Useful USB Gadgets on Linux

February, 2012

Gary Bisson
Adeneo Embedded

Embedded Linux Conference 2012

Agenda

- Introduction to USB
- USB Gadget API
- Existing Gadgets
- Design your own Gadget
- Demo
- Conclusion

Who am I?

- Software engineer at Adeneo Embedded (Bellevue, WA)
  - Linux, Android
  - Main activities:
    - BSP adaptation
    - Driver development
    - System integration

Context and objectives

- General knowledge of the API
  - Focused on USB not general driver development
  - Nothing on the host side
- Case study
  - Using a generic embedded device, see how we can create a USB Gadget
- Show potential
  - How to fulfill every need

Universal Serial Bus

- Industry standard developed in the mid-1990s
- Defines the cables, connectors and protocols used for connection, communication and power supply between computers and electronic devices
- 2 billion USB devices were sold each year (as of 2008)

Universal Serial Bus

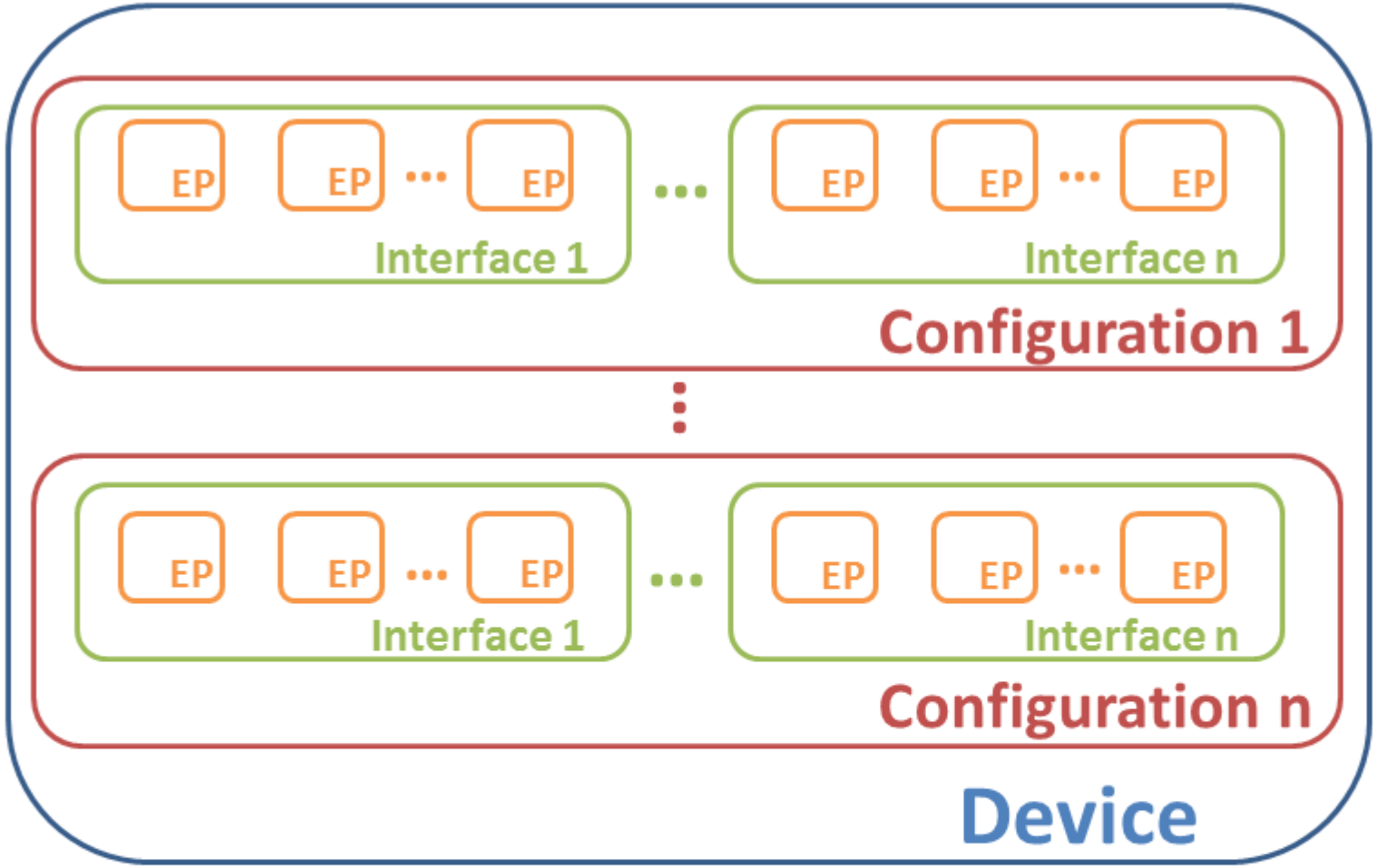
- Benefits:
  - Replace lots of old buses
  - Automatic configuration
  - Multiple speeds
  - Reliable
- Limits:
  - Distance
  - Peer-To-Peer
  - Broadcasting

Universal Serial Bus

- Architecture:
  - Master-Slave protocol
  - Up to 127 devices addressable
  - Can be hot-plugged
  - Identification to the host
  - Supports high speeds
  - Multifunction device possibility

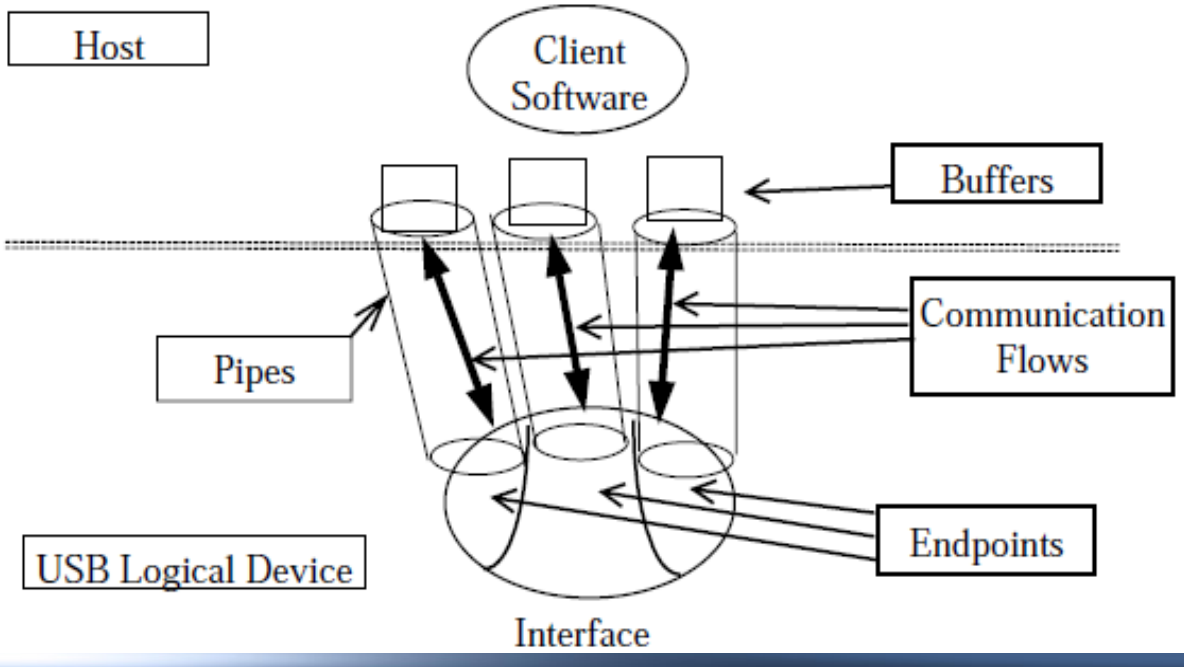
Universal Serial Bus

- Description:



Universal Serial Bus

- Endpoints
  - Source and Sink of data
  - Uniquely identifiable
  - Unique direction (except setup)

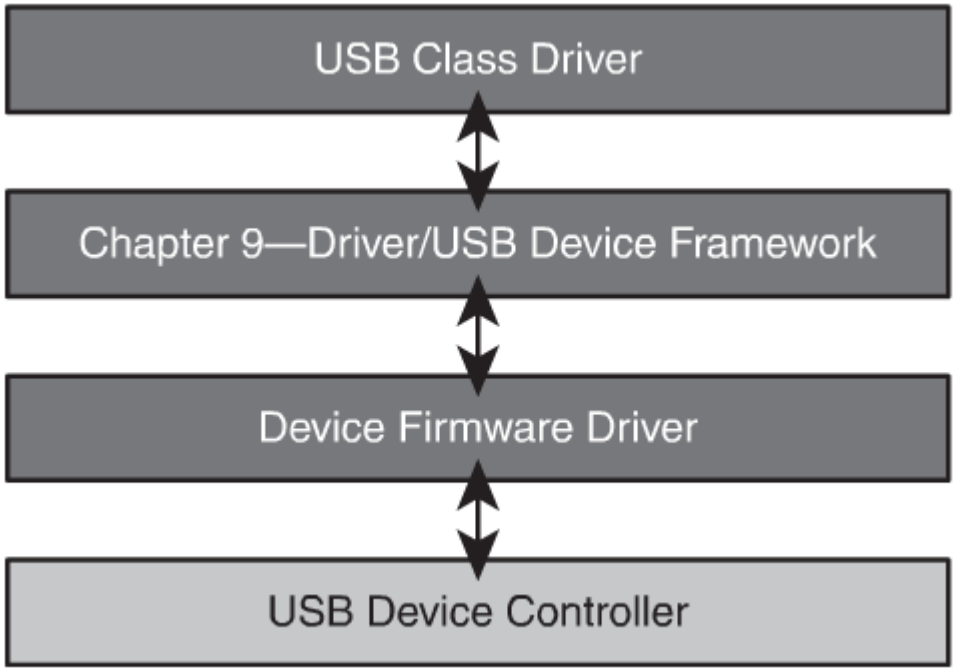


Universal Serial Bus

- 4 transfer types:
  - Control
    - Configuration and control information
  - Interrupt
    - Small quantities time-sensitive data
  - Bulk
    - Large quantities time-insensitive data
  - Isochronous
    - Real-time data at predictable bit rates

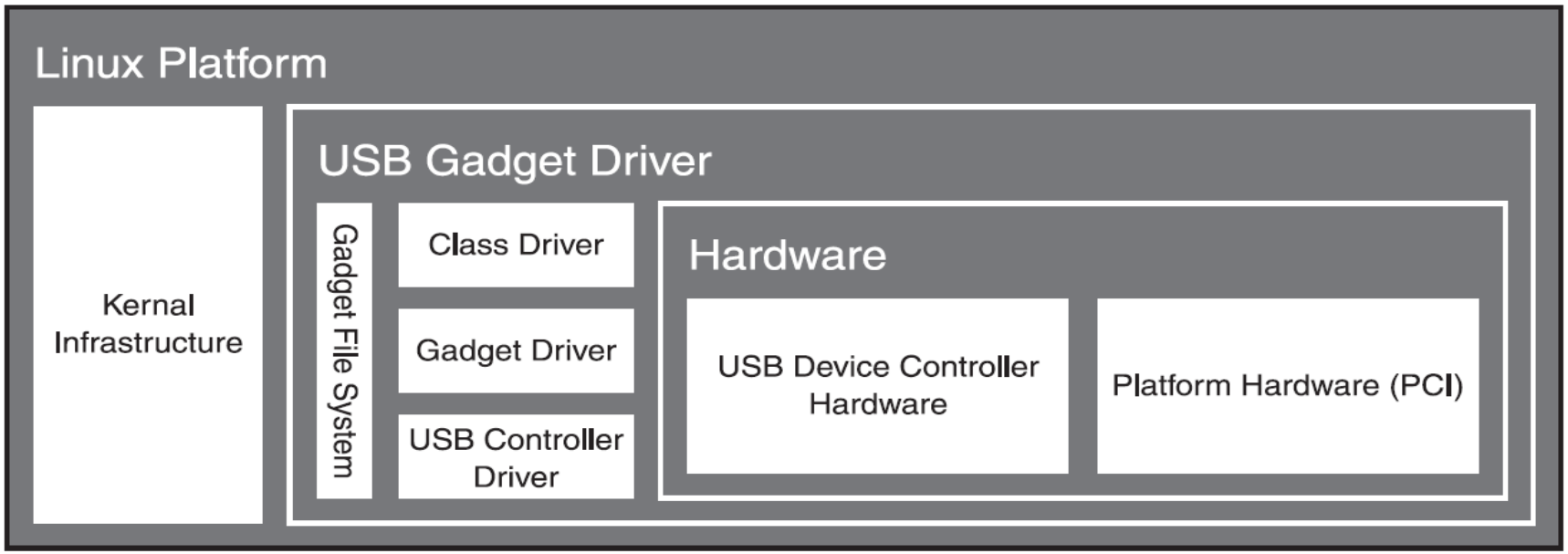
Typical Device Driver

- Device Firmware Driver
  - Hardware specific routines
  - USB interrupts/events
- Chapter 9
  - Enumeration process
  - Transfer data to upper layer
- USB Class Driver
  - Defines the behavior
  - Provides configuration



Gadget API

- Provides essential infrastructure
- Similar to Chapter 9 in typical USB device software
- Handles USB protocol specific requirements
- Flexible enough to expose more complex USB device capabilities



Gadget API vs. Linux-USB API

- Similarities
  - Share common definitions for the standard USB messages, structures and constants
  - Use queues of request objects to package I/O buffers
  - Both APIs bind and unbind drivers to devices
- Differences
  - Control transfers
  - Configuration management

=> Thanks to similarities, Gadget API supports OTG

Gadget API

Lower boundary:

- handling setup requests (ep0 protocol responses) possibly including class-specific functionality
- returning configuration and string descriptors
- (re)setting configurations and interface altsettings, including enabling and configuring endpoints
- handling life cycle events, such as managing bindings to hardware, USB suspend/resume, remote wakeup, and disconnection from the USB host
- managing IN and OUT transfers on all currently enabled endpoints

Gadget API

Upper layer:

- user mode code, using generic (gadgetfs) or application specific files in /dev
- networking subsystem (for network gadgets, like the CDC Ethernet Model gadget driver)
- data capture drivers, perhaps video4Linux or a scanner driver; or test and measurement hardware
- input subsystem (for HID gadgets)
- sound subsystem (for audio gadgets)
- file system (for PTP gadgets)
- block i/o subsystem (for usb-storage gadgets)

Gadget API – Main structures

`struct usb_gadget` – represents a gadget device

- `usb_gadget_ops` – contains callbacks for hardware operations

`struct usb_ep` – device hardware management

- `usb_ep_ops` – contains callbacks for endpoints operations

`struct usb_gadget_driver` – device functions management (bind, unbind, suspend etc...)

`struct usb_request` – USB transfers management

Gadget API – Main functions

General operations (`usb_gadget_x()`):

- `probe_driver / unregister_driver`
- `set_selfpowered / clear_selfpowered`
- `vbus_connect / vbus_disconnect`
- `connect / disconnect`
- `frame_number`

Endpoint operations (`usb_ep_x()`):

- `autoconf / autoconf_reset`
- `enable / disable`
- `alloc / free`
- `queue / dequeue`
- `set_halt / clear_halt`
- `fifo_status / fifo_flush`

Decriptor operations:

- `usb_descriptor_fillbuf`
- `usb_gadget_config_buf`

Gadget API

Driver life cycle:

- Register driver for a particular device controller
- Register gadget driver (bind)
- Hardware powered, enumeration starts
- Gadget driver returns descriptors (setup)
- Gadget driver returns interfaces configuration
- Do real work (data transfer) until disconnect
- Gadget driver unloaded (unbind)

Existing Gadgets

- Ethernet
  - Enumerate to the host as an Ethernet device
  - Can easily be bridging, routing, or firewalling access to other networks
  - Interoperability with hosts running Linux, MS-Windows among others
  - Possibility to set parameters such as MAC address, IP configuration or DHCP use thanks to the bootargs if using a boot firmware like U-Boot

Existing Gadgets

- GadgetFS
  - Provides User-Mode API
  - Each endpoint presented as single I/O file descriptor
  - Normal read() and write() calls
  - Async I/O supported
  - Configuration and descriptors written into files

Note that user mode gadget drivers do not necesarily need to be licensed according to the GPL.

Existing Gadgets

- File-backed Storage
  - Implements the USB Mass Storage Class
  - Up to 8 disk drives can be set
  - Store file or block device is called the “backing storage”
  - Backing storage requires preparation
    - If a file is used, it must created with its desired size before launching the driver
    - If a block device, it must match host reaquirements (DOS partition for MS-Windows host)
  - The backing storage must not change while FBS is running, only the host should access it

Existing Gadgets

- Webcam
  - Acts as a composite USB Audio and Video Class device
  - Provides a userspace API to process UVC control requests and stream video data

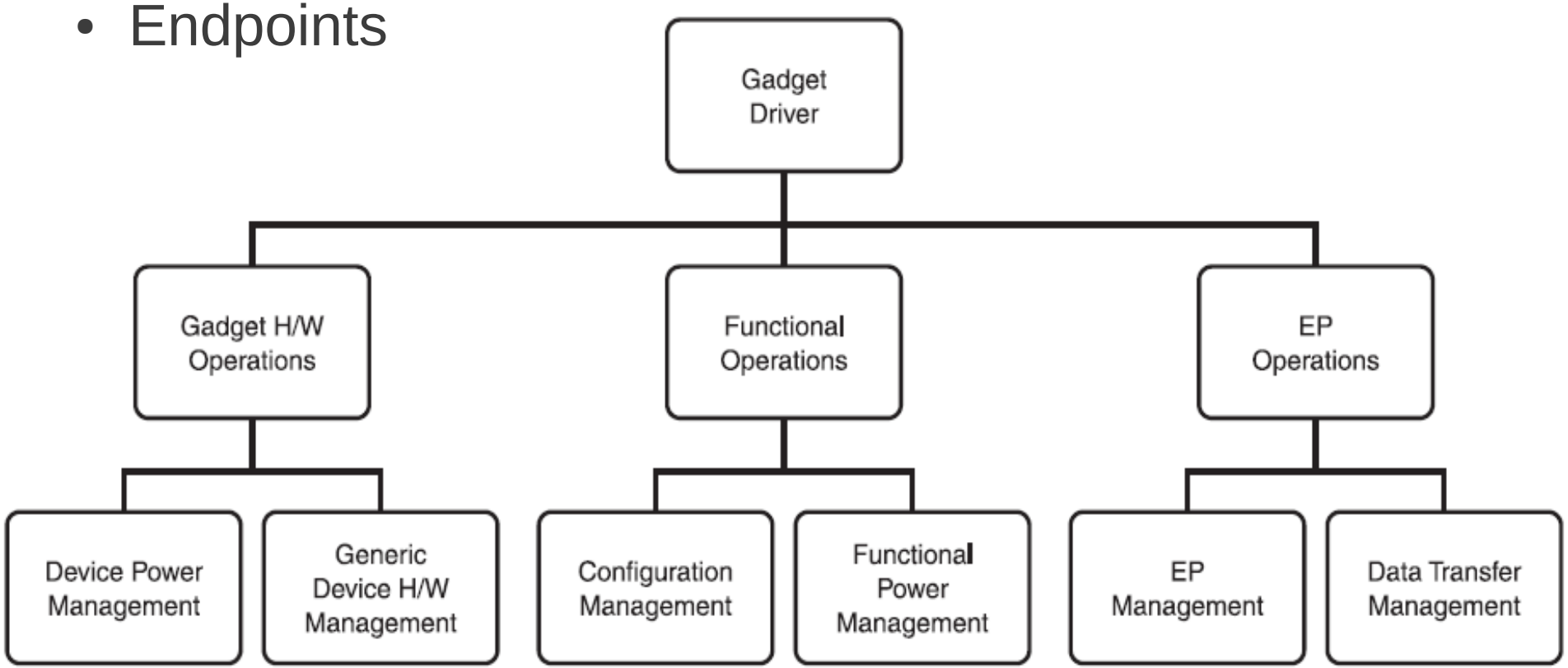
- Serial Gadget
  - Useful for TTY style operation
  - Supports a CDC-ACM module option

Existing Gadgets

- MIDI
  - Exposes an ALSA MIDI interface
  - Both recording and playback support
- GadgetZero
  - Useful to test device controller driver
  - Helps verify that the driver stack pass USB-IF (for USB branding)
  - On the host side, useful to test USB stack

Design your own Gadget

- 3 main operations to consider
  - Hardware
  - Functional
  - Endpoints

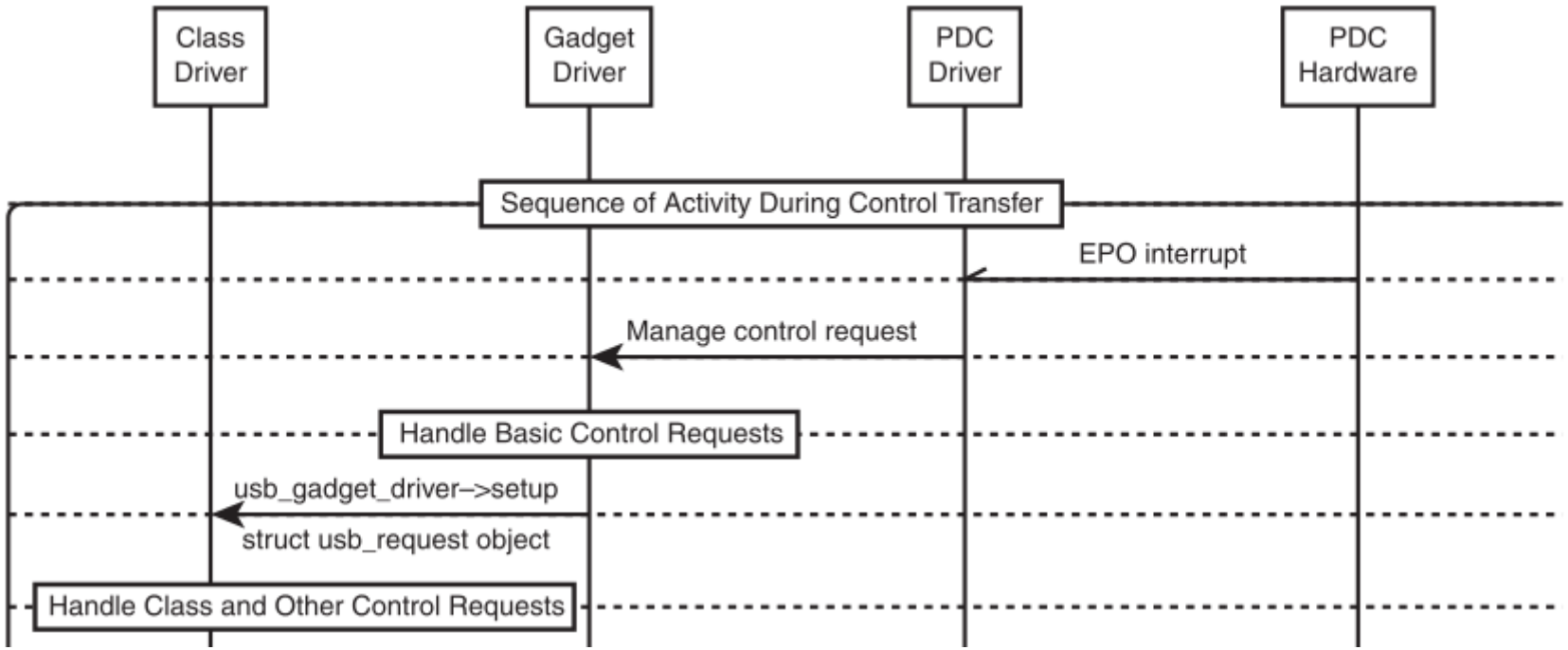


Design your own Gadget

- First implement the register/unregister functions
  - `usb_gadget_probe_driver`
    - Resgistration of the `usb_gadget_driver`
    - Responsible for binding the gadget driver and powering up the device
  - `usb_gadget_unregister_driver`
    - Responsible for unbinding the gadget from the functional driver and powering down the device
- Then define callbacks hardware related
  - Fill `usb_ep_ops` and `usb_gadget_ops`
  - Not necessary to support all functions

Design your own Gadget

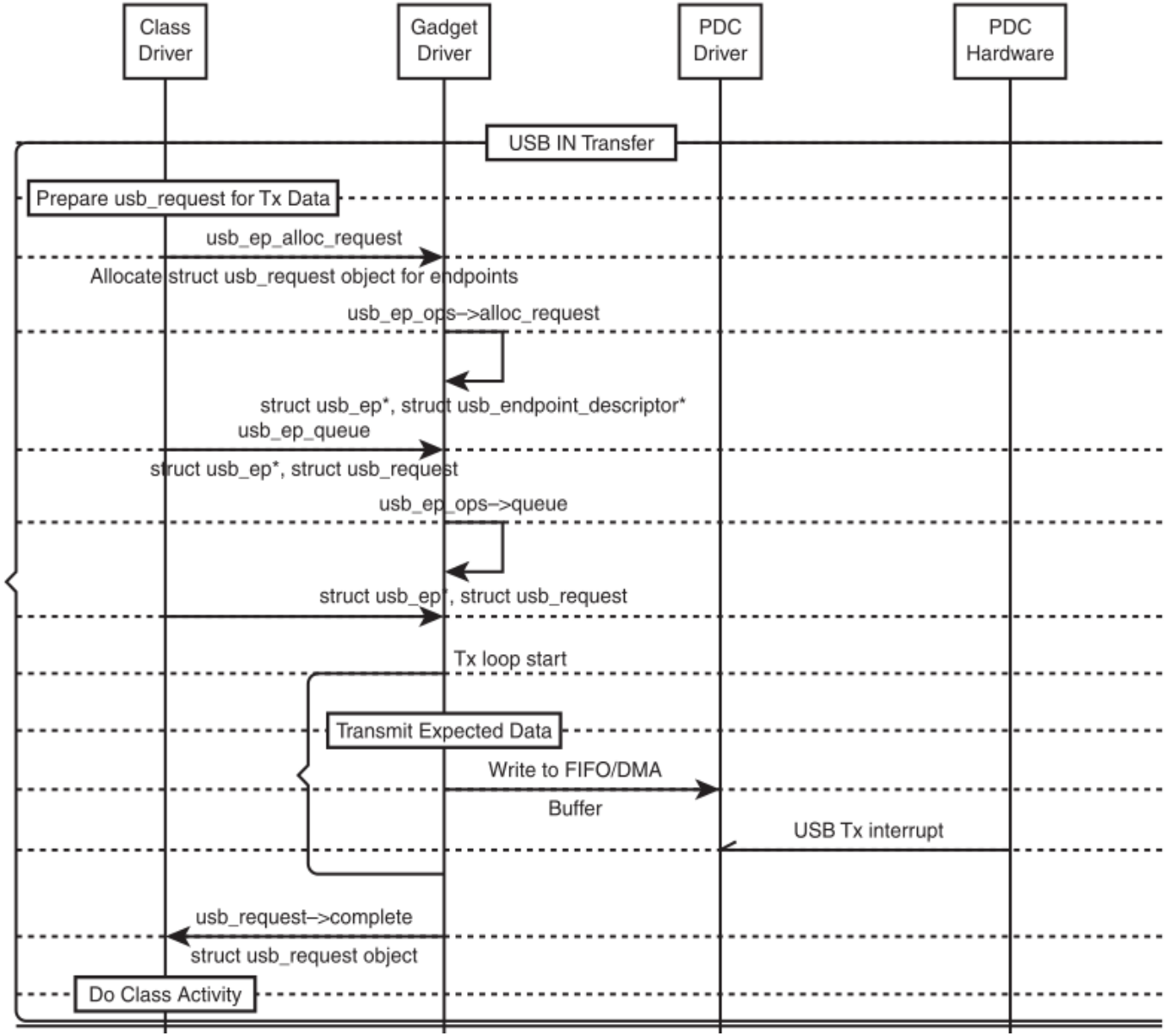
- Implement the control request handles (ep0)
  - Gadget driver handles only a part of it
  - The rest is routed to the class driver



Design your own Gadget

- Power Management requests
  - Comes from the PDC to the Gadget
  - The Gadget must pass the events to the class driver
- Once enumeration is done, class driver requests `usb_request` structure for IN/OUT transfers
  - Gadget receives data in interrupt routine (OUT)
    - Only when the expected amount is received the Gadget calls the `complete` function
  - Gadget sends data to the PDC (IN)
    - Wait send completion to inform the class driver

Design your own Gadget



Demo: Hardware

BeagleBoard xM

- ARM™ Cortex™-A8 1000 MHz
- USB connectivity:
  - 4 host ports
  - 1 OTG port (used as device)

Demo: Software

- Bootloader
  - U-boot 2011.12 r4
- Kernel
  - 3.0.17 r115c
- Root filesystem
  - Console image
    - Custom recipe (lighttpd)
  - Additional modules

Conclusion

- Easy to implement
- Hardware independent
- Scalability
- Large panel of existing gadgets
- Awareness of limitations

Questions?

Appendix: Files

The files used for this experiment should be attached with the presentation

- Rootfs:
  - Custom recipe provided if rebuild is necessary
- Additional modules:
  - Instructions to recompile them
- Demo script
- Lighttpd configuration file

Appendix: References

- Linux-USB Gadget API Framework: General presentation.
- USB Gadget API for Linux: Full description of the API.
- *Essential Linux Device Drivers (Sreekrishnan Venkateswaran)* : General device driver book containing a useful USB section.
- *Bootstrap Yourself with Linux-USB Stack (Rajaram Regupathy)*: Very detailed and easy-to-read book about Linux-USB.


deneo
Embedded