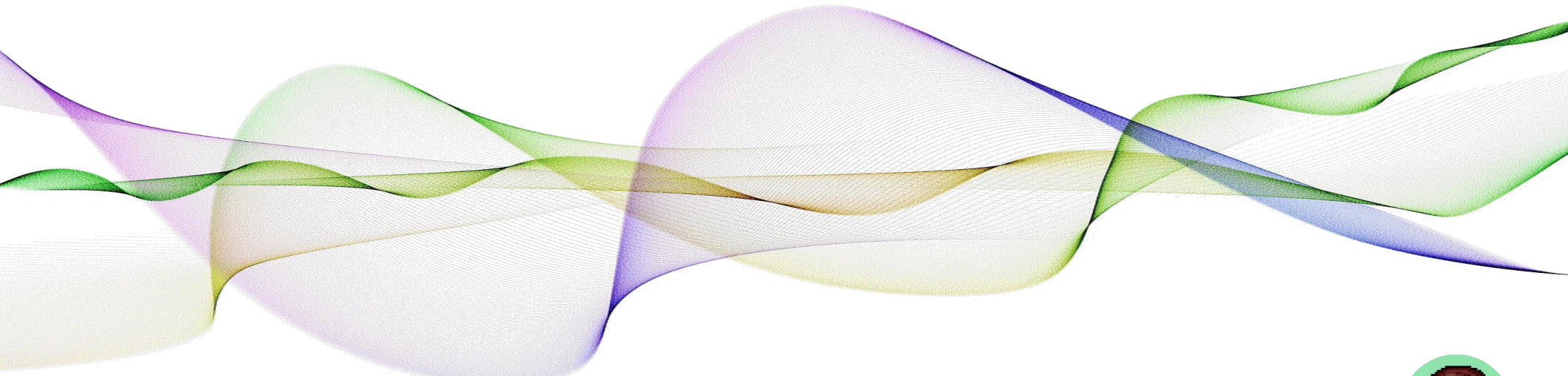
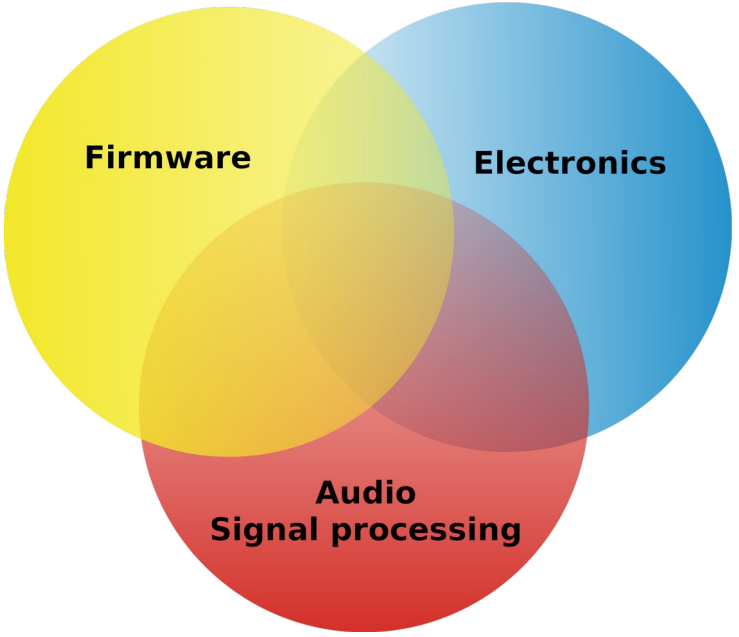
Playing audio on a SoC DAC output with Zephyr





Why is embedded audio complicated?



Goal: Get started with audio on Zephyr!



About us

Eve Redero – Pasqal, Redero Tech

- Electronics & embedded engineer
- Usually working for:
  - Consumer electronics
  - Start-up & scale-up
  - Paris
- ~ 10 years in audio companies

redero.fr



Ugo Marchand – Cloud Ex Machina

- Machine learning engineer
- Start-up aficionado



Outline

- Specific constraints for audio
- User-space implementation attempts
- Driver APIs
- Driver implementation attempts
- Debugging with tracing
- Demos & hardware

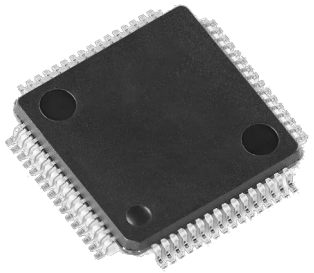


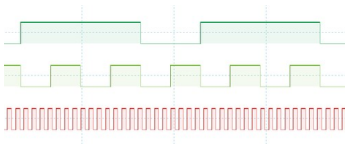
Digital audio design considerations



Audio chain

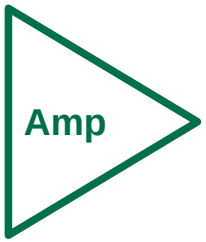
Digital Electronics





Digital-Analog Conversion

Analog Electronics





Acoustics





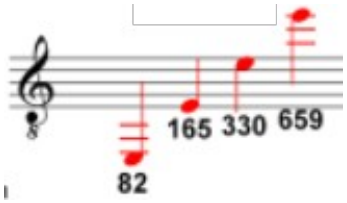




Human-ear frequency range





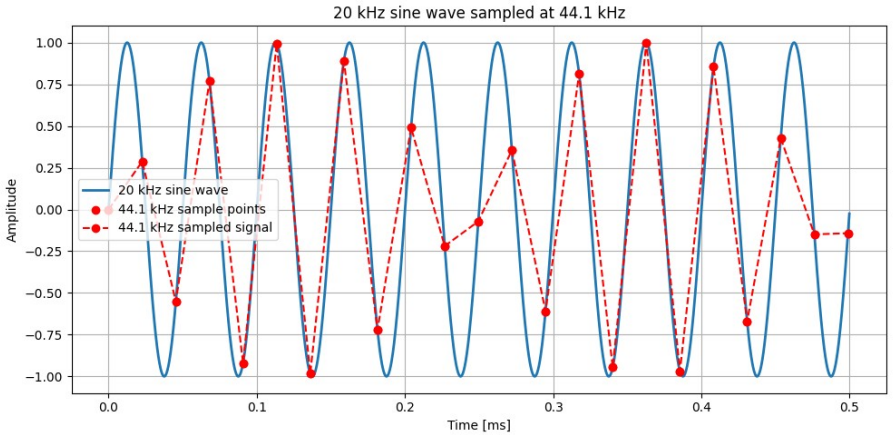


- Full human hearing range:
  20 Hz – 20 kHz
- Little brown bat hearing range:
  10 kHz – 115 kHz
- Usable voice frequency band range:
  300 - 3400 Hz
- Classical guitar notes: typically
  82 - 660 Hz
  But needs harmonics for audio definition

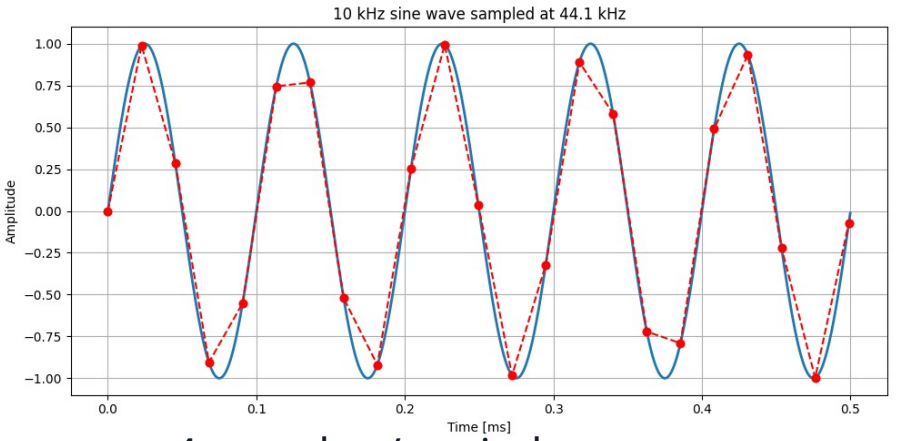


Shannon-Nyquist theorem

- Sampling frequency needs to be at least twice the max frequency you want to recreate
- 2x frequency is not accurate, it’s just to not totally wrong



~ 2 samples / period

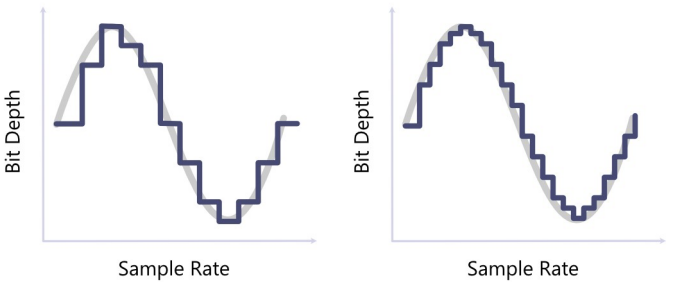


~ 4 samples / period



Digital audio definition

- Sample rate (44.1 kHz, 48 kHz, 96 kHz)
  - Number of samples per seconds
  - Should be « kSPS » and not « kHz »
- Bit depth (16 bits, 24 bits, 32 bits)
  - Number of bits per sample
- Audio CD : **44.1 kHz / 16 bits** (textbook reference)
- Sampling rates of 8 kHz in telephony (voice : max 3.4 kHz)
- Streaming HD: 96 kHz / 24 bits





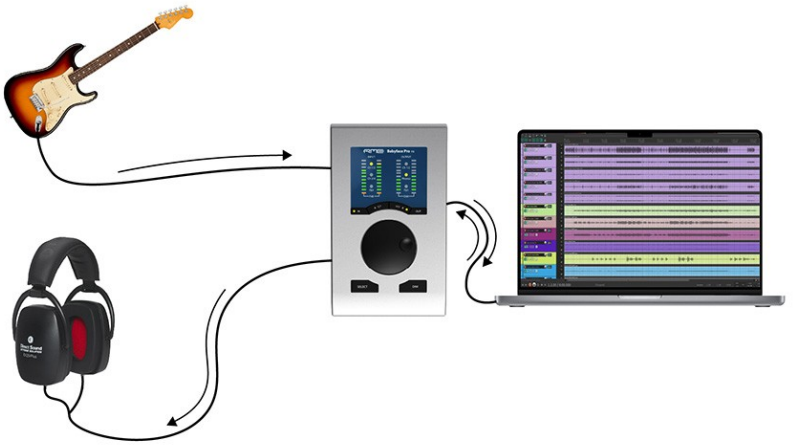
Latency

Humans do not like if the difference between what they see or feel and what they here is too important.

Latency values for live are:

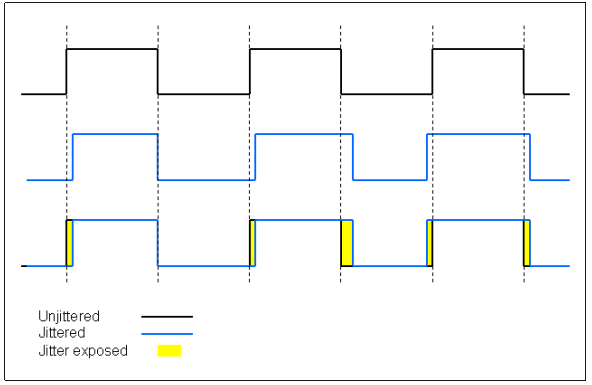
- **< 20 ms**
- < 10 ms preferably

Example: bedroom guitarist setup (Guitar + pedals + soundcard + computer processing + headphones) < 20ms





Jitter sensitivity



- Jitter : clock deviation from true periodicity
- Psychoacoustics say we can perceive down to several hundred nanoseconds jitter
- **< 500 ns** at 48 kHz means a max 2.4% jitter
- Better have the samples delivered on time





Trying to play audio on DAC
Part 1: userspace



The DAC peripheral

- Lots of SoC have an integrated DAC peripheral
- It can generate **analog voltages** (in the SoC analog supply voltage range)
- Can we use it for audio?

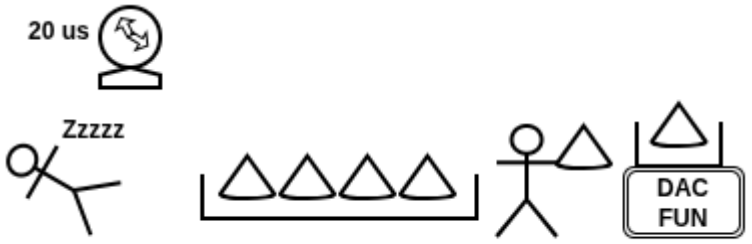
SoC family	Native DAC	Resolution	DMA-capable	Audio-rate capable
STM32 F/L/H/U	Yes	12-bit	Yes	Yes
NXP LPC	Yes	10-12 bit	Yes	Yes
Atmel SAM E/V/S	Yes	12-bit	Yes	Yes
ESP32	Yes	8-bit	Limited	Marginal
nRF	No			
RP2040	No			

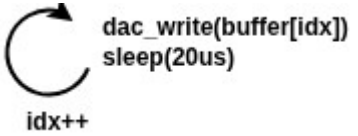


Experiment 1: for loop and sleep

Let’s start coding an audio driver in Zephyr

- Naive implementation
- For 48 kHz, SAMPLE_PERIOD_USEC = 20us
- For loop, with k_sleep
- Tested with oscilloscope, speaker and trace






```c
sample_index = 0;

while (1) {
        /* Custom trace event at every loop iteration. */
        sys_trace_named_event("sample", sample_index, 1);

        uint16_t value =
                wave_table[sample_index % ARRAY_SIZE(wave_table)];

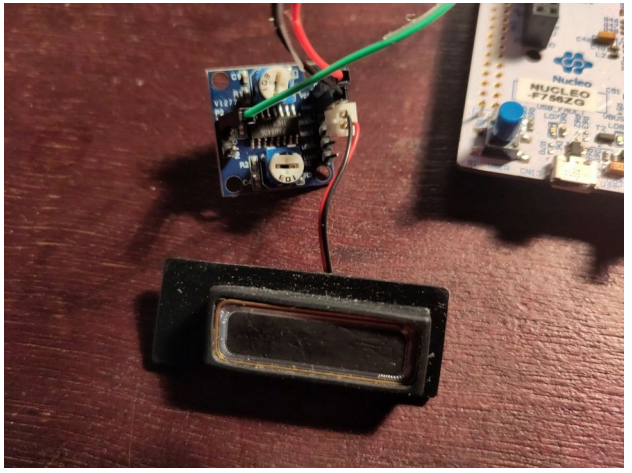
        dac_write_value(dac_dev, DAC_CHANNEL_ID, value);
        k_sleep(K_USEC(SAMPLE_PERIOD_USEC));

        sample_index++;
}
```



Experiment setup

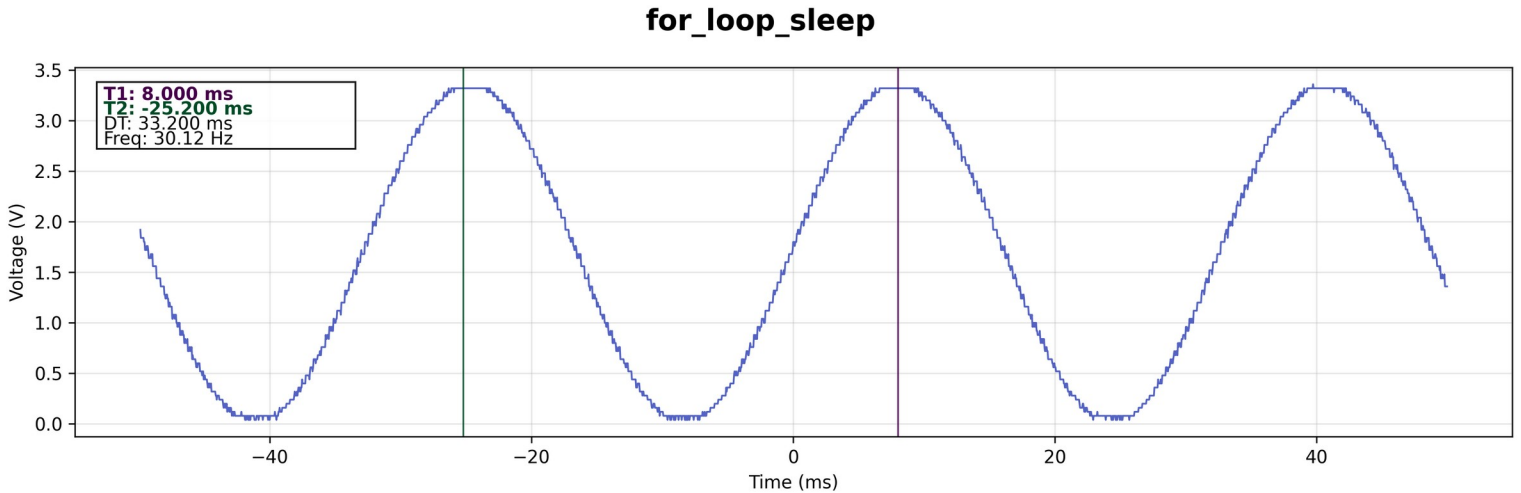
- Analog amp mini eval kit
- Small speaker (from a teardown)
- ST Nucleo F756ZG
  - Amp supply from USB 5V
  - Amp input on DAC output





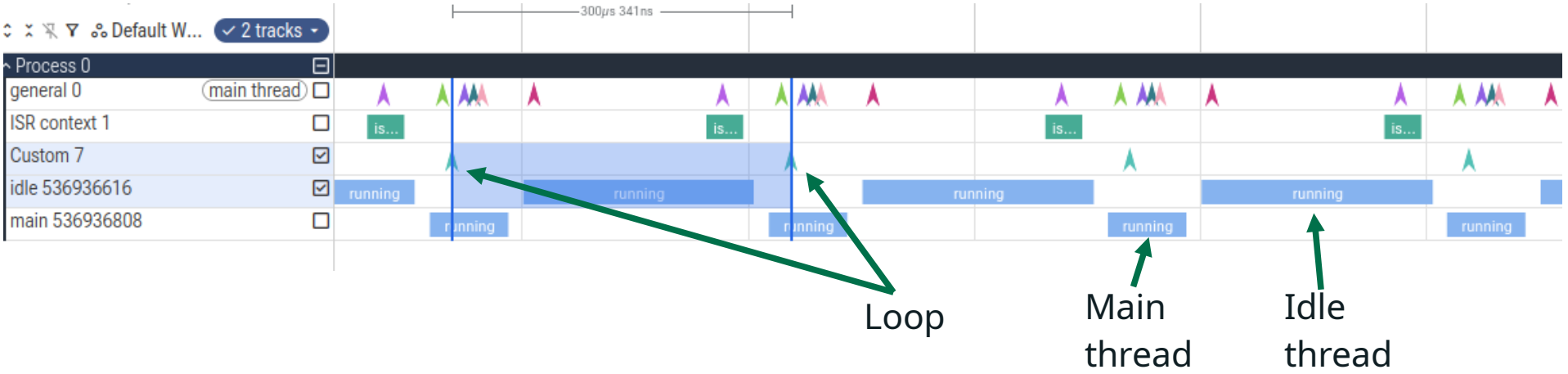
Experiment 1: for loop and sleep

- 30 Hz instead of 440 Hz
- A small speaker cannot even play this





Experiment 1
for loop and sleep



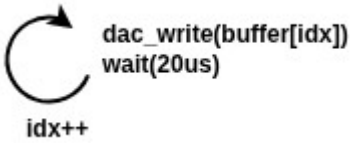
- **300us between 2 loop iterations instead of 20us**
- Sound is bad
- ~200us spent in idle thread
  - Minimum time unit is systick
  - Systick = 100us

SysTick: periodic interrupt used by the RTOS kernel to perform time-related tasks



Experiment 2: for loop and wait

- k_busy_wait instead of k_sleep
- More precise
- Will not take naps in IDLE anymore






```
sample_index = 0;

while (1) {
        /* Custom trace event at every loop iteration. */
        sys_trace_named_event("sample", sample_index, value);

        uint16_t value =
                wave_table[sample_index % ARRAY_SIZE(wave_table)];

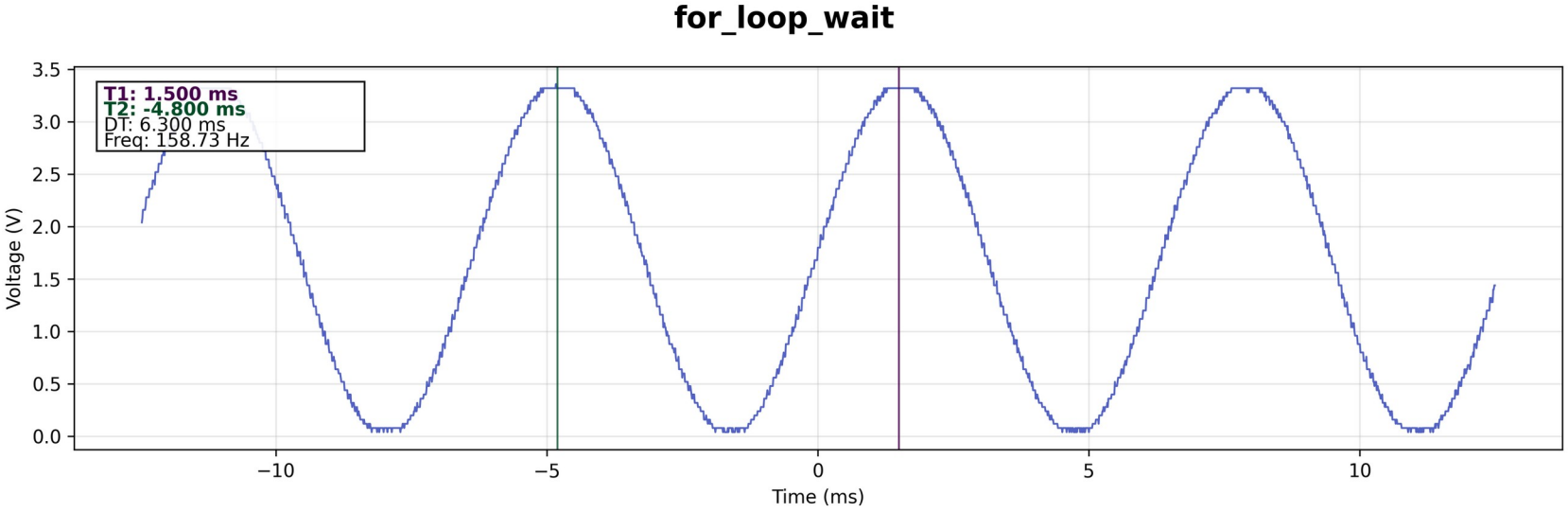
        ret = dac_write_value(dac_dev, DAC_CHANNEL_ID, value);

        k_busy_wait(SAMPLE_PERIOD_USEC);
        sample_index++;
}
```



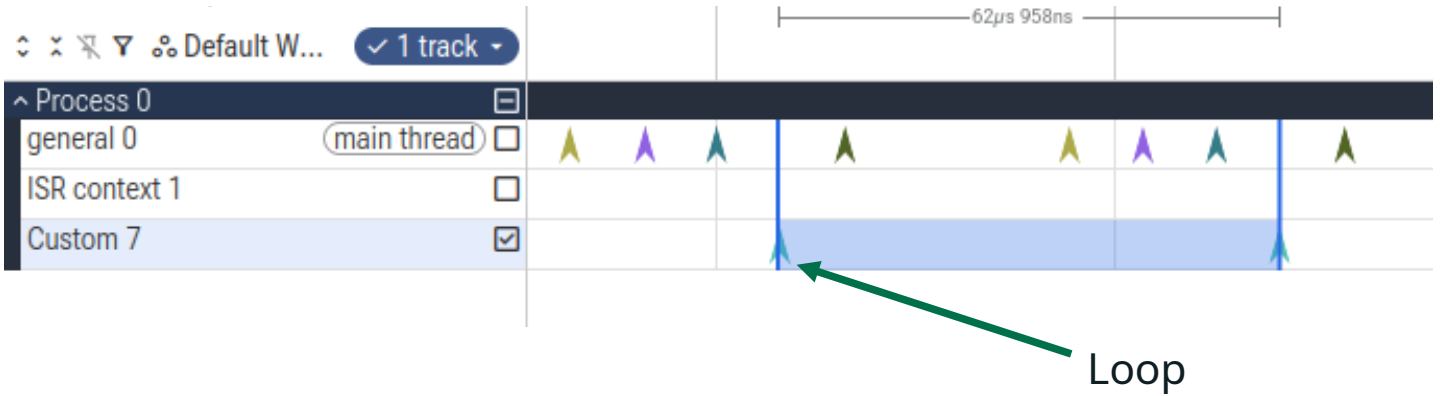
Experiment 2: for loop and wait

- 160 Hz instead of 440 Hz
- Faster, but still wrong





Experiment 2: for loop and wait

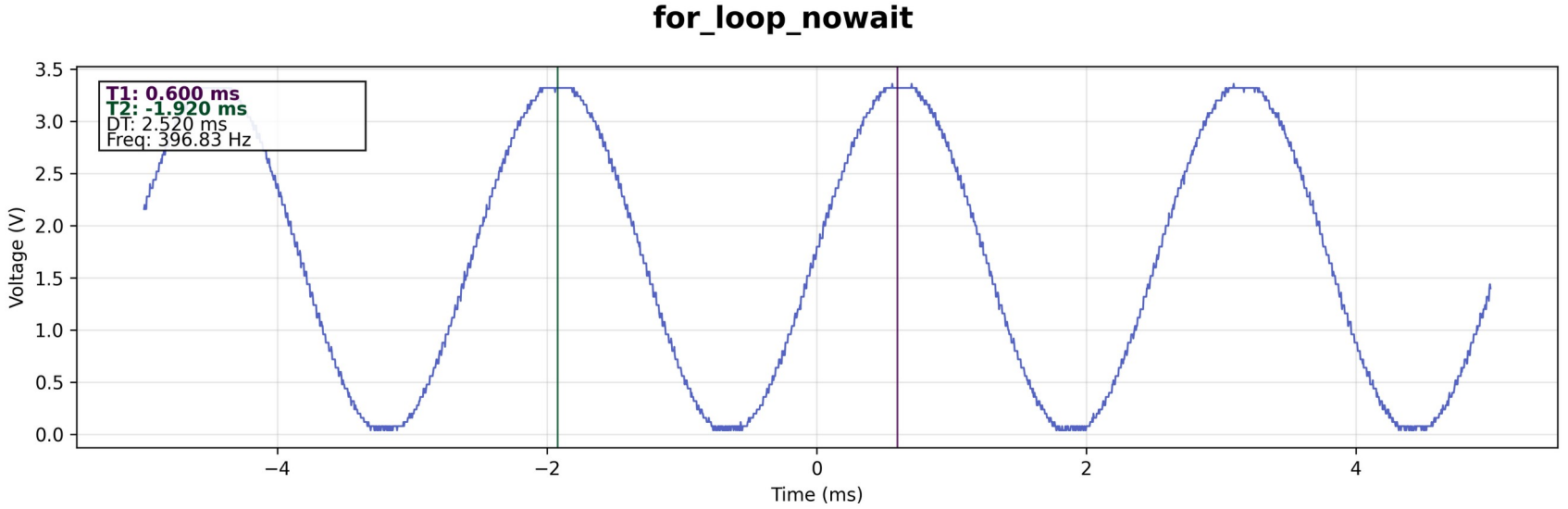


- 62us between loop iterations instead of 20us
- Busy wait is ~30us
- One fully busy thread



Experiment 2 bis: for loop, no wait

- 396 Hz instead of 440 Hz
- Still not right





Experiment 2 bis: for loop, no wait



- Removed all sleeps and waits
- Still 25us loop period > 20us
- One fully busy thread still
- Taking 8us between « dac_write_value » enter and exit
- We need to take this to a lower level



Trying to play audio on DAC
Part 2: driver



Zephyr driver API

- Abstraction for drivers
  - Factorization among different devices
  - All different brands and models are called by same functions, from a high-lever perspective

Exemple:

- Flash (in include/zephyr/drivers/flash.h)
  - flash_read
  - flash_write
  - flash_erase
- Working for both:
  - internal flash (flash_stm32h7x.c)
  - external flash (spi_flash_at25xv021a.c)



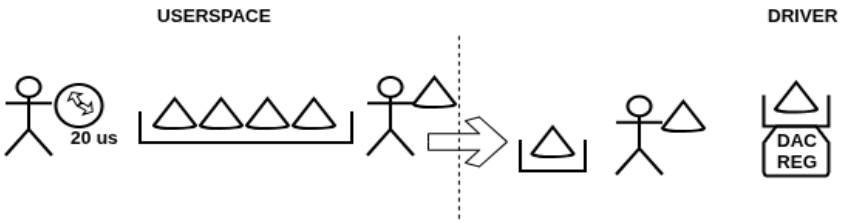


DAC driver API

- Zephyr has an API for « vanilla » DAC use: output an analog sample
- zephyr/include/zephyr/drivers/dac.h
- 2 functions:
  - channel_setup
  - write_value (one uint32_t)
- Designed for DC value output

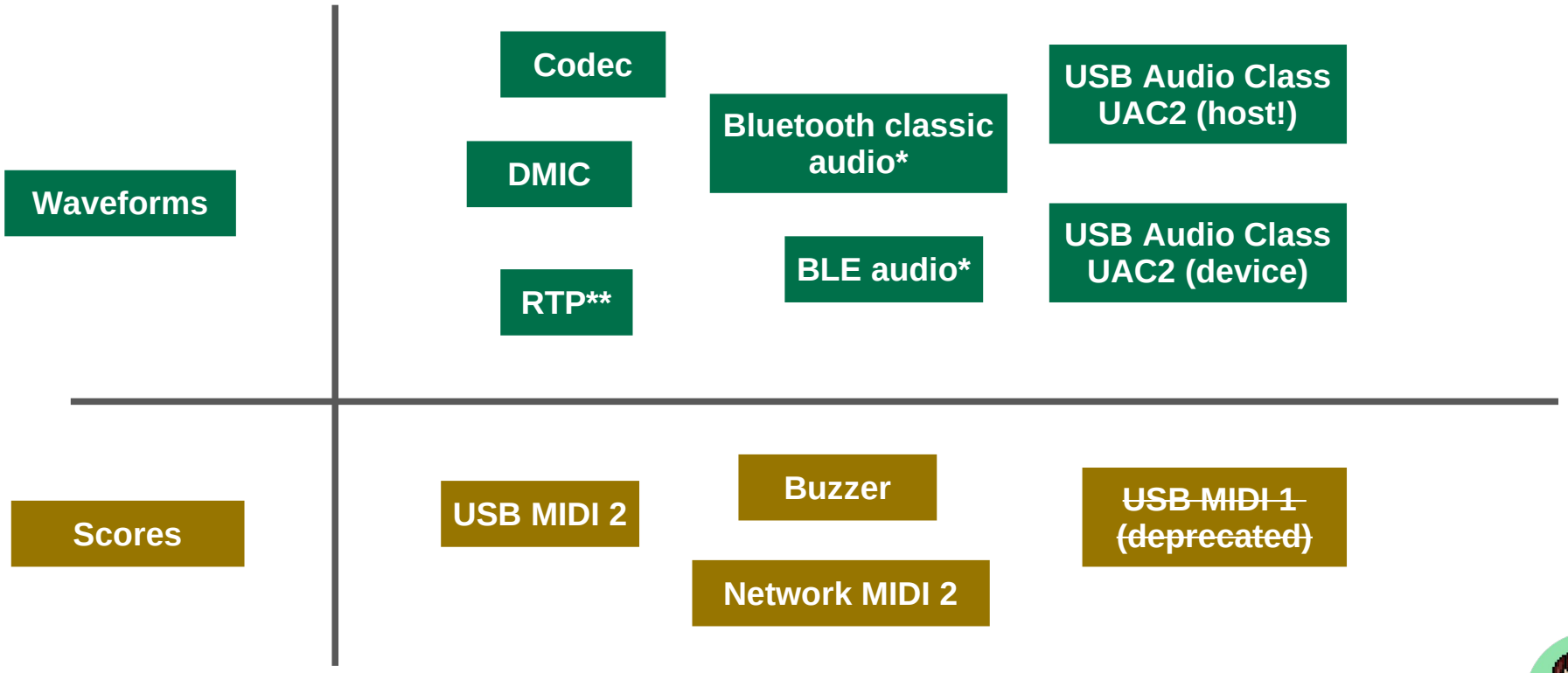
  Should we upgrade this?

```
typedef int (*dac_api_write_value)(const struct device *dev,
                                   uint8_t channel, uint32_t value);
```





Zephyr audio-related APIs





Including all their sub-profiles: BAP, CAP, LC3, A2DP, HFP…

Application layer for audio over UDP/IP

Codec driver API

- Functions to play audio
  - write (to send data for playback)
  - start
  - stop
- Setup functions
  - set_property / apply_properties
  - route_input
  - route_output
- Callbacks
  - register_error_callback
  - register_done_callback

Should we use this? Is it API legit use or abuse?





Codec API properties

Codec.h has all the structure and enums we need for audio!

```c
/**
 * PCM audio sample rates
 */
typedef enum {
        AUDIO_PCM_RATE_8K = 8000,       /**< 8 kHz sample rate */
        AUDIO_PCM_RATE_11P025K = 11025, /**< 11.025 kHz sample rate */
        AUDIO_PCM_RATE_16K = 16000,     /**< 16 kHz sample rate */
        AUDIO_PCM_RATE_22P05K = 22050,  /**< 22.05 kHz sample rate */
        AUDIO_PCM_RATE_24K = 24000,     /**< 24 kHz sample rate */
        AUDIO_PCM_RATE_32K = 32000,     /**< 32 kHz sample rate */
        AUDIO_PCM_RATE_44P1K = 44100,   /**< 44.1 kHz sample rate */
        AUDIO_PCM_RATE_48K = 48000,     /**< 48 kHz sample rate */
        AUDIO_PCM_RATE_96K = 96000,     /**< 96 kHz sample rate */
        AUDIO_PCM_RATE_192K = 192000,   /**< 192 kHz sample rate */
} audio_pcm_rate_t;

/**
 * PCM audio sample bit widths
 */
typedef enum {
        AUDIO_PCM_WIDTH_16_BITS = 16, /**< 16-bit sample width */
        AUDIO_PCM_WIDTH_20_BITS = 20, /**< 20-bit sample width */
        AUDIO_PCM_WIDTH_24_BITS = 24, /**< 24-bit sample width */
        AUDIO_PCM_WIDTH_32_BITS = 32, /**< 32-bit sample width */
} audio_pcm_width_t;
```

```c
typedef enum {
        AUDIO_CHANNEL_FRONT_LEFT,
        AUDIO_CHANNEL_FRONT_RIGHT,
        AUDIO_CHANNEL_REAR_LEFT,
        [...]
} audio_channel_t;
```

```c
typedef enum {
        AUDIO_PROPERTY_OUTPUT_VOLUME,
        AUDIO_PROPERTY_OUTPUT_MUTE,
        AUDIO_PROPERTY_INPUT_VOLUME,
        AUDIO_PROPERTY_INPUT_MUTE,
        AUDIO_PROPERTY_EQ_GAIN
} audio_property_t;
```



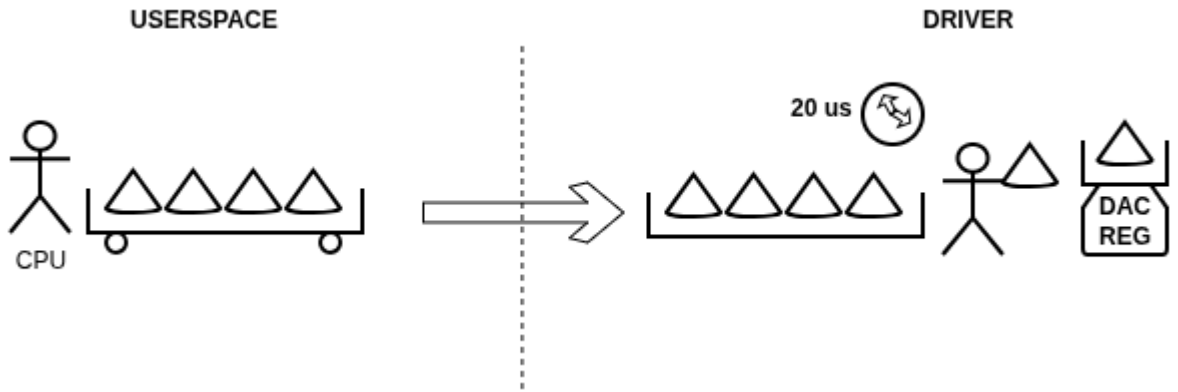
Experiment 3: DAC driver with interrupt

In the user code:

- Passing a buffer to the driver
- Pressing « play »

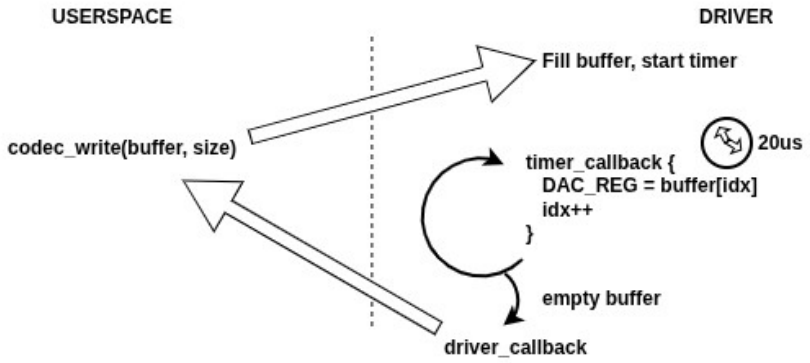
In the codec driver:

- Setting an interrupt to feed the DAC





DAC driver with interrupt




```
audio_codec_write(dev, audio_data_p, AUDIO_BLOCK_SIZE);
audio_codec_start(dev, AUDIO_DAI_DIR_TX);
```

```
static void stm32_dac_counter_callback(const struct device *counter_dev, void *user_data)
{
        [...]
        dac_data->play_index++;

        if ((dac_data->play_index != samples)) {
                return;
        }
        [...]

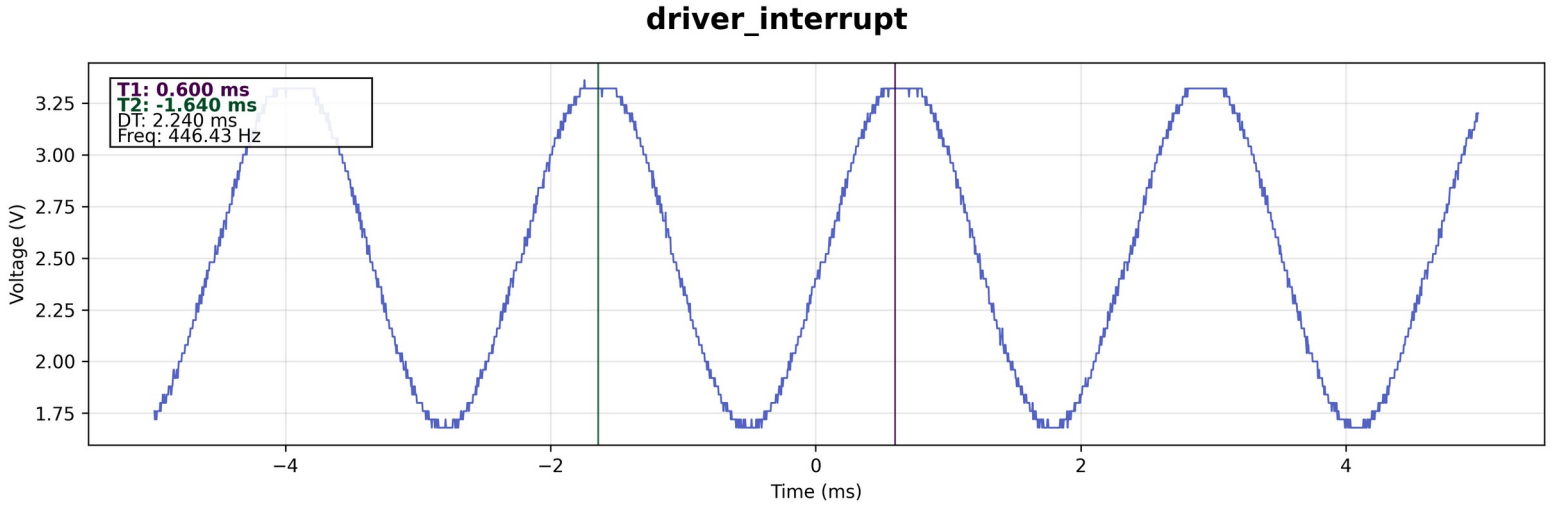
        if (dac_data->tx_cb != NULL) {
                dac_data->tx_cb(codec_dev, dac_data->tx_cb_user_data);
        }

}
```



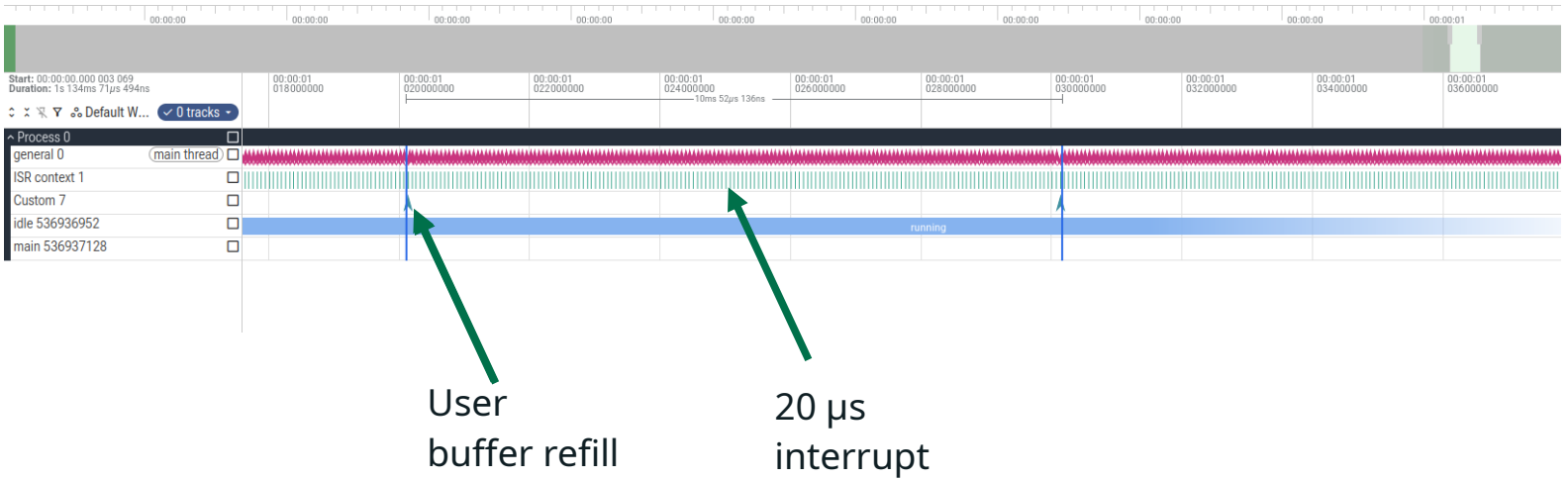
DAC driver with interrupt

- 440 Hz
- A bit noisy





DAC driver with interrupt



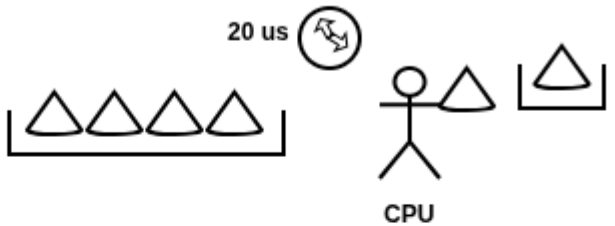
- 20 µs, the timing matches!
- We even have some spare time to run threads in parallel
- But the sound quality is not good: jitters



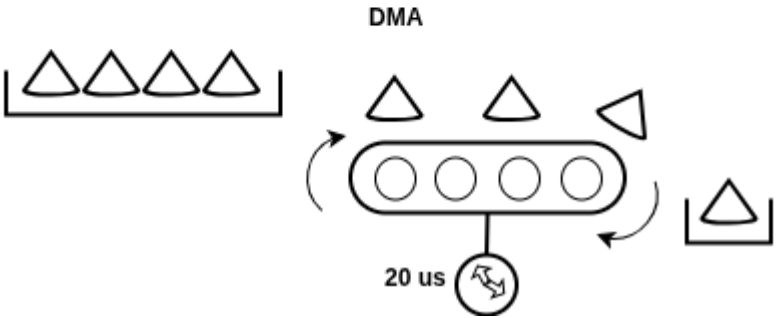
DMA peripherals

- Direct Memory Access
- Data conveyor belt
- Offloads CPU
- Perfect for audio, video, sensors

CPU moving samples on timer interrupt

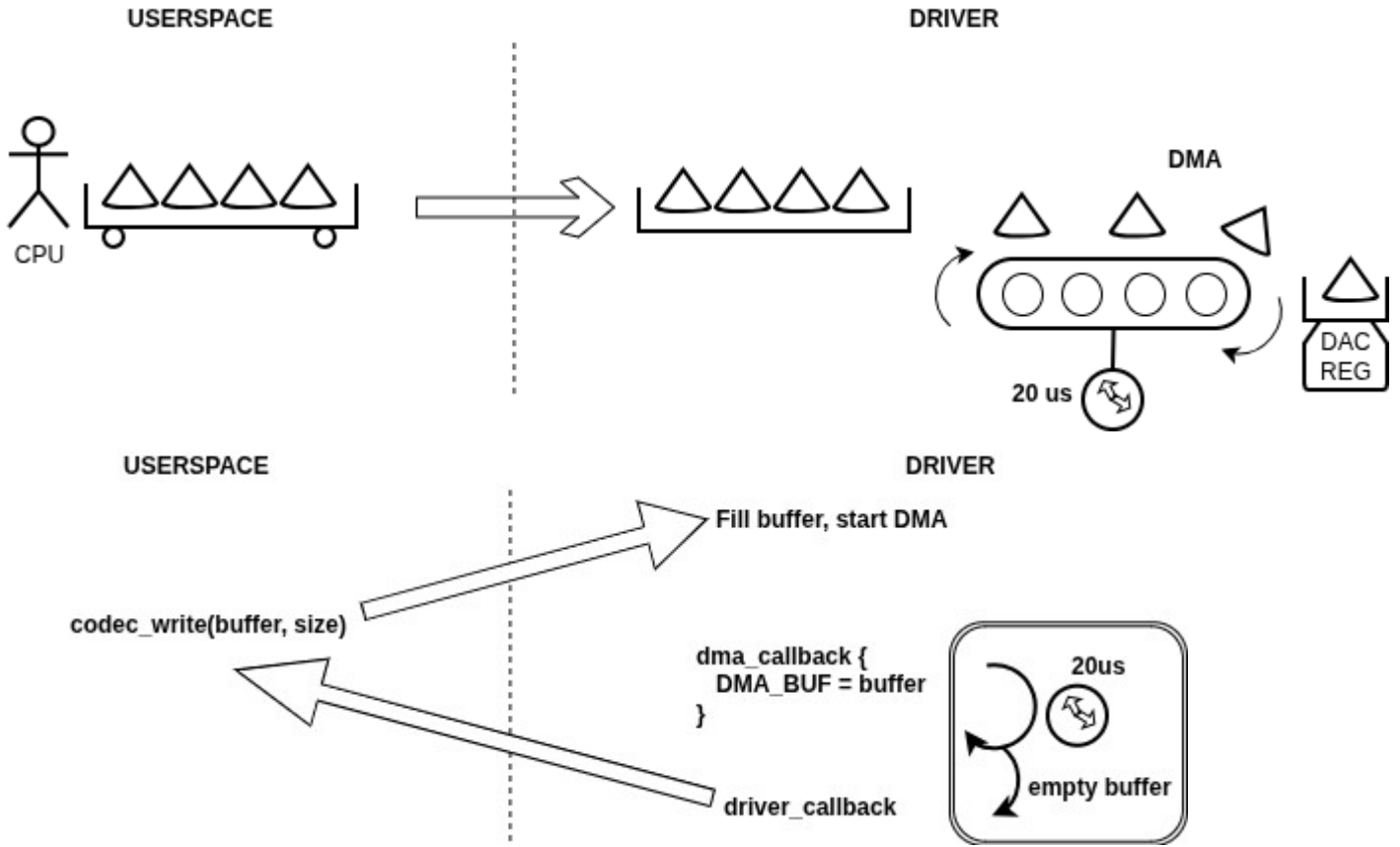


DMA moving samples automatically





Experiment 4: DAC driver with DMA





DAC driver with DMA Playback

- No API change!
  User application stays the same

```c
static void tx_done(const struct device *dev, void *user_data)
{
        audio_codec_write(dev, audio_data_p, AUDIO_BLOCK_SIZE);
        audio_data_p += AUDIO_BLOCK_SIZE;
    }
}
```

```c
static void stm32_dac_dma_callback(const struct device *dev, void *user_data, uint32_t channel,
                                   int status)
{
        [...]
        dac_data->write_buf = dac_data->buf;
        dac_data->writable = true;

        if (dac_data->tx_cb != NULL) {
                dac_data->tx_cb(codec_dev, dac_data->tx_cb_user_data);
        }
}
```

```c
static int stm32_dac_write(const struct device *dev, uint8_t *data, size_t data_size)
{
        [...]
        uint16_t *dst = dac_data->write_buf;
        size_t src_samples = data_size / STM32_DAC_BYTES_PER_SAMPLE;
        size_t dst_samples = dac_data->config.dai_cfg.pcm.block_size / STM32_DAC_BYTES_PER_SAMPLE;
        [...]

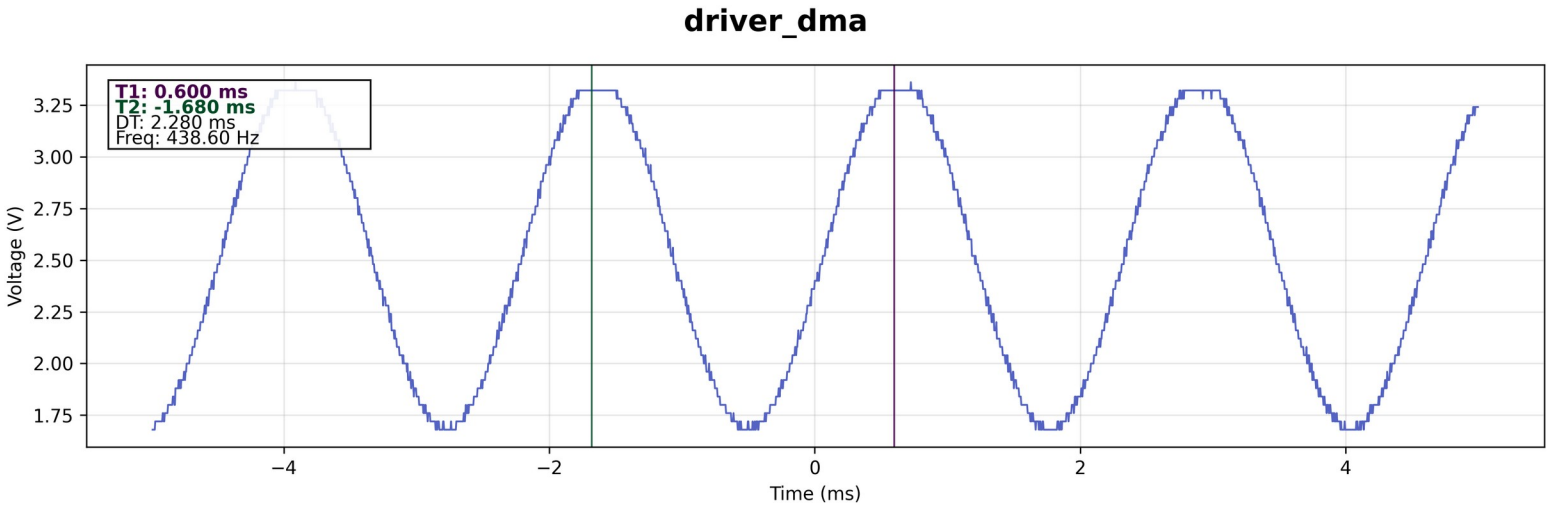
        for (size_t i = 0; i < src_samples; i++) {
                dst[i] = sys_get_le16(&data[2 * i]) ^ STM32_DAC_PCM_SIGN_BIT;
        }
        memset(&dst[src_samples], 0, (dst_samples - src_samples) * sizeof(*dst));

        dac_data->writable = false;

        return 0;
}
```

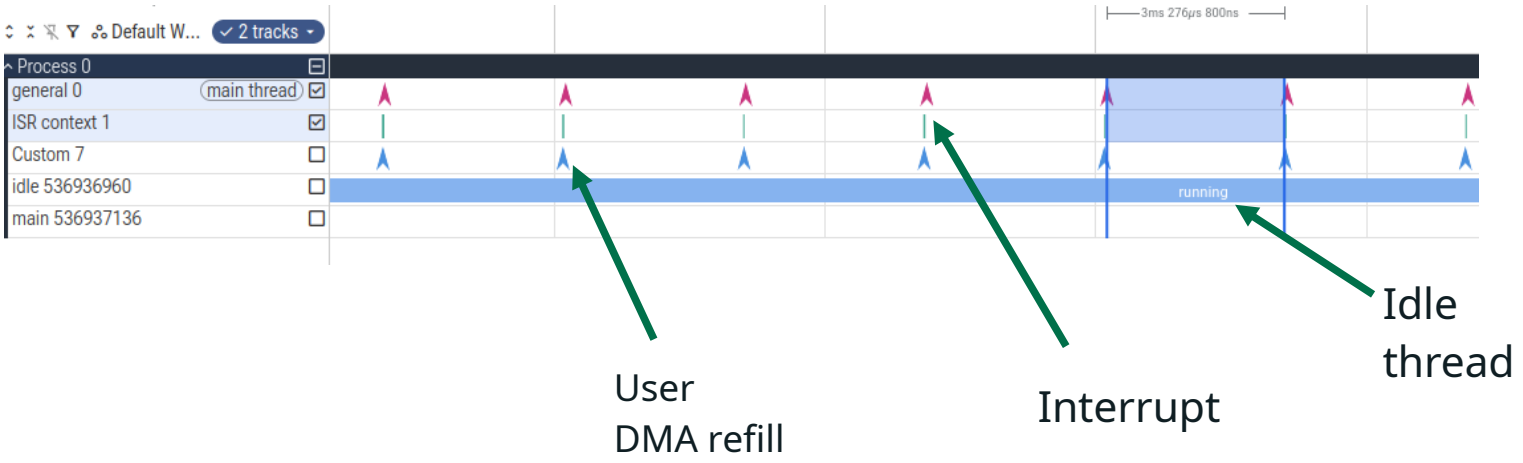
DAC driver with DMA

- 440 Hz
- Good quality





DAC driver with DMA



- DMA activated, ISR + custom event at DMA empty IRQ
- 3 ms between 2 DMA interrupts, playing 160 samples (320 bytes) = 48 kHz



DMA shortcomings

- Not easy to configure
- Not easy to observe
  - No CPU instructions → no breakpoints
  - Only sign of life: interrupts
  - Breakpoint on interrupts → bad idea
  - DMA may continue to run when execution stops!
- … which is where tracing is useful

```
dac_data->dma_block = (struct dma_block_config){
        .source_address = (uint32_t)dac_data->buf,
        .dest_address = LL_DAC_DMA_GetRegAddr(dac_cfg->dac_base, dac_cfg->dac_ll_channel,
                                              LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED),
        .block_size = 2U * dac_data->config.dai_cfg.pcm.block_size,
        .source_addr_adj = DMA_ADDR_ADJ_INCREMENT,
        .dest_addr_adj = DMA_ADDR_ADJ_NO_CHANGE,
        .source_reload_en = 1,
        .dest_reload_en = 1,
};

dac_data->dma_cfg = (struct dma_config){
        .dma_slot = dac_cfg->dma_slot,
        .channel_direction = MEMORY_TO_PERIPHERAL,
        .source_data_size = STM32_DAC_BYTES_PER_SAMPLE,
        .dest_data_size = STM32_DAC_BYTES_PER_SAMPLE,
        .block_count = 1,
        .head_block = &dac_data->dma_block,
        .dma_callback = stm32_dac_dma_callback,
        .user_data = (void *)dev,
};

int ret = dma_config(dac_cfg->dma_dev, dac_cfg->dma_channel, &dac_data->dma_cfg);
```



My Zephyr tracing how-to



What is tracing?

Measurement disturbs the system

- Debugging:
  - Stops execution on breakpoints
- Tracing:
  - The system runs, create a footprint
  - Almost real-time
  - Possible to retrieve the data later

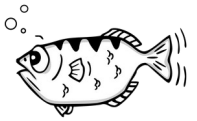




Method 1: Trace to RAM

Requires a **debugger probe**

- Zephyr trace subsystem
- Each time a trace point is reached, write some bits to RAM buffer
- Fast
  - CTF: minimalist binary format
  - Can even trace interrupts



```
.bss.ram_tracing
                0x20011354    0x10000 zephyr/libzephyr.a(tracing_backend_ram.c.obj)
                0x20011354              ram_tracing
```

- Use a GDB session
  - The buffer address is « ram_tracing »
  - Retrieve your RAM buffer directly

```
gdb> dump binary memory data/mytrace.ctf
           ram_tracing ram_tracing+65536
```

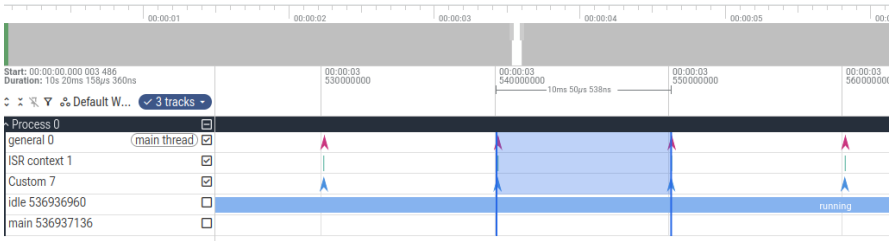
- Use ctf2perfetto and the Perfetto web UI for data visualisation

```
python3 scripts/tracing/ctf2perfetto.py data/mytrace.ctf
```



Custom trace event

- Custom trace event record :
  - a name
  - 2 uint32_t
- Can be anything you want!
- No extra Kconfig required
- A bit chunky




```c
#include <zephyr/tracing/tracing.h>

static void stm32_dac_dma_callback(const struct device *dev, void *user_data, uint32_t channel,
                                   int status)
{
        sys_trace_named_event("dac_dma_cb", channel, (uint32_t)status);
```



Method 2: GPIO tracing

Requires an **oscilloscope** or **logic analyzer**

- Set and reset a GPIO
  - gpio_pin_set has overhead
  - Prefer using direct register access
- Read it with oscilloscope or logic analyzer
- Like tracing, but hardware
- Good for audio applications
- Available on any system
- 2 options:
  - Configure a user trace hook in Zephyr
  - Or just toggle the GPIO in code :)

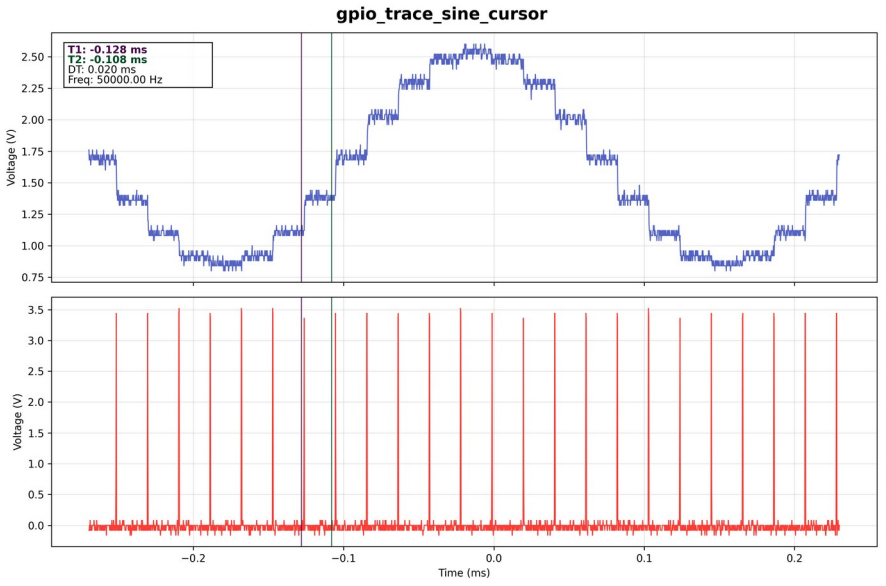
```
#define STM32_DAC_TRACE_HIGH(cfg) \
   LL_GPIO_SetOutputPin(GPIOA, LL_GPIO_PIN_5)
#define STM32_DAC_TRACE_LOW(cfg) \
   LL_GPIO_ResetOutputPin(GPIOA, LL_GPIO_PIN_5)
```



GPIO tracing: sample

- Code sample:
  DAC refill with an interrupt, no DMA
- GPIO:
  - set high at DAC interrupt start
  - set low at the end
- Measured period:
  - every ~20 µs, matching 48kHz

DAC output



Interrupt trace GPIO



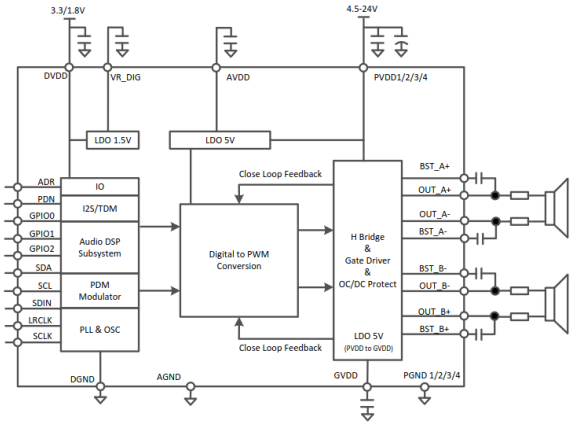
Why use an internal DAC peripheral for audio?

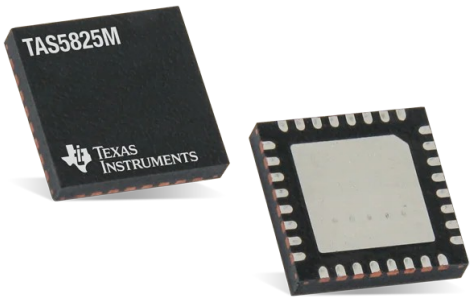


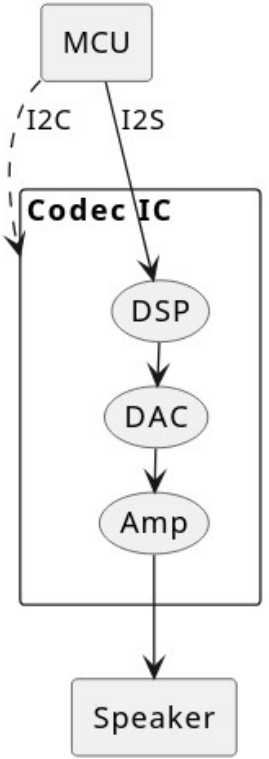
External Audio ICs

Need high-fidelity audio?

- Specialized IC can integrate DACs, ADC, DSP, amplifiers…
- I2S or TDM digital audio communication
- Example: TI TAS5825M « Digital Input Stereo Amp »





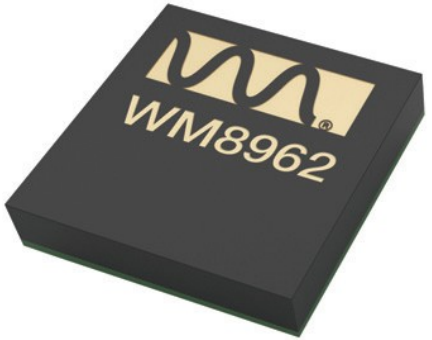


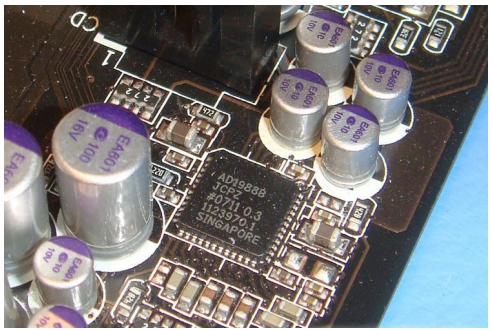


« Codec »

- Codec: Coder-Decoder
- In Zephyr audio driver context, used to name external ICs
  - Using digital audio interface
    - I2S: Inter-IC Sound
    - TDM: Time-Data Multiplexed
  - Embedding ADCs and / or DACs
- Some vendors call them ADC, DAC or amplifiers

- In the context of video, it means « a hardware or software block to encode/decode a media container or compression » (MPEG-4, H-265…)





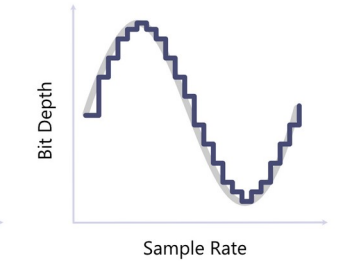


Digital audio interfaces

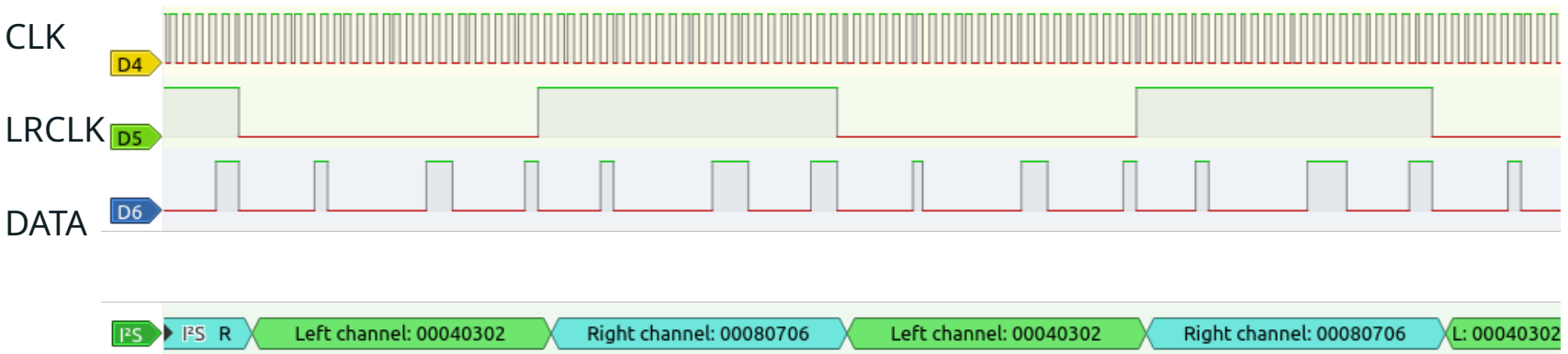
I2S: Audio transmission is:

- Synchronised
- Digital
- Muxed
- Not compressed

TDM: similar, with more channels muxed



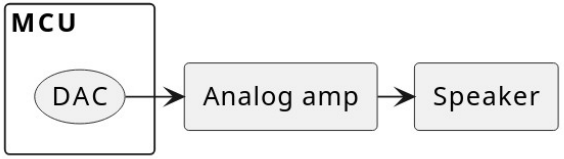
⚠ Your SoC I2S/TDM peripheral can also be called SAI, SSC, SSIE, McASP...



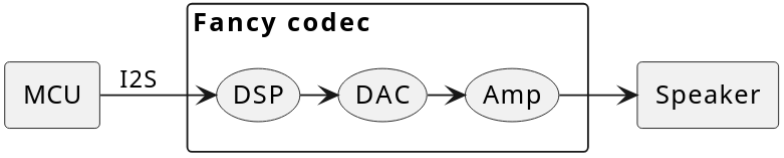


Two possible architectures

Analog amplifier + internal DAC



Digital amplifier



2 versions of the same Bluetooth speaker:

- JBL Flip 4:
  TI TPA3130 (analog input)

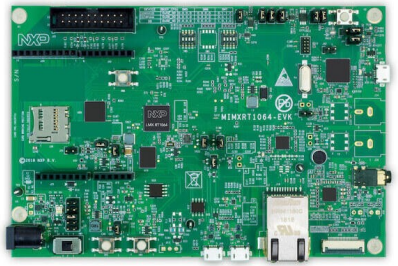


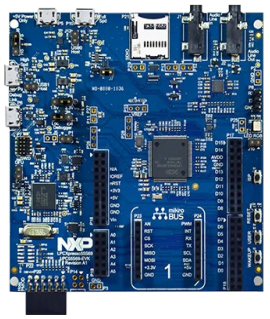
- JBL Flip 6:
  TI TAS5825P (digital input)





Zephyr board inventory







Over the 1180+ boards:

- 21 with external codec implemented and ready
- 359 with DAC peripheral presence
- 457 with I2S-capable peripheral presence

➔ DAC audio lets you have more options



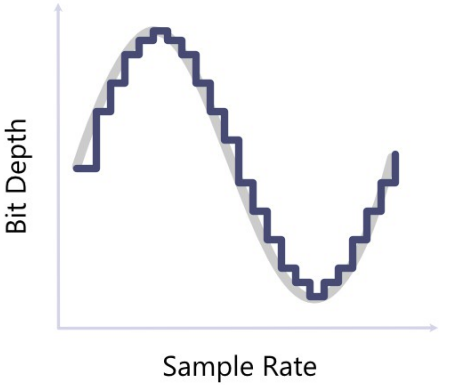
Demo

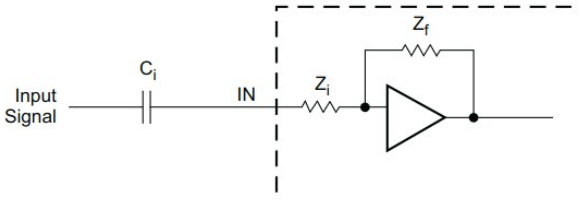


Minimum electronic requirements

- In order to drive a speaker we need:
  - Some power (small impedance)
  - Signal waving around zero
- DAC output has:
  - Limited output power
  - 0 to 3.3 V wave = 1.65V DC

  → We need some analog electronics

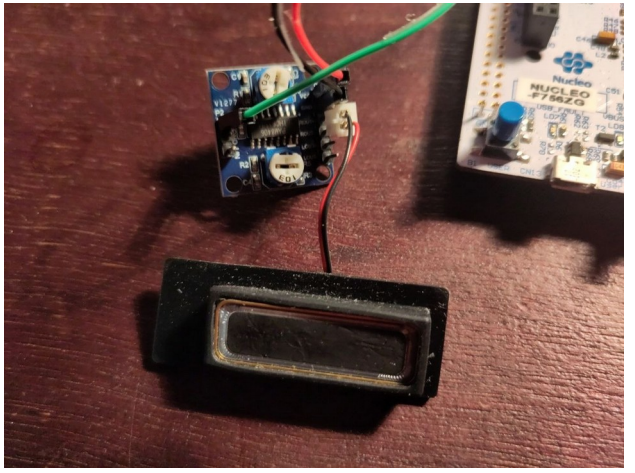






Demo: USB speaker

- Cheap analog amp eval kit
  - Battery-powered (for more watts!)
- Speaker from teardown
- Nucleo H7
  - USB UAC2 input
  - DAC output
- PoC code with MPipe





Conclusion

Music applications often require the use of expensive, specific audio gear, intricate codec ICs, specific boards...

Direct DAC audio can be used for:

- Learning!
- Non-HiFi audio products, voice
- Debugging with your ear
- Feedback loop, measurement
- Having fun



Thank you!



References

- https://en.wikipedia.org/wiki/Voice_frequency
- https://en.wikipedia.org/wiki/Hearing_range
- https://en.wikipedia.org/wiki/Delta_modulation
- https://audiojudgement.com/thiele-small-parameters-explained/
- https://www.geeksforgeeks.org/nlp/understand-audio-data/
- https://hackaday.io/project/162337-reverse-engineering-jbl-flip-4
- https://harman-cdn.servify.tech/vendor-partner-docs/Partner-Brand_Collateral_8_2024-05-08.JBLFlip6TL-ServiceManualV1.1.pdf
- https://www.ti.com/lit/ds/symlink/tas5825m.pdf
- Tracing to RAM https://redero.fr/?p=649&lang=en



Relevant PR & features

- drivers: stm32: dac: audio: add STM32 DAC audio playback driver - #115001
- Add samv71 dac codec driver - #118304
- Current discussions about audio dac https://github.com/zephyrproject-rtos/zephyr/issues/120240
- Fast GPIOs RFC https://github.com/zephyrproject-rtos/zephyr/issues/106831
- The new Multimedia Pipeline (mpipe) https://docs.zephyrproject.org/latest/services/mpipe/index.html

