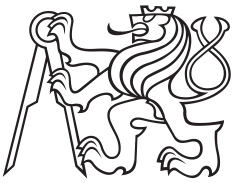
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering

Department of Measurement

# Learning and automation GPIO platform

Ondřej Hruška

Supervisor: doc. Ing. Radislav Šmíd, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Sensors and Instrumentation
2018

# MASTER'S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Hruška Ondřej** Personal ID number: **420010**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

Branch of study: **Sensors and Instrumentation**

## II. Master's thesis details

Master's thesis title in English:

**Learning and Automation GPIO Platform**

Master's thesis title in Czech:

**Výuková a automatizační GPIO platforma**

Guidelines:

Design and implement a modular system consisting of a motherboard and additional modules for connecting sensors, actuators and general inputs via I2C, SPI, UART, 1-Wire or other interfaces to the central system via USB, UART, and wireless interfaces. Allow access to built-in processor peripherals such as ADC, DAC, and timers (PWM, frequency measurement). Design a comfortable way to set the configuration without firmware changes. For the designed system, create a service library in C, Python, and MATLAB.

Bibliography / sources:

[1] STMicroelectronics datasheets, http://www.st.com
[2] Ganssle, J.: The Art of Designing Embedded Systems, Elsevier Science, 2008.
[3] Chi, Qingping & Yan, Hairong & Zhang, Chuan & Pang, Zhibo & Da Xu, Li. (2014).: A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment. Industrial Informatics, IEEE Transactions on. 10. 1417-1425. 10.1109/TII.2014.2306798.

Name and workplace of master's thesis supervisor:

**doc. Ing. Radislav Šmíd, Ph.D., Department of Measurement, FEL**

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

Date of master's thesis assignment: **10.01.2018** Deadline for master's thesis submission: _____

Assignment valid until:
**by the end of summer semester 2018/2019**

| doc. Ing. Radislav Šmíd, Ph.D. | Head of department's signature | prof. Ing. Pavel Ripka, CSc. |
|---|---|---|
| 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

# Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 27. května 2018

.....

# Acknowledgements

blabla

# Abstract

This work focuses on the design of an AC appliance degradation detector. The goal is to implement a device in the form of a power-plug adapter that could be used to monitor and study the characteristics of the AC current.

A prototype with a STM32 F3 processor and an ESP8266 programmable WiFi module has been realised, together with a custom firmware for both processors, which allows easy access to the measurements and charts using a web browser. The device also supports regular reporting to a Xively or ThingSpeak monitoring server.

**Keywords:**

**Supervisor:** doc. Ing. Radislav Šmíd, Ph.D.

# Abstrakt

Tato práce se zabývá implementací detektoru poruch a degradací síťového spotřebiče pomocí analýzy časového průběhu odebíraného proudu. Cílem je navrhnout a realizovat přístroj ve formě zásuvkového adaptéru, který by bylo možné použít k monitorování připojeného zařízení.

V rámci práce byl realizován prototyp přístroje s procesorem řady STM32 F3 a programovatelným WiFi modulem ESP8266. Zařízení umožňuje pohodlné ovládání a zobrazení grafů spektra a průběhu proudu pomocí webového rozhraní. Dále je podporováno pravidelné hlášení stavu na servery Xively a ThingSpeak.

**Klíčová slova:**

**Překlad názvu:** Výuková a automatizační GPIO platforma

# Contents

# Figures

# Chapter 1

# Introduction

Prototyping, design evaluation and the measurement of physical properties in experiments make a daily occurrence in the engineering praxis. This task typically involves the generation and capture of electrical signals coming to and from specialized sensors, actuators and other circuitry. As the technology advanced, mainly driven by the consumer electronics market and the automotive industry, a variety of affordable integrated sensors became available. Those devices can provide a sufficient accuracy and precision for the task at hand while keeping the circuit complexity down by integrating a large portion of the necessary circuitry together with the sensor on a single chip or in a compact module. Those modern components gives as a low cost alternative to expensive laboratory equipment and much larger instruments previously used.

concrete examples

However, the drive for miniaturization and the advent of modern hardware buses, in particular USB (Universal Serial Bus), lead to the disappearance of low level computer ports, such as the once ubiquitous parallel port, that would provide an easy way of connecting those digital devices (and low level hardware in general) to personal computers.

Today, when one wants to perform some measurements using a digital sensor, the usual route is to implement an embedded firmware for a microcontroller that can be connected to the PC through USB. This approach makes it possible to optimize the solution for a particular task and achieve high performance. However, building such a specialized tool, or even writing the firmware alone, is time-consuming and requires domain knowledge that is entirely removed from the measurements we want to perform with it.

Clearly it would be advantageous to have a way to easily attach those integrated devices and low other level hardware to a PC without having to burden ourselves with technicalities of the connection, even at the cost of lower performance compared to a specialized device or a professional tool. The design and implementation of such a user-friendly, general purpose hardware interfacing system is the object of this work.

# Chapter 2

# Goals and Requirements

This chapter discusses the project requirements and presents our expectations of the final outcome.

The work's main objective is the implementation of a reconfigurable embedded firmware, a physical module providing the required digital buses, signal generation and acquisition capabilities, and the PC libraries to work with it. It's expected that several prototypes of the hardware platform will be developed to evaluate different form factors.

## 2.1 Project Name

Every project needs a memorable name. During the development, the name "GEX" was chosen, an acronym originating in the term *GPIO Expander*, which, although not describing its scope perfectly, alludes to the project's primary purpose of providing low level GPIO capabilities to personal computers. The term GEX may be used through the text to refer to the whole project or hardware modules developed for it.

## 2.2 Expected Use Cases

First, we must consider in which situations the module could be helpful. As we explained in the introduction, GEX should allow the user to connect digital sensors and electronic circuits to a PC and work with them using high level application software.

This could be used to get familiar with a new chip or a module before using it in some hardware project, to measure characteristic curves of a component, to collect experimental data from a test setup, or, for instance, to control a positioning motor.

The applications can have a temporary character, a simple setup that is used once and then dismantled, or a more permanent one. An example of the latter could be students' laboratory tasks where the measurement setup is prepared beforehand by an instructor. Another example could be the use of GEX as a data acquisition module for process or environment monitoring.

The module should either be directly attached to the PC via a USB port, or controlled wirelessly (possibly powered by a battery or a solar cell). A wireless connection can find use in mobile robotic projects where the wired attachment isn't practical, or when used outdoors or in hardly accessible places.

## 2.3 Supported Hardware Interfaces

The project's scope is very broad and it's hardly possible to enumerate all the use cases. To achieve the greatest flexibility, it appears as a good strategy to divide the features into smaller functional blocks that can be used independently or combined as needed.

### 2.3.1 Direct Digital Input/Output

The most basic interaction with hardware is simply changing the logic levels of output pins, and reading input pins. With this feature alone it would be possible to analyze logic circuits, trigger some transient effect we want to observe using an oscilloscope, sense a button push, drive LED displays and more. Almost anything digital that doesn't require precise or fast timing[1] could be achieved by this simple function.

To make this feature more versatile, it should be possible to receive an asynchronous event on a pin state change, avoiding the need for polling loops in the control application.

### 2.3.2 Common Digital Buses

A popular way to attach peripheral devices to a microcontroller are hardware buses, the most well known of which are SPI (*Serial Peripheral Interface*), $I^2C$ (or IIC, *Inter-Integrated-Circuit*) and USART (*Universal Synchronous Asynchronous Receiver Transmitter*). There is a hardware support in most microcontrollers for those buses, removing the burden of precise timing from the firmware.

link to actual place

More information about those interfaces can be found in later chapters.

### 2.3.3 Specialized Buses

Some devices exist that use their own proprietary protocol, usually to reduce the number of data pins. An example of this group is the Dallas Semiconductor[2] 1-Wire bus, used by the popular DS18x20 digital thermometers. Another such protocol is used by some types of addressable LED strips.

A common characteristic of those buses is that they require precise timing and have no native hardware support in the microcontroller. We can't use the direct GPIO access here due to latencies and jitter; those protocols need a custom low level routine in the firmware that performs such "bit banging" more accurately.

### 2.3.4 Analog Input/Output

Microcontrollers typically include a 10-12 bit ADC (*Analog to Digital Converter*), often accompanied by a DAC (*Digital to Analog Converter*), its output counterpart. In the lack

[1] We're limited by the latencies of USB and the PC software.
[2] Acquired by Maxim Integrated in 2001

of a real DAC, the analog output, albeit with worse dynamic parameters, can be realized using a PWM signal (*Pulse Width Modulation*, pulse train) followed by a low-pass filter.

While we mainly focused on digital interfaces thus far, providing means of generating and capturing analog signals is also valuable. This capability makes it possible to read sensors with voltage output and it can substitute a simple oscilloscope when sampled periodically at a sufficient speed.

The analog output and input together can be used for automated characterization of electronic components, or for analog feedback regulation. Should the analog output be modulated, we could further use them to measure frequency-dependent characteristics, such as the frequency response of analog filters.

### 2.3.5 Frequency Measurement and Generation

Some sensors have a variable frequency or a pulse-width modulated (PWM) output. To capture those signals and convert them to a more useful digital value, we can use the Input Capture or External Clock function of a general purpose timer/counter in the used microcontroller. Those timers have a wide range of possible configurations and can be also used for pulse counting or PWM generation.

## 2.4 User Interface

USB will be the primary way of connecting the module to a PC. Thanks to USB's flexibility, it can present itself to the computer as any kind of device or even multiple devices at once.

The most straightforward method of interfacing the board is by passing binary messages in a fashion similar to USART. This can be done either using a "Virtual COM port" driver (the CDC/ACM class), or through a raw access to the corresponding USB endpoints. Using a raw access avoids potential problems with the operating system's driver interfering or not recognizing the device correctly; on the other hand, having GEX appear as a serial port makes it easier to integrate it into existing platforms that have a good serial port support (such as National Instruments LabWindows CVI or MATLAB).

The module should be reconfigurable. Given the settings are almost always going to be tied on the connected external hardware, it would be practical to have an option to store them permanently in the microcontroller's non-volatile memory.

We could load those settings into GEX using the serial interface, and indeed this should be implemented for its flexibility. The serial interface may be, in some form, also used for the wireless connection. However, having the power of USB at our disposal, we can make the board appear as a mass storage device and expose the configuration as text files. This approach, inspired by ARM mbed[3], avoids the need to create a configuration utility and it can work cross-platform thanks to using the same driver as a real removable disk. Further, we can expose any useful information (such as a README file with instructions, or a pin-out reference) using this virtual disk as separate files.

[3] A similar mechanism is used for flashing firmware images to mbed-enabled development kits

## 2.5 Form Factor and MCU Selection

It has been established that the designed device should be small and easily portable. It should support USB and wireless connection to a PC. Further, a plain UART interface should be offered in place of USB; this makes it easier to integrate GEX into other systems or use development kits fitted with a USB-UART converter.

The following protocols and features should be implemented as a minimum; Other functions may be added based on input from prototype testing in model situations.

- Low level hardware functions
  - I/O pin direct access
  - Input pin change reporting
  - Single pulse generation
  - PWM signal generation
  - Analog/Digital converter (analog input)
  - Digital/Analog converter (analog output)
  - Frequency measurement
  - Pulse measurement
- Common hardware interfaces
  - SPI
  - $I^2C$
  - UART/USART
- Low level support for proprietary buses
  - Dallas 1-Wire
  - NeoPixel (addressable LED strips)

Considering the requirements, availability, and cost, the STM32F072 microcontroller will be used for the first prototypes, with the expectation of later porting the project to more powerful or power-efficient devices, like the STM32L072, STM32F103, STM32F303, or other STM32 microcontrollers.

The STM32F072 is a Cortex M0 device with 128 KiB of flash memory, 16 KiB of RAM and running at up to of 48 MHz. It is equipped with a USB Full Speed peripheral block, a 12-bit ADC and DAC, a number of general purpose timers/counters, SPI, $I^2C$, and USART peripherals. It further supports a crystal-less USB operation, using the USB SOF packet for synchronization.

To utilize the time frame available for this work effectively, only the STM32F072 firmware will be initially developed, while making sure the planned future expansion will be as straightforward as possible.

info about form factor choices + some sketches

# Chapter 3

# Existing Solutions

The idea of making it easier to interact with low level hardware from a PC is not new. Several solutions to this problem have been developed over the past years, each with its own advantages and drawbacks. Some of the existing solutions will be presented in this chapter to give the reader some idea about the current possibilities.

## 3.1 Bus Pirate

pictures

Bus Pirate, developed by Ian Lesnet at Dangerous Prototypes and manufactured by Seeed Studio, is a USB-attached device providing access to hardware interfaces like SPI, $I^2C$, USART and 1-Wire (those will be described later ), as well as frequency measurement and direct pin access.

link

link

link to actual place

The board aims to make it easy for the users to familiarize themselves with new chips and modules; it also provides a range of programming interfaces for flashing microcontroller firmwares and memories. It communicates with the PC using a FTDI USB-serial bridge

Bus Pirate is open source and in scope it is similar to what we want to achieve here. It can be scripted and controlled from languages like Python or Perl, connects to USB and provides a wide selection of hardware interfaces.

The board is based on a PIC16 microcontroller running at 32 MHz. Its analog/digital converter (ADC) only has a resolution of 10 bits (1024 levels). There is no digital/analog converter (DAC) available on the chip, making applications that require a varied output voltage more difficult. Another limitation of the board is its low number of GPIO pins, which may be insufficient for certain applications, such as multi-channel sampling, parallel interfaces, or connecting multiple different devices at once. The Bus Pirate, at the time of writing, can be purchased for a price similar to some Raspberry Pi models.

## 3.2 Raspberry Pi

link, pictures

Another device worth mentioning, albeit of a different kind, is the Raspberry Pi. All models of the Raspberry Pi include a GPIO header which can be directly controlled by user applications. The pins broken out to this header can be used as general purpose I/O and some have alternate functions such as SPI or $I^2C$.

The responsibility of controlling the attached external hardware lies on the user application, which also commonly provides the user interface, which greatly simplifies the development process. The control application can be written in almost any programming language. Python is a popular choice thanks to its simplicity, but it's by no means the only way to interact with the pins.

The Raspberry Pi is commonly used in primary schools as a low-cost PC alternative that encourage students' interest in electronics and science. The board is, further, often built into more permanent projects that make use of its powerful processor, such as camera traps for wildlife observations.

The Raspberry Pi could be used for the quick evaluations or experiments we want to perform with GEX, however they would either have to be performed directly on the mini-computer itself (with attached monitor and keyboard), or use some form of remote access.

## 3.3 The Firmata protocol

links

Move this elsewhere

Firmata is a serial communication protocol based on MIDI (*Musical Instrument Digital Interface*) for passing data to and from embedded microcontrollers. MIDI is primarily used for attaching electronic musical instruments, such as synthesizers, keyboards, mixers etc., to each other or to a PC.

Firmata was designed for use with the Arduino firmware to allow easy construction of user programs (called *sketches* in the Arduino environment) that communicate with a client application running on the PC without having to worry about technical details.

citation

Implementing the Firmata protocol in a universal hardware interfacing module would make it possible to use existing Firmata client libraries. However, it is constricted by the limitations of the encompassing MIDI protocol and offers little flexibility.

## 3.4 Professional DAQ modules

A range of professional tools that would fulfill our needs exist on the market, however their common property is a high price. This makes them inaccessible for users with a limited budget, such as hobbyists or students who would like to keep such a device for personal use. An example falling into this category is the National Instruments "I²C/SPI Interface

Device", which also includes several GPIO lines, or some of the Total Phase I²C/SPI gadgets which sell for about $300 a piece.

The performance GEX can provide will certainly be far inferior to those professional tools, however this drawback is balanced by its greater flexibility and for most applications it should be perfectly sufficient, and at a fraction of the cost.

http://www.ni.com/en-gb/shop/select/i2c-spi-interface-device

pictures

# Chapter 4

# Universal Serial Bus

This chapter presents an overview of the *Universal Serial Bus* (USB) *Full Speed* interface, with focus on the features used in the GEX firmware. USB is a versatile but complex interface, thus explaining it in its entirety is beyond the scope of this text. References to external materials which explain the protocol in greater detail will be provided for the interested reader.

add those refs

## 4.1 Basic Principles and Terminology

USB is a hierarchical bus with a single master (*host*) and multiple slave devices. A USB device that provides functionality to the host is called a *function*. Communication between the host and a function is organized into virtual channels called *pipes*. Each pipe is identified by an *endpoint* number.

Endpoints can be either unidirectional or bidirectional; the direction from the host to a function is called *OUT*, the other direction (function the host) is called IN. A bidirectional endpoint is technically composed of a IN and OUT endpoint with the same number. All transactions (both IN and OUT) are initiated by the host; functions have to wait for their turn. Endpoint 0 is bidirectional, always enabled, and serves as a *control endpoint*. The host uses the control endpoint to read information about the device and configure it as needed.

There are four types of transfers: control, bulk, isochronous, and interrupt. Each endpoint is configured for a fixed transfer type.

- *Control* - initial configuration after device plug-in; also used for other aplication-specific control messages that can affect other pipes.
- *Bulk* - used for burst transfers of large messages, commonly e.g. for mass storage devices
- *Isochronous* - streaming with guaranteed low latency; designed for audio or video streams where some data loss is preferred over stuttering
- *Interrupt* - low latency short messages, used for human interface devices like mice and keyboards

The endpoint transfer type and other characteristics, together with other information about the device, such as the serial number, are defined in a *descriptor table*. This is a tree-like binary structure defined in the function's memory. The descriptor table is loaded by the host to learn about the used endpoints and to attach the right driver to it.

The function's endpoints are grouped into *interfaces*. An interface describes a logical connection of endpoints, such as the reception and transmission endpoint that belong together. An interface is assigned a *class* defining how it should be used. Standard classes are defined by the USB specification to provide a uniform way of interfacing devices of the same type, such as human-interface devices (mice, keyboards, gamepads) or mass storage devices. The use of standard classes makes it possible to re-use the same driver software for devices from different manufacturers. The class used for the GEX's "virtual COM port" function was originally meant for telephone modems, a common way of connecting to the Internet at the time the first versions of USB were developed. A device using this class will show as `/dev/ttyACM0` on Linux and as a COM port on Windows, provided the system supports it natively or the right driver is installed.

```
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass          239 Miscellaneous Device
  bDeviceSubClass         2
  bDeviceProtocol         1 Interface Association
  bMaxPacketSize0        64
  idVendor           0x0483 STMicroelectronics
  idProduct          0x572a
  bcdDevice            0.01
  iManufacturer           1 MightyPork
  iProduct                2 GEX
  iSerial                 3 0029002F-42365711-32353530
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           98
    bNumInterfaces          3
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0x80
      (Bus Powered)
    MaxPower              500mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           2
      bInterfaceClass         8 Mass Storage
      bInterfaceSubClass      6 SCSI
      bInterfaceProtocol     80 Bulk-Only
      iInterface              4 Settings VFS
      Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x81  EP 1 IN
        bmAttributes            2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0040  1x 64 bytes
        bInterval               0
      Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x01  EP 1 OUT
        bmAttributes            2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0040  1x 64 bytes
        bInterval               0
    Interface Association:
      bLength                 8
      bDescriptorType        11
      bFirstInterface         1
      bInterfaceCount         2
      bFunctionClass          2 Communications
      bFunctionSubClass       2 Abstract (modem)
      bFunctionProtocol       1 AT-commands (v.25ter)
      iFunction               5 Virtual Comport ACM
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        1
  bAlternateSetting       0
  bNumEndpoints           1
  bInterfaceClass         2 Communications
  bInterfaceSubClass      2 Abstract (modem)
  bInterfaceProtocol      1 AT-commands (v.25ter)
  iInterface              5 Virtual Comport ACM
  CDC Header:
    bcdCDC               1.10
  CDC Call Management:
    bmCapabilities       0x00
    bDataInterface          2
  CDC ACM:
    bmCapabilities       0x06
      sends break
      line coding and serial state
  CDC Union:
    bMasterInterface        1
    bSlaveInterface         2
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x83  EP 3 IN
    bmAttributes            3
      Transfer Type            Interrupt
      Synch Type               None
      Usage Type               Data
    wMaxPacketSize     0x0008  1x 8 bytes
    bInterval             255
Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        2
  bAlternateSetting       0
  bNumEndpoints           2
  bInterfaceClass        10 CDC Data
  bInterfaceSubClass      0
  bInterfaceProtocol      0
  iInterface              6 Virtual Comport CDC
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x02  EP 2 OUT
    bmAttributes            2
      Transfer Type            Bulk
      Synch Type               None
      Usage Type               Data
    wMaxPacketSize     0x0040  1x 64 bytes
    bInterval               0
  Endpoint Descriptor:
    bLength                 7
    bDescriptorType         5
    bEndpointAddress     0x82  EP 2 IN
    bmAttributes            2
      Transfer Type            Bulk
      Synch Type               None
      Usage Type               Data
    wMaxPacketSize     0x0040  1x 64 bytes
    bInterval               0
```

**Figure 4.1:** USB descriptors of a GEX prototype, read using `lsusb -vd vid:pid`

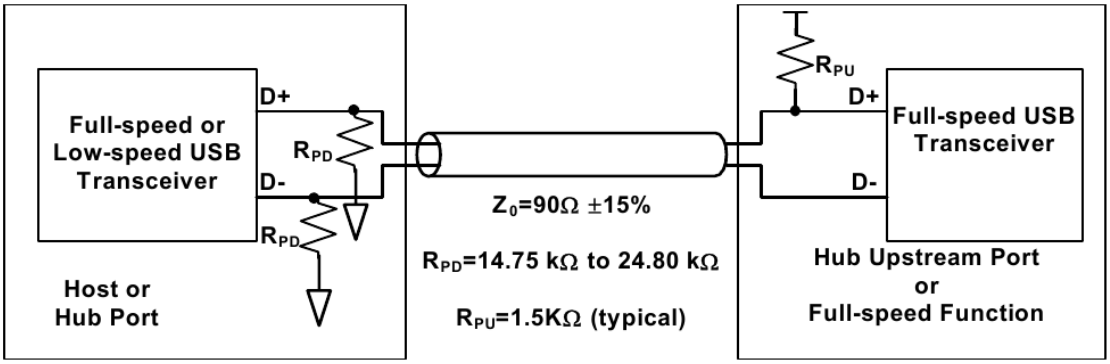



**Figure 4.3:** Pull-up and pull-down resistors of a Full Speed function, as prescribed by the USB specification rev. 2.0

## 4.2 USB Physical Layer

USB uses differential signaling with NRZI encoding (*Non Return to Zero Inverted*, fig. 4.2) and bit stuffing. The encoding, together with frame formatting, checksum verification, retransmission, and other low level aspects of the USB connection are entirely handled by the USB block in the microcontroller's silicon; normally we do not need to worry about those details. What needs more attention are the electrical characteristics of the bus, which need to be understood correctly for a successful schematic and PCB design.

The USB cable contains 4 conductors:

- $V_{BUS}$ (+5 V)
- D+
- D–
- Ground

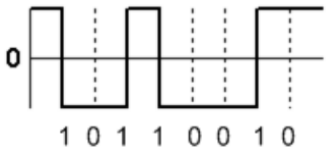



**Figure 4.2:** NRZI encoding example

The data lines, D+ and D–, are also commonly labeled DP and DM. This differential pair should be routed in parallel and kept at approximately the same length.

USB revisions are, where possible, backwards compatible, often even keeping the same connector shape. The bus speed is negotiated by the device using a 1.5 kΩ pull-up resistor (to 3.3 V) on one of the data lines: for Full Speed, D+ is pulled high (fig. ), for Low Speed the pull-up is on the D– line. The polarity of the differential signals is inverted depending on the used speed. Some microcontrollers integrate the correct pull-up resistor inside the USB block, removing the need for an external resistor.

When a function needs the host to re-enumerate it, that is, reload the descriptors and re-attach the correct drivers, it can momentarily remove the pull-up resistor, which the host will interpret as if the device was plugged out. In the case of an internal pull-up, this can be done by flipping a control bit. An external resistor could be connected through a transistor to facilitate re-enumeration, or it might be driven directly by a GPIO pin.

The $V_{BUS}$ line supplies power to the function in the case of a *bus-powered* device. *Self-powered* devices can leave this pin unconnected and instead use an external power supply. The maximal current drawn from the $V_{BUS}$ line is configured using a descriptor and should not be exceeded.

## 4.3 USB Classes

This section explains the function of the Mass Storage class and the CDC/ACM class that find use in the GEX firmware. A list of all standard classes with a more detailed explanation can be found in .

link to the ref manual

### 4.3.1 Mass Storage Class

The Mass Storage class (MSC) is natively supported by all modern operating systems (MS Windows, MacOS, GNU/Linux, FreeBSD etc.) and finds use in thumb drives, external disks, memory card readers and other storage devices.

The MSC specification defines multiple *transport protocols* that can be selected using the descriptors. For it's simplicity, the *Bulk Only Transport* (BOT) will be used. BOT uses two bulk endpoints for reading and writing blocks of data and for the exchange of control commands and status messages. For the device to be recognized by the operating system, it must also implement a *command set*. Most mass storage devices use the *SCSI Transparent command set* [1]. The defined commands let the host read information about the attached storage, such as its capacity, and check for media presence and readiness to write or detach.

The MSC class together with the SCSI command set are rather complicated, thankfully a driver library is provided by ST Microelectronics that can be used as given, or customized. The library also includes a CDC/ACM implementation.

In order to emulate a mass storage device without having a physical storage medium, we need to generate and parse the filesystem on-the-fly, as the host OS tries to access it. This will be discussed in chapter ??.

chapter num

### 4.3.2 CDC/ACM Class

Historically meant for modem communication, this class is now the de facto standard way of making USB devices appear as serial ports on the host OS. The CDC (*Communication Device Class*) class uses three endpoints: bulk IN and OUT, and an interrupt endpoint.

The interrupt endpoint is used for commands and notifications about the modem line, while the bulk endpoints are used for useful data. ACM stands for *Abstract Control Model* and it's a CDC's subclass that defines the control messages format. Since we don't use any physical serial port in this implementation and the line is virtual both on the PC and in the end device, the interrupt endpoint can mostly be ignored.

[1] To confirm this assertion, the descriptors of five thumb drives and an external hard disk were analyzed using `lsusb`. All but one device used the SCSI command set, one (incidentally the oldest) used *SFF-8070i*. A list of possible different command sets can be found in TODO (usb spec overview)

An interesting property of this class is that the bulk endpoints transport raw data without any wrapping frames. By changing the device class in the descriptor table to 255 (*Vendor Specific Class*), we can retain the messaging functionality of the designated endpoints and access the device directly using e.g. libUSB, whereas the OS will ignore the device and won't try to attach the serial port driver, which would otherwise interfere. The same trick can be used to hide the mass storage class, when not needed.

### 4.3.3 Interface Association: Composite Class

Since it's creation, the USB specification expected that each function will have only one interface enabled at a time. After it became apparent that there is a need for having multiple unrelated interfaces work in parallel, a workaround called the *Interface Association Descriptor* (IAD) was devised. IAD is an entry in the descriptor table that defines which interfaces belong together and should be handled by the same driver.

To use the IAD, the function's class must be set to 239 (EFh), subclass 2, protocol 1 (in the top level descriptor), so the OS knows to look for the presence of IADs before binding drivers to any interfaces.

In GEX, the IAD is used to tie together the CDC and ACM interfaces while leaving out the MSC interface which should be handled by a different driver. To make this work, a new *composite class* had to be created as a wrapper for the library-provided MSC and CDC/ACM implementations.

examples of descriptors?

# Chapter 5

# FreeRTOS