Hacking IOT devices using JTAG

HiTB 2024 – Hardware Village

Security Research Labs

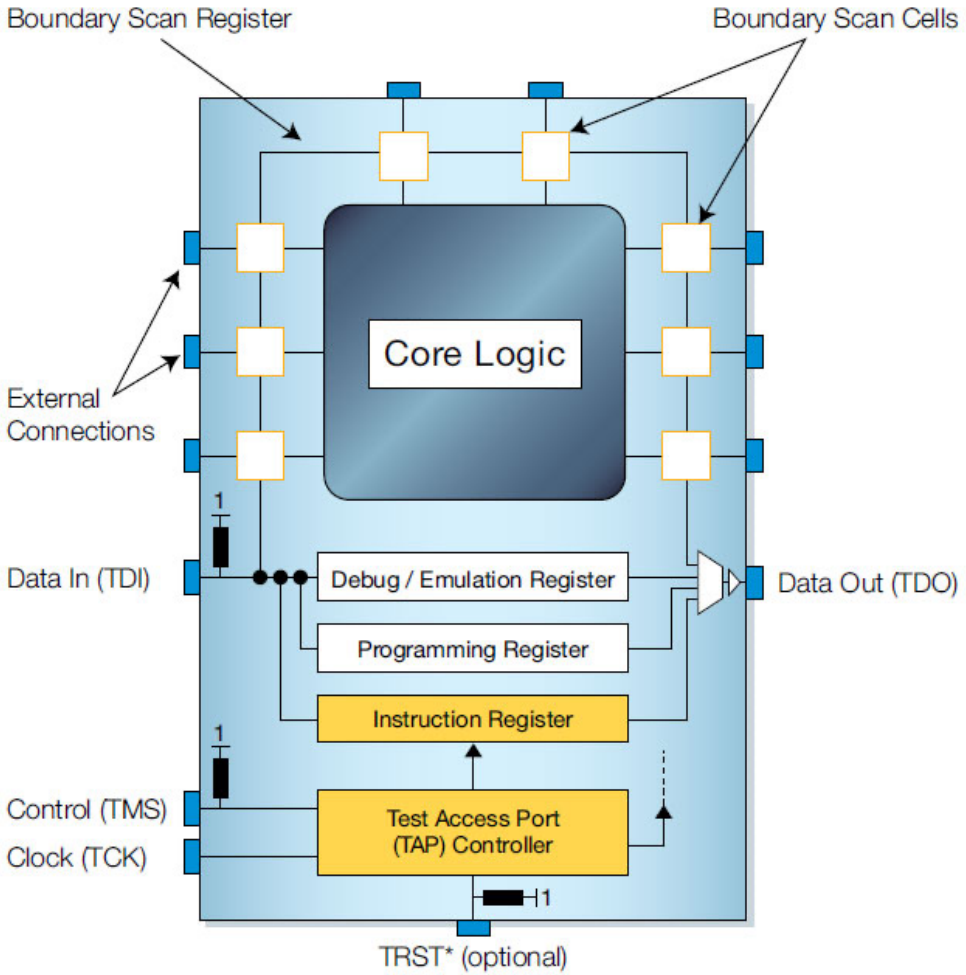
Agenda

1. What is JTAG?
2. Discovering a JTAG interface
3. Identifying JTAG pinout
4. Dumping firmware via JTAG
5. Challenges & Future works

What is JTAG?

- JTAG is the name used for the **IEEE 1149.1** standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports (TAP) used for testing printed circuit boards (PCB) using boundary scan

- Processors often use JTAG **to provide access to their debug/emulation functions** and all FPGAs and CPLDs use JTAG to provide access to their programming functions



Agenda

1. What is JTAG?
2. Discovering a JTAG interface
3. Identifying JTAG pinout
4. Dumping firmware via JTAG
5. Challenges & Future works

JTAG does not have standardized connection pinout. You might want to use a JTAGulator to bruteforce & verify the pinout

Common JTAG interfaces[1]

ARM JTAG header

1	VREF	VSUPPLY	2	
3	nTRST	GND	4	
5	TDI	GND	6	
7	TMS	GND	8	
9	TCK	GND	10	
11	RTCK	GND	12	
13	TDO	GND	14	
15	nSRST	GND	16	
17	DBGRQ	GND	18	
19	DGBACK	GND	20	

Toshiba MIPS JTAG header

1	nTRST	-	2	
3	TDI	GND	4	
5	TDO	GND	6	
7	TMS	GND	8	
9	TCK	GND	10	
11	VREF	GND	12	
13	nSRST	-	14	
15	-	-	16	
17	-	-	18	
19	-	-	20	

ARM JTAG header

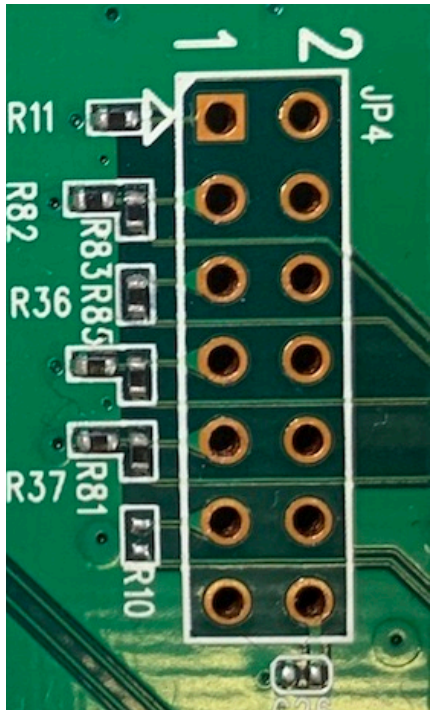
1	VREF	GND	2	
3	nTRST	GND	4	
5	TDI	GND	6	
7	TMS	GND	8	
9	TCK	GND	10	
11	TDO	nSRST	12	
13	VREF	GND	14	

MIPS EJTAG JTAG header

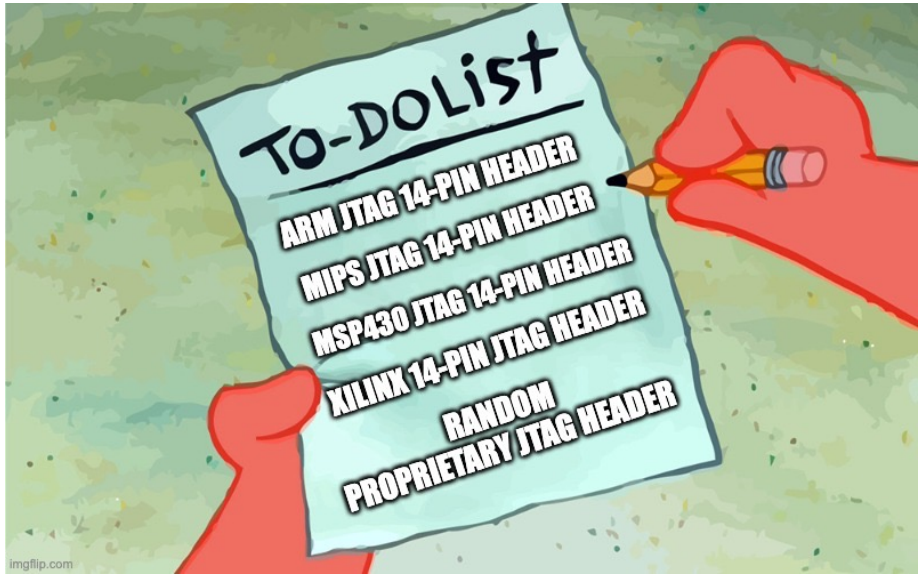
1	nTRST	GND	2	
3	TDI	GND	4	
5	TDO	GND	6	
7	TMS	GND	8	
9	TCK	GND	10	
11	nSRST		12	
13	DINT	VREF	14	

… etc

Real life JTAG interface



- No Pin labels
- No Chip information



[1] http://jtagtest.com/pinouts/

Agenda

1. What is JTAG?
2. Identifying a JTAG interface
3. Identifying JTAG pinout
4. Dumping firmware via JTAG
5. Challenges & Future works

JTAGulator supports IDCODE scan & BYPASS scan to bruteforce the JTAG pinout

What is a JTAGulator?

- JTAGulator is an open-source hardware tool that assists in identifying OCD (On-Chip Debug) interfaces from test points, vias, component pads, or connectors on a target device.

What is a IDCODE scan?

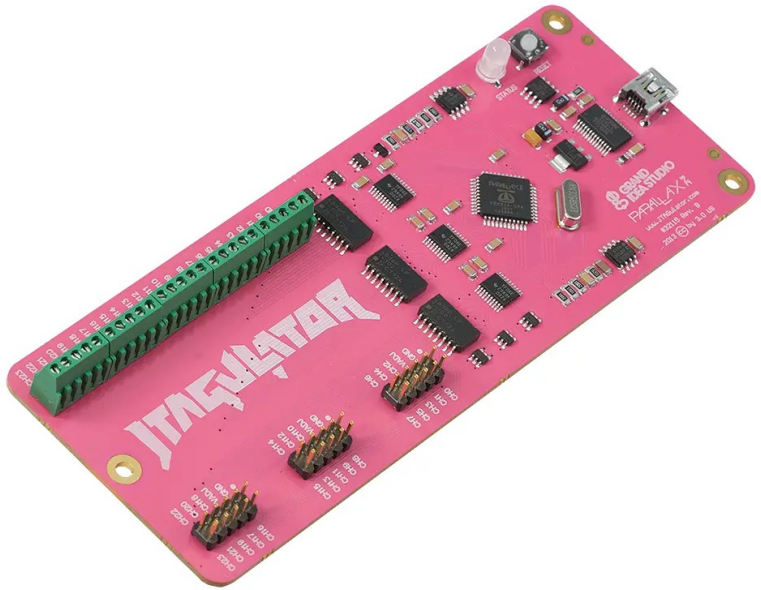
Expected results
- Identified TDO, TCK, TMS pins

Under the hood
- JTAGulator continuously send IDCODE command over the assumed pinout, until device identifier information is returned

What is a BYPASS scan?

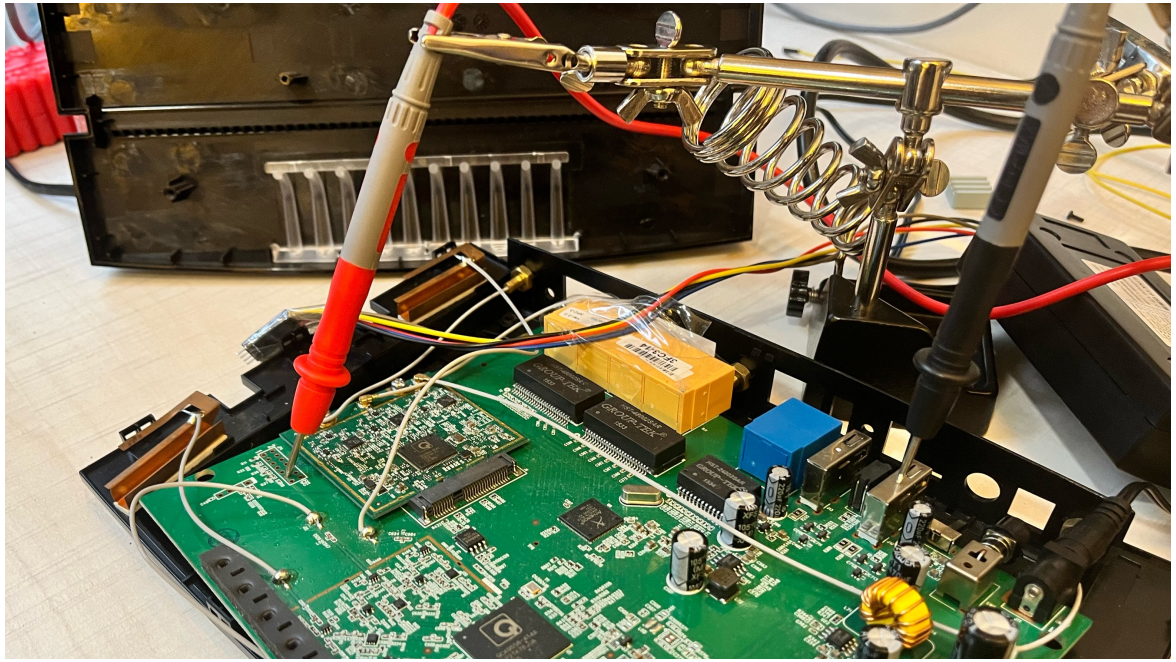
Expected results
- Identified TDI, TDO, TCK, TMS pins

Under the hood
- JTAGulator continuously send BYPASS command to the assumed pinout, until the same data input into TDI returned from TDO



Identifying pinout with JTAGulator (Part 1) – Identify ground pin(s) using multimeter

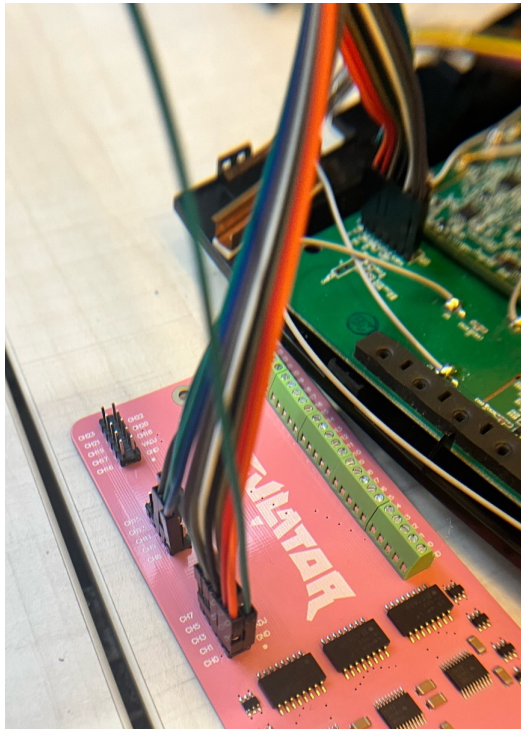
Pre-requisites	▪ Multi-meter ▪ Physical access to the target device
Step 1	Power off the target device for: - safe test - accurate measurement - avoiding damage to circuit board & meter

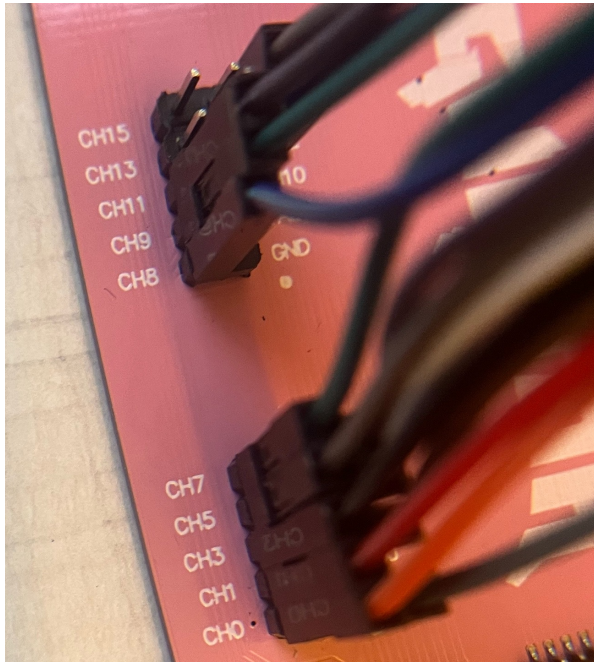


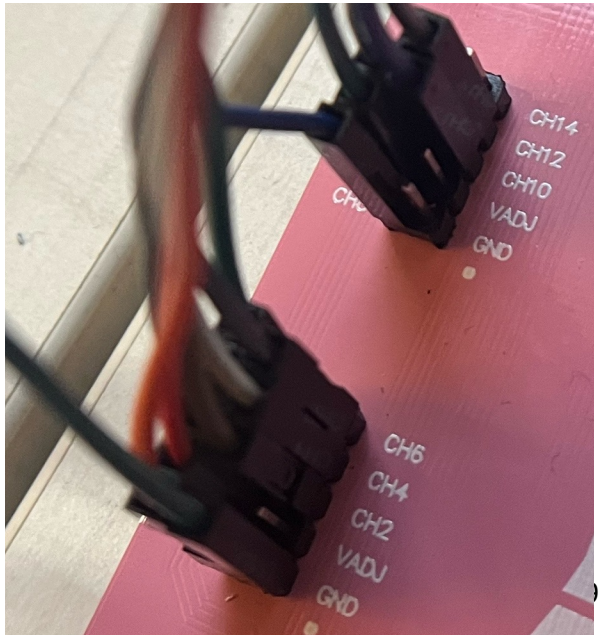
Step 2	Touch the black probe on a known ground (USB port case, power button case)
Step 3	Touch the red probe with each JTAG pin - the ones which form a circuit in the **continuity test** is the GND pin (give beep sound)

Identifying pinout with JTAGulator (Part 2) – Connect the pins with JTAGulator

Pre-requisites	▪ JTAGulator ▪ Jump wires
Step 1	Connect **one GND from the device** to JTAGulator's GND
Step 2	Connect the rest of the device's pins to the **channel pins** of JTAGulator **Make sure you can trace back the channel pins to the pins on the device







Identifying pinout with JTAGulator (Part 3) – Run IDCODE scan on JTAGulator

Pre-requisites	▪ JTAGulator ▪ Laptops (Ubuntu 20.04 or above is preferred)
Step 1	Connect the JTAGulator to laptop over USB cable - open console access to JTAGulator via command: **screen /dev/USBtty0 115200**
Step 2	Start IDCODE scan first with below commands - enter JTAG mode with "**J**" - set target device's voltage level with "**V**" - initiate IDCODE with "**I**" - configure bruteforce channels (channel 0-12 connected in previous part)
Step 3	Wait for the results of TDO, TCK, TMS

```
JTAG> I
Enter starting channel [0]:
Enter ending channel [7]:
Possible permutations: 336

Bring channels LOW before each permutation? [Y/n]:
Enter length of time for channels to remain LOW (in ms, 1 - 1000) [100]:
Enter length of time after channels return HIGH before proceeding (in ms, 1 - 1000) [100]:
Press spacebar to begin (any other key besides Enter to abort)...
JTAGulating! Press any key to abort...
----------------------------------------------------------------------------------------------------
----------------------------------------
TDI: N/A
TDO: 4
TCK: 2
TMS: 3
Device ID #1: 0000 0000000000000000 00000000000 1 (0x00000001)

----------------------------------------------------------------------------------------------------
------------
IDCODE scan complete.

JTAG>
```

Identifying pinout with JTAGulator (Part 4) – Run BYPASS scan on JTAGulator

Pre-requisites

- JTAGulator
- Laptops (Ubuntu 20.04 or above is preferred)

Step 1

Start BYPASS scan with below commands
- enter JTAG mode with “J”
- set target device’s voltage level with “V”
- initiate BYPASS scan with “B”
- configure bruteforce channels
(channel 0-12 connected in previous part)
- configure known pins to speed up the scan

Step 2

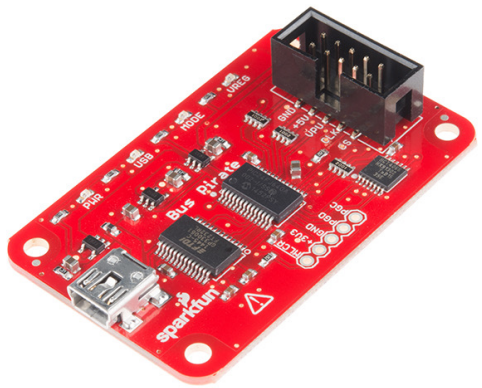
Wait for the results of TDI, TDO, TMS, TCK

```
JTAG> B
Enter starting channel [0]:
Enter ending channel [12]:
Are any pins already known? [Y/n]:
Enter X for any unknown pin.
Enter TDI pin [0]: X
Enter TDO pin [4]:
Enter TCK pin [2]:
Enter TMS pin [3]:
Possible permutations: 10

Bring channels LOW before each permutation? [Y/n]:
Enter length of time for channels to remain LOW (in ms, 1 - 1000) [100]:
Enter length of time after channels return HIGH before proceeding (in ms, 1 - 1000) [100]:
Press spacebar to begin (any other key besides Enter to abort)...
JTAGulating! Press any key to abort...
--
TDI: 5
TDO: 4
TCK: 2
TMS: 3
Number of devices detected: 1
--------
BYPASS scan complete.

JTAG>
```

Agenda

1. What is JTAG?
2. Identifying a JTAG interface
3. Identifying JTAG pinout
4. Dumping firmware via JTAG
5. Challenges & Future works

Connecting to JTAG



Bus Pirate

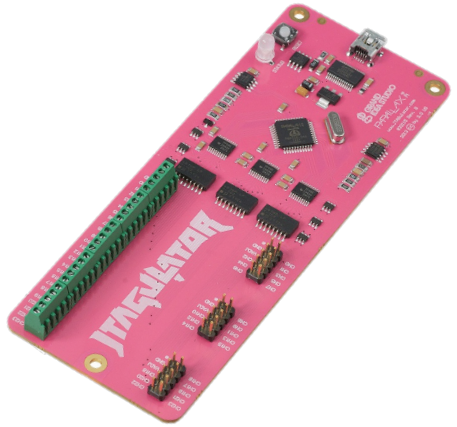
- **Price:** 30 – 60 USD
- **Pros:**
  - Support wide range of protocols (JTAG, SPI, I2C, UART, etc)
- **Cons:**
  - Relatively slow compared to dedicated JTAG debuggers

ST-Link v2

- **Price:** 13 – 46 USD
- **Pros:**
  - Cost Effective for JTAG debugging
- **Cons:**
  - Only compatible with STM32 devices (most cases)

J-LINK v9

- **Price:** ~450 USD
- **Pros:**
  - High performance with wide compatibility
- **Cons:**
  - Expensive for high-end version, e.g., commercial



JTAGulator

- **Price:** 90 - 200 USD
- **Pros:**
  - Specializes in identifying JTAG/ UART pinouts
- **Cons:**
  - Limited functions in JTAG debugging

Extracting firmware over JTAG (Part 1) – Upgrade the bus pirate version for JTAG & OpenOCD

Pre-requisites	▪ Bus Pirate v3.x ▪ Bus Pirate firmware version 6.1 r1676/ 6.0 r1625 ▪ Laptops (Ubuntu 20.04 or above is preferred)
Step 1	**## Download the firmware repository** git clone https://github.com/DangerousPrototypes/Bus_Pirate.git cd Bus_Pirate-master/BPv3-bootloader/pirate-loader
Step 2	**## Enter the bus pirate** screen /dev/ttyUSB0 115200 **## Enter bootloader mode in the bus pirate** $ **## Kill the screen session to prevent the connection occupied** Press the keys: Ctrl + A, k, y
Step 3	**## Flash the new firmware v6.1** ./pirate-loader_lnx --dev=/dev/ttyUBS0 --hex=../../package/BPv3-firmware/old-versions/BPv3-frimware-v6.1.hex
Step 4	**## Verify the firmware version in the bus pirate terminal** i

```
HiZ>$
Are you sure? y
BOOTLOADER
```

```
HiZ>i
Bus Pirate v3.5
Firmware v6.1 r1676  Bootloader v4.4
DEVID:0x0447 REVID:0x3046 (24FJ64GA002 B8)
http://dangerousprototypes.com
HiZ>
```

Extracting firmware over JTAG (Part 2) – Connect the bus pirate to the JTAG interface

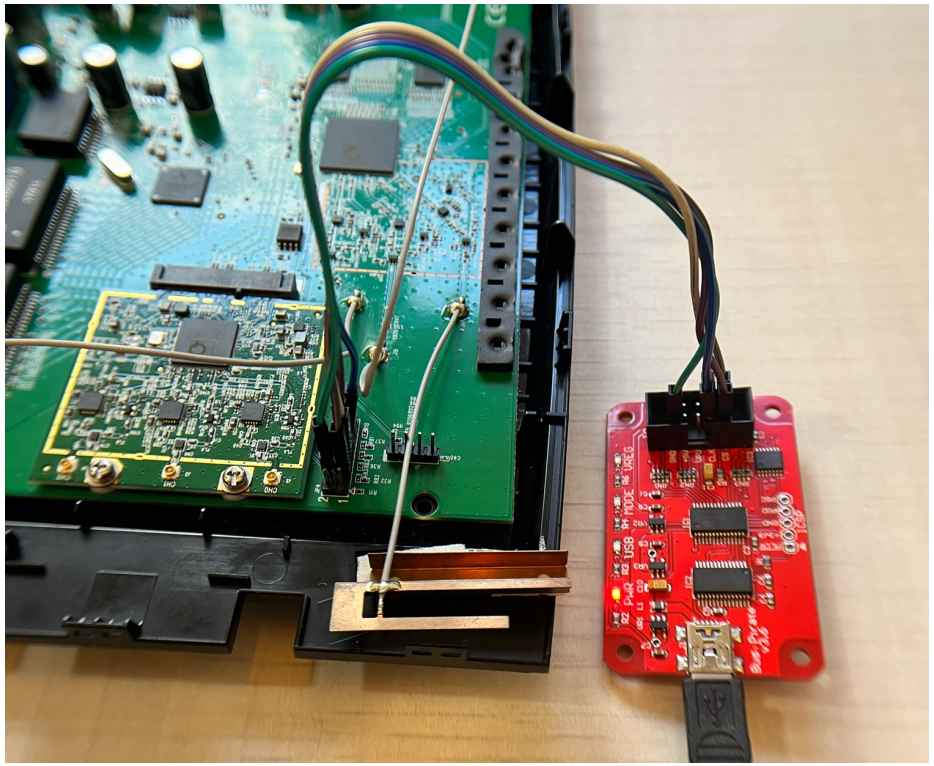
Step 1

Connect the Pins as identified:
Bus Pirate <-> Target JTAG interface

TDI (MOSI) <-> TDI
TCK (CLK) <-> TCK
TMS (CS) <-> TMS
TDO (MISO) <-> TDO
GND <-> GND

Bus pirate pin translation table

Mode	MOSI	CLK	MISO	CS
HiZ				
1-Wire	OWD			
UART	TX		RX	
I2C	SDA	SCL		
SPI	MOSI	CLOCK	MISO	CS
JTAG	TDI	TCK	TDO	TMS



Extracting firmware over JTAG (Part 3) – Connect to the JTAG using openOCD

Step 1	**## Install openocd, if not** sudo apt get install libtool autoconf texinfo libusb-dev libftdi-dev screen -y git clone git://git.code.sf.net/p/openocd/code ./bootstrap ./configure --enable-maintainer-mode --disable-werror --enable-buspirate
Step 2	**## Open JTAG connection via Openocd** **## openocd config in next slide as appendix** openocd -f MyBuspirate -f ath79.cfg

```
srlabs@srlabs:~/Workspace/device-testing/tp-link-ac1750v2$ openocd -f MyBuspirate.cfg -f /usr/local/share/openocd/scripts/interface/ath79.cfg
Open On-Chip Debugger 0.12.0+dev-00663-g16c114c05 (2024-08-26-16:03)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
jtag
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : Buspirate JTAG Interface ready!
Info : Note: The adapter "buspirate" doesn't support configurable speed
Info : JTAG tap: ath79.cpu tap/device found: 0x00000001 (mfg: 0x000 (<invalid>), part: 0x0000, ver: 0x0)
Info : [ath79.cpu] Examination succeed
Info : [ath79.cpu] starting gdb server on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'telnet' connection on tcp/4444
```

Appendix – Extracting firmware over JTAG (Part 3) – openOCD config

ath79.cfg [1]

```
# Atheros ATH79 MIPS SoC.
# tested on AP83 and AP99 reference board
#
# source: https://forum.openwrt.org/viewtopic.php?pid=297299#p297299

if { [info exists CHIPNAME] } {
  set _CHIPNAME $CHIPNAME
} else {
  set _CHIPNAME ath79
}

if { [info exists ENDIAN] } {
  set _ENDIAN $ENDIAN
} else {
  set _ENDIAN big
}

if { [info exists CPUTAPID] } {
  set _CPUTAPID $CPUTAPID
} else {
  set _CPUTAPID 0x00000001
}

jtag_ntrst_assert_width 200
jtag_ntrst_delay 1

reset_config trst_only

jtag newtap $_CHIPNAME cpu -irlen 5 -ircapture 0x1 -irmask 0x1f -expected-id $_CPUTAPID

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME mips_m4k -endian $_ENDIAN -chain-position $_TARGETNAME

$_TARGETNAME configure -event reset-init {
                        # disable flash remap
                        mww 0xbf000004 0x43
}

# serial SPI capable flash
# flash bank <driver> <base> <size> <chip_width> <bus_width>
set _FLASHNAME $_CHIPNAME.flash
flash bank $_FLASHNAME ath79 0xbf000000 0x01000000 0 0 $_TARGETNAME
```

MyBuspirate.cfg

```
source [find /usr/local/share/openocd/scripts/interface/buspirate.cfg]
buspirate port /dev/ttyUSB0
Transport select jtag
```

[1] https://openwrt.org/docs/guide-user/hardware/debrick.ath79.using.jtag#ath79cfg

Extracting firmware over JTAG (Part 4) – Halt the device before firmware extraction

Step 1

Telnet to openocd server
telnet localhost 4444
check the target chip's state
targets

```
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* ath79.cpu          mips_m4k   big    ath79.cpu          running
```

Step 2

Spam the halt command to halt the CPU
halt
Pause until you see two halt implemented and the state is steadily halted from "targets" output

- Sometimes you can only halt the CPU in the beginning of the system boot up

- Half-successful halt will trigger system reboot. Not a permanent halt

```
> halt
processor id not available, failed to read cp0 PRId register
isa info not available, failed to read cp0 config register: 0
target halted in MIPS32 mode due to debug-request, pc: 0x80109254
> halt
ISA implemented: MIPS32, MIPS16, release 2(AR=1)
DSP implemented: yes, rev 2
FPU implemented: no
target halted in MIPS32 mode due to debug-request, pc: 0x80109254
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> halt
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* ath79.cpu          mips_m4k   big    ath79.cpu          halted
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* ath79.cpu          mips_m4k   big    ath79.cpu          halted
```

Extracting firmware over JTAG (Part 5) – Dump the firmware

Step 1	**## Identify the flash memory location & size [1]** flash banks
Step 2	**## Identify the flash chip device name (optional)** flash probe 0
Step 3	**## Extract the firmware based on the offset identified in step 1** dump_image <output filename> 0xbf000000 0x0100000

```
> flash banks
#0 : ath79.flash (ath79) at 0xbf000000, size 0x01000000, buswidth 0, chipwidth 0
> flash probe 0
Found flash device 'win w25q128fv/jv' (ID 0x001840ef)
flash 'ath79' found at 0xbf000000
> dump_image ac1750v2_firmware.bin 0xbf000000 0x01000000
No working memory available. Specify -work-area-phys to target.
not enough working area available(requested 128)
No working area available
Falling back to non-bulk read
```

Step 4	**## Verify the image is being dumped**

```
srlabs@srlabs:~/Workspace/device-testing/tp-link-ac1750v2$ while true; do ls -l ac1750v2_firmware.bin ;sleep 60; done
-rw-rw-r-- 1 srlabs srlabs 0 Aug 26 16:40 ac1750v2_firmware.bin
-rw-rw-r-- 1 srlabs srlabs 0 Aug 26 16:40 ac1750v2_firmware.bin
-rw-rw-r-- 1 srlabs srlabs 4096 Aug 26 16:42 ac1750v2_firmware.bin
-rw-rw-r-- 1 srlabs srlabs 4096 Aug 26 16:42 ac1750v2_firmware.bin
```

Footnotes [1]

- Flash banks are pre-defined in the target chip's config file (ath79.cfg)
- Without the config, it will not work ☹

Agenda

1. What is JTAG?
2. Identifying a JTAG interface
3. Identifying JTAG pinout
4. Dumping firmware via JTAG
5. Challenges & Future works

The journey of JTAG hacking doesn't end here yet… Some challenges need more effort to work around

Challenge	Impact	Why this challenge happens?
Obtain chip specification (data sheet)	▪ **Unable to craft openocd config** -> Cannot connect the JTAG interface via OpenOCD ▪ Chip's details are in the datasheet, e.g. bus width, chip width, memory mapping	▪ Manufacturer does not publish all their chip's specification

Potential workaround

- An automotive brute-force approach to identify chip specification (See next slide)

The essential components of OpenOCD config files are bruteforceable

Essential config components

- **Bytes order**
  - Big-endian vs Little-endian
- **CPUTAPID**
  - Retrievable from IDCODE scan
- **Instruction Register Length**
  - Common configuration based on chip architecture, e.g.
  - 4-bit for simple devices, 5-bit for Cortex-M, 7-bit for ARM cores, etc
- **Flash memory offset**
  - Common values based on flash driver – readily available on OpenOCD page, e.g.
  - ath79: 0xbf000000/ 0x10000000/ 0x20000000; mrvlqspi: 0x46010000; etc
- **Flash memory size**
  - identifiable via physical inspection on the flash chip (refer to our SPI flash hacking!!)

Hypothetical automation steps

1. Populate the config files for bruteforce
2. Spawning openocd with new config files
3. Validate config by some means (to be researched)
4. Extract firmware with the few potential legitimate config
5. Extract and search file system from the firmwares

To answer after implementation & research

- Any essential components forgotten?
- Will the bruteforce time be acceptable?