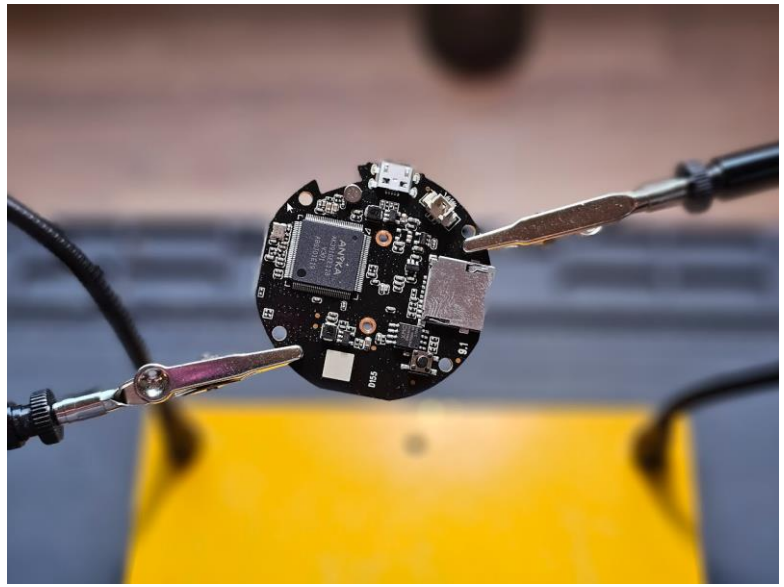
Anyone Can Hack IoT

A Beginner’s Guide to Hacking Your First IoT Device.

Andrew Bellini





The “S” in IoT stands for security



The “S” in IoT stands for security

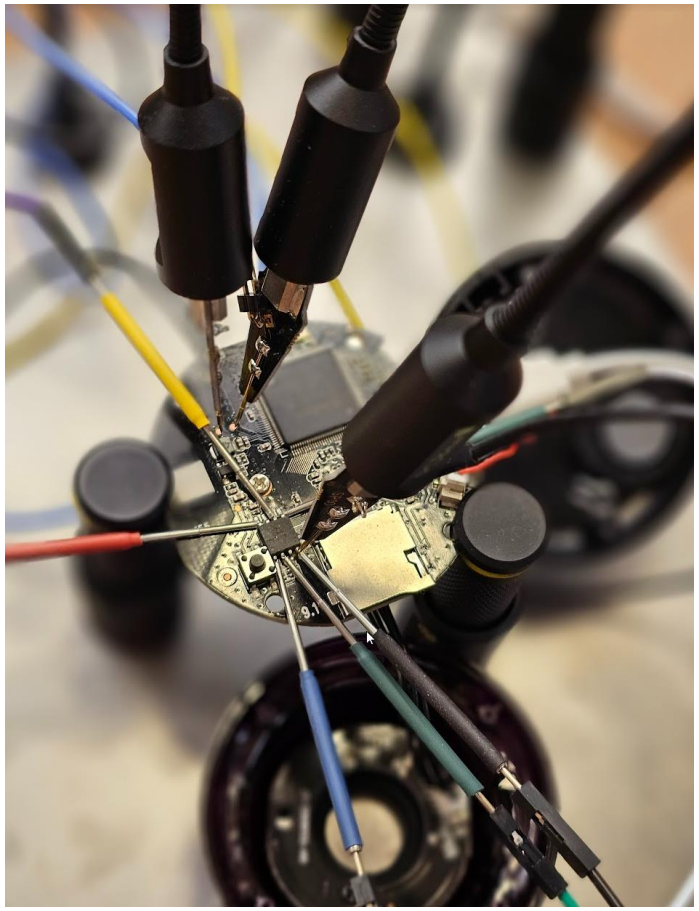
> 15 billion devices in 2024

~ 30 billion devices in 2030

Barriers to learning IoT security

- **$$$ Expensive**
  - Need tools, gear
  - Buy targets to test
  - Lack of affordable training

- **Complicated**
  - Electronics, Circuits, Hardware, Soldering
  - Special protocols



whoami

- Andrew Bellini(@DigitalAndrew)
- Electrical Engineer, P.Eng
- Content Creator at TCM Security
- Creator of TCMs IoT Hacking Course and PJIT Certification
- Link up with me: andrewbellini.com





Agenda

1. Safety and Legal Considerations
2. Picking a device to hack
3. Building out an affordable toolkit
4. Locating and using hardware interfaces
5. Acquiring firmware
6. Analyzing and reverse engineering firmware



Staying out of trouble

- Only test devices you own or are authorized to
- If you’re reverse engineering firmware, check your local laws and regulations
- Be ethical and follow responsible disclosure

Staying safe

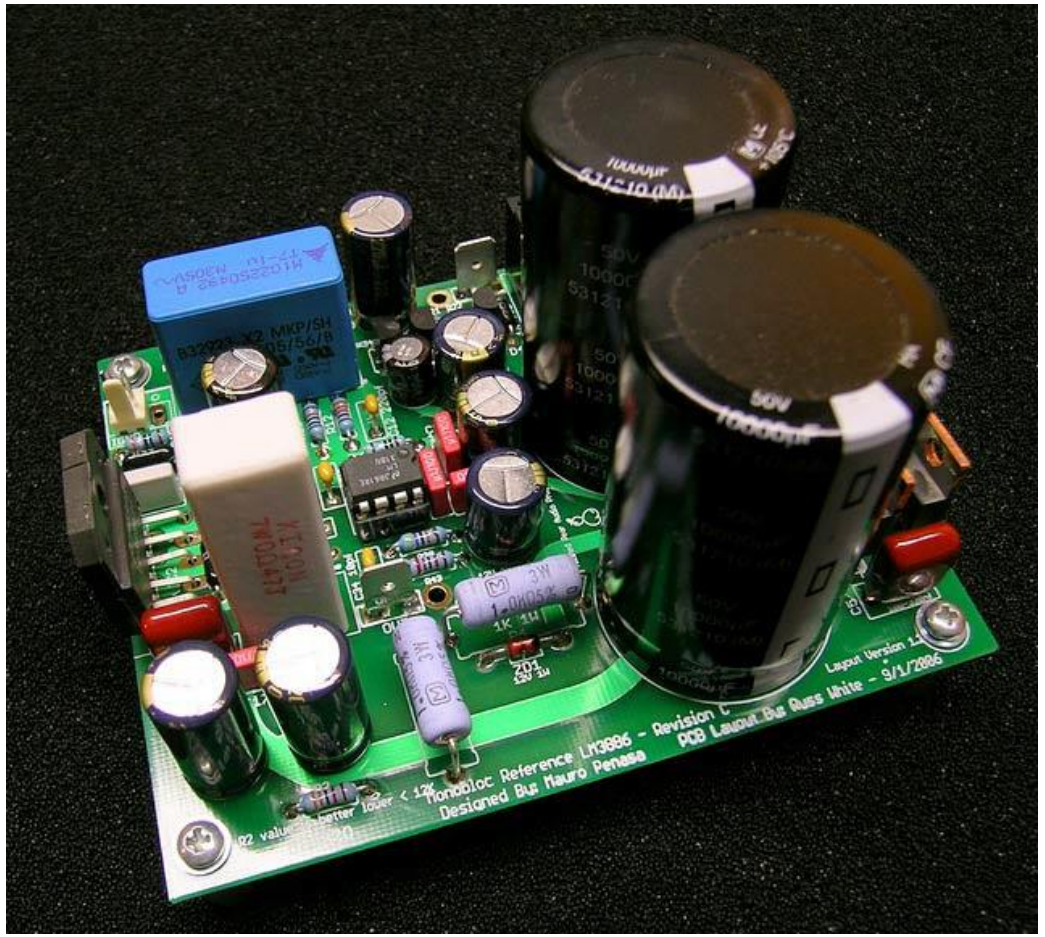
- IoT hacking inherently safe with a few constraints
- Never work on high voltage (the voltage from your outlets)
- Never used damaged or modified power supplies
- Power off circuits when not in use





Staying safe

- Be mindful of large capacitors
- Not common in most IoT devices

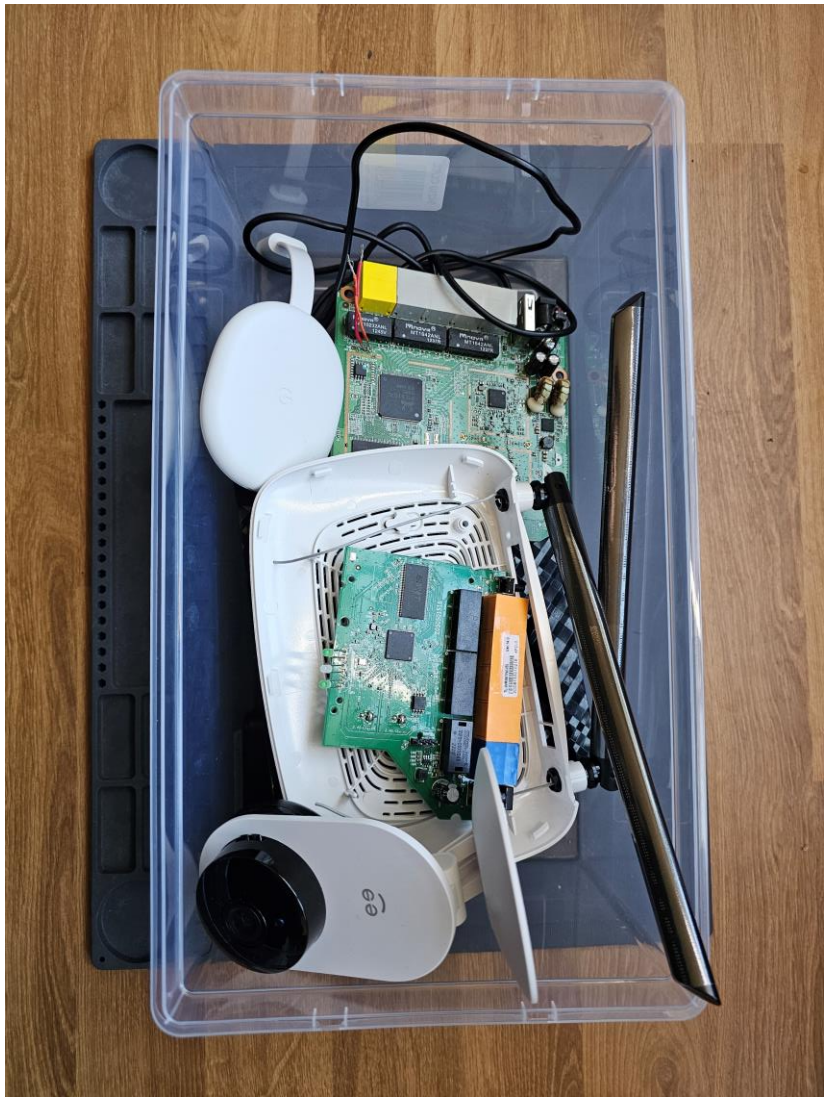


Picking a target

What device should I start with?

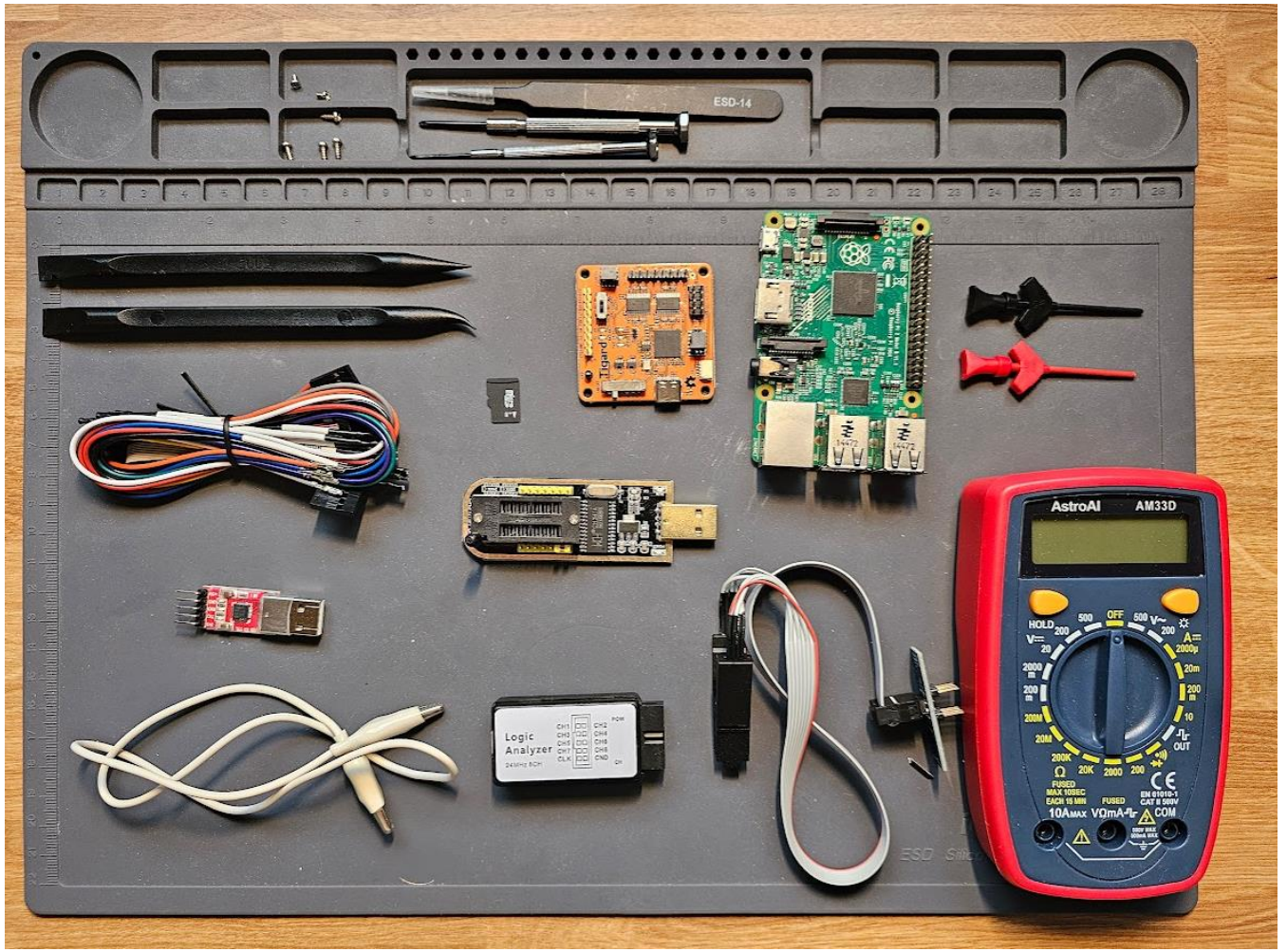
- Cheaper the better – Free?!
- Don’t hack anything you don’t mind bricking :P
- Cheap routers and smart/IP cameras are good to start

Just try something!



Building an affordable toolkit

- Goals is < $100
- Follow 80/20 rule
- Don’t need a dolphin toy…



Building an affordable toolkit

Digital Multimeter

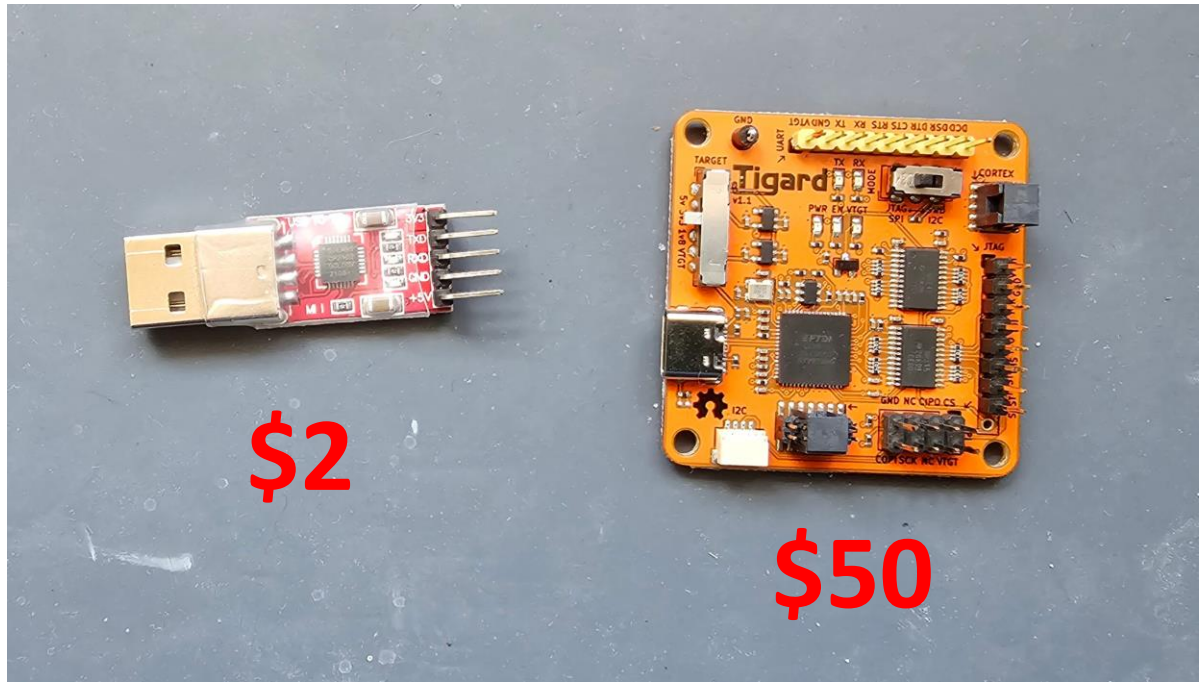
- Swiss army knife of electronics measurements
- Verify voltage levels
- Detect test/debug ports
- $5 - $100



Building an affordable toolkit

USB-X Adapter

- Something that can talk to debug ports
- Common protocols include UART, SPI, I2C, JTAG , SWD
- UART most common

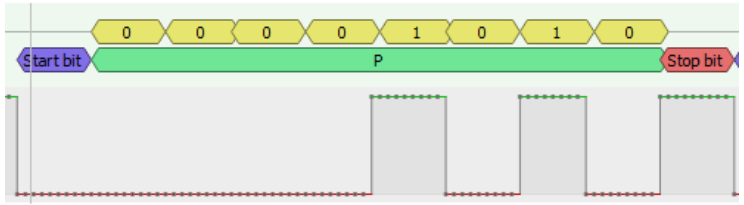


Building an affordable toolkit

Logic Analyzer

- Helps identify and decode debug ports/protocols
- May not even need one when starting
- Handy if things aren’t working

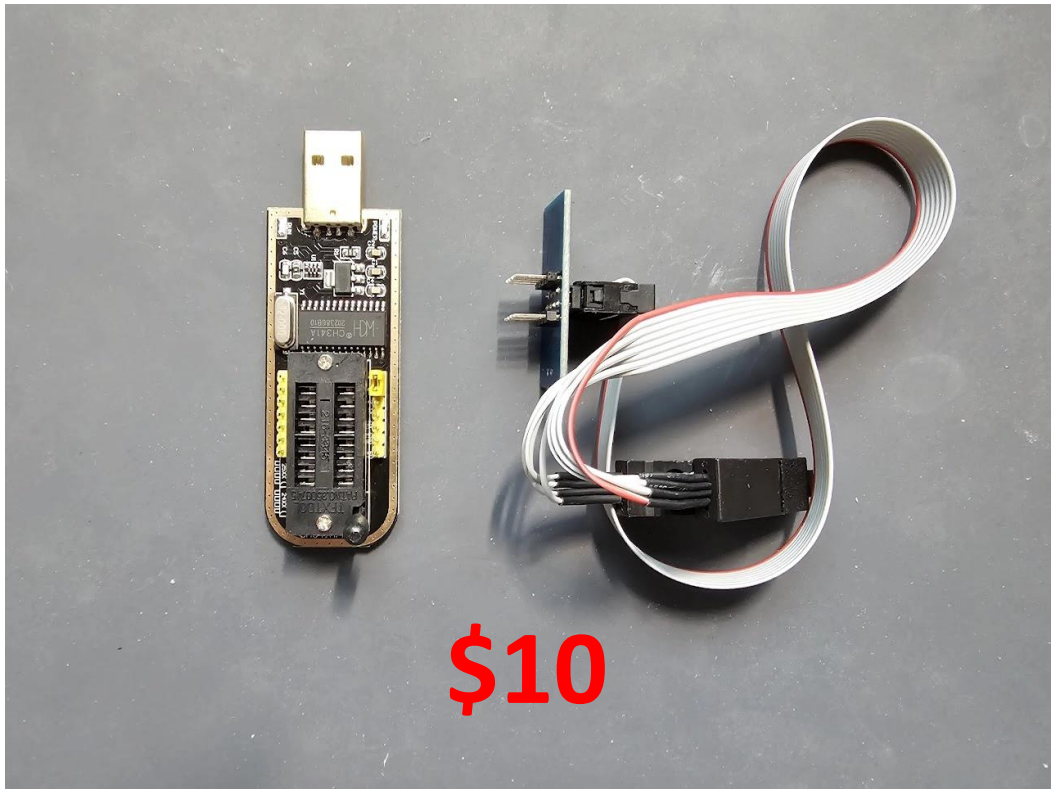




Building an affordable toolkit

Flash Programmer

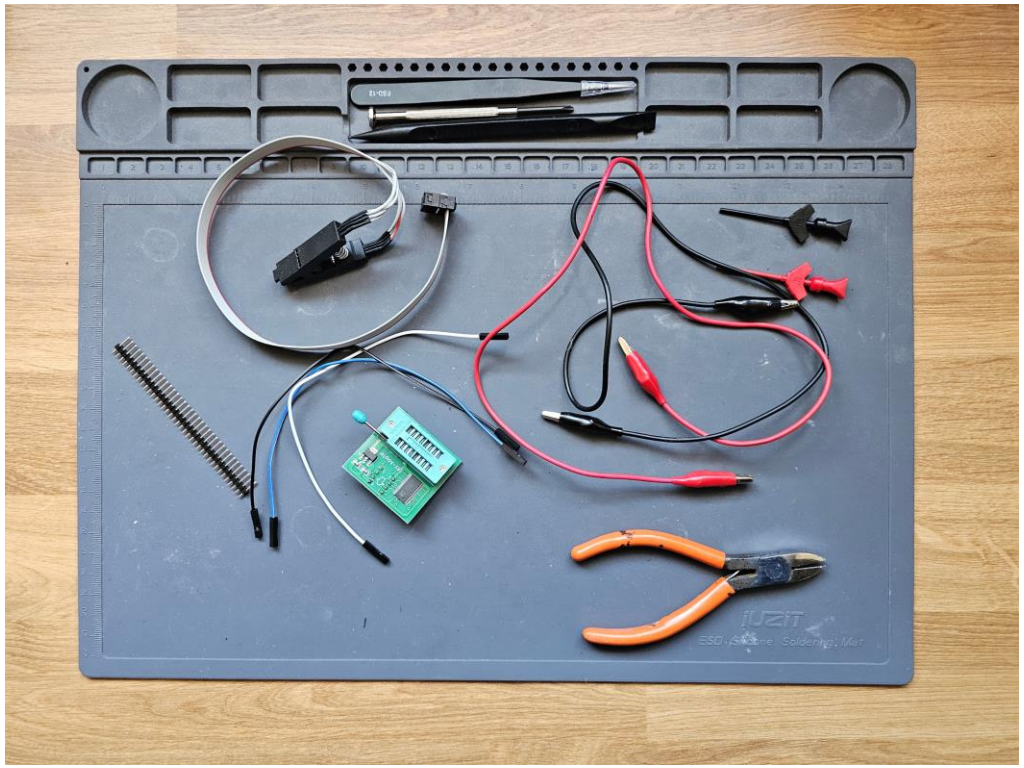
- Used to extract firmware and unbrick devices
- Most cheap devices use standard pinout SPI flash
- Tigard is a great option
- May not need when starting out



Building an affordable toolkit

Hand Tools, Cables, Adapters

- Buy these as required
- Will differ depending on device
- May not need any or very few when starting out



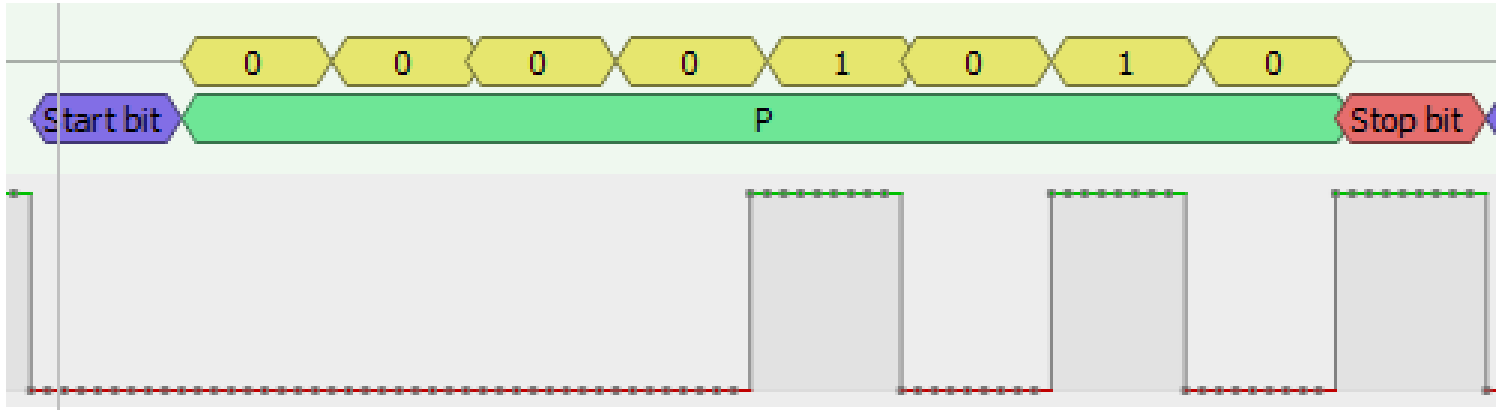
Hardware Interfaces

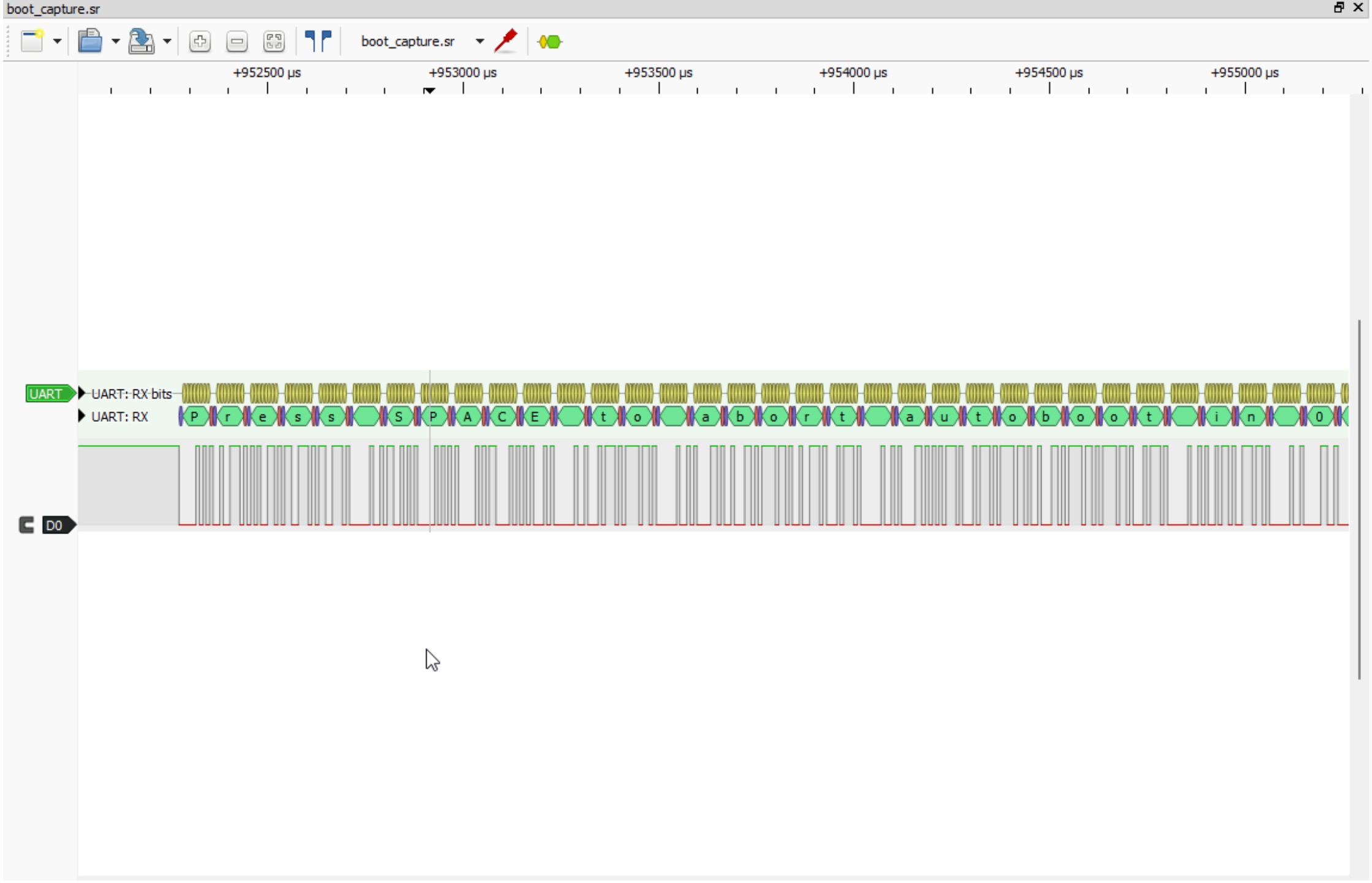
- Used for debugging, programming and Q/A
- Most common are UART and JTAG/SWD
- Frequently left on and unsecured
- We'll focus on UART



What is UART?

- Universal Asynchronous Receiver Transmitter
- Old and very common protocol for electrical communication
- Uses pulses of voltage to transmit binary (think like layer 1)




boot_capture.sr
boot_capture.sr
+952500 µs
+953000 µs
+953500 µs
+954000 µs
+954500 µs
+955000 µs
UART
UART: RX bits
UART: RX
Press SPACE to abort autoboot in 0
D0



What can we do with UART?

- View device logging
- Get a shell for further enumeration
- Access bootloader
- Dump firmware

```
File  Actions  Edit  View  Help
[04080B0F][04080C0C][8A7F0000][26253B38][00262539]
PU Setting Cal Done


U-Boot 1.1.3 (Feb  3 2021 - 10:10:08)

Board: Ralink APSoC DRAM:  32 MB
relocate_code Pointer at: 81fc0000
flash manufacture id: 1c, device id 70 16
Warning: un-recognized chip ID, please update bootloader!
================================================
Ralink UBoot Version: 4.3.0.0
------------------------------------------------
ASIC 7628_MP (Port5<->None)
DRAM component: 256 Mbits DDR, width 16
DRAM bus: 16 bit
Total memory: 32 MBytes
Flash component: SPI Flash
Date:Feb  3 2021  Time:10:10:08
================================================
icache: sets:512, ways:4, linesz:32 ,total:65536
dcache: sets:256, ways:4, linesz:32 ,total:32768

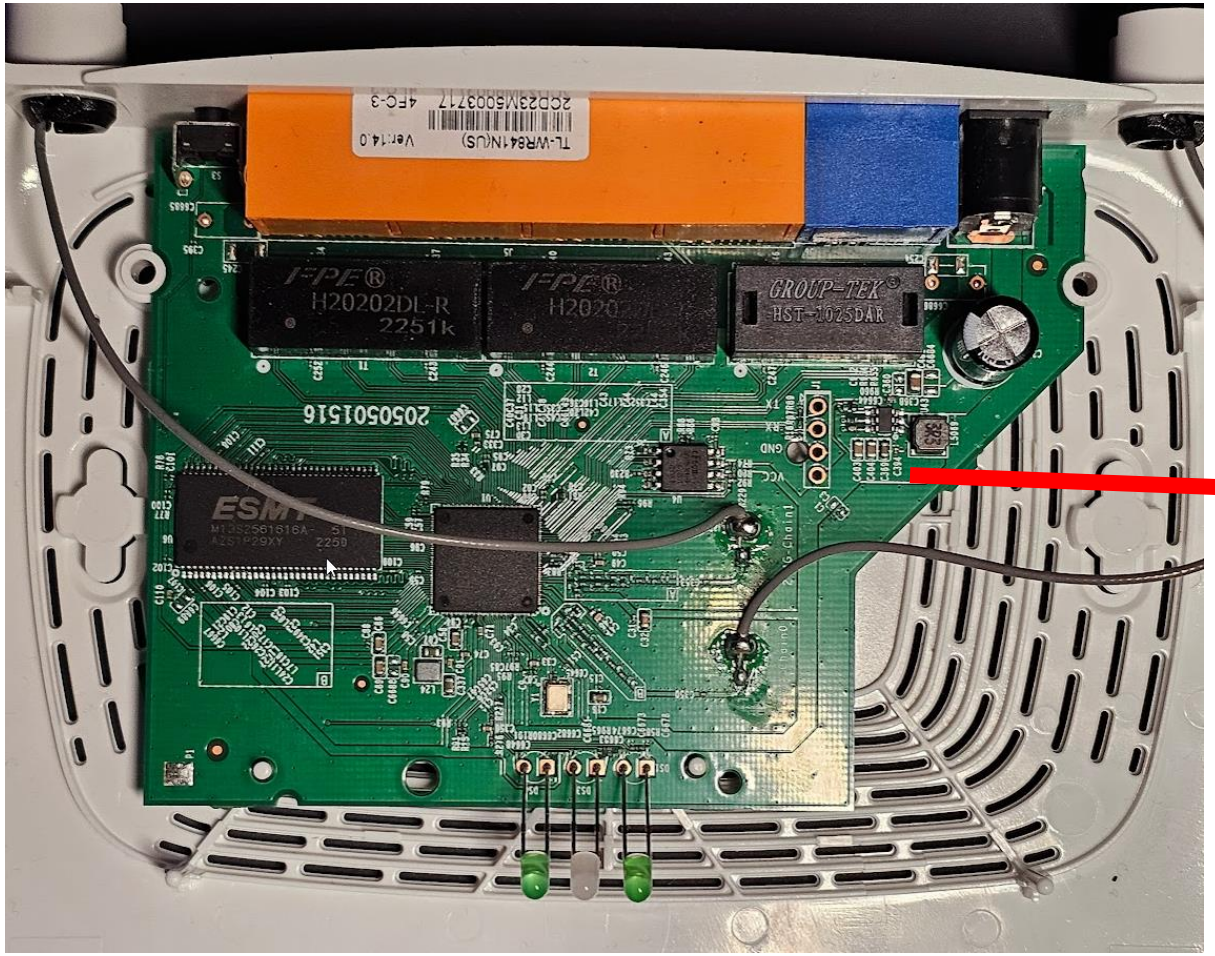
 ##### The CPU freq = 580 MHZ ####
 estimate memory size =32 Mbytes
RESET MT7628 PHY!!!!!!
continue to starting system.
disable switch phyport ...

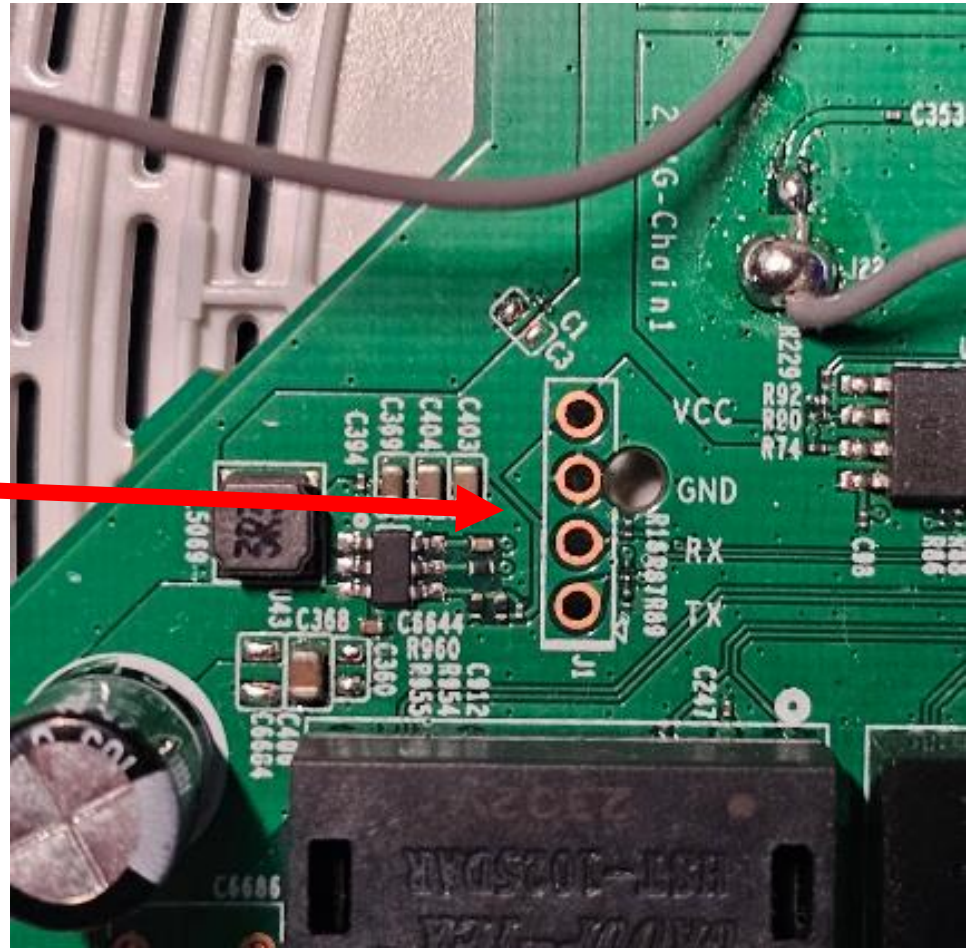
3: System Boot system code via Flash.(0xbc010000)
do_bootm:argc=2, addr=0xbc010000
## Booting image at bc010000 ...
   Uncompressing Kernel Image ... OK
No initrd
## Transferring control to Linux (at address 8000c150) ...
## Giving linux memsize in MB, 32

Starting kernel ...
```

How to identify UART connections

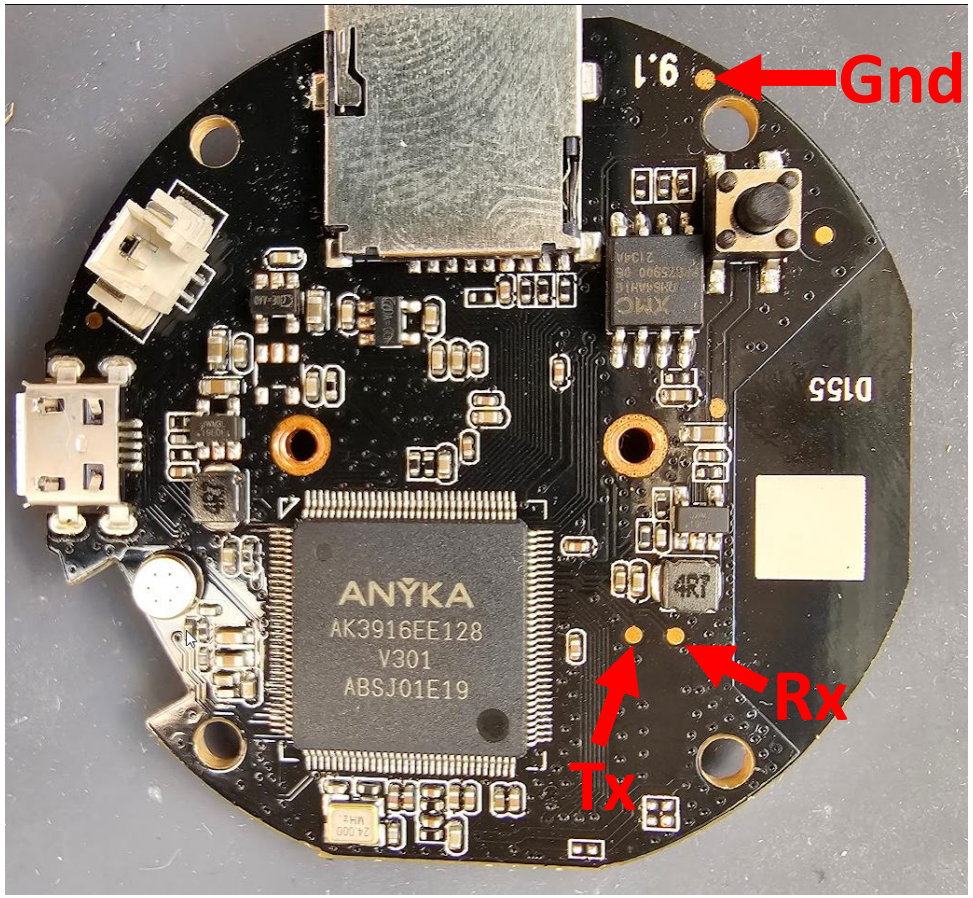
- UART at a minimum needs 3 contacts(Tx, Rx, Gnd)





How to identify UART connections

- They won't always make it this easy

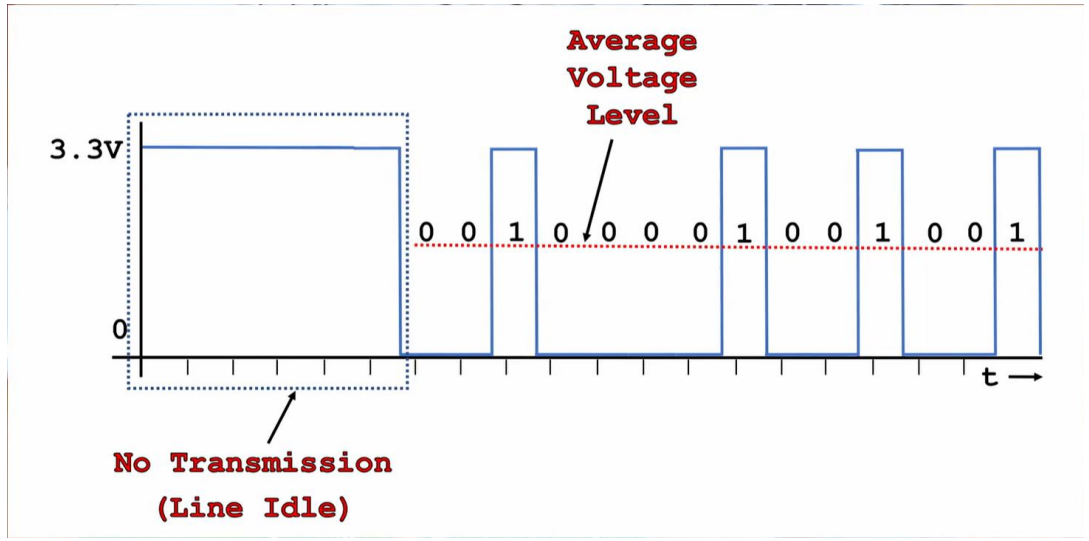




How to identify UART connections

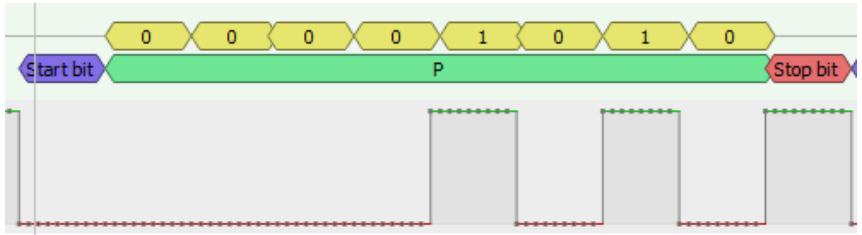
- Use a DMM to test pads/pins
- UART chatty on bootup
- Fluctating voltage on a pin good indication of Tx

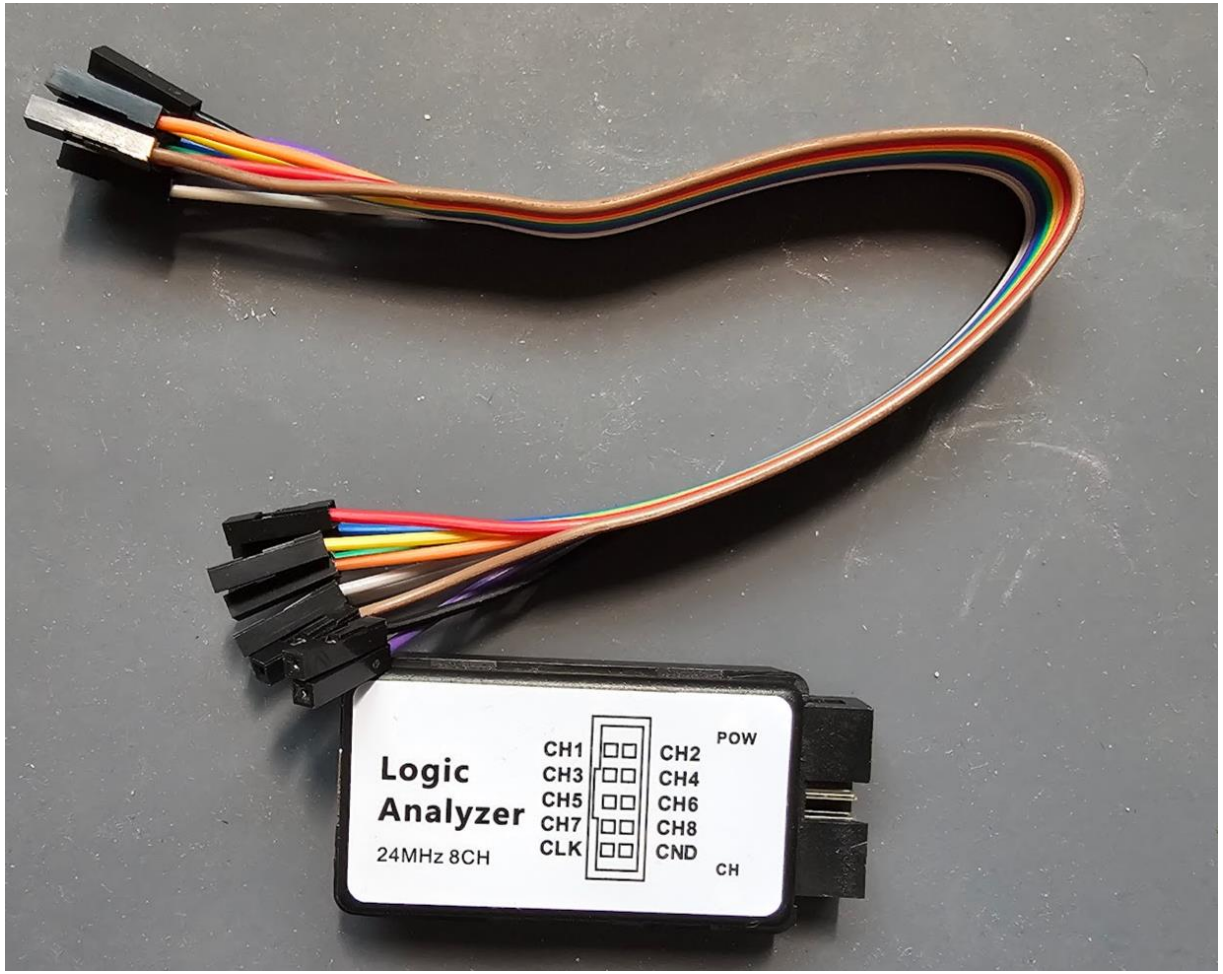




How to identify UART connections

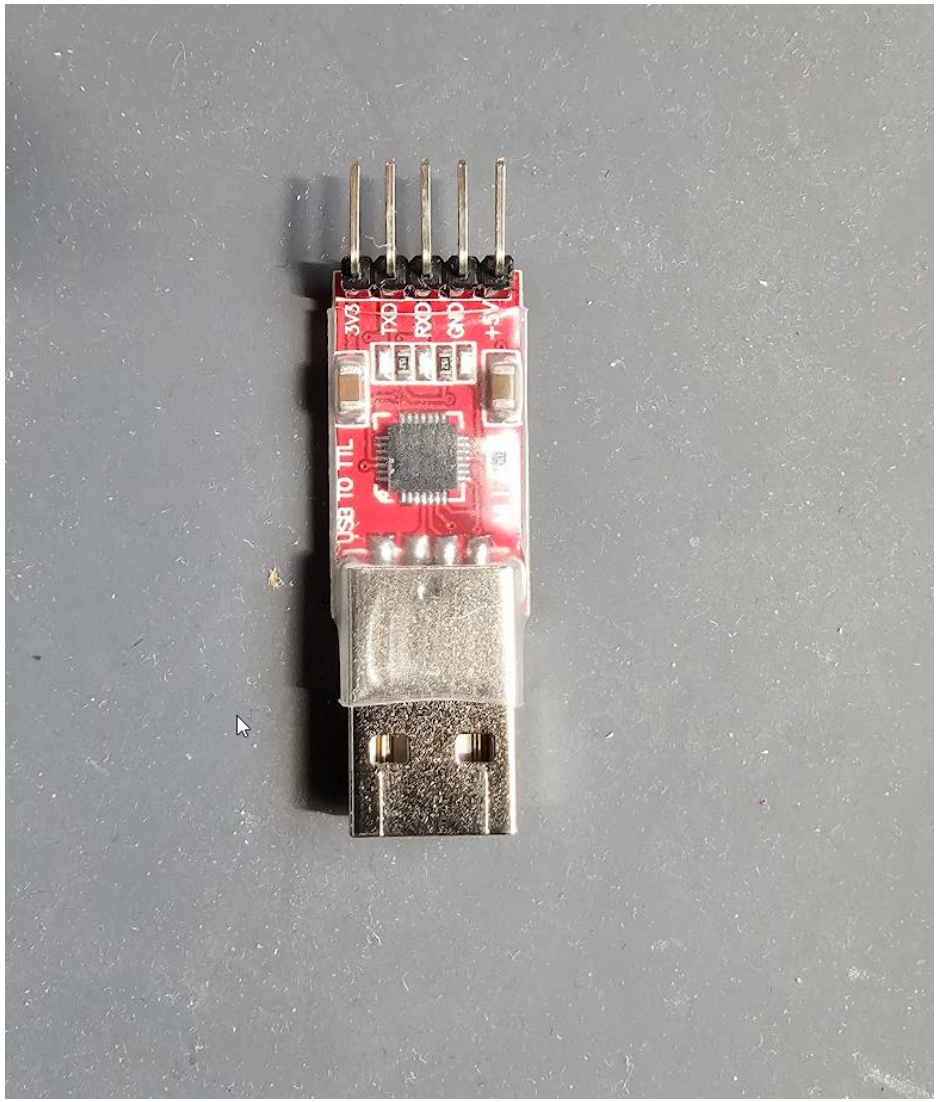
- Can go further to verify with logic analyzer
- Opensource Sigrok Pulseview software



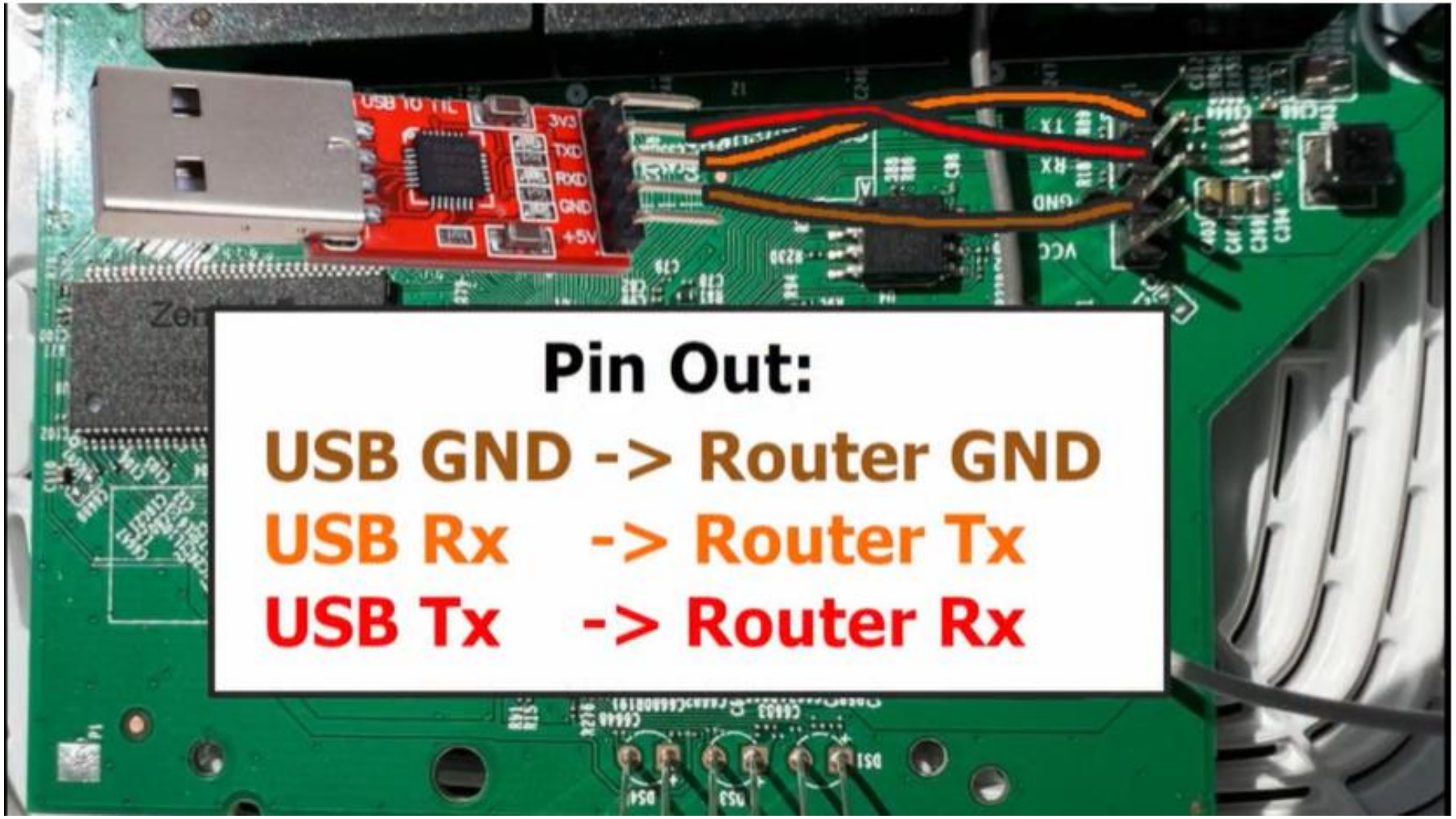


How to connect to it

- UART to USB bridge/adapter to connect to PC
- Rx -> Tx
- Tx -> Rx
- Gnd -> Gnd
- Ignore Vcc (if there is one)

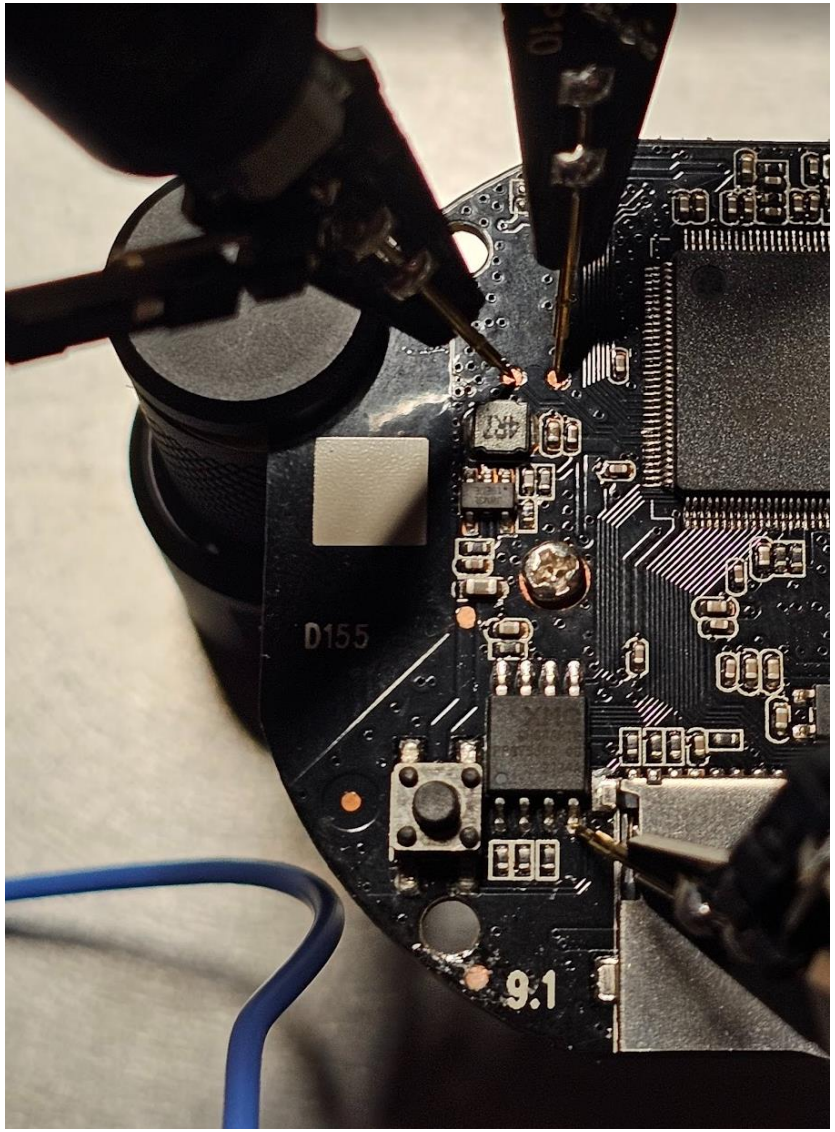


How to connect to it

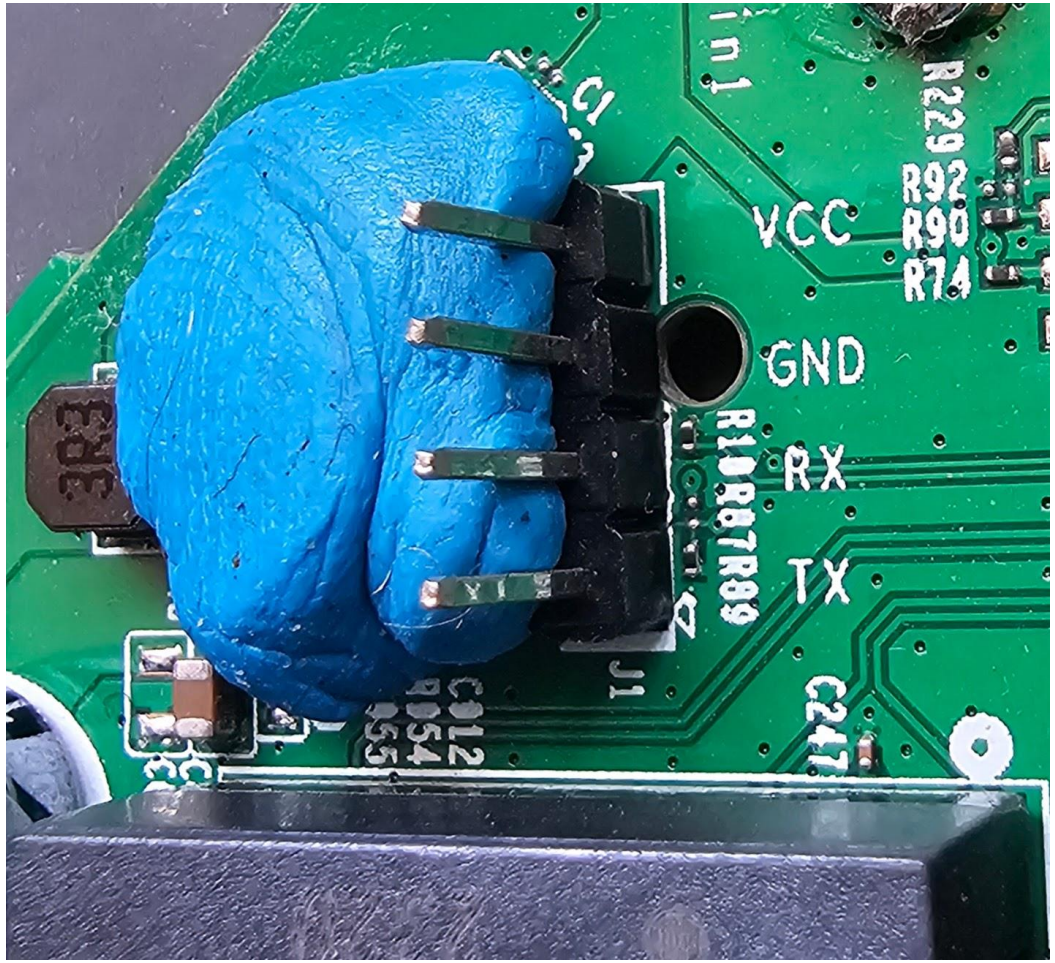


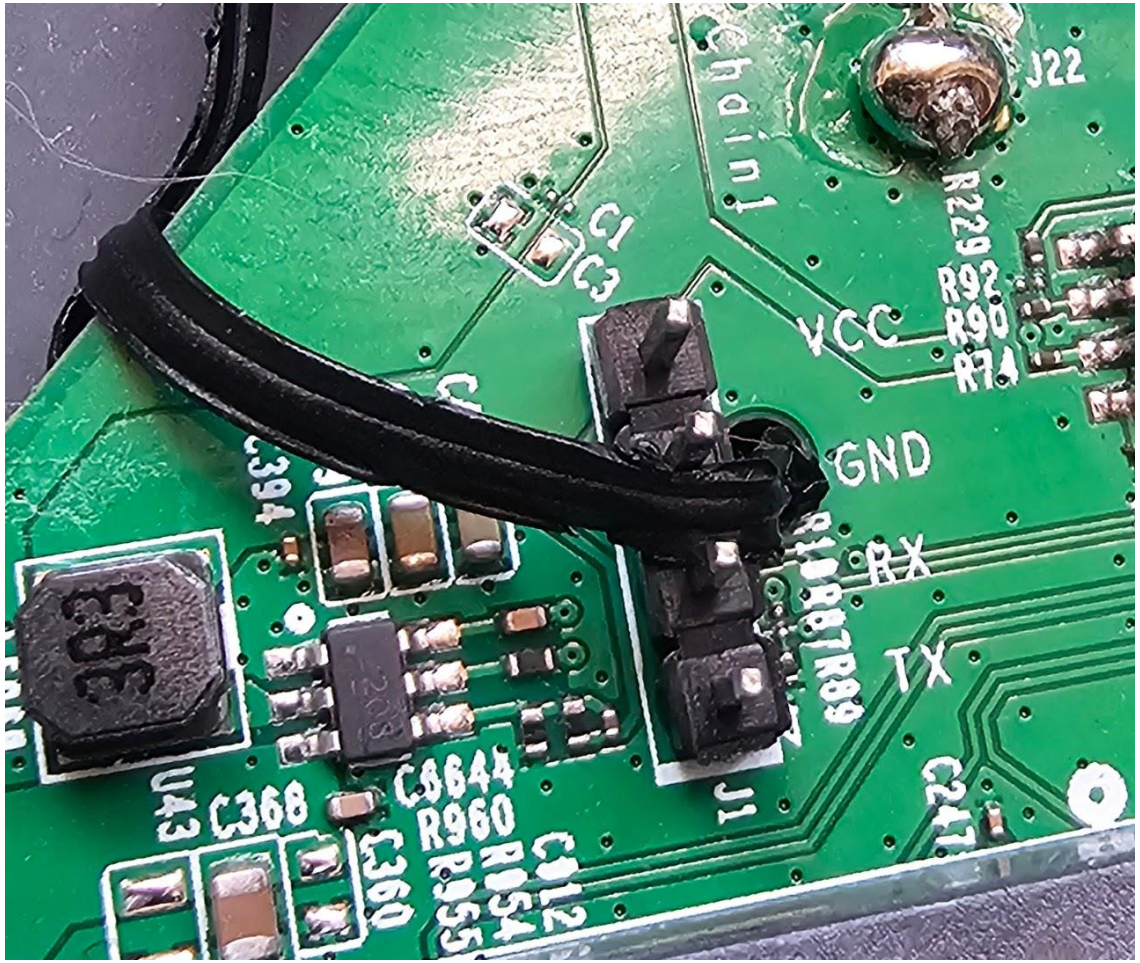
What if I don’t have header pins?

- Best option is to solder
- Attach dupont header pins to through hole
- Use clips, clamps, adapters to connect to test pads



What if I don't have header pins?







Interacting with UART

Steps for a Linux host

- Check UART device id (will be ttyUSB#):

```
ls -l /dev/serial/by-id
```

- Launch a terminal emulator (screen is the easiest):

```
sudo screen /dev/ttyUSB# 115200
```



Now what?

- If all worked should see boot logs on power up
- Hit enter to try for a shell
- Majority of routers and smart cameras run Linux

```
NR_IRQS:128
console [ttyS1] enabled
Calibrating delay loop... 386.04 BogoMIPS (lpj=772096)
pid_max: default: 4096 minimum: 301
Mount-cache hash table entries: 512
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
Switching to clocksource Ralink Systick timer
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 1024 (order: 1, 8192 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
NET: Registered protocol family 1
squashfs: version 4.0 (2009/01/31) Phillip Lougher
msgmni has been set to 57
io scheduler noop registered
io scheduler deadline registered (default)
Ralink gpio driver initialized
Serial: 8250/16550 driver, 2 ports, IRQ sharing enabled
serial8250: ttyS0 at MMIO 0x10000d00 (irq = 21) is a 16550A
serial8250: ttyS1 at MMIO 0x10000c00 (irq = 20) is a 16550A
brd: module loaded
flash manufacture id: 1c, device id 70 16
Warning: un-recognized chip ID, please update SPI driver!
EN25Q32B(1c 30161c30) (4096 Kbytes)
mtd .name = raspi, .size = 0x00400000 (4M) .erasesize = 0x00010000 (64K)
Creating 5 MTD partitions on "raspi":
0x000000000000-0x000000010000 : "boot"
0x000000010000-0x000000100000 : "kernel"
0x000000100000-0x0000003e0000 : "rootfs"
mtd: partition "rootfs" set to be root filesystem
0x0000003e0000-0x0000003f0000 : "config"
0x0000003f0000-0x000000400000 : "radio"
Register flash device:flash0
PPP generic driver version 2.4.2
PPP MPPE Compression module registered
NET: Registered protocol family 24
```

Now what?

Don't discount the value of logging! Especially while interacting with the device!

```
[ util_execSystem ] 141:  prepareDropbear cmd is "dropbearkey -t rsa -f /var/tmp/dropbear/dropbear_rsa_host_key"

Will output 1024 bit rsa secret key to '/var/tmp/dropbear/dropbear_rsa_host_key'
Generating key, this may take a while ...
[ util_execSystem ] 141:  prepareDropbear cmd is "dropbearkey -t dss -f /var/tmp/dropbear/dropbear_dss_host_key"

Will output 1024 bit dss secret key to '/var/tmp/dropbear/dropbear_dss_host_key'
Generating key, this may take a while ...
```

```
[ util_execSystem ] 141:  oal_wlan_ra_setWlanBasicCfg cmd is "iwpriv ra0 set A
utoChannelSel=2"

[ util_execSystem ] 141:  oal_wlan_ra_setWlanBasicCfg cmd is "iwpriv ra0 set C
hannel=0"

[ util_execSystem ] 141:  oal_wlan_ra_setWlanBasicCfg cmd is "iwpriv ra0 set S
SID='i''o''t''h''a''c''k''i''n''g'"

[ util_execSystem ] 141:  oal_wlan_ra_setWlanAdvCfg cmd is "iwpriv ra0 set TxP
ower=100"

[ util_execSystem ] 141:  oal_wlan_ra_setWlanAdvCfg cmd is "iwpriv ra0 set Bea
conPeriod=100"
```

```
spiflash_ioctl_read, Read from 0x003e0000 length 0x10000, ret 0, retlen 0x1000
0
.
spiflash_ioctl_erase, erase to 0x003f0000 length 0x0, nerase done
spiflash_ioctl_write, Write to 0x003e0000 length 0x10000, ret 0, retlen 0x1000
0
T#[ util_execSystem ] 141:  oal_startPing cmd is "ipping -c 1 -s 64 -w 1 192.1
68.0.100 -I 192.168.0.1 &"

[ ippingPacketResultHandler ] 4626:  stat: , time: 0, type: 4, pks: 0, result:
 PING 192.168.0.100 (192.168.0.100): 64 data bytes


PING 192.168.0.100 (192.168.0.100): 64 data bytes
[ ippingPacketResultHandler ] 4626:  stat: None, time: 1281, type: 0, pks: 0,
result: 72 bytes from 192.168.0.100: icmp_seq=0 ttl=64 time=1.281 ms
```



If you also got a shell

- Enumerate, enumerate, enumerate
- Places to start /etc /var
- Can you find passwords, hardcoded keys, endpoints?

```
/var #
/var # /var/tmp/_tools/busybox-mipsel find -name '*.xml'
./tmp/upnpd/gatedesc.xml
./tmp/upnpd/gateicfgSCPD.xml
./tmp/upnpd/gateconnSCPD.xml
./tmp/upnpd/dummy.xml
./tmp/wsc_upnp/WFADeviceDesc.xml
./tmp/wsc_upnp/WFAWLANConfigSCPD.xml
```

```
grep: ./tmp/21: No such device or address
grep: ./tmp/7: No such device or address
grep: ./tmp/6: No such device or address
grep: ./tmp/19: No such device or address
grep: ./tmp/25: No such device or address
./tmp/dconf/cmxdns.conf:password
./tmp/dconf/noipdns.conf:password
./tmp/dconf/dyndns.conf:password
./tmp/dropbear/dropbearpwd:password:62c8ad0a15d9d1ca38d5dee762a16e01
/var #
```

Acquiring Firmware

Firmware is the software running on an embedded device

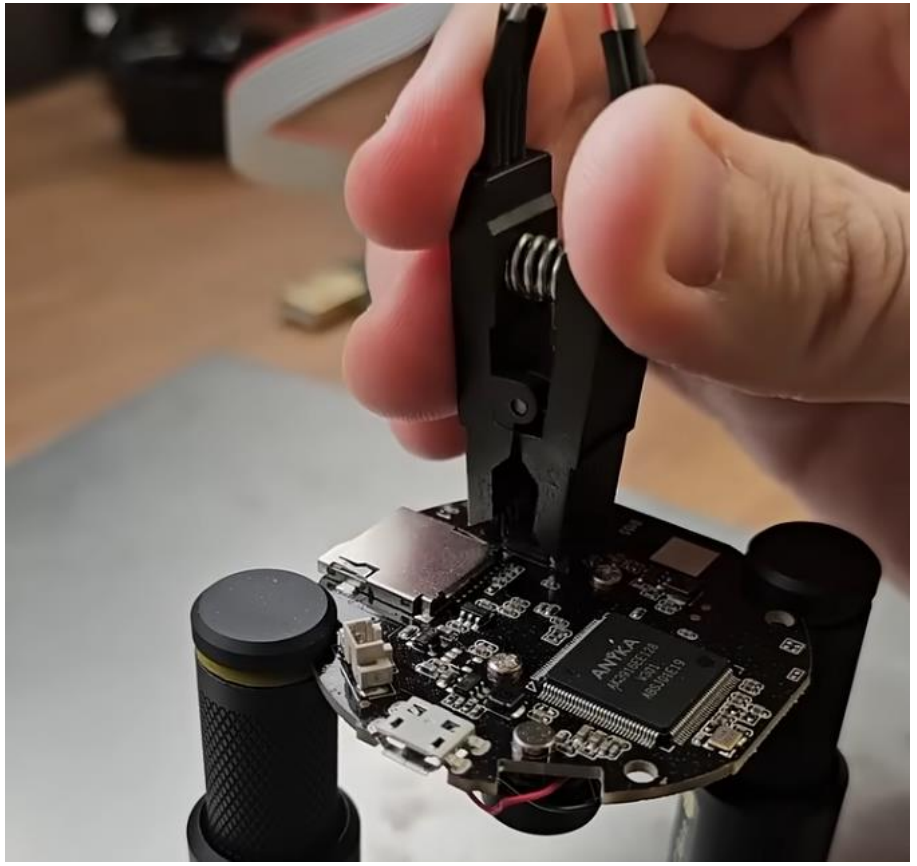
- Embedded Linux is comprised of:
  - bootloader
  - kernel
  - root file system

- Root file system has most of the goodies



Acquiring Firmware

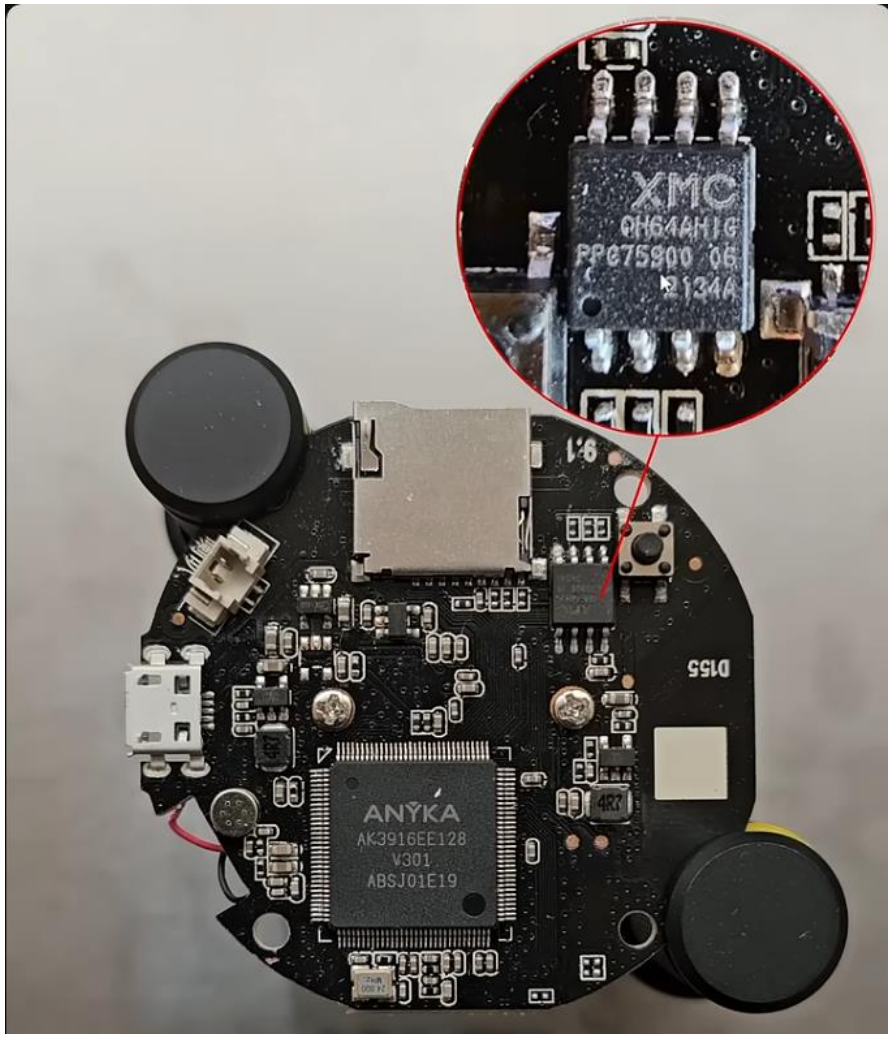
- Easiest method = download from vendor
- Transfer files of interest via UART
- Read from device memory



Acquiring Firmware

Reading SPI Flash Chips

- Use in majority of “cheap” IoT device
- Frequently in SOP8 package
- Readable using flash programmer and flasrhom(FOSS)



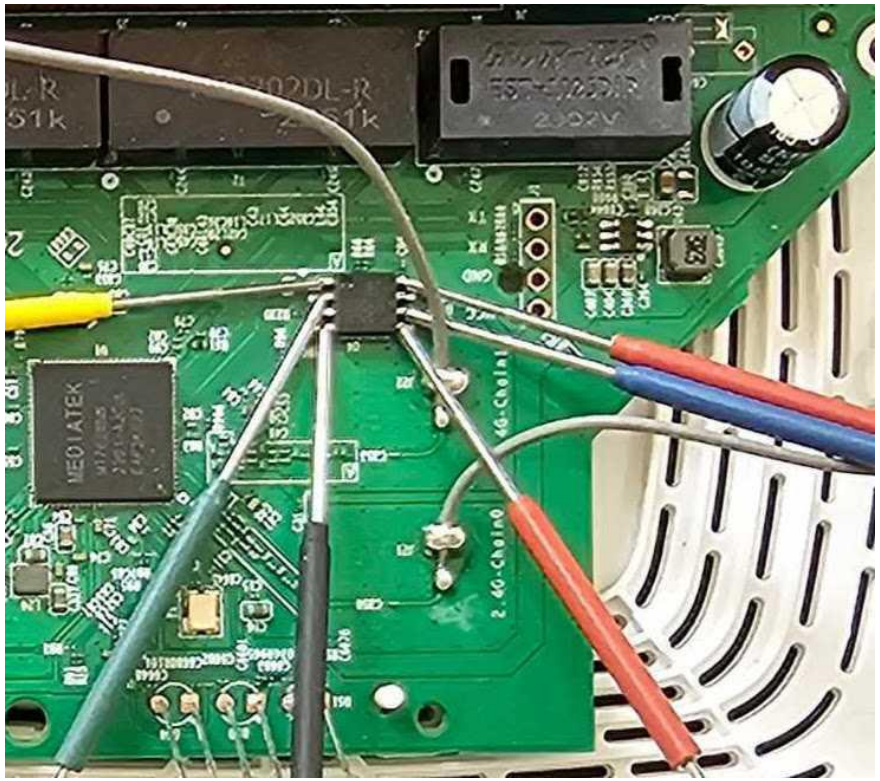
Acquiring Firmware

How to use flashrom

- Flashrom is free and opensource
- With a ch341a programmer:

```
flashrom -p ch341a_spi -r name.bin
```

- Can also use with Tigard



Unpacking firmware

- Cross fingers it’s not encrypted
- Use binwalk to check contents and extract

binwalk -Me firmware_name.bin

```
└─$ binwalk tp_link_wr841n_ext.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
53536         0xD120          U-Boot version string, "U-Boot 1.1.3 (Aug 16 2022 - 12:01:12)"
66048         0x10200         LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed
32 bytes
1048576       0x100000        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3001844 bytes
, blocksize: 262144 bytes, created: 2022-08-16 04:14:58
```



Analyzing firmware

- Now the fun starts!
- Focus on root file system
- Enumerate and look for files of interest
- Low hanging fruit includes config files and scripts

```
find -name "*.extension"
```

```
strings -f * | grep "pass"
```

Does this mean we hacked it?

- Depends on your definition
- Companies may not care about UART or extractable firmware
- Use this leverage to find remotely exploitable vulns

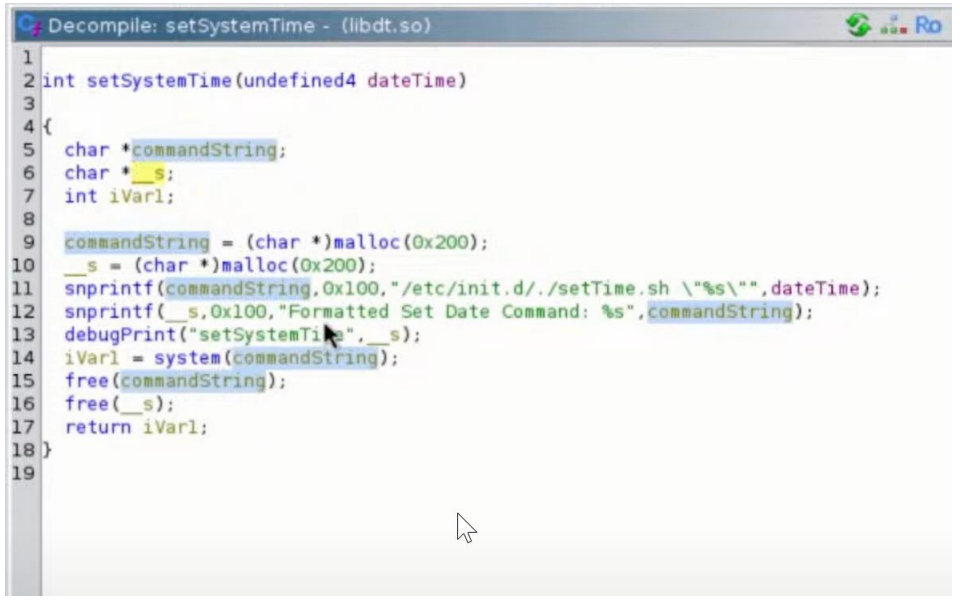




What's next?

The UART shell and Firmware are like an initial foothold

- Use the info found from enumeration to ID key binaries and libraries
- Decompile, reverse engineer to look for vulnerabilities
- Examples: command injection, buffer overflow, auth bypass





Thanks!

Thank You!