

A 3D rendered image of a yellow Pikachu character with red cheeks and black-tipped ears, standing on a pink surfboard. The surfboard is angled diagonally across the frame. The background is a beach scene with a sandy shore, a turquoise ocean, and a blue sky with light clouds. A green hill is visible on the right side of the horizon.

Reversing the Pokémon Snap Station without a Snap Station

James Chambers
@jamchamb_

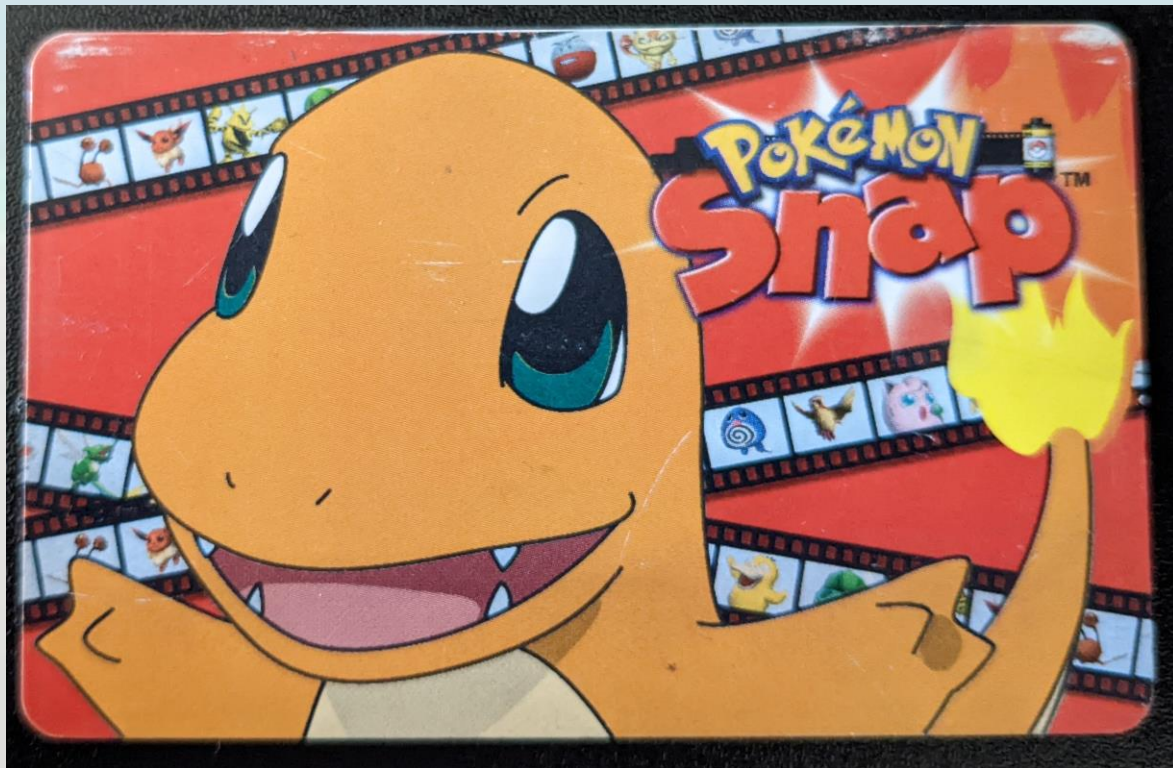
About Me

- Senior Security Consultant in NCC Group Hardware & Embedded Systems practice
- Learned RE for fun from fravia's Windows cracking & related sites like 3564020356
- Reversing retro games for fun, some nostalgia factor, no need to worry about NDAs/clients



Overview

- Reversing Pokémon Snap
- General highlights/challenges of the RE process
- Challenges from not having the kiosk hardware
- Joy Bus interface tool with iCEBreaker FPGA board
- Key things I stumbled on to go from WTFPGA/iCEBreaker FPGA Workshop tutorials or simple glitcher design to interfacing with an external bus





http://www.thecoverproject.net/view.php?game_id=575

Pokémon Snap

- Released in July 1999
- First 3D Pokémon game (outside Japan)
- Rail shooter style photography game
- Basically, a safari to take photos of Pokémon



<https://www.serebii.net/snap/> and https://assets.pokemon.com/assets/cms/img/video-games/snap/screenshots/Snap_ss9.jpg





James Artaius

<https://www.digitalcameraworld.com/tutorials/how-to-print-instax-photos-from-your-nintendo-switch-including-pokemon-snap>



Gotta Print 'em Out

One of the best features of Pokémon Snap is that you'll be able to convert your pictures into stickers. It's simple: just build up a portfolio of snapshots, then take your game down to the nearest Blockbuster Video store

and print out your favorites. It'll cost you a measly three bucks for 16 stickers—prices will be slightly higher in Canada—and the process takes no time at all. One-hour photo shops must be green with envy.

Turn Your Masterpieces into Stickers



The Gallery is made up of your four best shots, each of which will produce four stickers when you print them out at Blockbuster. You can shuffle these four around with pictures from the Report or your Album.

Sticker at Actual Size

Each of the 16 stickers on a sheet will be exactly this size and resolution and will adhere to most surfaces with ease.



The Sticker Machine Accepts All Major Pokémon Credit Cards



Look for the Pokémon display at your local Blockbuster store, then purchase a collectible smart card at the counter. Plug your Game Pak and the card into the display, choose your shots and print them up!

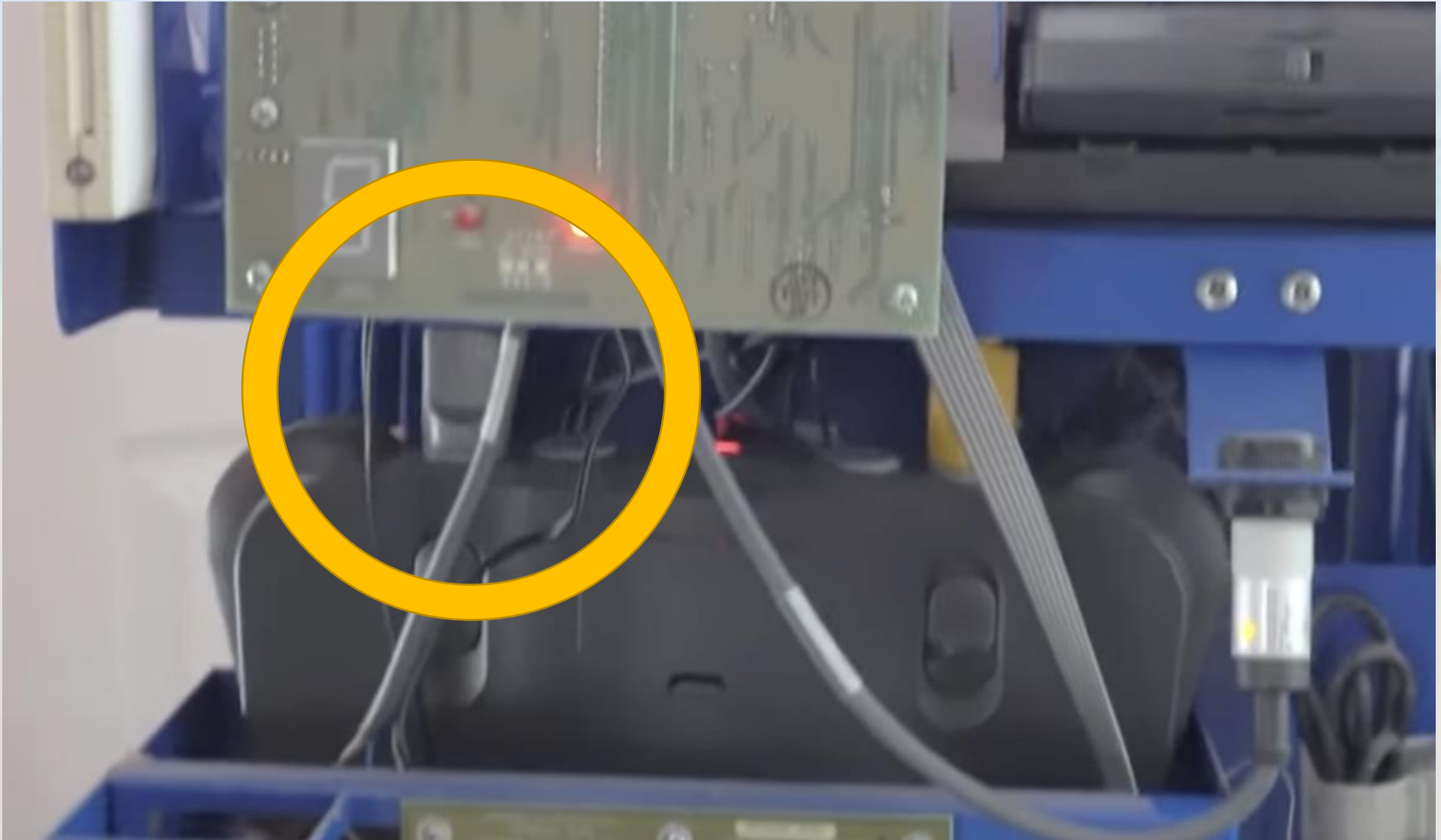


Put Your Best Shots in the Gallery






YouTube: PokeRev <https://youtu.be/tgOHRtbfxK0>



YouTube: PokeRev <https://youtu.be/tgOHRtbfxK0>

A yellow Pikachu is riding a pink surfboard on a sandy beach. The background shows a blue ocean and a clear sky. The text "Reverse Engineering" is overlaid on the image.

Reverse Engineering



00A08270	63	72	65	65	6E	2E	00	00	49	66	20	79	6F	75	20	6C	screen...If you l
00A08280	69	6B	65	20	74	68	65	73	65	20	70	69	63	74	75	72	ike these pictur
00A08290	65	73	2C	20	70	6C	65	61	73	65	0A	6D	61	6B	65	20	es, please.make
00A082A0	73	75	72	65	20	61	20	70	72	69	6E	74	20	63	72	65	sure a print cre
00A082B0	64	69	74	20	65	78	69	73	74	73	0A	74	68	65	6E	20	dit exists.then
00A082C0	70	72	65	73	73	20	5C	61	20	74	6F	20	70	72	69	6E	press \a to prin
00A082D0	74	2E	00	00	52	65	74	75	72	6E	20	74	6F	20	74	68	t...Return to th

*If you like these pictures, please
make sure a print credit exists
then press \a to print.*



US006383080B1

(12) **United States Patent**
Link et al.

(10) **Patent No.:** **US 6,383,080 B1**
(45) **Date of Patent:** **May 7, 2002**

(54) **KIOSK FOR PRINTING AND COMMUNICATING VIDEO GAME IMAGES**

(75) Inventors: **Patrick J. Link**, Carnation; **Ben Ong**, Seattle; **Yoshinobo Mantani**, Kirkland, all of WA (US)

5,184,830 A * 2/1993 Okada et al.
D443,623 S * 9/1999 Ohno
6,022,274 A * 2/2000 Takeda et al.
6,120,379 A * 9/2000 Tanaka et al.

* cited by examiner

(73) Assignee: **Nintendo Co., Ltd.**, Kyoto (JP)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

Primary Examiner—Mark Sager

(74) Attorney, Agent, or Firm—Nixon & Vanderhye P.C.

(57) **ABSTRACT**

A kiosk includes a game machine console having a processing system. The game machine has a connector that, in use, is connected to a cartridge comprising a memory. A video printer has a video input connected to a video output of the game machine console and a video output connected to a video input of a television. Control circuitry is connected to the game machine and to the video printer. The control circuitry is configured to enable a user to selectively print out images stored in the memory of the cartridge.

(21) Appl. No.: **09/567,022**

(22) Filed: **May 9, 2000**

(51) Int. Cl.⁷ **A63F 9/24**

(52) U.S. Cl. **463/47; 463/30**

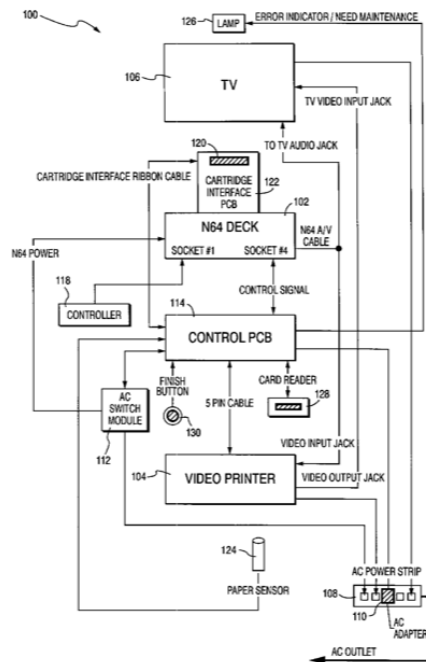
(58) Field of Search 463/1, 30-31,
463/36, 44-47, 40-42, 25, 29; 273/148 B;
355/18-19; 348/460, 552

References Cited

U.S. PATENT DOCUMENTS

4,095,791 A * 6/1978 Smith et al.

14 Claims, 5 Drawing Sheets



U.S. Patent

May 7, 2002

Sheet 2 of 5

US 6,383,080 B1

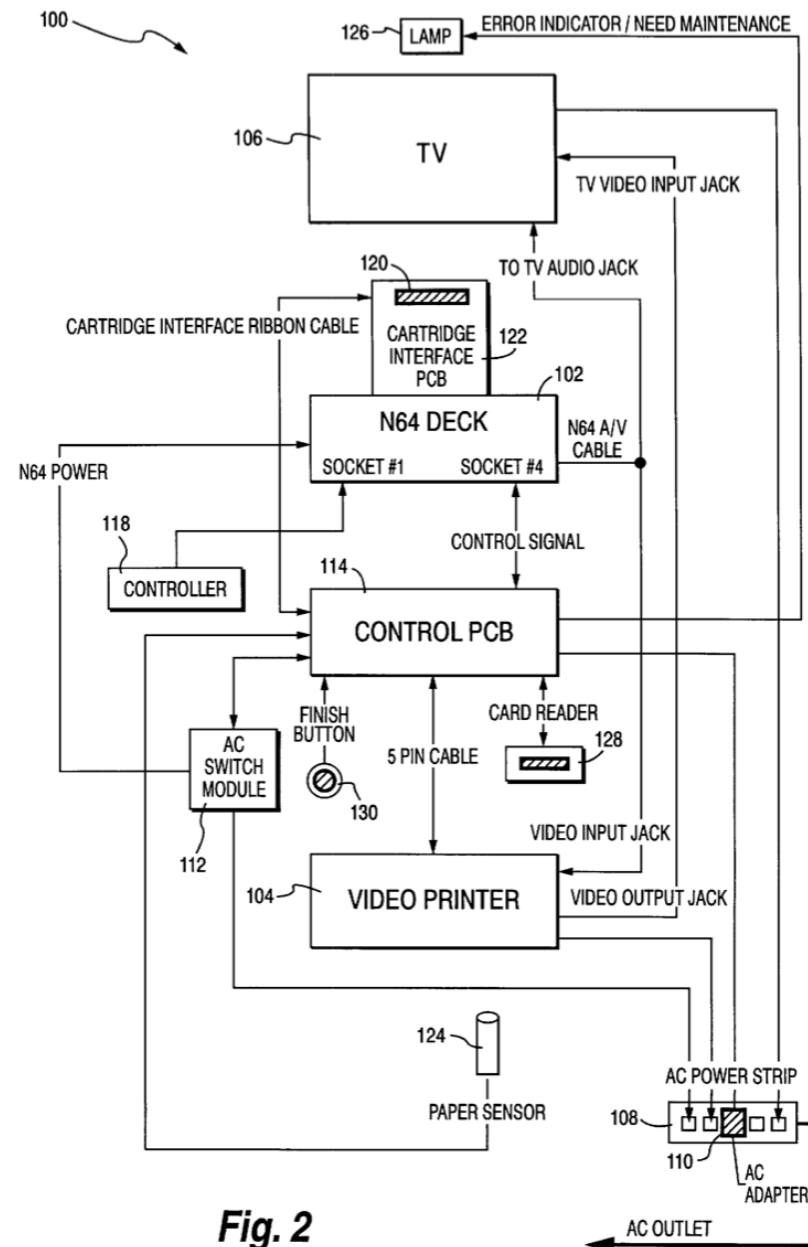


Fig. 2

The resetting of the console is performed under two circumstances. First, switching between the normal “play” mode of the game program executed by console **102** and the

“print” mode requires that the game be restarted. The game enters print mode if controller communication activity is detected on (controller) socket **4** of console **102** within the first few seconds after the game is started. Socket **4** is connected to control PCB **114**, which provides the communication activity only if the particular restart is for the purpose of entering the print mode. If the restart is not for the purpose of printing, socket **4** communication activity is delayed for about five seconds. This causes the game to start in a normal operating mode. Socket **4** communication becomes available after about five seconds so that control PCB **114** can be informed when, for example, a “Print” button is pressed in a game.

The printing operation is summarized as follows:

- (1) The user arranges images in the Gallery as desired.
- (2) The user presses a “Print” button on the Gallery screen. The game sends a Print Request to control PCB **114** via socket **4**.
- (3) If a card with credit is present, control PCB **114** restarts console **102**, enabling socket **4** communication immediately.
- (4) The game starts in print mode, and printer initialization commands are sent to control PCB **114** via socket **4**.
- (5) Control PCB **114** sends initialization commands (16-sticker mode, etc.) to video printer **104** via, for example, a five pin cable.

(6) The game draws the image for first sticker on the screen of television **106** and sends a Capture command to printer **104** via control PCB **114**. This step is repeated for each of the 16 images (even if the images are duplicated).

- 5 (7) Control PCB **114** deducts credit from card. If this fails for any reason (no card, no credit, etc.), the game is restarted in normal operating mode and the captured images are discarded.

(8) Control PCB **114** sends a Print command to video printer **104**.

10

(9) Upon completion of printing, control PCB **114** restarts console **102**, delaying socket **4** communication for 5 seconds so that the game starts in normal mode.

Control PCB **114** controls the overall operation of kiosk **100**. Control PCB **114** includes a microcontroller (Microchip 16C65), a Nintendo N64® controller IC (the same as found in the standard consumer controller), a one digit LED display for displaying a number of print credits available, a first indicator LED for indicating a card error, a second indicator LED for indicating a printer problem, and connectors for interfacing to socket **4** of console **102**, to switch module **112**, to cartridge interface PCB **122**, to paper sensor **124**, to card reader **128**, to problem lamp **126**, and to Finish button **130**. The microcontroller of control PCB **114** is a one-time programmable device that is programmed with code used for general operation of kiosk **100**.

In accordance with another aspect of the present invention, the kiosk also includes communication circuitry for communications over a communication network (e.g., the Internet, the public switched telephone network, a local area network, etc.). The communication circuitry is usable to selectively transmit images stored in the memory of a memory cartridge over the communication network. In this way, images may, for example, be electronically mailed (e-mailed) to others. Kiosk users may also submit images from their own memory cartridges to an on-line album maintained, for example, on a web server computer. Kiosk users may also view images submitted to the on-line album by others and/or selectively print out images from the on-line album. In the case of images captured during video game play, an on-line album can be used to build a gaming community of players having common interests and provides an opportunity for game players to compare the results of game playing with each other.

Static Analysis

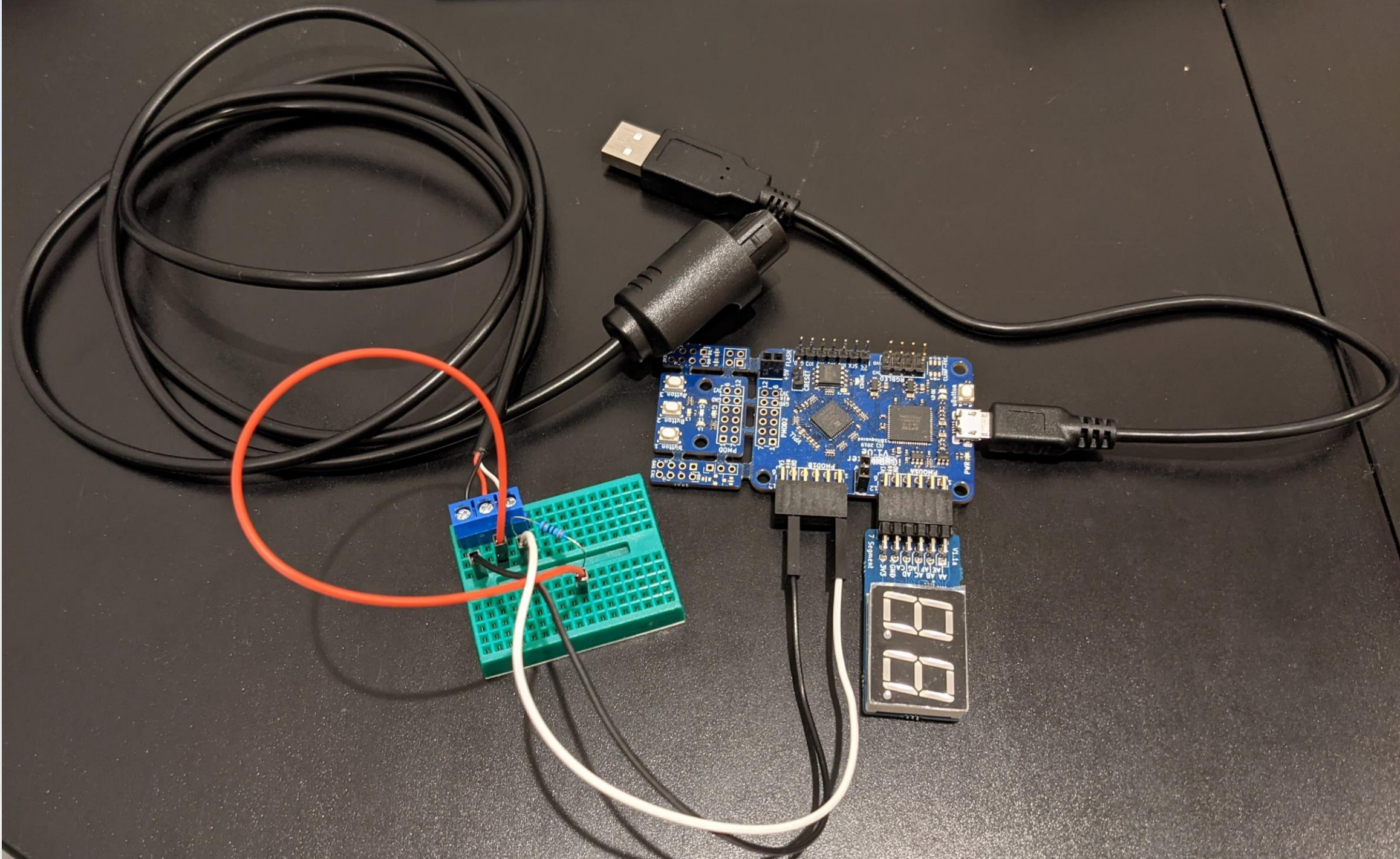
- Difficulty finding relevant code to reverse
- Without working flow based disassembly, very little code is identified, almost no cross-references to data
- Code moves around soon after the game boots
- Probably overlays; making it very difficult to interpret stripped assembly
 - Found some references to overlay reversing by LuigiBlood later on, and overlay docs in the N64 SDK
- Hard to build any context up to understand what's happening
- ROM is also full of all the textures, models, music, etc. assets

Dynamic Analysis

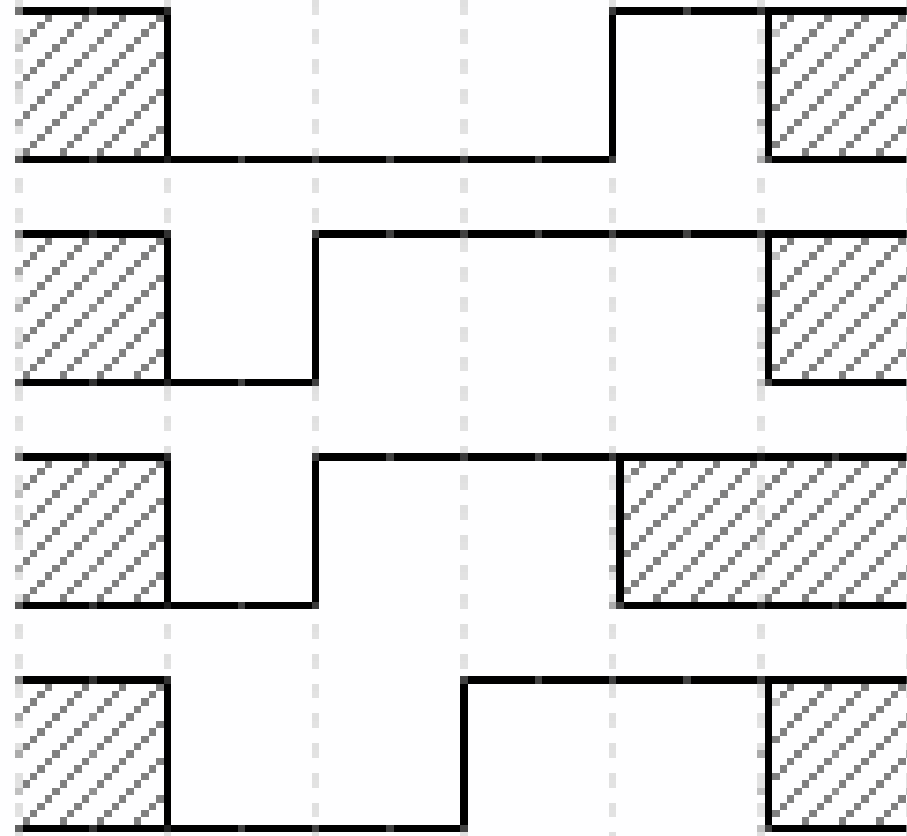
- Instead of painstaking work to get a nice disassembly I switch to dynamic analysis in emulator or with console
- Issues with emulation:
 - Photography feature (detecting Pokémon) must be directly linked to rendering, only worked with low-level graphics emulation plugin (slower)
 - Discovered later that the expansion pak (adds 4 MB to console's main RAM) is necessary for the print display, it just crashes without that
 - Breakpoints on PIF RAM/controller state in the emulator trigger too often
- Issue with console:
 - No real kiosk hardware to use for analysis...
 - But I do have the retail game and N64



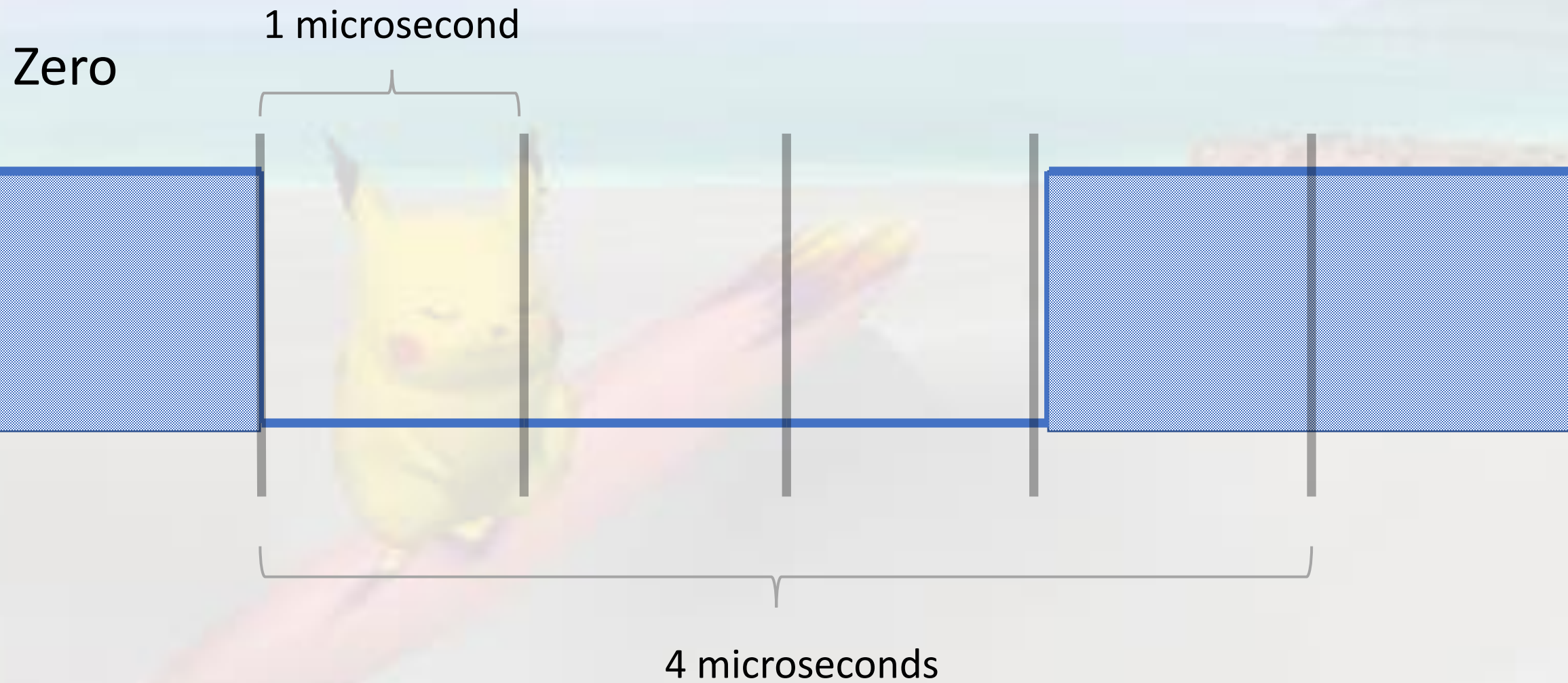
Interfacing with FPGA

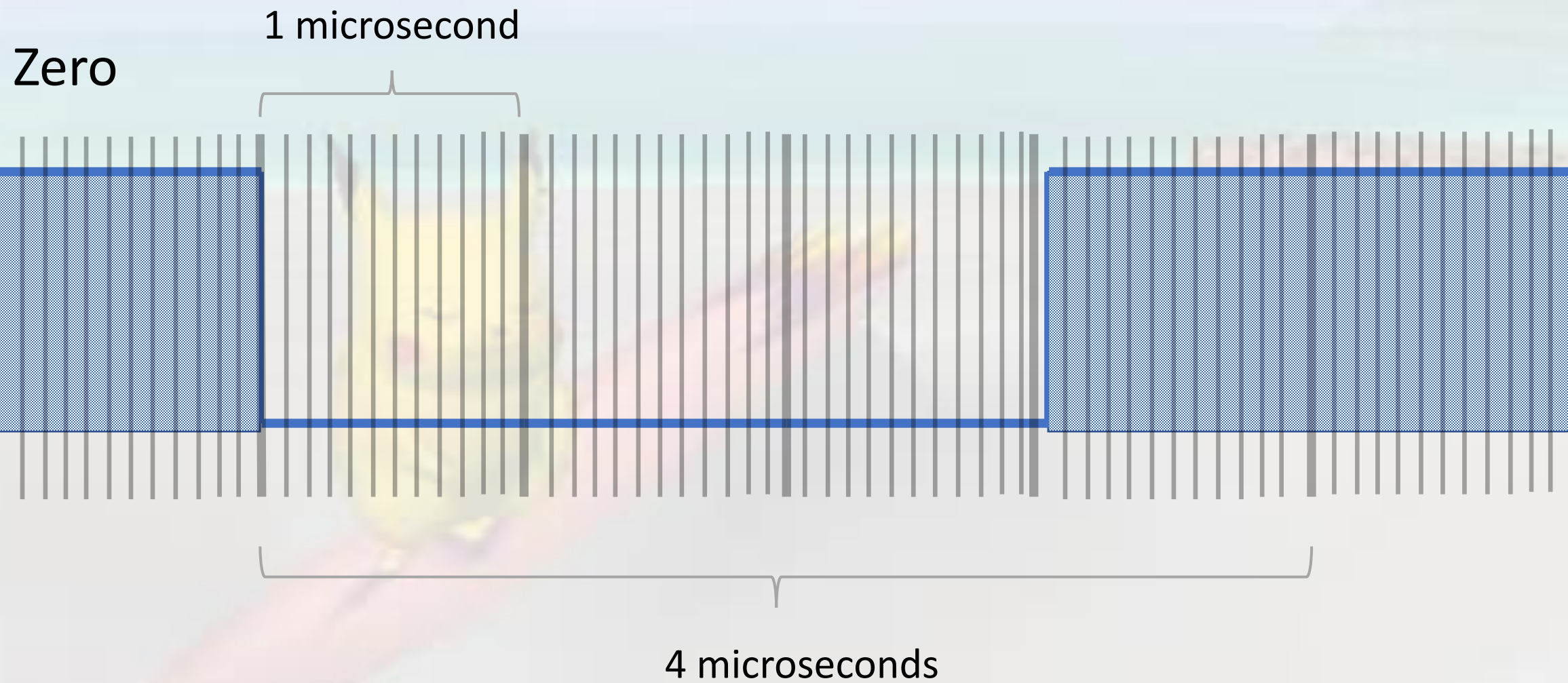


Zero
One
Console Stop Bit
Controller Stop Bit



<https://sites.google.com/site/consoleprotocols/home/nintendo-joy-bus-documentation>





Basic controller functionality

- Console commands consist of a one byte command ID followed by optional data bytes, such as a 16-bit read address for the Controller Pak read command
- Started by implementing the two basic commands needed to simulate a regular N64 controller:
- Device type and status query: FF or 00
- Check current state of buttons and analog stick: 01



<https://youtu.be/LN5hrpZ2cGI>

Sniffing the Joy Bus

- Forward request data to laptop over UART
- Look at hex dump of the requests
- When changing the controller status response to report that an accessory is plugged into the controller...

```
03_8001_fefefefefefefefefefe...fefefefefe
```

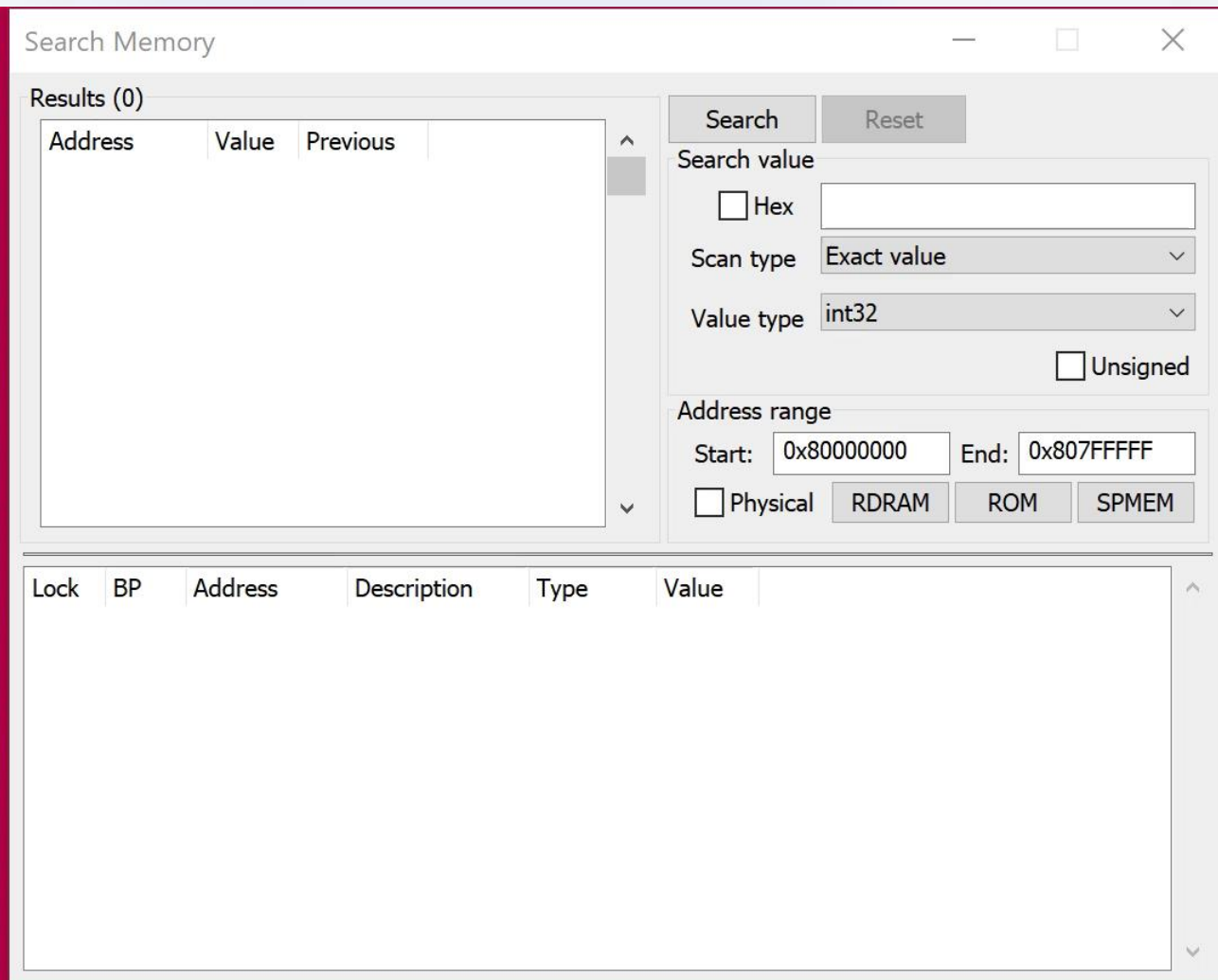
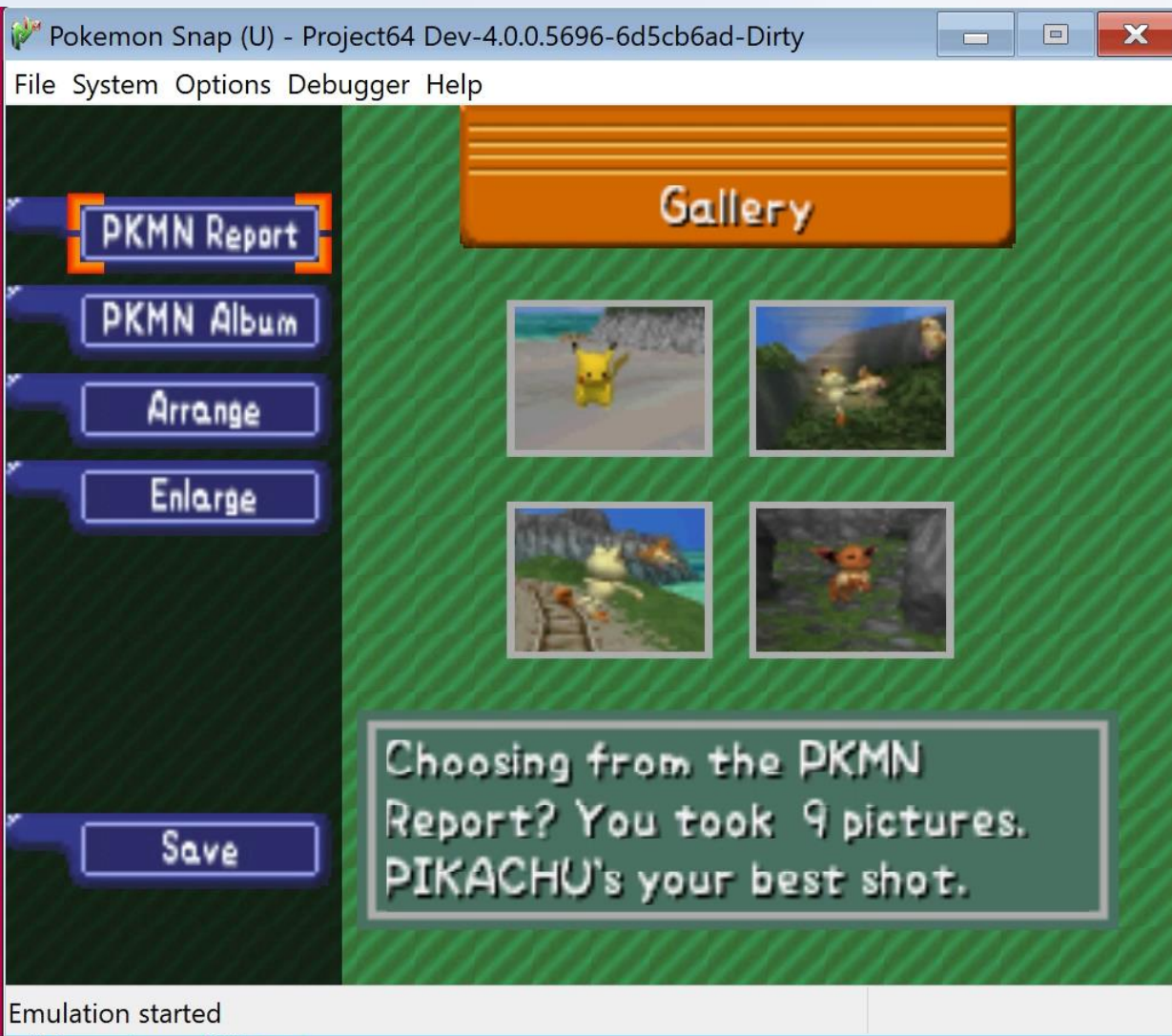
“Write **fefefefe**...to address 0x8000”



Emulator Dynamic Analysis

Dynamic Analysis

- Trying read/write breakpoints on the controller state was too spammy
- Quick & dirty solution: Using Cheat Engine style dynamic analysis of the menu to find the conditional code
- Assume Print code is present in the Gallery menu, just disabled...
- Looked at button/UI handling in the Gallery





Emulation started

Ghidra 9.2 mGBA.exe - Shortcut

Ghidra 10.1.2 snap test 9.2 2_2022_03_...

Commands

Add...	Co...	Parameters	Symbol
801...	LW	A1, 0x000...	
801...	AD...	AT, A1, R0	
801...	SLL	A1, AT, 2	
801...	SUBU	A1, A1, AT	
801...	SLL	A1, A1, 3	
801...	JAL	0x8036FFE0	
801...	AD...	A1, A1, 0x...	
801...	B	0x801DFA84	
801...	OR	V0, R0, R0	
801...	B	0x801DFA84	
801...	NOP		
801...	LW	RA, 0x001...	
801...	AD...	SP, SP, 0x18	
801...	JR	RA	
801...	NOP		
801...	AD...	SP, SP, -0...	
801...	SW	RA, 0x001...	
801...	SW	A0, 0x002...	
801...	SW	A1, 0x002...	
801...	SW	A2, 0x003...	
801...	SW	A3, 0x003...	
801...	SW	S0, 0x001...	
801...	LW	T6, 0x002...	
801...	BNEZ	T6, 0x801...	
801...	NOP		
801...	AD...	T7, R0, 0x...	
801...	LUI	AT, 0x801F	
801...	SW	T7, 0xA29...	
801...	AD...	T8, R0, 0x...	
801...	LUI	AT, 0x801F	
801...	SW	T8, 0xA2A...	
801...	B	0x801DFE60	
801...	OR	V0, R0, R0	
801...	LW	T9, 0x002...	
801...	LUI	AT, 0x0002	
801...	LW	T0, 0x001...	
801...	AND	T1, T0, AT	
801...	BEQZ	T1, 0x801...	
801...	NOP		

Search Memory

Results (1)

Address Value Previous

Search

Reset

801DFA58

...

Skip

Step

St.Ovr

Go

PC

801DFA58

View

R 802308A4

+

-

x

MI	VI	AI	PI	RI	SI	DD
GPR	FPR	COP0	RDRAM	SP	DPC	

CPU General Purpose Registers

R0	00000000	00000000	S0	FFFFFFFF	00000000
AT	00000000	00020000	S1	00000000	00000000
V0	00000000	800BEDF8	S2	00000000	00000000
V1	00000000	000000E2	S3	00000000	00000000
A0	FFFFFFFF	00000016	S4	00000000	00000000
A1	00000000	802308A4	S5	00000000	00000000
A2	00000000	00000000	S6	00000000	00000000
A3	00000000	0000A3E6	S7	00000000	00000000
T0	00000000	802308A4	T8	00000000	00000000
T1	00000000	00000000	T9	00000000	00000000
T2	00000000	00000000	K0	00000000	A430000C
T3	00000000	00000037	K1	00000000	0000AAA
T4	00000000	800BEDF8	GP	00000000	00000000
T5	00000000	00000000	SP	FFFFFFFF	80236648
T6	00000000	800BEDF8	FP	00000000	00000000
T7	00000000	800BEDF8	RA	00000000	801E0380
HI	00000000	00000000	LO	00000000	00000000

Copy Tab Registers

Copy All Registers

Dynamic Analysis

- After identifying a bit of code that way, can leverage the static analysis in a disassembler again
- Searched byte string of a chunk of the currently active code to find it in disassembly
- Repeated that process many times to identify individual functions

Dynamic Analysis

- Look at how button handler code maps button ID to an action
- Examine conditional stuff around here, where menu text is loaded from to try to get the "Print" string
- Finally found suspect global variable checked for a simple constant value to enable/disable certain menu entry, for the Print button
- Look into how that global flag is set (used write breakpoint)
- Found the 0x85 sequence...
 - Didn't know it yet, but this is how the console checks for a peripheral plugged into the controller like a Rumble Pak or Transfer Pak

PKMN Report

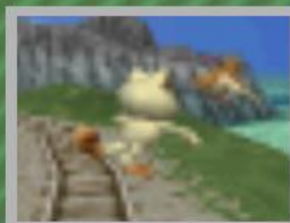
PKMN Album

Arrange

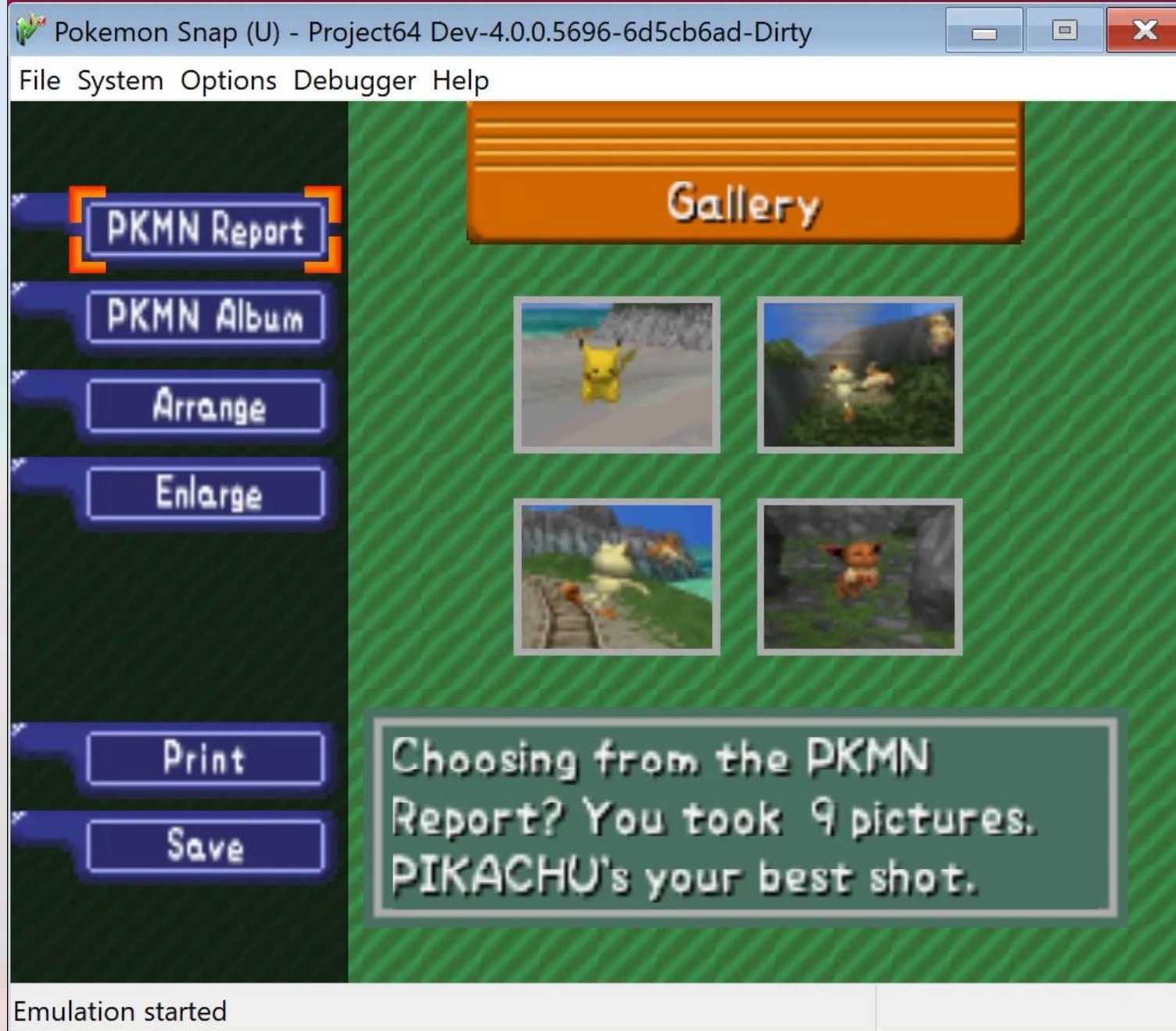
Enlarge

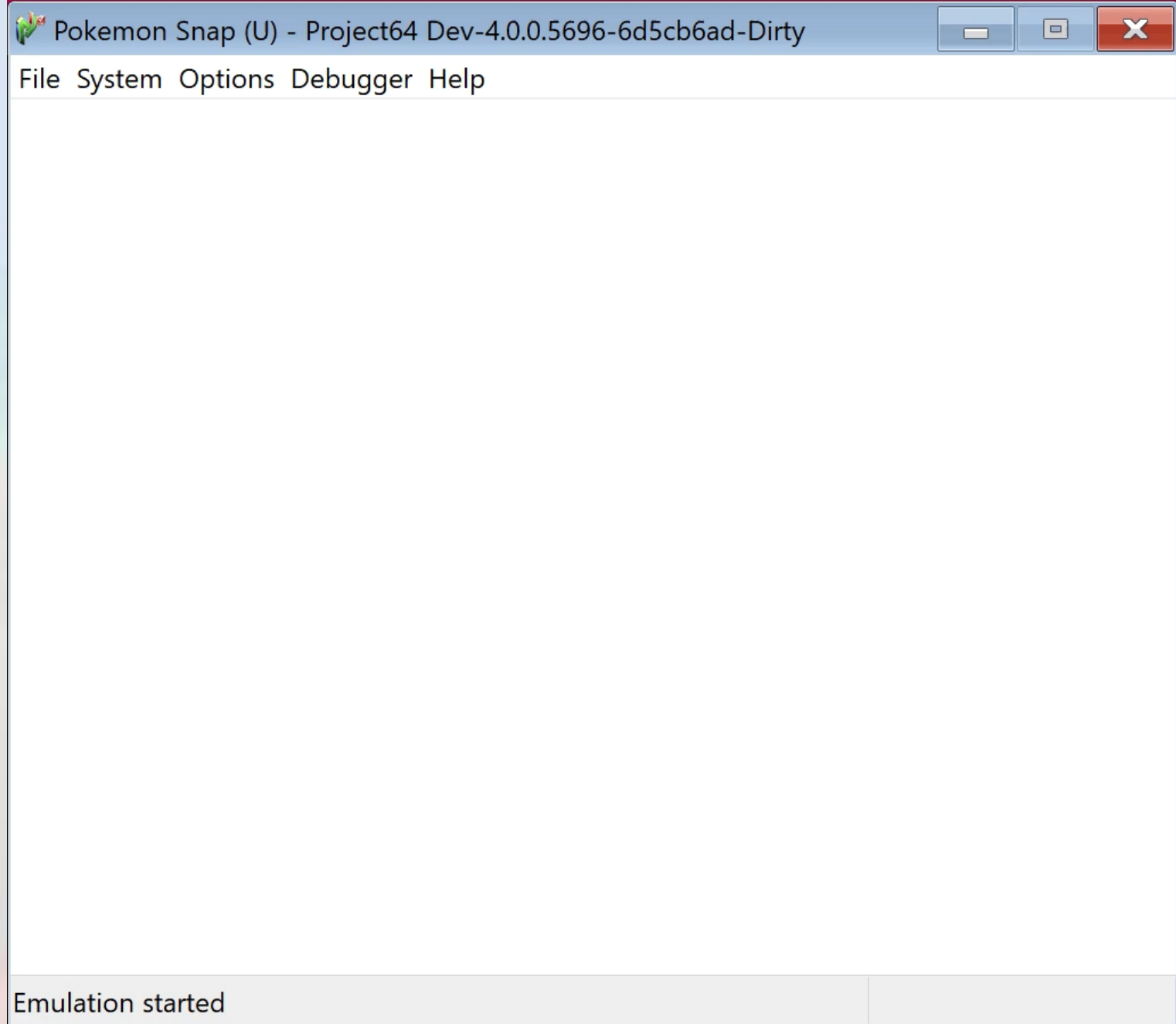
Save

Gallery



Choosing from the PKMN
Report? You took 9 pictures.
PIKACHU's your best shot.







Sniffing the Joy Bus 2

- Return “85” sequence (peripheral ID) to console when it tries to identify what’s plugged in to the controller
- When print is enabled, pressing the “Print” button or booting the game causes some requests to be sent to the peripheral
- Logged traffic with UART again

```
write cmd: 8000 (addr CRC-5 01)
  fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  response: e1
read cmd: 8000 (addr CRC-5 01)
  response: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
           00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
           00
write cmd: 8000 (addr CRC-5 01)
  85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85
  85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85
  response: f5
read cmd: 8000 (addr CRC-5 01)
  response: 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85
           85 85 85 85 85 85 85 85 85 85 85 85 85 85 85
           f5
```

Snap Station Protocol: Gallery menu

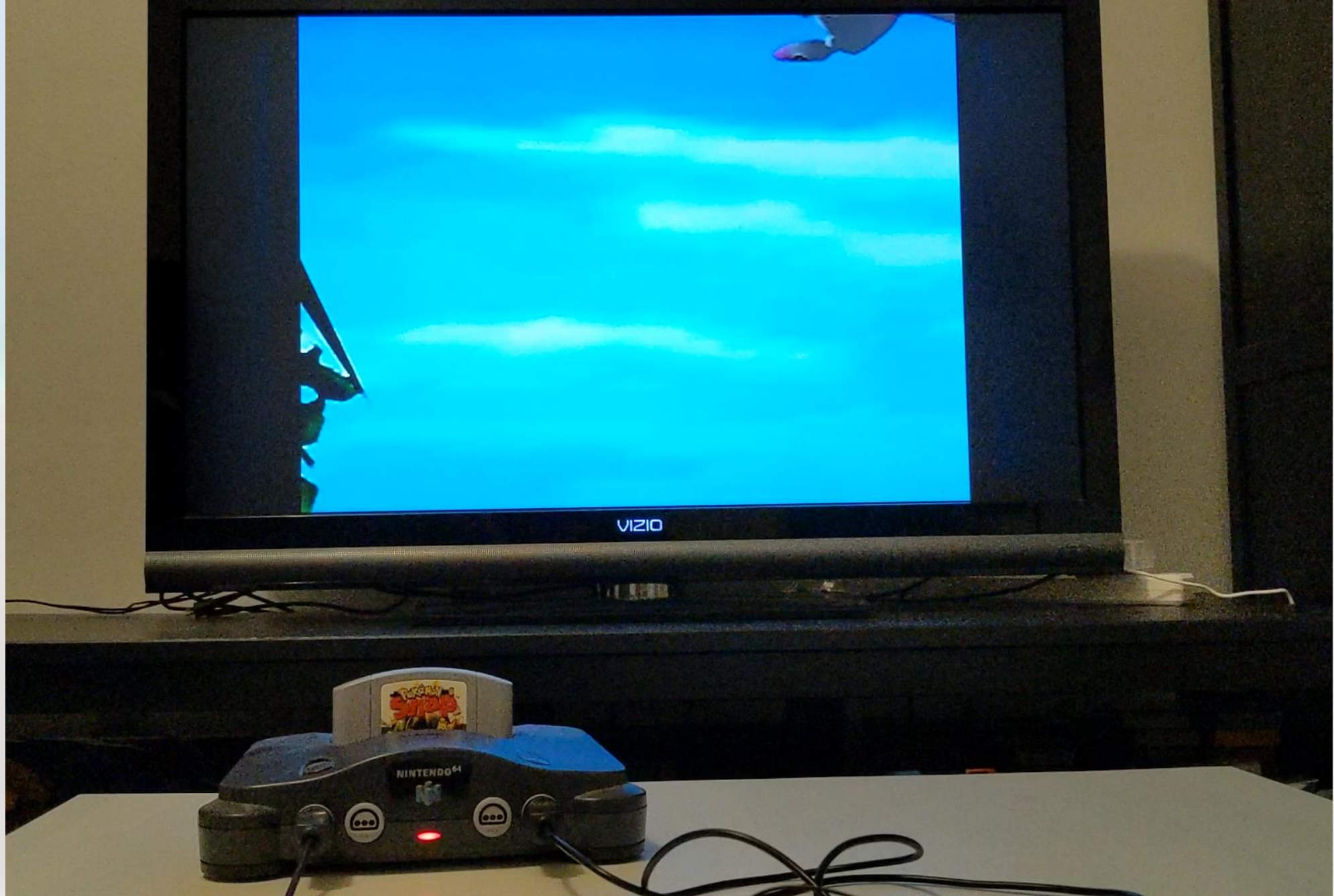
- Read from 0xC000 (just returned 00s...)
- Write to 0xC000: 00 00 ... 00 CC
- Read back 0xC000
- Write to 0xC000: 00 00 ... 00 33
- Read back 0xC000
- Write to 0xC000: 00 00 ... 00 5A
- Read back 0xC000
- CC/33 surround the saving process, 5A means ready to reset



Error message if save fails between CC/33

Snap Station Protocol: Photo display

- Write 01 to 0xC000 and read back
- Write 02 to 0xC000 and read back (16 times)
- Write 04 to 0xC000 and read back
- Note 16 photos are displayed...
- 01 – start
- 02 – displaying photo
- 04 – end
- Responding with 08 at any point triggers busy loop for syncing



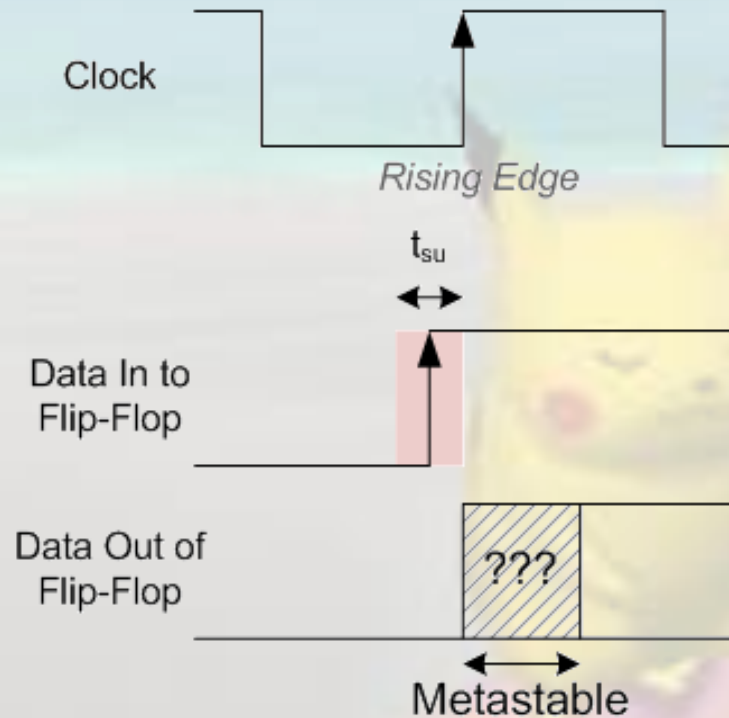
<https://youtu.be/krxyoXlhFw8>



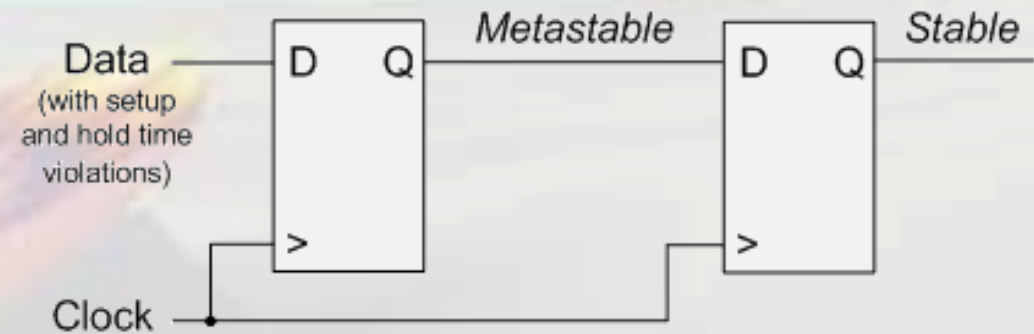
A yellow Pikachu is riding a pink surfboard on a beach. The background shows a blue ocean, a sandy beach, and a green hill under a blue sky with white clouds. The text "FPGA / Hardware Tips" is overlaid on the image.

FPGA / Hardware Tips

Tips: Metastability



Input changed during setup time (t_{su}),
output is metastable



Chain flip flops: second output is stable

<https://www.nandland.com/articles/metastability-in-an-fpga.html>

Tips: Memory inference

- Big difference between vector of 280 bits (`reg [279:0] rx_bits`) and 8-bit array (`reg [7:0] rx_bytes [0:34]`)
 - Quickly run out of LUTs...
- Block RAM: dedicated RAM components, UP5K has 30 blocks of 512 bytes
 - <https://projectf.io/posts/fpga-memory-types/>
- UP5K has SPRAM blocks with 32 KiB capacity, exactly the amount of storage needed for N64 memory card
 - Need to manually specify it with Yosys
 - <https://projectf.io/posts/spram-ice40-fpga/>
 - Using the SPI flash for persistent storage would also be nice

Tips: Open drain / open collector

- Signal on the bus by going to ground for 0, high impedance mode for 1 (pull-up resistor will pull signal high)
- Set output register to **z** to enter high impedance state

```
// Console data IO line (open drain output)
// setup is a little weird because trying to assign Z for 1 later didn't work
// We pull to ground for a 0, release the line for 1 and it should return to 3.3V
wire      console_input;
assign P1B1 = (tx_enabled && tx_output_bit == 0) ? 1'b 0 : 1'b z;
assign console_input = P1B1;
```

A yellow Pikachu is riding a pink surfboard on a sandy beach. The background shows a blue ocean and a blue sky with a few clouds. The text "Driving peripherals" is overlaid on the image.

Driving peripherals




```
$ ./uart_host.py /dev/ttyUSB1 --dump-tpak-ram pokemon_blue.sav
```

```
Using port: /dev/ttyUSB1
```

Pad type: 0005, joyport status: 01

```
transfer pak present: True
```

ROM header:

```
00000000: 00 C3 50 01 CE ED 66 66  CC 0D 00 0B 03 73 00 83  ..P...ff....s..
```

```
00000010: 00 0C 00 0D 00 08 11 1F 88 89 00 0E DC CC 6E E6 .....n.
```

```
00000020: DD DD D9 99 BB BB 67 63 6E 0E EC CC DD DC 99 9F .....gcn.....
```

```
00000030: BB B9 33 3E 50 4F 4B 45 4D 4F 4E 20 42 4C 55 45  ..3>POKEMON BLUE
```

```
00000040: 00 00 00 00 30 31 03 13 05 03 01 33 00 D3 9D 0A ....01.....3....
```

```
Raw title: b'POKEMON BLUE'
```

MBC type: MBC3

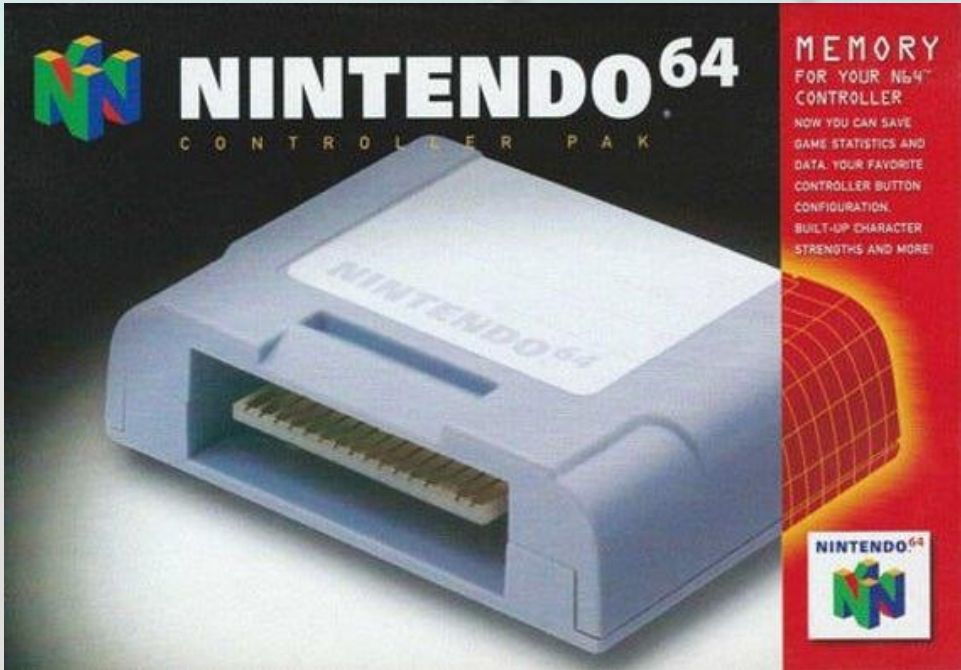
ROM size: 0x100000 bytes

```
RAM size: 0x8000 bytes
```

Dumping 4 RAM banks to pokemon_blue.sav...

100% | ██████████ | 32768/32768 [00:16<00:00, 1968.87it/s]

Controller Pak



```
$ ./uart_host.py /dev/ttyUSB1 --dump-cpak cpak_gray.bin
Using port: /dev/ttyUSB1
Pad type: 0005, joyport status: 01
dump controller pak to cpak_gray.bin...
100%|██████████████████████████████████████████| 1024/1024 [00:16<00:00, 62.55it/s]
$ hexdump cpak_gray.bin | head
000000 81 fe fd fc fb fa f9 f8 00 fe fd fc 08 08 08 08 >.....<
000010 ef ee ed ec 00 00 00 15 10 ee ed ec f5 00 00 f4 >.....<
000020 ff ff ff ff 04 bc 62 75 05 4c 46 f2 07 87 27 07 >.....bu.LF...'.<
000030 00 00 00 00 4f 4b 4b 0a 00 01 01 00 7d 51 82 a1 >....OKK.....}Q..<
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 >.....<
*
000060 ff ff ff ff 04 bc 62 75 05 4c 46 f2 07 87 27 07 >.....bu.LF...'.<
000070 00 00 00 00 4f 4b 4b 0a 00 01 01 00 7d 51 82 a1 >....OKK.....}Q..<
000080 ff ff ff ff 04 bc 62 75 05 4c 46 f2 07 87 27 07 >.....bu.LF...'.<
000090 00 00 00 00 4f 4b 4b 0a 00 01 01 00 7d 51 82 a1 >....OKK.....}Q..<
```

iCEBreaker design: <https://github.com/jamchamb/cojiro>

Project64 mod: <https://github.com/jamchamb/project64/tree/snapstation>

Blog post: <https://jamchamb.net/2021/08/17/snap-station.html>



SEE YOU SHMOOCON ...