Diploma thesis

Czech technical university in Prague
Faculty of Electrical engineering
Department of Measurement

Raspberry Pi Pico oscilloscope with a web-based user interface

Jan Fiala

Supervisor: doc. Ing. Jan Fischer, CSc.
May 2023

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Fiala Jan**	Personal ID number:	**475379**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Measurement**		
Study program:	**Open Informatics**		
Specialisation:	**Computer Engineering**		

II. Master's thesis details

Master's thesis title in English:

Raspberry Pi Pico oscilloscope with a web-based user interface

Master's thesis title in Czech:

Raspberry Pi Pico osciloskop s webovým uživatelským rozhraním

Guidelines:

Design and implement a software-defined, three-channel oscilloscope based on the Raspberry Pi Pico microcontroller. The oscilloscope should include a PWM generator with finely adjustable frequency to enable equivalent time sampling. Design an intuitive user interface according to the intended use in laboratory teaching at CTU FEE and in high schools. Implement the user interface as a web application, using the WebUSB API for communication with the microcontroller. Implement the required firmware. Test all functions of the oscilloscope and create a user manual in English and Czech.

Bibliography / sources:

[1] Vaněček, V.: Logic analyser based on Raspberry Pi Pico; ČVUT- FEL, 2022
[2] Raspberry Pi Ltd.: RP2040 Datasheet; 2022
[3] W3C: WebUSP API (Draft Community Group Report); 2022

Name and workplace of master's thesis supervisor:

doc. Ing. Jan Fischer, CSc. Department of Measurement FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **16.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until:
by the end of summer semester 2023/2024

doc. Ing. Jan Fischer, CSc.	Head of department's signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature		Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student's signature

Abstract

The goal of this thesis is to design and implement a software-defined oscilloscope using the Raspberry Pi Pico and a web-based user interface. Both parts of the system - the firmware for the Pico and the web-based user interface were implemented as part of this thesis. The oscilloscope includes three analog channels, a PWM generator and was designed to enable equivalent time sampling. The WebUSB API is used for communication between the microcontroller and the user interface.

Abstrakt

Cílem této práce je návrh a implementace softwarově definovaného osciloskopu založeného na mikrokontrolu Raspberry Pi Pico ve spojením s webovým uživatelským rozhraní. Součástí této práce je implementace obou částí osciloskopu – firmwaru pro Raspberry Pi Pico i webového rozhraní. Navržený osciloskop má tři kanály, PWM generátor a podporuje záznam signálu metodou equivalent time sampling. Pro komunikaci mezi mikrokontrolerem a uživatelským rozhraním je použito rozhraní WebUSB API.

Keywords

Oscilloscope, software-defined instrument, microcontroller, Raspberry Pi Pico, RP2040, WebUSB

Acknowledgement

I would like to thank my supervisor doc. Ing. Jan Fischer, CSc. for his help and patience.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Signature:
In Prague, May 26, 2023

Table of Contents

1 Introduction

Modern microcontrollers are powerful enough to be used for implementing instruments such as oscilloscopes or logic analysers on the low end of the performance spectrum. When the devices are implemented using readily available and cost-effective development boards such as the STM32 Nucleo or the Raspberry Pi Pico, they are particularly suited for use in classrooms - both in university courses or at high schools.

Multiple instruments based on microcontroller evaluation boards have been already created at the Department of measurement at CTU in Prague. For example, the LEO (Little Embedded Oscilloscope) [1] is an oscilloscope and a signal generator based on the STM32F303 Nucleo board which is widely used in laboratory lessons at CTU. The user interface for the LEO is a native Windows program written in C++. While native Windows applications are very efficient, they have an obvious drawback - they cannot run on other operating systems.

For this reason, I decided to create a new multiplatform (or platform independent) software-defined oscilloscope capable of running on all major operating systems – Windows, Linux and Mac OS X. There are multiple approaches to creating multiplatform applications, one of them being the currently popular Electron framework. Applications using the Electron framework are developed using standard web technologies – HTML, JavaScript and CSS and shipped with a modified version of the Google Chrome kernel, on which they are run. The modified browser kernel enables applications access to the file system, USB devices and other system capabilities. One obvious downside of this approach is the need to bundle a 120MB web browser kernel with every application. As an alternative, new web APIs have been created (in an effort led mainly by Google) to bring capabilities of standard web pages closer to native applications, potentially removing the need to install programs at all.

One of these APIs is the WebUSB API [2], which enables web pages client-side access to USB devices of the user. The WebUSB API is used for the oscilloscope developed in this thesis. With the use of this API, the user interface can be implemented as a standard web page, capable of running on Windows, Linux and Mac OS X without a need to be installed, which is a major advantage for use both in classrooms and at home. In case internet connection is an issue, the user interface can also be downloaded and run locally.

The Raspberry Pi Pico microcontroller board was selected as the hardware portion of the oscilloscope. The reasons for choosing the Pico are mainly its availability, low price and the presence of an USB peripheral (not just an USB-UART converter chip) on the board. The Pico has a relatively slow analog-digital converter, so the oscilloscope is designed to enable the equivalent time sampling technique. Using equivalent time sampling, sample rates up to 48MHz can be reached for certain types of captured signals, despite the modest speed of the ADC.

2 Overview of the oscilloscope design

An oscilloscope is a measurement instrument used to capture an analog voltage signal and display it as a two-dimensional time-voltage plot. While it is mainly used to inspect and verify the operation of electronic circuits, it is also a valuable tool for learning about electronics in a hands-on, visual way. Digital oscilloscopes, which are prevalent today, are based on analog-digital converters (ADCs). An ADC is an electronic circuit that converts analog voltage into a discrete digital representation. The ADC samples the voltage signal at a specific frequency (called the sample rate), as is shown in figure 1. The captured samples are saved into memory and when a specified number of samples is captured, they are displayed on the screen of the oscilloscope.

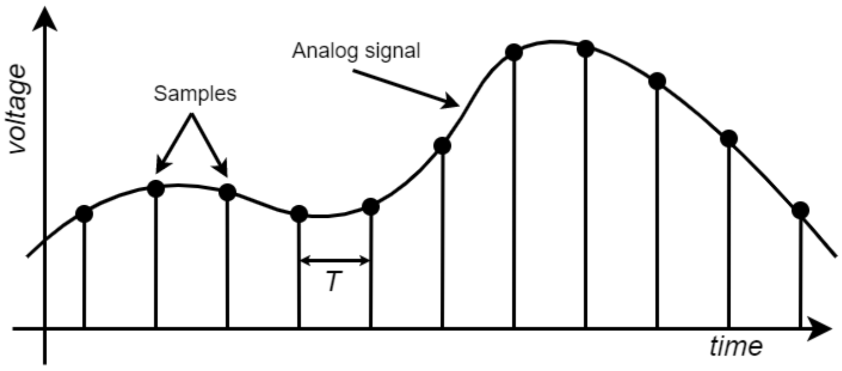


Figure 1: Analog voltage sampled at a frequency f = 1/T.

Practically, the oscilloscope developed in this thesis is divided into two parts: the hardware and the web-based user interface, as is shown in Figure 2. A crucial, and arguably the most novel part of the design is the communication interface between these two parts - the USB interface and the WebUSB API.

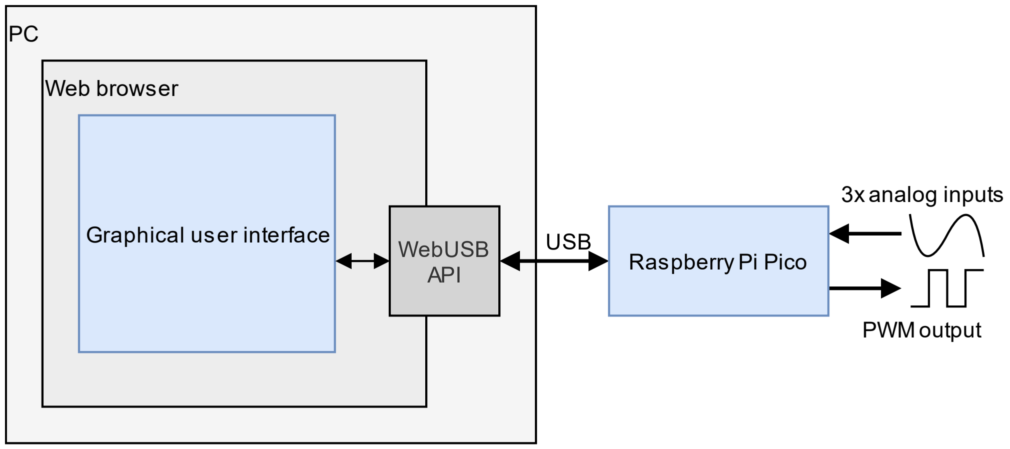


Figure 2: Overview of the oscilloscope design.

The Raspberry Pi Pico was chosen for the hardware part of the Oscilloscope. The main reasons for this decision are its availability, affordability and the presence of an USB peripheral on the board. A major downside of the Pico, or more precisely the RP2040 MCU present on the Pico board, is the low speed of the ADC peripheral, which cannot match the speed of instruments such as the LEO, which uses the STM32F303 microcontroller. However, the goal behind developing the oscilloscope is to make basic electrical measurements as accessible as possible, so a tradeoff in favor of the Raspberry Pi Pico was taken when choosing the hardware for the oscilloscope. The hardware features of the Pico are described in detail in the chapter 4.

In very simple terms, the Raspberry Pi Pico will have the following tasks:

1. Receive capture configuration from the user interface over USB.
2. Monitor the ADC for trigger condition.
3. Capture signal samples into a buffer when a trigger condition occurs.
4. Send the captured samples to the user interface over USB.
5. Repeat

Additionally, the PWM generator of the Pico will be used as a configurable PWM generator, so it will receive PWM configuration messages from the user interface and configure the PWM peripheral accordingly.
The user interface of the oscilloscope will be developed as a standard web page using HTML, CSS and JavaScript. The user interface will feature control panels for configuring the signal capture, such as capture depth, sample rate, number of channels, and trigger conditions. The GUI will plot the captured signals and allow the user to scale, zoom and pan the plotted signals.

To enable the use of the equivalent time sampling method (this method is described in detail in chapter X), the GUI must allow the user to set the oscilloscope sample rate and PWM generator frequency precisely. Because it is not always possible to reach the desired frequencies with the PWM and ADC peripherals, the actual frequencies of the PWM and ADC peripherals will be shown in the GUI too.

The user interface will also feature vertical and horizontal cursors, to enable precise measurement of time or voltage difference in the captured signals.

3 WebUSB API

3.1 Introduction

A crucial (and arguably the most novel) part of the proposed oscilloscope design is the communication between the hardware and the GUI. The WebUSB API serves exactly this purpose - communication between an USB device and software running in the browser.

Before describing the WebUSB API in more detail, I will cover a few terms and concepts relating to the USB interface itself. To accommodate devices with diverse communication requirements, USB is highly configurable. An USB device can use multiple (up to 32, although it is rare to use so many) virtual communication pipes, also called endpoints. There are three different types of endpoints: bulk, suitable for transferring large amounts of data, isochronous, which guarantee low latency, and control, mainly used for device configuration. When an USB device is connected to a host system, it sends the host a description of the required endpoints (this process is called enumeration), which the host system will then provide to the device.

On the host side of an USB connection is a device driver, which communicates with the device through endpoints exposed by the USB stack. Usually, an USB driver is written in C or C++ and runs inside the host operating system kernel. The WebUSB API exposes the USB device endpoints directly to the JavaScript runtime inside a web browser, so the device driver can be implemented directly using JavaScript.

Currently, WebUSB is in experimental phase and it is not a standard web API. Only web browsers based on the Chrome kernel (Chrome, Chromium and Microsoft Edge) support it at this time. There are officially no plans in the other two major browsers (Firefox and Safari) to implement this API, citing security concerns as the reason.
While these concerns are certainly valid, the way in which WebUSB exposes USB devices to the browser is more secure than it might sound. First, the user must explicitly select an USB device in a pop-up window for the web browser to access it. Second, it is not possible to access devices belonging to common USB classes such as HID (keyboards, mouses), Mass storage, Network interfaces and others. WebUSB can only access devices outside these classes, which are usually only very specialized hardware devices.

3.2 An example

Code snippet 1 demonstrates connecting to an USB device with the *vendor ID* equal to 0xCAFE. The requestDevice function does not select an USB device by itself: when it is called, a pop-up window appears in the browser, prompting the user themselves to select a particular USB device. The filters object passed to the function can be used to filter the USB devices listed to the user. The device *vendor ID*, *product ID* or serial number can be set to only allow a particular device(s) to be listed. After a device is selected, a device *configuration* and *interface* are chosen. It is rare for an USB device to have multiple interfaces and even rarer to have multiple configurations. In this example, the first configuration and interface are selected.

```
let device = await navigator.usb.requestDevice({
   filters: [{ vendorId: 0xcafe }]
});
await device.open();
await device.selectConfiguration(1);
await device.claimInterface(1);
```

Code snippet 1: Connecting to an USB device using the WebUSB API.

After connecting, we can start communicating with the device. An example of sending data to the device is shown in code snippet 2. The transferOut function parameters are the number of the endpoint (3 in this case) to which a message will be sent to, and the message itself. The message must be an ArrayBuffer object, such as an Uint8Array, which is basically an array of bytes.

```
let message = new Uint8Array([1, 2, 3, 4]);
device.transferOut(3, message);
```

Code snippet 2: Sending an array of bytes to an endpoind of an USB device.

Similarly, the transferIn function can be used to receive data from a particular endpoint. The second parameter of the function sets the maximum number of bytes to be received. The function returns a DataView object, which can be parsed to an Uint8Array or other array types.

```
let result = await device.transferIn(3, 4);
let bytes = result.data.getUint8();
```

Code snippet 3:Requesting a transfer with a size of 4 bytes from an USB device.

In code snippet 4, a more complete example of a web page using the WebUSB API is shown. The web page contains a button for connecting to an USB device and a button for reading data. Received data are shown inside a HTML div element on the page.

```
// HTML content of the page:
<button id="connect">Connect device</button>
<button id="readData">Read data</button>
<div id="display"></div>

// JavaScript code:
let device;

document.addEventListener('DOMContentLoaded', () => {
   document.getElementById('connect').addEventListener('click',
requestUSBDevice);
   document.getElementById('readData').addEventListener('click',
readData);
});

async function requestUSBDevice() {
   device = await navigator.usb.requestDevice({ filters: [] });
   console.log('USB device selected:', device);
   await device.open();
   await device.selectConfiguration(1);
   await device.claimInterface(0);
}

async function readData() {
  const endpointIn = device.configuration.interfaces[0].endpoints[0];

  const result = await device.transferIn(1, 1);
  let parsedData = result.data.getUint8();

  document.getElementById('display').textContent = 'Received: ' +
parsedData[0];
}
```

Code snippet 4: Example of a simple web page capable of connecting to an USB device, reading and displaying received data.

While it is possible to read data from an USB device as shown in code snippet 3, there is one issue – the `transferIn` can return less data than was requested. An obvious solution would be to simply request additional data until all the requested data were received, as shown in code snippet 5.

```
Let readSize = 128;
let result = [];
while (result.length < readSize) {
   let rec = await device.transferIn(3, result.length - readSize);
   result.append(rec.data.getUint8());
}
```

Code snippet 5: Reading data of size readSize inside a loop to deal with data being fragmented.

3.3 Practical insights of using the WebUSB API

Because WebUSB is not widely adopted, there are only a few resources available to help with development. The first is an article called *Access USB Devices on the Web* [3], which contains a simple example of using WebUSB and also considers platform-specific issues with using the API, which will be described later. The second resource is the official draft [2] of the WebUSB API, which describes components of the API in more detail.

These resources only include very basic examples, so a large amount of experimentation was required to use the API in practice. These struggles were heightened by my limited knowledge of JavaScript and the browser runtime. JavaScript can only run in a single thread inside a browser window tab, and this thread also handles re-rendering of the web page. This means that any blocking JavaScript code would cause the whole web page to stop responding.

3.4 Special USB descriptor on Windows

For the Windows operating system to allow the WebUSP API to communicate with an USB device, the device needs to include a special USB descriptor during its enumeration. This descriptor, called the *Microsoft OS 2.0 platform capability descriptor*, is fortunately included in the TinyUSB [4] open source USB stack, which was used for the oscilloscope firmware development.

3.5 Udev rules on Linux

Linux does not require a special USB descriptor, but it disables user space applications from accessing USB devices as a default behavior. To enable access to an USB device, an *udev rule* file needs to be added to the /etc/udev/rules.d/ folder. This file needs to have the .rules extension and content shown in code snippet 4:

```
SUBSYSTEM=="usb",ATTR{idVendor}=="cafe",MODE="0664",GROUP="plugdev"
```

Code snippet 6: Content of an udev rule file enabling WebUSB access to an USB device.

3.6 Debugging tools

Few tools proved to be useful for troubleshooting during the initial phase of development. The first is a utility inside the Chrome web browser, called *USB Internals*. It can be accessed by typing chrome://usb-internals/ into the search bar and lists all USB devices connected to the system. It also allows creating a virtual USB device (this device is only visible from inside the web browser, so it only has a limited utility, such as troubleshooting a WebUSB connection). Another utility, called Web USB tester [5], can be found on GitHub. It is a simple web page which can be used to test connection to an USB device with a specific USB *vendor ID* and *product ID*. Finally, the well-known WireShark packet analyzer can be used to capture and inspect “raw” USB communication on Linux, however this requires changing a few settings of the system. The required steps are described in a StackOverflow discussion [6].

4 Hardware platform – the Raspberry Pi Pico

4.1 Overview

There are several features of a microcontroller that are crucial for using it as an oscilloscope – an analog-digital converter to convert a signal into digital samples, a large enough RAM to store the captured samples, and a DMA peripheral to move the samples from the ADC into memory without overloading the microcontroller processing core. Additionally, a PWM generator is needed to implement a signal generator. A general overview of a microcontroller-based oscilloscope is shown in Figure 3 below, indicating communication between different parts of the microcontroller, with data transfer from the ADC peripheral to RAM being handled by the DMA peripheral without involvement of the CPU.

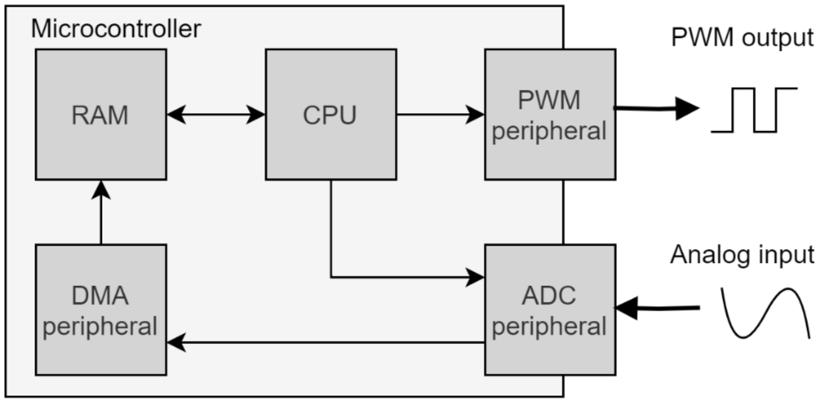


Figure 3: Overview of a microcontroller-based oscilloscope.

4.2 The Raspberry Pi Pico

The Raspberry Pi Pico, which was chosen as the hardware part of the oscilloscope developed in this thesis, contains the RP2040 microcontroller. The RP2040 features two ARM Cortex-M0 cores with clock speed up to 133MHz, 264kB of SRAM and peripherals such as the ADC, USB (including a PHY), DMA, PWM generator and others. The peripherals relevant to the oscilloscope are discussed in detail in this chapter. Besides the microcontroller, the Pico also contains a switching-mode DC-DC regulator for generating 3.3V from the USB power supply and a flash memory chip (the RP2040 does not have internal flash memory).

4.3 DMA peripheral

The DMA (direct memory access) peripheral of the RP2040 can be used to transfer data between a peripheral and the RAM independently of the microcontroller cores. In the oscilloscope, this is a crucial feature, because samples from the ADC peripheral need to be moved at regular intervals to a buffer in the memory, while the processor cores are doing other tasks such as trigger detection or handling USB communication.

Because ADC samples are saved to a circular buffer in the RAM, a “circular” DMA mode, as it is called in datasheets of certain STM32 microcontrollers, is required. While this mode is not available on the RP2040, it is possible to chain two DMA channels together to achieve the effect of a circular DMA transfer. This solution was inspired by a logic analyzer based on the Raspberry Pi Pico created as the diploma thesis of Ing. Vít Vaněček [8].

4.4 USB peripheral

The RP2040 features an USB peripheral including the PHY, which handles the physical layer of the USB protocol and converts the chip-level electrical signals to and from the 5V signals on the USB bus. The peripheral supports the USB 2.0 Full speed device mode, with communication speed of 12Mb/s. Both the USB peripheral and the USB protocol itself are relatively complex, so the configuration of the peripheral as well as tasks such as the enumeration of the device are handled by a port of the TinyUSB stack included in the Pico software development kit.

4.5 Analog-digital converter

The analog-digital converter of the RP2040 has five channels, with one channel connected to the internal temperature sensor. The structure of the ADC is shown in figure 4. On the Raspberry Pi Pico, one of the four remaining channels is used to measure the power supply voltage (V_{SYS}), leaving three usable ADC channels for the oscilloscope. The maximum sample rate of the ADC is 500kHz. All channels share the single analog-digital converter over an analog multiplexer, so the per-channel maximum sample rate is one half or one third of the maximum ADC sample rate when two or three channels are active, respectively.

When the ADC is in free-running mode, as is the case with the oscilloscope, conversions are started at regular intervals, triggered by the pacing timer inside the ADC peripheral. The pacing timer is fed with a 48MHz clock and features a configurable fractional divider with 16 integer and eight fractional bits to slow down the input clock. The fractional part of the divider causes jitter in the timing of the ADC sampling, which is undesirable (particularly when using the oscilloscope in equivalent time sampling mode), so only the integer part of the divider is used. The sample rate of the ADC can be calculated using equation 1.

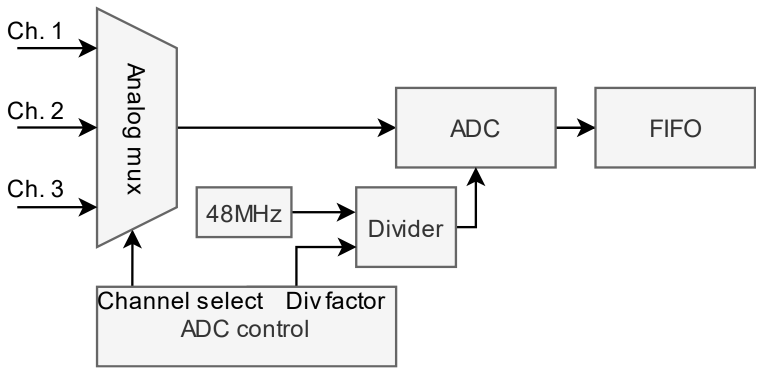


Figure 4:Analog-digital converter of the RP2040.

$$f_{ADC} = {}^{48MHz}/_{div}, div \geq 96 \tag{1}$$

The resolution of the ADC can be set to 8 or 12 bits. When it is set to 8 bits, each sample fits into one byte of memory. In 12 bit mode, two bytes are needed to store each sample, leading to wasted space in the memory. However, as the SRAM of the RP2040 is large enough, it was decided to use the 12-bit mode for the oscilloscope for greater resolution of the signal capture. Out of the 264kB of SRAM on the chip, 200kB could be used as the capture buffer, meaning up to one hundred thousand 12-bit samples can be saved at a time.

The RP2040 does not contain a reference voltage for the ADC - an external reference voltage must be supplied to a specific microcontroller pin. On the Raspberry Pi Pico board, the 3.3V output from the onboard SMPS (switching-mode power supply) is used as the reference voltage for the ADC. The SPMS chip present on the Pico features two regulation modes - pulse frequency modulation and pulse width modulation. One of the GPIO pins of the RP2040 is connected to a pin of the SMPS which selects the regulator mode, so it can be selected by the firmware. Because the switching mode of the SMPS can influence the stability of the 3.3V ADC reference during certain conditions, the switching mode can be configured in the oscilloscope GUI.

4.6 PWM generator

The PWM peripheral of the RP2040 will be used as a configurable PWM signal generator of the oscilloscope. In addition to being used as a configurable clock source, the generator is crucial in signal capturing using the equivalent time sampling method. The structure of the PWM peripheral is shown in figure 5.

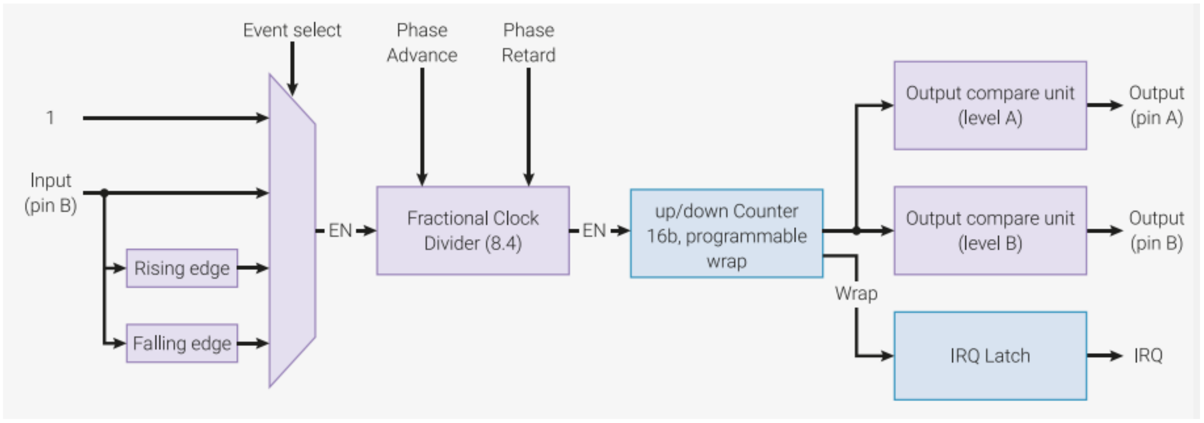

Event select
Phase Advance
Phase Retard
1
Input (pin B)
Rising edge
Falling edge
EN
Fractional Clock Divider (8.4)
EN
up/down Counter 16b, programmable wrap
Wrap
Output compare unit (level A)
Output compare unit (level B)
IRQ Latch
Output (pin A)
Output (pin B)
IRQ

Figure 5: PWM peripheral of the RP2040. Taken from [7].

The generator consists of a clock divider, an up/down counter and an output compare unit.
The clock divider, which is fed with the 125MHz system clock, allows division of the system clock by a fractional number *div* with eight integer and four fractional bits. However, the fractional part of the generator causes jitter in the output signal, which would make it unsuitable when using the oscilloscope in equivalent time sampling mode. With eight integer bits of the division factor, the divider can be configured to slow the output down by a factor from one to 255.

The output of the divider is fed into a 16 bit up/down counter. The counter has a configurable *wrap* value, which causes the counter to reset when the value is reached. The wrap value determines the period of the PWM signal relative to the divider output, so it can be configured along with the division factor of the divider to achieve the highest possible frequency range of the PWM generator. The frequency of the PWM output for given *wrap* and *div* values can be calculated using equation 2.

$$f_{PWM} = {}^{f_{SYS}}\!/_{div \cdot wrap} \tag{2}$$

The output of the counter is compared to a threshold value inside the output compare unit, which then drives an output pin of the microcontroller. When the counter output is lower than the threshold value, the output pin is set high. After the counter value reaches the threshold, the pin is kept low until the counter reaches the wrap value and is reset. The threshold value determines the duty cycle of the PWM duty signal, as is shown in equation 3.

$$duty = {}^{threshold}\!/_{wrap} \tag{3}$$

Obviously, the user of the oscilloscope will not be asked to set the div and wrap values of the PWM peripheral by themselves. It is possible to calculate the required div and wrap values for a desired frequency f_{SET} using equations 4 and 5:

$$div = \left\lceil \frac{f_{SYS}}{f_{SET} \cdot 2^{16}} \right\rceil \tag{4}$$

$$wrap = \left\lceil \frac{f_{SYS}}{f_{SET} \cdot div} \right\rceil \tag{5}$$

It is not always possible to generate the exact desired frequency. In that case, the actual frequency can be calculated using equation 6 with the *div* and *wrap* values obtained using equations 4 and 5.

$$f_{REAL} = \frac{f_{SYS}}{div \cdot wrap} \tag{6}$$

5 Oscilloscope firmware

5.1 Pico SDK and development setup

For development of the oscilloscope firmware, the *Raspberry Pi Pico SDK* (software development kit*)* [9] was used. The SDK contains drivers for the peripherals of the RP2040 and simple examples of the of using the peripherals of the Pico. One part of the SDK I found to be lacking is the USB stack. The Pico SDK uses a port of the TinyUSB USB stack, but it only implements a few basic USB classes. The WebUSB device class, which is present in TinyUSB, is missing in the Pico SDK port. However, the only special requirement of a WebUSB device is an additional *Platform capability descriptor*, which I was able to add from the main TinyUSB branch.

To make development easier, a second Raspberry Pi Pico flashed with the Picoprobe firmware was used. The Picoprobe firmware enables the Pico to function as a debugger and an USB-UART converter. The Pico with the oscilloscope firmware was connected to the Picoprobe with its SWD and UART ports, so it could be flashed without having to press the BOOT button on the flashed board each time. The UART port could be used for debugging too, when debugging using logging was preferred. The oscilloscope development setup is shown in figure 6.

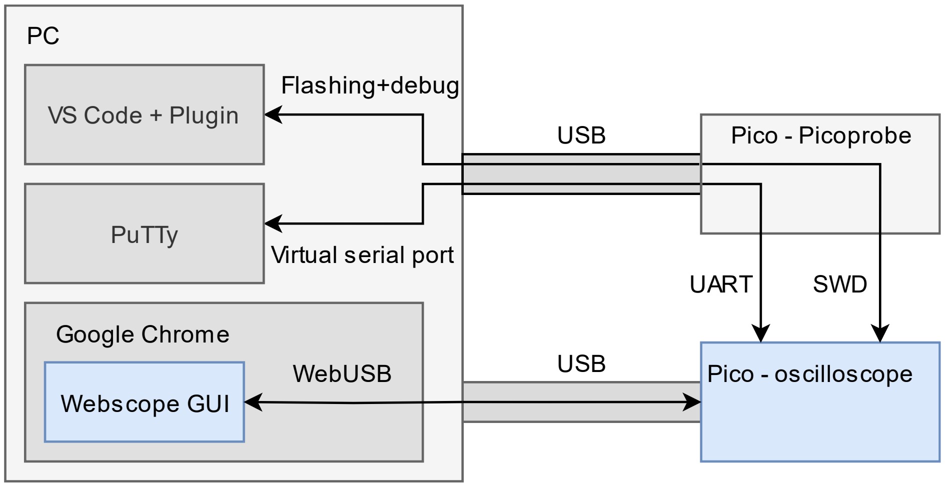
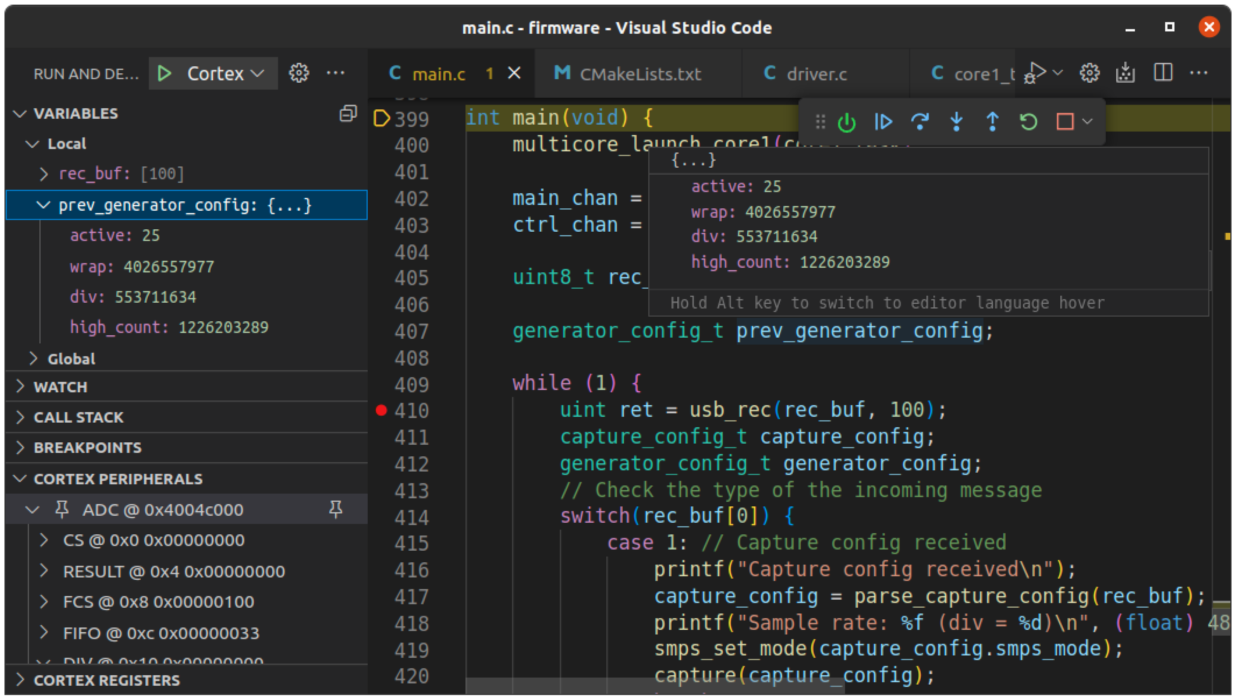


Figure 6: Overview of the development setup using Picobrobe firmware.

The Visual Studio Code IDE with the Cortex-Debug plugin was used for writing the firmware. As is shown in figure 7, the Cortex-Debug plugin features breakpoints and code stepping, as well as variable and microcontroller peripheral register inspection. The Picoprobe firmware, as well as configuration of the Visual Studio Code IDE for use with the Picoprobe are described in the *Getting started with Raspberry Pi Pico* document [10]. The wiring between a Pico flashed with the Picoprobe firmware and another Pico is shown in figure 8.



```
rec_buf: [100]
prev_generator_config: {...}
    active: 25
    wrap: 4026557977
    div: 553711634
    high_count: 1226203289
ADC @ 0x4004c000
    CS @ 0x0 0x00000000
    RESULT @ 0x4 0x00000000
    FCS @ 0x8 0x00000100
    FIFO @ 0xc 0x00000033
```

```
int main(void) {
    multicore_launch_core1(c

    main_chan =
    ctrl_chan =

    uint8_t rec_

    generator_config_t prev_generator_config;

    while (1) {
        uint ret = usb_rec(rec_buf, 100);
        capture_config_t capture_config;
        generator_config_t generator_config;
        // Check the type of the incoming message
        switch(rec_buf[0]) {
            case 1: // Capture config received
                printf("Capture config received\n");
                capture_config = parse_capture_config(rec_buf);
                printf("Sample rate: %f (div = %d)\n", (float) 48
                smps_set_mode(capture_config.smps_mode);
                capture(capture_config);
```

```
{...}
    active: 25
    wrap: 4026557977
    div: 553711634
    high_count: 1226203289
Hold Alt key to switch to editor language hover
```

Figure 7: Visual Studio Code configured for oscilloscope firmware development.

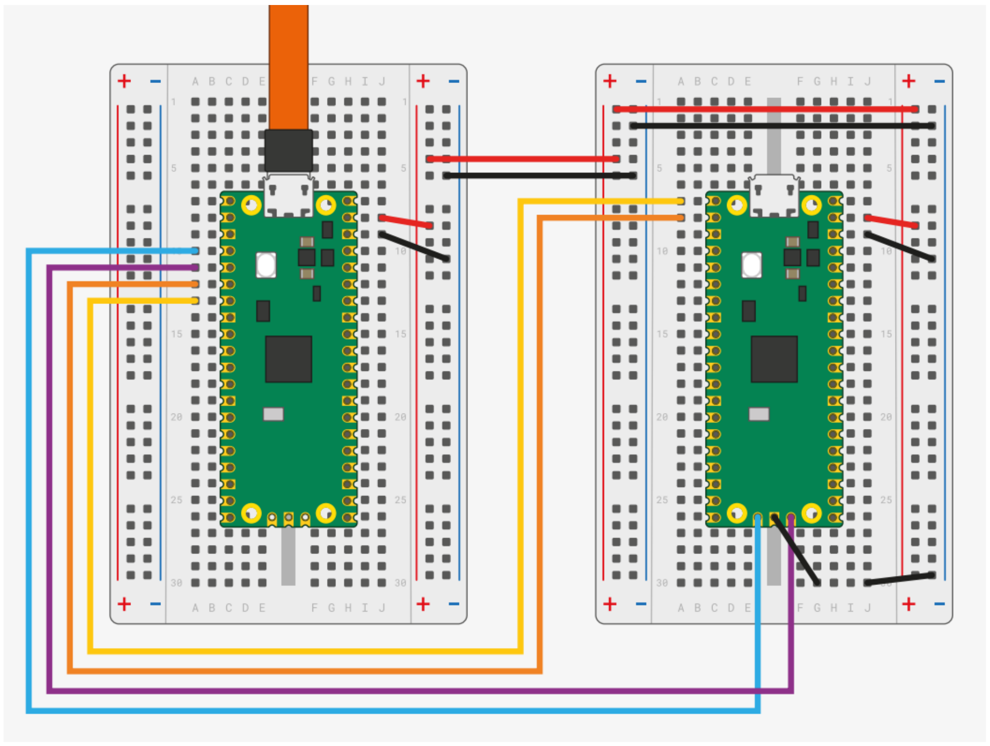


Figure 8: Connecting Picoprobe (left) to debug another Pico. Taken from [10].

5.2 USB stack and use of the second microcontroller core

While the first of the two cores of the RP2040 handles the “application” logic of the oscilloscope, the second core is dedicated to handling USB communication. The TinyUSB stack, which is used as the USB stack in the firmware, defers interrupts generated by the USB peripheral into a queue. A function that handles any interrupts in the queue must be called periodically by the application code, so by doing this on the second core, the first core can handle time-critical tasks such as trigger condition detection.

The RP2040 features a hardware queue to enable communication between the cores, however it was decided to use a simpler approach to share USB messages between the cores instead. The two firmware threads running on the two cores share two buffers: *usb_rec_buffer* and *usb_send_buffer* and two global variables: *usb_rec_bytes* and *usb_send_bytes*. When the second core receives data from the USB peripheral, it copies the data into *usb_rec_buffer* and sets *usb_rec_bytes* to the number of received bytes. When the first core wishes to read the data, it polls *usb_rec_bytes* until its value is greater than zero and then reads the received data from *usb_rec_buffer* and resets *usb_rec_bytes* to zero. Conversely, when the first processor core wishes to send data over the USB bus, it polls *usb_send_bytes* until it is set to zero and then copies the data into *usb_send_buffer* and sets *usb_send_bytes* to the number of bytes to be sent. The second core checks the value of *usb_send_bytes* periodically and when its value is greater than zero, it copies data from *usb_send_buffer* into the USB peripheral and resets *usb_send_bytes* to zero.

5.3 Firmware overview

The first core of the RP2040, which handles the signal capture, is executing a loop shown in figure 9. After the microcontroller is powered on and the USB peripheral is configured, the Pico waits for a message from the host PC. There are three types of messages: capture configuration messages, PWM configuration messages and capture abort messages. Capture configuration message contains capture parameters such as the sample rate, trigger conditions, number of active channels and others. PWM configuration message contains the PWM generator frequency and duty cycle, as well as an indication whether the PWM should be turned on or off. The capture abort message is used to abort an ongoing signal capture when the capture parameters are changed by the user, so a new capture with updated configuration could be started.

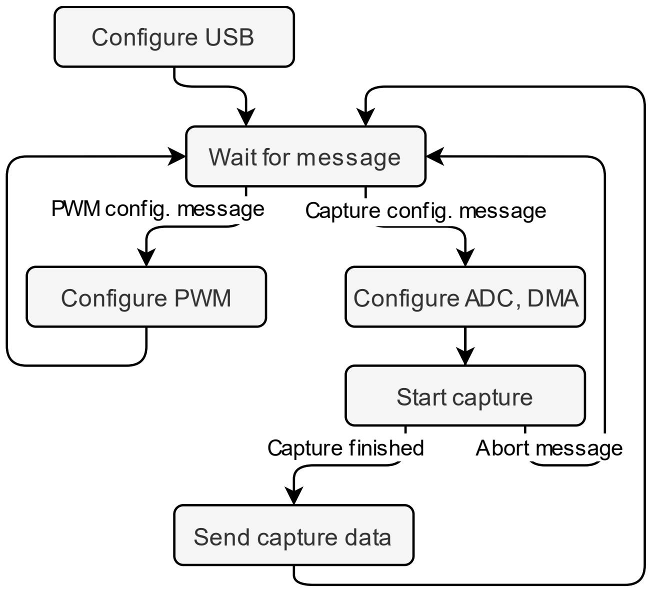


Figure 9: Simplified diagram of the oscilloscope firmware.

When a capture configuration is received, the ADC and DMA peripherals are configured accordingly, and the capture is started. When the capture is finished, the captured data is sent to the host PC and the Pico waits for another message. Sometimes, it is necessary to stop a capture before it is finished - most often when the user changes capture settings so the capture needs to be restarted with updated capture configuration. For this reason, there is another type of message called the abort message. When the Pico receives this message while a capture is running, it immediately aborts the capture and waits for another configuration message from the host PC.

When a PWM configuration message is received, it is compared to the current configuration of the PWM peripheral (this is done to avoid restarting the peripheral and causing a glitch in the PWM signal when it is not necessary). If the received configuration is different, the PWM peripheral is reconfigured accordingly.

5.4 GUI communication - message format

The first byte of each message indicates the message type and is followed by a fixed-length message payload. The content of the message does not follow any defined format such as JSON or Protobuf, messages are encoded and decoded with hand-written parsing methods in the GUI and in the firmware. As more features were added to the oscilloscope during development and additional fields were added to the configuration messages, this ad-hoc parsing of messages started to become time consuming and error-prone. Given this experience, I would use a message format such as JSON or Protobuf [11] if I were to implement the oscilloscope firmware again.

5.5 Signal capture

In a very simple digital oscilloscope, ADC samples would be monitored for a trigger condition and when a trigger condition would be detected, a defined number of samples would be captured and transferred to a buffer inside memory. However, one crucial feature of an oscilloscope would not be possible to implement with this approach: capturing a portion of the signal before the trigger appears. The length of this portion of the signal, or its ratio to the total capture length, is usually called the pretrigger setting. When the pretrigger setting is larger than zero, it is too late to start saving signal samples when a trigger condition appears. Instead, samples are transferred continually into a circular buffer from the moment the capture is started.

A circular buffer is a data structure implemented using a fixed-length continuous buffer inside memory. New data (samples) are inserted at the end of the buffer sequentially. When the buffer is filled with data, an additional sample added to the end of the buffer will overwrite the first element of the buffer - the least recent sample. This means that, once full, a buffer of size n will always contain the n most recent samples. The concept of the circular buffer is shown in figure 10 below.

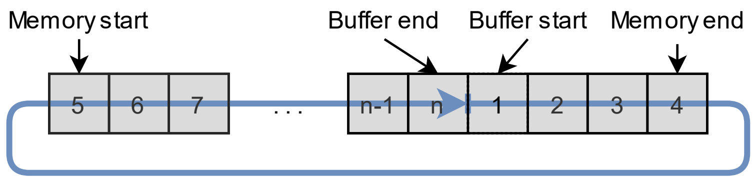


Figure 10: Conceptual diagram of a ring buffer.

As an example, let’s say the length of the capture buffer is 100 samples and the pretrigger ratio is set to 20%. After a trigger is detected, the capture is stopped after 80 more samples are captured. This leaves the buffer with 20 signal samples before the trigger and 80 after.

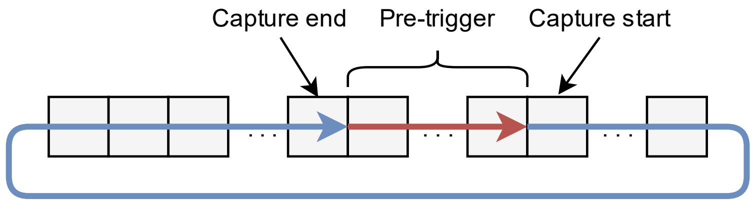


Figure 11: Storing a capture inside a ring buffer, including the pretrigger portion of the capture.

As was explained above, after a trigger condition is detected, it is necessary to stop the signal capture after a defined number of samples. While it is possible to configure the DMA peripheral to transfer a certain number of samples, it is not possible to change this number while the DMA is running on the RP2040. Instead, the first core of the microcontroller is continually checking the number of transferred samples in the register of the DMA peripheral. When the required number of samples is acquired, the ADC and the DMA are stopped. While this is not the most elegant solution, it works well given the slow speed of the ADC relative to the microcontroller cores.

6 The graphical user interface

6.1 Architecture overview

Being an interactive web page, the GUI of the oscilloscope was developed using standard web technologies: HTML for defining the web page elements, JavaScript for implementing the interactivity and CSS for styling. While it would be perfectly possible to use plain HTML, JavaScript and CSS, it was decided to use the React web development framework due to the complexity of GUI. React is a component-based framework, meaning the web page is built from components such as buttons or text fields, from which larger components such as entire control panels of the GUI are build. In React, each component is essentially a JavaScript function which, according to its input, generates and returns some HTML code. When the page is loaded, the React runtime calls each of these functions and builds the web page from the fragments generated by each function. When needed, components are generated again with updated parameters and portions of the GUI are redrawn.

The GUI is composed of control panels such as the PWM generator control panel, Trigger configuration panel or Channel configuration panels. These components are often similar, both style and functionality wise, so basic components such as buttons, numeric input fields or text fields were often reused with the help of the React framework. As an example, various control panels built using the same basic components are shown in figure 12 below.

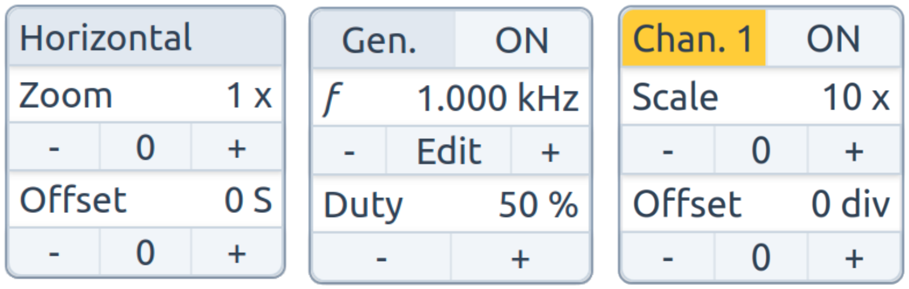
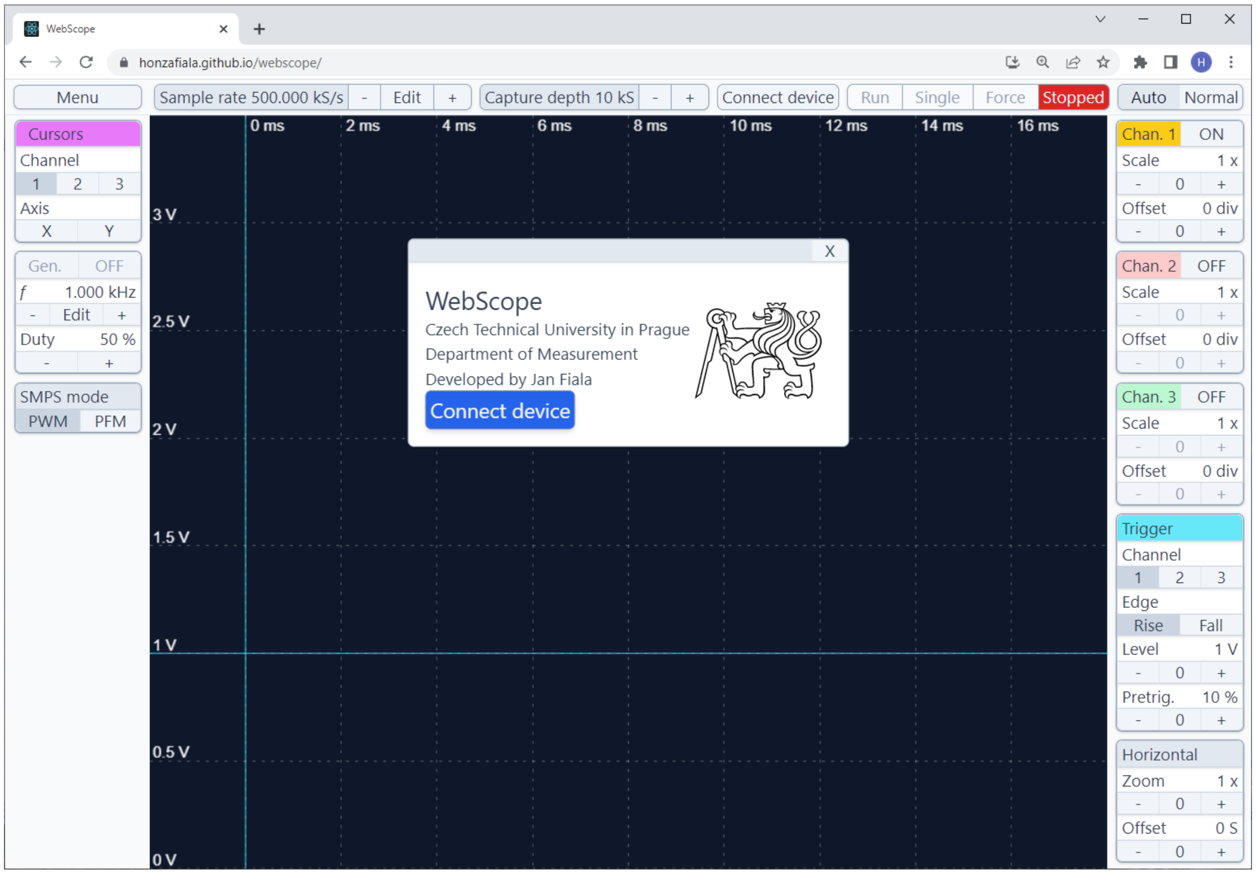


Figure 12: Different control panels of the user interface sharing the same basic components.

6.2 Interface design

The final design of the GUI is shown in figure 13. The interface is divided into four parts – the top bar, the left and right side panels and the signal plot in the center. Capture depth and sample rate settings, as well as buttons for starting the signal capture in various modes are located in the top bar. The right side bar contains settings for individual channels, trigger configuration and horizontal control panel used for zooming and panning the captured signal. Additional features of the oscilloscope – cursor settings, PWM generator settings and the SMPS power supply mode setting are located in the left side bar. In the future, any additional features of the oscilloscope would be probably added to the left side bar.



WebScope
Czech Technical University in Prague
Department of Measurement
Developed by Jan Fiala

Figure 13: Design of the oscilloscope GUI.

The GUI was designed to accommodate various browser window sizes. To achieve this, the main elements of the interface – the top bar, the side bars and the signal plot – are placed inside a CSS grid layout. The height of the top bar, as well as the widths of the side bar are constant, while the signal plot adjusts to the available remaining space on the screen. To respond to the browser window being resized during use, an *event listener* redraws the plot in the correct size when the web page *onresize* event is triggered.

6.3 Signal plotting

There are two general ways of drawing rendering graphics (such as signal plots) on a web page. One of them is the svg HTML element, in which basic vector graphics elements such as lines or bezier curves can be inserted. The other is the canvas HTML element, which allows pixel-wise manipulation of the element content, treating it as a bitmap. While a signal plot is essentially a complex curve or a series of lines, making the svg element seem more suitable, performance issues appeared with this approach. Thankfully, this was still in the early development phase, so it was possible to switch to using the canvas element.

6.4 Implementing cursors

In initial iterations of the user interface, the oscilloscope cursors were controlled using HTML slider elements located in the right side bar of the oscilloscope, as is shown in figure 14 - Dragging the slider elements with a mouse would move the cursors in the signal plot accordingly. However, because the sliders had to be only about 120 pixels wide to fit in the side bar, positioning the cursors precisely was difficult – a change of the cursor position in the plot view corresponded to a much smaller change of the position of the slider.

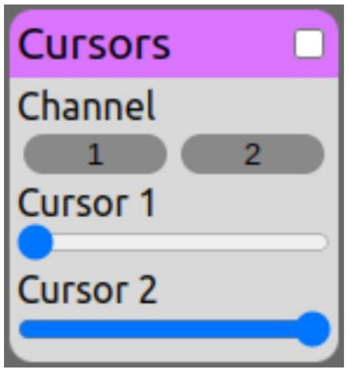


Figure 14: Cursor control panel in an early iteration of the GUI.

For this reason, a different, more precise way of positioning the oscilloscope cursors had to be devised. Arguably, the most intuitive way of controlling the cursors would be dragging the cursors with a mouse directly in the signal plot. There are no existing HTML control elements that would enable this behavior, so it had to be implemented in JavaScript. This was done by rendering the cursors on top of the signal plot as absolutely positioned HTML div elements as shown in figure 15. The cursor lines themselves are only a few pixels wide, which would make clicking on them with a mouse difficult – for this reason, the div elements are transparent and have a width of 20 pixels, with only their left or right borders visible. Clicking on and dragging the elements is detected using the *onclick* and *onmousemove* events, which trigger repositioning of the div elements to a new location.

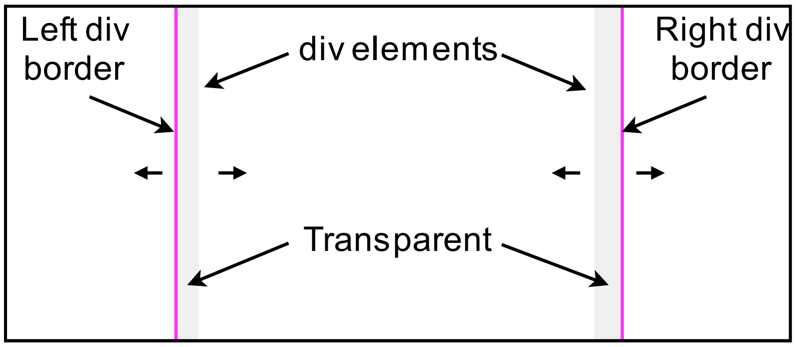
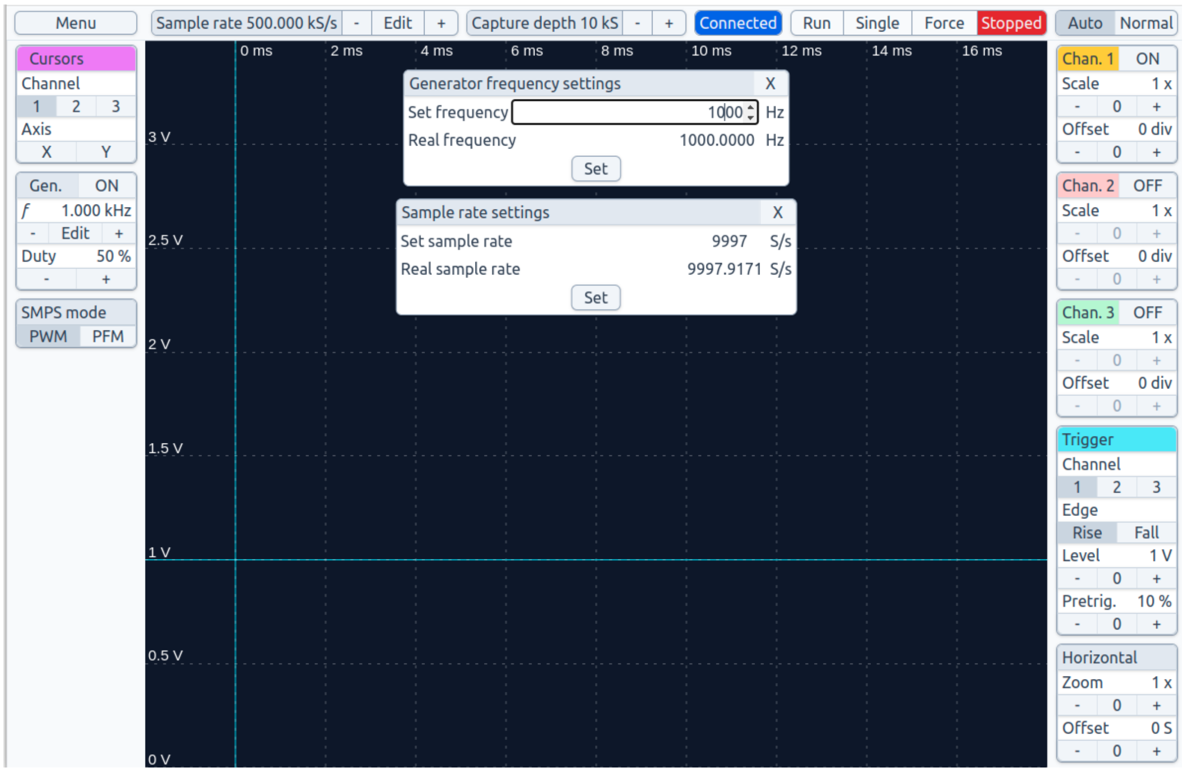


Figure 15: Horizontal cursors implemented as HTML div elements.

6.5 Configuration pop-up windows

To enable the use of the equivalent sampling method (which is described in detail in section 7.2), a precise way of configuring the oscilloscope sample rate and the PWM generator frequency was needed. For this reason, configuration pop-up windows, which can be seen in figure 16 below, were implemented. The windows contain an HTML text input form and allow the user to input the desired PWM frequency or sample rate using their keyboard. Because it is not always possible to reach the desired configuration with the PWM and ADC hardware, the actual configured PWM frequency or sample rate is shown below the value set by the user. The pop-up windows were implemented as absolutely positioned HTML div elements rendered on top of the rest of the user interface. To take advantage of the component-based architecture of the React framework, the pup-up windows (as well as the oscilloscope welcome screen, shown in figure 13) are all based on a single React component and only differ in the content.



Generator frequency settings X

Set frequency	1000	Hz
Real frequency	1000.0000	Hz

Set

Sample rate settings X

Set sample rate	9997	S/s
Real sample rate	9997.9171	S/s

Set

Figure 16: Oscilloscope GUI with sample rate and PWM generator pop-up windows opened.

6.6 Running the GUI locally

One clear disadvantage of web-based software is the dependency on internet connection. The oscilloscope GUI is, from the point of view of the server, a purely static website - the server only sends requested JavaScript and HTML files to the client and after that, all functionality of the GUI is handled by the client-side JavaScript code without further interaction with the server. It is therefore possible to run the GUI locally (by locating the index.html file of the application and opening it in a web browser). Only a small modification is needed in the NMP build configuration in this case - for this reason, two builds of the GUI exist - one to be hosted on a server and one to be downloaded and run locally.

7 Achieved results

7.1 Oscilloscope parameters and features

7.1.1 Sample rate

The oscilloscope has three analog channels. The maximum per-channel sample rate is 500kS/s with one active channel, 250kS/s with two active channels and 166kS/s with three active channels. To enable long signal captures, the minimum sample rate is 1kS/s. The sample rate can be selected in defined steps in the user interface and a pop-up window was implemented to allow the user to select the desired sample rate precisely.

7.1.2 Capture depth

The maximum total capture depth is 100kS. When more than one channels are active, the capture buffer is shared between all active channels, leading to a maximum per-channel capture depth of 50kS with two active channels and 33.3kS with three channels. The capture depth is configurable in the user interface in defined steps between 1kS and 100kS.

7.1.3 Capture modes

The oscilloscope has three capture modes called *Continuous*, *Single* and *Force.* In continuous mode, the oscilloscope performs new signal captures continuously until stopped. In single mode, only one signal capture is performed. Force mode is similar to single mode, with the difference that the trigger condition is forced immediately after the capture starts.

7.1.4 Trigger settings

Various parameters of the trigger condition can be configured: The trigger channel, rising or falling signal edge, trigger level and prettriger ratio.

7.1.5 PWM generator

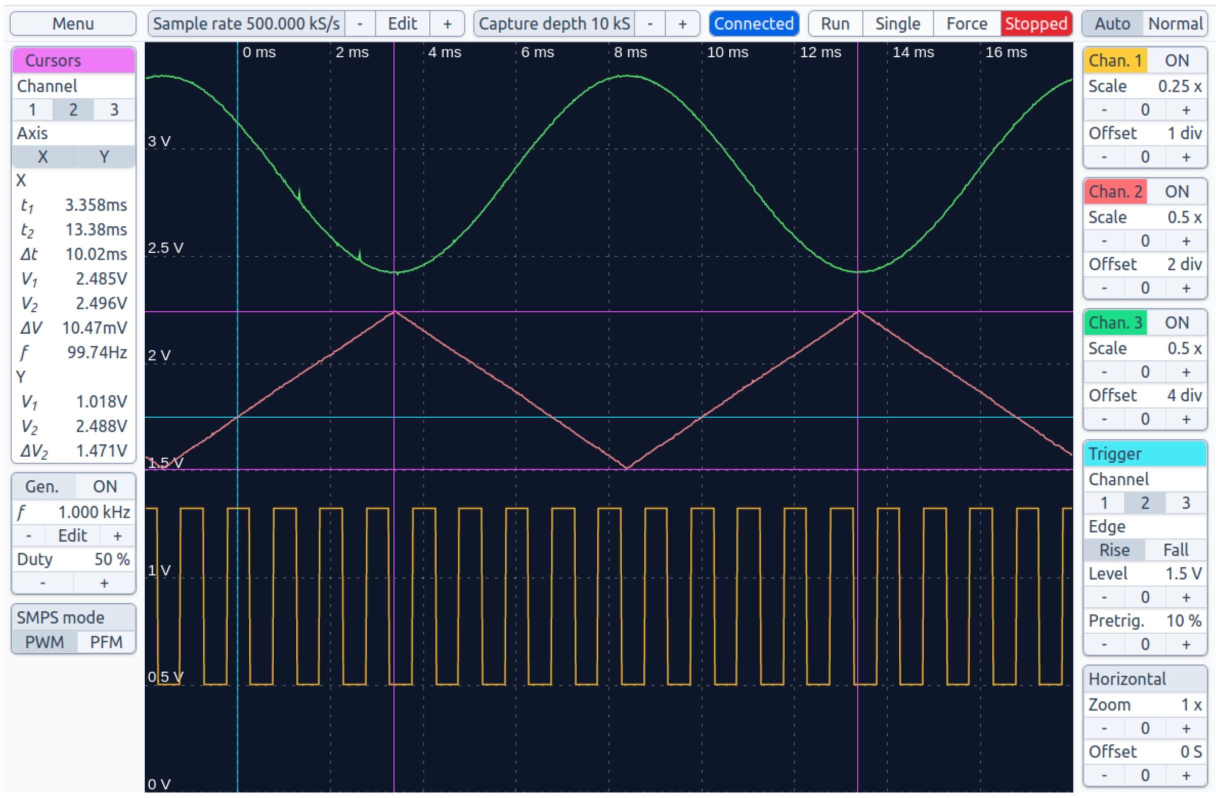
The oscilloscope includes a configurable PWM generator with configurable frequency between 7.5Hz and 50MHz and configurable duty cycle between 1% and 100%. A pop-up window was implemented to allow the user to select the desired generator frequency precisely.

7.1.6 Cursors

The oscilloscope features horizontal and vertical cursors, allowing the user to measure time or voltage difference between two points in the capture as well as the signal frequency.

7.2 Real time sampling

Example of a basic usage of the oscilloscope is shown in figure 17. In this example, the output of the PWM generator of the oscilloscope with a frequency of 1kHz is captured by channel 1. Channels 2 and 3 are connected to an external signal generator. The frequency and amplitude of a triangle shaped signal on channel 2 are determined by the horizontal and vertical cursors of the oscilloscope (cursors are indicated by purple colored vertical and horizontal lines in the signal plot). The trigger is set to the rising edge of channel 2 – trigger voltage and position are indicated by light-blue horizontal and vertical lines in the plot view, respectively. The capture channels are vertically scaled down and offset using the channel configuration panel settings, so they don't overlap in the signal plot.



Sample rate 500.000 kS/s
Capture depth 10 kS

Cursors
Channel: 1 [✓ 2] 3
Axis: X Y
X
t_1 3.358ms
t_2 13.38ms
Δt 10.02ms
V_1 2.485V
V_2 2.496V
ΔV 10.47mV
f 99.74Hz
Y
V_1 1.018V
V_2 2.488V
ΔV_2 1.471V

Gen. ON
f 1.000 kHz
Duty 50 %

SMPS mode: [✓ PWM] PFM

Chan. 1 ON
Scale 0.25 x
Offset 1 div

Chan. 2 ON
Scale 0.5 x
Offset 2 div

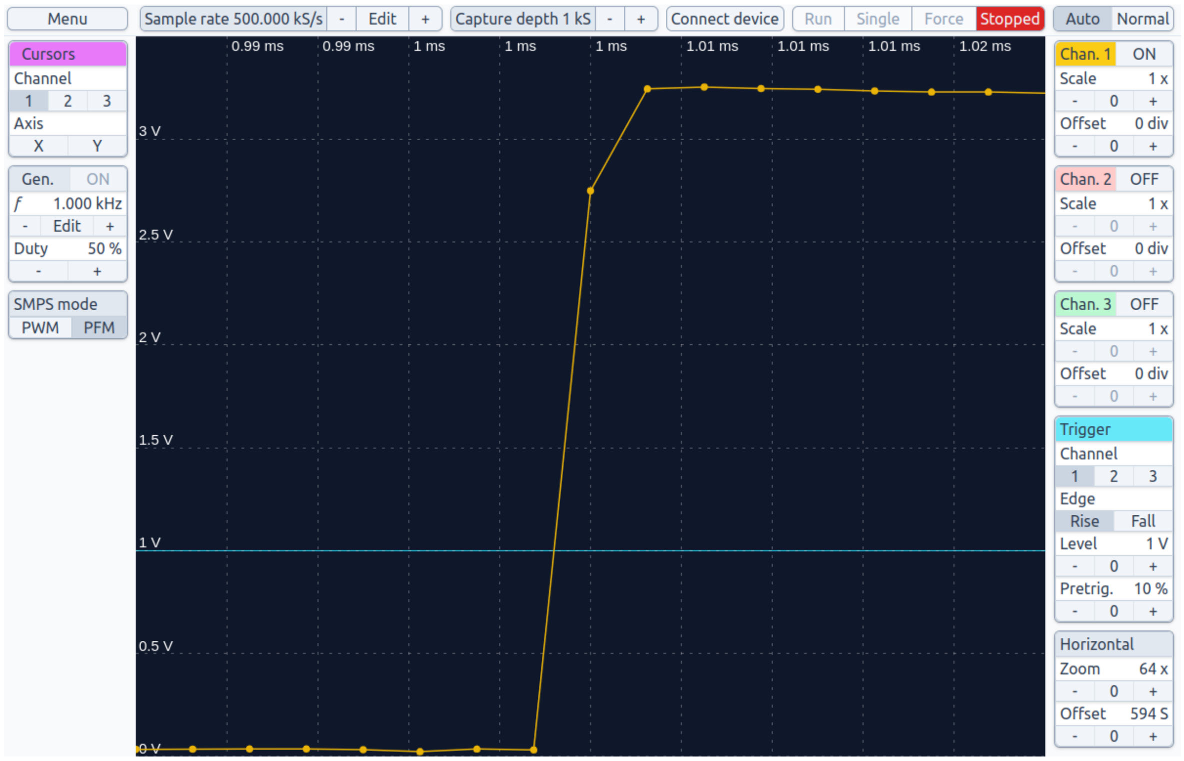
Chan. 3 ON
Scale 0.5 x
Offset 4 div

Trigger
Channel: 1 [✓ 2] 3
Edge: [✓ Rise] Fall
Level 1.5 V
Pretrig. 10 %

Horizontal
Zoom 1 x
Offset 0 S

Figure 17: Example of a capture with three analog channels.

In certain cases, the sample rate of the oscilloscope is not high enough. One such example is shown in figure 18 – a closeup of a rising edge of a 100kHz signal generated by the oscilloscope PWM generator, captured at the maximum oscilloscope sample rate of 500kS/s. Only a single signal sample was captured during the rising edge of the signal. To reach a higher sample rate, a method called equivalent time sampling might be used. This method is described in the following section.



Sample rate 500.000 kS/s
Capture depth 1 kS

Gen. ON
f 1.000 kHz
Duty 50 %

Chan. 1 ON
Scale 1 x
Offset 0 div

Chan. 2 OFF
Scale 1 x
Offset 0 div

Chan. 3 OFF
Scale 1 x
Offset 0 div

Trigger
Channel 1
Edge Rise
Level 1 V
Pretrig. 10 %

Horizontal
Zoom 64 x
Offset 594 S

Figure 18: Rising edge of a 100kHz signal captured using RTS.

7.3 Equivalent time sampling

For certain periodic signals, it is possible to effectively reach a much higher sample rate than the sample rate of the oscilloscope ADC using a technique called equivalent time sampling (ETS). ETS is based on the principle of capturing samples of a signal with frequency f_{PWM} at an appropriately selected frequency f_{SAMP}. This technique is described in detail in [12]. Because the performance of this method is dependent on being able to set the sampling frequency very close to the frequency of the sampled signal, the captured signal is often excited with a signal generator sharing the same clock as the oscilloscope. The interface of the oscilloscope was designed to make the sampling frequency and PWM generator frequency as precisely configurable as the PWM and ADC peripherals of the Raspberry Pi Pico allow.

When capturing a periodic signal with frequency f_{PWM} at an appropriately selected frequency f_{SAMP}, the achieved effective sampling frequency f_{EFF} can be obtained with equation 9 using the coefficient k_{AEQ} calculated using equation 8.

$$k_S \approx \frac{f_{PWM}}{f_{SAMP}} \tag{7}$$

$$k_{AEQ} = \frac{f_{PWM}}{f_{PWM} - k_S \cdot f_{SAMP}} \tag{8}$$

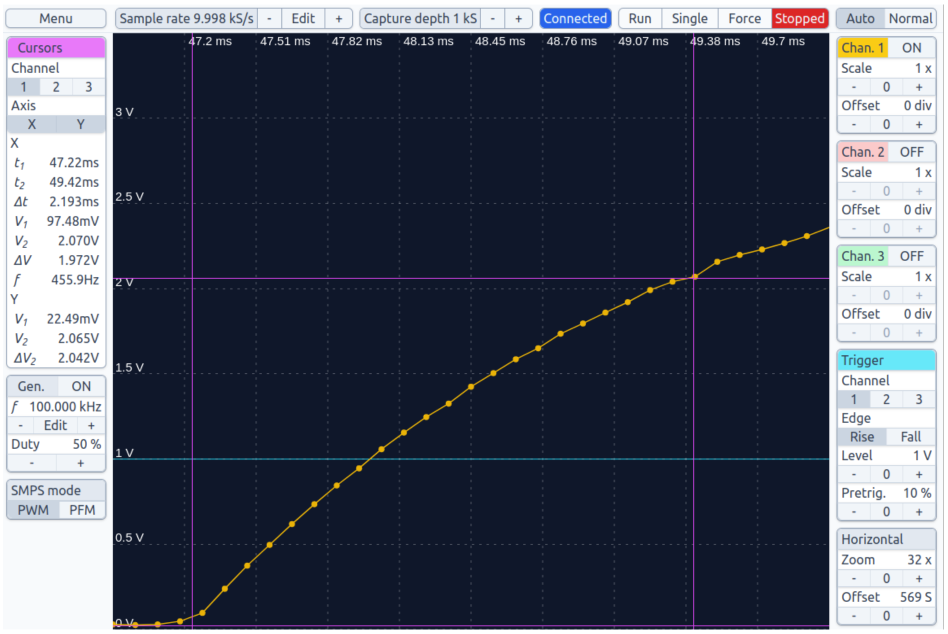
$$f_{EFF} = k_{AEQ} \cdot f_{SAMP} \tag{9}$$

As an example, when the sample rate of the oscilloscope is set to 9997.9171Hz (the closest frequency to 10kHz achievable by the ADC hardware) and the PWM generator frequency is set to 100kHz, the factor k_{AEQ} will be equal to approximately 4800. The effective sample rate achieved is is approximately 48MHz. The coefficient k_{AEQ} can also be used to convert the time scale of the signal captured using ETS to the real time scale of the signal by simply dividing the time scale of the captured signal by the coefficient.

To demonstrate the capabilities of the ETS method, the capacitance of one of the ADC inputs of the Pico was calculated using the configuration described in the above example. By connecting the PWM generator output to the ADC of the oscilloscope through a 10kΩ resistor, an RC circuit was formed. The RC time constant τ, which is the product of the capacitance of the capacitor and the resistance of the resistor in the RC circuit, can be obtained by measuring the time it takes to charge the capacitor to approximately 63% of the voltage applied to the circuit. The measured time, converted to the "real" time scale using the k_{AEQ} coefficient, is approximately 456.9ns. Using equation 10, the internal capacitance of the ADC (together with the capacitance of the breadboard used for connecting the components) was determined to be approximately 45.7pF.

$$\tau = t_{rise} = RC \tag{10}$$

The detail of the rising edge of a signal 100kHz signal generated by the PWM generator of the oscilloscope using the ETS method is shown in figure 19. For comparison, the same signal captured without ETS at the maximum sample rate of the oscilloscope is shown in figure 18.



X
t_1 47.22ms
t_2 49.42ms
Δt 2.193ms
V_1 97.48mV
V_2 2.070V
ΔV 1.972V
f 455.9Hz
Y
V_1 22.49mV
V_2 2.065V
ΔV_2 2.042V

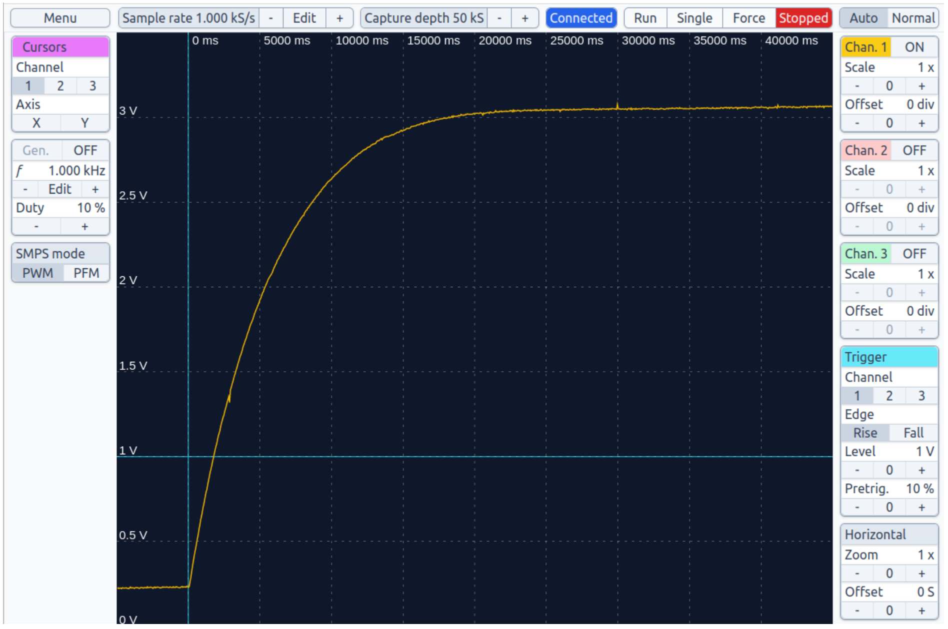
Gen. ON
f 100.000 kHz
Duty 50 %

Chan. 1 ON
Scale 1 x
Offset 0 div
Chan. 2 OFF
Scale 1 x
Offset 0 div
Chan. 3 OFF
Scale 1 x
Offset 0 div
Trigger
Channel 1
Edge Rise
Level 1 V
Pretrig. 10 %
Horizontal
Zoom 32 x
Offset 569 S

Figure 19: Rising edge of a 100kHz signal captured using ETS.

7.4 Long signal Capture

Sometimes, the ability to capture a long duration of a signal can be useful. The relatively large RAM of the RP2040 allows capturing up to one hundred thousand during a single capture. With the minimal sample rate of 1kS/s, up to 100 seconds of a signal can be captured. The Force button of the oscilloscope, which “forces” the capture of the signal immediately after starting can be useful in this case, allowing the oscilloscope to be used as a kind of an analog data logger. As an example, a 50 seconds long capture of a capacitor being charged is shown in figure 20.



Gen. OFF
f 1.000 kHz
Duty 10 %

Chan. 1 ON
Scale 1 x
Offset 0 div
Chan. 2 OFF
Scale 1 x
Offset 0 div
Chan. 3 OFF
Scale 1 x
Offset 0 div
Trigger
Channel 1
Edge Rise
Level 1 V
Pretrig. 10 %
Horizontal
Zoom 1 x
Offset 0 S

Figure 20: Capture of an 470uF capacitor being charged using Force mode.

8 User manual

8.1 Technical specification

Oscilloscope

Oscilloscope channels	3
ADC resolution	12 bits/channel
Input voltage range	0V – 3.3V
Maximum sample rate – real time mode	500kS/s (total for all active channels)
Maximum sample rate – ETS mode	48MHz
Maximum capture depth	100kS (total for all active channels)
Maximum capture time	100s at 1kS/s with one channel active

PWM generator

Frequency	7.5Hz to 50MHz
Duty cycle	1, 2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100%
Amplitude	Low: 0V, high: 3.3V

8.2 Connections (Pinout)

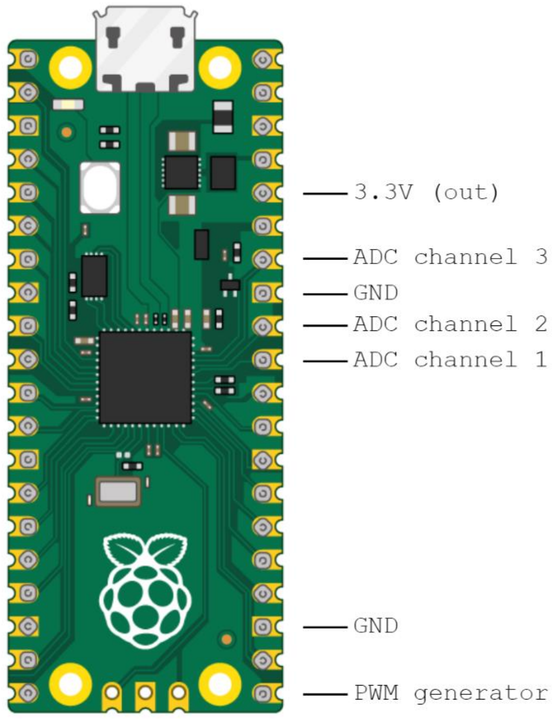


Figure 21:Oscilloscope ADC channels and PWM generator pin locations on the Pico.

8.3 Flashing the firmware

The Raspberry Pi Pico needs to be flashed with the oscilloscope firmware before it can first be used. To flash the firmware, the following steps need to be performed:

1. If the Pico is connected to the computer, disconnect it first
2. Press and hold the BOOT button on the Pico
3. While keeping the BOOT button pressed, connect the pico to the computer
4. Release the BOOT button
5. The Pico will appear as an USB storage device in the computer file system
6. Copy the *.uf2* file with the firmware to the Pico
7. The Pico is flashed with the firmware and is ready to be used

8.4 Browser and operating system compatibility

The oscilloscope depends on the WebUSB API for communication between the user interface and the Raspberry Pi Pico. Currently, WebUSB API is only available on Chrome, Edge and Opera web browsers. Current browser compatibility status, including compatible browser versions, is available at the CanIUse website [13], which tests web browser compatibility with web standards and APIs.

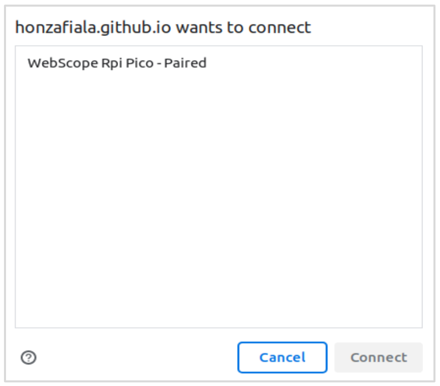
To use the oscilloscope on Linux, an *udev rule* needs to be added to enable the web browser access to USB devices. The udev rule can be added by performing the following terminal commands:

```
cd /etc/udev/rules.d
sudo echo SUBSYSTEM=="usb",ATTR{idVendor}=="cafe",MODE="0664", GROUP="plugdev" > pico.rules
```

Code snippet 7: Linux terminal commands to add an udev rule to the system.

8.5 Connecting to the Raspberry Pi Pico

Once the Pico is flashed with the WebScope firmware, load the GUI web page or open the *index.html* file in case you downloaded the offline version of the GUI. Clicking the **Connect device** button in the welcome screen or on the top bar will bring up a pop-up window similar to the one shown in figure 22. In case the pop-up window does not appear, the browser you are using does not support the WebUSB API required for communication with the Pico – consult section 8.4 to learn about browser compatibility.


honzafiala.github.io wants to connect

WebScope Rpi Pico - Paired

Figure 22: WebUSB connection pop-up browser window.

In the pop-up window, select *WebScope Rpi Pico* and press the **Connect** button. In case the Raspberry Pi Pico does not appear in the pop-up window, it is probably not flashed with the WebScope firmware – flashing the firmware is described in section 8.4.

8.6 Capture modes

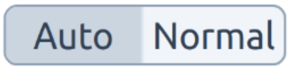


Figure 23: Auto/normal mode selection panel.

The oscilloscope has two trigger modes, auto and normal. Trigger mode can be set with the buttons **Auto** and **Normal**, located on the top bar.

- In Normal mode, the oscilloscope waits for a trigger condition indefinitely.
- In Auto mode, the oscilloscope first waits for a trigger condition for a defined time period, equal to three times the capture length. If the trigger condition does not occur in this time window, the oscilloscope then starts the signal capture automatically. The later mode is useful for capturing an unknown signal, where the user is not sure how to set the trigger condition.

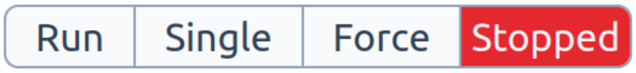


Figure 24: Capture control panel in stopped state.

Independently of the Auto and Normal trigger modes, there are three capture modes in which the oscilloscope can be started:

- In **Run** mode, the oscilloscope is capturing continuously - after a trigger occurs and the signal is captured, new capture is started immediately.
- In **Single** mode, only a single capture is performed, and the oscilloscope stops.
- In **Force** mode, the oscilloscope “forces” a trigger immediately after capture.

8.7 Capture settings

8.7.1 Sample rate setting

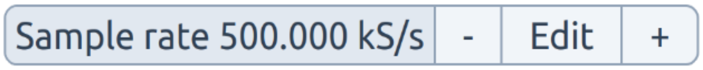


Figure 25: Sample rate configuration panel.

The sample rate of the oscilloscope is the frequency at which new signal samples are captured. The oscilloscope has only one analog-digital converter, which is shared between multiple channels when more than one channel is active. This means that the per-channel sample rate is equal to the sample rate divided by the number of active channels. The sample rate configuration panel, which is located on the top bar, allows the user to set the desired sampling frequency. The sample rate can be changed in defined steps using the **-** and **+** buttons.

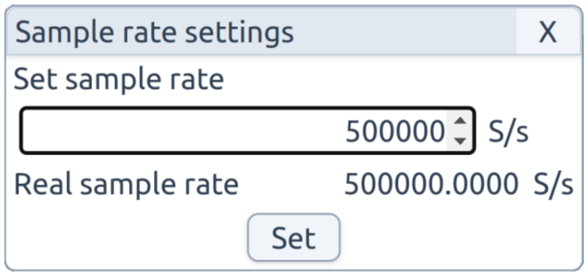


Figure 26: Sample rate configuration pop-up window.

The sample rate can be set more precisely by clicking the **Edit** button in the sample rate configuration panel, which opens a configuration pop-up window (shown on picture above). In the pop-up window, the value can be set using numerical keys on the keyboard. After the desired sample rate value is set, clicking the **Set** button in the pop-up window configures the oscilloscope to the new value. Due to limitations of the hardware, not all sample rate values can be achieved by the oscilloscope. The oscilloscope calculates the closest sample rate it can achieve - the actual sample rate is shown in the pop-up window, below the sample rate set by the user. The minimum sample rate of the oscilloscope is 1kS/s and the maximum is 500kS/s.

8.7.2 Capture depth setting

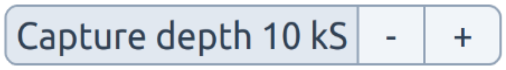


Figure 27: Capture depth configuration panel.

The capture depth is the number of samples that will be captured during a single capture. The capture buffer is shared between all active channels, so when more than one channel is active, the per-channel number of samples that will be captured is equal to the capture depth divided by the number of active channels. Capture depth can be configured using the **-** and **+** buttons in the capture depth configuration panel, which is located on the top bar. The minimum capture depth is 1kS and the maximum is 100kS.

8.7.3 Active channels

The three channels of the oscilloscope can be turned on and off individually using the **ON/OFF** buttons in the channel configuration panels, located on the right side bar. The number of active channels influences the maximum achievable per-channel sample rate and capture depth, so it is advised to turn off unnecessary channels when maximum performance is needed.

8.8 Trigger settings

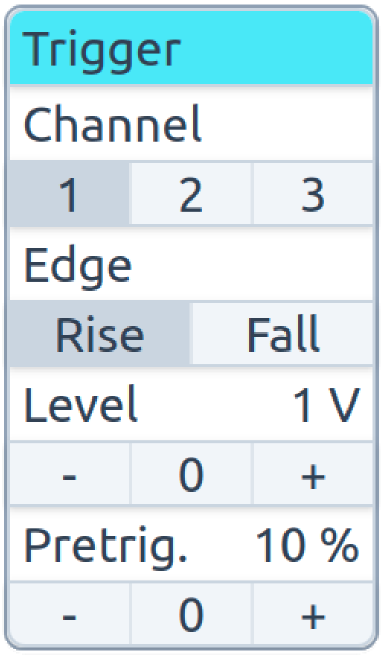


Figure 28: Trigger configuration panel.

The trigger settings define the conditions which cause the signal capture to start. The trigger channel, trigger edge, trigger voltage and pretrigger value can be set in the trigger configuration panel located on the right side bar.

- The trigger channel is the channel which the oscilloscope checks for the trigger condition. The channel can be set using the **1**, **2**, and **3** buttons in the *Channel* section of the trigger configuration panel
- The oscilloscope can trigger either on the rising edge or the falling edge of the signal. The behavior can be set using the **Rise** and **Fall** buttons in the *Edge* section of the trigger configuration panel.
- The trigger level is the voltage level at which the oscilloscope checks for the trigger condition. The trigger level voltage can be set using the **-** and **+** buttons in the *Level* section of the trigger configuration panel.
- The pretrigger value determines the portion of the signal before the trigger condition which will be included in the capture. The pretrigger value can be set using the **-** and **+** buttons in the *Pretrig.* section of the trigger configuration panel. For example, when the pretrigger value is set to 10%, the ratio of captured samples before and after the trigger will be 10%/90%. The pretrigger value can be set to values between 0% and 100%.

8.9 Vertical (channel) view settings

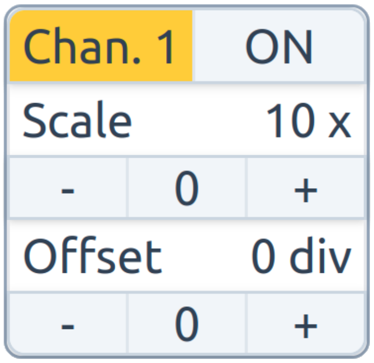


Figure 29: Channel 1 configuration panel.

The captured signals can be individually horizontally scaled using the **-** and **+** buttons in the *Scale* section of the channel configuration panels located on the right side bar. Similarly, signals can be moved up and down in the plot using the **-** and **+** buttons in the *Offset* section of the channel configuration panels. Both the scale and offset can be reset using the **0** buttons in the *Scale* and *Offset* sections of the configuration panel.

8.10 Horizontal view settings

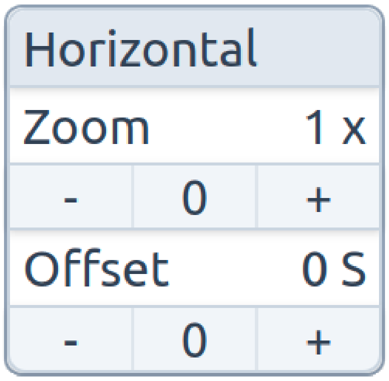


Figure 30: Horizontal control panel.

The captured signals can be horizontally zoomed in or out using the **-** and **+** buttons in the *Zoom* section of the horizontal control panel. A zoomed-in signal can be moved to the right or left using the **-** and **+** buttons in the *Offset* section of the horizontal control panel. Both the zoom and offset can be reset using the **0** buttons in the *Zoom* and *Offset* sections of the horizontal configuration panel. To make navigation in the captured signal easier, it is also possible to use the Up/Down arrow keys on the keyboard to zoom the signal in or out and the Left/Right arrow keys to move the signal to the left or right.

8.11 Cursor settings

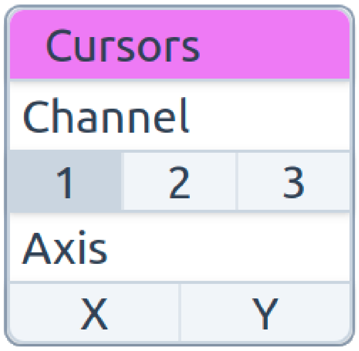


Figure 31: Cursor control panel

Oscilloscope cursors can be used to precisely measure voltage, time or voltage and time difference in the captured signal. The channel to be measured can be set using the **1**, **2**, and **3** buttons on the cursors panel, located on the left side bar. Clicking the **X** and **Y** buttons on the control panels activates the X (vertical) and Y (horizontal) cursors. The cursors are indicated by purple lines inside the signal view and can be moved by dragging them with a mouse over the signal view.

8.12 Generator settings

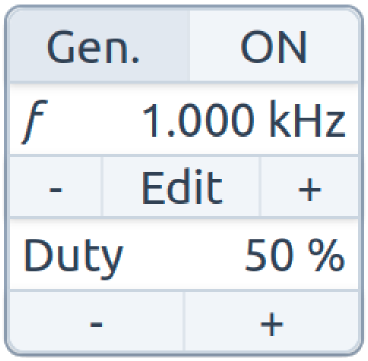
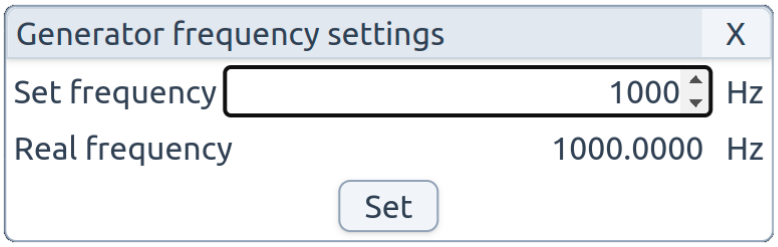


Figure 32:PWM generator configuration panel.

The oscilloscope features a configurable PWM signal generator. The generator can be activated using the **ON/OFF** button in the generator configuration panel located on the left side bar. The frequency of the signal can be adjusted by 100 Hz increments using the **-** and **+** buttons in the frequency section of the control panel. The duty cycle (the percentage of the signal period during which the signal is in high state) can be adjusted by 10% increments using the **-** and **+** buttons in the duty cycle section of the control panel.



Set frequency 1000 Hz

Real frequency 1000.0000 Hz

Figure 33: PWM generator frequency setting pop-up window.

The signal frequency can be set more precisely by clicking the **Edit** button in the generator configuration panel, which opens a configuration pop-up window (shown in picture above). In the pop-up window, the value can be set using numerical keys on the keyboard. After the desired frequency is set, clicking the **Set** button in the pop-up window configures the generator to the new value. Due to limitations of the hardware, not all frequencies can be achieved by the generator. The oscilloscope calculates the closest PWM frequency it can achieve - the actual frequency is shown in the pop-up window, below the frequency set by the user. The minimum frequency of the generator 7.5Hz and the maximum is 100MHz.

8.13 Capture export

TBD - CSV export button in the oscilloscope menu.
The captured signals can be downloaded in CSV format for additional processing. To download the capture, click the **Export CSV** button located in the menu (located in the top left corner of the GUI) of the oscilloscope. The csv file has the following format:

```
Ch1, .. Chn, Timestamp,
Ch1, .. Chn, Timestamp,
Ch1, .. Chn, Timestamp,
Ch1, .. Chn, Timestamp,
...
```

Where the Ch1, ... Ch2 values on each row are the samples of the active channels and Timestamp is the time of capture for each row in milliseconds.

9 Conclusion

The goal of this thesis was to design and implement a software-defined oscilloscope based on the Raspberry Pi Pico microcontroller and a web-based user interface. The first part of the thesis is devoted to implementing communication between the Pico and a web application using the WebUSB API. After the communication using the API was established, the firmware and the GUI web application were implemented. The concept of using a web application for controlling a software defined instrument has proven to be effective, even with a highly interactive interface of an oscilloscope. The motivation behind the thesis – developing a platform-independent oscilloscope capable of running on all major operating systems was also met.

The oscilloscope meets all major criteria that were set: it features three analog channels, cursors, PWM generator, precise sample rate and PWM frequency settings to enable equivalent time sampling.

While the performance of the oscillocope is limited by the used hardware, an effort was made to utilize all the hardware resources as much as possible. Using the PWM generator of the developed oscilloscope, an effective sample rate of up to 48MHz was reached for certain suitable signals using the equivalent time sampling method.

Hopefully, the oscilloscope will find its use in classrooms and in the homes of students and hobbyists, making electronics education and experiments more accessible.

10 List of figures

11 List of abbreviations

ADC	Analog-digital converter
API	Application interface
CSV	Comma separated values
CSS	Cascading style sheets
DMA	Direct memory access
ETS	Equivalent time sampling
GUI	Graphical user interface
HTML	Hypertext markup language
IDE	Integrated development environment
PFM	Pulse-frequency modulation
PWM	Pulse-width modulation
RTS	Real time sampling
SDI	Software defined instrument
SVG	Scalable vector graphics
SWD	Serial wire debug
UART	Universal asynchronous receiver-transmitter

12 Bibliography

[1] *Little Embedded Oscilloscope*.
URL: https://embedded.fel.cvut.cz/platformy/leo

[2] *WebUSB API, Draft Community Group Report*. 2023.
URL: https://wicg.github.io/webusb/

[3] *Accessing USB devices on the Web*. 2023.
URL: https://developer.chrome.com/en/articles/usb/

[4] *TinyUSB.* 2023.
URL: https://docs.tinyusb.org/

[5] *WebUSB Tester*. 2022
URL: https://larsgk.github.io/webusb-tester/

[6] *Capturing USB trafic on Linux using WireShark*. 2015.
URL: https://stackoverflow.com/questions/31054437/how-to-install-wireshark-on-linux-and-capture-usb-traffic

[7] *RP2040 Datasheet*. 2022.
URL: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[8] V. Vaněček, *Logic analyser based on Raspberry Pi Pico.* Diploma thesis. 2022.

[9] *Raspberry Pi Pico SDK.* 2023.
URL: https://github.com/raspberrypi/pico-sdk

[10] *Getting started with Raspberry Pi Pico*. 2022.
URL: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf

[11] *Protocol Buffers Documentation.* 2023.
URL: https://protobuf.dev/

[12] *Increasing of Sampling Rate of Internal ADC in microcontrollers by Equivalent-Time Sampling*. IMEKO International symposium proceedings. 2022.
URL:https://embedded.fel.cvut.cz/sites/default/files/kurzy/ETC22/Prednasky_ETC22_B/IMEKO_2022_Proceedings.pdf

[13] *Can I use*. 2023.
URL: https://caniuse.com/