

# Challenger RP2040 UWB (DWM3000)

## Progress Documentation

Siddharth Patel

HTW Berlin

### Document Summary

This document tracks the steps taken to bring up the iLabs Challenger RP2040 UWB board, program it using the Arduino toolchain, run initial UWB examples, and explore how distance data can be integrated into a larger robotics system (e.g. Raspberry Pi 5).

<b>Board</b>	iLabs Challenger RP2040 UWB (DWM3000)
<b>UWB Chip</b>	Qorvo DW3000 (DWM3000 module)
<b>Main MCU</b>	Raspberry Pi RP2040
<b>Toolchain</b>	Arduino IDE (RP2040 core)
<b>Date</b>	November 23, 2025

## Contents

---

### **List of Figures**

---

### **List of Tables**

---

## 1 Programming Options for the Challenger RP2040 UWB

### 1.1 RP2040 + DWM3000: Board Overview

The *Challenger RP2040 UWB* from iLabs combines

$$\text{ChallengerRP2040UWB} = \text{RP2040(MCU)} + \text{DWM3000(UWB)}.$$

The RP2040 defines how we can program the board, while the DWM3000 defines which UWB libraries we can actually use.

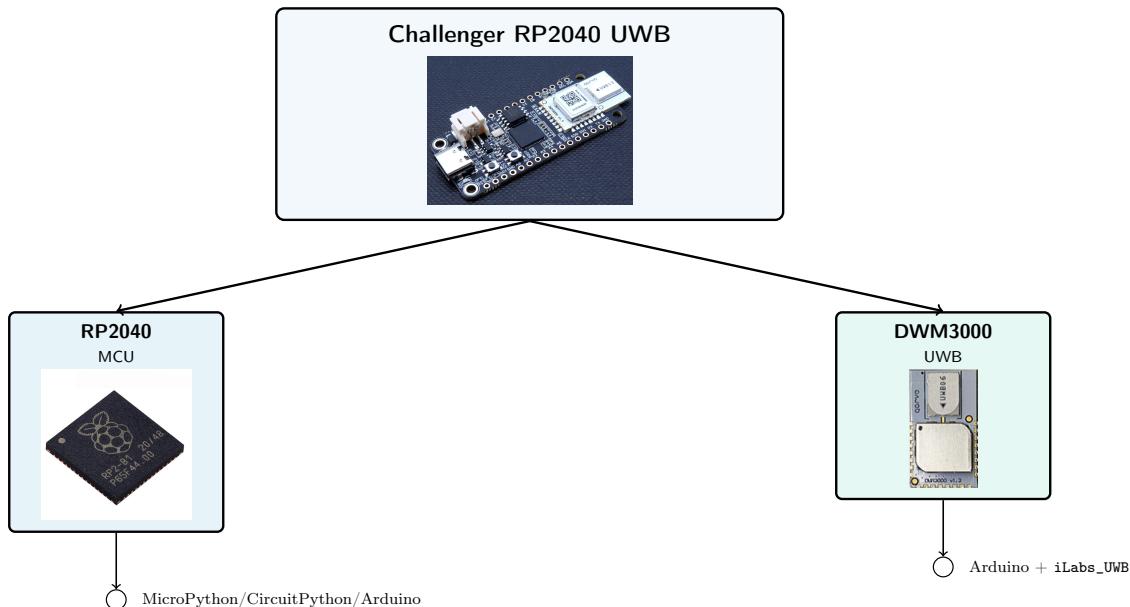


Figure 1: Conceptual view of the Challenger RP2040 UWB: RP2040 MCU, DWM3000 UWB module and their firmware options.

### 1.2 Firmware Options on the RP2040

Because the RP2040 is the same MCU used on the Raspberry Pi Pico, it can in principle be programmed with:

- **MicroPython**<sup>1</sup>
- **CircuitPython**<sup>2</sup>
- **Arduino (C++ with Arduino IDE)**<sup>3</sup>

### 1.3 Library Support for the DWM3000

For the UWB part, the situation is much more restricted:

- The iLabs product page notes that, although the board is compatible with CircuitPython, “*to date there does not exist a support library in Python for the DWM3000 module.*”<sup>4</sup>

<sup>1</sup><https://micropython.org/download/?mcu=rp2040>

<sup>2</sup>[https://circuitpython.org/board/raspberry\\_pi\\_pico/](https://circuitpython.org/board/raspberry_pi_pico/)

<sup>3</sup><https://ilabs.se/getting-your-challenger-rp2040-board-up-and-running-with-the-arduino-ide/>

<sup>4</sup><https://ilabs.se/product/challenger-rp2040-uwb/>

- iLabs provides an official **Arduino library** and examples for the DWM3000 via the [iLabs\\_UWB](#) repository.<sup>5</sup>

## 1.4 Conclusion for This Project

Because only the Arduino ecosystem currently offers a maintained UWB library for the DWM3000, all experiments in this document are based on:

$$RP2040\text{firmware} = \text{Arduino language} + \text{Arduino IDE} + \text{iLabs\_UWB library}.$$

MicroPython and CircuitPython remain theoretical options for the RP2040 itself, but are not used here due to the missing DWM3000 support.

---

<sup>5</sup>[https://github.com/Pontus0/iLabs\\_UWB](https://github.com/Pontus0/iLabs_UWB)

## 2 Arduino-Based Workflow for the Challenger RP2040 UWB

### 2.1 Following the iLabs Getting-Started Guide

The official starting point for programming the Challenger RP2040 UWB is the iLabs Arduino guide.<sup>6</sup> It explains how to install the iLabs board package and select the correct board in the Arduino IDE.

#### Basic setup (summary)

1. Install the iLabs RP2040 board package in the Arduino IDE as described in the guide.
2. In Tools → Board, select iLabs Challenger 2040 UWB.
3. Verify that a serial port for the board (e.g. /dev/cu.usbmodemXXXX) is visible.

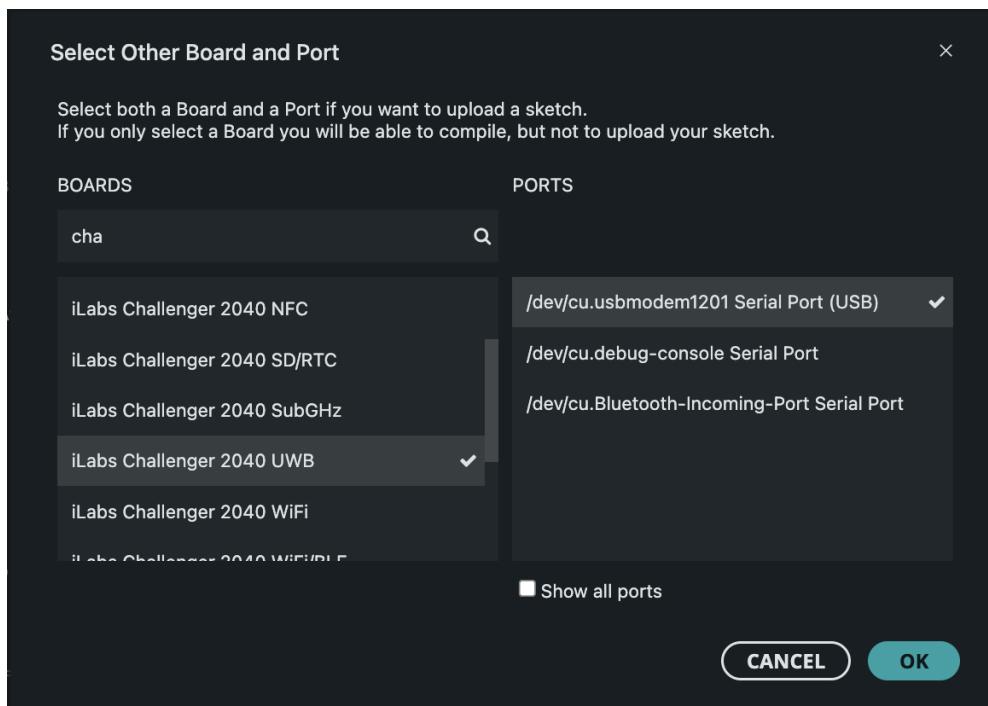


Figure 2: Selecting the iLabs Challenger 2040 UWB board and USB serial port in the Arduino IDE.

### 2.2 Bootloader Issue: “No Drive Detected”

When uploading directly from the Arduino IDE, the RP2040 bootloader is supposed to mount as a USB drive. On macOS this sometimes fails and the IDE reports:

```
Output  Serial Monitor
Sketch uses 80384 bytes (0%) of program storage space. Maximum is 8384512 bytes.
Global variables use 19228 bytes (7%) of dynamic memory, leaving 242916 bytes for
Resetting /dev/cu.usbmodem101
Converting to uf2, output size: 195072, start address: 0x2000
Scanning for RP2040 devices
No drive to deploy.
Failed uploading: uploading error: exit status 1
```

Figure 3: Typical Arduino upload error when the RP2040 UF2 drive is not detected.

To work around this, sketches are uploaded via a .uf2 file and manual drag&drop.

<sup>6</sup><https://ilabs.se/getting-your-challenger-rp2040-board-up-and-running-with-the-arduino-ide/>

## 2.3 Programming via UF2 Files

The UF2 method separates two things:

1. the Arduino IDE *compiles* a .uf2 file, and
2. the Mac *copies* that file to the RP2040 boot drive.

### 2.3.1 One-time setup in the Arduino IDE

#### Step 1: IDE configuration

1. Open your sketch in the Arduino IDE.
2. In Tools set:
  - Board → iLabs Challenger 2040 UWB,
  - Upload Method → Default (UF2),
  - USB Stack → Pico SDK.
3. Choose Sketch → Export compiled Binary.
4. Open Sketch → Show Sketch Folder and locate the generated .uf2 file (e.g. my\_sketch.ino.uf2).

### 2.3.2 Put the board into BOOTSEL mode

#### Step 2: Enter BOOTSEL mode

1. Unplug the USB cable from the Challenger RP2040 UWB.
2. Hold the **BOOT** button and plug the USB cable back in.
3. Release **BOOT** when a new USB drive named RPI-RP2 appears on the Mac.

### 2.3.3 Upload the program via UF2

#### Step 3: Copy UF2 file

1. Drag the .uf2 file from the sketch folder onto the RPI-RP2 drive in Finder.
2. Wait until RPI-RP2 automatically ejects.
3. Unplug and reconnect the board in normal mode. The new sketch is now running.

## 2.4 FreeRTOS Configuration Error

Some UWB example sketches use FreeRTOS on the RP2040. If FreeRTOS is not enabled, compilation fails with an error similar to:

```

 4 | #error "#define __FREERTOS 1 to use FreeRTOS in your application"
  | ^~~~~
exit status 1

Compilation error: exit status 1

```

Figure 4: Compilation error when FreeRTOS is not enabled for the Challenger RP2040 UWB.

#### 2.4.1 Enable FreeRTOS for the Challenger RP2040 UWB

Fix: enable FreeRTOS

1. Open the example sketch (e.g. `ex_01a_simple_tx.ino`) in the Arduino IDE.
2. In Tools set:
  - Board → iLabs Challenger 2040 UWB,
  - Upload Method → Default (UF2),
  - USB Stack → Pico SDK,
  - Operating System → **FreeRTOS SMP**.
3. Compile again with Sketch → Verify/Compile.
4. If compilation succeeds, upload the sketch via the UF2 workflow described above.

### 3 Which Data Can We Get from the DWM3000?

#### 3.1 Raw data directly from the DWM3000

When the RP2040 talks to the DWM3000 over SPI, it can read several types of *raw* information from the chip registers.<sup>7</sup>

Table 1: Typical raw data available from the DWM3000

---

##### Raw data type (examples)

---

- RX and TX timestamps in the internal UWB clock
  - Received UWB frame bytes (header fields and payload)
  - Channel impulse response (CIR) samples
  - RX quality and diagnostic values (power, noise, CRC status, ...)
  - Status and interrupt flags (RXOK, timeout, errors, ...)
- 

These low-level values are the building blocks. From them we can compute things like time-of-flight, distance or more advanced metrics.

#### 3.2 Derived data on RP2040 using the UWB library

With the iLabs UWB library running on the RP2040, these raw values are turned into more convenient, *processed* data structures:

Table 2: Examples of derived data on the RP2040

---

##### Processed data (examples)

---

- Distance between two nodes (single range result in m or mm)
  - Simple range quality or “RSSI” value per distance measurement
  - Logged packets that combine payload, timestamps and diagnostics
  - 2D or 3D position estimates of a tag based on multiple ranges
  - Debug recordings of CIR, timestamps and error counters
- 

<sup>7</sup>Full details are in the DWM3000 datasheet: <https://www.qorvo.com/products/p/DWM3000#documents>

## 4 Retrieving Raw UWB Data on the RP2040

### 4.1 How to run the raw-data demo

#### Step-by-step setup

1. **Board A (RX)**: Flash `rx_raw_data_demo.ino` to the first Challenger RP2040 UWB (using the Arduino + UF2 workflow from Section ??).
2. Open the Arduino **Serial Monitor** for Board A with the baudrate used in the sketch (e.g. 115200 Bd).
3. **Board B (TX)**: Flash a TX example sketch, e.g. `ex_01a_simple_tx.ino` from the iLabs examples.<sup>a</sup>
4. Power both boards. Board B periodically sends UWB frames; Board A prints one raw dump for every received frame.

<sup>a</sup>[https://github.com/Pontus0/iLabs\\_UWB/blob/main/examples/ex\\_01a\\_simple\\_tx/ex\\_01a\\_simple\\_tx.ino](https://github.com/Pontus0/iLabs_UWB/blob/main/examples/ex_01a_simple_tx/ex_01a_simple_tx.ino)

### 4.2 Example serial output

The TX board simply reports that frames are being sent, while the RX board prints a full raw dump for each frame (status, frame bytes, RX timestamp, CIR, power values, . . . ).

```
20:00:02.216 -> TX Frame Sent
20:00:02.742 -> TX Frame Sent
20:00:03.305 -> TX Frame Sent
20:00:03.831 -> TX Frame Sent
20:00:04.359 -> TX Frame Sent
20:00:04.887 -> TX Frame Sent
20:00:05.445 -> TX Frame Sent
20:00:05.976 -> TX Frame Sent
20:00:06.504 -> TX Frame Sent
20:00:07.032 -> TX Frame Sent
20:00:07.555 -> TX Frame Sent
20:00:08.118 -> TX Frame Sent
20:00:08.649 -> TX Frame Sent
```

```
20:00:54.562 -> === RX FRAME ===
20:00:54.562 -> SYS_STATUS (low 32b): 0x1806F07
20:00:54.562 -> Frame length: 12
20:00:54.562 -> Frame bytes: C5 3B 44 45 43 41 57 41 56 45 05 02
20:00:54.562 -> RX timestamp (ticks): 2006772321
20:00:54.562 -> RX timestamp (full 40b, hex): 0x779CEA61
20:00:54.562 -> CIR first 128 bytes:
20:00:54.595 -> AD 0A 00 00 17 00 00 F2 FF FF E7 FF FF 12 00 00 F0 FF FF 48 00
20:00:54.595 -> ipatovPeak: 1572883075
20:00:54.595 -> ipatovPower: 46
20:00:54.595 -> ipatovFpIndex: 47146
20:00:54.595 -> stsPeak: 0
20:00:54.595 -> stsPower: 0
20:00:54.595 -> stsFpIndex: 0
20:00:54.595 -> xtalOffset: 56
20:00:54.595 -> === END FRAME ===
```

Figure 5: \*

TX board: simple “TX  
Frame Sent” log from  
`ex_01a_simple_tx.ino`.

Figure 6: \*

RX board: raw frame dump from  
`rx_raw_data_demo.ino` (timestamps, CIR,  
diagnostics).

Figure 7: Serial output of the TX example (left) and the raw-data receiver demo (right).

### 4.3 Sketch: rx\_raw\_data\_demo.ino

The sketch `rx_raw_data_demo.ino` configures the Challenger RP2040 UWB as a receiver and prints the raw information from every incoming UWB frame (timestamps, frame bytes, CIR, diagnostics, ...) to the serial console.

```

1 #include "dw3000.h"
2
3 #define APP_NAME "RX RAW DATA DEMO"
4
5 // --- Basic UWB config (same as examples) ---
6 static dwt_config_t config = {
7     5,                      // Channel
8     DWT_PLEN_128,          // Preamble length
9     DWT_PAC8,              // PAC size
10    9,                     // TX preamble code
11    9,                     // RX preamble code
12    1,                     // SFD type
13    DWT_BR_6M8,           // Data rate
14    DWT_PHRMODE_STD,      // PHY header mode
15    DWT_PHRRATE_STD,      // PHY header rate
16    (129 + 8 - 8),        // SFD timeout
17    DWT_STS_MODE_OFF,
18    DWT_STS_LEN_64,
19    DWT_PDOA_MO
20 };
21
22 // Antenna delays (same as Qorvo examples)
23 #define TX_ANT_DLY 16385
24 #define RX_ANT_DLY 16385
25
26 // RX buffer
27 #define RX_BUF_LEN 127
28 static uint8_t rx_buffer[RX_BUF_LEN];
29
30 // CIR buffer: just grab first 128 bytes as a demo
31 #define CIR_BUF_LEN 128
32 static uint8_t cir_buf[CIR_BUF_LEN];
33
34 // Diagnostics struct from DW3000 API
35 static dwt_rxdiag_t rx_diag;
36
37 // For status flags
38 static uint32_t status_reg = 0;
39
40 // From library (TX power etc.)
41 extern dwt_txconfig_t txconfig_options;
42
43 // Print bytes as hex
44 static void print_bytes_hex(const uint8_t *buf, uint16_t len)

```

```
45 {
46     for (uint16_t i = 0; i < len; i++) {
47         if (buf[i] < 0x10) Serial.print('0');
48         Serial.print(buf[i], HEX);
49         Serial.print(' ');
50     }
51     Serial.println();
52 }
53
54 void setup()
55 {
56     Serial.begin(115200);
57     delay(1000);
58     Serial.println();
59     Serial.println(APP_NAME);
60
61     // Start SPI + DW3000
62     spiBegin();
63     spiSelect();
64     delay(200);
65
66     while (!dwt_checkidlerc()) {
67         Serial.println("IDLE FAILED");
68     }
69
70     dwt_softreset();
71     delay(200);
72
73     while (!dwt_checkidlerc()) {
74         Serial.println("IDLE FAILED (after reset)");
75     }
76
77     if (dwt_initialise(DWT_DW_INIT) == DWT_ERROR) {
78         Serial.println("INIT FAILED");
79         while (1);
80     }
81
82     if (dwt_configure(&config)) {
83         Serial.println("CONFIG FAILED");
84         while (1);
85     }
86
87     dwt_configuretxrf(&txconfig_options);
88
89     dwt_setrxantennadelay(RX_ANT_DLY);
90     dwt_settxantennadelay(TX_ANT_DLY);
91
92     // Enable LNA/PA
93     dwt_setlnapemode(DWT_LNA_ENABLE | DWT_PA_ENABLE);
```

```

94
95 // Enable CIA diagnostics so dwt_readdiagnostics() gives full data
96 // (1 = log full diagnostic set, see DW3xxx API Guide)
97 dwt_configciadiag(1);
98
99 Serial.println("Ready, waiting for RX frames...");
100 }
101
102 void loop()
103 {
104 // --- Put DW3000 into RX mode immediately ---
105 dwt_rxenable(DWT_START_RX_IMMEDIATE);
106
107 // Wait for good frame or RX error
108 while (!(status_reg = dwt_read32bitreg(SYS_STATUS_ID)) &
109         (SYS_STATUS_RXFCG_BIT_MASK | SYS_STATUS_ALL_RX_ERR))) {
110     // busy wait
111 }
112
113 if (status_reg & SYS_STATUS_RXFCG_BIT_MASK) {
114     // ----- 1) STATUS / FLAGS -----
115     Serial.println();
116     Serial.println("== RX FRAME ==");
117     Serial.print("SYS_STATUS (low 32b): 0x");
118     Serial.println(status_reg, HEX);
119
120     // Clear good RX flag
121     dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG_BIT_MASK);
122
123     // ----- 2) FRAME BUFFER -----
124     uint32_t frame_len = dwt_read32bitreg(RX_FINFO_ID) & RXFLEN_MASK;
125     if (frame_len > RX_BUF_LEN) frame_len = RX_BUF_LEN;
126
127     dwt_readrxdata(rx_buffer, frame_len, 0);
128
129     Serial.print("Frame length: ");
130     Serial.println(frame_len);
131
132     Serial.print("Frame bytes: ");
133     print_bytes_hex(rx_buffer, frame_len);
134
135     // ----- 3) RX TIMESTAMP -----
136     uint64_t rx_ts = get_rx_timestamp_u64();
137     Serial.print("RX timestamp (ticks): ");
138     Serial.println((unsigned long)(rx_ts & 0xFFFFFFFFFUL)); // low 32 bits
139     Serial.print("RX timestamp (full 40b, hex): 0x");
140     Serial.println((unsigned long)(rx_ts & 0xFFFFFFFFFUL), HEX);
141
142     // ----- 4) CIR ACCUMULATOR (first bytes) -----

```

```
143     dwt_readaccdata(cir_buf, CIR_BUF_LEN, 0); // read from accumulator mem
144
145     Serial.print("CIR first ");
146     Serial.print(CIR_BUF_LEN);
147     Serial.println(" bytes:");
148     print_bytes_hex(cir_buf, CIR_BUF_LEN);
149
150     // ----- 5) RX DIAGNOSTICS / QUALITY -----
151     dwt_readdiagnostics(&rx_diag);
152
153     Serial.print("ipatovPeak:    ");
154     Serial.println(rx_diag.ipatovPeak);
155     Serial.print("ipatovPower:   ");
156     Serial.println(rx_diag.ipatovPower);
157     Serial.print("ipatovFpIndex:");
158     Serial.println(rx_diag.ipatovFpIndex);
159     Serial.print("stsPeak:        ");
160     Serial.println(rx_diag.stsPeak);
161     Serial.print("stsPower:       ");
162     Serial.println(rx_diag.stsPower);
163     Serial.print("stsFpIndex:    ");
164     Serial.println(rx_diag.stsFpIndex);
165     Serial.print("xtalOffset:    ");
166     Serial.println(rx_diag.xtalOffset);
167
168     Serial.println("== END FRAME ==");
169
170 } else {
171     // RX error → clear flags and try again
172     Serial.println("RX ERROR");
173     dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_ERR);
174 }
175
176 delay(100); // small pause between RX cycles
177 }
```

## 5 Example: Distance Measurement Between Two Nodes

For a first end-to-end distance test we used the single-sided two-way ranging (SS TWR) examples from the iLabs UWB library:

- **Node A (initiator)**: ex\_06a\_ss\_twr\_initiator.ino<sup>8</sup>
- **Node B (responder)**: ex\_06b\_ss\_twr\_responder.ino<sup>9</sup>

The responder replies to each poll frame, and the initiator computes the time-of-flight and prints the resulting distance in meters on the serial console (Figure ??).

```
19:30:23.990 -> SS TWR INIT v1.0
19:30:24.417 -> PLL is locked..
19:30:24.417 -> PGF cal complete..
19:32:08.533 -> DIST: 2.04 m
19:32:10.545 -> DIST: 2.06 m
19:32:12.524 -> DIST: 2.02 m
19:32:13.544 -> DIST: 2.00 m
19:32:14.535 -> DIST: 2.00 m
19:32:16.549 -> DIST: 2.00 m
19:32:18.527 -> DIST: 1.97 m
19:32:20.538 -> DIST: 1.94 m
19:32:23.542 -> DIST: 1.99 m
19:32:24.531 -> DIST: 2.01 m
19:32:25.554 -> DIST: 2.02 m
19:32:28.559 -> DIST: 2.01 m
19:32:32.550 -> DIST: 2.06 m
19:32:33.540 -> DIST: 2.08 m
19:32:34.560 -> DIST: 2.06 m
```

Figure 8: Serial output of ex\_06a\_ss\_twr\_initiator.ino showing repeated distance estimates between the two nodes.

<sup>8</sup>[https://github.com/Pontus0/iLabs\\_UWB/blob/main/examples/ex\\_06a\\_ss\\_twr\\_initiator/ex\\_06a\\_ss\\_twr\\_initiator.ino](https://github.com/Pontus0/iLabs_UWB/blob/main/examples/ex_06a_ss_twr_initiator/ex_06a_ss_twr_initiator.ino)

<sup>9</sup>[https://github.com/Pontus0/iLabs\\_UWB/blob/main/examples/ex\\_06b\\_ss\\_twr\\_responder/ex\\_06b\\_ss\\_twr\\_responder.ino](https://github.com/Pontus0/iLabs_UWB/blob/main/examples/ex_06b_ss_twr_responder/ex_06b_ss_twr_responder.ino)