Designing a Game Boy Cartridge

By emulating it with a RP2040

Sebastian Quilitz

Backspace Bamberg, 2024-01-27

Chapters

- Overview RP2040
- PIO
- Designing the hardware
- Ordering the PCB
- Manufacturing the hardware
- Software Overview

Overview RP2040

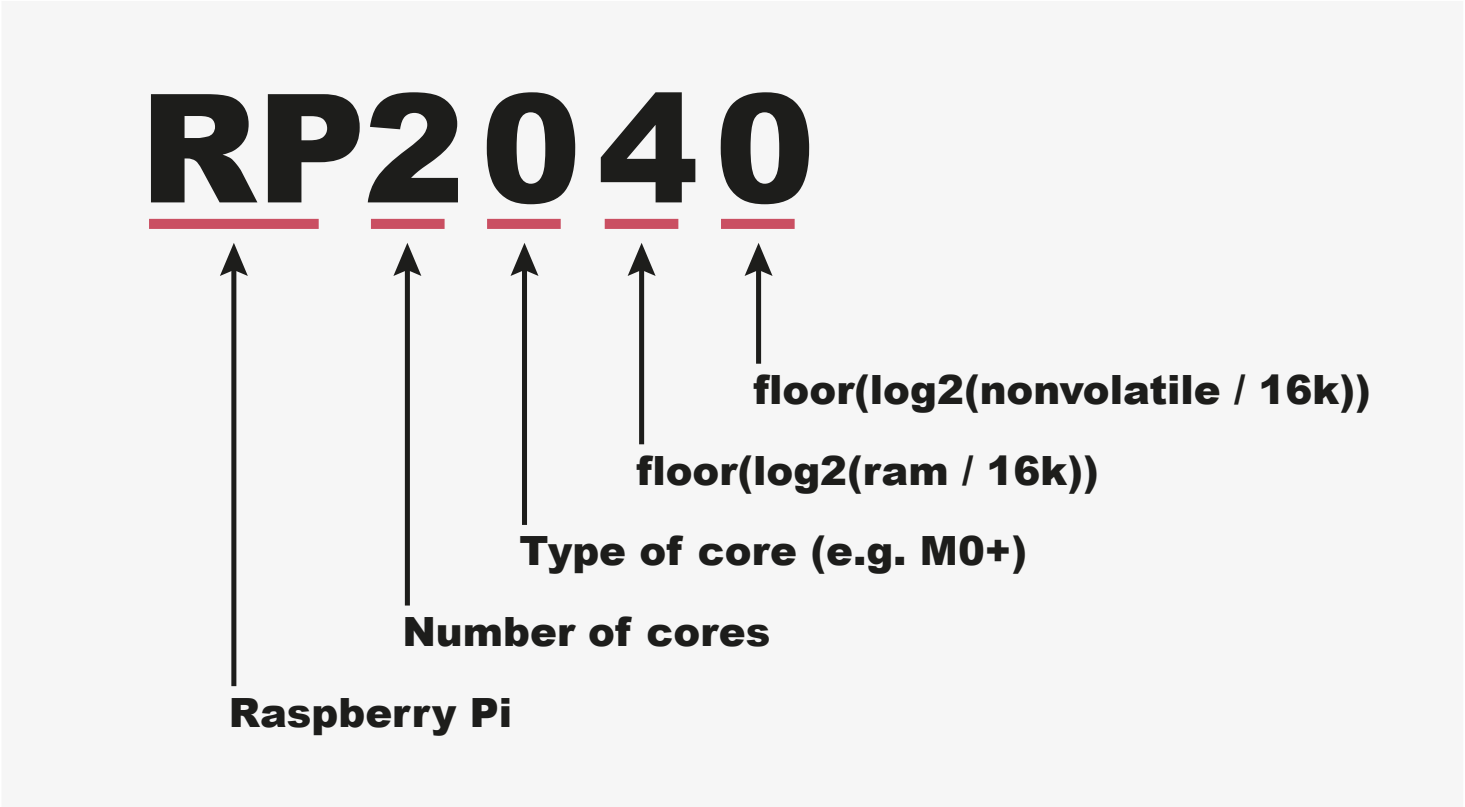
What is the RP2040?

- First microcontroller designed by Raspberry Pi Foundation
- Introduced firstly in the Raspberry Pico in January 2021
- Available as single chips in small numbers since June 2021 and broader availability in autumn/winter 2021
- Designed to be low-cost: Raspberry Pico 4$, MCU 1$

RP2040 Hardware Spec

- Dual Core ARM Cortex M0+
- Rated for 133 MHz (overclockable)
- 264 KB internal SRAM
- No internal Flash or EEPROM
- QSPI interface for external flash which allows XIP
- USB controller
- SPI, I2C, UART
- Programmable IO (PIO)

Why the name RP2040?



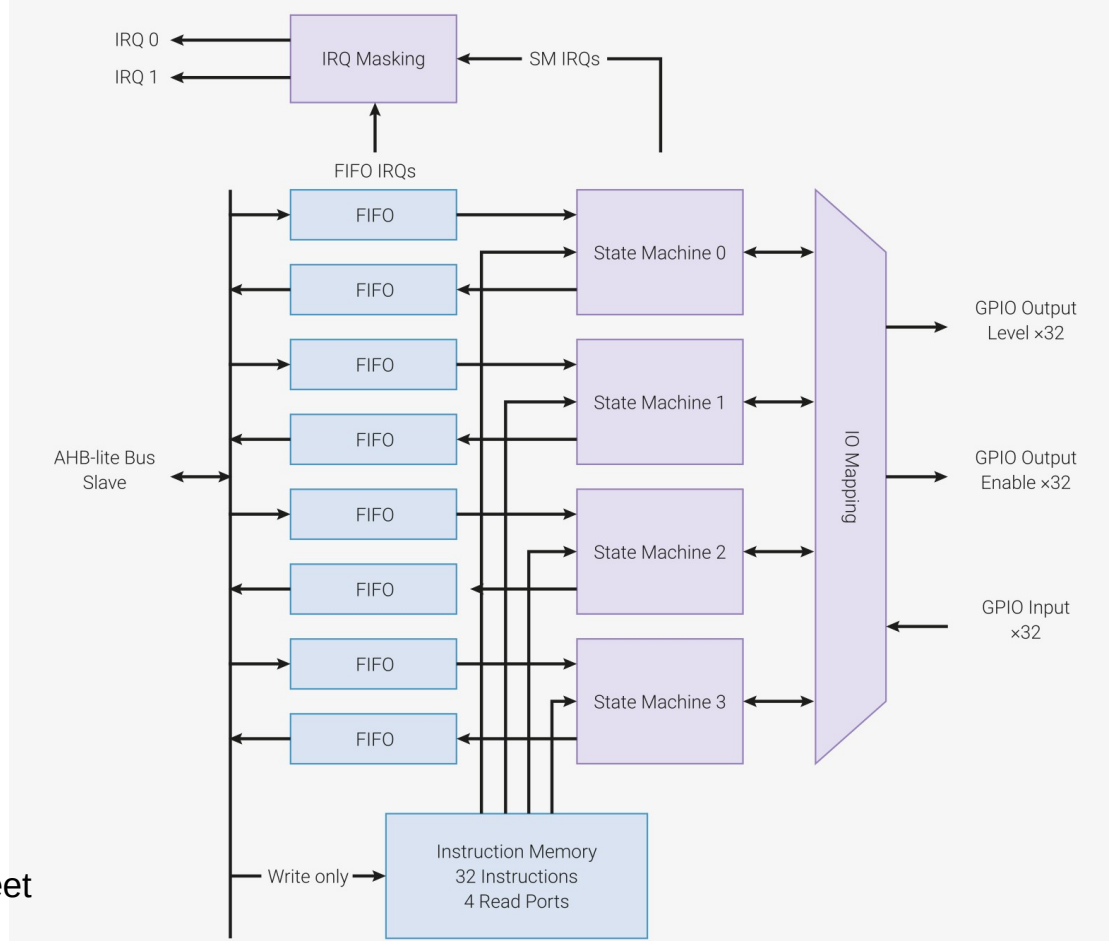
Source: RP2040 Datasheet

Programmable IO (PIO)

PIO Overview

- Instruction set highly optimized for IO operations
- RP2040 has 2 PIO blocks
- Each PIO block has 32 instructions
- Each PIO block has 4 state machines which share the instructions
- Each state machine has 2 FIFOs for communication with ARM cores

PIO Blockdiagram



Source: RP2040 Datasheet

Tightly packed PIO Assembler

For example one instruction can:

- Read all pins at once
- Sideset (output) pins
- Delay a few cycles
- Autopush to the FIFO
- Wrap back to the beginning of instruction loop

```
in pins 8 side 1 [3]
```

But PIO is bad at

- Compare operations
- Conditional jumps
  - Only one pin can act as jump
- Mathematical operations

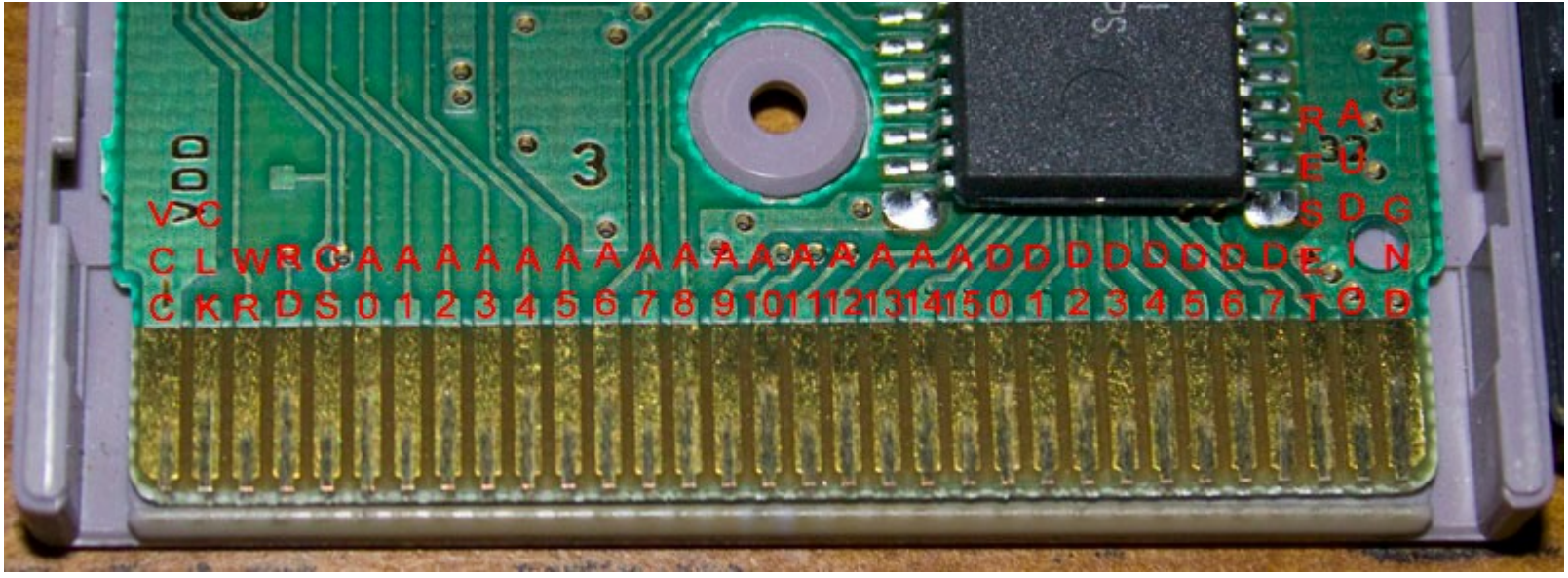
Game Boy

Game Boy Overview

- EU Release: September 1990
- Sharp LR35902 core @ 4.19 MHz
  - Similar to Zilog Z80 and Intel 8080
- Work RAM: 8KiB
- Video RAM: 8KiB
- Screen Resolution: 160x144

Game Boy Cartridge interface

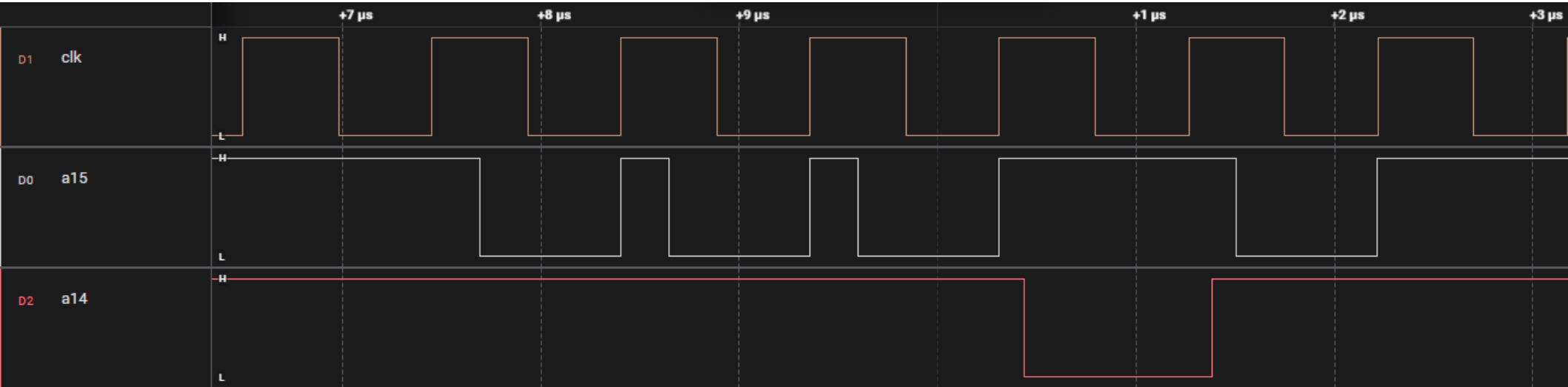
Parallel bus with 16 address lines and 8 data lines @~1Mhz



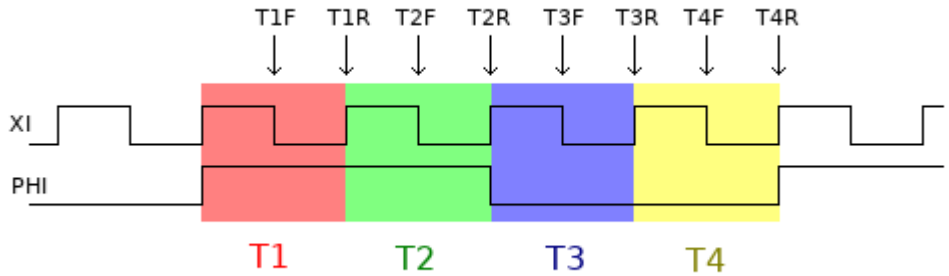
Source: https://www.insidegadgets.com/2016/12/06/gbxcartread-part-1-design-and-testing-gameboy-carts/

External memory interface

- A15 and CS are special pins
- CLK is not the CPU clock but a slower fetch clock @~1Mhz



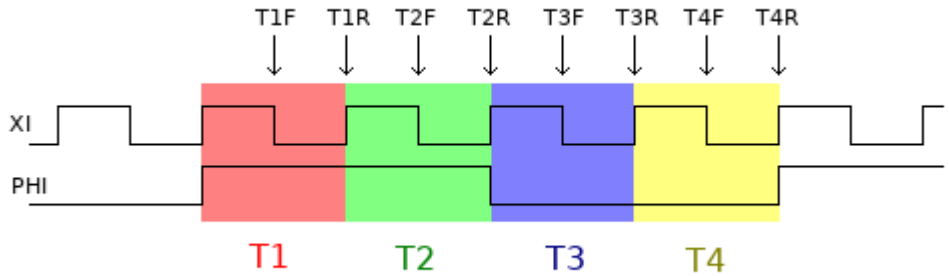
Bus timings



- T1F:
  - Only when CPU accesses External Bus, Internal Bus or Internal Video Bus:
    - A0-A14 change to new address
  - Only when CPU writes to External Bus:
    - #read goes high
- T1R:
  - Only when CPU accesses External Bus:
    - #cs or A15 goes low

Source: http://iceboy.a-singer.de/doc/mem_patterns.html

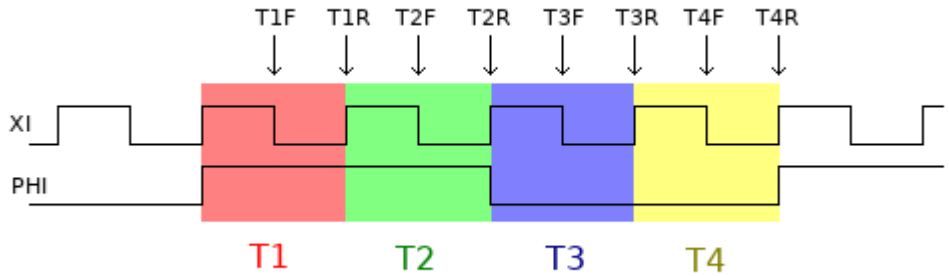
Bus timings



- T2F:
  - No changes
- T2R:
  - Only when CPU writes to External Bus:
    - #write goes low
    - D0-D7 get driven with the actual data

Source: http://iceboy.a-singer.de/doc/mem_patterns.html

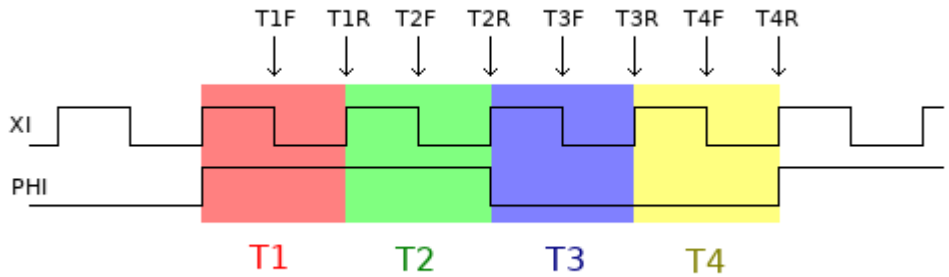
Bus timings



- T3F:
  - No changes
- T3R:
  - D0-D7 get sampled during read

Source: http://iceboy.a-singer.de/doc/mem_patterns.html

Bus timings



- T4F:
  - Only when CPU writes to External Bus:
    - #write goes high
- T4R:
  - Only when CPU accesses External Bus:
    - #cs or A15 goes high
  - Only when CPU writes to External Bus:
    - D0-D7 stop being driven (pull-up resistors take over)
    - #read goes low

Source: http://iceboy.a-singer.de/doc/mem_patterns.html

Gameboy Memory Map

Start	End	Description	Notes
0000	3FFF	16 KiB ROM bank 00	From cartridge, usually a fixed bank
4000	7FFF	16 KiB ROM Bank 01~NN	From cartridge, switchable bank via mapper (if any)
8000	9FFF	8 KiB Video RAM (VRAM)	In CGB mode, switchable bank 0/1
A000	BFFF	8 KiB External RAM	From cartridge, switchable bank if any
C000	CFFF	4 KiB Work RAM (WRAM)	
D000	DFFF	4 KiB Work RAM (WRAM)	In CGB mode, switchable bank 1~7
E000	FDFF	Mirror of C000~DDFF (ECHO RAM)	Nintendo says use of this area is prohibited.
FE00	FE9F	Object attribute memory (OAM)	
FEA0	FEFF	Not Usable	Nintendo says use of this area is prohibited
FF00	FF7F	I/O Registers	
FF80	FFFE	High RAM (HRAM)	
FFFF	FFFF	Interrupt Enable register (IE)	

Source: Pan Docs

Designing the hardware

Bus Translators

- RP2040 runs at 3,3V
- But Gameboy at 5 V
- Texas Instrument TXB0108
  - Bi-directional bus translator
  - Auto direction
  - 15 kV ESD protection

RGB LED

- WS2812b (Neopixel)
- 1-wire control of color & brightness

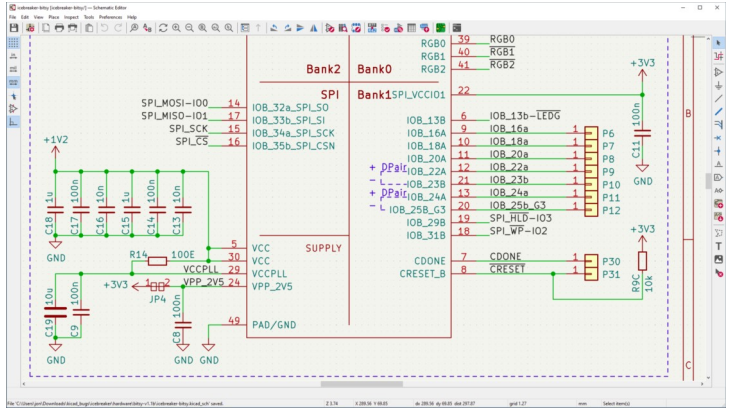
Design goals

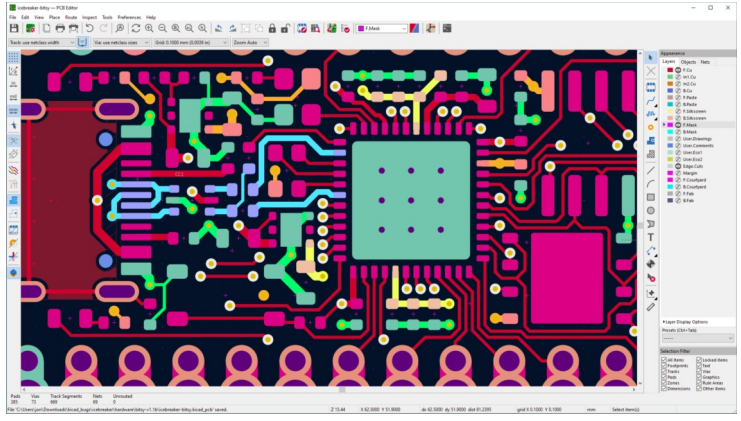
- As less components as possible

→ Mostly software defined

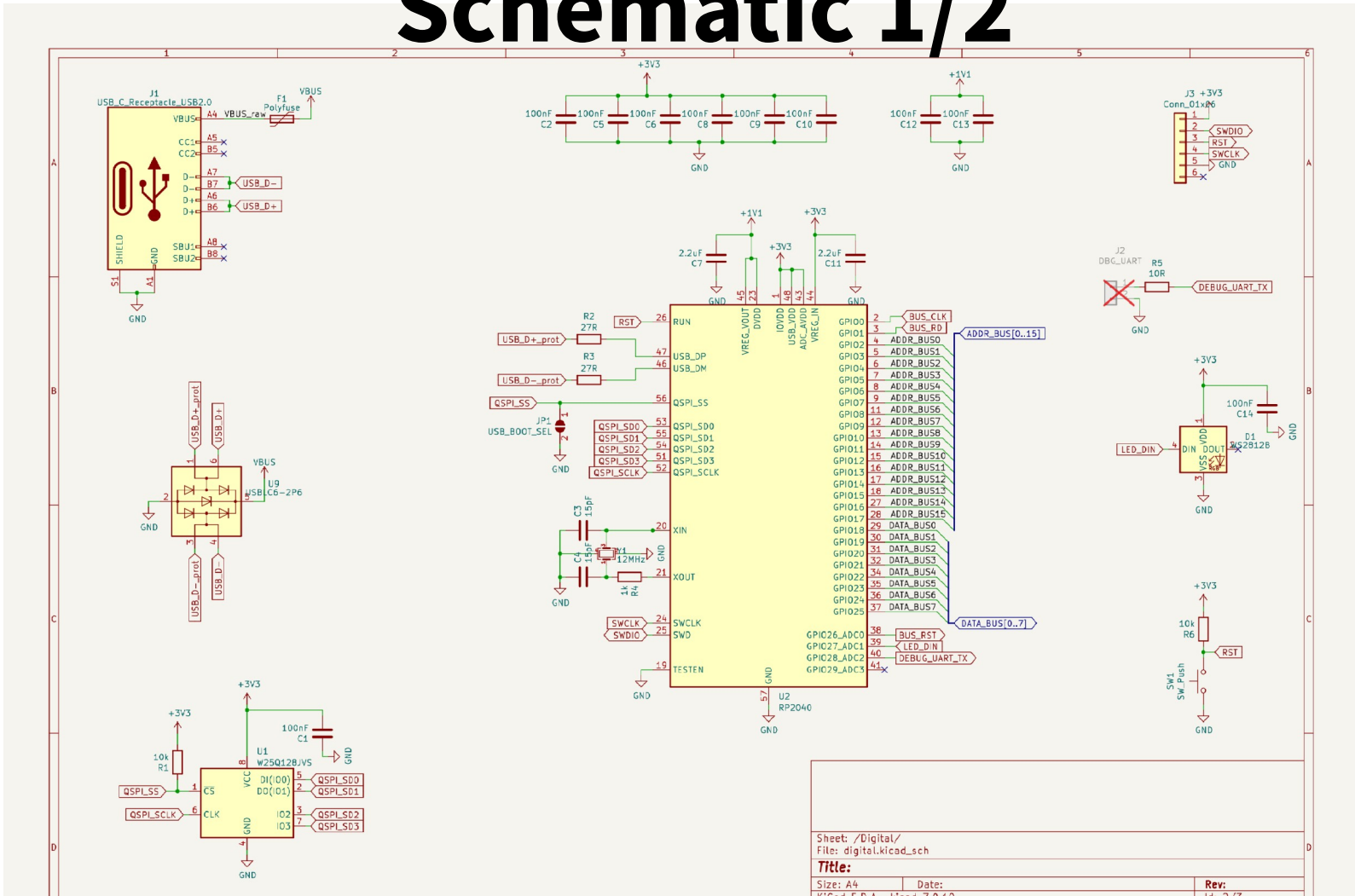
Design Tool

- KiCad
- Open Source www.kicad.org

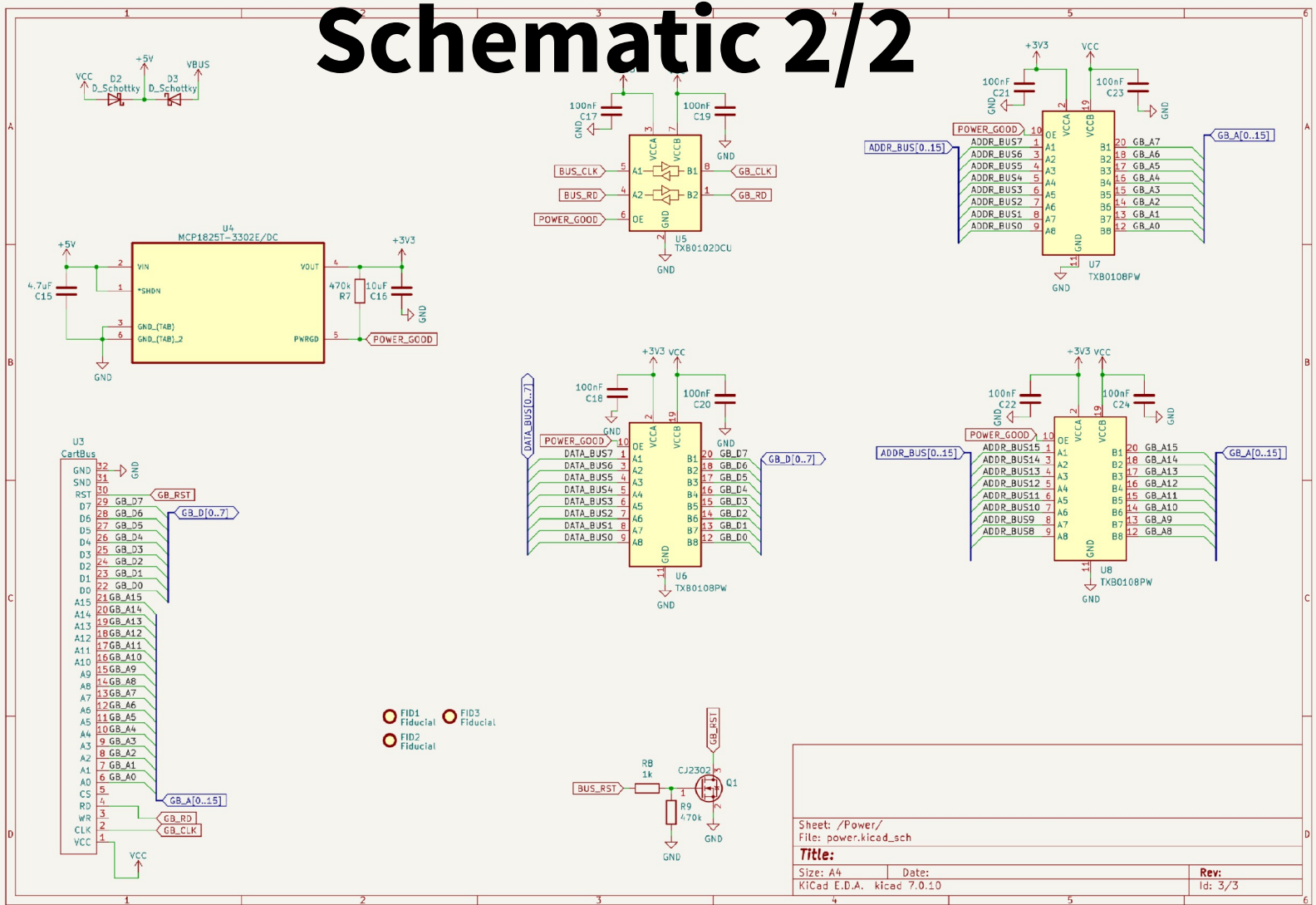




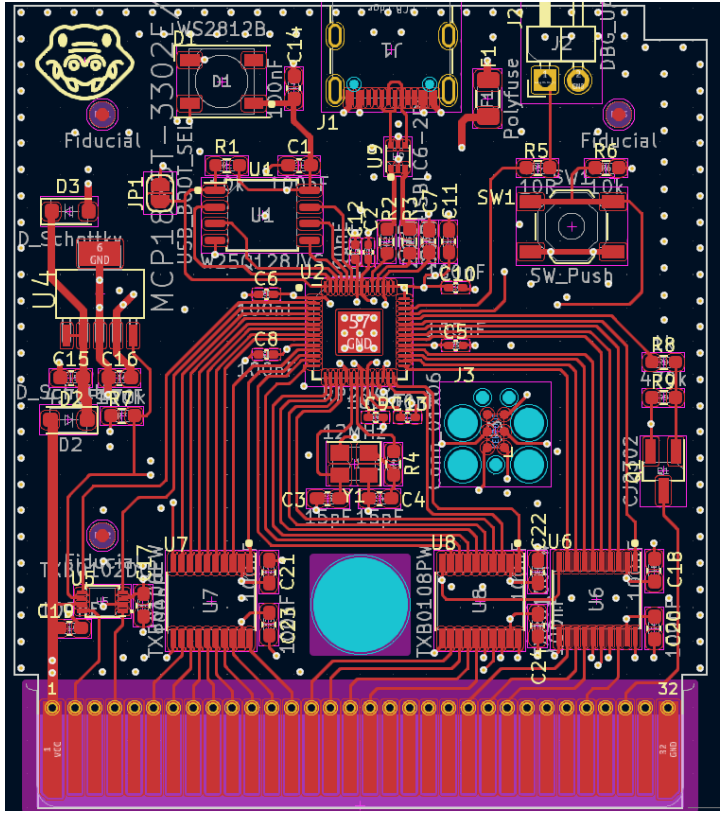
Schematic 1/2



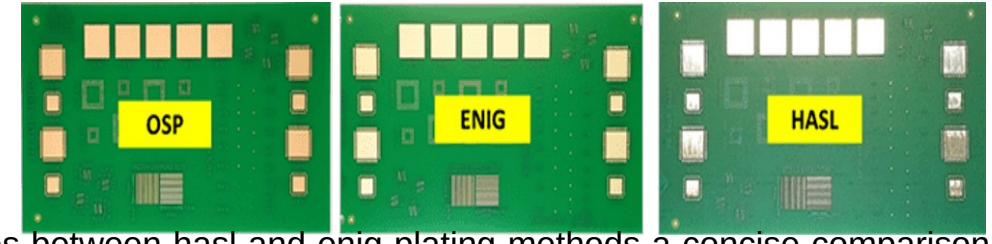
Schematic 2/2



After Routing

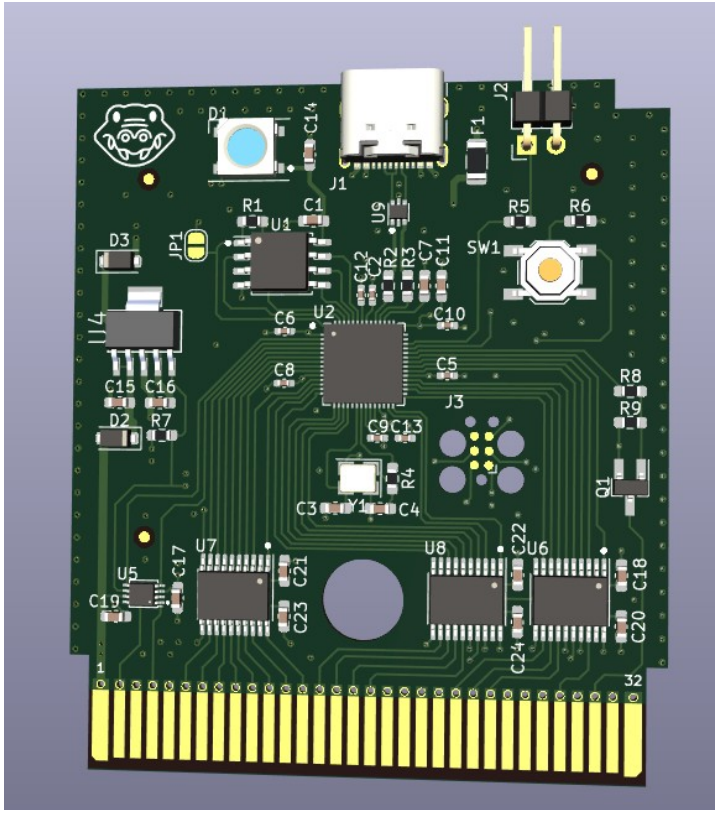


- 4 layer PCB
  - Reduces EMC and cross-talk between the signal lines as the distance from signal to power planes is shorter
  - Is not that much more expensive than 2 layer
    - 18$ vs 23$ for 10 PCBs
    - Expensive part is the ENIG plating and shipping
- 0.8 mm thick
- Inner layers
  - GND
  - Power supply and signal
- ENIG (Eloctroless nickel immersion gold) as surface finish to make the edge connector longer lasting



Source: https://hillmancurtis.com/10-differences-between-hasl-and-enig-plating-methods-a-concise-comparison/

3D View


D1
C14
J1
J2
F1
R1
C1
U1
U9
R5
R6
D3
JP1
C12
C2
R2
R3
C7
C11
SW1
U4
C6
U2
C10
C8
C5
R8
C15
C16
J3
R9
D2
R7
C9
C13
R4
Q1
C3
Y1
C4
U7
U8
C22
U6
U5
C17
C21
C18
C19
C23
C20
C24
1
32

Ordering the PCB

PCB ordering

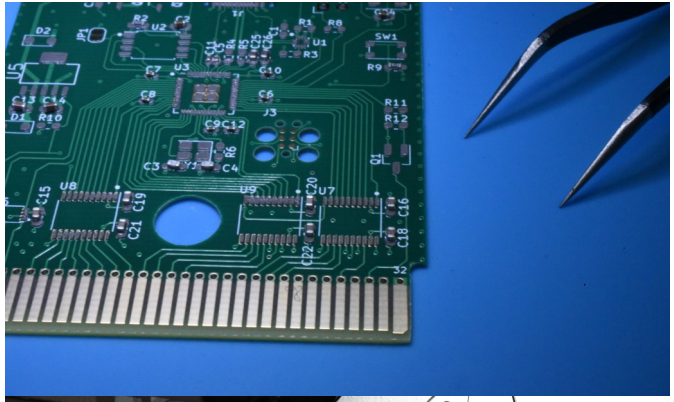
- Several online manufacturers available
- Examples:
  - PCBWay (China)
  - JLCPCB (China)
  - Eurocircuits (Hungary and Germany)
  - Aisler (Netherlands)

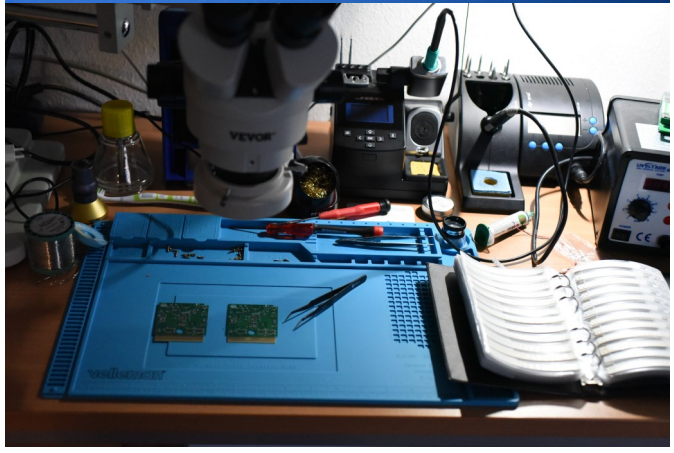
PCB assembly

- Solder yourself
  - Soldering iron
  - Get Stencil with PCB and reflow
- Order assembled
  - Let the PCB manufacturer do the assembly
  - Tip to save money:
    - Only SMD parts
    - Only one side

Manual assembly

PCB assembly





- Place solder paste with stencil
- Place components with tweezers
  - Microscope recommended

Bake it

- Reflow
  - IR oven
  - Iron (for clothes)
  - Pan
  - Hot plates
  - Pizza oven

Software

Software problems

- Timing is tight

  --> Fast / less instructions as possible

  --> Use DMA as much as possible

  --> Conditional bus handling must happen on PIO side

    --> PIO is not really designed for that

- Not enough RAM to buffer full ROM in most cases

  --> Must be loaded from flash in time

PIO

- Generally the following Statemachines are needed:
  - reading lower ROM
  - reading higher (bank-switched) ROM
  - reading Savegame RAM
  - writing Savegame RAM
  - Writing data back to the bus
  - “Master” statemachine to follow bus for bank switching
  - (Statemachine to control RGB LED)

Start	End	Description
0000	3FFF	16 KiB ROM bank 00
4000	7FFF	16 KiB ROM Bank 01~NN
8000	9FFF	8 KiB Video RAM (VRAM)
A000	BFFF	8 KiB External RAM

Master State Machine

```
31  .program gameboy_bus
32
33  a15_high:
34      irq set 4
35  .wrap_target
36      in pins 1 ; shift in RD pin
37      mov y isr ; store rd pin in y
38      in   pins 17                    ; shift rd pin and address into ISR
39      in null 15 ; fill up ISR to trigger auto push
40      wait 0 gpio PIN_CLK
41      jmp  !y idle[7]  ; Y holds read pin, skip to idle on reads
42      in   pins 25  ; sample read rd pin, addr pins and data pins
43      in   null 24  ; 7+17=24 shit read and addr pins out of the isr to only leave the data, trigger auto push
44  idle:
45  public entry_point:
46      mov  isr null  ; Clear ISR
47      wait 1 gpio PIN_CLK                    ; wait for clk
48
49      set  y DELAY_COUNT_ADDR_READ[6]
50  loop:
51      jmp  y-- loop[1]                      ; delay to let adress pins become available
52      jmp  pin a15_high                      ; if A15 is high jump to high area notification
53      irq set 5                           ; set irq for A15 low, though right now nobody needs it
54      .wrap ; wrap back to beginning
55
```

State Machines for savegame RAM

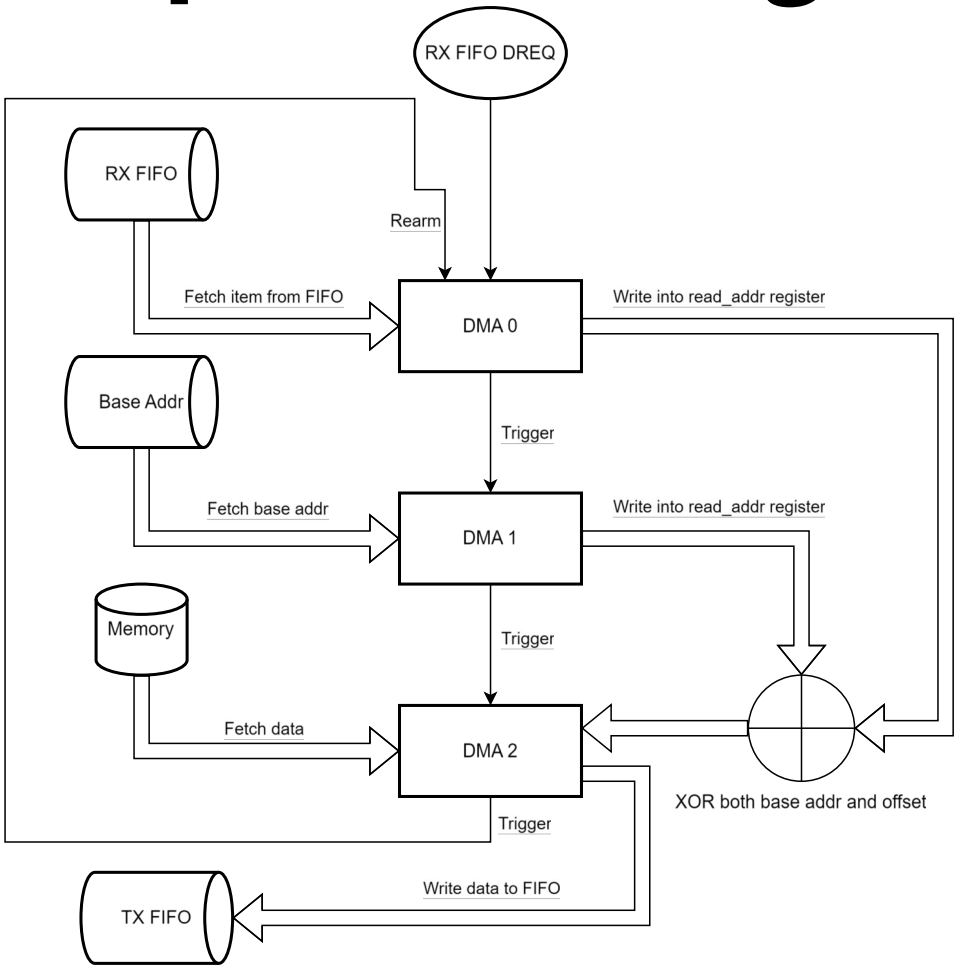
```
84  .program gameboy_bus_detect_a14
85  idle:
86  .wrap_target
87      wait 1 irq 4
88      jmp pin idle ; jmp back to idle if A14 is high
89      irq set 6
90      .wrap ; wrap back to beginning
91
92
93
94  .program gameboy_bus_ram
95  public read_wrap_target:
96      jmp  y-- idle_data                          ; Y=Rnw - skip the FIFO push on write cycles (RnW=0)
97      in   null 19                                  ; shift fixed part of ARM address (held in x) into ISR and trigger auto push
98
99  public entry_point:                                     ; *** Default Entry Point ***
100 public write_wrap_target:
101 idle_data:
102     mov  isr null    ; Clear ISR
103     wait 1 irq 6
104     jmp pin continue  ; jmp back to idle if A13 is high
105     jmp idle_data
106 continue:
107     in pins 1  ; shift in RD pin
108     mov y isr ; store rd pin in y
109
110     in   pins 14                    ; shift rd and A0 to A12 pins into ISR
111 public read_wrap:                                     ; *** READ state machine wraps to read_wrap_target ***
112
113     jmp  !y idle_data                         ; Y=Rnw - skip the FIFO push on read cycles (RnW=1)
114     in   null 19                                  ; shift fixed part of ARM address (held in x) into ISR and trigger auto push
115
116     wait 0 gpio PIN_CLK [7]                     ; wait for clk
117     in   pins 25                              ; sample read rd pin, addr pins and data pins
118     in   null 24                             ; 7+17=24 shit read and addr pins out of the isr to only leave the data
119 public write_wrap:                                     ; *** WRITE state machine wraps to write_wrap_target ***
```

State Machine to push data to the GB

```
122  .program gameboy_bus_write_to_data
123  .wrap_target
124      pull block                          ; pull the data from the Tx FIFO into the OSR
125      out  pins 8                        ; output 8 bits of data
126      mov  osr ~NULL                      ; OSR=FFFFFFFF
127      wait 0 gpio PIN_CLK                 ; wait for clk
128      out  pindirs 8                      ; start driving the data bus
129      mov  osr NULL
130      wait 1 gpio PIN_CLK                ; wait for clk
131      out  pindirs 8                     ; stop driving the data bus
132      .wrap ; wrap back to beginning
```

DMA loop for reading memory

Base addr points to either:
- address in RAM
- address in flash (XIP)

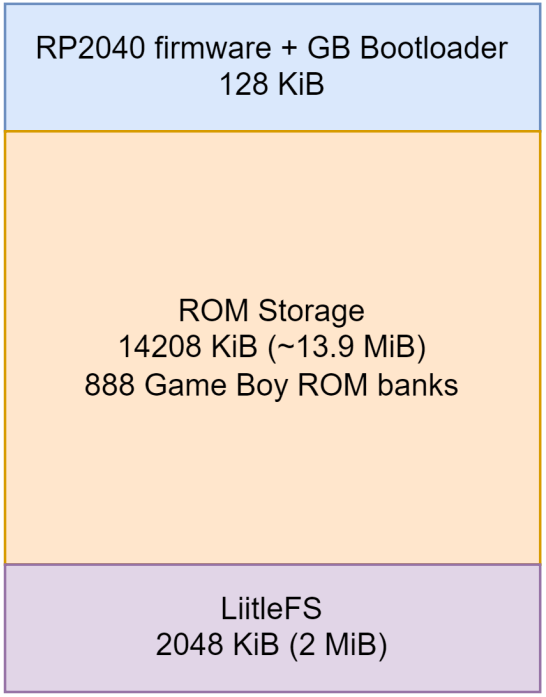


The CPU has shadow registers for peripherals which allows XOR operations

Game Boy bootloader

- Small Game Boy “Game” which shows the list of games
  - Communication with RP2040 through shared memory (savegame RAM)
- Written in C with the toolchain from GBDK-2020

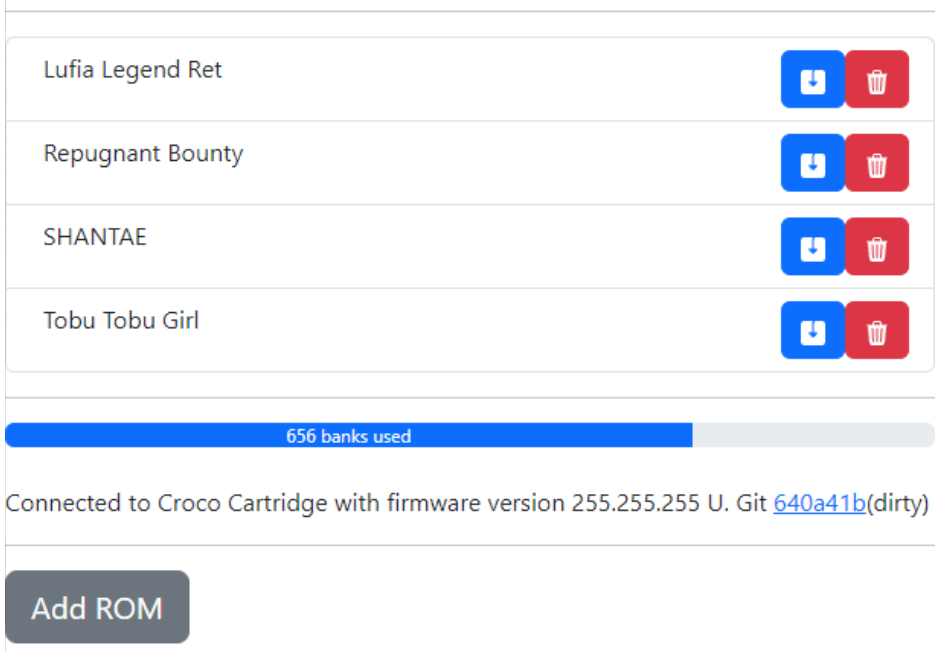
Flash layout



- 16 MiB total
- Each bank in ROM storage is aligned
- LittleFs holds savegames, config files, and ROM info

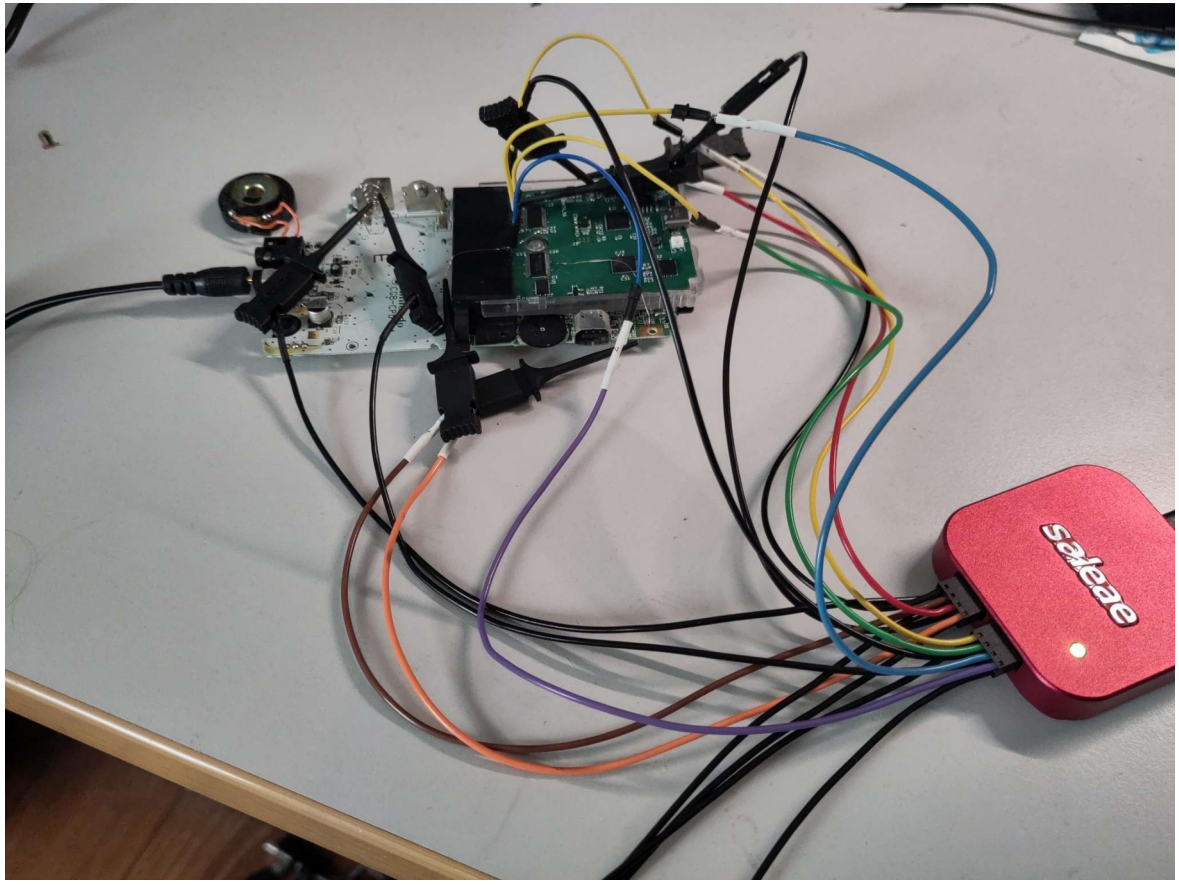
Management Software

- Uses WebUSB for connection
  - Chrome based browsers (also on Android)
- Written in React JS

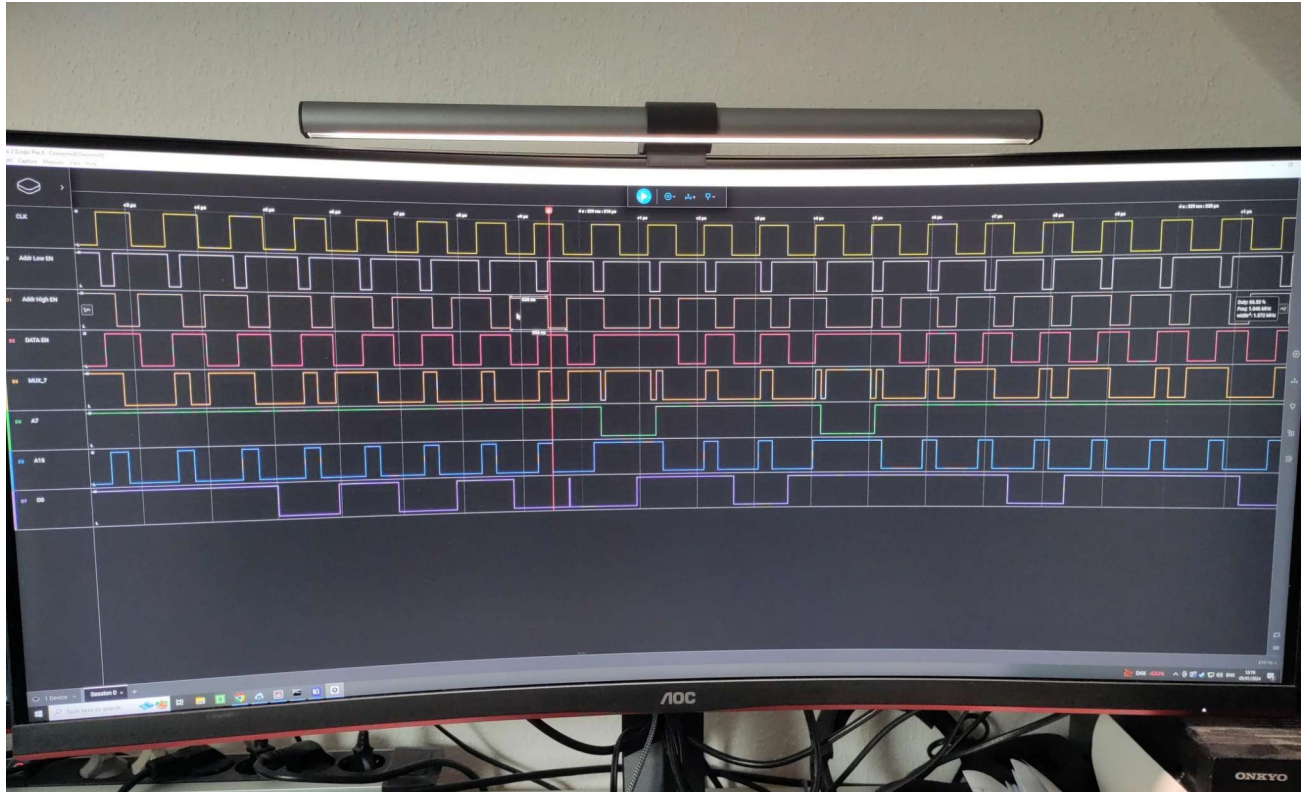


Debugging

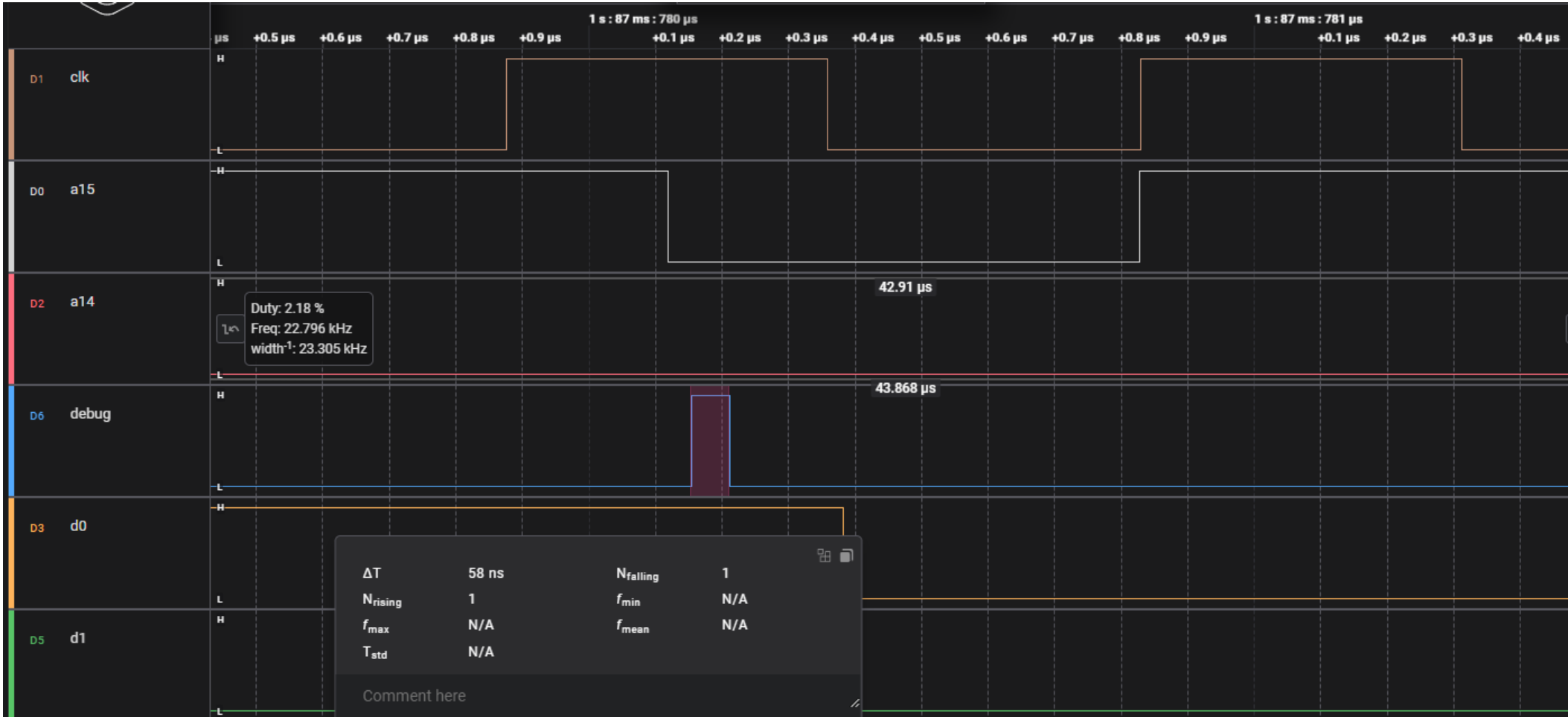
Debugging



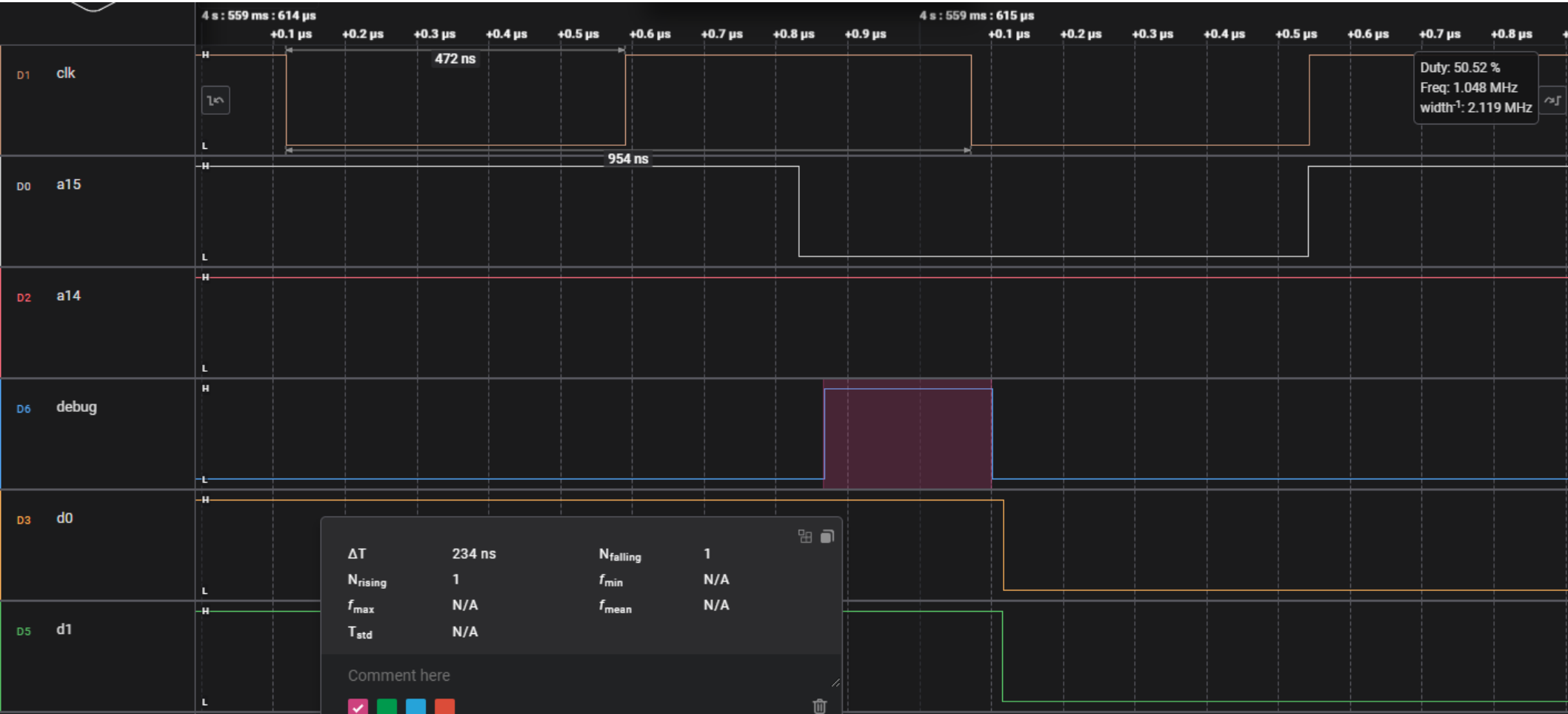
Debugging



Data fetch from RAM



Data fetch from flash



Sources and helpful documents

Sources

- Pandocs: https://gbdev.io/pandocs/
- (unoffical) Game Boy CPU Manual: D. P.
- Game Boy Technical Reference: gekkio.fi
- Game-Boy-KiCad-Library: HDR (Martin Refseth)
- PIO introduction: Youtube stacksmashing

Thank you!