USB Type-C connector class

Introduction

The typec class is meant for describing the USB Type-C ports in a system to the user space in unified fashion. The class is designed to provide nothing else except the user space interface implementation in hope that it can be utilized on as many platforms as possible.

The platforms are expected to register every USB Type-C port they have with the class. In a normal case the registration will be done by a USB Type-C or PD PHY driver, but it may be a driver for firmware interface such as UCSI, driver for USB PD controller or even driver for Thunderbolt3 controller. This document considers the component registering the USB Type-C ports with the class as "port driver".

On top of showing the capabilities, the class also offer user space control over the roles and alternate modes of ports, partners and cable plugs when the port driver is capable of supporting those features.

The class provides an API for the port drivers described in this document. The attributes are described in Documentation/ABI/testing/sysfs-class-typec.

User space interface

Every port will be presented as its own device under /sys/class/typec/. The first port will be named "port0", the second "port1" and so on.

When connected, the partner will be presented also as its own device under /sys/class/typec/. The parent of the partner device will always be the port it is attached to. The partner attached to port "port0" will be named "port0-partner". Full path to the device would be /sys/class/typec/port0/port0-partner/.

The cable and the two plugs on it may also be optionally presented as their own devices under /sys/class/typec/. The cable attached to the port "port0" port will be named port0-cable and the plug on the SOP Prime end (see USB Power Delivery Specification ch. 2.4) will be named "port0-plug0" and on the SOP Double Prime end "port0-plug1". The parent of a cable will always be the port, and the parent of the cable plugs will always be the cable.

If the port, partner or cable plug supports Alternate Modes, every supported Alternate Mode SVID will have their own device describing them. Note that the Alternate Mode devices will not be attached to the typec class. The parent of an alternate mode will be the device that supports it, so for example an alternate mode of port0-partner will be presented under /sys/class/typec/port0-partner/. Every mode that is supported will have its own group under the Alternate Mode device named "mode<index>", for example /sys/class/typec/port0/<alternate mode>/mode1/. The requests for entering/exiting a mode can be done with "active" attribute file in that group.

Driver API

Registering the ports

The port drivers will describe every Type-C port they control with struct typec_capability data structure, and register them with the following API:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 65)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_register_port typec_unregister_port
```

When registering the ports, the prefer_role member in struct typec_capability deserves special notice. If the port that is being registered does not have initial role preference, which means the port does not execute Try.SNK or Try.SRC by default, the member must have value TYPEC_NO_PREFERRED_ROLE. Otherwise if the port executes Try.SNK by default, the member must have value TYPEC_DEVICE, and with Try.SRC the value must be TYPEC_HOST.

Registering Partners

After successful connection of a partner, the port driver needs to register the partner with the class. Details about the partner need to be described in struct typec_partner_desc. The class copies the details of the partner during registration. The class offers the following API for registering/unregistering partners.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 84)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_register_partner typec_unregister_partner
```

The class will provide a handle to struct typec_partner if the registration was successful, or NULL.

If the partner is USB Power Delivery capable, and the port driver is able to show the result of Discover Identity command, the partner descriptor structure should include handle to struct usb_pd_identity instance. The class will then create a sysfs directory for the identity under the partner device. The result of Discover Identity command can then be reported with the following API:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 96)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_partner_set_identity
```

Registering Cables

After successful connection of a cable that supports USB Power Delivery Structured VDM "Discover Identity", the port driver needs to register the cable and one or two plugs, depending if there is CC Double Prime controller present in the cable or not. So a cable capable of SOP Prime communication, but not SOP Double Prime communication, should only have one plug registered. For more information about SOP communication, please read chapter about it from the latest USB Power Delivery specification.

The plugs are represented as their own devices. The cable is registered first, followed by registration of the cable plugs. The cable will be the parent device for the plugs. Details about the cable need to be described in struct typec_cable_desc and about a plug in struct typec_plug_desc. The class copies the details during registration. The class offers the following API for registering/unregistering cables and their plugs:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 117)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_register_cable typec_unregister_cable typec_register_plug typec_unregister_plu
```

The class will provide a handle to struct typec_cable and struct typec_plug if the registration is successful, or NULL if it isn't.

If the cable is USB Power Delivery capable, and the port driver is able to show the result of Discover Identity command, the cable descriptor structure should include handle to struct usb_pd_identity instance. The class will then create a sysfs directory for the identity under the cable device. The result of Discover Identity command can then be reported with the following API:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 129)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_cable_set_identity
```

Notifications

When the partner has executed a role change, or when the default roles change during connection of a partner or cable, the port driver must use the following APIs to report it to the class:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 139)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_set_data_role typec_set_pwr_role typec_set_vconn_role typec_set_pwr_opmode
```

Alternate Modes

USB Type-C ports, partners and cable plugs may support Alternate Modes. Each Alternate Mode will have identifier called SVID, which is either a Standard ID given by USB-IF or vendor ID, and each supported SVID can have 1 - 6 modes. The class provides struct typec_mode_desc for describing individual mode of a SVID, and struct typec_altmode_desc which is a container for all the supported modes.

Ports that support Alternate Modes need to register each SVID they support with the following API:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 154)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_port_register_altmode
```

If a partner or cable plug provides a list of SVIDs as response to USB Power Delivery Structured VDM Discover SVIDs message, each SVID needs to be registered.

API for the partners:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 163)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_partner_register_altmode
```

API for the Cable Plugs:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 168)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_plug_register_altmode
```

So ports, partners and cable plugs will register the alternate modes with their own functions, but the registration will always return a handle to struct typec_altmode on success, or NULL. The unregistration will happen with the same function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 176)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_unregister_altmode
```

If a partner or cable plug enters or exits a mode, the port driver needs to notify the class with the following API:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 182)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_altmode_update_active
```

Multiplexer/DeMultiplexer Switches

USB Type-C connectors may have one or more mux/demux switches behind them. Since the plugs can be inserted right-side-up or upside-down, a switch is needed to route the correct data pairs from the connector to the USB controllers. If Alternate or Accessory

Modes are supported, another switch is needed that can route the pins on the connector to some other component besides USB. USB Type-C Connector Class supplies an API for registering those switches.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 195)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/mux.c
   :functions: typec_switch_register typec_switch_unregister typec_mux_register typec_mux_unregiste
```

In most cases the same physical mux will handle both the orientation and mode. However, as the port drivers will be responsible for the orientation, and the alternate mode drivers for the mode, the two are always separated into their own logical components: "mux" for the mode and "switch" for the orientation.

When a port is registered, USB Type-C Connector Class requests both the mux and the switch for the port. The drivers can then use the following API for controlling them:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]typec.rst, line 207)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/usb/typec/class.c
   :functions: typec_set_orientation typec_set_mode
```

If the connector is dual-role capable, there may also be a switch for the data role. USB Type-C Connector Class does not supply separate API for them. The port drivers can use USB Role Class API with those.

Illustration of the muxes behind a connector that supports an alternate mode: