USB3 debug port

Author: Lu Baolu <baolu.lu@linux.intel.com>
Date: March 2017

GENERAL

This is a HOWTO for using the USB3 debug port on x86 systems.

Before using any kernel debugging functionality based on USB3 debug port, you need to:

```
1) check whether any USB3 debug port is available in
   your system;
2) check which port is used for debugging purposes;
3) have a USB 3.0 super-speed A-to-A debugging cable.
```

INTRODUCTION

The xHCI debug capability (DbC) is an optional but standalone functionality provided by the xHCI host controller. The xHCI specification describes DbC in the section 7.6.

When DbC is initialized and enabled, it will present a debug device through the debug port (normally the first USB3 super-speed port). The debug device is fully compliant with the USB framework and provides the equivalent of a very high performance full-duplex serial link between the debug target (the system under debugging) and a debug host.

EARLY PRINTK

DbC has been designed to log early printk messages. One use for this feature is kernel debugging. For example, when your machine crashes very early before the regular console code is initialized. Other uses include simpler, lockless logging instead of a full- blown printk console driver and klogd.

On the debug target system, you need to customize a debugging kernel with CONFIG_EARLY_PRINTK_USB_XDBC enabled. And, add below kernel boot parameter:

```
"earlyprintk=xdbc"
```

If there are multiple xHCI controllers in your system, you can append a host contoller index to this kernel parameter. This index starts from 0.

Current design doesn't support DbC runtime suspend/resume. As the result, you'd better disable runtime power management for USB subsystem by adding below kernel boot parameter:

```
"usbcore.autosuspend=-1"
```

Before starting the debug target, you should connect the debug port to a USB port (root port or port of any external hub) on the debug host. The cable used to connect these two ports should be a USB 3.0 super-speed A-to-A debugging cable.

During early boot of the debug target, DbC will be detected and initialized. After initialization, the debug host should be able to enumerate the debug device in debug target. The debug host will then bind the debug device with the usb_debug driver module and create the /dev/ttyUSB device.

If the debug device enumeration goes smoothly, you should be able to see below kernel messages on the debug host:

```
# tail -f /var/log/kern.log
[ 1815.983374] usb 4-3: new SuperSpeed USB device number 4 using xhci_hcd
[ 1815.999595] usb 4-3: LPM exit latency is zeroed, disabling LPM.
[ 1815.999899] usb 4-3: New USB device found, idVendor=1d6b, idProduct=0004
[ 1815.999902] usb 4-3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1815.999903] usb 4-3: Product: Remote GDB
[ 1815.999904] usb 4-3: Manufacturer: Linux
[ 1815.999905] usb 4-3: SerialNumber: 0001
[ 1816.000240] usb_debug 4-3:1.0: xhci_dbc converter detected
[ 1816.000360] usb 4-3: xhci_dbc converter now attached to ttyUSB0
```

You can use any communication program, for example minicom, to read and view the messages. Below simple bash scripts can help you to check the sanity of the setup.

```
===== start of bash scripts =============
#!/bin/bash

while true ; do
        while [ ! -d /sys/class/tty/ttyUSB0 ] ; do
                :
        done
```

```
cat /dev/ttyUSB0
done
===== end of bash scripts ===============
```

Serial TTY

The DbC support has been added to the xHCI driver. You can get a debug device provided by the DbC at runtime.

In order to use this, you need to make sure your kernel has been configured to support USB_XHCI_DBGCAP. A sysfs attribute under the xHCI device node is used to enable or disable DbC. By default, DbC is disabled:

```
root@target:/sys/bus/pci/devices/0000:00:14.0# cat dbc
disabled
```

Enable DbC with the following command:

```
root@target:/sys/bus/pci/devices/0000:00:14.0# echo enable > dbc
```

You can check the DbC state at anytime:

```
root@target:/sys/bus/pci/devices/0000:00:14.0# cat dbc
enabled
```

Connect the debug target to the debug host with a USB 3.0 super- speed A-to-A debugging cable. You can see /dev/ttyDBC0 created on the debug target. You will see below kernel message lines:

```
root@target: tail -f /var/log/kern.log
[  182.730103] xhci_hcd 0000:00:14.0: DbC connected
[  191.169420] xhci_hcd 0000:00:14.0: DbC configured
[  191.169597] xhci_hcd 0000:00:14.0: DbC now attached to /dev/ttyDBC0
```

Accordingly, the DbC state has been brought up to:

```
root@target:/sys/bus/pci/devices/0000:00:14.0# cat dbc
configured
```

On the debug host, you will see the debug device has been enumerated. You will see below kernel message lines:

```
root@host: tail -f /var/log/kern.log
[   79.454780] usb 2-2.1: new SuperSpeed USB device number 3 using xhci_hcd
[   79.475003] usb 2-2.1: LPM exit latency is zeroed, disabling LPM.
[   79.475389] usb 2-2.1: New USB device found, idVendor=1d6b, idProduct=0010
[   79.475390] usb 2-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[   79.475391] usb 2-2.1: Product: Linux USB Debug Target
[   79.475392] usb 2-2.1: Manufacturer: Linux Foundation
[   79.475393] usb 2-2.1: SerialNumber: 0001
```

The debug device works now. You can use any communication or debugging program to talk between the host and the target.