Explore Hardware

(old and new)

w/CircuitPython

Hi

Scott Shawcroft aka *tannewt*

Freelance software engineer

Project lead on CircuitPython for Adafruit

Plan

- Live demos!
  - GameBoy
- Vertical slice of CircuitPython
  - Software
  - Hardware

Demo

CircuitPython

Code + Community

Code + Community

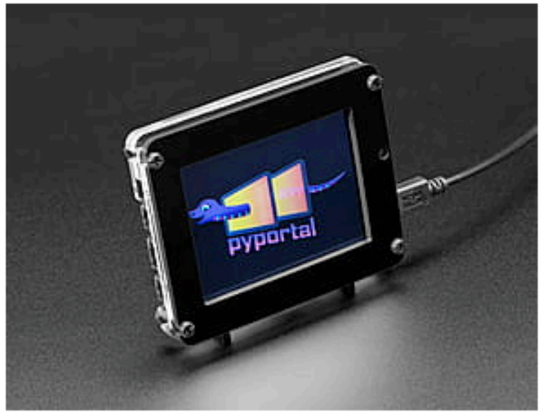
- Python is the easiest way to iterate on software
- CircuitPython code and toolchain travels with the device for ultimate hackability
- `https://github.com/adafruit/circuitpython`
- Built on MicroPython

Code **+ Community**

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

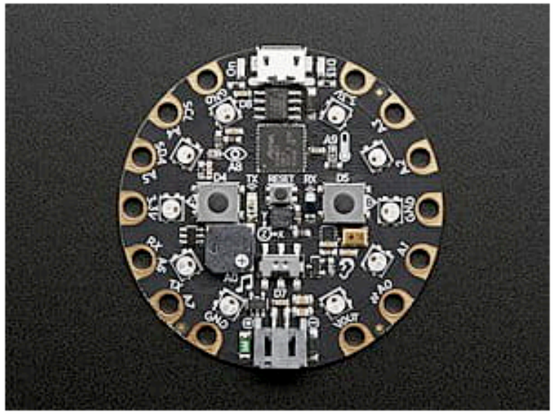
Downloads

Search for CircuitPython boards



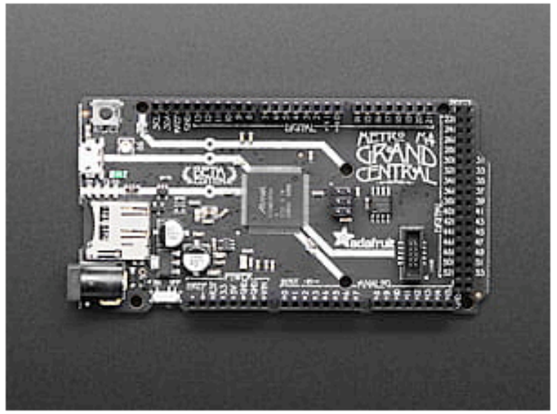
PyPortal

By Adafruit



Circuit Playground Express

By Adafruit

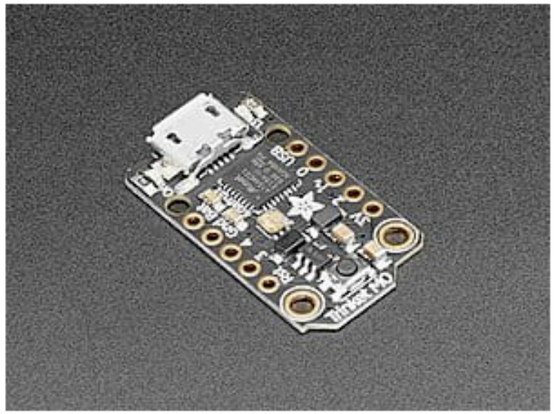


Grand Central M4 Express

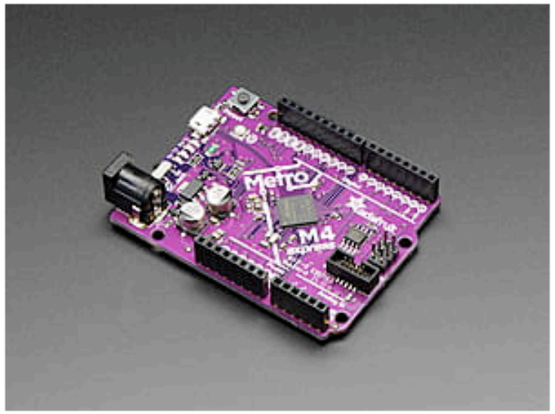
By Adafruit

NeoTrellis M4

By Adafruit

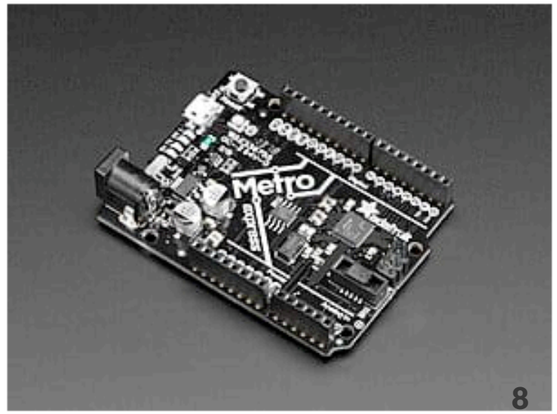


Trinket M0



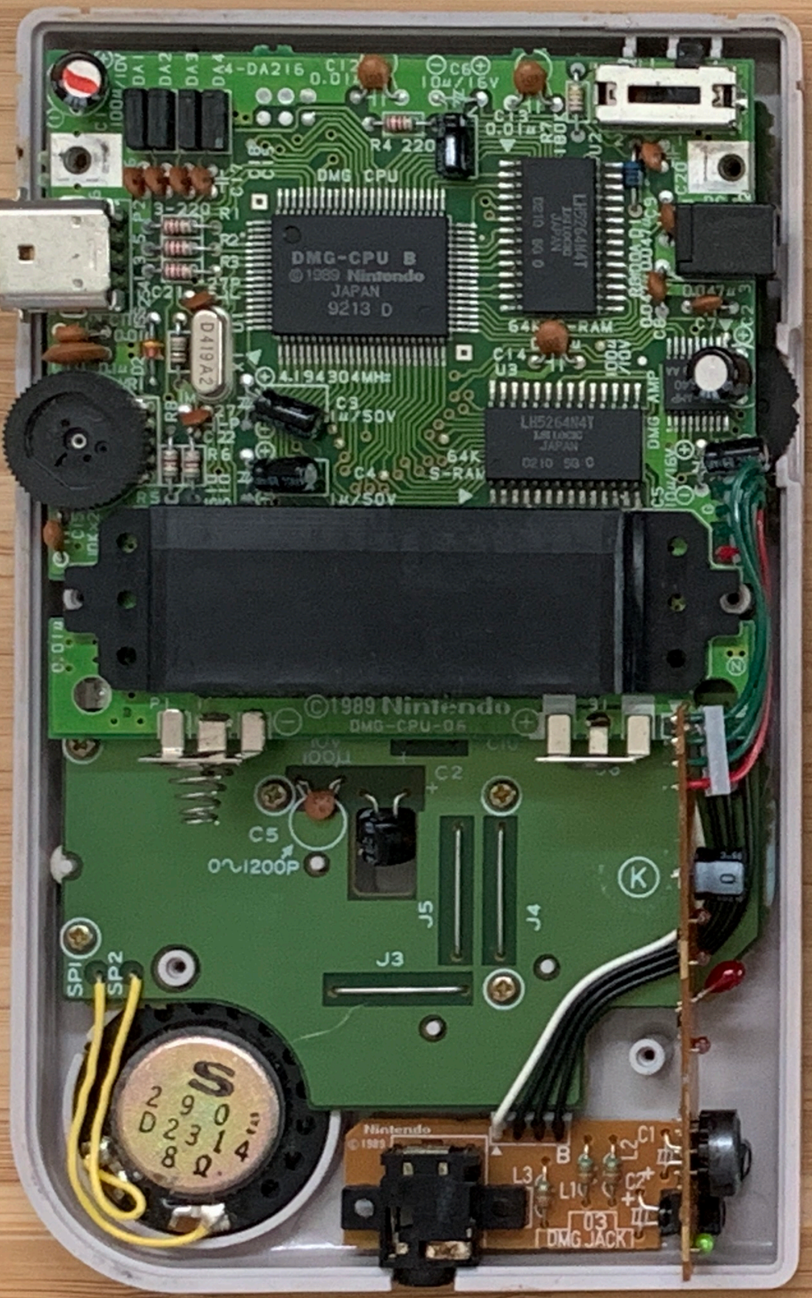
Metro M4 Express

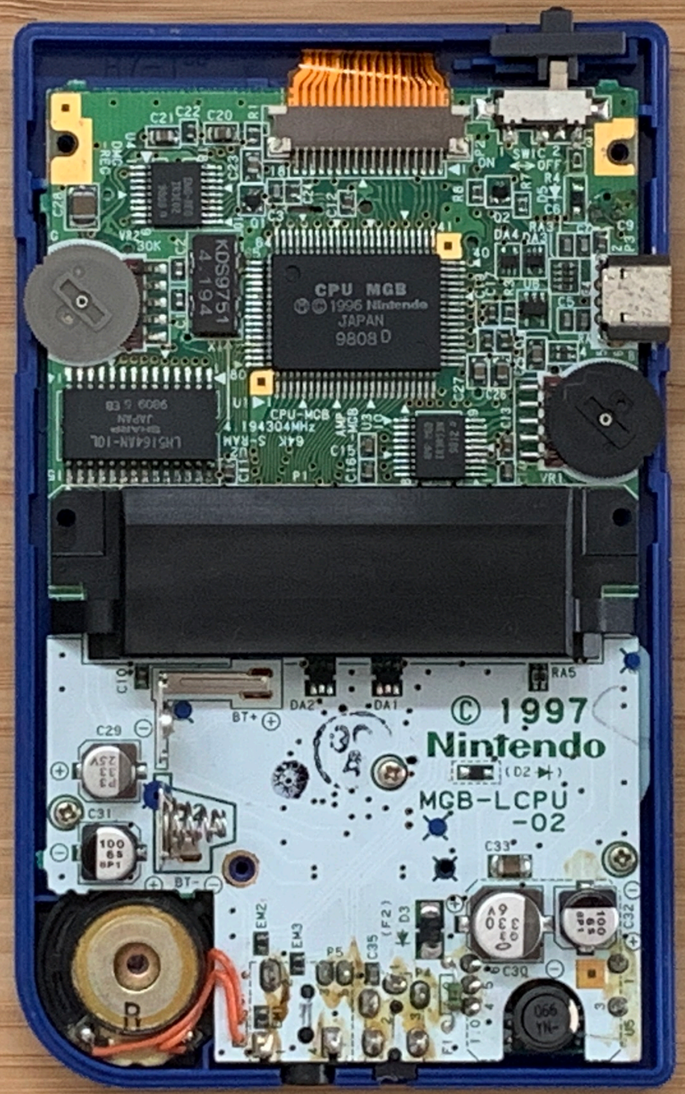
PyBadge



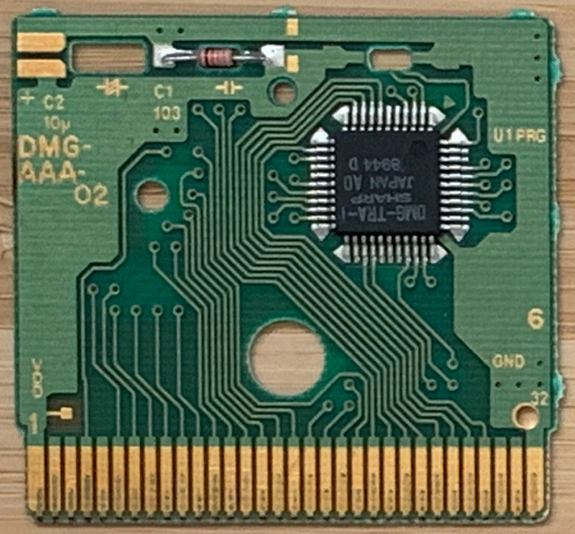
Metro M0 Express


DOT MATRIX WITH STEREO SOUND
BATTERY
Nintendo GAME BOY™
SELECT
START
B
A
POWER
GAME BOY pocket
SELECT START
POWER
GAME BOY COLOR

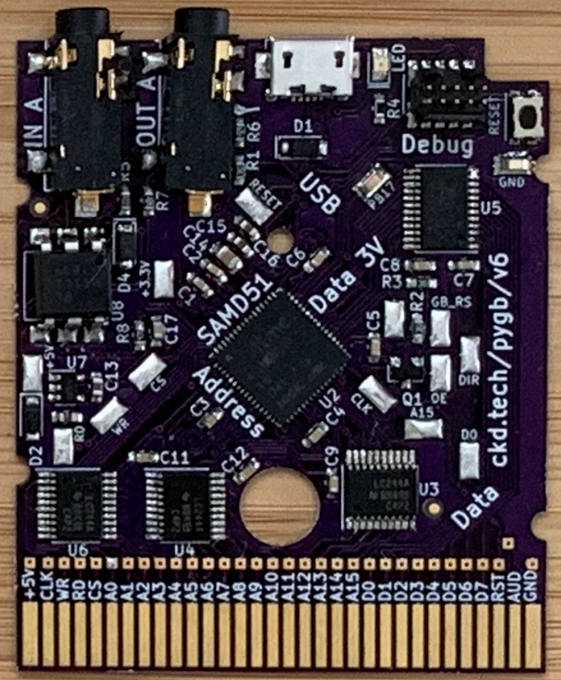

DMG CPU
DMG-CPU B
©1989 Nintendo
JAPAN
9213 D
4.194304MHz
64K S-RAM
©1989 Nintendo
DMG-CPU-06
C5
0~1200P
J5
J4
J3
SP1
SP2
DMG JACK


CPU MGB
©1996 Nintendo
JAPAN
9808 D
CPU-MGB
4.194304MHz
64K S-RAM
© 1997
Nintendo
MGB-LCPU
-02


CPU CGB D
©1998 Nintendo
JAPAN
0030 I
©1998
Nintendo
CGB-CPU-05

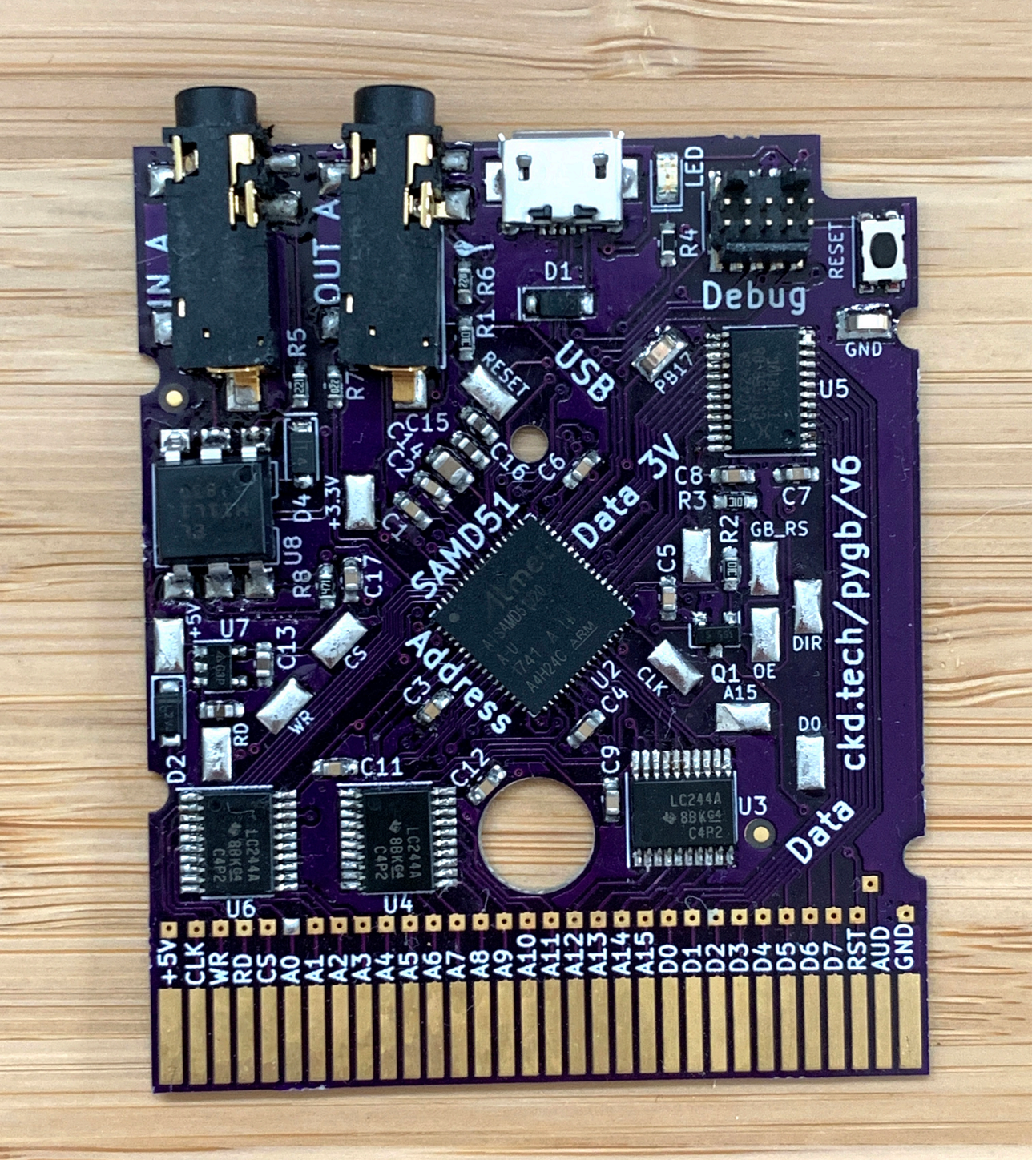

DMG-AAA-02
U1 PRG
GND


DMG-KFCN-01
Batt CR2025
MBC3A
U3 RAM
VDD
GND


IN A
OUT A
Debug
USB
SAMD51
Address
Data 3V
Data
ckd.tech/pygb/v6


Nintendo GAME BOY
TETRIS
DMG-TR-USA
THIS SIDE OUT
MADE IN JAPAN


mary-kateandashley
Pocket Planner
DMG-BMAE-USA
THIS SIDE OUT



MCU as Cart

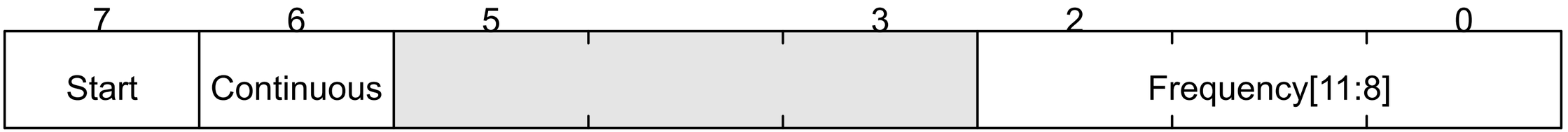
- DHole in late 2014 used an STM32F4 as a GameBoy cart.[1]
- Respond with a byte on the data bus for every 1 MHz clock where the address is in cart range (`0x0000-0x7fff`)
- Chose to use the SAMD51 because it is 120 MHz and already had CircuitPython support

[1] `https://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

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
```

The magic bit in `0xff14`



Layers

- Python Libraries - Expands the lowest level to simplify it
- Lowest Python - Barest Python that has no dependencies
- C <-> Python - hooks uniform C API to uniform Python API
- Lowest C - does time critical, chip-specific hardware interfacing
- The wire to the GameBoy CPU

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
```

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
```

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)
```

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)
```

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-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-bindings/gbio/__init__.c

```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-bindings/gbio/__init__.c

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

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

```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);
```

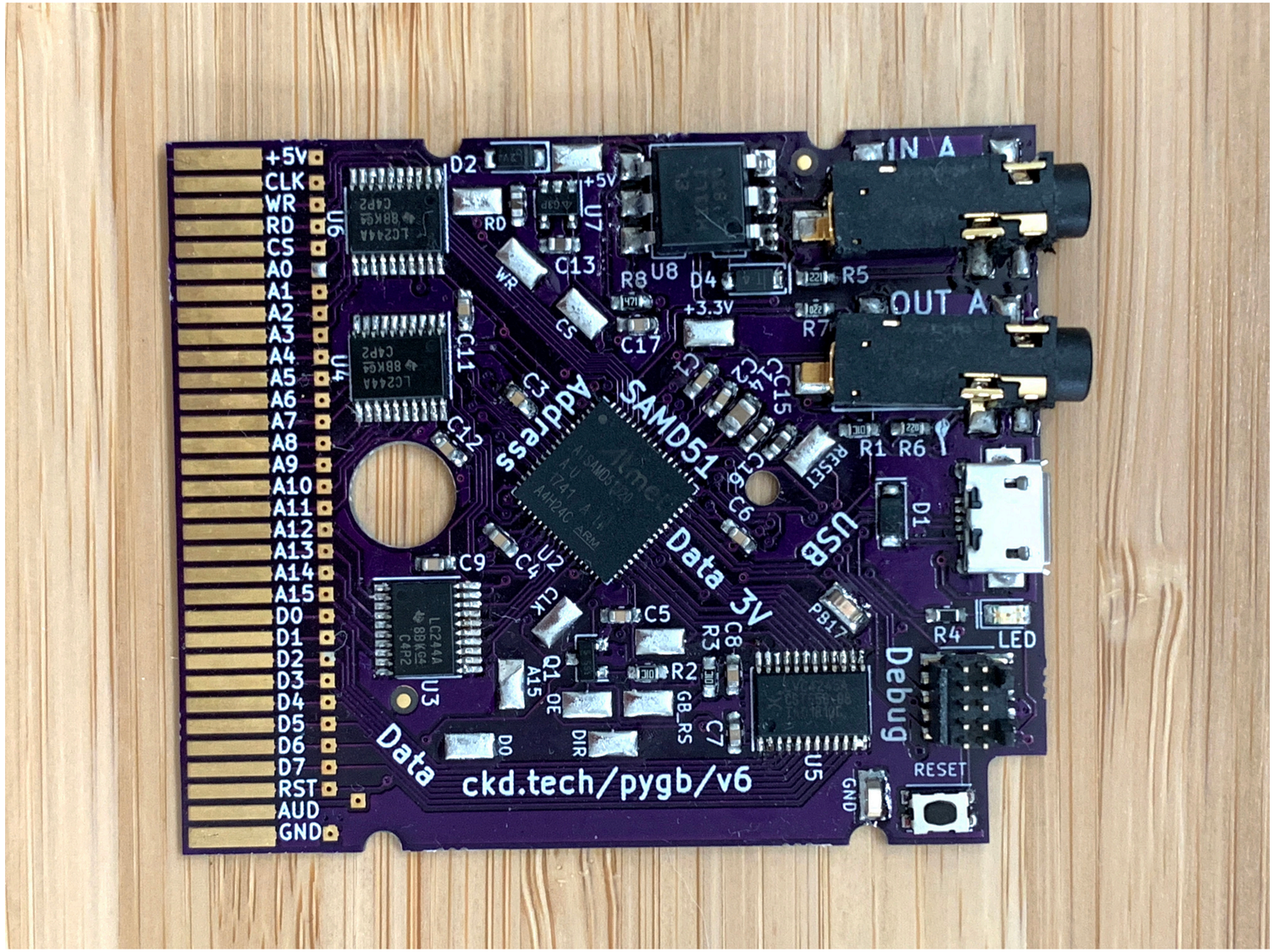
ports/atmel-samd/common-hal/gbio/__init__.c

```
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
}
```

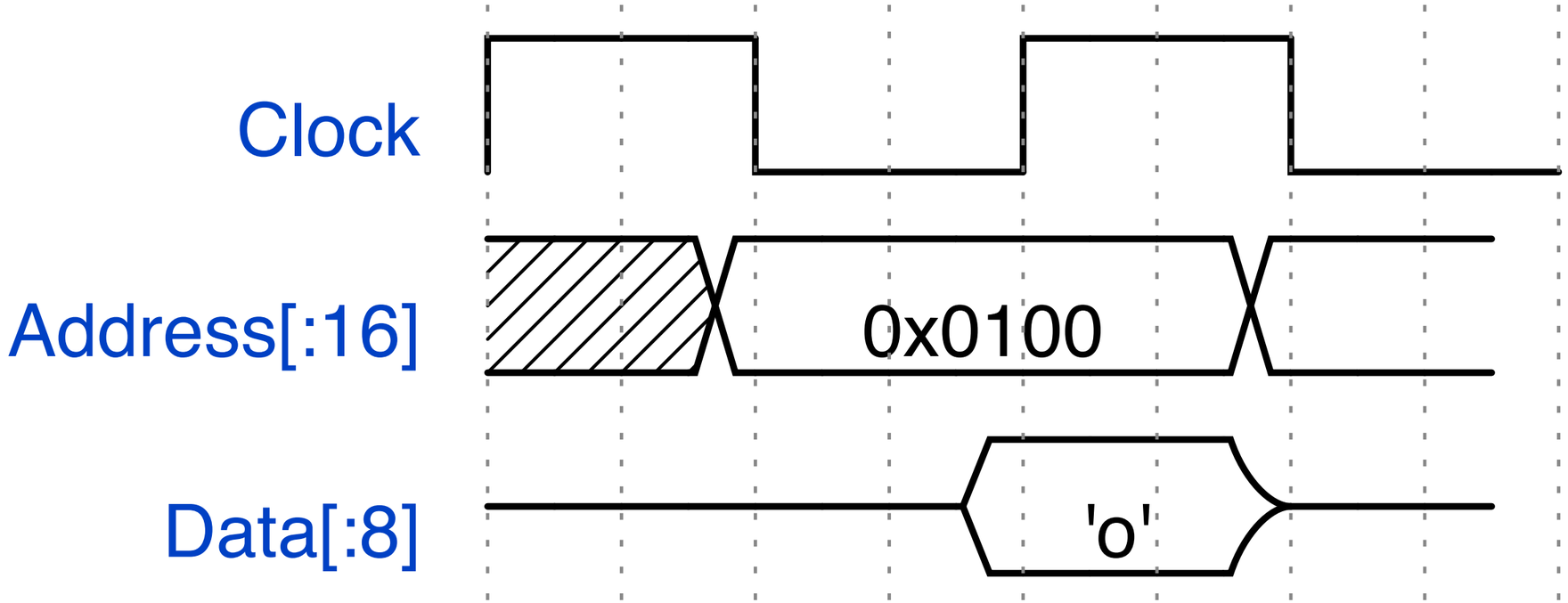
ports/atmel-samd/common-hal/gbio/__init__.c

```
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
}
```

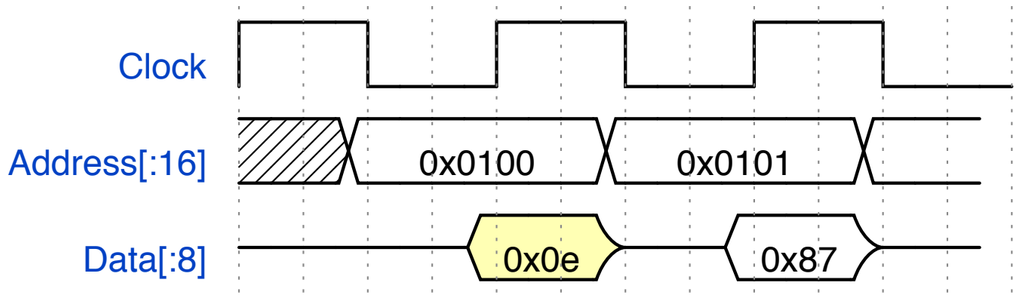
Wire


+5V
CLK
WR
RD
CS
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D0
D1
D2
D3
D4
D5
D6
D7
RST
AUD
GND
IN A
OUT A
USB
Debug
LED
RESET
Address
SAMD51
Data
3V
Data
ckd.tech/pygb/v6

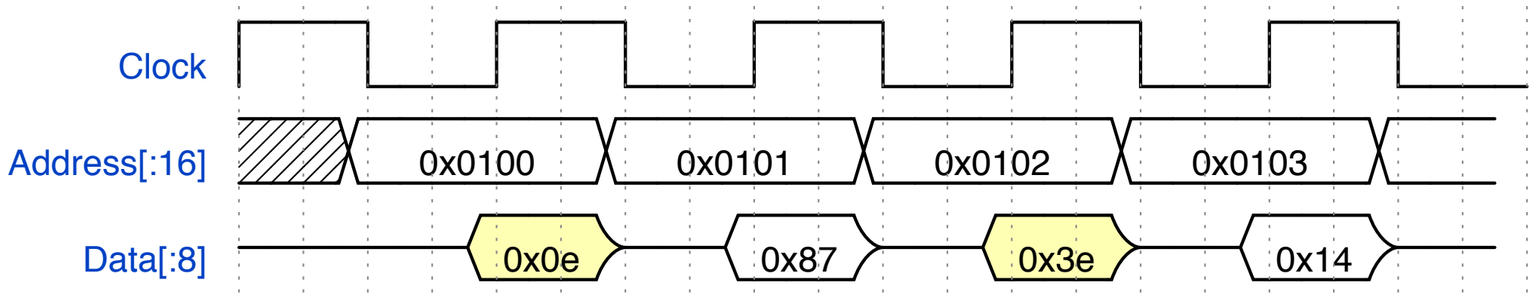
Timing Diagram



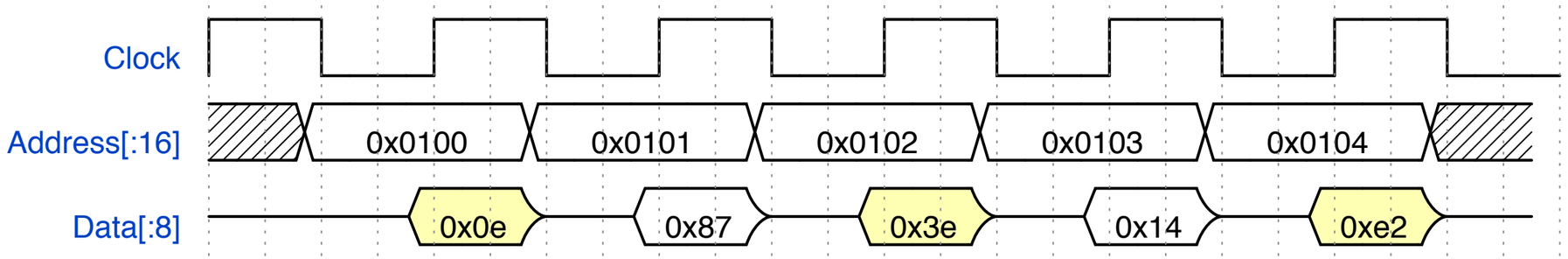
Load our new value into register A



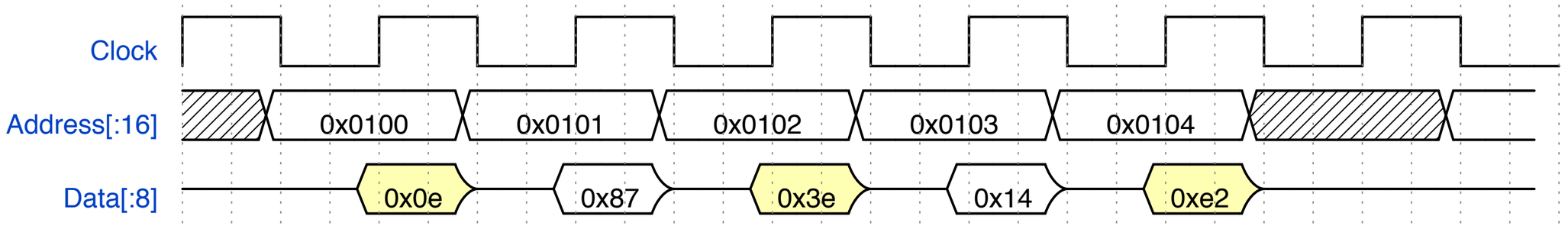
Load address into register C



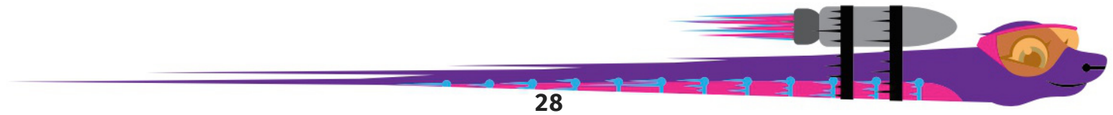
Store A into 0xff00 + C



Continue


Clock
Address[:16]
Data[:8]
0x0100
0x0101
0x0102
0x0103
0x0104
0x0e
0x87
0x3e
0x14
0xe2

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

Thank you!