BACHELOR PAPER

Thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Electronics and Business Distance Study (BEW-DL)

Wireless Button for FABI

By: Friedrich König, MSc

Student Number: 2210255002

Supervisor 1: Alijia Sabic, MSc

Munich, 13.02.2025

Declaration of Authenticity

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz/ Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and according to the rules currently applicable at the UAS Technikum Wien and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.”

|  |  |  |
| --- | --- | --- |
| Munich, |  |  |
| Place, Date |  | Digital Signature |

Kurzfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.

Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla.

Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien. Donec ut est in lectus consequat consequat.

Abstract

**Schlagwörter:** Puck.js, Raspberry Pi Pico W, FABI, Assistive Technologies

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.

Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

Fusce aliquet pede non pede. Suspendisse dapibus lorem pellentesque magna. Integer nulla.

Donec blandit feugiat ligula. Donec hendrerit, felis et imperdiet euismod, purus ipsum pretium metus, in lacinia nulla nisl eget sapien. Donec ut est in lectus consequat consequat.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

Aenean nec lorem. In porttitor. Donec laoreet nonummy augue.

Table of Contents

[1 Introduction 7](#_Toc190535561)

[2 State of the Art 8](#_Toc190535562)

[2.1 FABI 8](#_Toc190535563)

[2.2 Hardware 8](#_Toc190535564)

[2.2.1 Raspberry Pi Pico W 8](#_Toc190535565)

[2.2.2 Arduino Nano RP2040 Connect 9](#_Toc190535566)

[2.2.3 Raspberry Pi Pico 2 W 9](#_Toc190535567)

[2.2.4 Puck.js 9](#_Toc190535568)

[2.3 Software 10](#_Toc190535569)

[2.3.1 Earle Philhower Core 10](#_Toc190535570)

[2.3.2 BLE 10](#_Toc190535571)

[2.3.3 WIFI 10](#_Toc190535572)

[3 Requirements 11](#_Toc190535573)

[3.1 Required Hardware 11](#_Toc190535574)

[3.1.1 Raspberry Pi Pico W 11](#_Toc190535575)

[3.1.2 Puck.js 11](#_Toc190535576)

[3.2 Required Software 11](#_Toc190535577)

[3.2.1 C++ and Arduino IDE 11](#_Toc190535578)

[3.2.2 JavaScript and Espruino IDE 12](#_Toc190535579)

[3.3 Functional Features 12](#_Toc190535580)

[3.3.1 User Interface 12](#_Toc190535581)

[3.3.2 Connection of Puck.js 12](#_Toc190535582)

[3.3.3 Button Presses 13](#_Toc190535583)

[4 Design 14](#_Toc190535584)

[4.1.1 Hardware 14](#_Toc190535585)

[4.1.1.1 Wi-Fi Phase 14](#_Toc190535586)

[4.1.1.2 BLE Phase 14](#_Toc190535587)

[4.2 Software 16](#_Toc190535588)

[5 Implementation 17](#_Toc190535589)

[5.1 Raspberry Pico W 18](#_Toc190535590)

[5.1.1 Setup 18](#_Toc190535591)

[5.1.2 Wifi User Interface 18](#_Toc190535592)

[5.1.3 BLE 18](#_Toc190535593)

[5.1.4 Button Mapping 18](#_Toc190535594)

[5.1.5 Key Presses 18](#_Toc190535595)

[5.1.6 Main Loop 18](#_Toc190535596)

[5.2 Arduino Nano RP2040 Connect 18](#_Toc190535597)

[5.2.1 Setup 18](#_Toc190535598)

[5.2.2 Wifi User Interface 18](#_Toc190535599)

[5.2.3 BLE 18](#_Toc190535600)

[5.2.4 Button Mapping 18](#_Toc190535601)

[5.2.5 Key Presses 18](#_Toc190535602)

[5.2.6 Main Loop 18](#_Toc190535603)

[6 Evaluation 19](#_Toc190535604)

[6.1 Test 19](#_Toc190535605)

[6.2 Limitations 19](#_Toc190535606)

[6.2.1 One Button vs Multiple Buttons 19](#_Toc190535607)

[6.2.2 Wifi vs BLE 19](#_Toc190535608)

[6.2.3 USB HID vs BLE Periperal 19](#_Toc190535609)

[7 Results and Discussion 19](#_Toc190535610)

[7.1 Comparing the requirements 19](#_Toc190535611)

[7.1.1 Required Hardware 19](#_Toc190535612)

[7.1.2 User Interface 19](#_Toc190535613)

[7.1.3 Connection of Puck.js 19](#_Toc190535614)

[7.1.4 Button Presses 19](#_Toc190535615)

[7.2 Discussing Future Work 19](#_Toc190535616)

[8 Conclusion 19](#_Toc190535617)

[Bibliography 20](#_Toc190535618)

[List of Figures 23](#_Toc190535619)

[List of Tables 24](#_Toc190535620)

[List of Abbreviations 25](#_Toc190535621)

[Documentation table of AI-based tools 26](#_Toc190535622)

[Appendix A: Code for Puck.js 27](#_Toc190535623)

[Appendix B: Code for Raspberry Pico W 28](#_Toc190535624)

[Appendix C: Code for Arduino Nano RP2040 Connect 29](#_Toc190535625)

[Appendix D: Code for Puck.js (for the Connection with Arduino Nano RP2040 Connect) 35](#_Toc190535626)

# Introduction

Below there is a cross-reference to Table 1. The table format shown here serves as an example only. Tables may be formatted individually.

Tabelle 1: Schedule for “Applied Mathematics” (Quelle).

|  |  |  |
| --- | --- | --- |
| **Date** | **Subject** | **Room** |
| **20. 08. 2008** | Graph Theory | HS 3.13 |
| **01. 10. 2008** | Biomathematics | HS 1.05 |

This is a cross-reference to Equation (1):

|  |  |
| --- | --- |
|  | (1) |

Bibliography references should be automated especially when there is a long list of books. This is a sample reference to The style of citation and the Bibliography format used here is one of several possible ways, depending on the discipline and the functionality of the word processor.

# State of the Art

Before implementing any ideas, it is important to gain an understanding of the current state of the art. This chapter will provide an overview of FABI and describe the current state of the art of the used hard- and software.

## FABI

According to the AsTeRICS Foundation, “[t]he FABI (Flexible Assistive Button Interface) is an interface box which allows the connection of up to 9 momentary switches (buttons) or self-made electric contacts to a computer, tablet or smartphone” [1]. The user may specify the actions of each button in a “web-based config manager” [2, 3]. Buttons are either “momentary switches or self-made electrical contacts” [2] or “sip-puff sensor[s]” [2], that have to be connected via wire to the FABI system. The FABI itself may either communicate via USB to the target computer, or utilize a Bluetooth connection, if the user attaches a Bluetooth module to the FABI. The FABI system is an open-source project, wherefore it is possible to build a device oneself, using the provided building guides and user manual [4-7]. The main processing unit of the FABI system is an Arduino Pro Micro microcontroller, that is based on the ATmega32U4 chip [8], according to SparkFun Electronics [9], a vendor of electronic components, whose Pro Micro board is recommended by the AsTeRICS Foundation in the construction of a FABI system The Pro Micro board has been chosen, as it supports the USB HID protocol, meaning the device can act as an interface device, such as a mouse or a keyboard, when connected via an USB port [5]. It features 12 digital I/O pins, and is “[s]upported unter Arduino IDE v1.0.1” [9]. However, the Pro Micro board does not feature an onboard Bluetooth or Wifi chip, wherefore the FABI currently relies on an external Bluetooth module for wireless connections. To make the external Bluetooth module obsolete, a different board will be chosen for this project. Hence, the following chapter explores the available options for alternative boards.

## Hardware

An alternative to the Pro Micro board has to be compatible with Arduino IDE, support the USB HID protocol and needs to have digital I/O pins. One such candidate is the RP2040 chip developed and produced by Raspberry Pi Ltd [10], that is the basis of microcontrollers such as the Raspberry Pi Pico W or the Arduino Nano RP2040 Connect [11].

### Raspberry Pi Pico W

The RP2040 present on the Raspberry Pi Pico W board features a dual-core ARM Cortex-M0+ processor running at up to 133 MHz [12], significantly outperforming the ATmega32U4, which has a single-core 8-bit AVR processor operating at 16 MHz [8]. This increased clock speed and 32-bit architecture allow the RP2040 to handle more computationally intensive tasks efficiently. Both chips support native USB support without needing any additional hardware [8, 10]. While the Pro Micro costs $12.50 from the vendor SparkFun [9] at the time of writing, the Raspberry Pi Pico W is significantly cheaper priced at 6.70€ ($7.04) from the official vendor BerryBase GmbH [13]. The key advantage of the Raspberry Pi Pico W over the Arduino Pro Micro is the inclusion of the Infineon CYW43439 [14]. This chip supports “Wi-Fi 4 (802.11n), Single-band (2.4 GHz)“ [14], as well as Bluetooth 5.4 [14], which both are necessary for the present project.

### Arduino Nano RP2040 Connect

The Arduino Nano RP2040 Connect features the same RP2040 chip as the aforementioned Raspberry Pi Pico W. The main differences are the replacement of the CYW43439 chip by the u-blox Nina W102 module, which also provides single-band 2.5GHz Wi-Fi operation and supports Bluetooth 4.2 [11, 15]. It is currently priced at 30.74€ ($31.35), meaning it is significantly more expensive than the Raspberry Pi Pico W. It is more advanced technically speaking, as it features various additional sensors, such as a gyroscope, microphone or a cryptographic co-processor, that are not relevant for this project [11].

### Raspberry Pi Pico 2 W

In August of 2024 Raspberry Pi Ltd announced a successor to the Raspberry Pi Pico W, the Raspberry Pico 2 W, that is based on the enhanced RP2350 chip [16-18]. While the new board is more powerful than its predecessor, featuring a dual core ARM Cortex-M33 processor running at up to 150MHz [18], the same CYW43439 is present on the board, leading to no changes in wireless connectivity. The price of the Raspberry Pi Pico 2 W is given at 7.90€ ($8.30) by the official vendor BerryBase GmbH [19].

### Puck.js

The Puck.js is a compact, energy-efficient Bluetooth Low Energy (BLE) development board designed for wireless applications. It is based on the Nordic Semiconductor nRF52832 chipset [20], featuring an ARM Cortex-M4 processor and integrated radio communication capabilities [21]. The onboard nRF52832 chipset supports BLE 5.0, allowing for reliable data transmission and interaction with other BLE-enabled devices such as smartphones, computers, and embedded systems [20-22]. The Puck.js board is equipped with various built-in sensors and features, most importantly for this project a push-button for user interaction [22]. The Puck.js is designed for long battery life, operating on a standard coin-cell battery (CR2032) for extended periods [22]. Unlike many microcontroller boards that require compiled firmware development, Puck.js can be programmed using JavaScript through the Espruino interpreter, which makes development highly accessible, as code can be written and executed directly from a web browser via Web Bluetooth [21]. It currently retails for £32.40 ($40.88) in the official Espruino webshop [22].

## Software

After delving into the state of the art hardware, this chapter shall give an overview of the state of the art of the used software.

### Earle Philhower Core

The Earle Philhower Core is an Arduino-compatible core designed specifically for the RP2040 and RP2350 families of microcontrollers [23]. By bridging the gap between the robust features of the RP2040 & RP2350 and the ease-of-use provided by the Arduino ecosystem, this core allows developers to write, compile, and deploy code using the Arduino IDE [23]

The core is designed to be compatible with a wide range of existing Arduino libraries [24]. This compatibility minimizes the need for extensive code rewrites and allows for the quick migration of projects from other Arduino boards, such as the Arduino Pro Micro, that is used in the FABI sytem. The Earle Philhower Core provides many SDKs, that are needed in this project, such as an SDK for BLE and Wi-Fi, that are used in this project [24]. Additionally, Philhower [23] provides many detailed examples on how to use the SDKs on his GitHub page.

### BLE

### WIFI

# Requirements

This chapter will give an overview of the requirements of this project, i.e. which features need to be implemented. First, the hardware on which the project must run will be stated. Afterwards, the required user features will be discussed.

## Required Hardware

As this project features BLE, the required hardware features both an BLE central and a BLE peripheral device. Both will be stated in the next parts.

### Raspberry Pi Pico W

To provide an update to FABI, the main processing board and the BLE central devide has to be the Raspberry Pi Pico W, introduced in Chapter 2.2.1. While the Raspberry Pico 2 W is an updated version, it is rather more expensive and does not provide any additional features, that are required for this project. While not explicitly testing for compatibility, it is assumed, that the project will work on the Raspberry Pi Pico 2 W as well, as the Philhower Core explicitly also covers the board, and the wireless communication chip has not changed from the predecessor.

### Puck.js

The BLE peripheral device, that recognizes the user induced button presses has to be the Espruino Puck.js, which was introduced in Chapter 2.2.4. As stated before, that device fulfills all needs of this project, as it features an onboard push-button and is able to establish a BLE connection.

## Required Software

After having introduced the required hardware, next the required software will be introduced. This includes both the programming language, as well as the IDEs.

### C++ and Arduino IDE

The software, that will run on the Raspberry Pi Pico W, has to be programmed in C++ using the Arduino IDE, analogous to the original FABI project. The required core is the Philhower Core, that has been introduced in Chapter 2.3.1. This implies, that the included libraries in the Philhower Core have to be used as well.

### JavaScript and Espruino IDE

The Puck.js has to be programmed in JavaScript using the Espruino IDE. The Espruino IDE is a web-based IDE, that uses Web BLE to connect to the Puck.js [21, 25, 26]. It is therefore not needed to connect the Puck.js via cable to a computer nor the BLE Central device.

## Functional Features

This section outlines the key functional features that underpin the interactive nature of the system. These features include the user interface, the connection mechanism for the Puck.js board, and the recognition of button presses and button actions. Each of these elements contributes to a seamless and responsive user experience.

### User Interface

The user interface (UI) must designed to be intuitive and accessible, ensuring that users can easily interact with the system. Its core elements include:

* **Intuitive Layout:** The UI msut be organized in a clear and structured manner, presenting essential information and controls prominently. This design approach minimizes user confusion and enhances overall usability.
* **Real-Time Feedback:** Dynamic visual indicators provide immediate feedback on settings such as the number of Puck.js devices to be connected and the button actions. This real-time interaction helps users understand the current operation and any changes that occur.
* **Customizable Elements:** The interface can be adapted to suit individual user needs. Whether through configurable controls or adaptable display options, the UI offers flexibility in how information is presented and interacted with.

### Connection of Puck.js

A critical functional aspect of the system is its ability to establish a reliable connection with at least one Puck.js board. The connection process involves:

* **BLE:** The Puck.js is connected via BLE to the BLE Central device, i.e. a Raspberry Pi Pico W.
* **Automated Discovery and Pairing:** The system scans for nearby Puck.js devices and initiates a secure pairing process automatically. This ensures a quick and efficient connection without requiring manual intervention.
* **Connection Monitoring:** Once connected, the system continuously monitors the connection status. Any disconnects have to restart the discovery and pairing process.

### Button Presses

User interaction through physical button inputs on the Puck.js are the core component of the system’s functionality. The system handles these interactions by:

* **Accurate Button Detection:** The software distinguishes between various types of button presses—such as single taps, double taps, and long presses—to ensure that each interaction is correctly interpreted.
* **Action Mapping:** Each type of button press is linked to a specific function or command, that the user configured in the UI.
* **Feedback Mechanisms:** Upon detecting a button action, the system provides immediate feedback (visual or auditory) to confirm that the input has been received and processed. This helps users feel confident in their interactions.
* **Robust Debouncing:** To prevent unintended multiple triggers from a single press, a debouncing mechanism has to be implemented. This ensures that only deliberate and distinct button actions are registered, improving overall system reliability.

# Design

This chapter explains the design of the project. First, the hardware design and necessary connections will be explained. In a second step, the software design is shown.

### Hardware

As the CYW43439 chip is limited by providing “[s]imultaneous BT/WLAN reception with a single antenna” [27, p. 6], that means that hosting a web server while acting as a BLE Central device as the same time is not possible. Therefore, the design needs to be splitted into two phases. The Wi-Fi phase and the BLE phase.

#### Wi-Fi Phase

During the Wi-Fi phase, the Raspberry Pi Pico W needs to be connected via USB cable to user device, such as a PC or laptop. In this phase, the Raspberry Pi Pico W does not utilize the USB 1.1 protocol yet, as it only needs power from the user device. During the Wi-Fi phase, the Raspberry Pi Pico W hosts an Access Point (AP), to which the user device has to connect. A simplified schematic of the hardware connections is shown in figure 1.

Ein Bild, das Text, Elektronisches Bauteil, Elektronik, Elektrisches Bauelement enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 1: Hardware connections at the Wi-Fi phase (Source: Own depiction. Raspberry Pi Pico W: Own photograph. USB 1.1 logo: [28]. Wi-Fi 4 logo: [29])

#### BLE Phase

During the BLE phase, the Raspberry Pi Pico W has to stay connected to the user device via USB cable. It connects to at least one Puck.js via BLE. In the BLE phase the Raspberry Pi Pico W utilizes the USB 1.1 protocol, specifically the HID protocol [30], to send the respective button actions to the user device. A simplified schematic of the hardware connections is shown in figure 2.

Ein Bild, das Text, Screenshot, Schrift, Flashspeicher enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 2: Hardware connections at the BLE phase (Source: Own depiction. Puck.js: Own photograph. BLE logo: [31]. Raspberry Pi Pico W: Own photograph. USB 1.1 logo: [28])

## Software

Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 3: Simplified Flowchart (Source: Own depiction)

# Implementation

## Raspberry Pico W

### Setup

### Wifi User Interface

### BLE

### Button Mapping

### Key Presses

### Main Loop

## Arduino Nano RP2040 Connect

### Setup

### Wifi User Interface

### BLE

### Button Mapping

### Key Presses

### Main Loop

# Evaluation

## Test

## Limitations

### One Button vs Multiple Buttons

Multiple bottons are supported on the Arduino, but not on the Raspi

### Wifi vs BLE

Chip does not support both at the same time.

### USB HID vs BLE Periperal

If the Puck.js shall be connected via BLE, then the computer cannot be connected via BLE, as the Raspberry can be either LE Central or LE Peripheral

# Results and Discussion

## Comparing the requirements

### Required Hardware

### User Interface

### Connection of Puck.js

### Button Presses

## Discussing Future Work

# Conclusion

Bibliography

[1] AsTeRICS Foundation, “FABI - Flexible Assistive Button Interface.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.asterics-foundation.org/projects/fabi/

[2] AsTeRICS Foundation, “asterics/FABI: The repository for FABI (Flexible Assistive Button Interface).” Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/asterics/FABI

[3] AsTeRICS Foundation, “FABI Configuration.” Accessed: Feb. 14, 2025. [Online]. Available: https://fabi.asterics.eu/

[4] AsTeRICS Foundation, “FABI - User Manual.” Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/asterics/FABI/blob/master/Documentation/UserManual/Markdown/Fabi%20User%20Manual.md

[5] AsTeRICS Foundation, “FABI the Flexible Assistive Button Interface ‘Do-It-Yourself’ - Building Guide.” Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/asterics/FABI/blob/master/Documentation/ConstructionManual/SelfmadeBox/FABI\_Construction\_SelfmadeBox.pdf

[6] AsTeRICS Foundation, “FABI - Flexible Assistive Button Interface Construction Manual.” Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/asterics/FABI/blob/master/Documentation/ConstructionManual/3D-printedBox/en/FABI\_ConstructionManual.pdf

[7] AsTeRICS Foundation, “FABI Construction Manual.” Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/asterics/FABI/blob/master/Documentation/ConstructionManual/PCB-Version/FABIManual.pdf

[8] Atmel Corporation, “ATmega32U4 Datasheet,” 2010. Accessed: Feb. 14, 2025. [Online]. Available: https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/ATMega32U4.pdf

[9] SparkFun Electronics, “Pro Micro - 5V/16MHz.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.sparkfun.com/pro-micro-5v-16mhz.html

[10] Raspberry Pi Ltd, “RP2040 Datasheet,” Oct. 14, 2024. Accessed: Feb. 14, 2025. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[11] Arduino, “Arduino Nano RP2040 Connect Datasheet.”

[12] Raspberry Pi Ltd, “Raspberry Pi Pico W Datasheet,” Oct. 15, 2024. Accessed: Feb. 14, 2025. [Online]. Available: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

[13] BerryBase GmbH, “Raspberry Pi Pico W, RP2040 + WLAN Mikrocontroller-Board.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.berrybase.de/raspberry-pi-pico-w-rp2040-wlan-mikrocontroller-board

[14] Infineon Technologies AG, “CYW43439.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.infineon.com/cms/en/product/wireless-connectivity/airoc-wi-fi-plus-bluetooth-combos/wi-fi-4-802.11n/cyw43439/

[15] u-blox, “NINA-W10 series Datasheet,” Dec. 19, 2022. Accessed: Feb. 14, 2025. [Online]. Available: https://content.u-blox.com/sites/default/files/NINA-W10\_DataSheet\_UBX-17065507.pdf

[16] C. Windeck, “Raspberry Pi Pico 2 W: Mikrocontroller-Board mit WLAN vorgestellt | heise online.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.heise.de/news/Raspberry-Pi-Pico-2-W-Mikrocontroller-Board-mit-WLAN-vorgestellt-10082909.html

[17] Raspberry Pi Ltd, “Raspberry Pi Pico 2 W Datasheet,” Nov. 26, 2024. Accessed: Feb. 14, 2025. [Online]. Available: https://datasheets.raspberrypi.com/picow/pico-2-w-datasheet.pdf

[18] Raspberry Pi Ltd, “RP2350 Datasheet,” Oct. 16, 2024. Accessed: Feb. 14, 2025. [Online]. Available: https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf

[19] BerryBase GmbH, “Raspberry Pi Pico 2W, RP2350 + WLAN + Bluetooth Mikrocontroller-Board.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.berrybase.de/raspberry-pi-pico-2w-rp2350-wlan-bluetooth-mikrocontroller-board?utm\_source=google&utm\_medium=cpc&gad\_source=1&gbraid=0AAAAADSQJK5VOj\_pMDe3XJJt-wudJV3C0&gclid=CjwKCAiA8Lu9BhA8EiwAag16b6RfywPgdTiNy7ovvn2tIR\_ZSmJGJOEIz2sBuxc0K8q9uFuRaz1SjxoCGOkQAvD\_BwE

[20] Nordic Semiconductor, “nRF52832 Datasheet”.

[21] Espruino, “Puck.js.” Accessed: Feb. 14, 2025. [Online]. Available: https://www.espruino.com/Puck.js

[22] Espruino, “Espruino Puck.js v2.1.” Accessed: Feb. 14, 2025. [Online]. Available: https://shop.espruino.com/puckjs

[23] E. F. Philhower, “earlephilhower/arduino-pico: Raspberry Pi Pico Arduino core, for all RP2040 and RP2350 boards.” Accessed: Feb. 14, 2025. [Online]. Available: https://github.com/earlephilhower/arduino-pico

[24] E. F. Philhower, “Arduino-Pico — Arduino-Pico 4.4.3 documentation.” Accessed: Feb. 14, 2025. [Online]. Available: https://arduino-pico.readthedocs.io/en/latest/

[25] Espruino, “Espruino Web IDE.” Accessed: Feb. 15, 2025. [Online]. Available: https://www.espruino.com/ide/

[26] J. Dungen, “Puck.js Development Guide.” Accessed: Feb. 15, 2025. [Online]. Available: https://reelyactive.github.io/diy/puckjs-dev/

[27] Infineon Technologies AG, “CYW43439 Datasheet,” Mar. 2021, Accessed: Feb. 15, 2025. [Online]. Available: https://www.mouser.com/datasheet/2/196/Infineon\_CYW43439\_DataSheet\_v03\_00\_EN-3074791.pdf?srsltid=AfmBOoqh1Fsa7ADgs\_QJUSfQZn4mVc2rR7JT3VoX70H\_gDAn5DZDs7Fx

[28] USB Implementers Forum, “USB Logo Usage Guidelines,” 2024, Accessed: Feb. 15, 2025. [Online]. Available: https://www.usb.org/sites/default/files/usb-if\_original\_logo\_usage\_guidelines\_final\_2024.02.8.pdf

[29] Wi-Fi Alliance, “Generational Wi-Fi User Guide,” Apr. 2023, Accessed: Feb. 15, 2025. [Online]. Available: https://www.wi-fi.org/system/files/Generational\_Wi-Fi\_User\_Guide\_202304.pdf

[30] USB Implementers Forum, “Human Interface Devices (HID) Specifications and Tools.” Accessed: Feb. 15, 2025. [Online]. Available: https://www.usb.org/hid

[31] I. Bluetooth SIG, “Bluetooth Technology Website.” Accessed: Feb. 15, 2025. [Online]. Available: https://www.bluetooth.com/

List of Figures

[Figure 1: Hardware connections at the Wi-Fi phase 14](#_Toc190526463)

[Figure 2: Hardware connections at the BLE phase 14](#_Toc190526464)

[Figure 3: Simplified Flowchart 16](#_Toc190526465)

List of Tables

[Table 1: Schedule for “Applied Mathematics”. 6](#_Toc330300577)

List of Abbreviations

|  |  |
| --- | --- |
| FABI | Flexible Assistive Button Interface |
| USB HID |  |
| I/Os |  |
| BLE | Bluetooth Low Energy |
| GPIO? |  |
| SDK | Software Development Kit |
| AP | Access Point |
|  |  |

Documentation table of AI-based tools

|  |  |  |
| --- | --- | --- |
| **AI-based tools** | **Intended use** | **Prompt, source, page, paragraph...** |
| **DeepL Translate** | Translation of an article in English | Source (XXX), Chapter X on page X-X |
| **ChatGPT (4.0)** | Grammar and spelling | "Please list issues with spelling and grammar in the following text: ..." Entire document |
|  |  |  |

Appendix A: Code for Puck.js

1. const SERVICE\_UUID = '6e400001-b5a3-f393-e0a9-e50e24dcca9e'; // Service UUID

2. const CHARACTERISTIC\_UUID = '6e400003-b5a3-f393-e0a9-e50e24dcca9e'; // Characteristic UUID

3.

4. let buttonState = 0; // 0 = Released, 1 = Pressed

5.

6. // Function to set up services and start advertising

7. function setupBLE() {

8. // Set up GATT services and characteristics

9. NRF.setServices({

10. '6e400001-b5a3-f393-e0a9-e50e24dcca9e': {

11. '6e400031-b5a3-f393-e0a9-e50e24dcca9e': {

12. value: [buttonState], // Initial value

13. readable: true, // Allow reading the value

14. writable: false, // Disable writing to the characteristic

15. notify: true // Enable notifications

16. }

17. }

18. }, { advertise: [SERVICE\_UUID], uart: false });

19.

20. console.log("BLE services set up and advertising started.");

21. }

22.

23. // Function to notify the connected device of button state changes

24. function notifyButtonState() {

25. NRF.updateServices({

26. '6e400001-b5a3-f393-e0a9-e50e24dcca9e': {

27. '6e400003-b5a3-f393-e0a9-e50e24dcca9e': {

28. value: [buttonState], // Initial value

29. readable: true, // Allow reading the value

30. writable: false, // Disable writing to the characteristic

31. notify: true // Enable notifications

32. }

33. }

34. });

35. //console.log("Notification sent: " + buttonState);

36. }

37.

38. // Restart BLE and set up services

39. function restartBLE() {

40. console.log("Restarting BLE...");

41. //NRF.disconnect(); // Disconnect any connected devices

42. NRF.restart(() => {

43. setupBLE(); // Set up BLE after restart

44. console.log("BLE restarted.");

45. });

46. }

47.

48. // Watch for button press

49. setWatch(() => {

50. buttonState = 1;

51. console.log(buttonState); //Raspi recognizes only this

52. notifyButtonState();

53. LED1.write(buttonState);

54. }, BTN, { edge: "rising", debounce: 50, repeat: true });

55.

56. // Watch for button release

57. setWatch(() => {

58. buttonState = 0;

59. console.log(buttonState); //Raspi recognizes only this

60. notifyButtonState();

61. LED1.write(buttonState);

62. }, BTN, { edge: "falling", debounce: 50, repeat: true });

63.

Appendix B: Code for Raspberry Pico W

Appendix C: Code for Arduino Nano RP2040 Connect

1. #include "PluggableUSBHID.h"

2. #include "USBKeyboard.h"

3. #include <SPI.h>

4. #include <WiFiNINA.h>

5. #include <ArduinoBLE.h>

6.

7. //Keyboard

8. USBKeyboard Keyboard;

9.

10. //Wifi

11. bool wifiActive = true; // To track if WiFi is active

12.

13. char ssid[] = "FABI"; //network SSID name

14. char pass[] = "asterics"; //network password (use for WPA, or use as key for WEP)

15.

16. int status = WL\_IDLE\_STATUS;

17. WiFiServer server(80);

18.

19. int maxButtons = 2; // Default value for maximum number of buttons

20. String buttonActions[100]; // Array to store actions for each button

21.

22. //BLE

23. // UUIDs for the service and characteristic

24. const char\* PUCK\_SERVICE\_UUID = "6e400001-b5a3-f393-e0a9-e50e24dcca9e";

25. const char\* PUCK\_CHARACTERISTIC\_UUID = "6e400003-b5a3-f393-e0a9-e50e24dcca9e";

26.

27. BLEDevice puckDevices[100]; //Store up to 100 Pucks

28. BLECharacteristic buttonCharacteristics[100];

29. String puckNames[100]; // Store Puck.js names

30.

31. int puckCount = 0; //How many Puck.js were already found

32.

33. void setup() {

34. Serial.begin(9600);

35. while (!Serial);

36. // put your setup code here, to run once:

37. pinMode(LED\_BUILTIN, OUTPUT);

38. pinMode(LEDR, OUTPUT);

39. pinMode(LEDG, OUTPUT);

40. pinMode(LEDB, OUTPUT);

41.

42. digitalWrite(LEDR, HIGH);

43. delay(200);

44. digitalWrite(LEDR, LOW);

45. digitalWrite(LEDG, HIGH);

46. delay(200);

47. digitalWrite(LEDG, LOW);

48. digitalWrite(LEDB, HIGH);

49. delay(200);

50. digitalWrite(LEDB, LOW);

51. setup\_Wifi();

52. }

53.

54. void loop() {

55. // put your main code here, to run repeatedly:

56. while(wifiActive){

57. loop\_Wifi();

58. }

59. while(!wifiActive){

60. loop\_BLE();

61. Serial.println("!");

62. }

63. }

64.

65. void setup\_BLE(){

66. digitalWrite(LED\_BUILTIN, LOW);

67. // Start BLE central mode

68. if (!BLE.begin()) {

69. Serial.println("Failed to start BLE!");

70. while (1);

71. }

72.

73. Serial.println("BLE Central - Arduino RP2040 Connect");

74. BLE.scan(); // Start scanning for peripherals

75. }

76.

77. void loop\_BLE(){

78. // If less than maxButtons connected, keep scanning

79. if (puckCount < maxButtons) {

80. BLEDevice peripheral = BLE.available();

81.

82. if (peripheral) {

83. String deviceName = peripheral.localName();

84.

85. Serial.print("Found device: ");

86. Serial.println(deviceName);

87.

88. // Check if it's advertising the correct service UUID

89. if (peripheral.advertisedServiceUuid() == PUCK\_SERVICE\_UUID) {

90. Serial.println("Found a Puck.js device!");

91. BLE.stopScan(); // Stop scanning to connect

92.

93. if (peripheral.connect()) {

94. Serial.println("Connected to Puck.js");

95.

96. // Discover the service and characteristic

97. BLEService puckService = peripheral.service(PUCK\_SERVICE\_UUID);

98. if (peripheral.discoverService(PUCK\_SERVICE\_UUID)) {

99. BLECharacteristic buttonCharacteristic = peripheral.characteristic(PUCK\_CHARACTERISTIC\_UUID);

100.

101. if (!buttonCharacteristic || !buttonCharacteristic.canSubscribe() || !buttonCharacteristic.subscribe()) {

102. Serial.println("Failed to subscribe!");

103. peripheral.disconnect();

104. return;

105. } else {

106. Serial.println("Subscribed to button notifications");

107.

108. // Store device and characteristic

109. puckDevices[puckCount] = peripheral;

110. buttonCharacteristics[puckCount] = buttonCharacteristic;

111. puckNames[puckCount] = deviceName;

112.

113. puckCount++;

114. Serial.print("Connected to ");

115. Serial.println(deviceName);

116. }

117. }

118. } else {

119. Serial.println("Failed to connect!");

120. }

121.

122. BLE.scan(); // Continue scanning for more devices

123. }

124. }

125. }

126. checkForButtonPresses(); //The Keyboard Action

127. }

128.

129. //Examples can be adjusted if needed

130. #define KEY\_ENTER 0x28 // Keyboard Return (ENTER)

131. //#define KEY\_ENTER 0x1c // Keyboard Return (ENTER)

132. #define KEY\_SPACE 0x2c // Keyboard Spacebar

133. #define KEY\_RIGHT 0x4f // Keyboard Right Arrow

134. #define KEY\_LEFT 0x50 // Keyboard Left Arrow

135. #define KEY\_DOWN 0x51 // Keyboard Down Arrow

136. #define KEY\_UP 0x52 // Keyboard Up Arrow

137.

138. void buttonMapping(String txt){

139. if(txt == "KEY\_ENTER"){

140. Keyboard.key\_code\_raw(KEY\_ENTER);

141. Serial.println("HENLOO");

142. } else if (txt == "KEY\_SPACE"){

143. Keyboard.key\_code\_raw(KEY\_SPACE);

144. } else if (txt == "KEY\_RIGHT"){

145. Keyboard.key\_code\_raw(KEY\_RIGHT);

146. } else if (txt == "KEY\_LEFT"){

147. Keyboard.key\_code\_raw(KEY\_LEFT);

148. } else if (txt == "KEY\_DOWN"){

149. Keyboard.key\_code\_raw(KEY\_DOWN);

150. } else if (txt == "KEY\_UP"){

151. Keyboard.key\_code\_raw(KEY\_UP);

152. }

153. }

154.

155. void checkForButtonPresses() {

156. // Check for updates from each connected Puck.js device

157. for (int i = 0; i < puckCount; i++) {

158. if (puckDevices[i].connected() && buttonCharacteristics[i].valueUpdated()) {

159. byte value = 0;

160. buttonCharacteristics[i].readValue(value);

161.

162. // Check if the left button (second bit) was pressed

163. if (value & 0x02) {

164. Serial.print(puckNames[i]);

165. Serial.println(" button pressed");

166.

167. // Print the corresponding button action to USB using c\_str()

168. String buttonText = buttonActions[i].c\_str();

169. if (buttonText.startsWith("KEY\_")){

170. buttonMapping(buttonText);

171. digitalWrite(LED\_BUILTIN, HIGH); //RGB LED not compatibl with BLE

172. delay(200);

173. digitalWrite(LED\_BUILTIN, LOW);

174. } else {

175. Keyboard.printf(buttonActions[i].c\_str()); // Convert String to const char\*

176. Serial.println(buttonText);

177. digitalWrite(LED\_BUILTIN, HIGH);

178. delay(200);

179. digitalWrite(LED\_BUILTIN, LOW);

180. }

181. }

182. }

183. }

184. }

185.

186. void setup\_Wifi() {

187. Serial.println("Access Point Web Server");

188. digitalWrite(LED\_BUILTIN, HIGH);

189. // Check for the WiFi module:

190. if (WiFi.status() == WL\_NO\_MODULE) {

191. Serial.println("Communication with WiFi module failed!");

192. while (true);

193. }

194.

195. // Create access point

196. Serial.print("Creating access point named: ");

197. Serial.println(ssid);

198. status = WiFi.beginAP(ssid, pass);

199. if (status != WL\_AP\_LISTENING) {

200. Serial.println("Creating access point failed");

201. while (true);

202. }

203.

204. delay(10000); // Wait for connection

205.

206. // Start the web server on port 80

207. server.begin();

208.

209. // You're connected now, so print out the status

210. printWiFiStatus();

211. }

212.

213. void loop\_Wifi() {

214. WiFiClient client = server.available(); // listen for incoming clients

215. if (client) { // if you get a client

216. Serial.println("New client connected");

217. String currentLine = ""; // Stores current line of HTTP request

218. String request = ""; // Stores full HTTP request line

219. bool isRequestLine = true; // Flag for first request line

220.

221. while (client.connected()) { // loop while the client is connected

222. if (client.available()) { // if there’s bytes to read from the client

223. char c = client.read(); // read a byte

224. Serial.write(c); // print it out to the serial monitor

225.

226. // Accumulate the current line

227. if (c != '\r' && c != '\n') {

228. currentLine += c; // Accumulate the HTTP request line

229. }

230.

231. // If the line is complete (newline), process it

232. if (c == '\n') {

233. // If this is the request line (first line), store it

234. if (isRequestLine) {

235. request = currentLine;

236. Serial.print("Received Request: ");

237. Serial.println(request);

238. isRequestLine = false; // Ensure this is only done once

239. }

240.

241. // If the line is blank, this means the headers are done

242. if (currentLine.length() == 0) {

243. // Send the HTML response (this part happens for every request)

244. client.println("HTTP/1.1 200 OK");

245. client.println("Content-type:text/html");

246. client.println();

247.

248. // Generate the HTML form response

249. client.print("<html><body><h1>Configure Bluetooth Buttons</h1>");

250. client.print("<form action=\"/save\" method=\"GET\">");

251. client.print("<label for=\"maxButtons\">Max Buttons: </label>");

252. client.print("<input type=\"number\" id=\"maxButtons\" name=\"maxButtons\" value=\"" + String(maxButtons) + "\" min=\"1\" max=\"20\"/><br/><br/>");

253.

254. // Generate input fields for each button action

255. for (int i = 0; i < maxButtons; i++) {

256. client.print("<label for=\"button" + String(i) + "\">Button " + String(i + 1) + " Action: </label>");

257. client.print("<input type=\"text\" id=\"button" + String(i) + "\" name=\"button" + String(i) + "\" value=\"" + buttonActions[i] + "\"/><br/><br/>");

258. }

259.

260. client.print("<input type=\"submit\" value=\"Save Configuration\"/></form>");

261. client.print("<form action=\"/startBLE\" method=\"GET\">");

262. client.print("<input type=\"submit\" value=\"Start FABI\"/></form>");

263. client.println("</body></html>");

264.

265. client.println(); // End the response

266. break; // Exit the client loop after sending the response

267. }

268.

269. // Clear the current line for the next line

270. currentLine = "";

271. }

272. }

273. }

274.

275. // Now that the response is sent, handle the request logic

276. client.stop(); // Close the connection after serving the page

277. Serial.println("Client disconnected");

278.

279. // Handle the GET request (check after the client is disconnected)

280. if (request.startsWith("GET /save?")) {

281. Serial.println("Processing save request");

282. parseConfig(request); // Process the configuration form data

283. Keyboard.key\_code\_raw(KEY\_F5);

284. } else if (request.startsWith("GET /startBLE")) {

285. Serial.println("Processing Start FABI request");

286. startFABI(); // Start BLE and disable WiFi

287. wifiActive = false; // Mark WiFi as inactive

288. setup\_BLE(); // Setup BLE

289. }

290.

291. // Reset the request variable after processing

292. request = ""; // Clear request to prevent old data from affecting future requests

293. }

294. }

295.

296. void parseConfig(String line) {

297. // Extract the maximum number of buttons

298. int idx = line.indexOf("maxButtons=");

299. if (idx != -1) {

300. int endIdx = line.indexOf('&', idx);

301. if (endIdx == -1) endIdx = line.length();

302. maxButtons = line.substring(idx + 11, endIdx).toInt();

303. }

304.

305. // Ensure maxButtons is within a valid range

306. maxButtons = constrain(maxButtons, 1, 20); // Limit maxButtons

307.

308. // Extract actions for each button

309. for (int i = 0; i < maxButtons; i++) {

310. String buttonParam = "button" + String(i) + "=";

311. idx = line.indexOf(buttonParam);

312. if (idx != -1) {

313. int endIdx = line.indexOf('&', idx);

314. if (endIdx == -1) endIdx = line.length();

315. String action = line.substring(idx + buttonParam.length(), endIdx);

316. action.replace("+", " "); // Replace '+' with space

317. action.trim(); // Remove leading/trailing whitespace

318.

319. // Remove anything starting with "HTTP"

320. int httpIdx = action.indexOf("HTTP");

321. if (httpIdx != -1) {

322. action = action.substring(0, httpIdx); // Truncate before "HTTP"

323. }

324. action.trim();

325. // Store the cleaned action

326. buttonActions[i] = action; // Save the action for the button

327. Serial.println("Button " + String(i + 1) + " Action: " + buttonActions[i]);

328. }

329. }

330.

331. // Reset the buttonActions array if necessary

332. for (int i = maxButtons; i < 20; i++) {

333. buttonActions[i] = ""; // Clear any unused actions

334. }

335. }

336.

337. void startFABI() {

338. // Close Wifi Connection so BLE can start

339. WiFi.end();

340. Serial.println("WiFi off, BLE started");

341. }

342.

343. void printWiFiStatus() {

344. Serial.print("SSID: ");

345. Serial.println(WiFi.SSID());

346.

347. IPAddress ip = WiFi.localIP();

348. Serial.print("IP Address: ");

349. Serial.println(ip);

350.

351. Serial.print("To see this page in action, open a browser to http://");

352. Serial.println(ip);

353. }

Appendix D: Code for Puck.js (for the Connection with Arduino Nano RP2040 Connect)

1. // Service UUID and Characteristic UUID (using a valid 16-bit alias)

2. const SERVICE\_UUID = '6e400001-b5a3-f393-e0a9-e50e24dcca9e';

3. const CHARACTERISTIC\_UUID = '6e400003-b5a3-f393-e0a9-e50e24dcca9e';

4.

5. var state = 1;

6. function press() {

7. /\* switch(state) {

8. case 0:

9. state = 1;

10. LED1.write(1);

11. break;

12. case 1:

13. state = 0;

14. LED1.write(0);

15. break;

16. } \*/

17.

18. LED1.write(1);

19. setTimeout(function(){

20. LED1.write(0);

21. }, 100);

22. sendButtonState();

23. }

24.

25. // Detect button press and release

26. setWatch(press, BTN1, { edge: 'rising', debounce: 50, repeat: true });

27.

28. // Start advertising with specified service

29. NRF.setAdvertising({}, {

30. name: "Puck.js Button 1",

31. connectable: true,

32. services: [SERVICE\_UUID] // Ensure the service UUID is included

33. });

34.

35. // Function to send button state over BLE

36. function sendButtonState(state) {

37. // Advertise with services that notify button press state

38. NRF.setServices({

39. '6e400001-b5a3-f393-e0a9-e50e24dcca9e': {

40. '6e400003-b5a3-f393-e0a9-e50e24dcca9e': {

41. value: [state],

42. readable: true,

43. notify: true

44. }

45. }

46. });

47. NRF.setAdvertising({}, {

48. name: "Puck.js Button 2",

49. connectable: true,

50. services: [SERVICE\_UUID], // Ensure the service UUID is included

51. manufacturer: 0x0590,

52. manufacturerData: JSON.stringify({state}),

53. });

54. }

Appendix E: Link to the Github Repository

<https://github.com/FriedrichKoenig/WirelessButtonforFABI>