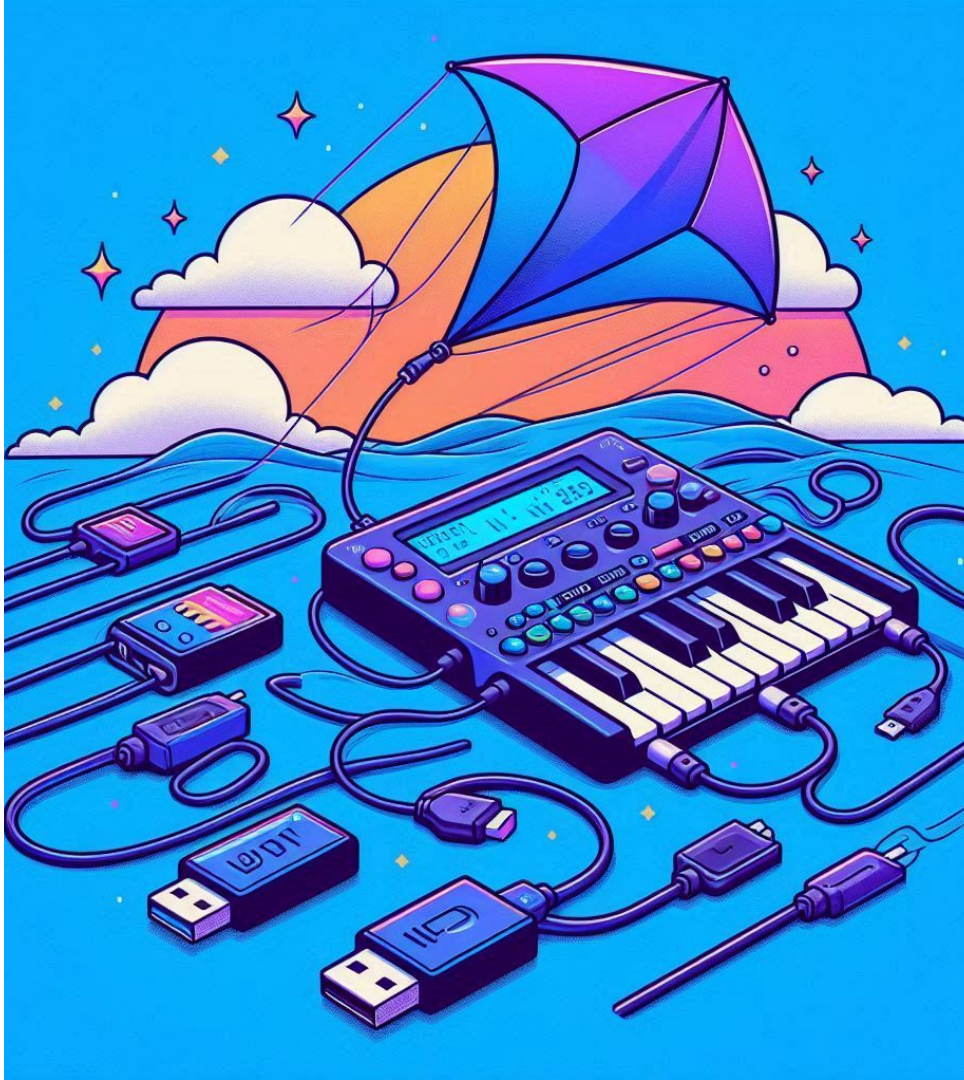
Making USB-MIDI2 instruments with Zephyr

Beep-bloop ! I'm a kite

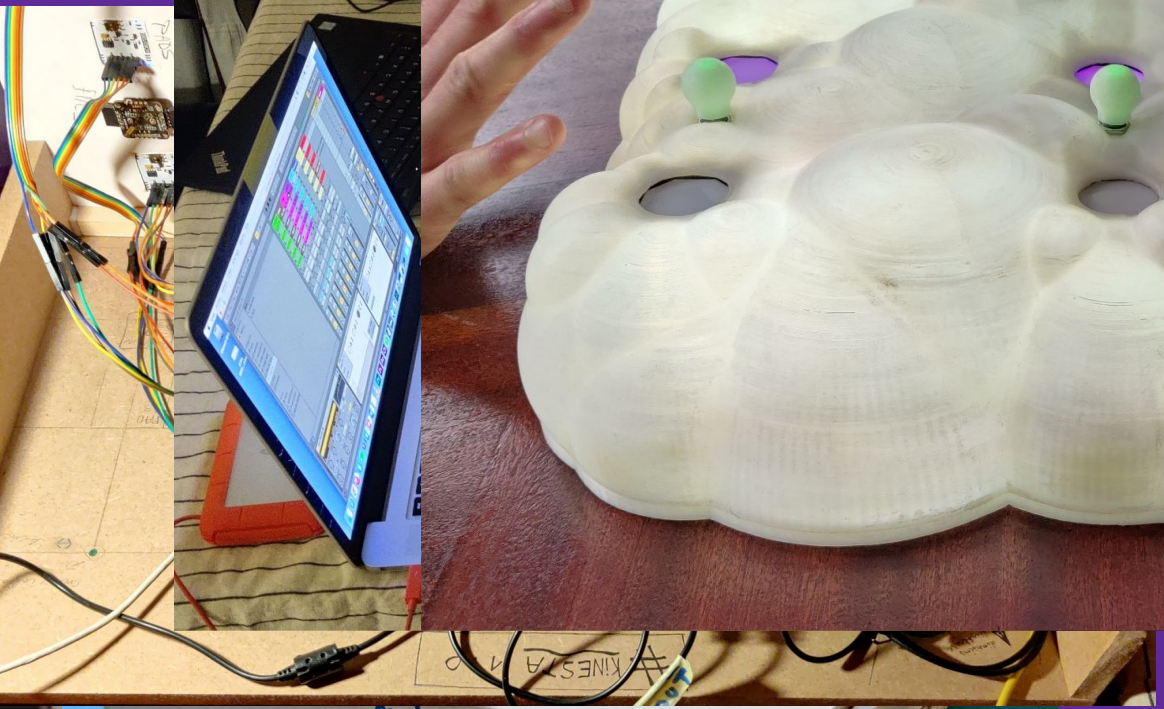
iTitou - Smartmonday March 2025



Who is he ?

- iTitou
- Embedded & Backend developer
- Ex-UrLab (~2013-2018) / FOSDEM delegate
- Ex-ULB (BA1-BA3) / -VUB (MA1-MA2)

- Btw, I like trains



How it all started

Zephyr®

Enters the room

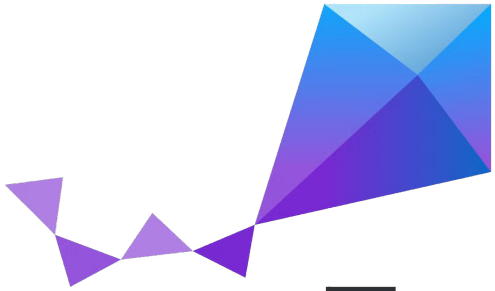
https://zephyrproject.org

zephyrproject-rtos/zephyr

Things running on Zephyr

What is Zephyr ?

- Operating System for very small devices (microcontrollers)
  - 100's MHz clock, "small" cores / 100's kB RAM
  - Dedicated peripherals (energy optimized)
- Multi-platform
- Provides a framework for
  - Multithreading with priorities
  - Networking (incl. BLE, Wifi, LoRa, etc...)
  - Security (TLS, crypto accelerators, etc...)
  - Device drivers
  - Utilities (logging, shell, etc...)
- Higher level than "bare-metal" programming
- Compile-time configuration framework
- west command-line tool

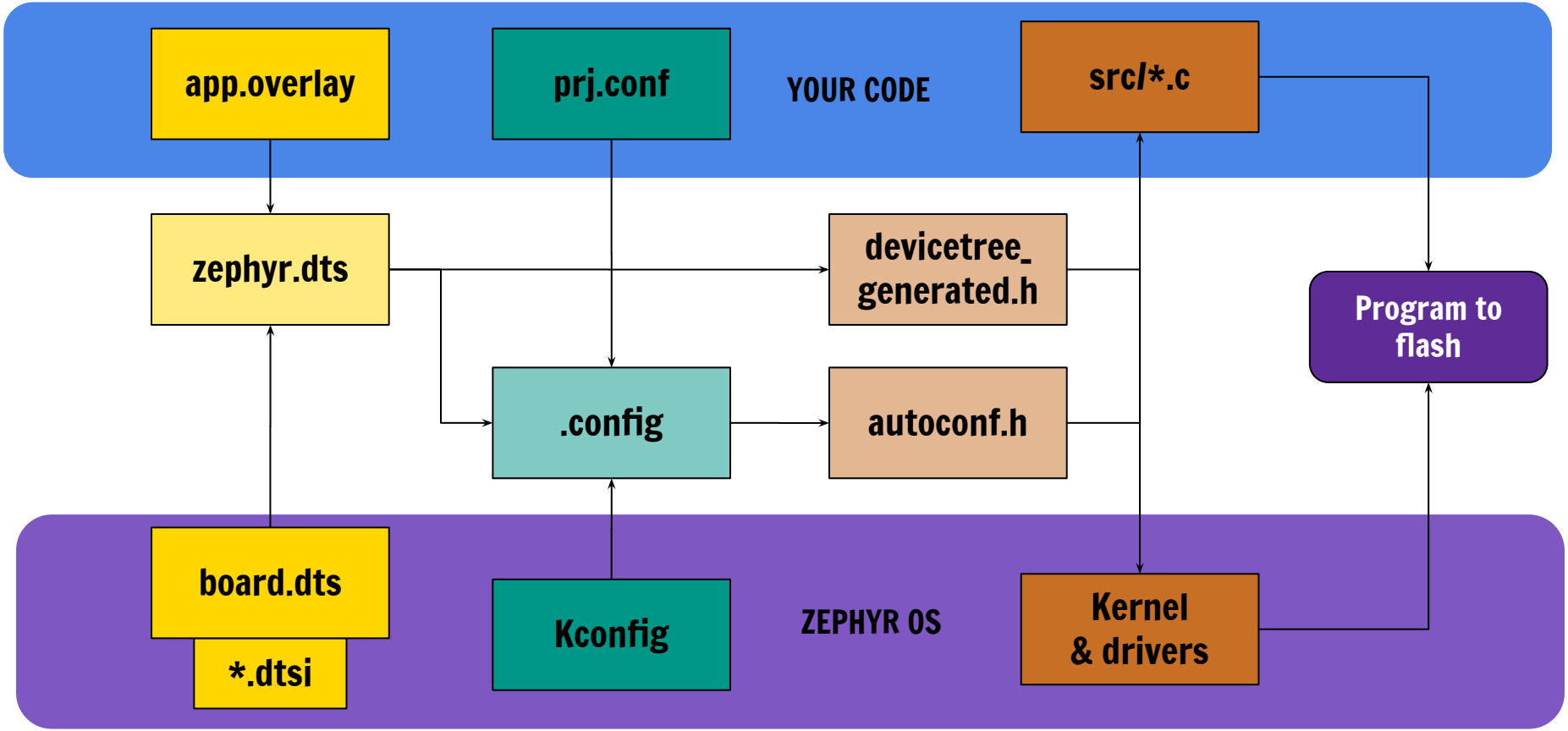


Zephyr®

Zephyr vs. Arduino

	Zephyr	Arduino
Writing, compiling and running your code	Your IDE + west command line tool (integrate with VSCode; platformio)	Arduino IDE (platformio)
Adding external libraries	West modules	Copy / Paste libraries
Writing code for the target	Main function, threads, tasks and callback functions	setup() + loop() (callbacks with some libs)
Declaring devices	Device tree (KConfig for driver options)	C++ class instantiation in code
Accessing devices	Driver API with opaque const struct *device	C++ class accessors and methods

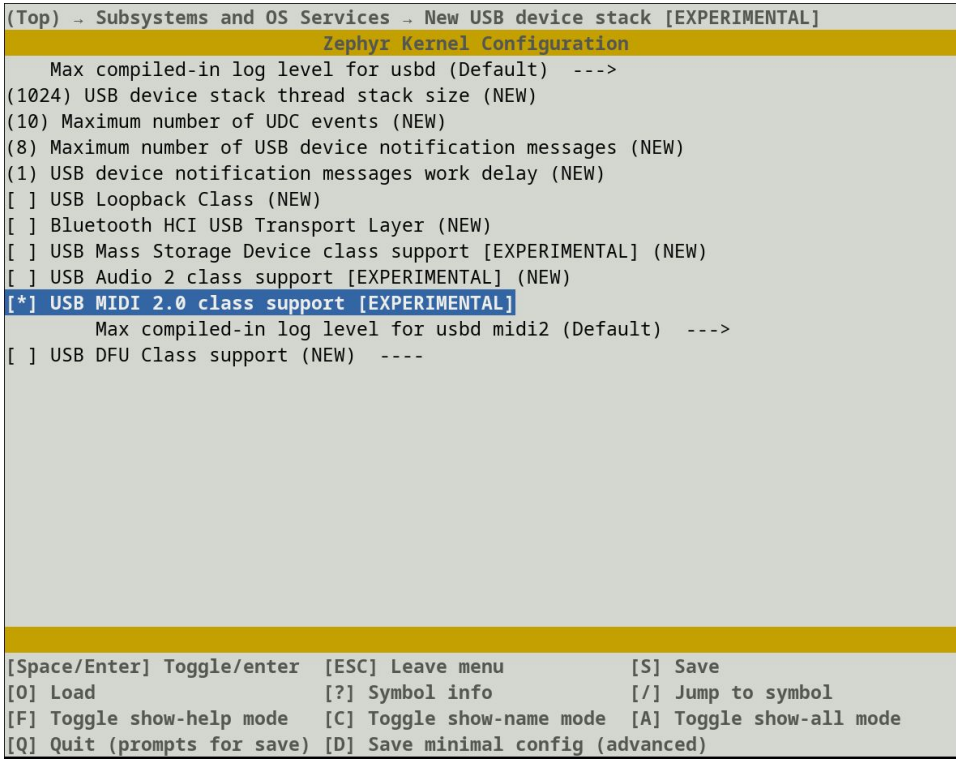
Building a Zephyr application



What is this Kconfig thing ?

- Originally, the (Linux) Kernel Config
- Define configurable items
  - Booleans (y/n)
  - Integers
  - Choices
  - Strings
- Available to Make
  - (what should be built ?)
- Available to C/C++ files (defines)

```
west build -t menuconfig
```



What is this devicetree thing ?

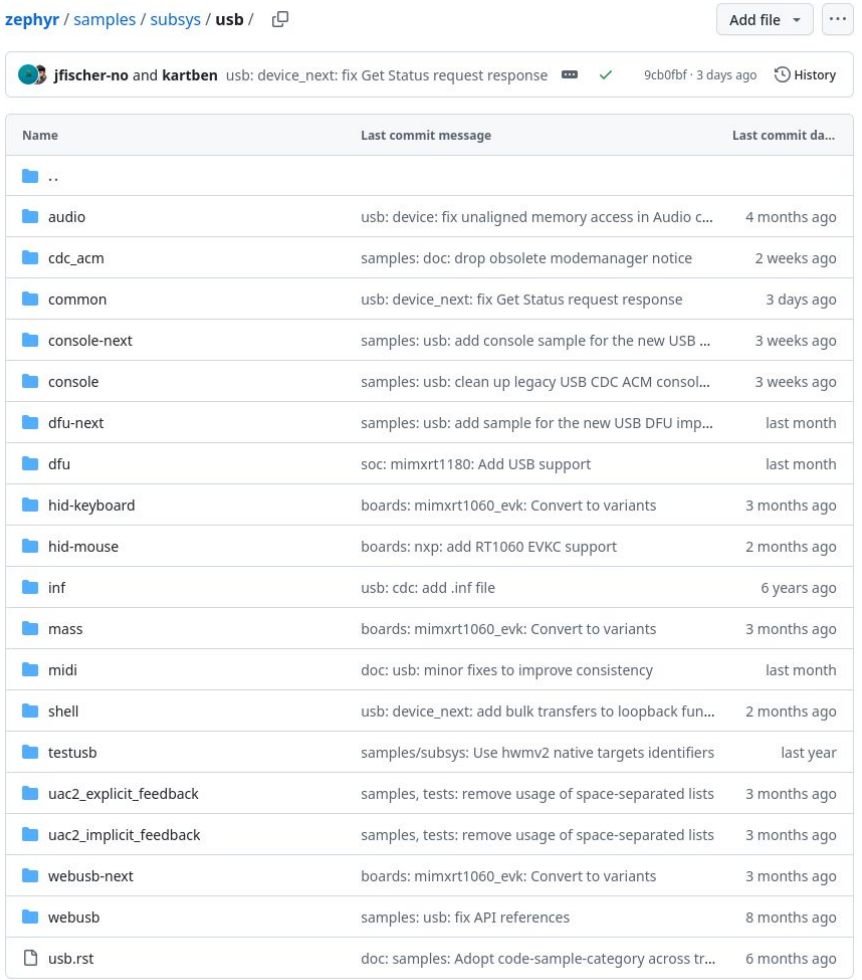
- Originates from embedded Linux (again)
  - ⚠️ Compile-time in Zephyr vs. run-time in Linux
- Describe topology of all static peripherals
  - (ie. not dynamic like USB, PCI, ...)
- Includes & fragments
- Assemble, compile and decompile with dtc

=> When changing hardware: swap the device tree, <u>not the code</u> !

```
/dts-v1/;
#include <st/f4/stm32f407Xg.dtsi>
#include <st/f4/stm32f407v(e-g)tx-pinctrl.dtsi>
#include <zephyr/dt-bindings/input/input-event-codes.h>

/ {
        model = "STMicroelectronics STM32F4DISCOVERY board";
        compatible = "st,stm32f4discovery";

        chosen {
                zephyr,console = &usart2;
                zephyr,shell-uart = &usart2;
                zephyr,sram = &sram0;
                zephyr,flash = &flash0;
                zephyr,ccm = &ccm0;
                zephyr,canbus = &can2;
        };

        leds {
                compatible = "gpio-leds";
                orange_led_3: led_3 {
                        gpios = <&gpiod 13 GPIO_ACTIVE_HIGH>;
                        label = "User LD3";
                };
                green_led_4: led_4 {
                        gpios = <&gpiod 12 GPIO_ACTIVE_HIGH>;
                        label = "User LD4";
                };
                red_led_5: led_5 {
                        gpios = <&gpiod 14 GPIO_ACTIVE_HIGH>;
                        label = "User LD5";
                };
                blue_led_6: led_6 {
                        gpios = <&gpiod 15 GPIO_ACTIVE_HIGH>;
                        label = "User LD6";
                };
        };

        gpio_keys {
                compatible = "gpio-keys";
                user_button: button {
                        label = "Key";
                        gpios = <&gpioa 0 GPIO_ACTIVE_HIGH>;
                        zephyr,code = <INPUT_KEY_0>;
                };
        };
```

Creating a Zephyr application

- **Easiest way: find related example in** samples/ **and modify it to your needs**
- **Create from scratch:**
  a. **Create a new directory**
  b. **Acc CMakeLists.txt (build rules)**
  c. **Add prj.conf (Kconfig options)**
  d. **(Optional) add app.overlay**
  e. **Add C files in src/**



zephyr / samples / subsys / usb / Add file

jfischer-no and kartben usb: device_next: fix Get Status request response 9cb0fbf · 3 days ago History

Name	Last commit message	Last commit da...
..		
audio	usb: device: fix unaligned memory access in Audio c...	4 months ago
cdc_acm	samples: doc: drop obsolete modemanager notice	2 weeks ago
common	usb: device_next: fix Get Status request response	3 days ago
console-next	samples: usb: add console sample for the new USB ...	3 weeks ago
console	samples: usb: clean up legacy USB CDC ACM consol...	3 weeks ago
dfu-next	samples: usb: add sample for the new USB DFU imp...	last month
dfu	soc: mimxrt1180: Add USB support	last month
hid-keyboard	boards: mimxrt1060_evk: Convert to variants	3 months ago
hid-mouse	boards: nxp: add RT1060 EVKC support	2 months ago
inf	usb: cdc: add .inf file	6 years ago
mass	boards: mimxrt1060_evk: Convert to variants	3 months ago
midi	doc: usb: minor fixes to improve consistency	last month
shell	usb: device_next: add bulk transfers to loopback fun...	2 months ago
testusb	samples/subsys: Use hwmv2 native targets identifiers	last year
uac2_explicit_feedback	samples, tests: remove usage of space-separated lists	3 months ago
uac2_implicit_feedback	samples, tests: remove usage of space-separated lists	3 months ago
webusb-next	boards: mimxrt1060_evk: Convert to variants	3 months ago
webusb	samples: usb: fix API references	8 months ago
usb.rst	doc: samples: Adopt code-sample-category across tr...	6 months ago

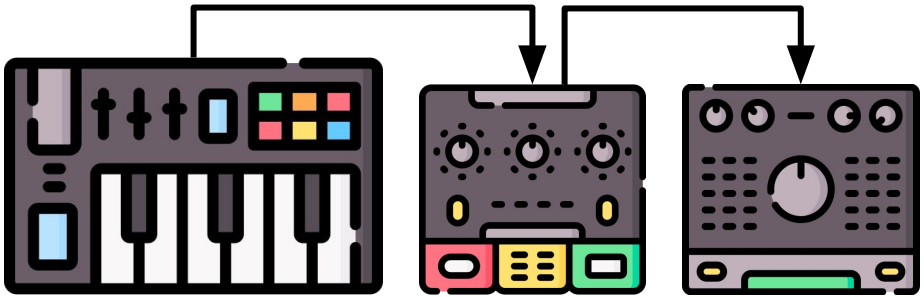
But wait...

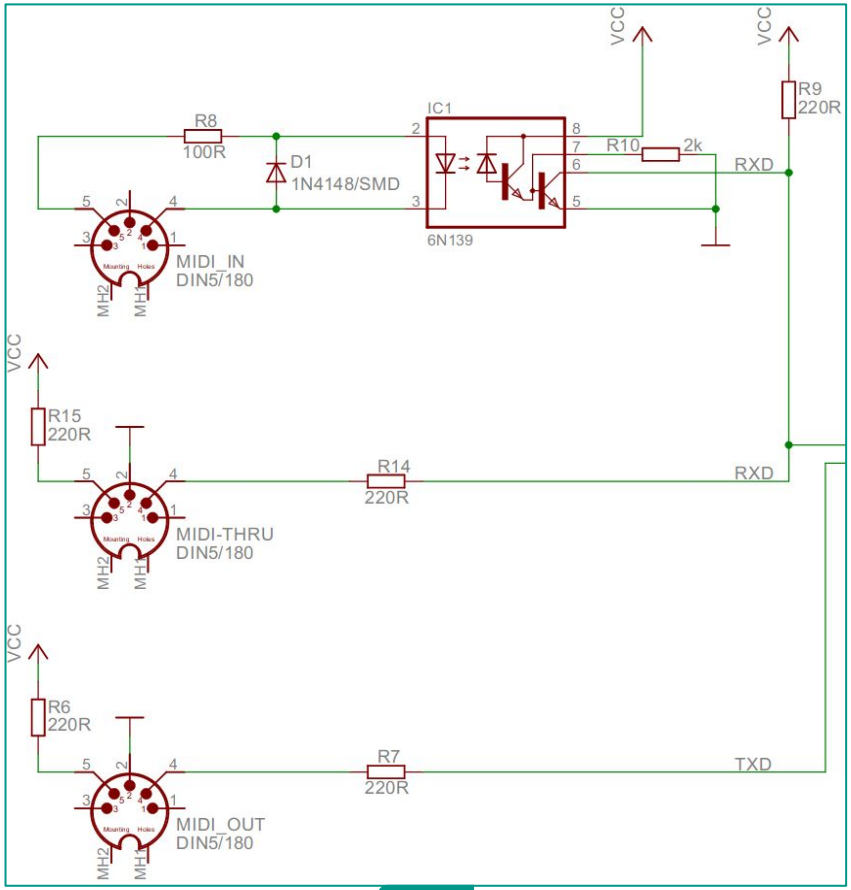
There is no support for USB-MIDI in Zephyr

So let's build it !

MIDI (1) - physical layer

- “Musical scores” communication protocol
- Born in the 1980’s
- Iconic DIN5 connector
- Connect **controllers** to **synthesizers**
  - unidirectional + pass-through (daisy-chaining
  - 16 channels
- UART @ 31.25 kb/s 8N1





[1]

MIDI (1) - messages

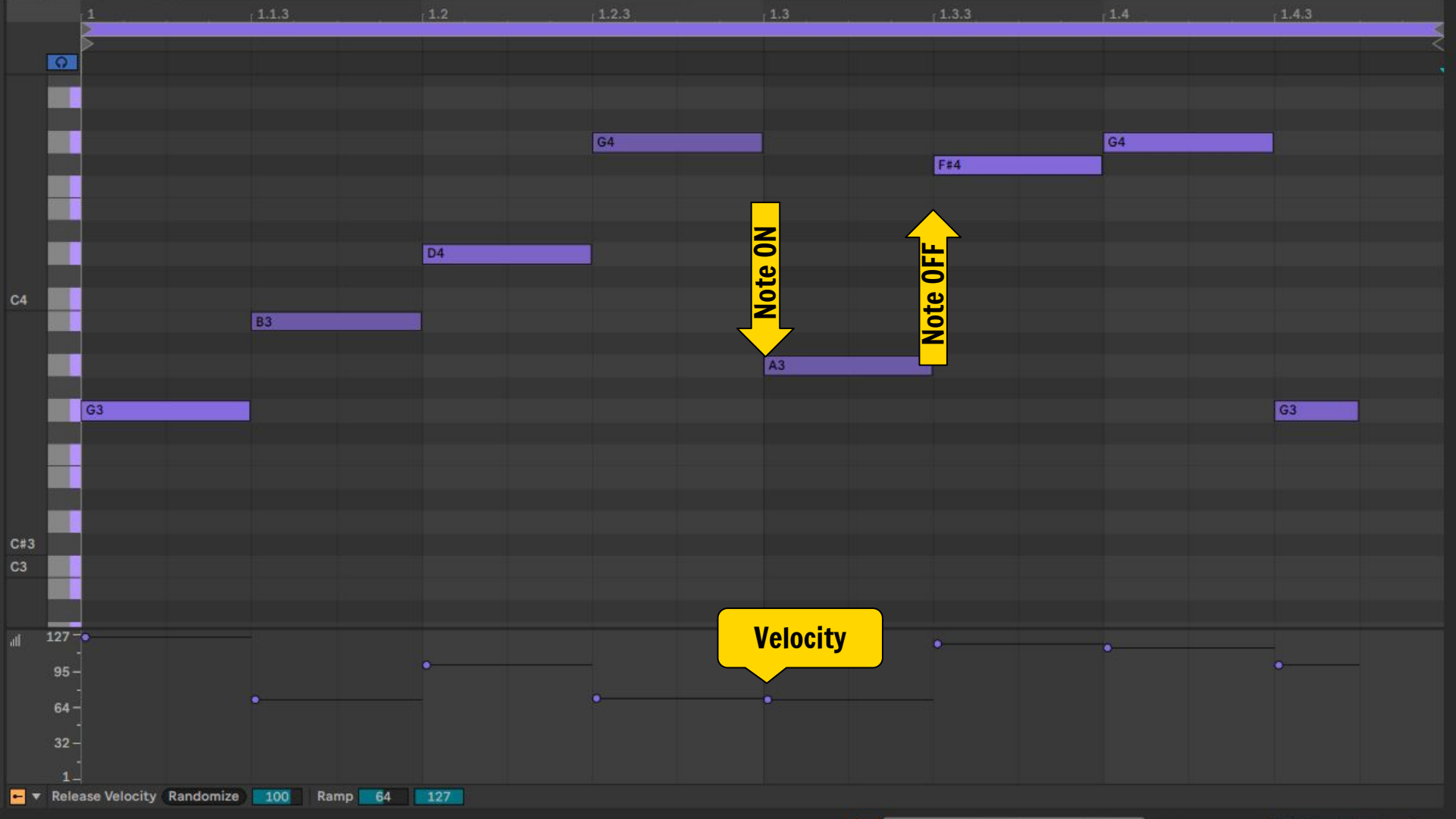
`90 46 63` Play note A#4 on ch. 1

`CF 2A` Load program 42 on ch. 16

`F0 46 4F 53 44 45 4D F7`
Send ASCII "FOSDEM" as SysEx

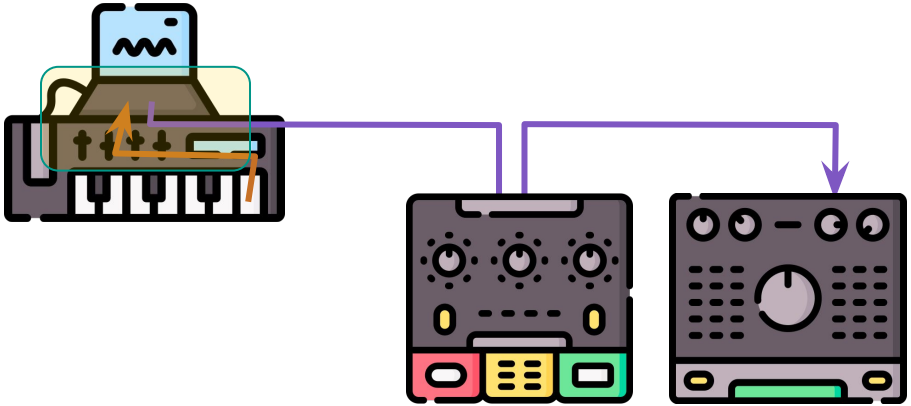
- Variable length uint8 packet (1b start + 7b data)
  - Start of packet = msb is 1 otherwise always 0
- 16 chans, commands with 0, 1 or 2 params
- Var. length SysEx for proprietary payloads
- Send "music scores" events in real time
  - .mid file format with timing infos for replay

Some common MIDI 1 messages		Byte 0		Byte 1	Byte 2
Channel voice messages	Note off / on 🎹	8/9	Ch (0-F)	Note (0-7F)	Velocity (0-7F)
	Control change 🎚️	B	Ch (0-F)	Ctl number (0-7F)	Value (0-7F)
	Program change	C	Ch (0-F)	Prog number (0-7F)	
	Pitch bend (value = signed int14)	E	Ch (0-F)	Value LSB (0-7F)	Value MSB (0-7F)
System common	Start System Exclusive (SysEx) 📟	F	0	(0-7F)[...] until "End of SysEx"	
	End of System Exclusive 📟	F	7		
System realtime	Clock (1 tick per quarter-note) ⏱️	F	8		
	Start / Continue / Stop ⏯️	F	A/B/C		


1
1.1.3
1.2
1.2.3
1.3
1.3.3
1.4
1.4.3
G4
G4
F#4
D4
B3
G3
A3
G3
Note ON
Note OFF
C4
C#3
C3
Velocity
127
95
64
32
1
Release Velocity
Randomize
100
Ramp
64
127

USB-MIDI 1.0 (1999)

- MIDI jacks
  - Represent USB-MIDI inputs / outputs
  - Embedded in/out = from/to host
  - External in/out = from/to the outside world
- Elements for internal processing (synthesizers and effects)



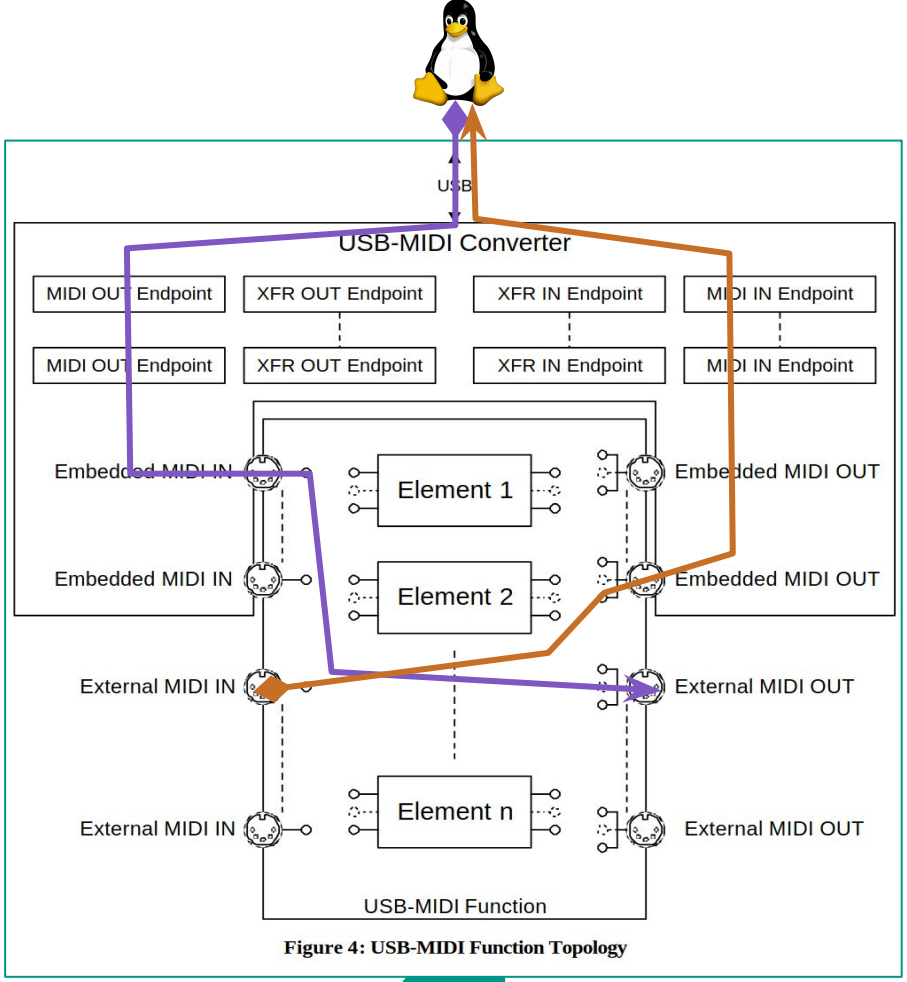
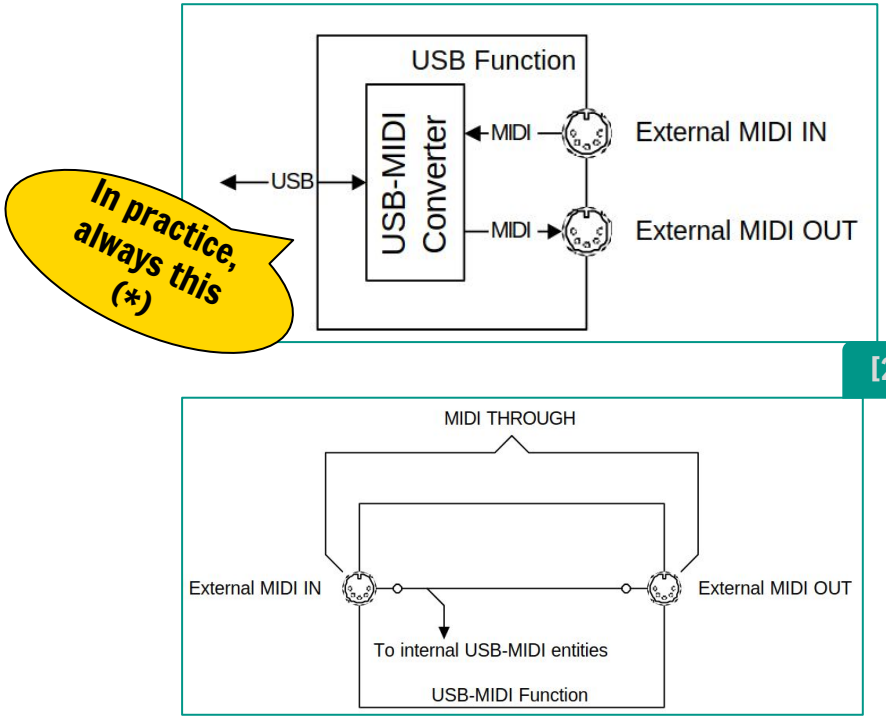


Figure 4: USB-MIDI Function Topology

[2]

USB-MIDI 1.0: Example topologies



[2]

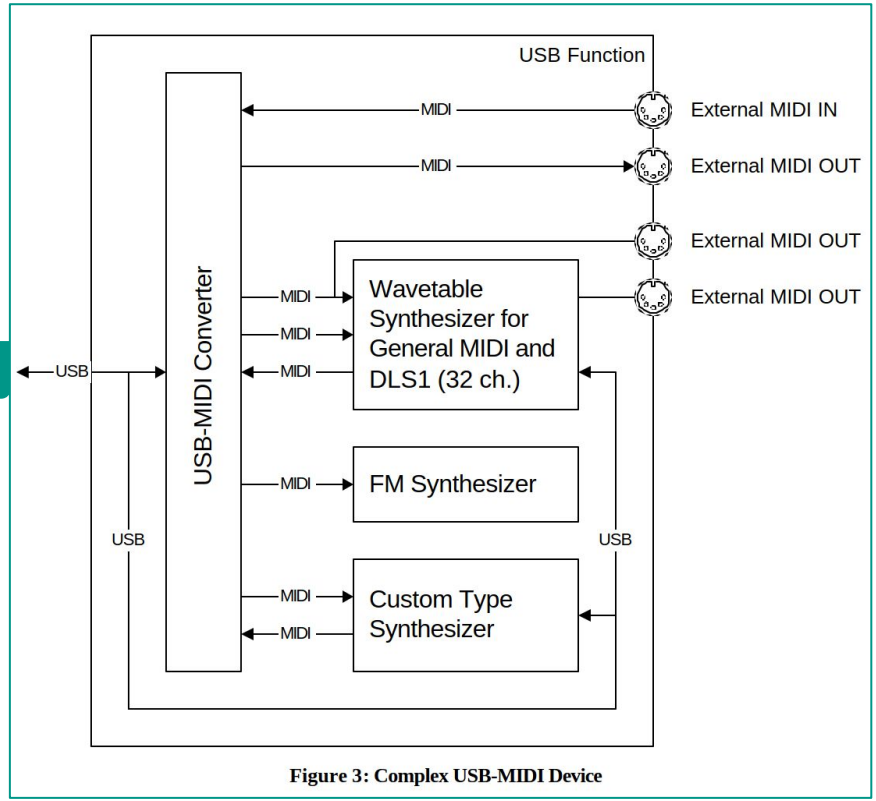
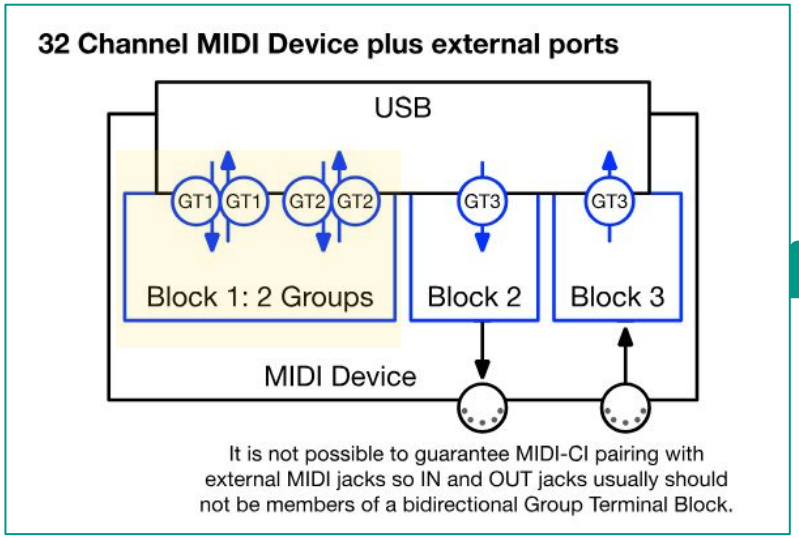


Figure 3: Complex USB-MIDI Device

(*) At least in all consumer-grade devices I had in my hands

USB-MIDI 2.0 (2020)

- Simplified operational model
- Group Terminal Blocks
  - Functional relationship b*l*w ports



[3]

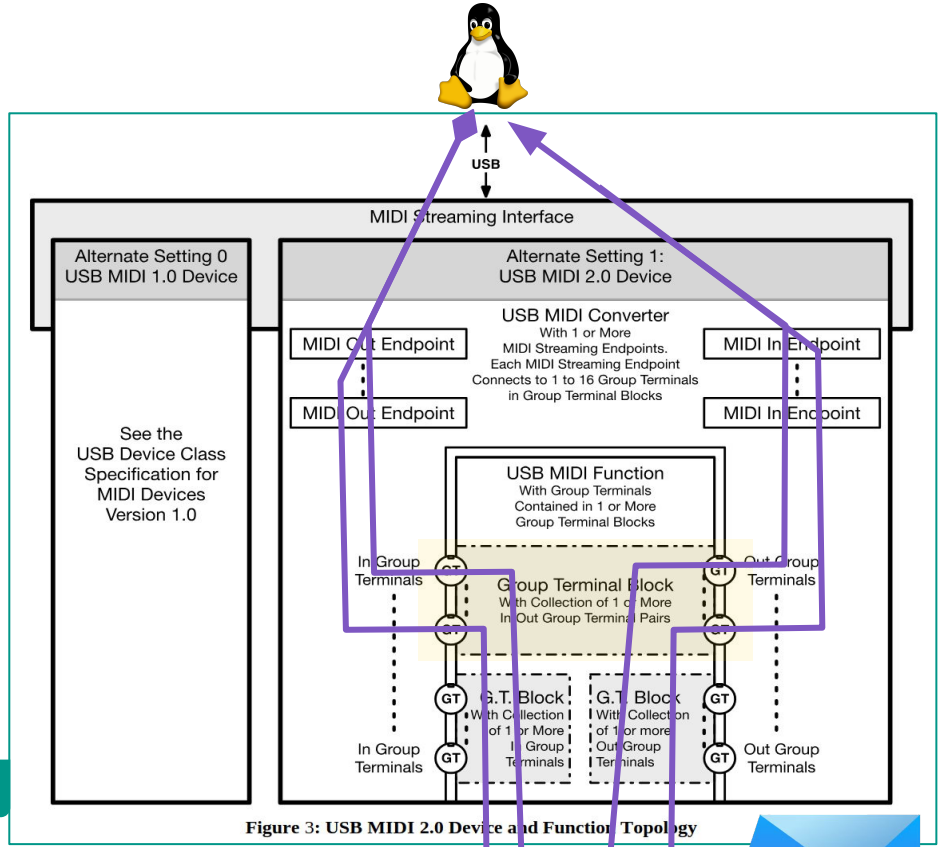


Figure 3: USB MIDI 2.0 Device and Function Topology

MIDI2 - Universal MIDI Packet endpoint

- New generic container format
  - 1 to 4 `uint32`
- Carry MIDI1 and MIDI2 messages:
  - Extended numerical ranges
  - Note attributes, tonal alterations
  - Tunings, chords, tempo,
  - time signature, lyrics, ...
- 16 Groups alloc. In Function Blocks
  - Discovery and dynamic configuration with UMP Stream (Groupless)

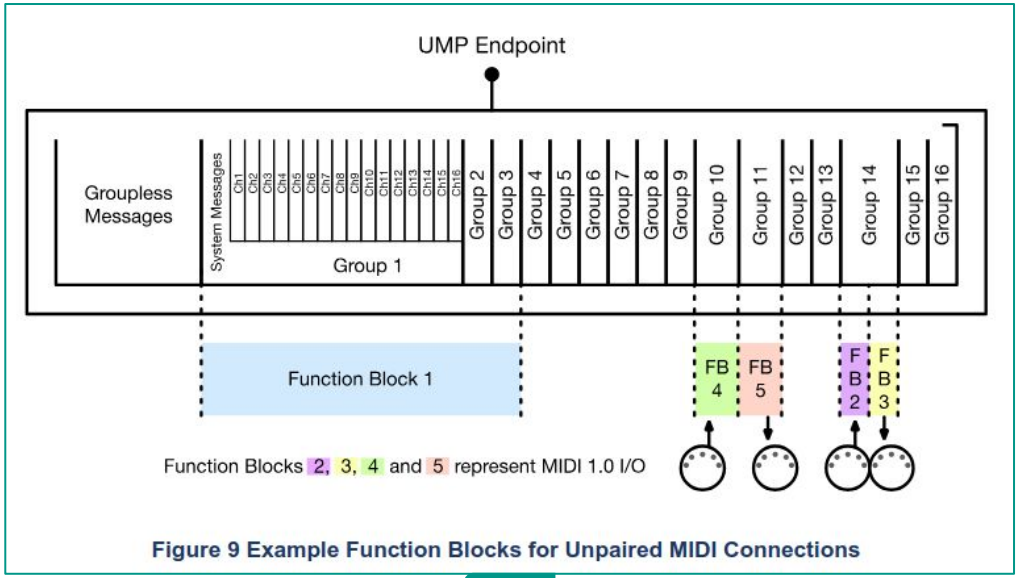


Figure 9 Example Function Blocks for Unpaired MIDI Connections

[4]

=> Acknowledge that most modern MIDI devices communicate on bidirectional, hi speed link (USB)

MIDI2 → Presentation

is not

USB-MIDI2.0 → Transport

is not

USB 2.0 → Levels below

ex: send MIDI1 messages with USB-MIDI2 on USB1

Example Universal MIDI Packets

MIDI1 - Note Off / On (1 word - 32b)

MT=2	Group (0-F)	8 / 9	Channel (0-F)	Note number (0-7F)	Velocity (0-7F)
MSB					LSB

MIDI2 - Note Off / On (2 words - 64b)

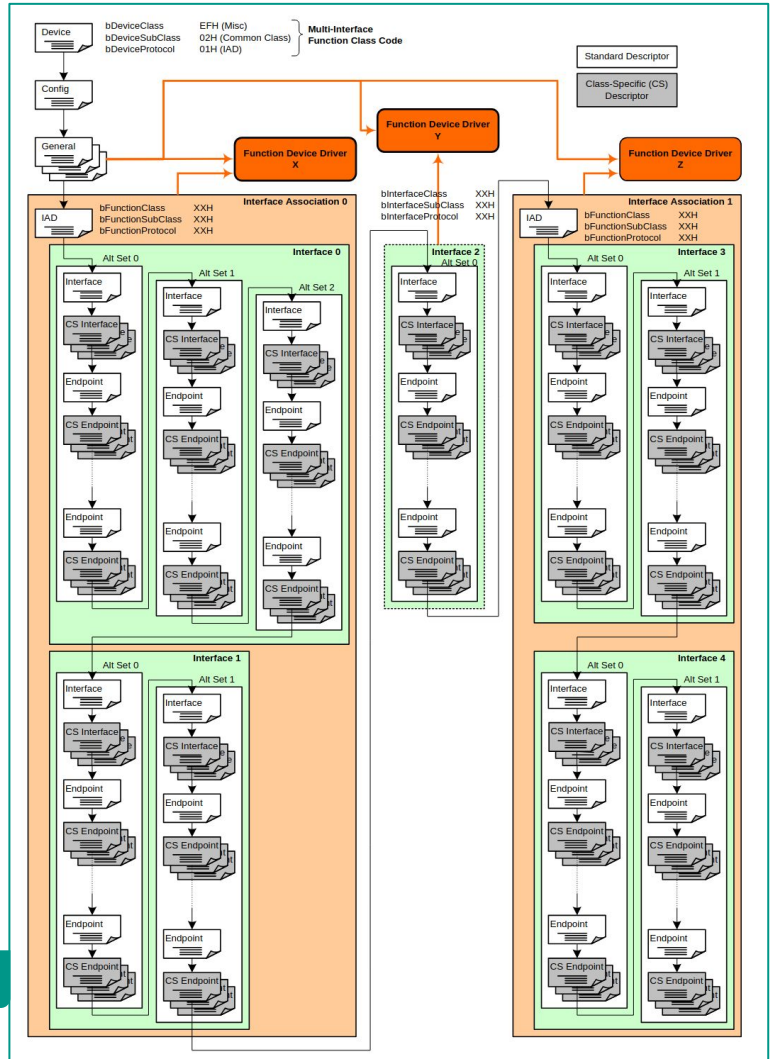
MT=4	Group (0-F)	8 / 9	Channel (0-F)	Note number (0-7F)	Attribute type
Velocity (0-FFFF)				Attribute value	
MSB					LSB

UMP Stream - Endpoint Name Notification (4 words - 128b)

MT=F	Form (0-3)	Status = 0x03	Name [0]	Name [1]
Name [2..5]				
Name [6..9]				
Name [10..13]				

USB descriptors, endpoints

- Descriptors
  - Expose capabilities to the host
  - Device class specific, iterable with `bLength`
  - “Key:value” mapping wrt. `bDescriptor(Sub)Type`
- Interfaces
  - Set of descriptors for one class “function”
- Endpoints
  - Unidirectional communication channels
  - Bulk, isochronous or interrupt
  - Declared in descriptors
- Class-specific requests
  - Out-of-band data exchange



[5]

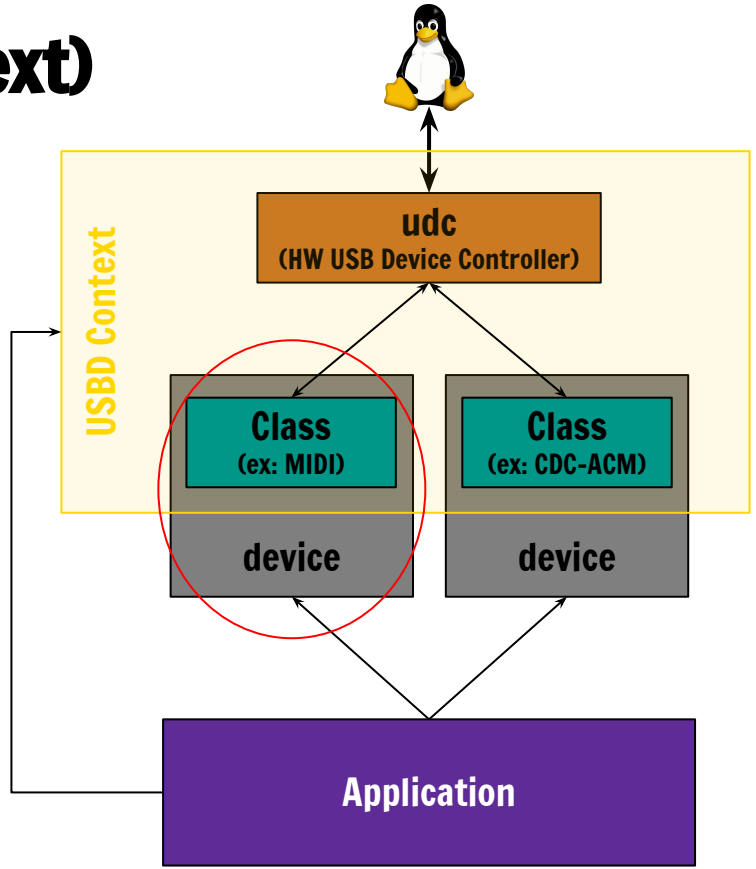
Our goal

Implement a USB-MIDI device

- Expose descriptors to the USB host (device capabilities)
  - USB class enumeration
- Respond to class-specific USB requests (topology)
  - Configured via the Zephyr device tree
- Exchange Universal MIDI packets over bulk USB endpoints
  - Runtime API

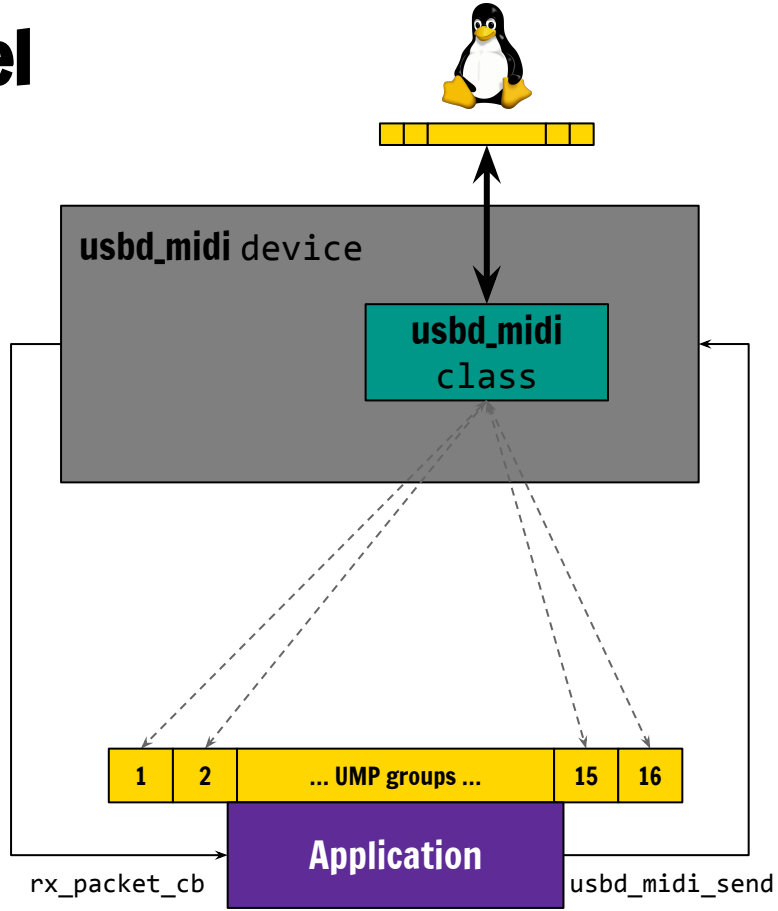
Zephyr udc, usbd (usb device_next)

- USBD Context manages wiring between
  - Low-level USB device controller
  - Registered USB device classes
  - Runtime state
- Easier than in old USB stack
  - Organize and pack descriptors
  - Handle multiple classes
- USB class drivers have to implement `usbd_class_api`
  - Corresponding Zephyr `device`



Zephyr usbd_midi: Operational model

- (up to) 16 UMP groups exchanged b/w host and Zephyr application
  - Group terminal blocks defined in device tree
- Send Universal MIDI Packet
  - Message Type (MT) field determines length
  - UMP Group field (except UMP_STREAM)
  - `-> usbd_midi_send(dev, pkt)`
- Receive Universal MIDI Packet
  - Register event handlers (all optional)
  - `-> usbd_midi_set_ops(dev, {rx_packet_cb, ready_cb})`
  - (avoid unneeded buffering)



Zephyr usbd_midi descriptors

- Empty USB-MIDI1.0 on altsetting 0
  - (ie. no jacks or elements)
  - Required by the spec, won't enumerate without
- USB-MIDI2.0 on altsetting 1
  - Group terminal blocks (topology) retrieved via separate class-specific request (as per spec.)
  - MUST be selected by host
- One pair of endpoints per interface

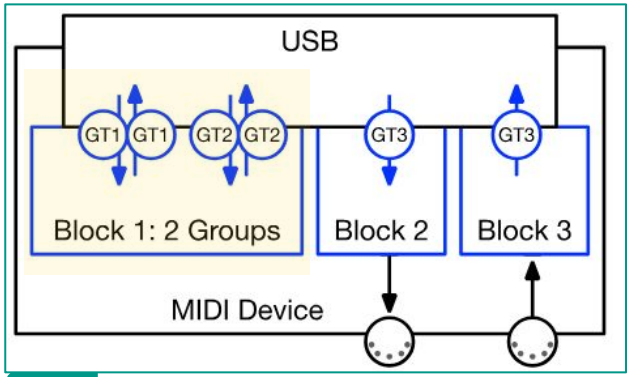
Class			Interface association
	If 0	Alt 0	Audiocontrol Interface
			Audio class-specific
	If 1	Alt 0	MIDIStreaming Interface 1.0
			MIDIStreaming class-specific
			Endpoint
			Endpoint class-specific
			Endpoint
			Endpoint class-specific
		Alt 1	MIDIStreaming Interface 2.0
			MIDIStreaming class-specific
			Endpoint
			Endpoint class-specific
			Endpoint
			Endpoint class-specific

Let's make some noise !

Set up usbd_midi in Zephyr application

prj.conf

```
CONFIG_USB_DEVICE_STACK_NEXT=y
CONFIG_USBD_MIDI2_CLASS=y
```



[3]

app.overlay

```
/ {
	usb_midi: usb_midi {
		compatible = "zephyr,midi2-device";
		status = "okay";
		#address-cells = <1>;
		#size-cells = <1>;

		block1 {
			reg = <0 2>;
			protocol = "midi2";
			terminal-type = "bidirectional";
			status = "okay";
		};
		/* ... */
		block3 {
			reg = <2 1>;
			protocol = "midi1-up-to-64b";
			terminal-type = "input-only";
			status = "okay";
		};
	};
};
```

Start at UMP grp 1

Span 2 groups

Register application event handlers

- Register callbacks
  - MIDI packet from host
  - Connection status change

[3]

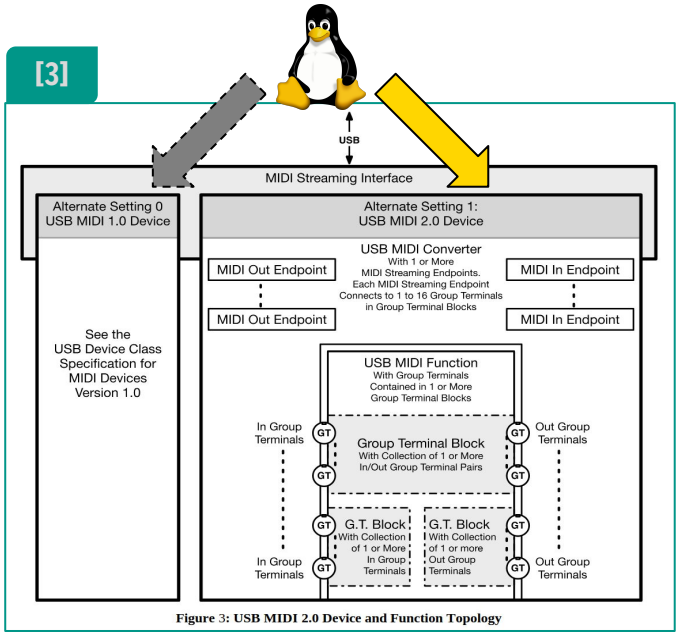


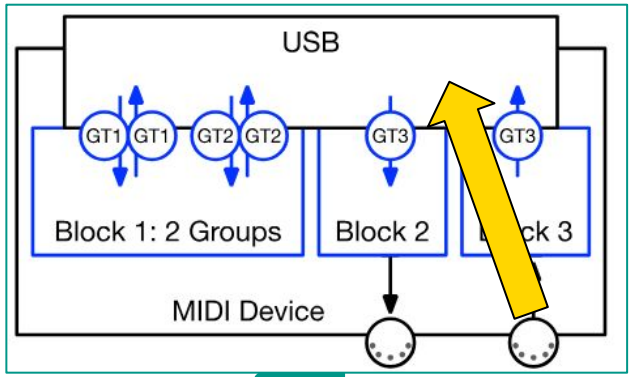
Figure 3: USB MIDI 2.0 Device and Function Topology

```c
#include <zephyr/drivers/gpio.h>
#include <zephyr/usb/class/usbd_midi2.h>

static void on_device_ready(const struct device *dev,
                            const bool ready)
{
      gpio_pin_set_dt(&led, ready);
}

static const struct usbd_midi_ops ops = {
      .ready_cb = on_device_ready,
      .rx_packet_cb = rx_midi,
};

int main(void)
{
      /* … Initialize application state … */

      usbd_midi_set_ops(midi, &ops);
}
```

More in 2 slides...

MIDI1 keyboard

- Leverage input subsys
- Build UMP, then send it



[3]

```c
#include <zephyr/input/input.h>
#include <zephyr/usb/class/usbd_midi2.h>

static const struct device *usb_midi =
       DEVICE_DT_GET(DT_NODELABEL(usb_midi));

static void key_press(struct input_event *evt, void *_)
{
      if (evt->type != INPUT_EV_KEY || evt->code > 0x7f) {
            return;
      }
      uint8_t command = evt->value
                      ? UMP_MIDI_NOTE_ON
                      : UMP_MIDI_NOTE_OFF;
      uint8_t note = evt->code;
      const struct midi_ump ump = UMP_MIDI1_CHANNEL_VOICE(
            2,
            command, 0,
            note, 100
      );
      usbd_midi_send(usb_midi, ump);
}
INPUT_CALLBACK_DEFINE(NULL, key_press, NULL);
```

Key press

Key release

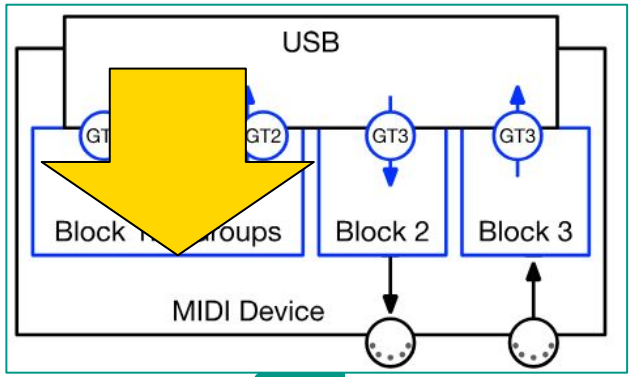
UMP group 3

MIDI chan 1

Velocity ≅ “fortissimo”

MIDI2 Synthesizer

- React to MIDI2 Note On/Off sent by the host



[3]

Let's just assume we have these

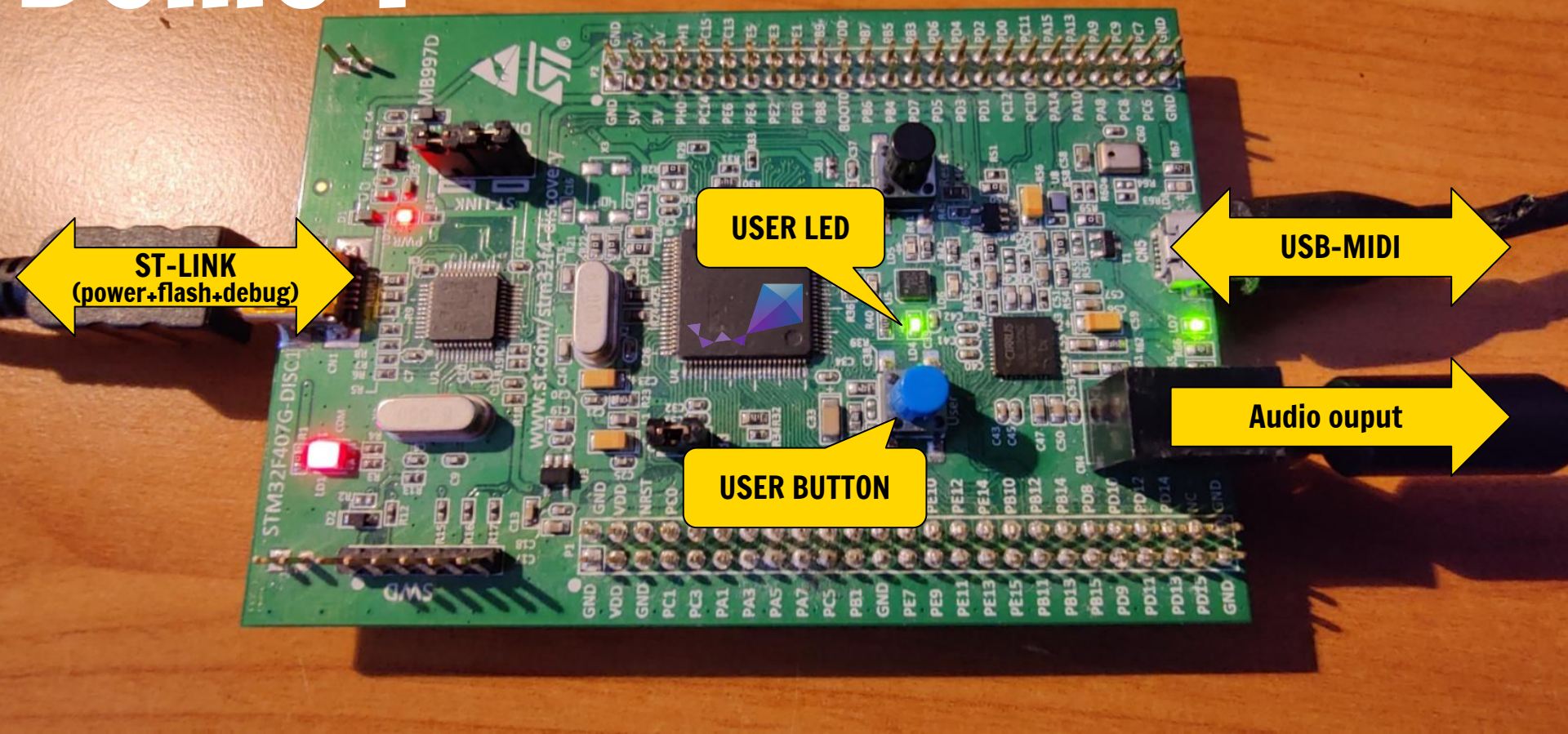
```
static void rx_midi(const struct device *dev,
                    const struct midi_ump pkt)
{
      if (UMP_MT(pkt) != UMP_MT_MIDI2_CHANNEL_VOICE) {
            return;
      }

      uint8_t channel = (UMP_GROUP(pkt) << 4)
                      | UMP_MIDI_CHANNEL(pkt);
      if (channel >= 32)
            return;

      switch (UMP_MIDI_COMMAND(pkt)) {
      case UMP_MIDI_NOTE_ON:
            .start_tone(channel,
                  UMP_MIDI1_P1(pkt),
                  UMP_MIDI2_VELOCITY(pkt)
            );
            break;
      case UMP_MIDI_NOTE_OFF:
            .stop_tone(channel);
            break;
      }
}
```

2 groups x16 chans

Note pitch

Note volume

Demo !


ST-LINK
(power+flash+debug)
USER LED
USB-MIDI
Audio ouput
USER BUTTON

USB-MIDI2 is fairly new

Let's build fancy instruments and controllers in the open !

(Coming in Zephyr 4.1)

Thank you !

iTitou titouanc iTitou

Going further and beyond

- Pull Request “add new MIDI 2.0 device class” in Zephyr
- MIDI 2.0 on Linux (kernel documentation)
- USB Device Support in Zephyr RTOS from the Application Perspective - Johann Fischer (Video from EOSS 2023)
- Open Source MIDI driver in Windows

- World's First MIDI Shellcode :: portasynthinca3's blog

Figures and images

[1] Olimex SHIELD MIDI schematics

[2] USB Device Class Definition for MIDI Devices

[3] USB Device Class Definition for MIDI Devices Release 2

[4] Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol

[5] USB Interface Association Descriptor Device Class Code and Use Model

[*] Flaticon “Audio Edition” pack

[*] https://commons.wikimedia.org/wiki/File:Orange_Banjo.svg