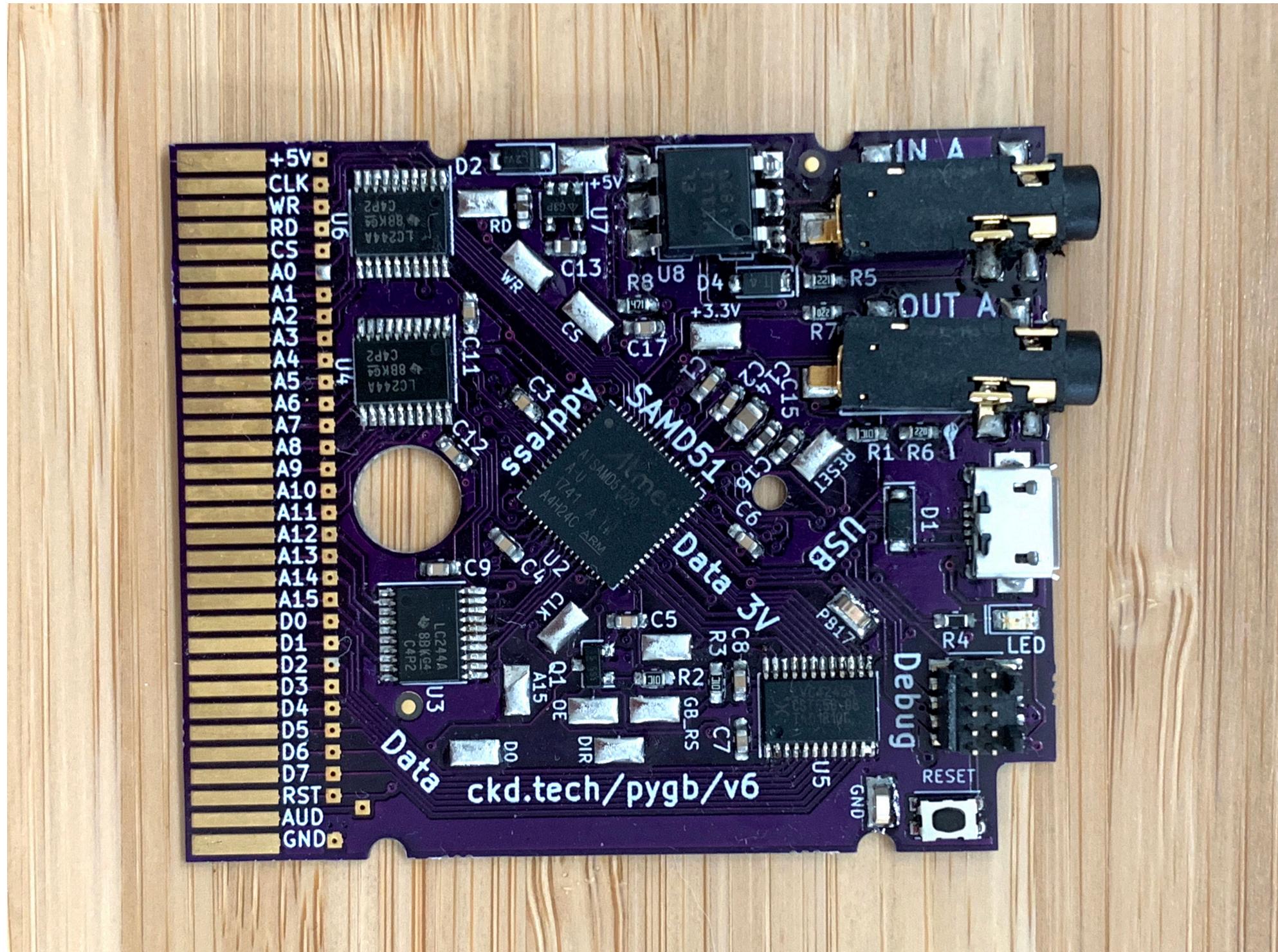
    dma_enable_channel(dma_out_channel);

    // Wait for DMA
}
```



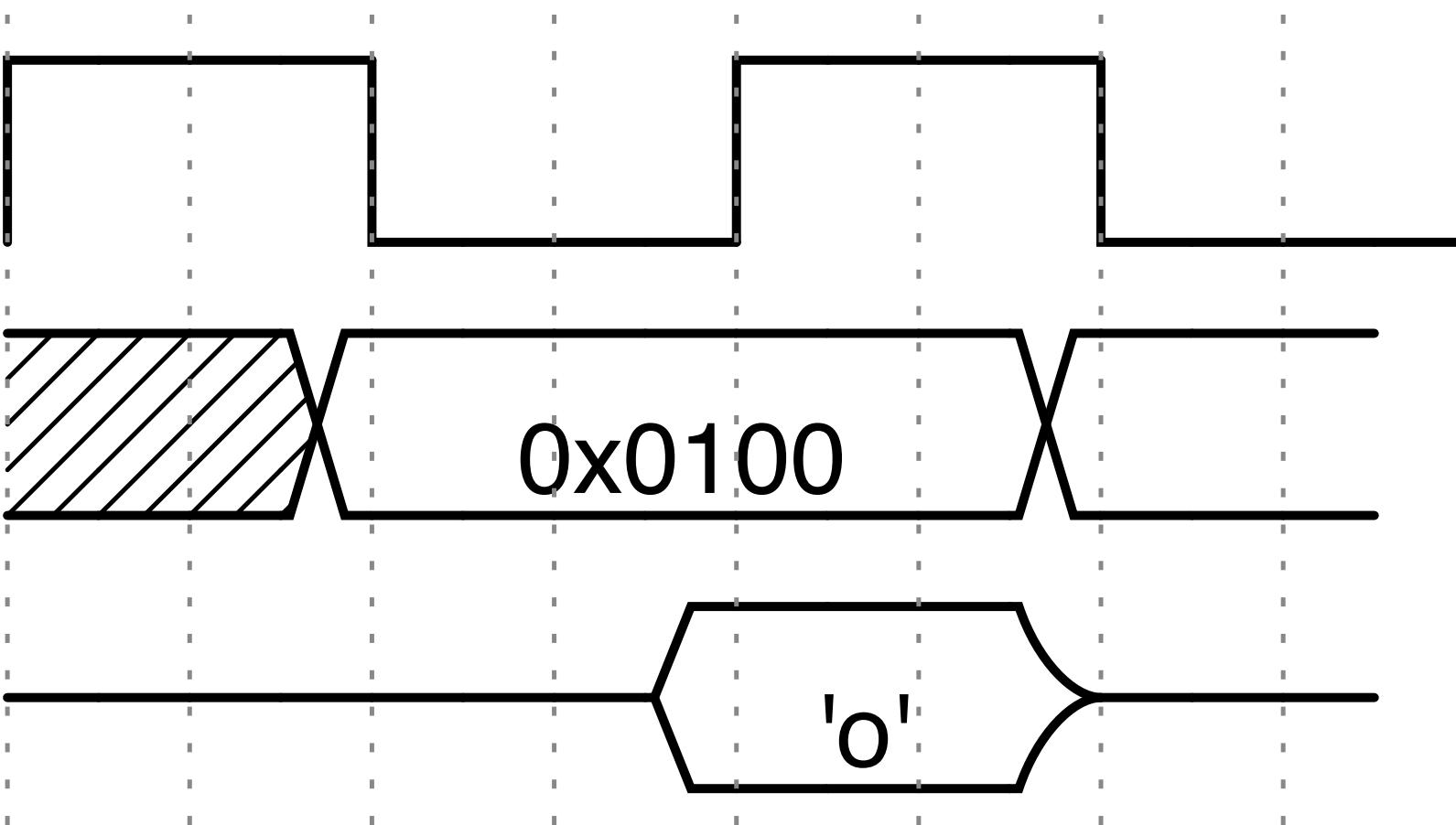
Wire



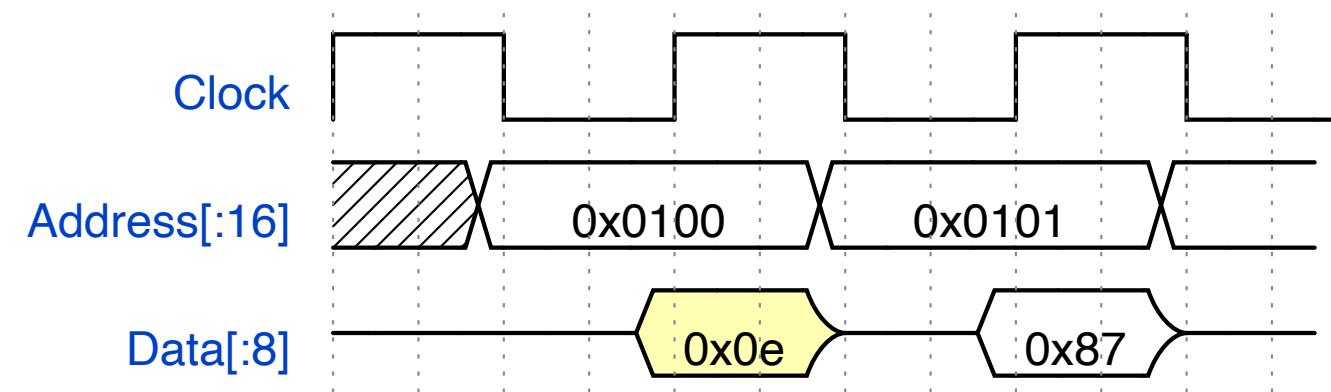


Timing Diagram

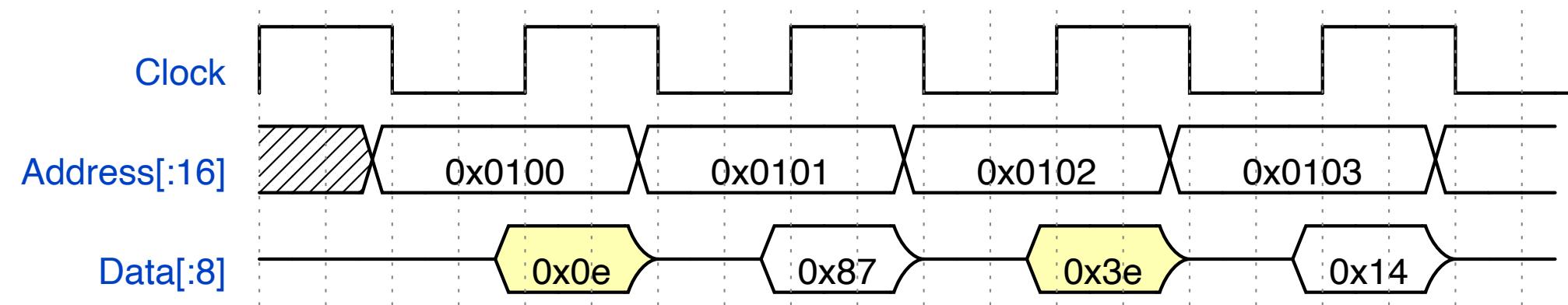
Clock
Address[:16]
Data[:8]



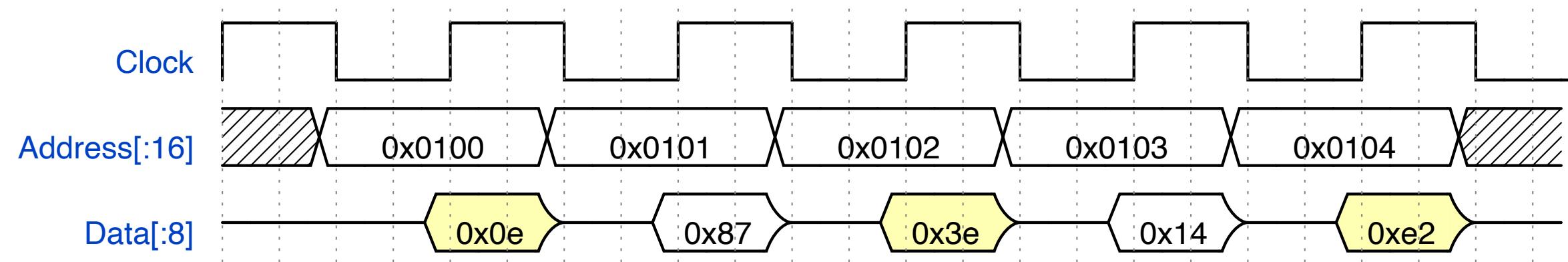
Load our new value into register A



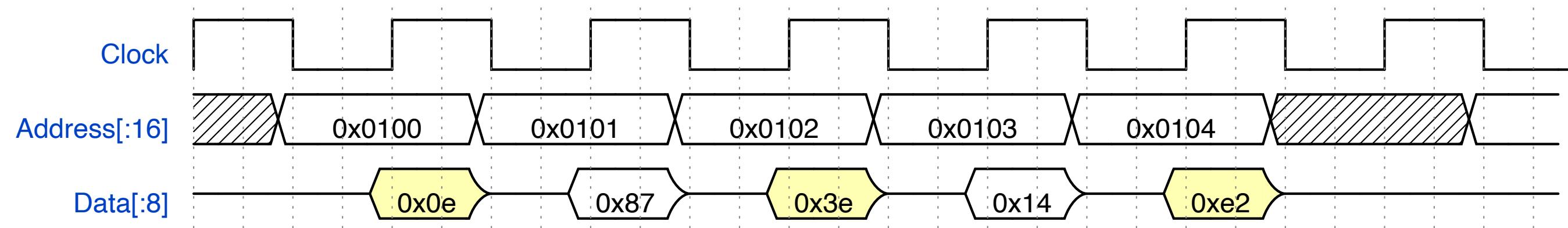
Load address into register C



Store A into 0xff00 + C



Continue



GameBoy Demo





https://media.ccc.de/v/33c3-8029-the_ultimate_game_boy_talk

Getting started

- Reach out on our Discord chat: <https://adafru.it/discord>
- Help us make CircuitPython the easiest way to learn to code and the best example of Python on hardware.



Contact

@tannewt

scott@adafruit.com

circuitpython.org

github.com/tannewt/presentations



Thank you!

