USB on Embedded Linux Systems Deep Dive

Presented by Toradex
WITH YOU TODAY...

- Joined Toradex 2011
- Spearheaded Embedded Linux Adoption
- Introduced Upstream First Policy
- Top 10 U-Boot Contributor
- Top 10 Linux Kernel Arm SoC Contributor
- Industrial Embedded Linux Platform Torizon Fully Based on Mainline Technology
  - Mainline U-Boot with Distroboot
  - KMS/DRM Graphics with Etnaviv & Nouveau
  - OTA with OSTree
  - Docker resp. Podman
WHAT WE’LL COVER TODAY...

• Introduction to the USB Specification
• USB in Embedded Systems
• USB Recovery Mode
• USB in U-Boot
• USB in the Linux Kernel
• USB from Userspace
• USB Tooling
• USB Role Switching
• USB Debugging
• Live Demonstration
Introduction to the USB Specification

• Connectors
  • USB-A: USB 2.0 and 3.0 variants
  • USB-B: Fullsize, mini, micro and USB 3.0 variants
  • USB-C: One size fits all, right?

• USB transfer speed
  • Low-speed: up to 1.5 Mbps
    • Since USB 1.0
  • Full-speed: up to 12 Mbps
    • Since USB 1.1
  • High-speed: up to 480 Mbps
    • Since USB 2.0
  • SuperSpeed: up to 5 Gbps
    • Since USB 3.0
  •...
Introduction to the USB Specification (cont.)

- USB protocol
  - Device: Entity connected to the bus
  - Configuration: State of a device
    - Initialisation, standby, active
    - Bundles a bunch of interfaces
  - Interface: Logical device
    - Each interface encapsulates a single high-level function (e.g. webcam: video stream, audio stream, buttons)
    - One driver is needed for each interface!
    - Alternate settings: Each USB interface may have different parameter settings (e.g. for different bandwidth)
    - The initial state is always in the first setting (number 0)
    - Alternate settings often used for isochronous endpoints (endpoints use different amounts of reserved bandwidth)
Introduction to the USB Specification (cont.)

• Endpoint: Unidirectional communication pipe
  • Control endpoints
    • For configuration, get information, send commands, and retrieve status information
    • Simple, small data transfers
    • Every device has a control endpoint (endpoint 0)
    • USB protocol guarantees corresponding data transfers will always have enough (reserved) bandwidth
  • Interrupt endpoints
    • Transfer small amounts of data at a fixed rate
    • Guaranteed, reserved bandwidth
    • For devices requiring guaranteed response time, such as USB human interface devices (HID) e.g. mice and keyboards
    • Note: Different from hardware interrupts, really requires constant polling from the host
Introduction to the USB Specification (cont.)

• Endpoints: Unidirectional communication pipes (cont.)
  • Bulk endpoints
    • Large sporadic data transfers
    • Using all remaining available bandwidth
    • However, no guarantee on bandwidth or latency
    • Only guarantee that no data is lost
    • Typically used when there is no quality of service requirement (Network, printer, storage devices et al.)
  • Isochronous endpoints
    • Also for large amounts of data
    • Guaranteed speed (often but not necessarily as fast as possible).
    • No guarantee that all data makes it through
    • Used by real-time data transfers (typically for audio and video devices with quality of service requirements)
Introduction to the USB Specification (cont.)

- USB request blocks (URBs)
  - Communication between host and device done asynchronously using URBs
  - Similar to packets in network communication
  - Every endpoint can handle a queue of URBs
  - Every URB has a completion handler
  - A driver may allocate many URBs for a single endpoint or reuse same URB for different endpoints
  - See Documentation/usb/URB.txt in kernel sources

- URB scheduling interval
  - For interrupt and isochronous transfers
  - Low-speed and full-speed devices: The interval unit is frames (ms)
  - Hi-speed devices: The interval unit is microframes (1/8 ms)
USB in Embedded Systems

- Most modern SoCs feature at least one USB port often with accompanying PHY
- Dedicated differential signals
  - D+/D- for up to USB 2.0 low/full/high-speed
  - SSRX+/SSRX- and SSTX+/SSTX- for SuperSpeed beginning with USB 3.0
- Supporting signals
  - May be dedicated or realised by regular GPIOs
  - ID: usually low for host and not connected (pulled-up) for device
  - OverCurrent: device draws too much VBUS current (output from USB power switch chips)
- VBUS
  - Input in device role
  - May influence connection/suspend state
  - Often not 5 volt tolerant requiring a voltage divider
- VBUS enable
  - Output in host role (enable for USB power switch chips)
USB in Embedded Systems (cont.)

- External hub and/or PHY chips
- Designed-in chips
  - USB-to-Ethernet bridges
  - USB-to-serial adapters
  - ...
- USB-C
  - Special companion chips taking care of signalling details
  - Either compatible to legacy signalling (e.g. ID and VBUS)
  - Or using out-of-band signalling (e.g. I2C or SPI) mandating special driver
  - May further take care of power delivery requirements
  - Blog posts and webinars from Toradex about the topic (see references)
USB Recovery Mode

- Most modern SoCs allow multiple so-called “boot modes”
- Selected by either strapping pins or fusing (done during production)
- Functionality of the Boot ROM aka initial program loader (IPL)
- Once initial “stage” is loaded/executed other mechanisms may be used to load/execute later “stages”
- NXP i.MX 6/7/8 and Vybrid support USB serial download protocol (SDP)
  - Basically former serial aka UART download protocol encapsulated in USB
  - Two implementations thereof exist:
    - imx_loader aka imx_usb
    - mfgtools 3.0 aka universal update utility (uuu)
USB Recovery Mode (cont.)

- TI AM62x Sitara support USB device firmware upgrade (DFU)
  - Official USB device class
  - Relatively low transfer speed for large files
  - Imposed utilization of only EP0 for transfer
  - Host side implementation: dfu-util
- For convenience further configuration/scripting may be required
  - What USB vendor/product ID to act upon
  - What binaries to use for what “stages”
- Toradex easy installer uses those mechanisms to allow loading full fledged Linux/Qt based installer
USB in U-Boot

- Device side aka gadget
  - Manually start one functionality at a time: DFU, Fastboot, UMS
- Device Firmware Upgrade (DFU)
  - CONFIG_DFU and CONFIG_CMD_DFU plus at least one backend like CONFIG_DFU_RAM
  - Environment variable
    
    ```
    dfu_alt_info_ram=tispl.bin ram 0x80080000 0x200000;u-boot.img ram 0x81000000 0x400000;loadaddr ram 0x88200000 0x80000;scriptaddr ram 0x90280000 0x80000;ramdisk_addr_r ram 0x90300000 0x800000
    ```
  - dfu <USB_controller> [<interface> <dev>] [<timeout>]
- Android Fastboot
  - CONFIG_USB_FUNCTION_FASTBOOT depends on CONFIG_USB_GADGET_DOWNLOAD, CONFIG_USB_GADGET_VENDOR_NUM, CONFIG_USB_GADGET_PRODUCT_NUM and CONFIG_USB_GADGET_MANUFACTURER
  - Requires large memory buffer via CONFIG_FASTBOOT_BUF_ADDR and CONFIG_FASTBOOT_BUF_SIZE
  - Further configuration like partition aliases, raw partition descriptors and variable overrides possible
  - fastboot usb 0
USB in U-Boot (cont.)

- Device side aka gadget (cont.)
  - USB mass storage class (ums): shares a U-Boot block device via USB
    - CONFIG_CMD_USB_MASS_STORAGE depends on CONFIG_USB_USB_GADGET and CONFIG_BLK
    - ums <dev> [<interface>] <devnum:[partnum]>
    - Where <dev> is the USB gadget device number (usually zero unless multiple device controller instances)
    - Further arguments are specific to the block device

- Host side
  - CONFIG_CMD_USB depends on a low-level host controller driver
  - USB is NOT automatically started during start-up due to potential interference with OS e.g. Linux kernel boot
  - Therefore requires manually starting it with "usb start" and stopping with "usb stop"
  - Enumeration is also rather slow due to timeouts
  - "usb tree" shows all USB devices in a tree like display
  - Supports keyboards, storage as well as USB-to-Ethernet adapters (with their resp. configs)
USB in the Linux Kernel

- USB core: Implements the USB bus specification
  - Architecture independent kernel subsystem
- USB host controller drivers
  - Architecture and platform dependent
  - Different driver depending on USB host controller hardware (OHCI/UHCI, EHCI, xHCI et al.)
- USB device drivers
  - Platform independent
  - Drivers for specific peripheral on the USB bus
- USB device controller (UDC) drivers
  - Architecture and platform dependent
  - Different driver depending on USB device controller hardware
- USB gadget drivers
  - Platform independent
  - Different driver depending on peripheral functionality to provide (Ethernet, serial, storage et al.)
USB from Userspace

- `/proc/bus/usb/devices`
- `usbutils`: Utilities for inspecting devices connected to a USB bus
  - `lsusb`: List USB devices, tree like view with `-t` resp. `--tree`
  - `usb-devices`: Print USB device details
  - `usbhid-dump`: Dump USB HID device report descriptors and streams
- `usbview`: Display information on USB devices
  - GTK+ 3.x graphical application
USB from Userspace (cont.)

- libusb: A cross-platform user library to access USB devices
  - C library providing generic access to USB devices
  - Intended to be used by developers to facilitate the production of applications that communicate with USB hardware
  - Portable: Using a single cross-platform API on Android, Linux, macOS, Windows, etc.
  - User-mode: No special privilege or elevation is required for the application to communicate with a device
- uhubctl: USB hub per-port power control
  - Utility to control USB power per-port on smart USB hubs
  - Smart hub defined as one that implements per-port power switching
USB Tooling

- FTDI USB-to-serial aka UART adapters
- USB analyzer
  - BEAGLE
  - Cynthion (formerly Luna)
    - A multi-tool for building, analyzing, and hacking USB devices
    - Completely open source hardware and software
- USB CAN analyzer
- USB logic analyzer
  - DreamSourceLab DSLogic
  - Saleae Logic
- USB oscilloscope
  - DreamSourceLab DSCope
USB Role Switching

- Device/host resp. on-the-go (OTG) or dual role device (DRD) switching
- Fixed in device tree
  - dr_mode property
    - May be host, otg (usually defaults to peripheral) or peripheral
- USB GPIO extcon device driver (e.g. as used on Colibri iMX6/7)
  - Documentation/devicetree/bindings/extcon/extcon-usb-gpio.txt
  - Virtual device used to generate USB cable states from USB ID pin connected to a GPIO pin (obsolete)
  - CONFIG_EXTCON_USB_GPIO
    - drivers/extcon/extcon-usb-gpio.c
    - compatible = "linux,extcon-usb-gpio"
    - id-gpio and/or vbus-gpio
    - Reference it in actual USB node
    - Here zero means no VBUS detection capability, ID pin aka device/host only
USB Role Switching (cont.)

- USB connector subsystem (e.g. as used on Verdin iMX8M Plus)
- Documentation/devicetree/bindings/connector/usb-connector.yaml
- USB GPIO based connection detection driver
  - CONFIG_USB_CONN_GPIO
  - drivers/usb/common/usb-conn-gpio.c
- Simple GPIO VBUS sensing driver for B peripheral devices
  - CONFIG_USB_GPIO_VBUS
  - drivers/usb/phy/phy-gpio-vbus-usb.c
  - compatible = "gpio-usb-b-connector", "usb-b-connector";
  - label = "Type-C";
  - type: mini/micro in case of non-fullsize connector
  - self-powered and more optional power related properties
  - id-gpio and/or vbus-gpio
  - vbus-supply

```c
/* Verdin USB_1 */
&usb3_0 {
    fsl,disable-port-power-control;
    fsl,over-current-active-low;
    pinctrl-names = "default";
    pinctrl-0 = &pinctrl_usb_1_oc_n;
};

&usb_dwc3_0 {
    /* dual role only, not full featured OTG */
    adp-disable;
    dr_mode = "otg";
    hnp-disable;
    maximum-speed = "high-speed";
    role-switch-default-mode = "peripheral";
    srp-disable;
    usb-role-switch;

    connector {
        compatible = "gpio-usb-b-connector",
                    "usb-b-connector";
        id-gpios = <&gpio2 10
                   GPIO_ACTIVE_HIGH>;
        label = "Type-C";
        pinctrl-names = "default";
        pinctrl-0 = &pinctrl_usb_1_id;
        self-powered;
        type = "micro";
    vbus-supply = &reg_usb1_vbus;
    }
};
```
USB Device Functionality

- USB gadget functions configurable through configfs

--- USB Gadget Support
- Debugging messages (DEVELOPMENT)
- Debugging information files (DEVELOPMENT)
- Debugging information files in debugfs (DEVELOPMENT)
- Maximum VBUS Power usage (2-500 mA)
- Number of storage pipeline buffers
- Serial gadget console support
- USB Peripheral Controller

USB Gadget functions configurable through configfs
- Generic serial bulk in/out
- Abstract Control Model (CDC ACM)
- Object Exchange Model (CDC OBEX)
- Network Control Model (CDC NCM)
- Ethernet Control Model (CDC ECM)
- Ethernet Control Model (CDC ECM) subset
- RNDIS
- Ethernet Emulation Model (EEM)
- Mass storage
- Loopback and sourcesink function (for testing)
- Function filesystem (FunctionFS)
- Audio Class 1.0
- Audio Class 1.0 (legacy implementation)
- Audio Class 2.0
- MIDI function
- HID function
- USB Webcam function
- Printer function

USB Gadget precomposed configurations
configfs

- Userspace-driven kernel object configuration
- Ram-based filesystem that provides the converse of sysfs's functionality
- Where sysfs is a filesystem-based view of kernel objects, configfs is a filesystem which allows userspace instantiation of kernel objects, or config_items
- Two types of configfs attributes
  - Normal attributes: Small ASCII text files
  - Binary attributes
- USB Gadget ConfigFS: Interface that allows definition of arbitrary functions and configurations to define an application specific USB composite device from userspace
  - Create gadget device and bind to a UDC driver from userspace
configfs (cont.)

- First needs to be mounted
- If USB gadget configfs support enabled usb_gadget subdirectory present
- By creating the g1 directory instantiated new gadget device filled by template
- Write our vendor/product IDs
- Instantiate English language strings

```bash
~# mount -t configfs none /sys/kernel/config
~# cd /sys/kernel/config/
/sys/kernel/config# ls
pci_ep  usb_gadget
~# cd usb_gadget/
/sys/kernel/config/usb_gadget# mkdir g1
/sys/kernel/config/usb_gadget# cd g1/
/sys/kernel/config/usb_gadget/g1# ls
UDC  bDeviceSubClass  bcdUSB  idProduct  os_desc
bDeviceClass  bMaxPacketSize0  configs  idVendor  strings
bDeviceProtocol  bcdDevice  functions  max_speed  webusb
/sys/kernel/config/usb_gadget/g1# echo "0x1b67" > idVendor
/sys/kernel/config/usb_gadget/g1# echo "0x4058" > idProduct
/sys/kernel/config/usb_gadget/g1# mkdir strings/0x409
/sys/kernel/config/usb_gadget/g1# ls strings/0x409/
manufactured  product  serialnumber
```
configfs (cont.)

- Write our manufacturer, product and serialnumber descriptor strings
- Create function instances
- Note: Multiple function instances of the same type must have a unique extension
- Create configuration instance
- Create English language strings and write description for this device configuration
- Bind each of our function instances to this configuration
- Check which UDC instances available
- Attach created gadget device to desired UDC

```bash
# echo "Toradex" > strings/0x409/manufacturer
# echo "verdin-imx8mp" > strings/0x409/product
# echo "07106916" > strings/0x409/serialnumber
/sys/kernel/config/usb_gadget/gl# mkdir functions/ncm.usb0
/sys/kernel/config/usb_gadget/gl# mkdir configs/c.1
/sys/kernel/config/usb_gadget/gl# ls configs/c.1/
MaxPower bmAttributes strings
# mkdir configs/c.1/strings/0x409/
/sys/kernel/config/usb_gadget/gl# ls configs/c.1/strings/0x409/configuration
# echo "WINNCM" > configs/c.1/strings/0x409/configuration
# ln -s functions/ncm.usb0 configs/c.1/
/sys/kernel/config/usb_gadget/gl# ls /sys/class/udc/38100000.usb
/sys/kernel/config/usb_gadget/gl# echo "38100000.usb" > UDC
```
configfs (cont.)

- **libusbgx**
- Library providing C API to USB gadget configfs
- Basically programmatic way to go about creation and removal of gadgets
USB Host Functionality

- USB device class drivers

*** USB Device Class drivers ***

< M> USB Modem (CDC ACM) support
< > USB Printer support
< > USB Wireless Device Management support
< > USB Test and Measurement Class support

*** NOTE: USB_STORAGE depends on SCSI but BLK_DEV_SD may ***
*** also be needed; see USB_STORAGE Help for more info ***

< > USB Mass Storage support

[ ] USB Mass Storage verbose debug
< > Realtek Card Reader support
< > Datafab Compact Flash Reader support
< > Freecom USB/ATAPI Bridge support
< > ISO-200 USB/ATA Bridge support
< > USBAT/USBAT02-based storage support
< > SanDisk SDDR-09 (and other SmartMedia, including DFCM) support
< > SanDisk SDDR-55 SmartMedia support
< > Lexar Jumpshot Compact Flash Reader
< > Olympus MAUSB-10/Fuji DPC-R1 support
< > Support OneTouch Button on Maxtor Hard Drives
< > Support for Rio Karma music player
< > SAT emulation on Cypress USB/ATA Bridge with ATACB
< > USB ENE card reader support
< > USB Attached SCSI

*** USB Imaging devices ***

< > USB Mustek NDC800 Digital Camera support
< > Microtek X6 USB scanner support
< > USB/IP support
Debugging USB

- **usbmon**
  - Linux kernel facility used to collect traces of I/O on the USB bus
  - May be compiled as built-in or Linux kernel module requiring separate loading
  - Analogous to packet socket used by network monitoring tools such as tcpdump
  - As a matter of fact tcpdump comes with support for usbmon: tcpdump --list-interfaces

- **Virtual USB Analyzer**
  - Tool for visualizing logs of USB packets from hardware or software USB sniffer tools
  - Developed at VMware
  - Python 2.7 PyGTK based
  - Probably abandoned rather obsolete project

- **Wireshark**
  - Has built-in USB analysis functionality
  - But how to do that on an Embedded device?
  - `ssh <target> "tcpcump -i usbmon2 -U -w -" | flatpak run --filesystem=host --file-forwarding=host --share=network org.wireshark.Wireshark -k -i -`
### Debugging USB (cont.)

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0.006411</td>
<td>host</td>
<td>2.2.0</td>
<td>USB</td>
<td>95</td>
<td>GET DESCRIPTOR Response CONFIGURATION</td>
</tr>
<tr>
<td>13</td>
<td>0.006471</td>
<td>host</td>
<td>2.1.0</td>
<td>USB</td>
<td>64</td>
<td>GET DESCRIPTOR Request DEVICE</td>
</tr>
<tr>
<td>14</td>
<td>0.006483</td>
<td>host</td>
<td>2.1.0</td>
<td>USB</td>
<td>82</td>
<td>GET DESCRIPTOR Response DEVICE</td>
</tr>
<tr>
<td>15</td>
<td>0.006492</td>
<td>host</td>
<td>2.1.0</td>
<td>USB</td>
<td>64</td>
<td>GET DESCRIPTOR Request CONFIGURATION</td>
</tr>
<tr>
<td>16</td>
<td>0.006498</td>
<td>host</td>
<td>2.1.0</td>
<td>USB</td>
<td>73</td>
<td>GET DESCRIPTOR Response CONFIGURATION</td>
</tr>
<tr>
<td>17</td>
<td>0.006507</td>
<td>host</td>
<td>2.1.0</td>
<td>USB</td>
<td>64</td>
<td>GET DESCRIPTOR Request CONFIGURATION</td>
</tr>
<tr>
<td>18</td>
<td>0.006515</td>
<td>host</td>
<td>2.1.0</td>
<td>USB</td>
<td>95</td>
<td>GET DESCRIPTOR Response CONFIGURATION</td>
</tr>
<tr>
<td>19</td>
<td>0.036492</td>
<td>host</td>
<td>2.3.2</td>
<td>USBMS</td>
<td>95</td>
<td>SCSI: Unit Ready LUN: 0x00</td>
</tr>
<tr>
<td>20</td>
<td>0.036529</td>
<td>host</td>
<td>2.3.2</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>21</td>
<td>0.036540</td>
<td>host</td>
<td>2.3.1</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK in</td>
</tr>
<tr>
<td>22</td>
<td>0.036562</td>
<td>host</td>
<td>2.3.1</td>
<td>USBMS</td>
<td>77</td>
<td>SCSI: Response LUN: 0x00 (Test Unit Ready) (Good)</td>
</tr>
<tr>
<td>23</td>
<td>0.036549</td>
<td>host</td>
<td>2.3.2</td>
<td>USB</td>
<td>95</td>
<td>SCSI: Unit Ready LUN: 0x00</td>
</tr>
<tr>
<td>24</td>
<td>0.052556</td>
<td>host</td>
<td>2.3.2</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>25</td>
<td>0.052568</td>
<td>host</td>
<td>2.3.1</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK in</td>
</tr>
<tr>
<td>26</td>
<td>0.052586</td>
<td>host</td>
<td>2.3.1</td>
<td>USBMS</td>
<td>77</td>
<td>SCSI: Response LUN: 0x00 (Test Unit Ready) (Good)</td>
</tr>
<tr>
<td>27</td>
<td>0.068488</td>
<td>host</td>
<td>2.3.2</td>
<td>USB</td>
<td>95</td>
<td>SCSI: Test Unit Ready LUN: 0x00</td>
</tr>
<tr>
<td>28</td>
<td>0.068517</td>
<td>host</td>
<td>2.3.2</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>29</td>
<td>0.068528</td>
<td>host</td>
<td>2.3.1</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK In</td>
</tr>
<tr>
<td>30</td>
<td>0.068552</td>
<td>host</td>
<td>2.3.1</td>
<td>USBMS</td>
<td>77</td>
<td>SCSI: Response LUN: 0x00 (Test Unit Ready) (Good)</td>
</tr>
</tbody>
</table>

Frame 24: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
USB URB

- [Source: 2.3.2]
- [Destination: host]
- URB id: 0xffff000000a00000
- URB type: URB_COMPLETE ('c')
- URB transfer type: URB_BULK (0x03)
- Endpoint: 0x82, Direction: OUT
- Device: 3
- URB bus id: 2
- Device setup request: not relevant ('-')
- Data: not present ('-')
Live Demonstration

- Nothing fancy, just a regular Toradex board in the wild running upstream Linux (:-p)
References

- USB Specification
  https://www.usb.org/documents

- USB-C
  https://www.toradex.com/blog/add-usb-c-to-your-next-carrier-board-design-1
  https://www.toradex.com/blog/add-usb-c-to-your-next-carrier-board-design-2
  https://www.toradex.com/webinars/add-usb-c-to-your-next-carrier-board-design

- imx_loader
  https://github.com/boundarydevices/imx_usb_loader

- uuu
  https://github.com/nxp-imx/mfgtools

- dfu-util
  https://dfu-util.sourceforge.net

- Toradex Easy Installer
  https://www.toradex.com/tools-libraries/toradex-easy-installer

- usbview
  http://www.kroah.com/linux-usb
References (cont.)

- libusb
  https://libusb.info

- uhubctl
  https://github.com/mvp/uhubctl

- Virtual USB Analyzer
  https://vusb-analyzer.sourceforge.net

- Wireshark
  https://wiki.wireshark.org/CaptureSetup/USB

- usbmon
  https://docs.kernel.org/usb/usbmon.html
THANK YOU
FOR YOUR INTEREST

community.toradex.com | labs.toradex.com