USB support in Zephyr OS

Zephyr Project Developer Summit 2022

Johann Fischer
johann.fischer@nordicsemi.no

Mountain View, CA / June 9, 2022

Contents I

1. Introduction
2. New USB device controller API
3. New USB device stack
4. USB host support
5. USB testing, emulated controller drivers, USBIP
6. USB support namespaces
7. References

What is it about today?

- USB support in Zephyr RTOS
- Continuation on the presenstation from ZDS2021
- Not about current USB device support
- ... which is not so bad
- ... but has few drawbacks and limitations

What is going on in USB development?

- New USB device controller (UDC) driver API
- New USB device stack implementation
- Current device support will stay maintained

Hot and new

- USB host controller (UHC) driver API
- Initial USB host stack implementation

USB device controller (UDC) API

- Support for multiple drivers (and instances)
- Support to query controller capabilities (FS, HS, rwup...)
- Support to check endpoint configuration
- Single asynchronous API to enqueue transfers
- ... (no direct read/write accesses to endpoint buffers)
- Uses net_buf for endpoint transfers
- Thin common layer between driver and device stack
- Implementation for nRF USBD and Kinetis USBFSOTG controllers

3. New USB device controller API

```c
/* Driver facing API */

struct udc_api {
        int (*ep_enqueue)(const struct device *dev,
                          struct udc_ep_config *const cfg,
                          struct net_buf *const buf);
        int (*ep_dequeue)(const struct device *dev,
                          struct udc_ep_config *const cfg);
        int (*ep_flush)(const struct device *dev,
                        struct udc_ep_config *const cfg);
        int (*ep_set_halt)(const struct device *dev,
                           struct udc_ep_config *const cfg);
        int (*ep_clear_halt)(const struct device *dev,
                             struct udc_ep_config *const cfg);
        int (*ep_enable)(const struct device *dev,
                         struct udc_ep_config *const cfg);
        int (*ep_disable)(const struct device *dev,
                          struct udc_ep_config *const cfg);
        int (*host_wakeup)(const struct device *dev);
        int (*set_address)(const struct device *dev,
                           const uint8_t addr);
        int (*enable)(const struct device *dev);
        int (*disable)(const struct device *dev);
        int (*init)(const struct device *dev);
        int (*shutdown)(const struct device *dev);
        int (*lock)(const struct device *dev);
        int (*unlock)(const struct device *dev);
};
```

3. New USB device controller API

```c
/* Upper layer facing API */

    int udc_init(const struct device *dev, udc_event_cb_t event_cb);
    int udc_enable(const struct device *dev);
...
    int udc_shutdown(const struct device *dev);
...
    int udc_set_address(const struct device *dev, const uint8_t addr);
    int udc_host_wakeup(const struct device *dev)
...
    int udc_ep_try_config(const struct device *dev,
                          const uint8_t ep,
                          const uint8_t attributes,
                          uint16_t *const mps,
                          const uint8_t interval);
...
    int udc_ep_enable(const struct device *dev,
                      const uint8_t ep,
                      const uint8_t attributes,
                      const uint16_t mps,
                      const uint8_t interval);
...
    int udc_ep_set_halt(const struct device *dev, const uint8_t ep);
...
    int udc_ep_enqueue(const struct device *dev, struct net_buf *const buf);
```

UDC drivers organization

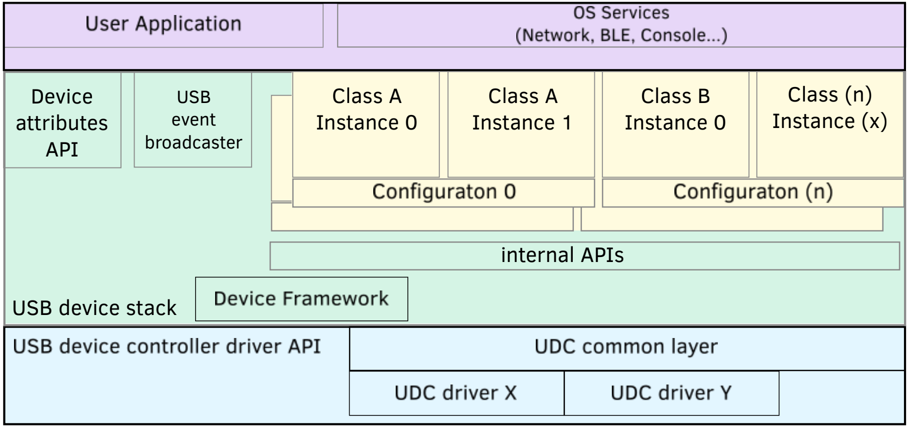
include/zephyr/drivers/usb/udc.h

```
drivers
└── usb
    ├── device
    │   └── ...
    └── udc
        ├── CMakeLists.txt
        ├── Kconfig
        ├── Kconfig.nrf
        ├── ...
        ├── udc_common.h ........................UDC common layer header
        ├── udc_common.c ..................................UDC common layer
        ├── udc_nrf.c .................................... nRF USBD driver
        └── ...
```

New USB device stack implementation

- Support for multiple UDC instances
- Support for multiple device configurations
- Interface to update/set device attributes at runtime
- Class/function assignment to a configuration at runtime
- Endpoint assignment by the stack
- Interface configuration by the stack
- Managed endpoint transfer events

USB device controller (UDC) API



4. New USB device stack

```
#include <zephyr/usb/usbd.h>
...
USBD_CONFIGURATION_DEFINE(config_foo, USB_SCD_SELF_POWERED, 200);
USBD_CONFIGURATION_DEFINE(config_baz, USB_SCD_REMOTE_WAKEUP, 200);

USBD_DESC_LANG_DEFINE(lang);
USBD_DESC_STRING_DEFINE(mfr, "ZEPHYR", 1);
...
USBD_DEVICE_DEFINE(uds_ctx, DEVICE_DT_GET(DT_NODELABEL(zephyr_udc0)),
                   0x2fe3, 0xffff);
...
        err = usbd_add_descriptor(&uds_ctx, &lang);
...
        err = usbd_add_descriptor(&uds_ctx, &mfr);
...

        err = usbd_add_configuration(&uds_ctx, &config_foo);
...

        err = usbd_register_class(&uds_ctx, "foobaz", 1);
...
        err = usbd_init(&uds_ctx);
...
        err = usbd_enable(&uds_ctx);
...
```

How does it work? (simplified)

- Device configurations are initialized by usbd_init()
- Device stack provides an event callback using udc_init()
- UDC submits an event (to stack's message queue) using callback
- Device is not recognized by the host until udc_enable()
- Finally device is enabled by usbd_enable()
- Host enumerates device, set configuration and interface alternate
- Stack configures endpoints according to class interface descriptors

USBD class API)

- Internal API used by the stack for the classes (functions)
- Looks like a driver API but much simpler
- Class instances use iterable section and a specific name to be recognizable
- Describes class configuration using (interface) descriptors
- API provides callbacks for configuration update,
- ... control and interface endpoint events,
- ... suspend and resume events

4. New USB device stack

```c
/* USBD class facing API */
struct usbd_class_api foobaz_api = {
        .update = foobaz_update,
        .control = foobaz_control,
        .request = foobaz_ep_request,
        .suspended = foobaz_suspended,
        .resumed = foobaz_resumed,
        .init = foobaz_init,
};

static struct usbd_class_data foobaz_data = {
        .desc = (struct usb_desc_header *)&foobaz_desc,
        .v_reqs = &foobaz_vregs,
};

USBD_DEFINE_CLASS(foobaz, &foobaz_api, &foobaz_data);

/* Stack facing internal class API (truncated) */
size_t usbd_class_desc_len(struct usbd_class_node *node);
...
struct usbd_class_node *usbd_class_get_by_ep(struct usbd_contex *uds_ctx,
                                             uint8_t ep);
...

/* Application facing USB device stack API (truncated) */
int usbd_register_class(struct usbd_contex *uds_ctx,
                        const char *name,
                        uint8_t cfg);
...
```

USB host support

- Similar in structure to new USB device support
- Asynchronous host controller driver API
- Simpler, initial host support
- Original driven by the need to test device support
- Should ideally map the device support features

Should ideally map the device support features

- Device CDC ACM <-> Host CDC ACM
- Device class-foo <-> Host class-foo

USB host controller (UHC) API

- Similar to UDC API
- Support for multiple driver (and instances)
- Support to obtain controller capabilities (FS, HS...)
- Single API for bus and transfer events
- Uses a container and net_buf for transfers
- Thin common layer between driver and host stack
- Implementation for MAX3421E host controller

USB testing

- The best way to test is on the hardware
- ... higher coverage and authenticity

Tests using Zephyr RTOS only (WIP)

- Closed loop using real host and device controller
- Closed loop using virtual host and device drivers

Tests using foreign host support or host stack

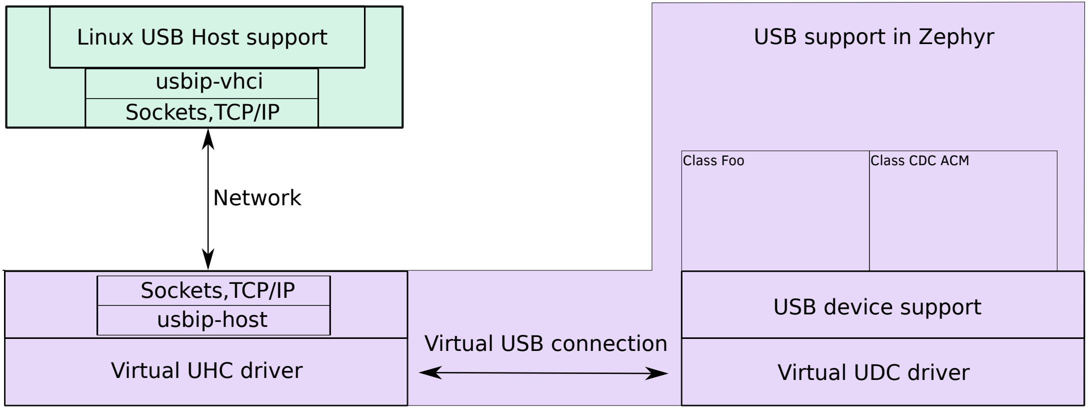
- Device support testing using testusb and Linux host
- Host controller testing using USBIP

- There are none emulated controller implemented yet (WIP)

USBIP

- Passes USB device from server to client over TCP
- ... actually exports a host controller over TCP
- Description and implementation available in Linux kernel[1] [2]
- New implementation based on USB host controller API is WIP
- Could be used to export a real host controller
- ... or virtual host connected to virtual device controller

USBIP support overview



Namespaces in USB support

- USB device controller driver API - udc_
- USB host controller driver API - uhc_
- current USB device controller driver API - usb_dc_
- USB device stack - usbd_
- USB host stack - usbh_
- Common Device Framework defines (Chapter 9) - usb_ and USB_
- current USB device stack - usb_

Questions?

9. References

[1] *USBIP Protocol*. https://elixir.bootlin.com/linux/latest/source/Documentation/usb/usbip_protocol.rst.

[2] *USBIP Tool*. https://elixir.bootlin.com/linux/latest/source/tools/usb/usbip/README.