This article is written for people from a technical background who are seeking to retrofit Thunderbolt to older / low end PCs. It does not assist with basic Thunderbolt troubleshooting.

Presently everything here concerns Thunderbolt 3 add-in cards. The situation is no better for Thunderbolt 4 however many technical details differ from what is written on this page.

I’ve long been intrigued by Thunderbolt add-in cards. Apparently regular looking PCIe expansion cards, but shipped with a mystery interface cable to the motherboard, of which there is a small list of supported models.

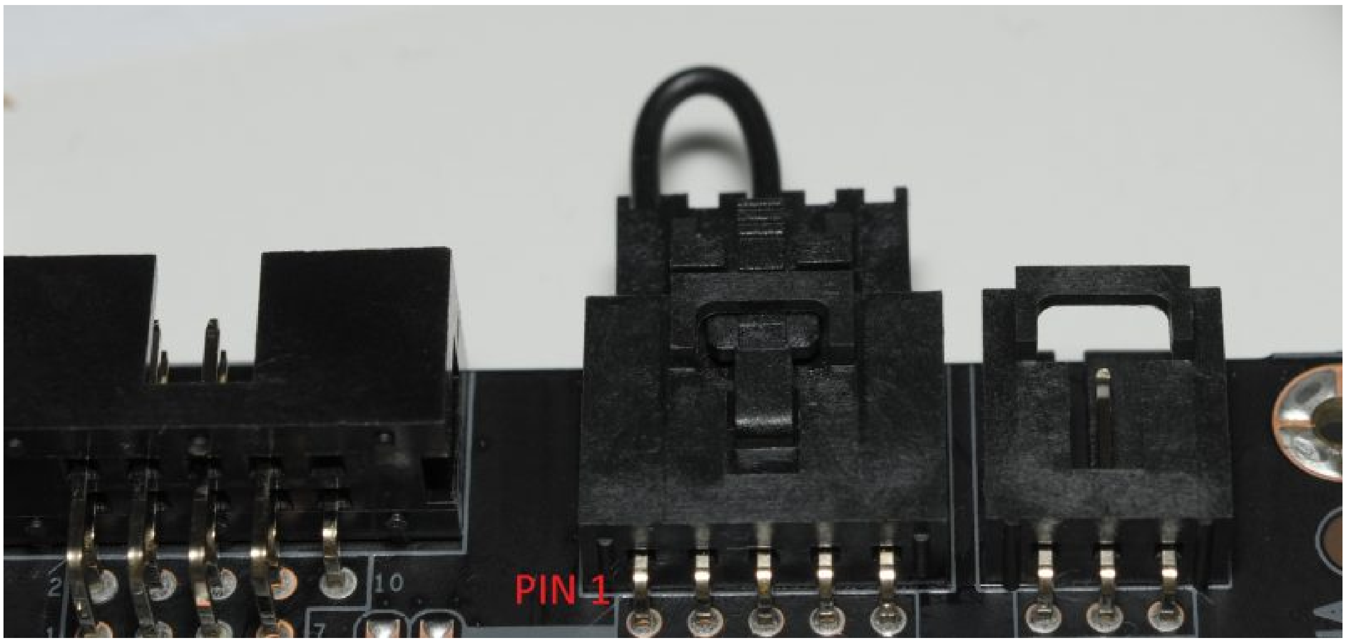
It’s not a secret that these cards *may* work in a motherboard which isn’t supported, but full functionality is not a given. I have spent the past few evenings trawling through many forums, reading about the many different experiences people are having, and have also purchased some hardware to play around with myself, so we can dig into these problems and see what (if any) solutions there are.

Common Problem #1: It doesn’t work at all

Card not detected, nor any attached devices, nada.

This is a nice easy one to fix. One of the pins on the mystery Thunderbolt header interface on the rear of the card is an “enable” (technically ‘force power’) signal. On the Gigabyte GC-TITAN RIDGE (one of the most popular cards for Thunderbolt hacking) it is regularly reported that you can “short pins 3 and 5”. **Note that you are actually shorting pins 1 and 3**. Pin 1 is clearly marked on the connector as being at the top of the card, and in the drawing of the connector.

This modification works thanks to a weak pull-up on the input at pin 3 meaning there is +3.3V present. It will not necessarily work for every AIC. In some cases a separate +3.3V supply will need to be applied (via a 10K resistor for example) to the “force power” pin, which on the GC-TITAN RIDGE is pin 1.



Jumpering the GC-TITAN RIDGE into action. This is likely to differ for other cards.

This will not work for all AICs. I have noticed that some (Alpine Ridge for example) cards come up with a PCI Vendor/Device ID of 0xFFFF/0xFFFF which would mean that they wouldn't be enumerated by the system. These cards will need some more initialisation that I'll be looking at in future.

Common Problem #2: When peripherals are connected at boot time, they're not detected

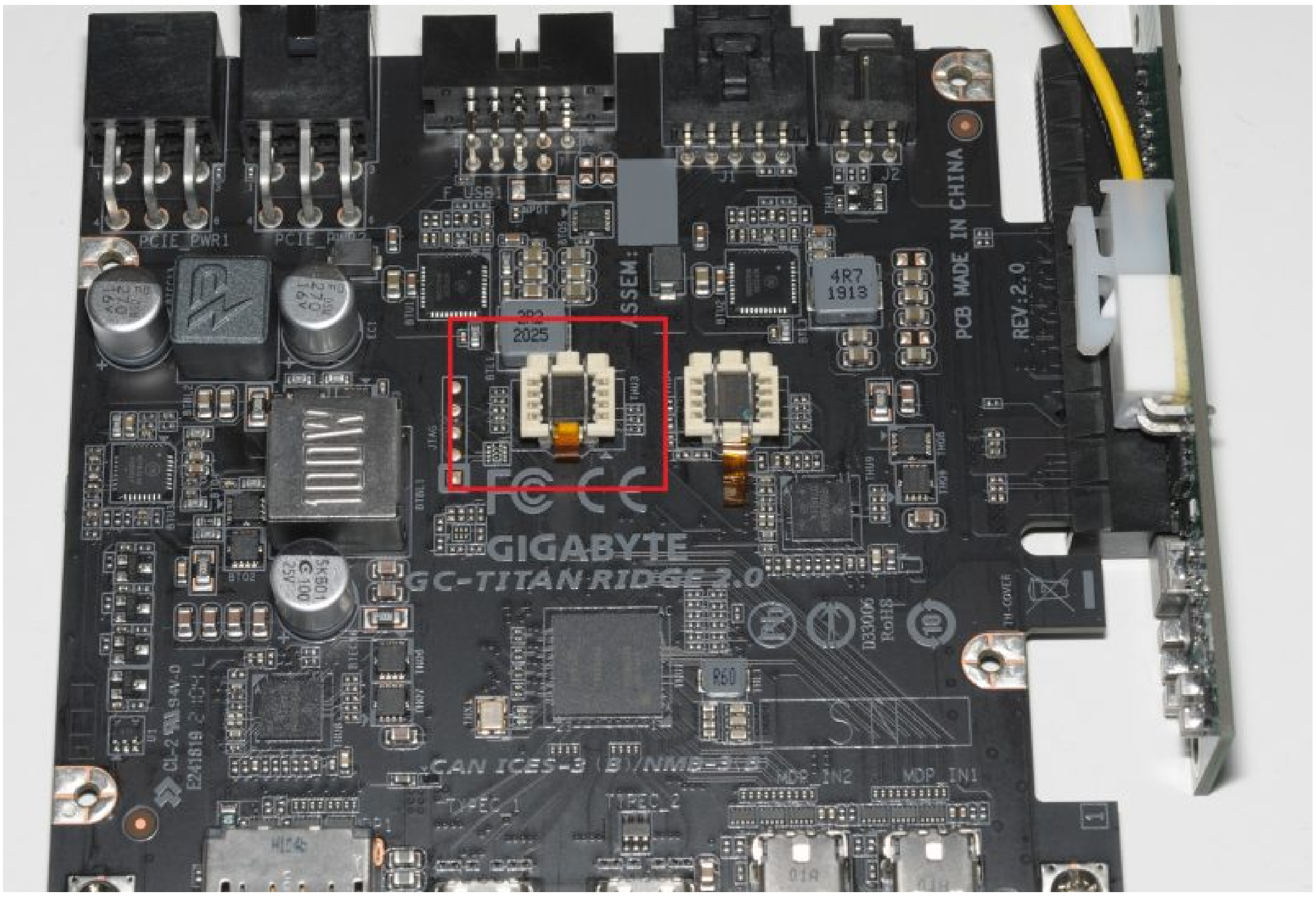
But if I subsequently unplug it, and plug it back in, then it works?

This cause of this problem is that all Thunderbolt Add-in cards by default power up at security level "SL1" which means users have to manually approve attached peripherals. Because you likely don't have any Thunderbolt options in your UEFI setup menu, you can't change this either. You will have previously installed the Intel Thunderbolt software, seen your device in the list, and approved it, and it might have even sprung into life.

When you approve the device, the Intel Thunderbolt software enables the PCIe endpoint for the peripheral in question which brings the peripheral online. The problem is that it only does this on first approval and hot-plug events. We can prove this by opening up the Thunderbolt software with a non functioning device that was attached at boot time, un-approve it, and re-approve it. It now works.

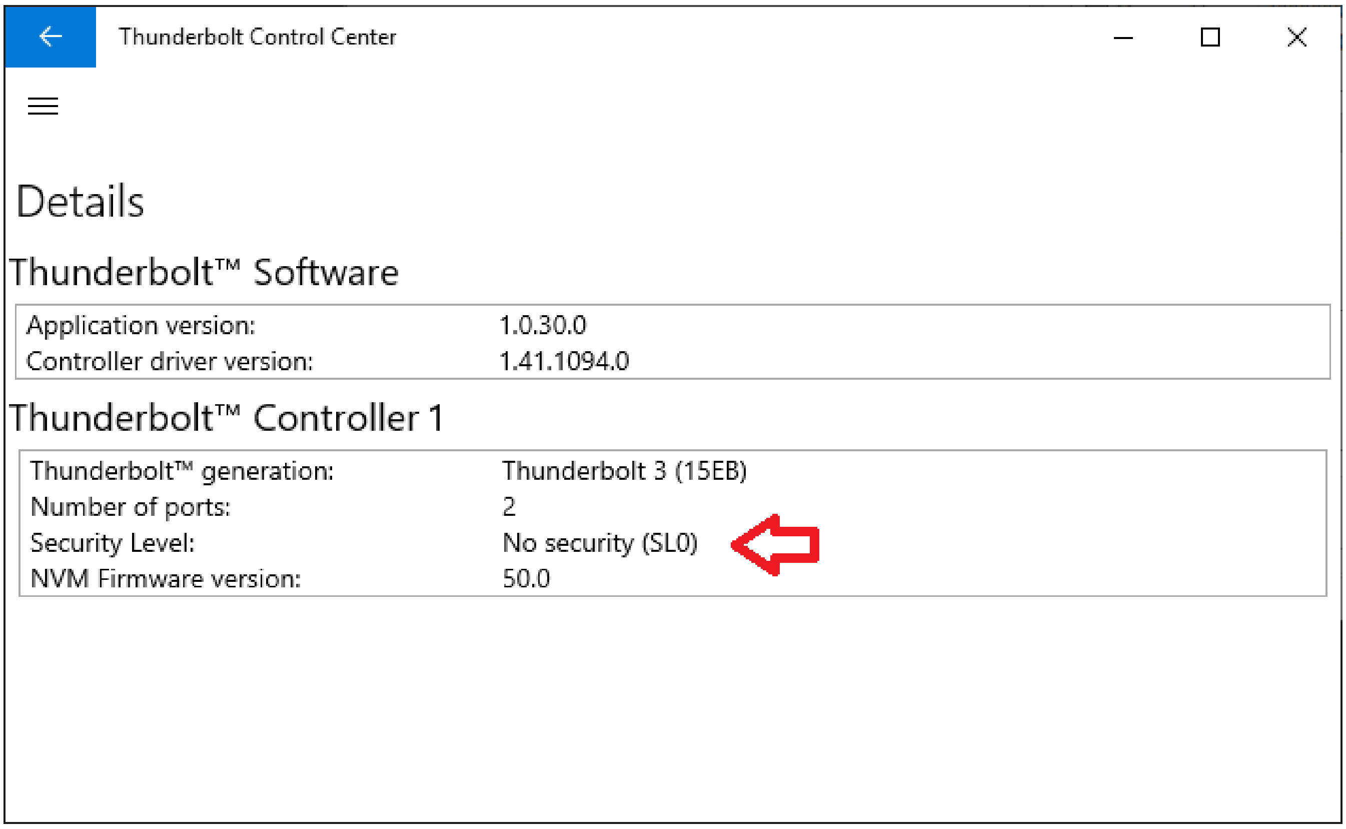
Because your motherboard likely has no Thunderbolt support whatsoever in the UEFI, the boot time activation process is missing. It may be possible to somehow hack the Thunderbolt software to force it to activate approved devices at boot time, but this isn't an ideal solution.

If you aren't concerned about security, there is a good fix out there for some Thunderbolt 3 add-in cards. Björn Ruytenberg is well known for Thunderbolt security research and one of the things he's produced (albeit for a topic completely unrelated to this post) is a Python script which can patch the Thunderbolt controller firmware to bring the card up in "SL0" mode (No security). The purpose of his hack is to create an "evil maid" Thunderbolt controller for performing DMA attacks. Us? we just want the darn thing to work.



By socketing the Thunderbolt firmware flash chip, I was able to patch it to force the card to SL0 security level. It is also possible to do this with an in-system programming arrangement.

I have personally confirmed this as a solution on the Gigabyte GC-TITAN RIDGE. By having SL0 as the default, all periperhals are automatically approved at the time of connection or system power-on. We wouldn't need the Intel Thunderbolt software anymore but it doesn't hurt to have it installed.



Details

Thunderbolt™ Software

Application version:	1.0.30.0
Controller driver version:	1.41.1094.0

Thunderbolt™ Controller 1

Thunderbolt™ generation:	Thunderbolt 3 (15EB)
Number of ports:	2
Security Level:	No security (SL0)
NVM Firmware version:	50.0

It does unfortunately require attaching an external programmer to the SPI chip on the card. I won't be covering this subject as there are many guides out there already where Thunderbolt firmware is re-flashed for other reasons. One small note is that I was unable to patch the firmware already on the card, however the BIN file of the firmware on Gigabyte's website patched and flashed no problem.

Common Problem #3: I can't hot plug

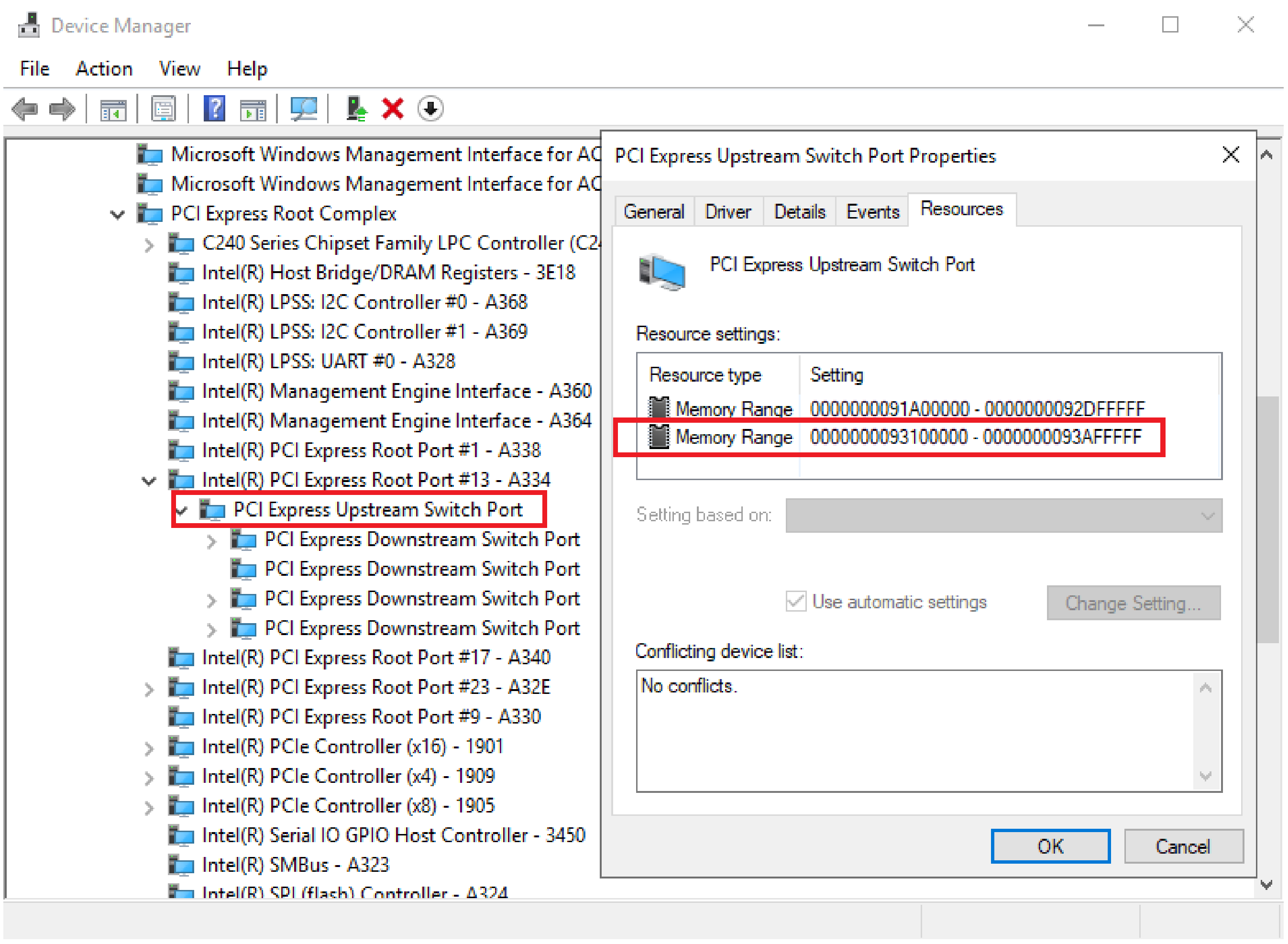
Or perhaps you can hot plug on port 2, but not port 1. This is assuming you are not having problem #2.

This is because the Thunderbolt controller hasn't been setup properly by the UEFI. To be able to hot plug, two things are required:

1) Memory space allocation

Thunderbolt controllers are typically allocated around 1GB of memory mapped IO space to allow the CPU to address any peripherals attached to it. Most PCIe peripherals (excluding GPUs) only use kilobytes, at most a few megabytes of memory mapped IO space.

From my own experimentation the UEFI on unsupported boards did allocate the Thunderbolt controller 10 megabytes of MMIO space **if** attached to the chipset PCIe lanes, which is enough to get basic peripherals working i.e. NICs, NVMe SSDs etc. This appears to because the Intel reference code does this by default and if you are lucky your motherboard manufacturer has left this alone. Anecdotally it appears that AMD do something similar.



Microsoft Windows Management Interface for AC
Microsoft Windows Management Interface for AC
PCI Express Root Complex
C240 Series Chipset Family LPC Controller (C2
Intel(R) Host Bridge/DRAM Registers - 3E18
Intel(R) LPSS: I2C Controller #0 - A368
Intel(R) LPSS: I2C Controller #1 - A369
Intel(R) LPSS: UART #0 - A328
Intel(R) Management Engine Interface - A360
Intel(R) Management Engine Interface - A364
Intel(R) PCI Express Root Port #1 - A338
Intel(R) PCI Express Root Port #13 - A334
PCI Express Upstream Switch Port
PCI Express Downstream Switch Port
PCI Express Downstream Switch Port
PCI Express Downstream Switch Port
PCI Express Downstream Switch Port
Intel(R) PCI Express Root Port #17 - A340
Intel(R) PCI Express Root Port #23 - A32E
Intel(R) PCI Express Root Port #9 - A330
Intel(R) PCIe Controller (x16) - 1901
Intel(R) PCIe Controller (x4) - 1909
Intel(R) PCIe Controller (x8) - 1905
Intel(R) Serial IO GPIO Host Controller - 3450
Intel(R) SMBus - A323
Intel(R) SPI (flash) Controller - A324

PCI Express Upstream Switch Port Properties

PCI Express Upstream Switch Port

Resource settings:

Resource type	Setting
Memory Range	000000091A00000 - 0000000092DFFFFF
Memory Range	0000000093100000 - 0000000093AFFFFF

Conflicting device list:

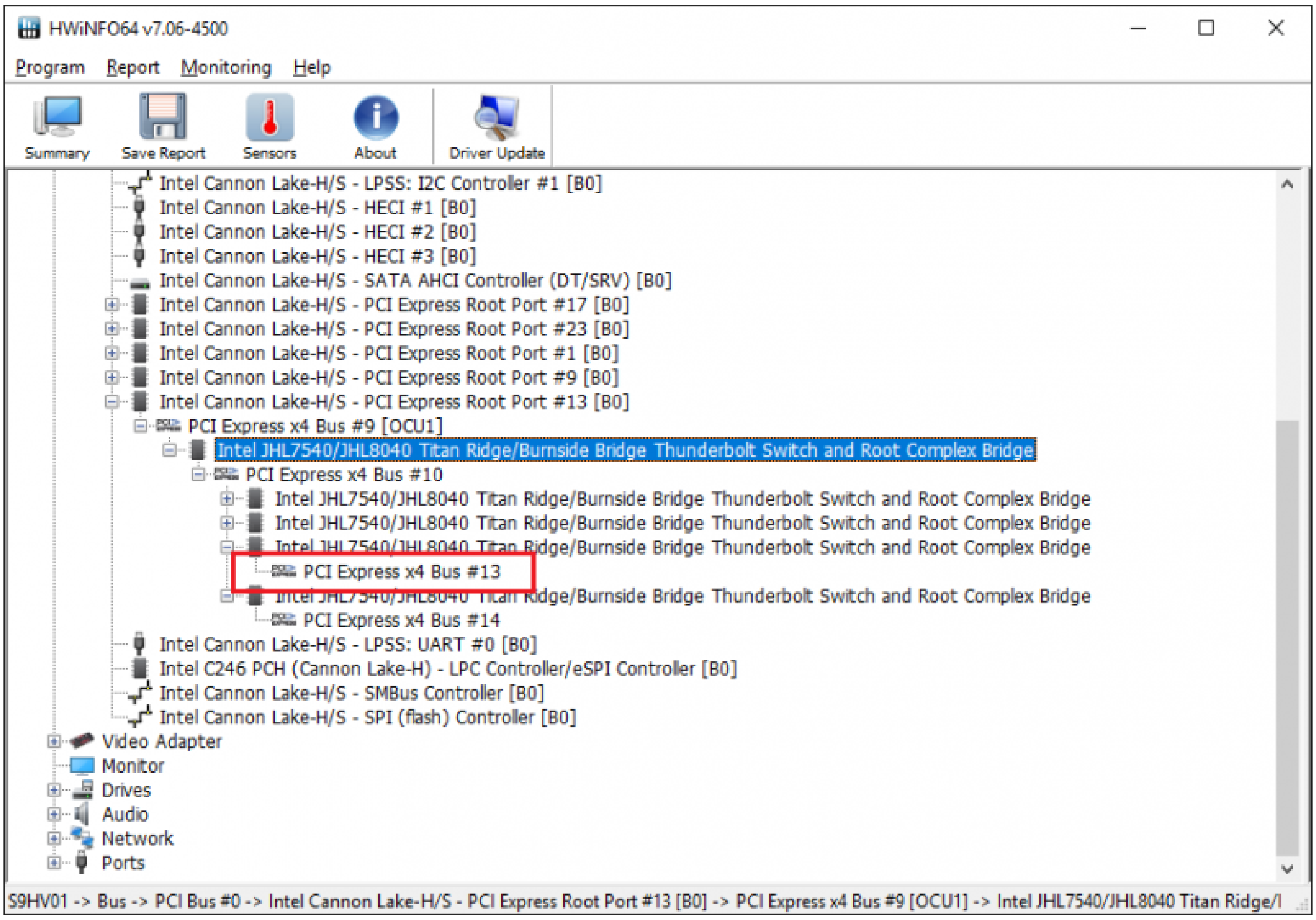
No conflicts.

This non Thunderbolt supporting board allocates 10 megabytes of MMIO to the Thunderbolt controller if connected to the chipset PCIe lanes. Well short of the usual 1GB, but enough for simple use cases.

2) Reserve PCIe bus indexes

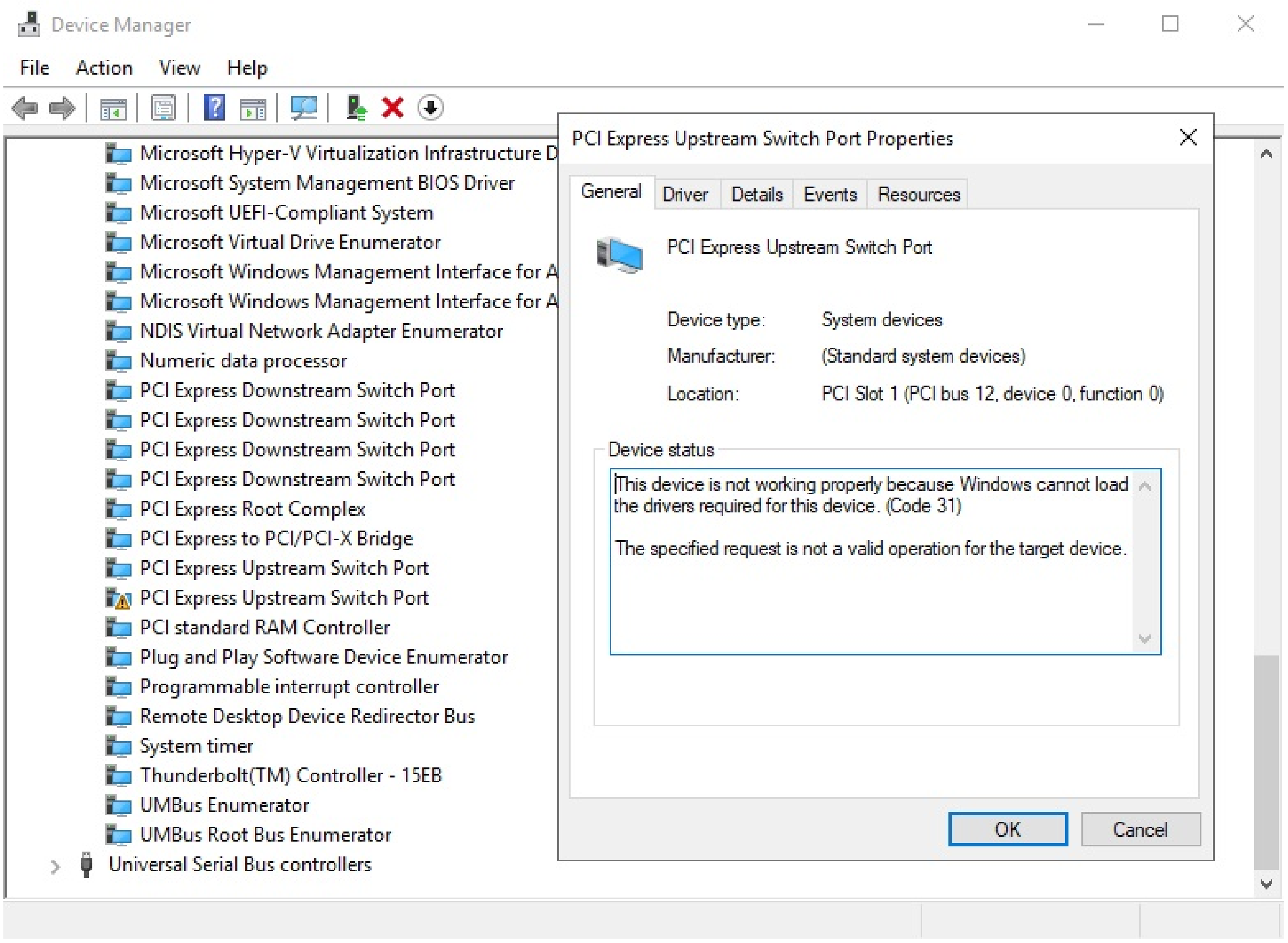
The PCI specification, dating back to the beginning in the 1990s has a frustrating limitation that buses must be numbered in the order that they are enumerated, which is a fixed order based on the order they are physically and logically connected.

When we plug Thunderbolt peripherals in, we add more PCIe buses, which need to have an index number assigned. If that new bus appears in the middle of a tree of PCIe buses, then it cannot be used as there are no free indexes to assign to it, and buses cannot be dynamically re-numbered while the operating system is running.



```
Intel Cannon Lake-H/S - LPSS: I2C Controller #1 [B0]
Intel Cannon Lake-H/S - HECI #1 [B0]
Intel Cannon Lake-H/S - HECI #2 [B0]
Intel Cannon Lake-H/S - HECI #3 [B0]
Intel Cannon Lake-H/S - SATA AHCI Controller (DT/SRV) [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #17 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #23 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #1 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #9 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #13 [B0]
  PCI Express x4 Bus #9 [OCU1]
    Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
      PCI Express x4 Bus #10
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
          PCI Express x4 Bus #13
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
          PCI Express x4 Bus #14
Intel Cannon Lake-H/S - LPSS: UART #0 [B0]
Intel C246 PCH (Cannon Lake-H) - LPC Controller/eSPI Controller [B0]
Intel Cannon Lake-H/S - SMBus Controller [B0]
Intel Cannon Lake-H/S - SPI (flash) Controller [B0]
Video Adapter
Monitor
Drives
Audio
Network
Ports
S9HV01 -> Bus -> PCI Bus #0 -> Intel Cannon Lake-H/S - PCI Express Root Port #13 [B0] -> PCI Express x4 Bus #9 [OCU1] -> Intel JHL7540/JHL8040 Titan Ridge/I
```

In the above example, the UEFI does not reserve any bus indexes. The bus highlighted belongs to Port 1 on the AIC. Nothing can be connected to it at run-time, because it would immediately want to assign bus index 14 to it, already in use by Port 2.


Device Manager
File
Action
View
Help
Microsoft Hyper-V Virtualization Infrastructure D
Microsoft System Management BIOS Driver
Microsoft UEFI-Compliant System
Microsoft Virtual Drive Enumerator
Microsoft Windows Management Interface for A
Microsoft Windows Management Interface for A
NDIS Virtual Network Adapter Enumerator
Numeric data processor
PCI Express Downstream Switch Port
PCI Express Downstream Switch Port
PCI Express Downstream Switch Port
PCI Express Downstream Switch Port
PCI Express Root Complex
PCI Express to PCI/PCI-X Bridge
PCI Express Upstream Switch Port
PCI Express Upstream Switch Port
PCI standard RAM Controller
Plug and Play Software Device Enumerator
Programmable interrupt controller
Remote Desktop Device Redirector Bus
System timer
Thunderbolt(TM) Controller - 15EB
UMBus Enumerator
UMBus Root Bus Enumerator
Universal Serial Bus controllers
PCI Express Upstream Switch Port Properties
General
Driver
Details
Events
Resources
PCI Express Upstream Switch Port
Device status
OK
Cancel

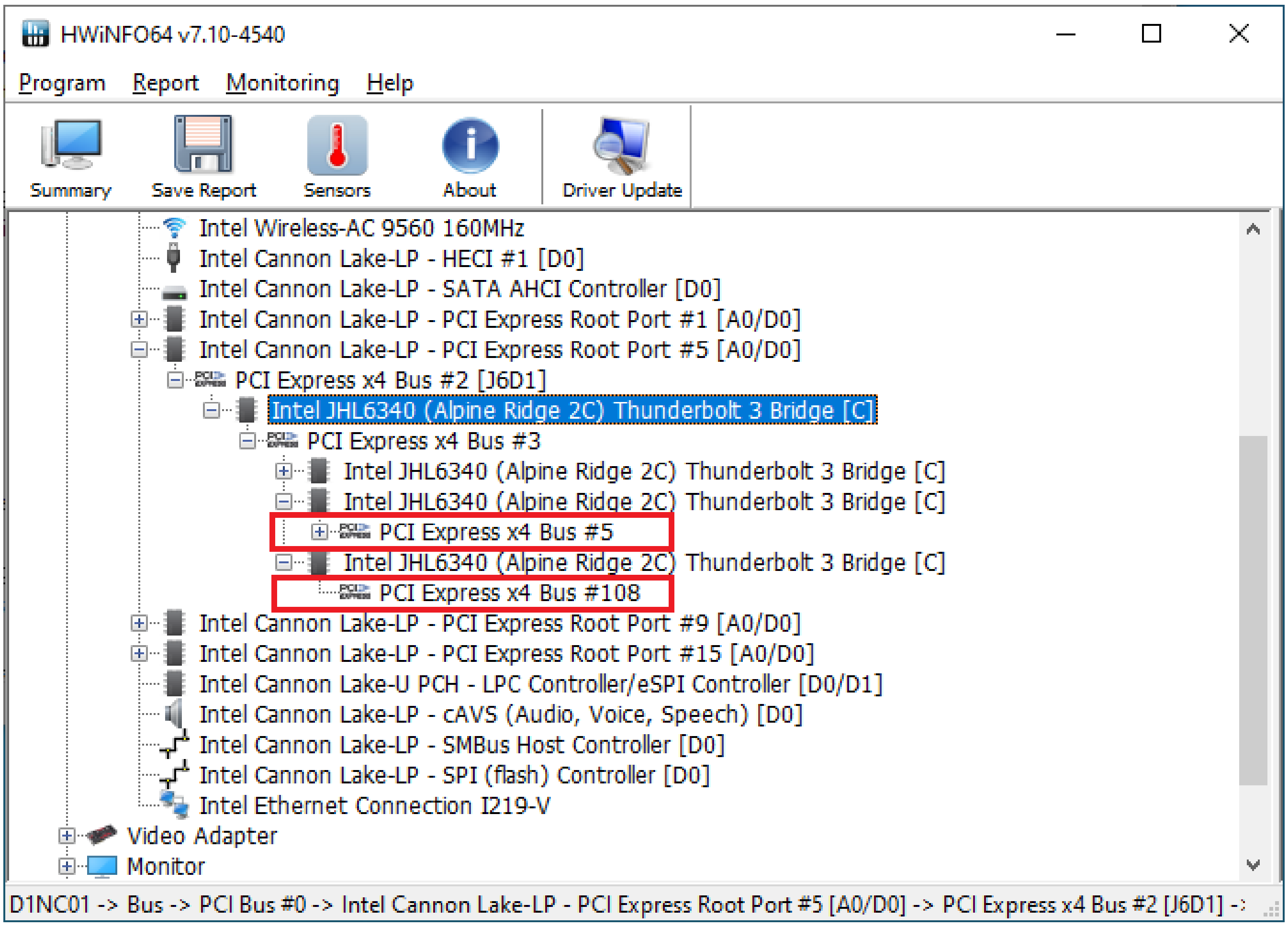
Device type:	System devices
Manufacturer:	(Standard system devices)
Location:	PCI Slot 1 (PCI bus 12, device 0, function 0)

```
This device is not working properly because Windows cannot load
the drivers required for this device. (Code 31)

The specified request is not a valid operation for the target device.
```

If we hot plug something into port 1, all we get is an error code 31 on the associated port (which means there isn't enough resources available to activate it).

A Thunderbolt supporting UEFI will "reserve" typically quite a large number, at least 60, of downstream bus indexes for each Thunderbolt port. It cannot go bonkers reserving huge numbers of bus indexes all over the place as there is a hard limit of 256 per system.



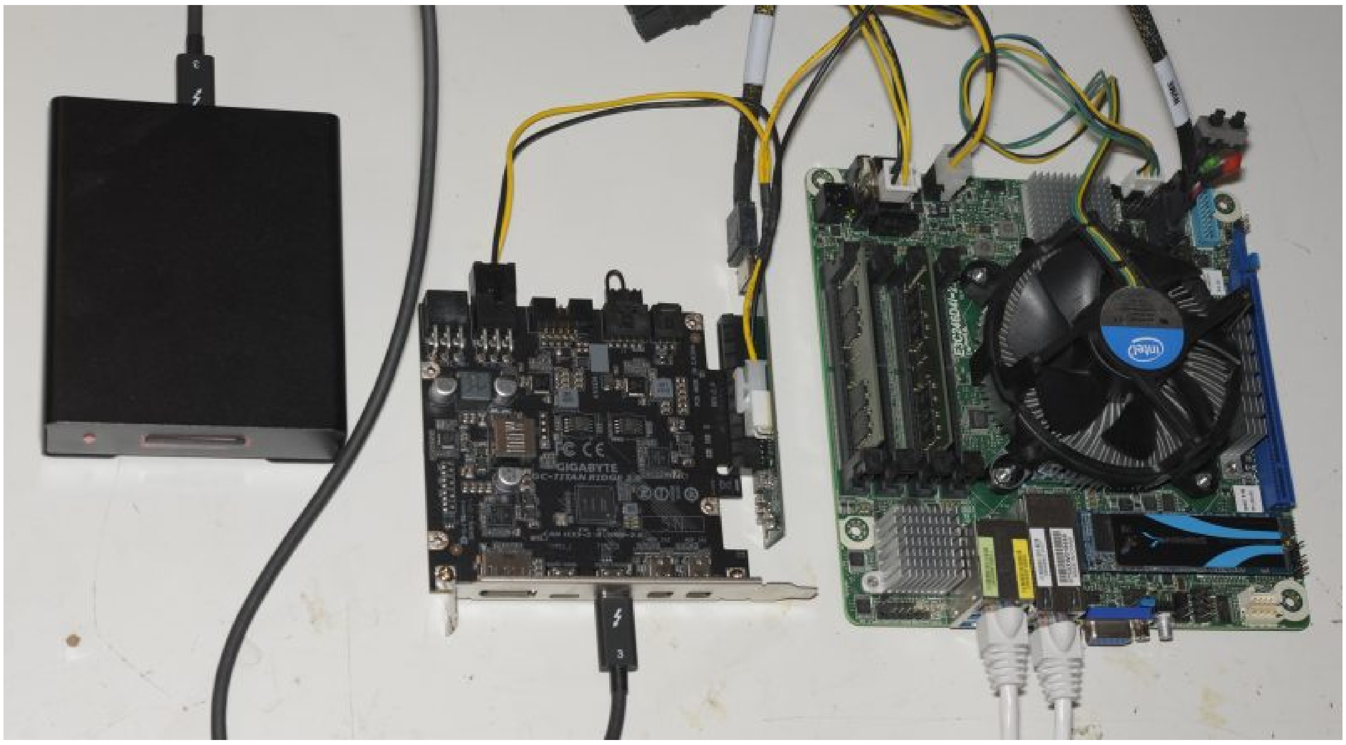
```
Intel Wireless-AC 9560 160MHz
Intel Cannon Lake-LP - HECI #1 [D0]
Intel Cannon Lake-LP - SATA AHCI Controller [D0]
Intel Cannon Lake-LP - PCI Express Root Port #1 [A0/D0]
Intel Cannon Lake-LP - PCI Express Root Port #5 [A0/D0]
  PCI Express x4 Bus #2 [J6D1]
    Intel JHL6340 (Alpine Ridge 2C) Thunderbolt 3 Bridge [C]
      PCI Express x4 Bus #3
        Intel JHL6340 (Alpine Ridge 2C) Thunderbolt 3 Bridge [C]
        Intel JHL6340 (Alpine Ridge 2C) Thunderbolt 3 Bridge [C]
          PCI Express x4 Bus #5
        Intel JHL6340 (Alpine Ridge 2C) Thunderbolt 3 Bridge [C]
          PCI Express x4 Bus #108
Intel Cannon Lake-LP - PCI Express Root Port #9 [A0/D0]
Intel Cannon Lake-LP - PCI Express Root Port #15 [A0/D0]
Intel Cannon Lake-U PCH - LPC Controller/eSPI Controller [D0/D1]
Intel Cannon Lake-LP - cAVS (Audio, Voice, Speech) [D0]
Intel Cannon Lake-LP - SMBus Host Controller [D0]
Intel Cannon Lake-LP - SPI (flash) Controller [D0]
Intel Ethernet Connection I219-V
Video Adapter
Monitor
D1NC01 -> Bus -> PCI Bus #0 -> Intel Cannon Lake-LP - PCI Express Root Port #5 [A0/D0] -> PCI Express x4 Bus #2 [J6D1] ->
```

In the above example, a correctly configured Thunderbolt controller has a whopping 103 (108 minus 5) bus indexes reserved for the first port.

Just because your UEFI doesn't reserve any bus numbers, that doesn't mean you can't hot plug

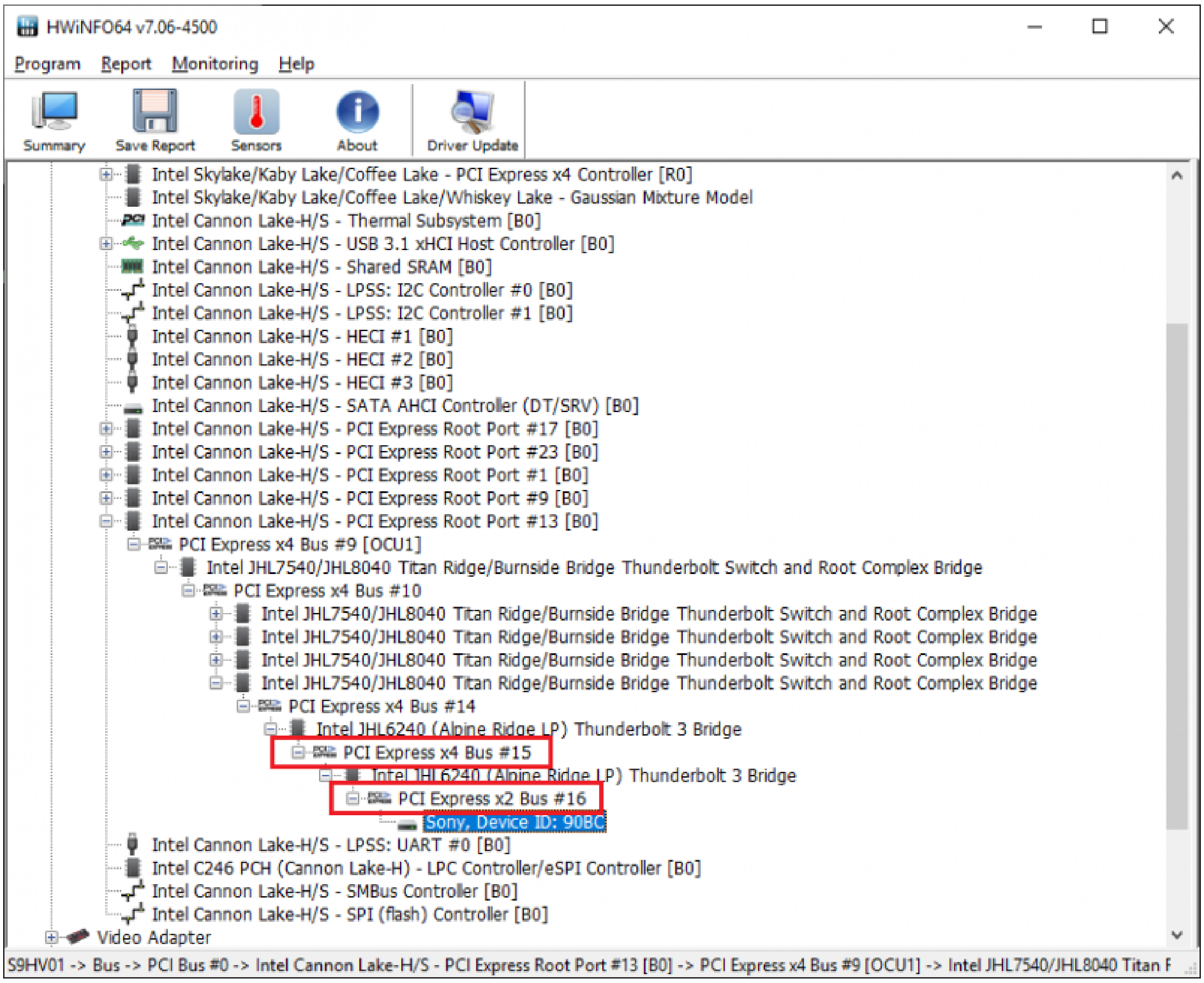
If you attach your Thunderbolt AIC to the *highest numbered* PCIe bus in your system, assuming there is sufficient MMIO mapping, and that you can physically get at it, you will be able to hot plug on the second port of the Thunderbolt AIC. This works because none of the bus index numbers after the second port are in use, therefore the operating system can assign them to the connected peripheral.

I was able to prove this in the below scenario:



On the above board (AsRock E3C246D4I-2T), the OCuLink port the Thunderbolt AIC is attached to is the highest numbered bus in the system, making port 2 on the AIC the absolute last bus, therefore another bus can be attached to it at run-time.

I wasn’t able to find any product which adapts from OCuLink to PCIe, fortunately I had previously designed my own. For the test I’m using a PCIe protocol XQD/CFExpress reader.

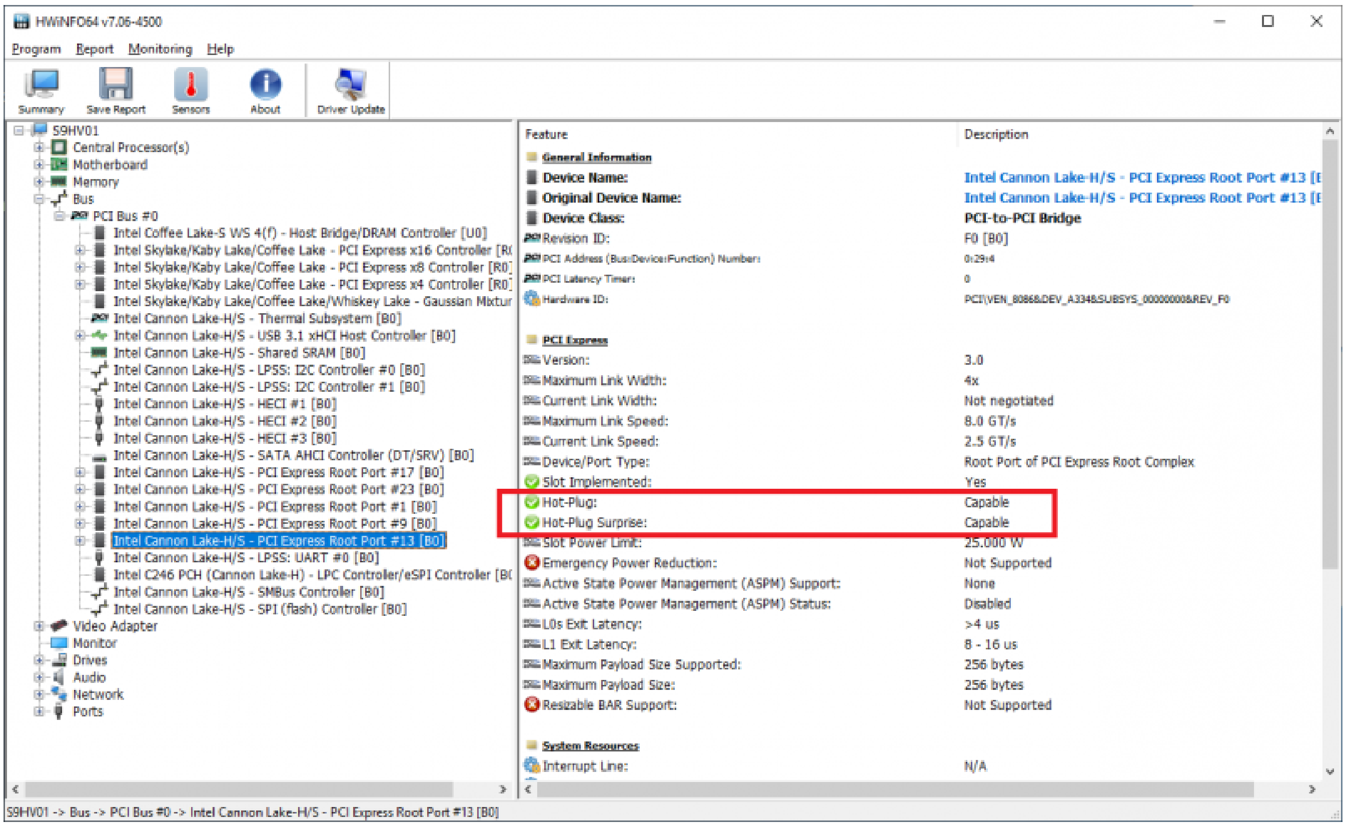


```
Intel Skylake/Kaby Lake/Coffee Lake - PCI Express x4 Controller [R0]
Intel Skylake/Kaby Lake/Coffee Lake/Whiskey Lake - Gaussian Mixture Model
Intel Cannon Lake-H/S - Thermal Subsystem [B0]
Intel Cannon Lake-H/S - USB 3.1 xHCI Host Controller [B0]
Intel Cannon Lake-H/S - Shared SRAM [B0]
Intel Cannon Lake-H/S - LPSS: I2C Controller #0 [B0]
Intel Cannon Lake-H/S - LPSS: I2C Controller #1 [B0]
Intel Cannon Lake-H/S - HECI #1 [B0]
Intel Cannon Lake-H/S - HECI #2 [B0]
Intel Cannon Lake-H/S - HECI #3 [B0]
Intel Cannon Lake-H/S - SATA AHCI Controller (DT/SRV) [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #17 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #23 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #1 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #9 [B0]
Intel Cannon Lake-H/S - PCI Express Root Port #13 [B0]
  PCI Express x4 Bus #9 [OCU1]
    Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
      PCI Express x4 Bus #10
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
        Intel JHL7540/JHL8040 Titan Ridge/Burnside Bridge Thunderbolt Switch and Root Complex Bridge
          PCI Express x4 Bus #14
            Intel JHL6240 (Alpine Ridge LP) Thunderbolt 3 Bridge
              PCI Express x4 Bus #15
                Intel JHL6240 (Alpine Ridge LP) Thunderbolt 3 Bridge
                  PCI Express x2 Bus #16
                    Sony, Device ID: 90BC
Intel Cannon Lake-H/S - LPSS: UART #0 [B0]
Intel C246 PCH (Cannon Lake-H) - LPC Controller/eSPI Controller [B0]
Intel Cannon Lake-H/S - SMBus Controller [B0]
Intel Cannon Lake-H/S - SPI (flash) Controller [B0]
Video Adapter
S9HV01 -> Bus -> PCI Bus #0 -> Intel Cannon Lake-H/S - PCI Express Root Port #13 [B0] -> PCI Express x4 Bus #9 [OCU1] -> Intel JHL7540/JHL8040 Titan F
```

Because the OCuLink port has the highest numbered bus in the system, and port 2 is the highest numbered bus, every other bus index after it is free to use. In this case Windows is able to dynamically allocate bus indexes 15 and 16

Another interesting thing to note is that if a device is connected at boot time (assuming the card is in SL0 mode), PCIe indexes will be reserved. The card could then be unplugged, then re-plugged later on just fine.

(But maybe not)



```
S9HV01
  Central Processor(s)
  Motherboard
  Memory
  Bus
    PCI Bus #0
      Intel Coffee Lake-S WS 4(f) - Host Bridge/DRAM Controller [U0]
      Intel Skylake/Kaby Lake/Coffee Lake - PCI Express x16 Controller [R(
      Intel Skylake/Kaby Lake/Coffee Lake - PCI Express x8 Controller [R0]
      Intel Skylake/Kaby Lake/Coffee Lake - PCI Express x4 Controller [R0]
      Intel Skylake/Kaby Lake/Coffee Lake/Whiskey Lake - Gaussian Mixtur
      Intel Cannon Lake-H/S - Thermal Subsystem [B0]
      Intel Cannon Lake-H/S - USB 3.1 xHCI Host Controller [B0]
      Intel Cannon Lake-H/S - Shared SRAM [B0]
      Intel Cannon Lake-H/S - LPSS: I2C Controller #0 [B0]
      Intel Cannon Lake-H/S - LPSS: I2C Controller #1 [B0]
      Intel Cannon Lake-H/S - HECI #1 [B0]
      Intel Cannon Lake-H/S - HECI #2 [B0]
      Intel Cannon Lake-H/S - HECI #3 [B0]
      Intel Cannon Lake-H/S - SATA AHCI Controller (DT/SRV) [B0]
      Intel Cannon Lake-H/S - PCI Express Root Port #17 [B0]
      Intel Cannon Lake-H/S - PCI Express Root Port #23 [B0]
      Intel Cannon Lake-H/S - PCI Express Root Port #1 [B0]
      Intel Cannon Lake-H/S - PCI Express Root Port #9 [B0]
      Intel Cannon Lake-H/S - PCI Express Root Port #13 [B0]
      Intel Cannon Lake-H/S - LPSS: UART #0 [B0]
      Intel C246 PCH (Cannon Lake-H) - LPC Controller/eSPI Controller [B(
      Intel Cannon Lake-H/S - SMBus Controller [B0]
      Intel Cannon Lake-H/S - SPI (flash) Controller [B0]
  Video Adapter
  Monitor
  Drives
  Audio
  Network
  Ports
```

Feature	Description
General Information	
Device Name:	Intel Cannon Lake-H/S - PCI Express Root Port #13 [E
Original Device Name:	Intel Cannon Lake-H/S - PCI Express Root Port #13 [E
Device Class:	**PCI-to-PCI Bridge**
Revision ID:	F0 [B0]
PCI Address (Bus:Device:Function) Number:	0:29:4
PCI Latency Timer:	0
Hardware ID:	PCI\VEN_8086&DEV_A334&SUBSYS_00000000&REV_F0
PCI Express	
Version:	3.0
Maximum Link Width:	4x
Current Link Width:	Not negotiated
Maximum Link Speed:	8.0 GT/s
Current Link Speed:	2.5 GT/s
Device/Port Type:	Root Port of PCI Express Root Complex
Slot Implemented:	Yes
Hot-Plug:	Capable
Hot-Plug Surprise:	Capable
Slot Power Limit:	25.000 W
Emergency Power Reduction:	Not Supported
Active State Power Management (ASPM) Support:	None
Active State Power Management (ASPM) Status:	Disabled
L0s Exit Latency:	>4 us
L1 Exit Latency:	8 - 16 us
Maximum Payload Size Supported:	256 bytes
Maximum Payload Size:	256 bytes
Resizable BAR Support:	Not Supported
System Resources	
Interrupt Line:	N/A

```
S9HV01 -> Bus -> PCI Bus #0 -> Intel Cannon Lake-H/S - PCI Express Root Port #13 [B0]
```

HWINFO showing that hot-plug is enabled on the port I'm plugging my Thunderbolt AIC into. Lucky me.

Another potential blocker is whether or not the motherboard manufacturer has enabled hot-plugging for the PCIe root port that you're trying to attach the Thunderbolt AIC to. In my case they have, however this is not a given.

If hot-plug isn't enabled for the root port in question, you still *sort of* can still hot plug. After plugging in a new Thunderbolt device, you'd have to open device manager and manually press the "Scan for hardware changes" button. Just as you would have to when plugging a disk into a non hot-plugging SATA port. This does not get around the bus numbering problem I covered earlier.

Operating system hacks to the rescue?

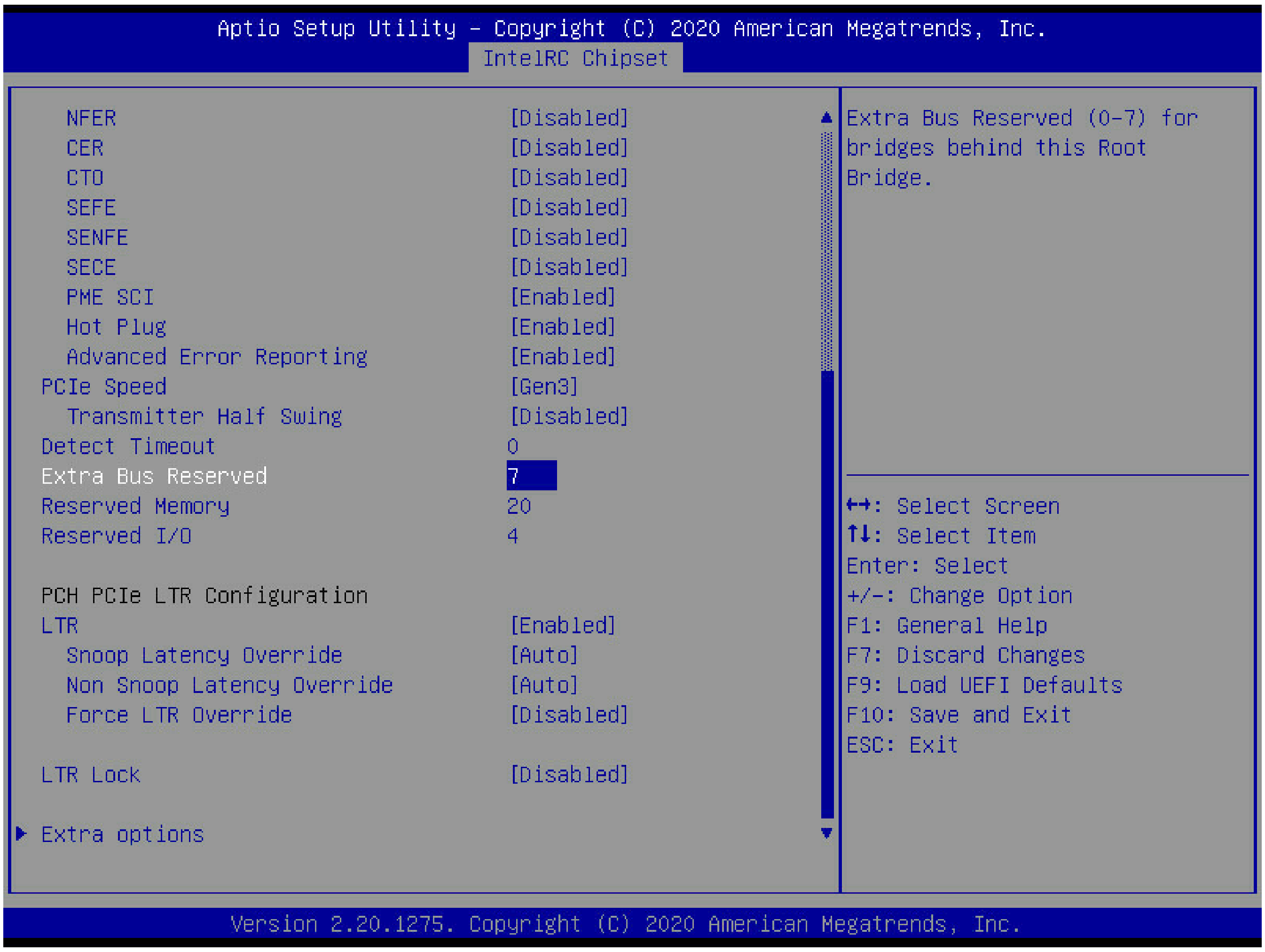
Unfortunately, this problem is where we hit the buffers. The Linux kernel is able to re-number PCIe buses with some advanced hacking – a solution is detailed here.

For Windows, we're in serious difficulty. There are a couple of dead-end threads on OSR where people have tried to tackle this problem, essentially trying to create something like what Linux is able to do, renumbering buses at boot time. Windows does not do this, and by the looks of it, never will.

One commentator suggests it is possible to create a filter driver which re-numbers PCIe buses just before PCI.sys loads. Even if we could, we'd either have to run with driver signature enforcement disabled, or pay thousands of dollars to have the driver signed for 64-bit use. Also, as the commentator from Microsoft points out, it wouldn't work in every scenario.

Correct configuration from the UEFI

Ultimately this is the best solution. Reserving resources for hot pluggable PCI systems is a problem which predates Thunderbolt, and UEFI solutions have been available (to OEMs at least) for a long time.



```
Aptio Setup Utility - Copyright (C) 2020 American Megatrends, Inc.
                       IntelRC Chipset

    NFER                               [Disabled]       Extra Bus Reserved (0-7) for
    CER                                [Disabled]       bridges behind this Root
    CTO                                [Disabled]       Bridge.
    SEFE                               [Disabled]
    SENFE                              [Disabled]
    SECE                               [Disabled]
    PME SCI                            [Enabled]
    Hot Plug                           [Enabled]
    Advanced Error Reporting           [Enabled]
  PCIe Speed                           [Gen3]
    Transmitter Half Swing             [Disabled]
  Detect Timeout                       0
  Extra Bus Reserved                   7
  Reserved Memory                      20               ↔: Select Screen
  Reserved I/O                         4                ↑↓: Select Item
                                                        Enter: Select
  PCH PCIe LTR Configuration                            +/-: Change Option
  LTR                                  [Enabled]        F1: General Help
    Snoop Latency Override             [Auto]           F7: Discard Changes
    Non Snoop Latency Override         [Auto]           F9: Load UEFI Defaults
    Force LTR Override                 [Disabled]       F10: Save and Exit
                                                        ESC: Exit
  LTR Lock                             [Disabled]

▶ Extra options

     Version 2.20.1275. Copyright (C) 2020 American Megatrends, Inc.
```

The Intel UEFI Reference Code has an option called "Extra Bus Reserved", this allows for up to 7 bus indexes to be reserved for each PCIe downstream port on a bridge, well short of the minimum 60 which Thunderbolt initialisation normally allocates. It would't be enough for daisy chaining or multi-slot Thunderbolt enclosures, *but it would be enough for anything else*.

Also present here is the option to enable hot-plug and to reserve MMIO space. It's limited to 20MB *but* as I previously mentioned, that is enough for the majority of use-cases.

Unfortunately it is ridiculously unlikely that you will find these options readily available to you, supported, and actually working. Motherboard manufacturers have no reason to expose this kind of functionality to end-users.

What are you talking about? It works for me!

Some people apparently find that it all just works – this may be because the motherboard manufacturer has left some actual Thunderbolt initialisation code in the UEFI, or, they've just decided to be generous and put in some half decent PCIe hot-plugging initialisation code. Are you one of these people? Please tell me about it!

How else could we get around this problem?

For Linux, there is a solution, but for Windows, something will have to be hacked. At this time I don't have sufficient understanding to know whether hacking Windows, or the UEFI is the best starting point. It definitely does look to be interesting fodder for a future project. Please get in touch with me if you have researched this subject.

What about the Thunderbolt header?

I haven't spent much time talking about this because in the context of getting basic functionality working, it's not a significant part of the problem. The header provides a collection of GPIO signals which form a back channel between the Thunderbolt controller and an SMM Module in the UEFI, which is invisible to the operating system.

The signals on the header mostly appear to be related to power management however there is one notable exception where it is possible for the system to vector into an SMI on hot-plug events. *I am not clear why this is necessary and quite frankly it looks a bit nasty*. I will be studying all of this in detail in future.

Some of the source code for this mechanism is disclosed in TianoCore. But not all of it. Inevitably vendors can add their own hacky little bits, potentially explaining why the stability of Thunderbolt varies from vendor to vendor.

Is there any light at the end of the tunnel?

From reading through forums, it is clear that there are many people hoping that one day, some kind of Thunderbolt AIC will appear which can be popped into an AMD or that cranky old 3rd gen Intel system and work perfectly. *I have been one of those people for years now.*

Unfortunately as we can now see, there are already limitations baked into these platforms which make such an addition impossible without hacking. A PCIe enabled USB4 AIC (when, and if one ever surfaces) will not change the situation because it is still a hot pluggable PCIe system with all of the security worries of Thunderbolt and therefore constrained by every problem I've outlined on this page.

I'd bet the only difference we'll see is that it has an ASMedia chip on it, instead of Intel. Otherwise it'll be the same deal: Made by a motherboard manufacturer, requires a proprietary header, and has a short list of boards it's supported in. At least that will include AMD boards.

There is nothing preventing Intel (and other stakeholders) from offering a vendor independent solution to these problems, however it would take years of discussion and engineering efforts from all of the relevant parties (of which there are many). Assuming this is already happening, a one-size-fits-all solution is likely still years, and a new motherboard away.

For now, it's easier for OEMs to tell us to plug the Thunderbolt header cable in to the motherboard. Haven't got the header on your board? Goodbye.

Related Posts

- Hacking Thunderbolt Part 2: Retrofitting PCI Hot-Plug To The UEFI

Posted in PC & Software, Thunderbolt

⟵ Blasted fake CA3080's AGAIN

27 thoughts on "Hacking Thunderbolt Part 1: Why that add-in card doesn't work properly in your

unsupported PC"

Pingback: Why that Thunderbolt add-in card doesn't work properly in your unsupported PC – OSnews Edit

chx says:

September 11, 2021 at 8:00 am Edit

"I wasn't able to find any product which adapts from OCuLink to PCIe" Find a SFF-8611 to SFF-8643 PCI Express cable say Delock 85694, a riser from that is readily available from say MicroSataCables , part number is SFF-1332-U2X4.

Reply

admin says:

September 11, 2021 at 8:24 am Edit

Thanks for the link but that is not an OCuLink to PCIe conversion, nor is that going to fit into a 1u chassis, and allow me to put an AIC wherever I please. That was the reason I built my own.

Reply

bs says:

September 12, 2021 at 9:47 am Edit

Could #3 be solved by a pre-bootloader that re-enumerates PCIe? Would cause issues with Secure Boot though…

Reply

admin says:

September 12, 2021 at 10:00 am Edit

Yes that would be able to fix it. I don't know to build such a thing at present however.

So long as buses needed for the boot process aren't touched.

Reply

admin says:

September 13, 2021 at 6:47 am Edit

Björn Ruytenberg has an example of an EFI pre-bootloader: https://github.com/BjornRuytenberg/kdmap-patcher

It's Thunderbolt related but for enabling Kernel DMA protection. Interesting. I may see if I can look into how this could be modified to reserve some PCIe indexes.

Reply

MHB says:

September 12, 2021 at 7:35 pm Edit

I've got an AMD 5900X waiting to go into my new desktop build. All that's left to get is the motherboard. Also expecting a new TB4 laptop from work that comes with a TB dock (all Dell).

So I'm hoping to find an AMD board with either built-in TB (Gigabyte Vision D-P TB3), or support a TB AIC, so far the new revisions of the Gigabyte boards look the most promising for the latter. The Asus ProArt B550 and X570 seem to be one step ahead, and support TB4 built-in.

Reply

Chris says:

September 21, 2021 at 5:15 am Edit

Thanks for writing this overview, it's very helpful. Would you be so kind though and link a guide to the best way to flash the SPI chip? I'm seeing conflicting methods and most seem to focus on Macs rather than Windows systems. If you have any guide somewhere that you can link to for how to flash the SPI chip exactly, that would be wonderful.

Reply

admin says:

September 21, 2021 at 6:59 am Edit

https://youtu.be/R6jrc3TxQDc

Worked for me. As I said though I did end up socketing mine as it's just easier and safer.

Reply

Chris says:

September 25, 2021 at 12:42 am Edit

What is socketing exactly, for those who are new to it, and how to do it like you did it? Just a tutorial would be helpful as I'm planning to do the same thing you did in your article.

Reply

admin says:

September 25, 2021 at 1:37 am Edit

It means de-soldering the chip and putting a socket in its place. Still want a tutorial? 😉

Reply

Chris says:

October 5, 2021 at 10:46 pm Edit

Yes please!

admin says:

October 6, 2021 at 7:48 am Edit

I'm a little busy with some Thunderbolt related UEFI hacking at present – don't have time to put up a detailed tutorial.

You need to de-solder the chip I've highlighted, and solder one of these in its place: https://www.adafruit.com/product/4726

I personally used a hot air gun with a fine tip (Leister Hot-Jet S). You can also do it with a soldering iron. https://www.youtube.com/watch?v=1c5_UP-qwxE – I would add to that video that the pads should be cleaned up with braid before soldering the socket. Make sure it's oriented properly.

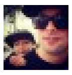
Chris says:

October 12, 2021 at 11:53 pm Edit

Thanks for this. I would love to see a more fool-proof tutorial to continue this topic however. I think it would be very helpful!

After de-soldering the old chip and soldering on the new one, I understand you still need to flash it – what's the procedure like for that?

Will keep checking back on your website and hope you'd be so kind to extend this tutorial with the steps on how to solder + flash the firmware to get this working. Thanks so much.

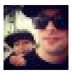


Aaron says:

March 21, 2022 at 2:30 am Edit

You mentioned you took the gigabyte firmware from their site and patched it prior to flashing to enable sl0? I'm looking for what you had to patch. My card also requires plugging in to detect after boot.

Reply



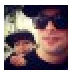
Aaron says:

March 22, 2022 at 6:19 pm Edit

I tried to use the gigabyte firmware updater tool to flash nvm50 while patching it to sl0 via running tcfp on the temp path. Got stuck at 99% 😊.

Ill try the spi flasher next to hopefully recover it (it's bricked 😅)

Reply



Aaron says:

April 5, 2022 at 2:06 am Edit

The Intel flasher tool
Actually worked. After a few reboots it came back to life. Pulled it out and read the fw off via spi and confirmed it was patched to sl0. Thx!

Chris says:

October 1, 2021 at 11:10 pm Edit

Absolutely, sounds like a challenge 😳

Reply

Karl Thunder says:

November 18, 2021 at 7:29 pm Edit

My ASUS z490 MBoard has a 14 Pin Tunderbolt header.

Would I be able to Plug a Gigabyte GC Titan Ridge into that 14 PIN Header with the 5 Pin Cable that comes with the Card? Or would I have to solder my own connection cable?

Reply

admin says:

November 18, 2021 at 7:51 pm Edit

Unlikely. I did a bit of experimentation of mixing cards across vendors. Header isn't really a consideration, unless the pinout is completely wrong in which case the system may not power on.

It seems that vendors have code in their UEFIs which look for their specific AICs, if not found, no Thunderbolt initialisation is done and the card doesn't work.

Reply

Muhammad Bangash says:

December 29, 2021 at 6:57 am Edit

Where to by this jumper wire to short? Also I want to do it for the Asus thunderbolt ex4 card, how will I know which one to jump it on? The thunderbolt pins layout is different than the GC titan ridge

Reply

Dustin says:

February 9, 2022 at 11:39 pm Edit

I have a z370 prime with a 5 pin thunderbolt header was hoping to use a ex3 or ex4 on it but seems that they are both 10 or 14 pin any way you jumper the newer cards.

Reply

Dreamcat4 says:

July 22, 2022 at 4:59 pm Edit

hello, just arrived today my gigabyte titan ridge (to be used for linux). it works after step 1 (jumping the header pins 1-3). and is detected correctly in linux, and works as usb 3.1 controller. However in terms of my motherboard, it is a z170 asrock oc formula, running a latest custom bios (to enable 8700k cpu on older generation). This bios does have a thunderbolt menu, with enable / disable "thunderbolt". And it also has a setting for security, which can be configured to use enable ids, or 'legacy mode'… and it has both a 5-pin thunderbolt header (which the manual claims is for TB2). Plus next to that a 10-pin thunderbolt header (which is supposed to be for TB3 devices). So that is cool possibility to try to hook up a cable in future. Should that in fact be needed. (but aparrently perhaps not so important). Anyhow ATM my problem is that if the thunderbolt feature is enabled in the BIOS menu, then the linux kernel fails to boot. It says "error out of memory, press any key to continue". So it seems like there is some conflict with the memory reservations region for the thunderbolt. Such that the loading of the operating system fails / gives up. Or some other memory mapping conflict occurs. Another question is what has dsanke (the bios modder) has done in his custom bios changes, perhaps if he changed something in the BIOS image that then caused this to happen. However I actually cannot go back to the original BIOS to check. Because then my newer 8700k is not supported on z170 platform. And the thing will never boot (not without that missing intel microcode for the 8700k, which was the whole point of the custom BIOS modding…). So am hoping it is not unavoidable. Or maybe if the thunderbolt bios setting only matters for hotplugging feature. Then that might be ok too.

Anyhow many thanks for this article, it is going to be very helpful 🙂

Reply

G says:

September 21, 2022 at 7:36 pm Edit

Sorry for the necro-post! I have a gigabyte wrx80 motherboard with a lot of esoteric (to me) thunderbolt settings. One in particular is the the ability to assign "thunderbolt resources" before or after PCIe enumeration. How does this affect TB hot plug, detection, setup etc? Thanks!

Reply

Andre says:

November 24, 2022 at 3:56 pm Edit

This is exactly why people love Macs. It just works.

Reply

admin says:

November 24, 2022 at 4:01 pm Edit

Thanks. You have added precisely zero value to this post.

Reply

Gabriel says:

July 11, 2023 at 8:46 am Edit

Hey there, I was wondering how exactly would I short the pins to work on an Asus Z490e ROG STRIX mobo?

Reply

Leave a Reply

Logged in as admin. Log out?

Comment

Post Comment