René Ramsauer

Bachelor Project: IOT Localization Tracker

Integration of embedded systems.
Last change: 02.04.2023
Project name: IoT - Localization ESP32 (Galaxy Extension)
Short name: BIC_IoT_LT_ESP32_GE(A)
Version: V1.2

1. UWB BLE Tracker (DW1000, ESP32)

(Implementierung of UWB Galaxy Extension: "UWB Anchor")

1.1. Introduction

This embedded software is an implementation of a so called key tracker system. Compared to conventional systems the implementation has a scientific background. For this reason the software only consists of the following tasks:

- Provision of a BLE server with the following sevices:
  - BLE Batery level service.
  - UWB shared shord adress

    - UWB shared unique identifier
- Provision of a UWB distance measuring service
- Provision of a battery measuring service.

Notice: We used a usb power bank for the test setup. Thus the battery service was deactivated with the help of preprocessor stadement in ble_server.h.

As you can see below, this description consists of two hardware systems. A target prototype and a 3rd party prototype. Since there were some problems with the implementation of the target prototype system and these problems could not be solved in a reasonable time frame, the 3rd party hardware was used.

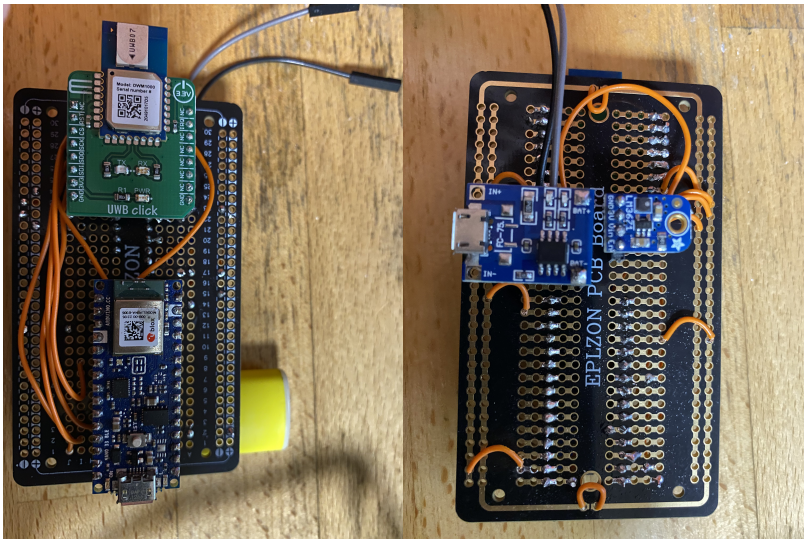
1.2. Prototype

1.2.1. Target prototype.

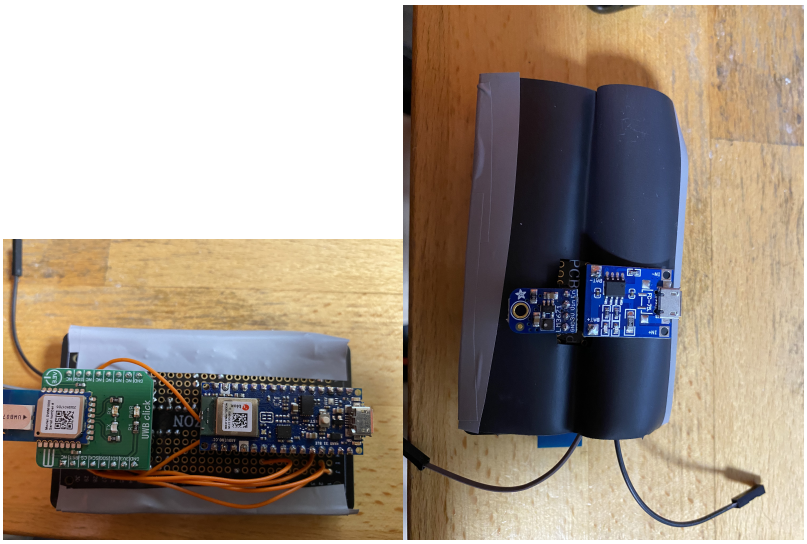
Attention: Since there were some problems with the prototypes, this system was not commissioned in working order. More detailed information on the problem can be found under "2.2 Problem with target prototype".

The target prototype components were selected based on the performance and size of the target system.

Without insulation



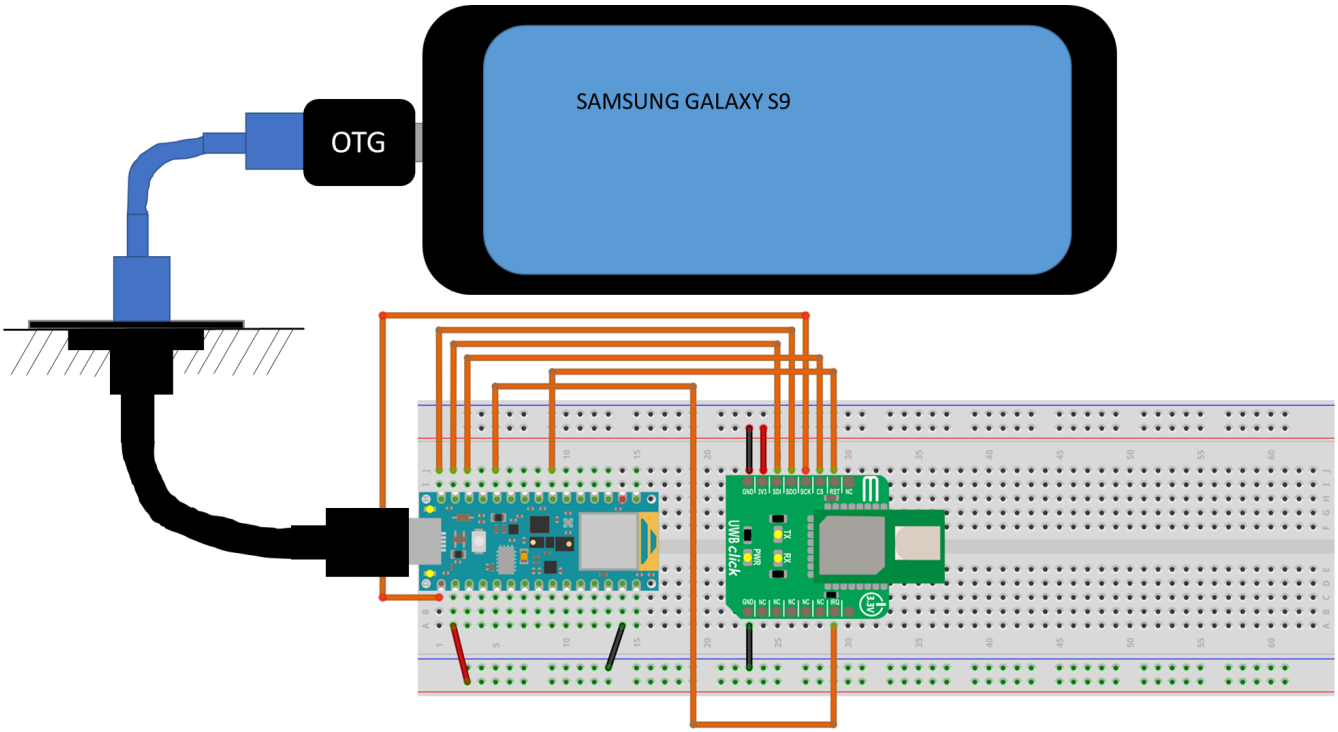
With insulation for the back to avoid short circuits.



1.2.1.1. Components of target Prototype

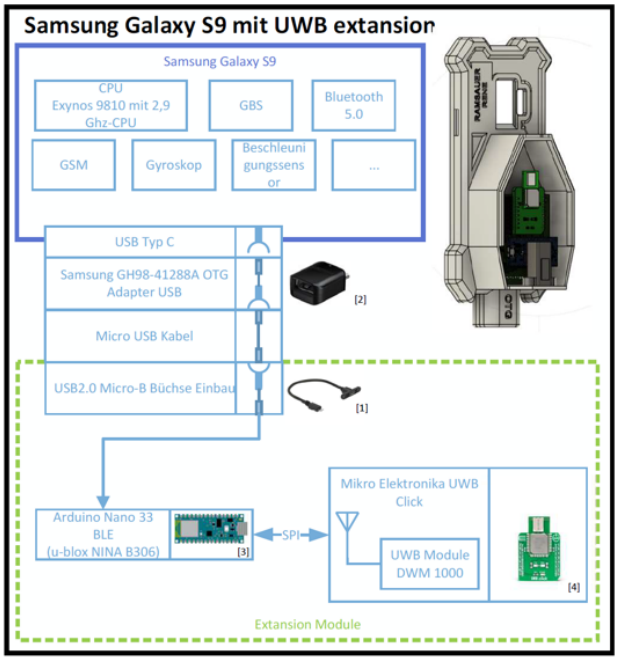
- Mikroe UWB click (UWB Module DWM1000)
- Arduino Nano 33 BLE (yC = nRF52840, Module = NINA-B306)
- Adafruit LM3671 3.3V Buck Converter Breakout (Buck Converter Breakout)
- AZ-Delivery TP4056 (Laderegler)
- EEMB LP542730 (Li-ion Battery)
- Wires, boards and resistors
- OTG Adapter

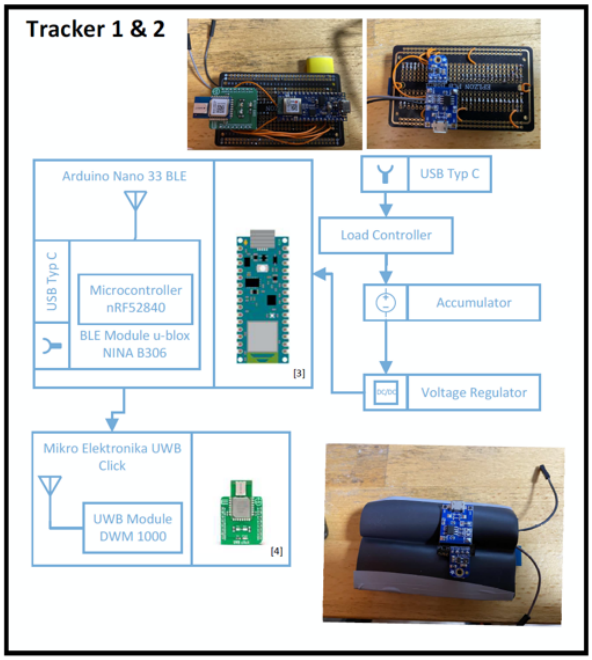
1.2.1.2. Target prototype wiring



As you can see from the prototype picture, this has already been soldered on a prototype board.

1.2.1.3. Target prototype hardware block diagram (Red Marked)





Quelle Image:
[1] https://www.delock.de/produkt/85245/merkmale.html
[2] https://www.samsung.com/at/mobile-accessories/usb-adapter-un930-ee-un930bbegww/
[3] https://docs.arduino.cc/hardware/nano-33-ble
[4] https://www.mikroe.com/uwb-click

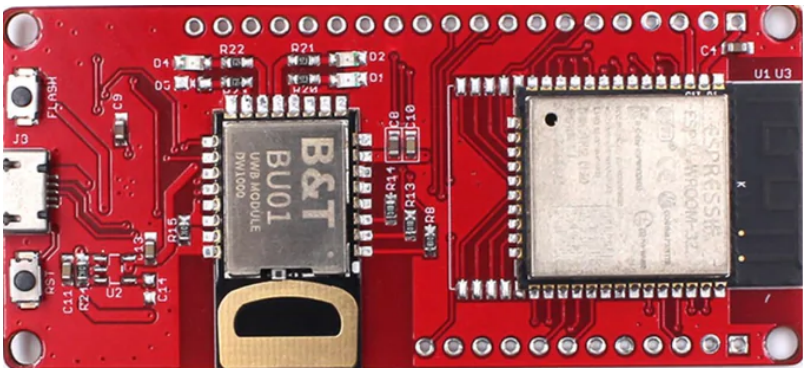
For better testing of the system, however, the board was split and the assembly was distributed to the rear and forme. Refer to the photos.

1.2.1.4. Problem with target prototype.

During the implementation we had problems with the libary of Arduino. Due to time constraints, we did not pursue fixing the problem. However, we could exclude that the error occurred due to the wrong representation. This was determined with the help of simultaions and measurements with oscilloscope and lapornetzgerät. All inputs and outputs detected the signal with the same level.

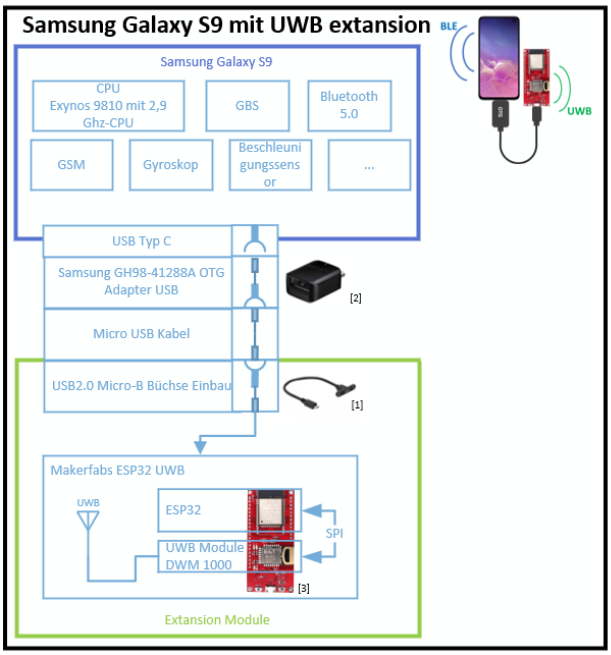
1.2.2. Aktueller Prototype (3rd Party)

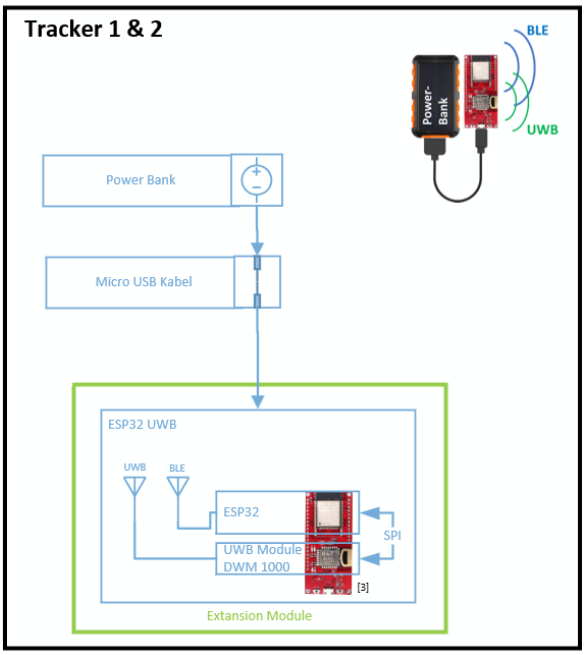
The system "ESP32 UWB" from Makerfabs was used as the current prototype. Which is equipped with an ESP32 and a DW1000 module. See picture:



Weitere Informatationen finden sie unter "iot-localization_tracker/docs/project_documentation/000_Specification _Requirement/002_Hardware-Specification/...".

1.2.2.1. Aktueller Prototype (3rd Party) hardware block diagram





Quelle Image:
[1] https://www.delock.de/produkt/85245/merkmale.html
[2] https://www.samsung.com/at/mobile-accessories/usb-adapter-un930-ee-un930bbegww/
[3] https://www.makerfabs.com/esp32-uwb-ultra-wideband.html

1.3. Software

1.3.1. File Description

```
└── BLE_UWB_Tracker
    ├── .pio                    // PlatformIO File
    ├── .vscode                 // PlatformIO File
    ├── img                     // Image for Readme.md
    ├── include                 // Directory for includes
    │   ├── main.h              // Header for Main (main.cpp)
    │   ├── log_settings.h      // Header for Logging Settings
    │   ├── uwb_anchor.h        // Header for DWM1000 UWB module (uwb_anchor.cpp)
    ├── lib
    ├── src                     // Source-Code File
    │   ├── main.cpp            // Implementation of Main
    │   ├── uwb_anchor.cpp      // Implementation of DWM1000 UWB module.
    └── test                    // No tests were implemented due to the
implementation size
```

1.3.2. Integrierte Entwicklungsumgebung

The software was implemented using the PlatformIO IDE.

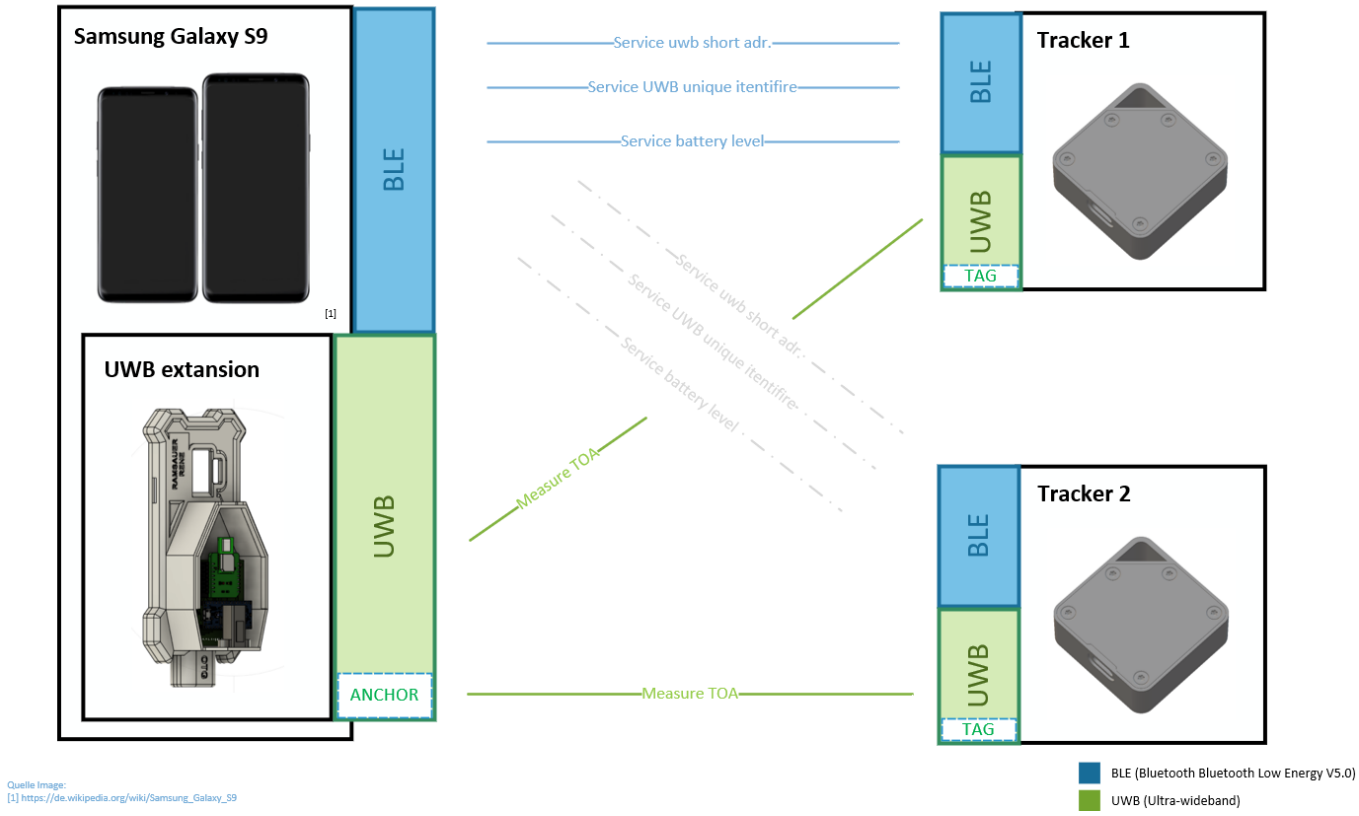
https://platformio.org/

1.3.3. Used libraries (3rd Party)

Include	Creator	Description
<SPI.h>	Arduino	Library for SPI communication
<ArduinoLog.h>	Thijs Elenbaas	An minimalistic Logging framework for Arduino devices
<Arduino.h>	Arduino Libraries	librarie für nRF52840
<DW1000.h>	Thomas Trojer	Biblioteck for UWB module (DWM1000)

1.3.4. Software Block Diagram

Software Block Diagram overview:(The device implemented here is red marked.)



Software Block Diagram of Anchor:

1.3.5. Software struct

The software is divided into three modules and one logging, with each module implemented in a separate C++ file:

- UWB Anchor Service

All modules are merged in the main.cpp.

1.3.5.1. UWB Anchor Service (uwb_anchor.h | uwb_anchor.cpp)

The uwb service makes it possible to measure discharges via uwb. To integrate the UWB anchor into the application, only two functions need to be called. The function `initUwb()` must only be called during initialization and the function `uwbLoop()` must be called in an interval.

```
/** This function provides the functionalety of the uwb module. */
void initUwb();
/** This function init the uwb module as a Anchor. */
void uwbLoop();
```

Furthermore, the uwb short address settings and the uwb unique itentifier can be made via the header `uwb_anchor.h`.

```
// Define Anchor Adress
#define ADRESS                      "84:00:5B:D5:A9:9A:E2:9C"
// Settings Short Adr.
#define IS_RENDOM_SHORT_ADR             (true)
// Write measuring data to seriel
#define WRITE_MEASURING_DATA_TO_SERIEL  (true)
```

More information you can finde in the code documentation.

1.3.5.2. Logging Settings (log_settings.h)

The loglevel is set with the parameter:

```
/*
 * 0 - LOG_LEVEL_SILENT     no output
 * 1 - LOG_LEVEL_FATAL      fatal errors
 * 2 - LOG_LEVEL_ERROR      all errors
 * 3 - LOG_LEVEL_WARNING    errors and warnings
 * 4 - LOG_LEVEL_INFO       errors, warnings and notices
 * 4 - LOG_LEVEL_NOTICE     Same as INFO, kept for backward compatibility
 * 5 - LOG_LEVEL_TRACE      errors, warnings, notices, traces
 * 6 - LOG_LEVEL_VERBOSE    all
 */
#define DEBUG_LV (LOG_LEVEL_VERBOSE)
```

Furthermore, additional debug output can be output with the following parameters:

```
#define SHOW_UWB_DEVICE_INFO            (true)
```

1.4. Created by:

Name	Email
René Josef Ramsauer	ic18b066@technikum-wien.at