Hardware fault injection attacks for everyone

Voltage glitching workshop @ Fri3d Camp 2022

PoroCYon

PDF & code: `https://pcy.be/nl22`

TOC

Whoami

- Demoscener and hardware hacker
- Dumped DSi ARM7 boot ROM and Wii Fit U Meter flash using glitching
- Linux demoscene 4k intro tooling, ...

DSi boot ROM dump

- EMFI attack leveraging design issue
- Presentation at Newline 2021[1]
- Well received

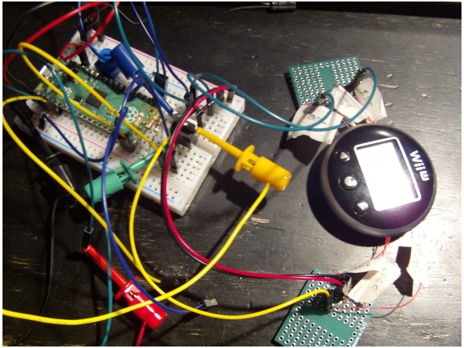
[1] https://events.hackerspace.gent/en/newline2021/public/events/72

DSi boot ROM dump

- EMFI attack leveraging design issue
- Presentation at Newline 2021[1]
- Well received
- "Nobody else can do this"

[1] https://events.hackerspace.gent/en/newline2021/public/events/72

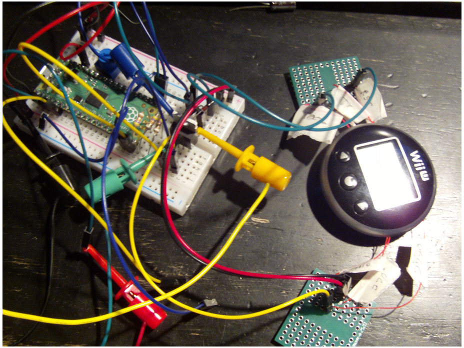
Wii Fit U Meter



- Similar to Pokéwalker, but different MCU
- No IR exploit known (↔ Pokéwalker)
- VFI attack inspired by fail0verflow on the PS4 Syscon[2]

[2] https://fail0verflow.com/blog/2018/ps4-syscon/

Wii Fit U Meter



- Similar to Pokéwalker, but different MCU
- No IR exploit known (↔ Pokéwalker)
- VFI attack inspired by fail0verflow on the PS4 Syscon[2]
- Very easy to pull off!

[2] https://fail0verflow.com/blog/2018/ps4-syscon/

Materials

Participation fee: €8,5

for extra components (€170 total)

You should have:

- Camp badge
- USB-C cable
- Laptop
- ESP-IDF installation
- **Firmware: get at** `https://pcy.be/fc22`

On your table:

- Breadboard, wires, micro:bit breakout connector (leave here)
- RL78 target on PCB (bring home)
- MOSFET (bring home)
- Potentiometer (bring home)
- Diode (bring home)

(need help with custom firmware? Ask me!)

Concept

"ICs need to be operated under specified conditions, eg. within the rated supply voltage, clock stability, temperature, and electromagnetic field ranges. This dependency can be misused to force faulty behavior during the chip's operation."[1]

[1] https://arxiv.org/pdf/2108.06131.pdf

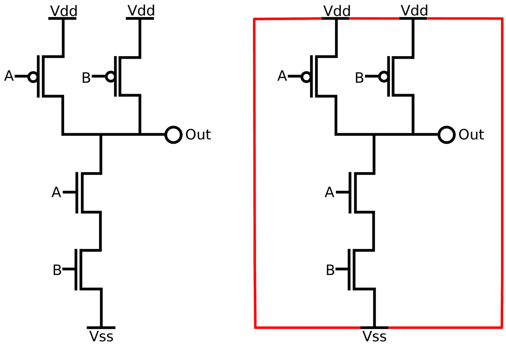
Concept

"ICs need to be operated under specified conditions, eg. within the rated supply voltage, clock stability, temperature, and electromagnetic field ranges. This dependency can be misused to force faulty behavior during the chip's operation."[1]

Glitching (colloquial) = "Fault Injection" (academic)

[1] https://arxiv.org/pdf/2108.06131.pdf

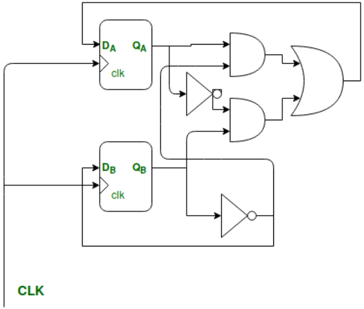
Voltage Fault Injection



Source: Wikimedia

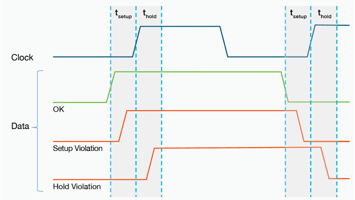
- Easy to pull off, can be very cheap
- Very common ⇒ most countermeasures

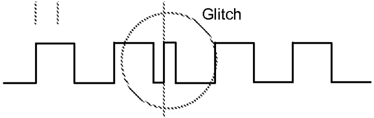
Clock glitching



Source: GeeksforGeeks

Source: Dave Pereles at DesignNews

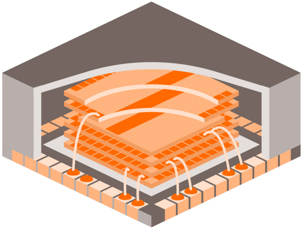




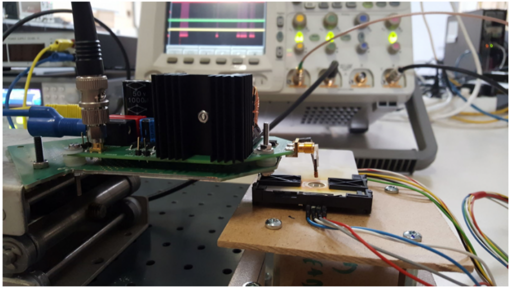
Source: US patent 08319524

- Easy to pull off
- Needs direct clock input (not always available)

Electromagnetic Fault Injection



Source: Applus

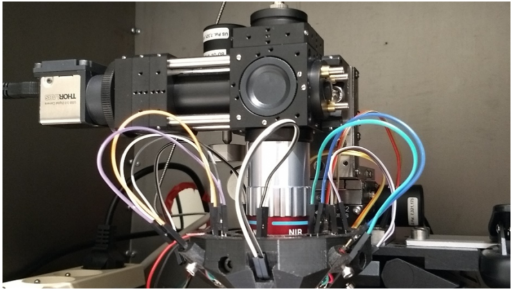


Source: COSIC

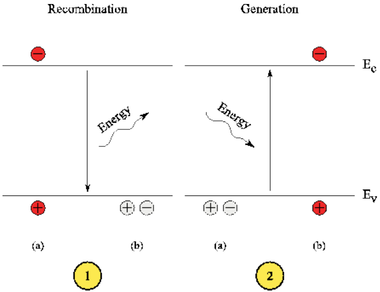
- Harder to pull off (but: PicoEMP[2])
- Larger parameter search space

[2] https://github.com/newaetech/chipshouter-picoemp

Laser Fault Injection



Source: COSIC



Source: TU Wien

- Needs specialized equipment, very large search space
- Needs chip decapsulation

"Fault model"

"Fault model" describes what could go wrong

- Skip instruction
- Skip register/memory write
- Clear/set/toggle bits in accessed data
- Faulty instruction decoding
- Perturb result of cryptographic operation → *differential fault analysis* (DFA)
- ...

Use these to create possible bugs

“Parameter search space”

Effect of glitch depends on many things:

- Moment of glitch
- Glitch length/power/...
- Abnormal voltage (some VFI)
- Glitch location (EMFI/LFI)
- Environment temperature, ...

⇒ Need to find the right combination for the desired effect!

Sidenote: “Side Channel Analysis”

“Inverse” idea of fault injection:
Perturb operating conditions to cause bad behavior
$\updownarrow$
Monitor effects to the environment closely to learn about the system

- Time it takes to perform an operation
- Fluctuations in power usage
- Small EM emissions
- ...

⇒ Need very precise measurement system..

How to attack a target

1. Acquire target
2. ???
3. ???
4. ???
5. ???
6. Hack it
7. ???
8. Profit

How to attack a target

1. Acquire target
2. Define your goal
3. Read as much info about it as possible
4. Find exploit
5. Find glitching method
6. Test glitching method (search ‘parameter space’)
7. Perform exploit
8. Profit

RL78?

- 8/16-bit microcontroller
- Descendant of old 78K and 78K0R lines
- Used in small automotive stuff, household applicances, ...
- Not very popular

RL78 info

- Instruction set[3]: Z80, but worse
- Hardware manual[4]: flash & RAM size, pinout, I/O registers
- *Serial Flash Programming* manual[5]: protocol to access flash contents from outside

Goal: Can we read out the flash memory contents?

[3] https://www.renesas.com/eu/en/document/mah/rl78-family-users-manual-software-rev230?r=1054286

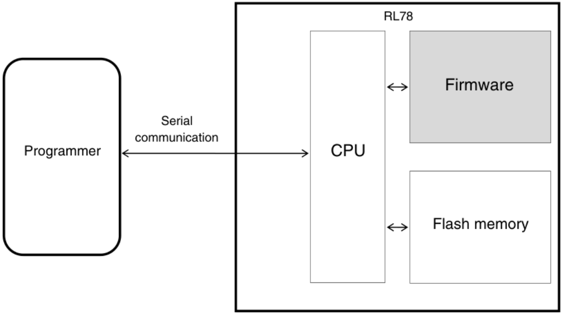
[4] https://www.renesas.com/eu/en/document/man/rl78g13-users-manual-hardware?r=1054286

[5] https://www.renesas.com/eu/en/document/apn/rl78-microcontrollers-rl78-protocol-programmer-edition-application-note-rev100?r=1054286

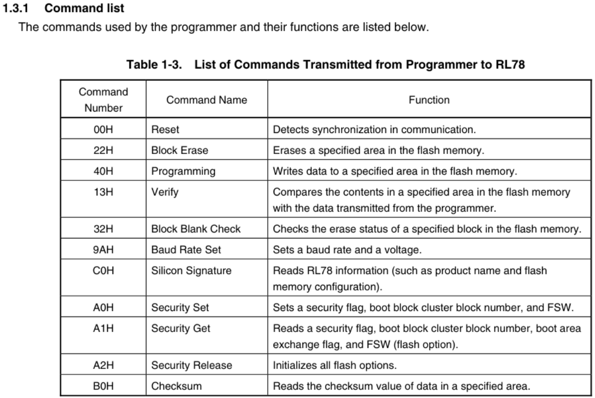
“Serial flash protocol”?

Interesting...

Figure 1-1. System Outline of Flash Memory Programming in RL78



Serial flash protocol!



1.3.1 Command list

The commands used by the programmer and their functions are listed below.

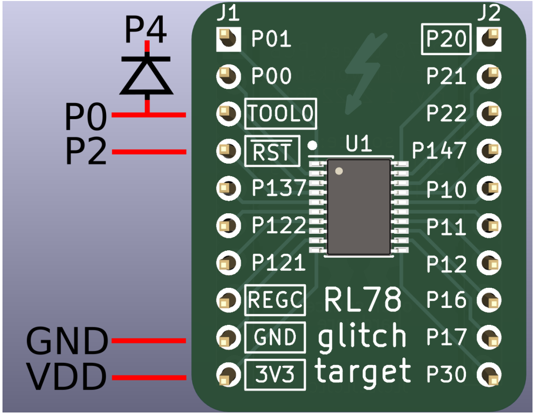
Table 1-3. List of Commands Transmitted from Programmer to RL78

Command Number	Command Name	Function
00H	Reset	Detects synchronization in communication.
22H	Block Erase	Erases a specified area in the flash memory.
40H	Programming	Writes data to a specified area in the flash memory.
13H	Verify	Compares the contents in a specified area in the flash memory with the data transmitted from the programmer.
32H	Block Blank Check	Checks the erase status of a specified block in the flash memory.
9AH	Baud Rate Set	Sets a baud rate and a voltage.
C0H	Silicon Signature	Reads RL78 information (such as product name and flash memory configuration).
A0H	Security Set	Sets a security flag, boot block cluster block number, and FSW.
A1H	Security Get	Reads a security flag, boot block cluster block number, boot area exchange flag, and FSW (flash option).
A2H	Security Release	Initializes all flash options.
B0H	Checksum	Reads the checksum value of data in a specified area.

No read command?

Can still be used to check if hardware is alive!

Hardware setup: connect pins



Hardware setup: load firmware

Web:

1. If Windows: install `https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers`
2. Go to `https://gitlab.ulyssis.org/pcy/nl22-vfi/tree/main/bin`
3. Get `esp32.zip` from git repo
4. Upload zip to badge via `https://fri3d-flasher.vercel.app/`
5. Use PuTTY or Arduino IDE for serial port

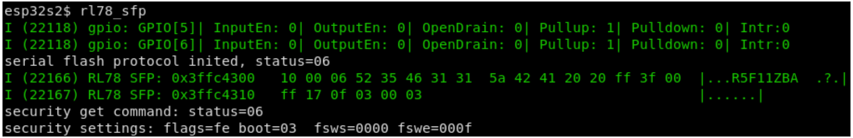
Terminal (with ESP-IDF):

1. `git clone https://gitlab.ulyssis.org/pcy/nl22-vfi`
2. `cd bin/esp32`
3. `source ~/.espressif/export.sh`
4. `./flash.sh /dev/ttyUSB0`
5. `idf.py monitor` or `picocom -b 115200 --imap lfcrlf /dev/ttyUSB0 115200`

Or, ask me for help.

Hardware setup: test run

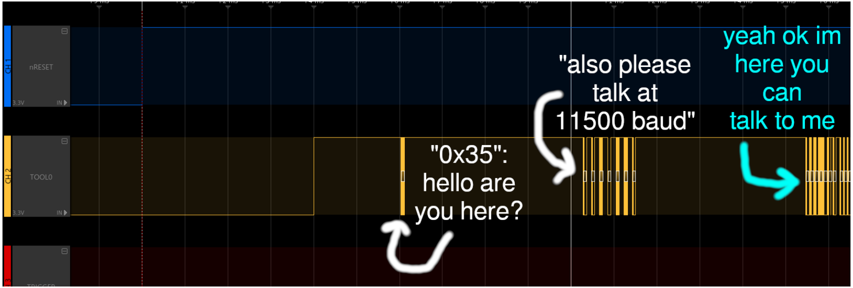
`rl78_sfp`



```
esp32s2$ rl78_sfp
I (22118) gpio: GPIO[5]| InputEn: 0| OutputEn: 0| OpenDrain: 0| Pullup: 1| Pulldown: 0| Intr:0
I (22118) gpio: GPIO[6]| InputEn: 0| OutputEn: 0| OpenDrain: 0| Pullup: 1| Pulldown: 0| Intr:0
serial flash protocol inited, status=06
I (22166) RL78 SFP: 0x3ffc4300   10 00 06 52 35 46 31 31  5a 42 41 20 20 ff 3f 00  |...R5F11ZBA  .?.|
I (22167) RL78 SFP: 0x3ffc4310   ff 17 0f 03 00 03                                 |......|
security get command: status=06
security settings: flags=fe boot=03  fsws=0000 fswe=000f
```

Alt. serial port: `https://console.zacharyschneider.ca/`

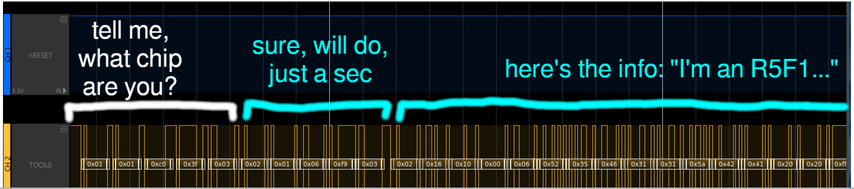
Test run: what does it do?



"0x35": hello are you here?

"also please talk at 11500 baud"

yeah ok im here you can talk to me



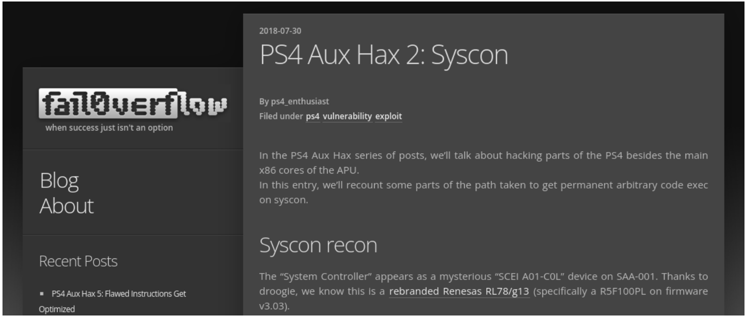
tell me, what chip are you?

sure, will do, just a sec

here's the info: "I'm an R5F1..."

```
0x01 0x01 0xc0 0x3f 0x03 0x02 0x01 0x06 0xf9 0x03 0x02 0x16 0x10 0x00 0x06 0x52 0x35 0x46 0x31 0x31 0x5a 0x42 0x41 0x20 0x20 0xff
```

Second protocol?!



fail0verflow
when success just isn't an option
Blog
About
Recent Posts
▪ PS4 Aux Hax 5: Flawed Instructions Get Optimized

2018-07-30

PS4 Aux Hax 2: Syscon

By ps4_enthusiast
Filed under ps4 vulnerability exploit

In the PS4 Aux Hax series of posts, we'll talk about hacking parts of the PS4 besides the main x86 cores of the APU.
In this entry, we'll recount some parts of the path taken to get permanent arbitrary code exec on syscon.

Syscon recon

The "System Controller" appears as a mysterious "SCEI A01-C0L" device on SAA-001. Thanks to droogie, we know this is a rebranded Renesas RL78/g13 (specifically a R5F100PL on firmware v3.03).

- Serial Flash Protocol implemented in ROM
- Reverse-engineer ROM: has second protocol: debug
- Allows more access...

On-Chip Debug

Capabilities

- Read from RAM and MMIO
- Write to RAM and MMIO
- Execute code in RAM

Protection

- Password-protected
- Can be disabled by `OCDEN` bit
- ~~Setting to erase flash on wrong password~~ — A lie, doesn't happen

On-Chip Debug

Capabilities

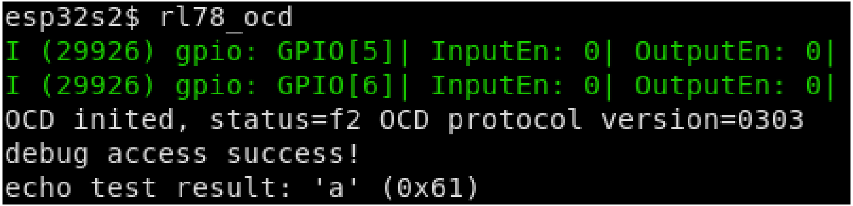
- Read from RAM and MMIO
- Write to RAM and MMIO
- Execute code in RAM
- ⇒ Upload code to dump flash!

Protection

- Password-protected
- Can be disabled by `OCDEN` bit
- ~~Setting to erase flash on wrong password~~ — A lie, doesn't happen
- ⇒ Needs to be circumvented ⇒ Glitching!

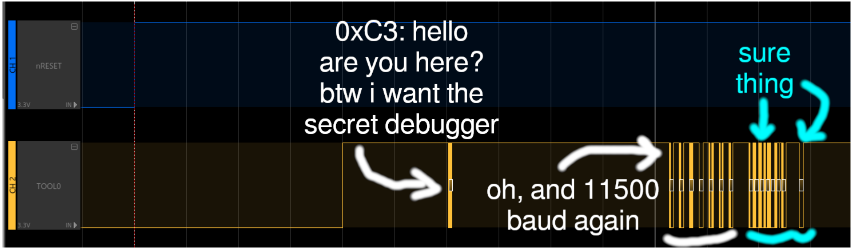
Demonstration

rl78_ocd



```
esp32s2$ rl78_ocd
I (29926) gpio: GPIO[5]| InputEn: 0| OutputEn: 0|
I (29926) gpio: GPIO[6]| InputEn: 0| OutputEn: 0|
OCD inited, status=f2 OCD protocol version=0303
debug access success!
echo test result: 'a' (0x61)
```

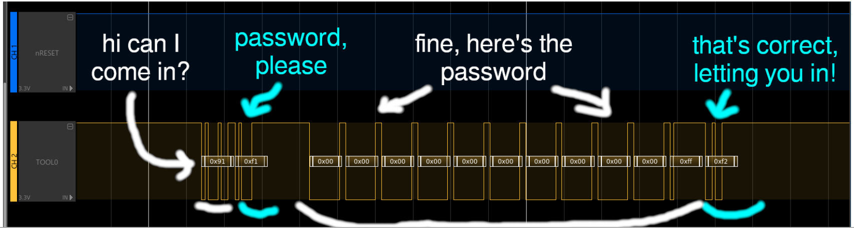
Demonstration: what does it do?



0xC3: hello
are you here?
btw i want the
secret debugger

oh, and 11500
baud again

sure
thing



hi can I
come in?

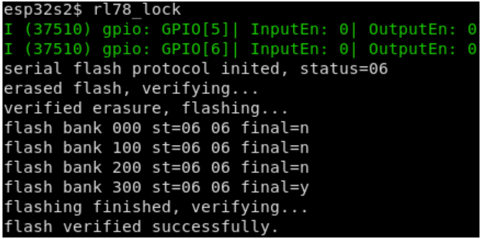
password,
please

fine, here's the
password

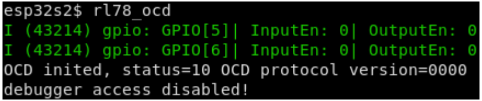
that's correct,
letting you in!

OCD lock?

`rl78_lock`
`rl78_ocd`

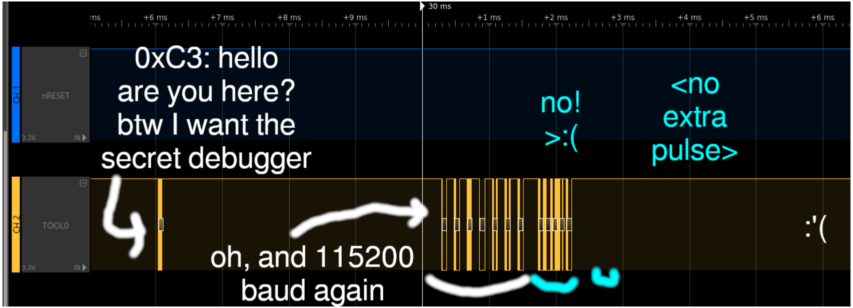


```
esp32s2$ rl78_lock
I (37510) gpio: GPIO[5]| InputEn: 0| OutputEn: 0
I (37510) gpio: GPIO[6]| InputEn: 0| OutputEn: 0
serial flash protocol inited, status=06
erased flash, verifying...
verified erasure, flashing...
flash bank 000 st=06 06 final=n
flash bank 100 st=06 06 final=n
flash bank 200 st=06 06 final=n
flash bank 300 st=06 06 final=y
flashing finished, verifying...
flash verified successfully.
```



```
esp32s2$ rl78_ocd
I (43214) gpio: GPIO[5]| InputEn: 0| OutputEn: 0
I (43214) gpio: GPIO[6]| InputEn: 0| OutputEn: 0
OCD inited, status=10 OCD protocol version=0000
debugger access disabled!
```

Locked!



0xC3: hello are you here? btw I want the secret debugger

oh, and 115200 baud again

no! >:(

<no extra pulse>

:'(

Now do `rl78_unlock` before we continue

Looking for vulnerabilities

RL78 ROM (pseudocode):

```
byte mode = tool_rx();
if (mode == 0x35) {
  do_sfp();
} else if (mode == 0xc3) {
  if (!OCDEN) { // OCD locked?
    // infinite loop
    while (true) ;
  }
  do_ocd();
}

```

Looking for vulnerabilities (2)

```
  // infinite loop in assembly:
  brtrue OCDEN, jump_to_ocd

hang:
  br hang
jump_to_ocd:
  call do_ocd

```

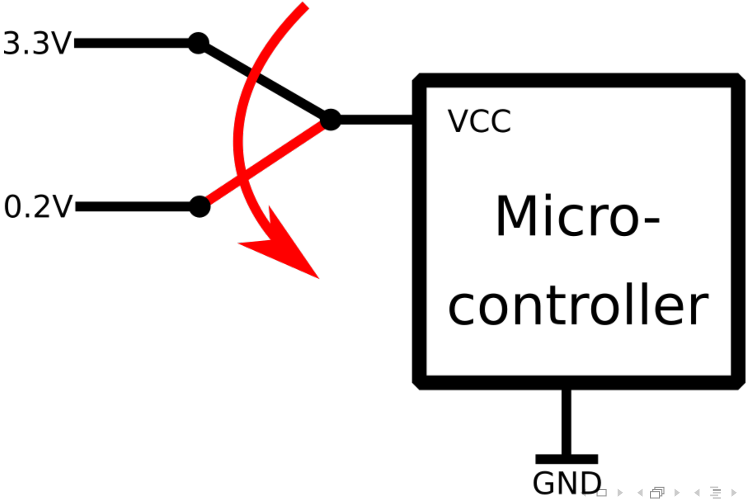
Recall: possible fault model: skip instruction

Target acquired

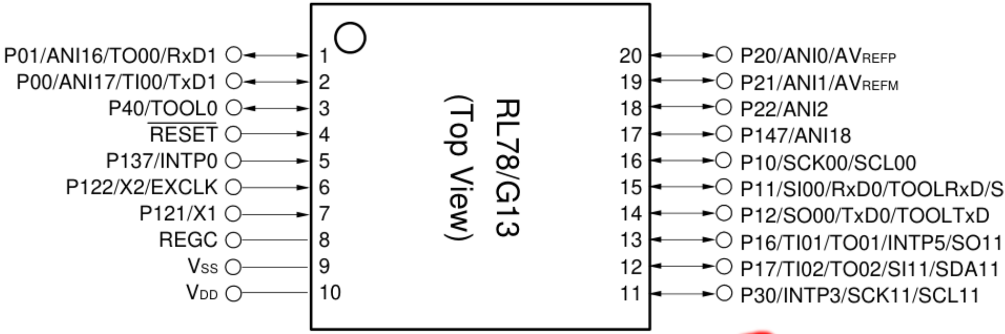
We now know what to glitch!

... how do we glitch it?

Voltage Fault Injection, take 1



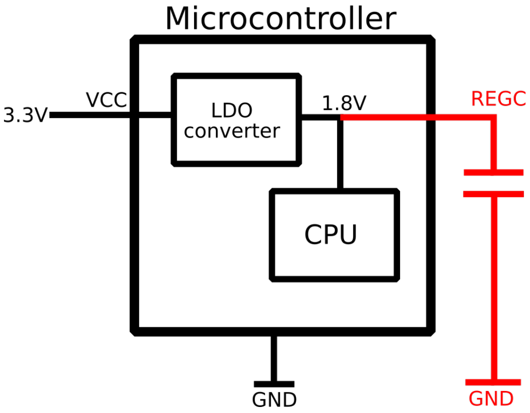
What this pin?



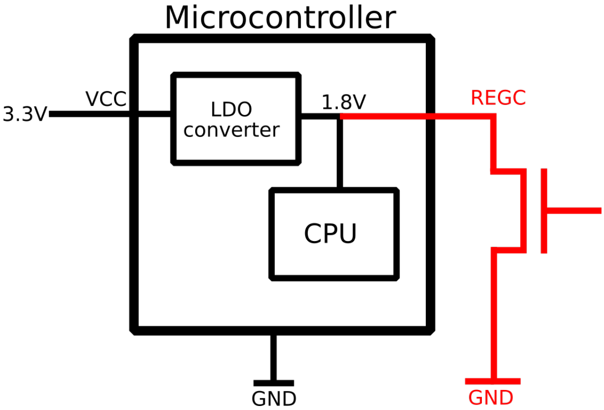
Caution Connect the REGC pin to Vss via a capacitor (0.47 to 1 µF).

?

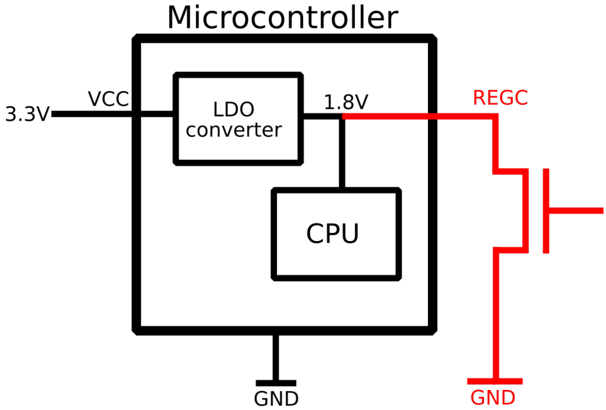
Voltage regulator



Voltage Fault Injection, take 2



Voltage Fault Injection, take 2



This pin gives us direct access to what we want to glitch!

Parameter search

Parameters to optimize:

offset	No
	CPU stuck in infinite loop → not applicable
low voltage	No
X/Y/Z pos.	No
...	No
length	**Yes**!

How to search

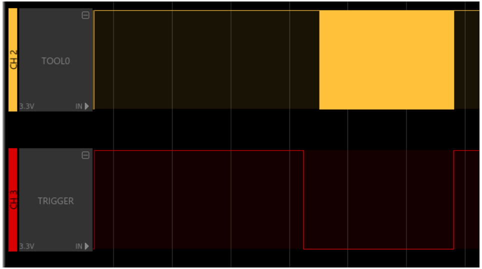
Simple glitch test setup:

1. Upload test code using debug access
2. Code increments variable in a loop, sends it back
3. Try glitching the loop

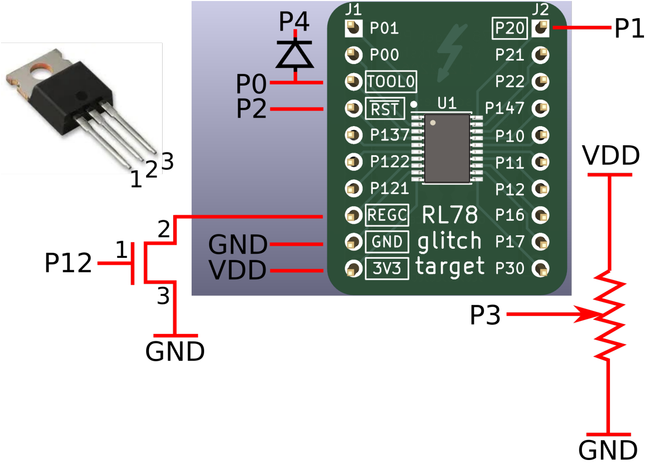
too short : result = `0xff`

just right : result ≠ `0xff`!

too long : chip reset



Assembling the setup



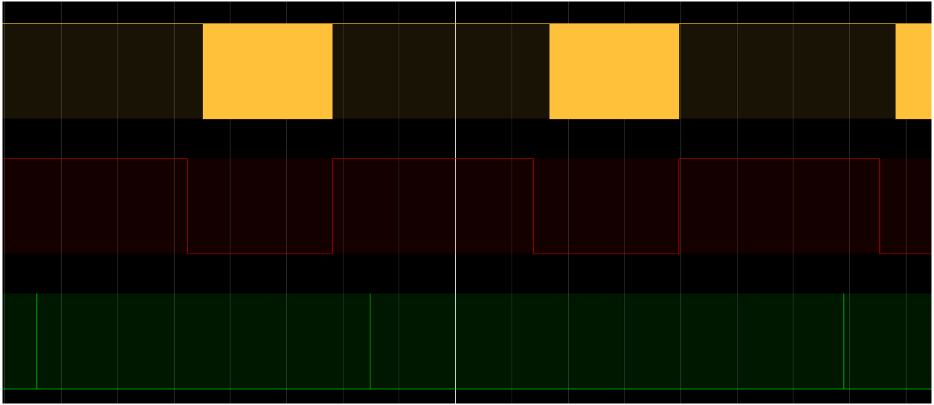
Implementing the parameter search

Flow:

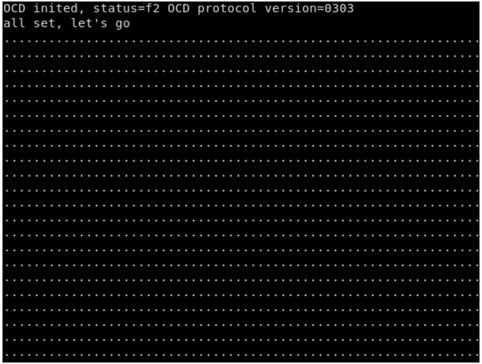
1. Wait for `P20` line to go high
2. Do glitch somewhere while `P20` is high
3. Wait for data on `TOOL0`. No data = chip reset!
4. Check first two bytes, should be `Hi`. Wrong = chip crash!
5. Receive 256 data bytes on `TOOL0`
6. If any byte is wrong, we have a glitch!
7. GOTO 1

Running the setup

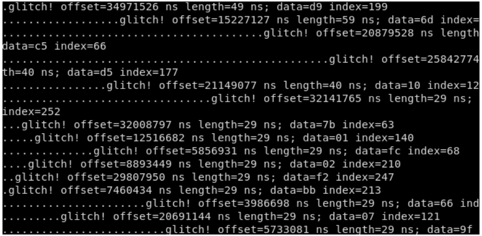
glitch_param



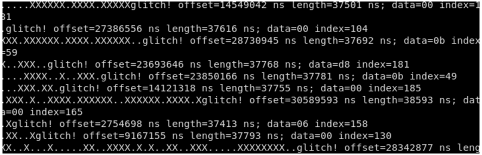
Running the setup



```
OCD inited, status=f2 OCD protocol version=0303
all set, let's go
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
```

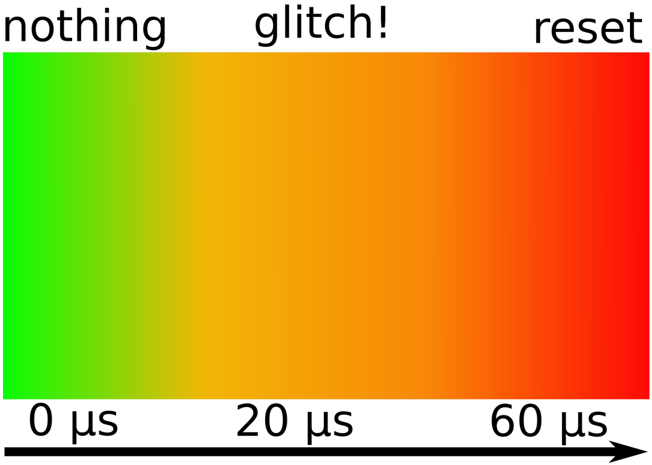


```
.glitch! offset=34971526 ns length=49 ns; data=d9 index=199
..................glitch! offset=15227127 ns length=59 ns; data=6d index=
........................................glitch! offset=20879528 ns length
data=c5 index=66
..................................................glitch! offset=25842774
th=40 ns; data=d5 index=177
................glitch! offset=21149077 ns length=40 ns; data=10 index=12
................................glitch! offset=32141765 ns length=29 ns;
index=252
...glitch! offset=32008797 ns length=29 ns; data=7b index=63
.....glitch! offset=12516682 ns length=29 ns; data=01 index=140
..............glitch! offset=5856931 ns length=29 ns; data=fc index=68
....glitch! offset=8893449 ns length=29 ns; data=02 index=210
..glitch! offset=29807950 ns length=29 ns; data=f2 index=247
.glitch! offset=7460434 ns length=29 ns; data=bb index=213
......................glitch! offset=3986698 ns length=29 ns; data=66 ind
.........glitch! offset=20691144 ns length=29 ns; data=07 index=121
.........................glitch! offset=5733081 ns length=29 ns; data=9f
```



```
.....XXXXXX.XXXX.XXXXXglitch! offset=14549042 ns length=37501 ns; data=00 index=1
31
.glitch! offset=27386556 ns length=37616 ns; data=00 index=104
XXX.XXXXXX.XXXX.XXXXXX..glitch! offset=28730945 ns length=37692 ns; data=0b index
=59
X..XXX..glitch! offset=23693646 ns length=37768 ns; data=d8 index=181
....XXXX..X..XXX.glitch! offset=23850166 ns length=37781 ns; data=0b index=49
...XXX.XX.glitch! offset=14121318 ns length=37755 ns; data=00 index=185
.XXX.X..XXXX.XXXXXX..XXXXXX.XXXX.Xglitch! offset=30589593 ns length=38593 ns; dat
a=00 index=165
.Xglitch! offset=2754698 ns length=37413 ns; data=06 index=158
.XX..Xglitch! offset=9167155 ns length=37793 ns; data=00 index=130
XX..X...X.....XX..XXXX.X.X..XX..XXX.....XXXXXXXX..glitch! offset=28342877 ns leng
```

Evaluating the search



Try finding a sweet spot

Need this in the next part!

Exploit time

`rl78_lock`

No more cheating.

Quick reminder

What we need to do:

1. Ask chip for debug access
2. Chip says `no!`
3. Code in chip now in infinite loop
4. **Glitch here: break out of loop!**
5. Chip now gives us debug access!
6. Ask chip to give up all its secrets

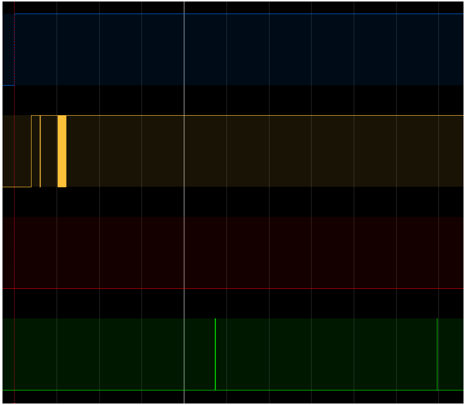
Implementing the glitching

Flow:

1. Ask chip for access, get told “`no`”
2. Do (up to) 16 times:
   2.1 Do a glitch
   2.2 If we now receive a `0x00` byte, we have glitched successfully!
   2.3 Else, wait a bit
3. No response: glitches either did nothing or crashed the chip, we can’t know.
4. In case latter happened, reset chip
5. GOTO 1

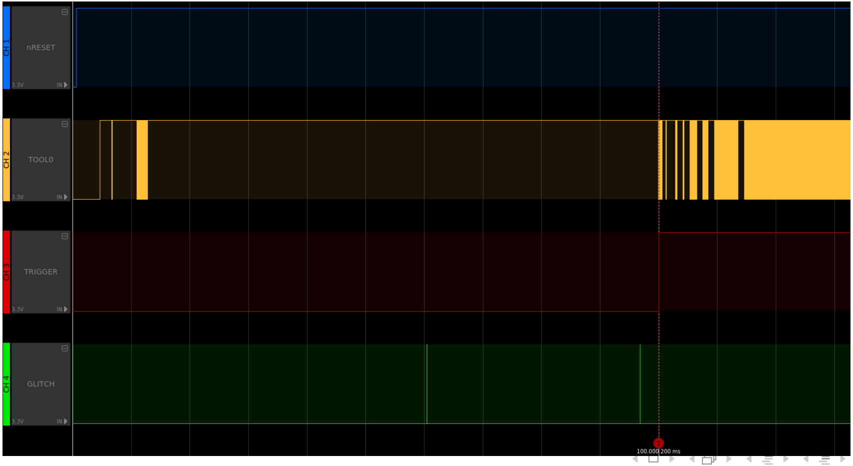
Glitching time

`glitch_dump`



Eventually...

Success!



Wasn't there a password?

In Renesas SDK examples, the password is always 10 null bytes

Everyone uses this default

- Wii Fit U Meter
- PS4 Syscon
- PSVita Syscon
- ...

Conclusion

- **Glitching is not impossible**
- **Can be a powerful tool**
- **This is not the only glitching method!**

Thanks to:

- Aitec/My-Tec, for a slight components discount
- FabLab Leuven and the people there, for PCB assembly help
- Many friends, for giving feedback
- Renesas, for a good educational VFI target

Questions

Questions?

Fedi/masto: `@pcy@icosahedron.website`
Mail me at `p@pcy.be`
melonDS, GodMode9, demoscene IRC/Discord
Slides available at `https://pcy.be/nl22`

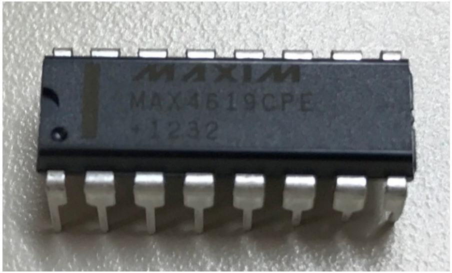
Price breakdown

Item	Part name	Price
PCBs	Fab @ Aisler	€21.84
PCBs (broken)	Fab @ JLC + stencil @ Aisler	€12.98
RL78 MCUs	R5F1006CASP	€45.98
MOSFETs	PSMN017-30PL	€30.55
Diodes	1N4148	€3.75
Headers	HEAD1R40	€3.40
Potentiometers	3386F / PPA50K	€48.51
Solder	Velleman Sn99Cu1	€2.20
Total		€169.21
Per participant		€8.46 ≈ €8.5

SPDT glitching

What if there is no REGC pin?

⇒ MAX4619



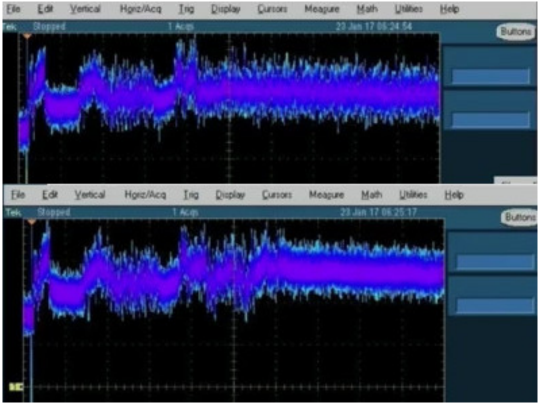
⇒ Also use "low" voltage as glitch parameter!

- ReCon 2017: Chris Gerlinsky: "Breaking Code Read Protection on the NXP LPC-family Microcontrollers"
- 36C3: noopwafel: iceGLITCH
- 36C3: Thomas 'stacksmashing' Roth: "TrustZone-M(eh)"

What if I don’t know the exact timing for the glitch?

Two options:

1. Guess / brute force
2. Power side channel!


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Math Utilities Help
Tek Stopped 1 Acqs 23 Jan 17 05:24:54
Buttons
File Edit Vertical Horiz/Acq Trig Display Cursors Measure Math Utilities Help
Tek Stopped 1 Acqs 23 Jan 17 05:25:17
Buttons

Source: Chris Gerlinsky: “Breaking Code Read Protection on the NXP LPC-family Microcontrollers”