arm

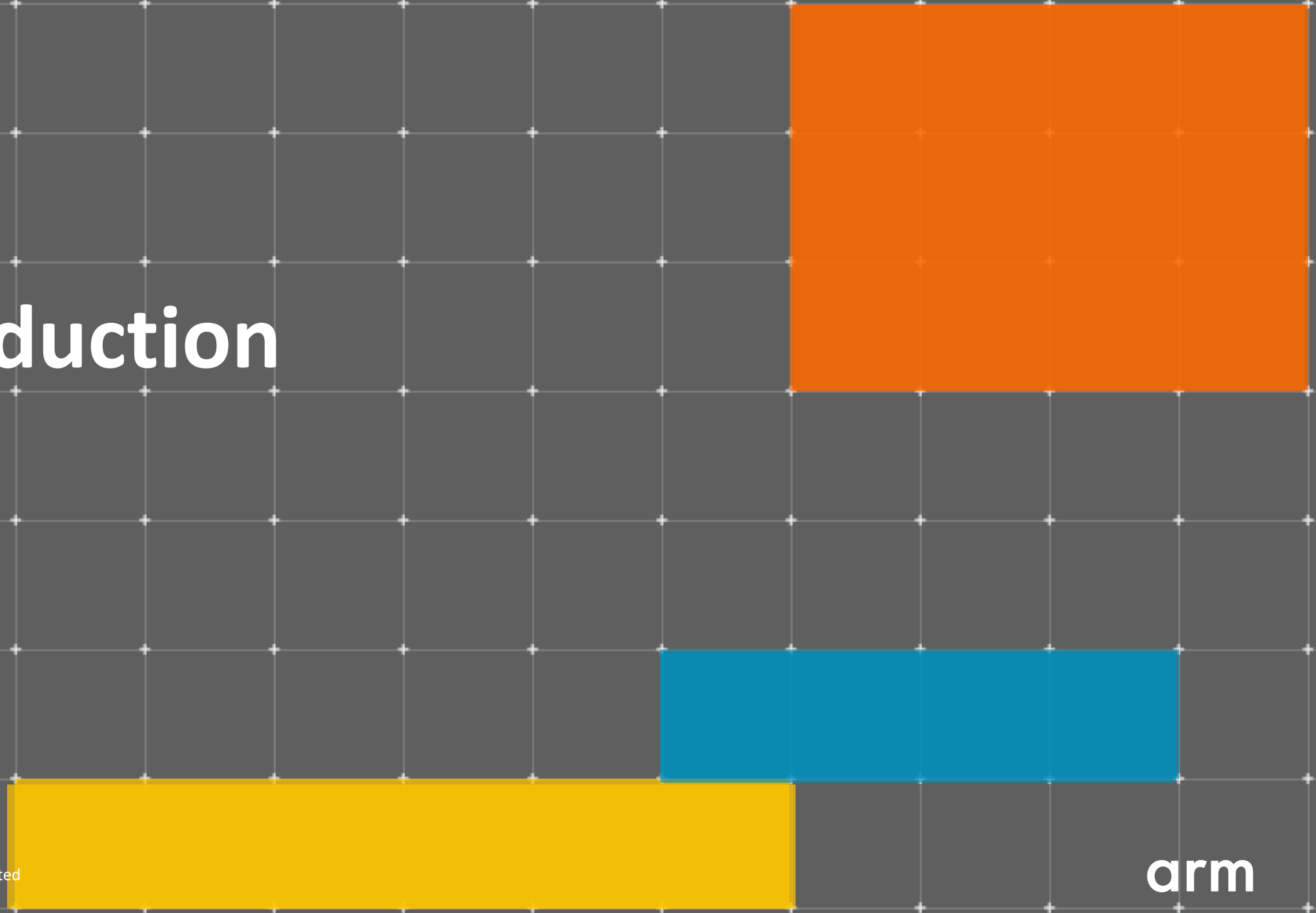
Mbed Silicon Partner Workshop

USB Improvements

28th of March 2018

Paris, France

Introduction



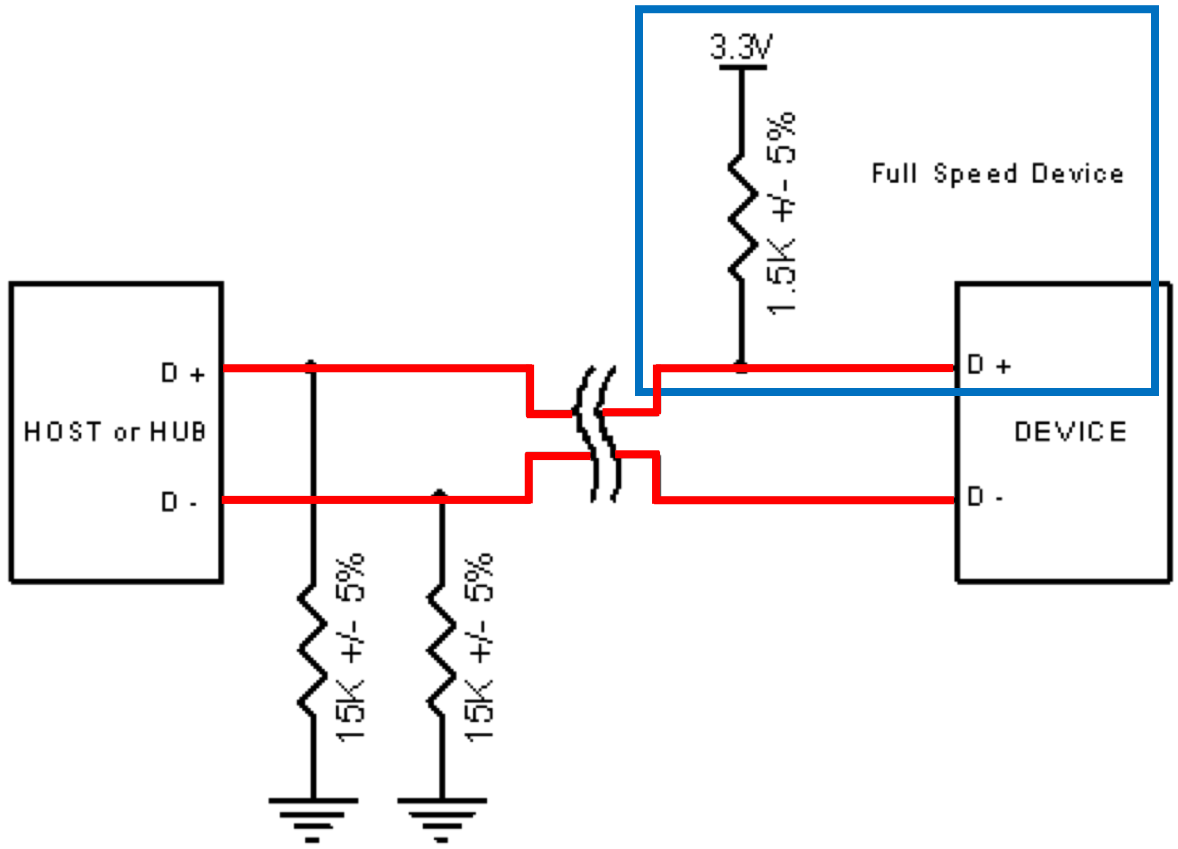
arm

Terms in the context of this presentation

- USB device – mbed board which performs some USB function
- USB host – computer with a USB host controller
- USB phy – Part of the chip providing physical access to the USB lines
- Endpoint – USB data stream between a USB host and a USB device
- Interface – Group of zero or more endpoint serving a dedicated purpose
- Descriptor – Information reported by a USB device which describes it
- Enumeration – Process by which a USB host activates a USB device

USB 101 – Electrical

- Data sent on a pair of wires - D+ D-
- Pullup on D+ (full speed) or D- (low speed) is used by USB host to detect USB device
- USB speeds
  - High speed – 480 Mbit/s
  - Full speed - 12 Mbit/s
  - Low speed – 1.5 Mbit/s



USB 101 – Packets

Token packets – These are used to start transfers. Contain a 7-bit address, and 4 bit endpoint address. Examples include Setup, In and Out packets.

- Setup – Used to start control transfer
- In – Request data from USB device
- Out – Send data to USB device

Data packets – These are used to carry data. Examples are Data0 and Data1

Handshake packets – These are used to report a status

- Ack – Data received
- Nak – I'm not ready for your data yet
- Stall – Command failed or something went wrong

arm

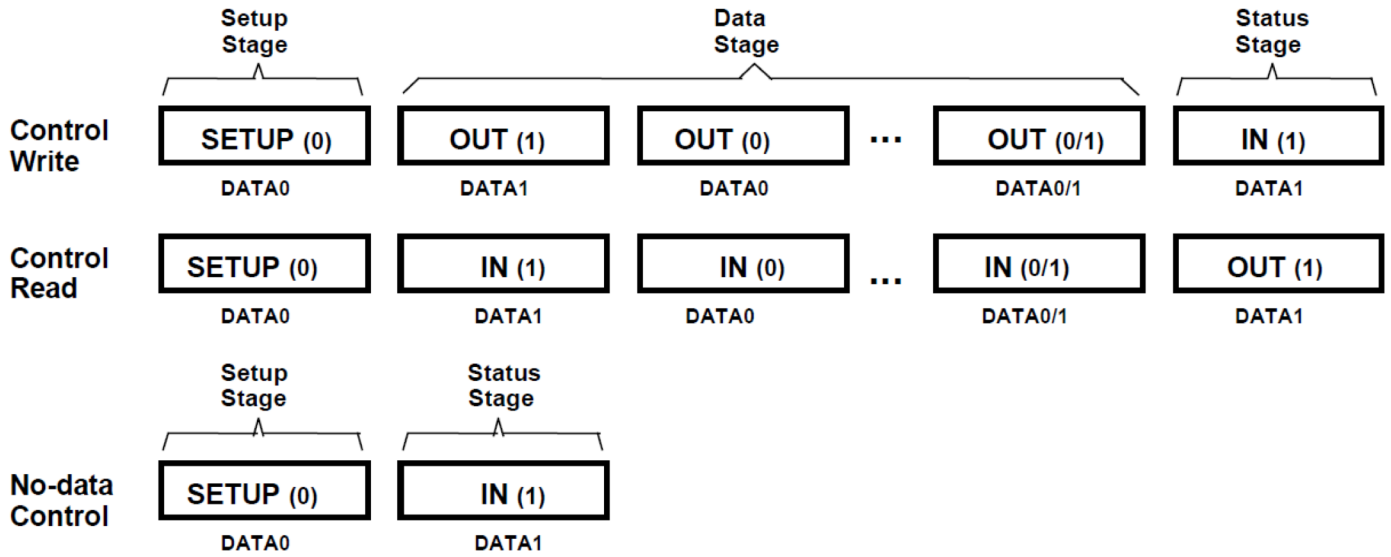
USB 101 - Protocols

The USB protocol layer defines how USB packets are used to transfer data. USB has 4 primary protocols:

- Control – Error free delivery and flow control for device management and data transfers
- Isochronous – Guaranteed/Fixed bandwidth, no error recovery
- Bulk – Error free delivery and flow control for data transfers
- Interrupt – Guaranteed/Fixed bandwidth, error free delivery and flow control

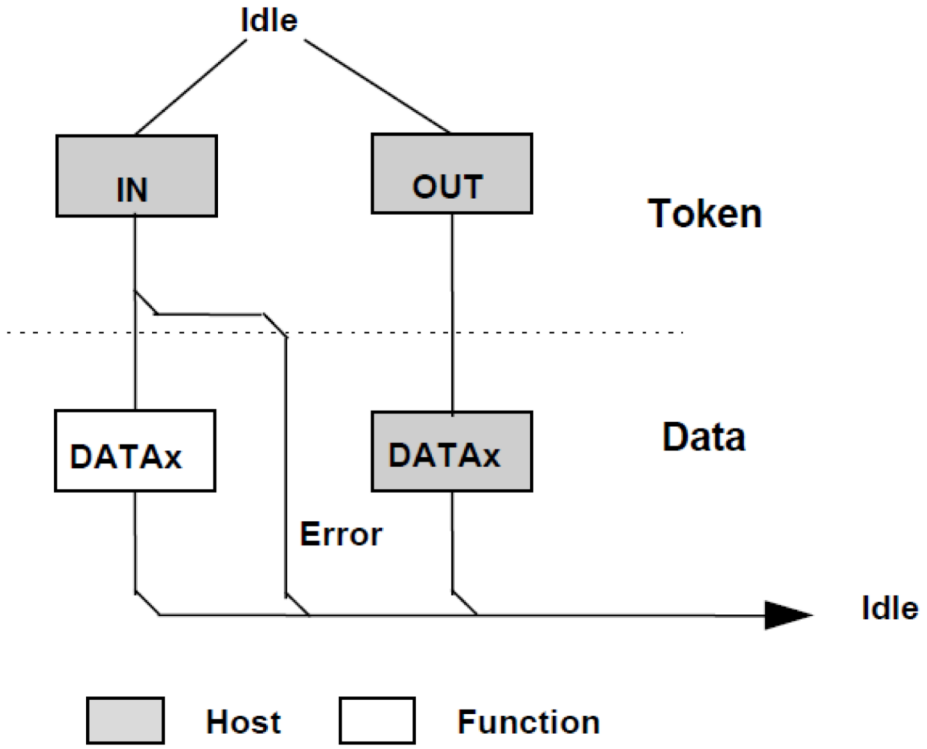
USB 101 – Protocols - Control

- Used to manage the device and can be used by all interfaces
- Can send or receive data
- Consists of a setup stage, an optional data stage and a status stage
- Flow control by sending Nak packets



USB 101 – Protocols - Isochronous

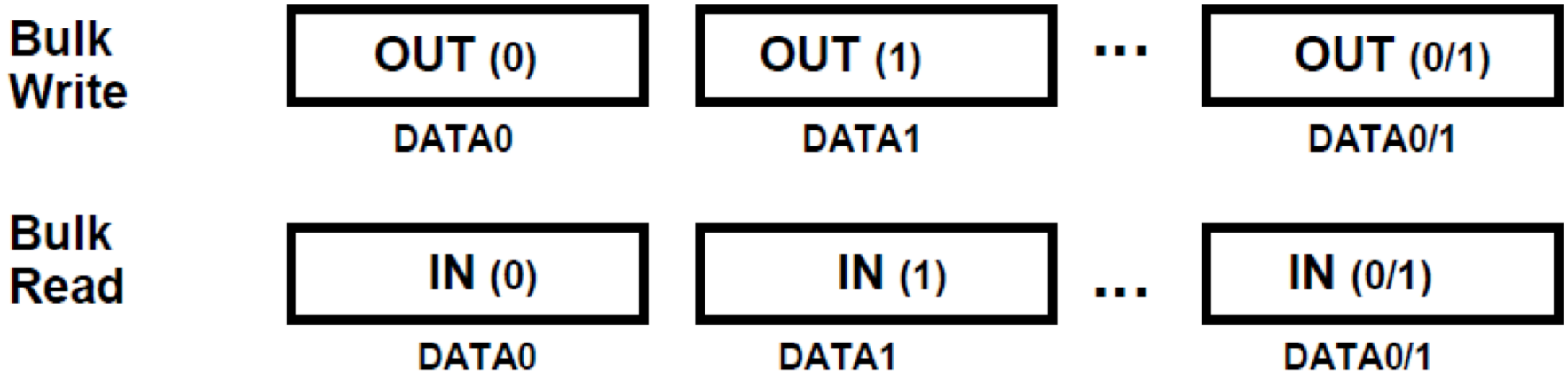
- Typical use case is for sending audio or video where it is better to drop a frame rather than lag or glitch
- Starts with an In or Out token
- No status stage
- Corrupt packets are not retried
- Fixed bandwidth is reserved by USB host



arm

USB 101 – Protocols - Bulk

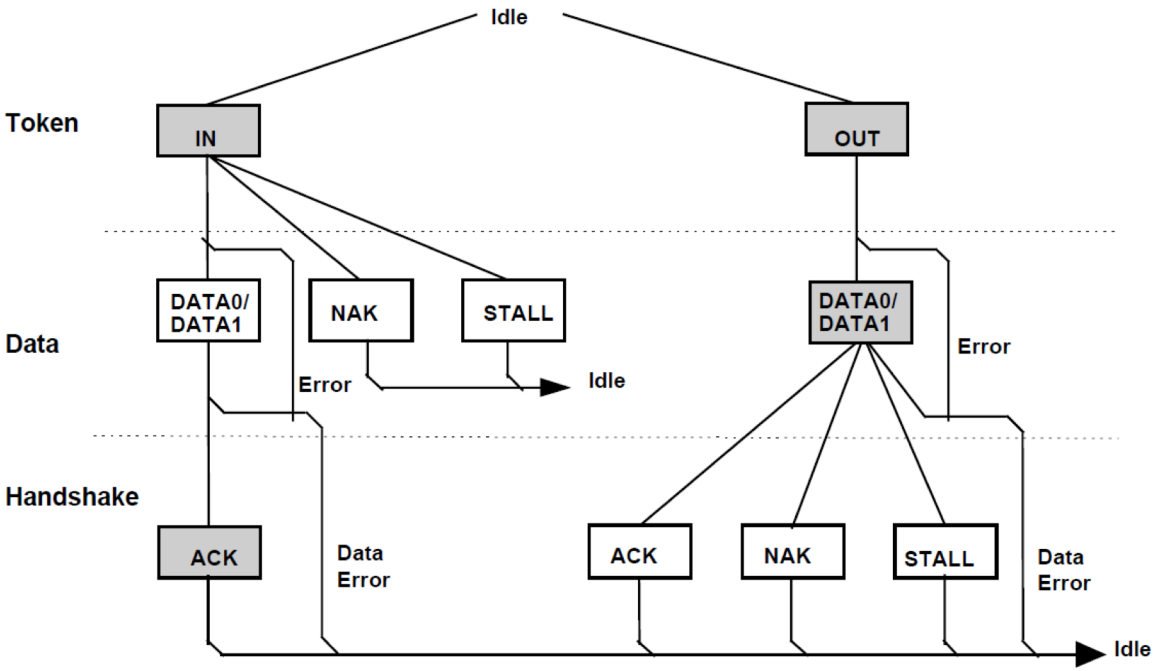
- Typically used in mass storage devices where data transfers must be error free
- Starts with an In or Out token, followed by data followed by status
- Application layer can report errors by stalling the endpoint
- Flow control by sending Nak packets



arm

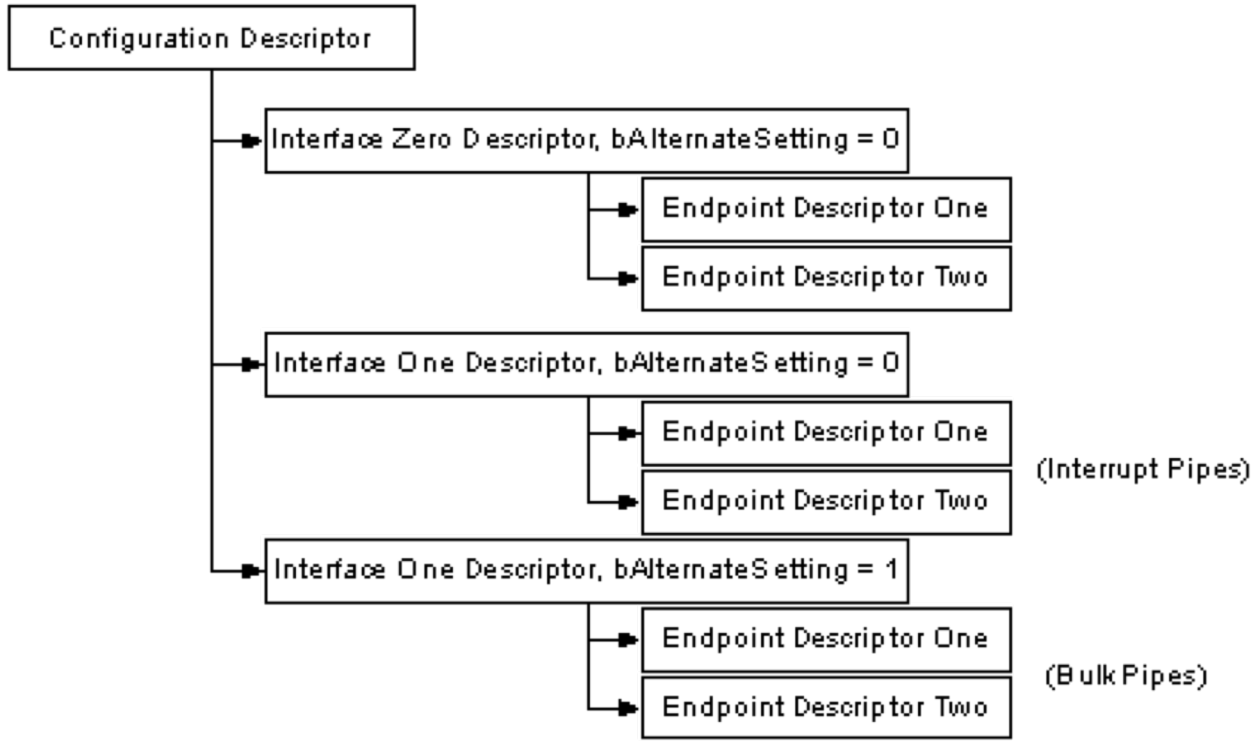
USB 101 – Protocols - Interrupt

- Typical use case is for sending data from a keyboard or mouse where responsiveness and reliability are important
- Starts with an In or Out token
- Corrupt packets are retried
- Fixed bandwidth is reserved by USB host



USB 101 - Descriptors

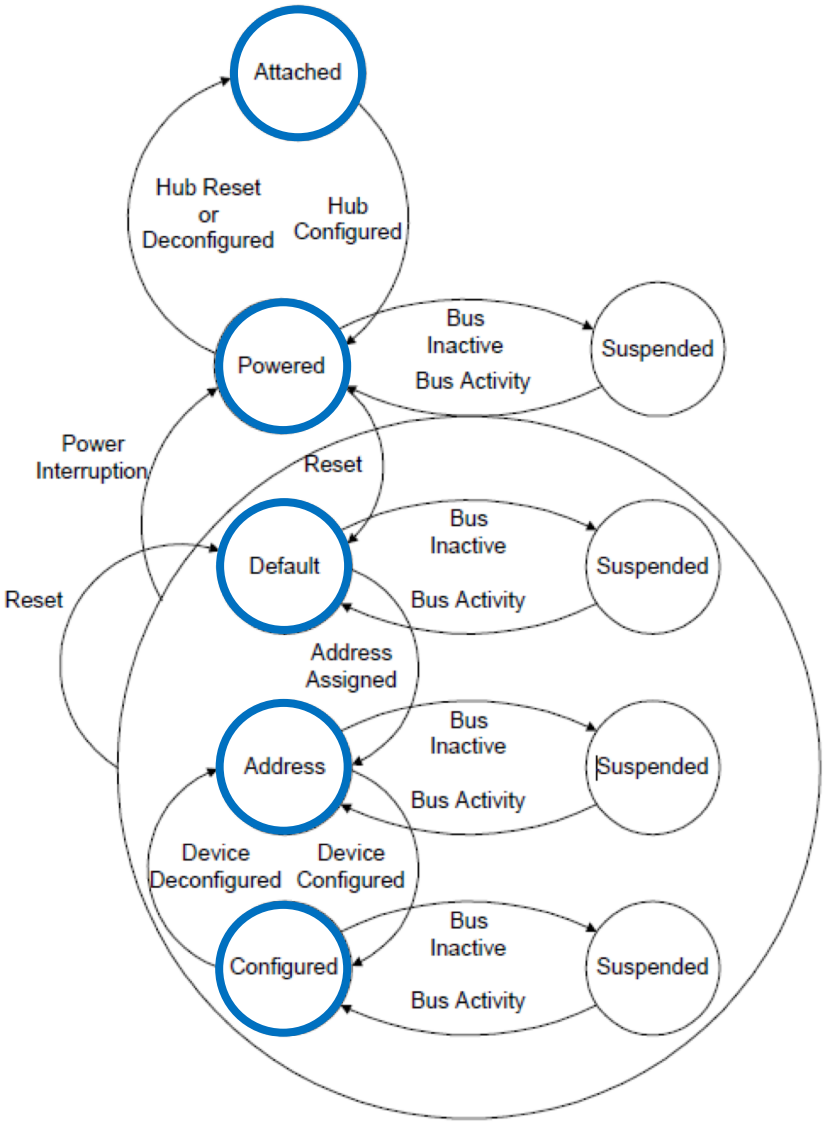
- Used to report device capabilities and info
- Sent over control endpoint 0
- Device descriptor contains general info like vendor id and product id
- Configuration descriptor contains all interfaces and endpoints the device supports
- String descriptors are human readable Unicode data



arm

USB 101 – USB states

- Attached – device is connected to USB host but unpowered
- Powered – device is connected to USB host and powered
- Default – USB host has reset the device
- Addressed – USB host has given USB device an address
- Configured – USB host has selected a configuration of the USB device. All interfaces of the configuration are active

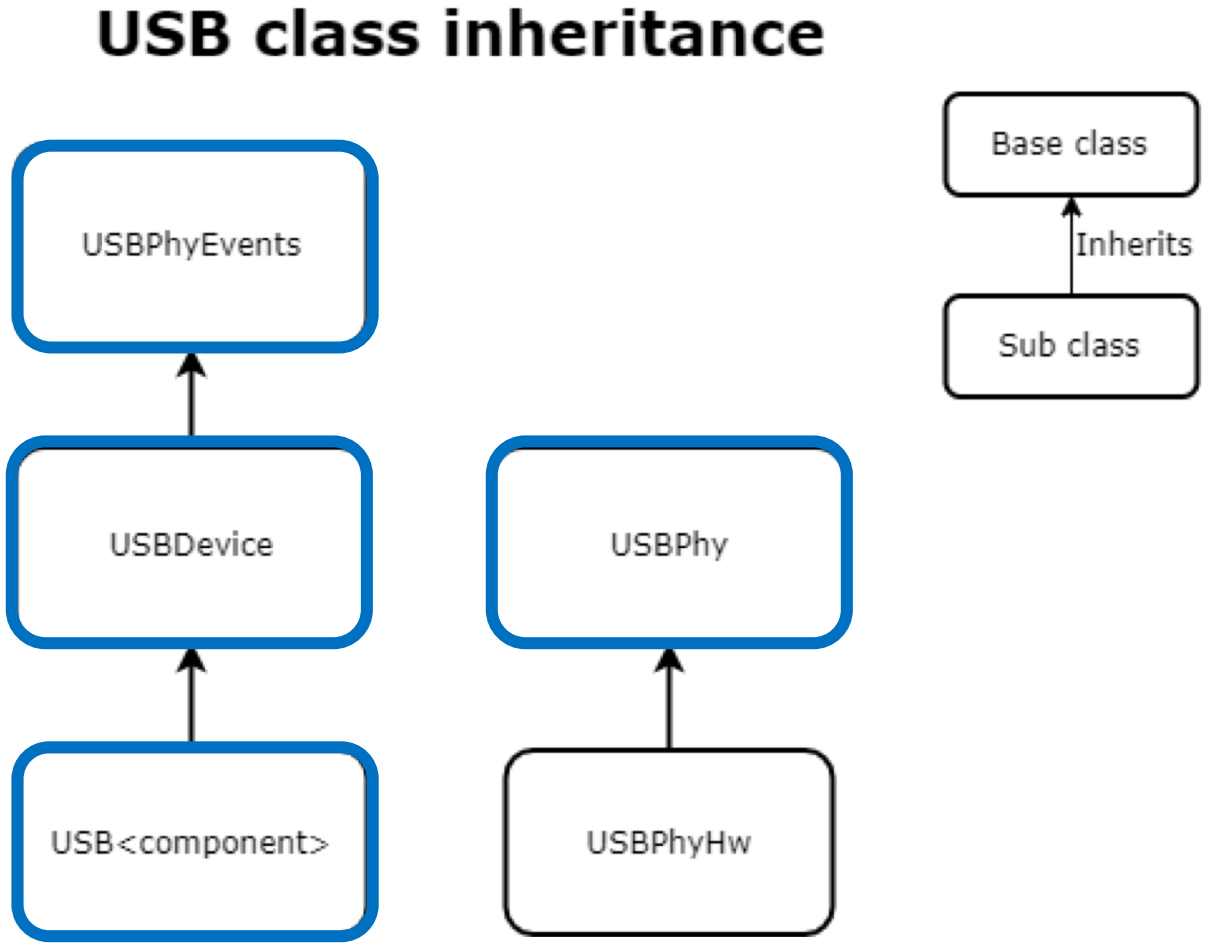


Mbed OS USB device

arm

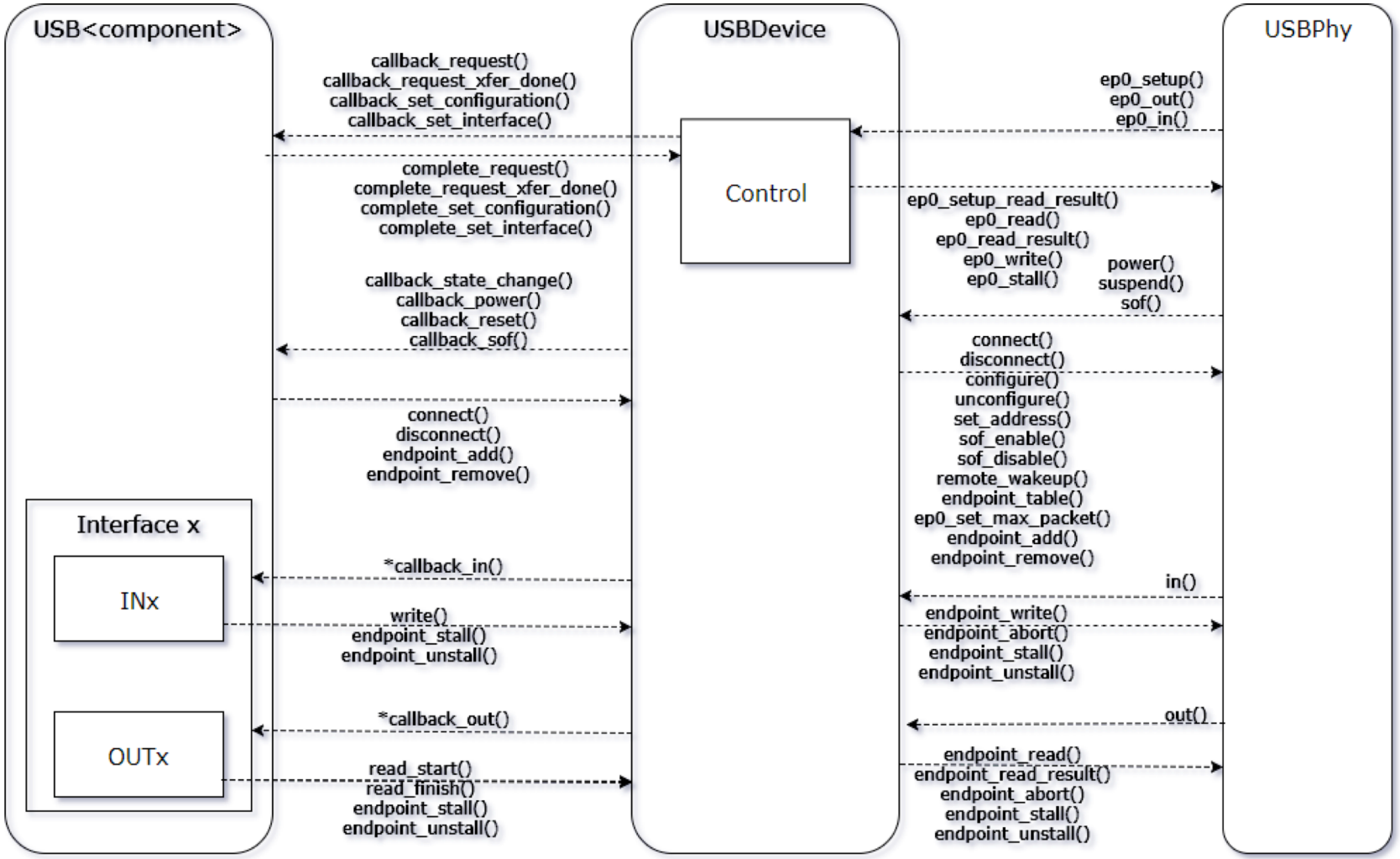
USB class hierarchy

- `USBPhy` - provides raw access to USB. The target should implement this to support USB
- `USBPhyEvents` - Interface class of all events the `USBPhy` can send
- `USBDevice` - core of USB stack. It manages the `USBPhy`
- `USB<component>` - class providing end-user functionality, such as MSD, CDC, HID



arm

USB classes interactions



USB device synchronization

- The class `USBDevice` is safe to use from interrupt context
- All access to `USBDevice` is serialized with a lock implemented as a critical section
- USB components can make use of this lock but it is not required

```
{
    lock();

    AsyncOp wait_op(NULL);
    wait_op.start(&_connected_list);
    if (_terminal_connected) {
        wait_op.complete();
    }

    unlock();
    wait_op.wait();
}
```

```
void USBCDC::callback_state_change(DeviceState new_state)
{
    assert_locked();

    if (new_state != Configured) {
        _change_terminal_connected(false);
    }
}
```


arm

USB device states

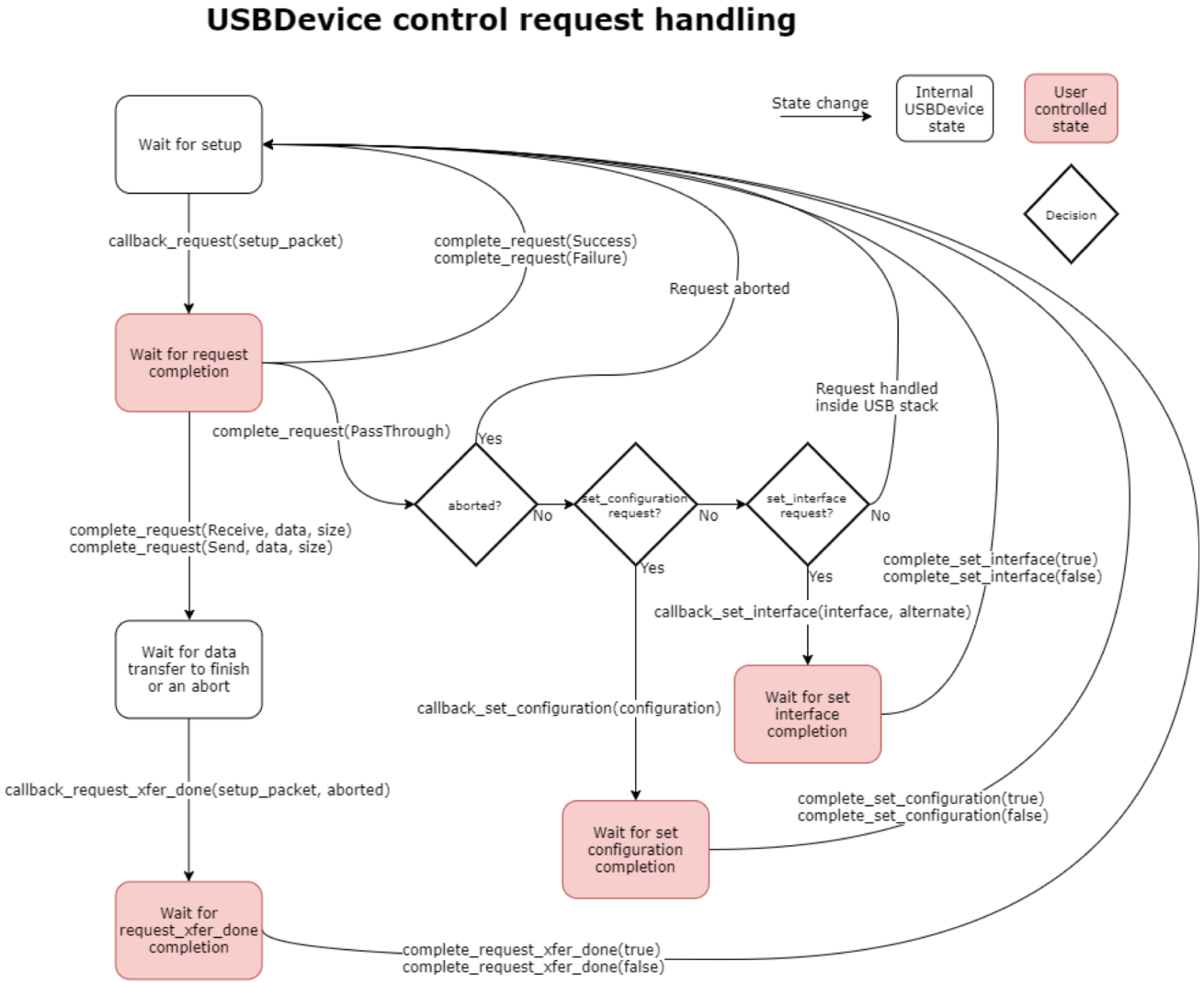
- `USBDevice` has a state machine which matches the 2.0 specification
- Each state adds increasing functionality with Configured having the most
- At any time the device could enter a lower state due to a reset or disconnect
- USB component can delay entry into configured state by only responding to control request when ready

State	Functionality
Attached	Power events
Powered	+Reset events
Default	+Control endpoint 0 active
Address	No new functionality
Configured	+All enabled endpoints are functional

arm

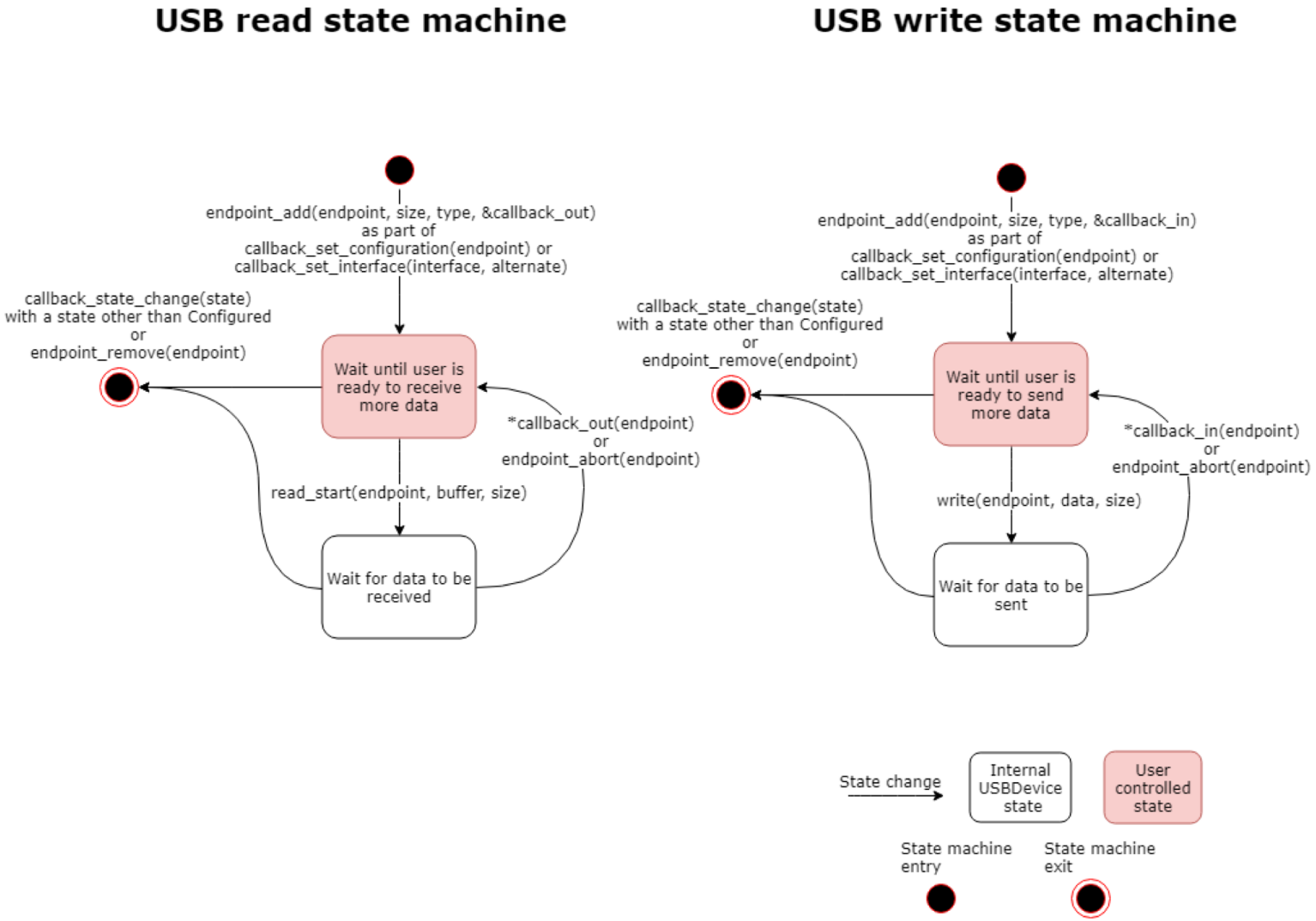
Control transfer handling

- Control transfer can be throttled without delaying other endpoints
- Status can be returned after setup stage or after the data stage



Endpoint handling

- Reads and writes are explicitly started
- Caller supplies the buffer for read and write operations
- Buffer must remain unchanged until the transfer is completed or aborted

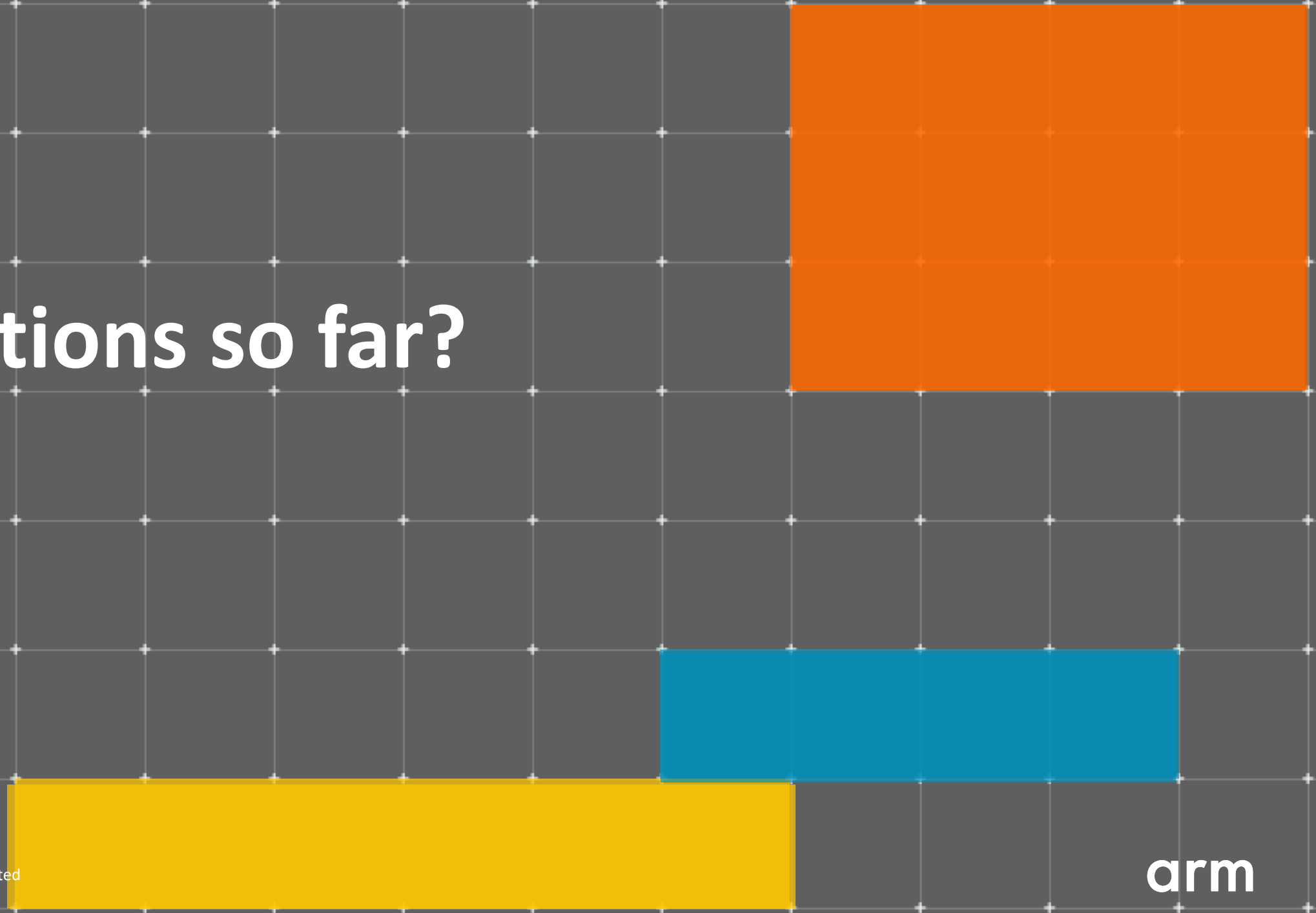


Endpoint configuration

- Mbed boards may have different numbers of endpoints, functionality can be limited for specific endpoints or limited by USB ram
- To make a portable USB component the endpoints you are using must be looked up at runtime
- `EndpointResolver` utility class makes it easy to request endpoints and check that the configuration fits within the devices USB ram.

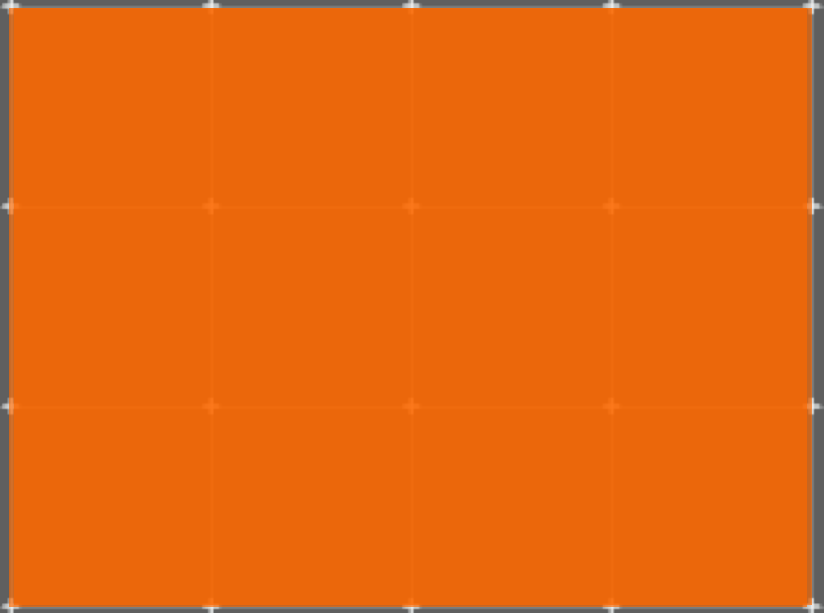
```
EndpointResolver resolver(endpoint_table());
resolver.endpoint_ctrl(CDC_MAX_PACKET_SIZE);
bulk_in = resolver.endpoint_in(USB_EP_TYPE_BULK,
CDC_MAX_PACKET_SIZE);
bulk_out = resolver.endpoint_out(USB_EP_TYPE_BULK,
CDC_MAX_PACKET_SIZE);
int_in = resolver.endpoint_in(USB_EP_TYPE_INT,
CDC_MAX_PACKET_SIZE);
MBED_ASSERT(resolver.valid());
```

Questions so far?



arm

Porting a new USBPhy



arm

Porting

- Create and implement a subclass of `USBPhy`. A template for this can be found in the directory mbed-os/usb/device/targets/TARGET_Template.
- Add the function `USBPhy *get_usb_phy()` which returns an instance of your `USBPhyHw`
- Add the `USBDEVICE` label to `device_has` for your target in targets.json

```
USBPhy *get_usb_phy()
{
    static USBPhyHw usbphy;
    return &usbphy;
}
```

```
"LPC1768": {
        "inherits": ["LPCTarget"],
        "core": "Cortex-M3",
        "extra_labels": ["NXP", "LPC176X", "MBED_LPC1768"],
        "supported_toolchains": ["ARM", ..., "GCC_CR", "IAR"],
        "detect_code": ["1010"],
        "device_has": ["ANALOGIN", ..., "USBDEVICE"],
        "release_versions": ["2", "5"],
        "features": ["LWIP"],
        "device_name": "LPC1768",
        "bootloader_supported": true
    },
```


arm

USBPhy and USBPhyEvents API

USBPhy Doxygen

```
class USBPhy {
public:
    USBPhy() {};
    virtual ~USBPhy() {};
    virtual void init(USBPhyEvents *events);
    virtual void deinit();
    virtual bool powered();
    virtual void connect();
    virtual void disconnect();
    virtual void configure();
    virtual void unconfigure();
    virtual void sof_enable();
    virtual void sof_disable();
    virtual void set_address(uint8_t address);
    virtual void remote_wakeup();
    virtual const usb_ep_table_t* endpoint_table();
    virtual uint32_t ep0_set_max_packet(uint32_t max_packet);
    virtual void ep0_setup_read_result(uint8_t *buffer, uint32_t size);
    virtual void ep0_read(uint8_t *data, uint32_t size);
    virtual uint32_t ep0_read_result();
    virtual void ep0_write(uint8_t *buffer, uint32_t size);
    virtual void ep0_stall();
    virtual bool endpoint_add(usb_ep_t endpoint, uint32_t max_packet, usb_ep_type_t type);
    virtual void endpoint_remove(usb_ep_t endpoint);
    virtual void endpoint_stall(usb_ep_t endpoint);
    virtual void endpoint_unstall(usb_ep_t endpoint);
    virtual bool endpoint_read(usb_ep_t endpoint, uint8_t *data, uint32_t size);
    virtual uint32_t endpoint_read_result(usb_ep_t endpoint);
    virtual bool endpoint_write(usb_ep_t endpoint, uint8_t *data, uint32_t size);
    virtual void endpoint_abort(usb_ep_t endpoint);
    virtual void process();
};
```

USBPhyEvents Doxygen

```
class USBPhyEvents {
public:
    USBPhyEvents() {};
    virtual ~USBPhyEvents() {};
    virtual void reset();
    virtual void ep0_setup();
    virtual void ep0_out();
    virtual void ep0_in();
    virtual void power(bool powered);
    virtual void suspend(bool suspended);
    virtual void sof(int frame_number);
    virtual void out(usb_ep_t endpoint);
    virtual void in(usb_ep_t endpoint);
    virtual void start_process();
};
```


arm

USBPhy specification

Defined behavior

- You can use any endpoint configurations that fit in the parameters of the table returned by USBPhy::endpoint_table.
- You can use all endpoints in any valid endpoint configuration concurrently.
- The device supports use of at least one control, bulk, interrupt and isochronous in each direction at the same time - at least 8 endpoints.
- USBPhy supports all standard endpoint sizes (wMaxPacketSize).
- USBPhy can handle an interrupt latency of at least 100ms if the host PC is not performing a reset or setting the device's address.
- USBPhy only sends USBPhyEvents when it is in the initialized state.
- When unpowered, USBPhy only sends the USBPhyEvents::power event.
- On USB reset, all endpoints are removed except for endpoint 0.
- A call to USBPhy::ep0_write results in the call of USBPhyEvents::in when the PC reads the data unless a power loss, reset or a call to USBPhy::disconnect occurs first.
- A call to USBPhy::endpoint_write results in the call of USBPhyEvents::in when the PC reads the data unless a power loss, reset or a call to USBPhy::endpoint_abort occurs first.
- A call to USBPhy::endpoint_read results in the call of USBPhyEvents::out when the PC sends data unless a power loss, reset or a call to USBPhy::endpoint_abort occurs first.
- Endpoint 0 naks all transactions aside from setup packets until higher-level code calls one of USBPhy::ep0_read, USBPhy::ep0_write or USBPhy::ep0_stall.
- Endpoint 0 stall automatically clears on reception of a setup packet.

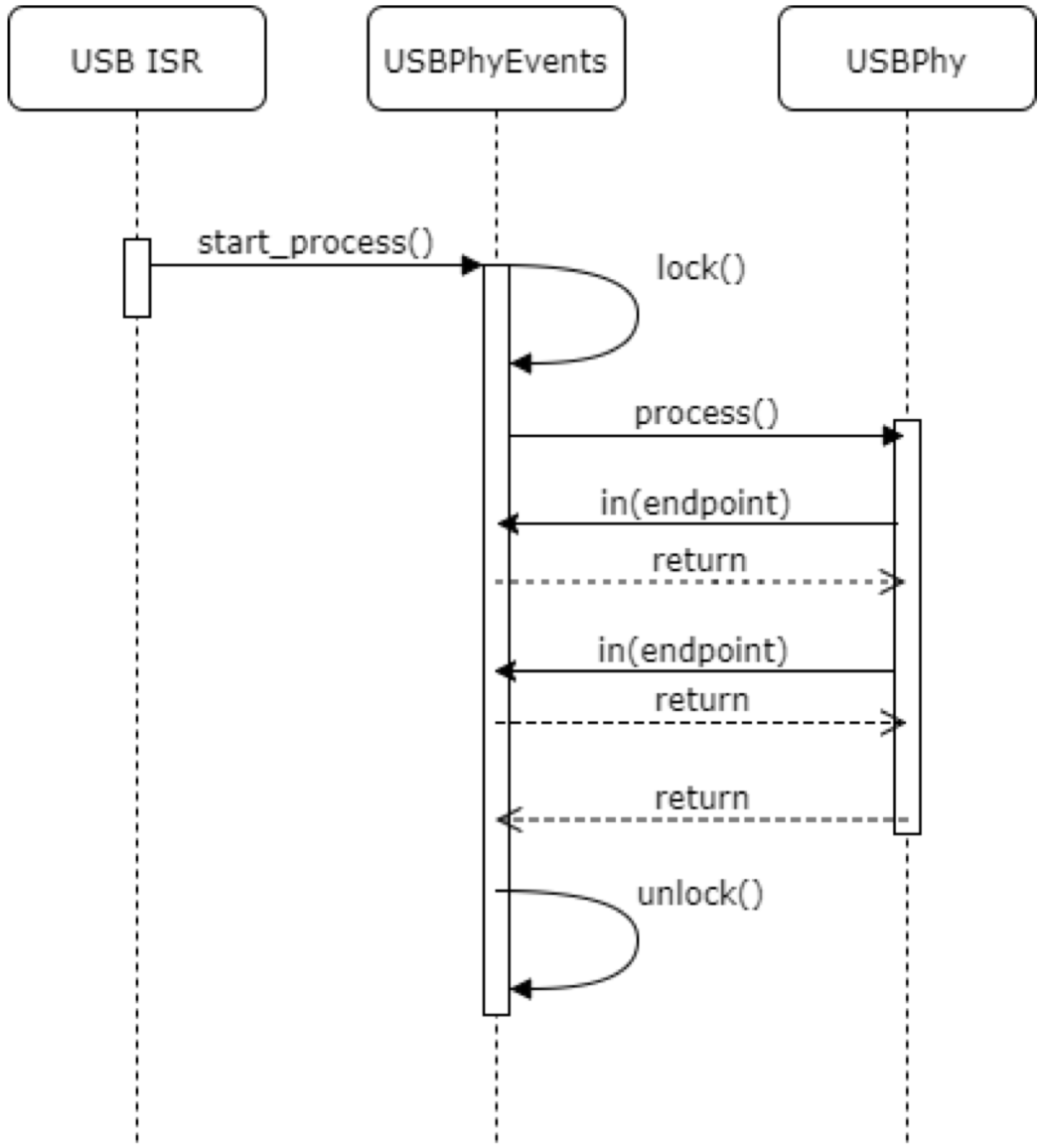
USBPhy specification

Undefined behavior

- Calling USBPhy::endpoint_add or USBPhy::endpoint_remove outside of the control requests SetInterface or SetConfiguration.
- Calling USBPhy::endpoint_remove on an endpoint that has an ongoing read or write operation. To avoid undefined behavior, you must abort ongoing operations with USBPhy::endpoint_abort.
- Devices behavior is undefined if latency is greater than 2ms when address is being set - see USB spec 9.2.6.3.
- Devices behavior is undefined if latency is greater than 10ms when a reset occurs - see USB spec 7.1.7.5.
- Calling any of the USBPhy::endpoint_* functions on endpoint 0.

USBPhy Synchronization

- USBPhy is synchronized externally, so no locking or critical section is needed
- Only event the USBPhy is allowed to call from an interrupt is **USBPhyEvents::start_process()** which leads to USBPhy::process() being called in a protected context
- All other USBPhyEvents must be sent inside the call to USBPhy::process()



USBPhy endpoint table

Allows supported configurations to be documented and tested

- Number of endpoints
- Endpoint functionality
- Endpoint RAM requirements
- Endpoint ram available
- Returned by endpoint_table()

```
static const usb_ep_table_t endpoint_table = {
    4096 - 32 * 4, // USB ram available
    {
        {USB_EP_ATTR_ALLOW_CTRL | USB_EP_ATTR_DIR_IN_AND_OUT, 1, 0},
        {USB_EP_ATTR_ALLOW_INT | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_BULK | USB_EP_ATTR_DIR_IN_AND_OUT, 2, 0},
        {USB_EP_ATTR_ALLOW_ISO | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_INT | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_BULK | USB_EP_ATTR_DIR_IN_AND_OUT, 2, 0},
        {USB_EP_ATTR_ALLOW_ISO | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_INT | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_BULK | USB_EP_ATTR_DIR_IN_AND_OUT, 2, 0},
        {USB_EP_ATTR_ALLOW_ISO | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_INT | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_BULK | USB_EP_ATTR_DIR_IN_AND_OUT, 2, 0},
        {USB_EP_ATTR_ALLOW_ISO | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_INT | USB_EP_ATTR_DIR_IN_AND_OUT,  1, 3},
        {USB_EP_ATTR_ALLOW_BULK | USB_EP_ATTR_DIR_IN_AND_OUT, 2, 0},
        {USB_EP_ATTR_ALLOW_BULK | USB_EP_ATTR_DIR_IN_AND_OUT, 2, 0}
    }
};
```

USBPhy endpoint table continued

Each entry in the endpoint table consists of {<endpoint type> | <endpoint direction>, <size per byte>, <size for endpoint>}

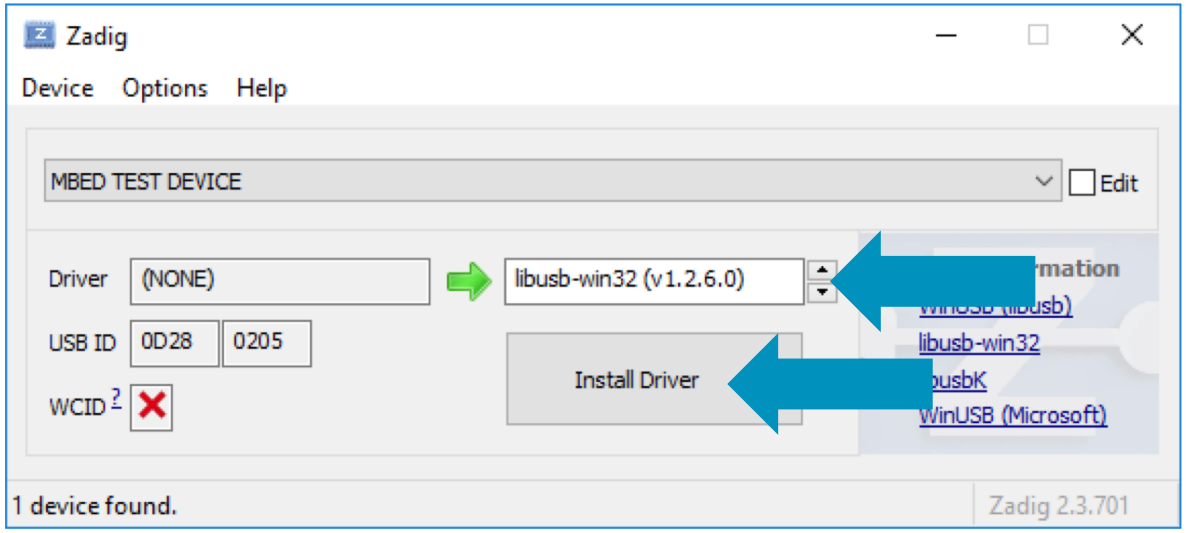
Endpoint type	Endpoint directions
USB_EP_ATTR_ALLOW_CTRL	USB_EP_ATTR_DIR_IN
USB_EP_ATTR_ALLOW_BULK	USB_EP_ATTR_DIR_OUT
USB_EP_ATTR_ALLOW_INT	USB_EP_ATTR_DIR_IN_OR_OUT
USB_EP_ATTR_ALLOW_ISO	USB_EP_ATTR_DIR_IN_AND_OUT

Testing Setup

- To run testing your mbed board must have both device and interface connected to the PC running the tests.
  - Be sure to plug in the extra USB cable.
- Use branch - `feature-hal-spec-usb-device`
- Install mbed-os requirements - `` `pip install -r requirements.txt` ``

Windows testing setup

- Download Zadig http://zadig.akeo.ie/



- Run

```
$ mbed test -t <toolchain> -m <target> -n "tests-usb_device-*"
```

Note: the tests will fail. This is expected.

- Open Zadig. MBED TEST DEVICE should be listed.
- Select **libusb-win32** for the driver. Click Install Driver.
- You are now setup for testing

arm

OSX testing setup

Install homebrew https://brew.sh/

Install libusb - `brew install libusb`

You are now setup for testing

arm

Run Tests

Run:

```
$ mbed test -t <toolchain> -m <target> -n "tests-usb_device-*"
```

```
Building library mbed-build (K64F, GCC_ARM)
...
+--------------+--------------+-------------------------------+-----------------------------------------+--------+--------+--------+--------------------+
| target       | platform_name | test suite                   | test case                               | passed | failed | result | elapsed_time (sec) |
+--------------+--------------+-------------------------------+-----------------------------------------+--------+--------+--------+--------------------+
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb control basic test                  | 1      | 0      | OK     | 0.64               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb control sizes test                  | 1      | 0      | OK     | 0.58               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb control stall test                  | 1      | 0      | OK     | 0.48               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb control stress test                 | 1      | 0      | OK     | 0.63               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb device reset test                   | 1      | 0      | OK     | 3.05               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb device suspend/resume test          | 1      | 0      | OK     | 6.35               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb repeated construction destruction test | 1      | 0      | OK     | 2.78               |
| K64F-GCC_ARM | K64F         | mbed-os-tests-usb_device-basic | usb soft reconnection test              | 1      | 0      | OK     | 3.01               |
+--------------+--------------+-------------------------------+-----------------------------------------+--------+--------+--------+--------------------+
mbedgt: test case results: 8 OK
mbedgt: completed in 50.17 sec
```

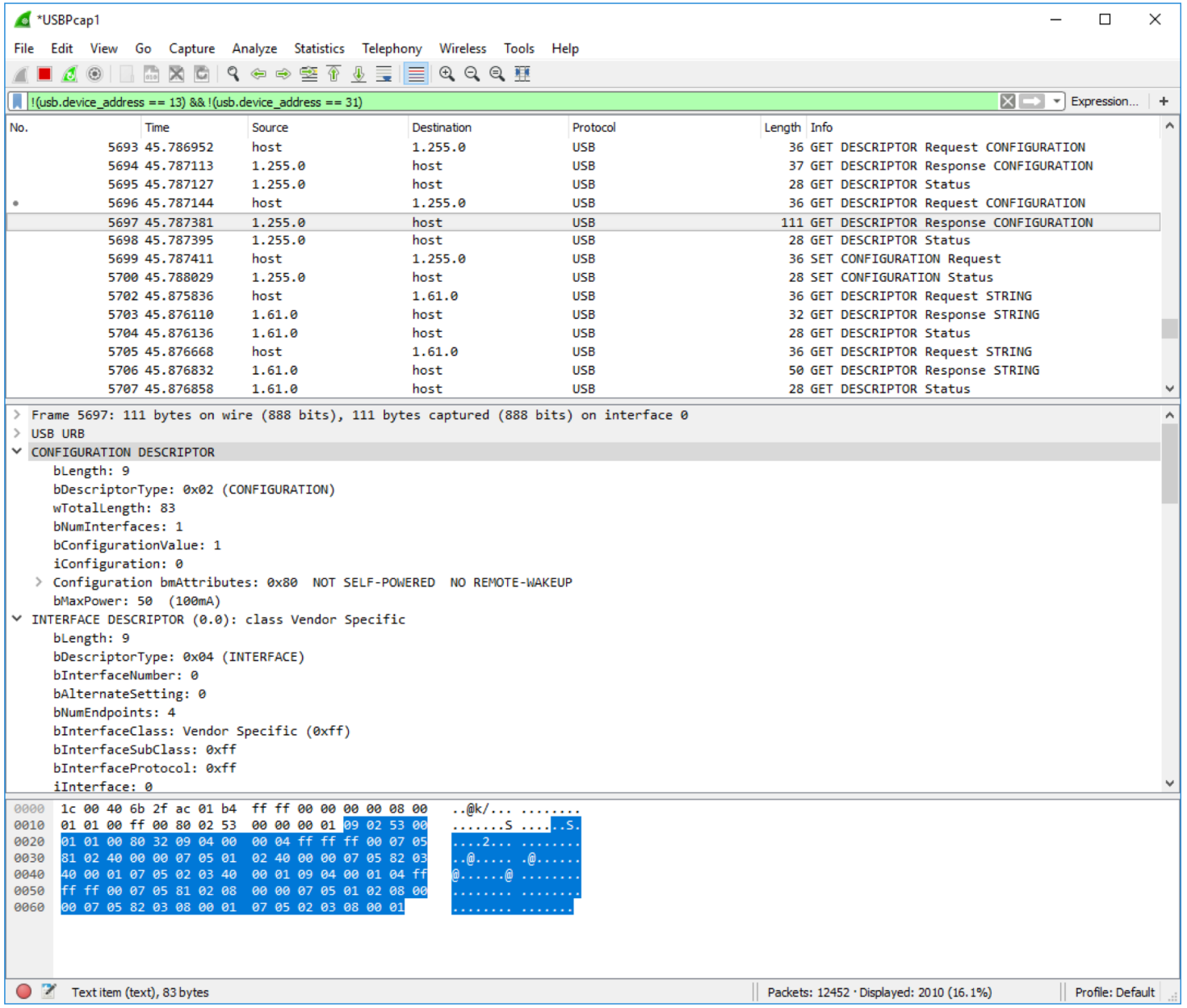
Tests are still under development

- Tests are actively being written
- Not all defined behavior is tested yet
- USBPhy's passing tests today may start failing when these new tests are added

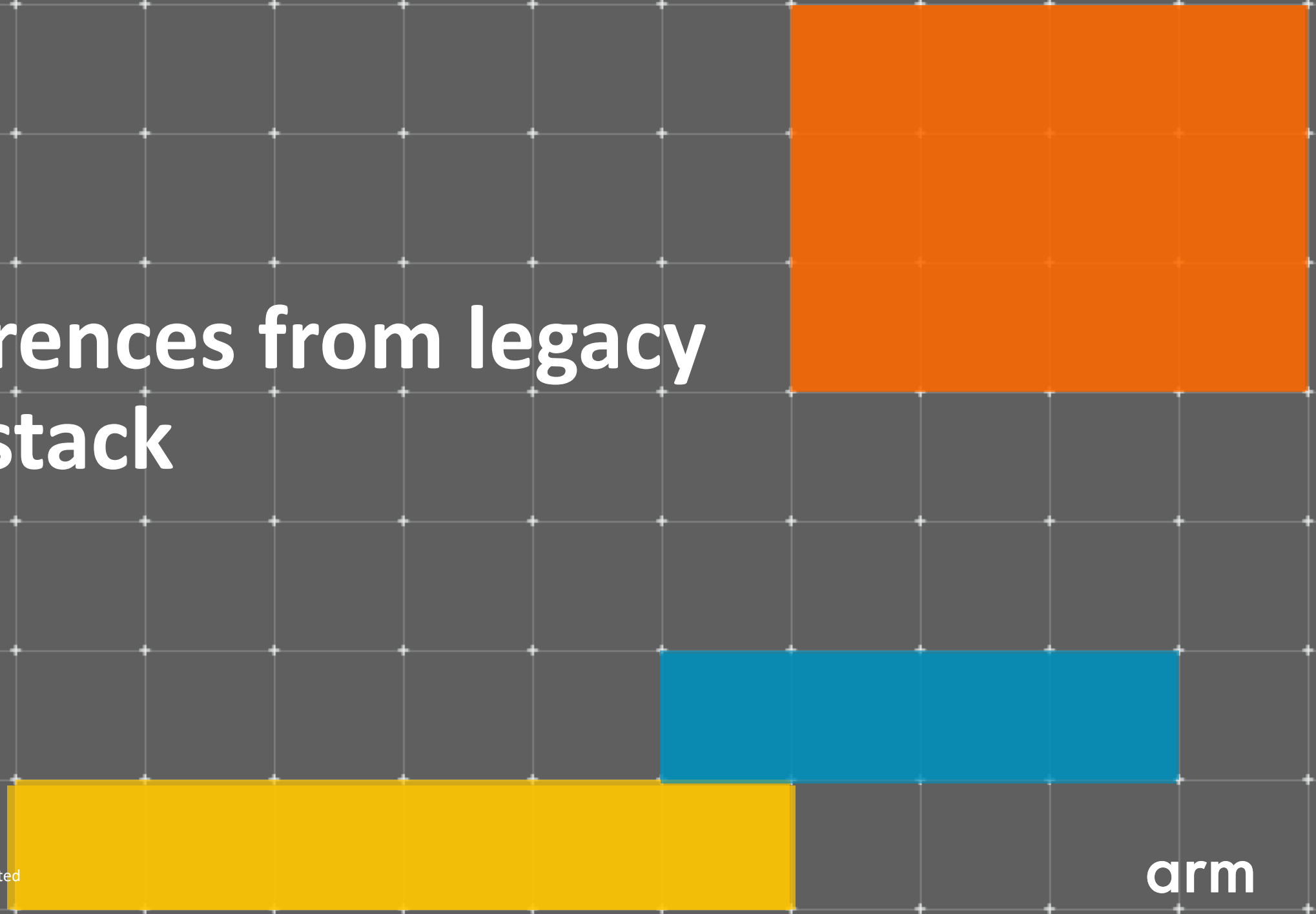
Tests	State
Endpoint Table	Not implemented
Control	Done
Endpoint	Not implemented
Concurrency	Not implemented
Reset Events	Partial

USB debugging tools

- Wireshark - https://www.wireshark.org/
- Total Phase USB analyzer
- (ISR)Printf - https://github.com/ARMmbed/ISRSerial



Differences from legacy USB stack



arm

High level differences

- Blocking removed from `USBDevice`. USB classes handle blocking reads and writes
- Control requests can be throttled by starting the next phase of the request outside of the request callback when the USB component is ready
- Endpoint reads and writes do not copy the data passed to them. Instead data is written directly to the buffer passed it. The buffer must remain unmodified until the transfer is completed or aborted
- Endpoint are not hardcoded. Instead endpoint functionality is reported by the device and appropriate endpoints can be selected at runtime
- A callback for power loss is available to handle states better
- Function names match the mbed-os naming convention

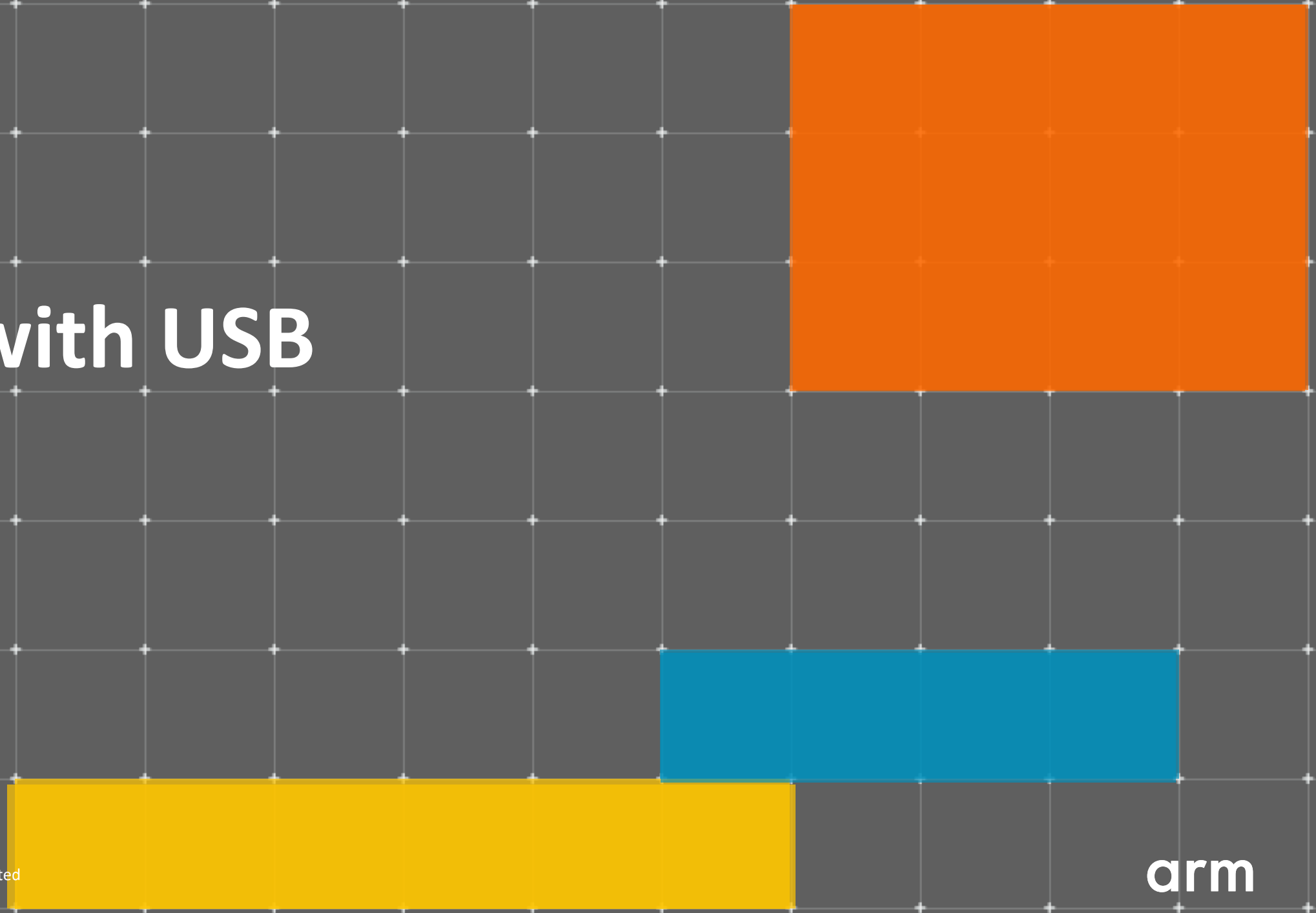
USBPhy mapping to legacy USB

Legacy USB API	New USB API
None	USBPhyHw::USBPhyHw()
None	USBPhyHw::~USBPhyHw()
USBHAL::USBHAL()	void USBPhyHw::init(USBPhyEvents *events)
USBHAL::~USBHAL()	void USBPhyHw::deinit()
None	bool USBPhyHw::powered()
void USBHAL::connect()	void USBPhyHw::connect()
void USBHAL::disconnect()	void USBPhyHw::disconnect()
void USBHAL::configureDevice()	void USBPhyHw::configure()
void USBHAL::unconfigureDevice()	void USBPhyHw::unconfigure()
None	void USBPhyHw::sof_enable()
None	void USBPhyHw::sof_disable()
void USBHAL::setAddress(uint8_t address)	void USBPhyHw::set_address(uint8_t address)
void USBHAL::remoteWakeup()	void USBPhyHw::remote_wakeup()
None	const usb_ep_table_t* USBPhyHw::endpoint_table()
None	uint32_t USBPhyHw::ep0_set_max_packet(uint32_t max_packet)
void USBHAL::EP0setup(uint8_t *buffer)	void USBPhyHw::ep0_setup_read_result(uint8_t *buffer, uint32_t size)
void USBHAL::EP0readStage(void)	None
void USBHAL::EP0read()	void USBPhyHw::ep0_read(uint8_t *data, uint32_t size)
uint32_t USBHAL::EP0getReadResult(uint8_t *buffer)	uint32_t USBPhyHw::ep0_read_result()
void USBHAL::EP0write(uint8_t *buffer, uint32_t size)	void USBPhyHw::ep0_write(uint8_t *buffer, uint32_t size)
void USBHAL::EP0getWriteResult()	None
void USBHAL::EP0stall()	void USBPhyHw::ep0_stall()
bool USBHAL::realiseEndpoint(uint8_t endpoint, uint32_t maxPacket, uint32_t flags)	bool USBPhyHw::endpoint_add(usb_ep_t endpoint, uint32_t max_packet, usb_ep_type_t type)
None	void USBPhyHw::endpoint_remove(usb_ep_t endpoint)
void USBHAL::stallEndpoint(uint8_t endpoint)	void USBPhyHw::endpoint_stall(usb_ep_t endpoint)
void USBHAL::unstallEndpoint(uint8_t endpoint)	void USBPhyHw::endpoint_unstall(usb_ep_t endpoint)
bool USBHAL::getEndpointStallState(uint8_t endpoint)	None
EP_STATUS USBHAL::endpointRead(uint8_t endpoint, uint32_t maximumSize)	bool USBPhyHw::endpoint_read(usb_ep_t endpoint, uint8_t *data, uint32_t size)
EP_STATUS USBHAL::endpointReadResult(uint8_t endpoint, uint8_t * buffer, uint32_t *bytesRead)	bool USBPhyHw::endpoint_read_result(usb_ep_t endpoint)
EP_STATUS USBHAL::endpointWrite(uint8_t endpoint, uint8_t *data, uint32_t size)	bool USBPhyHw::endpoint_write(usb_ep_t endpoint, uint8_t *data, uint32_t size)
EP_STATUS USBHAL::endpointWriteResult(uint8_t endpoint)	None
None	void USBPhyHw::endpoint_abort(usb_ep_t endpoint)
void USBHAL::usbisr()	void USBPhyHw::process()

USBPhyEvents mapping to legacy USB

Legacy USB callbacks	USBPhyEvent callbacks
busReset()	events->reset()
EP0setupCallback()	events->ep0_setup()
EP0out()	events->ep0_out()
EP0in()	events->ep0_in()
None	events->power(powered);
suspendStateChanged(suspended)	events->suspend(suspended)
SOF(frameNumber)	events->sof(frameNumber)
EP*_OUT_callback()	events->out(usb_ep_t endpoint)
EP*_IN_callback()	events->in(usb_ep_t endpoint)
usbisr()	events->start_process()

Fun with USB



arm

USBSerial class

- Use it like the normal serial class
- Read and write functionality
- Thread safe

Instructions:

- Create a file main.cpp
- Paste in the contents
- Make sure your target and toolchain are set, compile and flash with:

```
$ mbed compile -f
```

```
#include "mbed.h"
#include "USBSerial.h"

USBSerial serial;

int main()
{
    serial.printf("Hello world from mbed serial\r\n");
    while (true) {
        serial.printf("Enter a number\r\n");
        int number = 0;
        int args = serial.scanf("%i", &number);
        if (args == 1) {
            serial.printf("You entered: %i\r\n",
number);
        }
    }
}
```


arm

USBKeyboard class

- Print to simulate a key press
- Sent various key values
- Read the state of caps and other keys with `lock_status`
- Have fun on April 1st?

Instructions:

- Checkout usb_keyboard branch
- Create a file main.cpp
- Paste in the contents
- Make sure your target and toolchain are set, compile and flash with:

```
$ mbed compile -f
```

```
#include "mbed.h"
#include "USBKeyboard.h"

USBKeyboard key;

int main(void)
{
  while (1) {
      key.printf("Hello World\r\n");
      wait(1);
  }
}
```


arm

Hands-on workshop

- Use branch - `feature-hal-spec-usb-device`
- Workshop materials - https://github.com/ARMmbed/mbed-os-workshop-18q1
- USB 2.0 Specification - http://www.usb.org/developers/docs/usb20_docs
- Helpful USB basics - http://www.beyondlogic.org/usbnutshell

arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद