Make your own USB device without pain and money!

Krzysztof Opasiak

SAMSUNG

Samsung R&D Institute Poland

FOSDEM '16

Brussels 30 & 31 January 2016

Agenda

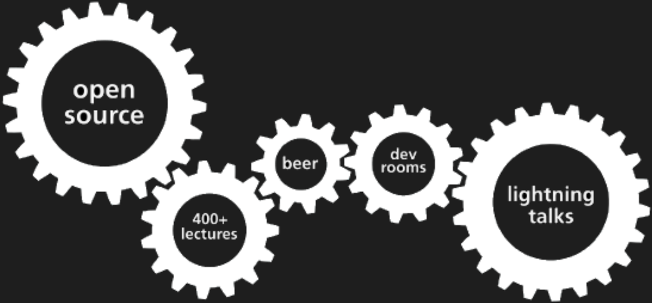
What USB is about?

How to compose a device?

How to create a function?

Summary

Q & A

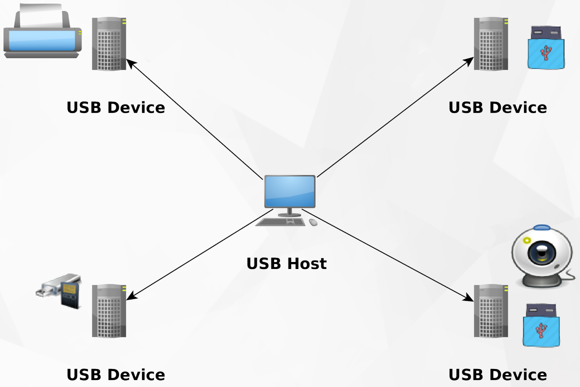


What USB is about?

What USB is about?

It is about providing services!

- Storage
- Printing
- Ethernet
- Camera
- Any other



What is USB device?

- Piece hardware for USB communication
- USB protocol implementation
- Some useful protocol implementation
- Piece of Hardware/Software for providing desired functionality

Endpoints…

- Device may have up to 31 endpoints (including ep0)
- Each of them gets an unique Endpoint address
- Endpoint 0 may transfer data in both directions
- All other endpoints may transfer data in one direction:

IN	Transfer data from device to host
OUT	Transfer data from host to device

Endpoint types

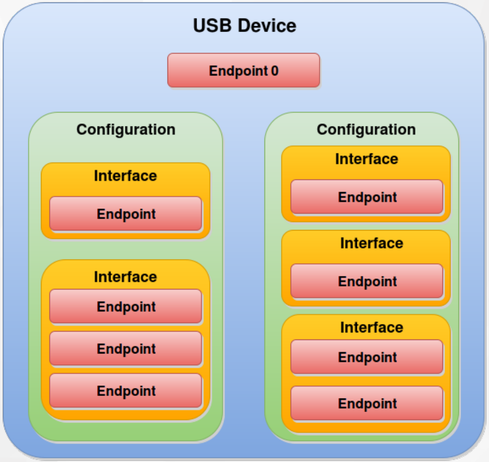
- **Control**
  - Bi-directional endpoint
  - Used for enumeration
  - Can be used for application
- **Interrupt**
  - Transfers a small amount of low-latency data
  - Reserves bandwidth on the bus
  - Used for time-sensitive data (HID)

Endpoint types

- **Bulk**
  - Used for large data transfers
  - Used for large, time-insensitive data (Network packets, Mass Storage, etc).
  - Does not reserve bandwidth on bus, uses whatever time is left over

- **Isochronous**
  - Transfers a large amount of time-sensitive data
  - Delivery is not guaranteed (no ACKs are sent)
  - Used for Audio and Video streams
  - Late data is as good as no data
  - Better to drop a frame than to delay and force a re-transmission

USB device

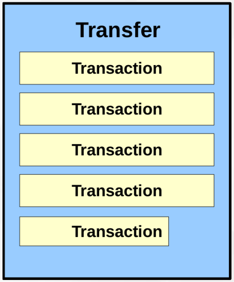

USB Device
Endpoint 0
Configuration
Interface
Endpoint
Interface
Endpoint
Endpoint
Endpoint
Configuration
Interface
Endpoint
Interface
Endpoint
Interface
Endpoint
Endpoint

USB bus

- **USB is a Host-controlled bus**
- **Nothing on the bus happens without the host first initiating it.**
- **Devices cannot initiate any communication.**
- **The USB is a Polled Bus.**
- **The Host polls each device, requesting data or sending data.**

USB transfer vs transaction

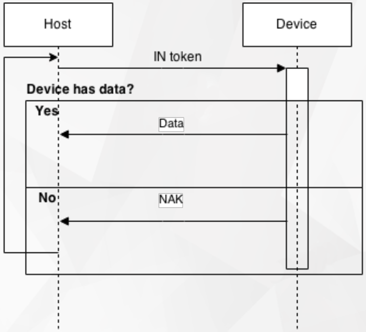
- **Transaction**
  - Delivery of data to endpoint
  - Limited by wMaxPacketSize
- **Transfer**
  - One or more transactions
  - May be large or small
  - Completion conditions



USB transport

IN

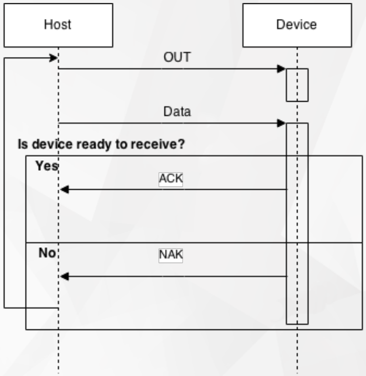
- **Host sends an IN token**
- **If the device has data:**
    - Device sends data
    - Host sends ACK
- **else**
    - Device sends NAK
    - Host will retry until timeout



USB transport

OUT

- **Host sends an OUT token**
- **Host sends the data (one packet)**
- **If device accepts data transfer:**
  - Device sends an ACK
- **else**
  - Device sends an NAK
  - Host will retry until success or timeout



∗ PING, NYET - bandwidth savers

Plug and Play - step by step

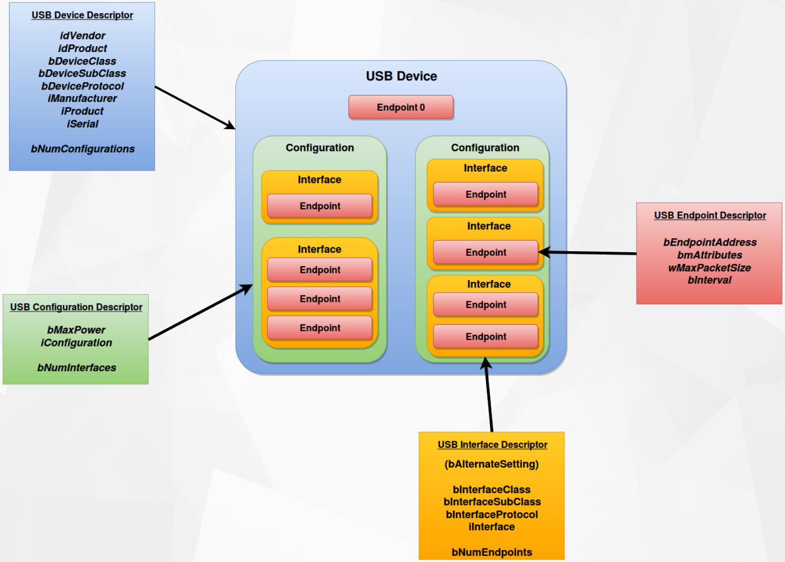
- **Plug in device**
- **Detect Connection**
- **Set address**
- **Get device info**
- **Choose configuration**
- **Choose drivers for interfaces**
- **Use it ;)**

Device details

- **Each USB world entity is described by data structure called descriptor**
- **Descriptors have different types, sizes and content**
- **But they all have a common header**

Field	Size	Value	Description
bLength	1	Number	Size of the Descriptor in Bytes
bDescriptorType	1	Constant	Device Descriptor (0x01)
<data>	bLength - 2	NA	Payload

USB descriptors



USB classes

00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Fh	Interface	Personal Healthcare
10h	Interface	Audio/Video Devices
11h	Device	Billboard Device Class
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

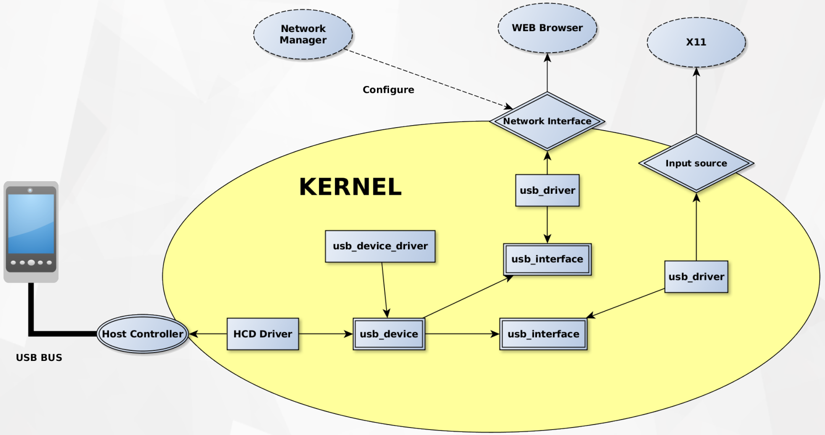
What USB driver really is?

- **Piece of kernel code / libusb app**
- **Usually provides something to userspace (network interface, block device, tty, etc.)**
- **Implementation of some communication protocol**
- **…so a little bit equivalent to your Web browser, ssh client etc.**

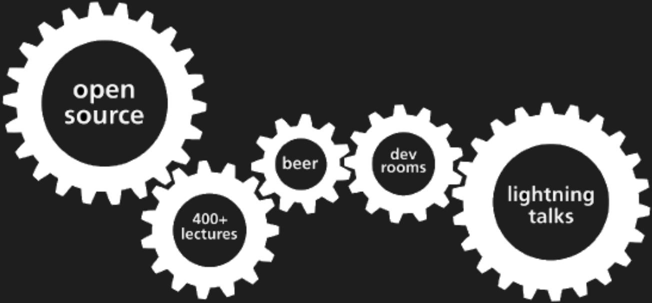
How to choose a suitable driver?

- *struct usb_device_driver* **vs** *struct usb_driver*
- **When device needs special handling:**
  - Using VID and PID and interface id
  - Driver probe()s for each interface in device that match vid and pid
- **When driver implements some well defined, standardized protocol**
  - Using bInterfaceClass, bInterfaceSubClass etc.
  - Driver probe() for each interface which has suitable identity
  - No matter what is the VID and PID
  - Driver will not match if interface hasn't suitable class

Big picture



Network Manager
WEB Browser
X11
Configure
Network Interface
Input source
KERNEL
usb_driver
usb_device_driver
usb_interface
usb_driver
Host Controller
HCD Driver
usb_device
usb_interface
USB BUS



How to compose a device?

FOSDEM '16
Brussels 30 & 31 January 2016

What is needed?

Need	Solution
Suitable hardware	Get some board with UDC controller (BBB, Odroid etc.)
Implementation of USB protocol	Use one from Linux kernel!
Implementation of some useful protocol	A lot of protocols are available out of the box in Linux kernel!
Desired functionality provider	Let's use our system infrastructure!

Terminology

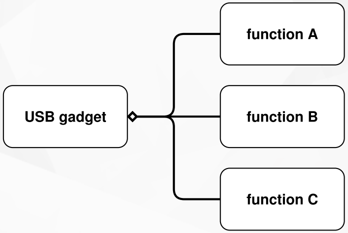
USB device = USB gadget + UDC

UDC driver Driver for USB Device Controller

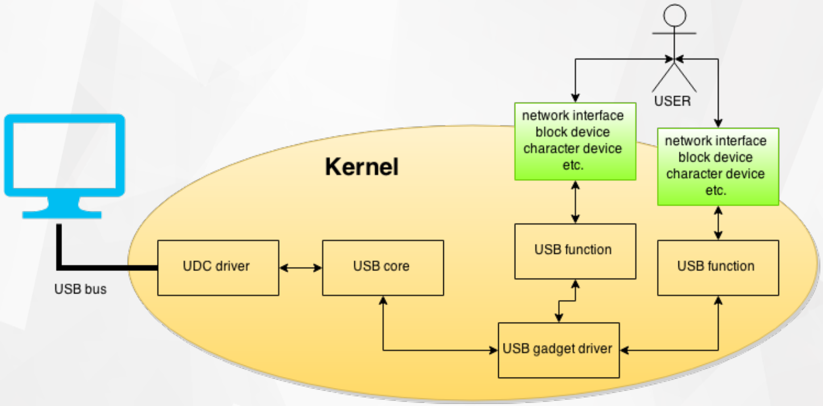
USB function (type) driver which implements some useful protocol (HID, Mass storage)

USB gadget Glue layer for functions.

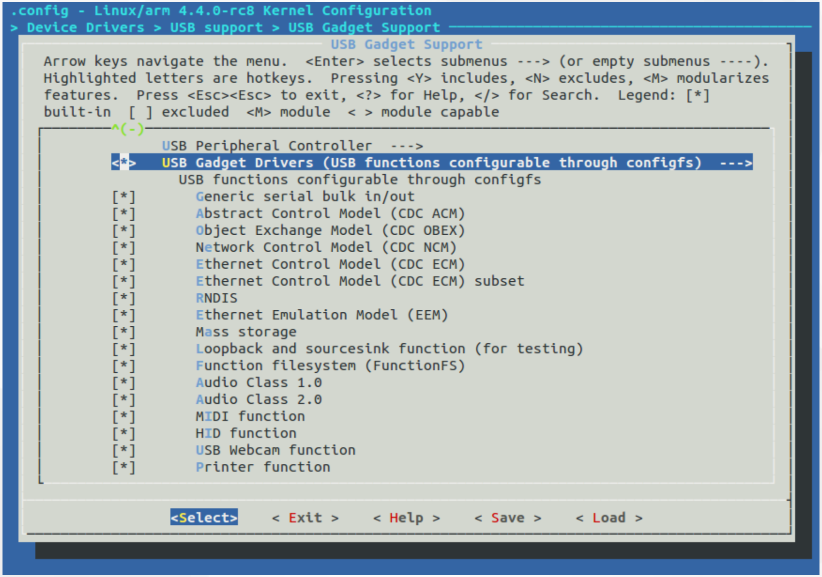
- **Handle enumeration**
- **Respond to most general requests**



Device architecture overview



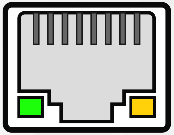
Prerequisites - menuconfig

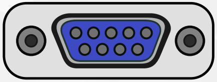


```
.config - Linux/arm 4.4.0-rc8 Kernel Configuration
> Device Drivers > USB support > USB Gadget Support ─────────────────────────
                           USB Gadget Support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable

        ^(-)
              USB Peripheral Controller  --->
        <*>   USB Gadget Drivers (USB functions configurable through configfs)  --->
                USB 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 2.0
        [*]       MIDI function
        [*]       HID function
        [*]       USB Webcam function
        [*]       Printer function

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Available functions

- **Ethernet**
    - ECM
    - EEM
    - NCM
    - Subset
    - RNDIS
- **Serial**
    - ACM
    - Serial
    - OBEX
- **Mass Storage**
- **HID**
- **UVC**
- **UAC**
- **Printer**
- **Phonet**
- **Loopback and SourceSink**





Base composition

- **Fill the identity of gadget**
  - Vendor ID
  - Product ID
  - Device Class details
  - Strings (manufacturer, product and serial)
- **Decide what functions**
- **Decide how many configurations**
- **Decide what functions are available in each configuration**

But how to do this?

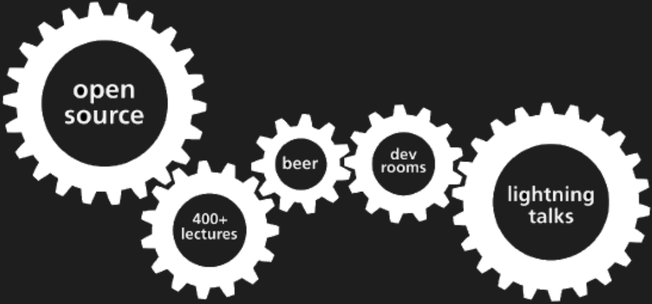
- **Use bare kernel ConfigFS interface**
  `Documentation/ABI/testing/configfs-usb-gadget*`
- **Use libusbgx to create a program**
  `https://github.com/libusbgx/libusbgx`
- **Use gt to create a simple script**
  `https://github.com/kopasiak/gt`
- **Use gt to load gadget scheme**

What gadget schemes really are?

- **Declarative gadget description**
- **Simple configuration file**
- **libconfig syntax**
- **Interpreted by libusbgx**
- **Can be easily loaded using gt load**

```
attrs = {
 idVendor = 0x1D6B
 idProduct = 0xe1ce
}
strings = ({
  lang = 0x409;
  manufacturer = "Samsung"
  product = "Sample␣gadget"
  serialnumber = "FOSDEM2016"
})
functions = {
  our_net = {
    instance = "net1"
    type = "ecm"
  }
}
configs = ({
  id = 1
  name = "c"
  strings = ({
    lang = 0x409
    configuration = "The␣only␣one"
  })
  functions = ("our_net")
})
```

Let's compose some device



How to create a function?

We can write a…?

Kernel function

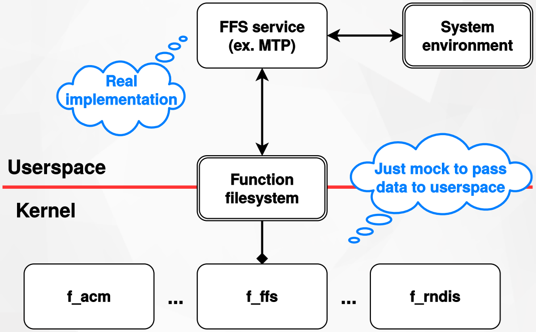
- Requires writing a kernel module
- May provide some generic entity to userspace
- Direct access to hardware

FunctionFS service

- Simple userspace service
- Only single service may communicate with host
- Easier access to system infrastructure (fs, net, dbus, etc.)

What FunctionFS really is?

- **USB function**
- **File system**
- **…which forwards all USB traffic to userspace**



Basic concepts

- **ep0 - communication + events + descriptors**
- **read() - schedule USB OUT request**
- **write() - schedule USB IN request**

Device cannot initiate any communication!

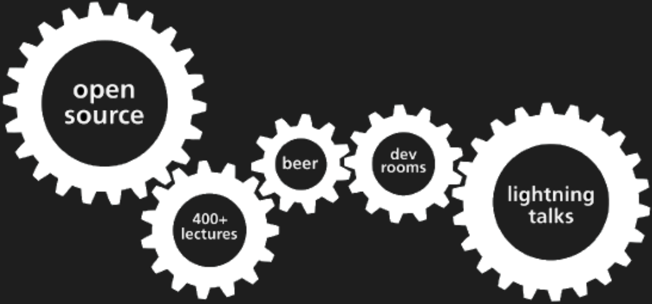
FunctionFS - HOWTO setup

- **Create function instance (ConfigFS)**
- **mount file system (pass instance name as dev)**
- **open ep0 file**
- **Write function descriptors**
- **Write function strings**
- **Open epXX (if any)**
- **Read events from ep0**
    - BIND
    - UNBIND
    - ENABLE
    - DISABLE
    - SETUP
- **read()/write() to other eps**

Sample source & demo

Why we should use aio?

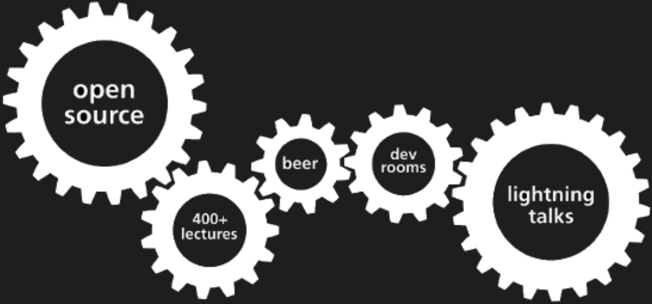
- **read() and write() on ep blocks till end of transfer**
- **ep0 should be also handled for events**
- **Usually function has more than one endpoint**
- **Threads are not the best solution…**



Summary

Summary

- **USB is about providing a services**
- **Communication protocol is implemented on both sides**
  - USB driver on host side
  - USB function on device side
- **There is a lot of USB functions in Linux kernel which can be easily used**
- **Use some helpers instead of bare ConfigFS interface**
- **While implementing own function consider using FunctionFS**
- **Remember to use AIO in FFS**
- **Have fun with USB!**



Q & A

FOSDEM '16

Brussels 30 & 31 January 2016

Thank you!

Krzysztof Opasiak

Samsung R&D Institute Poland

+48 605 125 174
k.opasiak@samsung.com