Building a WebUSB or Web Serial Gadget

Have you ever dreamed of creating a cool microcontroller-based gadget to connect to your PC, tablet, or phone via USB? Many $5 to $10 microcontrollers have USB built-in now and you just need a USB connector to use it!

But then, have your dreams ever been squashed by the thought of creating an "app" for a half-dozen different platforms, involving multiple languages, device drivers, app stores, etc.? If so, you are not alone -- and it was for this reason that some smart people developed the WebUSB and Web Serial APIs that run directly in a Chromium-based web browser!

With WebUSB and Web Serial APIs your dreams can again come true -- you can write a simple webpage that can be opened on a Windows, Mac, or Linux PC, a Chromebook, or even an Android phone, and you can communicate with the USB interface of your microcontroller-based gadget using only platform-independent JavaScript running in a webpage in your web browser!

You can write your "app" once, in JavaScript, and never even need to install software on a host computer to use it -- just open a webpage and you are up and running, with your host computer talking to your cool microcontroller-based gadget!

My Dream...

As for me, for many years, I have dreamed of taking an advanced USB-capable microcontroller (MCU) and creating a trivial oscilloscope using its A/D Converter! I dreamed of building a tiny piece of hardware with a USB connector on one side and a BNC connector on the other side, which could interface between a host computer and a normal oscilloscope probe. I dreamed of just hooking this up to a PC or tablet or phone for power and having instant access to probe signals and traces, without the need of any heavy-weight/bloated oscilloscope software that always seems to fall out-of-date!

And now I have finally built it -- it is called Flea-Scope™, based on the PIC32MK MCU! And it works with a Chromium-based web browser!

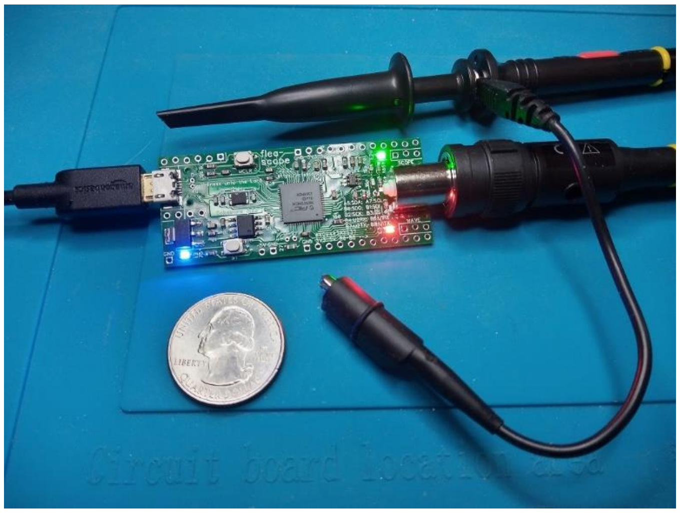


Figure 1: Flea-Scope™

You can read all about it (and build one yourself if you want!) here:

Flea-Scope™ USB Oscilloscope (18 Msps, $13 BoM, WebUSB)

`https://hackaday.io/project/192598-flea-scope-usb-o-scope-18-msps-13-bom-webusb`

All the code is on github (links below), as well, so you can even make your own custom project following my pattern!

The Flea-Scope™ Hardware

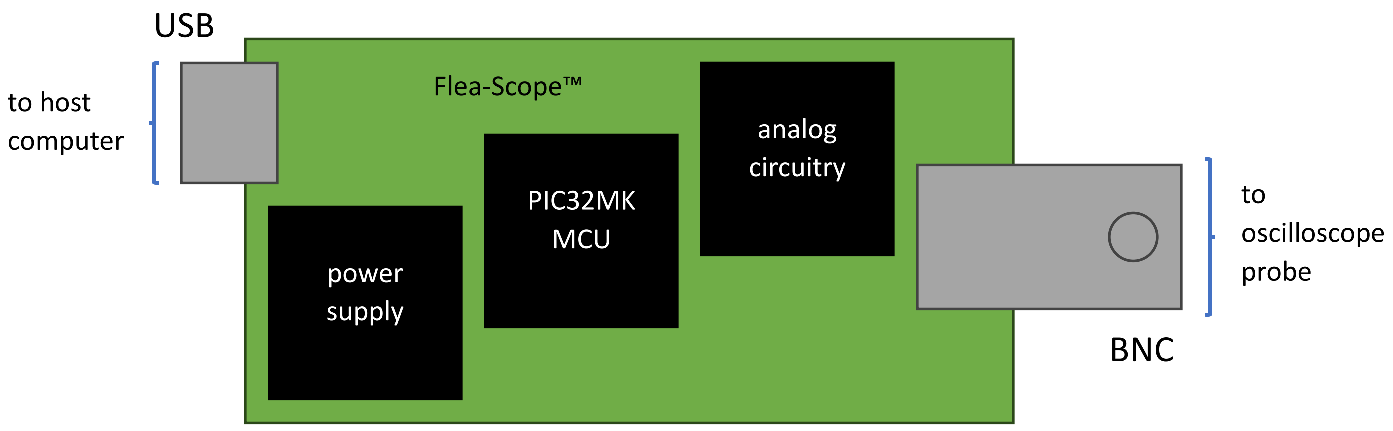


Figure 2:The Flea-Scope™ Hardware

Power Supply

I get 5V from the USB and generate -5V from the 5V using a DC-to-DC converter, for use by the analog circuitry. I also regulate 3.3V from 5V, for use by the MCU.

The MCU

I use a PIC32MK microcontroller that has an array of five fast A/D converters as well as a full-speed USB interface. Each A/D converter can run 3.75 million samples per second (Msps), and I configure them to run interleaved to achieve a combined rate of just over 18 Msps! I use built-in analog comparators and analog voltage reference (using a D/A converter) to automatically detect a “trigger level or edge” on the input signal, and then start the A/D converters gathering samples into a RAM buffer once trigger has been detected. I then cook the RAM buffers and upload them to the USB interface using the CDC/ACM serial protocol, for display by the webpage user interface!

Analog Circuitry

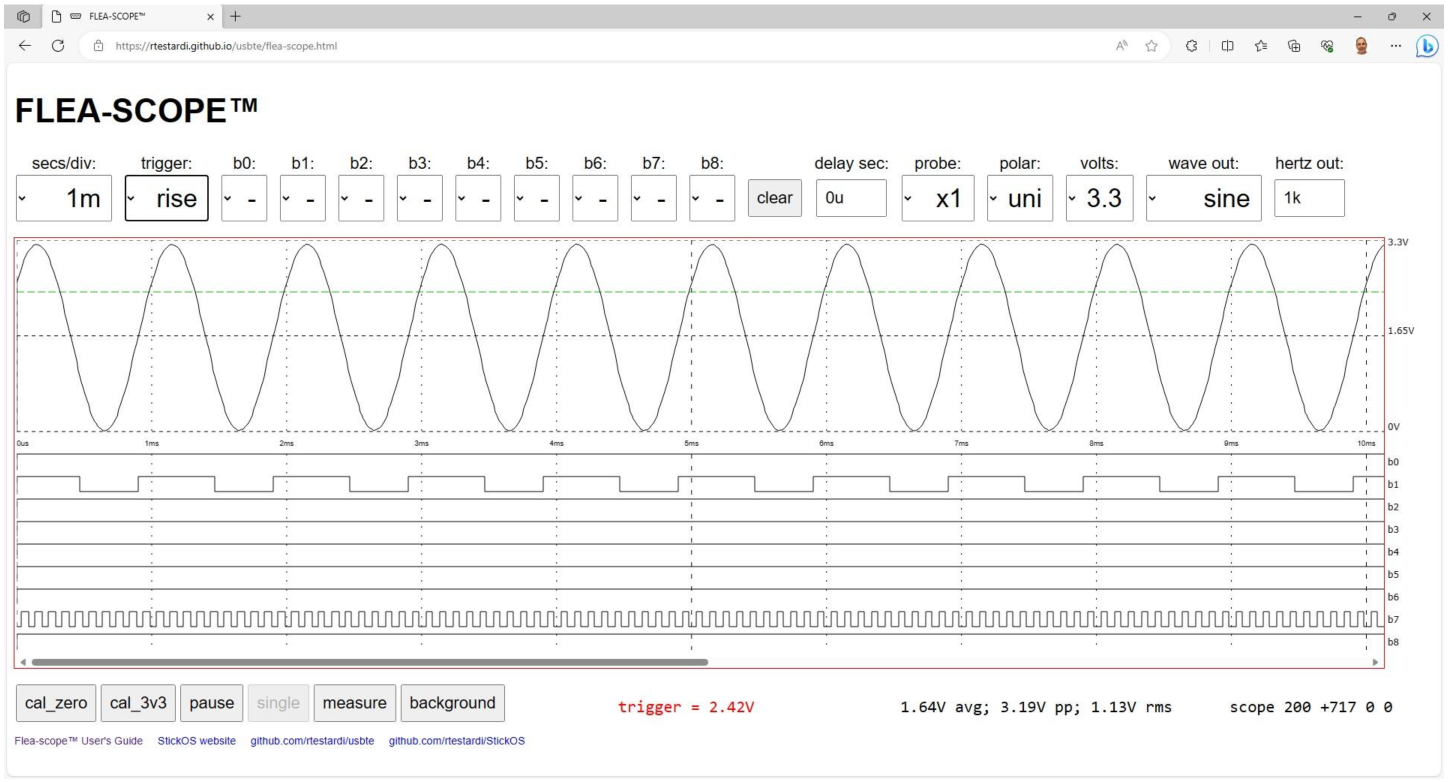
The analog circuitry includes an op amp to achieve 1 megaohm input impedance (and 12pF) to be compatible with most 1x and 10x oscilloscope probes. It also includes passive pull-down/pull-up circuitry so that the 1x oscilloscope probe voltages of -6.6V to +6.6V correspond to MCU A/D input voltages of 0V to 3.3V, since the MCU A/D converters can only measure that more limited range. With a 10x oscilloscope probe, Flea-Scope™ can measure input voltages from -66V to 66V.

The Flea-Scope™ Firmware

The firmware inside Flea-Scope™ is based on the StickOS® BASIC operating system. It exposes the CDC/ACM serial protocol out the USB port from where commands can be received from the host computer on USB endpoint 2, and results can be returned to the host computer on USB endpoint 3. It uses a simple command/response mechanism, where the webpage user interface sends a command to the Flea-Scope™ to initiate trigger and capture, including parameters like the trigger level or edge voltage and sample rate, and then after trigger and capture has occurred, the Flea-Scope™ returns the sample data.

The firmware inside the Flea-Scope™ also manages all the analog and digital peripherals inside the MCU, which is quite an intricate dance between many moving pieces. See the Flea-Scope™ User’s Guide (link below) for a “how it works” section with block diagrams at the end.

The Flea-Scope™ Software (i.e., the GUI Webpage!)



FLEA-SCOPE™

secs/div: 1m | trigger: rise | b0: - | b1: - | b2: - | b3: - | b4: - | b5: - | b6: - | b7: - | b8: - | delay sec: 0u | probe: x1 | polar: uni | volts: 3.3 | wave out: sine | hertz out: 1k

```
trigger = 2.42V          1.64V avg; 3.19V pp; 1.13V rms      scope 200 +717 0 0
```

Figure 3:The Flea-Scope™ Software (i.e., the Webpage!)

Now we get to the main point of this article -- how to communicate with your cool microcontroller-based gadget using WebUSB or Web Serial APIs -- in other words, how to write your “app” once, in JavaScript, and never even need to install software on a host computer to use it!

First, let’s clarify the difference between WebUSB and Web Serial... These are two very similar but different API protocols which talk to drivers on the host platform to talk to your USB gadget. Which one you use depends on whether the host platform (Windows, Mac, Linux, ChromeOS, or Android) has bound its own “serial driver” to the CDC/ACM port exposed to the USB by your gadget or not.

Basically, if the host platform has bound its “serial driver” to the CDC/ACM port, like Windows or ChromeOS do on the left below, then WebUSB cannot also bind to the CDC/ACM port, since only a single driver can bind to a given port (since the driver gets exclusive ownership of the port). So, in that case, you would use Web Serial to bind on top of the serial driver bound by the host platform.

If, on the other hand, the host platform does not bind a “serial driver” to the CDC/ACM port, like Mac or Linux or Android on the right below, then WebUSB is free to bind to the CDC/ACM port directly, since it is the only driver using the port (and can get exclusive ownership of the port).

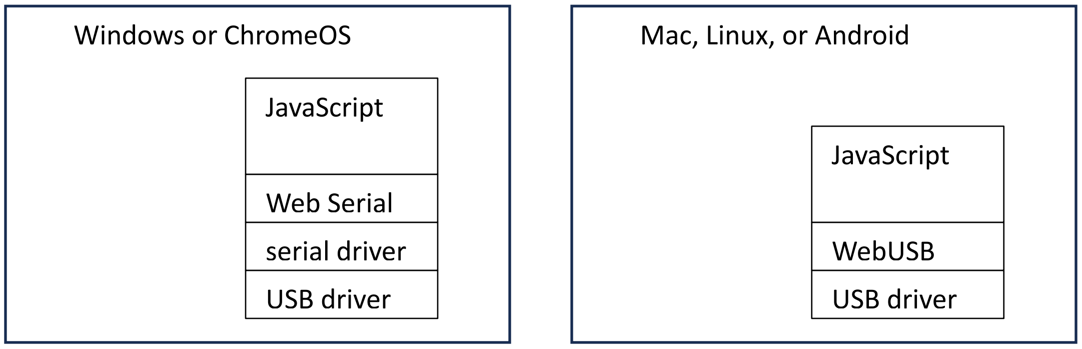


Figure 4: WebUSB vs. Web Serial

The advantage of using the well-known CDC/ACM protocol in your device is you don't need an *.inf file for Windows.

This is a tiny bit messy to support both models dependent on the host platform, but not terribly so. Both models have similar APIs to open the CDC/ACM port and read or write data from or to it, so we just use an if-else based on the host platform type.

Connect to the CDC/ACM Port

For starters, for security reasons, you must be in a user-initiated action function to connect to a USB device, so the browser can ask the user for permission. The easiest way to do this is to just put a button on the webpage that the user must click, like:

```
Select:
<button type='button'
        onclick="if (/\b(CrOS|Windows)\b/.test(navigator.userAgent)) {
                     Comm();
                 } else {
                     Usb();
                 }">Connect</button>
```

The regular expression magic there just checks to see if this is Windows or ChromeOS, and if so, uses Web Serial; otherwise, it uses WebUSB.

Alternately, you can have two buttons and have the user decide, like:

```
Select:
<button type='button' onclick="Comm();">COM</button>
or
<button type='button' onclick="Usb();">USB</button>
```

The actual JavaScript functions Comm() and Usb() are below:

```
var usb;
var comm;
var epin = 2;
var epout = 3;
var reader;
var writer;
var string = "";

// the user wants to connect to a USB device's bulk endpoints directly
async function Usb()
{
    var filter = [
        { vendorId: 0x04D8, productId: 0xE66E },
        // XXX -- add any vid/pid filters for your device here
    ];

    if ('usb' in navigator) {
        try {
            usb = await navigator.usb.requestDevice({ filters: filter });
            await usb.open();
            await usb.selectConfiguration(1);
            await usb.claimInterface(1);

            Send(...);  // XXX -- send any strings up front to the device

            setTimeout(Receive, 1);
        } catch(err) {
            // XXX -- something went wrong; show err.message
        }
    } else {
        // XXX -- this browser does not support WebUSB; show an error
    }
}

// the user wants to connect to an emulated COM port
async function Comm()
{
    var filter = [
        { usbVendorId: 0x04D8, usbProductId: 0xE66E },
        // XXX -- add any vid/pid filters for your device here
    ];

    if ('serial' in navigator) {
        try {
            comm = await navigator.serial.requestPort({ filters: filter });
            await comm.open({ baudRate: 115200, bufferSize: 1024 });
            reader = comm.readable.getReader();
            writer = comm.writable.getWriter();

            Send(...);  // XXX -- send any strings up front to the device

            setTimeout(Receive, 1);
        } catch(err) {
            // XXX -- something went wrong; show err.message
        }
    } else {
        // XXX -- this browser does not support Web Serial; show an error
    }
}
```

These functions just ask WebUSB or Web Serial to ask the user for permission to open a USB device matching the vendor and product ids in the filter. If the user agrees, they then open the device and perform some minimal configuration. Then optionally, they might send some data to the device. Finally, they start a background function to receive data from the device using a state machine.

Write to the CDC/ACM Port

Writing to the CDC/ACM port is as simple as sending a string to the function below:

```
// send a string to the device
async function Send(str)
{
    if (usb) {
        await usb.transferOut(epout, enc.encode(str).buffer);
    } else {
        await writer.write(enc.encode(str).buffer);
    }
}
```

Read from the CDC/ACM Port

Reading from the CDC/ACM port is a bit more complicated, as you read in the background, potentially in pieces as the data is received, not necessarily in response to what you sent above, and have to aggregate and parse the data in a state machine -- in my case, I'm looking for a line with the word "done" to tell me I have all the results needed to display the trace:

```
// receive a string from the device
async function Receive()
{
    var str;
    var result;

    if (usb) {
        result = await usb.transferIn(epin, 1024);
        str = dec.decode(result.data.buffer);
    } else {
        result = await reader.read();
        str = dec.decode(result.value);
    }
    string += str.replace(/</g,'&lt;').replace(/>/g,'&gt;');

    // XXX -- do something with the data in string, like look for the word "done"!
    // XXX -- clear string to "" when done doing something with the data

    setTimeout(Receive, 1);
}
```

More on JavaScript

JavaScript can do so much! You can store persistent data in the web browser "cookies and other site data" store, so we use this to store calibration data. You can perform graphics operations, like we do for drawing analog waveforms or digital traces. If you want your program to run in the background, you must go thru a few extra push-ups -- a simple way to achieve this is to generate some periodic noise, which the Flea-Scope™ GUI has an option for.

Conclusion: “You don’t even need an ‘app’ for that!!!”

Using just JavaScript and WebUSB and/or Web Serial in a Chromium-based web browser, you can write a user interface for your cool microcontroller-based gadget just once, and deploy it across most USB host computers -- a Windows, Mac, or Linux PC, a Chromebook, or even an Android phone!

“You don’t even need an ‘app’ for that!!!”

Links

The hackaday page for Flea-Scope™, with full build instructions (give me a like, and make me smile! :-), is here: `https://hackaday.io/project/192598-flea-scope-usb-o-scope-18-msps-13-bom-webusb`

See a video of Flea-Scope™ in action here: `https://photos.onedrive.com/share/90675E0DD8A0AA7E!86984?cid=90675E0DD8A0AA7E&resId=90675E0DD8A0AA7E!86984&authkey=!AI9ZTZLr4fAe8N0&ithint=video&e=Q0UZW3`

A number of examples of WebUSB and Web Serial usage, ranging from a simple terminal emulator to the Flea-Scope™ GUI are here: `https://github.com/rtestardi/usbte`

The Flea-Scope™ firmware for the PIC32MK MCU, including both the USB CDC/ACM port as well as all the logic to manage the A/D converters, etc., is here: `https://github.com/rtestardi/StickOS2`

The Flea-Scope™ User’s Guide, including a “how it works” section with block diagrams at the end, is here: `https://rtestardi.github.io/usbte/flea-scope.pdf`

The PIC32MK datasheet starts here: `https://www.microchip.com/en-us/product/PIC32MK0512GPK064`

WebUSB documentation is here: `https://wicg.github.io/webusb/`

Web Serial documentation is here: `https://wicg.github.io/serial/`

CDC/ACM documentation starts here: `https://en.wikipedia.org/wiki/USB_communications_device_class`

About the Author

Richard Testardi has a wife and 17yo daughter and lives in Colorado. He is grateful most of the time and Christian. He loves anything outdoors or math/science related. In the future, he hopes to be teaching high school students. He lives without a cell phone (well, except a sim-less phone for interoperability testing!)!