Many More Tamagotchis Were Harmed in the Making of this Presentation

Natalie Silvanovich

About Me

- Security Researcher at BlackBerry
  - (But I don't represent them)
- Studied electrical engineering, but mostly into software hacking
- First-time hardware hacker/ reverse engineer
- Tamagotchi enthusiast

What are Tamagotchis?

- The same virtual pet toys you remember from the 90's
- Functionality has evolved substantially
  - Now they can go to school, have jobs, make friends, get married and have kids!
- Newer versions have an IR interface

so that they can communicate with other Tamagotchis

TamaTown Tama-Go

- The “Christmas” Tamagotchi from 2010
- Same functionality for smaller hands
- Supports detachable ‘figures’ with extra games and stores

Goals

- Dump Tamagotchi code
- Answer the ‘deeper questions’ of Tamagotchi life
- Make my gotchis rich and happy
- Make a TV-B-Gotchi
- Have fun!

Posted by: bitbot on June 15, 2005 at 1:33 AM

xoscope for Linux
http://xoscope.sourceforge.net/

Reply to this comment

Posted by: Epicanis on June 15, 2005 at 1:06 PM

Do I remember correctly that these critters could "have children" when paired up (and you would, I guess, have multiple virtual critters running around on your Tamagotchi device?)

The idea of decoding the IR and eventually running around having my notebook/pda "impregnating" innocent tamagotchi devices amuses me greatly...

Reply to this comment

Posted by: globo on June 30, 2005 at 4:51 AM

what device did you use send a messege back to
globo@olhc.woll.catholic.edu.au

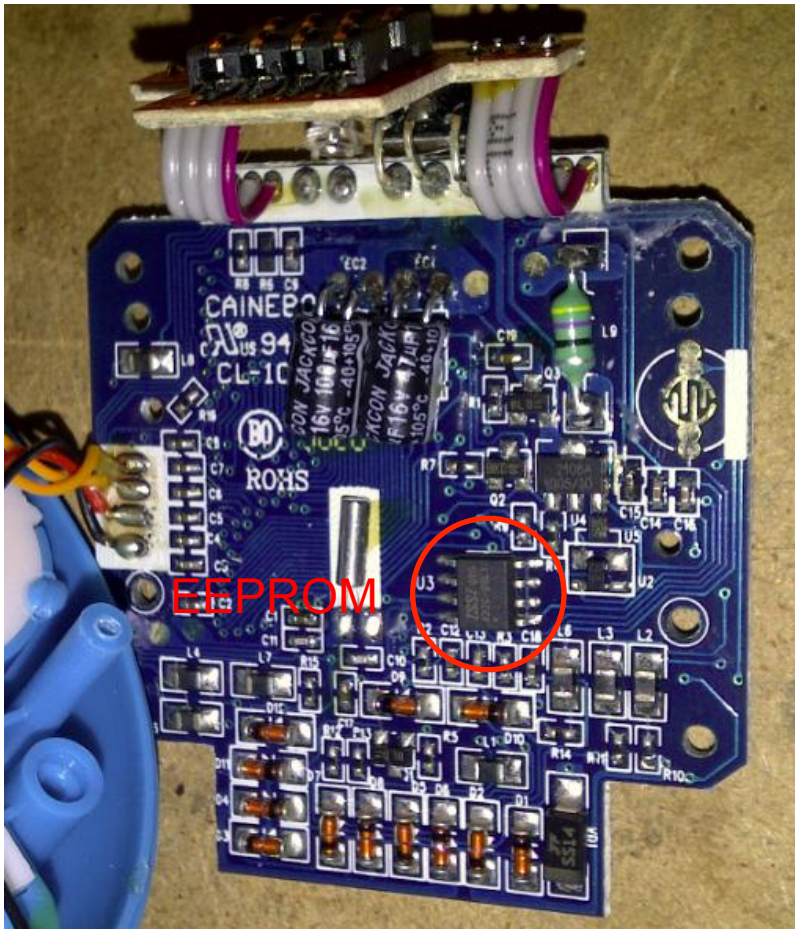

Previous Work

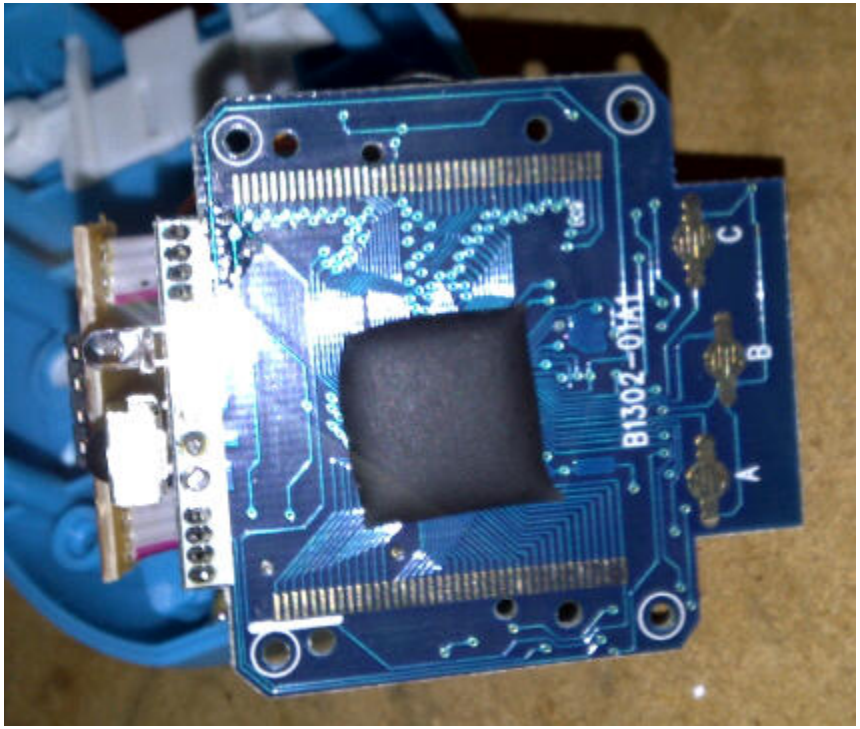
Teardown

Hardware Teardown

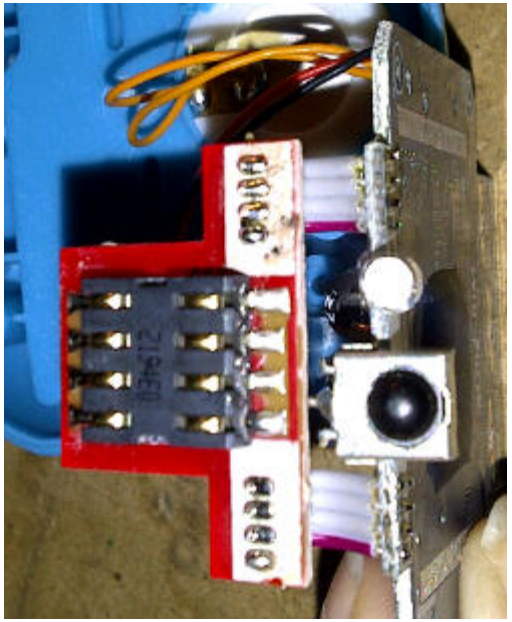
- Took apart a Tama-Go and Tamagotchi to determine if code dumping was a possibility
- Looked for helpful interfaces
- Also took apart a figure

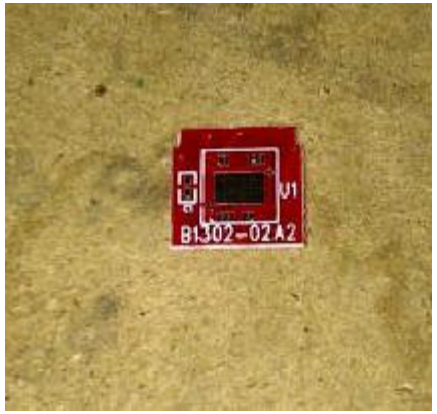
Tama-Go Board

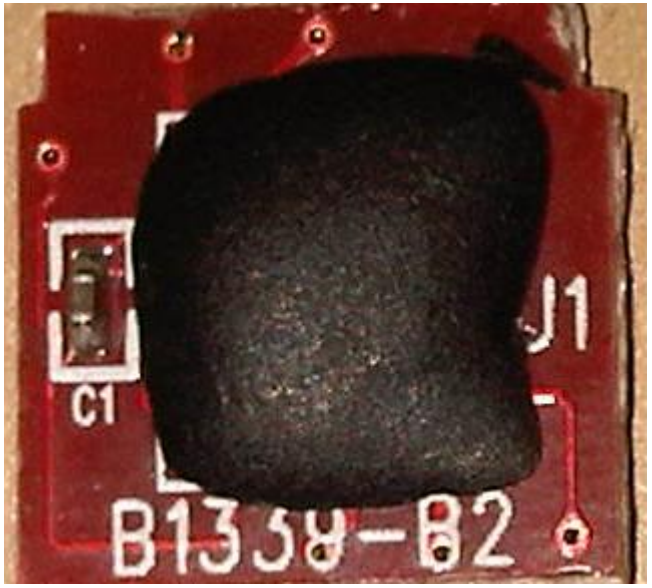




Tama-Go Figure



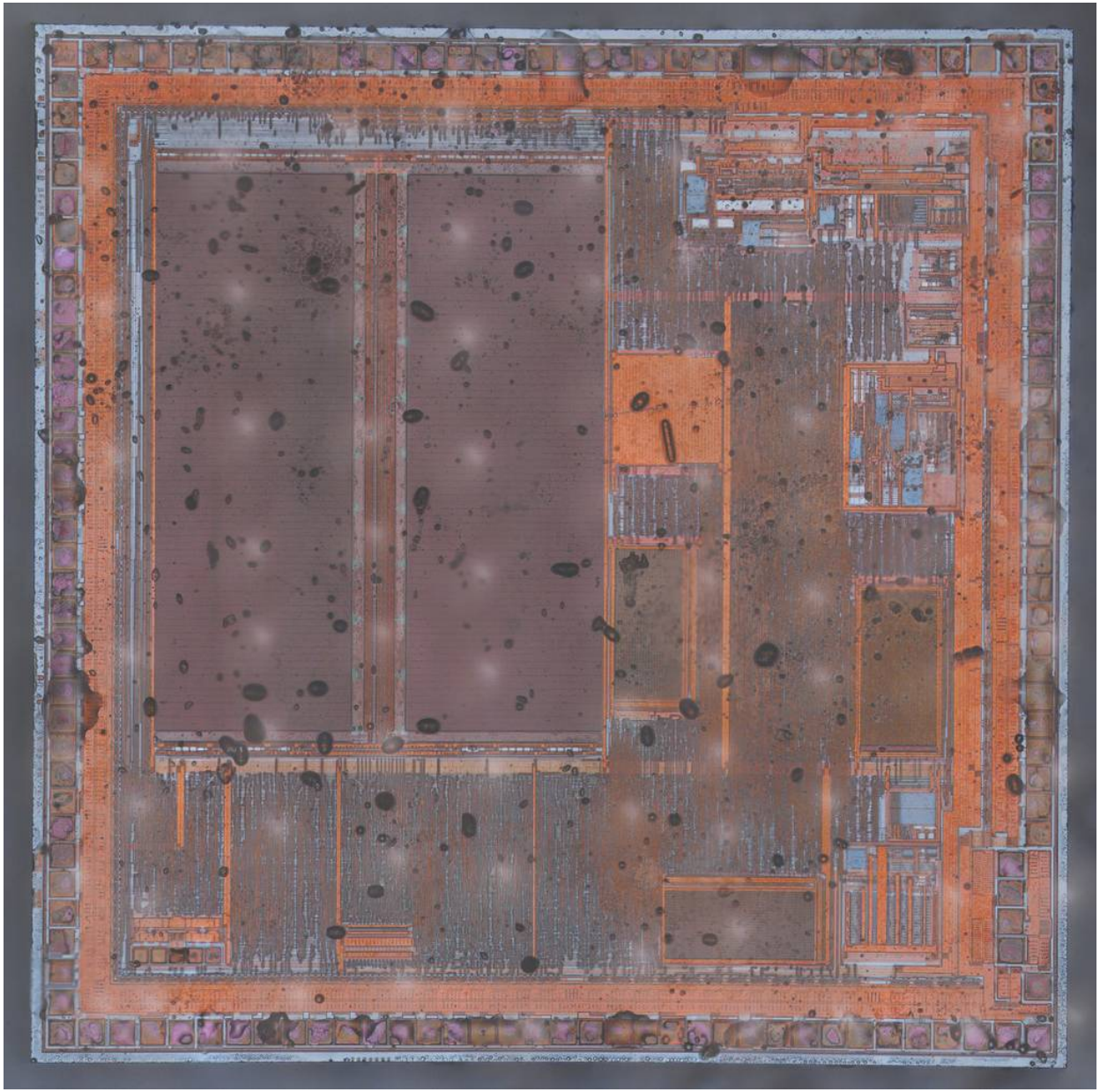




Microcontroller Identification

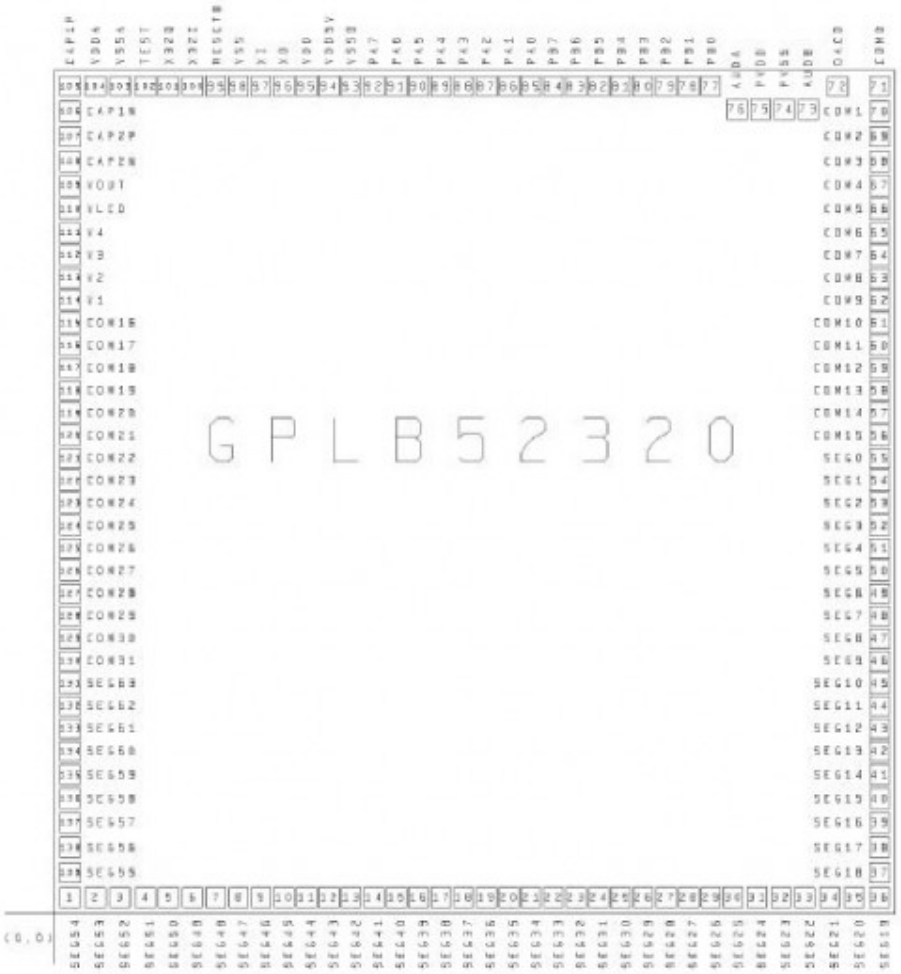
Identifying the Microcontroller

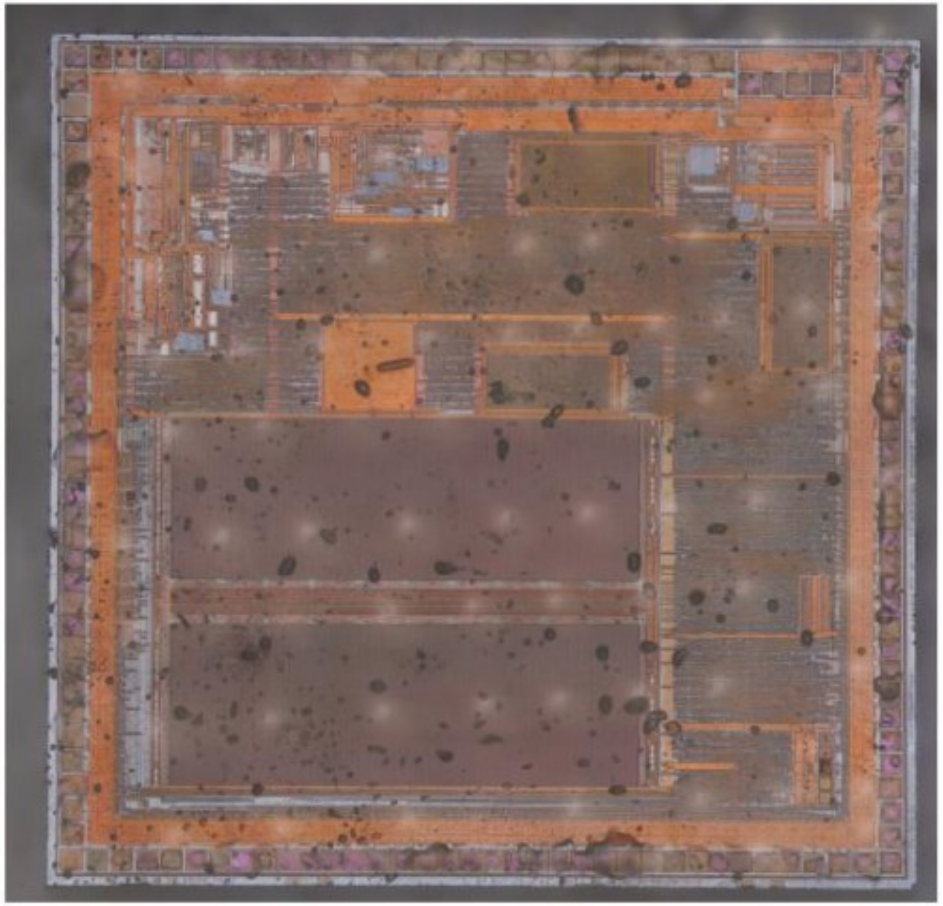
- Considering the lack of external hardware, MCU was likely under the 'blob'
- Tried several methods to remove, including acetone, heat, a razor blade and a chopstick
- Travis Goodspeed kindly offered to decap the chip with acid




S08C3CR35732

- Eventually, success!





GPLB5X Series LCD Controller

- 8 bit 6502 microprocessor
- 1536 bytes RAM
- 320 or 640 kbyte mask ROM (depending on model), baked to perfection for each customer
- 512 bytes LCD RAM
- 4 color grayscale LCD controller
- SPI
- Audio DAC

Dumping Mask ROM

- Not sure how to dump mask ROM, but had a few ideas
  - Restore a bad state from EEPROM
  - Look for test functionality
  - Exploit a vulnerability in figure or IR processing
  - Read ROM with a microscope
  - Pin manipulation


Test
Program

Test Program?

- GeneralPlus mask ROMs contain a GP test program that can probably dump code
- Contacted GeneralPlus for a devkit
  - Requires an NDA
- Looked around online
  - No one seems to have a devkit or know the test program

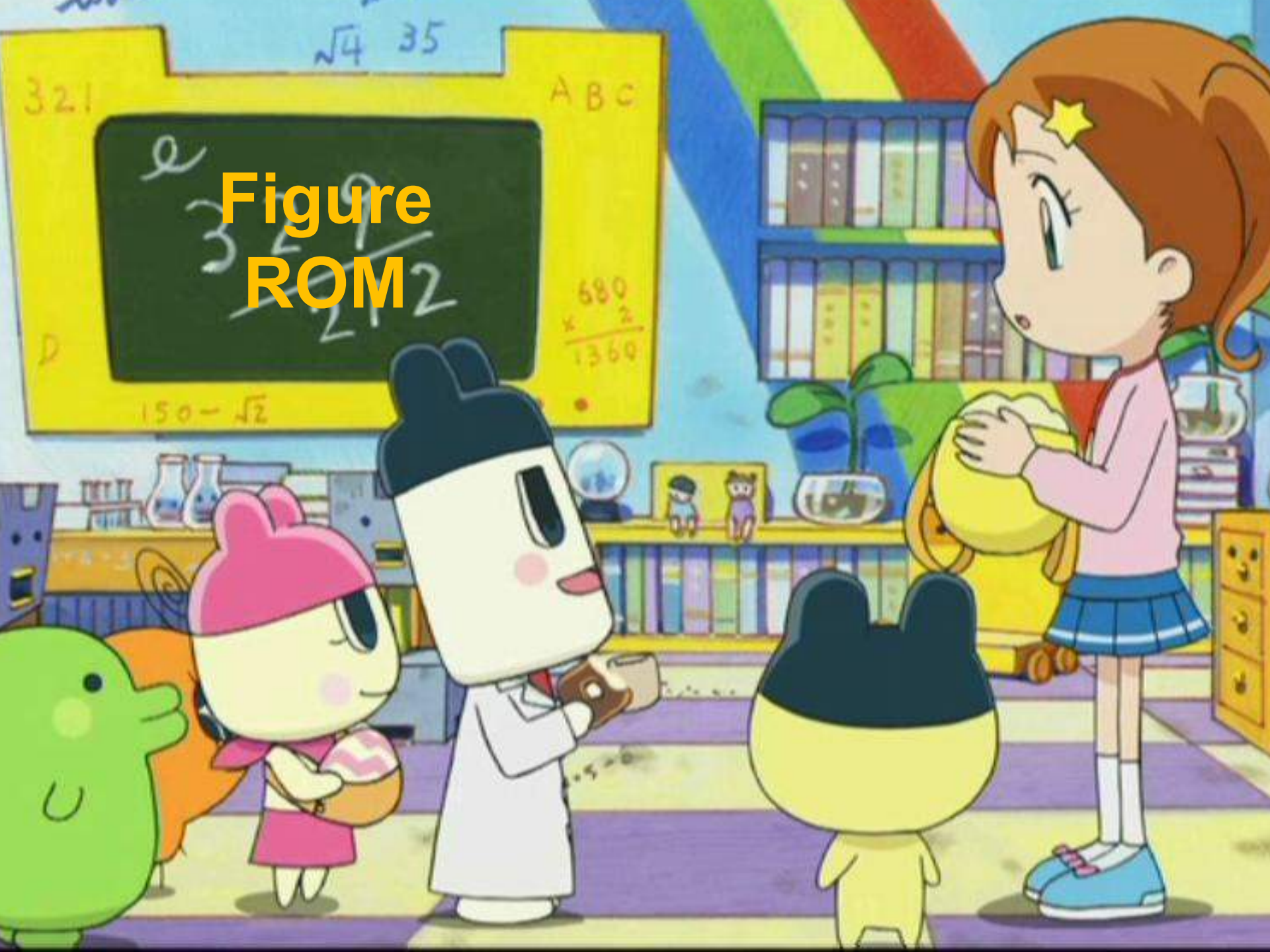


Figure
ROM2

Figure ROM

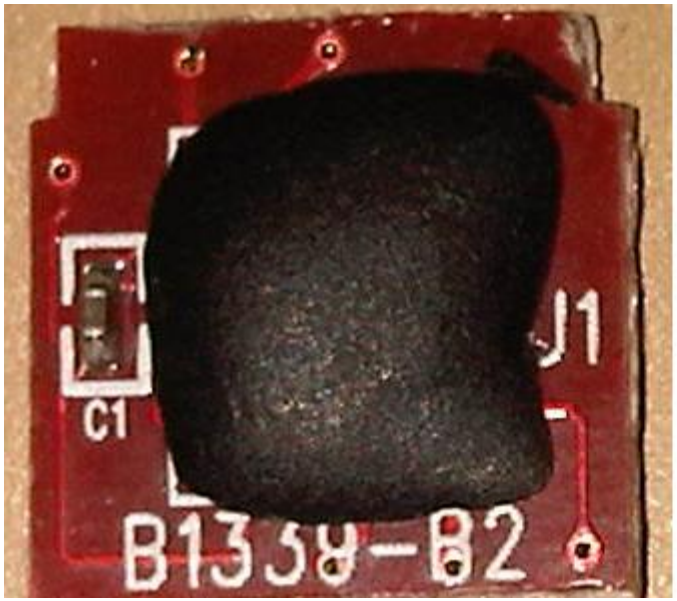
- Decoding the figure ROM could be useful in a few ways
  - Making your own Tamagotchi games
  - Executing code on the Tamagotchi
  - Dumping mask ROM
  - Understanding Tamagotchi behaviour

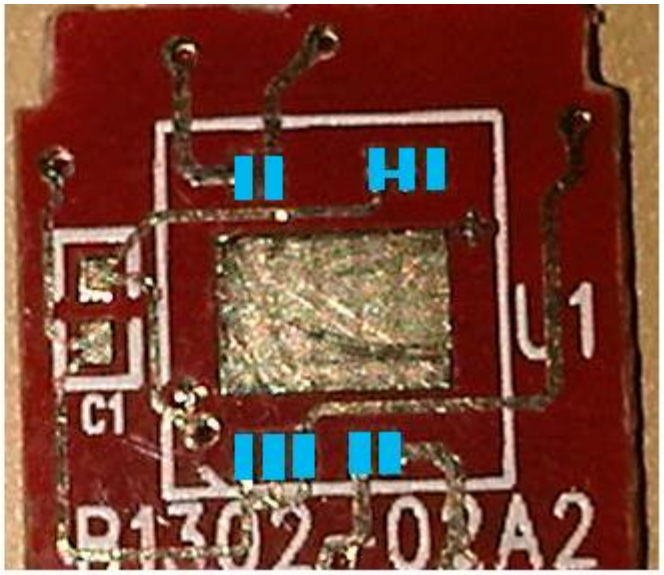
Figure Types

- There are two types of Tamagotchi figures, ‘regular’ and ‘lite’
- Regular figures contain PCBs with blobs
- Lite figures contain unpopulated PCB
  - Act as jumpers
- Tried jumper-ing regular figures
  - Saw functionality of different figures!
- Extremely likely figures contain mask ROM

Figure ROM Pads

- The unpopulated PCBs in lite figures appear to be the same boards used in regular figures

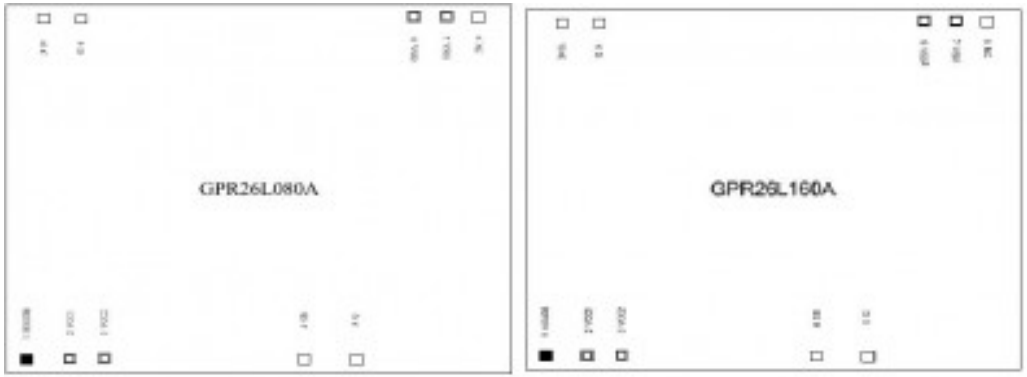




- Makes the mask ROM pad layout visible

Figure ROM Chip

- GeneralPlus makes an SPI ROM with a similar layout



- Assumed figures use this ROM

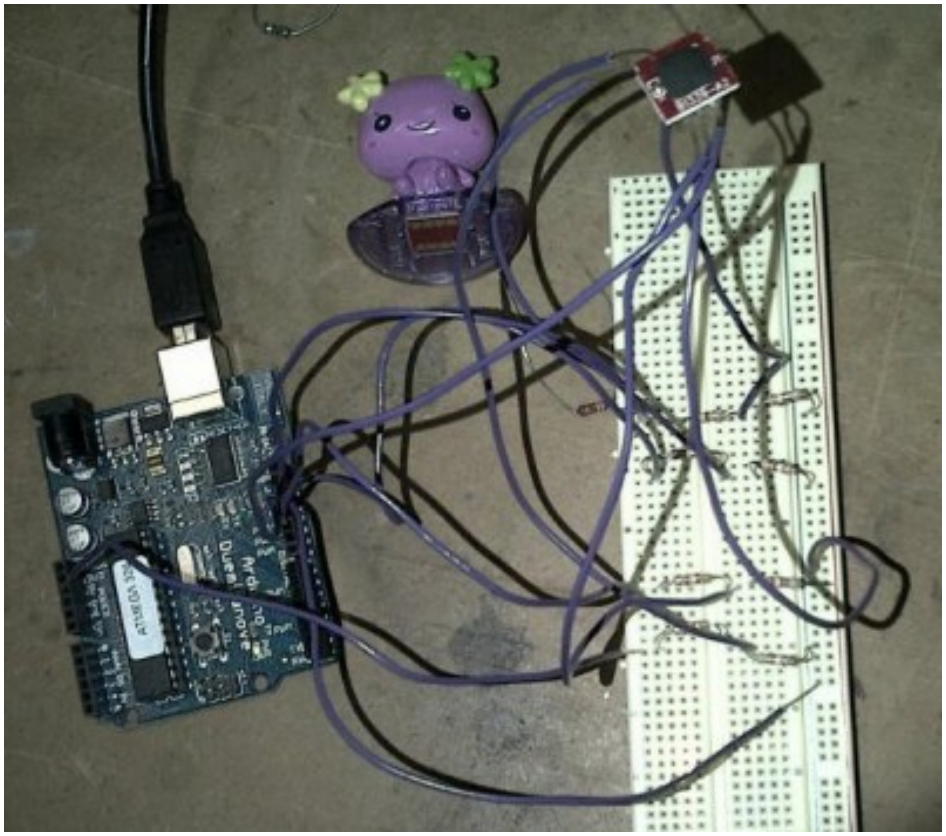
Figure ROM Pins

- Based on the GeneralPlus ROM datasheet, was able to identify the figure pins

1, 4 and 8: Ground/Jumper
2: Serial clock (C)
3: Serial data input (D)
5: Power
6: Chip Select (SB)
7: Serial Data Output (Q)

ROM Dump

- Dumped the ROM using an Arduino as SPI master



Decoding ROM

- The Tamagotchi has a four-tone display, so looked for strings of 0x00, 0x55, 0xAA and 0xFF, representing images
- Noticed that these strings were preceded by values which were reasonable for length and width

Decoding Images

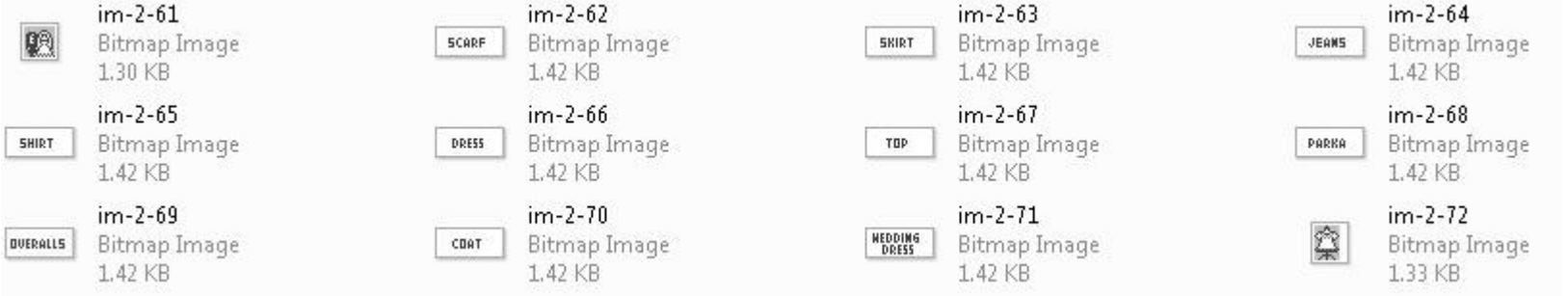
- Tried decoding these images



- Eventually, it worked!

Images

- The figure contained a lot of images
- Text displays appear to be images



- Animations are series of images

The Rest of the ROM

- The ROM contains a lot of non-image data
- None of this data is GeneralPlus code
  - Wrote a dissasembler
- Likely logic information in some sort of serialized format

Simulating the ROM

- Could not obtain compatible flash
- Attempted to simulate the ROM using an Arduino, but chip is too slow
- Switched to a Chipkit Uno, this was also too slow
- Eventually used a STM32F4 Discovery board

Simulating the ROM

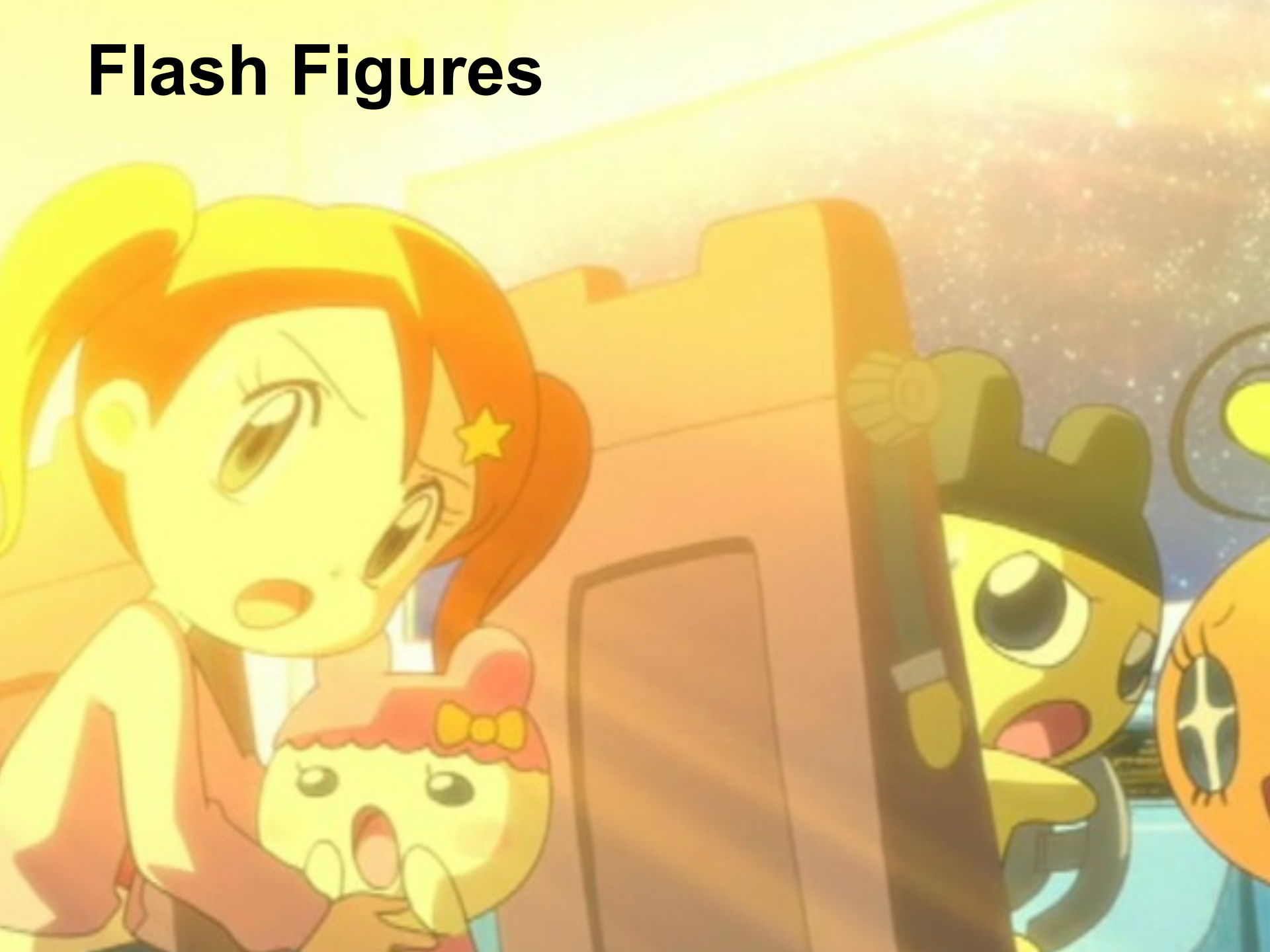
- Knew the image format, so could alter images

Game Logic

- The Tama-Go reads less than 50 bytes of non-image data during all figure functionality
- Game logic is represented by a one byte code
  - This logic is executed with images from figure
- Changing this code can cause a jump to non-game screens
  - Stats, food, death, etc. Every screen was available
- Many codes caused freezing

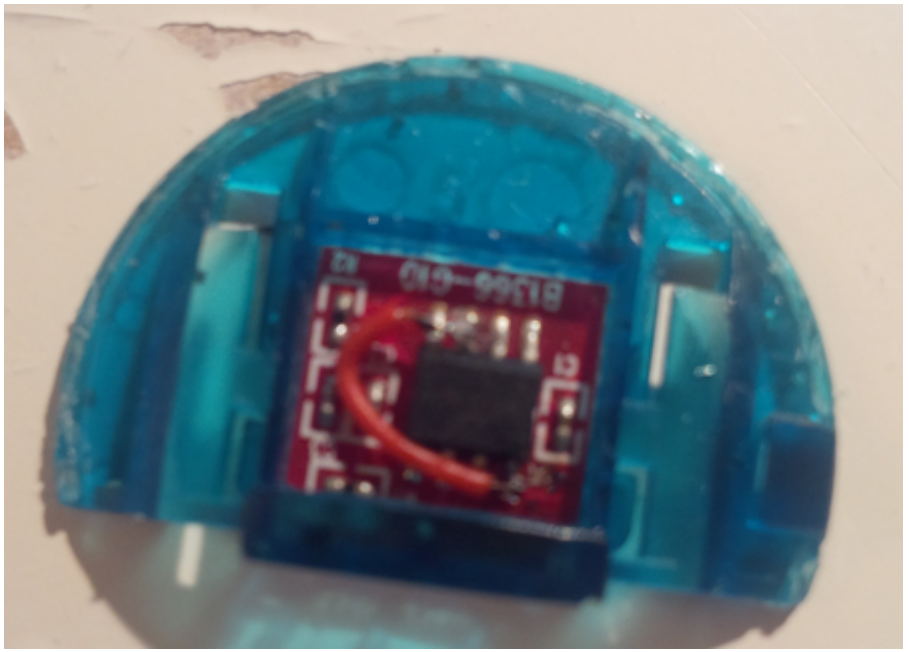
Evolve Demo

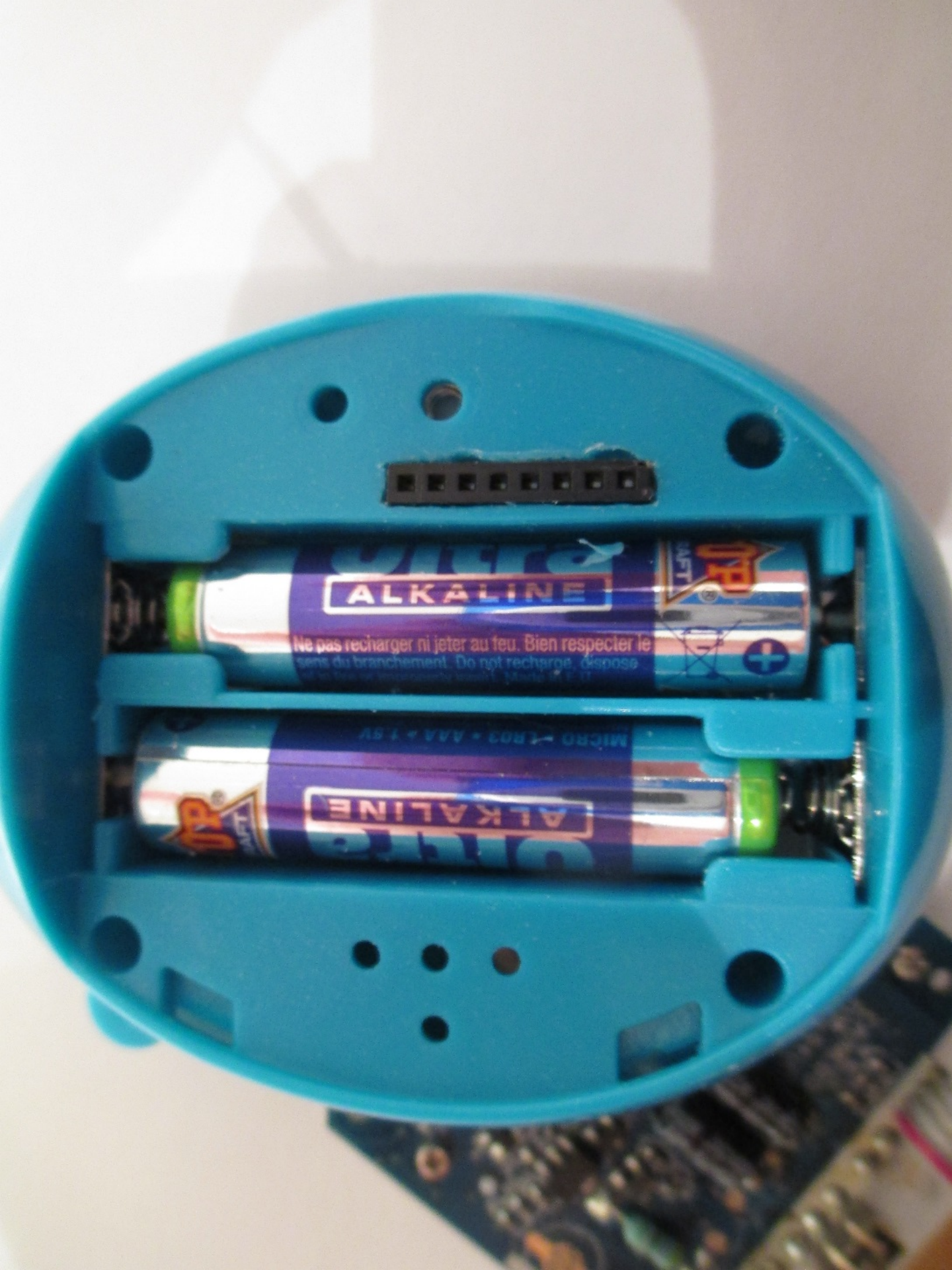
Flash Figures

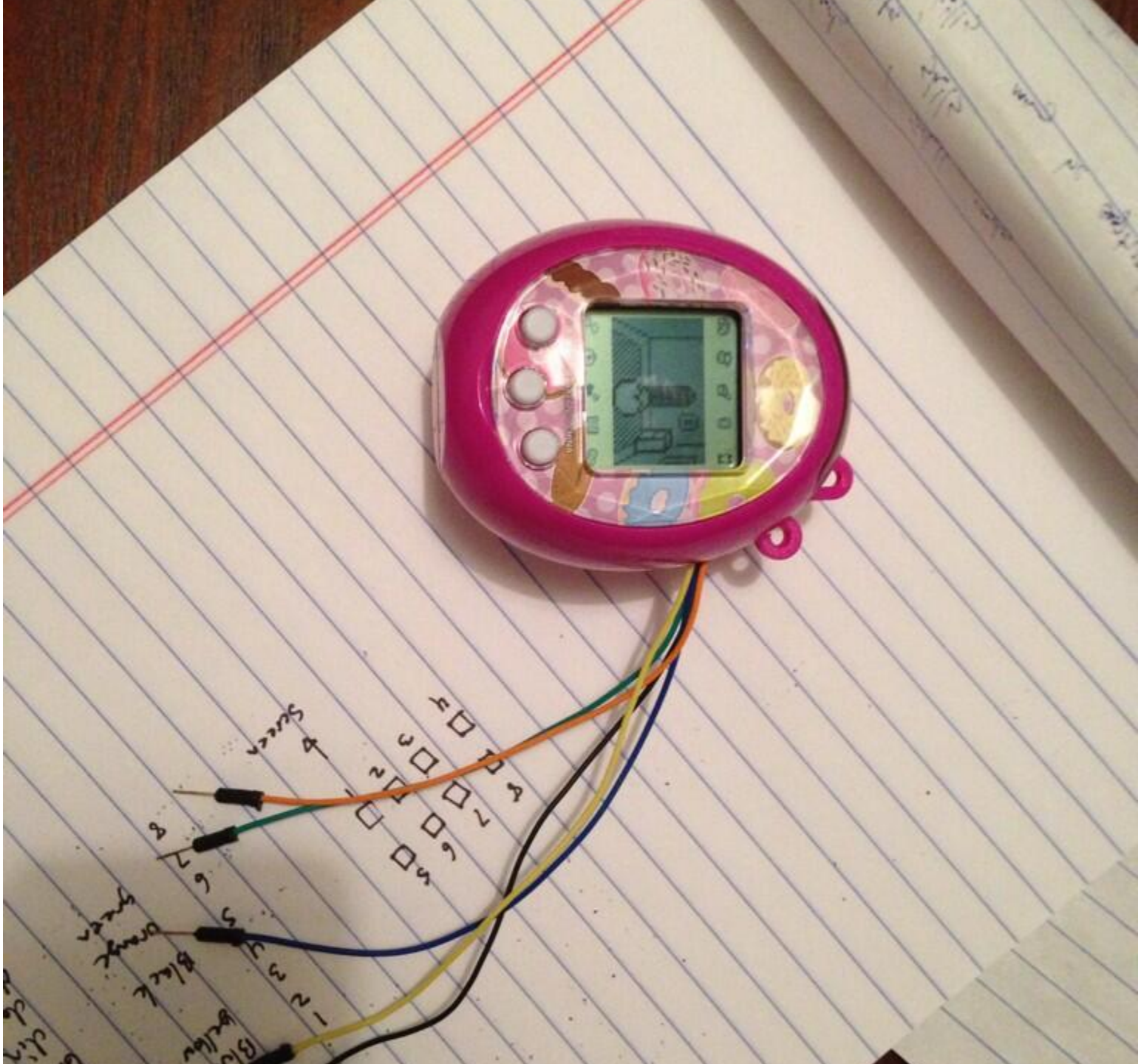


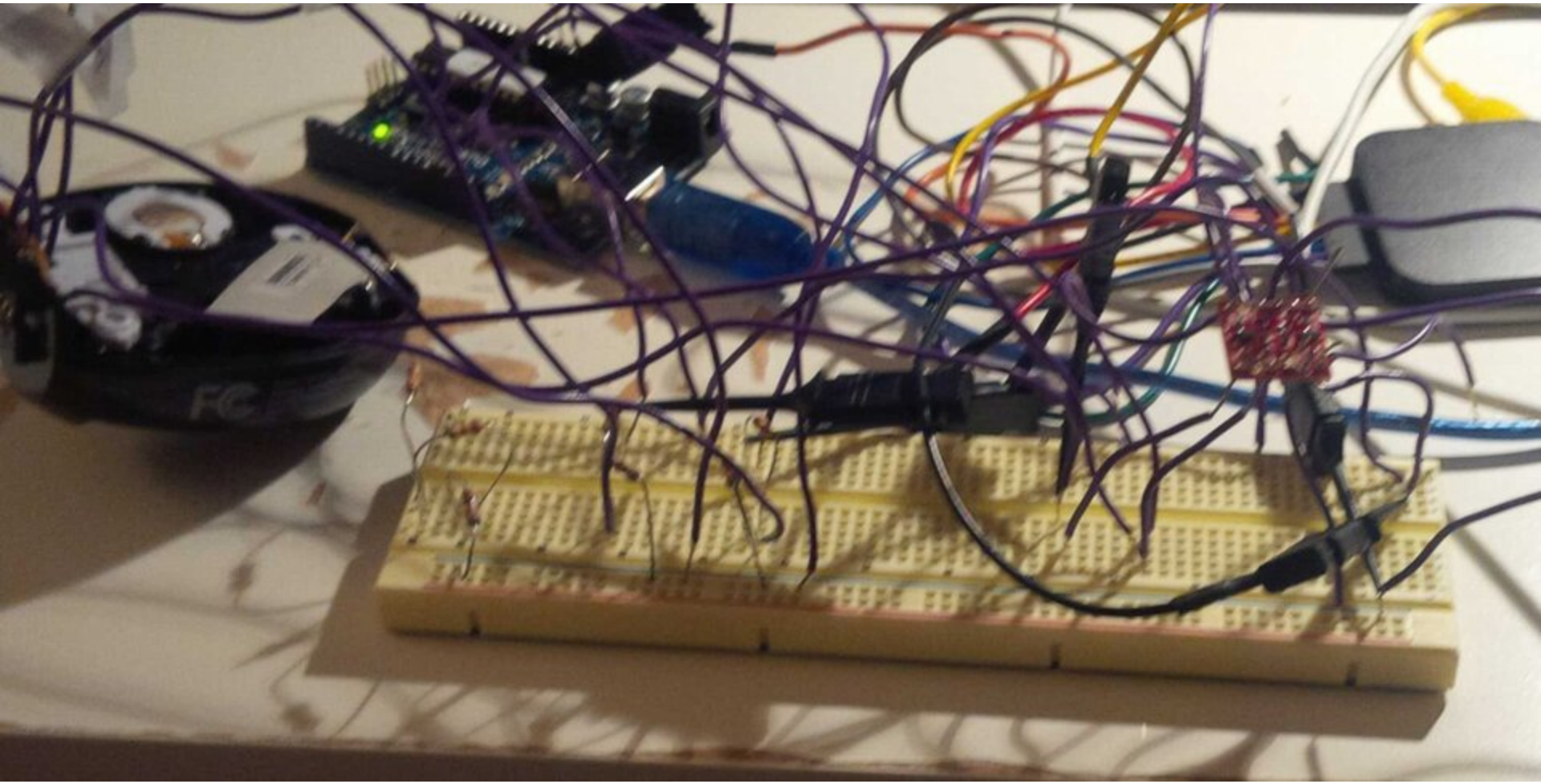
Flash Figures

- MrBlinky ordered a set of figures to experiment with
  - They contained flash!
  - Built a figure programmer
  - The ability to re-flash figures made testing much easier



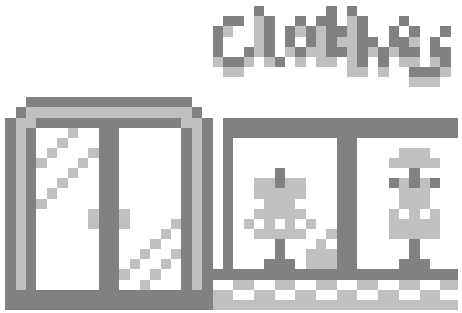

ALKALINE
Ne pas recharger ni jeter au feu. Bien respecter le
ALKALINE

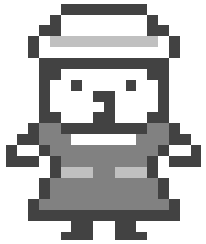




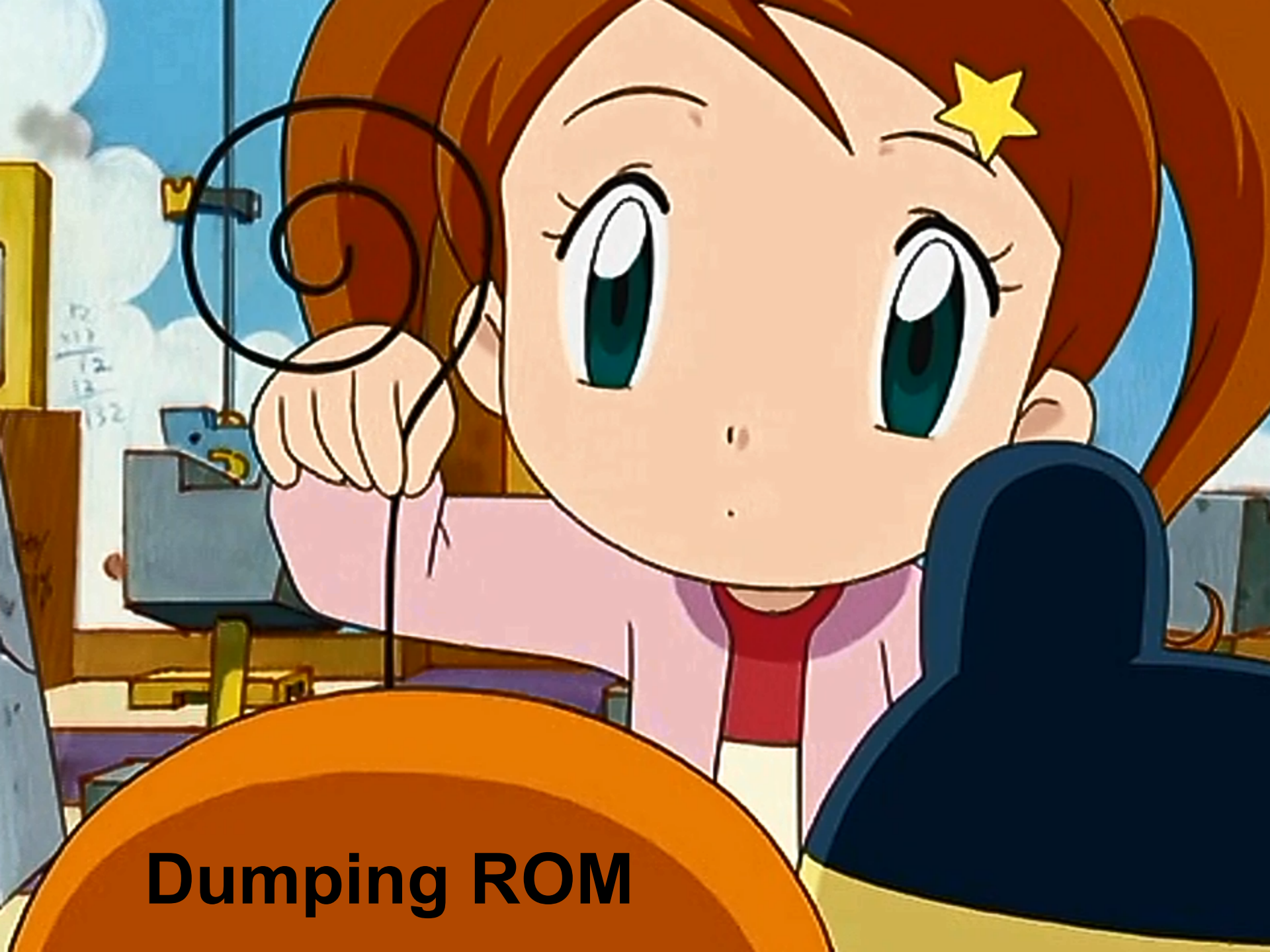
Items

- Items are implemented using a byte code format
  - Instructions include showing images, playing sounds and changing Tamagotchi stats
  - Some unusual behaviour for invalid instructions
  - Posted ‘dev tools’ on github





Demo


Dumping ROM

How to Dump the ROM of a Tamagotchi

(According to the Internet)

100+ 1. Use rompar → Fun with Masked ROMs - Atmel MARC4 http://adamsblog.aperturelabs.com/2013/01/fun-with-masked-roms.html

~20 2. "Use ROP"

~20 3. Circumvent Windows Stack Cookies → Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows Server 2003 http://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-litchfield.pdf

~15 4. Probe the chip

~10 5. "Glitch it"

< 5 6. Differential power analysis

How to Dump the ROM of a Tamagotchi

(According to the Internet)

1. Use rompar
2. "Use ROP"
3. Circumvent Windows Stack Cookies
4. Probe the chip
5. "Glitch it"
6. Differential power analysis

Hardware Methods

Software Methods


Hardware
Methods

Hardware Possibilities

1. Use rompar (optical decoding)
2. Probe the chip
3. “Glitch it”
4. Differential power analysis

Hardware Possibilities

- My problems are:
  - Don't have the equipment to get a high resolution image of ROM
  - Don't have a test device
- My problems are not:
  - Hardware security protections

Hardware Suggestions

1. Use rompar (optical decoding)
2. Probe the chip
3. “Glitch it”
4. Differential power analysis

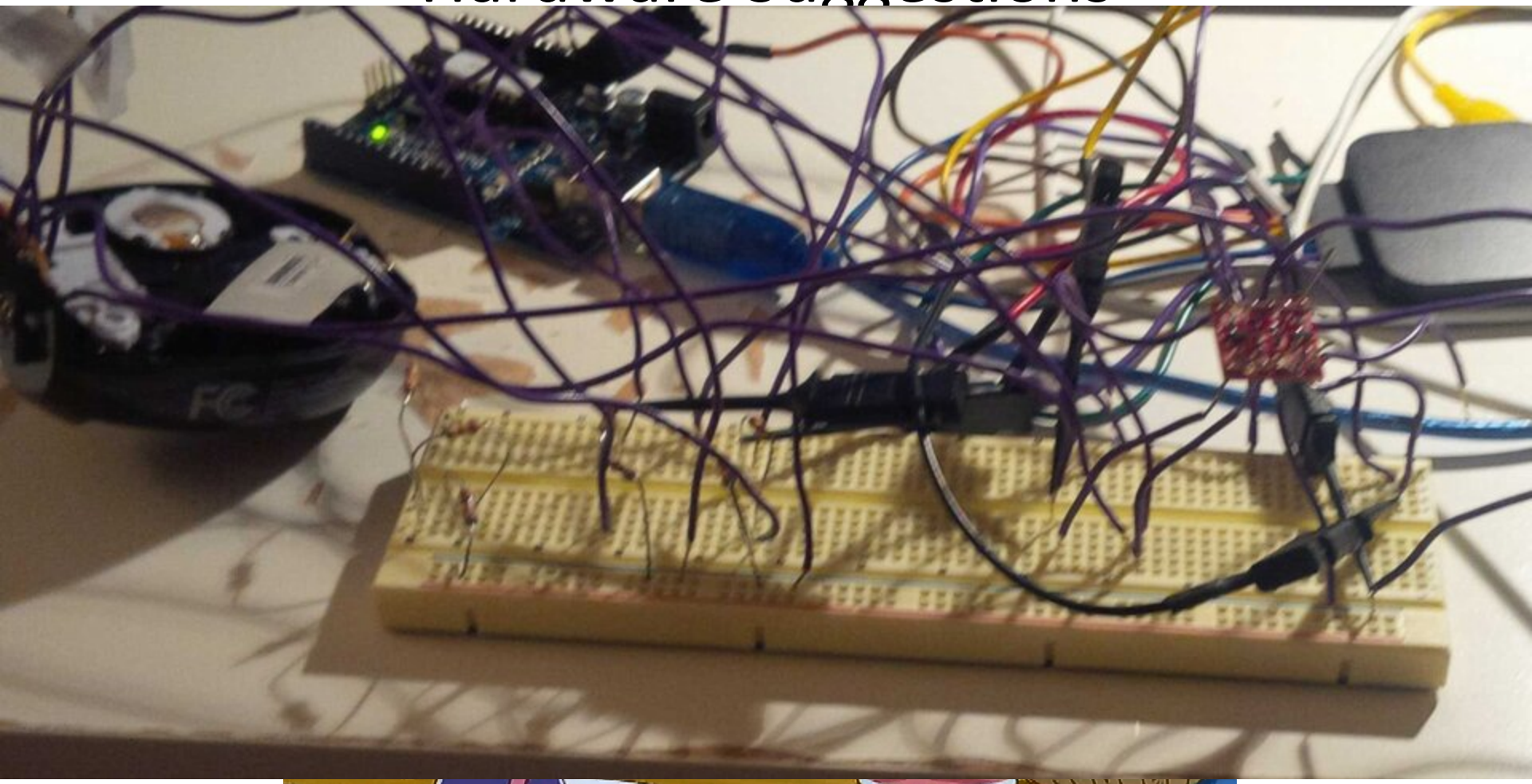
Hardware Suggestions

1. Use rompar (optical decoding)
2. Probe the chip
3. ~~"Glitch it"~~
4. Differential power analysis

Hardware Suggestions

1. Use rompar (optical decoding)
2. Probe the chip
3. ~~"Glitch it"~~
4. ~~Differential power analysis~~

Hardware Suggestions

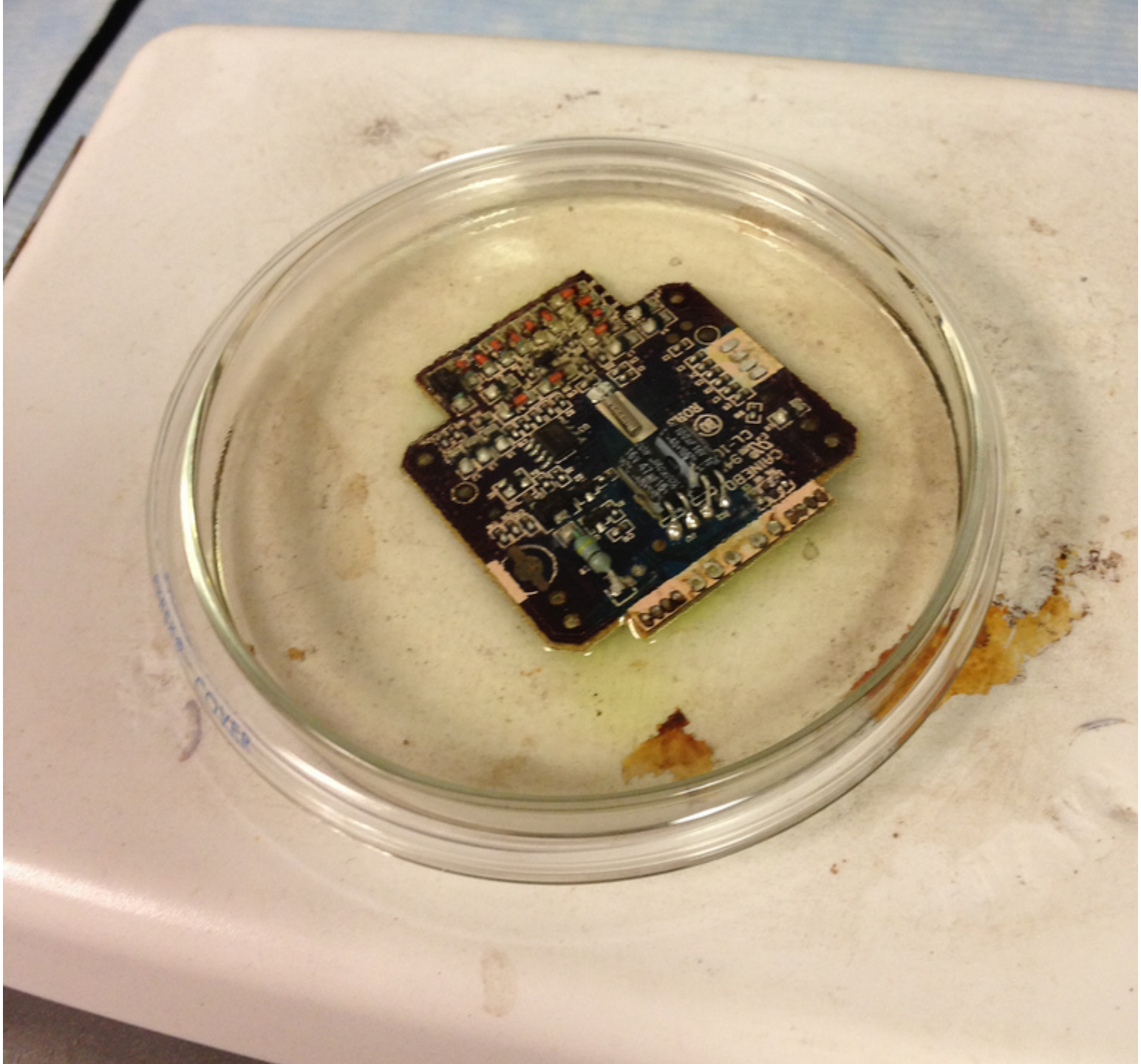


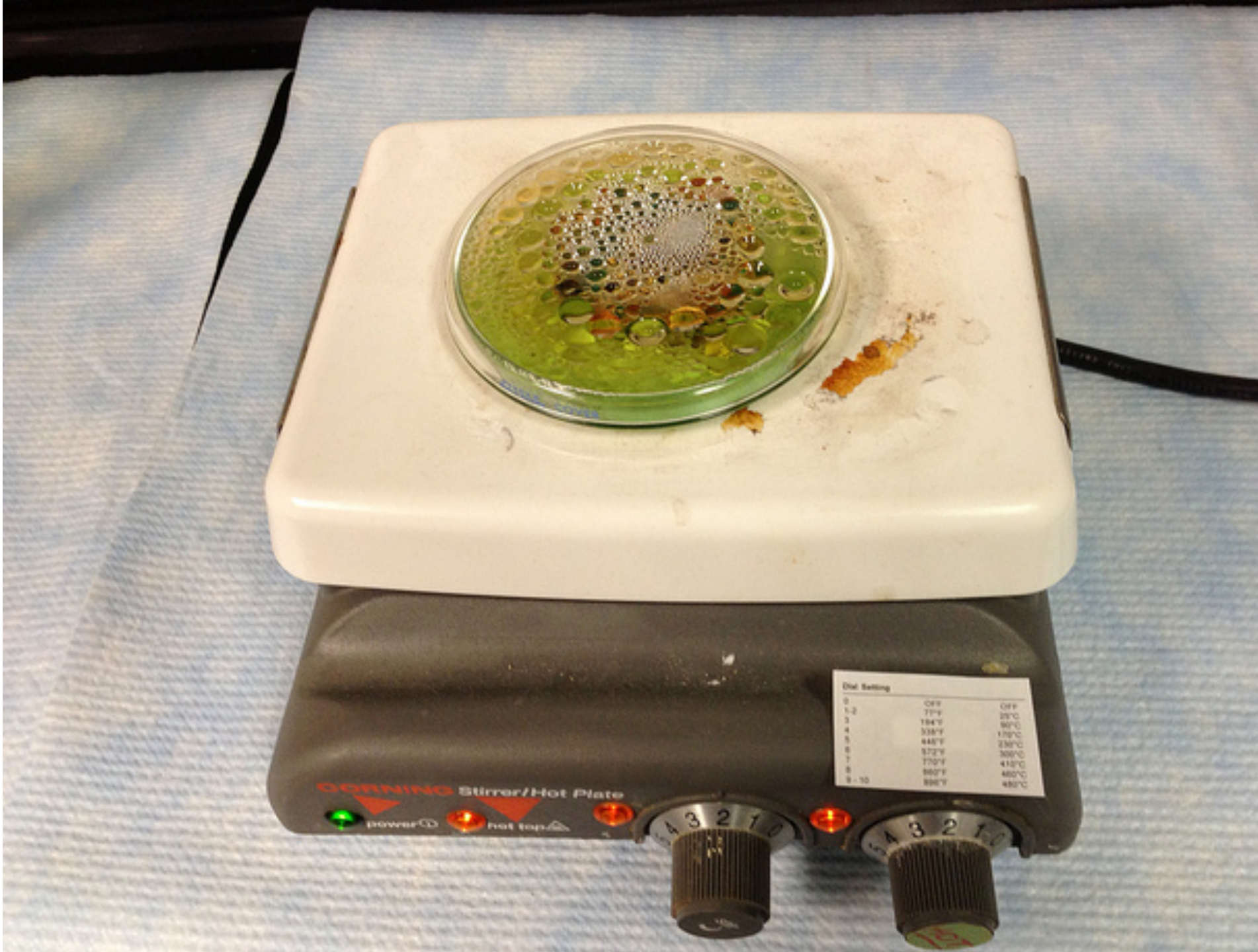
Hardware Suggestions

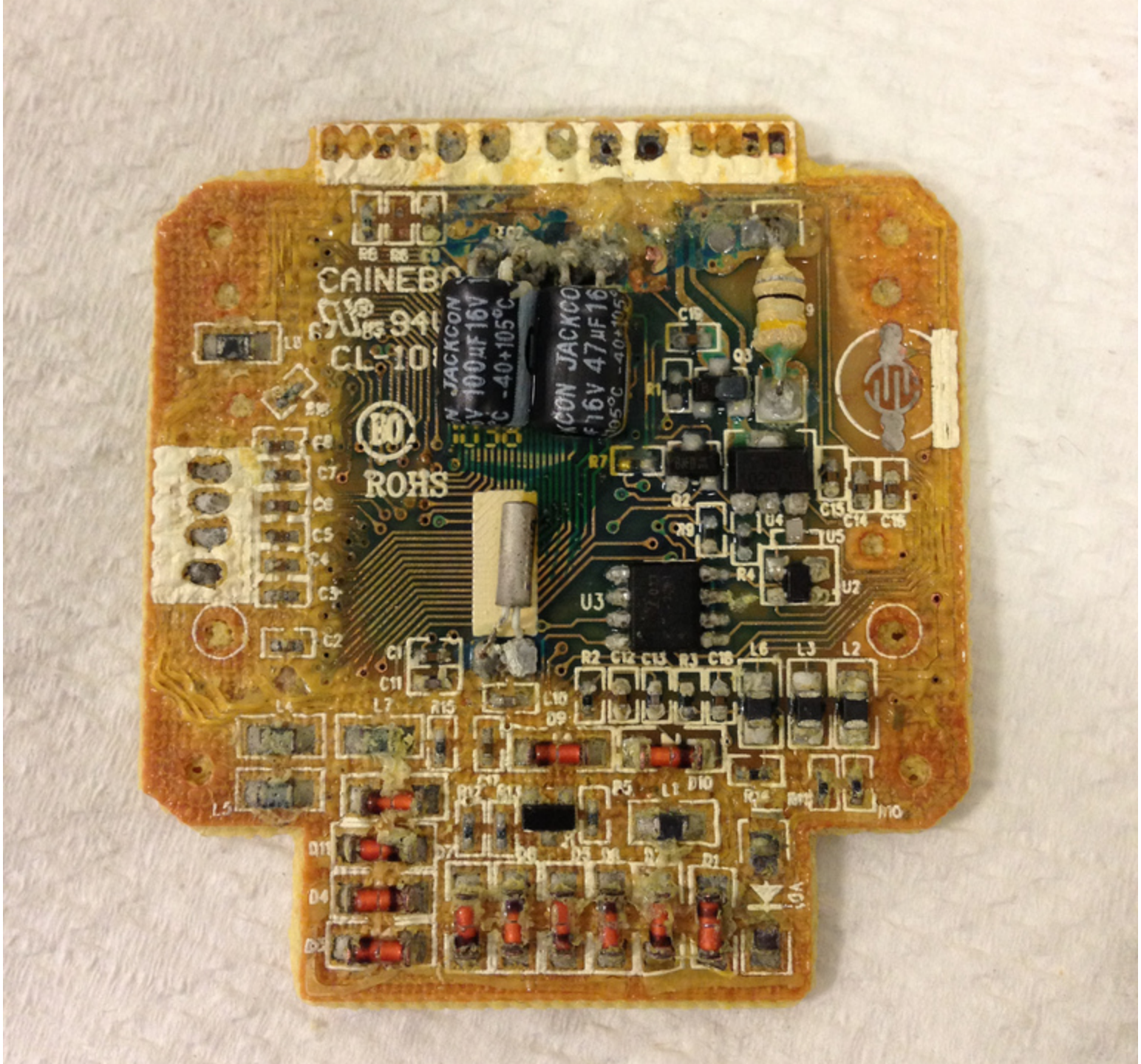
1. Use rompar (optical decoding)
2. ~~Probe the chip~~
3. ~~“Glitch it”~~
4. ~~Differential power analysis~~

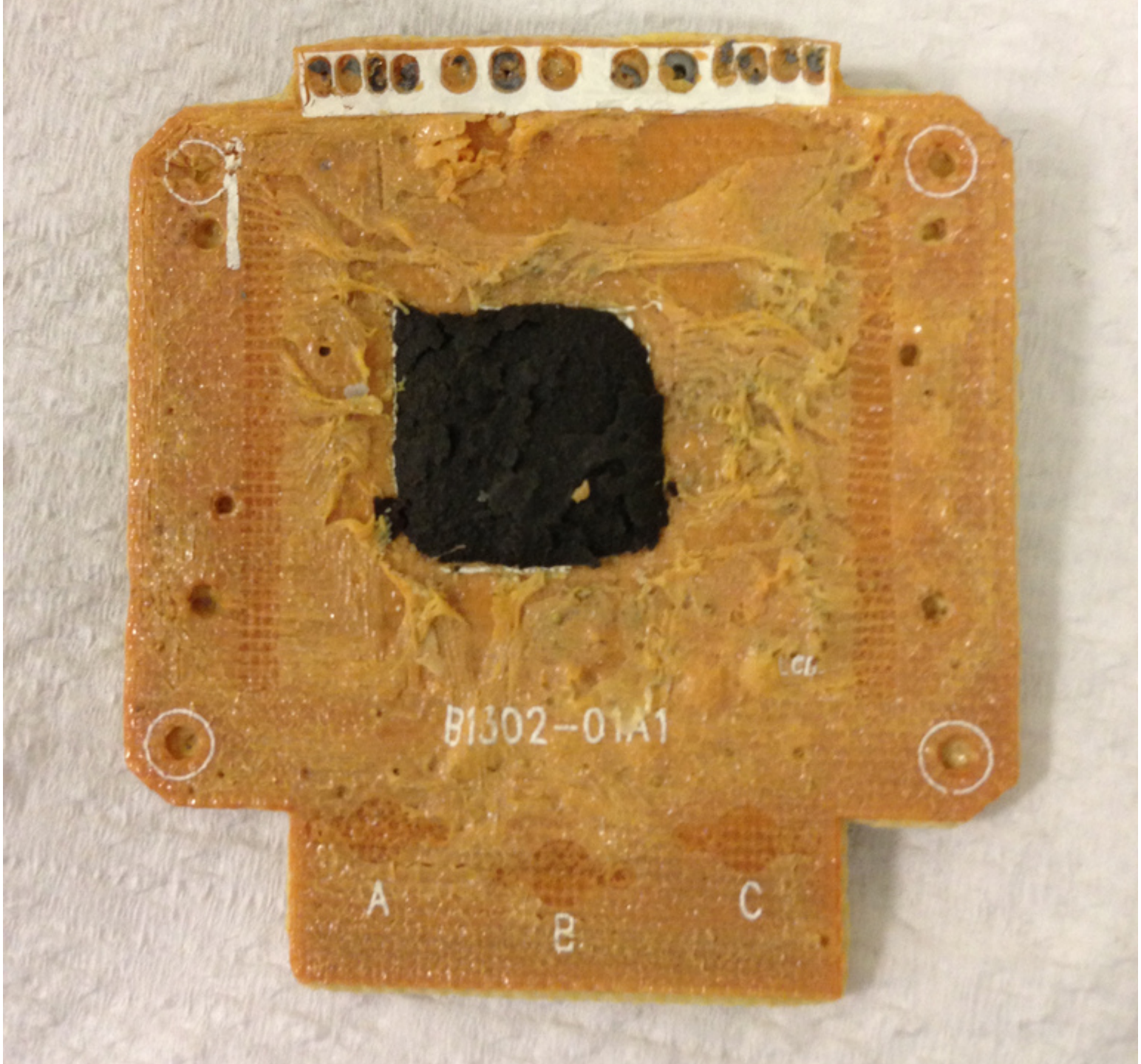
Decoding with a Light Microscope

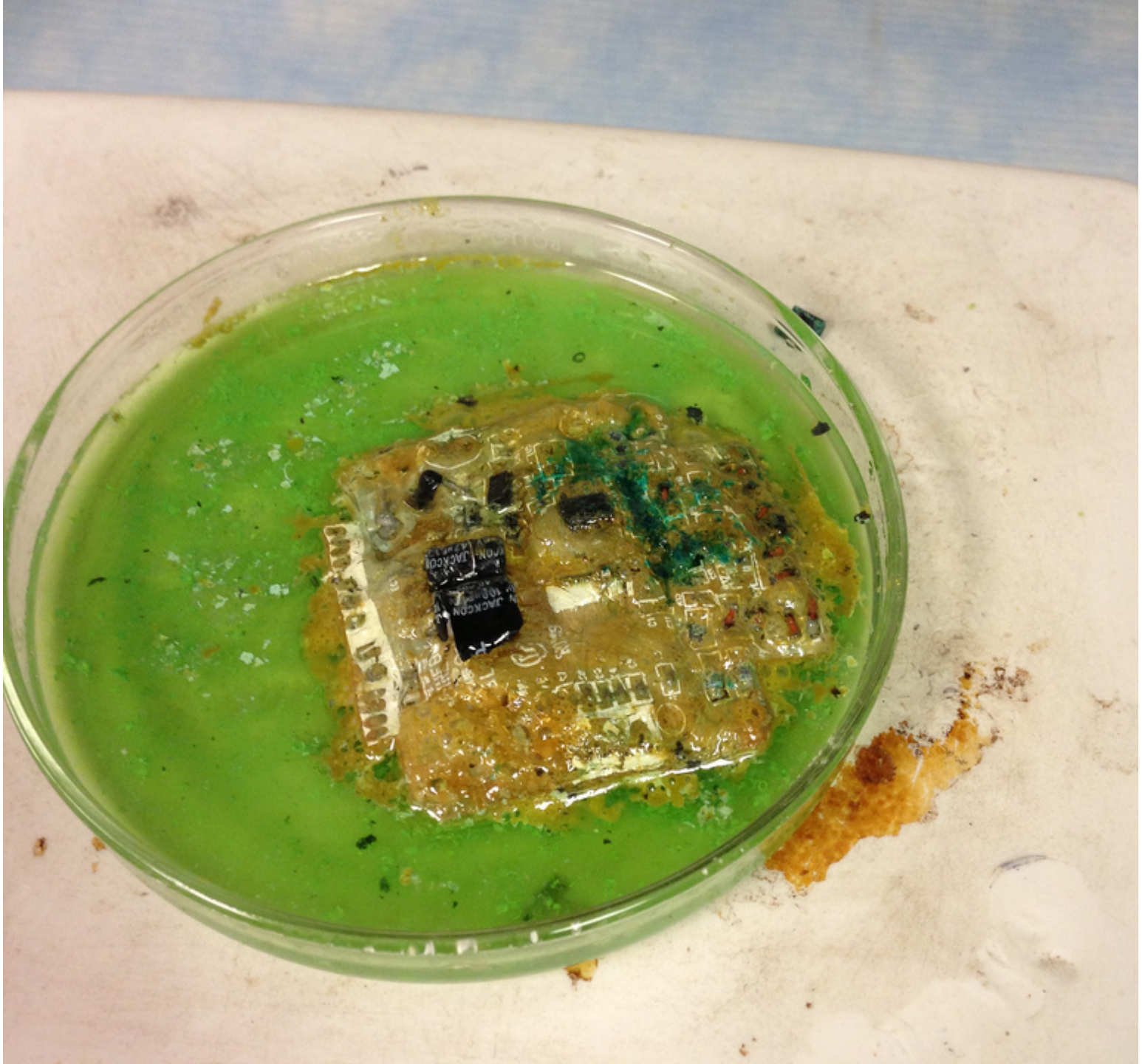
- John Maushammer attempted to decode the Tamagotchi ROM with a light microscope
- Could not see bits
  - Theoretically should have been one bit/four pixels
- Bits might be too small
- Could need grinding or polishing

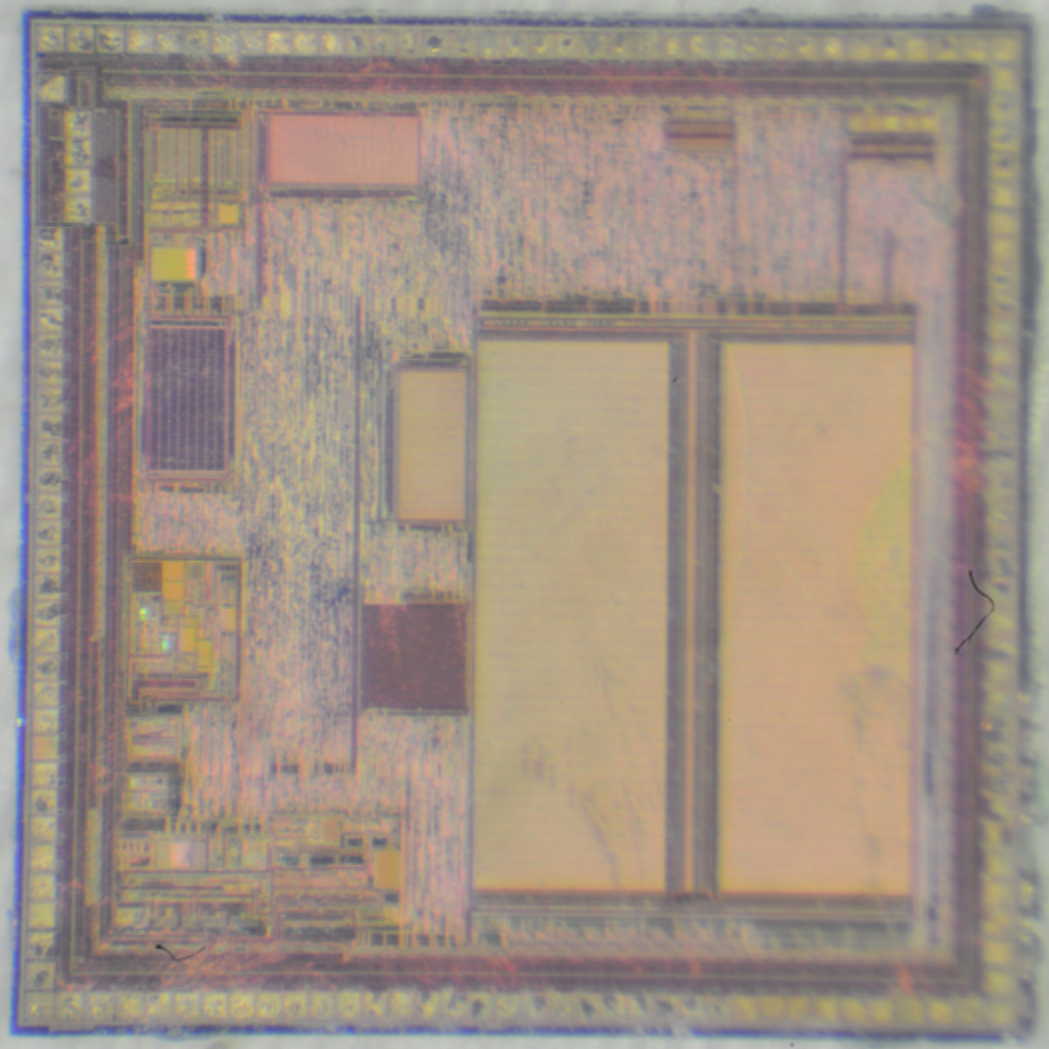


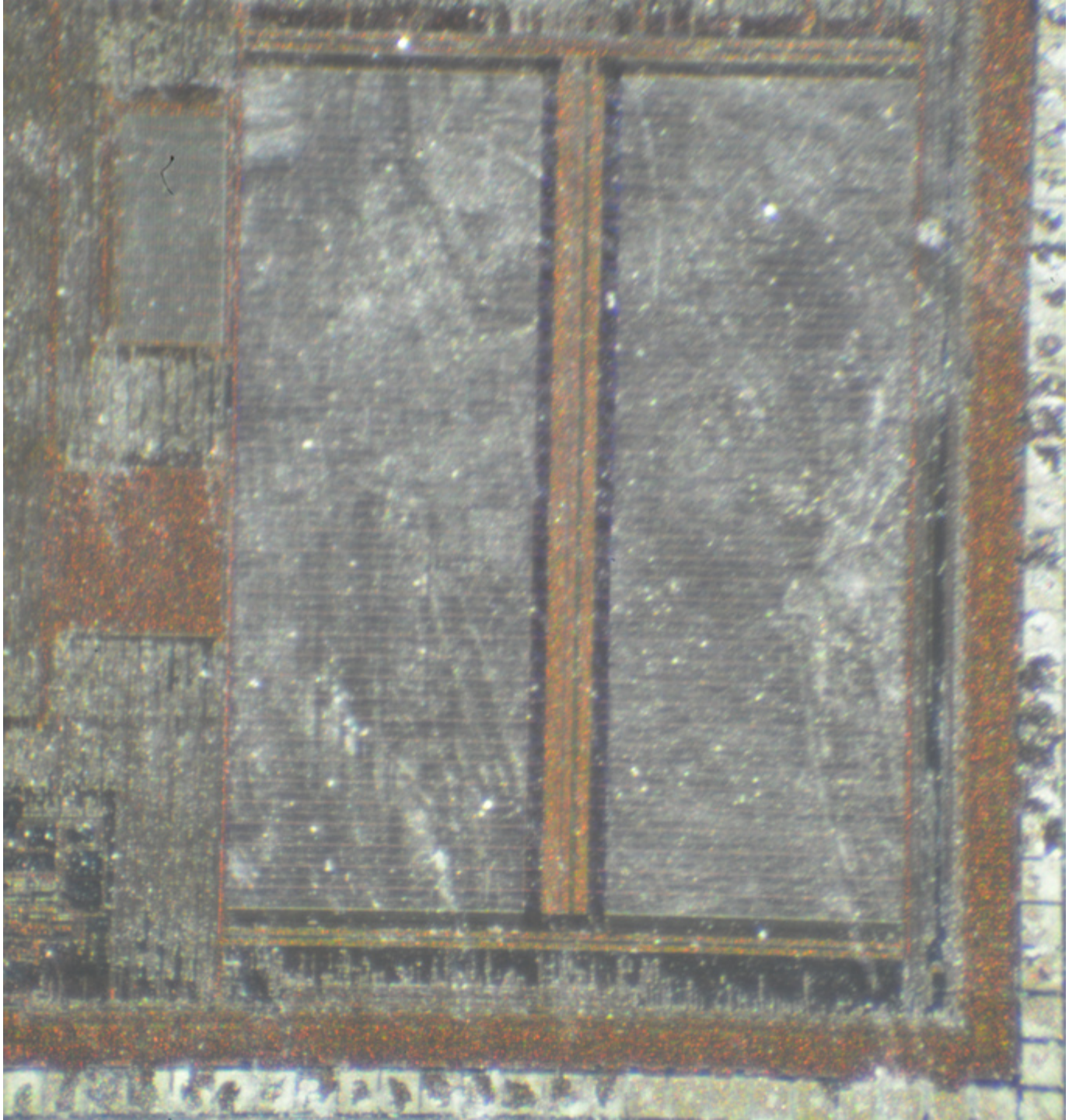

CORNING Stirrer/Hot Plate
power
hot top
Dial Setting
0 OFF OFF
1-2 77°F 25°C
3 194°F 90°C
4 338°F 170°C
5 446°F 230°C
6 572°F 300°C
7 770°F 410°C
8 860°F 460°C
9 - 10 896°F 480°C
4 3 2 1 0
5 4 3 2 1 0

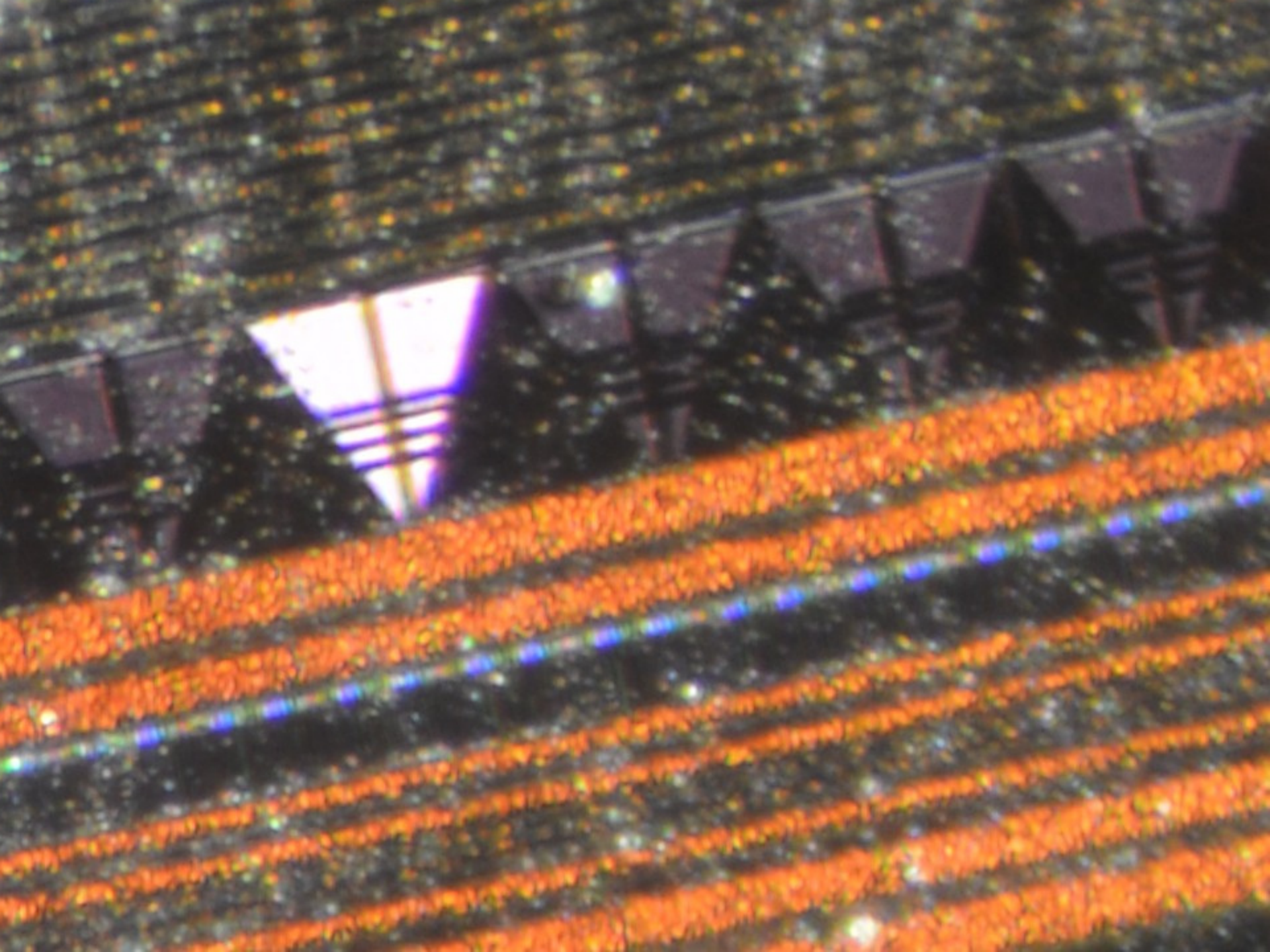

ROHS
U3
U4
U5
U2
Q1
Q2
Q3

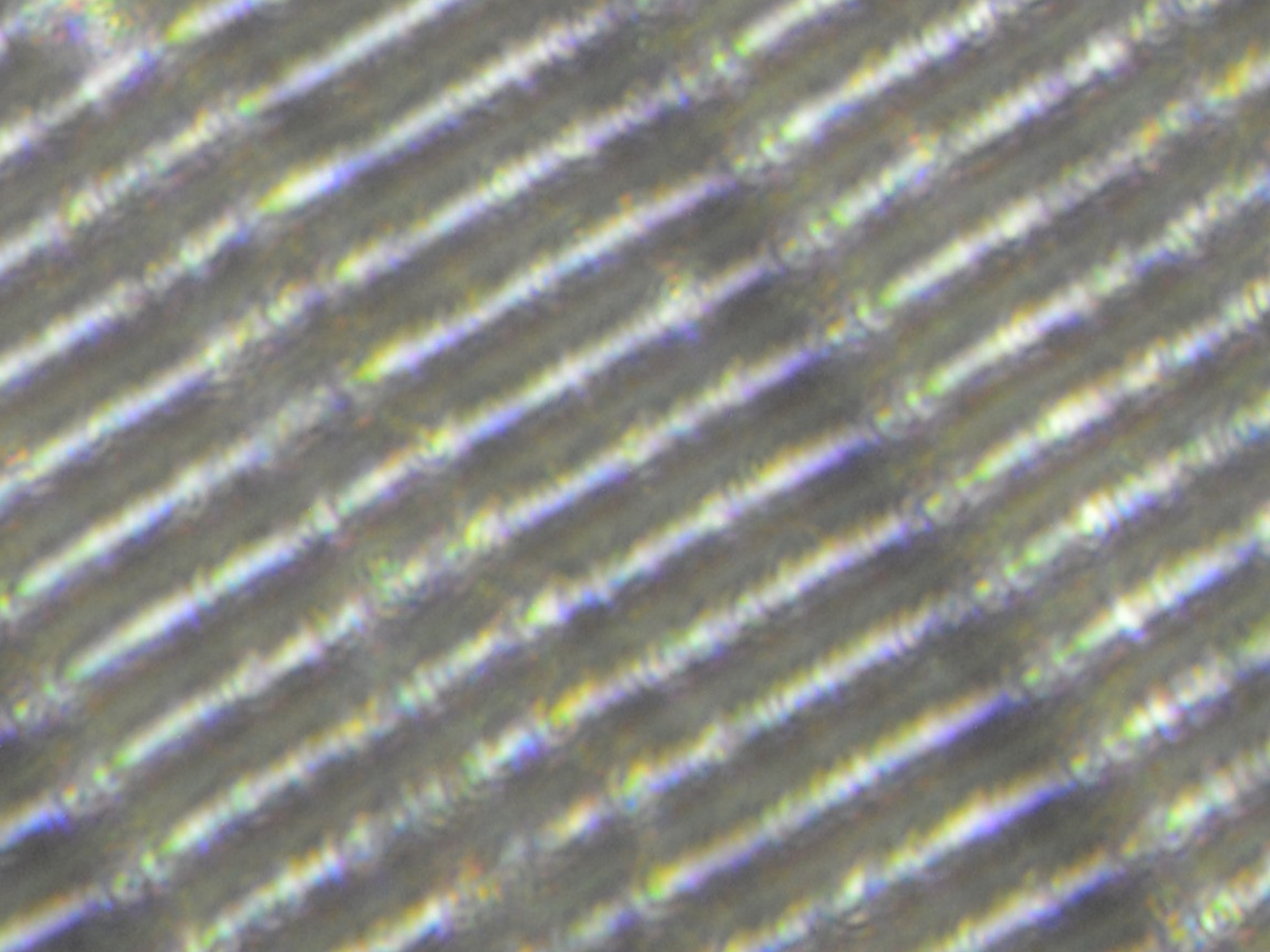

B1302-01A1
A
B
C





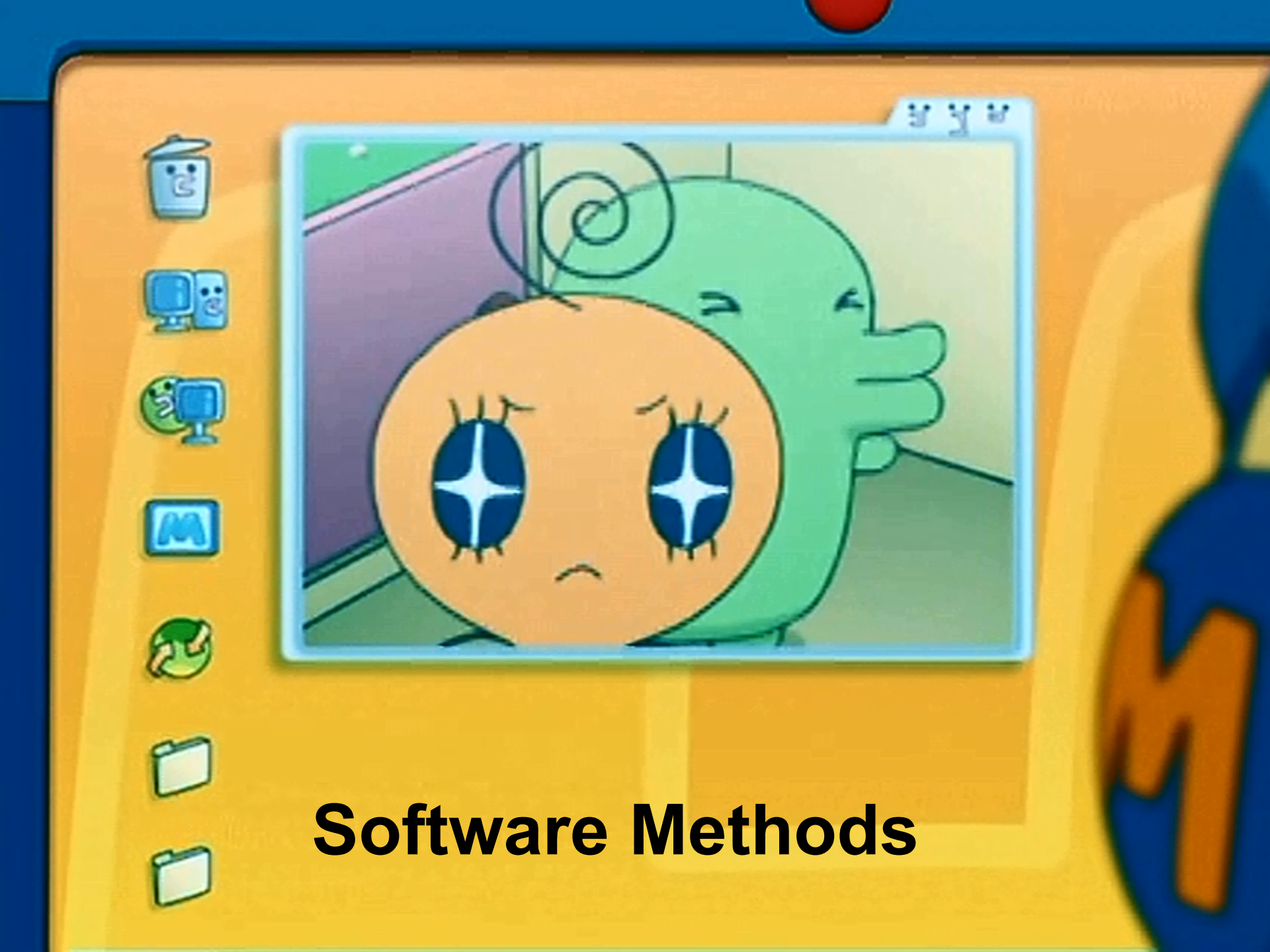






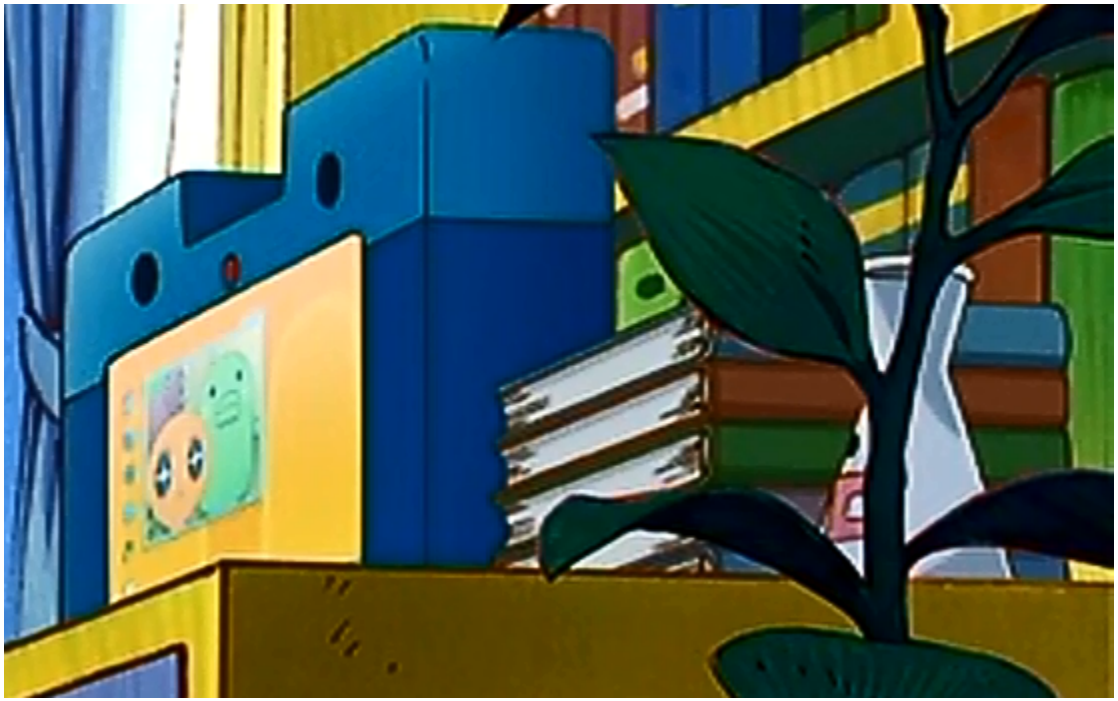
Decoding with a SEM

- An oil company employee offered to look at the ROM with a SEM and FIB
  - Apparently mining and oil companies have tons of them sitting around
- Unfortunately, there was a long queue
- Started looking at software methods in the meantime


Software Methods

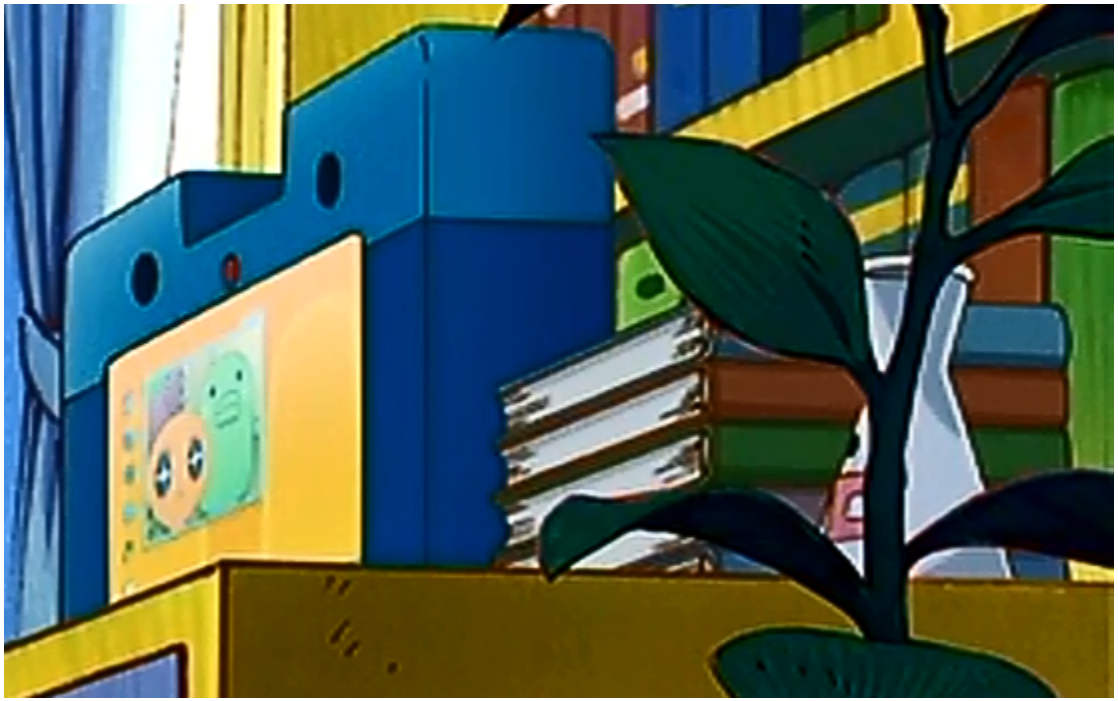
Software Possibilities

1. “Use ROP”
2. Circumvent Windows stack cookies



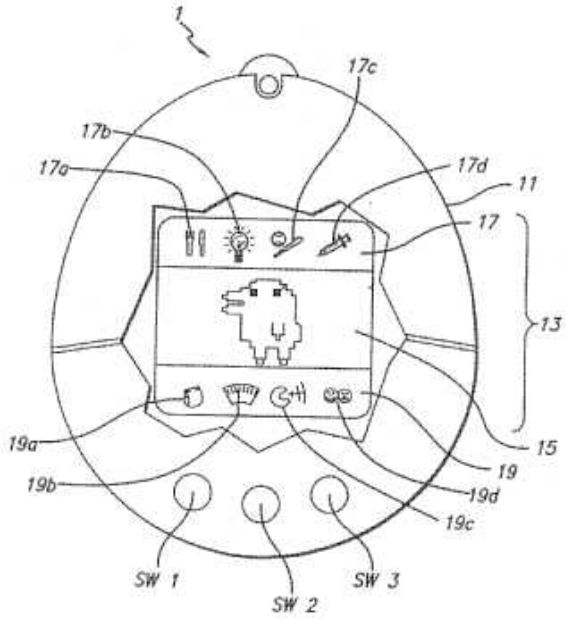
Software Possibilities

1. ~~“Use ROP”~~
2. ~~Circumvent Windows stack cookies~~



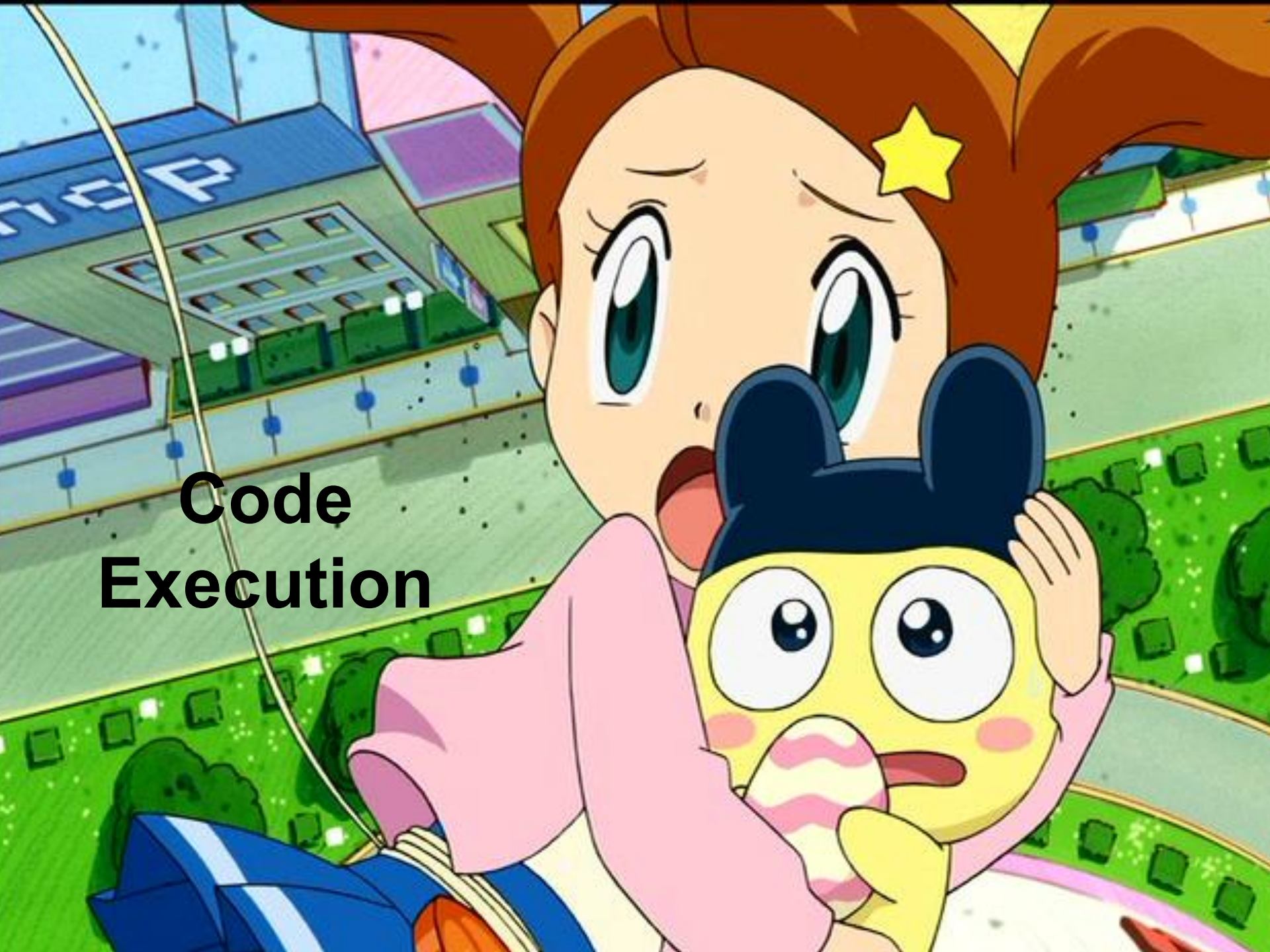
Software Possibilities

- My problems are:
  - Don't know how to execute code or dump ROM
  - Don't have any debug outputs
  - Don't have any working code or shell code
- My problems are not:
  - Exploit mitigations
  - Non-executable memory



The Hannah Montana Hackers

- Dumped the ROM of a Hannah Montana toy by dumping ROM to LCD and photographing it
- Unable to get further details
- Possible on a Tamagotchi?


Code
Execution

Game Logic

- The Tama-Go reads less than 50 bytes of non-image data during all figure functionality
- Game logic is represented by a one byte code
  - This logic is executed with images from figure
- Changing this code can cause a jump to non-game screens
  - Stats, food, death, etc. Every screen was available
- Many codes caused freezing

6502 Facts

- Memory mapped into a single address space
- No MMU
  - Unmapped addresses return 0 (usually)
  - Invalid instructions execute undefined behaviour
- Reset is rare
  - Great for explotation

First Attempt

- Assumed ‘game codes’ were indexes into a jump table
  - Invalid indexes would cause jumps (RTS) to non-pointer data
- Only controllable RAM is LCD RAM
  - 0x1000-0x1200
- Made a NOP sled and hoped

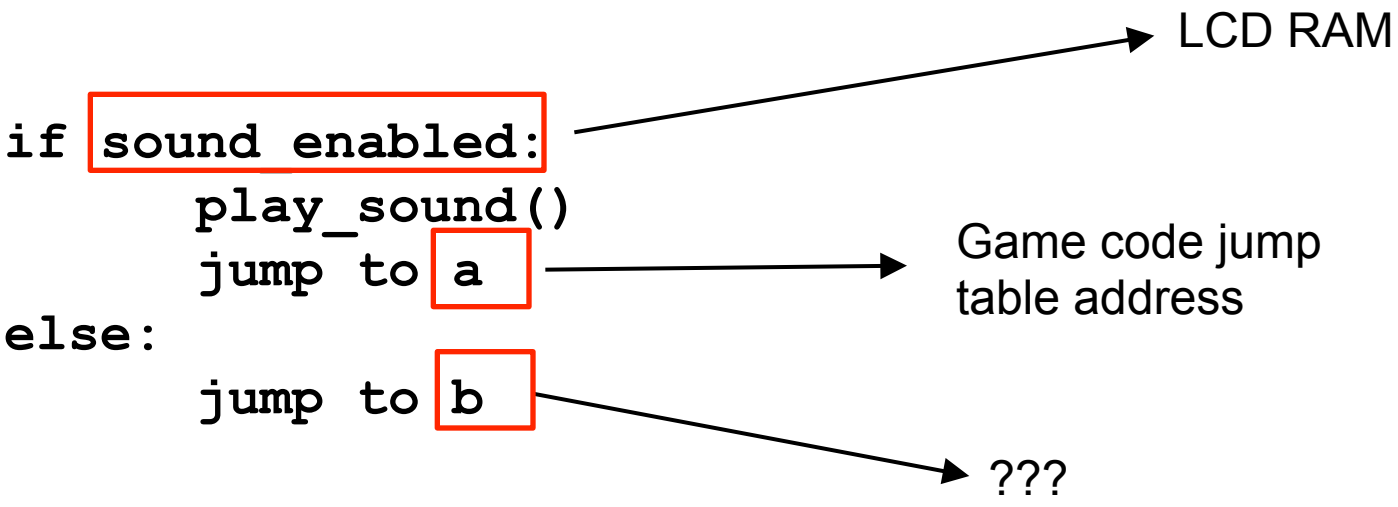

© B.W. / T.T. Bandai 2010 CHINA

Code 0xCC

- Did not work, but code 0xCC had interesting behavior
  - Buzzed when bit 3 of byte 68 was set and detected figure detach
  - Froze otherwise
- Also noticed that some middle indexes

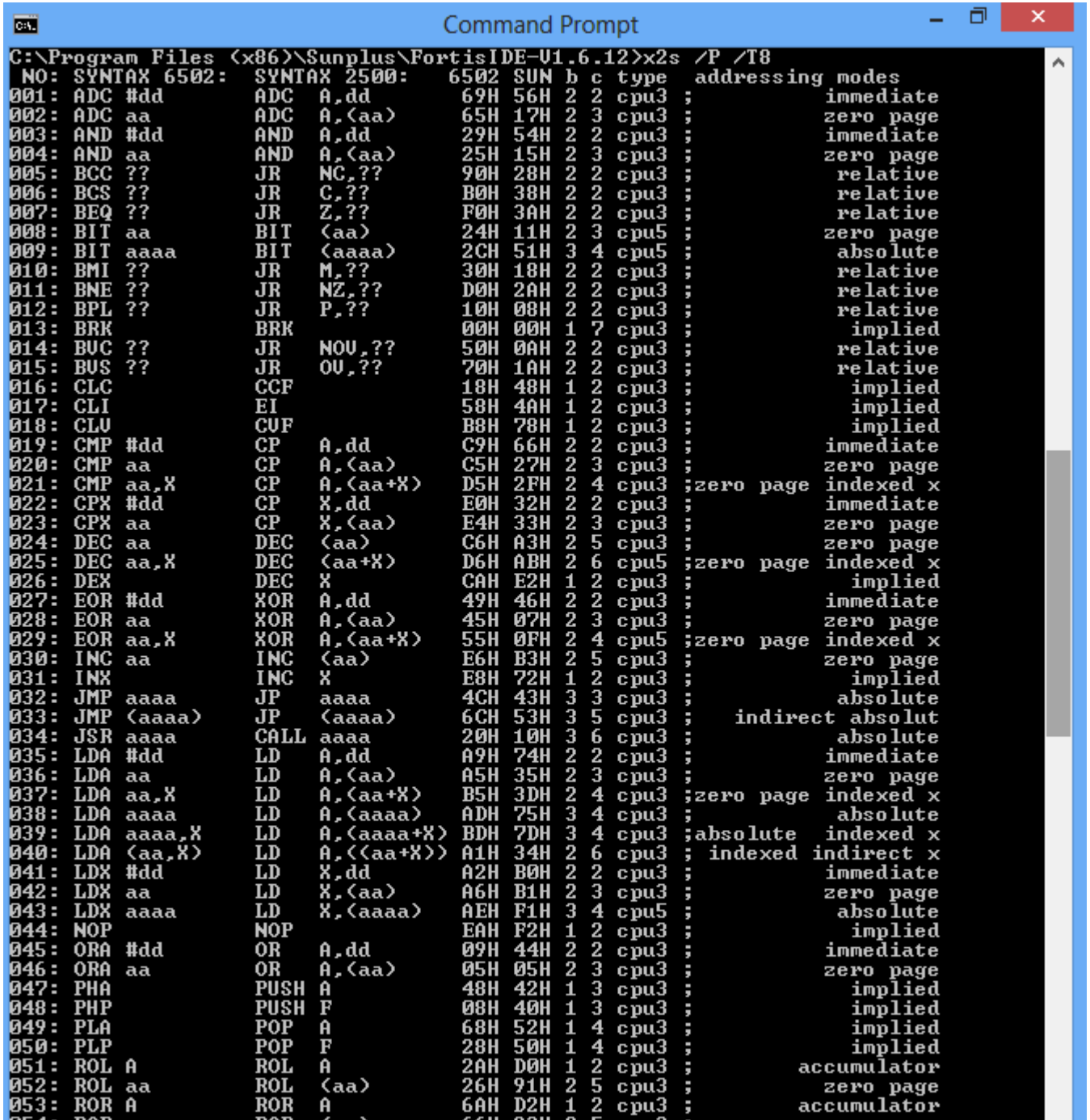
New Theory

- All indexes are valid, but the stack isn't set up correctly
- 0xCC plays the noise when button pressed



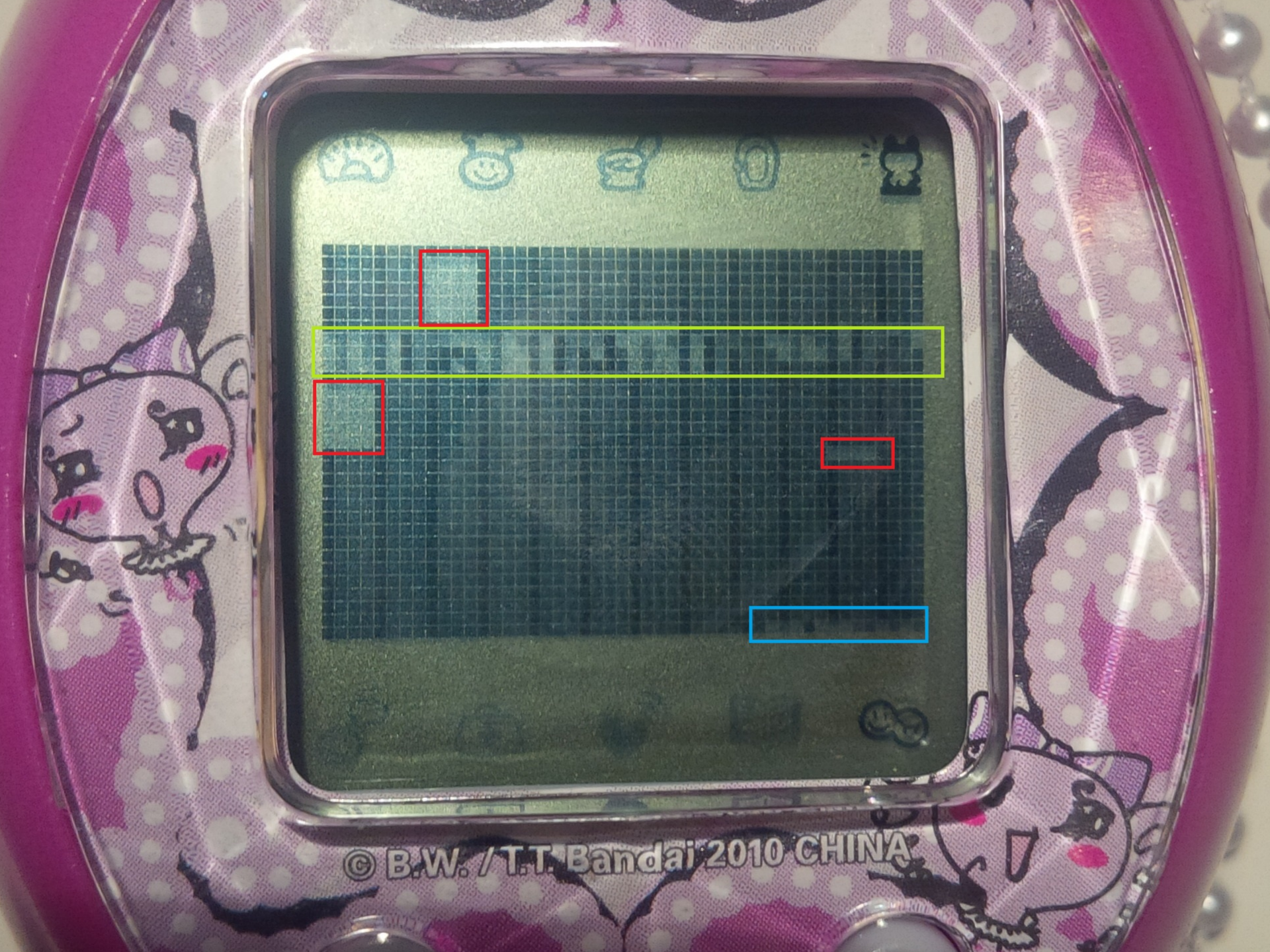
New Theory

- But if
  - A pointer to the LCD RAM is on the stack
  - Stack confusion is occurring
  - There's 255 possibilities
- Why isn't it working?


Command Prompt
C:\Program Files (x86)\Sunplus\FortisIDE-V1.6.12>x2s /P /T8
NO: SYNTAX 6502: SYNTAX 2500: 6502 SUN b c type addressing modes
001: ADC #dd ADC A,dd 69H 56H 2 2 cpu3 ; immediate
002: ADC aa ADC A,(aa) 65H 17H 2 3 cpu3 ; zero page
003: AND #dd AND A,dd 29H 54H 2 2 cpu3 ; immediate
004: AND aa AND A,(aa) 25H 15H 2 3 cpu3 ; zero page
005: BCC ?? JR NC,?? 90H 28H 2 2 cpu3 ; relative
006: BCS ?? JR C,?? B0H 38H 2 2 cpu3 ; relative
007: BEQ ?? JR Z,?? F0H 3AH 2 2 cpu3 ; relative
008: BIT aa BIT (aa) 24H 11H 2 3 cpu5 ; zero page
009: BIT aaaa BIT (aaaa) 2CH 51H 3 4 cpu5 ; absolute
010: BMI ?? JR M,?? 30H 18H 2 2 cpu3 ; relative
011: BNE ?? JR NZ,?? D0H 2AH 2 2 cpu3 ; relative
012: BPL ?? JR P,?? 10H 08H 2 2 cpu3 ; relative
013: BRK BRK 00H 00H 1 7 cpu3 ; implied
014: BVC ?? JR NOV,?? 50H 0AH 2 2 cpu3 ; relative
015: BVS ?? JR OV,?? 70H 1AH 2 2 cpu3 ; relative
016: CLC CCF 18H 48H 1 2 cpu3 ; implied
017: CLI EI 58H 4AH 1 2 cpu3 ; implied
018: CLV CVF B8H 78H 1 2 cpu3 ; implied
019: CMP #dd CP A,dd C9H 66H 2 2 cpu3 ; immediate
020: CMP aa CP A,(aa) C5H 27H 2 3 cpu3 ; zero page
021: CMP aa,X CP A,(aa+X) D5H 2FH 2 4 cpu3 ;zero page indexed x
022: CPX #dd CP X,dd E0H 32H 2 2 cpu3 ; immediate
023: CPX aa CP X,(aa) E4H 33H 2 3 cpu3 ; zero page
024: DEC aa DEC (aa) C6H A3H 2 5 cpu3 ; zero page
025: DEC aa,X DEC (aa+X) D6H ABH 2 6 cpu5 ;zero page indexed x
026: DEX DEC X CAH E2H 1 2 cpu3 ; implied
027: EOR #dd XOR A,dd 49H 46H 2 2 cpu3 ; immediate
028: EOR aa XOR A,(aa) 45H 07H 2 3 cpu3 ; zero page
029: EOR aa,X XOR A,(aa+X) 55H 0FH 2 4 cpu5 ;zero page indexed x
030: INC aa INC (aa) E6H B3H 2 5 cpu3 ; zero page
031: INX INC X E8H 72H 1 2 cpu3 ; implied
032: JMP aaaa JP aaaa 4CH 43H 3 3 cpu3 ; absolute
033: JMP (aaaa) JP (aaaa) 6CH 53H 3 5 cpu3 ; indirect absolut
034: JSR aaaa CALL aaaa 20H 10H 3 6 cpu3 ; absolute
035: LDA #dd LD A,dd A9H 74H 2 2 cpu3 ; immediate
036: LDA aa LD A,(aa) A5H 35H 2 3 cpu3 ; zero page
037: LDA aa,X LD A,(aa+X) B5H 3DH 2 4 cpu3 ;zero page indexed x
038: LDA aaaa LD A,(aaaa) ADH 75H 3 4 cpu3 ; absolute
039: LDA aaaa,X LD A,(aaaa+X) BDH 7DH 3 4 cpu3 ;absolute indexed x
040: LDA (aa,X) LD A,((aa+X)) A1H 34H 2 6 cpu3 ; indexed indirect x
041: LDX #dd LD X,dd A2H B0H 2 2 cpu3 ; immediate
042: LDX aa LD X,(aa) A6H B1H 2 3 cpu3 ; zero page
043: LDX aaaa LD X,(aaaa) AEH F1H 3 4 cpu5 ; absolute
044: NOP NOP EAH F2H 1 2 cpu3 ; implied
045: ORA #dd OR A,dd 09H 44H 2 2 cpu3 ; immediate
046: ORA aa OR A,(aa) 05H 05H 2 3 cpu3 ; zero page
047: PHA PUSH A 48H 42H 1 3 cpu3 ; implied
048: PHP PUSH F 08H 40H 1 3 cpu3 ; implied
049: PLA POP A 68H 52H 1 4 cpu3 ; implied
050: PLP POP F 28H 50H 1 4 cpu3 ; implied
051: ROL A ROL A 2AH D0H 1 2 cpu3 ; accumulator
052: ROL aa ROL (aa) 26H 91H 2 5 cpu3 ; zero page
053: ROR A ROR A 6AH D2H 1 2 cpu3 ; accumulator

Code Execution

- Switched instruction sets
- Used simpler shellcode
- Using the correct instruction set, it worked on the fourth index I tried, 0xd4


© B.W. / T.T. Bandai 2010 CHINA


ROM Dump

Dumping Memory

- Wrote code to dump entire memory space of Tamagotchi
- Output memory over SPI using port A (buttons)
- Decoded output with signal analyzer

Paging

- The ROM is larger than the memory space
- First page is always mapped
- Other pages are mapped one at a time
- Determined 0x3000 is page port
- Dumped all 19 pages

Pages

- Quickly identified pages by inspection
  - Pages 0 to 6 are code
  - Pages 7 to 9 are blank
  - Page 10 contains images and a image pointer table
  - Pages 11 to 18 contain image data

Images

- Dumped images from image pages


Test
Program

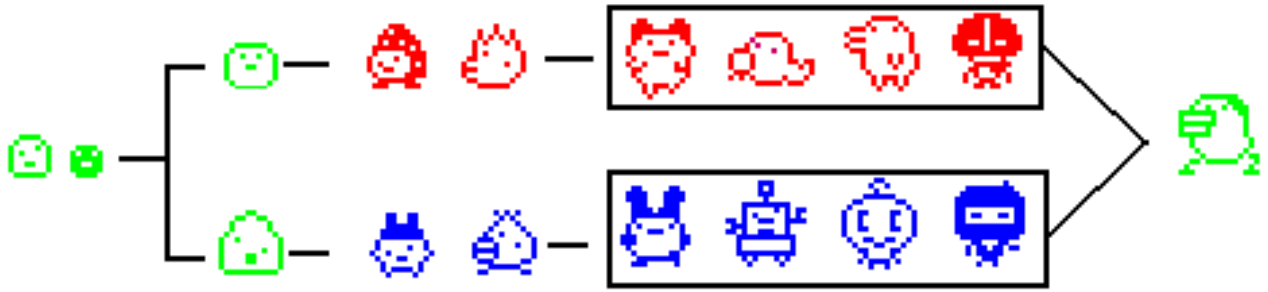
GeneralPlus Test Program

- Analyzed GeneralPlus Test Program
- Hoped it would make dumping other GP ROMs easier

BFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	80	90	88	97	A0	B0	A6	B5	90	9E	98	A5	AC	BA	B2	BF	C0	D0	C4	D3	E0
C015	F0	E2	F1	C8	D6	CC	D9	E4	F2	E6	F3	A0	AC	A8	B3	B8	C4	BE	C9	B0	BA	B8	C1	C4	CE	CA	D3	D0	DC	D4	DF	E8	F4	EA	F5	D8	E2
C03A	DC	E5	EC	F6	EE	F7	00	10	00	0F	20	30	1E	2D	00	0E	00	0D	1C	2A	1A	27	40	50	3C	4B	60	70	5A	69	38	46	34	41	54	62	4E
C05F	5B	00	0C	00	0B	18	24	16	21	00	0A	00	09	14	1E	12	1B	30	3C	2C	37	48	54	42	4D	28	32	24	2D	3C	46	36	3F	C0	C8	C8	CF
C084	D0	D8	D6	DD	D0	D6	D8	DD	DC	E2	E2	E7	E0	E8	E4	EB	F0	F8	F2	F9	E8	EE	EC	F1	F4	FA	F6	FB	E0	E4	E8	EB	E8	EC	EE	F1	F0
C0A9	F2	F8	F9	F4	F6	FA	FB	F0	F4	F4	F7	F8	FC	FA	FD	F8	FA	FC	FD	FC	FE	FE	FF	00	08	00	07	10	18	0E	15	00	06	00	05	0C	12
C0CE	0A	0F	20	28	1C	23	30	38	2A	31	18	1E	14	19	24	2A	1E	23	00	04	00	03	08	0C	06	09	00	02	00	01	04	06	02	03	10	14	0C
C0F3	0F	18	1C	12	15	08	0A	04	05	0C	0E	06	07	1B	1F	1C	20	78	A2	FF	9A	A9	00	8D	76	30	8D	70	30	8D	71	30	8D	72	30	8D	54
C118	30	8D	56	30	8D	06	30	8D	65	30	8E	55	30	8E	73	30	8E	74	30	8E	75	30	AD	05	30	F0	37	29	08	F0	06	20	8C	C1	4C	13	C8
C13D	AD	05	30	29	04	F0	08	A9	08	8D	16	30	4C	12	C3	AD	05	30	29	02	F0	08	A9	07	8D	16	30	4C	12	C3	AD	05	30	29	01	F0	08
C162	A9	0A	8D	16	30	4C	12	C3	20	32	C3	20	18	C3	AD	12	30	29	40	D0	05	A9	00	8D	01	30	A2	5A	8E	0B	30	A9	00	8D	0C	30	20
C187	5C	C3	4C	12	C3	A9	00	8D	05	30	60	A0	FF	A2	FF	EA	EA	EA	EA	EA	EA	EA	EA	EA	EA	EA	CA	D0	F2	88	D0	ED	60	48	8A	48	98
C1AC	48	AD	00	30	48	A5	92	09	80	85	92	A5	90	29	7F	85	90	8D	70	30	68	8D	00	30	68	A8	68	AA	68	8D	97	30	40	48	8A	48	98
C1D1	48	AD	00	30	48	A5	93	09	20	85	93	A5	91	29	DF	85	91	8D	71	30	68	8D	00	30	68	A8	68	AA	68	8D	9D	30	40	48	8A	48	98

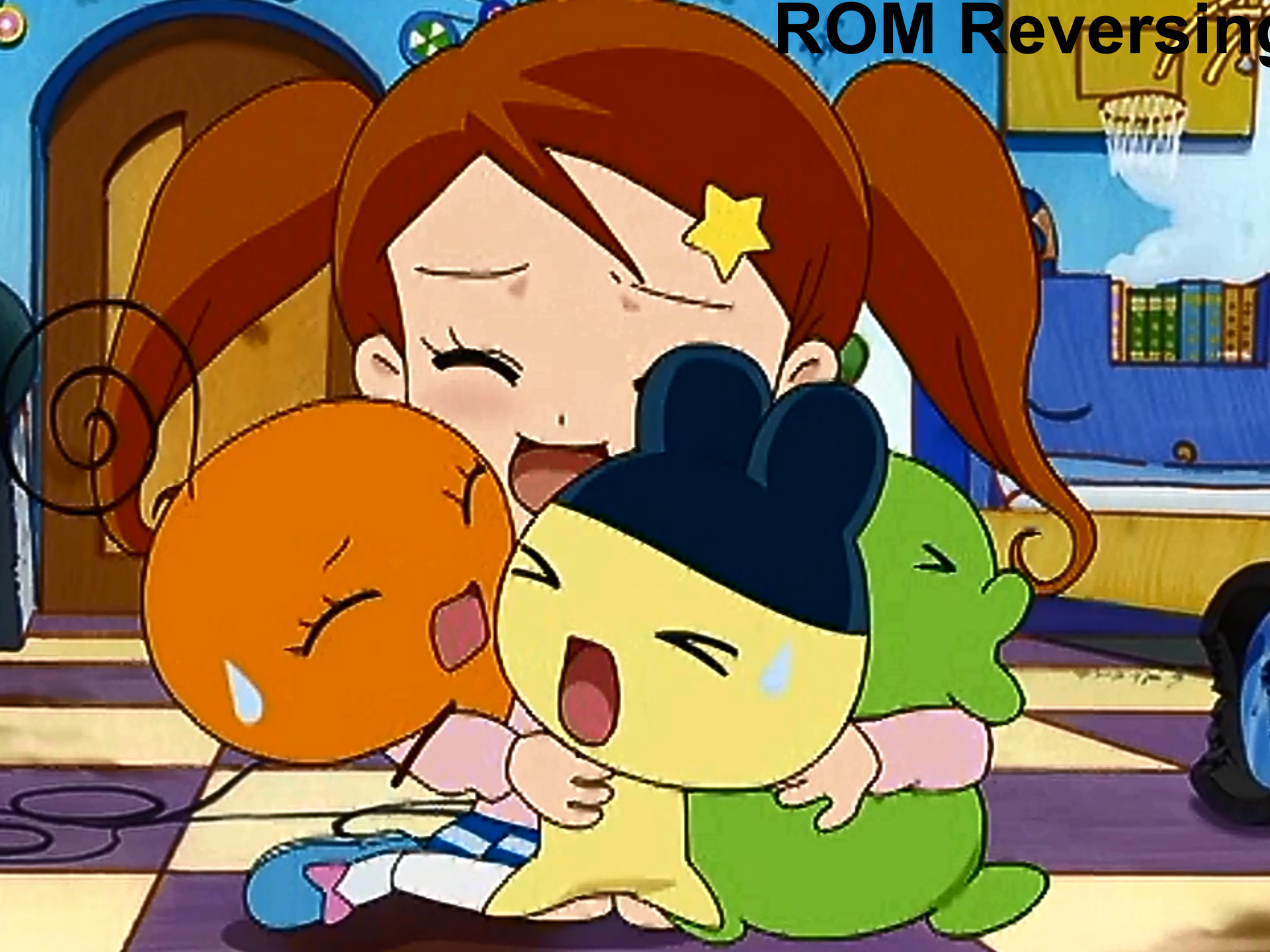
GeneralPlus Test Program

- Polls port A for a code, runs test and outputs results on port B
- Two interesting codes, 3 and 0x16
- Code 3 checksums custom address range
  - Unfortunately contains a bug so it doesn't work



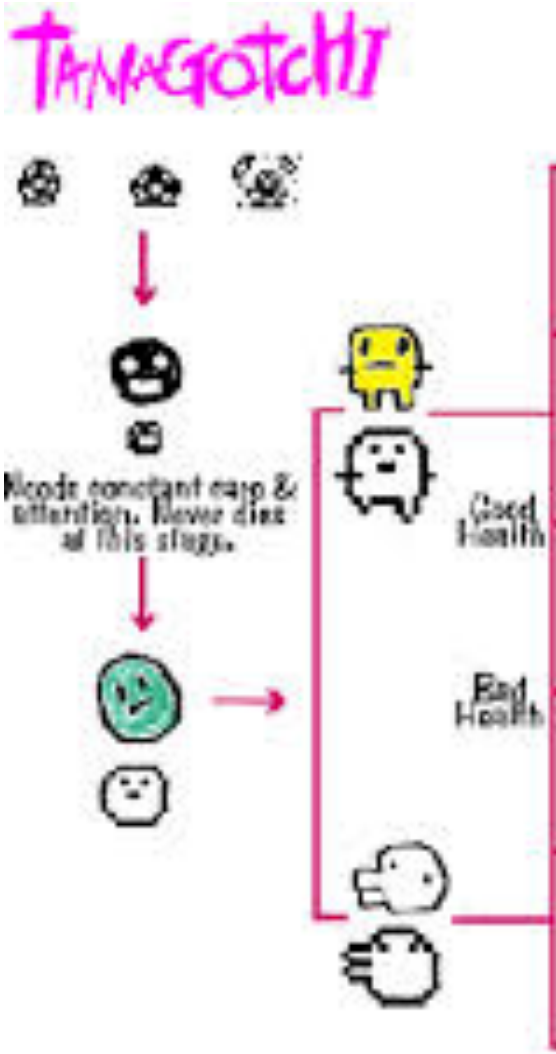
Test Program Code Dump

- Code 16 fills RAM up with code from Port B and jumps to it!
- Can dump code from any GeneralPlus LCD controller so long as Port A, Port B and TEST are bonded


ROM Reversing

ROM Reversing

- Started using IDA
  - Learning curve was steep
  - No paging support
- Eventually wrote a simulator based on py65



Tamagotchi Internals

- After start-up, Tamagotchis cycle through a single loop, driven by tm1 interrupts
- Always in one of 0x41 states
  - Table determines state actions
  - Can have substates and subsubstates and …
  - State entry behaves differently
  - States are responsible for all behaviour (buttons, sound) except for physical LCD update and SPI poll
  - A LOT of pointer tables

Secrets So Far …

- What makes a Tamagotchi a boy or a girl?
  - Determined from entropy source C4, based on how many times tm1 has fired since the Tamagotchi started up
- What toddler a baby grows into is random
  - Intentionally evened out
  - Some toddlers are higher-maintenance than others

Secrets So Far …

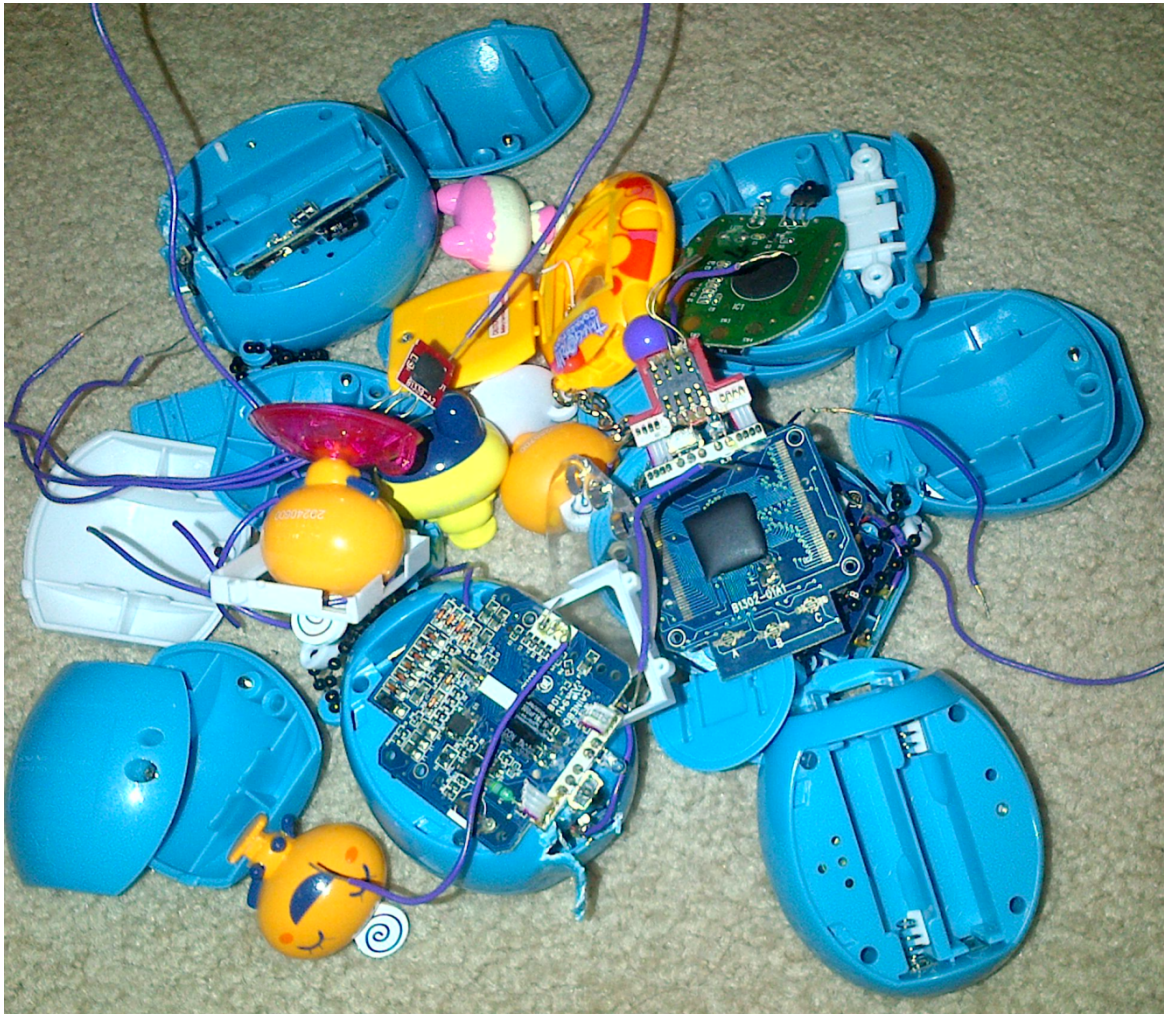
- What teen a toddler becomes is based on care
  - Two factors
- What adult a teen becomes depends on care and training
  - Toddler care matters
- You can potty train your Tamagotchi


Conclusion

Conclusions

- Dumped Tamagotchi code
- Learned to dump all GeneralPlus chips
- Learned about Tamagotchi internals
- Learned the secrets of Tamagotchi life
- Most importantly, good times were had by all…

Except for the Tamagotchis




Questions?
natashenka@kwartzlab.ca
@natashenka

More Info




http://www.kwartzlab.ca/author/natalies/

natashenka@kwartzlab.ca

@natashenka