Zynq Linux USB Device Driver

创建者Confluence Wiki Admin
由 anuragku 最近更新于 Nov 04, 2019

Table of Contents

On-the-Go. Two identical controllers are in the Zynq-7000 device. Each controller is configured and controlled independently. The USB controller I/O uses the ULPI protocol to connect external ULPI PHY via the MIO pins. The ULPI interface provides an 8-bit parallel SDR data path from the controller's internal UTMI-like bus to the PHY. The ULPI interface minimizes device pin count and is controlled by a 60 MHz clock output from the PHY

HW/IP Features

The USB controller has the following key features:

- USB 2.0 High Speed Host controller (480 Mb/s)
  Intel® EHCI software programming model.
- USB 2.0 HS and FS Device controller.
  Up to 12 Endpoint: Control Endpoint plus 11 configurable Endpoints
  USB 1.1 legacy FS/LS.
  Embedded Transaction Translator to support FS/LS in Host mode.
- On-the-Go, OTG 1.3 supplement.
  Host Negotiation Protocol (HNP).
  Session Request Protocol (SRP).
- All USB Transaction types
  Control, Bulk, Interrupt, Isochronous
- Local DMA Engine.
- AHB Bus Master.
- Transfers data between system memory and controller FIFOs.
  Processes transfer descriptors for Device Endpoints and Host Schedules.
- Protocol Engine
  Interprets USB packets
  Responds in real-time based on controller status
- Port/Transceiver Controller
  8-bit parallel data pass-thru bus
- ULPI Link Wrapper
  Translates Rx and Tx transfers between ULPI I/O interface and a UTMI-like interface.
  Bridge between the protocol engine and the ULPI interface.
  Rx and Tx commands
- ULPI I/O interface
  8-bit SDR data plus clock, direction, next, stop signals.
  12 ULPI PHY signals via MIO pins.
  Clocked by PHY in Clock-out mode.
  Viewport access to ULPI PHY registers

Features supported by driver

- All the HW/IP features are supported by driver

Missing features, Known Issues, limitations

None

Host Mode

Kernel Configuration

Ensure the below config parameters are selected

```
Device Drivers
USB support
    <*> Support for Host-side USB
    <*> EHCI HCD (USB 2.0) support
    <*> USB Mass Storage support
    <*> ChipIdea Highspeed Dual Role Controller
    <*> ChipIdea host controller
        USB Physical Layer drivers --->
        <*> Generic ULPI Transceiver Driver
```

Devicetree

```
    usb_0: usb@e0002000 {
```

```
            interrupt-parent = <&intc>;
            interrupts = <0 21 4>;
            reg = <0xe0002000 0x1000>;
            usb-phy = <&usb_phy0>;
        };

    usb_phy0: phy0 {
        compatible = "ulpi-phy";
        #phy-cells = <0>;
        reg = <0xe0002000 0x1000>;
        view-port = <0x170>;
        drv-vbus;
    }
```

Performance

Host Mode	25.00 MB/sec	Tool: hdparm

Test Procedure

Tested with mas-storage device.
Connect the mass storage device and perform the file read/write operations.

Peripheral Mode

Kernel Configuration

Ensure the below config parameters are selected
Mass Storage

```
    <*> ChipIdea device controller
    <*> USB Gadget Support
          <M> USB Gadget Drivers
          <M> USB functions configurable through configfs
          [*] Mass storage
```

Ethernet

```
<M>  USB Gadget Support
     <M>  USB Gadget Drivers
     <M>  USB functions configurable through configfs
     [*]  RNDIS
```

Devicetree

```
      usb_0: usb@e0002000 {
           compatible = "xlnx,zynq-usb-2.20.a", "chipidea,usb2";
           clocks = <&clkc 28>;
           dr_mode = "peripheral";
           interrupt-parent = <&intc>;
           interrupts = <0 21 4>;
           reg = <0xe0002000 0x1000>;
           usb-phy = <&usb_phy0>;
       };

   usb_phy0: phy0 {
      compatible = "ulpi-phy";
      #phy-cells = <0>;
      reg = <0xe0002000 0x1000>;
      view-port = <0x170>;
```

Performance

Peripheral mode	32.00 MB/sec	Tool: hdparm

Test Procedure

Tested with Mass storage and Ethernet gadget.

Mass Storage
Please refer above kernel configuration for enabling required modules for mass storage gadget.
After building the source code, copy the required modules found in the above given paths into sdcard

Steps for mounting the sdcard for accessing the compiled modules
zynq> mount /dev/mmcblk0p1 /mnt

Install the following modules
zynq> insmod /mnt/configfs.ko
zynq> insmod /mnt/libcomposite.ko
zynq> insmod /mnt/usb_f_mass_storage.ko

zynq> dd if=/dev/zero of=/tmp/mydev count=10 bs=1M
zynq> mount -t configfs none /sys/kernel/config
zynq> cd /sys/kernel/config/usb_gadget
zynq> mkdir g1
zynq> cd g1
zynq> echo "64" > bMaxPacketSize0
zynq> echo "0x200" > bcdUSB
zynq> echo "0x100" > bcdDevice
zynq> echo "0x03FD" > idVendor
zynq> echo "0x0500" > idProduct
zynq> mkdir functions/mass_storage.ms0
Number of LUNs=8
Mass Storage Function, version: 2009/09/11
LUN: removable file: (no medium)
zynq> mkdir configs/c1.1
zynq> echo /tmp/mydev > functions/mass_storage.ms0/lun.0/file
zynq> echo 1 > functions/mass_storage.ms0/lun.0/removable
zynq> ln -s functions/mass_storage.ms0 configs/c1.1/

Ethernet
Please refer above kernel configuration for enabling required modules for ethernet gadget.
After building the source code, copy the required modules found in the above given paths into sdcard

Steps for mounting the sdcard for accessing the compiled modules
zynq> mount /dev/mmcblk0p1 /mnt

Install the following modules
zynq> insmod /mnt/configfs.ko
zynq> insmod /mnt/libcomposite.ko
zynq> insmod /mnt/u_ether.ko
zynq> insmod /mnt/usb_f_rndis.ko

zynq> mount -t configfs none /sys/kernel/config
zynq> cd /sys/kernel/config/usb_gadget
zynq> mkdir g1
zynq> cd g1
zynq> echo "64" > bMaxPacketSize0
zynq> echo "0x200" > bcdUSB
zynq> echo "0x100" > bcdDevice
zynq> echo "0x03FD" > idVendor
zynq> echo "0x0500" > idProduct
zynq> mkdir functions/rndis.rn0
zynq> mkdir configs/c1.1
zynq> ln -s functions/rndis.rn0 configs/c1.1/
zynq> echo "ci_hdrc.0" > UDC
zynq> ifconfig usb0 10.10.70.1
zynq> ifconfig usb0 up

Run the standard network tests like ping, iperf,netperf...

OTG Mode

Kernel Configuration

Ensure the below config parameters are selected

```
<*> OTG support
<*> EHCI HCD (USB 2.0) support
<*> USB Mass Storage support
<*> ChipIdea Highspeed Dual Role Controller
<*> ChipIdea host controller
<*> ChipIdea device controller
    USB Physical Layer drivers --->
    <*> NOP USB Transceiver Driver
<*> USB Gadget Support
      <M> USB Gadget Drivers
      <M> USB functions configurable through configfs
      [*] Mass storage
```

All the loadable modules (.ko) for Peripheral/OTG configuration will be generated in below kernel source paths:
fs/configfs/configfs.ko
drivers/usb/gadget/libcomposite.ko
drivers/usb/gadget/function/usb_f_mass_storage.ko
drivers/usb/gadget/function/usb_f_rndis.ko

Devicetree

```
usb_0: usb@e0002000 {
     compatible = "xlnx,zynq-usb-2.20.a", "chipidea,usb2";
     clocks = <&clkc 28>;
     dr_mode = "otg";
     interrupt-parent = <&intc>;
     interrupts = <0 21 4>;
     reg = <0xe0002000 0x1000>;
     usb-phy = <&usb_phy0>;
 };
```

Xilinx Wiki
- Xilinx Phy Vi...
- Video_Mixer...
- Zynq EDAC ...
- Zynq Emacp...
- Zynq Pl353 ...
- Zynq QSPI ...
- Zynq SDIO ...
- Zynq Ultrasc...
- Zynq Ultrasc...
- Video Scene...
- Zynq Linux ...
- CCF SI5324 ...
- Common Cl...
- Cortexa53 E...
- CPU freque...

```
    #phy-cells = <0>;
    reg = <0xe0002000 0x1000>;
    view-port = <0x170>;
    drv-vbus;
}
```

Test Procedure

Using the correct cables is the key to OTG operation. Testing was done using two cables joined together to create an OTG cable. An OTG cable has a micro A connector on one end and a micro B connector on the other end. The micro A connector is the host side of the cable and the micro B connector is the device side by default.

Testing for OTG was done with 2 ZC706 boards connected together. An adapter with a Micro-A plug on one end and a Standard-A receptacle on the other end was used for testing. The adapter is connected to the board that defaults to being a host. A cable with a Micro-B plug on one end and a Standard-A plug on the end is connected to the board that defaults to being a device. The cable is then connected to the adapter with the Standard-A receptacle and Standard-A plug.

```
After booting linux, insert the gadget drivers on both the boards.

This step is necessary as OTG device should work as both host and device.
1. Do the above steps used for testing mass storage gadget

2. Connect Micro-A cable to USB interface of the board#1.
   This board will act as USB A-device.

3. Connect Micro-B cable to USB interface of board#2.
   This board will act as a USB B-device.

Now, both board#1 and board#2 are connected.

4. Make sure there is no hub in between. This makes the back-to-back connection
   between the two boards.
```

```
now acts as Host and enumerates board#1 as Mass storage device
```

Mainline Status

The Zynq USB driver is currently in sync with mainline kernel 4.9

Change Log

2016.3
Summary:

- None

2016.4
Summary:

- None

2017.1
Summary:

- None

2017.2
Summary:

- None

2017.3
Summary:

- None

2017.4
Summary:

- None

- Merged to 4.14 kernel

Commits:
818f1

2018.2
Summary:

- None

2018.3
Summary:

- None

2019.1
Summary:

- None

2019.2
Summary:

- None

Related Links

- Linux Device Drivers