CTVPDs and Making Your Own USB-C® Thingamajig

Mark Hayter & David Schneider

Hardware Engineering, Chrome OS, Google Inc

USB Developer Days 2019 – Taipei, Taiwan

November 19, 2019

USB
Enabling Connections™

Chromebook/USB-C®: State of the union is good

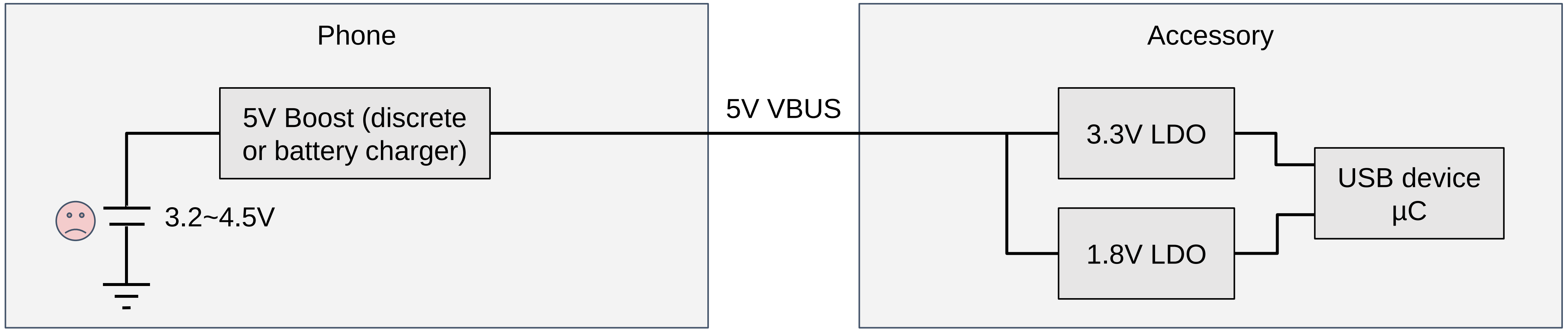
- All new Chromebooks (even low cost ones) have USB-C
  - Power sink: wide input range (5-20V on most, 5V-PDP limit on others)
  - Power source: min 7.5W, some 15W
  - USB Data
  - DP Alt mode video
- Most devices have 2xUSB-C: normally one on each side
  - Good to also have USB Standard-A if it fits in the device
- USB PD 2.0 now has wide adoption and optimized cost structures
- Migrating to USB PD 3.0: CTVPD, FRS, PPS, Power adapter/battery info
- Not much changed in the past two years...this is a Good Thing

On the last episode of USB Developer Days (Vancouver 2017)...

SUPERSPEED+ CERTIFIED USB 10Gbps

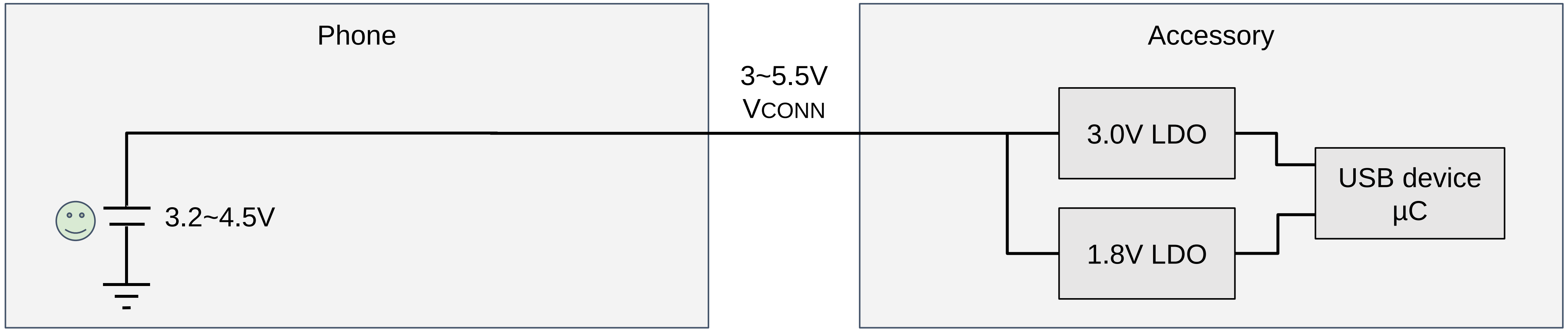
VCONN-Powered USB Devices

Does this look familiar?



- This is how many USB accessories (especially compact audio devices) work with phones today
- Generously assuming boost is ~90% efficient, USB device's core power is coming in at 33% efficiency. Yikes!
- A USB2 device at 0.5A could consume 2.7W at battery

Ideal



- Now we’re at 40~56% efficiency, without really changing the accessory
- Limiting maximum current in this mode to 100mA brings the worst-case battery load to 450mW (320mW at low battery)
- Your headphones can no longer ruin your phone’s battery life!

VPDs enable the ideal, but keep compatibility

- VPD must be able to run from both VBUS (4.75-5.5V) and VCONN (3-5.5V), so it works with any kind of host
- VPD adds an eMarker (simplest PD silicon possible) to advertise its capability. Host can probe SOP' to find out it can remove VBUS
- VCONN is only available on cable plugs, so accessory must be direct-attach or have a captive cable

- Optimizing power for VPDs in a phone (or any other USB PD-capable source or DRP) does not require additional hardware!
- VPD DiscoverIdentity VDO is only defined in PD3, but PD2 devices can safely use PD3 **just** to query eMarkers (minor header change)

VPDs enable something else…(Soon™)

- The VPD architecture leaves room for adding charge-through to a future USB Type-C® / USB PD spec
- This charge-through approach would be dramatically simpler and cheaper than what's required of multi-port adapters today
- Most DRP implementations out now would be able to support it with a software update
- ...but it's not fully specified yet, so I'll have something to talk about next year

VPDs enable charge-through...CTVPDs

- USB Type-C® / USB PD spec contains a charge-through variant of the VPD architecture
- This charge-through approach is dramatically simpler and cheaper than what's required of multi-port adapters today
- Most DRP implementations out now would be able to support it with a software update
- So let's step through the process of...

Today's episode:

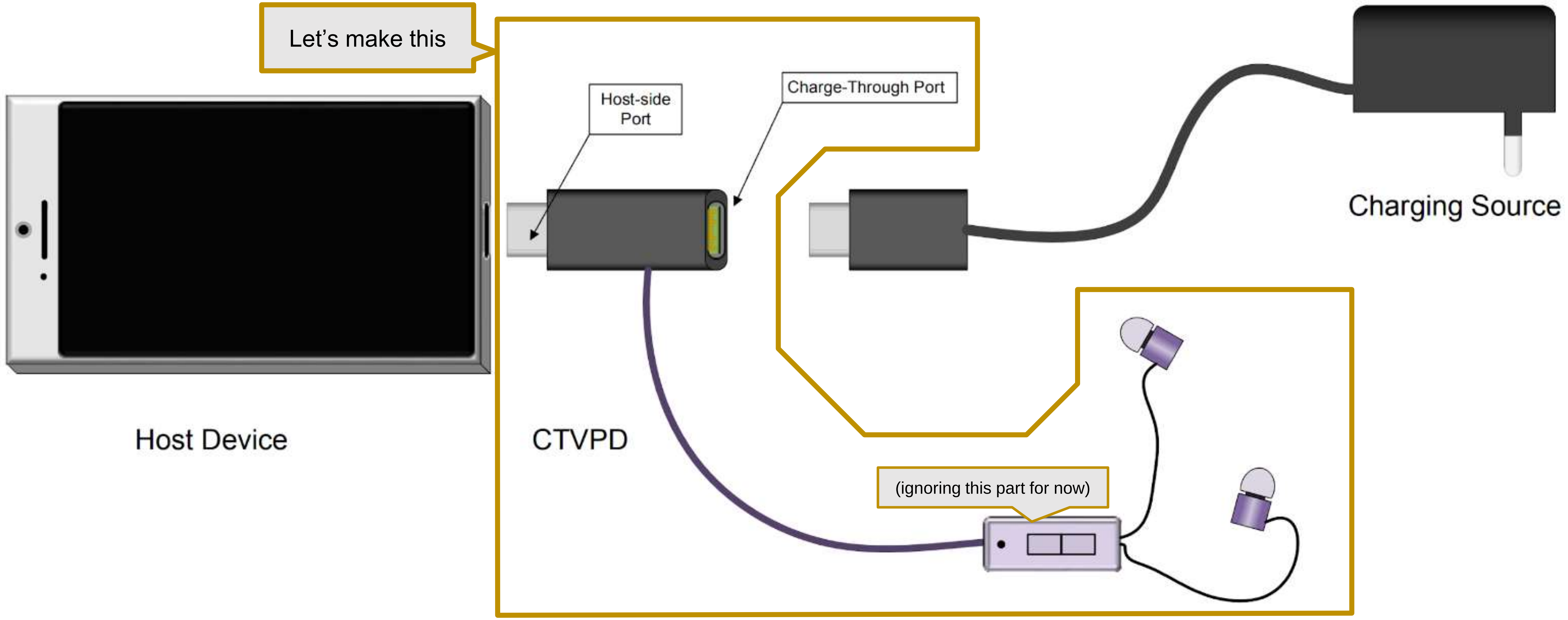
Making Your Own USB-C™ Thingamajig

in 9 Easy Steps

"FETs, States, and Extreme Isolation"

Our goal for today:

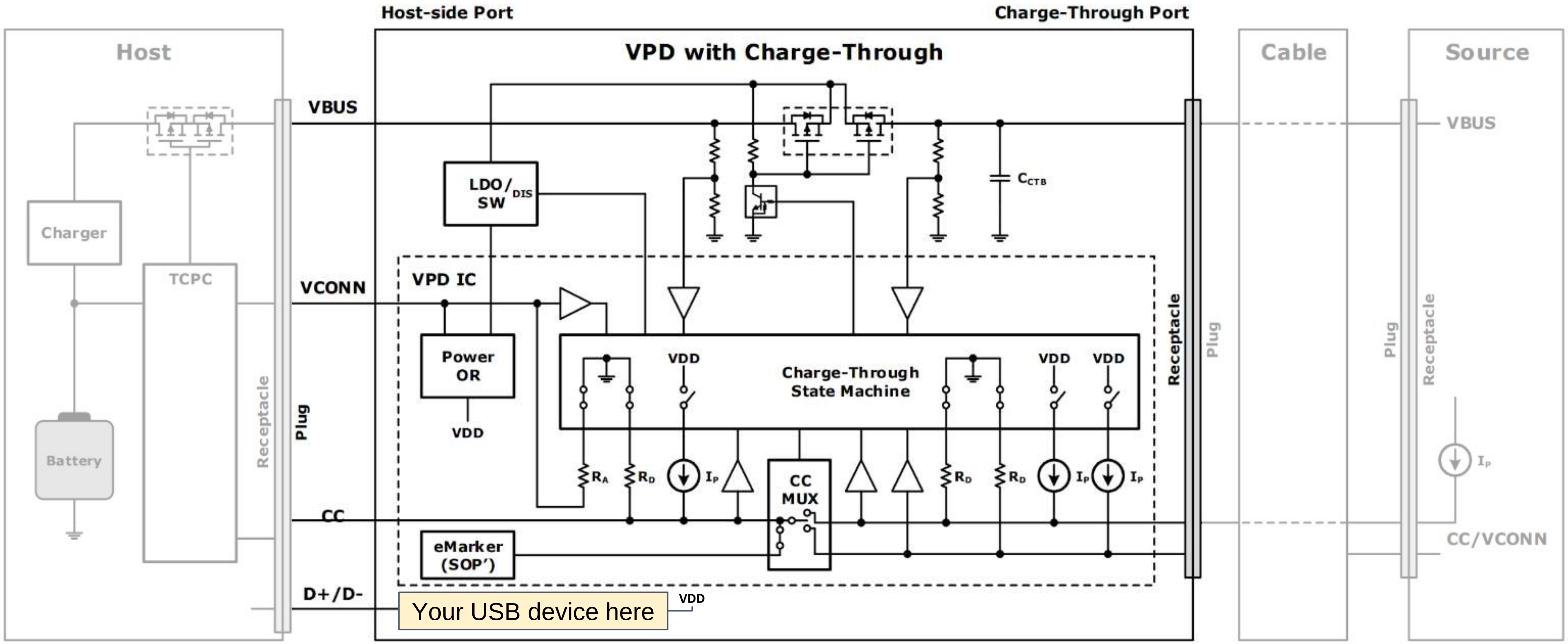
Figure 4-44 Example Charge-Through VCONN-Power USB Device Use Case



Source: USB Type-C Specification Release 1.4

Step 1: Read the specs

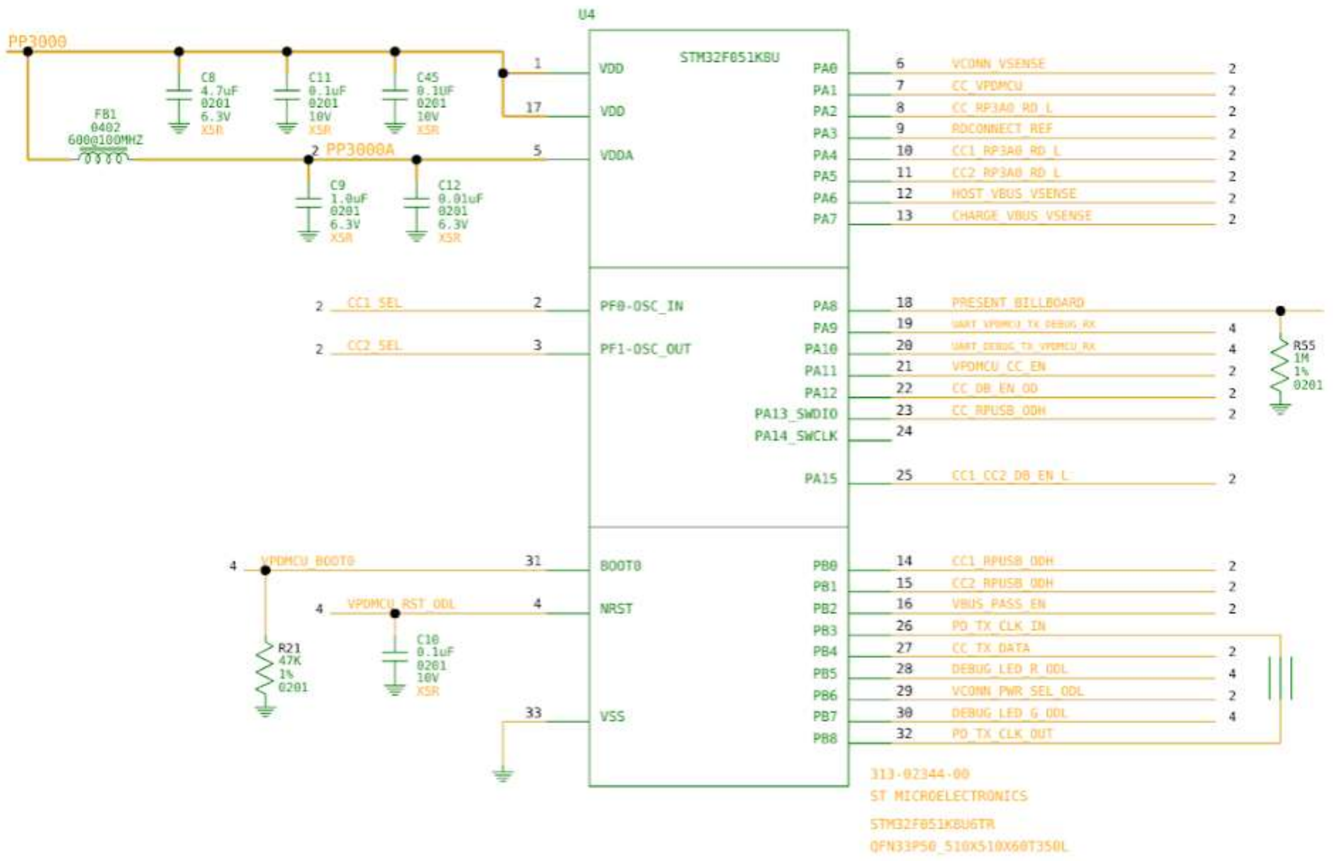
Figure 4-30 Example DRP to Charge-Through VCONN-Powered USB Device Model

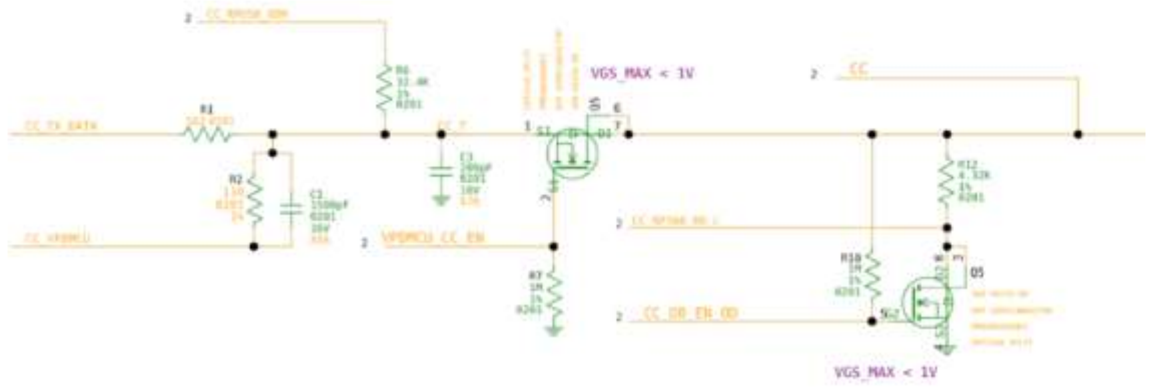


Source: USB Type-C Specification Release 1.4

Step 2: Start with a solid platform

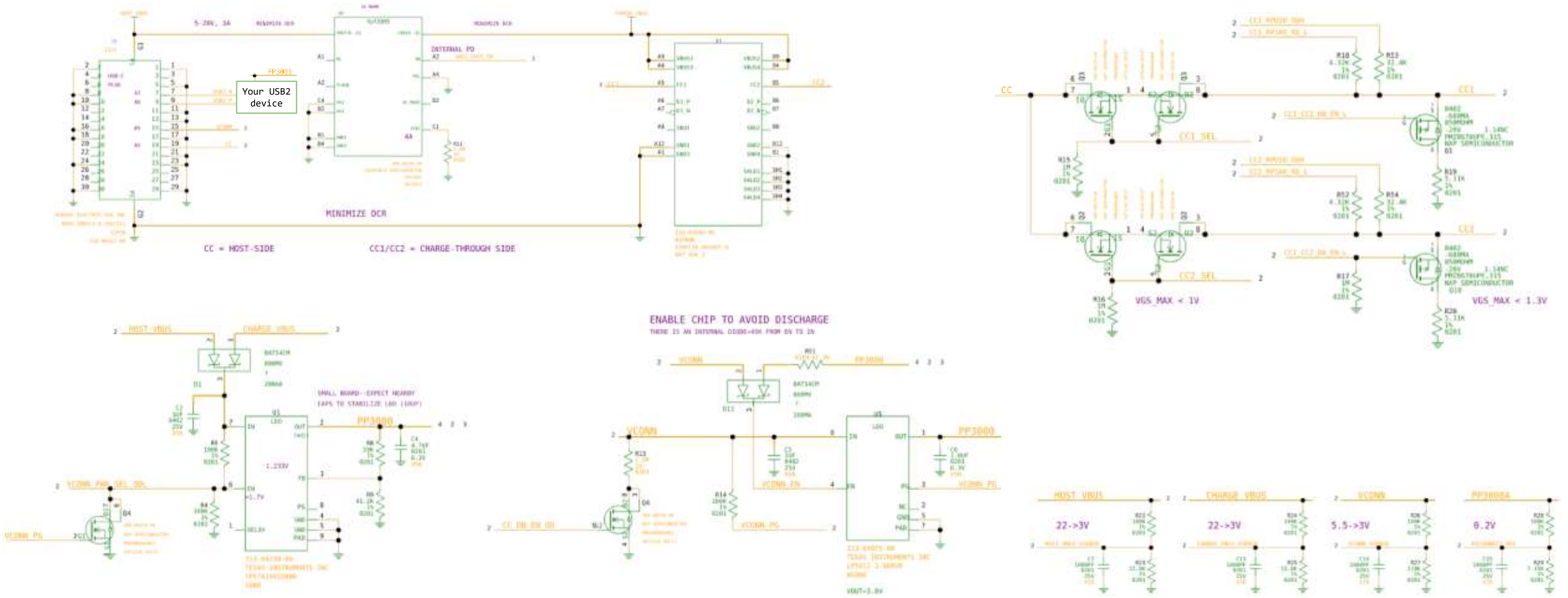
Or, barring that, something flexible enough to update in the field (securely!)





Source: https://chromium.googlesource.com/chromiumos/platform/ec/+/master/board/chocodile_vpdmcu/chocodile.html

Step 3: Add missing USB-C® features

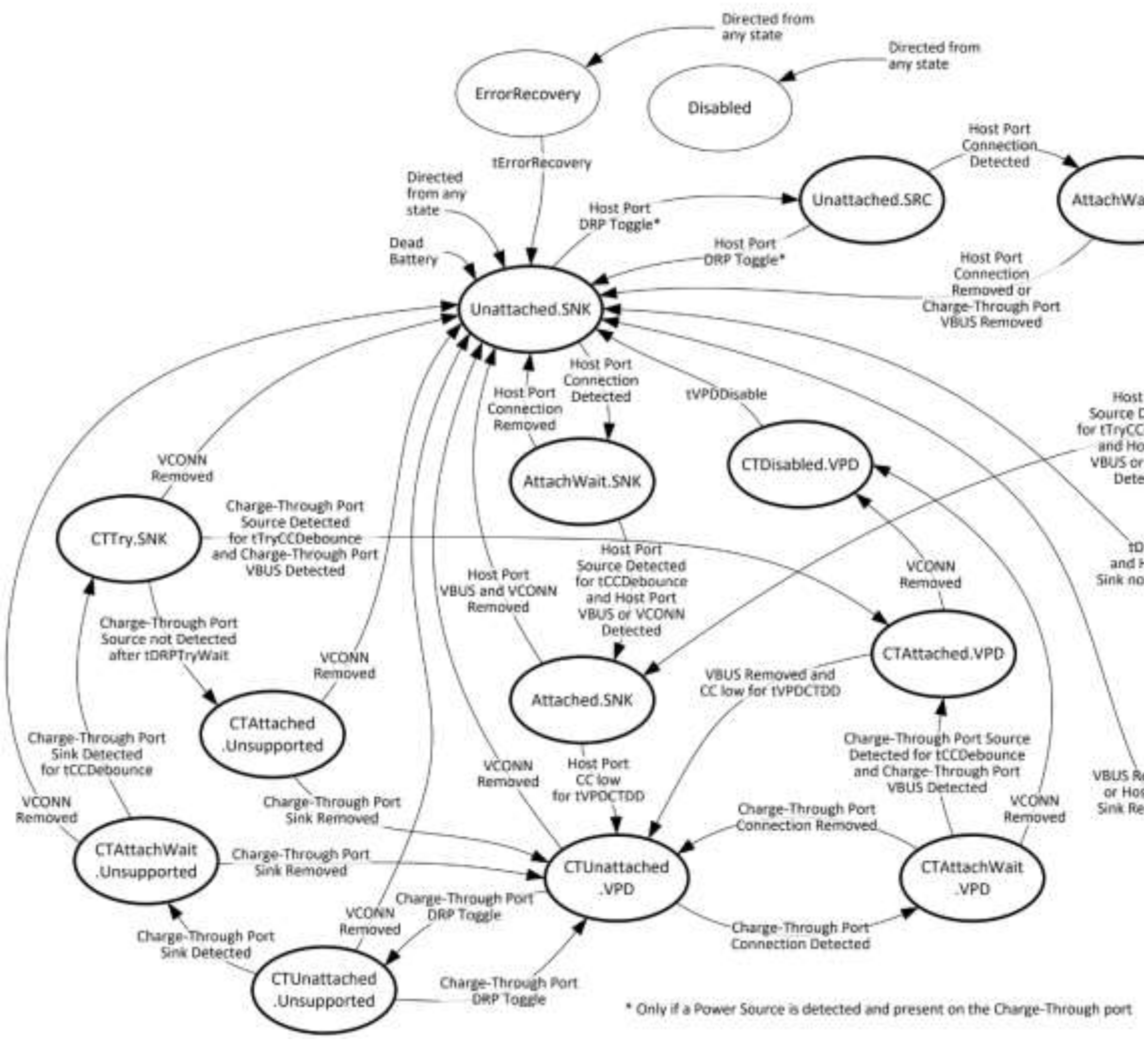


You may have to implement them with discrete parts if your foundation is old...

Source: https://chromium.googlesource.com/chromiumos/platform/ec/+/master/board/chocodile_vpdmcu/chocodile.html

Step 4: Read the specs again

Figure 4-18 Connection State Diagram: Charge-Through VPD


Directed from any state
ErrorRecovery
Directed from any state
Disabled
tErrorRecovery
Directed from any state
Dead Battery
Host Port DRP Toggle*
Host Port Connection Detected
Unattached.SRC
AttachWait.SRC
Host Port DRP Toggle*
Host Port Connection Removed or Charge-Through Port VBUS Removed
Host-Port VBUS at vSafe0V and Host Port Sink Detected for tCCDebounce
Unattached.SNK
Host Port Connection Detected
Host Port Connection Removed
tVPDDisable
Host Port Source Detected for tTryCCDebounce and Host-Port VBUS or VCONN Detected
Try.SNK
AttachWait.SNK
CTDisabled.VPD
VCONN Removed
CTTry.SNK
Charge-Through Port Source Detected for tTryCCDebounce and Charge-Through Port VBUS Detected
Host Port Source not Detected for tTryCCDebounce after tDRPTry
tDRPTry and Host Port Sink not Detected
Host Port VBUS and VCONN Removed
Host Port Source Detected for tCCDebounce and Host Port VBUS or VCONN Detected
VCONN Removed
Charge-Through Port Source not Detected after tDRPTryWait
VCONN Removed
CTAttached.VPD
TryWait.SRC
Attached.SNK
VBUS Removed and CC low for tVPDCTDD
CTAttached .Unsupported
Host-Port VBUS at vSafe0V and Host Port Sink Detected for tTryCCDebounce
Charge-Through Port Sink Detected for tCCDebounce
VCONN Removed
Host Port CC low for tVPDCTDD
Charge-Through Port Source Detected for tCCDebounce and Charge-Through Port VBUS Detected
VBUS Removed or Host Port Sink Removed
VCONN Removed
Charge-Through Port Sink Removed
Charge-Through Port Connection Removed
VCONN Removed
CTAttachWait .Unsupported
Charge-Through Port Sink Removed
CTUnattached .VPD
CTAttachWait .VPD
Attached.SRC
Charge-Through Port DRP Toggle
VCONN Removed
Charge-Through Port Sink Detected
Charge-Through Port Connection Detected
CTUnattached .Unsupported
Charge-Through Port DRP Toggle
* Only if a Power Source is detected and present on the Charge-Through port

Going over the state machines and their respective requirements is a great way to check that your hardware supports the high-level features needed, e.g.:

- Various isolation and power states
- Detection and monitoring
- Response times

Source: USB Type-C Specification Release 1.4

Step 5: Read the specs a third time

Figure 8-119 Responder Structured VDM Discover Identity State Diagram

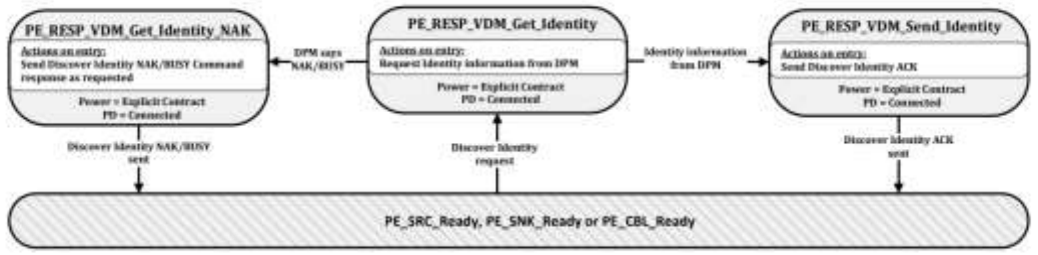
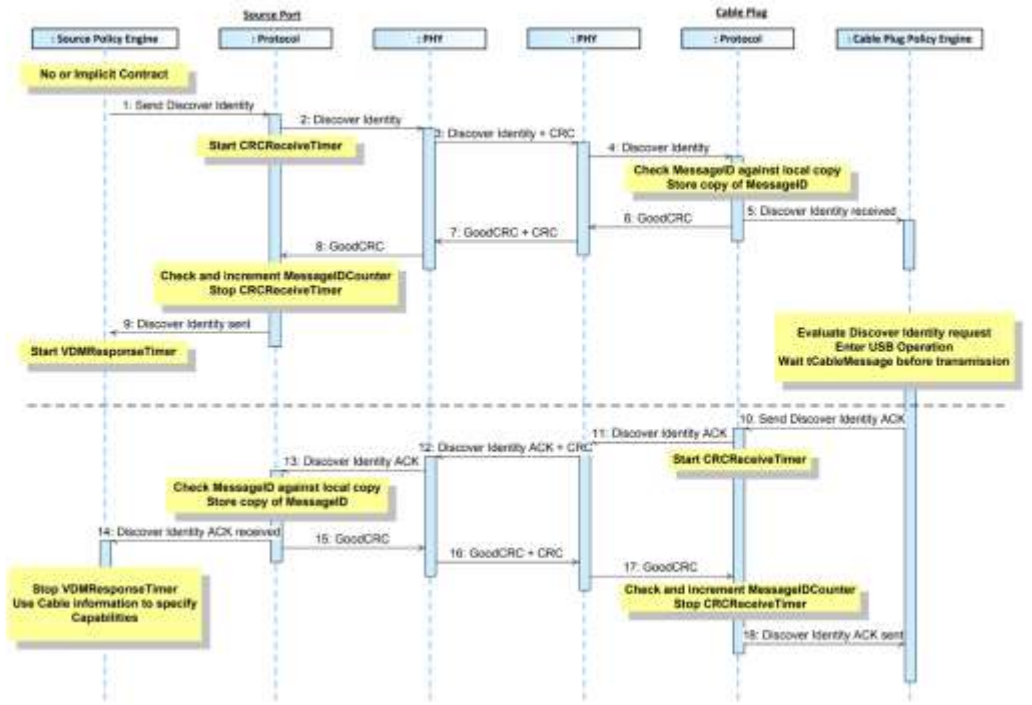

PE_RESP_VDM_Get_Identity_NAK
Actions on entry:
Send Discover Identity NAK/BUSY Command response as requested
Power = Explicit Contract
PD = Connected
DPM says NAK/BUSY
PE_RESP_VDM_Get_Identity
Actions on entry:
Request Identity information from DPM
Power = Explicit Contract
PD = Connected
Identity information from DPM
PE_RESP_VDM_Send_Identity
Actions on entry:
Send Discover Identity ACK
Power = Explicit Contract
PD = Connected
Discover Identity NAK/BUSY sent
Discover Identity request
Discover Identity ACK sent
PE_SRC_Ready, PE_SNK_Ready or PE_CBL_Ready

Figure 8-54 Source Port to Cable Plug Discover Identity


Source Port
Cable Plug
: Source Policy Engine
: Protocol
: PHY
: PHY
: Protocol
: Cable Plug Policy Engine
No or Implicit Contract
1: Send Discover Identity
2: Discover Identity
3: Discover Identity + CRC
4: Discover Identity
Start CRCReceiveTimer
Check MessageID against local copy
Store copy of MessageID
5: Discover Identity received
6: GoodCRC
7: GoodCRC + CRC
8: GoodCRC
Check and increment MessageIDCounter
Stop CRCReceiveTimer
9: Discover Identity sent
Start VDMResponseTimer
Evaluate Discover Identity request
Enter USB Operation
Wait tCableMessage before transmission
10: Send Discover Identity ACK
11: Discover Identity ACK
12: Discover Identity ACK + CRC
13: Discover Identity ACK
Start CRCReceiveTimer
Check MessageID against local copy
Store copy of MessageID
14: Discover Identity ACK received
15: GoodCRC
16: GoodCRC + CRC
17: GoodCRC
Stop VDMResponseTimer
Use Cable information to specify Capabilities
Check and increment MessageIDCounter
Stop CRCReceiveTimer
18: Discover Identity ACK sent

OK, this is getting quite abstract.

How do we transform this into code?

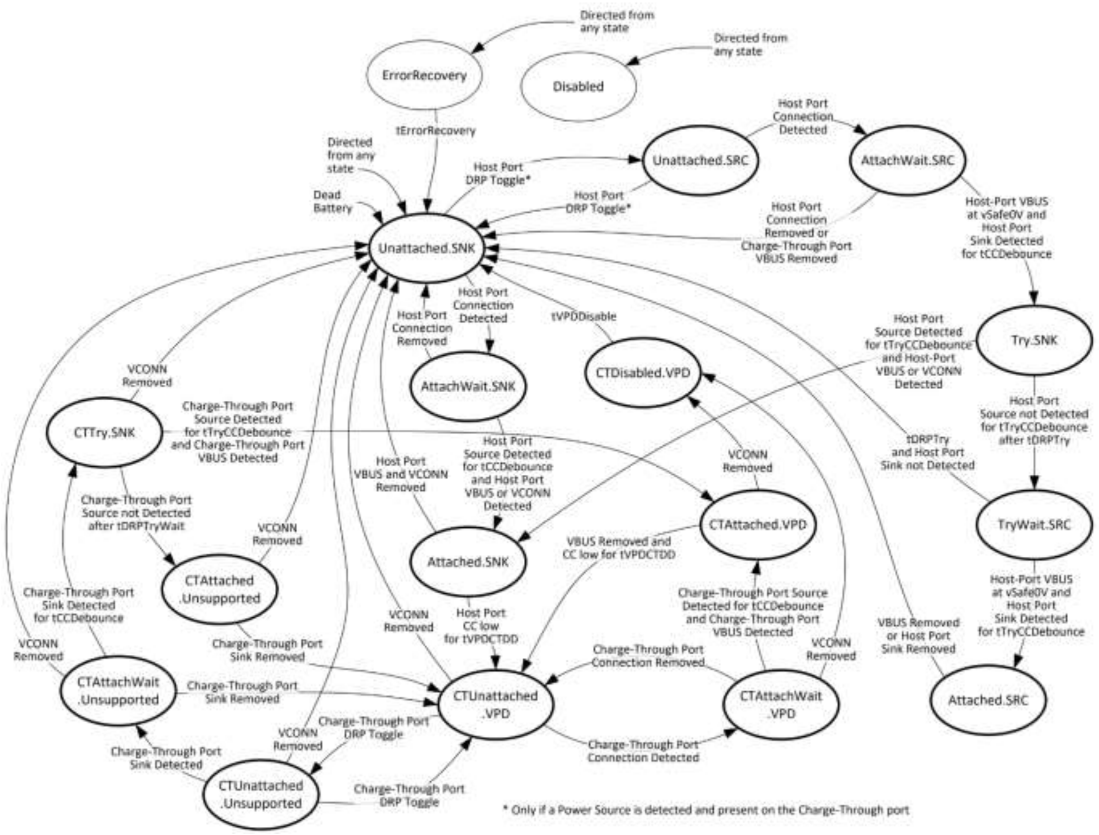
Source: USB Power Delivery Specification Revision 3.0, Version 1.2

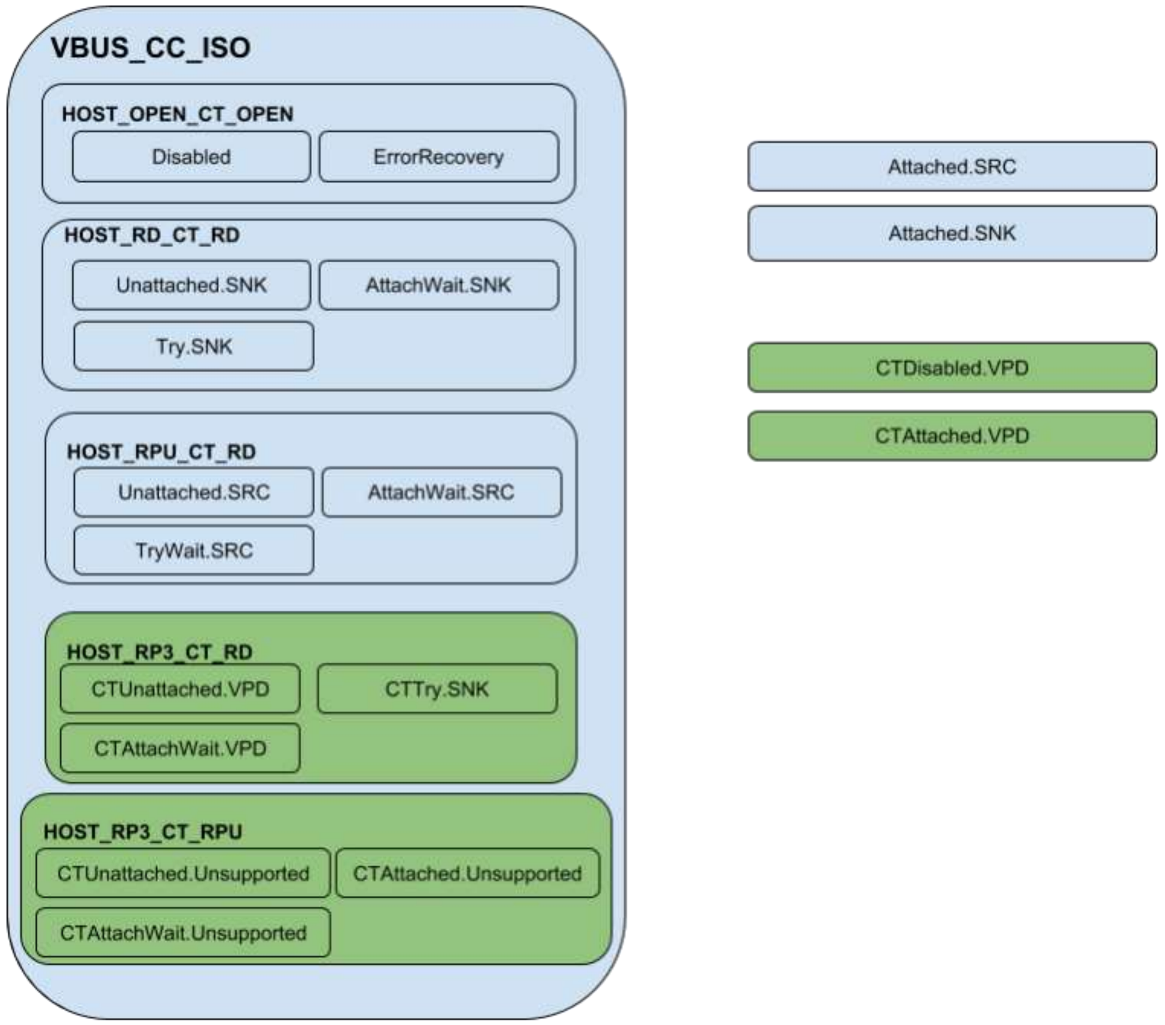
Enter the NEW! Chromium OS EC USB Type-C® Stack

- Complete rewrite of protocol and policy engines as state machines
- Close to 1:1 state correspondence with the specs
  - Some trivial USB PD transitory states were merged
  - Easy to visually confirm that everything matches the spec
- Adding new features is easy thanks to separation of policy layer
- Automated regression testing infrastructure
  - Intend to implement a bulk of the relevant tests from the compliance suite
- Supports USB PD3 (finally!) but takes ~9KB more of flash space (oh no)
- Interfaces with existing wide variety of TCPC hardware drivers (in EC)
- Runs as its own task, so should be easy to port to other OSes
- Rolling out to Chromebooks both new and old (as space allows)

Step 6: Translate the states from speccese to C

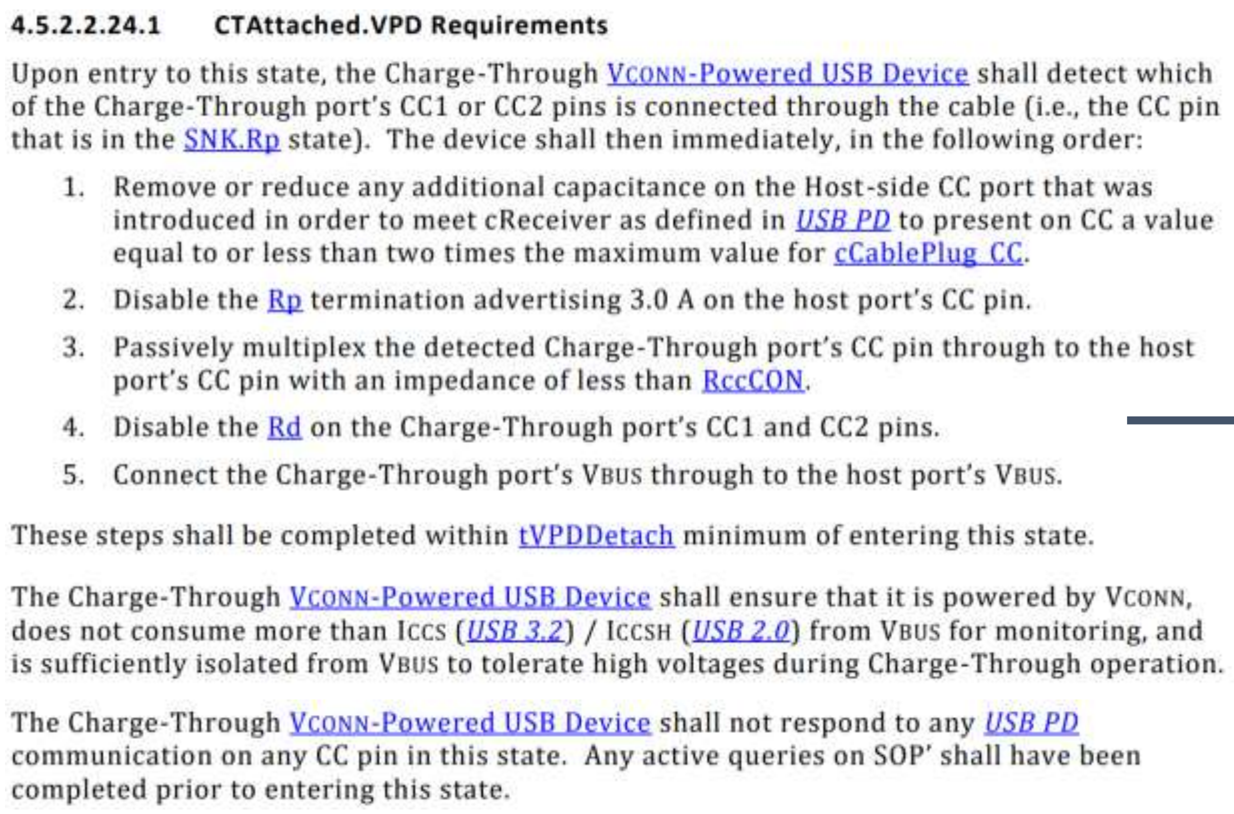
Figure 4-18 Connection State Diagram: Charge-Through VPD


Directed from any state
ErrorRecovery
Directed from any state
Disabled
tErrorRecovery
Directed from any state
Dead Battery
Host Port DRP Toggle*
Unattached.SRC
Host Port Connection Detected
AttachWait.SRC
Host Port DRP Toggle*
Host Port Connection Removed or Charge-Through Port VBUS Removed
Host-Port VBUS at vSafe0V and Host Port Sink Detected for tCCDebounce
Unattached.SNK
Host Port Connection Detected
Host Port Connection Removed
tVPDDisable
CTDisabled.VPD
Host Port Source Detected for tTryCCDebounce and Host-Port VBUS or VCONN Detected
Try.SNK
AttachWait.SNK
VCONN Removed
CTTry.SNK
Charge-Through Port Source Detected for tTryCCDebounce and Charge-Through Port VBUS Detected
Host Port Source not Detected for tTryCCDebounce after tDRPTry
Host Port VBUS and VCONN Removed
Host Port Source Detected for tCCDebounce and Host Port VBUS or VCONN Detected
VCONN Removed
tDRPTry and Host Port Sink not Detected
Charge-Through Port Source not Detected after tDRPTryWait
VCONN Removed
CTAttached.VPD
TryWait.SRC
CTAttached .Unsupported
Attached.SNK
VBUS Removed and CC low for tVPDCTDD
Charge-Through Port Sink Detected for tCCDebounce
VCONN Removed
Host Port CC low for tVPDCTDD
Charge-Through Port Source Detected for tCCDebounce and Charge-Through Port VBUS Detected
Host-Port VBUS at vSafe0V and Host Port Sink Detected for tTryCCDebounce
VCONN Removed
Charge-Through Port Sink Removed
Charge-Through Port Connection Removed
VCONN Removed
VBUS Removed or Host Port Sink Removed
CTAttachWait .Unsupported
Charge-Through Port Sink Removed
CTUnattached .VPD
CTAttachWait .VPD
Attached.SRC
Charge-Through Port DRP Toggle
VCONN Removed
Charge-Through Port Sink Detected
Charge-Through Port Connection Detected
CTUnattached .Unsupported
Charge-Through Port DRP Toggle
* Only if a Power Source is detected and present on the Charge-Through port



Source: USB Type-C Specification Release 1.4

Step 6: Translate the states from speccese to C



4.5.2.2.24.1 CTAttached.VPD Requirements

Upon entry to this state, the Charge-Through VCONN-Powered USB Device shall detect which of the Charge-Through port's CC1 or CC2 pins is connected through the cable (i.e., the CC pin that is in the SNK.Rp state). The device shall then immediately, in the following order:

1. Remove or reduce any additional capacitance on the Host-side CC port that was introduced in order to meet cReceiver as defined in *USB PD* to present on CC a value equal to or less than two times the maximum value for cCablePlug_CC.
2. Disable the Rp termination advertising 3.0 A on the host port's CC pin.
3. Passively multiplex the detected Charge-Through port's CC pin through to the host port's CC pin with an impedance of less than RccCON.
4. Disable the Rd on the Charge-Through port's CC1 and CC2 pins.
5. Connect the Charge-Through port's VBUS through to the host port's VBUS.

These steps shall be completed within tVPDDetach minimum of entering this state.

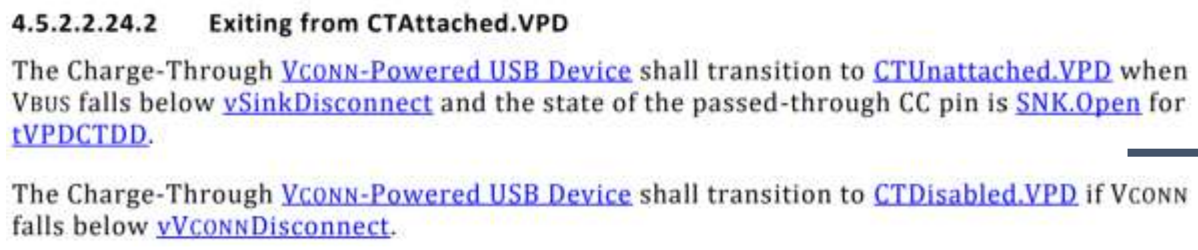
The Charge-Through VCONN-Powered USB Device shall ensure that it is powered by VCONN, does not consume more than ICCS (*USB 3.2*) / ICCSH (*USB 2.0*) from VBUS for monitoring, and is sufficiently isolated from VBUS to tolerate high voltages during Charge-Through operation.

The Charge-Through VCONN-Powered USB Device shall not respond to any *USB PD* communication on any CC pin in this state. Any active queries on SOP' shall have been completed prior to entering this state.

```c
/* CTAttached.VPD */
static void tc_ct_attached_vpd_entry(const int port) {
    int cc1, cc2;
    /* Get power from VCONN */
    vpd_vconn_pwr_sel_odl(PWR_VCONN);
    /** Detect which of the Charge-Through port's CC1 or CC2
     * pins is connected through the cable */
    vpd_ct_get_cc(&cc1, &cc2);
    tc[port].ct_cc = cc_is_rp(cc2) ? CT_CC2 : CT_CC1;
    /* 1. Remove or reduce any additional capacitance on the Host-side CC port */
    vpd_mcu_cc_en(0);
    /* 2. Disable the Rp termination advertising 3.0 A on the host port's CC pin */
    vpd_host_set_pull(TYPEC_CC_OPEN, 0);
    /** 3. Passively multiplex the detected Charge-Through port's
     * CC pin through to the host port's CC */
    vpd_ct_cc_sel(tc[port].ct_cc);
    /* 4. Disable the Rd on the Charge-Through port's CC1 and CC2 pins */
    vpd_ct_set_pull(TYPEC_CC_OPEN, 0);
    /* 5. Connect the Charge-Through port's VBUS through to the host port's VBUS */
    vpd_vbus_pass_en(1);
    tc[port].cc_state = PD_CC_UNSET;
}
```

Source: USB Type-C Specification Release 1.4
Source: https://chromium.googlesource.com/chromiumos/platform/ec/+/master/common/usbc/usb_tc_ctvpd_sm.c

Step 6: Translate the states from speccese to C



4.5.2.2.24.2 Exiting from CTAttached.VPD

The Charge-Through VCONN-Powered USB Device shall transition to CTUnattached.VPD when VBUS falls below vSinkDisconnect and the state of the passed-through CC pin is SNK.Open for tVPDCTDD.

The Charge-Through VCONN-Powered USB Device shall transition to CTDisabled.VPD if VCONN falls below vVCONNDisconnect.

```
static void tc_ct_attached_vpd_run(const int port) {
    int new_cc_state, cc1, cc2;
    /** A Charge-Through VCONN-Powered USB Device shall transition to
     * CTDisabled.VPD if VCONN falls below vVCONNDisconnect. */
    if (!vpd_is_vconn_present()) {
        set_state_tc(port, TC_CT_DISABLED_VPD);
        return;
    }
    /* Check CT CC for connection */
    vpd_ct_get_cc(&cc1, &cc2);
    if ((tc[port].ct_cc ? cc2 : cc1) == TYPEC_CC_VOLT_OPEN)
        new_cc_state = PD_CC_NONE;
    else
        new_cc_state = PD_CC_DFP_ATTACHED;
    /* Debounce the cc state */
    if (new_cc_state != tc[port].cc_state) {
        tc[port].cc_state = new_cc_state;
        tc[port].cc_debounce = get_time().val + PD_T_VPDCTDD;
        return;
    }
    if (get_time().val < tc[port].pd_debounce)
        return;
    /** A Charge-Through VCONN-Powered USB Device shall transition to
     * CTUnattached.VPD when VBUS falls below vSinkDisconnect and the
     * state of the passed-through CC pin is SNK.Open for tVPDCTDD. */
    if (tc[port].cc_state == PD_CC_NONE && !vpd_is_ct_vbus_present())
        set_state_tc(port, TC_CT_UNATTACHED_VPD);
}
```

Source: USB Type-C Specification Release 1.4
Source: https://chromium.googlesource.com/chromiumos/platform/ec/+/master/common/usbc/usb_tc_ctvpd_sm.c

Step 7: Test test test test test test test test!

Testing basic functionality against a Macbook and charger is a great start, but not a representative sampling of the wide diversity of the USB Type-C® ecosystem nor sufficient coverage of all the edge-cases.

Have you considered:

- e-marked cables?
- non-PD phones?
- non-Thunderbolt hosts/devices?
- USB2-only hosts/devices?
- PPS sources?
- sources/sinks that dynamically change PDOs?
- sinks that strictly interpret “unconstrained power”?
- self-powered sinks?
- sinks that have high minimum power requirements?
- hosts/devices that combine the two USB2 pairs?
- power role/data role/Vconn/fast role swaps?
- buggy devices that get caught in hard-reset loops?
- random stuff found on the internet?

Test labs can help expand your test coverage, but it’s ultimately up to you and your engineering creativity to ensure all the weird combinations and potential behaviors get tested.

Step 7b: Report bugs!

- Often, novel USB-C® devices can expose bugs in shipping products
- Report bugs to the vendor rather than shipping with workarounds
- Bugs on complicated products in the market can generally be fixed with software/firmware updates
- Hacky workarounds on simple accessories often cannot, and poison the ecosystem with ever-growing test matrices and weird corner-cases
- Workarounds are usually “wrong” in the context of spec compliance

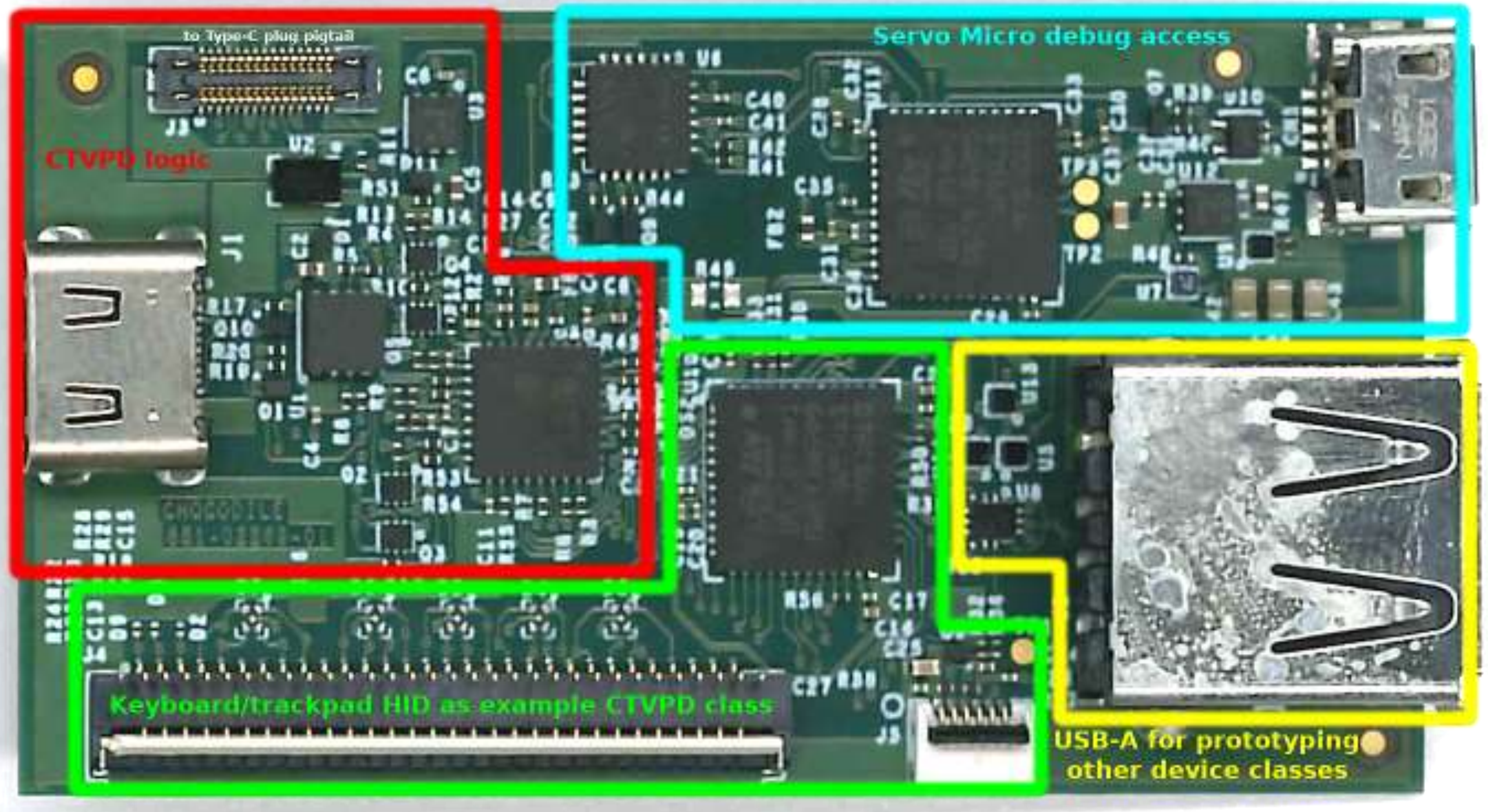
Apple bug report system:	**https://bugreport.apple.com**
Chromebook issue tracker:	**https://bugs.chromium.org/p/chromium/issues/entry**
Android/Pixel Phone issue tracker:	**https://issuetracker.google.com/issues/new?component=190923**

Your Company's bug report system?
Please provide a public link to report issues to make the ecosystem healthier!

Step 8: Compliance

- If there’s an official compliance category for your class of device, get certified. The logos are pretty alright and using them improves the ecosystem for everyone
- Either way, buy/rent/barter for time on more than one of the several compliance testers available
- Beware: compliance doesn’t catch everything and isn’t a substitute for a wide variety of testing (see step 7)
- Leave time in your schedule for multiple rounds of testing and compliance
- If you find a bug or a blind spot in compliance, report it to **techadmin@usb.org**

Step 9: Show off your cool thingamajig at USB DevDays


to Type-C plug pigtail
CTVPD logic
Servo Micro debug access
Keyboard/trackpad HID as example CTVPD class
USB-A for prototyping other device classes

Q&A