

BY NIKO ZURSTRASSEN, RWTH AACHEN UNIVERSITY
SYSTEMC FIKR 2025-03-27



WHAT IS THE TLMBOY?

 README

 Apache-2.0 license

https://github.com/not-chciken/TLMBoy





 Build and Test CI

passing

 coverage

86%

 code quality

A

TLMBoy

A Game Boy Simulator written in C++/SystemC TLM-2.0.

Building

Use the following commands to build the TLMBoy:

```
cd TLMBoy
```



gb-ts

YET ANOTHER GAME BOY EMULATOR?

GasyBoy

Boy

GameLad

xgbc

Peanut-GB

Dromai Anemulator

php-gameboy

Emu Boy

vba-gb

Plutoboy

AI Plays
Pokemon

GBS

jsGBC

node-ga

GGB

binjgb

miniBoy

Gearb

LuaGB

Goodboy

gameboi

gbemu

JBoyEmu

Castor

POPBoy

NoobBoy

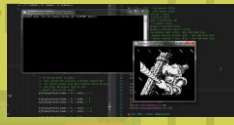
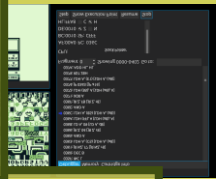
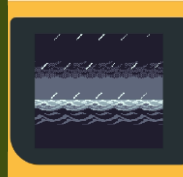
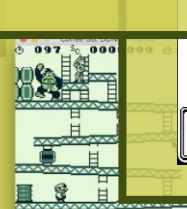
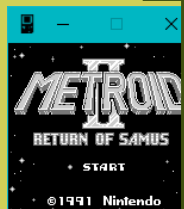
Dawn GB

gameboyGO

Kotlin-
Gameboy-
Emulator

gBoy

Frame Boy



YET ANOTHER GAME BOY EMULATOR?

<https://github.com/gabrielrcouto/php-terminal-gameboy-emulator>

```
root@2a708e75221a:/app# php boot.php drmario.rom
```

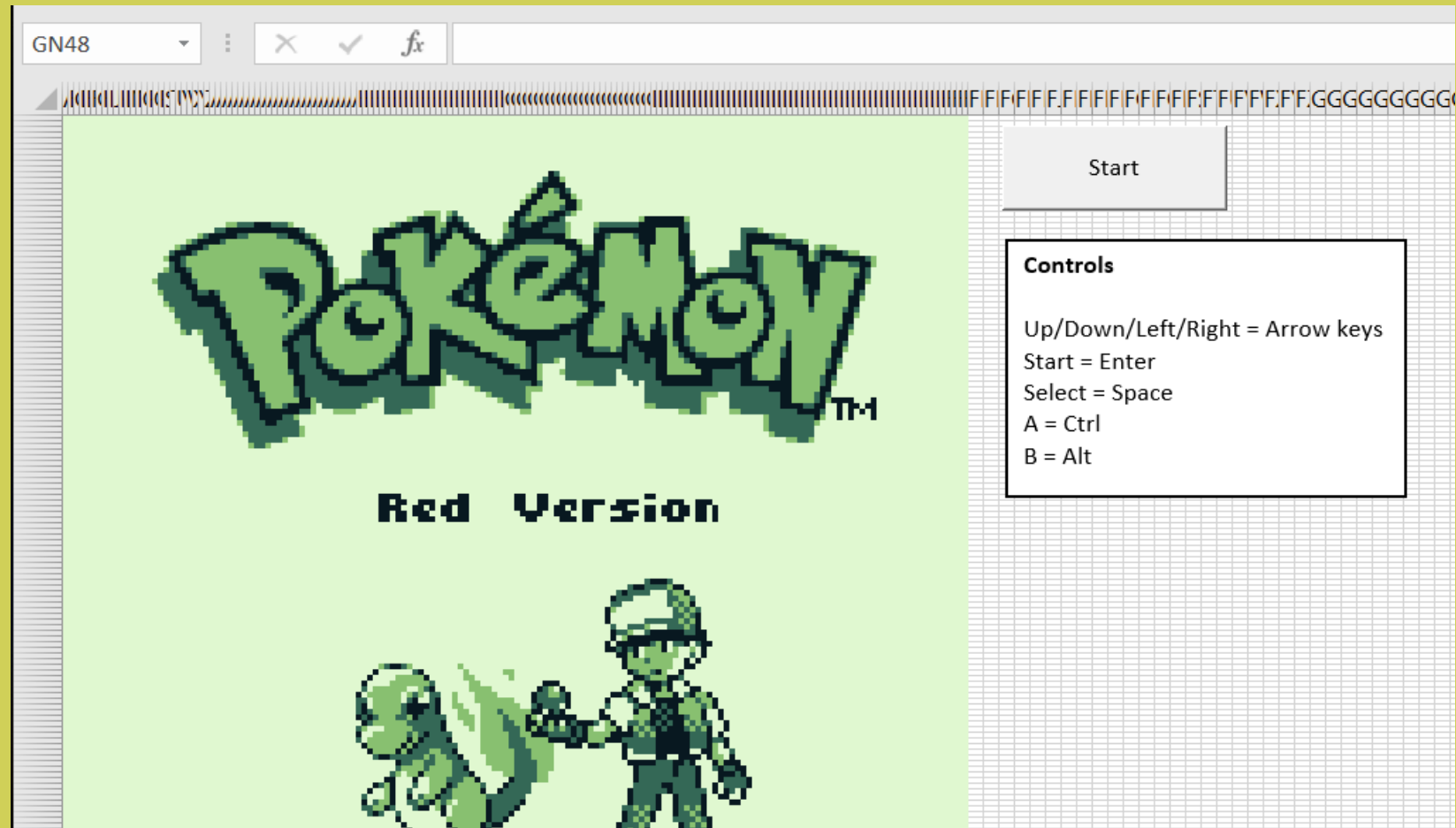
YET ANOTHER GAME BOY EMULATOR?

<https://www.youtube.com/watch?v=VcMLJ4ZFfGE>



YET ANOTHER GAME BOY EMULATOR?

<https://github.com/KatrinaJames/vba-gb>



A GAME BOY EMULATOR IN SYSTEMC?





r/EmuDev



"systemc"



Log In



Posts

Comments

Media

Relevance ▾

All time ▾

Show results from [all of Reddit](#) →



Hm... we couldn't find any results for
"systemc"

Double-check your spelling or try different keywords to
[adjust your search](#)

Developing Emulators

Community of people building
emulators. Discord:
<https://discord.gg/dkmJAes>

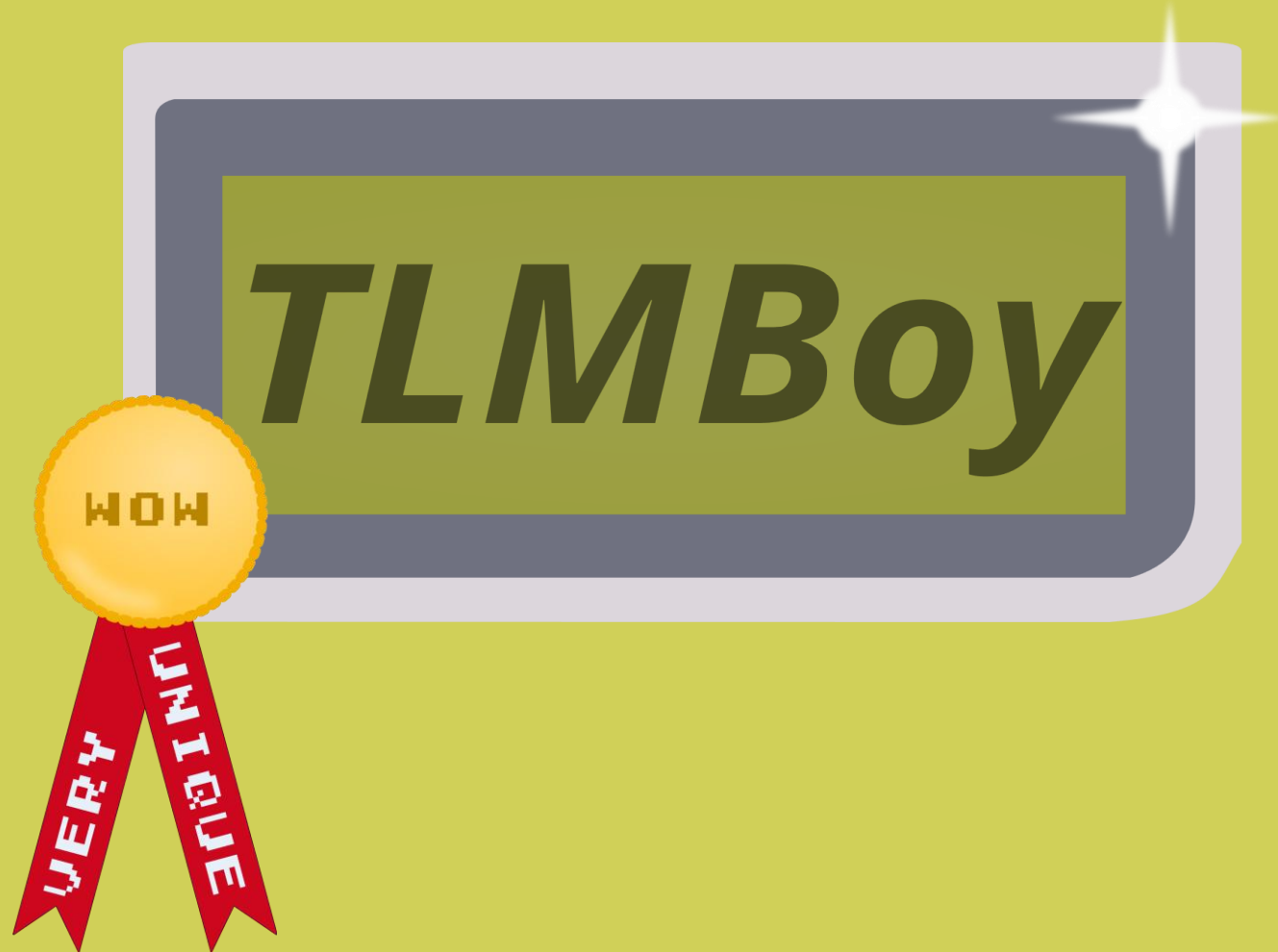
23K

Members

2

● Online

A UNIQUE PROJECT



HOW TO START?

HOW TO START?



Screen?

Graphics Card?

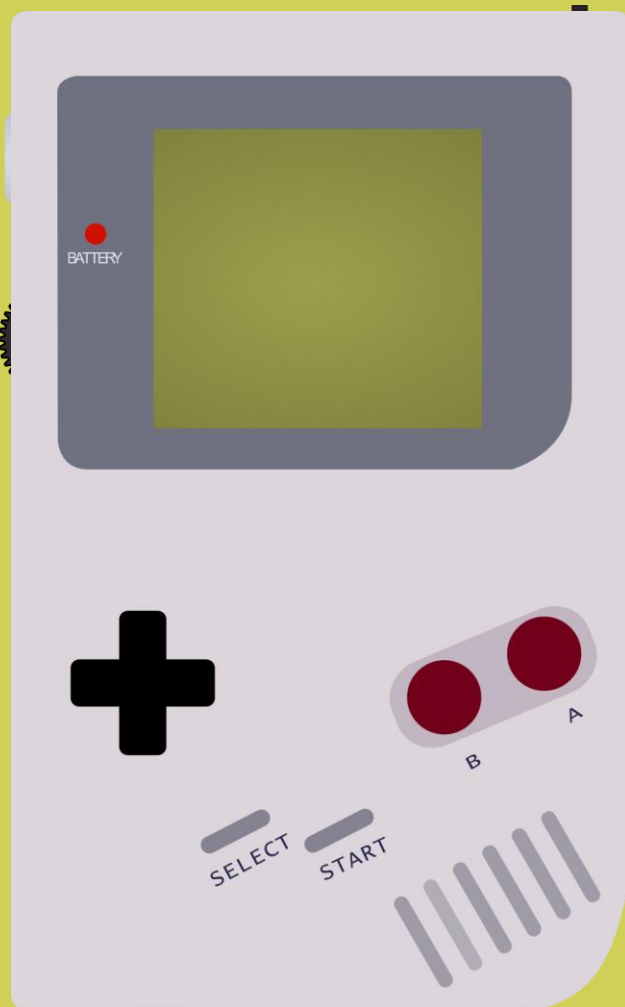
CPU?

Buttons?

Sound?

...

HOW TO START?



README Code of conduct License <https://github.com/gbdev/awesome-gbdev>

Awesome Game Boy Development

Join us on Discord chat 981 online

A curated list of awesome Game Boy (Color) Development resources, tools, docs, related projects and open-source ROMs. Inspired by the [awesome](#) list thing.

You can find a (way cooler) web version of this list [here](#).

Contents

- [Introduction](#)
 - [Disambiguation](#)
- [Community](#)
- [Documentation](#)
 - [Misc](#)
 - [Opcodes](#)
 - [Game Boy Color](#)

BUILDING BLOCKS



\$89.99

Released in 1989
(EU 1990)

by **Nintendo®**

„DMG-01“







Source: <https://www.youtube.com/watch?v=HyzD8pNlpwI>

BUILDING BLOCKS



BUILDING BLOCKS



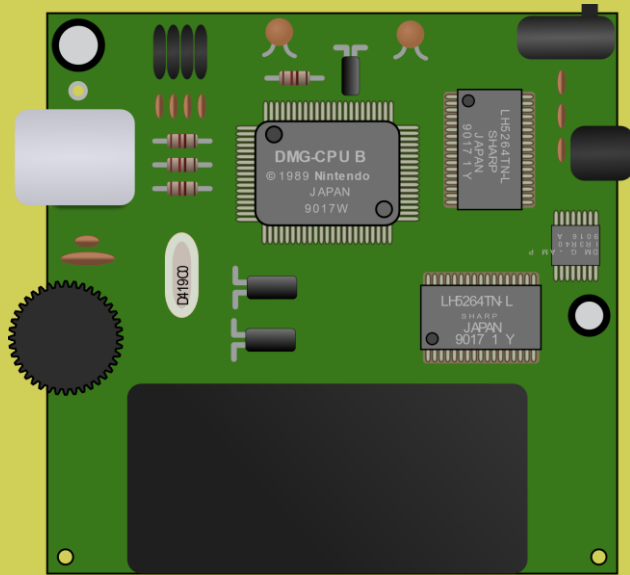
1. Screen (160x144, no backlight)

BUILDING BLOCKS



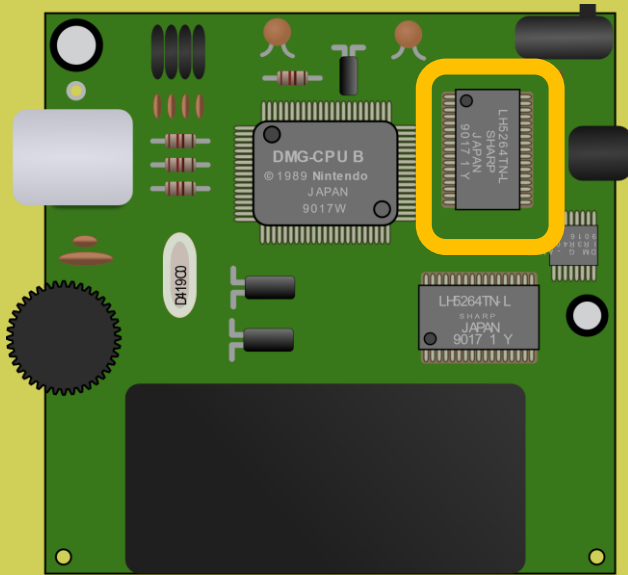
1. Screen (160x144, no backlight)
2. D-pad + buttons

BUILDING BLOCKS



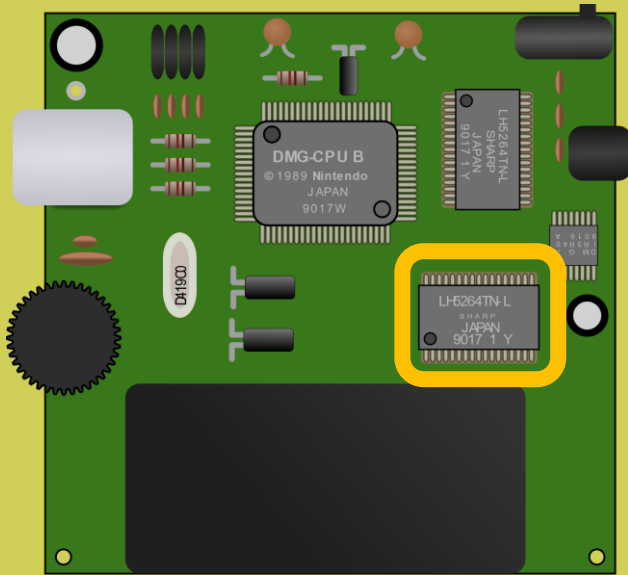
1. Screen (160x144, no backlight)
2. D-pad + buttons

BUILDING BLOCKS



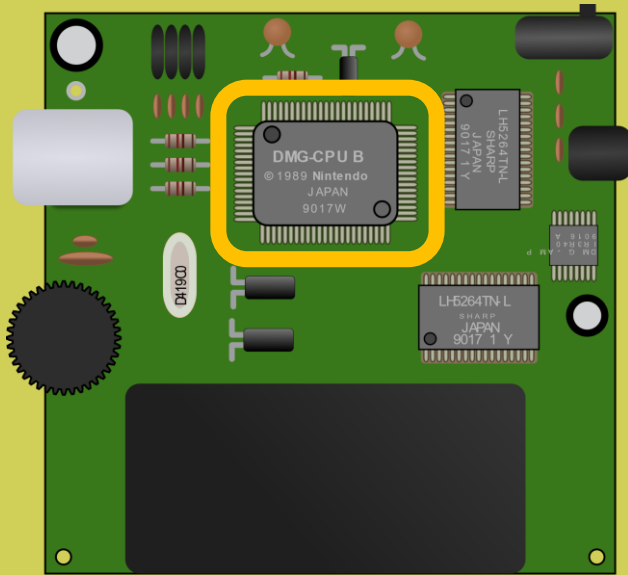
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)

BUILDING BLOCKS



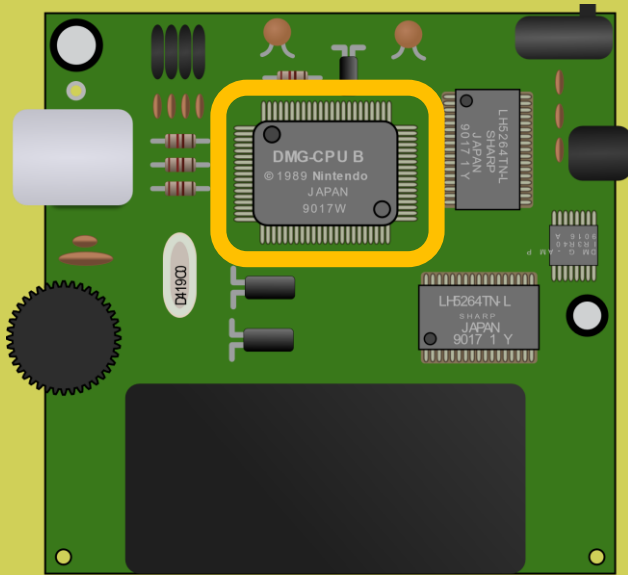
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)

BUILDING BLOCKS



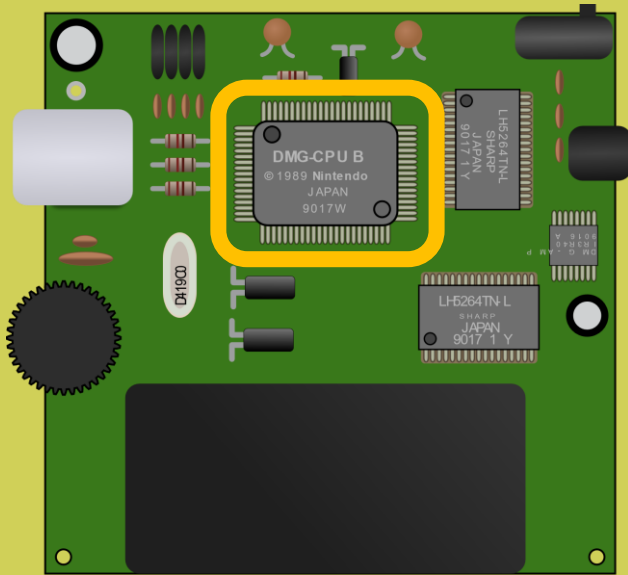
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)

BUILDING BLOCKS



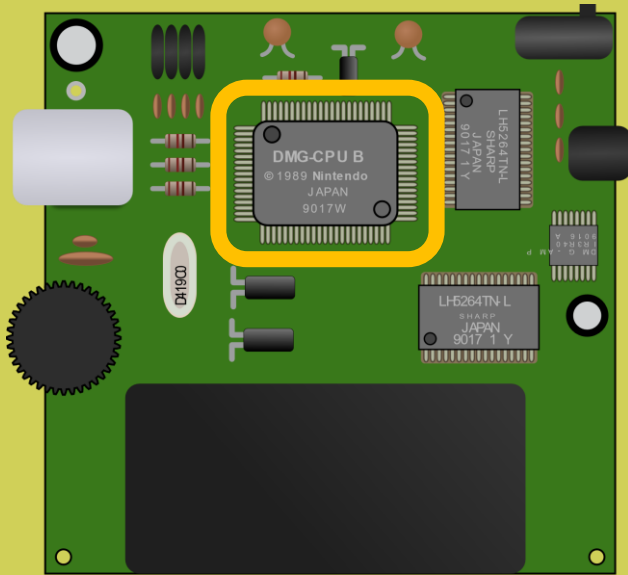
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)
6. PPU (Pixel Processing Unit)

BUILDING BLOCKS



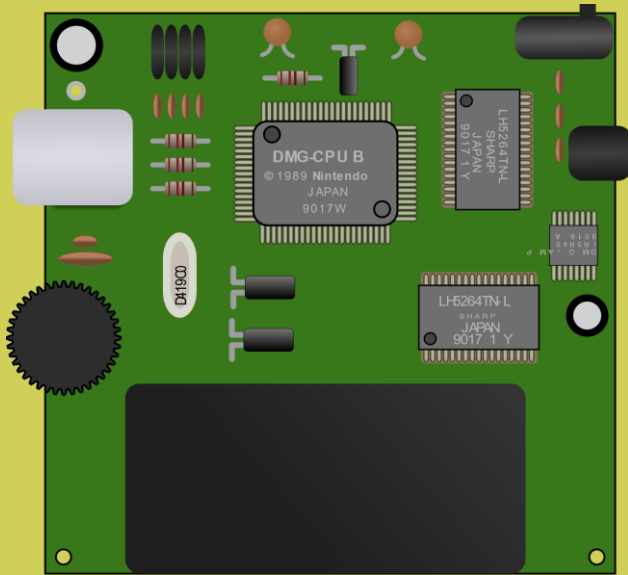
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)
6. PPU (Pixel Processing Unit)
7. APU (Audio Processing Unit)

BUILDING BLOCKS



1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)
6. PPU (Pixel Processing Unit)
7. APU (Audio Processing Unit)
8. Boot ROM

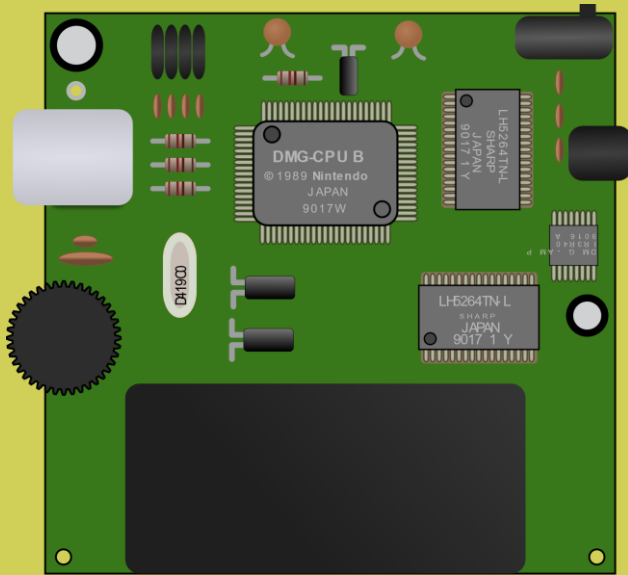
BUILDING BLOCKS



- 1.Screen (160x144, no backlight)
- 2.D-pad + buttons
- 3.RAM (8 kiB)
- 4.VRAM (8 kiB)
- 5.CPU (8-bit Z80 @ 4MHz)
- 6.PPU (Pixel Processing Unit)
- 7.APU (Audio Processing Unit)
- 8.Boot ROM

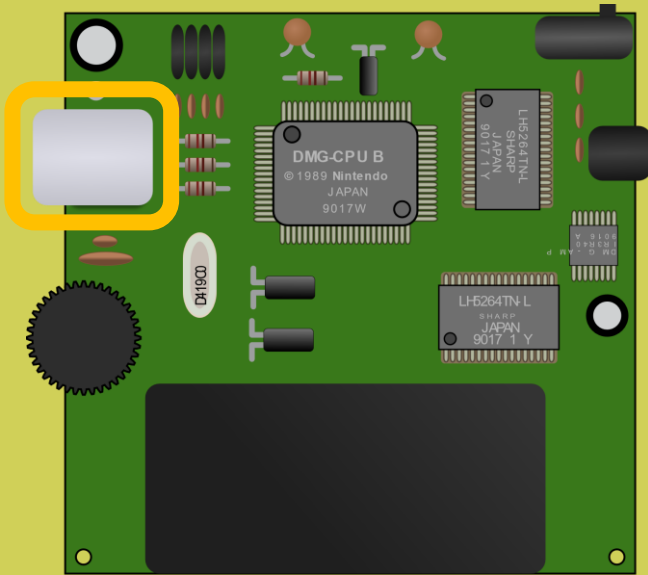


BUILDING BLOCKS



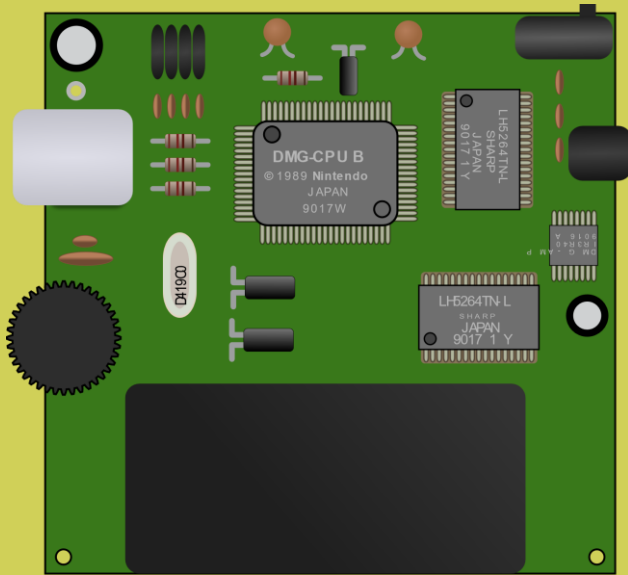
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)
6. PPU (Pixel Processing Unit)
7. APU (Audio Processing Unit)
8. Boot ROM
9. Cartridge

BUILDING BLOCKS



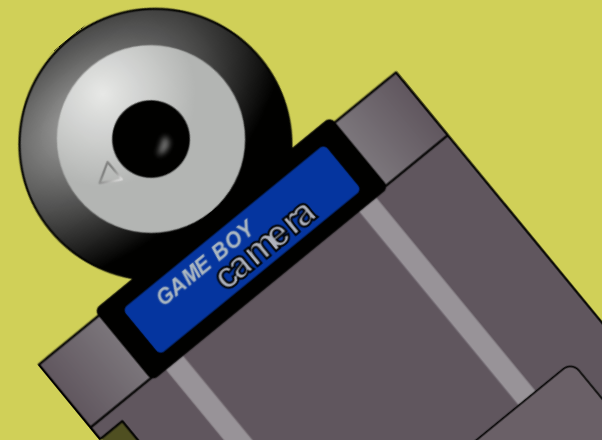
1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)
6. PPU (Pixel Processing Unit)
7. APU (Audio Processing Unit)
8. Boot ROM
9. Cartridge
10. Serial I/O

BUILDING BLOCKS



1. Screen (160x144, no backlight)
2. D-pad + buttons
3. RAM (8 kiB)
4. VRAM (8 kiB)
5. CPU (8-bit Z80 @ 4MHz)
6. PPU (Pixel Processing Unit)
7. APU (Audio Processing Unit)
8. Boot ROM
9. Cartridge
10. Serial I/O
11. Camera

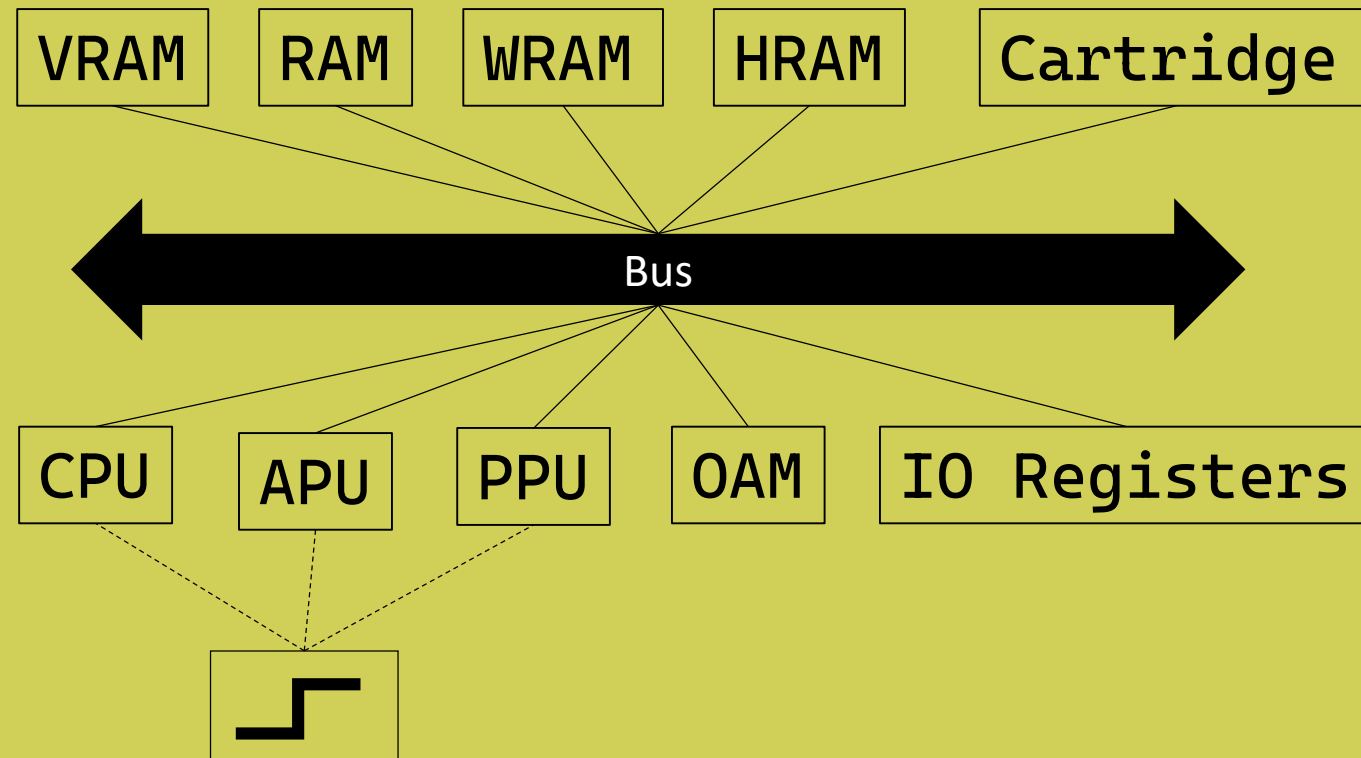
...



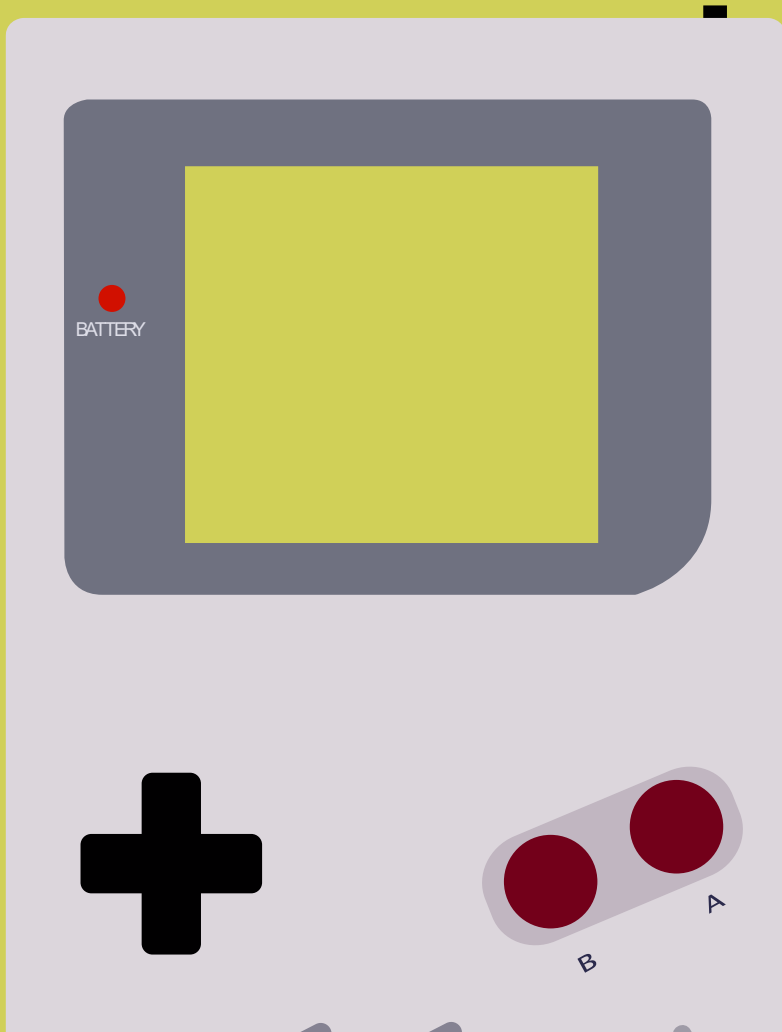
MEMORY MAP

Description	Start	End
Cartridge fixed bank	0000	3fff
Cartridge switchable bank	4000	7fff
8 kiB VRAM	8000	9fff
8 kiB external RAM	a000	bfff
8 kiB internal RAM	c000	dfff
Echo RAM	e000	fdff
Object Attribute Memory (OAM)	fe00	fe9f
Don't use	fea0	feff
I/O registers	ff00	ff7f
High RAM (HRAM)	ff80	ffffe
Interrupt Enable (IE) register	ffff	ffff

TLMB0Y STRUCTURE

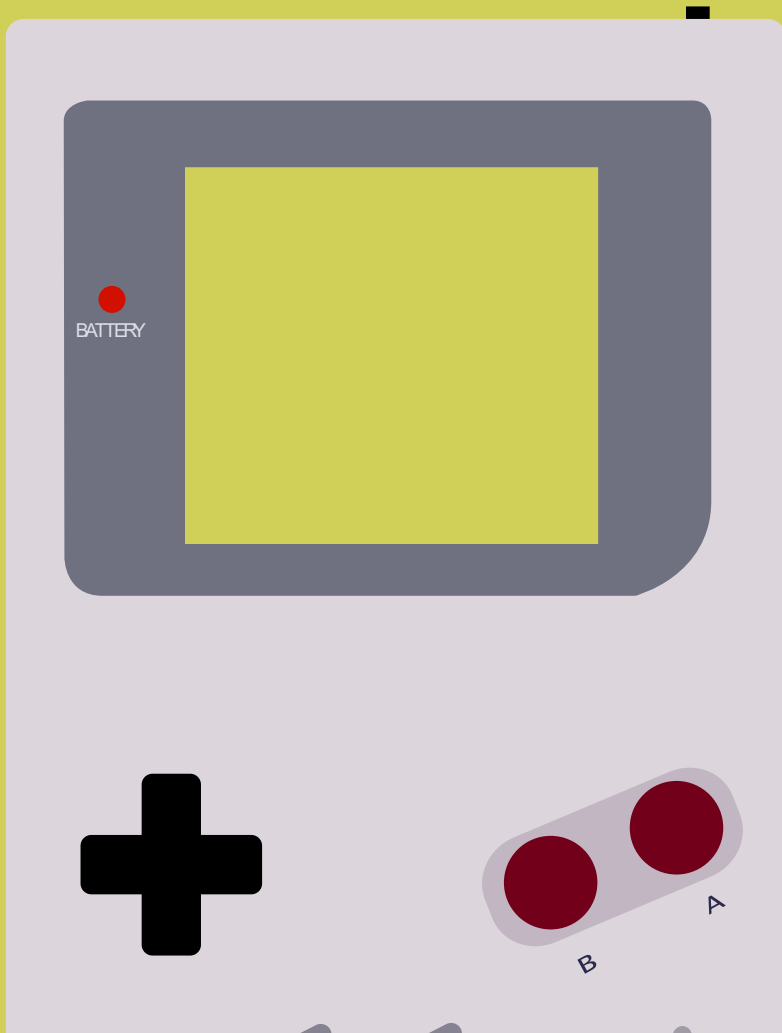


HOW TO BOOT



HOW TO BOOT

Click!



HOW TO BOOT



HOW TO BOOT



Memory

← PC=0x0000

16 kiB Cartridge

16 kiB Cartridge Bank N

8 kiB VRAM

...

HOW TO BOOT



Memory

256B Boot ROM
16 kiB Cartridge
16 kiB Cartridge Bank N
8 kiB VRAM
...

← PC=0x0000

HOW TO BOOT



Memory

256B Boot ROM
16 kiB Cartridge
16 kiB Cartridge Bank N
8 kiB VRAM
...

← PC=0x00FE

[0xff50] = 1

HOW TO BOOT



Memory

16 kiB Cartridge

16 kiB Cartridge Bank N

8 kiB VRAM

...

← PC=0x0100

[0xff50] = 1

HOW TO BOOT

Memory

256B Boot ROM

16 kiB Cartridge

16 kiB Cartridge Bank N

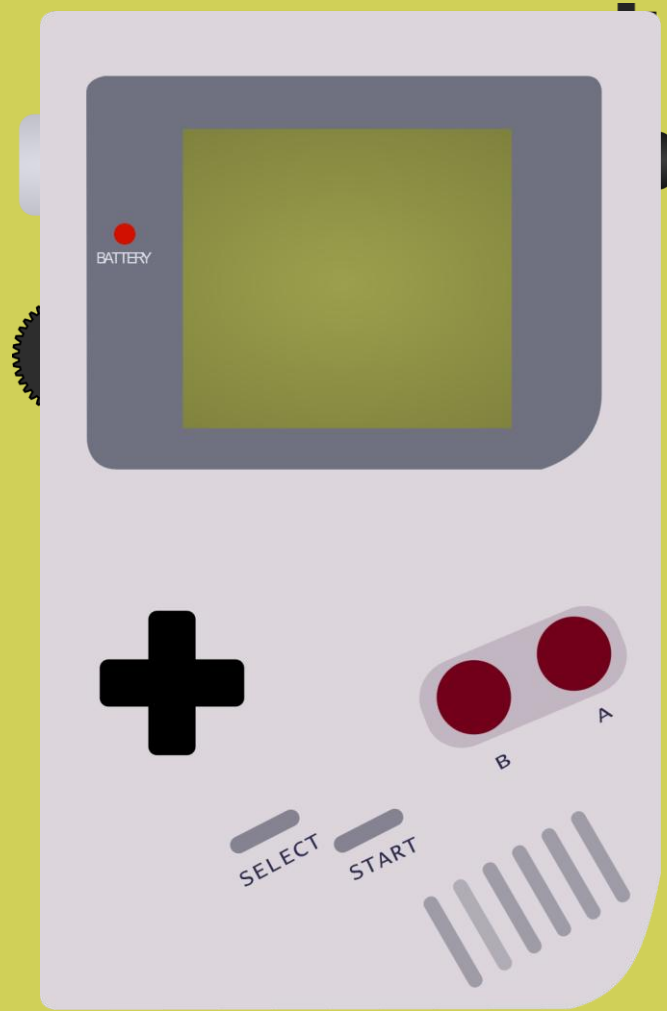
8 kiB VRAM

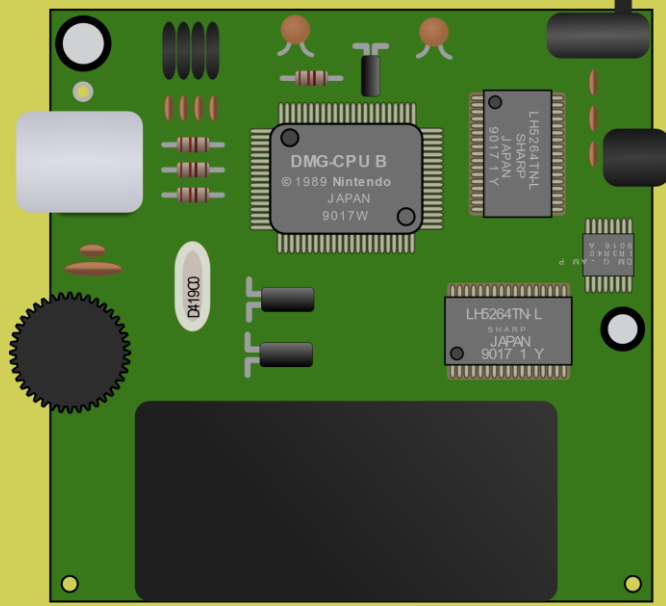
...

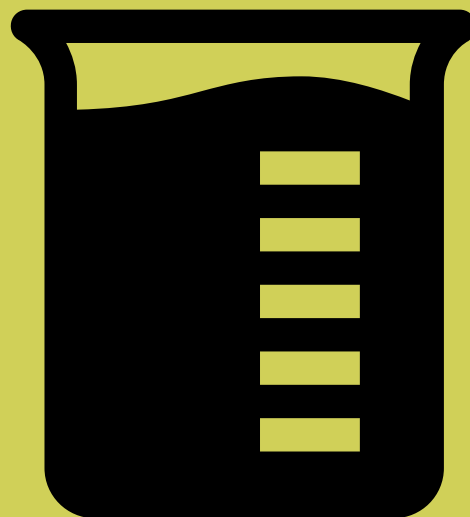
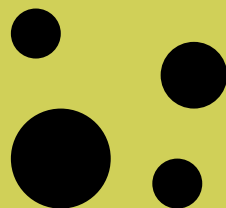
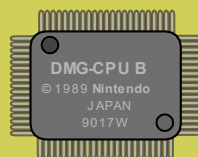
What does it do?
How to get it?



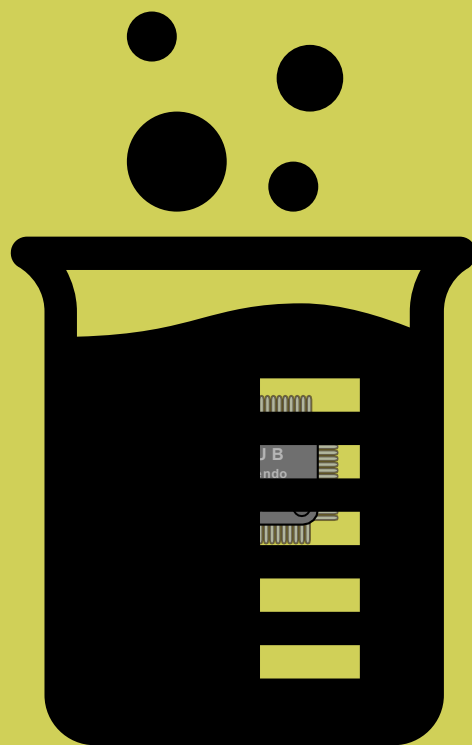
2003







HN03



HN03



HNO_3

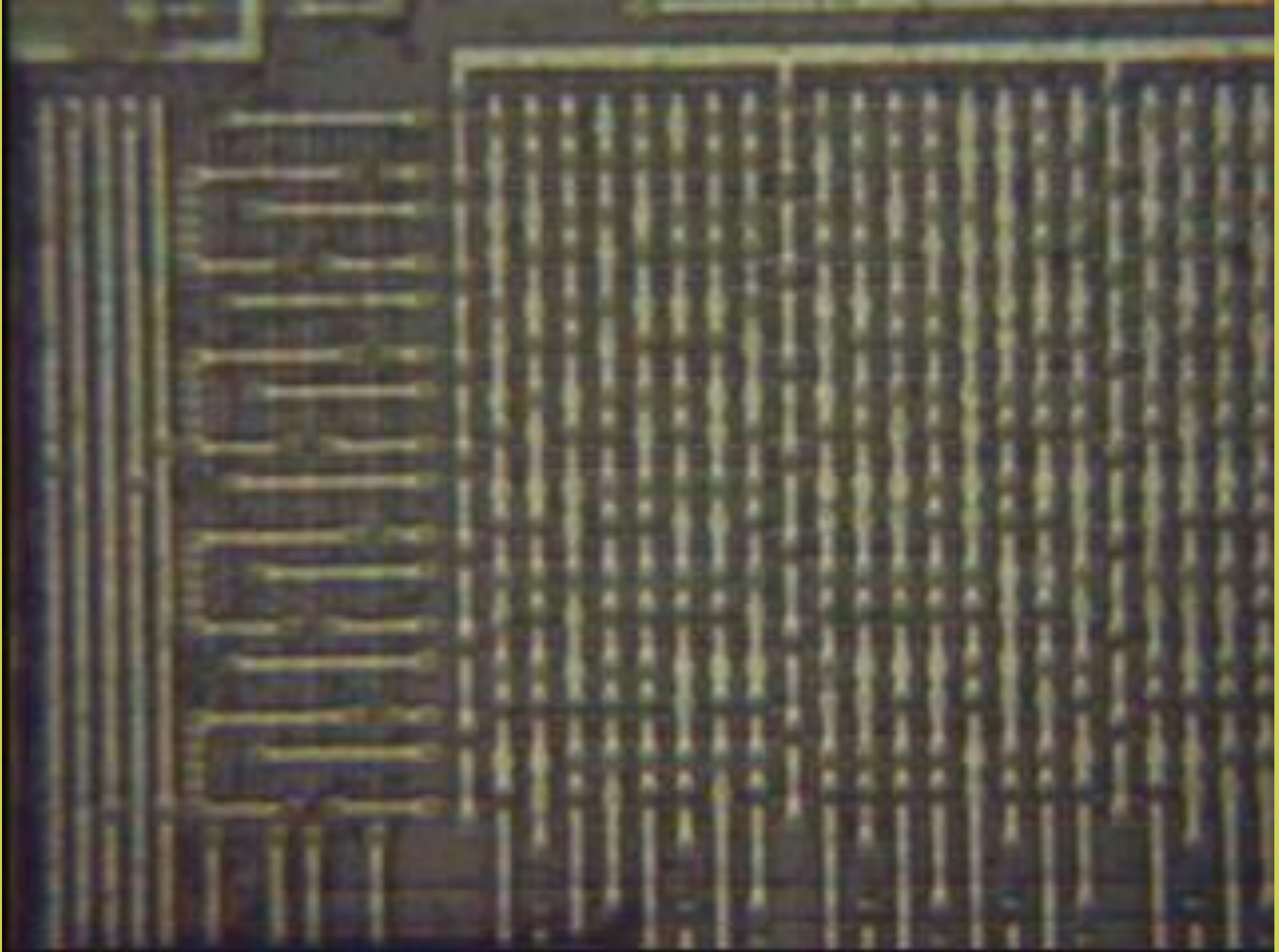




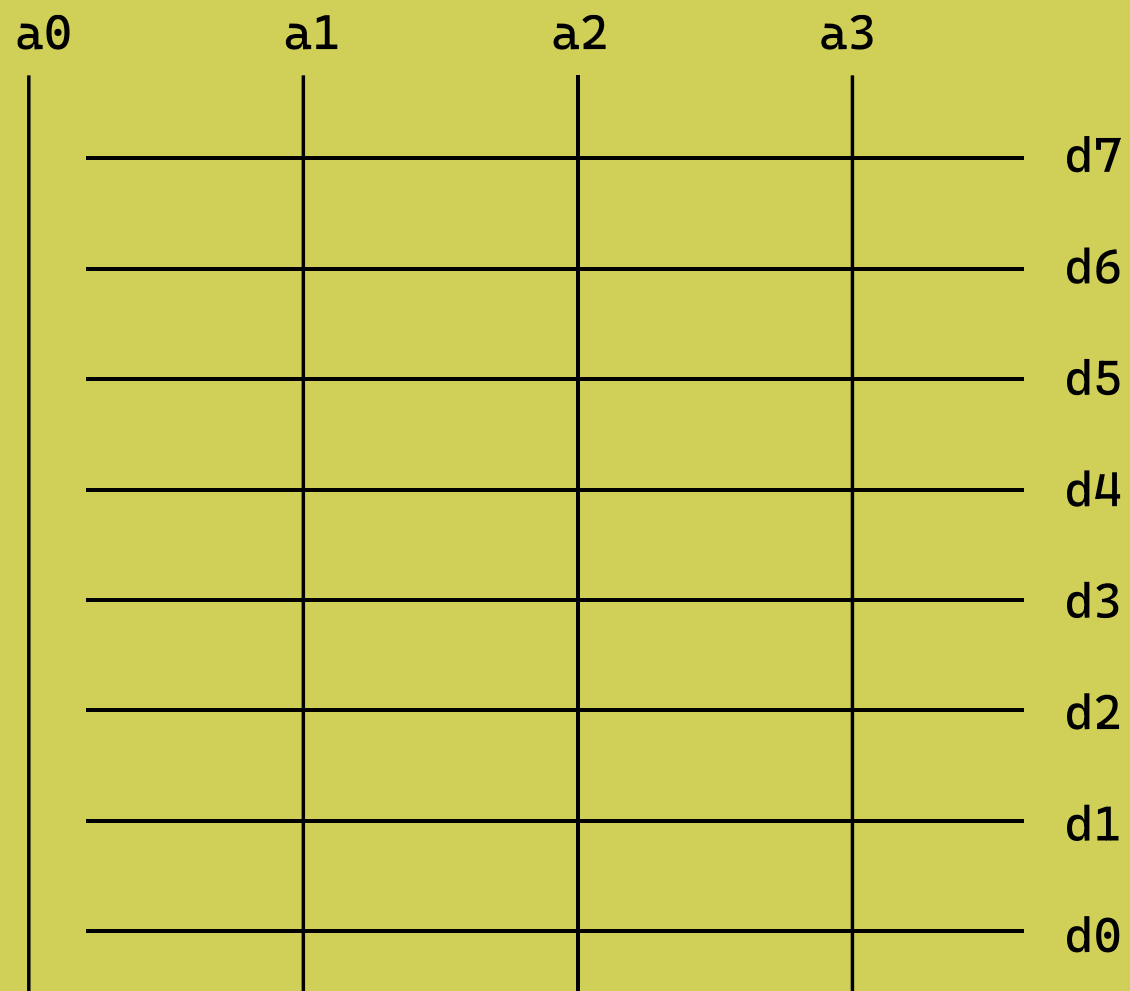
source: <https://www.neviksti.com/DMG/>

16 Bytes

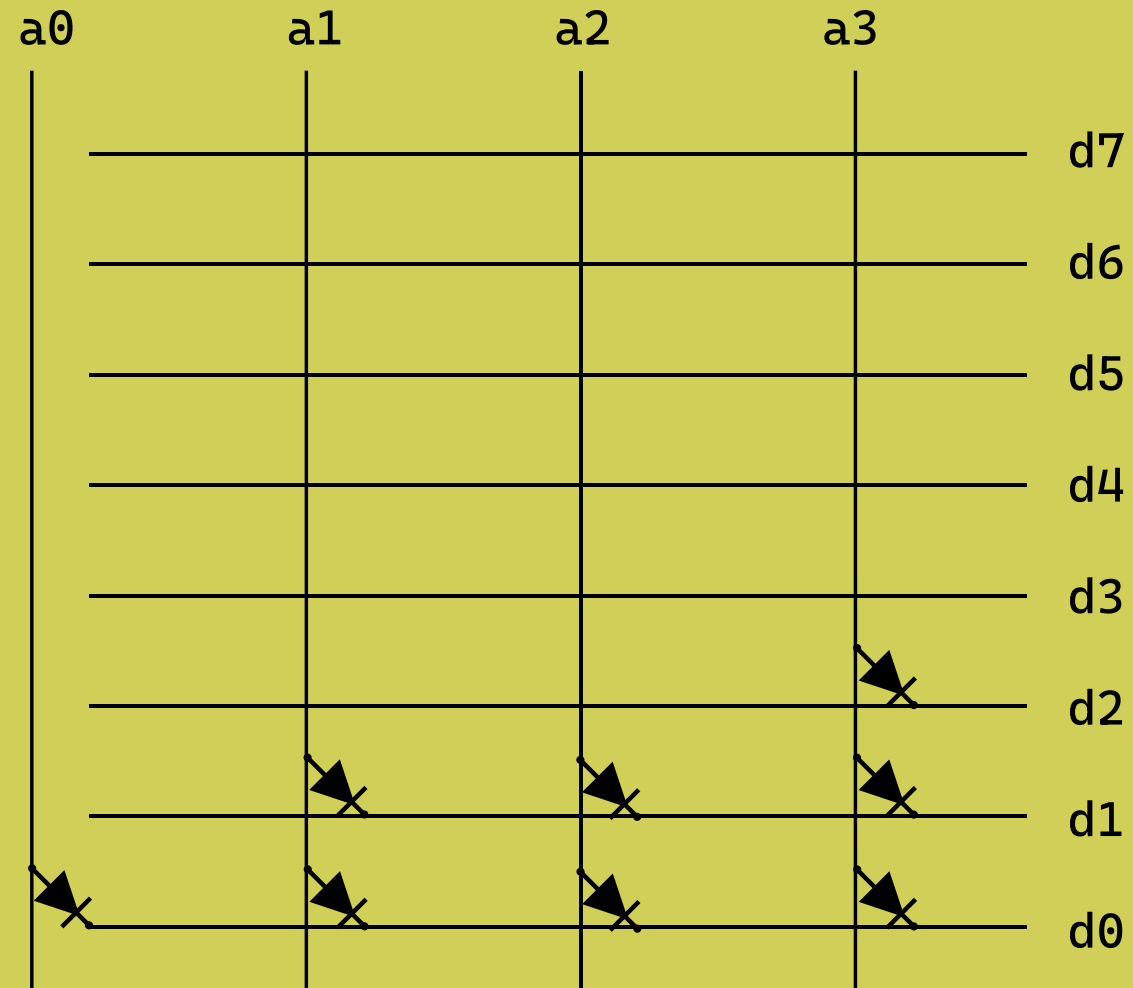
256 Bytes



MASKED ROM

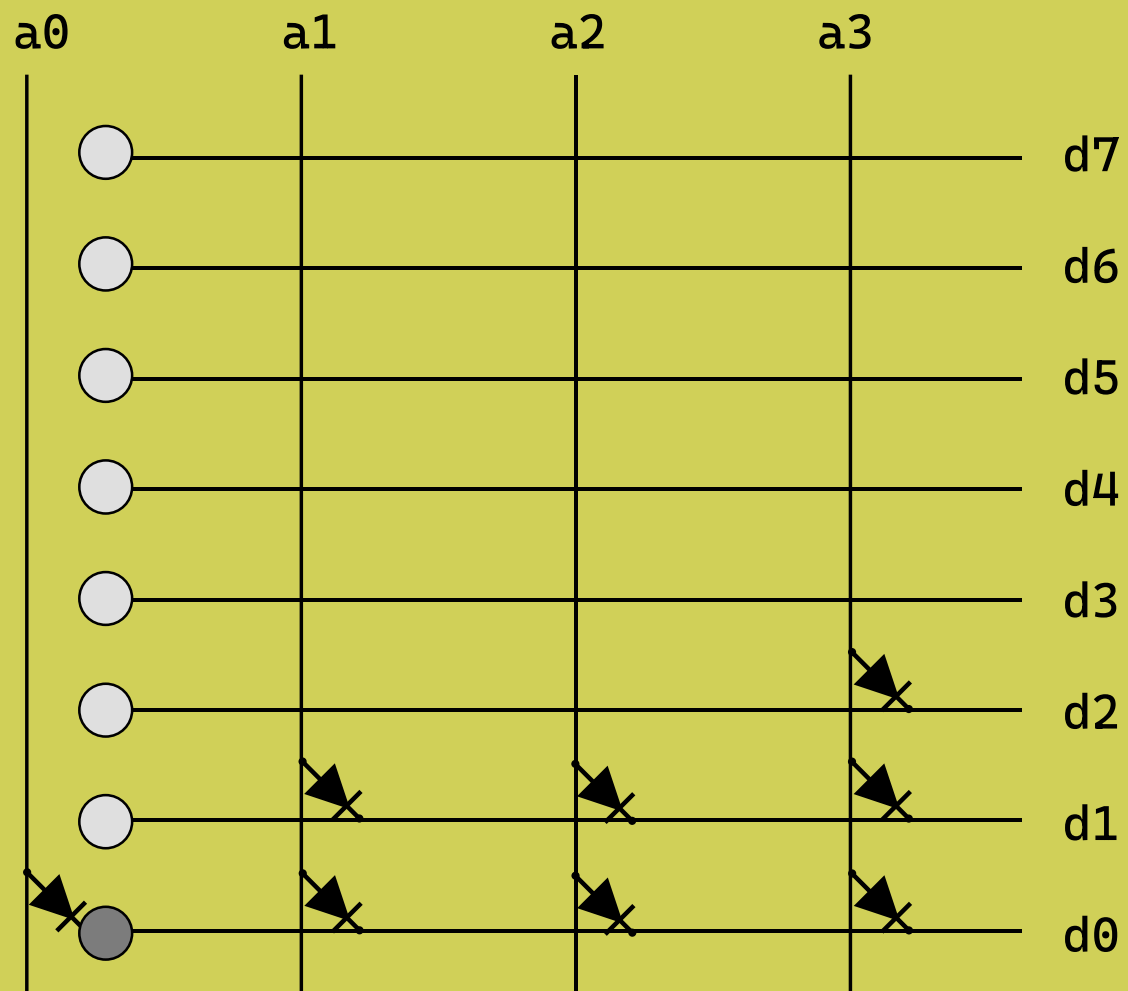


MASKED ROM



MASKED ROM

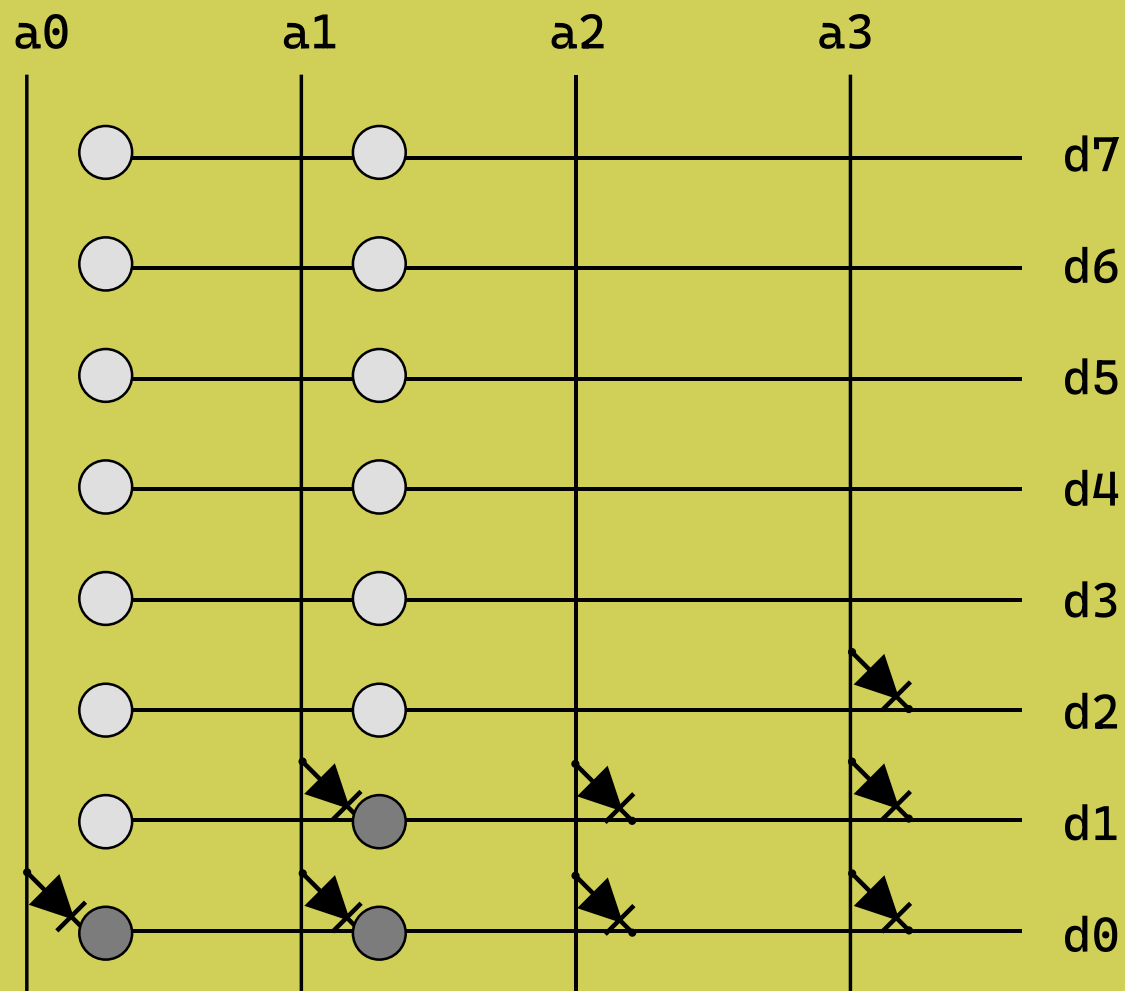
a0 = 0x01



MASKED ROM

a0 = 0x01

a1 = 0x03

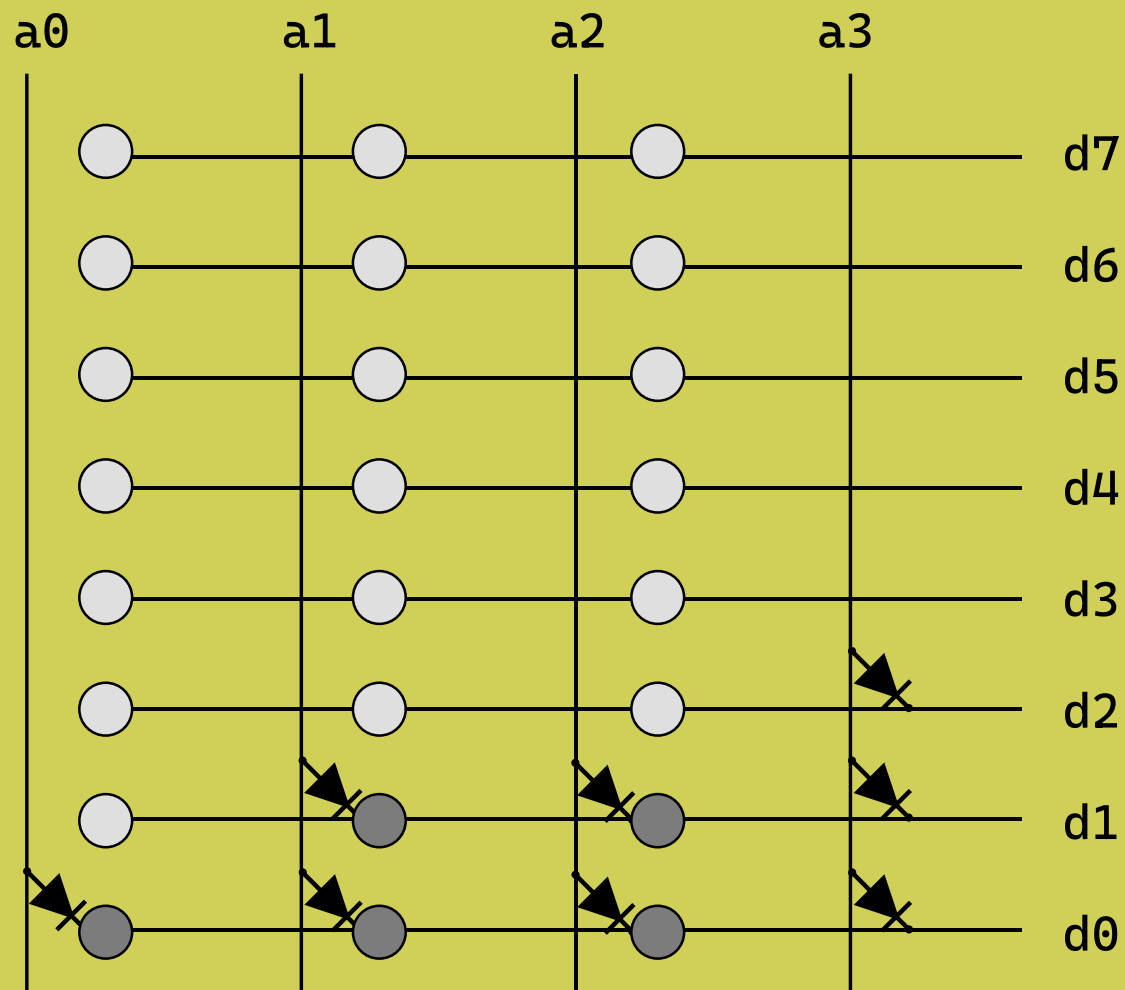


MASKED ROM

a0 = 0x01

a1 = 0x03

a2 = 0x03



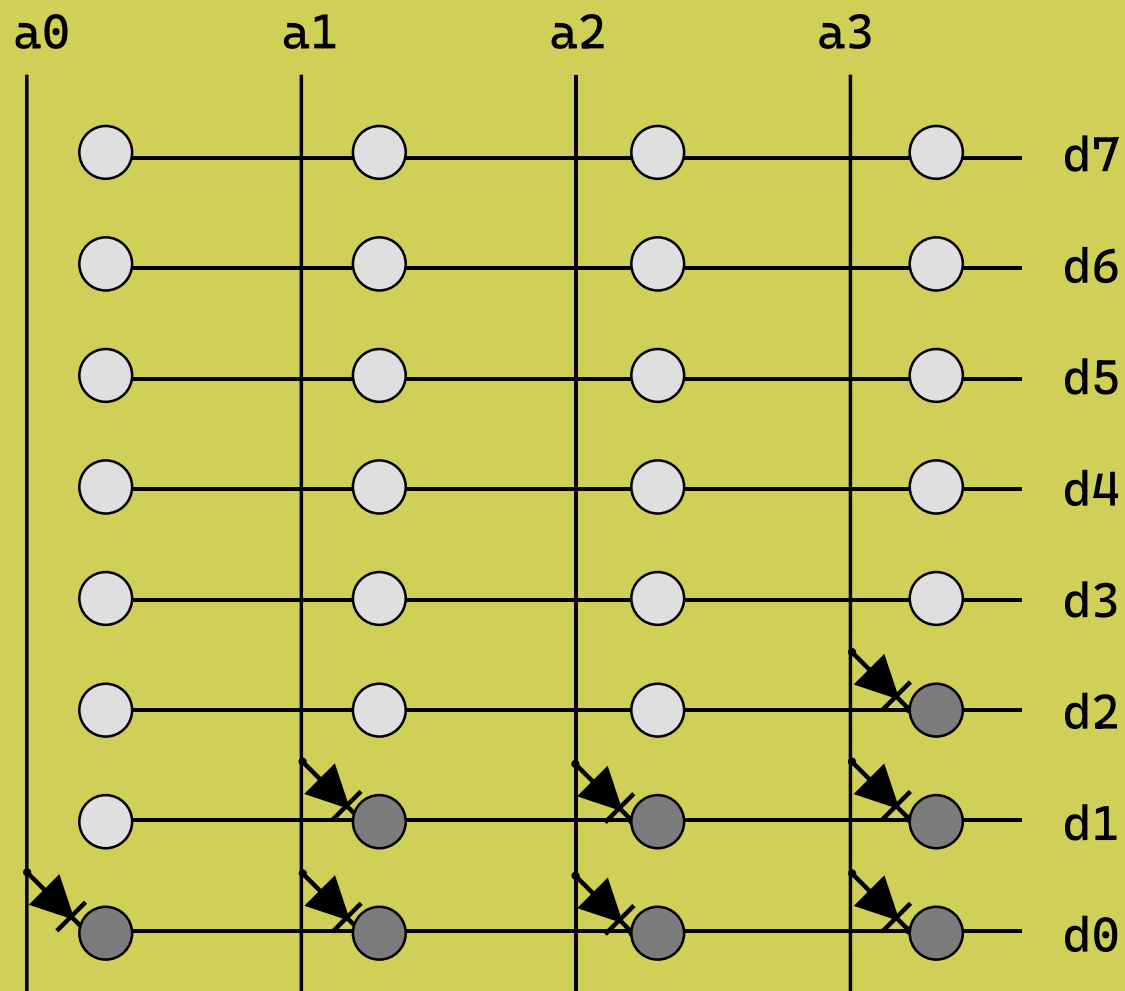
MASKED ROM

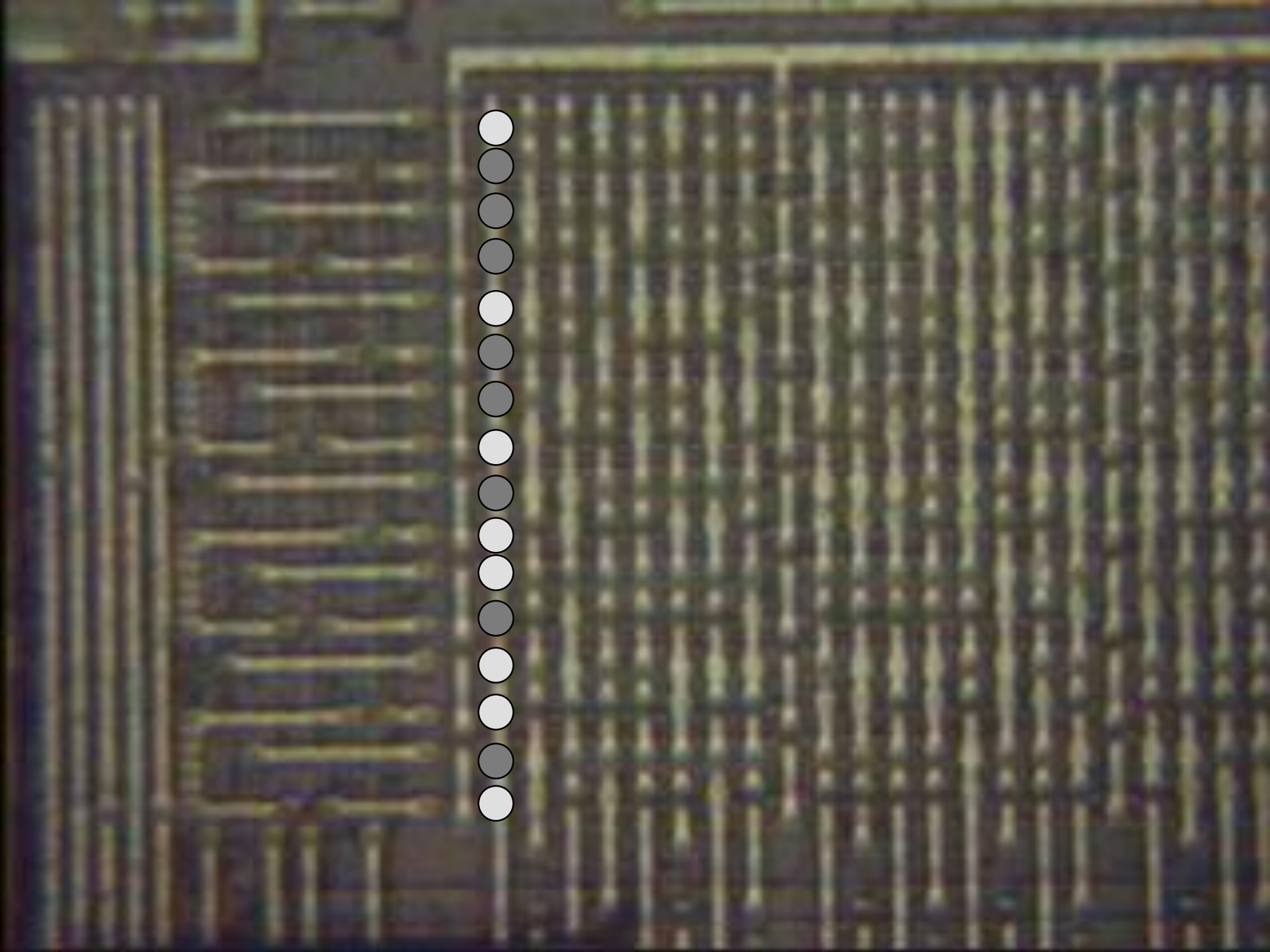
a0 = 0x01

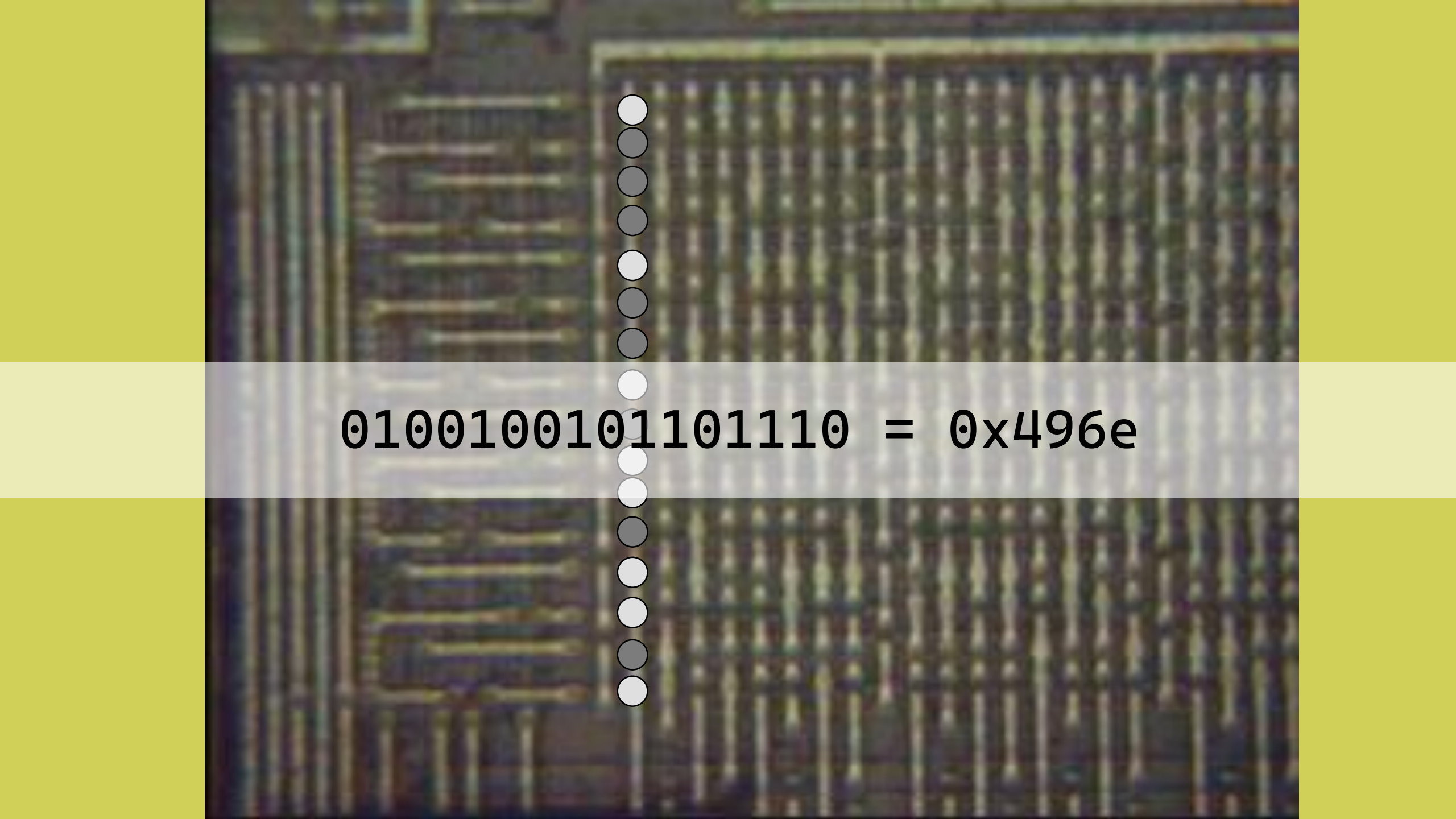
a1 = 0x03

a2 = 0x03

a3 = 0x07





The background of the slide is a close-up photograph of a microchip, showing its intricate circuitry and gold-colored pins. A vertical column of 16 circles is positioned in the center of the slide. The circles are arranged in a single column, with the 8th circle from the top being white and the others being gray. A semi-transparent horizontal bar is overlaid on the middle of the slide, containing the text '0100100101101110 = 0x496e'.

0100100101101110 = 0x496e

<https://github.com/travisgoodspeed/maskromtool>

 **maskromtool** Public

Watch 13 Fork 21 Starred 345

master 1 Branch 12 Tags

Add file Code

 **travisgoodspeed** typo ✓

e86fd57 · last week 382 Commits

 .github/workflows	Disabling macOS in CI/CD builds as we can't sign them.	6 months ago
 Deployment	Version bump.	5 months ago
 gatotests	Disassembler now supports Radare2 and GoodASM. Some ar...	7 months ago
 icons.iconset	First post!	3 years ago
 screenshots	Updated instructions, solver screenshot.	7 months ago
 tests	Better tests.	3 years ago
 .gitignore	Ignore .asm files from the GatoROM tests.	6 months ago
 CMakeLists.txt	RomAlignerTilting implementing a rotation matrix. #120	5 months ago

About

A CAD tool for extracting bits from Mask ROM photographs.

reverse-engineering rom maskrom

 Readme

 View license

 Activity

 344 stars

 13 watching

 21 forks

[Report repository](#)

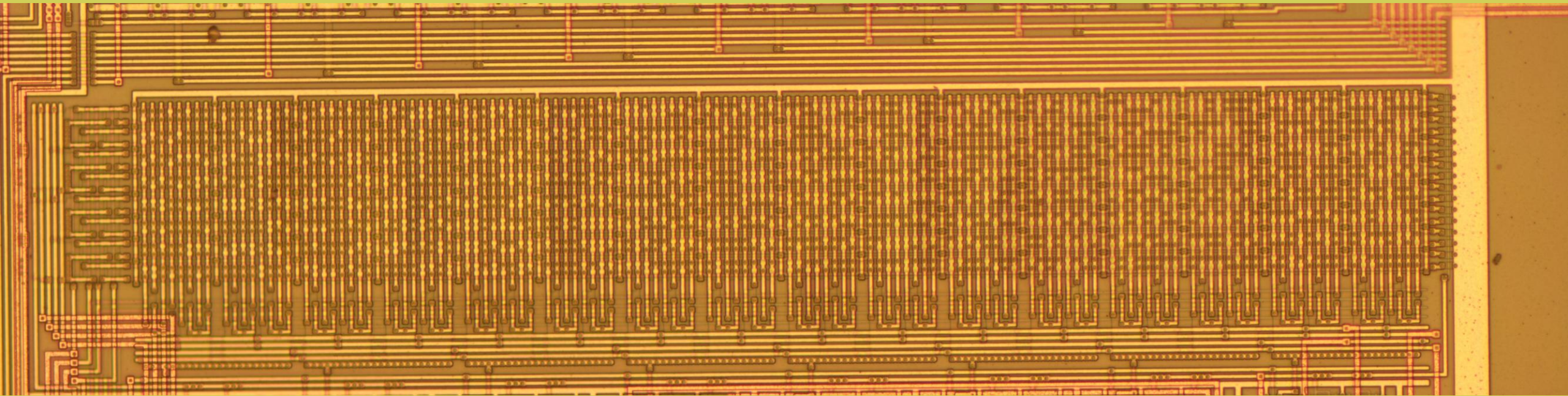
Releases

12

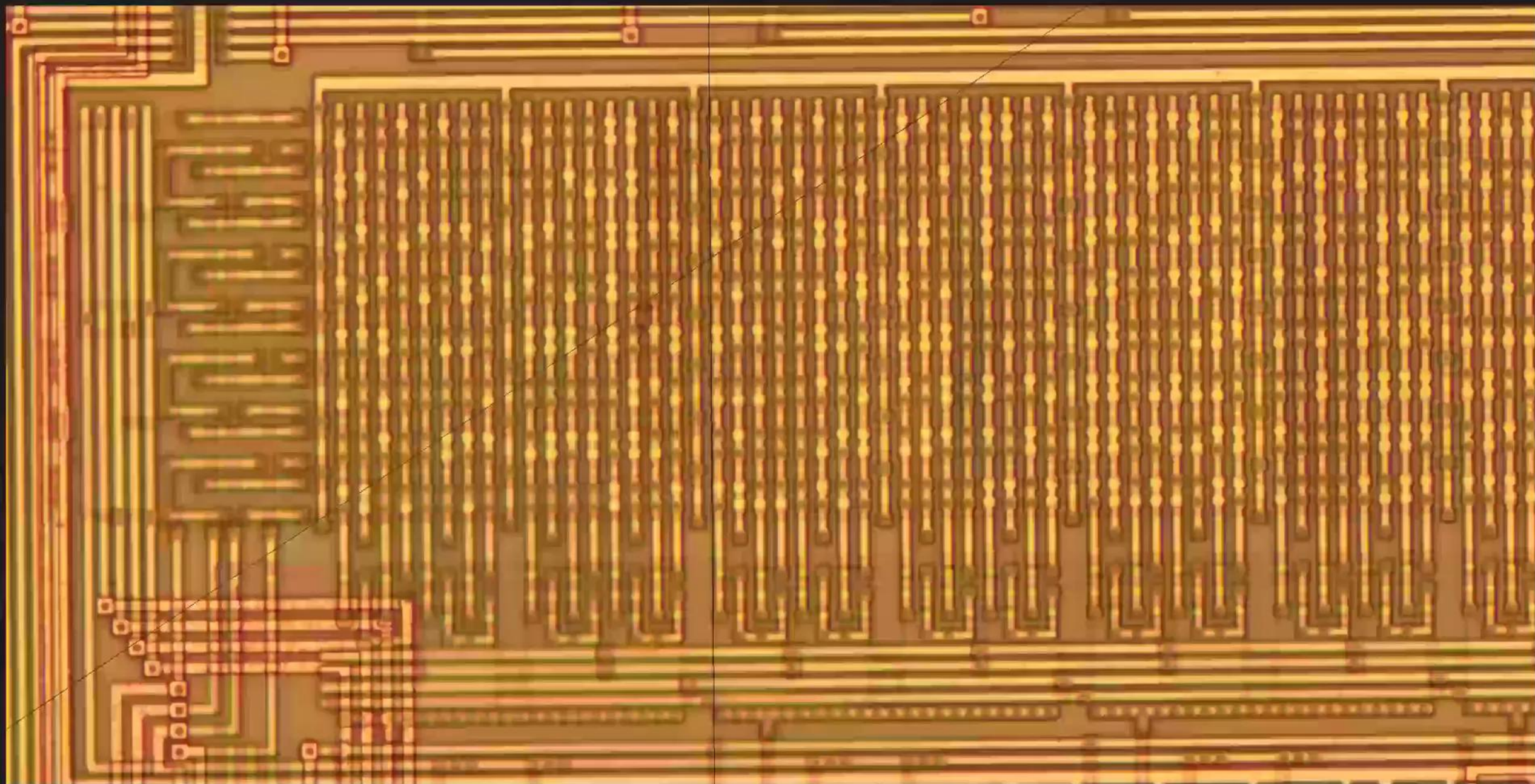
 v2024-08-18

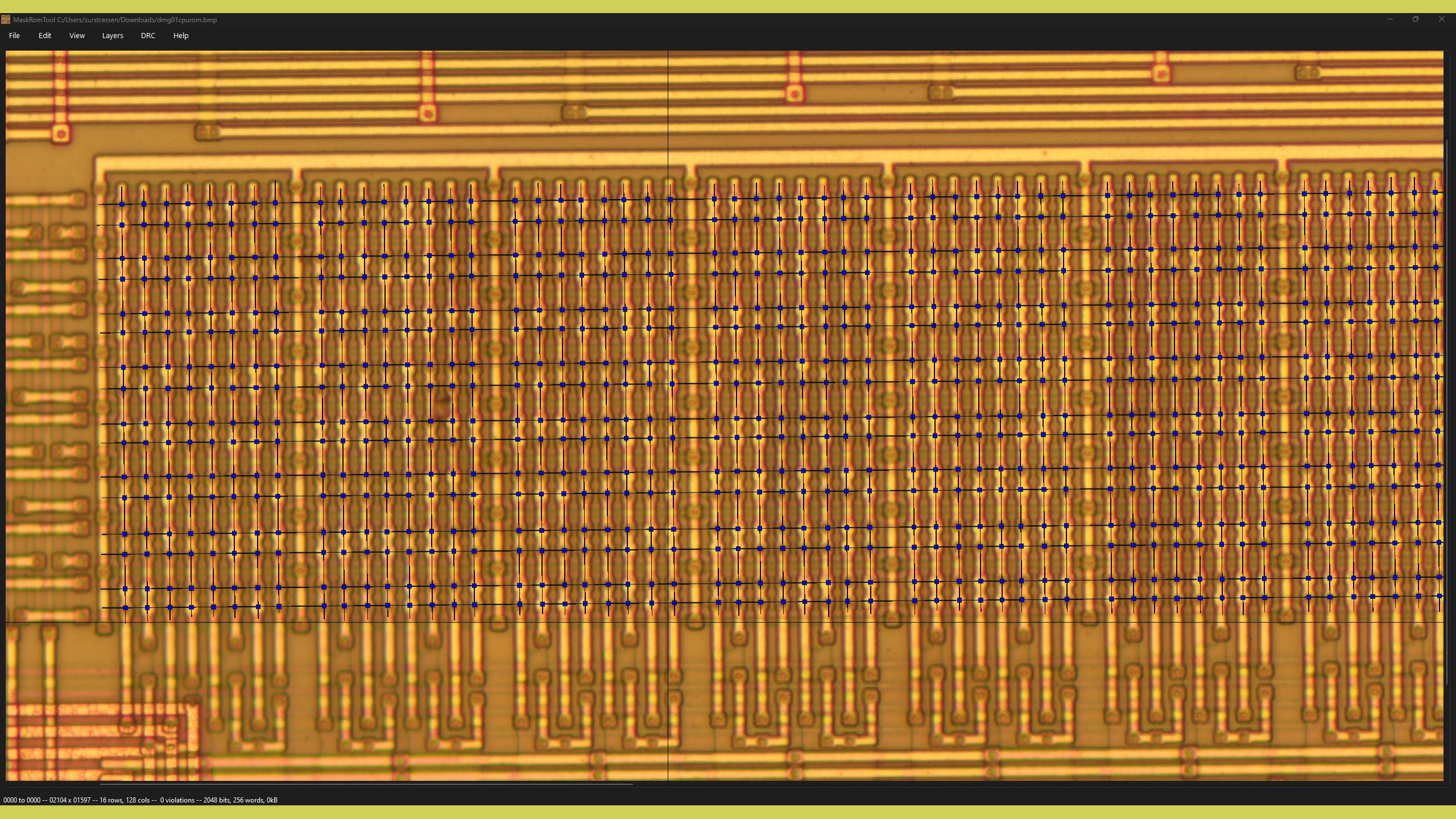
Latest

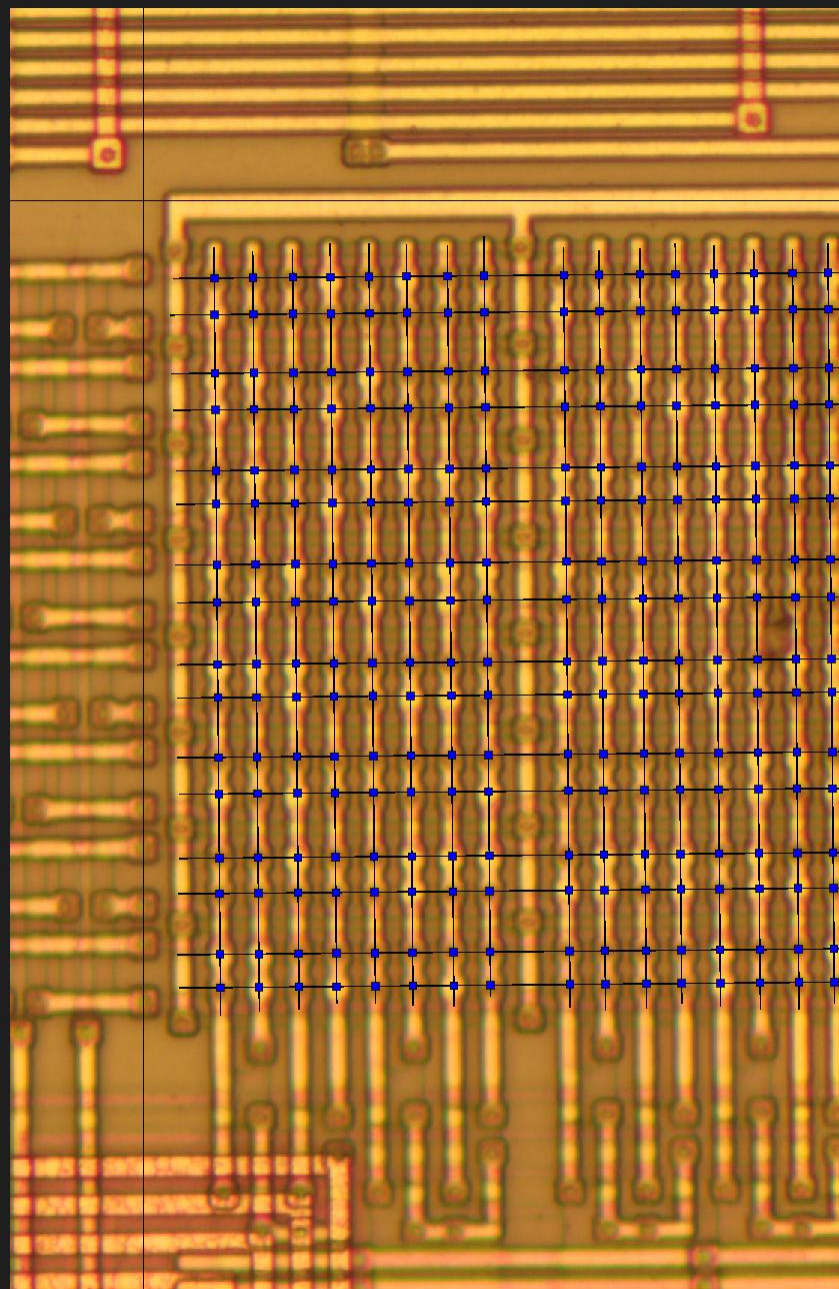
DIE SHOT



<https://github.com/travisgoodspeed/gbrom-tutorial>







Settings

Sampling **Display**

Red

Green

Blue

Sampler

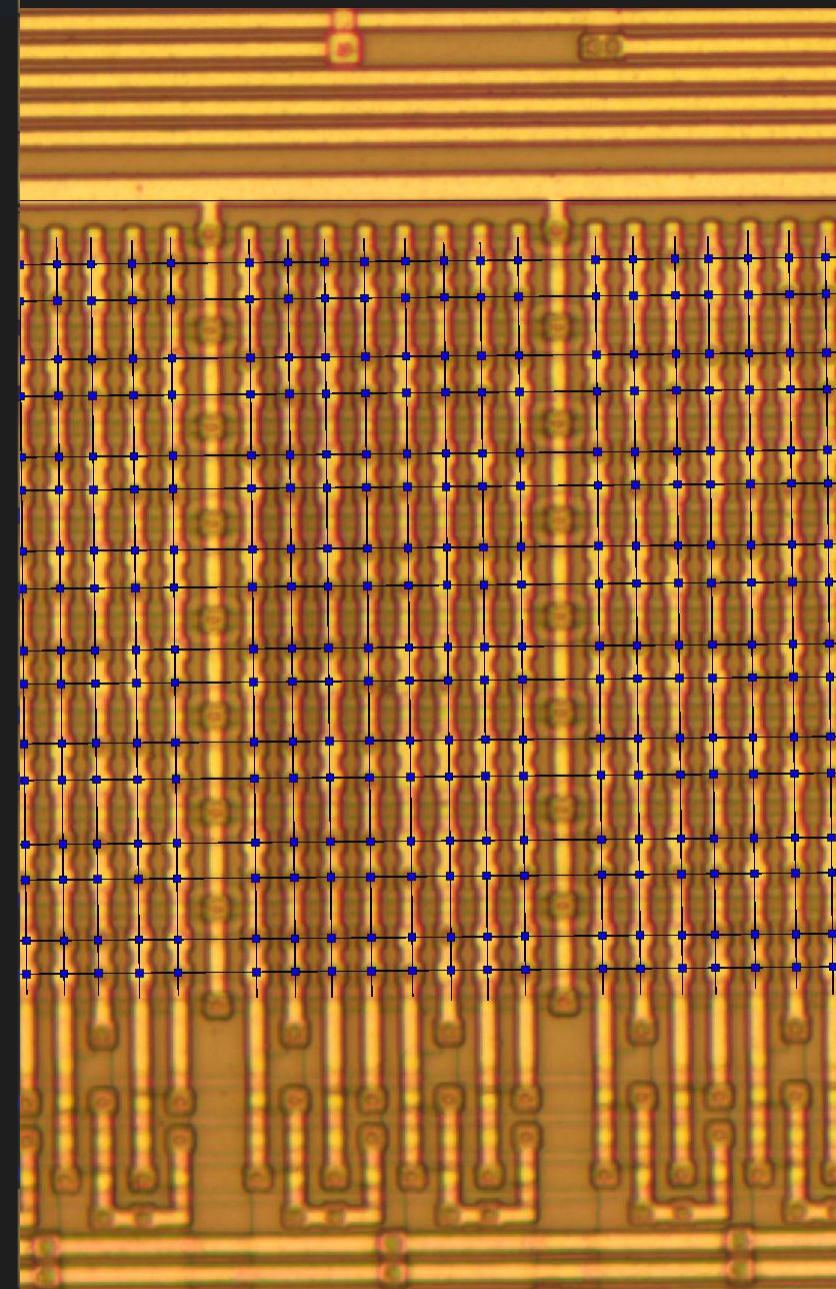
Size

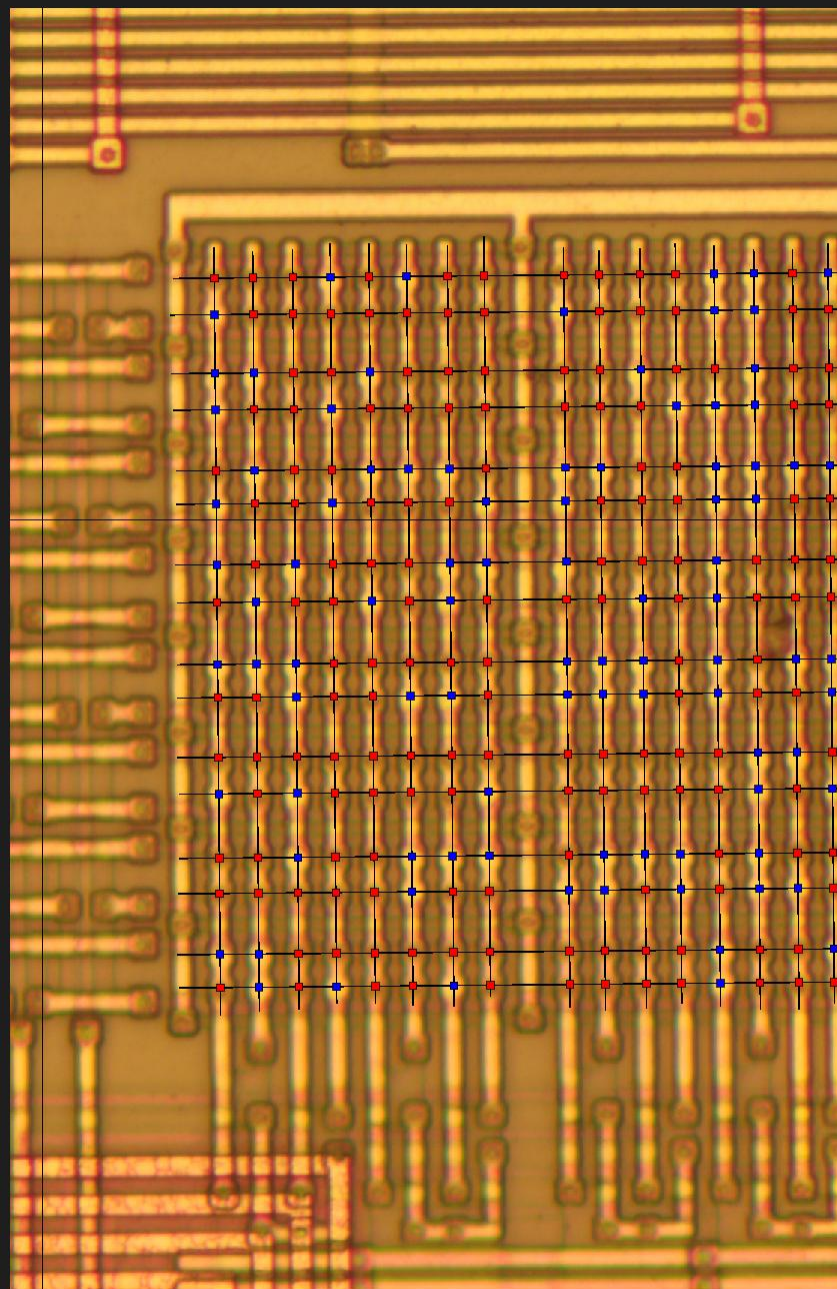
☐ Inverted Bits

Bit Histogram

The Bit Histogram graph displays the distribution of bit values for the Red, Green, and Blue channels. The x-axis represents the bit value (0.0 to 255.0) and the y-axis represents the frequency (0.0 to 118.0). The Red channel (red line) shows a peak around 191.3. The Green channel (green line) shows a peak around 127.5. The Blue channel (blue line) shows a peak around 63.8.

Channel	Peak Value (approx.)	Peak Frequency (approx.)
Red	191.3	59.0
Green	127.5	59.0
Blue	63.8	118.0





Settings

Sampling Display

Red — 0

Green — 57

Blue — 0

Average

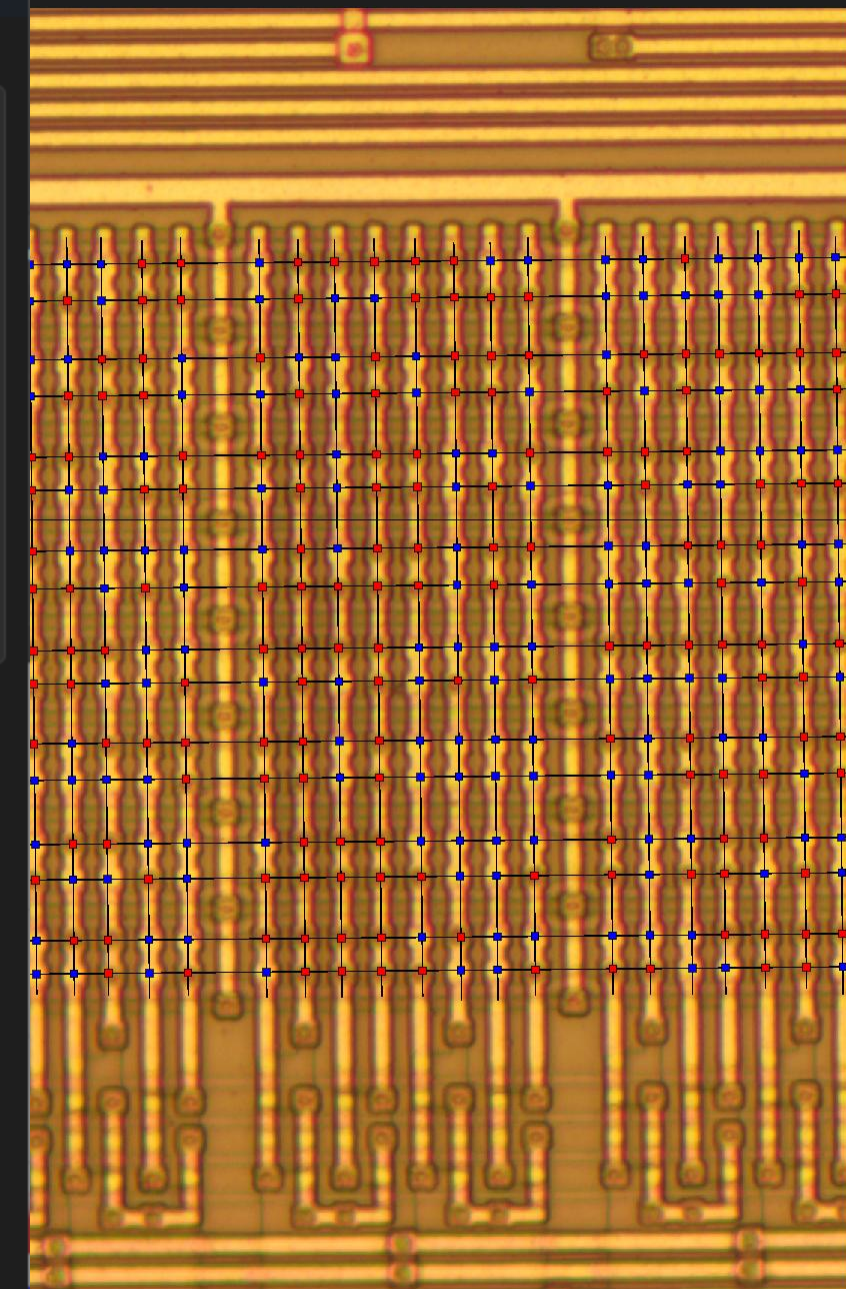
Sampler Default

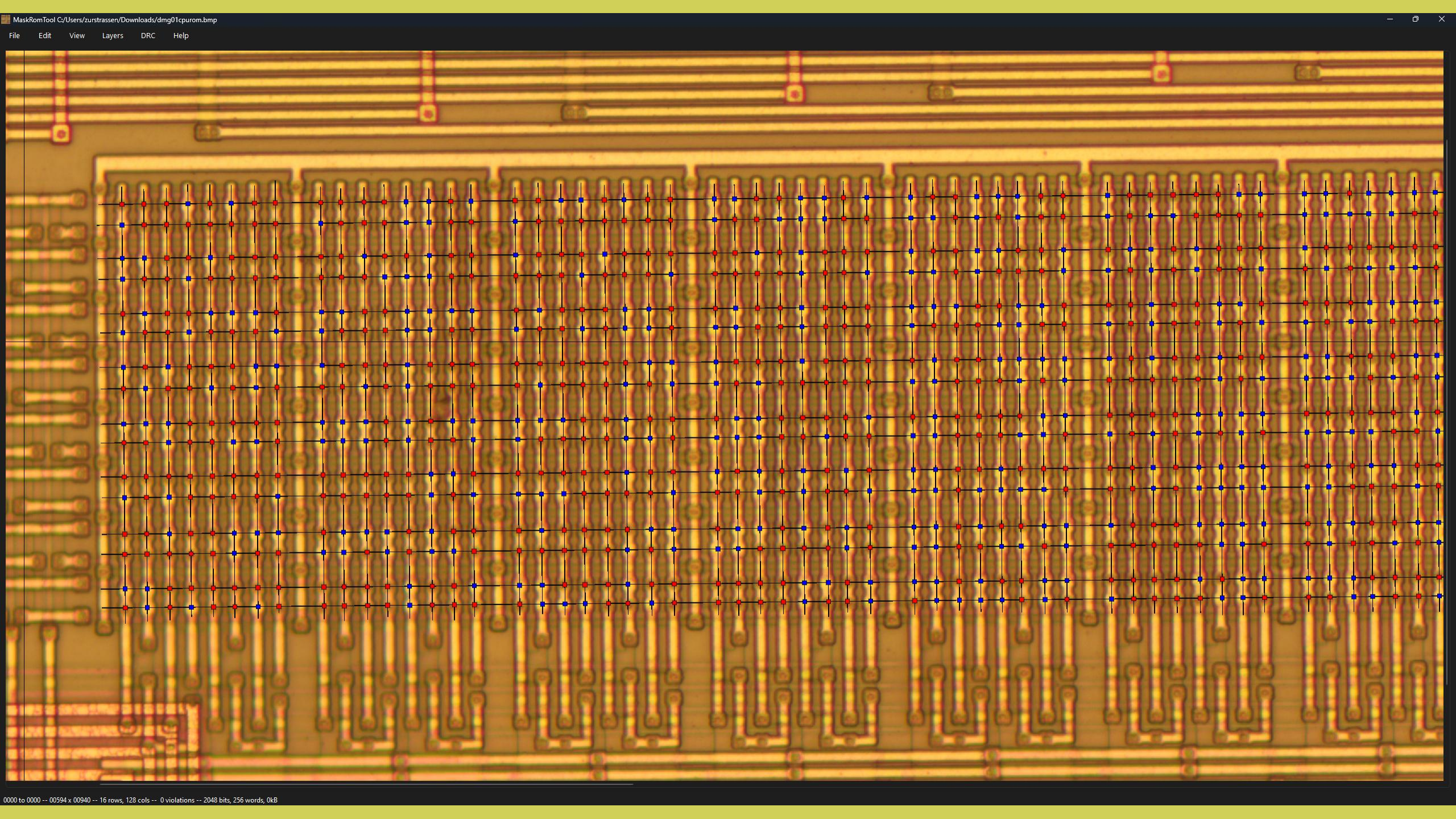
Size 2

☐ Inverted Bits

Bit Histogram

The Bit Histogram graph displays three curves: a blue curve peaking at approximately 118.0 around x=63.8, a green curve peaking at approximately 59.0 around x=127.5, and a red curve peaking at approximately 59.0 around x=191.3. The x-axis ranges from 0.0 to 255.0, and the y-axis ranges from 0.0 to 118.0.





Solver

BytesByte StringASCIIYara

This solves for optionally masked bytes at absolute addresses. Comma or space delimited, the records are colon separated with an address, a byte, and an optional mask. Use the Byte String solver if you know the bytes, but not their address.

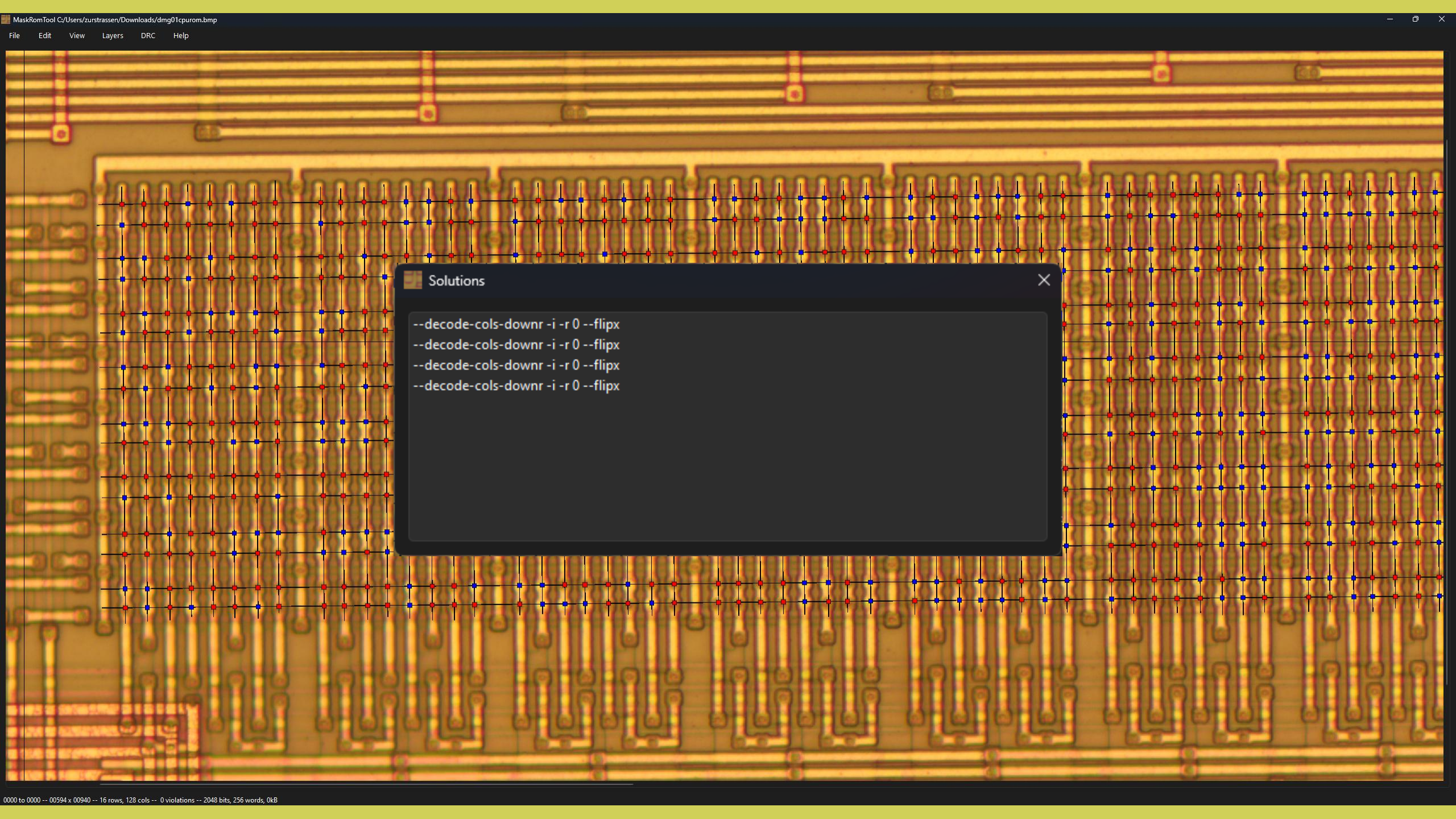
Examples:
GameBoy: 0:31,1:fe,2:ff
MYK82 Clipper Chip: 0:9b,7:ea

Query:

0:31,1:fe,2:ff

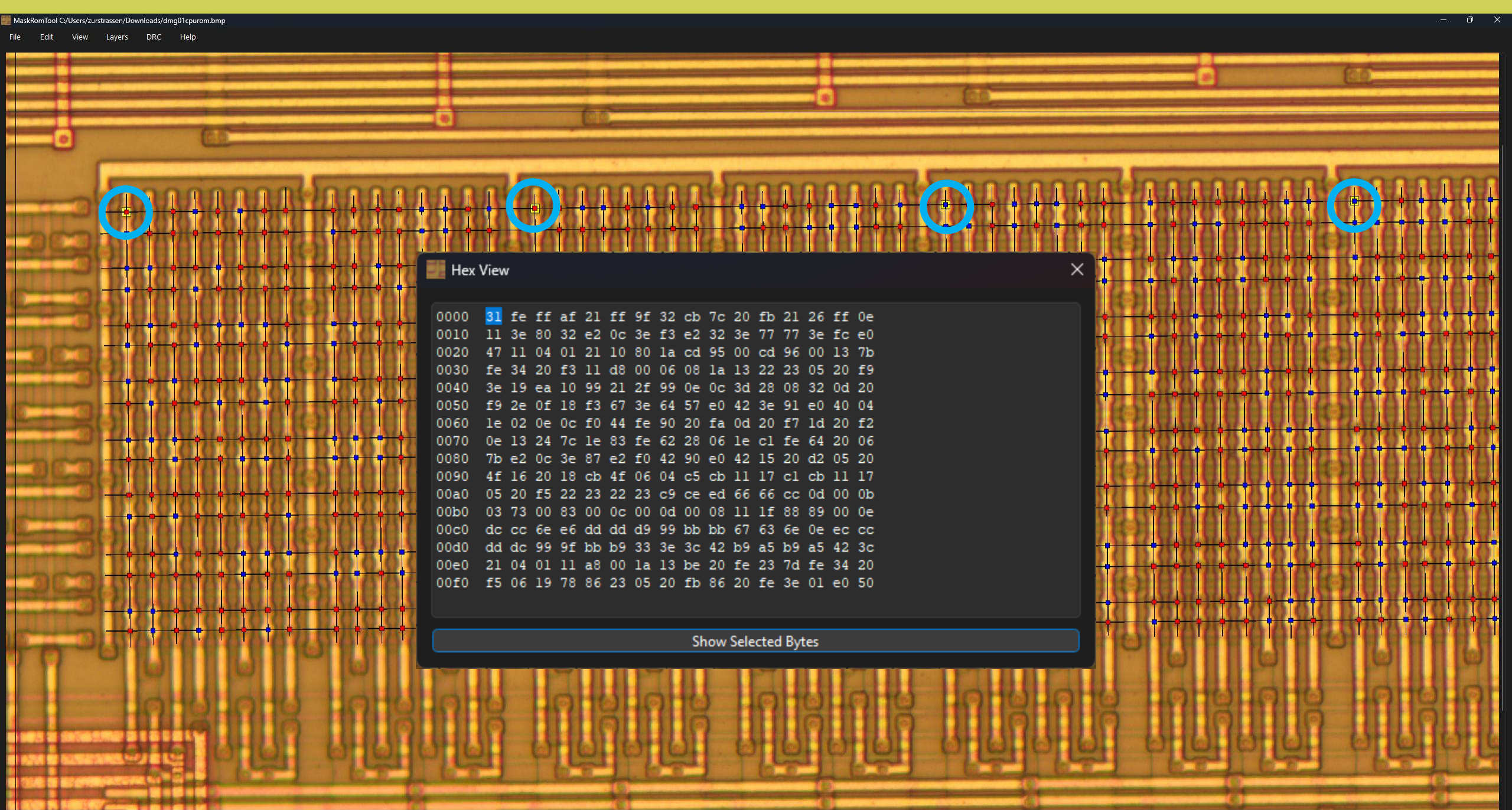
LD SP,\$fffe

Solve



Solutions

```
--decode-cols-downr -i -r 0 --flipx  
--decode-cols-downr -i -r 0 --flipx  
--decode-cols-downr -i -r 0 --flipx  
--decode-cols-downr -i -r 0 --flipx
```



unidasmlr35902 Disassembly

```
00: 31 fe ff ld sp,$FFFE
03: af xor a
04: 21 ff 9f ld hl,$9FFF
07: 32 ld (hl-),a
08: cb 7c bit 7,h
0a: 20 fb jr nz,$0007
0c: 21 26 ff ld hl,$FF26
0f: 0e 11 ld c,$11
11: 3e 80 ld a,$80
13: 32 ld (hl-),a
14: e2 ld ($FF00+c),a
15: 0c inc c
16: 3e f3 ld a,$F3
18: e2 ld ($FF00+c),a
19: 32 ld (hl-),a
1a: 3e 77 ld a,$77
1c: 77 ld (hl),a
1d: 3e fc ld a,$FC
1f: e0 47 ld ($FF47),a
21: 11 04 01 ld de,$0104
24: 21 10 80 ld hl,$8010
27: 1a ld a,(de)
28: cd 95 00 call $0095
2b: cd 96 00 call $0096
2e: 13 inc de
2f: 7b ld a,e
30: fe 34 cp $34
32: 20 f3 jr nz,$0027
34: 11 d8 00 ld de,$00D8
37: 06 08 ld b,$08
39: 1a ld a,(de)
3a: 13 inc de
3b: 22 ld (hl+),a
3c: 23 inc hl
3d: 05 dec b
3e: 20 f9 jr nz,$0039
40: 3e 19 ld a,$19
42: ea 10 99 ld ($9910),a
45: 21 2f 99 ld hl,$992F
```

unidasn/lr35902 Disassembly

```
00: 31 fe ff ld sp,$FFFE
03: af xor a
04: 21 ff 9f ld hl,$9FFF
07: 32 ld (hl-),a
08: cb 7c bit 7,h
0a: 20 fb jr nz,$0007
0c: 21 26 ff ld hl,$FF26
0f: 0e 11 ld c,$11
11: 3e 80 ld a,$80
13: 32 ld (hl-),a
14: e2 ld ($FF00+c),a
15: 0c inc c
16: 3e f3 ld a,$F3
18: e2 ld ($FF00+c),a
19: 32 ld (hl-),a
1a: 3e 77 ld a,$77
1b: 32 ld (hl-),a
1c: 3e f3 ld a,$F3
1f: e2 ld ($FF00+c),a
21: 04 ld de,$0104
24: 21 10 80 ld hl,$8010
27: 1a ld a,(de)
28: cd 95 00 call $0095
2b: cd 96 00 call $0096
2e: 13 inc de
2f: 7b ld a,e
30: fe 34 cp $34
32: 20 f3 jr nz,$0027
34: 11 d8 00 ld de,$00D8
37: 06 08 ld b,$08
39: 1a ld a,(de)
3a: 13 inc de
3b: 22 ld (hl+),a
3c: 23 inc hl
3d: 05 dec b
3e: 20 f9 jr nz,$0039
40: 3e 19 ld a,$19
42: ea 10 99 ld ($9910),a
45: 21 2f 99 ld hl,$992F
```

What is it doing?

BOOT ROM OVERVIEW



Setup stack pointer



Setup Video



Setup Audio



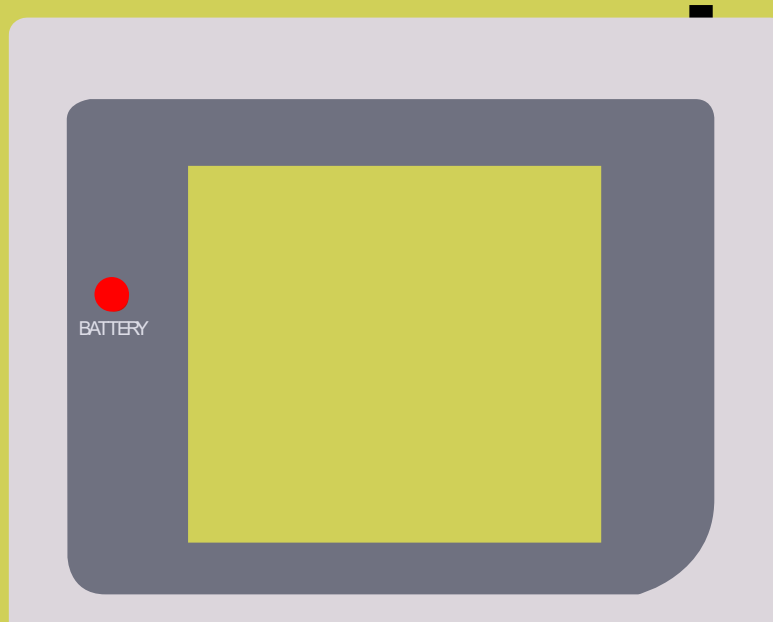
Load the Nintendo logo

LOGO CHECK

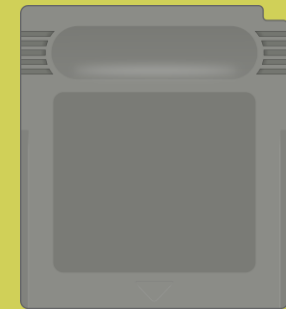
ⓔ Load the Nintendo logo



Boot ROM



Nintendo®



Cartridge

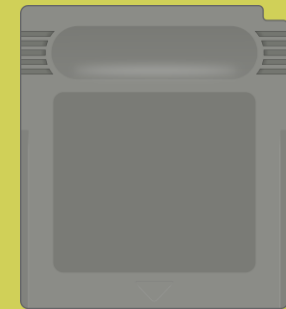
LOGO CHECK

ⓔ Load the Nintendo logo

Nintendo®



Boot ROM



Cartridge

LOGO CHECK

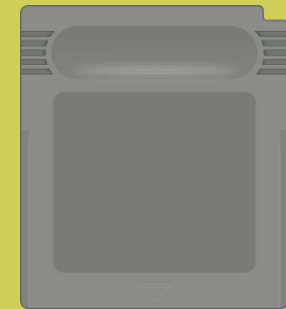
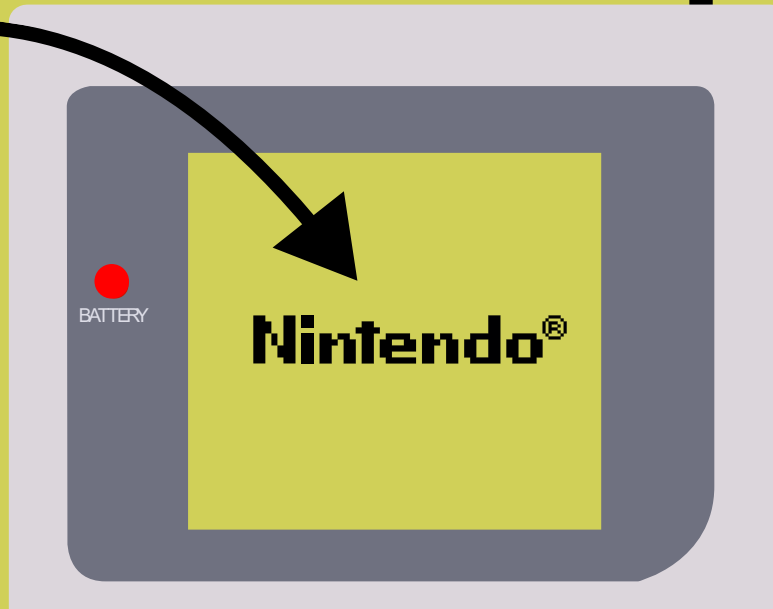
ⓔ Load the Nintendo logo



Nintendo®



Boot ROM



Cartridge

LOGO CHECK

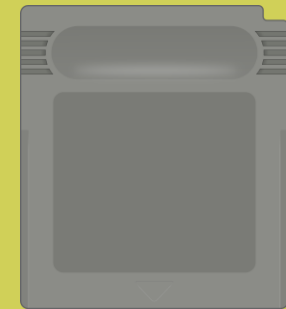
ⓔ Load the Nintendo logo



Nintendo®



Boot ROM



Cartridge

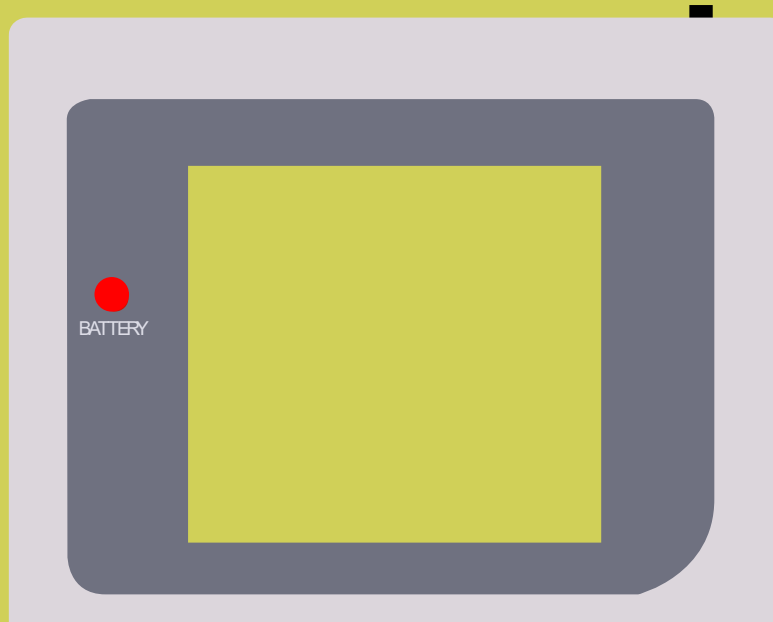
LOGO CHECK

Ⓡ Load the Nintendo logo

Nintendo®



Boot ROM



SystemC®



Cartridge

LOGO CHECK

 Load the Nintendo logo



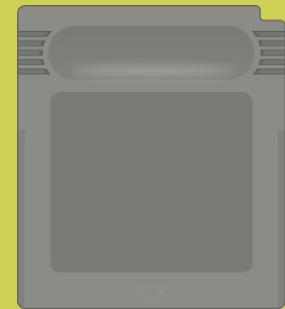
Nintendo®



Boot ROM

BATTERY

SystemC®



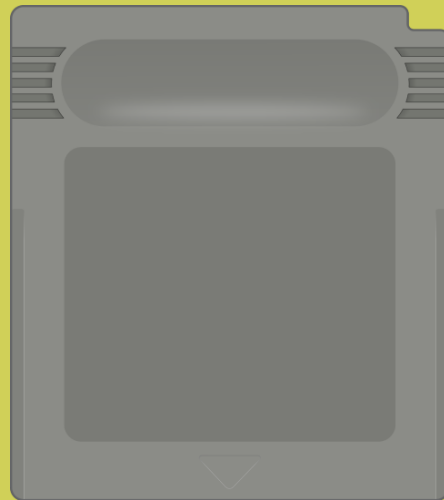
Cartridge

LOGO CHECK

WHY?

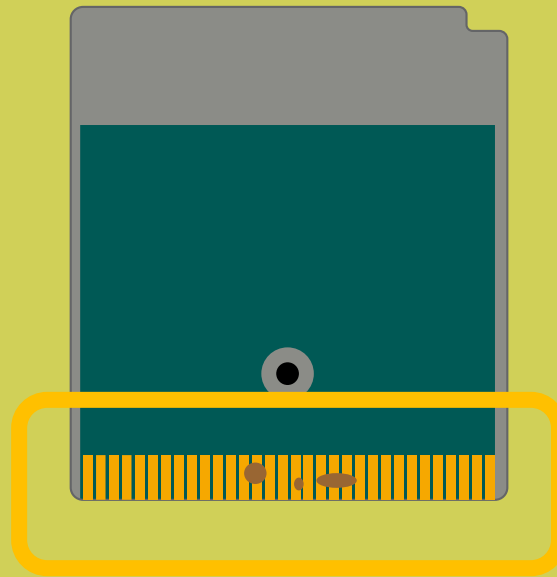
LOGO CHECK

1. Connection check



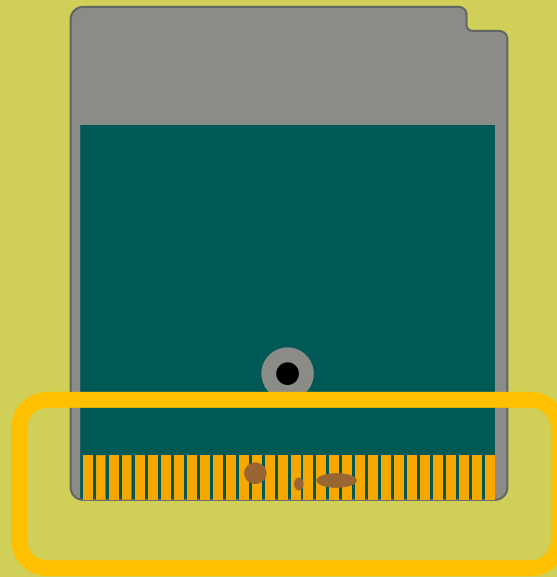
LOGO CHECK

1. Connection check



LOGO CHECK

1. Connection check

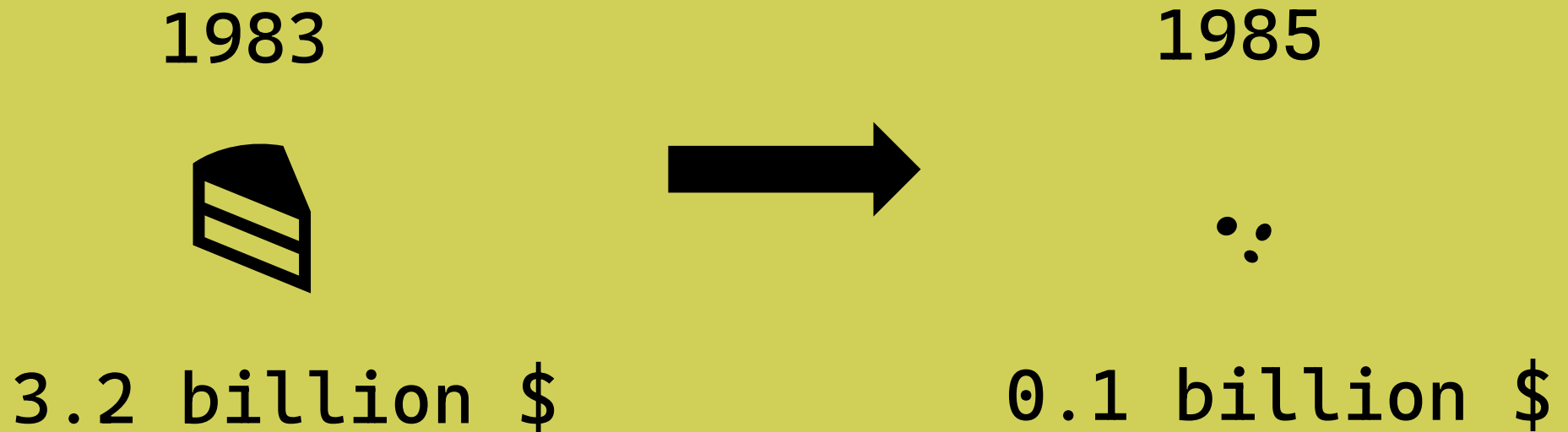


LOGO CHECK

1. Connection check
2. Prevention of unauthorized games

A BIT OF HISTORY

US Home Video Game Revenue*



*https://en.wikipedia.org/wiki/Video_game_crash_of_1983

A BIT OF HISTORY

*



"Atari collapsed because they gave too much freedom to third-party developers and the market was swamped with rubbish games"

Hiroshi Yamauchi, Nintendo President,
1986

*https://nintendo.fandom.com/wiki/Hiroshi_Yamauchi

NINTENDO'S IDEA



SYSTEMC THINGS

SYSTEMC PERFORMANCE

Performance goal

CPU -> ~1 million instructions per second

PPU -> ~1 million pixels per second



SYSTEMC PERFORMANCE

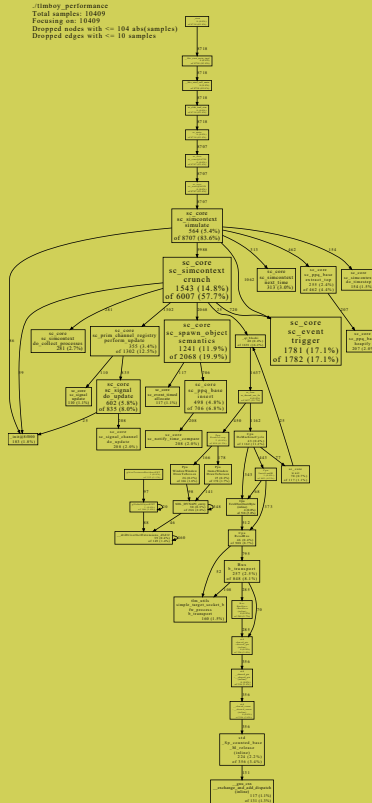
Performance goal

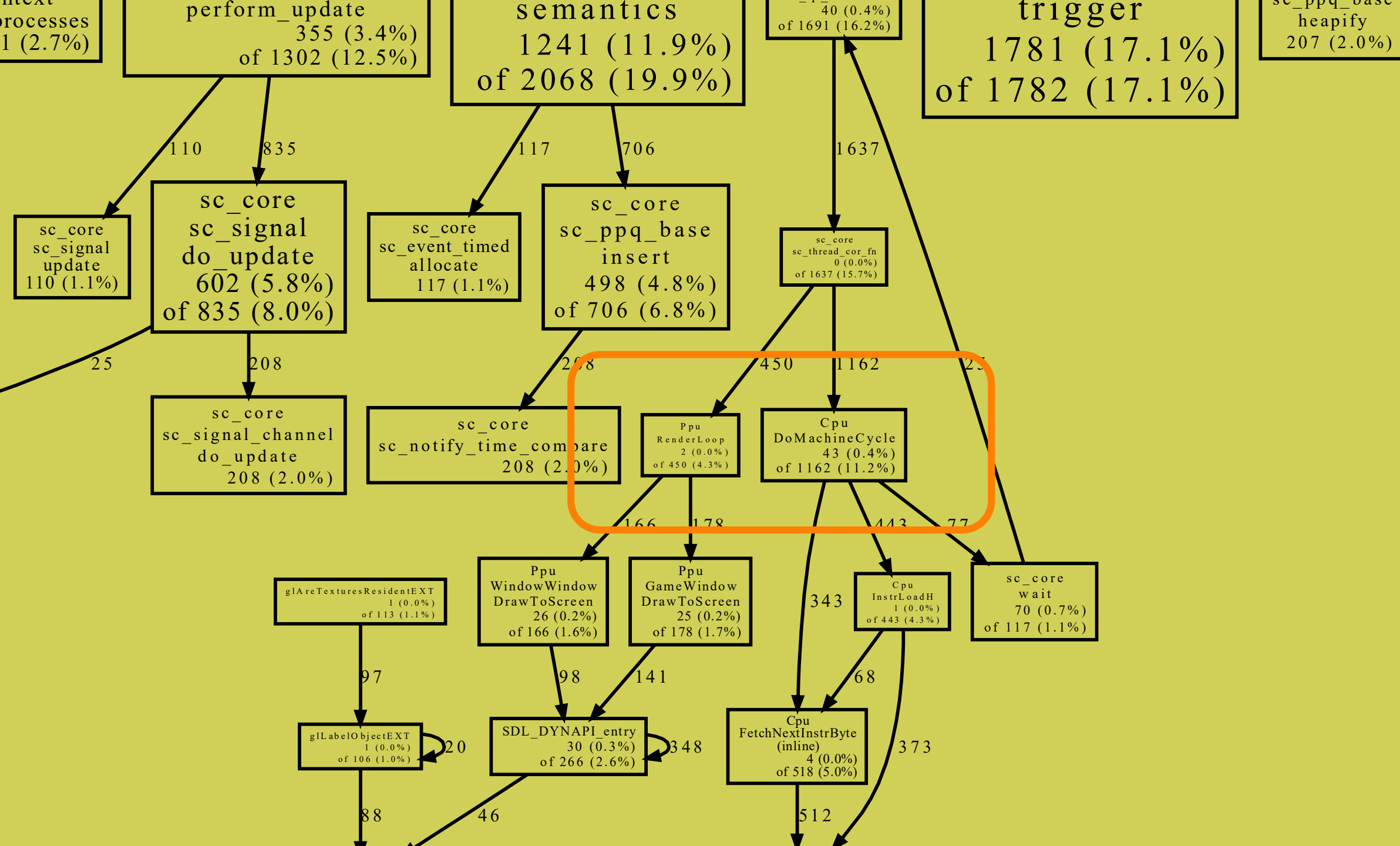
CPU -> ~1 million instructions per second

PPU -> ~1 million pixels per second



SYSTEMC PERFORMANCE





SYSTEMC PERFORMANCE

SystemC is the bottleneck!

SYSTEMC EVALUATION



A beginner's view!

SYSTEMC PROS

- + SystemC coroutine → clean code
- + Timekeeping
- + Ports, Signals

SYSTEMC CONS

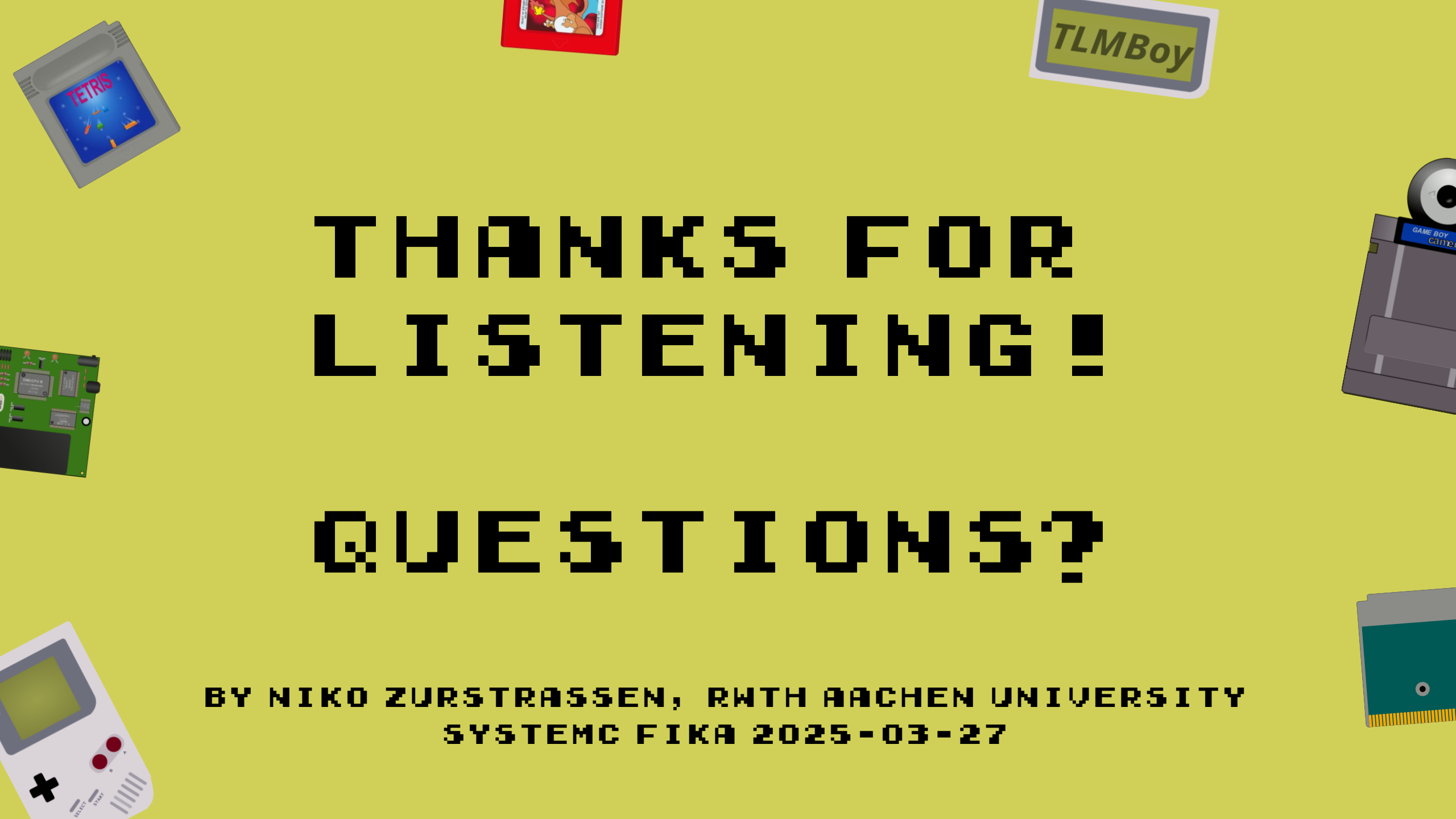
- A lot of leaky magic in the background
- Tons of macros

“You're not really going to define a macro, are you?”

Google C++ Style Guide

SYSTEMC CONS

- A lot of leaky magic in the background
- Tons of macros
- Fuzzy errors
- No checkpointing



THANKS FOR LISTENING! QUESTIONS?

BY NIKO ZURSTRASSEN, RWTH AACHEN UNIVERSITY
SYSTEMC FIKR 2025-03-27