

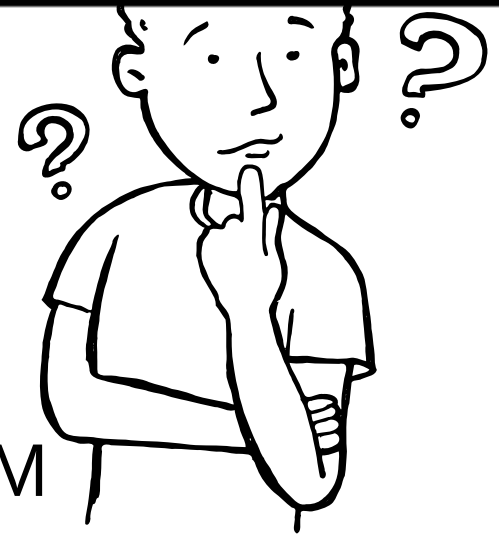
From off-the-shelf embedded devices to research platforms. Two case studies: a PLC and a SSD

Lucian Cojocar

Herbert Bos

Vrije Universiteit of Amsterdam

Who I am



- Find back-doors in firmwares:
 - Wrote a translator from binary to LLVM
 - Used it for parser detection (to appear @ACSAC)
- Binary analysis, especially for firmwares
 - Started with dynamic (firmware emulation system)
 - Switched to static analysis
 - Currently trying to use the best from both worlds
- Extract and analyze various firmwares

Presentation outline

1. An introduction to *hardware* hacking
 - What is a research platform?
 - Why do we need re-purposing a device?
2. First showcase: a PLC
 - An example of re-purposing a high-profile device.
3. Second showcase: a SSD
 - Another embedded device that was re-purposed.
4. Call for contribution:
 - We gather a list of such devices on this public wiki:
<http://embedded.labs.vu.nl>

Background

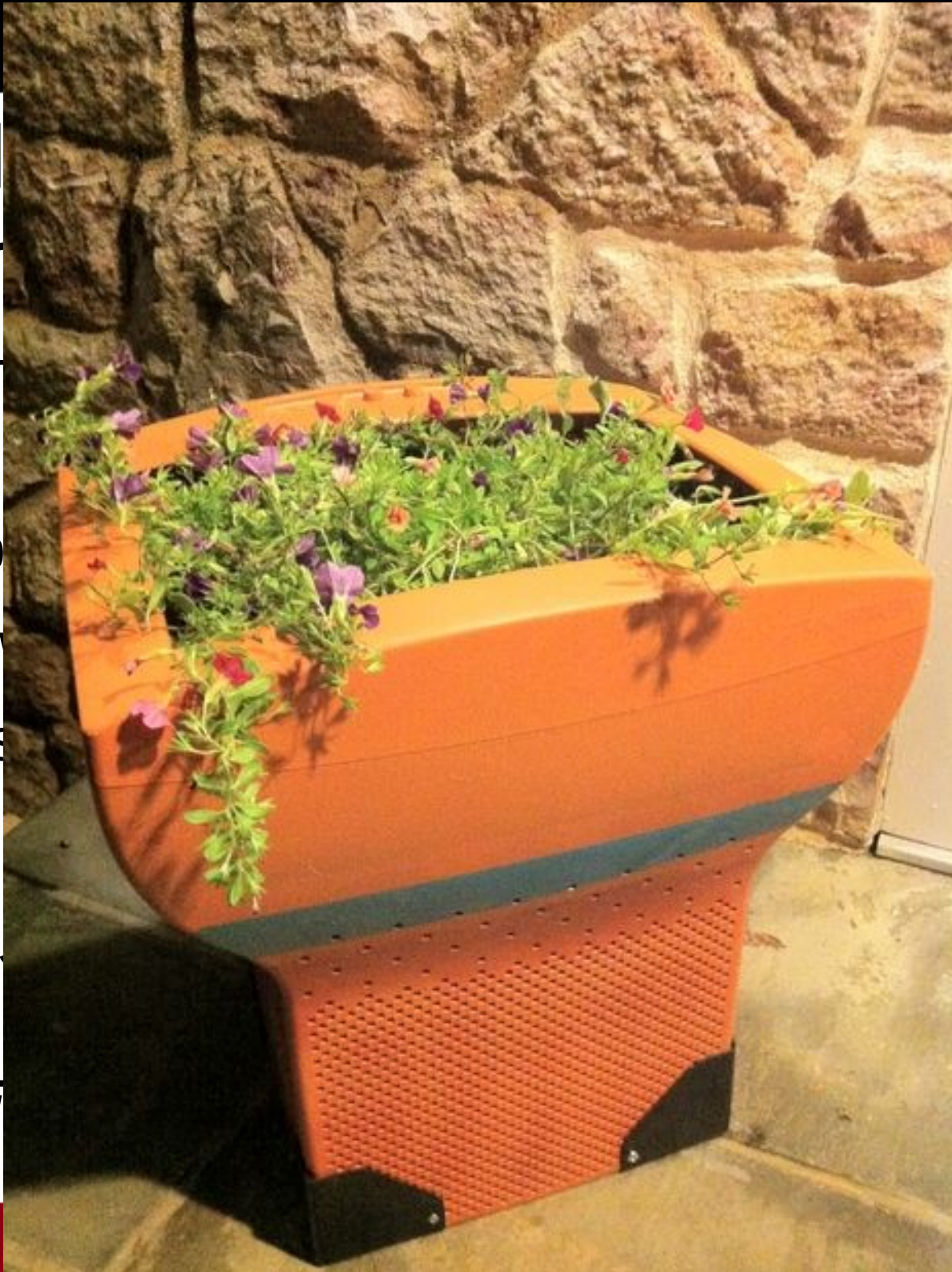
- Embedded devices are ubiquitous nowadays
 - IoT buzzword (Shodan project)
 - Old devices, out-dated code
 - They are also cheap :-)
- We apply various security related techniques to the firmware of these devices
 - The techniques are not *necessarily* new
 - Most of them are binary based
- Regardless of the goal we have to extract the firmware and, in some cases, we have to execute code on them

Look at relevant devices and reuse them for testing

M OR

- It's fun :-)
- Analyze
- Develop
- Test new
- Deploy s

*A real
time
findi*



g y?

*of the
d not in
t them*

Motivation of re-purposing or why *hardware* hacking?

- It's fun :-)
- Analyze the firmware that runs
- Develop new generic techniques
- Test new security oriented ideas
- Deploy security mechanism to old devices

A researcher should invest most of the time in developing new ideas and not in finding the suitable device to test them

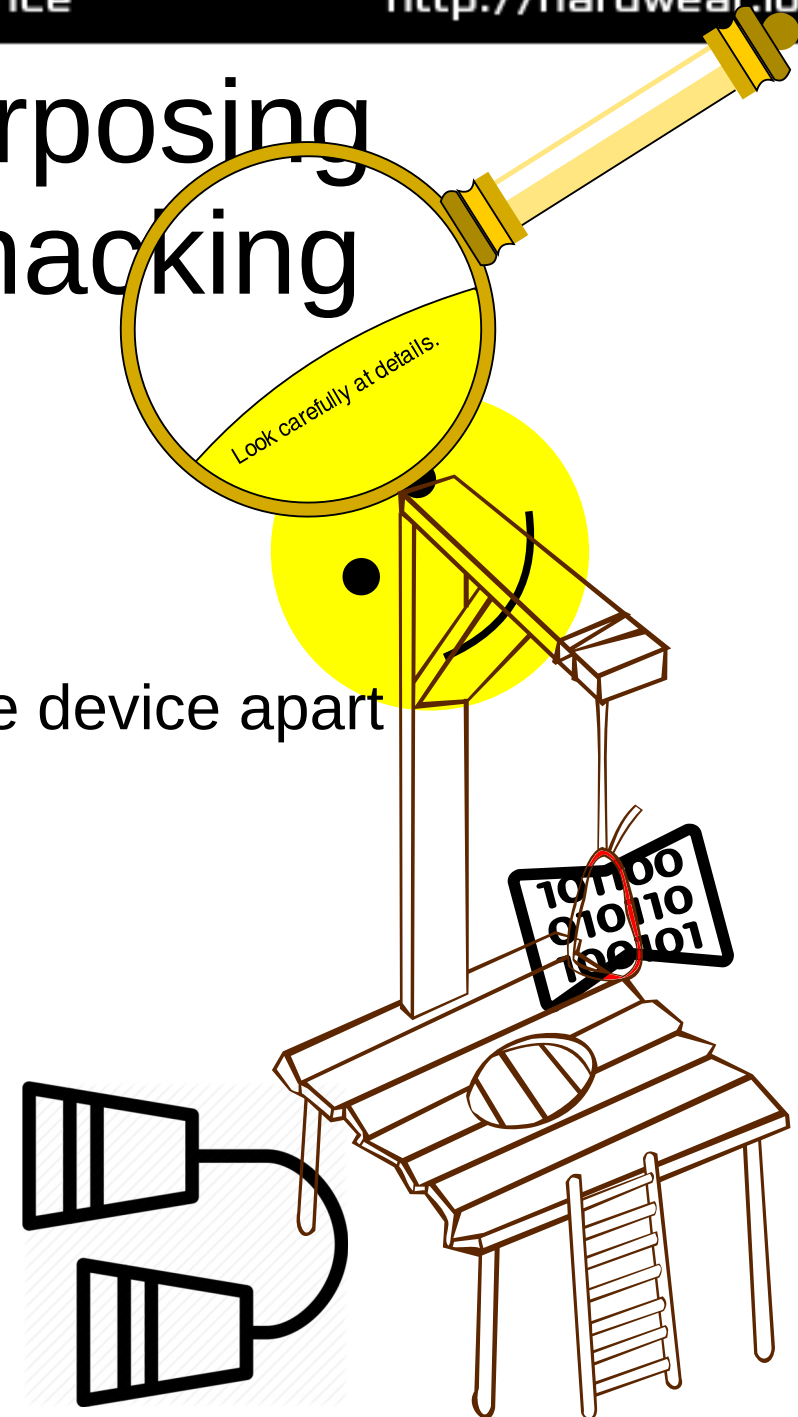
What does repurposing mean?

- Reusing an off-the-shelf embedded device with the goal of testing security related frameworks
- Roughly, this boils down to:
 - Running new (or partially new) code and,
 - *Communicate* with the target device.
- Examples:
 - Avatar: Dynamic firmware analysis (Zaddach et al.) – showcased on a GSM phone and a HDD
 - Firmalice: Detection of Authentication Bypass (Shoshitaishvili et al.) – tested on a camera and a printer
 - PoC back-doors on printers, HDDs, IPCamera, SatPhones

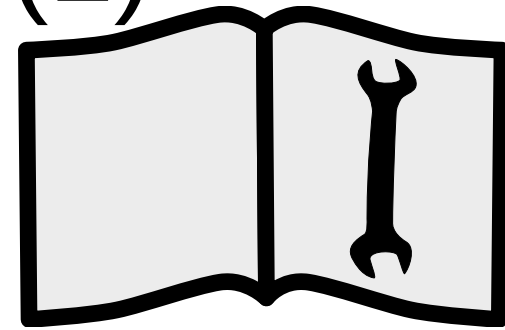
A primer on repurposing or on *hardware* hacking

Roughly three steps:

- Reconnaissance phase
 - Read the documentation and take the device apart
- Getting code execution
 - JTAG, debug channel
- Communication channel
 - UART interface, GPIO pin, display



Reconnaissance phase (1)

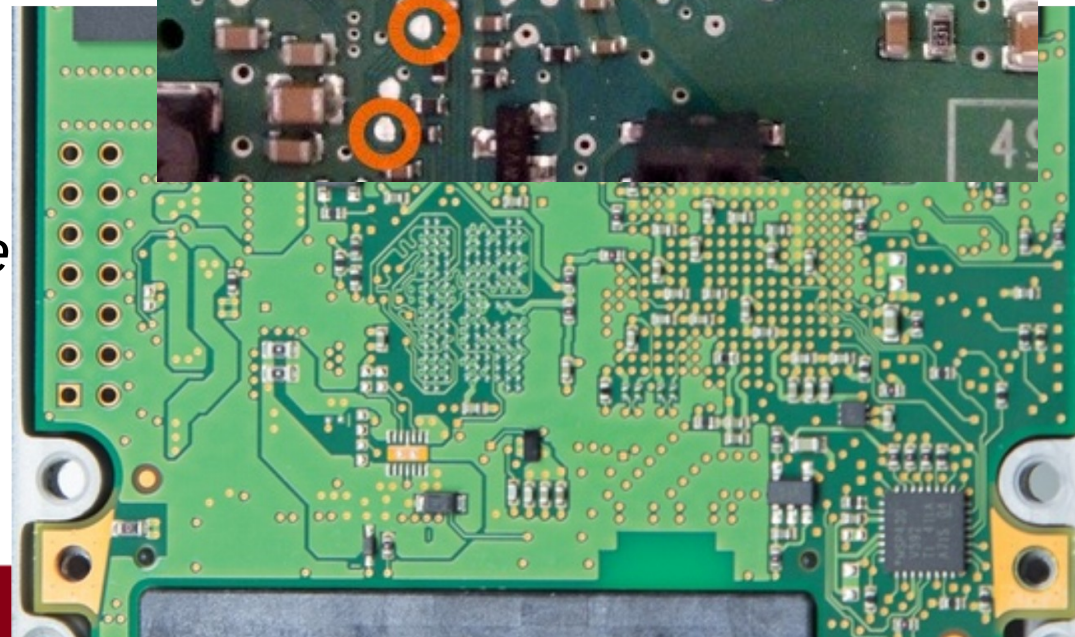
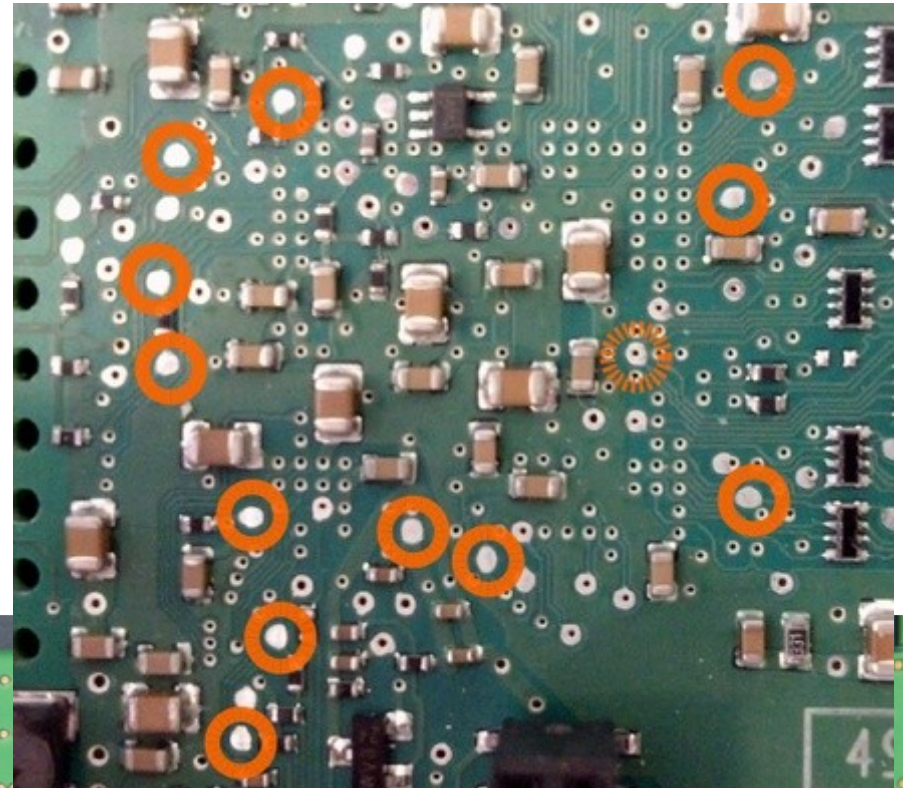


- Read the description of the device
- Read reference manual and SDKs
- Previous errors (CVEs) reported for this device
- Firmware updates, different versions, change-logs
- How widely used is this device?
- Are there other researchers that are working with this device?
- In short: *gather as much information as possible from the publicly available sources*

Reconnaissance phase (2)

Take it apart and look for:

- Test pads
- Known/Unknown chips
- Main SoC
- Unpopulated footprints
- Hidden headers
- Power lines
- Data-sheets
- *Build a high-level diagram of the system*



Code execution (1)

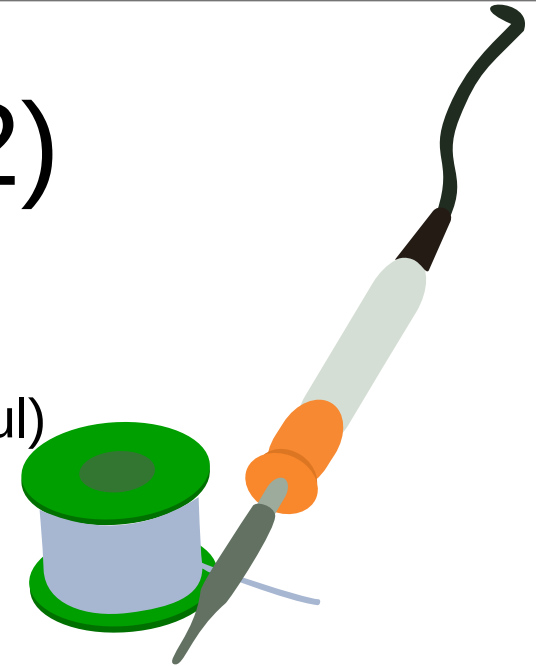
- Software (via *RFU* or command injection)
 - A bit of reverse engineering (on the **FU**) is required along with some trial and error
 - Signature/checksum of the FU (if any)
 - (Un)packing of the FU
 - check of the FU (is it off-line or on-line)
 - Are there any **buggy** software components used?
 - Can we exploit these bugs?
 - Updates are rare
 - (manual) Fuzzing still effective in some cases



Code execution (2)

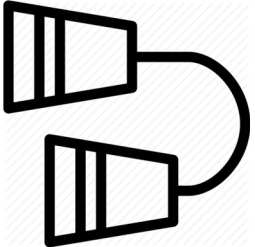
- Hardware

- Debug signals of the main SoC (data-sheet is useful)
 - JTAG, SWI, etc.
 - Debug facilities are sometimes still enabled
 - Look for unpopulated foot-prints at test patterns
 - Good candidates for JTAG signals can be identified (Breeuwsma)
- Flash chips that may store code (don't be afraid to use the soldering iron)
 - SPI flash is easy to access
 - *Sniff* some data, identify when the chip is used
 - Read and reprogram the chip
 - Simple is better – start looking at smaller (in terms of storage) chips first



Start simple: use “`while (1) ;`” patterns for reprogramming and observe the behavior

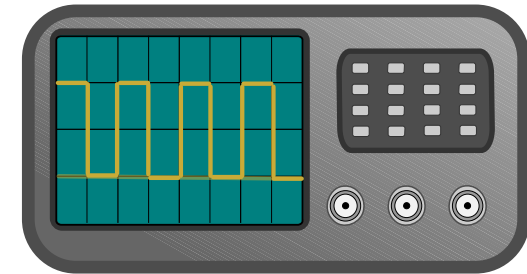
Communication channel (1)

- We need a way to communicate with the device 
 - Send and receive data
- Any controllable and observable signal can be used
 - Most of the SoCs have an UART interface
- Usually, it requires reverse engineering of the firmware
 - Identify the memory map (MMIO area)
 - Polling code patterns – `while (*MMIO_ADDR & 0x40) ;`
 - Search GPIO ports (LEDs indicating statuses might be connected to such ports)
 - Exception handling routines may help

Communication channel (2)

UART communication

- How to find the TX signal:
 - Is there output?
 - Trace (in firmware) the sync point of strings
 - Look for polling patterns followed by a single byte write
 - If it is DMA, things are more complicated :-)
 - It is rarely DMA
 - Probe with the oscilloscope potential candidates on the PCB
- How to find the RX signal:
 - Usually at the same (or very similar) MMIO address as the TX signal
 - Same polling pattern
 - Trial and error process: write code that is *verbose* **after** a byte is received through the RX signal



Recap

- In principle repurposing has three steps:
 - Reconnaissance phase
 - Data-sheets, PCB inspection
 - Code execution phase
 - JTAG, SWD etc.
 - Communication channel phase
 - UART, GPIO etc.
- We will repurpose two embedded devices:
 - PLC
 - SSD

PLC

(Programmable logic controller)

- Part of SCADA system
 - S7-1200 Series
 - Similar device was attacked by Stuxnet
 - High-profile device
- Exact details are in the paper



PLC goals

- We needed a test case for a research project
- The research framework used a GDB connection to a live system

We implement a GDB server on this device

PLC – reconnaissance (1)

Plenty of documentation available on-line

- On how to use the device,
- And how to add expansion boards,
- And how to program (application) the device with,
- And how to connect a *communication* module,
- But nothing denoting what hardware is inside.

PLC – reconnaissance (2)

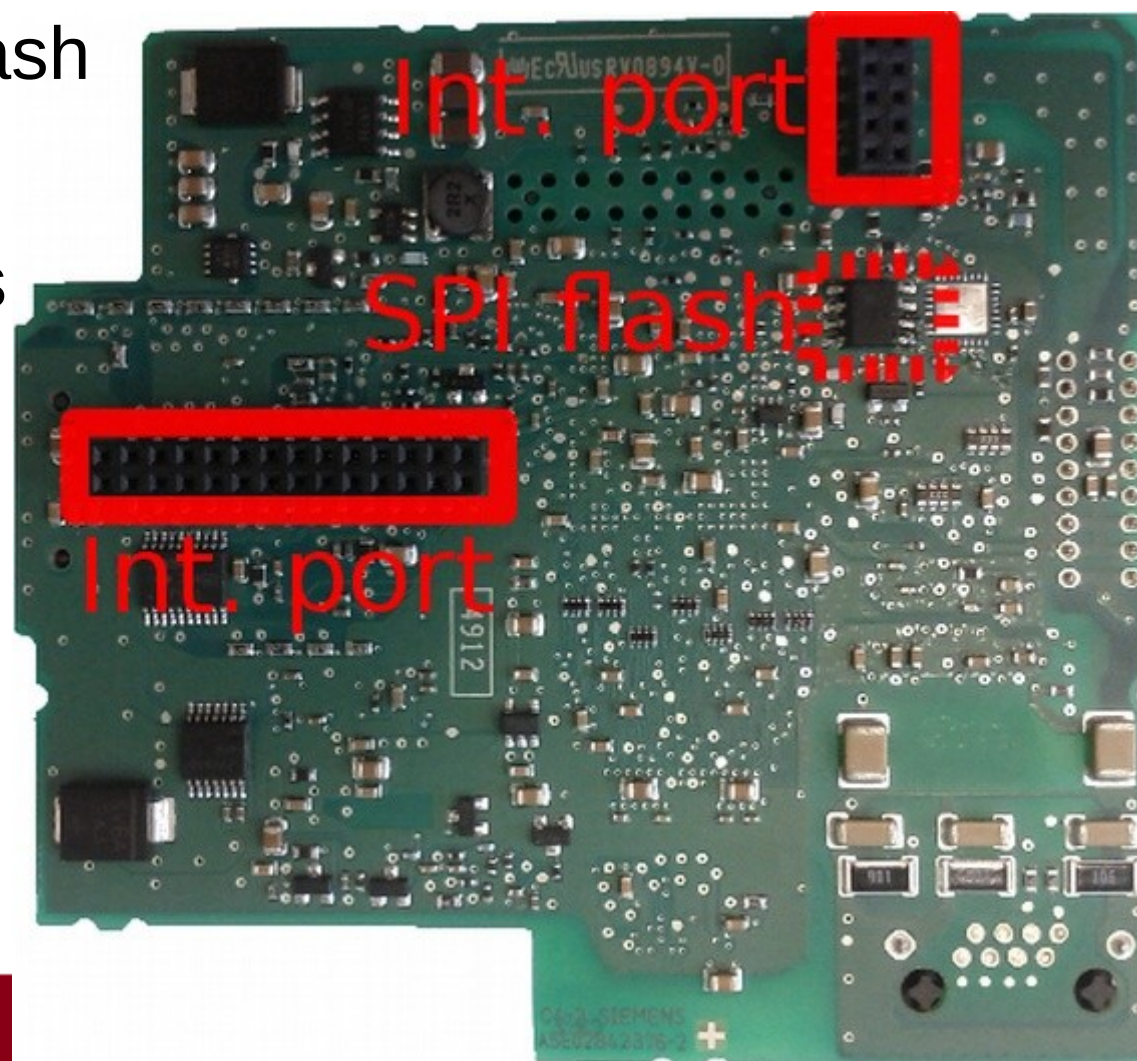
Firmware updates were available

- Packed with unknown algorithm
- Not signed, only checksummed
- The checking was done online
- Known text strings present in the firmware update
- The update can be performed through:
 - A special MMC card, or
 - Through a **webserver**

We tried to reverse the algorithm but it turned out to be faster to gain code execution by other means

PLC – reconnaissance (4)

- Take it apart! (bottom)
 - A nice SPI (1Mbit) flash
 - Data-sheet available
 - Two internal headers



PLC – code execution (1)

- Firmware was checksummed and compressed
 - The unpacking was done off-line
 - We dropped the idea of modifying the FU
- Unknown SoC, no data-sheet available
 - Previous versions of this PLC were ARM
 - No obvious pattern of unpopulated header (JTAG)
- Let's investigate the SPI flash → to the scope!

Anatomy of a bootloader

- Used only after the power-up
- Fairly small
- Does basic configs and check (RAM patterns)
- Loads a *bigger* code
- Finally, it jumps to the loaded code

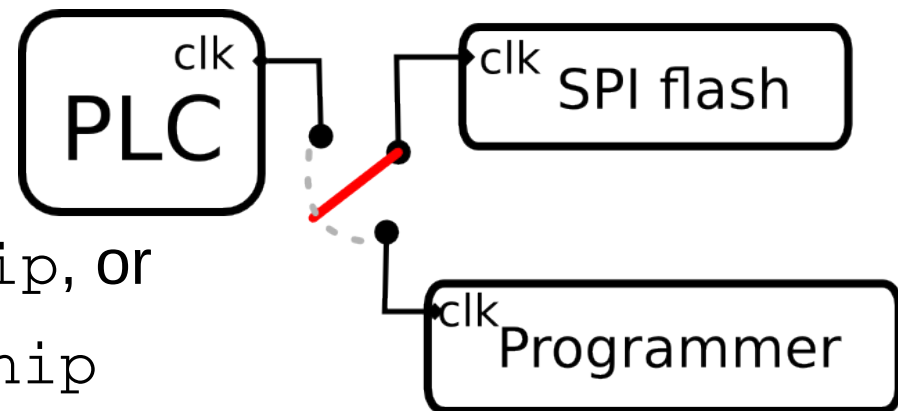
The code in the SPI flash is a good candidate.

PLC – code execution (2)

- Reflashed the bootloader with our code (j .)
- For testing: reflashed back the original bootloader
- The PLC was in good shape :-)
- We didn't had a stable version of the GDB stub
 - Solution: man-in-the middle on SPI Flash
 - Other solution might work

PLC – code execution (3)

- Man-in-the middle on the SPI:
 - Desolder only chip-select (CS), clock (CLK), data-in (DI)
 - Either:
 - `clk_prog` → `clk_chip`, or
 - `clk_board` → `clk_chip`



- We achieved code execution – proof: `j`. blocks the boot process, we can see this on the LEDs

PLC – communication channel (1)

- Two expansion ports (on each side of the CPU/PLC)
- CM 1241 RS232 is a nice module ... and it is referenced in the manual ... and it is connected to the above mentioned ports
- Reverse engineering:
 - We bet that the serial port is used in the simplest configuration: polling. **Idea:** search for tight loops that are checking statuses
 - There were not too many loops and not too many serial port types.
 - `while (* (base+offset) & 0x40); * (base) = x;`

PLC – communication channel (2)

- How do we test this?
- Tight loop that writes characters at the *presumably* serial MMIO output register
- Use (again) the oscilloscope to probe around.

Cursors

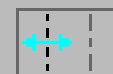
Type



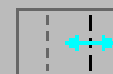
Time



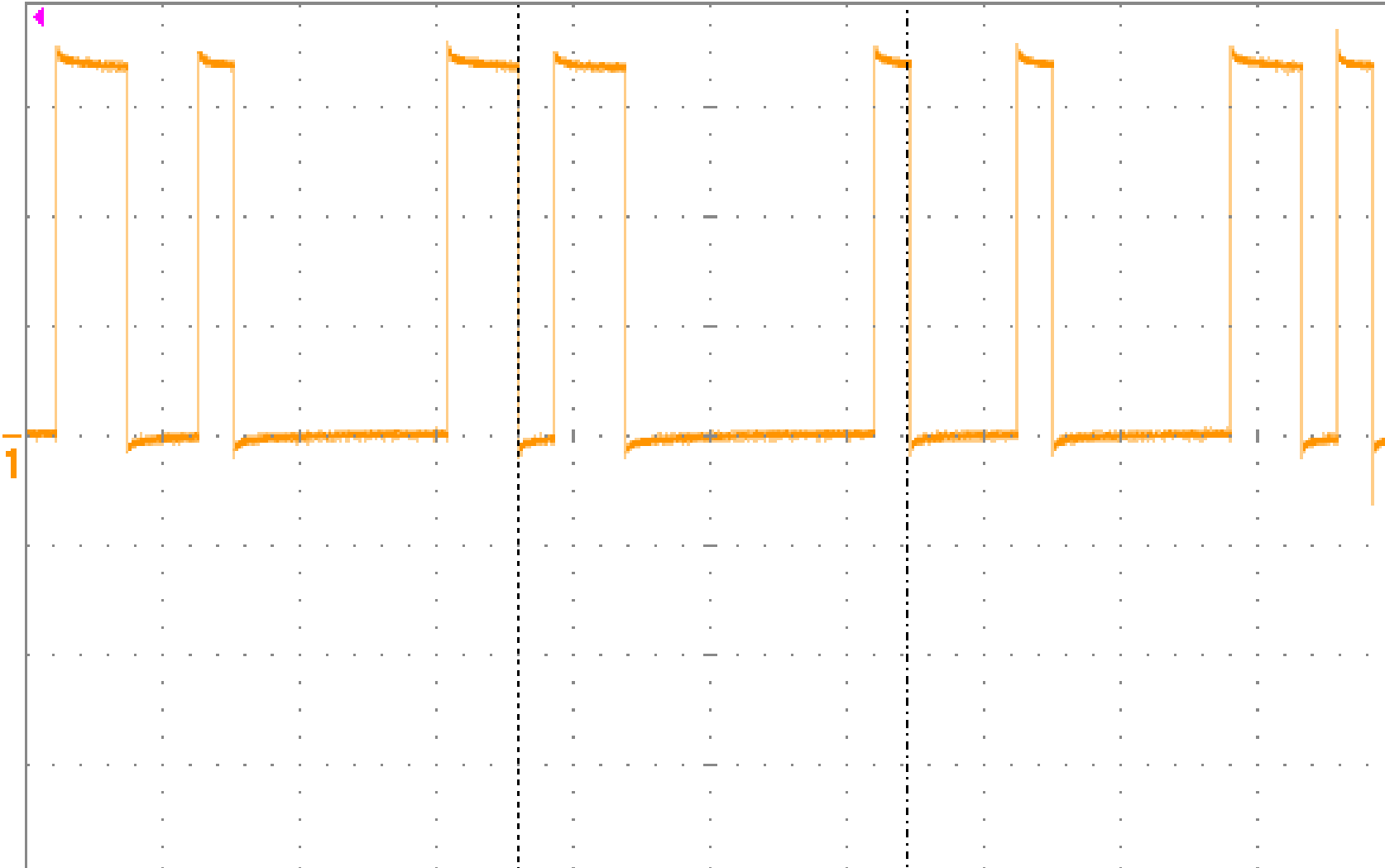
Cursor1



Cursor2



Track



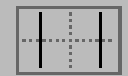
A: **1** +Pulse Width 52.00µs
 B: **1** Frequency 12.82kHz
 C: **1** Top 3.37V
 D: **1** -Pulse Width 25.99µs

$\Delta t=284\mu s$ $1/\Delta t=3.52kHz$ Edge **2** DC 0.00V

1: 1.00V	2: 50.0mV BW	3: 50.0mV BW	4: 500mV BW
DC1MΩ	DC1MΩ	DC1MΩ	DC1MΩ
ofs 0.00V	Empty	Empty	Empty

Cursors

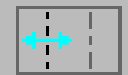
Type



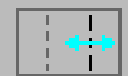
Time



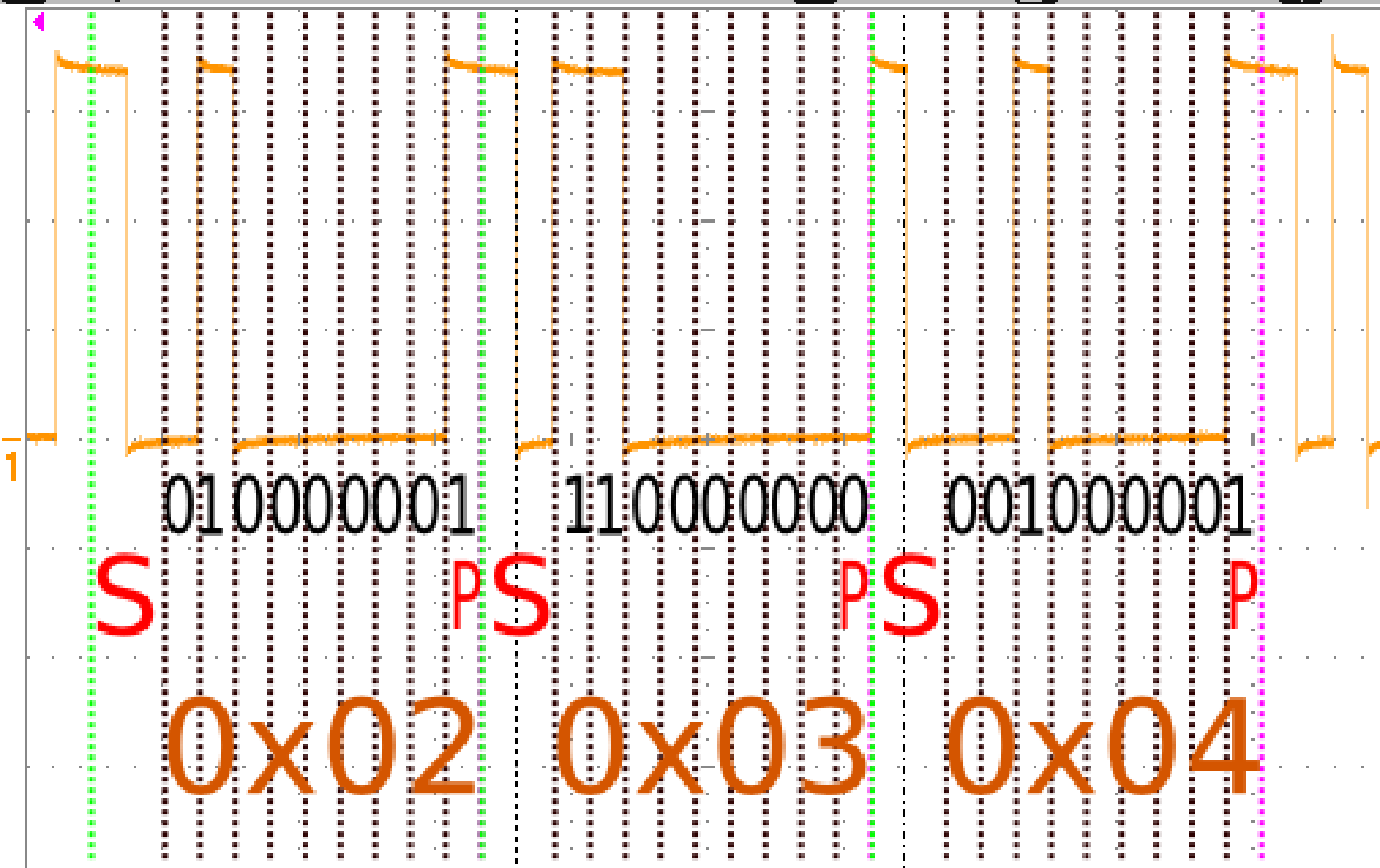
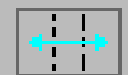
Cursor1



Cursor2



Track



- A: 1 +Pulse Width 52.00µs
- B: 1 Frequency 12.82kHz
- C: 1 Top 3.37V
- D: 1 -Pulse Width 25.99µs

Δt=284µs 1/Δt=3.52kHz Edge 2 DC 0.00V

1: 1.00V	2: 50.0mV	BW	3: 50.0mV	BW	4: 500mV	BW
DC1MΩ	DC1MΩ		DC1MΩ		DC1MΩ	
ofs	0.00V	Empty	Empty		Empty	

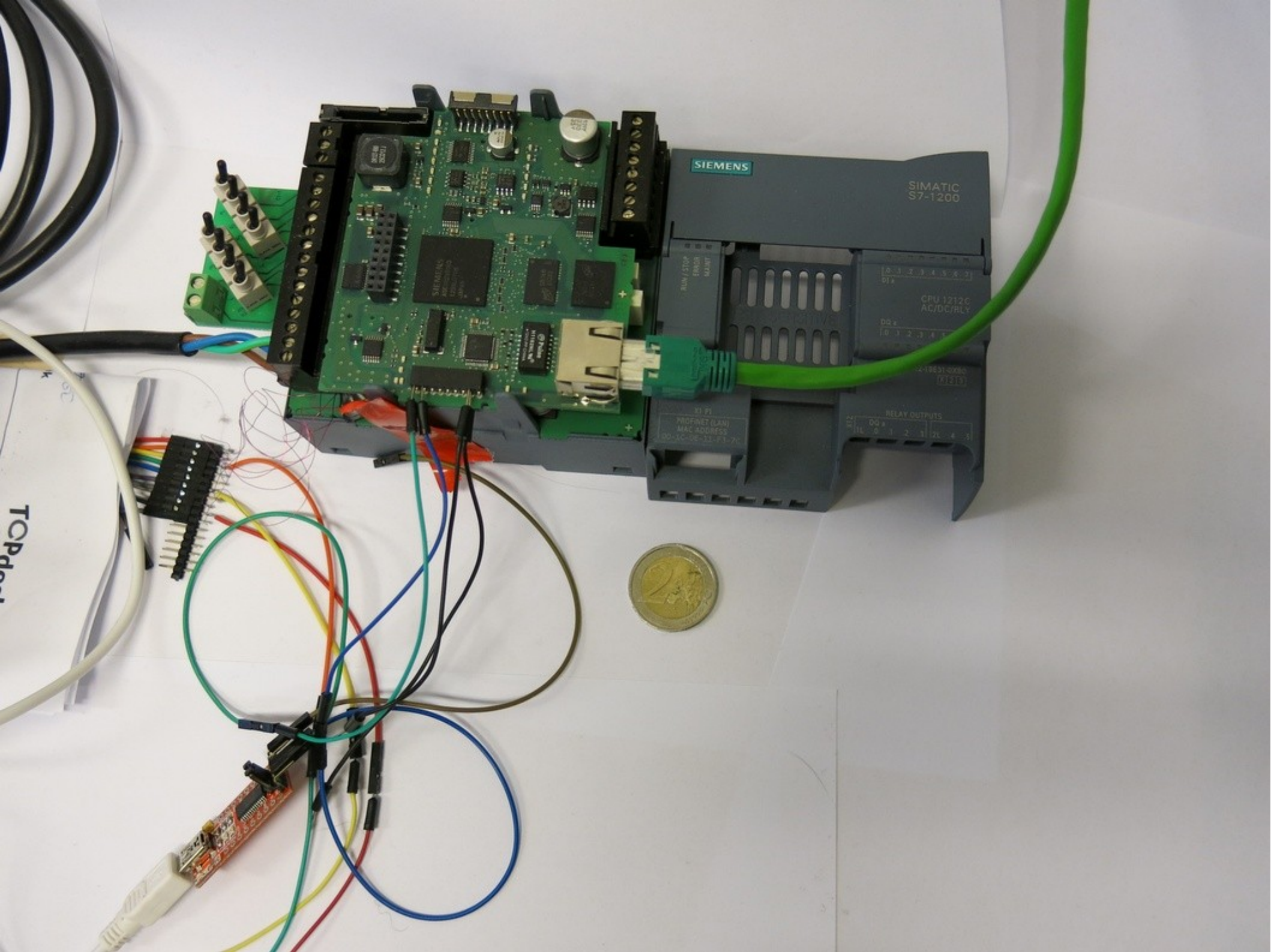
PLC – communication channel (2)

- How do we test this?
- Tight loop that writes characters at the *presumably* serial MMIO output register
- Use the oscilloscope to probe around

- Even parity
- 26 μ s pulse width \rightarrow 38400 bps
- two start bits
- LSB first

PLC – results

- We applied the three phases to the PLC
 - Achieved code execution by reprogramming the flash containing the bootloader
 - The communication channel is established through the stock UART interface



SIEMENS

SIMATIC
S7-1200

0 1 2 3 4 5 6 7

CPU 1212C
AC/DC/RLY

0 1 2 3 4

2-18631-0000
(1212C)

SI P1
PROFINET (LAN)
MAC ADDRESS
90-AC-0E-33-F1-7C

RELAY OUTPUTS

0 1 2 3 4 5



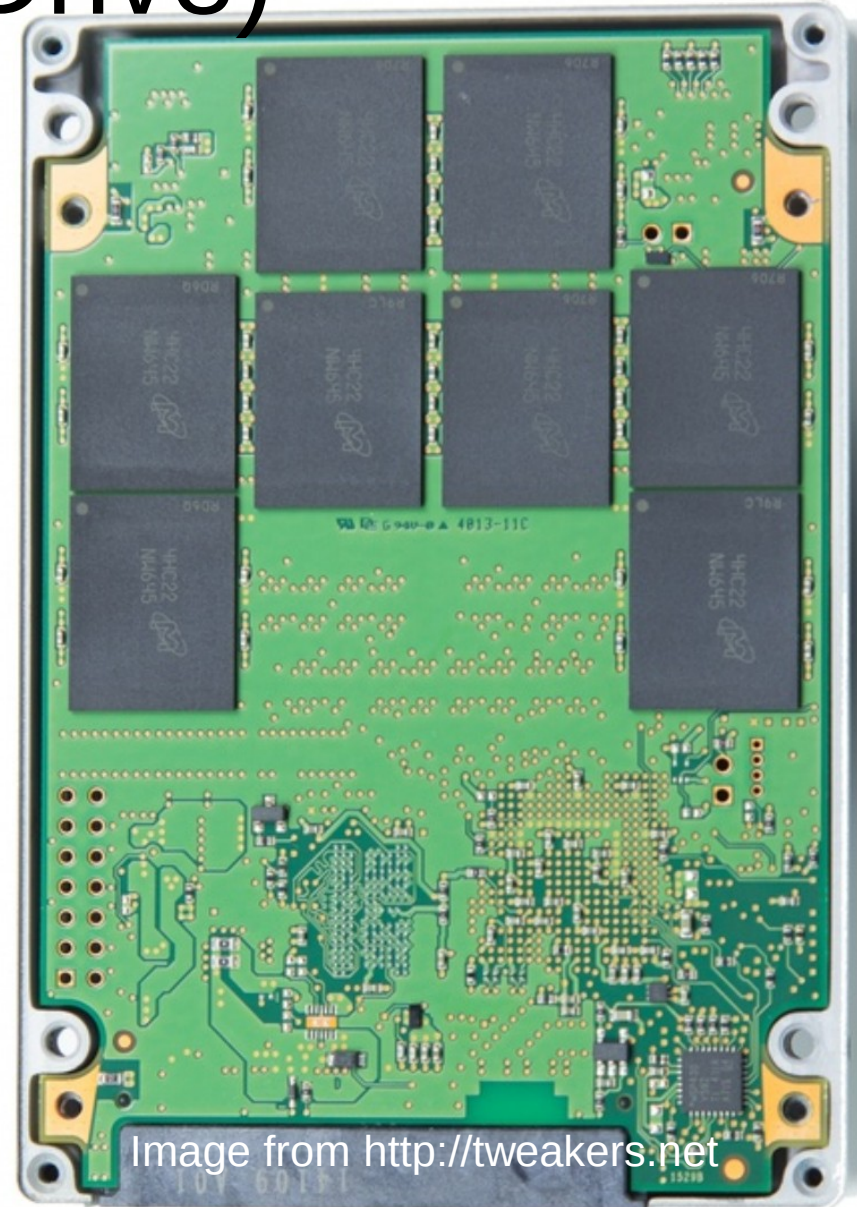
TCPD...

Next device

- We applied the three phases to the PLC
- Let's move on to the SSD

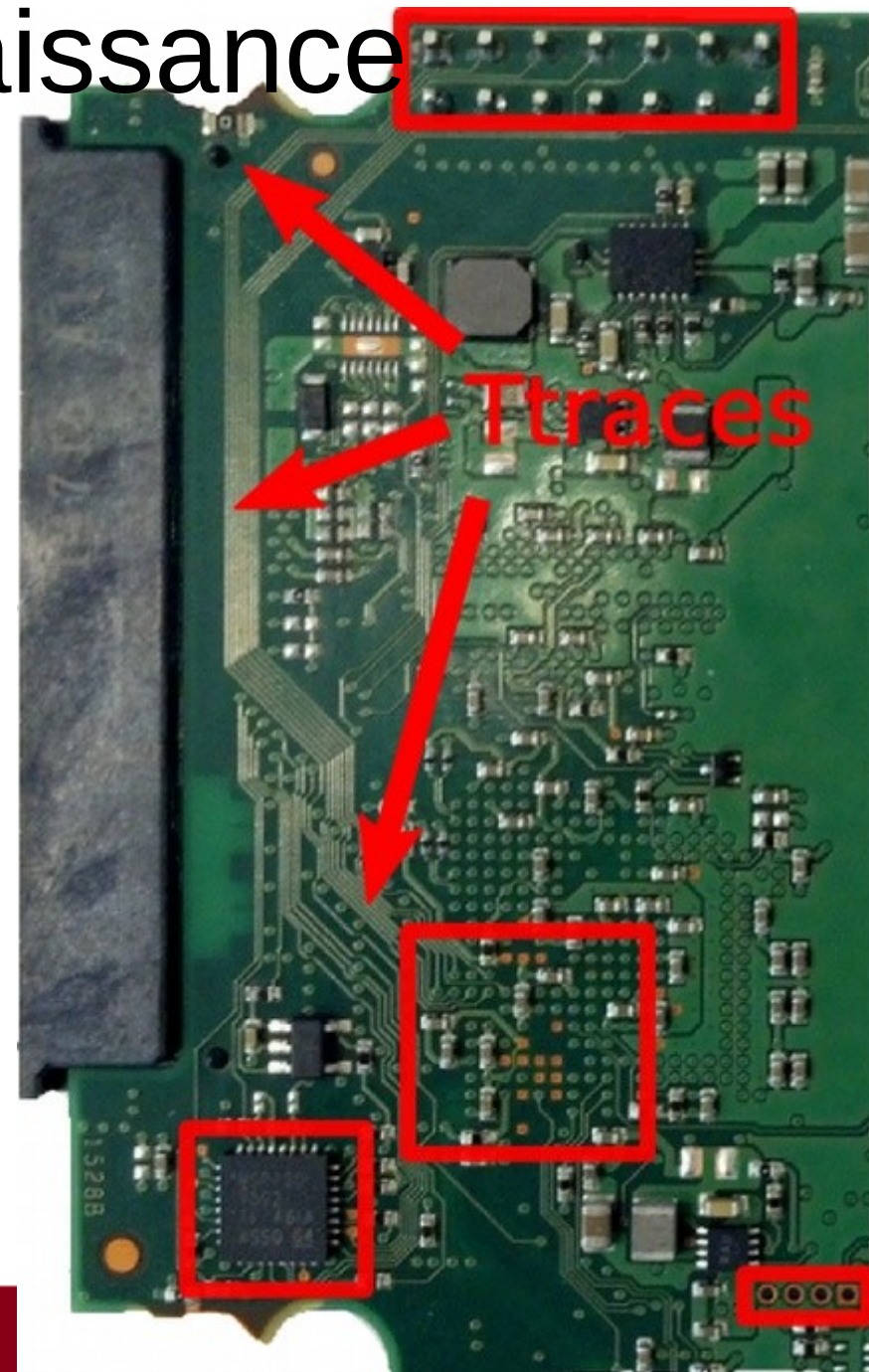
SSD (Solid State Drive)

- Crucial MX100
 - 128GB SATA 6Gb/s
- Pictures of the PCB on-line
- PCB is very light
 - Not many components



SSD – reconnaissance

- FU are present
- More interesting things are on the PCB:
 - Test pads
 - Unpopulated (promising) footprints
 - Known MCUs
 - (mostly) Known SoC



SSD – code execution (1)

- JTAG candidate
 - Checked the ground pins
 - It matched *standard* ARM pinout
- OpenOCD worked out of the box
 - `jtag newtap core0 cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id 0x121003d3`
 - `jtag newtap core1 cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id 0x121003d3`
 - `target create ssd_core0 dragonite -endian little -chain-position core0.cpu`
 - `target create ssd_core1 dragonite -endian little -chain-position core1.cpu`

SSD – code execution (2)

- Two ARM cores
- Memory read and write is working
- Code execution successfully tested
 - Tested with a tight loop over a set of NOPs
 - We are able to break code execution by halting the CPU via JTAG
 - No caching problems
 - Watchdog interferes when only one core is halted

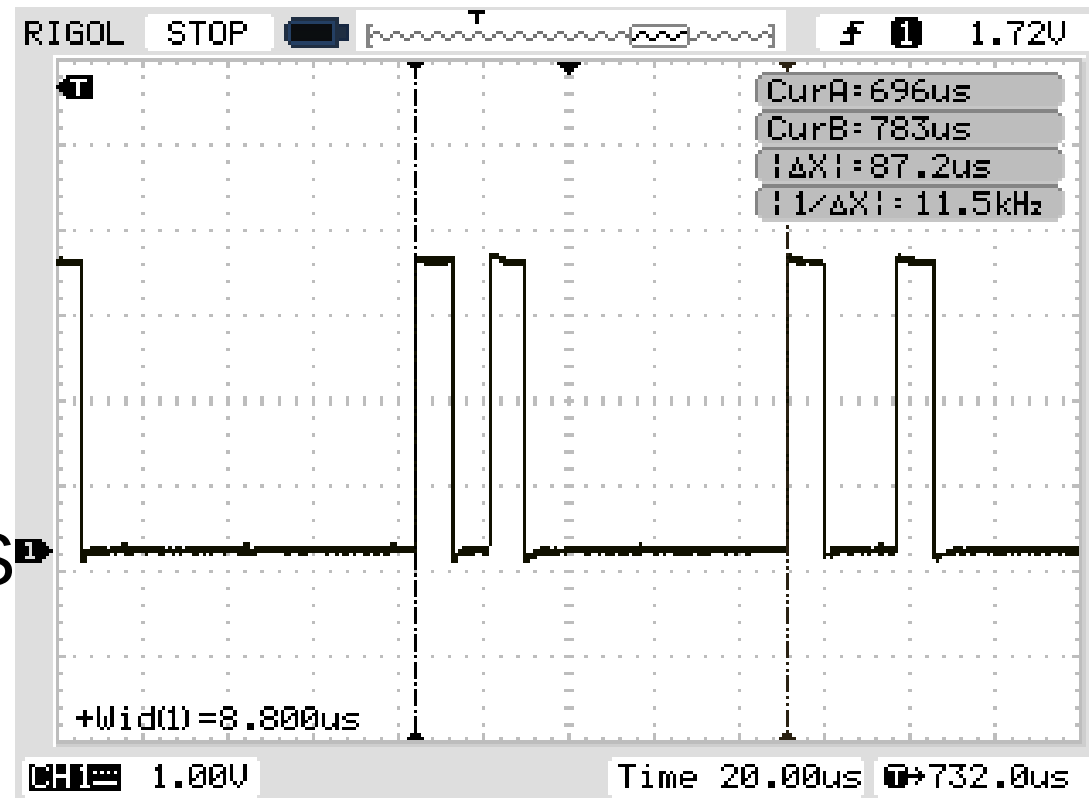
SSD – communication channel (1)

- Dump the memory through JTAG and inspect it:
 - Debug strings are present in memory
 - They **should** be printed
 - The *error logging routine* is not hit during normal operation
 - Tested by putting a breakpoint
 - A MMIO address (allegedly of the UART port) is used by this routine

Run our own code and use the
oscilloscope.

SSD – communication channel (2)

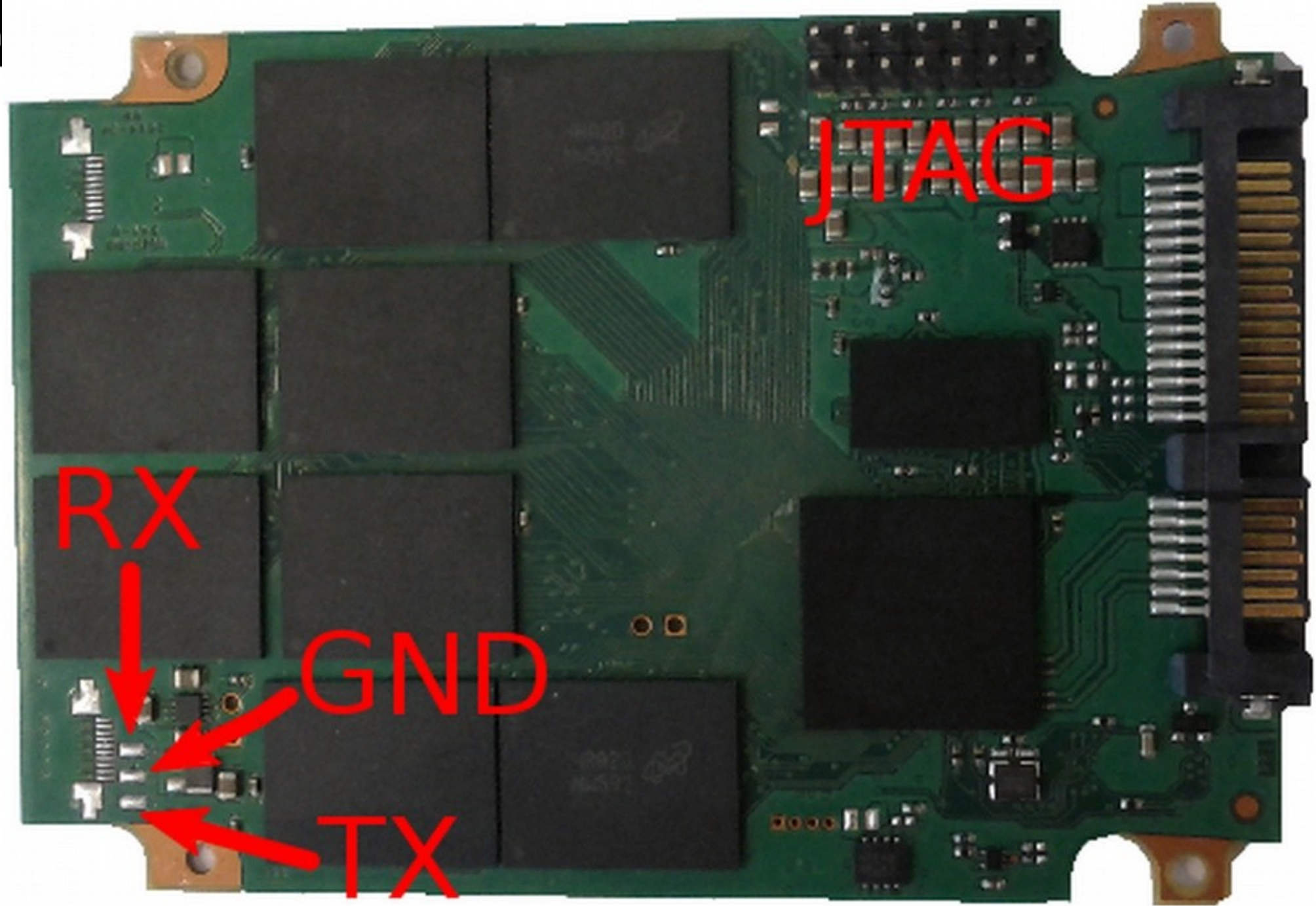
- Default config:
 - one stop bit
 - no parity
 - LSB order
 - The pulse width $8.8\mu\text{S}$



The position of the RX signal, on the PCB, is *obvious*.

SSD – results

- We applied the three phases to the SSD
 - Achieved code execution by making use of JTAG
 - The communication channel is established through the UART interface



Results

- We showed how to repurpose two off-the-shelf embedded devices
- What we did with the PLC:
 - Used dynamic analysis on it (symbex, taint tracking),
 - Reported a bug in the PLC webserver (CVE-2014-2258)
 - Used it as a test-case for parser detection (ACSAC 2015, to appear)
- What we did with the SSD:
 - Used it as a test-case for parser detection
 - Designed CTF challenges on it (work-in-progress)
 - There are more things than can be done
- We believe that reproducibility of results is valuable for research, especially in this area.

Let's share information about these embedded devices!

- Wiki: <http://embedded.labs.vu.nl>
- Gather the information that is needed for repurposing:
 - We do not share (or host) binaries
 - We want to share the method of obtaining:
 - Code execution
 - Communication channel (if available)
- We do not want to overlap with *-wrt (SoHo routers may not be that interesting)



Conclusion

- Repurposing of off-the-shelf embedded devices:
 - We want to develop and test security related ideas
- How to do this:
 - Three steps: reconnaissance, code execution and communication channel
- Share the information: <http://embedded.labs.vu.nl>
 - We want to focus on ideas instead of **random** hacking
- Two devices: an PLC and an SSD