

# Funderbolt

Adventures in Thunderbolt DMA Attacks

Russ Sevinsky



# Background

- Thunderbolt
  - Apple and Intel collaboration
  - Expansion port
  - PCI Express (PCIe) and DisplayPort using the same port
- DMA
  - Direct Memory Access
  - Processor becomes bottleneck for high-speed transfers
  - Lets devices read and write directly to RAM



- Why external buses matter for security experts?
  - Digital Forensics
    - Getting data to solve a mystery
  - User protection
    - So RAM contents can be safe
  - Sneaky DRM
    - Bus encryption



# Background

- Current DMA research
  - I/O Attacks in Intel-PC Architectures and Countermeasures<sup>1</sup>
  - Understanding DMA Malware<sup>2</sup>
- Current Thunderbolt attacks?
  - Daisy chaining Thunderbolt and Firewire<sup>3</sup>
  - Inception<sup>4</sup>
  - De Mysteriis Dom Jobsivs (Mac EFI Rootkits)<sup>5</sup>



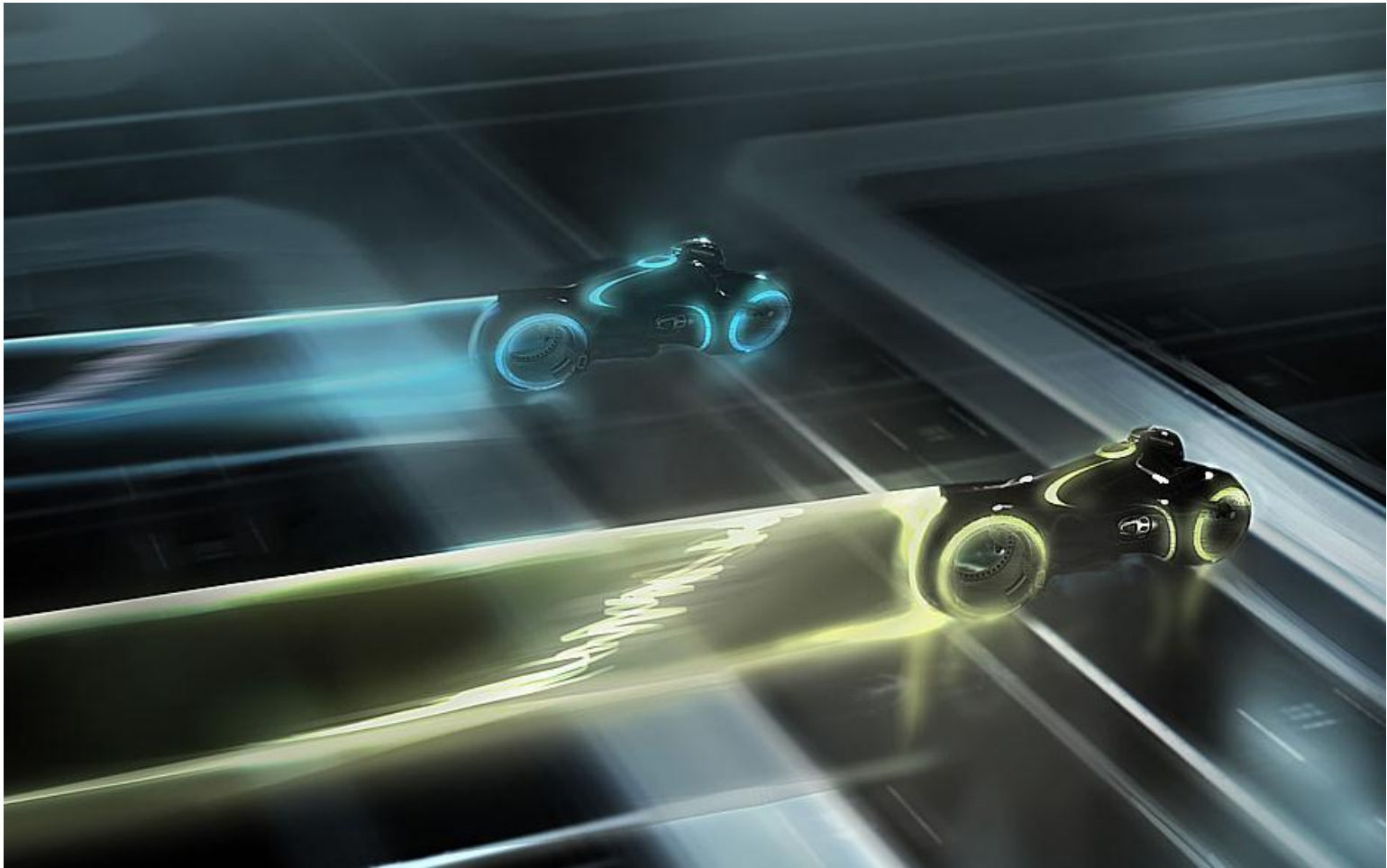
- Mitigations?
  - Epoxy (really?)
  - Input/Output Memory Management Units (IOMMUs)
    - Maps physical memory addresses to logical addresses
    - Think “VM for DMA”
    - Prevents devices from requesting physical addresses directly
    - Currently not implemented on Apple hardware
  - Secure Configurations<sup>6</sup>
    - Prevents attacks when computer is locked
    - Boot attacks and social engineering still possible



# Background



# A Trip Down Memory Lanes



# A Trip Down Memory Lanes

- PCI Express (PCIe)
  - High-speed serial bus
  - Data sent via “lanes”
  - A lane is made up of differential wire pairs
    - One + and one – wire
    - Helps to reduce noise
  - One lane (x1) is made up of two differential pairs
    - Transmit pair (PET)
    - Receive pair (PER)





# A Trip Down Memory Lanes

- PCIe (cont)
  - Four lanes (x4) has eight wires, etc...
  - All lanes CAN use another differential pair for clock
    - REFCLK
    - Not required (could use a XTAL)
  - So per spec, x1 only needs 4 wires for data communication!
    - PET and PER
  - Typical x1 setups use a REFCLK



# A Trip Down Memory Lanes

- PCIe (cont)
  - PCIe device types
    - Endpoints (Legacy and Native)
    - Switch/Bridge
    - Root Complex
  - Legacy endpoints
    - More permissive than Native endpoints for backwards compatibility
    - Supports “Locked” transactions, IO transactions and 32-bit addressing only

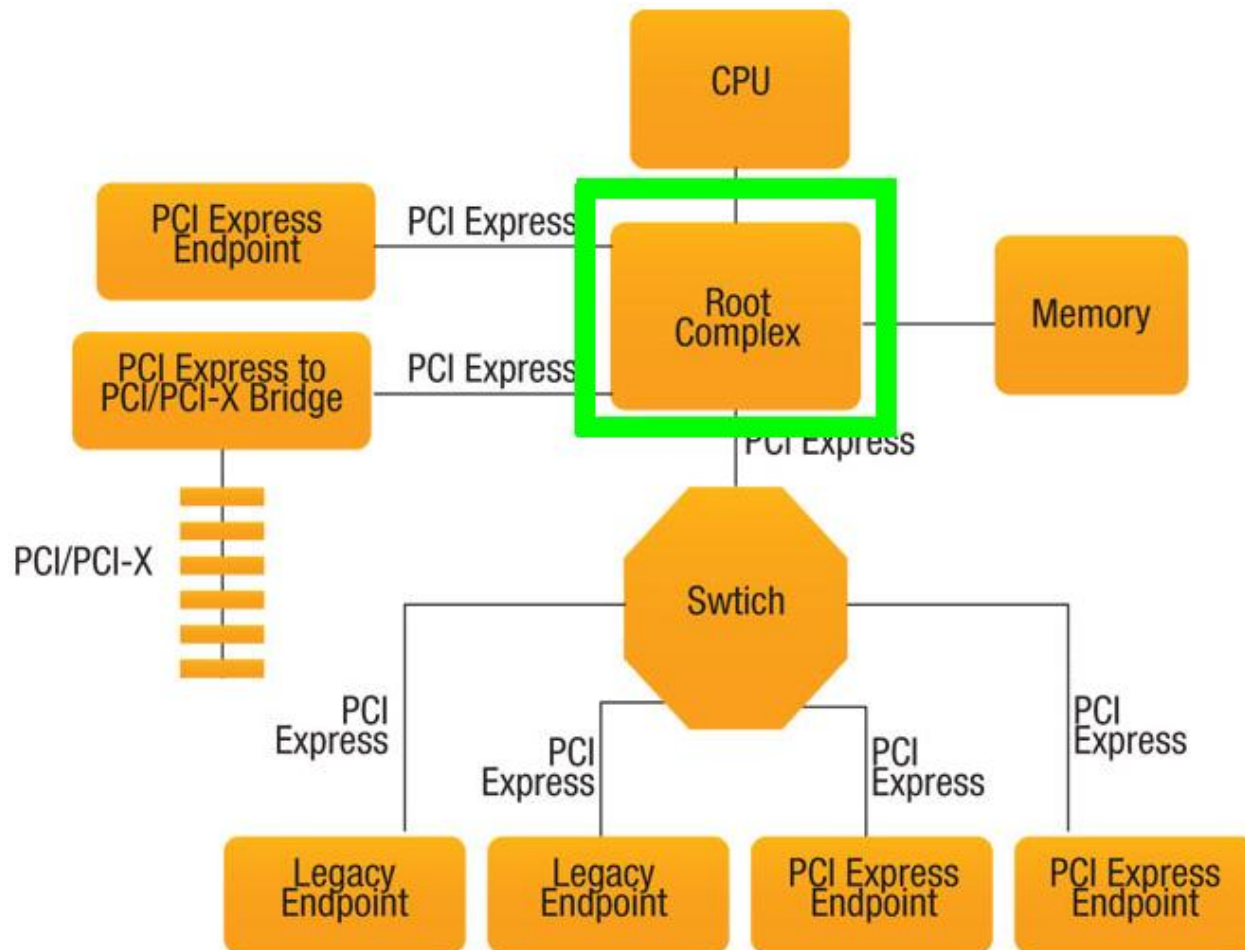


# A Trip Down Memory Lanes

- PCIe (cont)
  - Data sent via “packets” called Transaction Layer Packet (TLP)
  - Point-to-point topology
    - No arbitration
    - Devices do not need to be “granted” access to the bus
  - Packets routed by address
    - Peer-to-peer transfers are possible
  - Packets going to RAM go through “Root Complex”



# A Trip Down Memory Lanes



# A Trip Down Memory Lanes

- PCIe (cont)
  - PCIe devices are discovered via “enumeration”
    - Starts at Bus 0 (root complex), Device 0, Function 0
    - Checks if it’s a bridge or an endpoint
    - Traverses tree of devices
    - Assigns ids and addresses
  - Hot plug device?
    - Enumerate just that port
  - Device info stored in expansion ROMs and read into configuration space



# How My Adventures Went

- Improvised Tools for Analysis
  - Multimeter
  - Soldering station
  - Heat gun
  - Desoldering tools
  - Ethernet cable
  - Epoxy (really?)
  - Logic Analyzer
  - Image Editor



# How My Adventures Went

- Reversing Thunderbolt – The Process
  - Research a product
  - Take it apart
  - Locate important ICs
  - Trace connections between ICs
  - Look for datasheets
  - Sniff buses
  - Develop a map



# How My Adventures Went

- Looking at consumer products
  - Buffalo MiniStation Thunderbolt/USB3 Hard Drive
    - 500GB and 1TB model
    - USB3 and Thunderbolt
    - Decent form factor for reversing
  - Apple Thunderbolt to Gigabit Ethernet Adapter
    - Tiny
    - Small
    - Little





# How My Adventures Went

- Buffalo MiniStation Thunderbolt/USB3 Hard Drive
  - Excellent Anandtech review<sup>7</sup>
  - Tear-down instructions
  - Identified ICs for us!

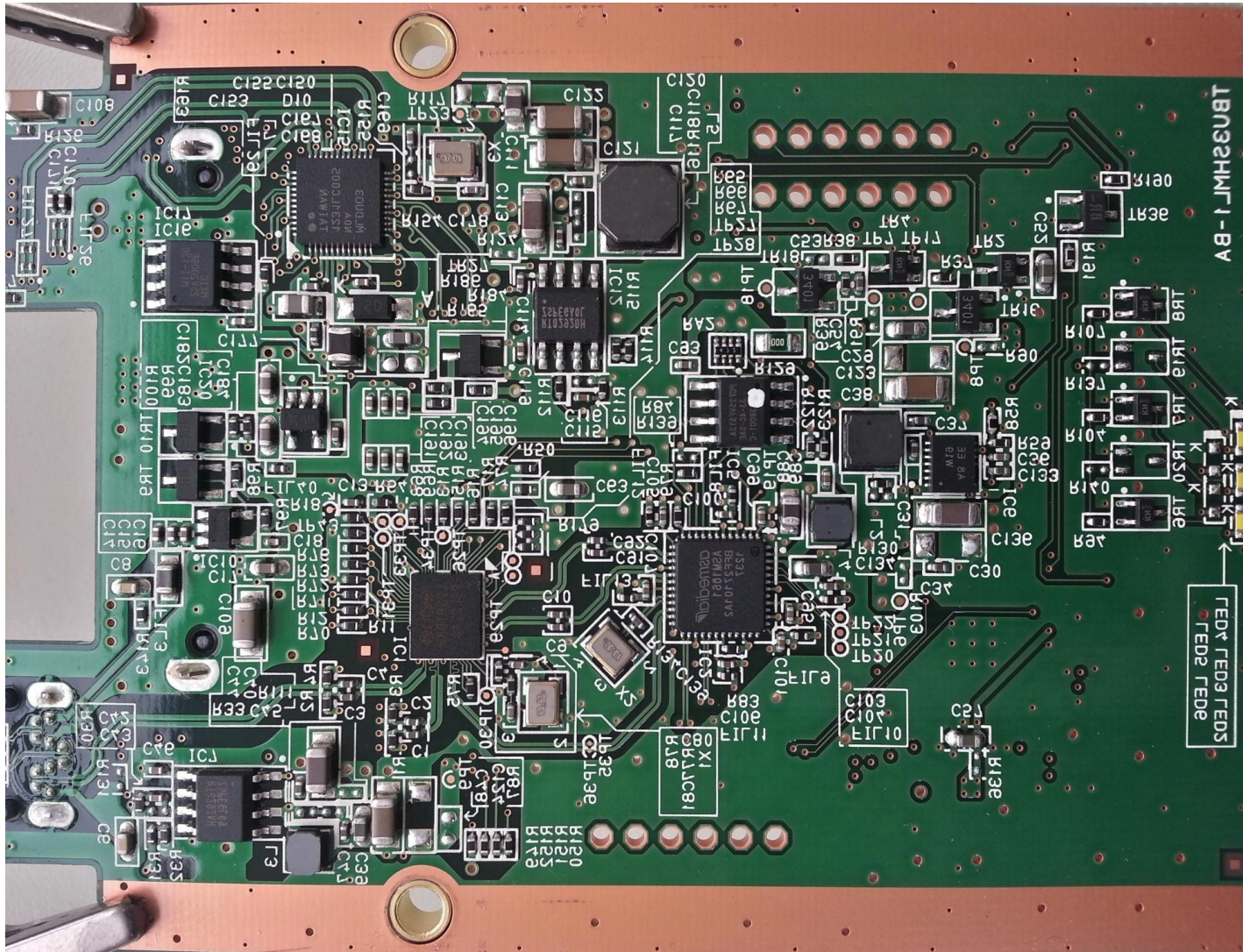


# How My Adventures Went

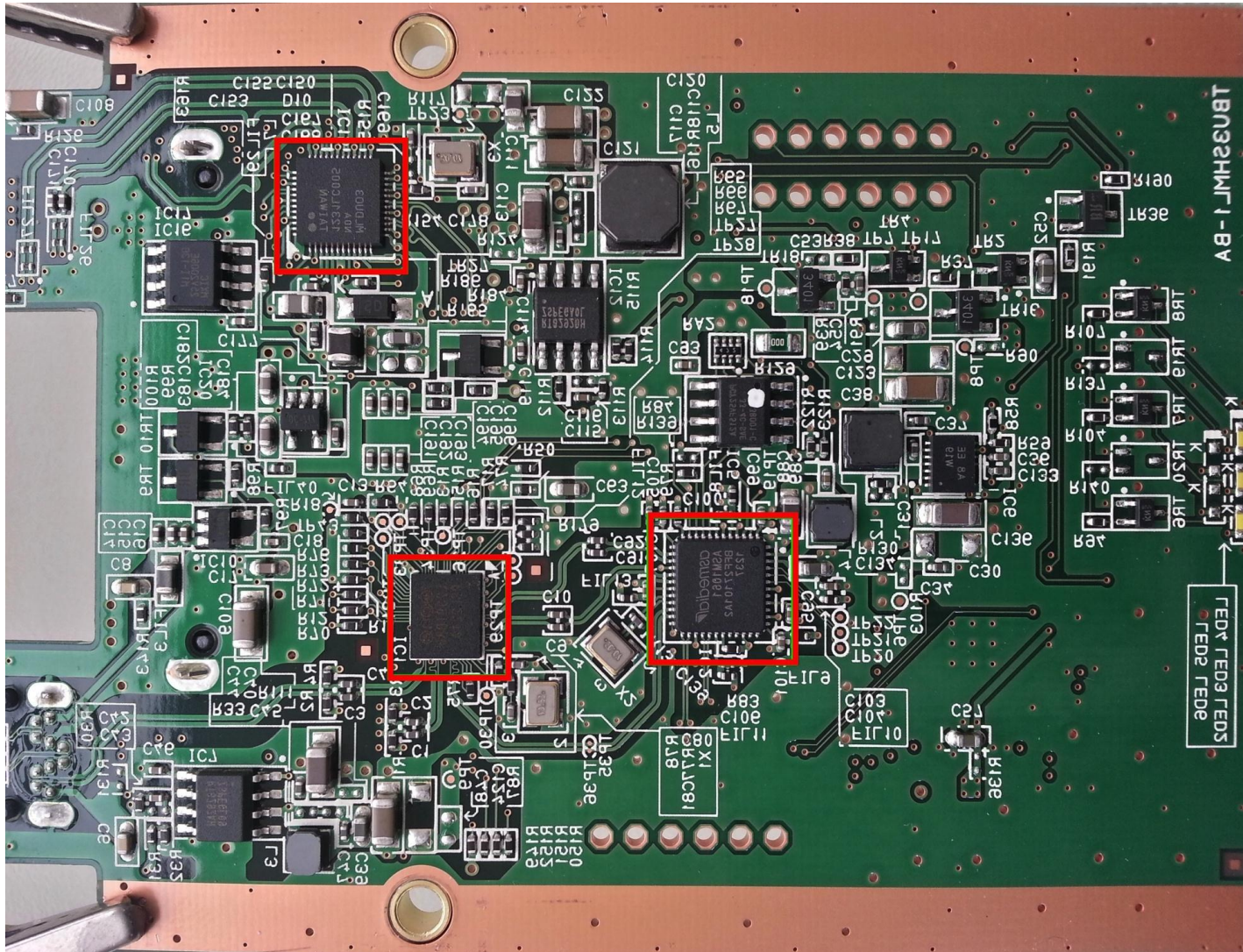
- Main ICs
  - MLDU03
    - Medial Logic USB3.0 to SATA 6G Bridge
  - ASM1061
    - ASMedia PCIe to SATA Controller
  - DSL2210 (Peak Ridge)
    - Intel Thunderbolt Controller
    - Supports PCIe x1
  - LPC1114
    - NXP ARM Cortex Mo



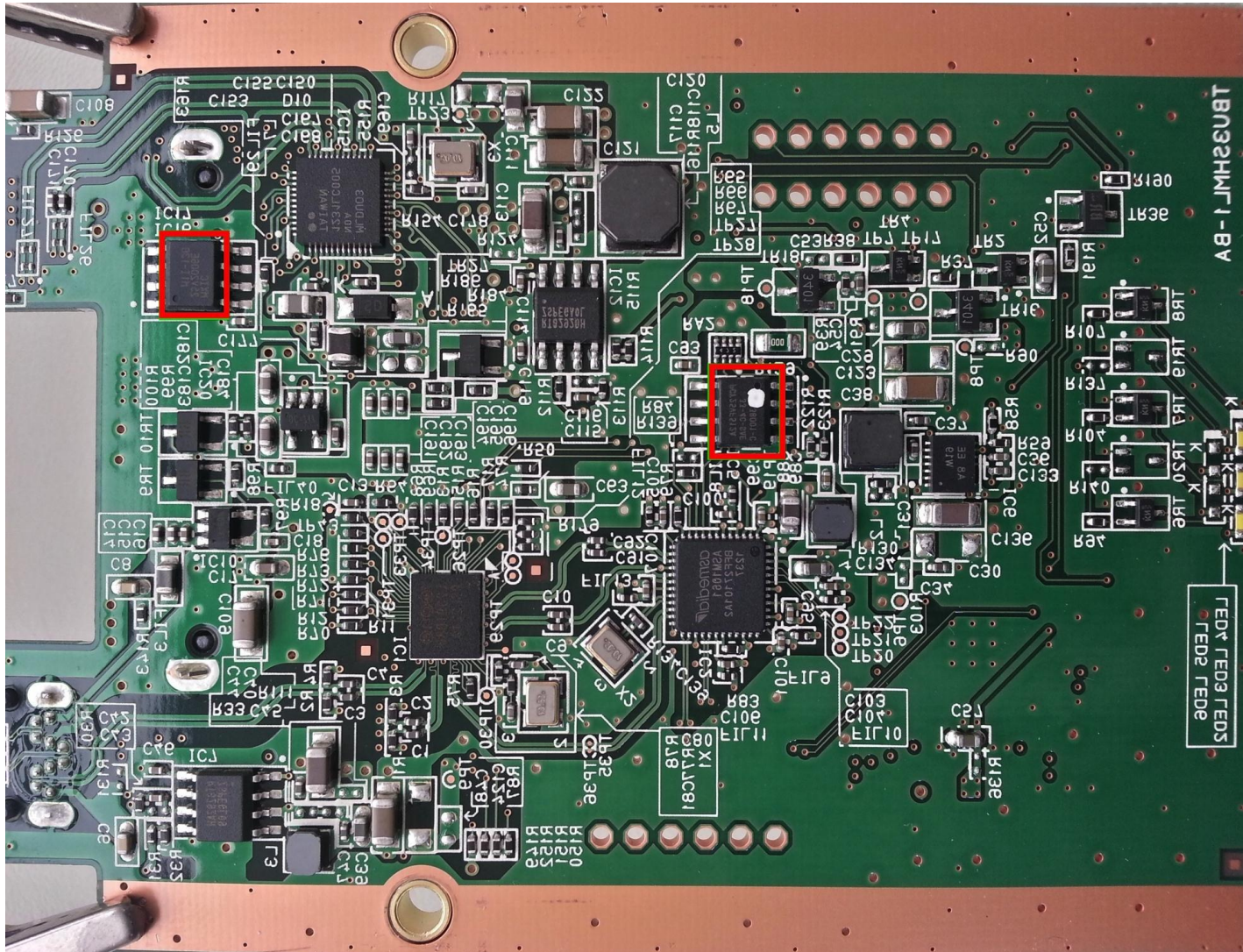
# How My Adventures Went



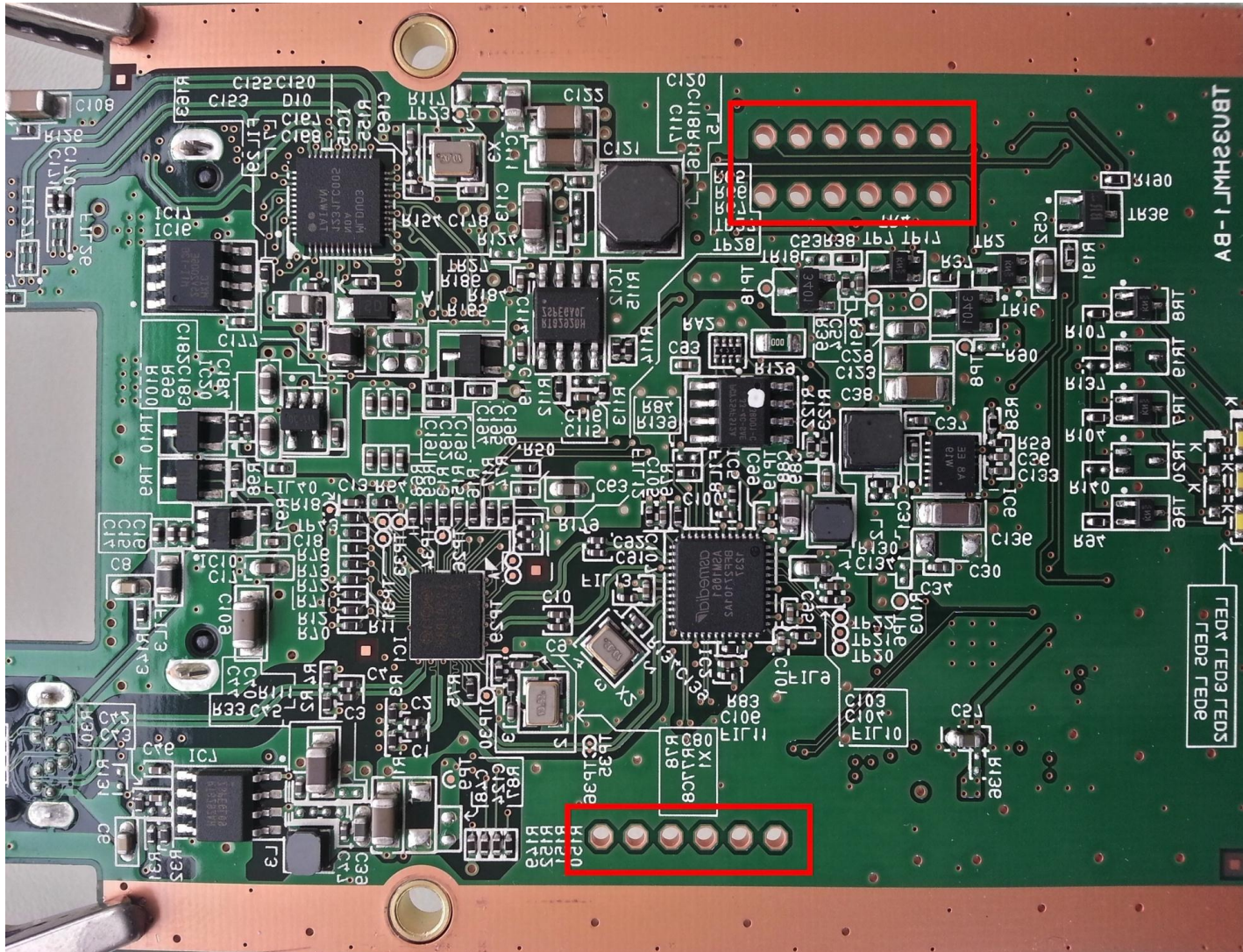
# How My Adventures Went



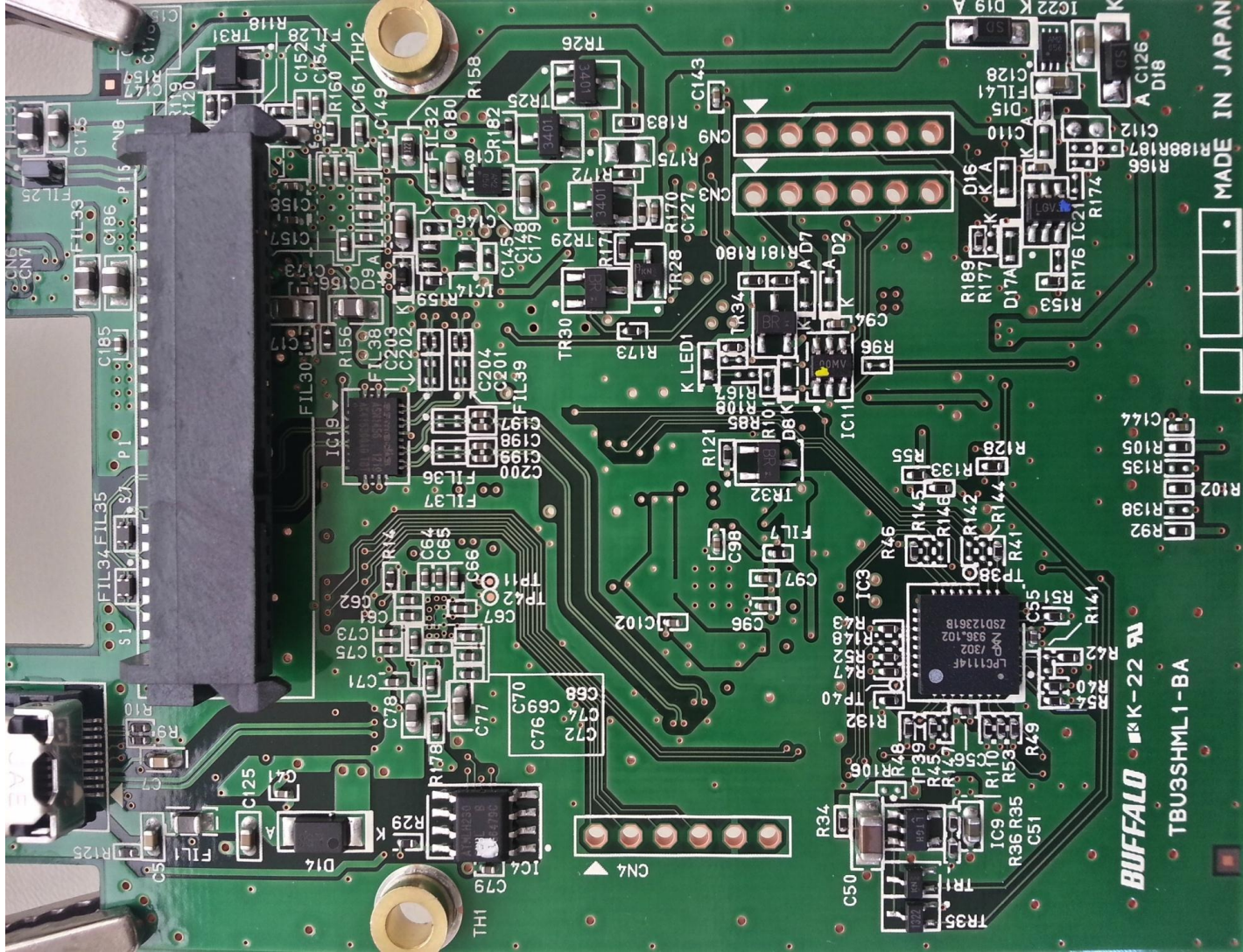
# How My Adventures Went



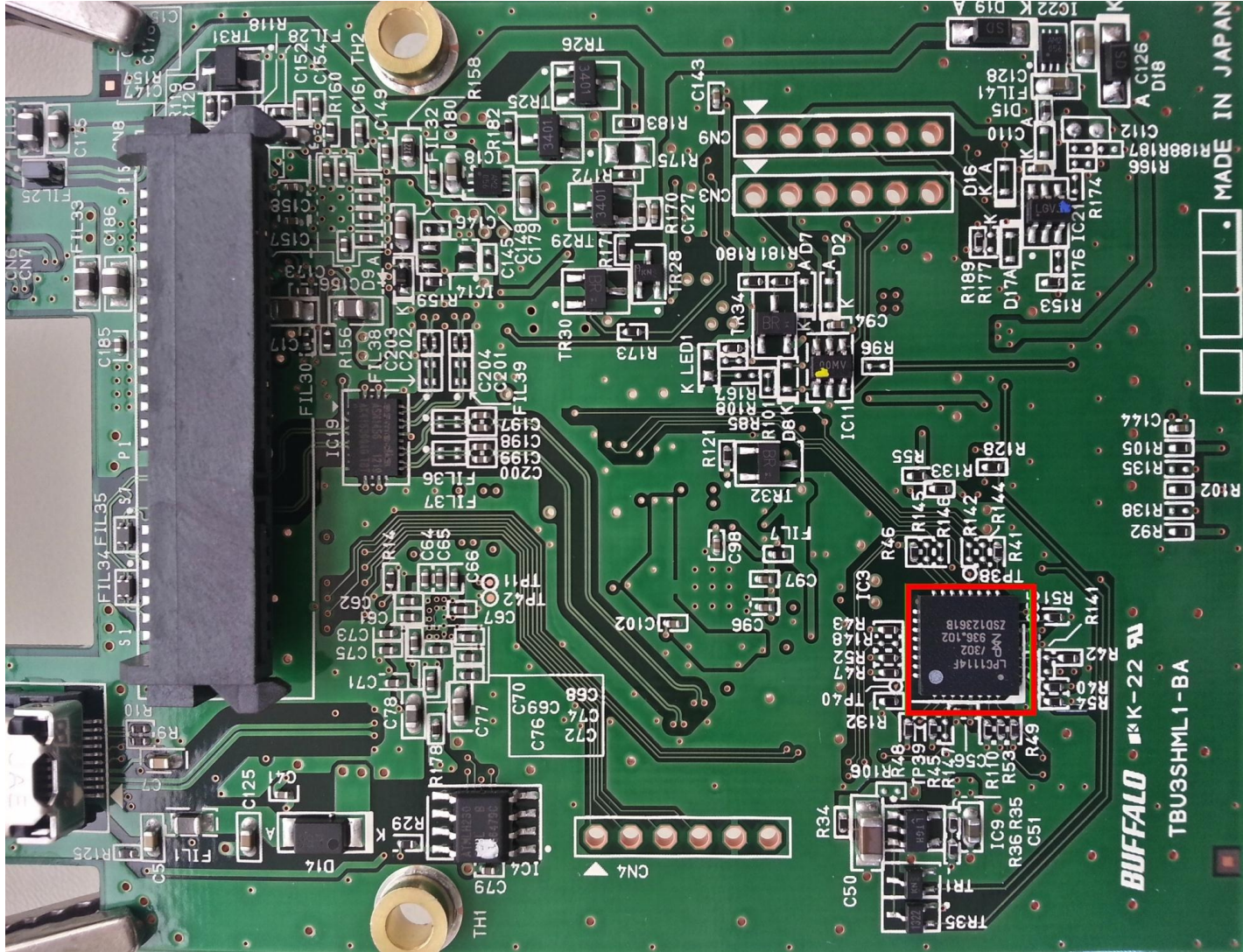
# How My Adventures Went



# How My Adventures Went

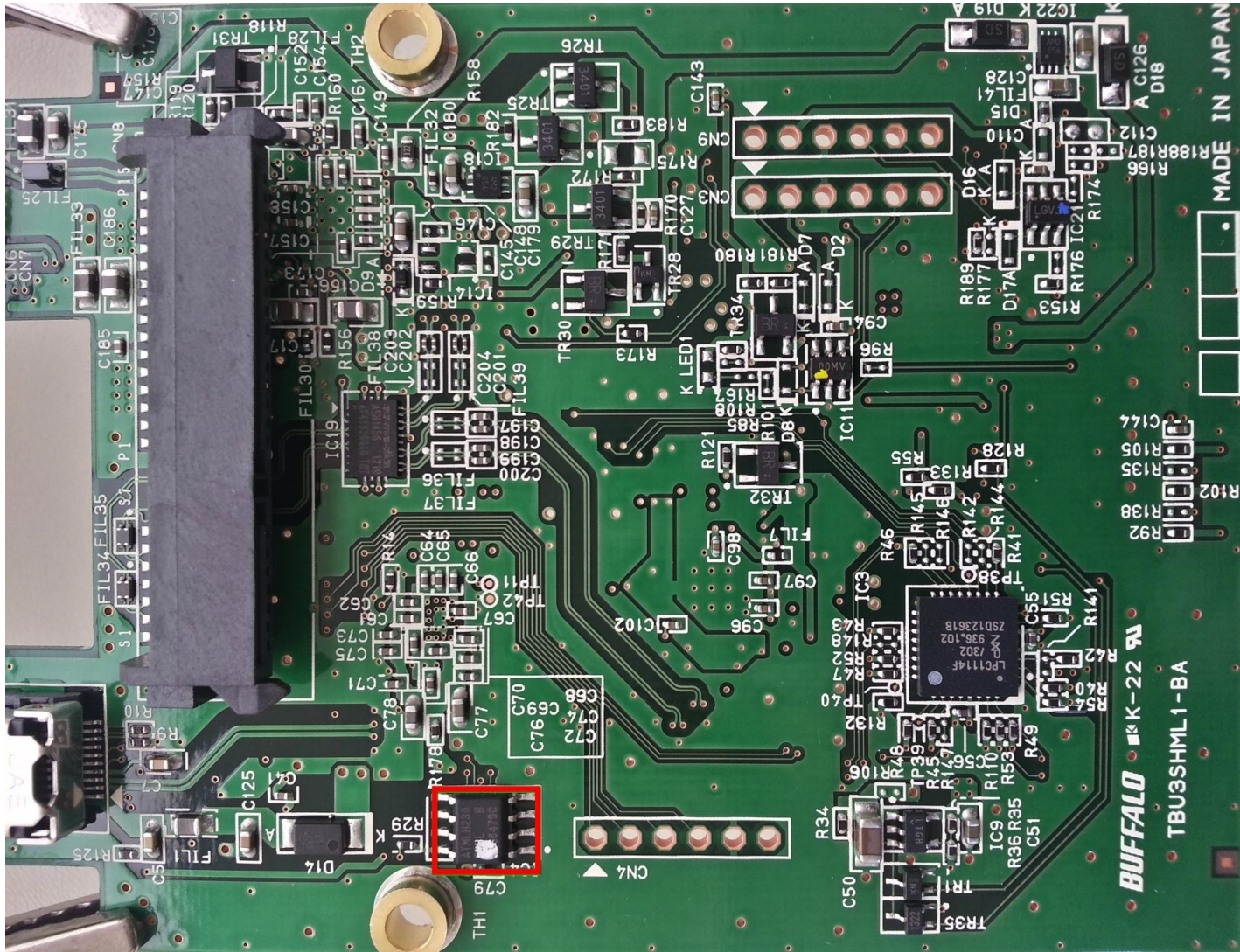


# How My Adventures Went

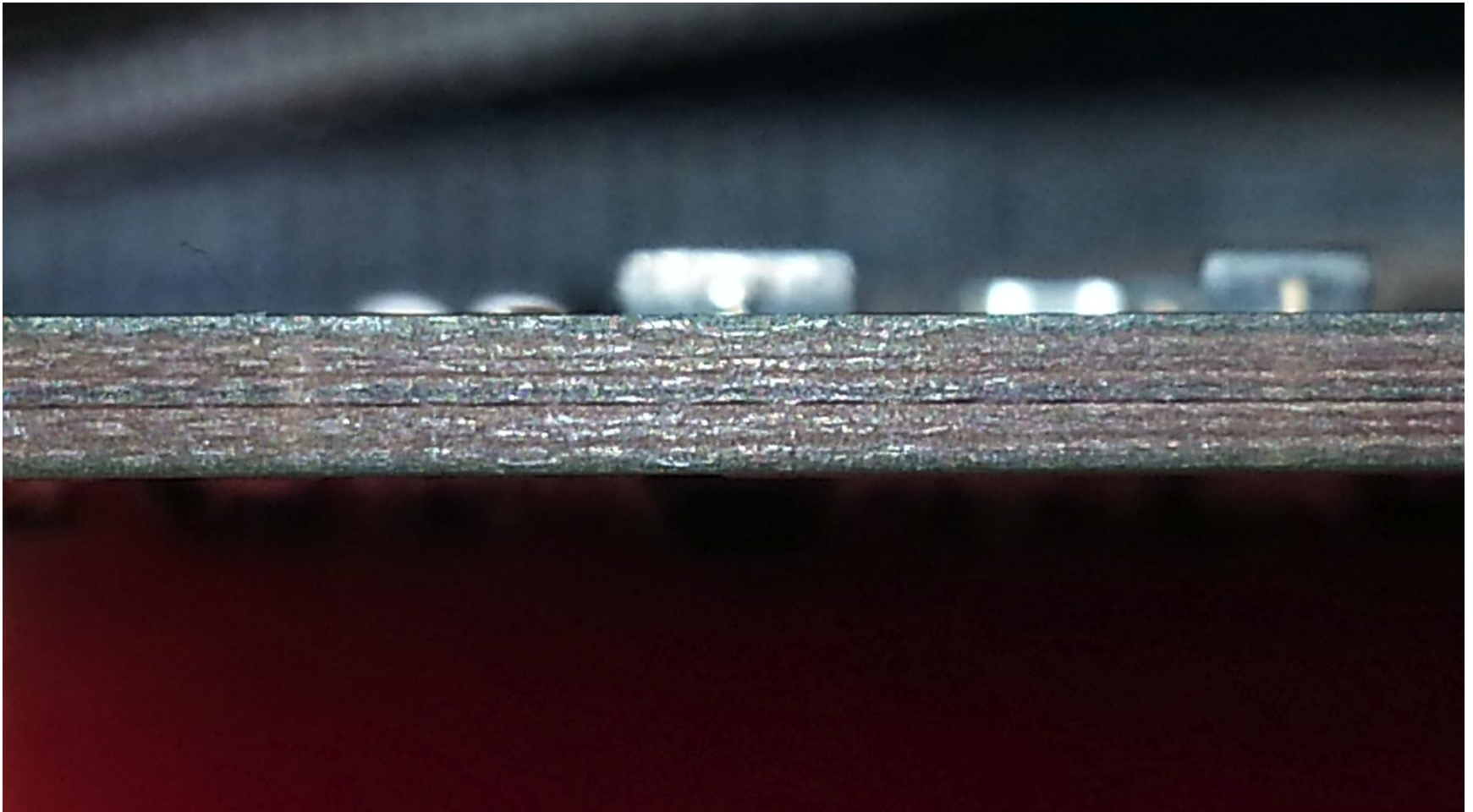




# How My Adventures Went



# How My Adventures Went



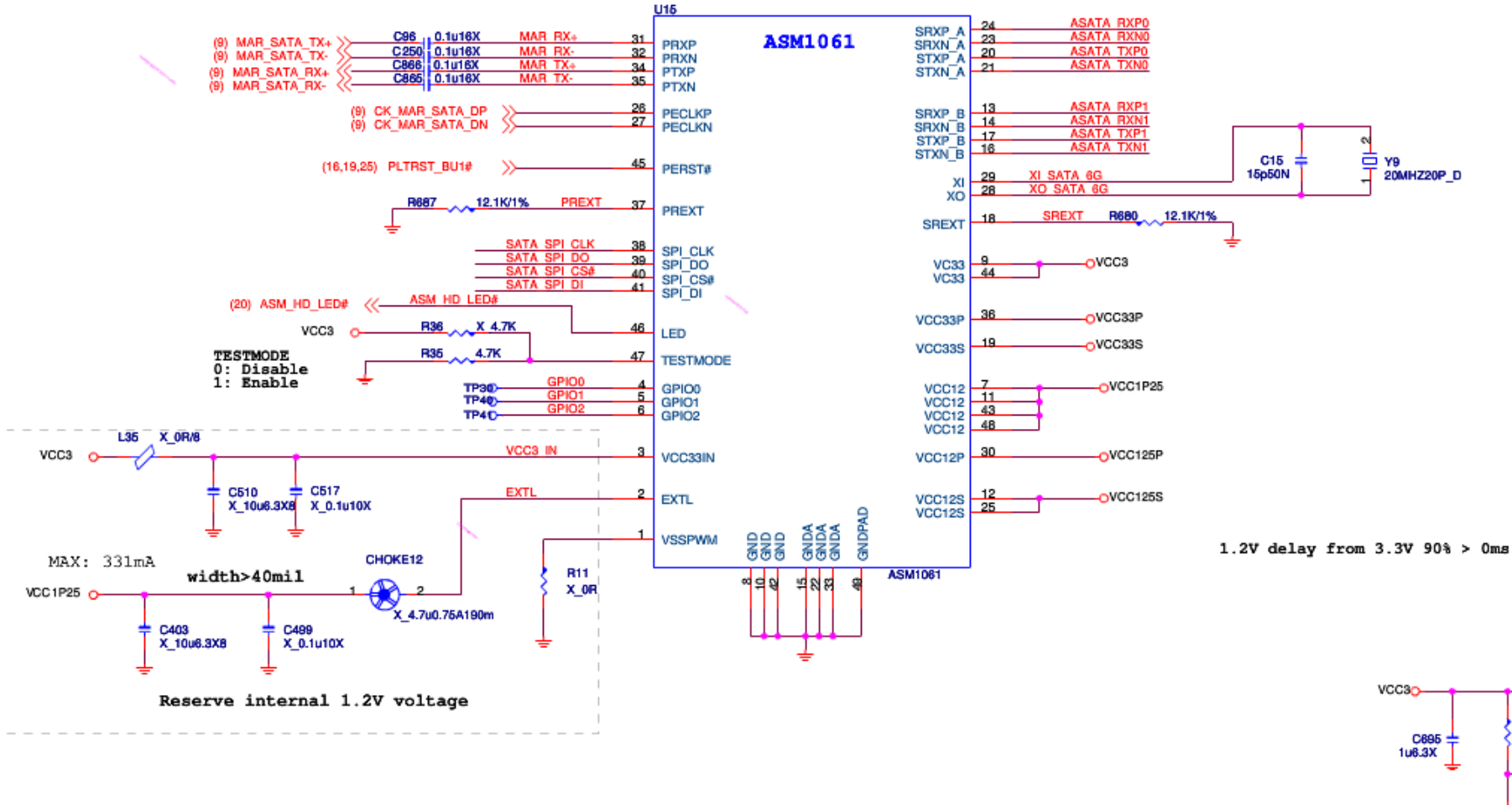
# How My Adventures Went

- ASMedia ASM1061
  - PCIe/SATA Controller
  - Datasheets?
  - ROMs/Flashes?

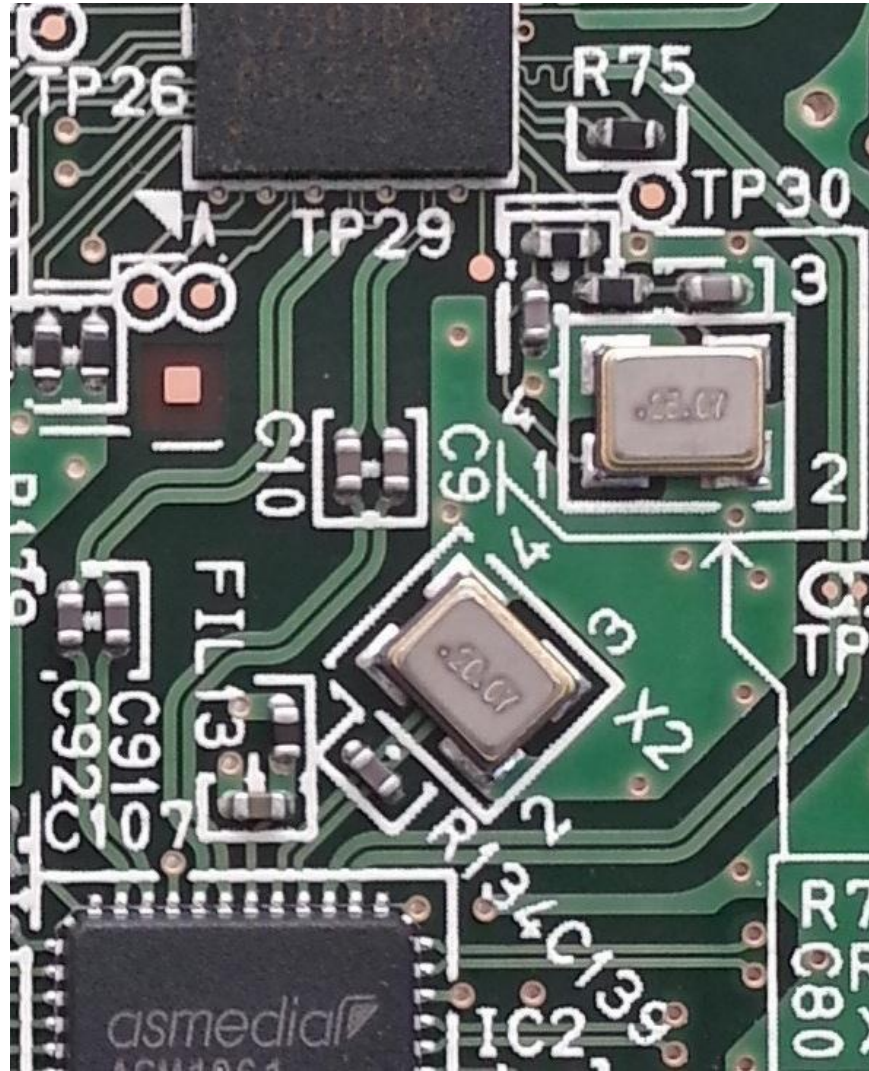


# How My Adventures Went

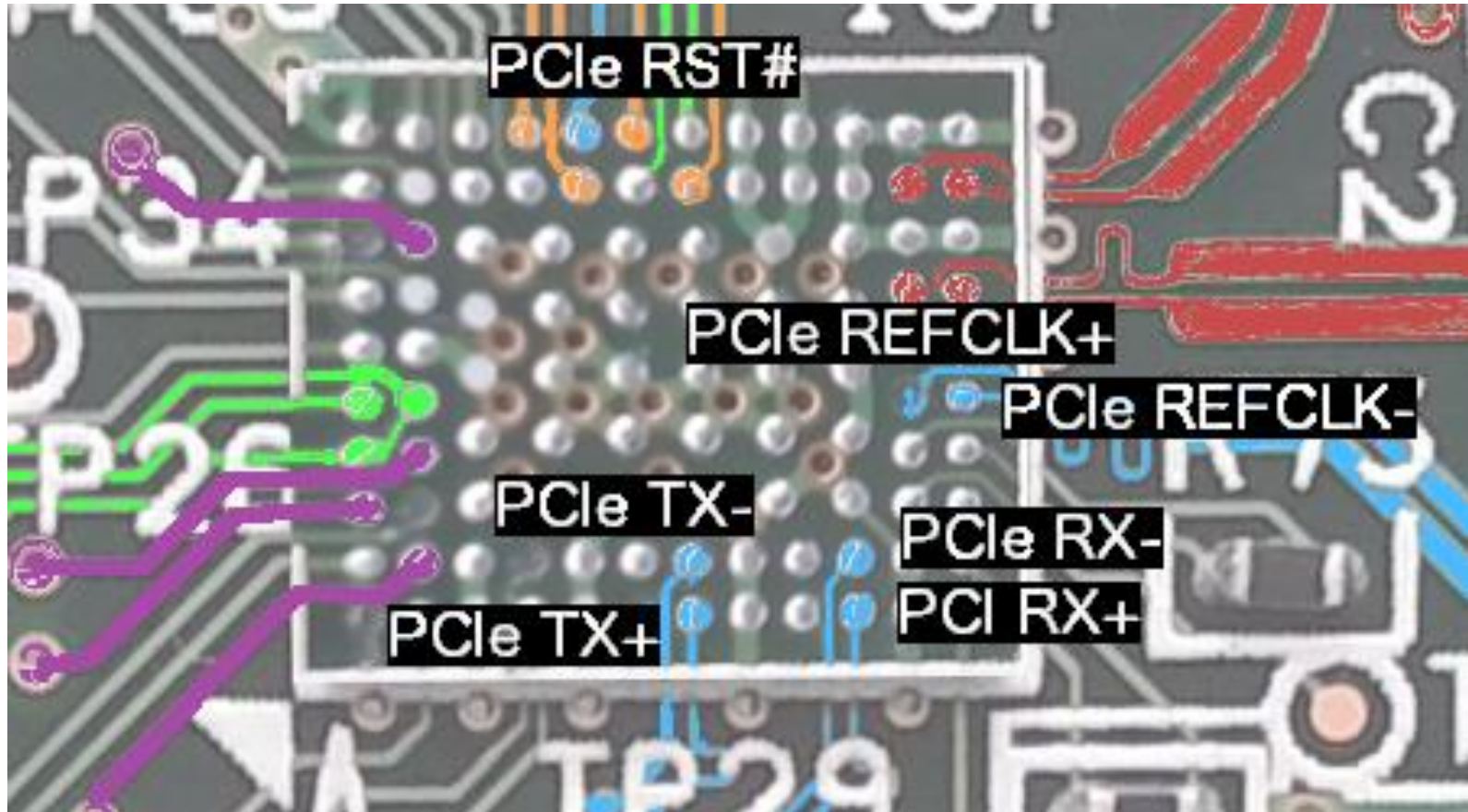
## ASM1061 SATA6G



# How My Adventures Went



# How My Adventures Went



■ PCIe

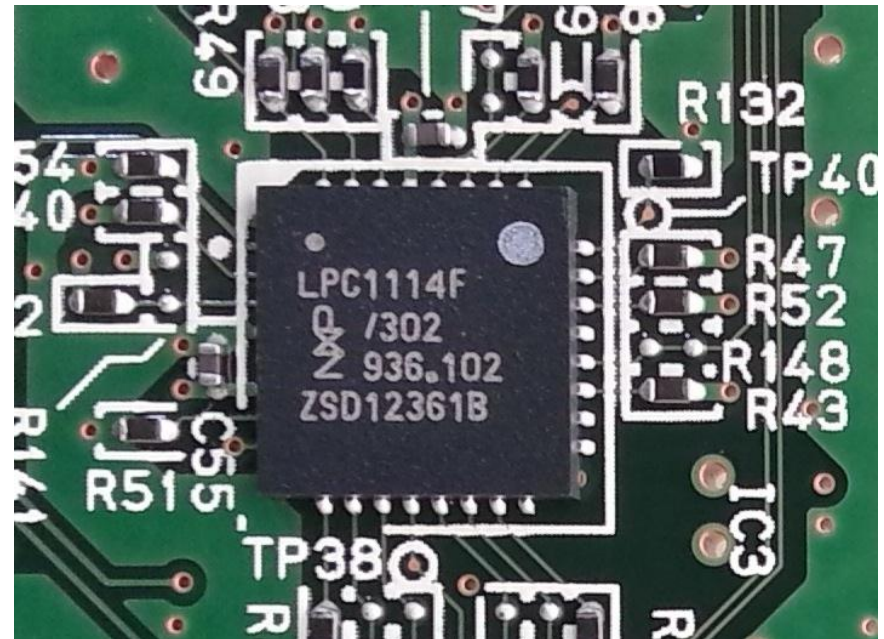
# How My Adventures Went

- Patch PCIe Controllers' SPI ROM to send DMA read requests?



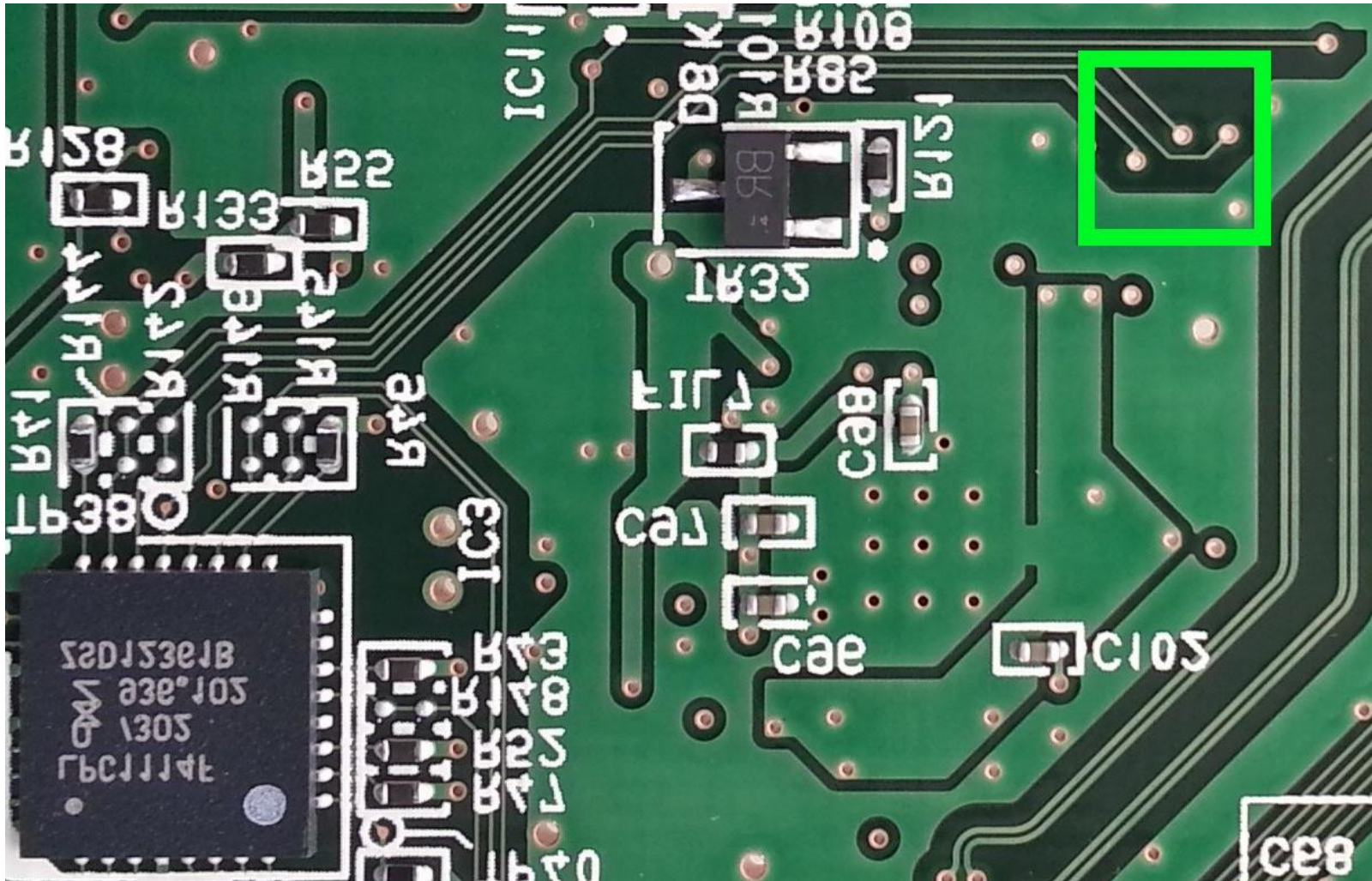
# How My Adventures Went

- NXP LPC1114
  - ARM Cortex Mo
  - Used for... ??
  - No ROMs or Flashes
  - TONS of info online
  - Connects into DSL2201
    - How do I know?

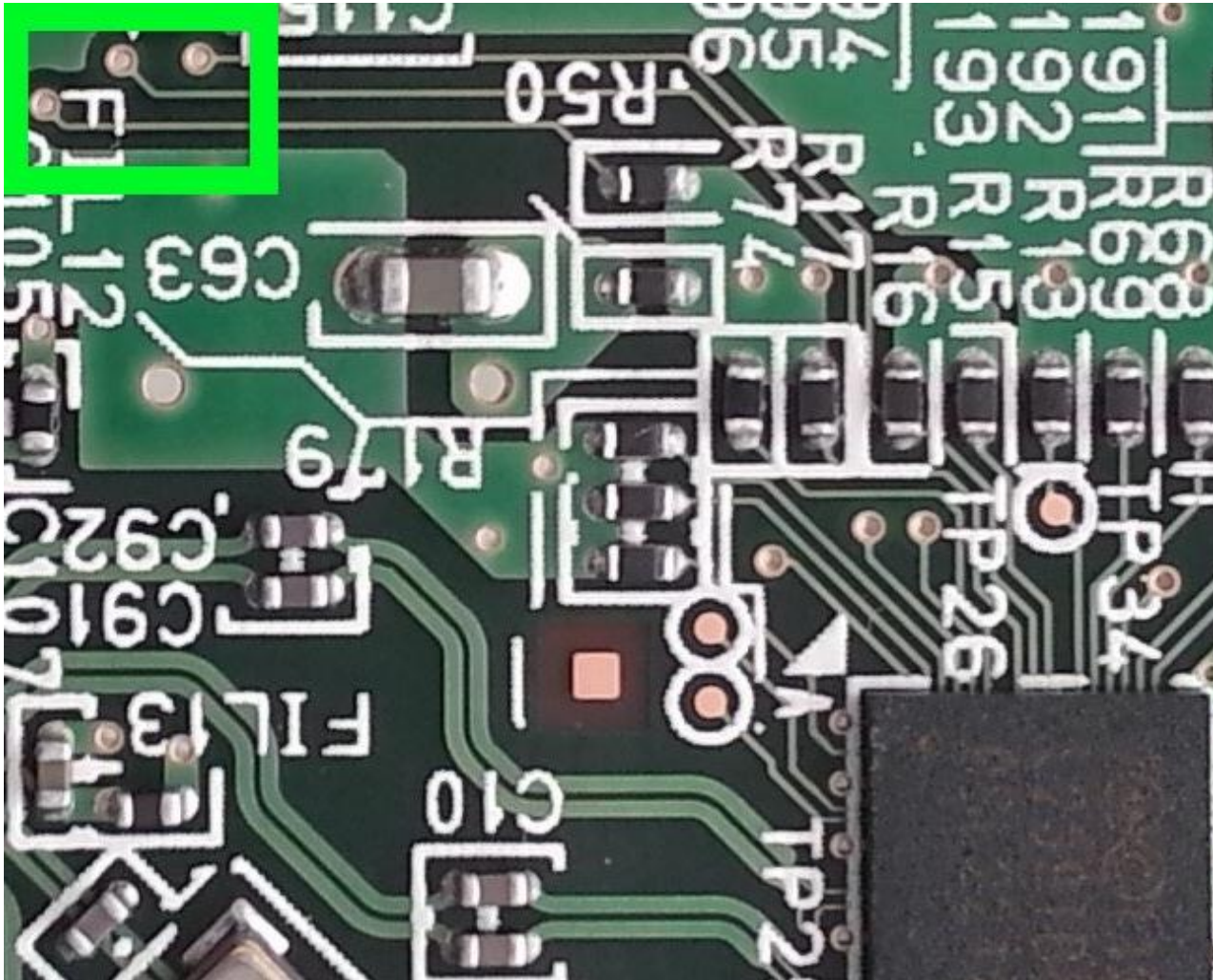




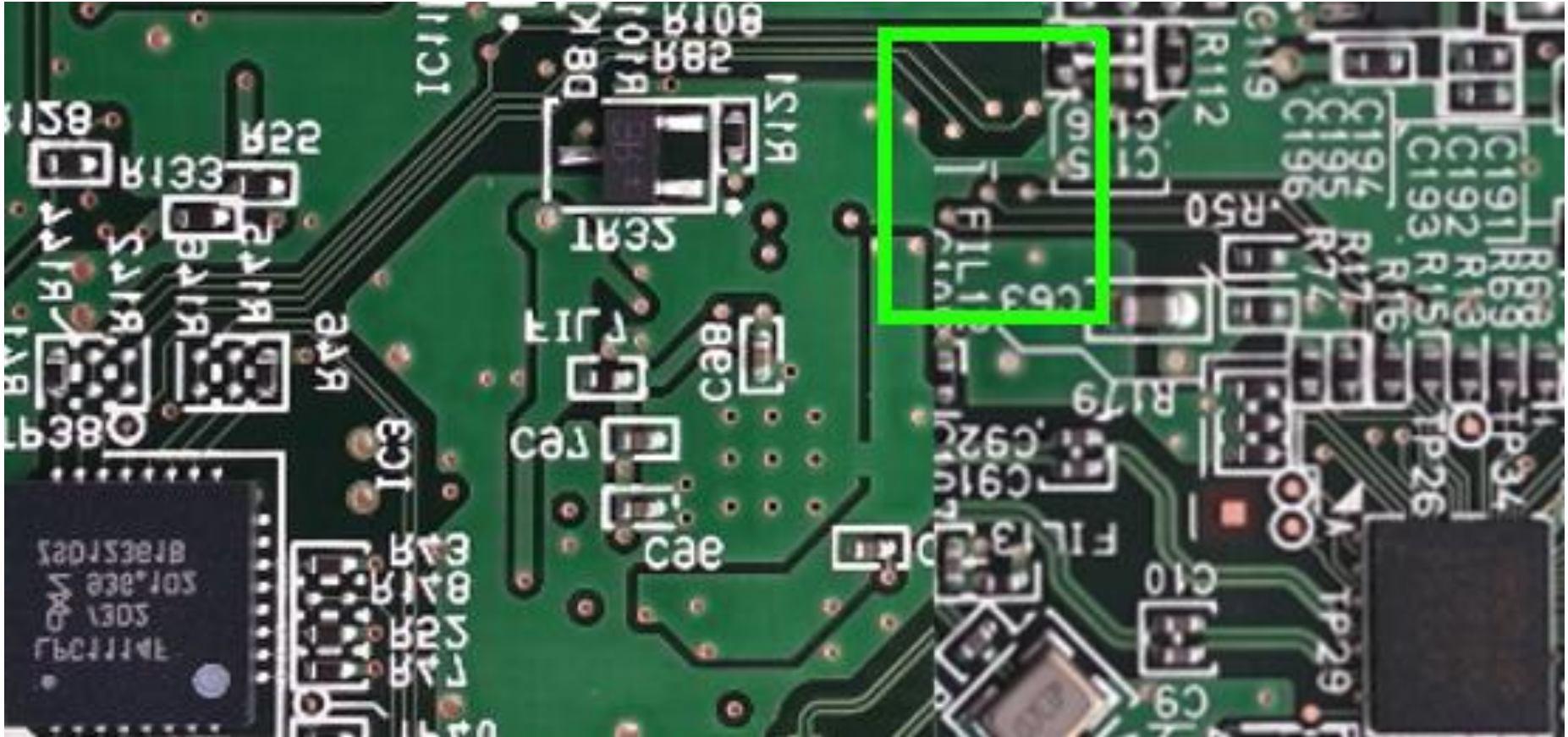
# How My Adventures Went



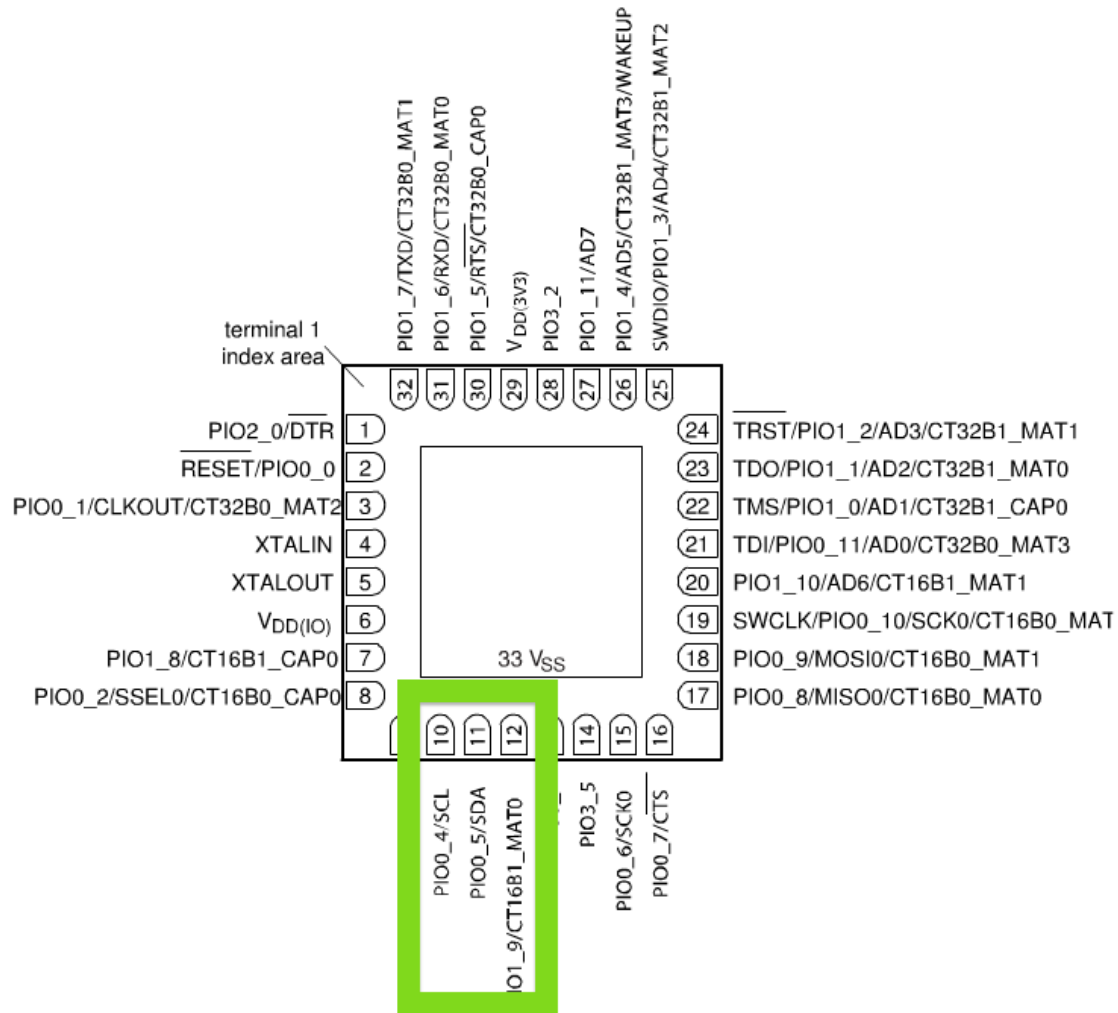
# How My Adventures Went



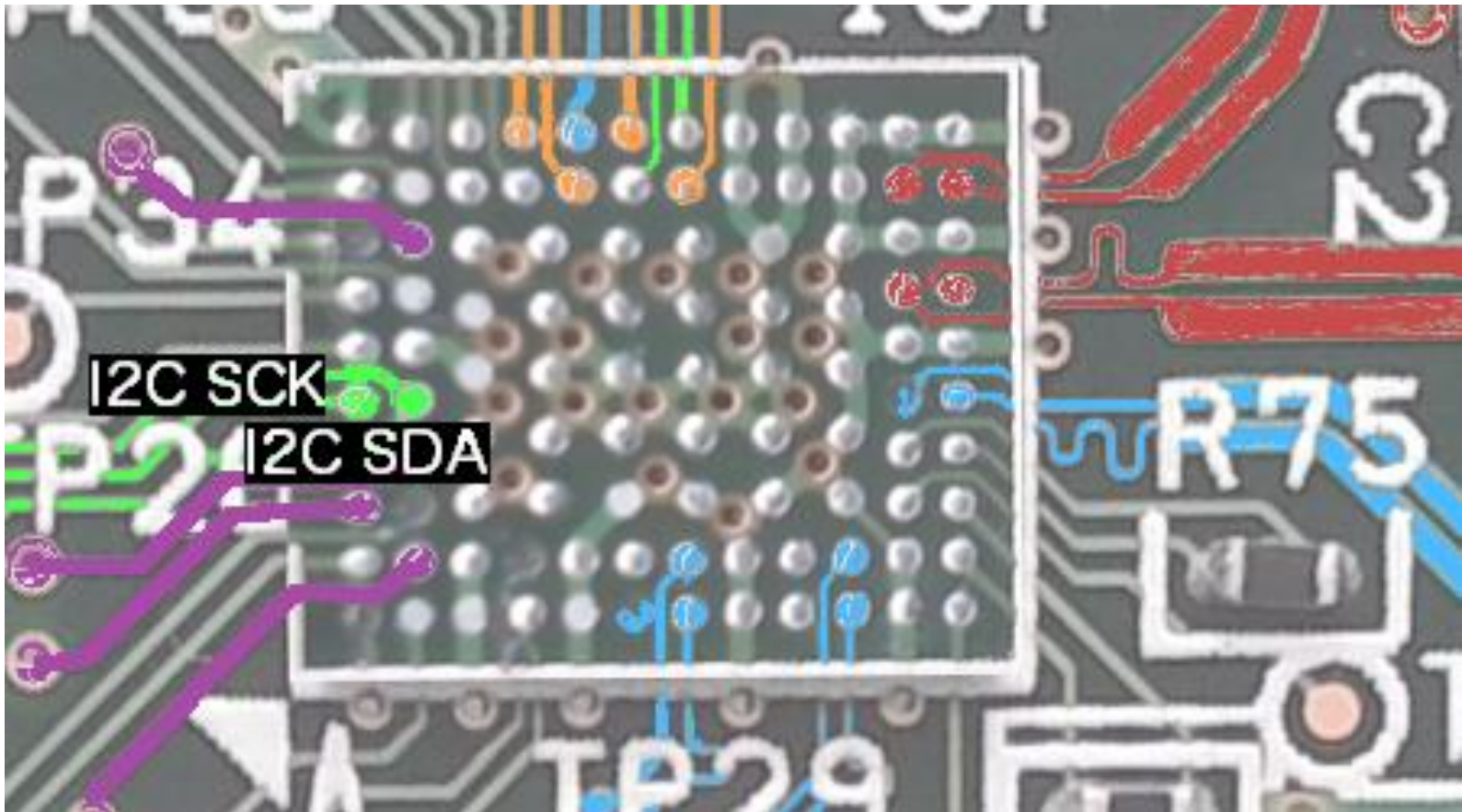
# How My Adventures Went



# How My Adventures Went



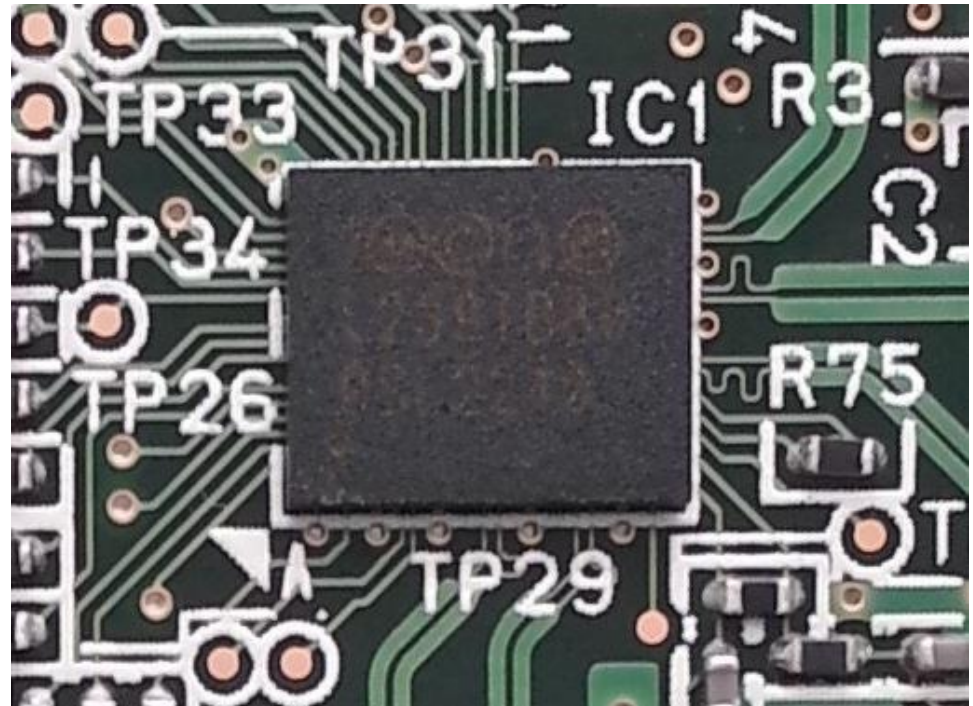
# How My Adventures Went



 ARM

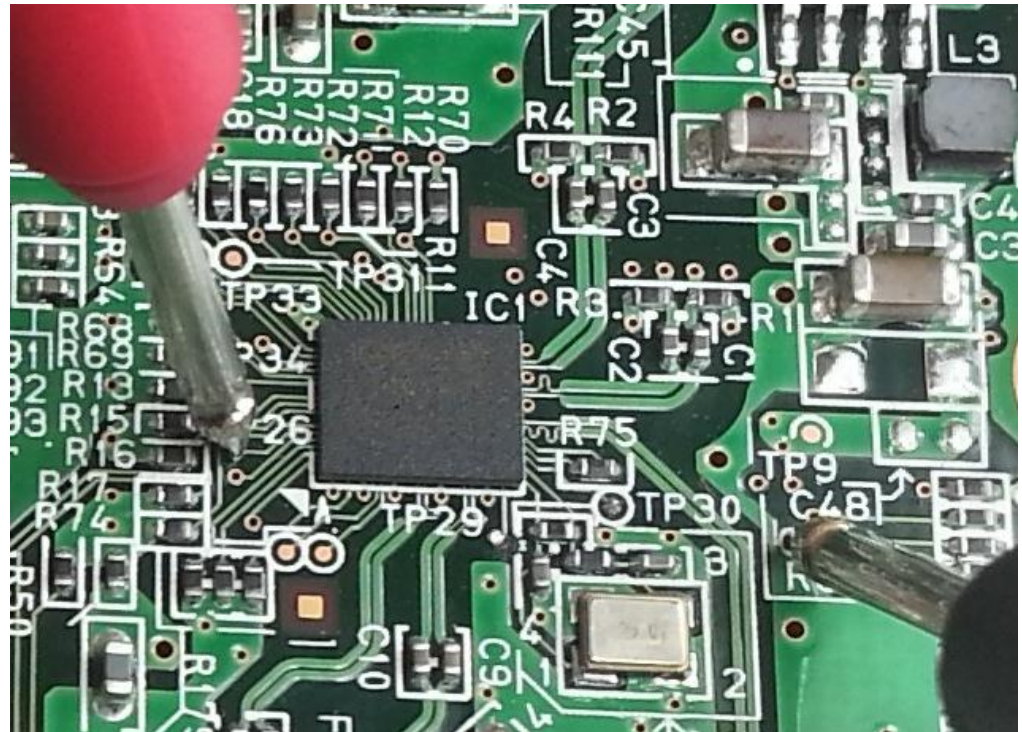
# How My Adventures Went

- Intel DSL2210
  - Thunderbolt Controller
  - No Datasheets
  - Promo info only
  - ROMs/Flashes?

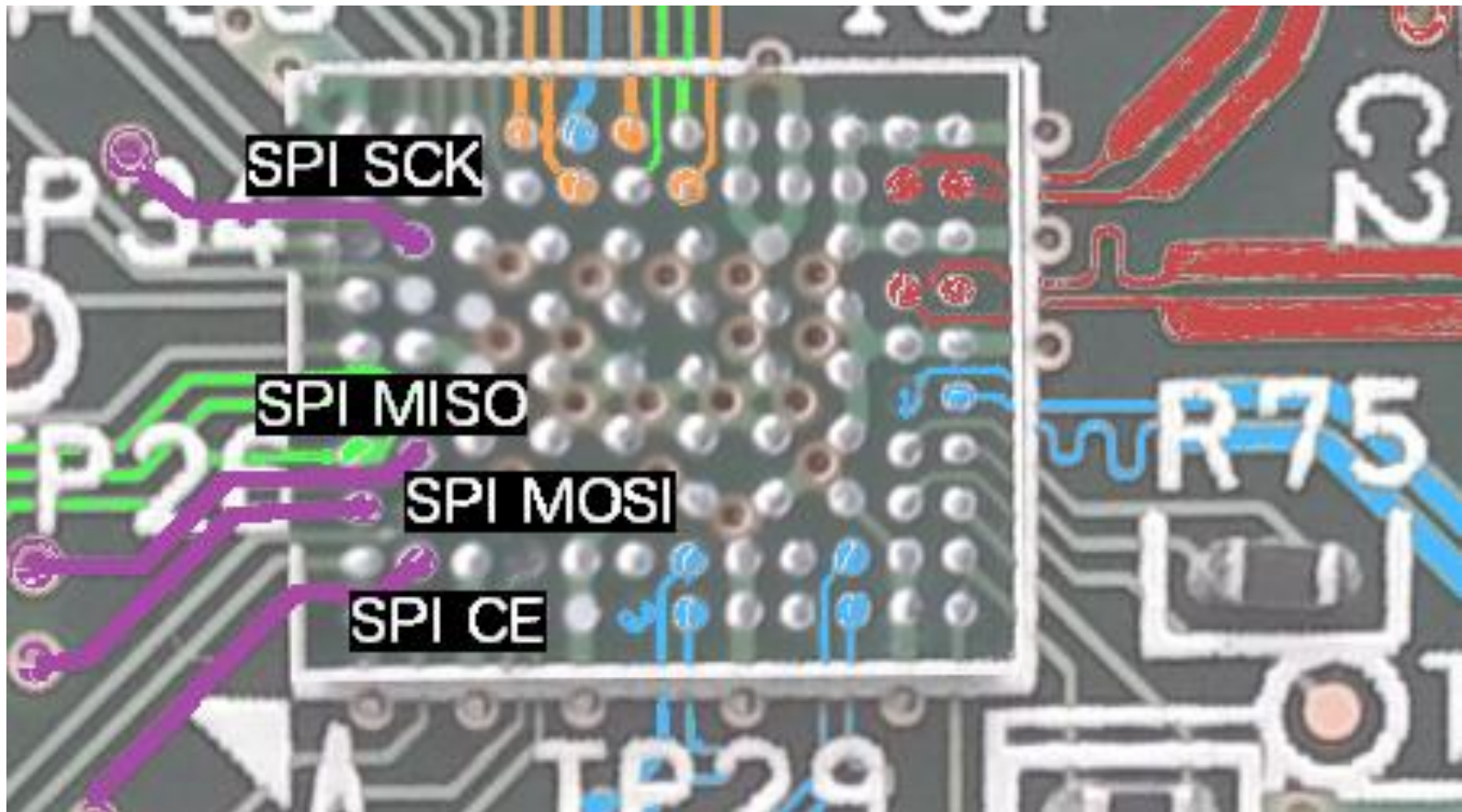


# How My Adventures Went

- Continuity testing SPI ROM...
  - Beep! It's Thunderbolt!
  - hasROM = TRUE;



# How My Adventures Went

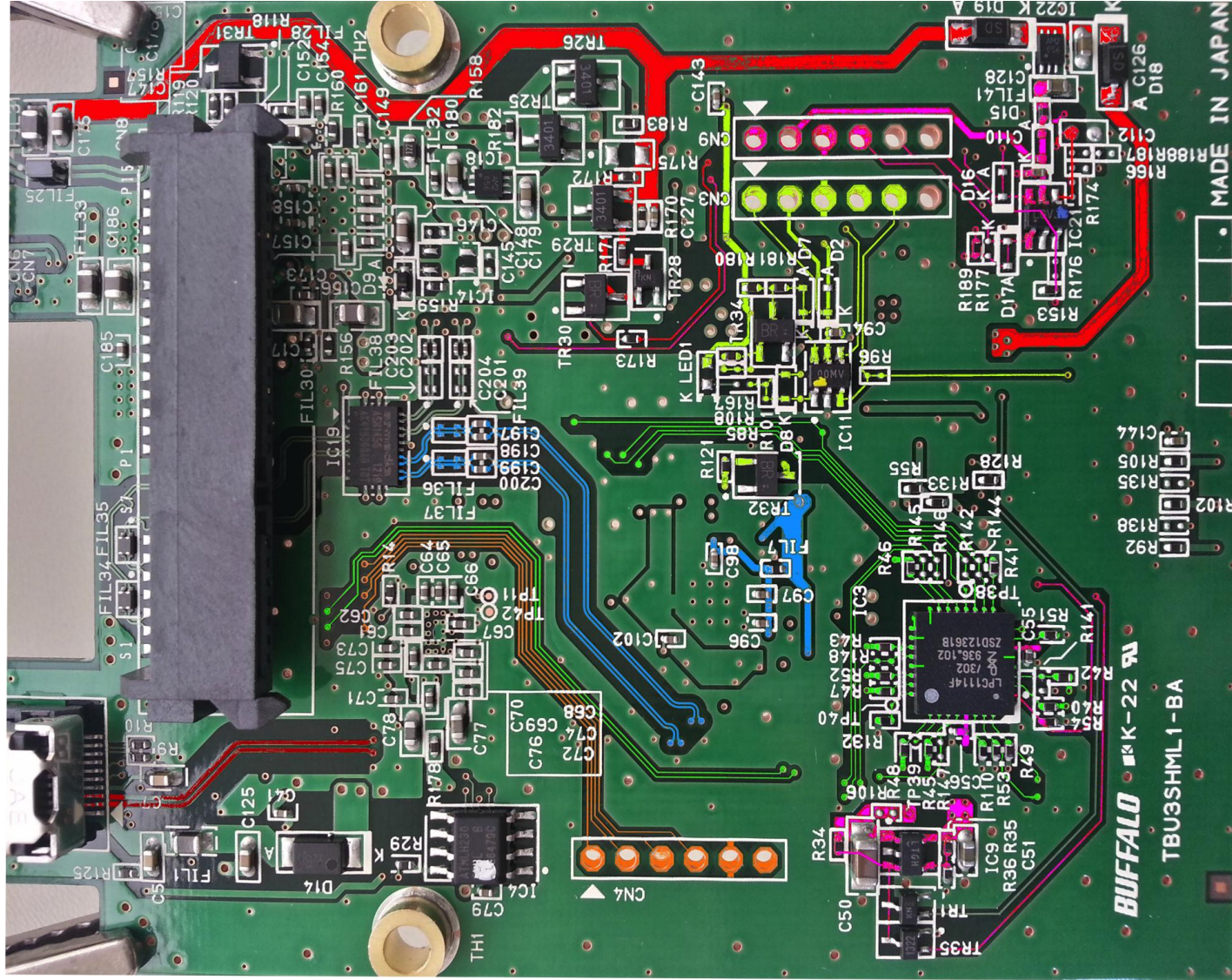


SPI

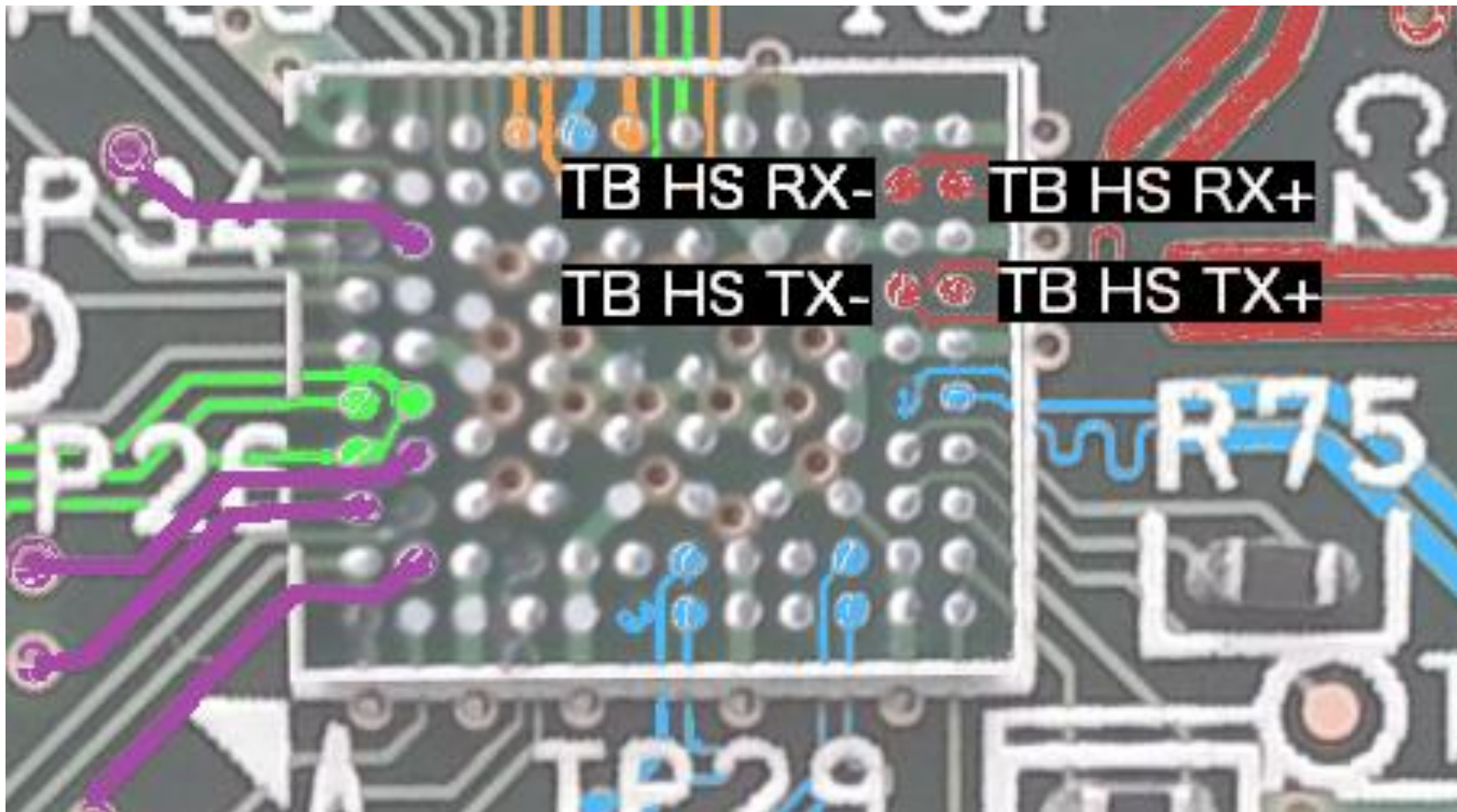




# How My Adventures Went



# How My Adventures Went



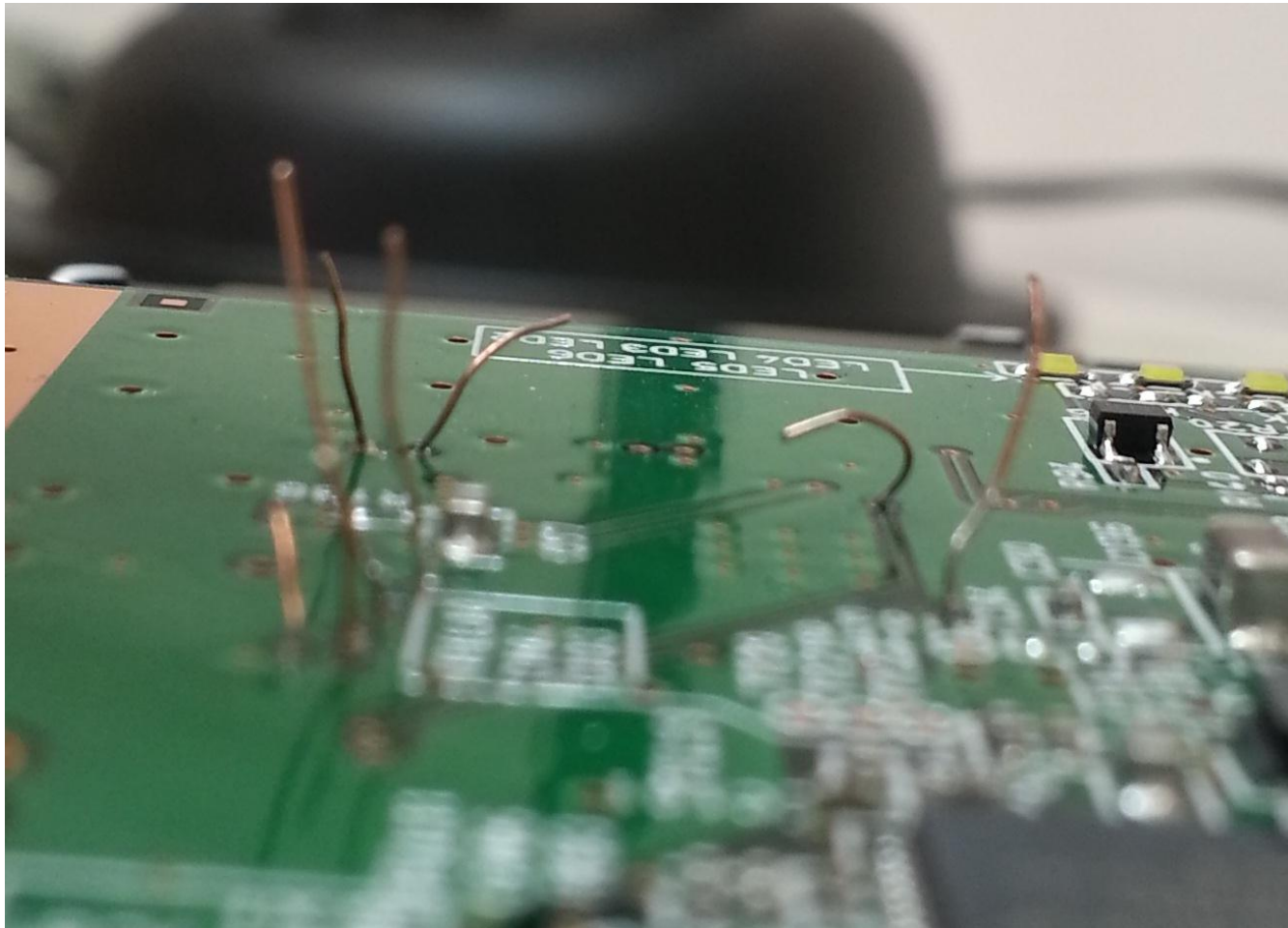
Thunderbolt  
Connector

# How My Adventures Went

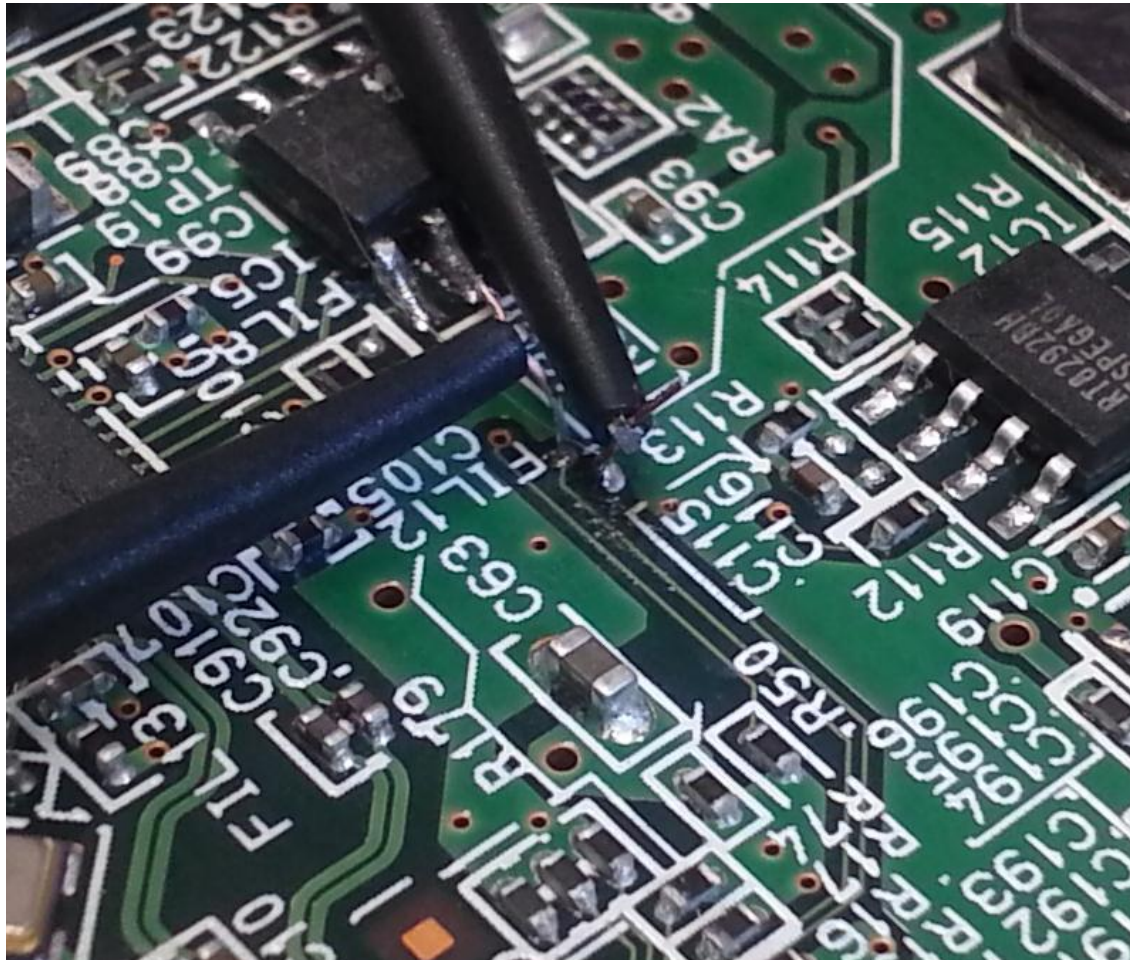
- Thunderbolt Connector
  - 1 pair of HighSpeed lanes
    - TX and RX
  - All others pulled to ground
  - “LowSpeed” lines go to ARM?

Pin out		
Pin 1	GND	Ground
Pin 2	HPD	Hot plug detect
Pin 3	HS0TX(P)	HighSpeed transmit 0 (positive)
Pin 4	HS0RX(P)	HighSpeed receive 0 (positive)
Pin 5	HS0TX(N)	HighSpeed transmit 0 (negative)
Pin 6	HS0RX(N)	HighSpeed receive 0 (negative)
Pin 7	GND	Ground
Pin 8	GND	Ground
Pin 9	LSR2P TX	LowSpeed transmit
Pin 10	GND	Ground (reserved)
Pin 11	LSP2R RX	LowSpeed receive
Pin 12	GND	Ground (reserved)
Pin 13	GND	Ground
Pin 14	GND	Ground
Pin 15	HS1TX(P)	HighSpeed transmit 1 (positive)
Pin 16	HS1RX(P)	HighSpeed receive 1 (positive)
Pin 17	HS1TX(N)	HighSpeed transmit 1 (negative)
Pin 18	HS1RX(N)	HighSpeed receive 1 (negative)
Pin 19	GND	Ground
Pin 20	DPPWR	Power

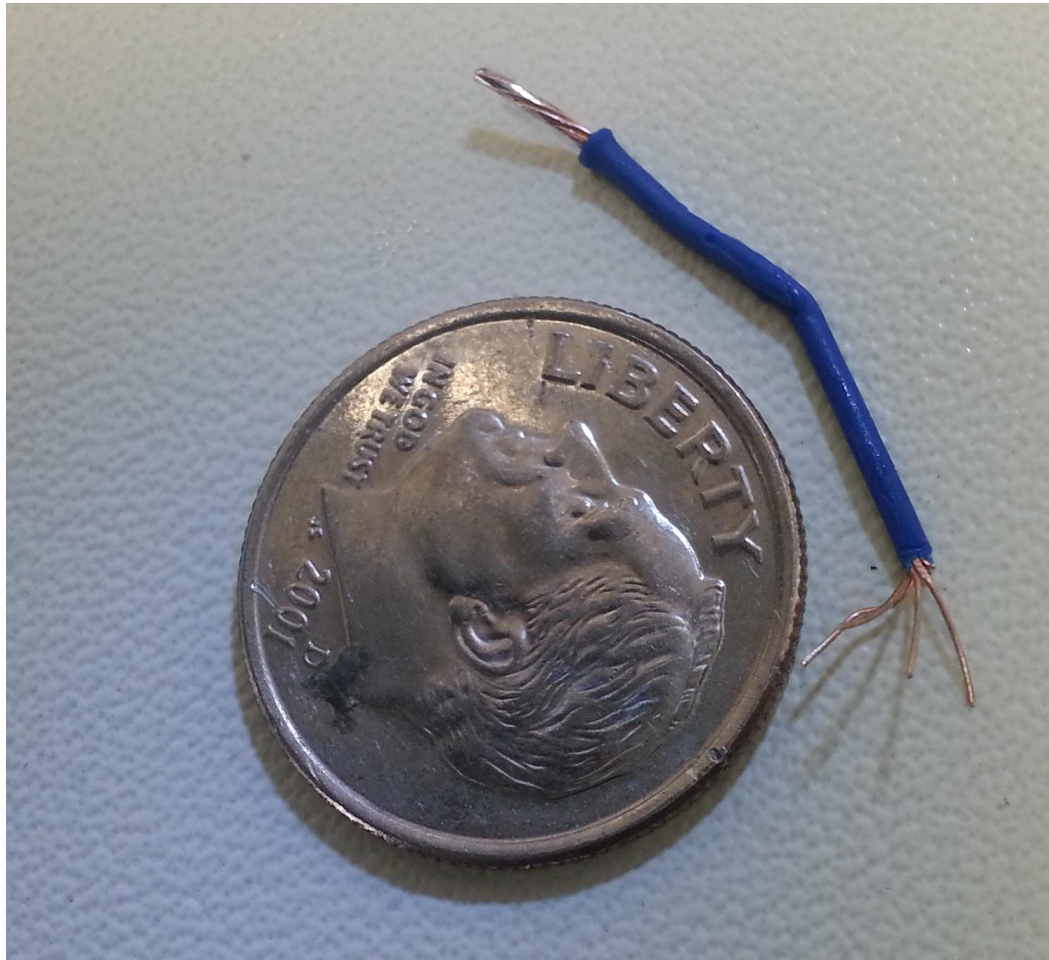
# How My Adventures Went



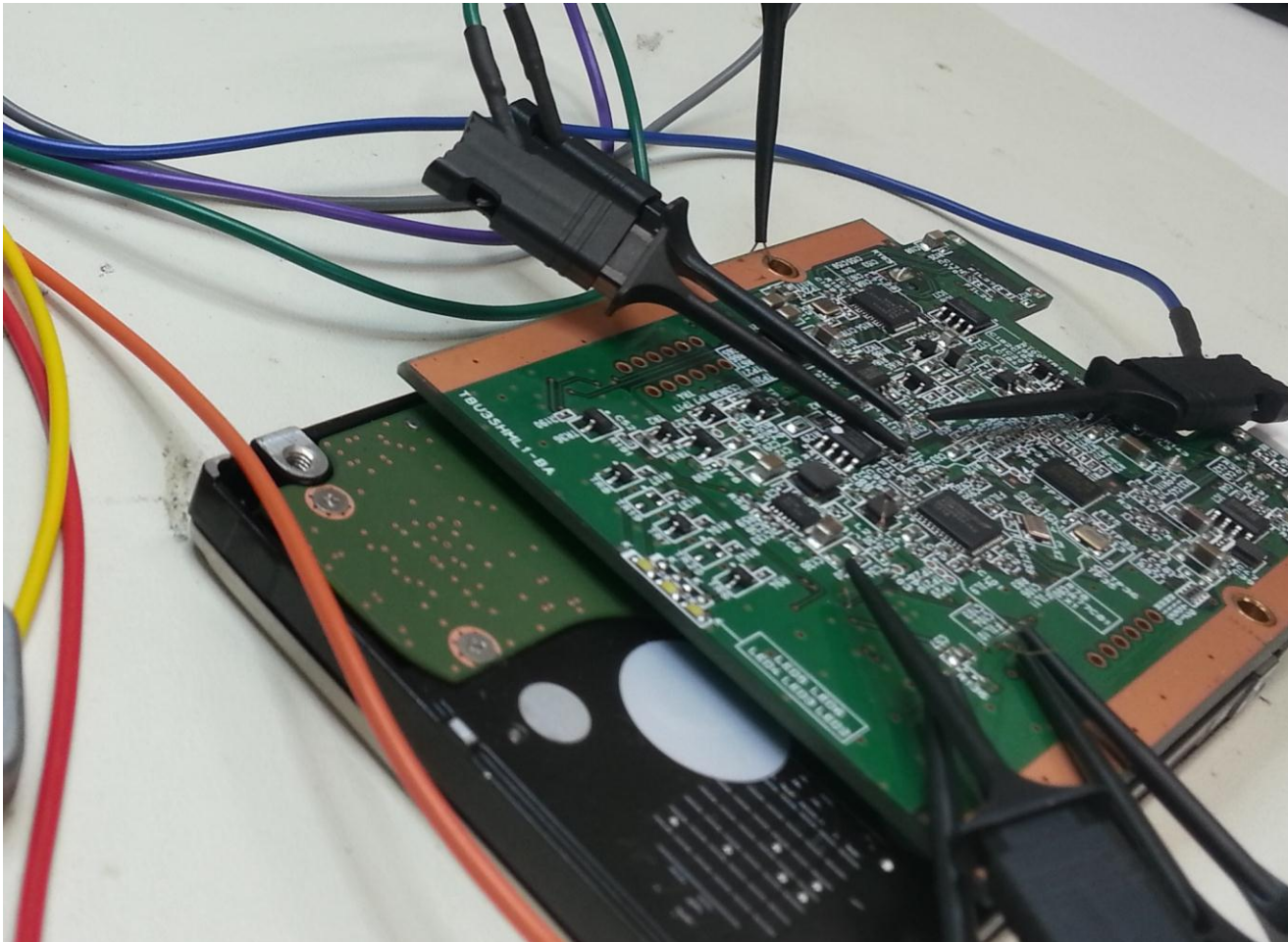
# How My Adventures Went



# How My Adventures Went

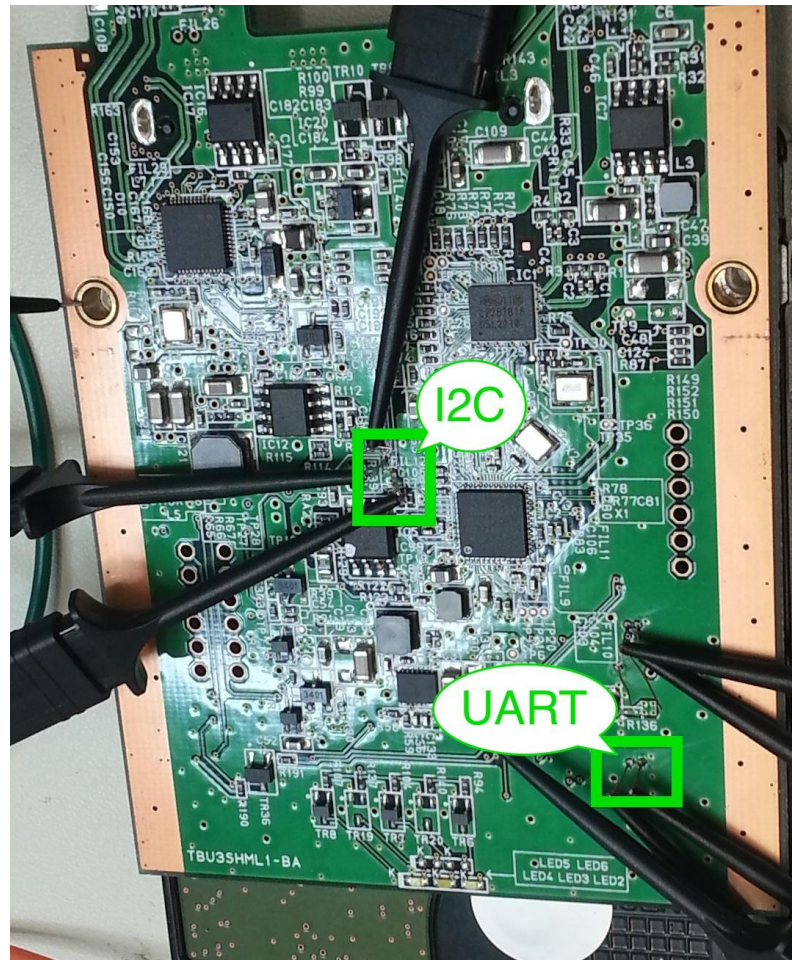


# How My Adventures Went

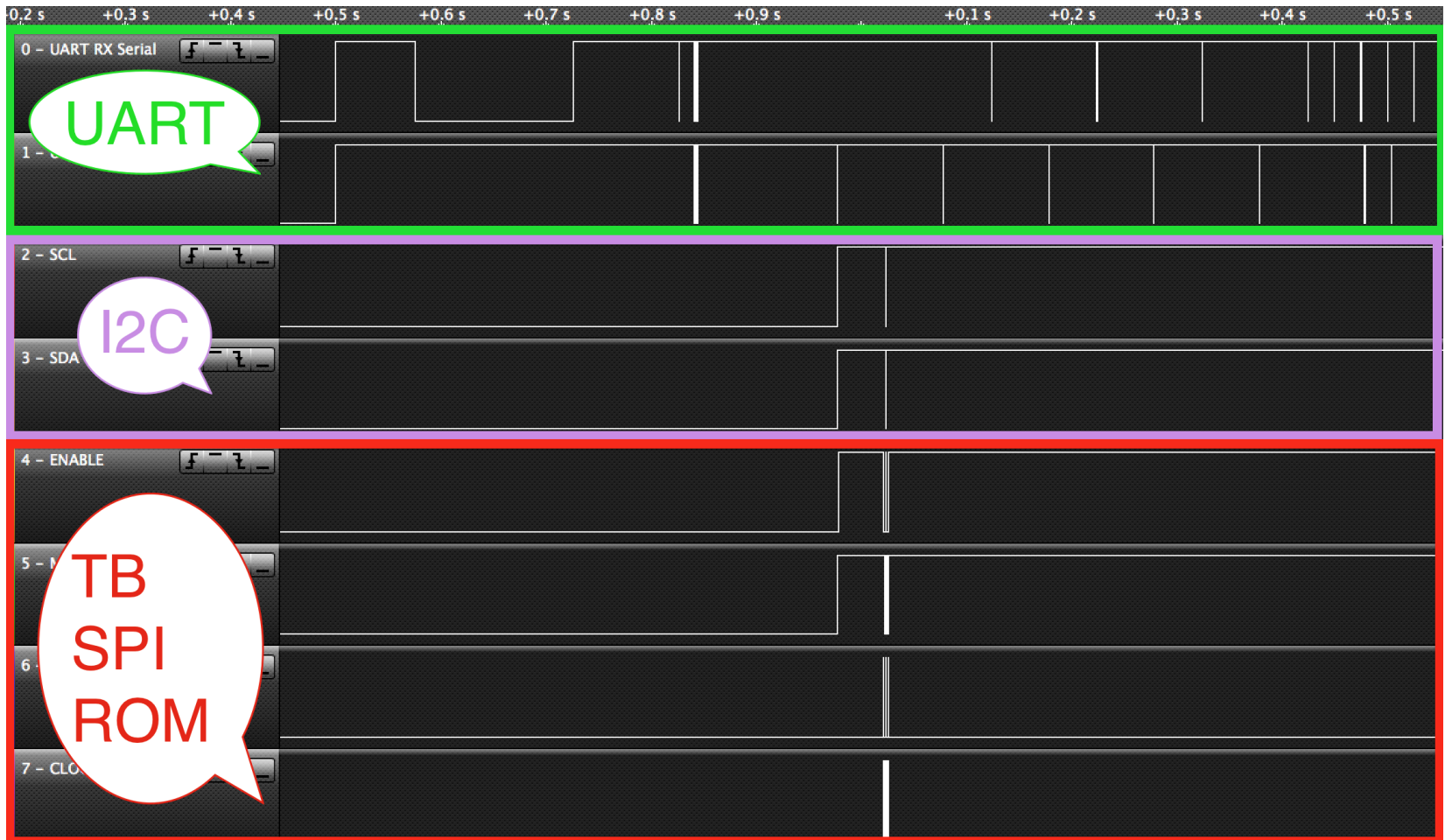




# How My Adventures Went

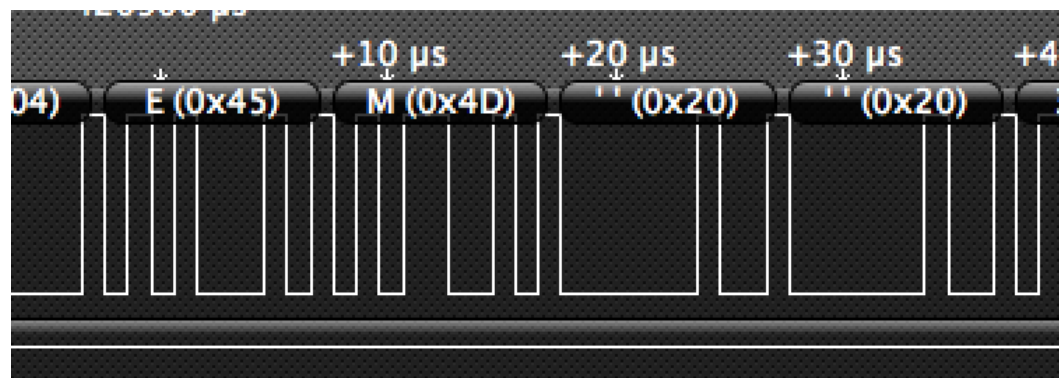
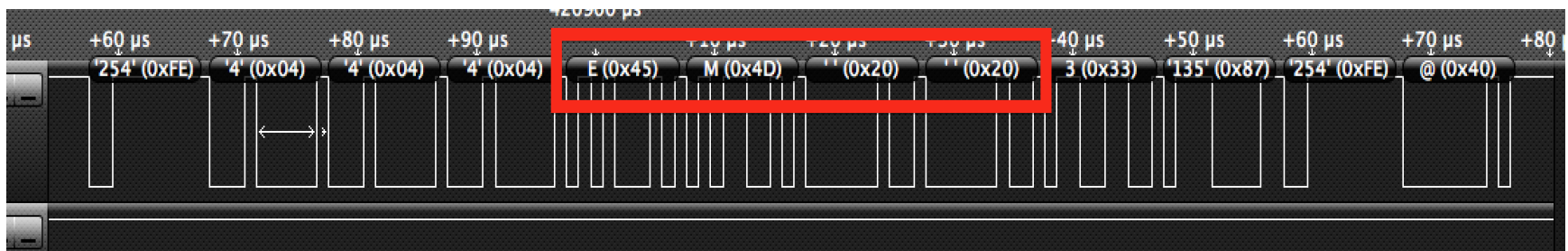


# How My Adventures Went



# How My Adventures Went

- ARM UART Traffic
  - String "EM "



# How My Adventures Went

- Thunderbolt Firmware Update
  - Display contents of Application Package
  - Decompress "Payload" file

```
TBUpdate — bash — 90x29

$ ls
Bom                postinstall
PackageInfo        postinstall_actions
Payload             preinstall
Scripts            preinstall_actions
ThunderboltFirmwareUpdate.pkg

$ file Payload
Payload: gzip compressed data, from Unix

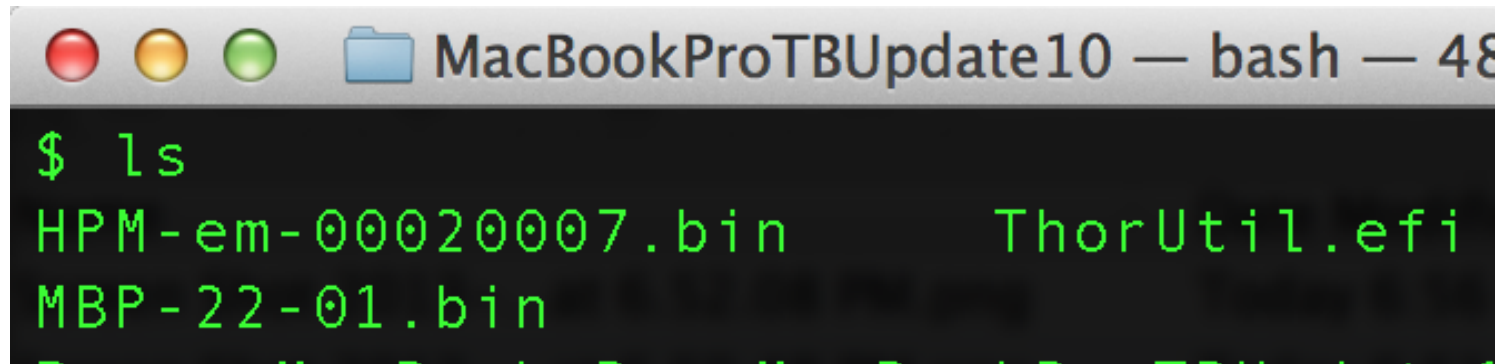
$ mv Payload Payload.tgz

$ tar zxvf Payload.tgz
x .
x ./System
x ./System/Library
x ./System/Library/CoreServices
x ./System/Library/CoreServices/Firmware Updates
x ./System/Library/CoreServices/Firmware Updates/MacBookProTBUpdate10
x ./System/Library/CoreServices/Firmware Updates/MacBookProTBUpdate10/HPM-em-00020007.bin
x ./System/Library/CoreServices/Firmware Updates/MacBookProTBUpdate10/MBP-22-01.bin
x ./System/Library/CoreServices/Firmware Updates/MacBookProTBUpdate10/ThorUtil.efi

$
```

# How My Adventures Went

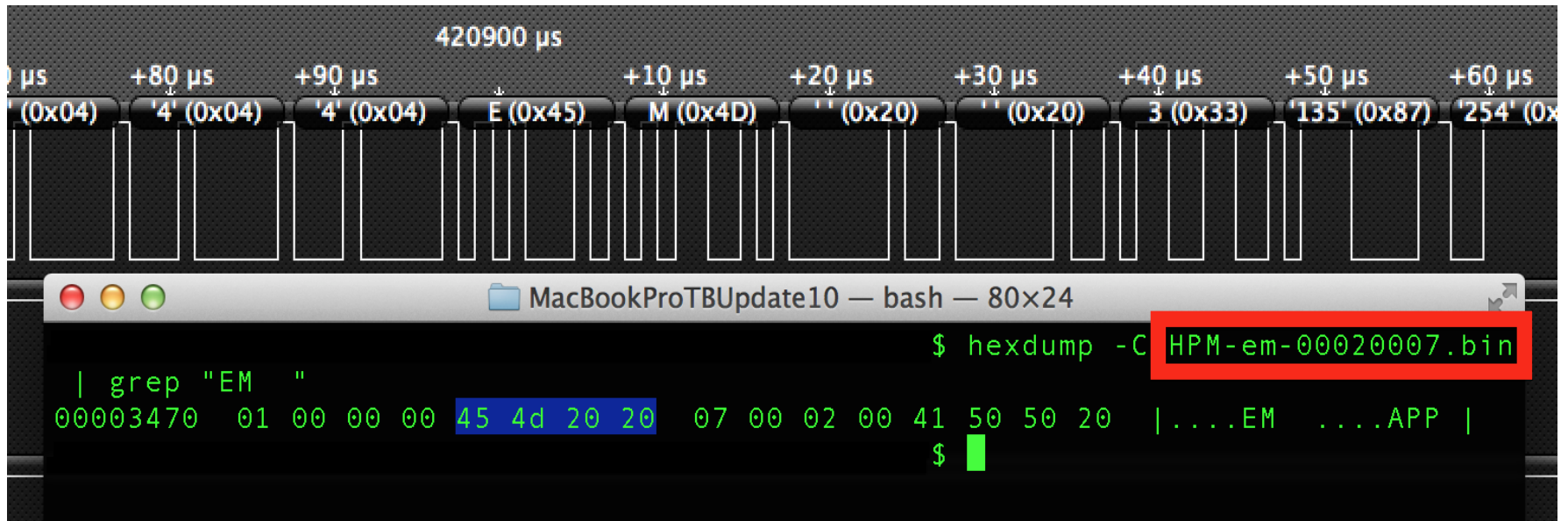
- Two Firmwares for Thunderbolt?
  - One is probably ARM
  - Let's look for string "EM "



```
MacBookProTBUpdate10 — bash — 48
$ ls
HPM-em-00020007.bin      ThorUtil.efi
MBP-22-01.bin
```

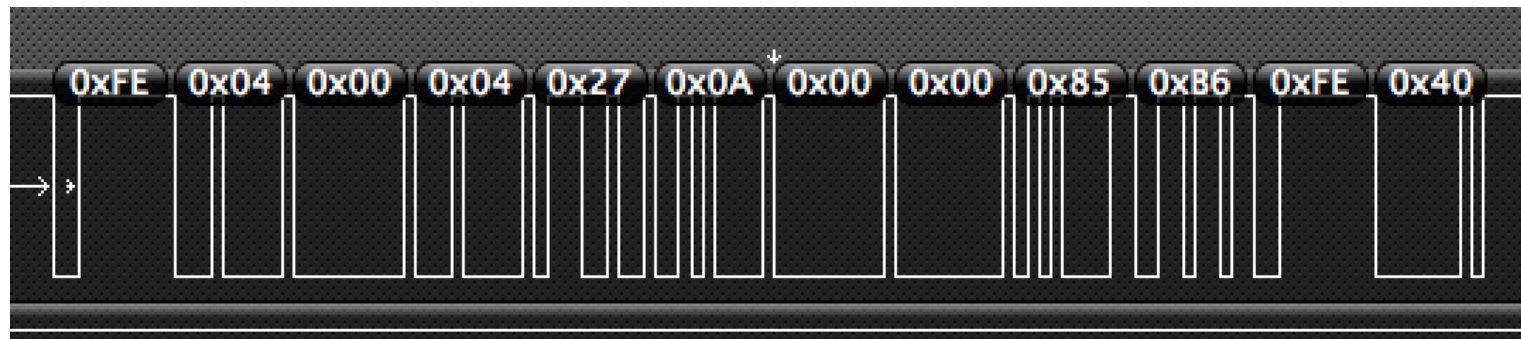
# How My Adventures Went

Jackpot!



# How My Adventures Went

- Round 2...
  - String "\x27\x0a\x00\x00"



# How My Adventures Went

Successaroo!

The image displays two screenshots related to a binary search. The top screenshot is a hex dump of a file, with a red box highlighting the sequence: `' (0x27) \n (0x0A) '0' (0x00) '0' (0x00)`. The bottom screenshot is a terminal window titled "MacBookProTBUupdate10 — bash — 60x25". It shows the command `$ hexdump -C` being executed on the file `HPM-em-00020007.bin`. The terminal output shows the hex dump of the file, with a blue box highlighting the sequence `27 0a 00 00`, which matches the highlighted sequence in the hex dump above.

```
0x00) '4' (0x04) ' (0x27) \n (0x0A) '0' (0x00) '0' (0x00) '133' (0x85) '182
```

```
MacBookProTBUupdate10 — bash — 60x25
```

```
$ hexdump -C
```

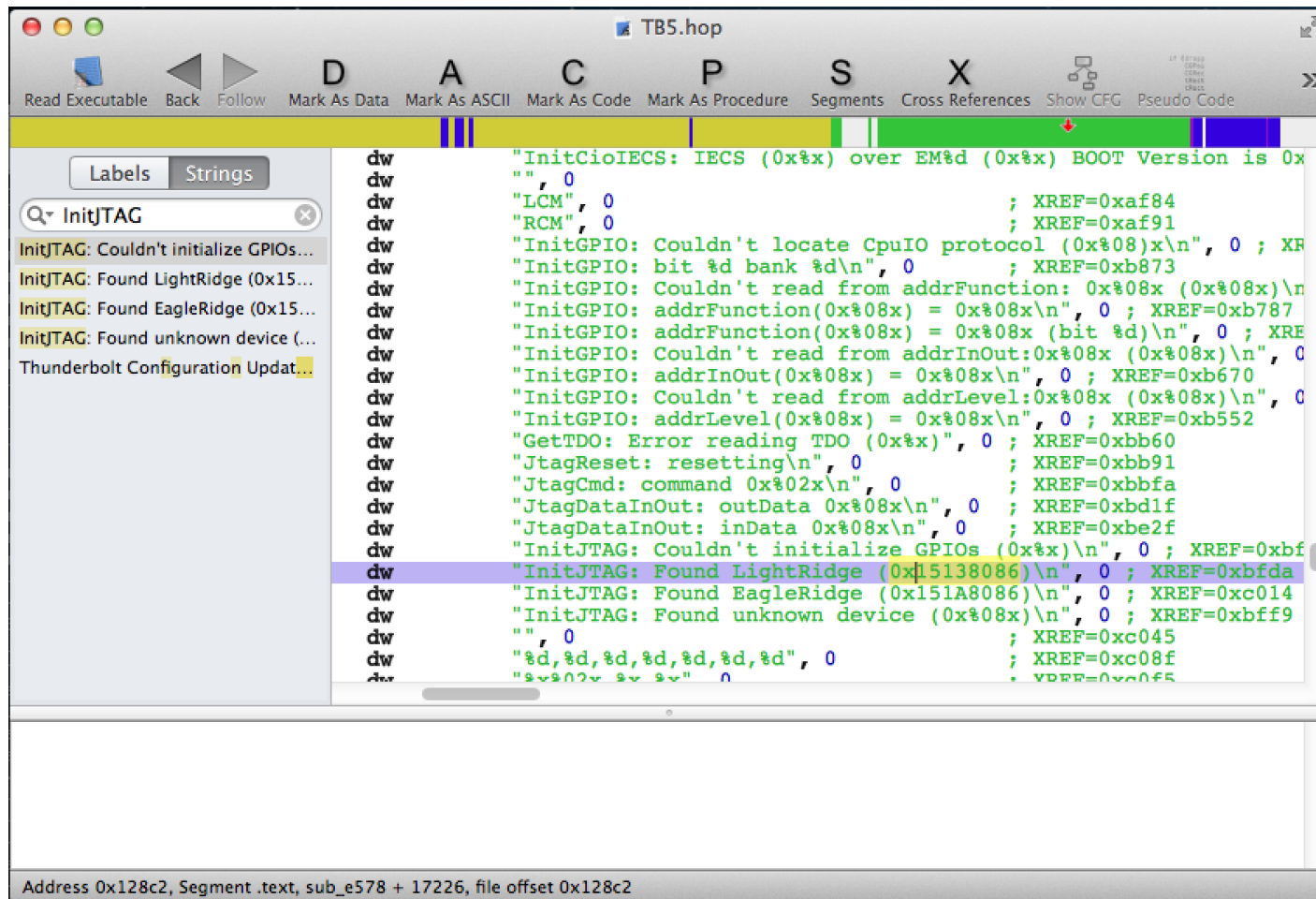
```
HPM-em-00020007.bin | grep "27 0a 00 00"
```

```
00003400 70 02 00 10 18 00 00 00 02 00 00 00 27 0a 00 00
```

```
|p.....'|...|
```



# How My Adventures Went



The screenshot shows a debugger window titled "TB5.hop" with a menu bar including "Read Executable", "Back", "Follow", "Mark As Data", "Mark As ASCII", "Mark As Code", "Mark As Procedure", "Segments", "Cross References", "Show CFG", and "Pseudo Code". The main window displays assembly code with a search filter "InitJTAG" applied to the left-hand side. The search results list several entries, with the following line highlighted:

```
dw "InitJTAG: Found LightRidge (0x15138086)\n", 0 ; XREF=0xbfda
```

The assembly code in the background includes various instructions and strings, such as:

```
dw "InitCioIECS: IECS (0x%x) over EM%d (0x%x) BOOT Version is 0x", 0 ; XREF=0xaf84
dw "LCM", 0 ; XREF=0xaf91
dw "RCM", 0 ; XREF=0xaf91
dw "InitGPIO: Couldn't locate CpuIO protocol (0x%08x)\n", 0 ; XREF=0xb873
dw "InitGPIO: bit %d bank %d\n", 0 ; XREF=0xb873
dw "InitGPIO: Couldn't read from addrFunction: 0x%08x (0x%08x)\n", 0 ; XREF=0xb787
dw "InitGPIO: addrFunction(0x%08x) = 0x%08x\n", 0 ; XREF=0xb787
dw "InitGPIO: addrFunction(0x%08x) = 0x%08x (bit %d)\n", 0 ; XREF=0xb670
dw "InitGPIO: Couldn't read from addrInOut: 0x%08x (0x%08x)\n", 0 ; XREF=0xb670
dw "InitGPIO: addrInOut(0x%08x) = 0x%08x\n", 0 ; XREF=0xb670
dw "InitGPIO: Couldn't read from addrLevel: 0x%08x (0x%08x)\n", 0 ; XREF=0xb552
dw "InitGPIO: addrLevel(0x%08x) = 0x%08x\n", 0 ; XREF=0xb552
dw "GetTDO: Error reading TDO (0x%x)", 0 ; XREF=0xbb60
dw "JtagReset: resetting\n", 0 ; XREF=0xbb91
dw "JtagCmd: command 0x%02x\n", 0 ; XREF=0xbbfa
dw "JtagDataInOut: outData 0x%08x\n", 0 ; XREF=0xbd1f
dw "JtagDataInOut: inData 0x%08x\n", 0 ; XREF=0xbe2f
dw "InitJTAG: Couldn't initialize GPIOs (0x%x)\n", 0 ; XREF=0xbfda
dw "InitJTAG: Found LightRidge (0x15138086)\n", 0 ; XREF=0xbfda
dw "InitJTAG: Found EagleRidge (0x151A8086)\n", 0 ; XREF=0xc014
dw "InitJTAG: Found unknown device (0x%08x)\n", 0 ; XREF=0xbff9
dw " ", 0 ; XREF=0xc045
dw "%d,%d,%d,%d,%d,%d,%d", 0 ; XREF=0xc08f
dw "%x%02x %x %x", 0 ; XREF=0xc0f5
```

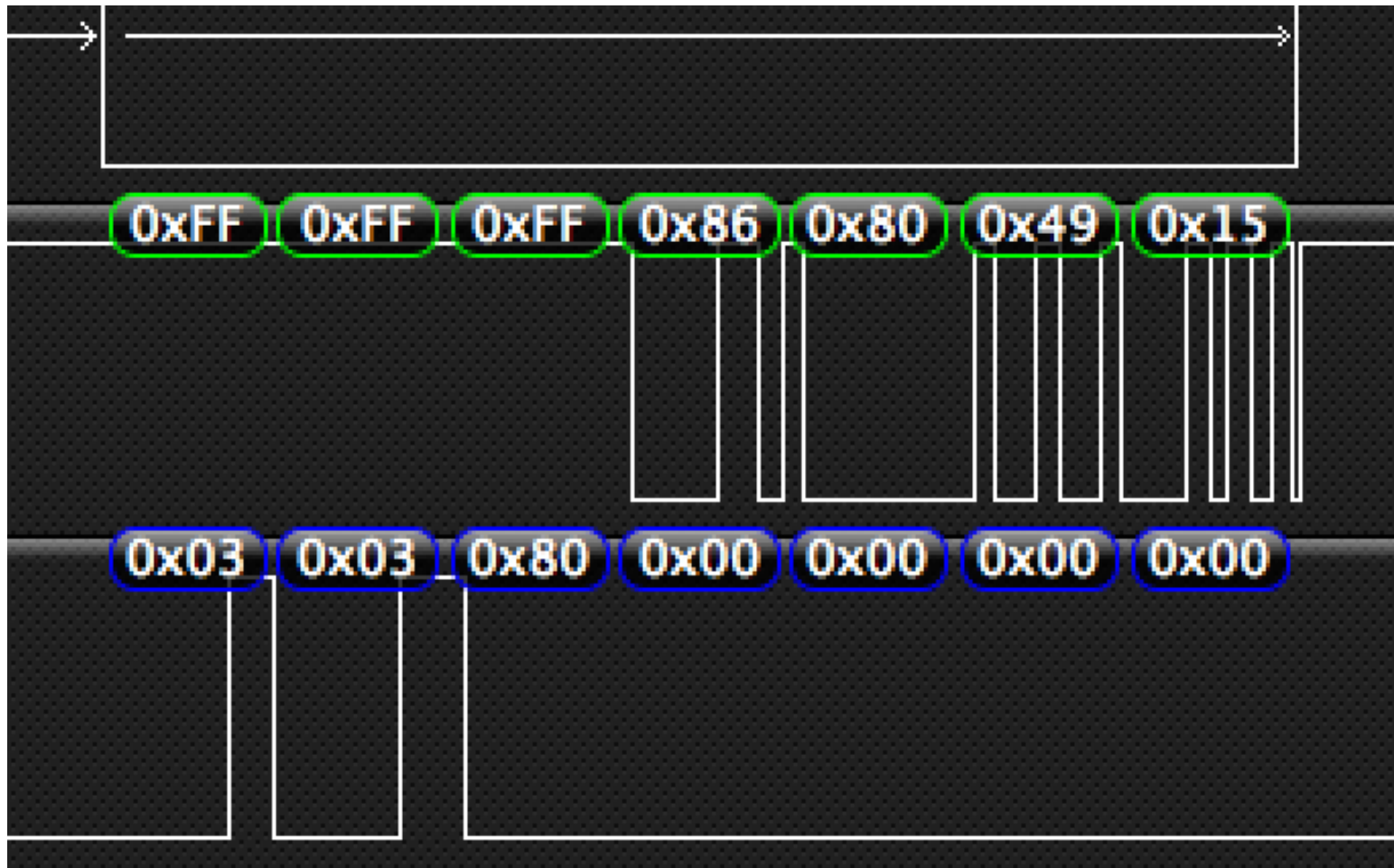
At the bottom of the window, the address is shown as "Address 0x128c2, Segment .text, sub\_e578 + 17226, file offset 0x128c2".

# How My Adventures Went

GitHub, Inc. [US] <https://github.com/aosm/IOPCIFamily/blob/master/IOPCIBridge.cpp>

```
379         // hot plug bridge, but use Legacy if avail
380         uint8_t line = device->configRead8(kIOPCIConfigInterruptLine);
381         if (tunnelLink)
382         {
383             tunnelLink = (0x15138086 != vendorProd)
384                         && (0x151a8086 != vendorProd)
385                         && (0x151b8086 != vendorProd)
386                         && (0x15498086 != vendorProd)
387                         && ((0x15478086 != vendorProd) || ((revIDClass & 0xff) > 1));
388             DLOG("tunnel bridge 0x%08x, %d, msi %d\n",
389                 vendorProd, (revIDClass & 0xff), tunnelLink);
390         }
391         if (tunnelLink || (line == 0) || (line == 0xFF))
392         {
393             // no Legacy ints, need one MSI
394             numVectors = 1;
395         }
```

# How My Adventures Went



# How My Adventures Went

The screenshot shows a debugger window for a file named 'TB5.hop'. The interface includes a menu bar with options like 'Read Executable', 'Back', 'Follow', 'Mark As Data', 'Mark As ASCII', 'Mark As Code', 'Mark As Procedure', 'Segments', 'Cross References', 'Show CFG', 'Pseudo Code', and 'GDB'. A search bar on the left contains the text 'InitJTAG'. Below the search bar, a list of search results is shown, including 'InitJTAG: Couldn't initialize GPIOs (0x%x)\n', 'InitJTAG: Found LightRidge (0x15138086)...', 'InitJTAG: Found EagleRidge (0x151A8086)...', 'InitJTAG: Found unknown device (0x%08x...)', and 'Thunderbolt Configuration Update Utility...'. The main window displays assembly code with columns for address, hex value, disassembly, and comment. The code is divided into two sections: one starting at 0000bfd1 and another starting at 0000c014. The first section contains instructions like 'je sub\_c014', 'cmp eax, 0x15138086', 'jne 0xbff5', 'mov dword [ss:esp+0x4], 0x128c2', and 'mov dword [ss:esp], 0x1'. The second section, labeled 'sub\_c014', contains instructions like 'mov dword [ss:esp+0x4], 0x12914', 'mov dword [ss:esp], 0x1', 'call debug\_output', 'mov eax, 0x10002c', 'mov dword [ds:0x15fe8], eax', 'xor esi, esi', 'mov eax, esi', and 'add esp, 0x14'. The status bar at the bottom indicates 'Address 0xbfda, Segment .text, InitJTAG + 153, file offset 0xbfda'.

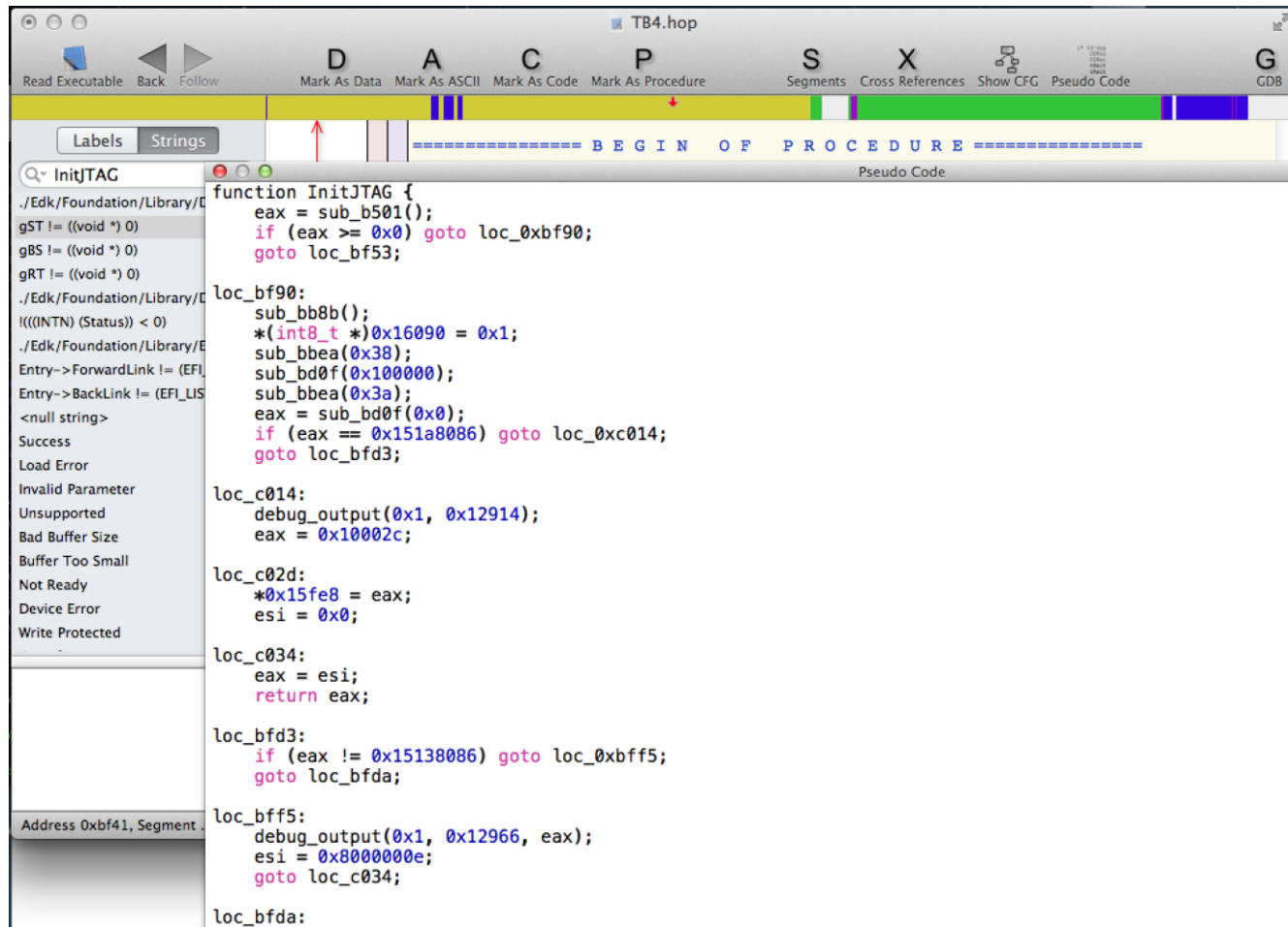
```
0000bfd1 7441          je      sub_c014
0000bfd3 3D86801315   cmp    eax, 0x15138086
0000bfd8 751B        jne    0xbff5
0000bfda C7442404C2280100 mov   dword [ss:esp+0x4], 0x128c2
0000bfe2 C7042401000000 mov   dword [ss:esp], 0x1

000128c2          dw    "InitJTAG: Found LightRidge (0x15138086)\n",
00012914          dw    "InitJTAG: Found EagleRidge (0x151A8086)\n",
00012966          dw    "InitJTAG: Found unknown device (0x%08x)\n",
000129b8          dw    " ", 0 ; XREF
000129ba          dw    "%d,%d,%d,%d,%d,%d", 0 ; XREF
000129e4          dw    "%x%02x.%x.%x", 0 ; XREF
000129fe          dw    "%x.%x.%x", 0 ; XREF
00012a10          dw    "TBGetCioSpiAddress: Found capability @ offset 0x%02x\n", 0 ; XREF
00012aa2          dw    "Found SPI reg at offset 0x%02x\n", 0 ; XREF
00012ae2          dw    "TBGetCioSpiAddress: Couldn't find SPI register\n", 0 ; XREF
00012b4a          dw    "TBGetCioI2CAddress: Found capability @ offset 0x%02x\n", 0 ; XREF
00012bdc          dw    "Found I2C capability base at offset 0x%02x\n", 0 ; XREF

sub_c014:
0000c014 C744240414290100 mov   dword [ss:esp+0x4], 0x12914
0000c01c C7042401000000 mov   dword [ss:esp], 0x1
0000c023 E8081C0000   call  debug_output
0000c028 B82C001000   mov   eax, 0x10002c
0000c02d A3E85F0100   mov   dword [ds:0x15fe8], eax
0000c032 31F6        xor   esi, esi
0000c034 89F0        mov   eax, esi
0000c036 83C414     add   esp, 0x14
```

Address 0xbfda, Segment .text, InitJTAG + 153, file offset 0xbfda

# How My Adventures Went



```
function InitJTAG {
    eax = sub_b501();
    if (eax >= 0x0) goto loc_0xbf90;
    goto loc_bf53;
}

loc_bf90:
    sub_bb8b();
    *(int8_t *)0x16090 = 0x1;
    sub_bbea(0x38);
    sub_bd0f(0x100000);
    sub_bbea(0x3a);
    eax = sub_bd0f(0x0);
    if (eax == 0x151a8086) goto loc_0xc014;
    goto loc_bfd3;

loc_c014:
    debug_output(0x1, 0x12914);
    eax = 0x10002c;

loc_c02d:
    *0x15fe8 = eax;
    esi = 0x0;

loc_c034:
    eax = esi;
    return eax;

loc_bfd3:
    if (eax != 0x15138086) goto loc_0xbff5;
    goto loc_bfda;

loc_bff5:
    debug_output(0x1, 0x12966, eax);
    esi = 0x8000000e;
    goto loc_c034;

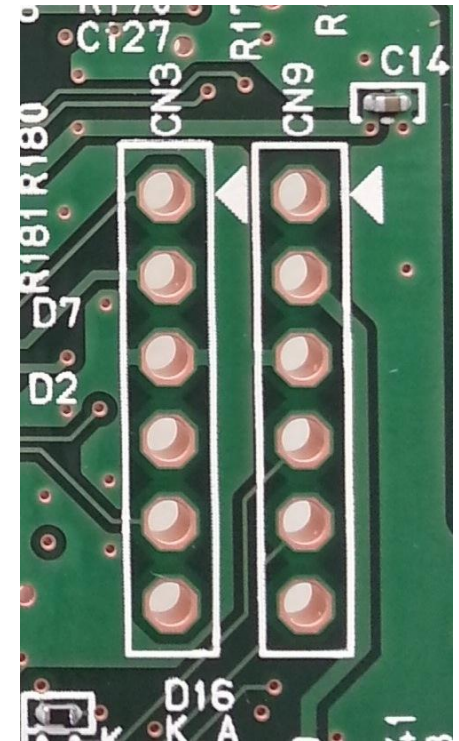
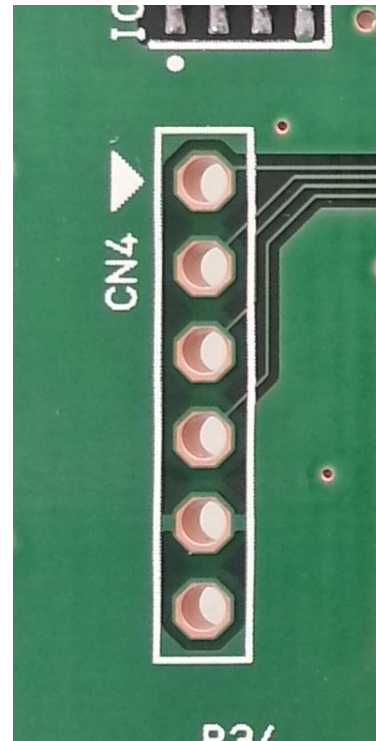
loc_bfda:
```

# How My Adventures Went

```
function InitJTAG {
    eax = InitGPIO();
    if (eax >= 0x0) {
        sub_bb8b();
        *(int8_t *)0x16090 = 0x1;
        sub_bbea(0x38);
        sub_bd0f(0x100000);
        sub_bbea(0x3a);
        eax = sub_bd0f(0x0);
        if (eax == 0x151a8086) {
            debug_output(0x1, "InitJTAG: Found EagleRidge (0x151A8086)\n");
            eax = 0x10002c;
        }
        else if (eax != 0x15138086) {
            debug_output(0x1, "InitJTAG: Found unknown device (0x%08x)\n", eax);
            esi = 0x8000000e;
        }
        else {
            debug_output(0x1, "InitJTAG: Found LightRidge (0x15138086)\n");
            eax = 0x100028;
        }
        *0x15fe8 = eax;
        esi = 0x0;
    }
    else
    {
        eax = debug_output(0x7, "InitJTAG: Couldn't initialize GPIOs (0x%x)\n", esi);
        if (**0x14a84 <= 0x6) {
            *(int16_t *)(*0x14a88 + eax * 0x2) = 0x100;
        }
        *(*0x14a88 + 0x10) = esi;
    }
    return esi;
}
```

# How My Adventures Went

- Header rows
  - CN<sub>3</sub> = LED Voltages
  - CN<sub>9</sub> = Power rails
  - CN<sub>4</sub> = JTAG!



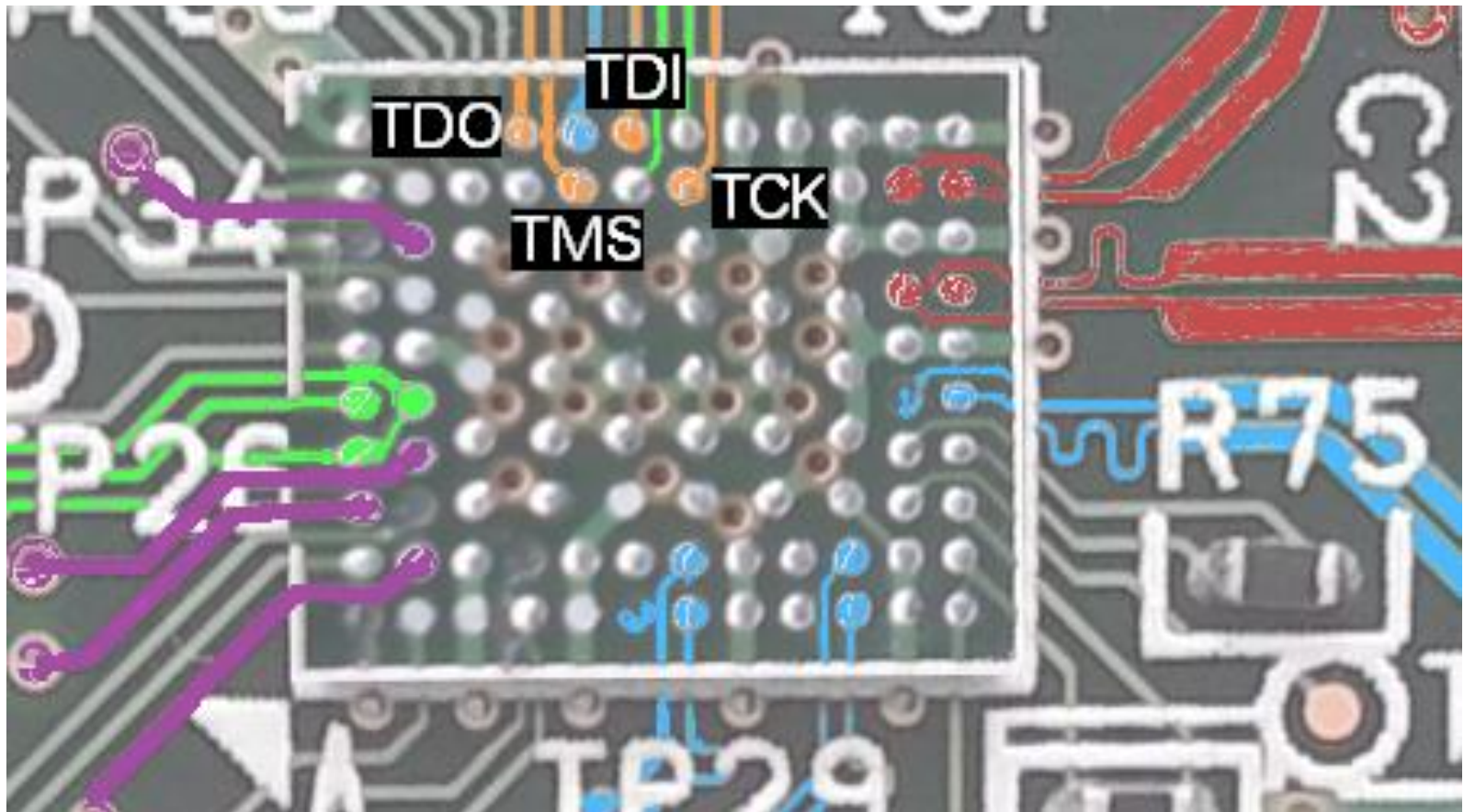
# How My Adventures Went

- JTAG Reversing
  - Trail and error
  - They make devices for this
  - Pin 1 => TCK
  - Pin 2 => TDI
  - Pin 3 => TMS
  - Pin 4 => TDO

```
jtag> cable ft2232 vid=0x0403 pid=0x8a98
Connected to libftdi driver.
jtag> idcode
Reading 0 bytes of idcode
Read 00000000(0x00) 00000000(0x00) 00000000(0x00) 00000000(0x00)
jtag> detect
IR length: 6
Chain length: 1
Device Id: not supported (bit 0 was not a 1)
jtag> █
```



# How My Adventures Went



■ JTAG

# How My Adventures Went

- JTAG Reversing
  - ARM Cortex Mo uses SWD, not JTAG...
  - Obfuscated with a start sequence
  - We saw this code earlier...

```
jtag> discovery
Detecting IR length ... 6
Detecting DR length for IR 111111 ... 1
Detecting DR length for IR 000000 ... 25
Detecting DR length for IR 000001 ... 25
Detecting DR length for IR 000010 ... warning: TDO seems to be stuck at 1
-1
Detecting DR length for IR 000011 ... warning: TDO seems to be stuck at 1
-1
```

# How My Adventures Went

- Thunderbolt Device ROM

- Vendor Name
- Device Name
- Vendor ID
- Device ID
- Device Revision
- UID

HD-PATU3:

Vendor Name:	BUFFALO INC.
Device Name:	HD-PATU3
Vendor ID:	0x29
Device ID:	0x1
Device Revision:	0x1
UID:	0x002900010005A0E0
Route String:	3
Firmware Version:	5.1
Port:	
Status:	Device connected
Link Status:	0x2
Port Micro Firmware Version:	0.2.5
Cable Firmware Version:	0.1.24



# How My Adventures Went

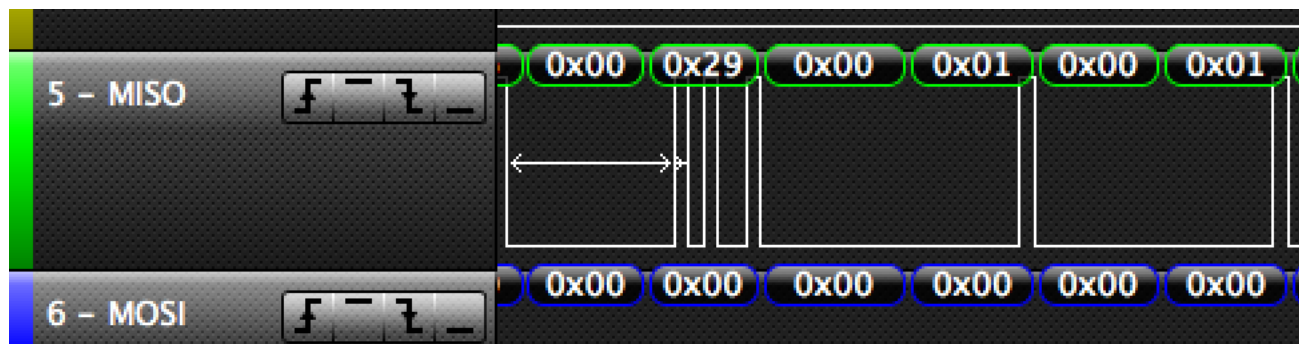
- Thunderbolt Device ROM

- Vendor Name
- Device Name
- **Vendor ID**
- **Device ID**
- **Device Revision**
- UID

**HD-PATU3:**

Vendor Name: BUFFALO INC.  
Device Name: HD-PATU3  
Vendor ID: 0x29  
Device ID: 0x1  
Device Revision: 0x1  
UID: 0x002900010005A0E0  
Route String: 3  
Firmware Version: 5.1  
Port:

Status: Device connected  
Link Status: 0x2  
Port Micro Firmware Version: 0.2.5  
Cable Firmware Version: 0.1.24



# How My Adventures Went

- Thunderbolt Device ROM

- Vendor Name
- Device Name
- Vendor ID
- Device ID
- Device Revision
- **UID**

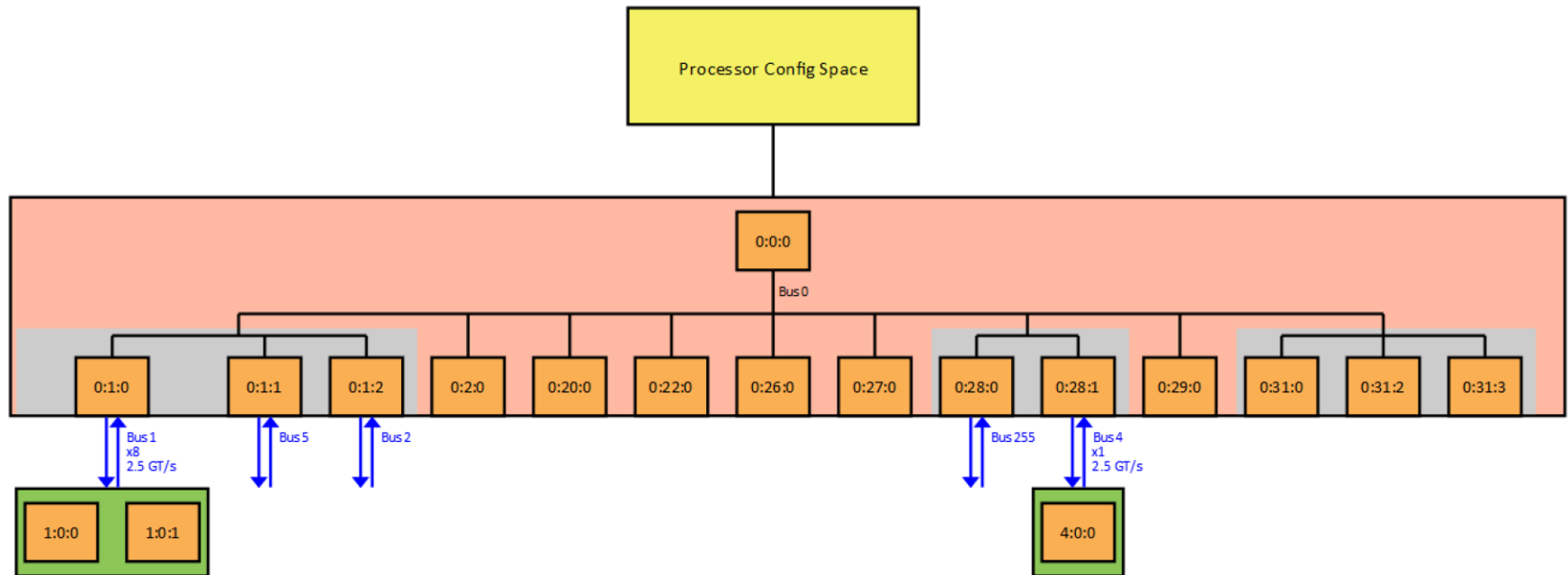
HD-PATU3:

Vendor Name: BUFFALO INC.  
Device Name: HD-PATU3  
Vendor ID: 0x29  
Device ID: 0x1  
Device Revision: 0x1  
UID: 0x002900010005A0E0  
Route String: 3  
Firmware Version: 5.1  
Port:  
Status: Device connected  
Link Status: 0x2  
Port Micro Firmware Version: 0.2.5  
Cable Firmware Version: 0.1.24

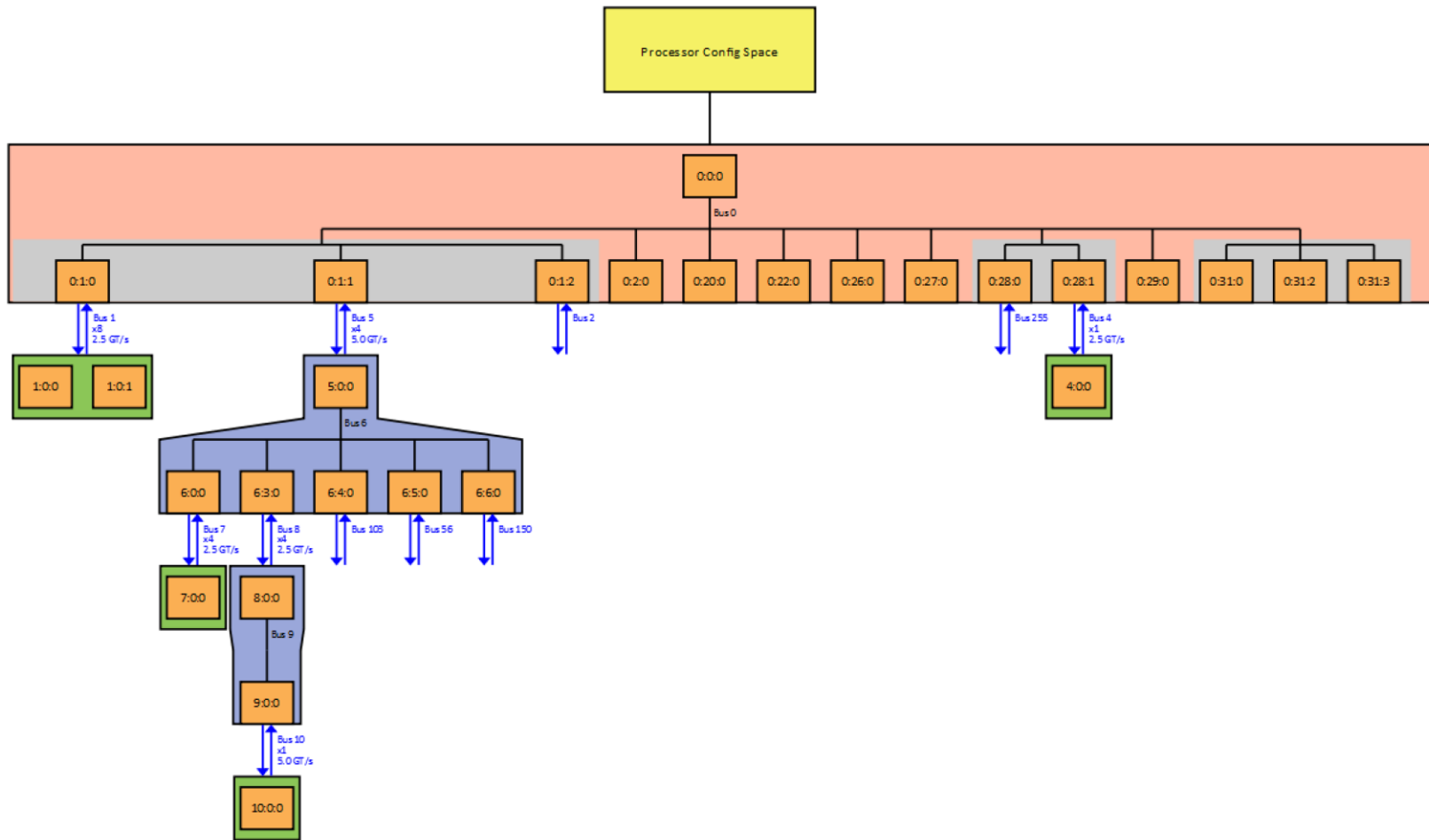


# How My Adventures Went

- PCIe Info
  - Thunderbolt is a PCIe Bridge



# How My Adventures Went



# How My Adventures Went

## Type 1 Header

Device ID 15 49		Vendor ID 80 86		00h
Status 00 10		Command 00 07		04h
Class Code 06 04 00			RevID 00	08h
BIST 00	HdrType 01	LatTimer 00	\$LineSze 20	0Ch
BAR0 00 00 00 00				10h
BAR1 00 00 00 00				14h
SecLaTmr 00	SubBus# 0A	SecBus# 0A	PriBus# 09	18h
Secondary Status 00 00		IO Limit 41	IO Base 41	1Ch
Memory Limit 8F B0		Memory Base 8F B0		20h
Prefetch Mem Limit 00 01		Prefetch Mem Base 00 01		24h



# How My Adventures Went

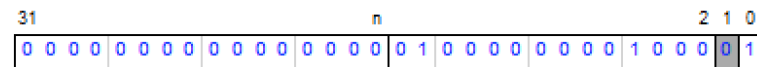
B:D:F	Class	Device Description	Device Type
2! 0:1:1 (0,5,198)	PCI/PCI bridge	Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor PCI...	Root Port
2! 5:0:0 (5,6,198)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Port [Cactus Ridge]	Switch Upstream Port
6:0:0 (6,7,7)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Port [Cactus Ridge]	Switch Downstream Port
7:0:0	System peripheral	Intel Corporation DSL3510 Thunderbolt Port [Cactus Ridge]	PCIe Endpoint
2! 6:3:0 (6,8,10)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Port [Cactus Ridge]	Switch Downstream Port
8:0:0 (8,9,10)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Controller [Cactus Ridg...	Switch Upstream Port
9:0:0 (9,10,10)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Controller [Cactus Ridg...	Switch Downstream Port
10:0:0	SATA controller - AHCI 1.0	ASMedia Technology Inc. ASM1062 Serial ATA Controller	Leg. PCIe Endpoint
6:4:0 (6,103,103)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Port [Cactus Ridge]	Switch Downstream Port
6:5:0 (6,58,58)	PCI/PCI bridge	Intel Corporation DSL3510 Thunderbolt Port [Cactus Ridge]	Switch Downstream Port

decode raw | **10:0:0 - ASMedia Technology Inc. ASM1062 Serial ATA Controller**

### Type 0 Header

Device ID 06 12		Vendor ID 1B 21		00h
Status 00 10		Command 04 06		04h
Class Code 01 06 01			RevID 01	08h
BIST 00	HdrType 00	LatTimer 00	\$LineSize 20	0Ch
BAR0 00 00 40 21				10h
BAR1				

### Base Address Register 0



BAR0

Address: 4020h  
Type: IO

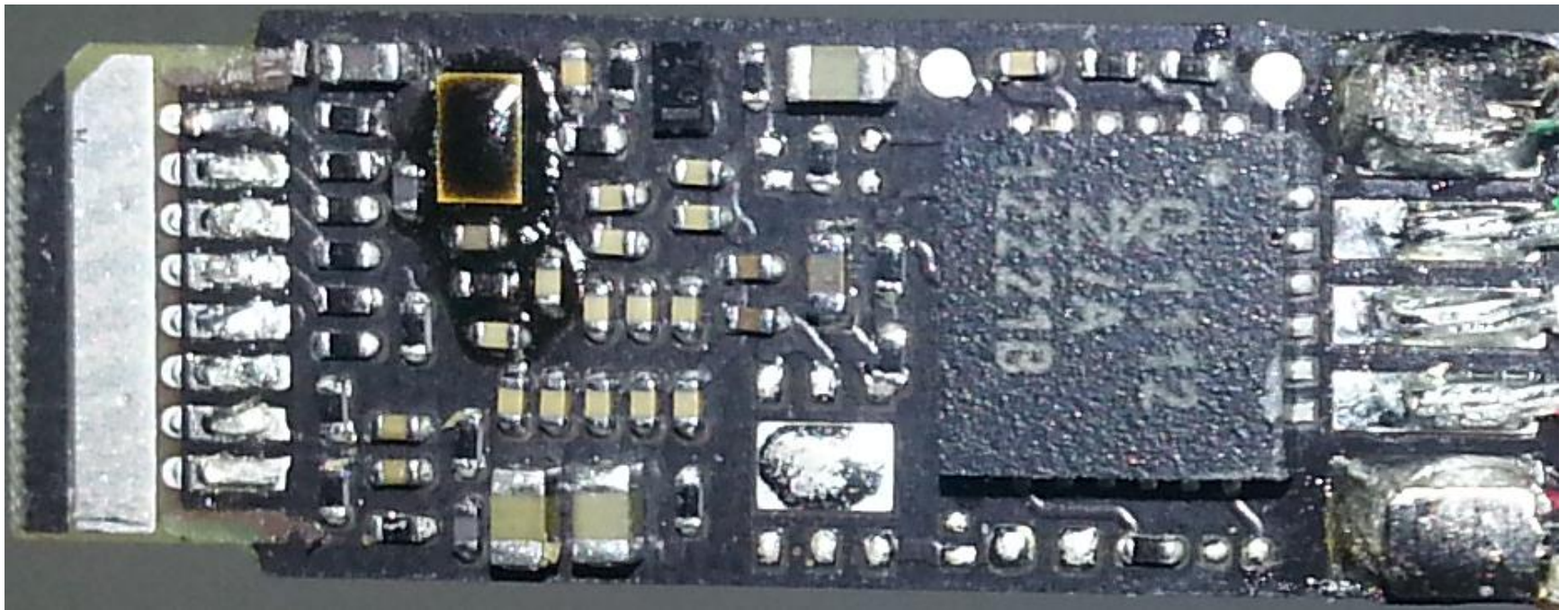
The BARs are used to determine what type of address space(s) each function may need and requested by that BAR. If a function supports decoding addresses larger than 32 bits, two con

# How My Adventures Went

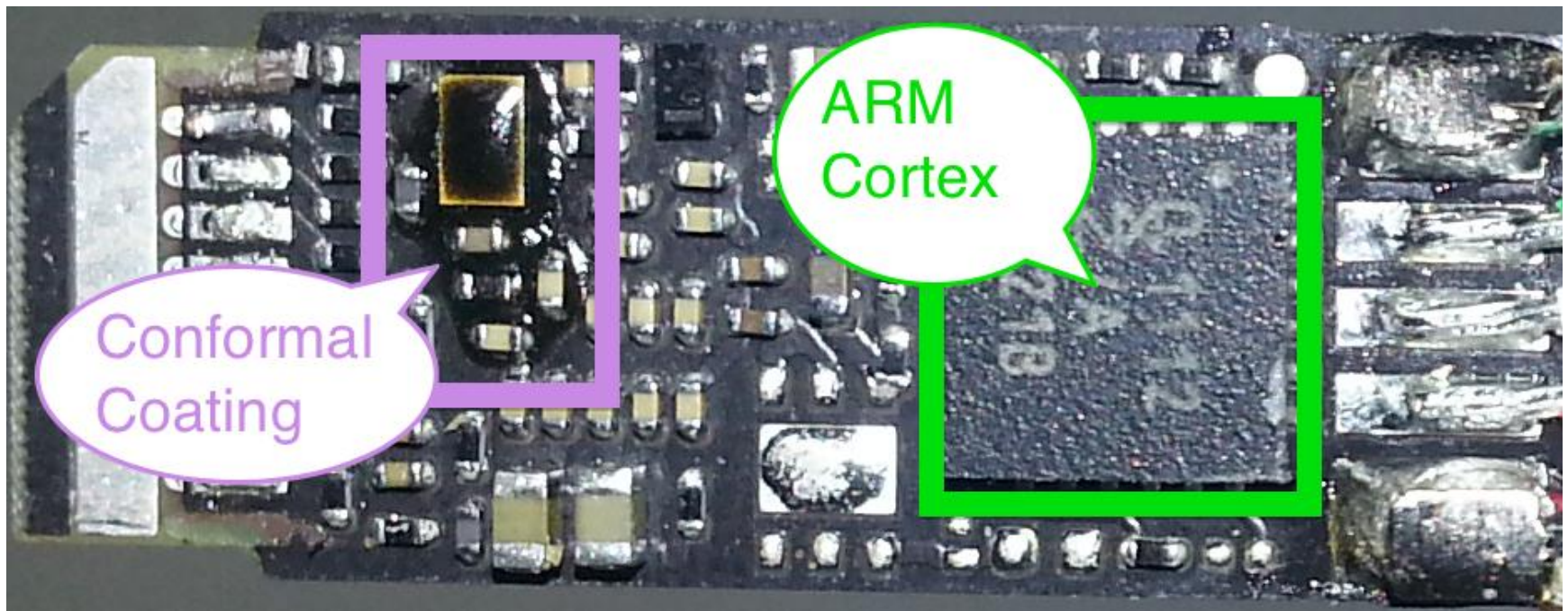
- Gigabit Ethernet Adapter
  - Researching the product
  - Taking it apart
  - Attack vectors



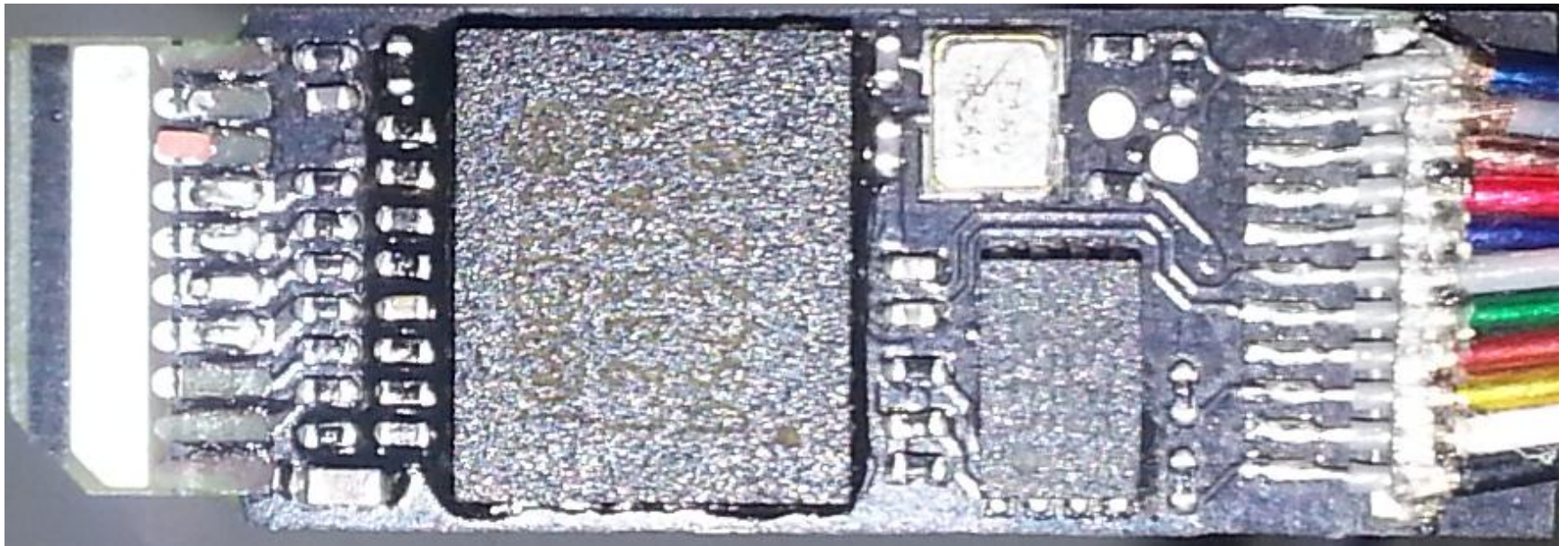
# How My Adventures Went



# How My Adventures Went



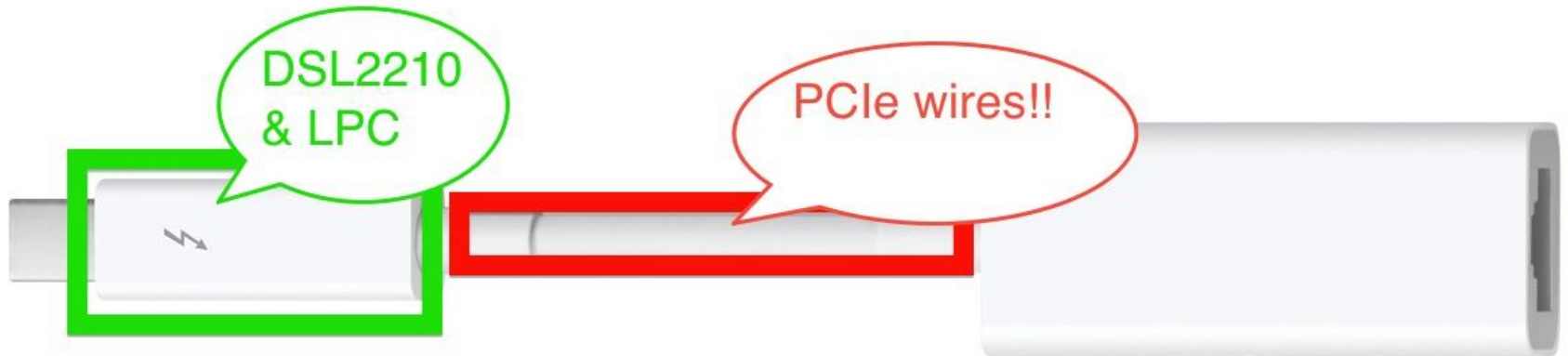
# How My Adventures Went



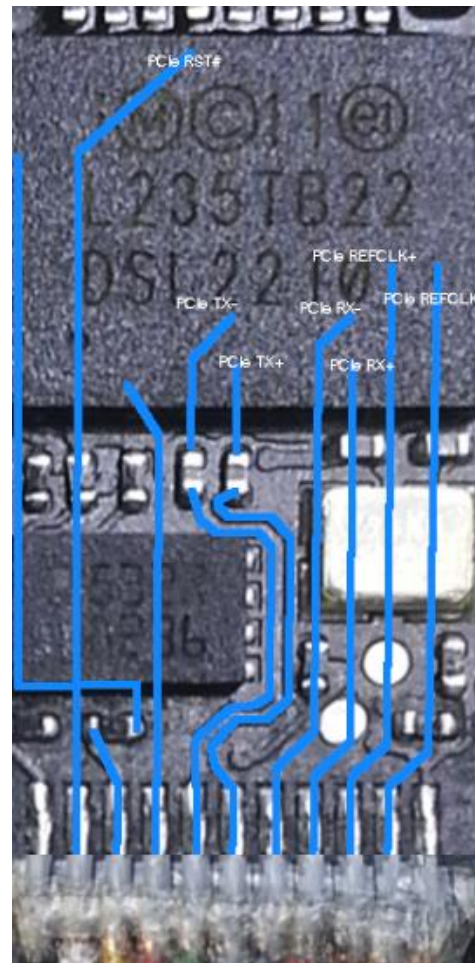
# How My Adventures Went



# How My Adventures Went

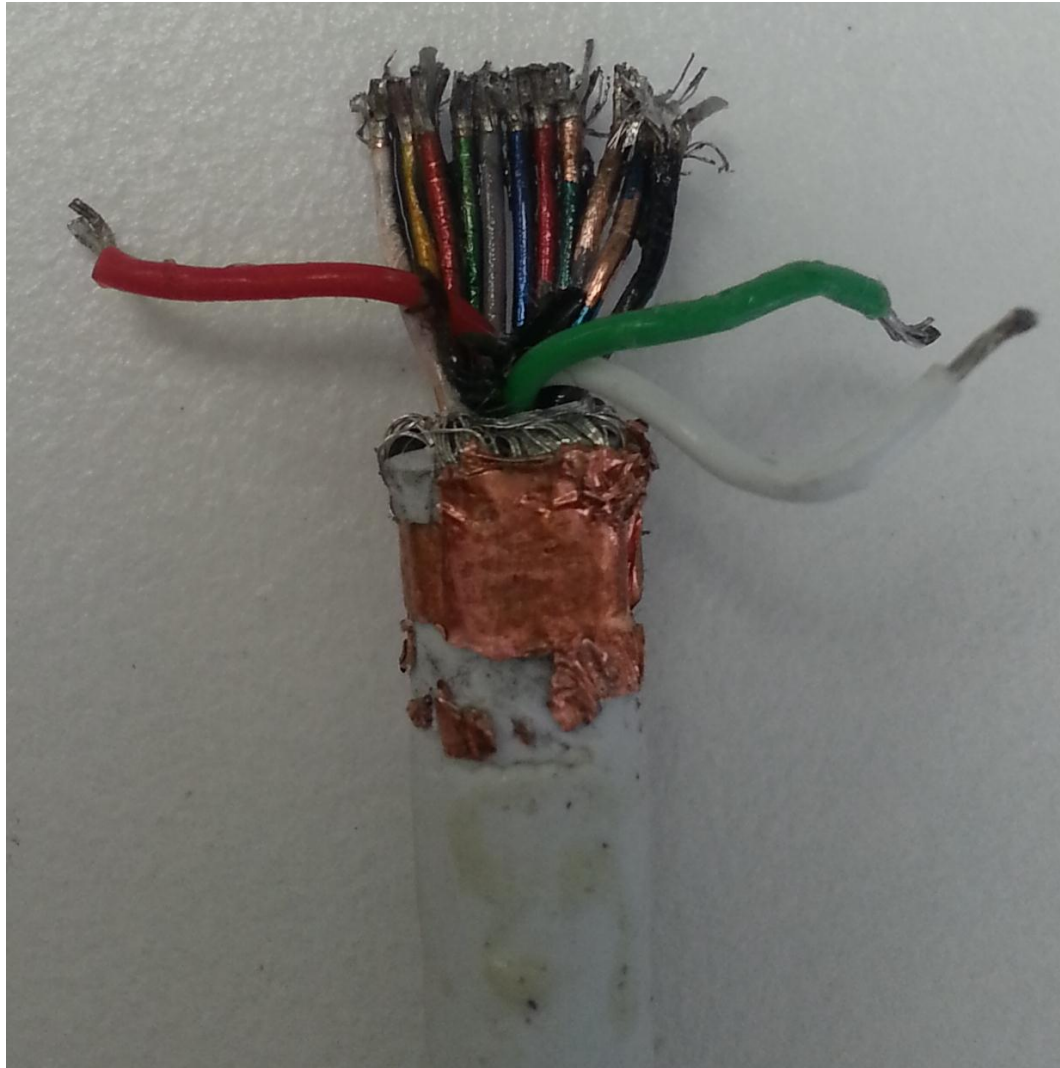


# How My Adventures Went

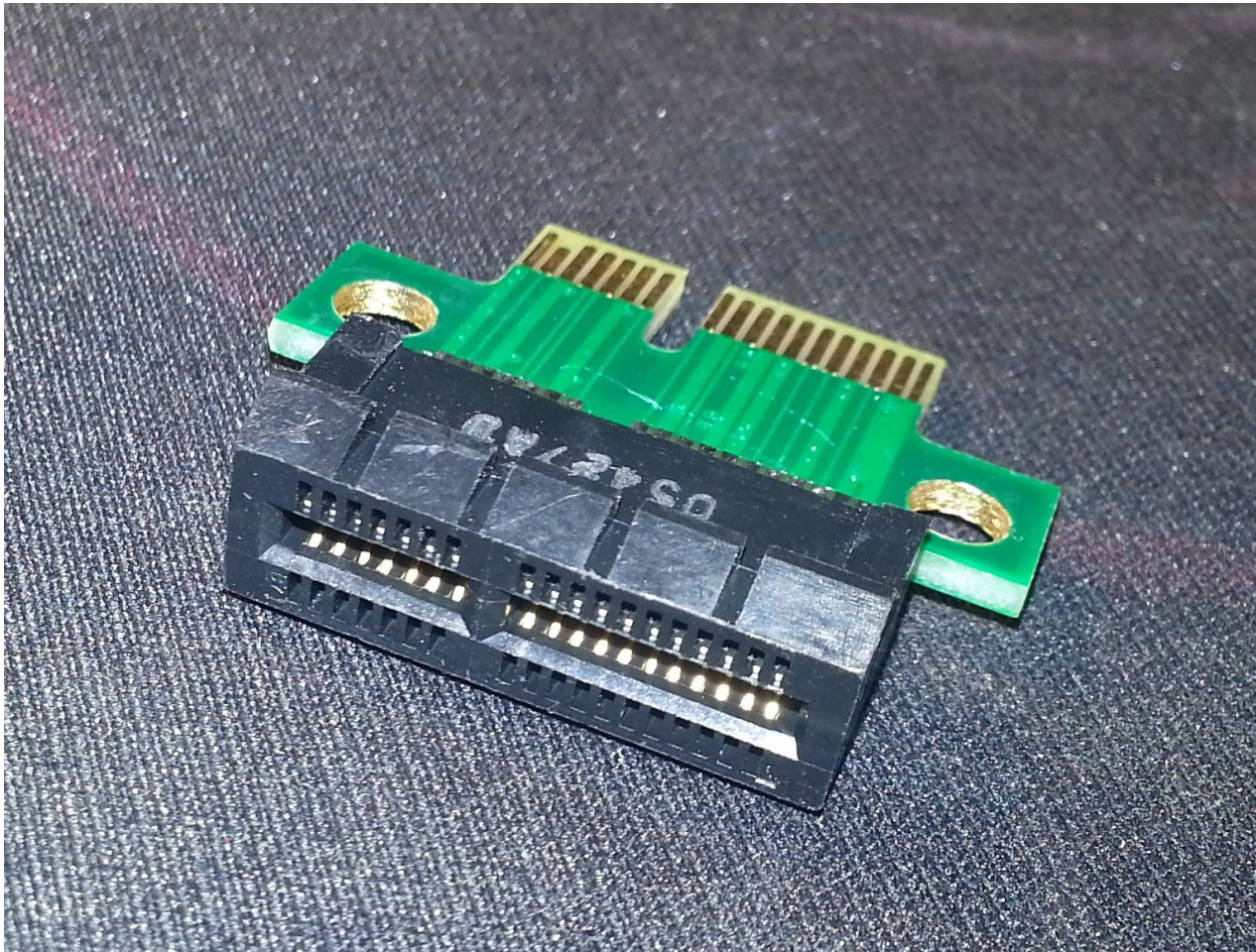




# How My Adventures Went



# How My Adventures Went

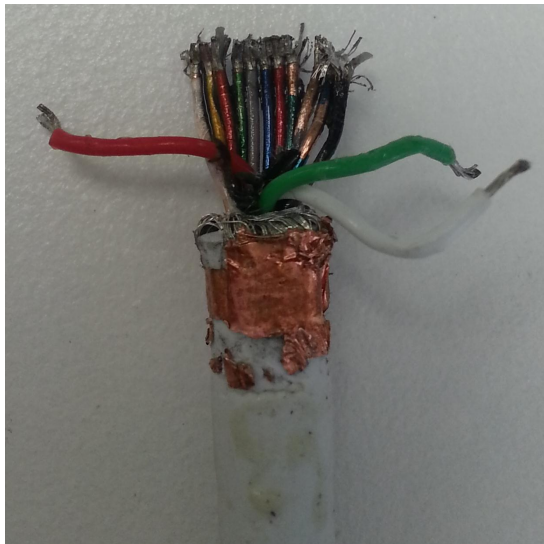
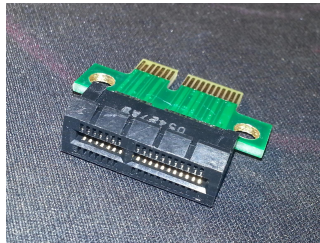


# How My Adventures Went

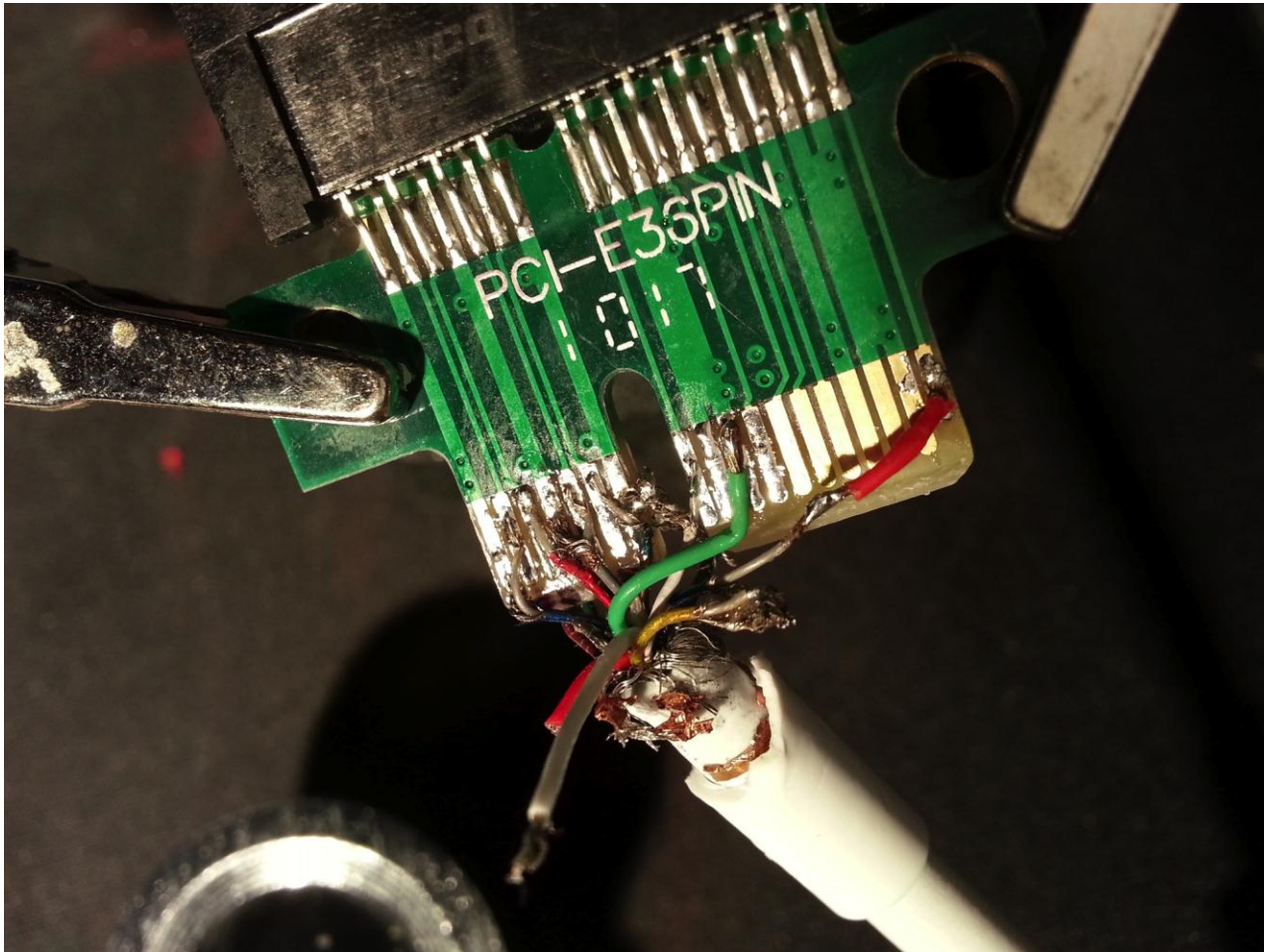
- Altera Cyclone IV GX Transceiver Starter Kit
  - Hard IP for PCIe
  - PCIe x1
  - ~\$400



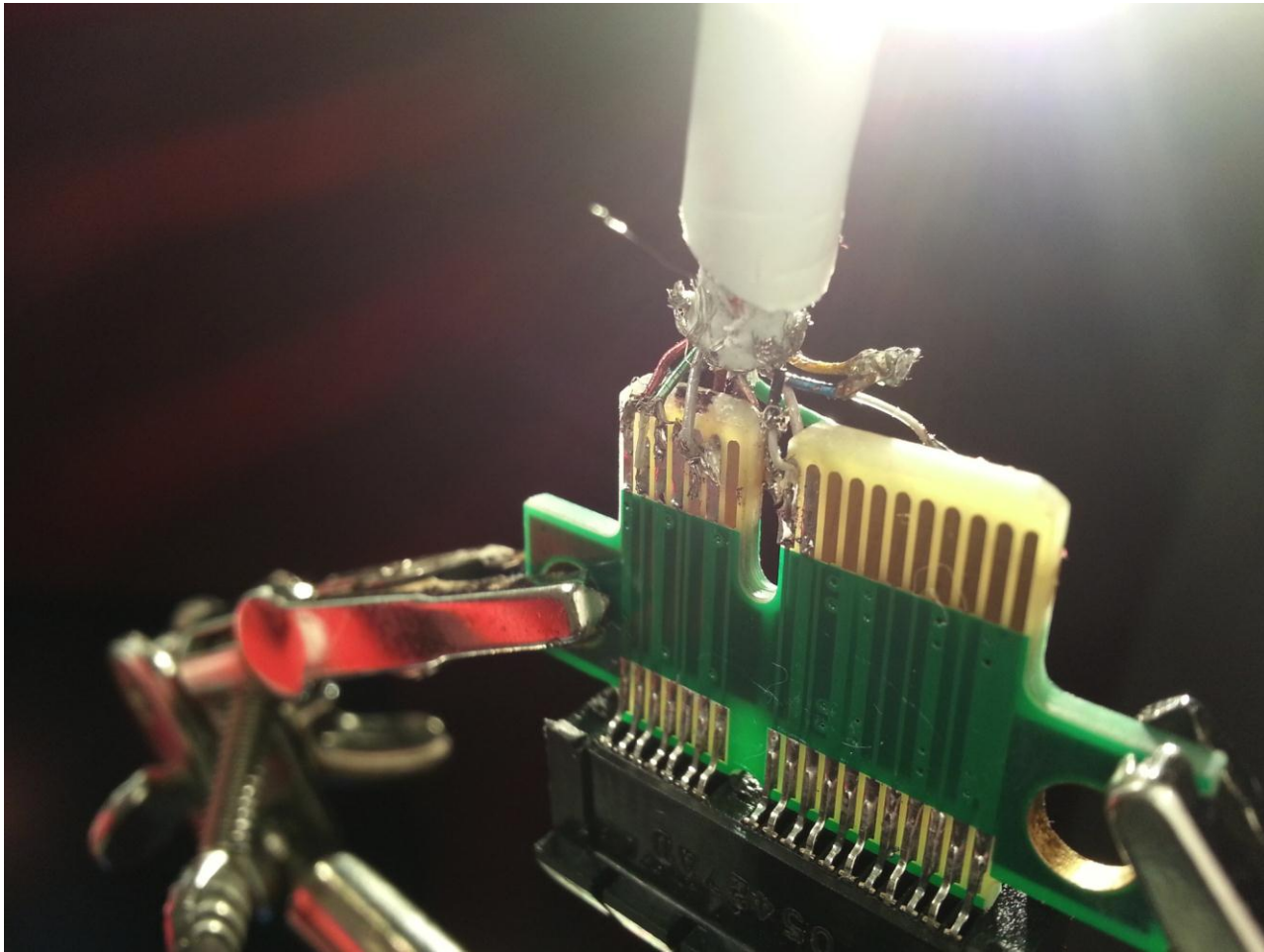
# How My Adventures Went



# How My Adventures Went



# How My Adventures Went



# How My Adventures Went

---

- Tips and Tricks
  - Get A LOT of devices!
  - Heat up everything SLOWLY!
  - Continuity testing WINS
  - Sniff EVERYTHING
  - Read all ROMs/Flashes

# Thank You

- Russ Sevinsky
  - Security Consultant at iSEC Partners
  - [rsevinsky@isecpartners.com](mailto:rsevinsky@isecpartners.com)
- Special thanks to:
  - Everyone @ iSEC, specifically
    - Jesse Burns
    - Mike Warner
  - Craig Heffner





# References

- 1.) "I/O Attacks in Intel-PC Architectures and Countermeasures"  
<http://www.syssec-project.eu/media/page-media/23/syssec2011-s1.4-sang.pdf>
- 2.) "Understanding DMA Malware":  
<http://www.stewin.org/papers/dimvap15-stewin.pdf>
- 3.) "Adventures with Daisy in Thunderbolt-DMA-land: Hacking Macs through the Thunderbolt interface":  
<http://www.breaknenter.org/2012/02/adventures-with-daisy-in-thunderbolt-dma-land-hacking-macs-through-the-thunderbolt-interface/>



# References

4.) Inception: <http://www.breaknenter.org/projects/inception/>

5.) "De Mysteriis Dom Jobsivs" Blackhat Paper:  
[http://reverse.put.as/wp-content/uploads/2011/06/De\\_Mysteriis\\_Dom\\_Jobsivs\\_Black\\_Hat\\_Paper.pdf](http://reverse.put.as/wp-content/uploads/2011/06/De_Mysteriis_Dom_Jobsivs_Black_Hat_Paper.pdf)

6.) "Protecting yourself against Firewire DMA attacks on 10.7.x":  
<http://derflounder.wordpress.com/2012/02/05/protecting-yourself-against-firewire-dma-attacks-on-10-7-x/>

7.) Anandtech Buffalo Ministation  
Review:<http://www.anandtech.com/show/6127/buffalo-ministation-thunderbolt-review-an-external-with-usb-30-and-thunderbolt>





**UK Offices**

Manchester - Head Office  
Cheltenham  
Edinburgh  
Leatherhead  
London  
Thame



**North American Offices**

San Francisco  
Atlanta  
New York  
Seattle



**Australian Offices**

Sydney

**European Offices**

Amsterdam - Netherlands  
Munich – Germany  
Zurich - Switzerland