https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Debug Capability (DbC) Support on FreeBSD, Revised

Hiroki Sato <hrs@FreeBSD.org>
BSDCan 2024 / 2024.5.31

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Outline

- **Who am I?**
    - A Japanese FreeBSD committer since 2000, working in various areas
- **Outline of This Talk**
    - Background
    - USB Debug Capability
        - High-level Overview
        - USB Host/Device Controller Basics
            - Pipes and Endpoints
            - TRBs
        - Implementation Details
    - Demo and Future Work

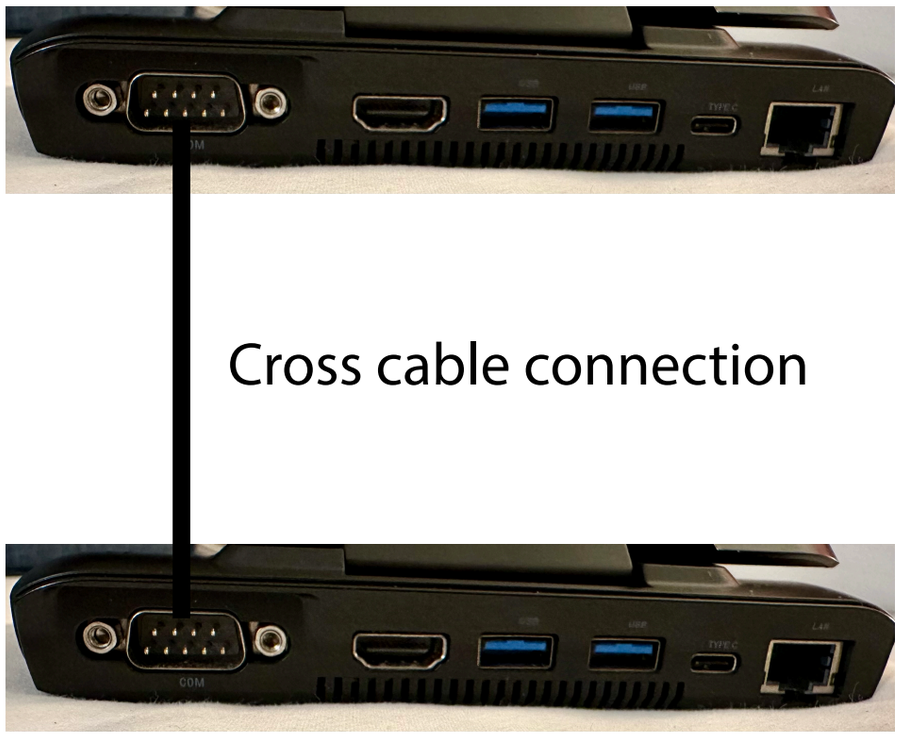
https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Background

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Background

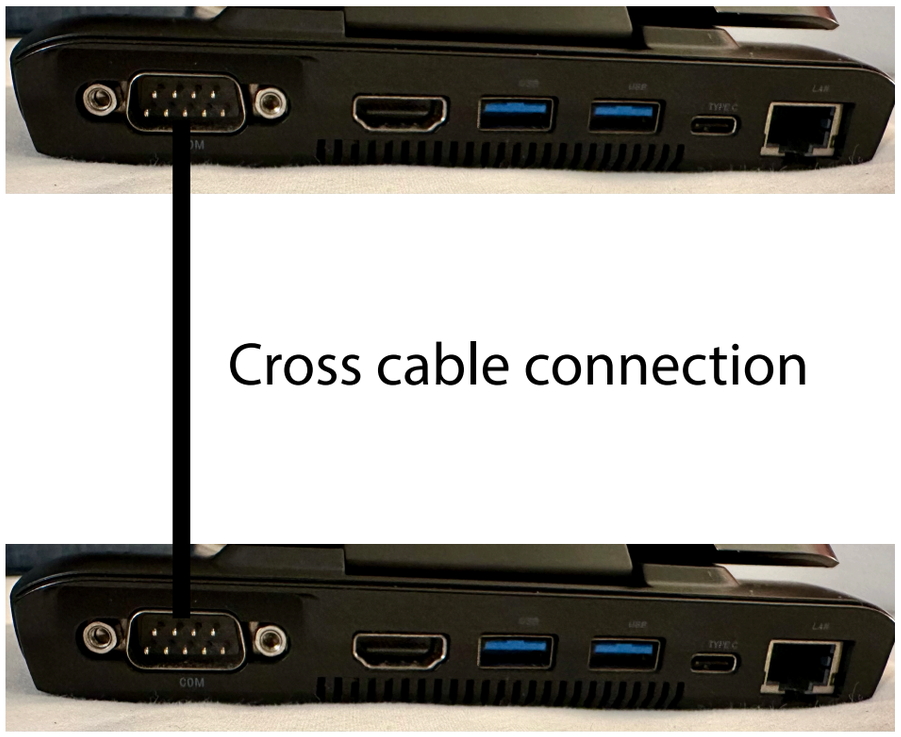
- **Debugging work using serial console:**
  - Remote access to a headless machine, including firmware (BIOS/UEFI) configuration
  - Device driver hacking
    - Remote GDB session



https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Background

- **Debugging work using serial console:**
  - Remote access to a headless machine, including firmware (BIOS/UEFI) configuration
  - Device driver hacking
    - Remote GDB session

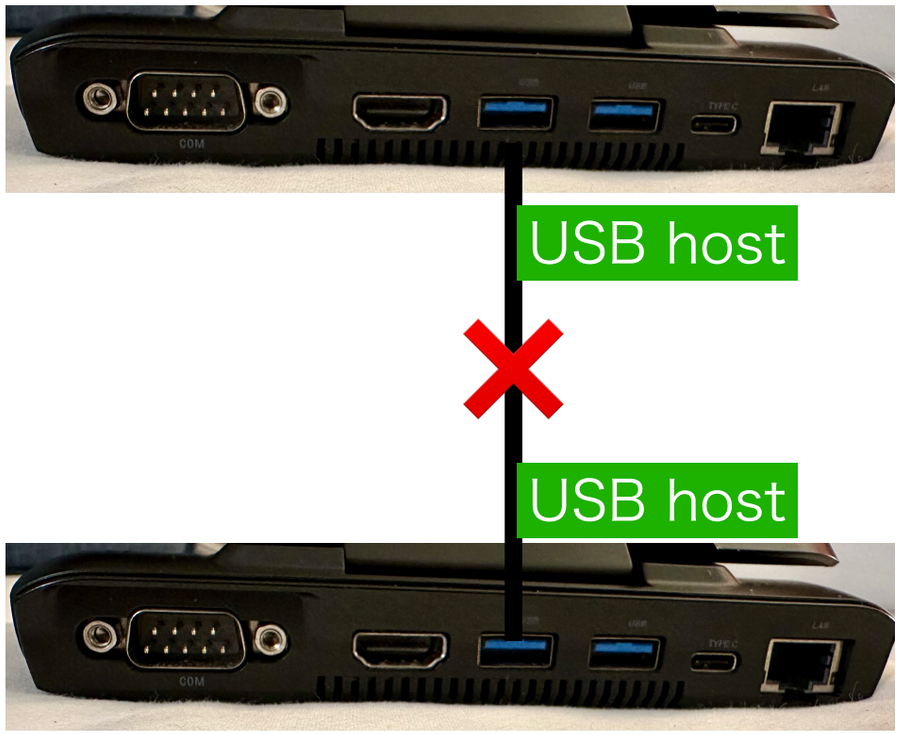


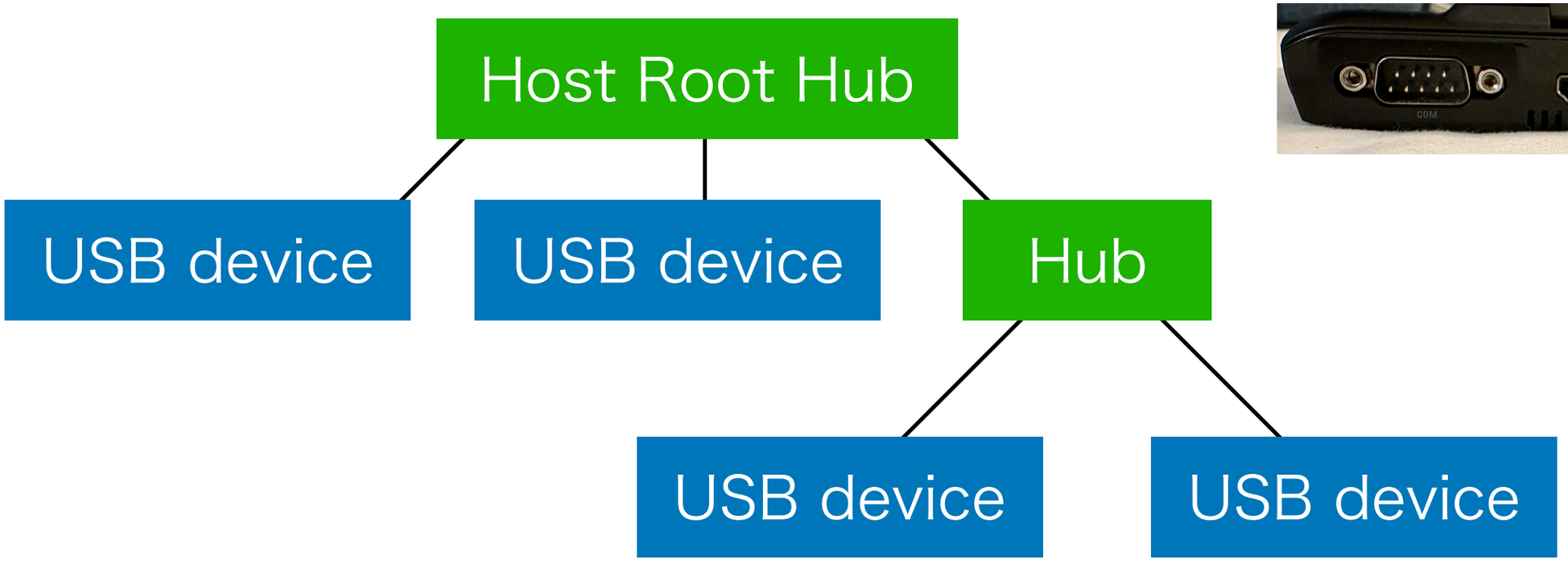
- **No serial port on modern hardware, however...**
  - A legacy interface
  - Server-grade machines have BMC with "console redirection"
    - BMC: baseboard management controller
      - An embedded processor that runs independently
      - Provides virtual serial ports over IPMI SoL (Serial-over-LAN, 623/udp)

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Background

- **USB is the replacement of legacy interfaces including the serial ports**
  - USB basically requires tiered star topology
  - No direct connection of USB hosts is allowed





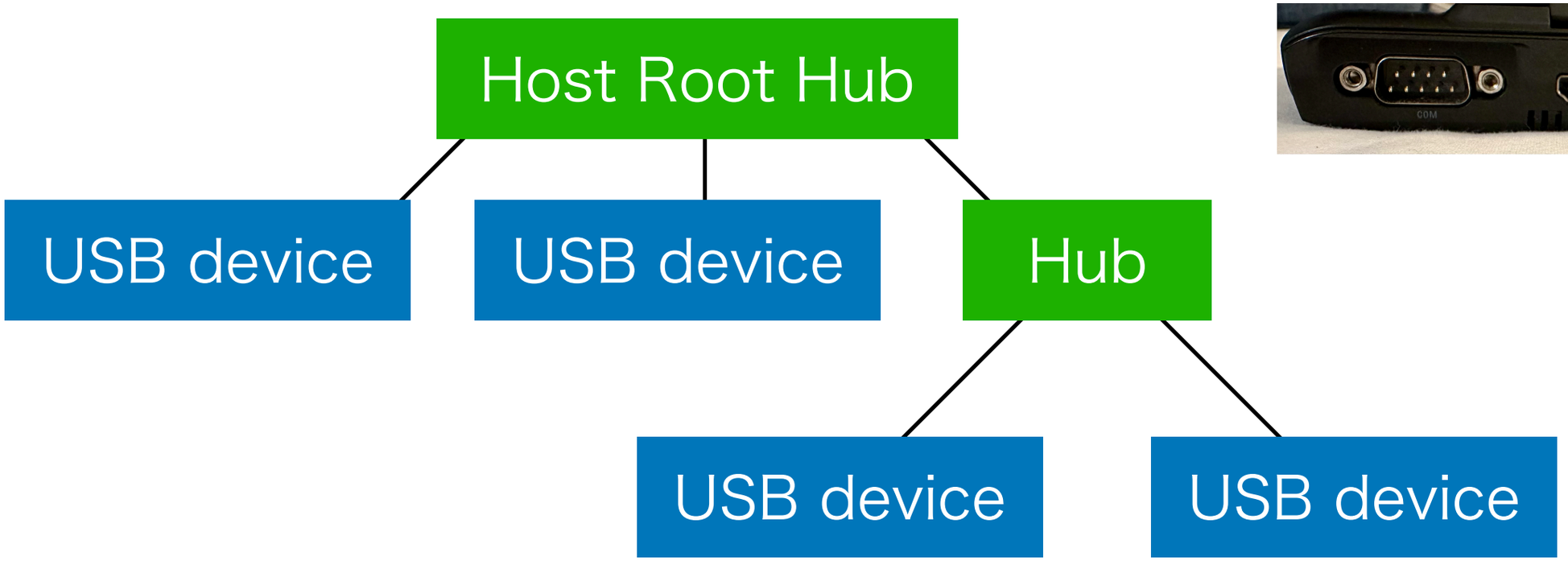
https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Debug Capability

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Debug Capability

- **1-sentence summary: USB DbC changes one of the USB ports on a USB host for a USB device**
  - Not a point-to-point connection
  - An optional feature in USB 3.0 Specification
  - Most of xHCI controllers support it



https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

A-to-A Cable?

- **A-to-A USB3 Cross Cable is required**
  - No A-A for USB 2.0. It is not allowed.
  - USB3 spec has five cables including A-A. **A-A is always a cross cable.**
  - Note that non-standard A-A cables can be found in the market.

Reference: USB 3.1 Legacy Connector and Cable Specification

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Similar Technologies

- **IEEE 1394 (FireWire) supports point-to-point connection and physical memory access**
  - OHCI specification
  - You can read/write memory
  - dcons(4) is a serial communication driver using this
  - Firewire is considered a legacy interface

- **USB2.0 also supports debug capability**
  - EHCI specification
  - Requires a special repeater hardware

Signal Name	Wire Number
VBUS	1
D−	2
D+	3
GND	4

By Nicola02nb - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=117668227

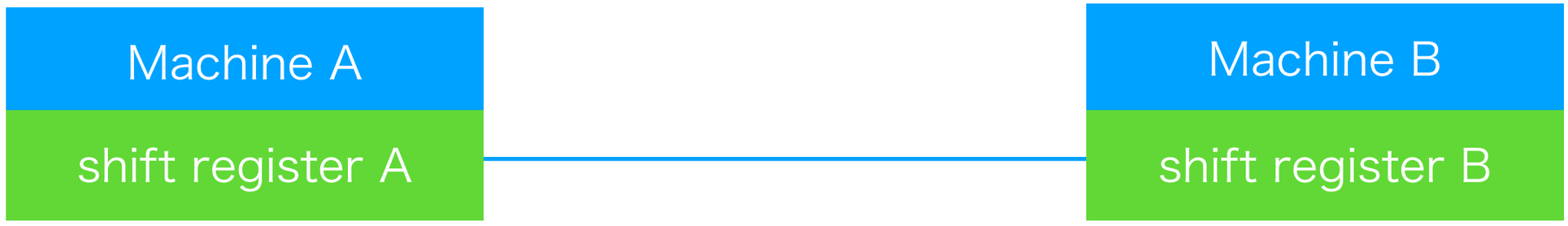
https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Implementation Details

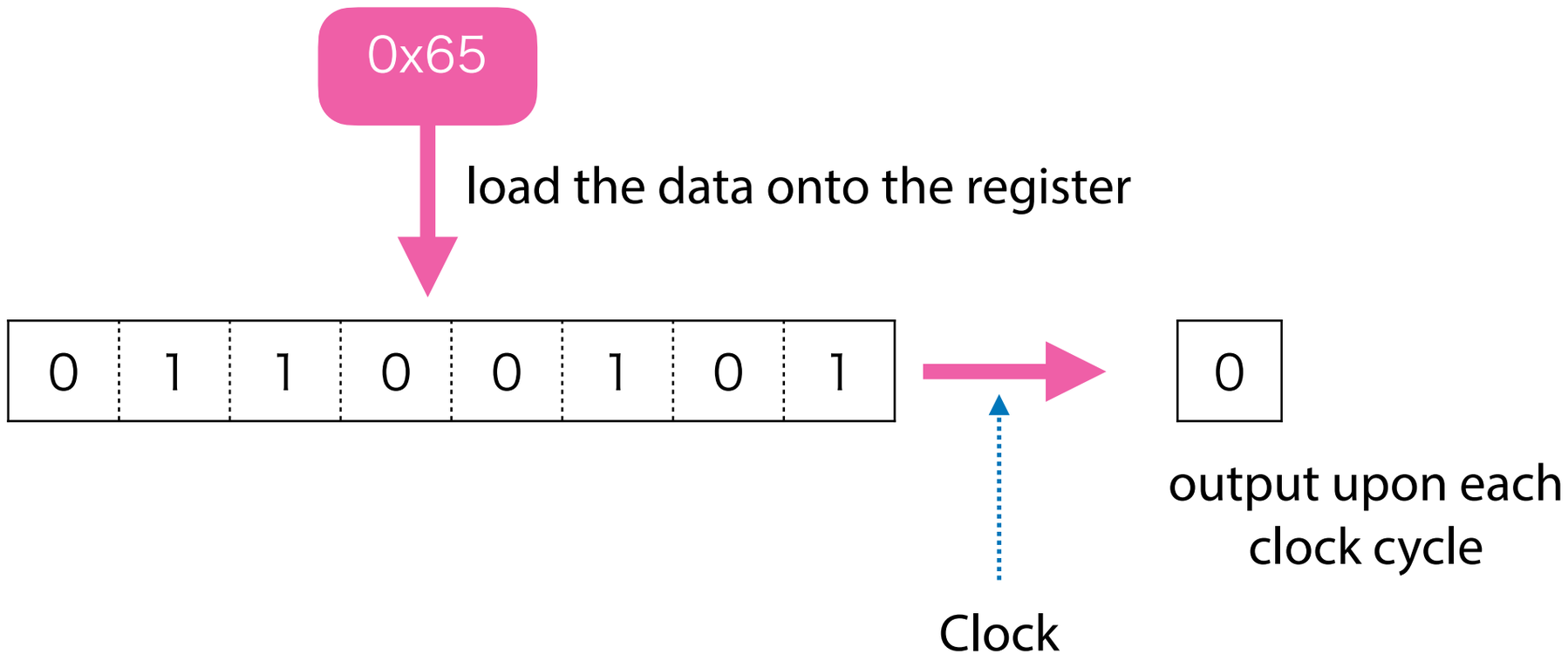
https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Host/Device Controller Basics

- **Serial communication over the legacy serial ports**



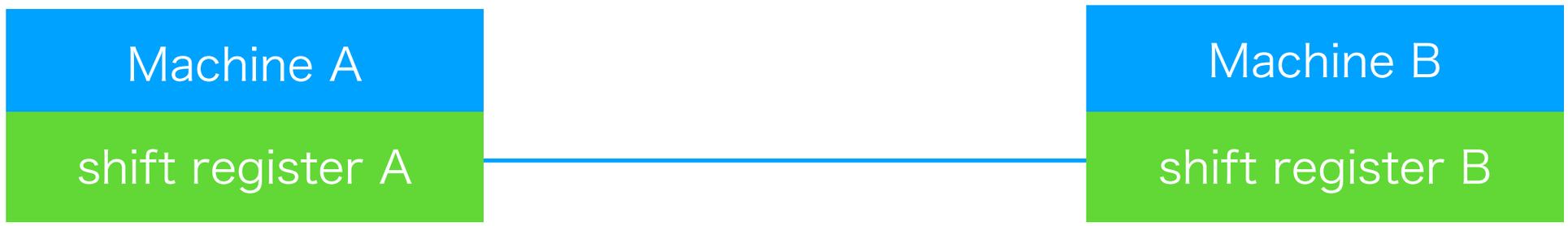
Shift register to convert data into a pulse sequence

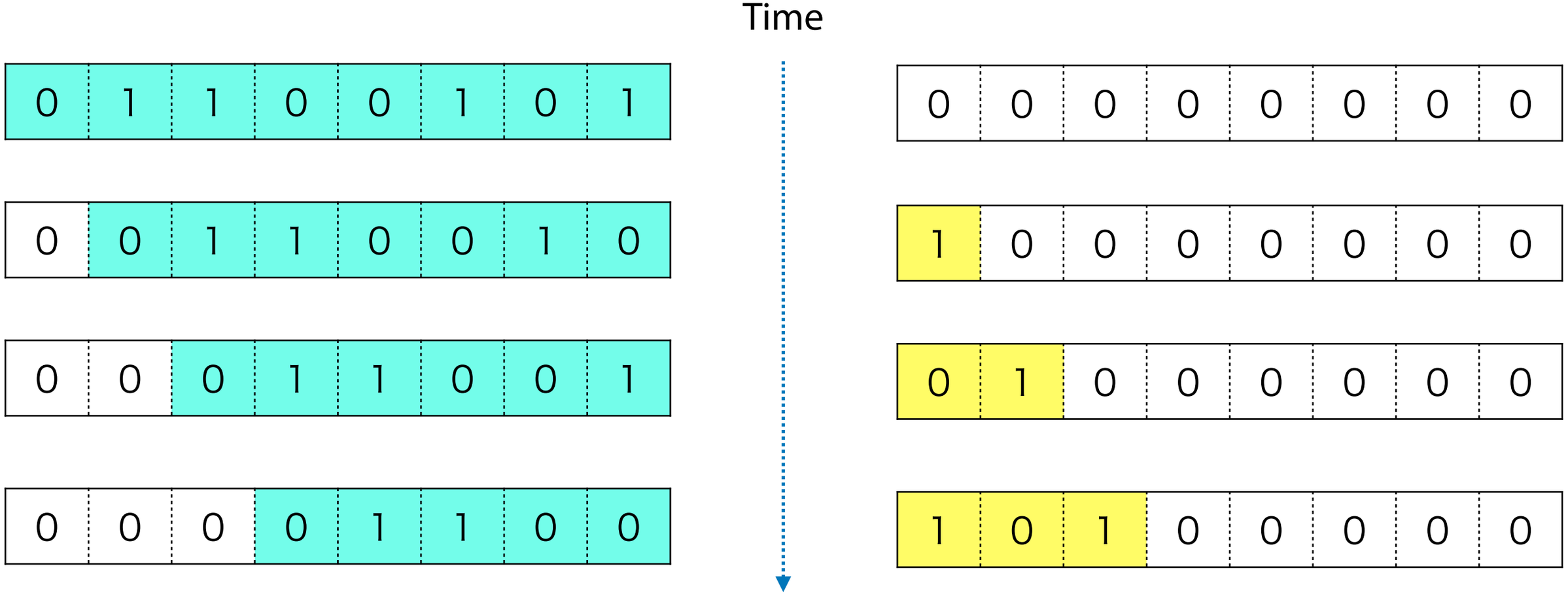


https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Host/Device Controller Basics

- **Serial communication over the legacy serial ports**

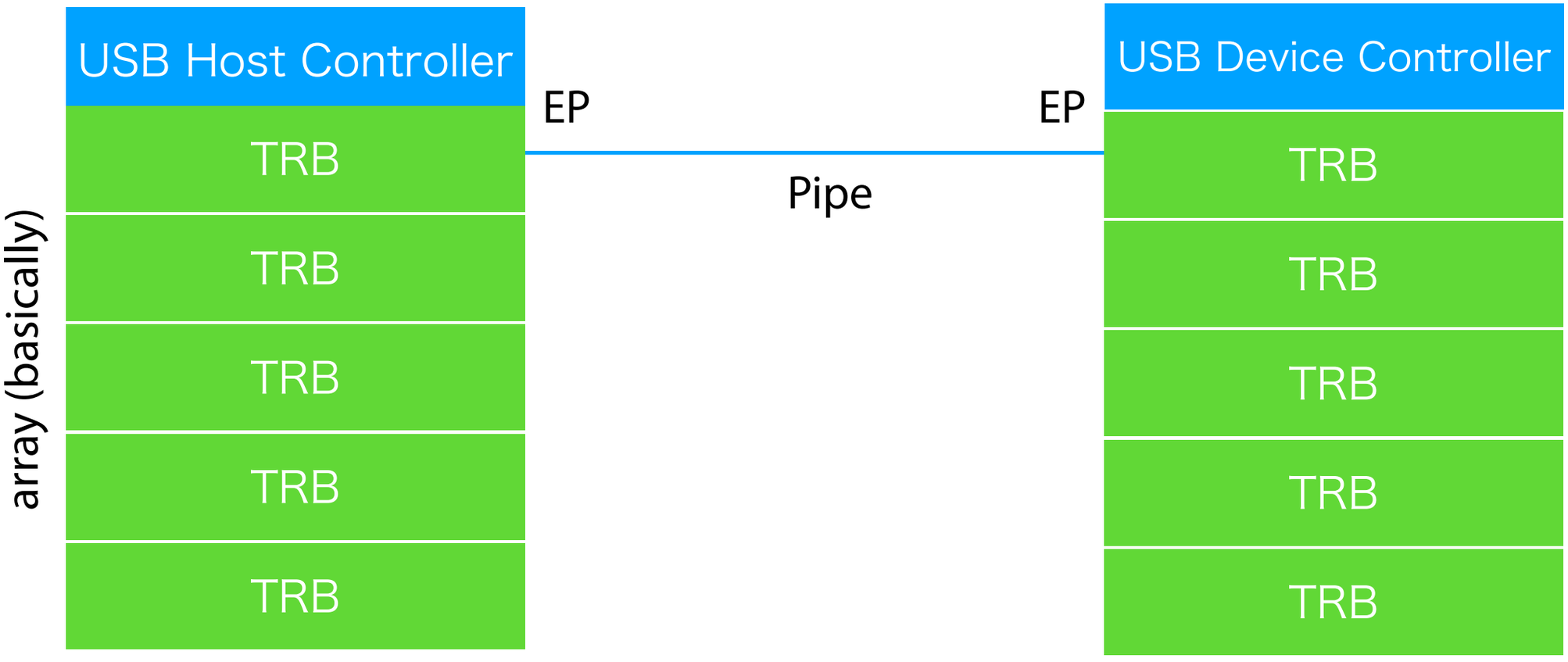




https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Host/Device Controller Basics

- **Serial communication over USB using xHCI**

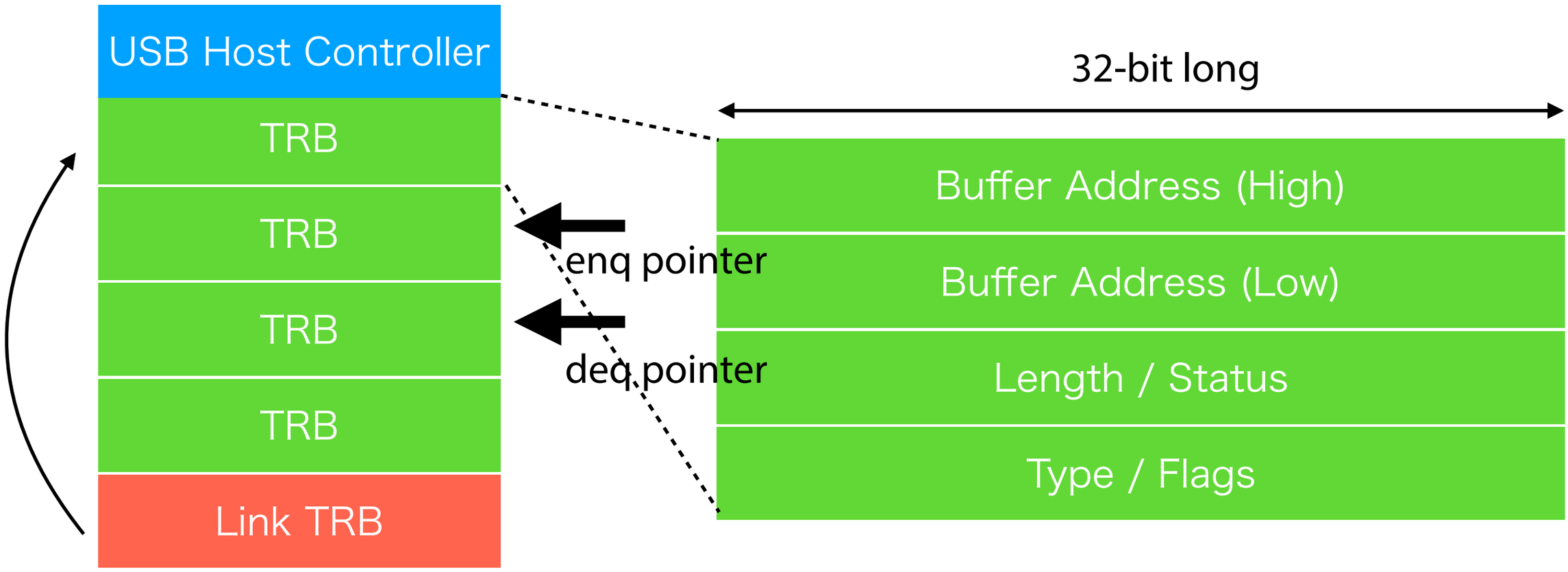


- xHCI uses ring buffers of TRBs (Transfer Request Block)
- Data on a TRB will be transferred to another end by the controllers.
- Multiple virtual serial communications are managed by EPs (End Point)

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

USB Host/Device Controller Basics

- **TRB and ring structure**



- A 16-byte TRB for transfer holds a pointer
  - Normal TRB type is used to specify data transfer
  - Link TRB type can point another TRB as the next one
    - A segmented TRB buffer helps when memory is non-contiguous

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Functions of USB DbC

- A virtual "device-side" controller with the minimal functionality on one of the ports on "host-side" controller:
    - Two pipes: IN and OUT
    - SuperSpeed (5Gbps) at least.
    - The max size of USB packet is 1024 bytes
    - **The host controller does not see the port after initialization**

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Functions of USB DbC

- A virtual "device-side" controller with the minimal functionality on one of the ports on "host-side" controller:
    - Two pipes: IN and OUT
    - SuperSpeed (5Gbps) at least.
    - The max size of USB packet is 1024 bytes
    - **The host controller does not see the port after initialization**

- **No full USB stack is required**
    - After specifying addresses for TRB ring buffers of the two pipes, what you have to do is to place your data into the ring buffer (or read it).

- getchar()/putchar() will be more than "`inb 0x3f8` + offset", but writing/reading the TRB ring is still simple

- DbC is designed as a transport for more sophisticated debug feature, such as JTAG and Intel DCI (exposing processor internal states and memory region)

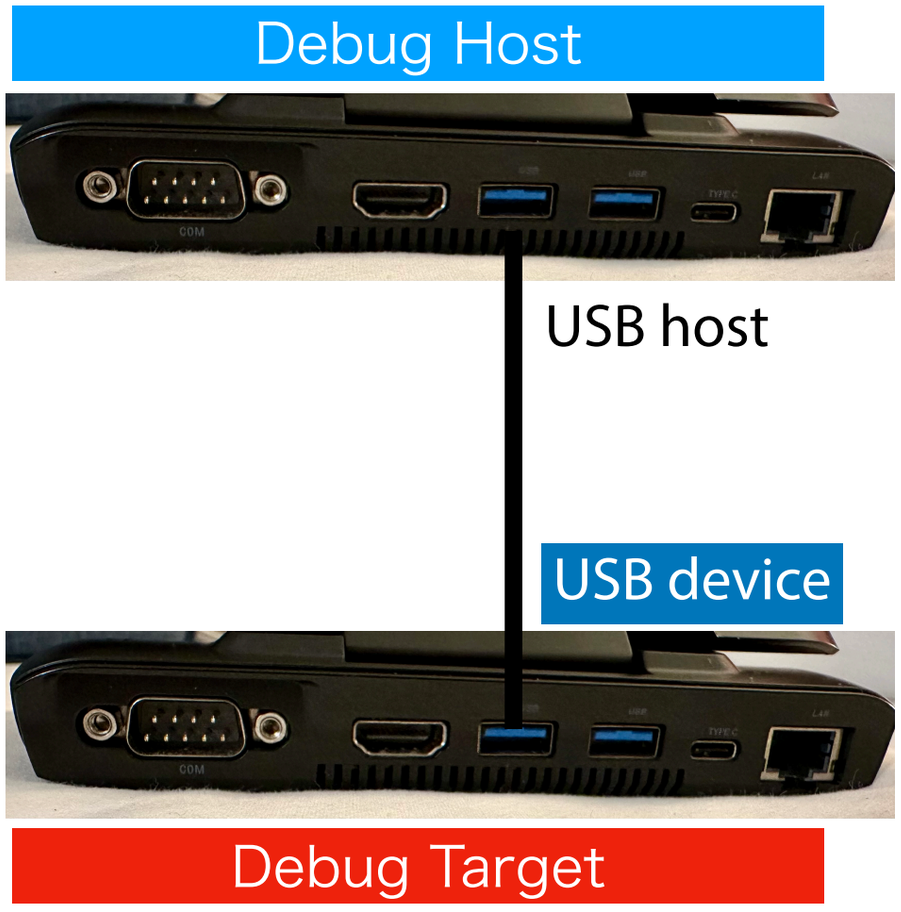
Use Cases and Security Concerns

- Just like a legacy serial port:
  - Console login access to headless servers
  - DDB access
  - Remote GDB
- There are a lot of "X over serial line", such as file transfer, IP communication, and etc.

- Safer (in terms of security) than solutions using Firewire or Thunderbolt, which exports access to bus and memory. Same as a serial port at all.

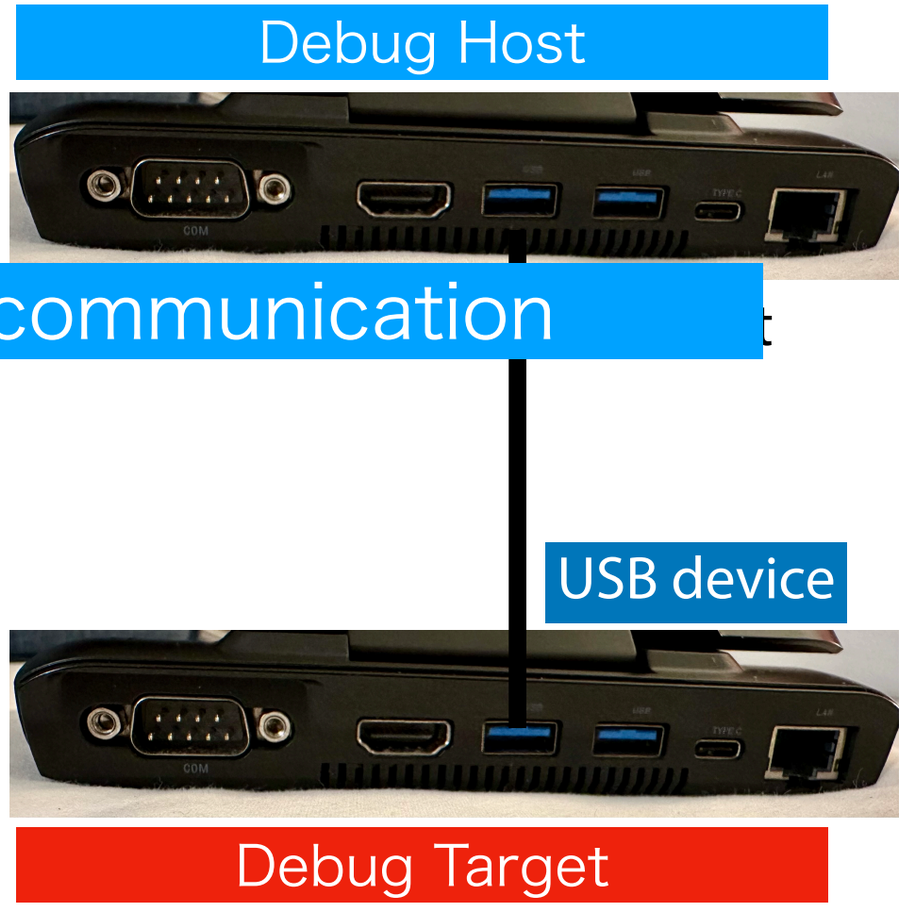
Software Components for DbC

- **On the Debug Host**
  - A normal USB3 stack is sufficient. No DbC required.
  - A client driver is required. This is because the USB device has USB Debug Class (0xdc in the bInterfaceClass field)



Software Components for DbC

- **On the Debug Host**
  - A normal USB3 stack is sufficient. No DbC required.
  - udbc(4) driver for simple serial communication because the USB device has USB Debug Class (0xdc in the bInterfaceClass field)

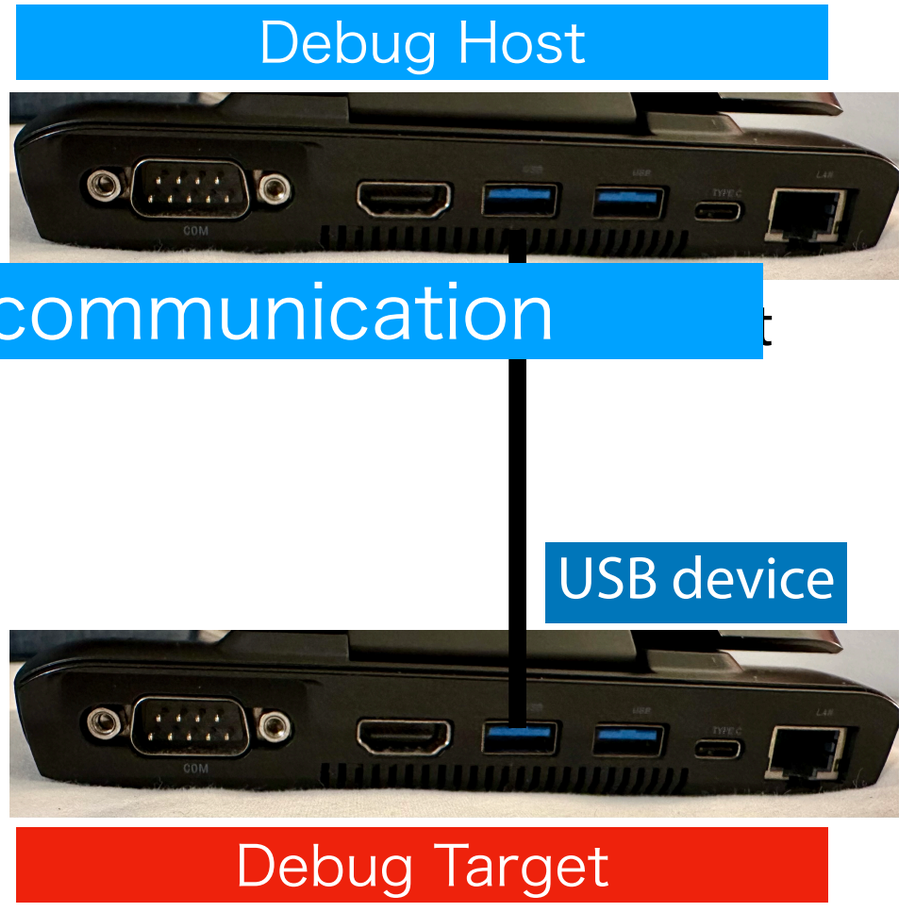


https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Software Components for DbC

- **On the Debug Host**
  - A normal USB3 stack is sufficient. No DbC required.
  - udbc(4) driver for simple serial communication because the USB device has USB Debug Class (0xdc in the bInterfaceClass field)

- **On the Debug Target**
  - Activation of DbC is required.
  - DbC has two endpoints (IN and OUT) for bulk transfer
  - TRB ring buffers for IN and OUT must be allocated in memory (DMA will handle them)

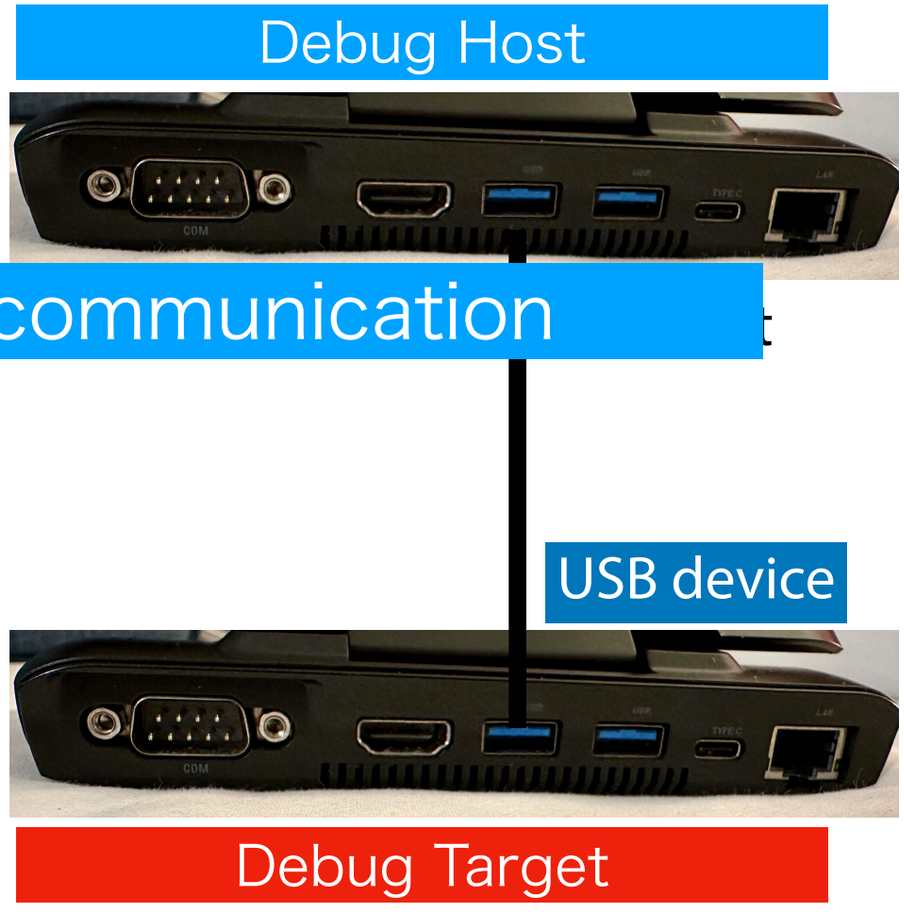


https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Software Components for DbC

- **On the Debug Host**
  - A normal USB3 stack is sufficient. No DbC required.
  - udbc(4) driver for simple serial communication
  - because the USB device has USB Debug Class (0xdc in the bInterfaceClass field)

- **On the Debug Target**
  - Activation of DbC is required.
  - DbC has two endpoints (IN and OUT) for bulk transfer
  - Console backend in the loader and the kernel
  - TRB ring buffers for IN and OUT must be allocated in memory (DMA will handle them)

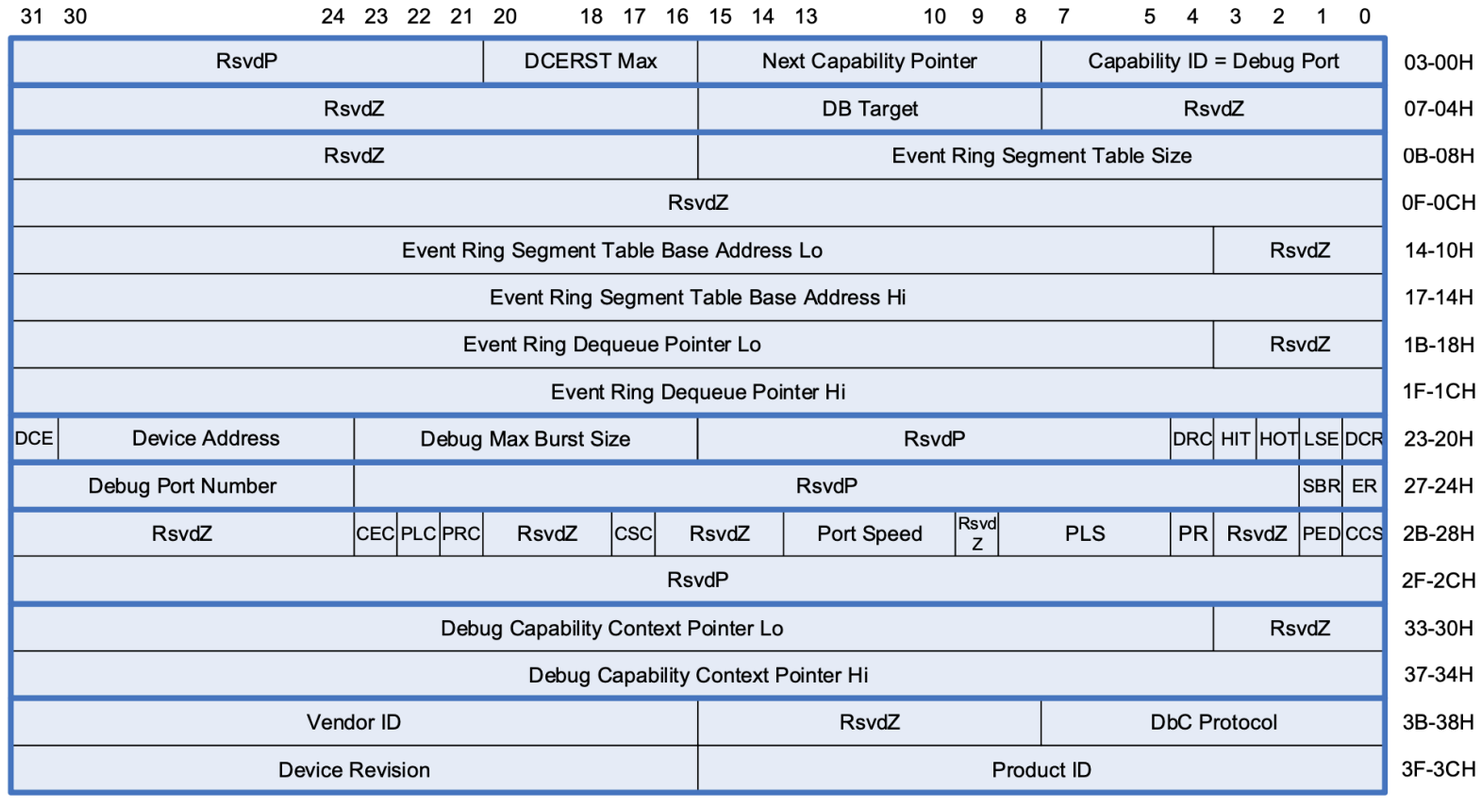


https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Enabling DbC on Target

- Extended capability ID=0x0a in PCI configuration space
- Configure the DbC register:
  - Three TRB ring for Tx, Rx, and event handling
  - TRB memory region for DMA

Figure 7-9: Debug Capability Register Layout



https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

FreeBSD Console Framework

- In loader:

stand/efi/loader/conf.c

```
struct console *consoles[] = {
        &efi_console,
        &eficom,
        &udb_console,
        &comconsole,
        &nullconsole,
        &spinconsole,
       NULL
};
```

```
OK set console=udb,efi
```

stand/efi/loader/usb_dbc.c

```
struct console udb_console = {
        .c_name = "udb",
        .c_desc = "USB DbC serial port",
        .c_flags = 0,
        .c_probe = udb_probe,
        .c_init = udb_init,
        .c_out = udb_putc,
        .c_in = udb_getc,
        .c_ready = udb_ischar
};
```

- udb_probe() -> udb_init(). c_in/c_out methods are used.

FreeBSD Console Framework

- In kernel:

sys/dev/usb/controller/xhci.c

```
% conscontrol
Configured: ttyv0,udbcons,gdb
 Available: udbcons,ttyv0,gdb
    Muting: off
```

```
static cn_probe_t xhci_debug_cnprobe;
static cn_init_t xhci_debug_cninit;
static cn_term_t xhci_debug_cnterm;
static cn_getc_t xhci_debug_cngetc;
static cn_putc_t xhci_debug_cnputc;
static cn_grab_t xhci_debug_cngrab;
static cn_ungrab_t xhci_debug_cnungrab;

const struct consdev_ops xhci_debug_cnops = {
        .cn_probe       = xhci_debug_cnprobe,
        .cn_init        = xhci_debug_cninit,
        .cn_term        = xhci_debug_cnterm,
        .cn_getc        = xhci_debug_cngetc,
        .cn_putc        = xhci_debug_cnputc,
        .cn_grab        = xhci_debug_cngrab,
        .cn_ungrab      = xhci_debug_cnungrab,
};

CONSOLE_DRIVER(xhci_debug);
```

- cninit() (kern/kern_cons.c) is called in MD init routines and handle probing. cn_getc() and cn_putc() are used.

FreeBSD Console Framework

- In kernel:

sys/dev/usb/controller/xhci.c

```
% ls -al /dev/udbcons
crw------- 1 root wheel 0x33 \
    May 31 23:00 /dev/udbcons
```

```
static tsw_outwakeup_t xhci_debug_tty_outwakeup;

static struct ttydevsw xhci_debug_ttydevsw = {
        .tsw_flags      = TF_NOPREFIX,
        .tsw_outwakeup  =
xhci_debug_tty_outwakeup,
};

...

cons->tp = tty_alloc(&xhci_debug_ttydevsw,
cons);
tty_makedev(cons->tp, NULL, "%s", UDBCONS_NAME);
        tty_init_console(cons->tp, 0);
....

callout_init(&cons->callout, 1);
callout_reset(&cons->callout, cons->polltime,
      xhci_debug_timeout, cons->tp);
```

- /dev/udbcons is another entry point used by getty(8). tty_makedev() is called during the DbC initialization. The callouts are for polling of data arrival.

Memory region for TRB

- Both loader and kernel need to access the same TRB rings.
- The memory region are initialized using UEFI service in loader.efi:

```
status = pci->Map(pci, EfiPciIoOperationBusMasterCommonBuffer,
(void *)virt, &mapped, &paddr, &mapping);
```

- Is this mapping valid (ore possible to reuse) even after kernel loaded? The current code ignores and reconfigures it completely.
- The XHCI register has physical address configured by the loader and the kernel can read later.

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Physical Setup

- **A-to-A USB3 Cable between the two**
  - On the debug target, **one of the ports on Root Hub** will become USB device.
  - This means that you have to find ports associated with the Root Hub. **Any USB 2.0 ports do not work.**

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Physical Setup

- **A-to-A USB3 Cable between the two**
  - On the debug target, **one of the ports on Root Hub** will become USB device.
  - This means that you have to find ports associated with the Root Hub. **Any USB 2.0 ports do not work.**

- **A-A cross cable + A-A extension + A-C adapter + Beastie charm for 30 USD/40 CAD here.** 6 sets are available. Catch me if you are interested in them.

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Demo and Call for Test

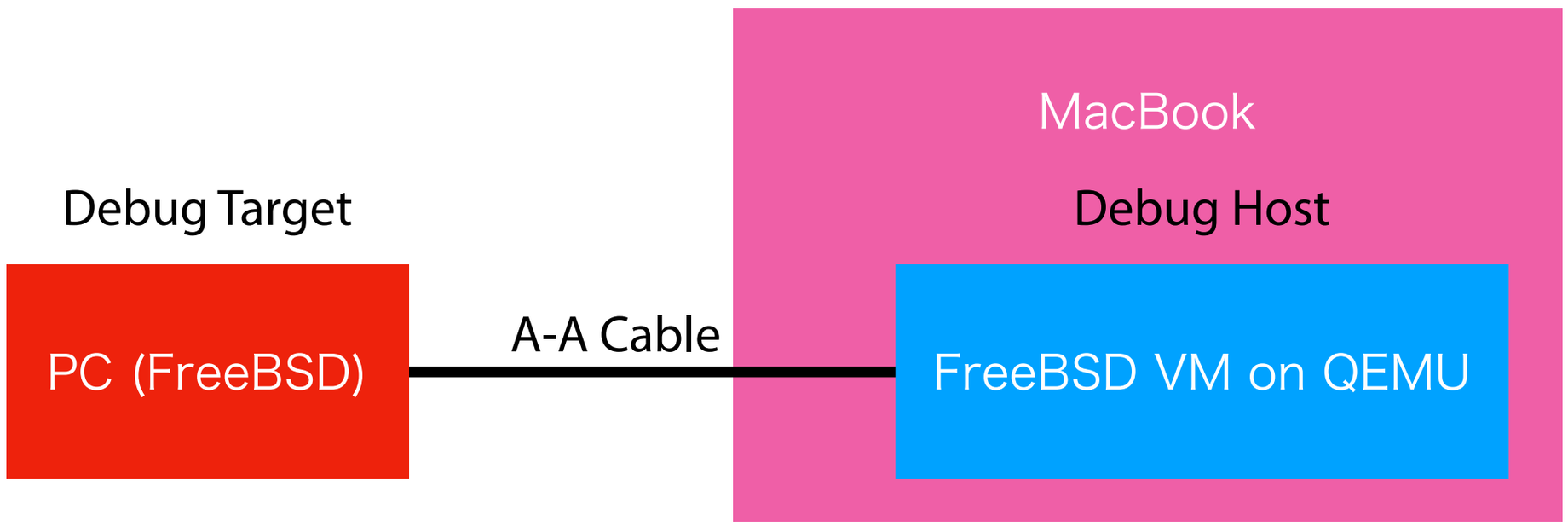
- https://people.allbsd.org/~hrs/FreeBSD/udbc/20240531/
  - udbc-kernel-14-20240531.tar.gz
    - 14-stable kernel, including udbc(4) driver
  - udbc-loader-14-20240531.tar.gz
    - 14-stable loader (you need to use UEFI)
  - udbc-src-14-20240531.tar.gz
    - Source, still work in progress
  - udbc-patch-14-58be9203662d0dd2072002ceb9a78c81bb64d3b3.20240531.diff
    - Patch set against the branch point (58be9203...)

- **A unidirectional communication test from Target to Host**
  - See README.udbc

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Demo and Call for Test

- **Demo**



https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

TODOs and Future Work

- **DONE**
  - Tested using 20 different machines and there seems good availability. Bidirectional communication is also confirmed.

  - udbc(4): almost commit ready
    - udbconsole backend in UEFI loader
      - Needs to reduce duplication with in-kernel xhci(4) driver

    - /dev/udbcons except for putc()/getc() handler
      - TRB memory region issue

TODOs and Future Work

- **In progress and plan to complete in the next three weeks:**

  - Import working code for UEFI loader and kernel:
    - Does enabling it by default make sense? If you do not connect A-A cable, nothing happened. One of the USB ports can be used for DbC only when the cable is attached.
    - USB port or bus sometimes stall. Some countermeasures must be implemented.

- **Need help:**

  - More testing for USB-C connection. Flipping the mode from host to device sometimes requires another insertion/removal cycle.

  - Support in non-UEFI loader.

https://people.allbsd.org/~hrs/FreeBSD/udbc/sato-20240531-usbdbc.pdf

Summary

- USB DbC is a feature to change one of the ports on a USB host for a USB device.

- The USB device has two EPs. You can receive/send any data over the IN and OUT pipes (virtual serial channels).

- A-A USB3 cable is required (again, catch me you want one). 5Gbps speed is supported at least.

- I need more information about device compatibility. Please try the test and let me know your xHCI device id and if it works or not.

Questions/Comments/Suggestions?

Please send your feedback to hrs@FreeBSD.org