René Ramsauer

Bachelor Project: IOT Localization Tracker

Integration of embedded systems.
Last change: 18.02.2023
Project name: IoT - Localization ESP32 (Tracker)
Short name: BIC_IoT_LT_(T)
Version: V1.2

1. UWB BLE Tracker (DW1000, ESP32)

(Implementierung of Tracker 1 & 2: "BLE Server" and "UWB Tag")

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.

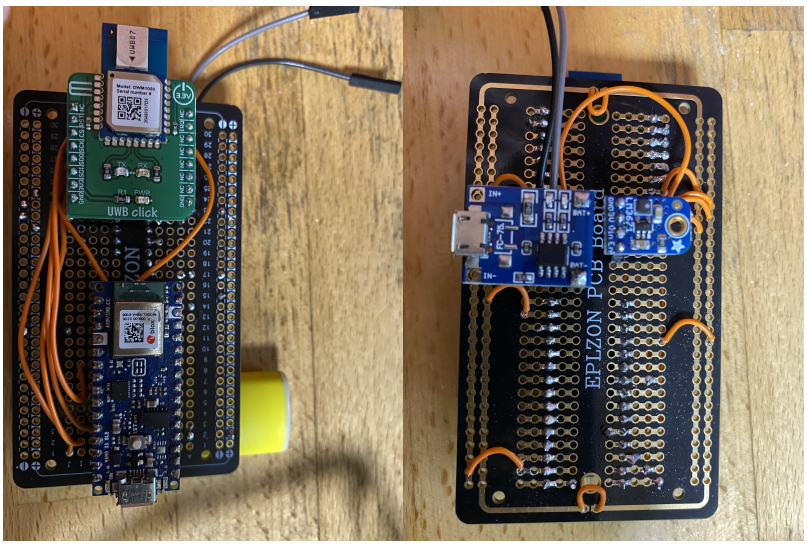
1.2. Prototype

1.2.1. Target prototype.

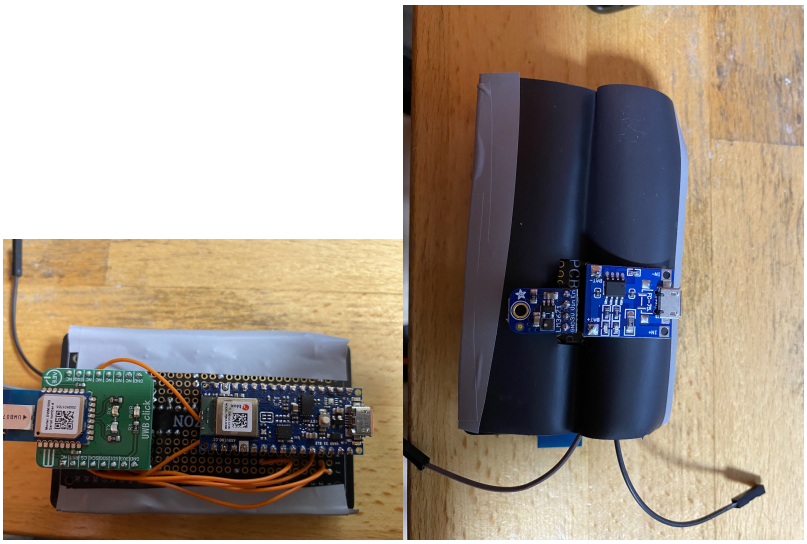
Attention: Since there were some problems with the prototypes, this system was not commissioned in working order. More detailed information on the problem can be found under "2.2 Problem with target prototype".

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

Without insulation



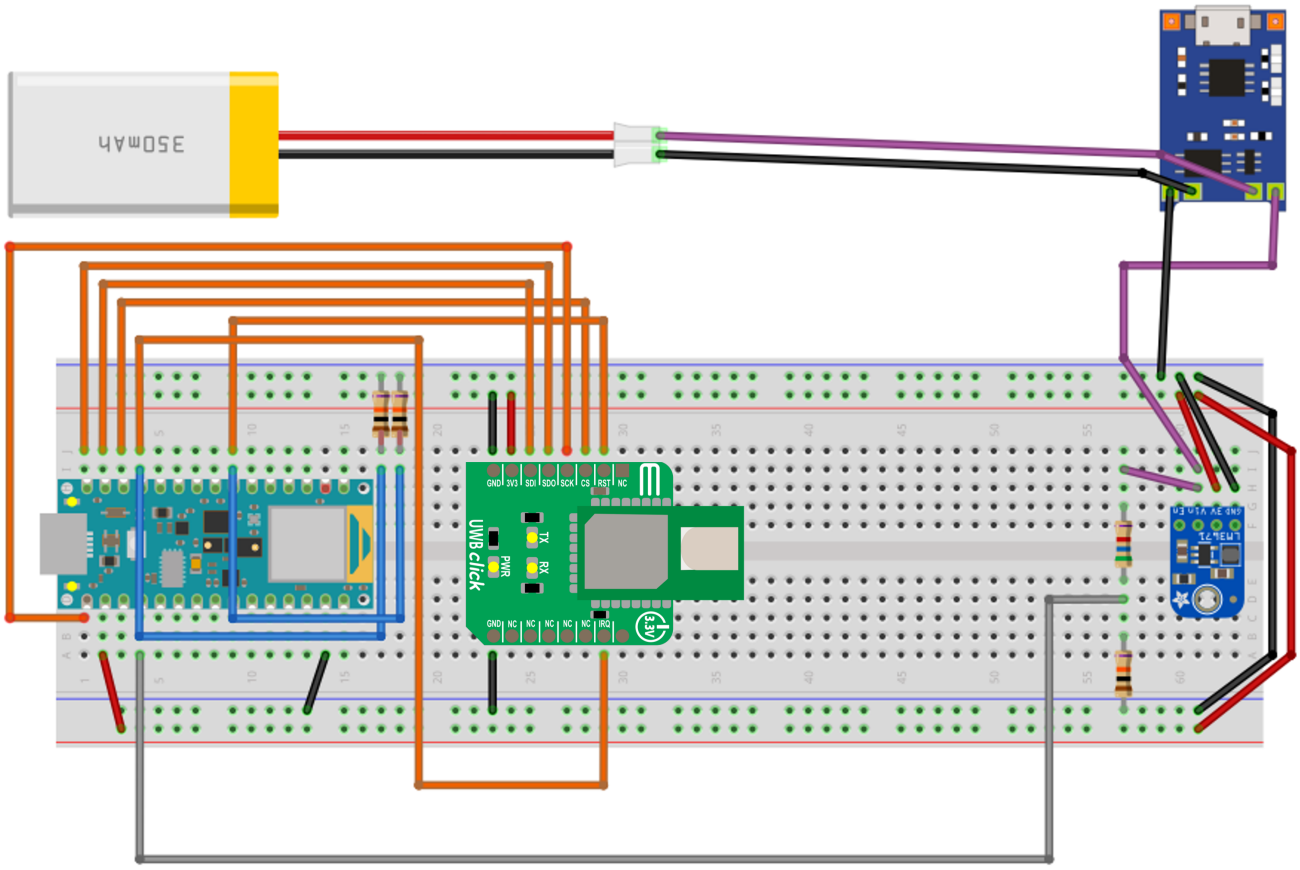
With insulation for the back to avoid short circuits.



1.2.1.1. Components of target Prototype

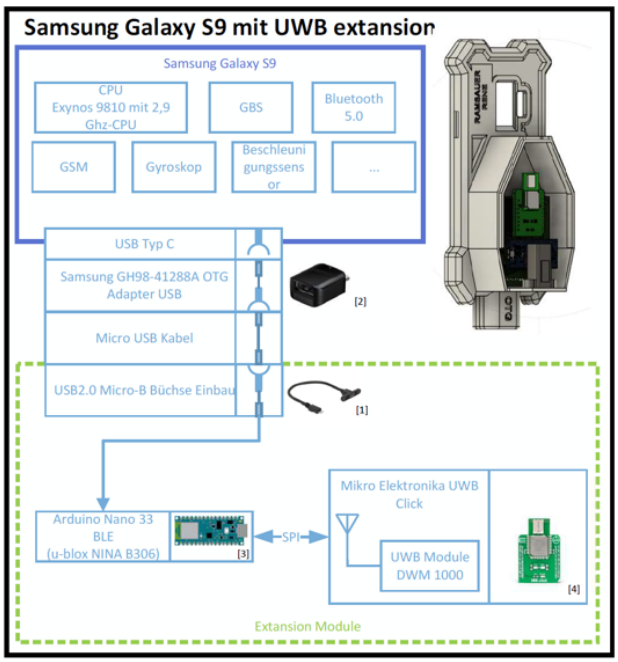
- Mikroe UWB click (UWB Module DWM1000)
- Arduino Nano 33 BLE (yC = nRF52840, Module = NINA-B306)
- Adafruit LM3671 3.3V Buck Converter Breakout (Buck Converter Breakout)
- AZ-Delivery TP4056 (Laderegler)
- EEMB LP542730 (Li-ion Battery)
- Wires, boards and resistors

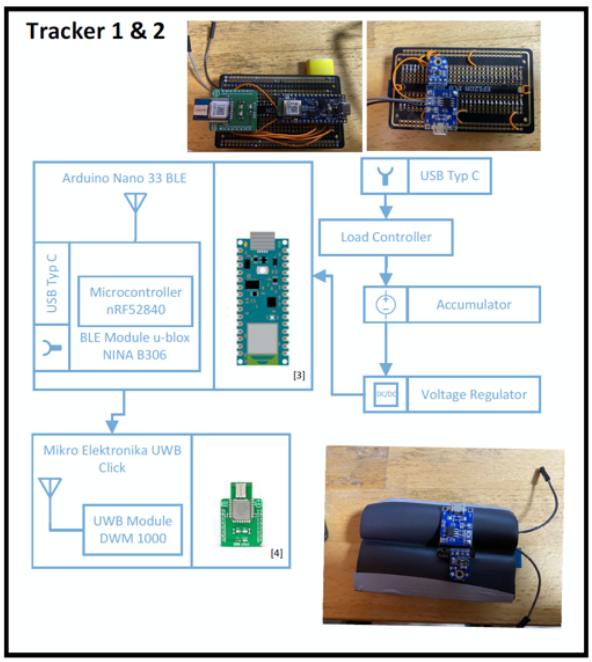
1.2.1.2. Target prototype wiring



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

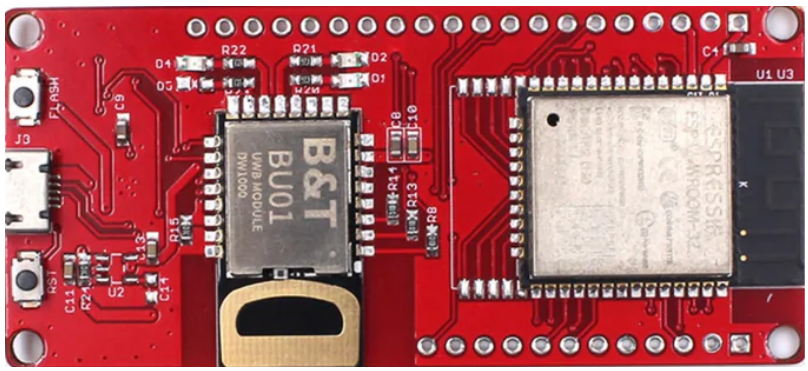
For better testing of the system, however, the board was split and the assembly was distributed to the rear and forme. Refer to the photos.

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.

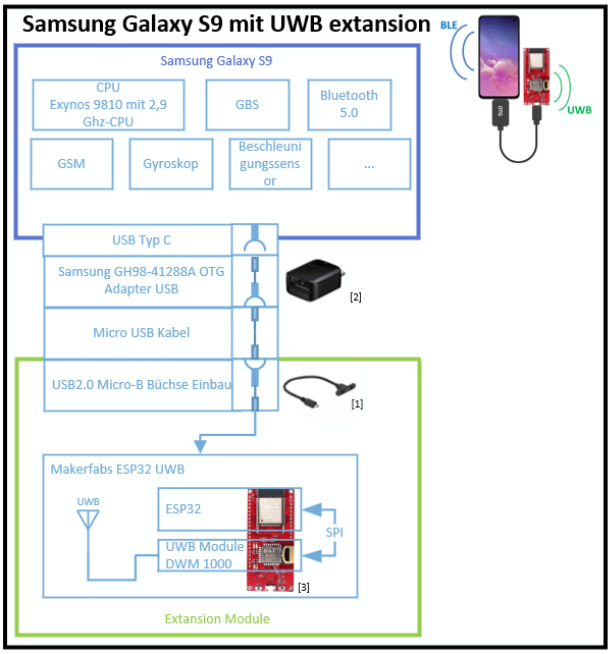
1.2.2. Aktueller Prototype (3rd Party)

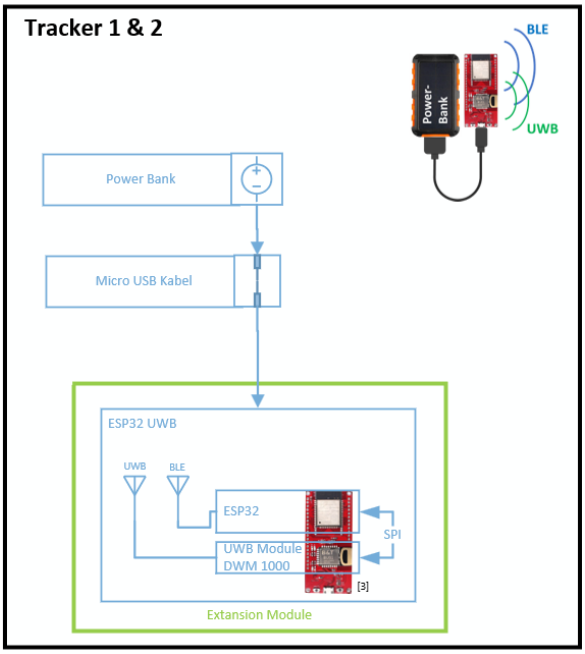
The system "ESP32 UWB" from Makerfabs was used as the current prototype. Which is equipped with an ESP32 and a DW1000 module. See picture:



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)
    │   ├── batt_monitoring.h   // Header Battery monitoring (batt_monitoring.cpp)
    │   ├── log_settings.h      // Header for Logging Settings
    │   ├── ble_server.h        // Header for ESP32 BLE (ble_server.cpp)
    │   ├── uwb_tag.h           // Header for DWM1000 UWB module (uwb_service.cpp)
    ├── lib
    ├── src                     // Source-Code File
    │   ├── main.cpp            // Implementation of Main
    │   ├── batt_monitoring.cpp // Implementation of Batt monitoring.
    │   ├── ble_server.cpp      // Implementation of ESP32 BLE.
    │   ├── uwb_tag.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.

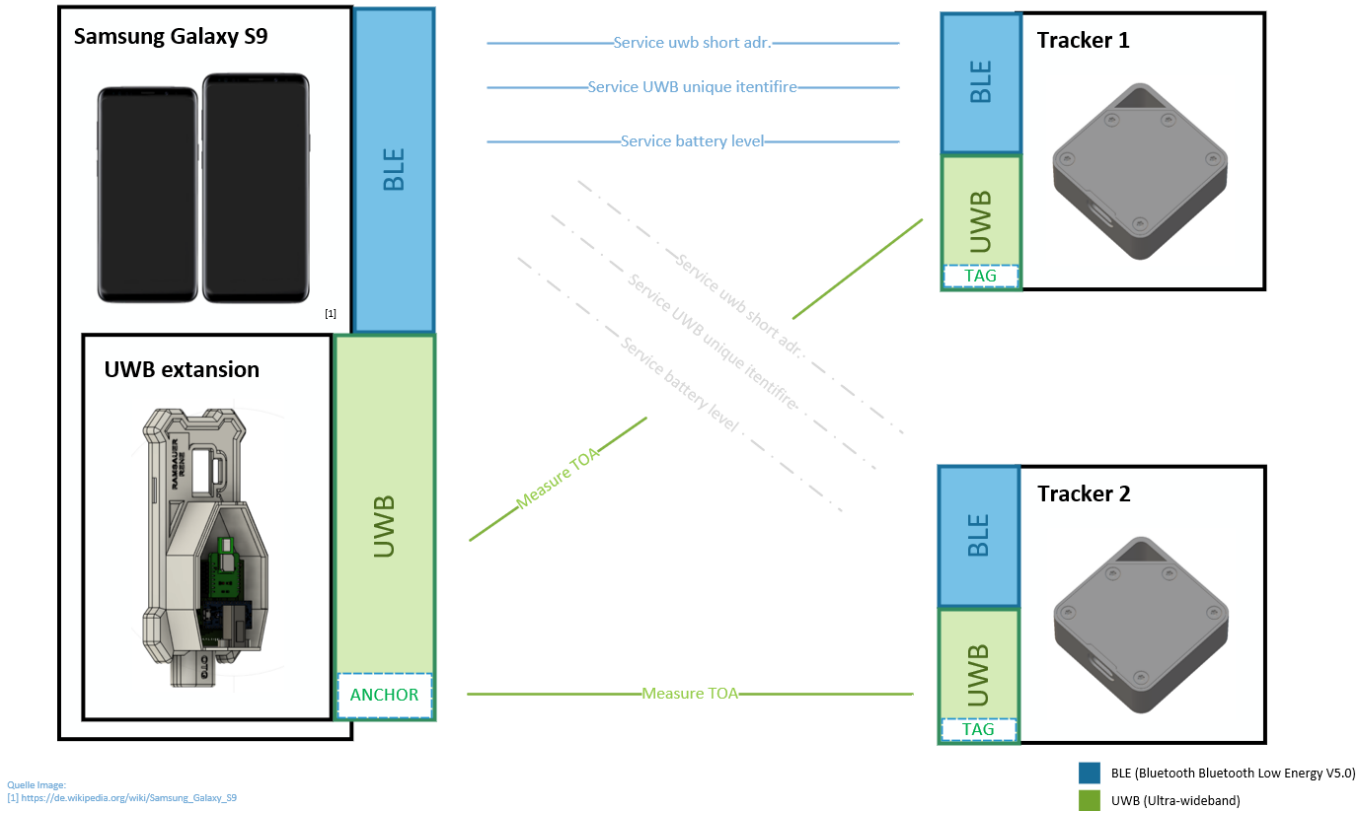
https://platformio.org/

1.3.3. Used libraries (3rd Party)

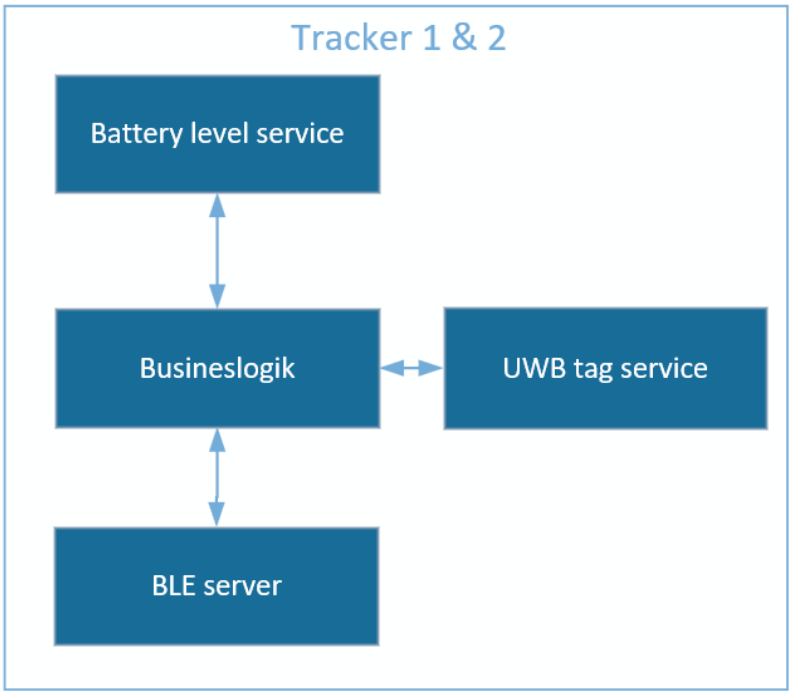
Include	Creator	Description
<SPI.h>	Arduino	Library for SPI communication
<ArduinoLog.h>	Thijs Elenbaas	An minimalistic Logging framework for Arduino devices
<Arduino.h>	Arduino Libraries	librarie für nRF52840
<BLEDevice.h>	Mr. Kolban	Bluetooth® Low Energy librarie für ESP32
<BLEUtils.h>	Mr. Kolban	Bluetooth® Low Energy librarie für ESP32
<BLEServer.h>	Mr. Kolban	Bluetooth® Low Energy librarie für ESP32 (Server)
<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 Tracker:



1.3.5. Software struct

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

- Battery monitoring service
- UWB Tag Service
- BLE Server

All modules are merged in the main.cpp.

1.3.5.1. Battery monitoring service (batt_monitoring.h | batt_monitoring.cpp)

The Battery monitoring service allows to determine the battery level via the battery voltage. Necessary parameters can be defined in the header.

```
#define RESOLUTION_ADC 1023
#define VOLTAGE_IO 3.3
#define VOLTAGE_DEVIDER 0.625
```

This service provides two function:

```
/* Return Voltage of IO */
float batt_reatVoltage(int i_pin, float f_voltIO, int i_resolutionADConverter);
//Bsp.: batt_reatVoltage(34, VOLTAGE_IO, RESOLUTION_ADC);

/* Estimation the state of the battery charge level of LIPO */
int batt_estimationStateOfBatteryCharge_LIPO(float voltage, int countCell);
//Bsp.: batt_estimationStateOfBatteryCharge_LIPO(batt_reatVoltage(34, VOLTAGE_IO,
RESOLUTION_ADC),1);
```

1.3.5.2. UWB Tag Service (uwb_tag.h | uwb_tag.cpp)

The uwb service makes it possible to measure discharges via uwb. To integrate the UWB tag 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 Tag. */
void uwbLoop();
```

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

```
// Define Anchor Adress
#define ADRESS                      "7D:09:22:EA:82:60:3B:9C"
// Settings Short Adr.
#define IS_RENDOM_SHORT_ADR         (false)
```

More information you can finde in the code documentation.

1.3.5.3. BLE Server (ble_server.h | ble_server.cpp)

To integrate the BLE server into the application, only two functions have to be called. The function `initBLE()` must only be called during initialization and the function `bleLoop()` must be called in an interval.

```
/** This function initiates the BLE server with a specific name. */
void initBLE(std::__cxx11::string bleDeviceName);
/** This function provides an lopp for update BLE notfication. */
void bleLoop(uint8_t battLevelValue, std::__cxx11::string uwbShortID,
std::__cxx11::string uwbUID);
```

Since the passing of the parameters is described in detail in the code documentation, this is not explained in more detail here.

Furthermore, the ble services to be provided can be activated and deactivated via `ble_server.h`. In the implementation here, for example, the battery service has been disabled because it is not needed. As can be seen in the graphic above, the system is supplied with power via a powerbank.

```
/* Define which service is enabled. */
#define BETERRY_LVL_SERVICE_AKTIVATE (false)
#define UWB_SERVICE_AKTIVATE (true)
```

It should also be mentioned that when implementing the BLE server, care was taken to ensure that it is possible to extend the services without any problems.

More information you can finde in the code documentation.

1.3.5.4. 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_DISTANCE               (true)
#define SHOW_UWB_DEVICE_INFO            (true)
#define SHOW_BLE_UPDATE_VALUE_SERVICE   (true)
#define SHOW_VOLTAGE_BAT                (false)
#define SHOW_STATE_BAT                  (false)
```

1.4. Futur work and bugs

1.4.1. Bug and

- After a disconnect with the device a reset is necessary to re-establish a connection with the same device.

1.4.2. Futur Work

- Implementation target system.

1.5. Created by:

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