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
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
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

Let’s make this

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
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

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
Step 5: Read the specs a third time

OK, this is getting quite abstract.

How do we transform this into code?
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 specceces to C
Step 6: Translate the states from speccese to C

```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/usbc_tc_ctvpd_sm.c
Step 6: Translate the states from speccese to C

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

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.
Step 7: Test test 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](https://bugreport.apple.com)
Chromebook issue tracker: [https://bugs.chromium.org/p/chromium/issues/entry](https://bugs.chromium.org/p/chromium/issues/entry)
Android/Pixel Phone issue tracker: [https://issuetracker.google.com/issues/new?component=190923](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
Q&A