EMBEDDED DEVICE DRIVERS

Linux Device Drivers on Beaglebone Black

Linux device model

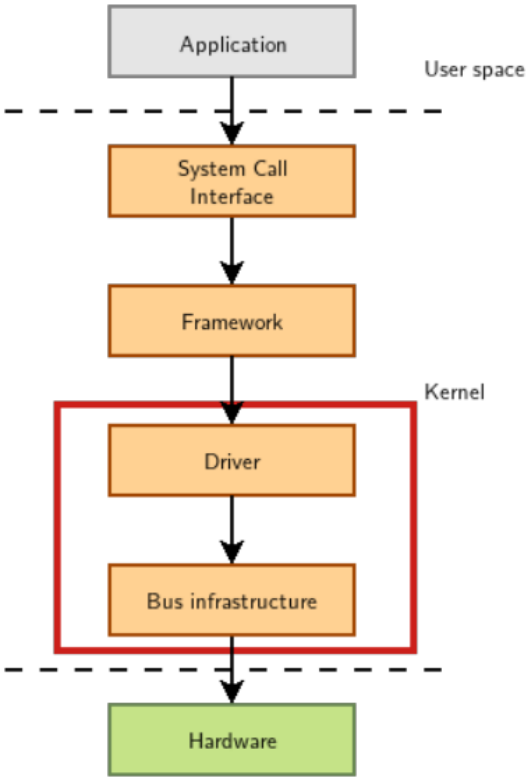
- Linux kernel needs to handle various device connect scenarios
  - The same device needs to use the same driver
  - On multiple CPU architectures
  - But the controllers may be different

  - A single driver
  - Needs to support multiple devices
  - Of the same kind

- This necessitates a separation
  - Controller drivers
    - Control behavior of controllers for complex protocol busses
  - Device drivers
    - Control behavior of specific device instantiations on the bus

Linux device model illustrated

- In Linux, a (device) driver talks to
  - A framework
    - That allows the driver to expose
    - Hardware in a generic manner
  - A bus infrastructure
    - To detect and communicate
    - With the actual hardware



Device model data structures

- The device model is organized around:
  - ***struct bus_type***
    - Represents a bus (USB, I2C, SPI, etc.)
  - ***struct device_driver***
    - Represents a driver capable of handling
      - Certain devices on a certain bus
  - ***struct device***
    - Represents one device connected to a bus
- The kernel uses inheritance
  - To create more specialized versions
    - For each bus subsystem

USB: What?

- Universal Serial Bus
  - Easy connection of multiple peripheral devices
  - To ports (in a tree hierarchy)
  - With auto-config, hotplug (attach/detach)

- Allows
  - Data exchange
  - Power delivery
    - Upto 500mA per controller
      - Endpoint device gets up to 100mA
        - Can get up to 500mA (if feasible) on ‘negotiation’

- Array of compatible devices
  - Scanners, storage, cameras, network cards, mice, keyboards, printers, serial adapters, hubs, joysticks, wireless devices, …

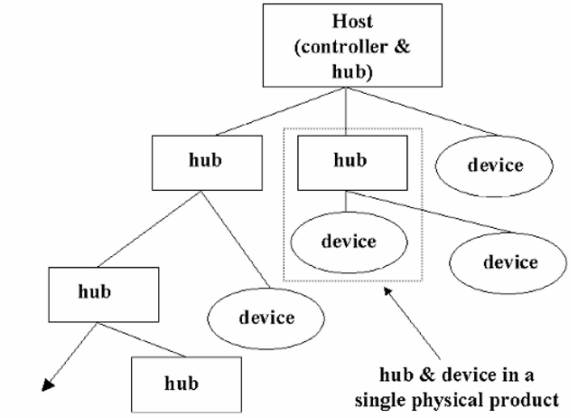
USB: Bus history

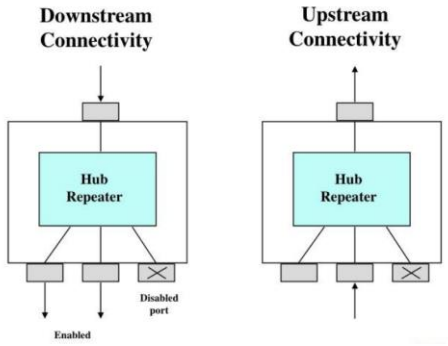
- Version 1.0 (1996), 1.1 (1998)
  - Low-speed (LS) 1.5Mbits/s, Full-speed (FS) 12 Mbits/s
- Version 2.0 (1999-2001)
  - High-speed (HS) 480 Mbits/s
- Version 3.0 (2008)
  - SuperSpeed (SS) USB 5Gbps
- Version 3.1 (2013)
  - SuperSpeed+ (SS+) USB 10Gbps
- Version 3.2 (2017)
  - SuperSpeed+ 2-lane USB 20Gbps
- Version USB4 (2019)
  - USB4 40 Gbps (2-lane)
- Version USB4 v2.0 (2022)
  - USB4 80-120Gbps

USB: Topology (physical)

- Star-tree topology rather than a true bus
  - Host controller as the central hub
- Hubs fan out to multiple devices

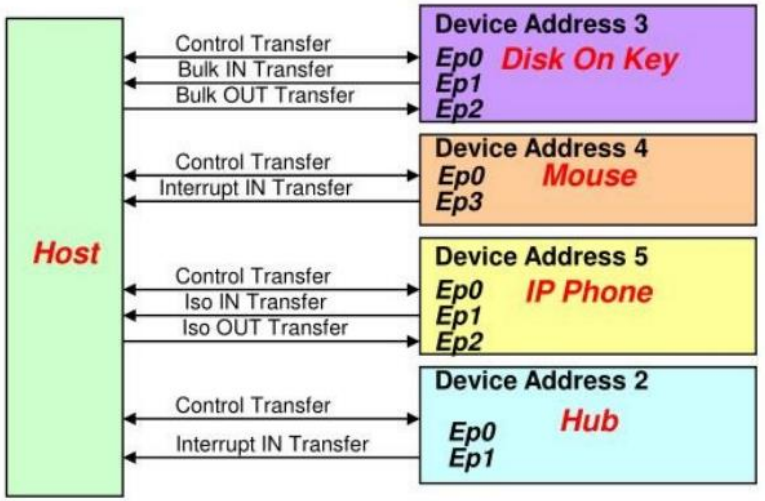
- Directions:
  - Downstream (host), Upstream (endpoints)
  - Both (hubs)





USB: Transfer types

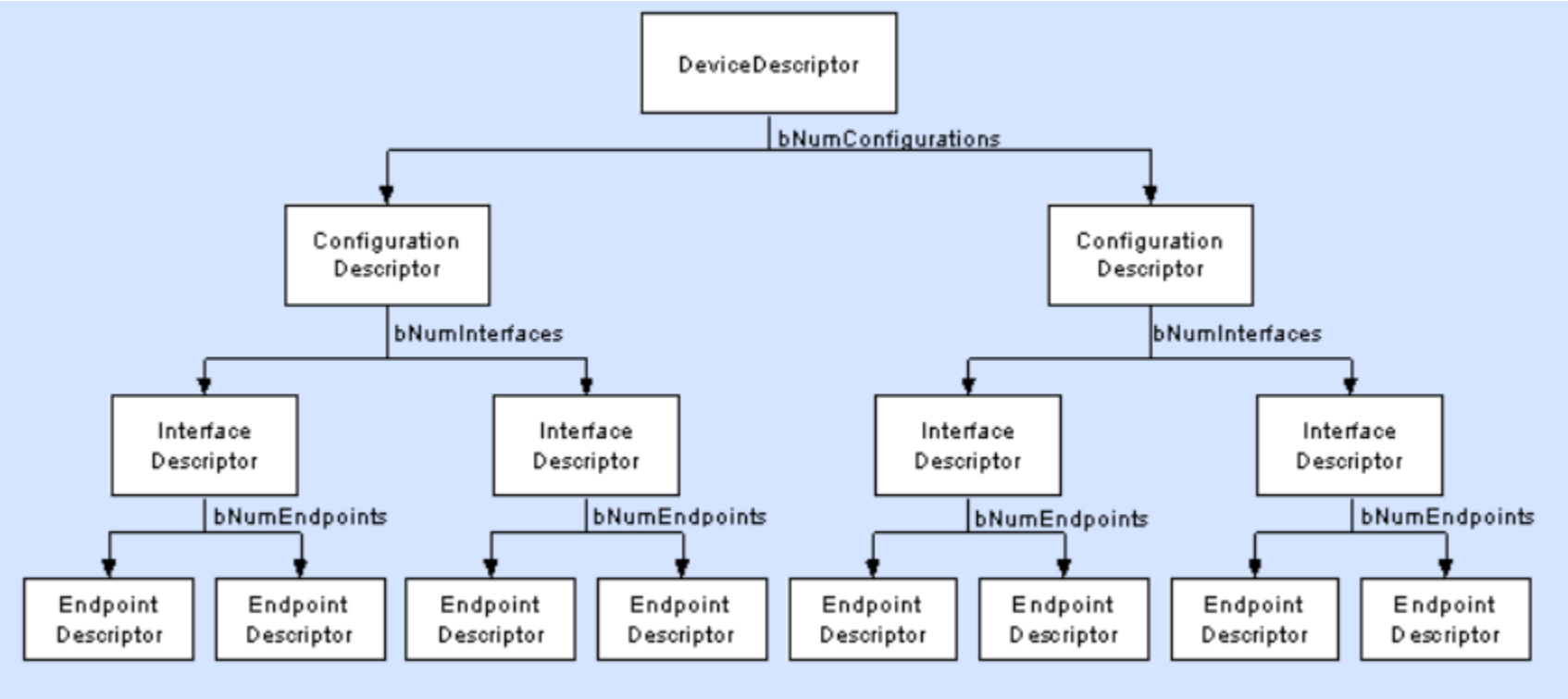
- Control transfers - bidirectional
  - Short commands to configure
  - Used to obtain device state
- Bulk transfers - unidirectional
  - Send information using full bandwidth
    - Reliable – printers, network cards
- Interrupt transfers - unidirectional
  - Sent in response to periodic polling
  - Interruptible, host may retry
    - Keyboard, mouse
- Isochronous transfers - unidirectional
  - Take up full bandwidth
    - Unreliable – cameras, audio



USB: Descriptors in devices

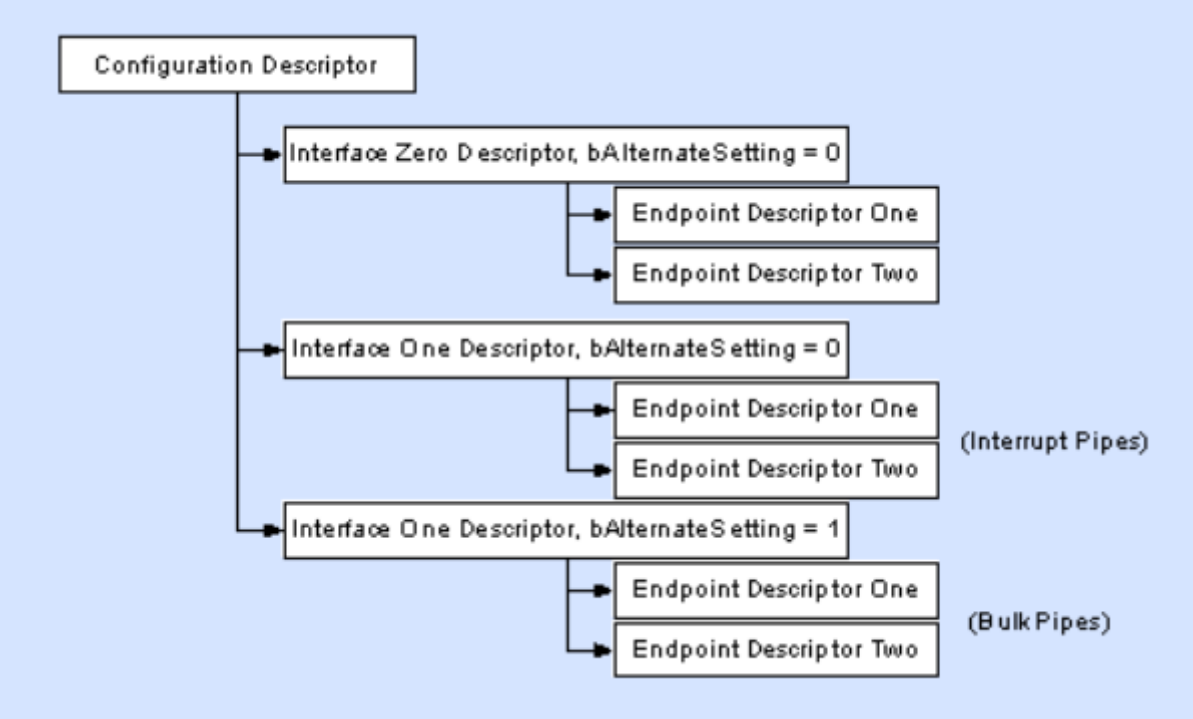
- Device Descriptor (1 per device)
  - USB revision, product and vendor IDs
  - Gives number of possible configurations
- Configuration Descriptor (1/more per device)
  - Gives device power, whether self / bus powered, device transfer speed
  - Gives number of interfaces
  - Only 1 configuration enabled at a time
- Interface Descriptor (1/more per device
  - Functional grouping of endpoints
  - Have a ***bInterfaceNumber*** and ***bAlternateSettings***
- Endpoint Descriptor
  - Specifies transfer type, direction, polling interval, max packet size
  - Endpoint 0 is always control – no descriptor

USB: Descriptor hierarchy



USB: Multi-configuration device

- 3 interface descriptors
  - #1 simple device
  - #2 interrupt pipe transfer
  - #3 bulk transfer pipe



USB: Device classes

Class	Usage	Description	Examples
00	Device	-	Interface dependent
01	Interface	Audio	Speaker, mic, sound card, MIDI
02	Both	Comms, CDC	UART RS-232, Wifi adapters
03	Interface	HID	Keyboard, mouse, joystick
05	Interface	PID	Force feedback joystick
06	Interface	Media	Scanner, camera
07	Interface	Printer	Laser, inkjet, CNC
08	Interface	Mass storage	Flash drive, card reader
09	Device	Hub	Hi-speed hub
…	…	…	…
0E	Interface	Video	Webcam
…	…	…	…
E0	Interface	Wireless controller	Bluetooth adapter, M$ RNDIS
…	…	…	…

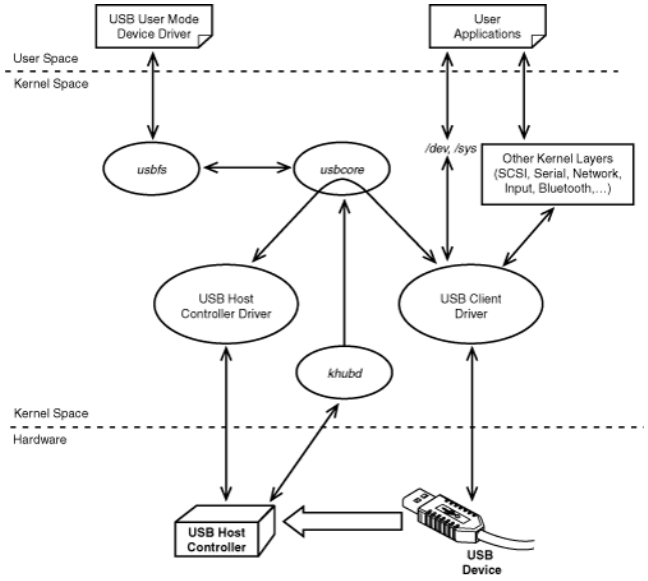
USB: Pros and cons

- Pros:
  - Extremely popular, self configuring devices
  - Standardized connectors
  - Hot-swappable
  - Defines various protocols for error recovery
  - Needs minimal operator actions and driver installs

- Cons:
  - Limited cable lengths
  - Data transfer rates slower than PCIe / SATA / NVMe
  - Strict tree topology – all transactions through host
  - No concept of ‘broadcast’ from host to all devices

LKM: USB subsystem in kernel

- Linux kernel handles USB stack in 3 layers
  - USB Host Controller Driver (OHCI, UHCI, EHCI)
  - USB Core
    - APIs for both controller and device drivers
    - A library of common routines that both can use
  - USB Client (Device) Drivers
    - Control the actual hardware
      - By using the USB Core APIs



LKM: USB driver exercise #1

- Refer ***mod14*** directory
  - The file ***mod14-1.c*** contains code for simple probing
    - We set up the ***usb_device_id*** table
      - Using definitions from ***"my_usb.h"***
    - We set up the struct ***usb_driver***
      - With appropriate probing and disconnect functions
    - We register the driver in the init call
      - Which calls our probing function
    - We deregister the driver in the exit call
      - Which cleans up
    - Also notice the disconnect is called when the device gets detached
  - Compile and load the module on BBB
    - Unload the system modules – before loading our driver
      # rmmod uas
      # rmmod usb_storage
    - Insert / detach the USB stick as per contents of ***my_usb.h***
      - Observe **dmesg** output

LKM: USB driver exercise #2

- Refer ***mod14*** directory
  - The file ***mod14-2.c*** contains code for simple probing
    - We now get more details about the USB device
      - Attached to the port
      - In the probing function
        - Device, configuration, interface, and endpoint descriptors
  - Compile and load the module on BBB
    - Unload the system modules – before loading our driver

      # rmmod uas

      # rmmod usb-storage
    - Insert / detach the USB stick as per contents of ***my_usb.h***
      - Observe **dmesg** output

THANK YOU!