FROM ZERO TO XHCI DRIVER

Josh Felmeden – Capgemini Engineering

CURRENT STATE OF PROJECT

- Native seL4 xHCI USB driver for Avnet MaaXBoard.
- Supports mouse, keyboard, and touchscreen
- Can support all three simultaneously through one port with a USB hub.
- Available at https://github.com/sel4-cap/sel4-xhci

INITIAL PROJECT GOALS

- Create an seL4 native USB driver
  - Specifically xHCI (extensible Host Controller Interface)
- Develop a driver with sustainability in mind
  - Easy to keep updated
- Will help lower barrier for entry
  - Drivers for useful protocols encourage and facilitate use of seL4
- Provide an experience report

seL4 DEVELOPER KIT

- Previous work package focused on creating extensive driver support (https://github.com/sel4devkit)
- Using U-Boot driver
- Allows a wide range of platforms to have access to drivers
- Easy to extend the library

DEVICE DRIVERS

- Focus today is specifically on an xHCI driver

- Previous work package (seL4 developer kit) provides a working driver with caveats
  - No interrupts
    - Relies on polling – less performant than interrupt driven drivers
  - Licensing
    - U-Boot is GPL licensed

- Current development offers independent xHCI driver without the above

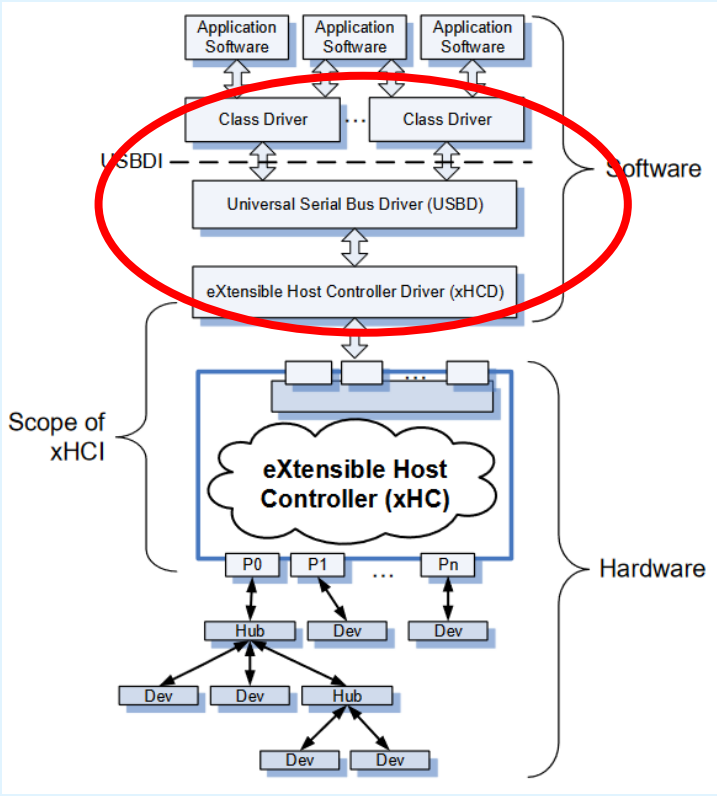
NetBSD

Why NetBSD?

- Boasts portability
  - Generic code lends itself to supporting a wide range of hardware
- BSD licensed
  - Minimal restrictions on use and distribution of software
- Updated often
  - Easy to keep up to date, and ensures safety
- Utilises interrupts
- Provides base to pull other drivers/services in future
  - E.g. Filesystem for mass storage

xHCI – EXTENSIBLE HOST CONTROLLER INTERFACE

- Protocol used for communicating with USB devices
- Advancement over OHCI/UHCI/EHCI, supporting all speeds
  - No need to explicitly include other drivers
- Describes communication between software and host controller
  - How memory should be set up
  - Data structures
  - MMIO (Memory Mapped Input/Output)
  - DMA (Direct Memory Access)



Capgemini engineering

CAmkES

Component Architecture for microkernel-based Embedded Systems

INITIAL RESEARCH

- First challenge was interfacing with hardware

- Compile debug NetBSD image for trace of xHCI initialisation
  - Take notes of first MMIO read/writes
  - Attempt to recreate them

- Initially make use of hardcoded register addresses
  - Make use of device tree to setup memory regions instead

- Then, initialise host controller
  - Account for ‘controller quirks’
  - DMA library utilised to setup DMA regions
  - Setup Interrupts
  - Tell host controller to start
  - Host controller initialised!

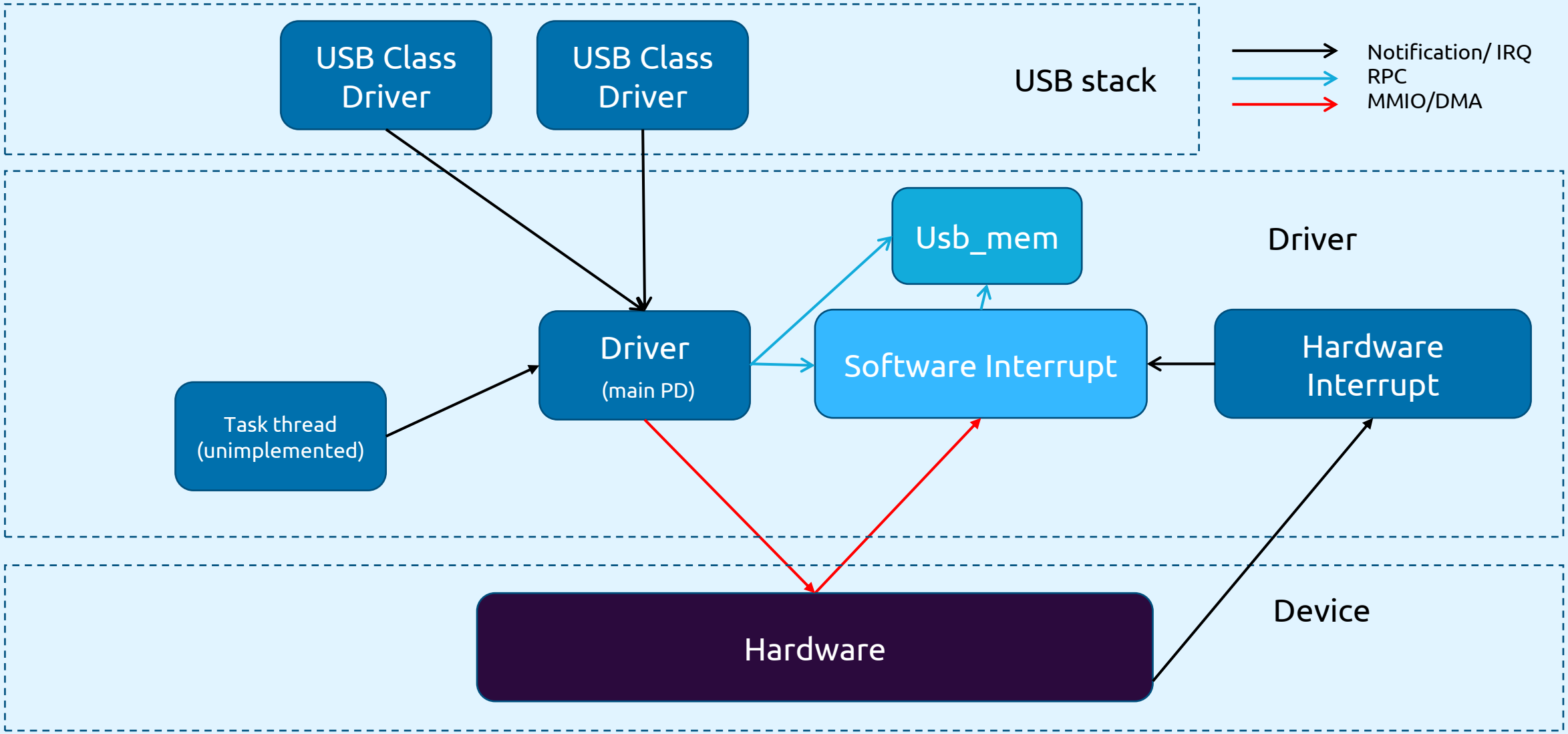
Capgemini engineering

MICROKIT

TRANSITIONING TO MICROKIT

- New system description language
  - Took some time to readjust, but overall very positive experience
- No standard libraries
  - Imported external, minimalist replacement for needed functionality (print/memory handling libraries)
  - Memory regions
- No DMA library
  - Created a simple DMA handler
- No inbuilt build system
  - Opted for simple Makefile

SYSTEM ARCHITECTURE



NEW DEVICES - KEYBOARD

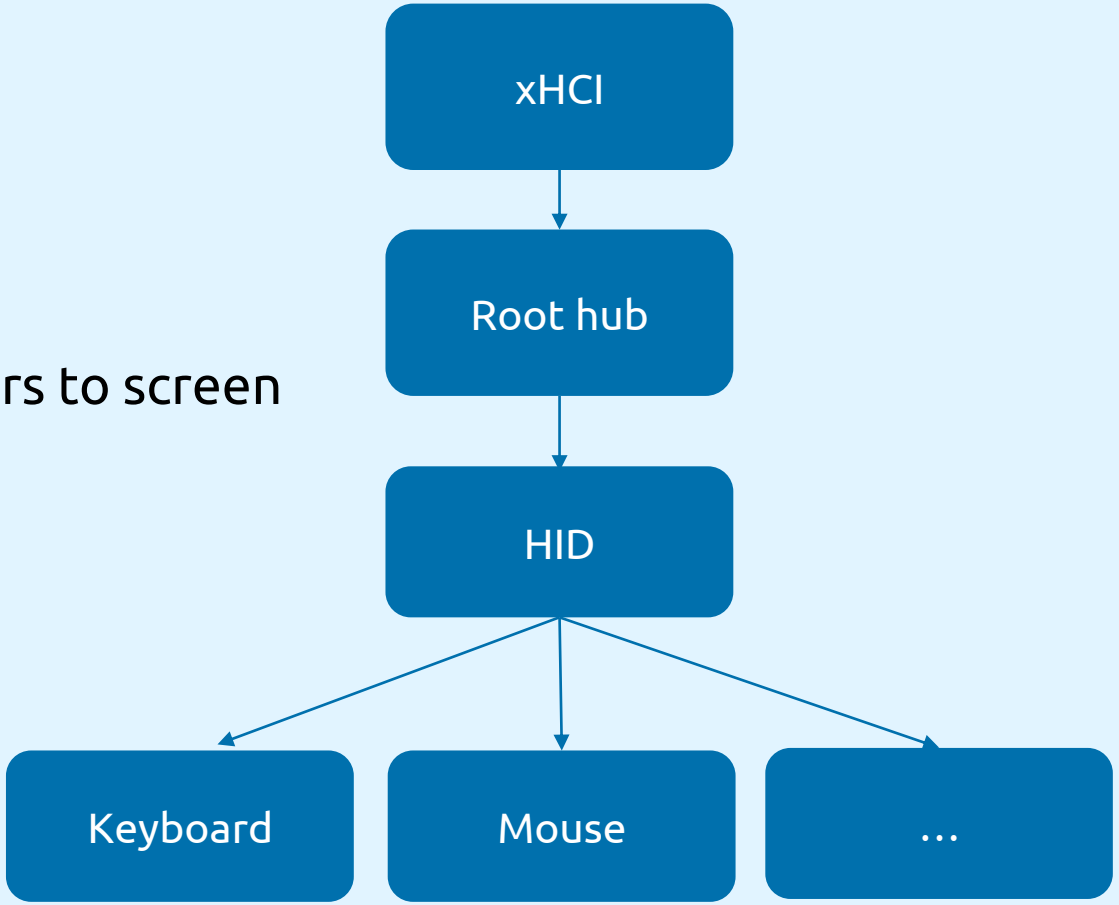
- With root hub initialised and attached, begin looking at connecting a device

- Started with keyboard
  - Fairly straight forward
  - Requires fewer additional header files

- Aim to get proof of concept by outputting characters to screen

- Human Interface Device (HID) required as a base

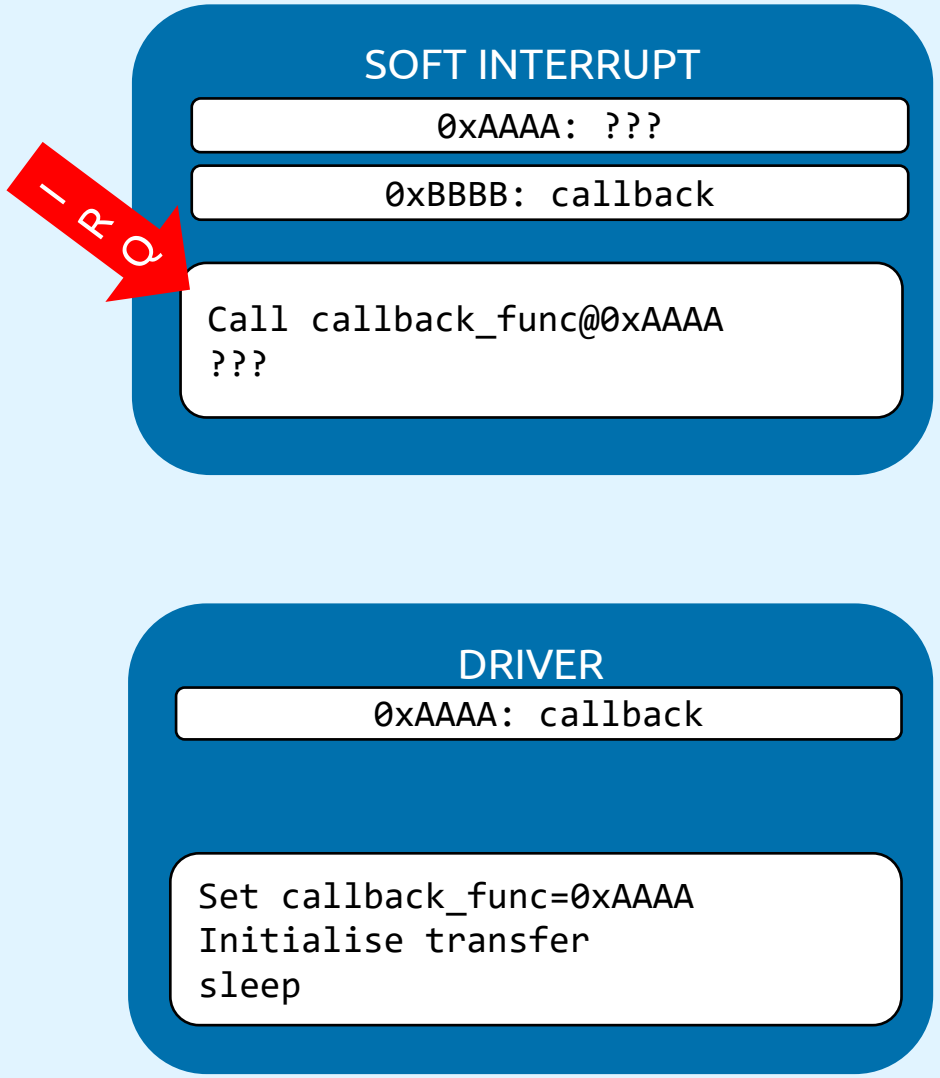
- Issues with hard coded functions
  - USB transfer callbacks



MEMORY DIFFICULTIES

- Memory addresses not the same between protection domains (PD)
  - E.g. if variable is initialised in general computation PD, the function memory address is different in software interrupt PD

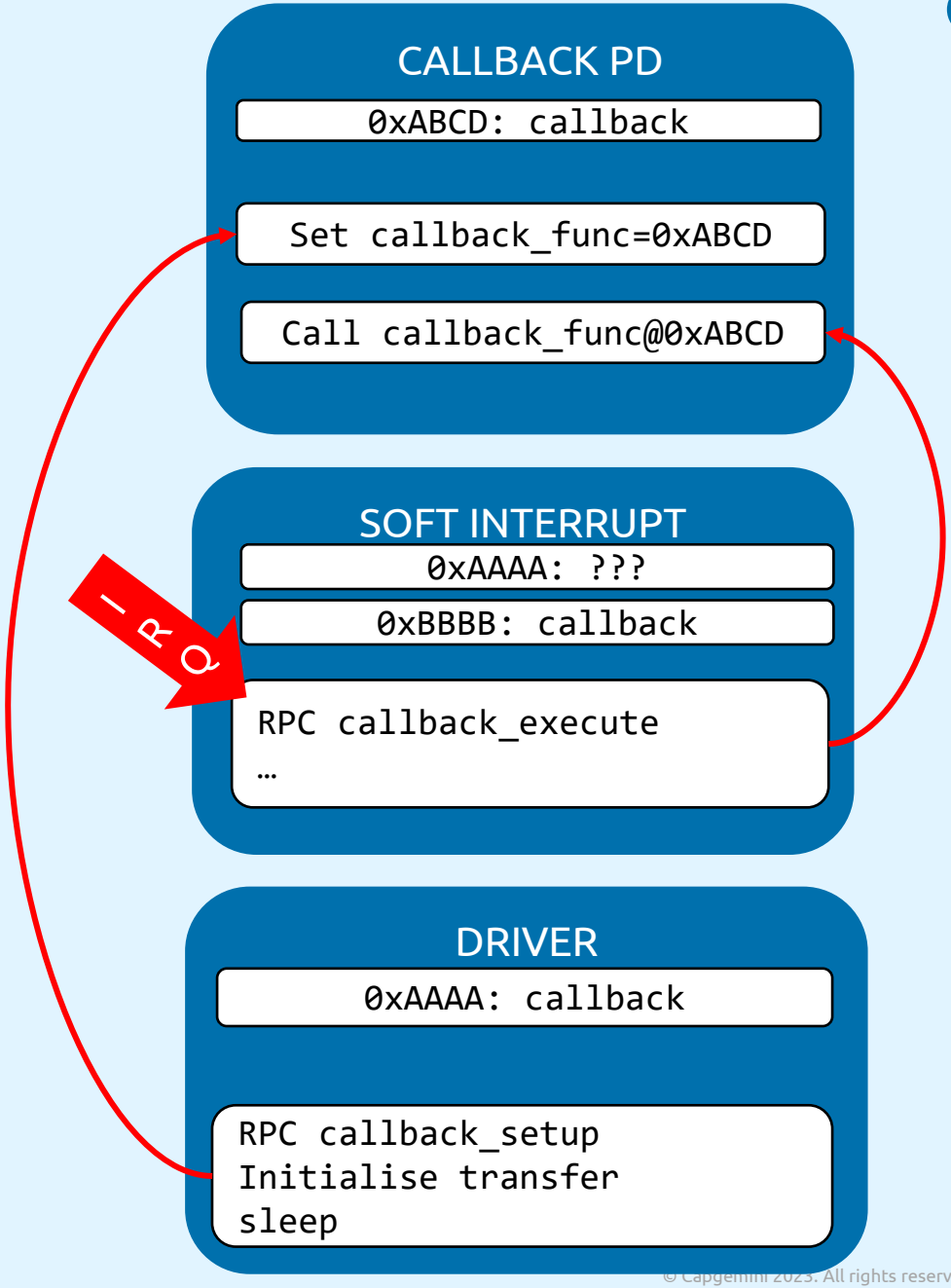
- Two possible solutions
  - Master PD that handles all such functions
  - “Context switch”, compare address with function value, switch to local function if mismacth



MEMORY DIFFICULTIES

- Memory addresses not the same between protection domains (PD)
  - E.g. if variable is initialised in general computation PD, the function memory address is different in software interrupt PD

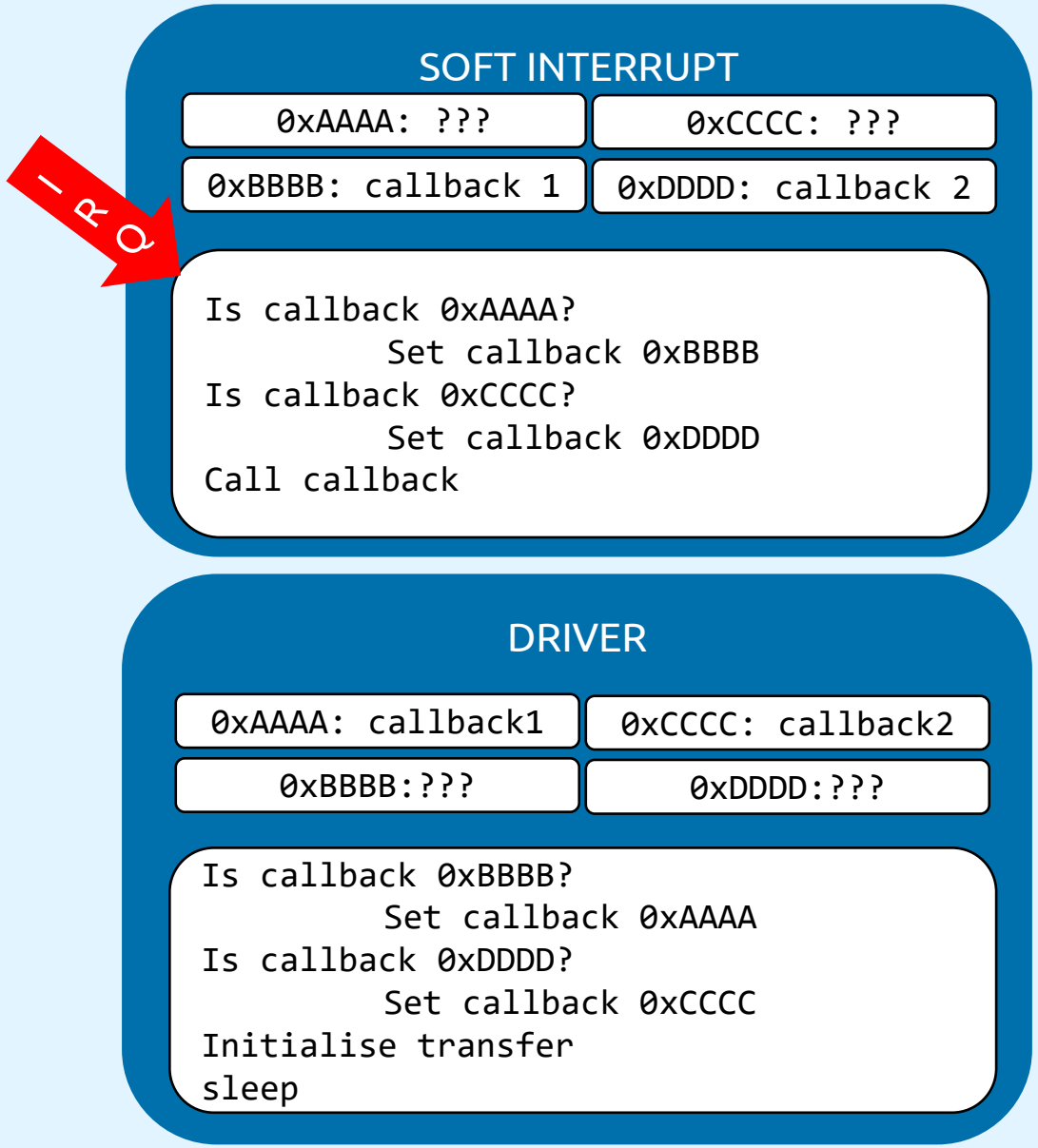
- Two possible solutions
  - Master PD that handles all such functions
  - “Context switch”, compare address with function value, switch to local function if mismatch



MEMORY DIFFICULTIES

- Memory addresses not the same between protection domains (PD)
  - E.g. if variable is initialised in general computation PD, the function memory address is different in software interrupt PD

- Two possible solutions
  - Master PD that handles all such functions
  - “Context switch”, compare address with function value, switch to local function if mismatch



NEW DEVICES – MOUSE/TOUCHSCREEN

- Keyboard added, next up is a mouse
  - Conceptually very similar, both HIDs
  - Require same fundamental layer
- Touchscreen conceptually very similar to mouse
  - Able to make use of generic code used for mouse and add touchscreen specific code on top
- When probing the bus for devices, have to explicitly say what device is being attached
  - Inflexible; requires code modification and recompile when switching between devices
  - Introduce NetBSD auto configuration
- USB Hub
  - Able to combine all three devices for use simultaneously

AUTO CONFIGURATION

- NetBSD makes use of “match” function to evaluate what USB device driver to assign

- Auto configuration uses a list of devices for reference created at kernel compile time
  - We can extract this list by using a prebuilt file
  - Trim down unnecessary devices (e.g. non-xHCI/not implemented)
  - Avoids explicit setup requirement

- Allows our driver to determine the best match automatically for each device

LINKING WITH OTHER PROJECTS

- Extracting keypress using channels
- Exporting to other projects
  - sDDF ethernet driver, sending keypress over ethernet
- Proves compatibility with seL4 philosophy
- Provides example of how to interact with driver

FUTURE WORK

- Mass storage
  - First device that does not have HID fundamentals
  - Requires more complicated concepts (file systems)
- Update to seL4 devkit
- LibC for Microkit
- xHCI driver available for use at https://github.com/sel4-cap/sel4-xhci!

Capgemini engineering

QUESTIONS?

Capgemini engineering

About Capgemini Engineering

Capgemini Engineering combines, under one brand, a unique set of strengths from across the Capgemini Group: the world-leading engineering and R&D services of Capgemini Engineering – acquired by Capgemini in 2020 – and Capgemini's digital manufacturing expertise. With broad industry knowledge and cutting-edge technologies in digital and software, Capgemini Engineering supports the convergence of the physical and digital worlds. Combined with the capabilities of the rest of the Group, it helps clients to accelerate their journey towards Intelligent Industry. Capgemini Engineering has more than 52,000 engineer and scientist team members in over 30 countries across sectors including aeronautics, automotive, railways, communications, energy, life sciences, semiconductors, software & internet, aerospace & defence, and consumer products.

Capgemini Engineering is an integral part of the Capgemini Group, a global leader in partnering with companies to transform and manage their business by harnessing the power of technology. The Group is guided every day by its purpose of unleashing human energy through technology for an inclusive and sustainable future. It is a responsible and diverse organization of 290,000 team members in nearly 50 countries. With its strong 50 year heritage and deep industry expertise, Capgemini is trusted by its clients to address the entire breadth of their business needs, from strategy and design to operations, fueled by the fast-evolving and innovative world of cloud, data, AI, connectivity, software, digital engineering, and platforms. The Group reported in 2020 global revenues of €16 billion.

Get The Future You Want | www.capgemini-engineering.com | www.capgemini.com

SOURCES

NetBSD
xHCI
USB stack

Wrapper
MMIO reads/writes
DMA access
Interrupt callbacks
Microkit Interaction

External Sources
Memory allocation
Printf