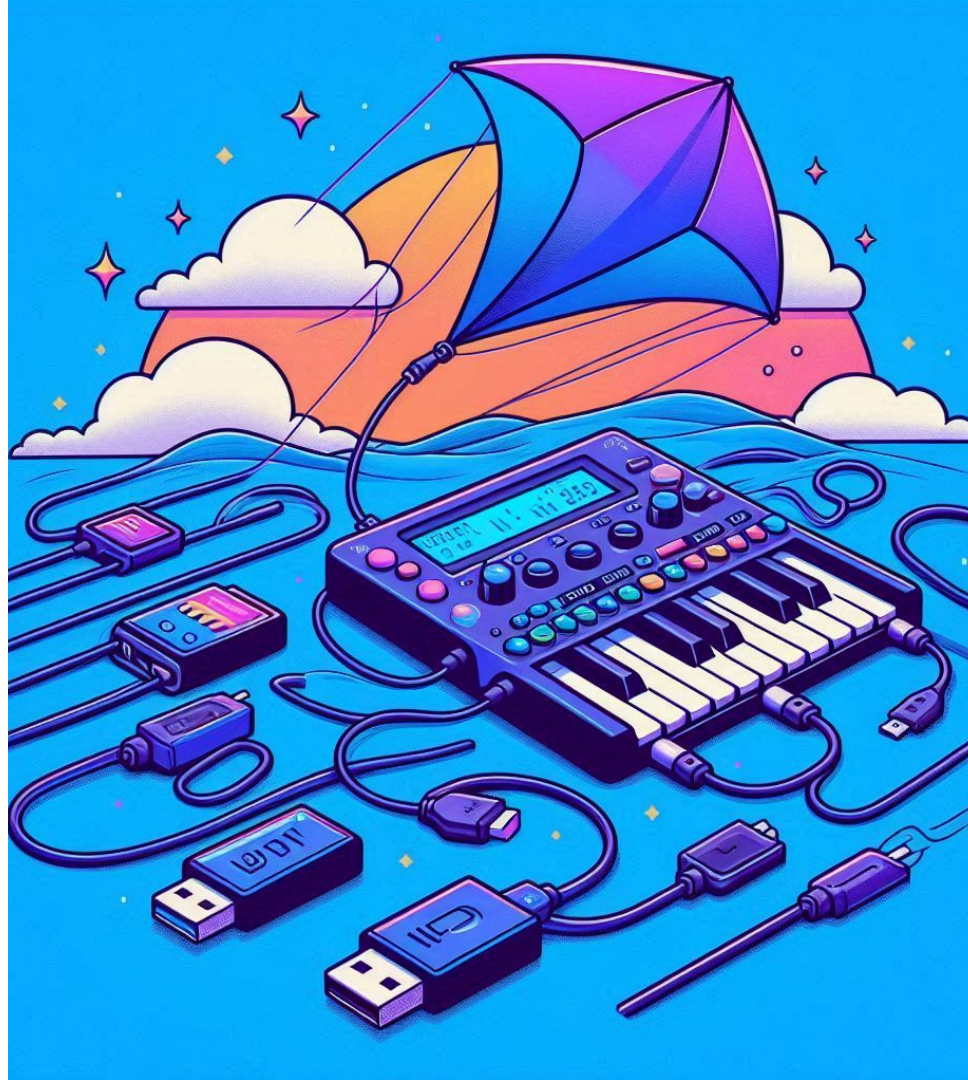


The USB-MIDI 2.0 device class in Zephyr

Making beeps and bloops with a kite

iTitou (Titouan CHRISTOPHE) - FOSDEM 2025





How it all started

So, why ?



Zephyr[®]

- **Devicetree, Kconfig, west**
 - **Free & Open source**
 - **Large collection of drivers**
 - **Subsys & OS services**
 - ***“Feels at home”* when coming from Linux**
-

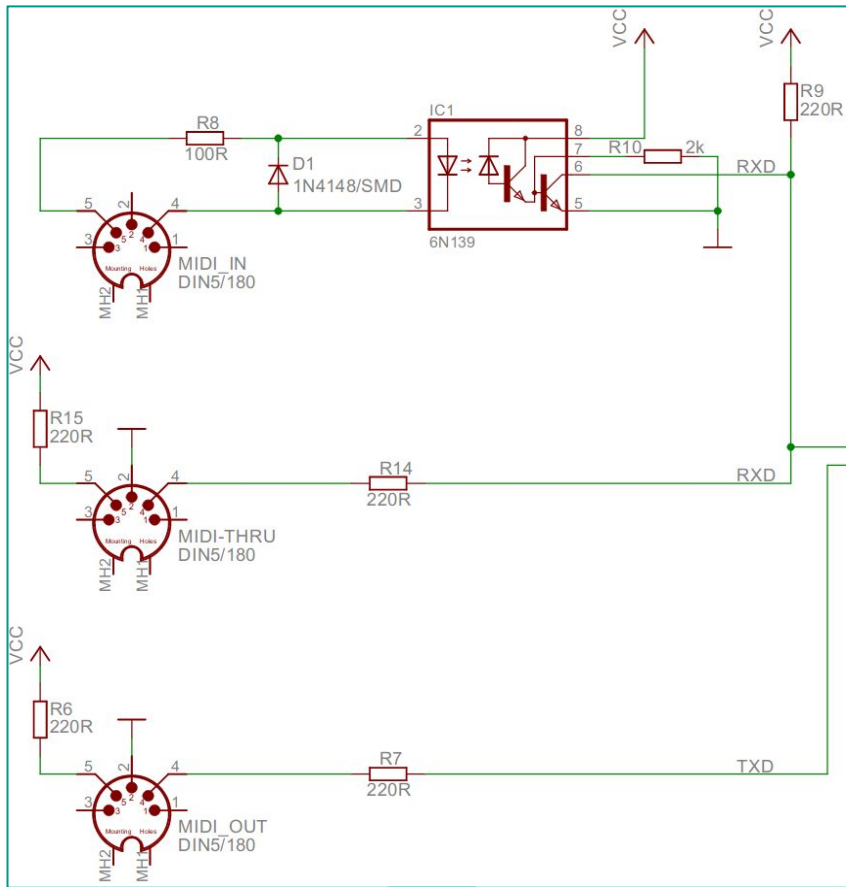
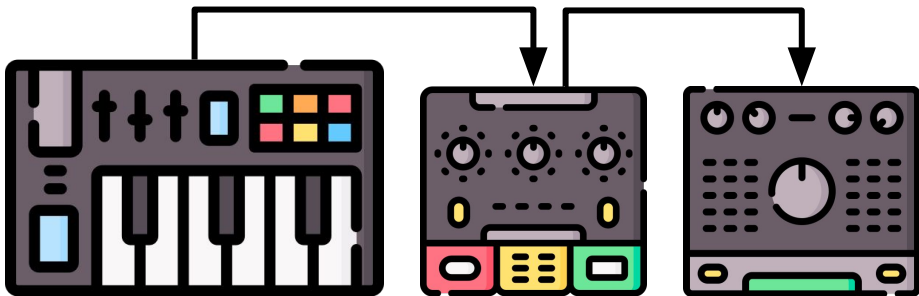
But wait...

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**



MIDI (1) - messages

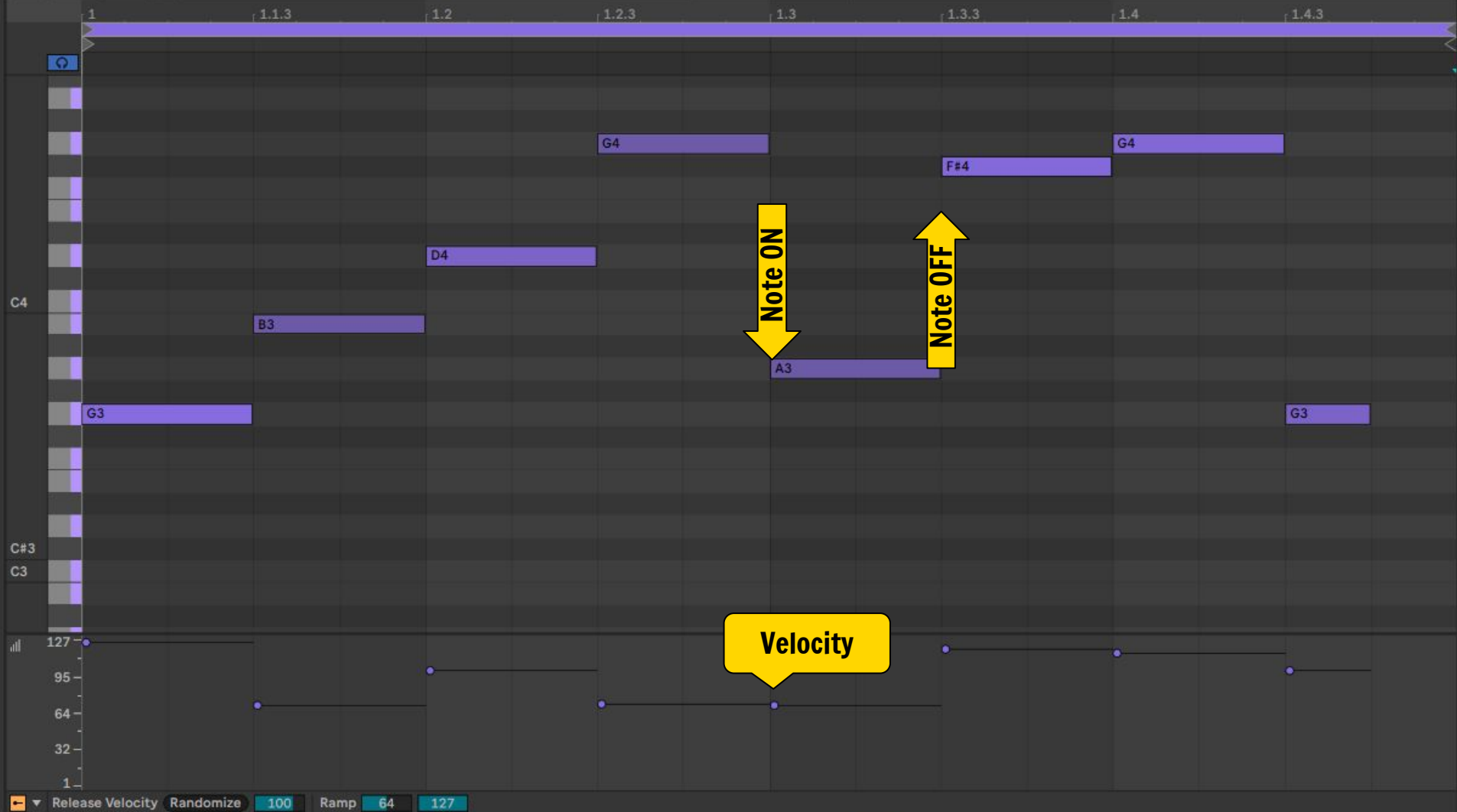
90 46 63 Play note A#4 on ch. 1

CF 2A Load program 42 on ch. 16

F0 46 4F 53 44 45 4D E7
Send ASCII "FOSDEM" as SysEx

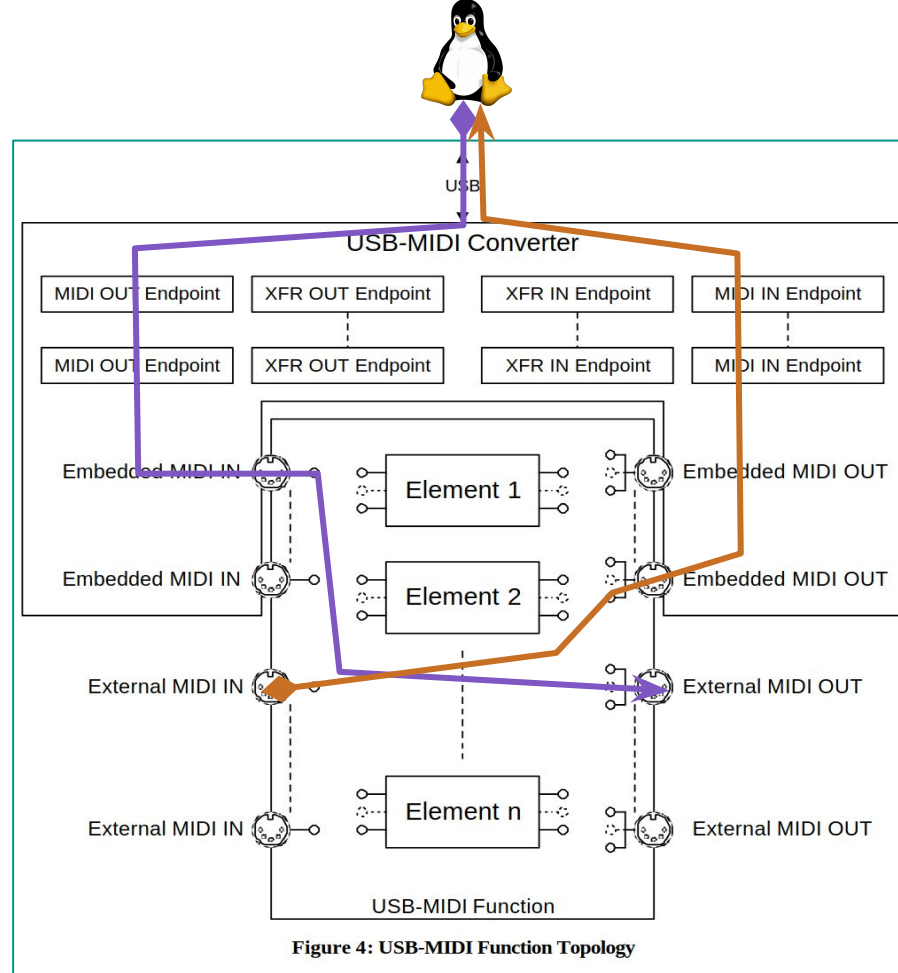
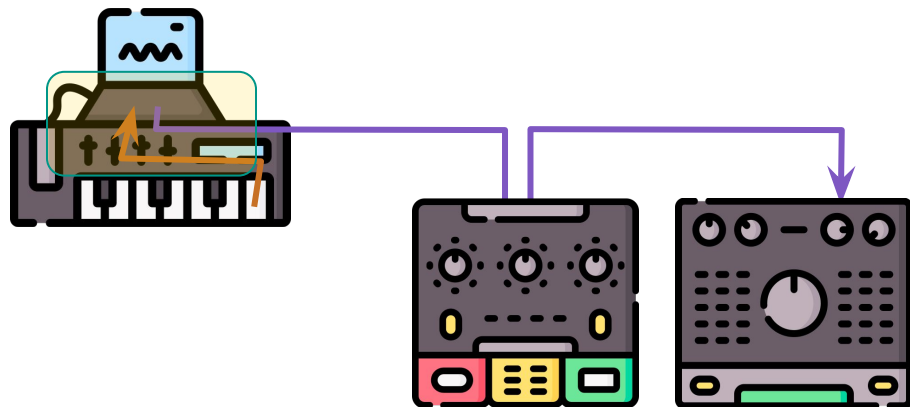
- Variable length uint8 packet (1b start + 7b data)
 - Start of packet = msb is 1 otherwise always 0
- 16 chans, commands with 0, 1 or 2 params
- Var. length SysEx for proprietary payloads
- Send "music scores" events in real time
 - .mid file format with timing infos for replay

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

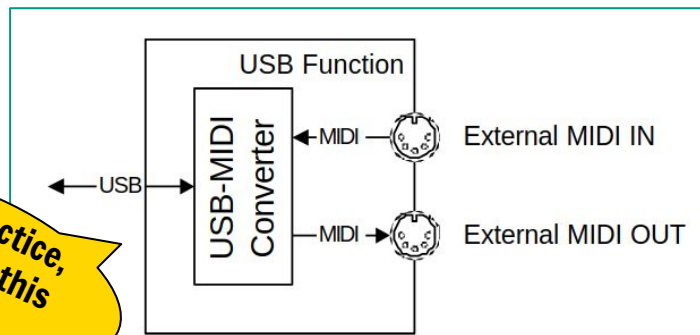


USB-MIDI 1.0 (1999)

- **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)



USB-MIDI 1.0: Example topologies



*In practice,
always this
(*)*

[2]

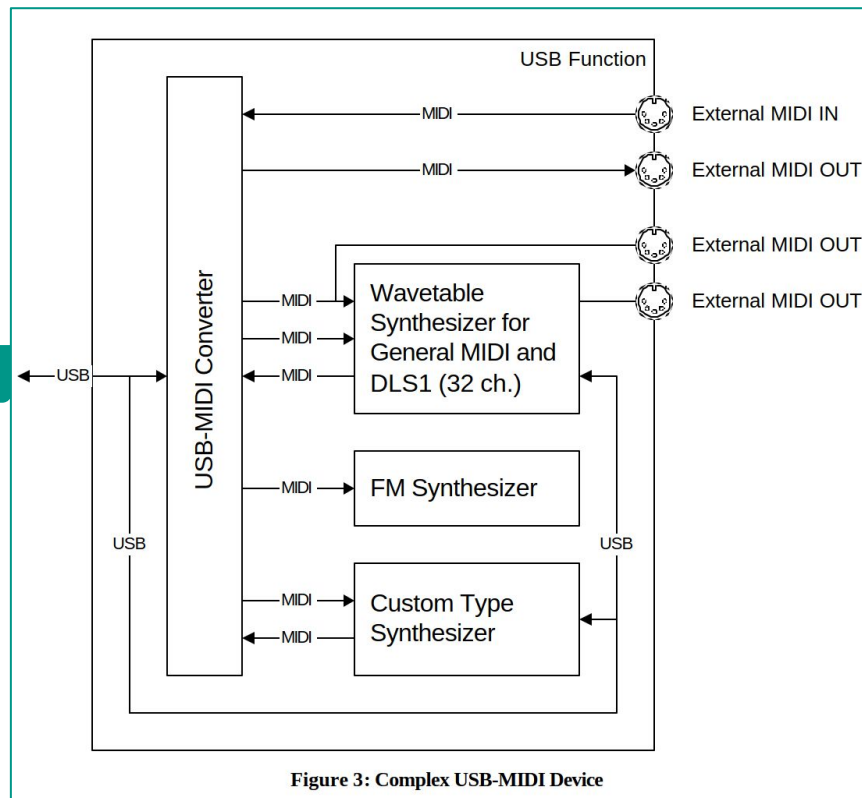
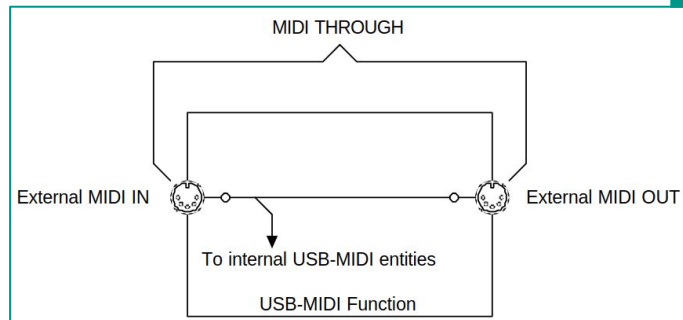


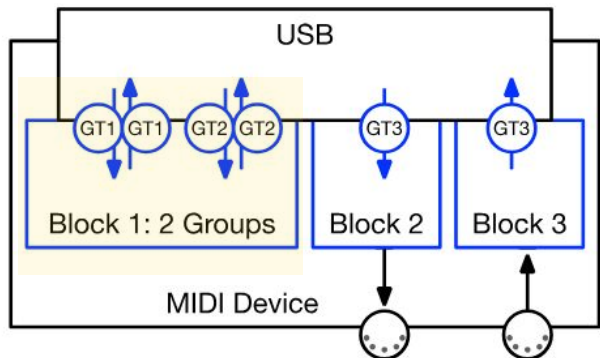
Figure 3: Complex USB-MIDI Device

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

USB-MIDI 2.0 (2020)

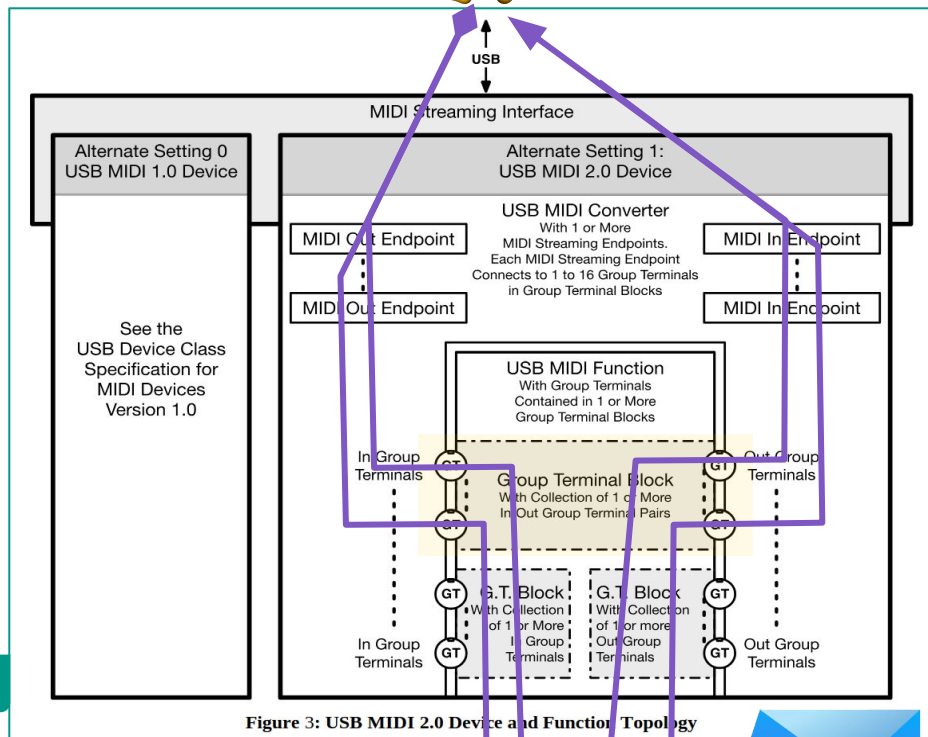
- Simplified operational model
- Group Terminal Blocks
 - Functional relationship b/w ports

32 Channel MIDI Device plus external ports



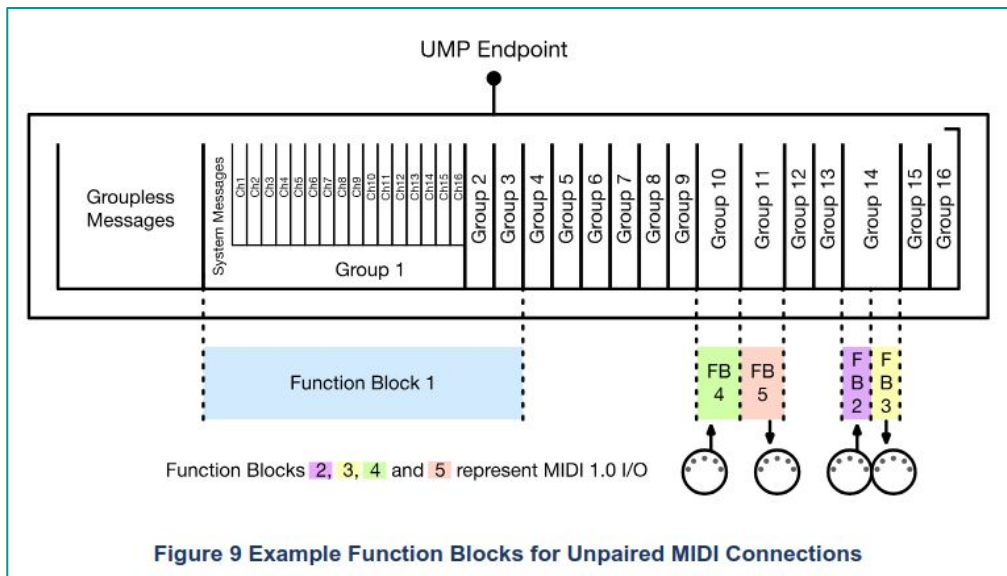
It is not possible to guarantee MIDI-CI pairing with external MIDI jacks so IN and OUT jacks usually should not be members of a bidirectional Group Terminal Block.

[3]



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)



[4]

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

MIDI2

is not

USB-MIDI2.0

is not

USB 2.0



Presentation



Transport



Levels below

ex: send MIDI1 messages with USB-MIDI2 on USB1

Example Universal MIDI Packets

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

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)

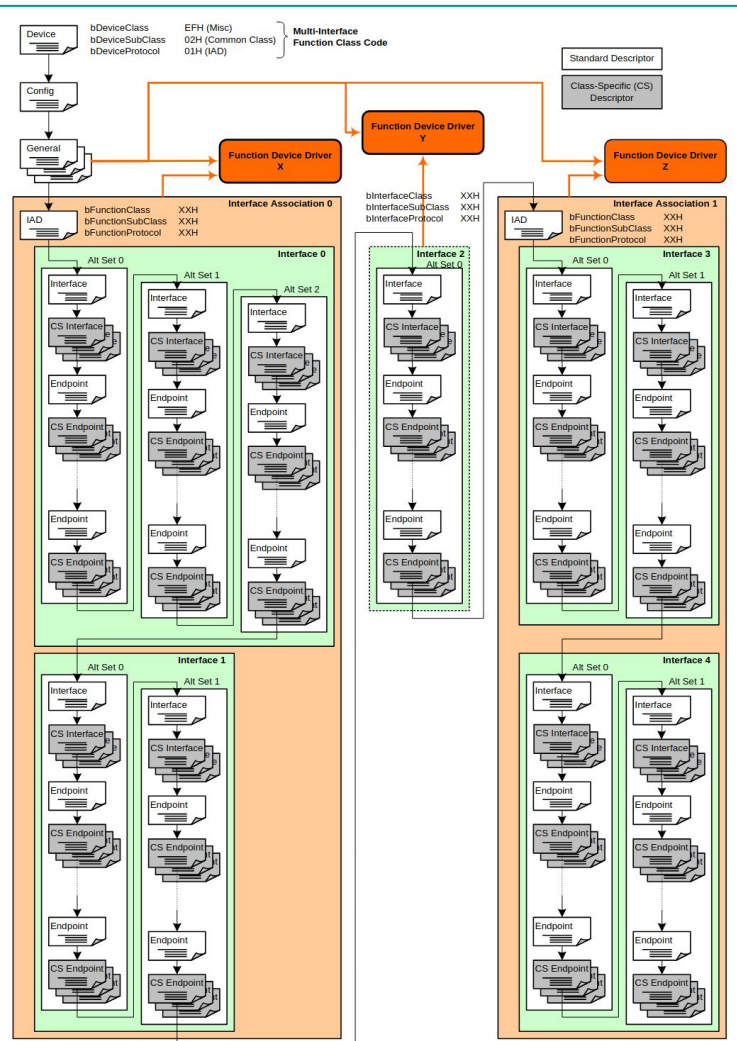
MT=4	Group (0-F)	8 / 9	Channel (0-F)	Note number (0-7F)	Attribute type
Velocity (0-FFFF)				Attribute value	
MSB					LSB

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



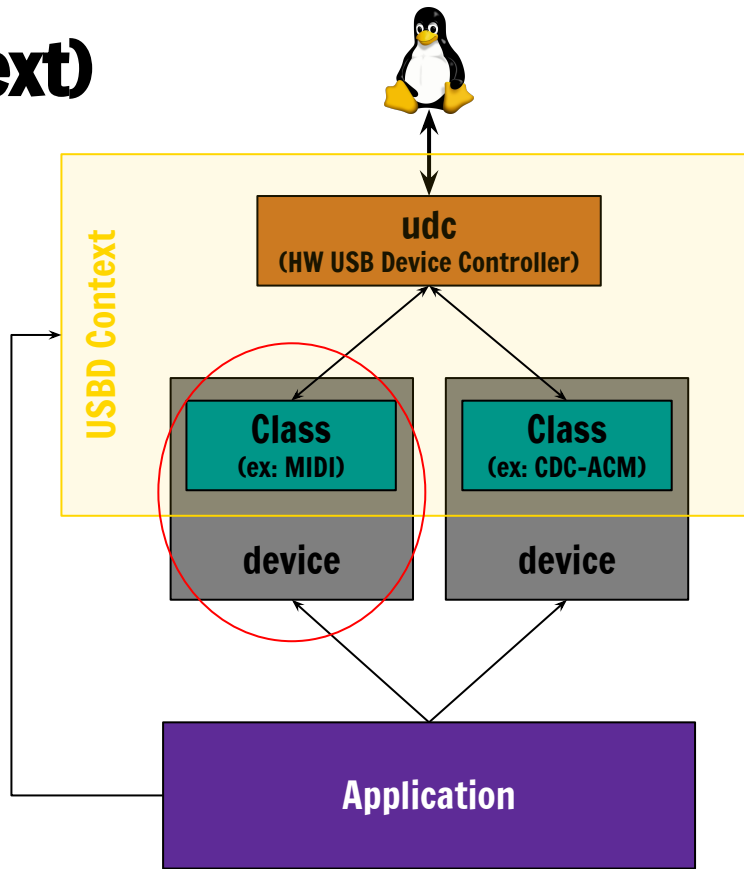
Our goal

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
-

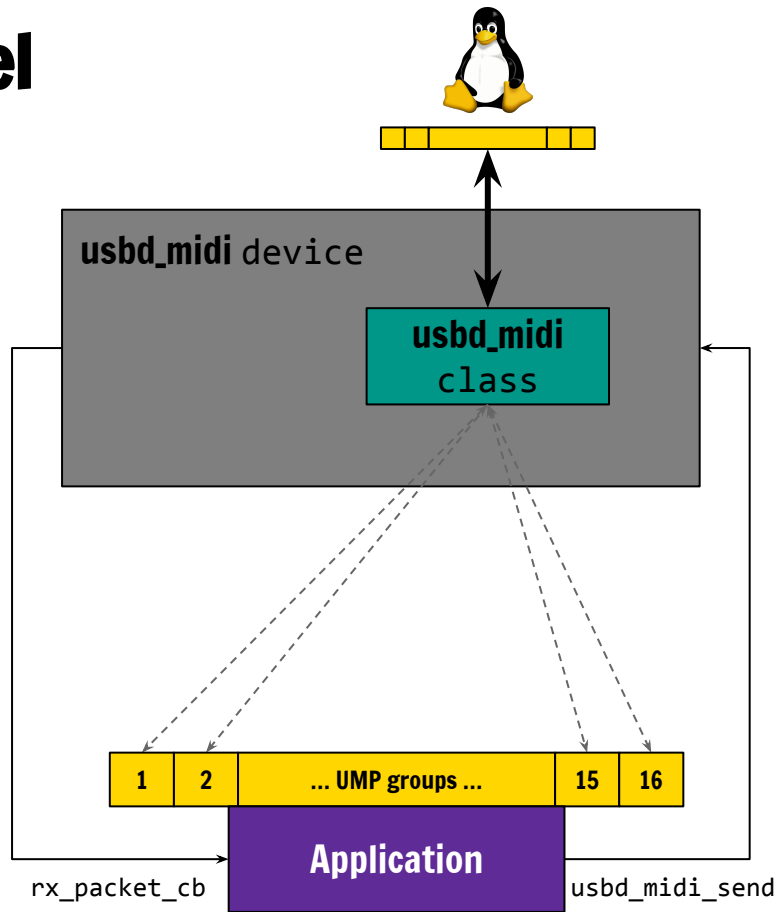
Zephyr udc, usbd (usb device_next)

- **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
	If 0	Alt 0	Audio class-specific
			MIDIStreaming Interface 1.0
			MIDIStreaming class-specific
			Endpoint
			Endpoint class-specific
			Endpoint
	If 1	Alt 0	Endpoint class-specific
			MIDIStreaming Interface 2.0
			MIDIStreaming class-specific
			Endpoint
			Endpoint class-specific
			Endpoint
	If 1	Alt 1	Endpoint class-specific
			MIDIStreaming Interface 2.0
			MIDIStreaming class-specific

**Let's make
some noise !**

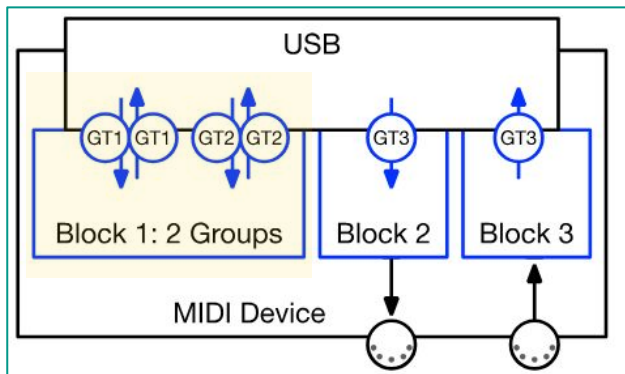


Set up usbd_midi in Zephyr application

app.overlay

prj.conf

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



[3]

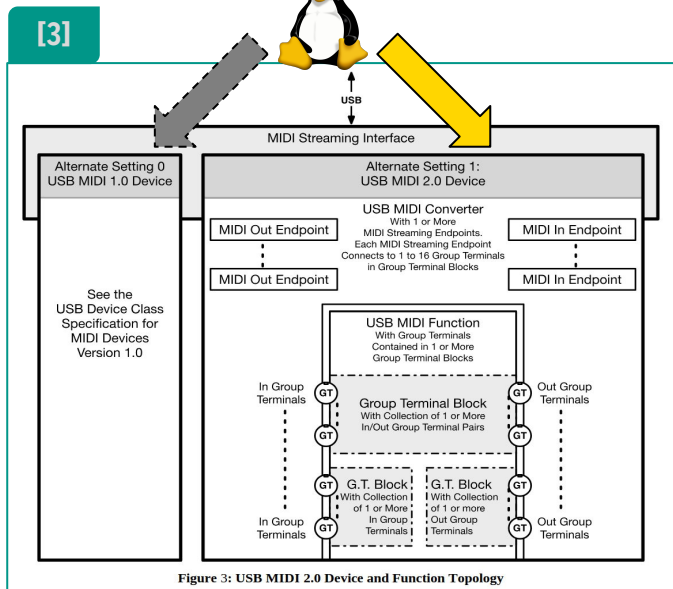
```
/ {  
    usb_midi: usb_midi {  
        compatible = "zephyr,midi2-device";  
        status = "okay";  
        #address-cells = <1>;  
        #size-cells = <1>;  
  
        block1 {  
            reg = <0 2>;  
            protocol = "midi2";  
            terminal-type = "bidirectional";  
            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



```
#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,
};

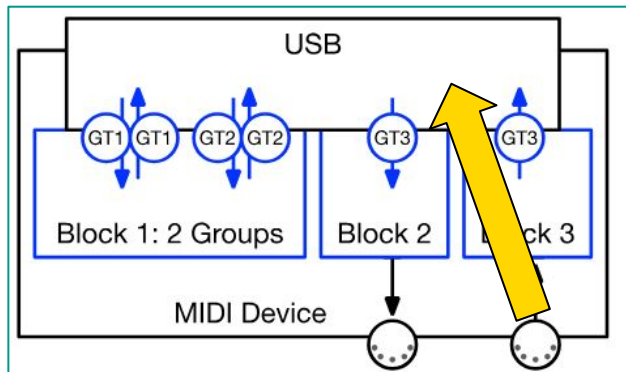
int main(void)
{
    /* ... Initialize application state ... */

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

More in 2 slides...

MIDI1 keyboard

- Leverage input subsys
- Build UMP, then send it



[3]

```
#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

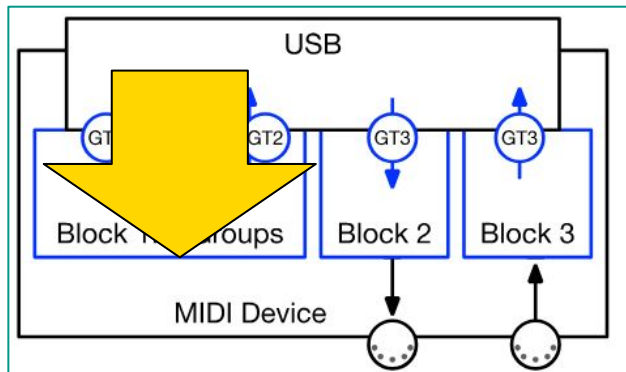
UMP group 3

MIDI chan 1

Velocity = "fortissimo"

MIDI2 Synthesizer

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



[3]

```
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;
    }
}
```

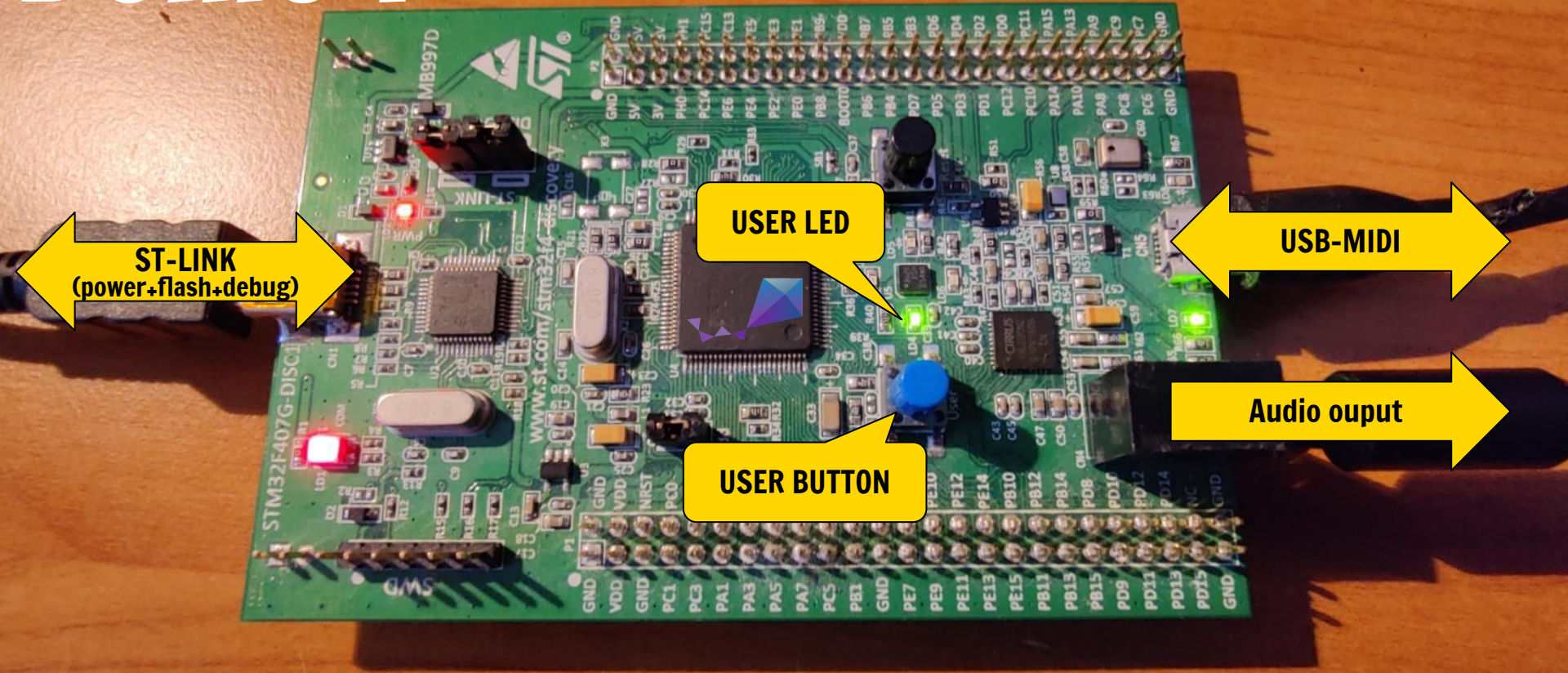
2 groups
x16 chans

Note
pitch

Note
volume

Let's just
assume we
have these

Demo !





USB-MIDI2 is fairly new

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

(Coming in Zephyr 4.1)

Thank you !



Come and visit the Zephyr stand !

- **Building K level 1 (*near the infodesk*)**
 - Meet users (including me) and contributors
 - Demos and showcases
 - Goodies and kites that fly for real



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