|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | **Protokół projektu układu testującego lokalizację trilateracją w systemie Człowiek za burtą przy użyciu UWB** | | | | | | Data wystawienia: | |
|  | 01/09/2021 | |
|  | Doc# | 3/CZB/004 |
|  | Nr wniosku NCBR: | | POIR.01.01.01-00-0196/19 | | | Nazwa projektu: | | Smart Yacht |
|  | Rozpoczęcie testów: | | 29-07-2021 | | Zakończenie testów: | | 01-09-2021 | |

#### Założenia

Celem jest zbudowanie układu testującego metodę trilateracji UWB. Układ ma się składać z dwóch urządzeń- inicjatora wyposażonego w wyświetlacz LCD do wyświetlania dystansu oraz respondera- urządzenia mającego odpowiadać na ramkę inicjatora.

#### Hardware

* 1. Inicjator

Układ inicjatora składa się z płytki Nucleo-64 z mikrokontrolerem STM32F103 do której podłączony został shield DWS1000 wyposażony w moduł DWM1000 marki Quorvo. Całość została podłączona do wyświetlacza alfanumerycznego 16x2 przy pomocy płytki prototypowej. Całość może zostać zasilona z gniazda USB bądź z power banka.

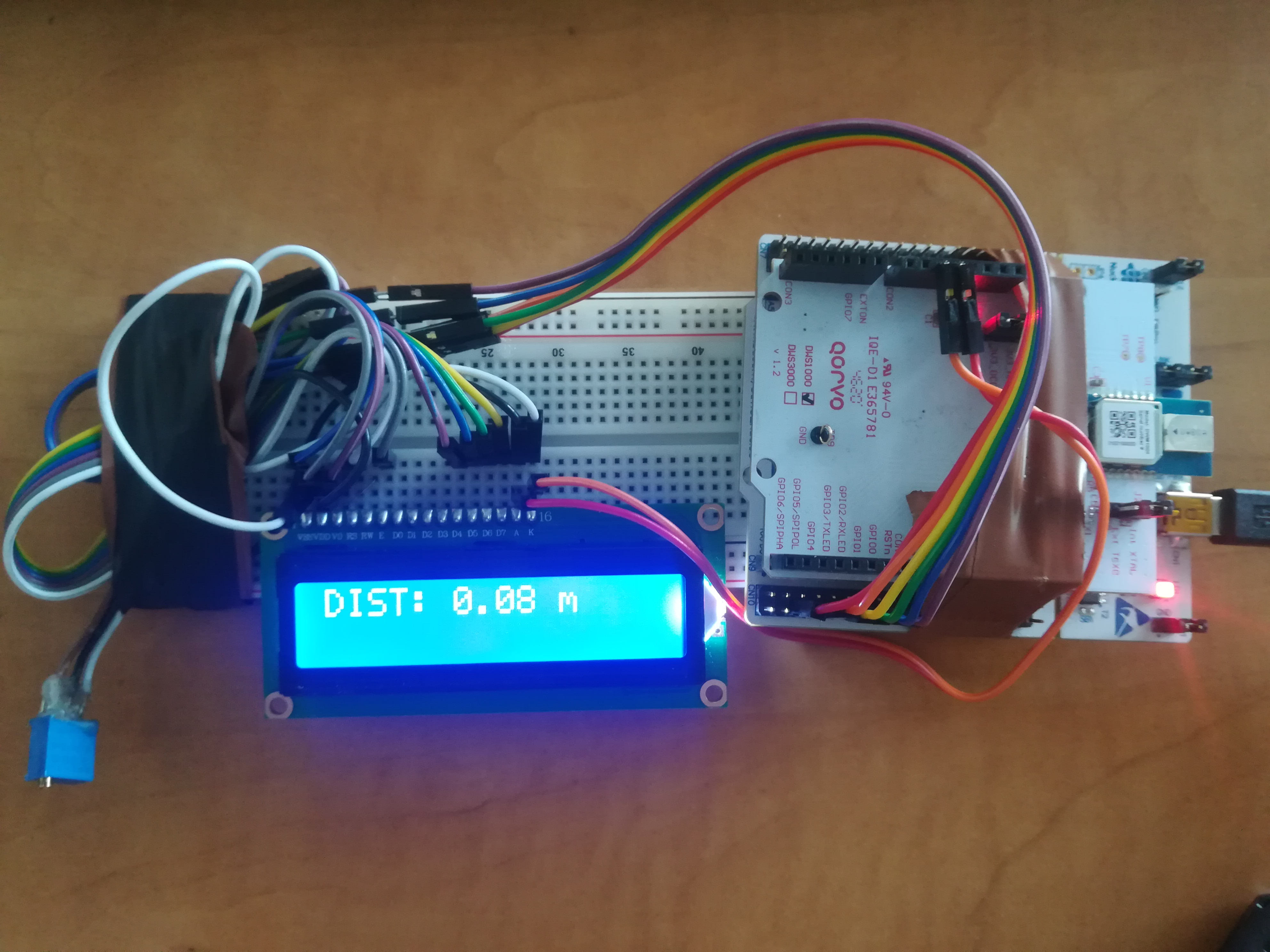
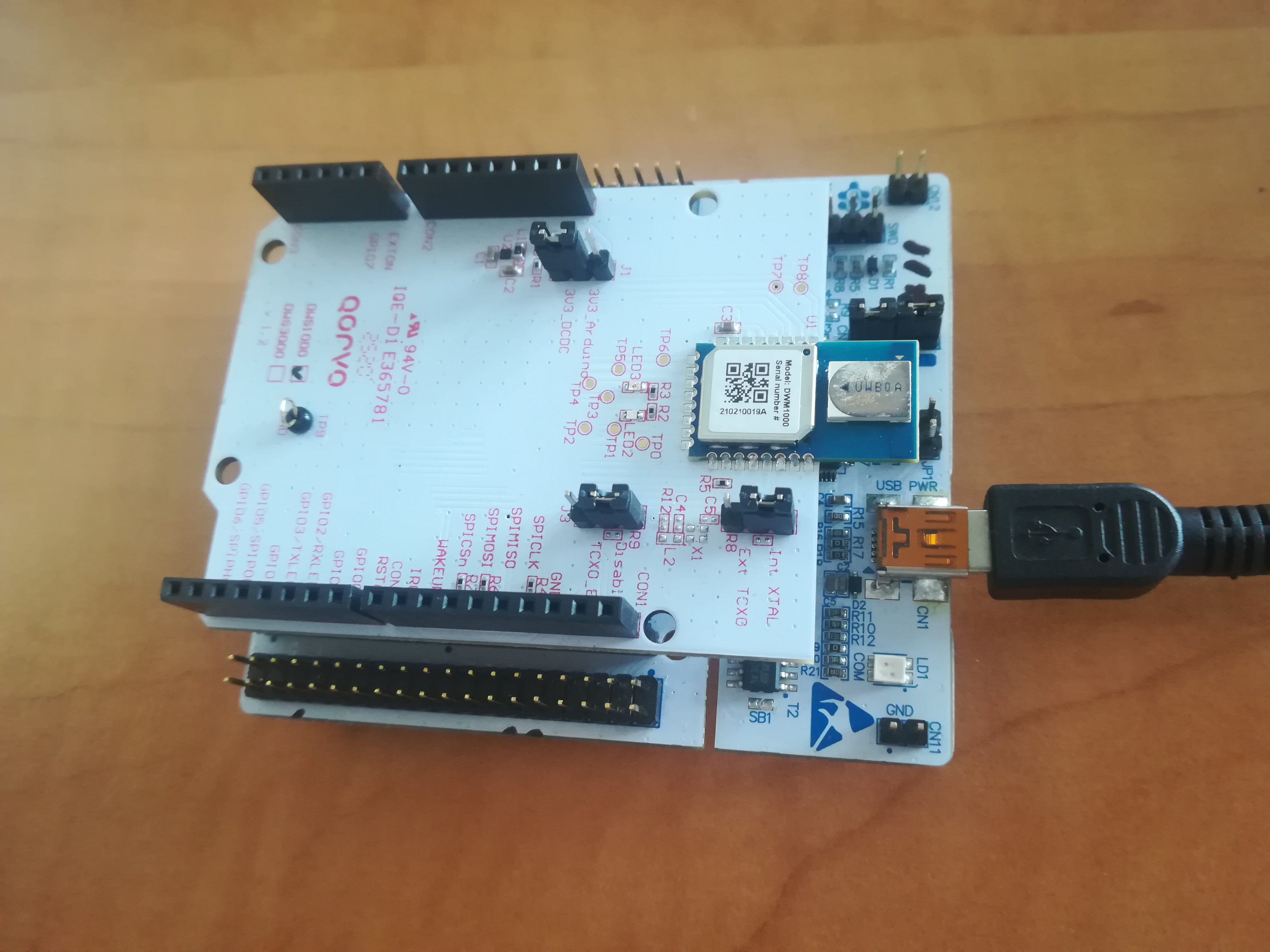


Figura 1: Układ inicjatora

* 1. Responder

Jako urządzenie lokalizowane posłużył analogiczny układ, składający się z płytki nucleo-64 z mikrokontrolerem STM32L010RBT wyposażony w ten sam shield(DWS1000).

Figura 2: Układ respondera

#### Software

Oprogramowanie testowe to kod przykładowy współpracujący z firmware od producenta układu UWB. Przykład został zmodyfikowany ze względu na użycie innych mikrokontrolerów niż w przykładnie.

Wybrane przykłady są częścią paczki DWS\_1000\_ExampleCode\_v1.0.1.

* 1. Inicjator

Kod inicjatora bazuje na przykładzie ex\_06a\_ss\_twr\_init z wprowadzonymi modyfikacjami.

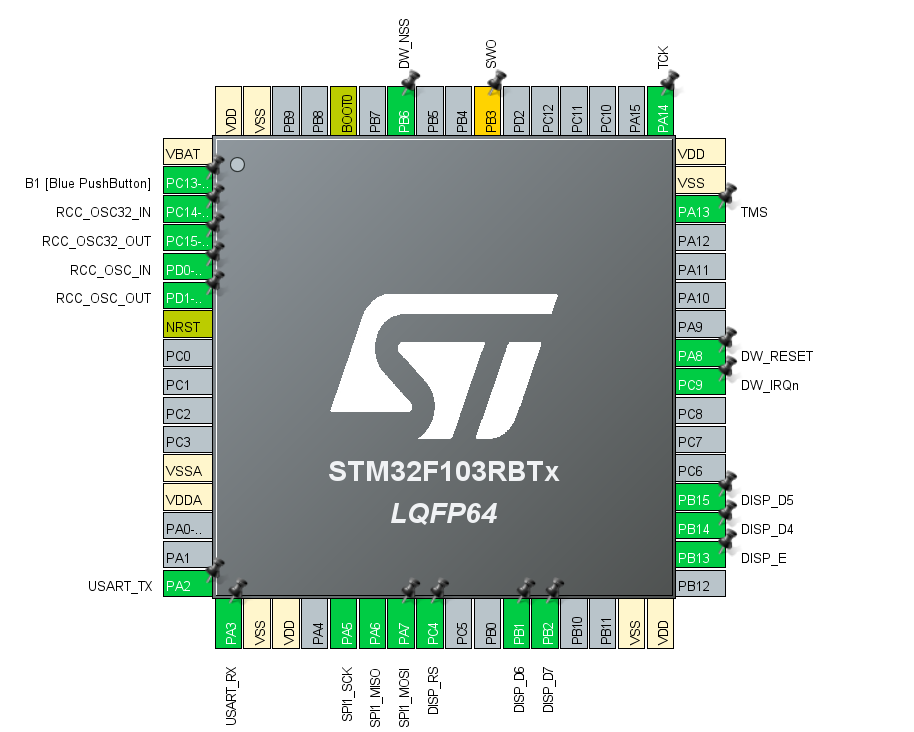
Dyrektywa RESP\_RX\_TIMEOUT\_UUS jest czasem oczekiwania na odpowiedź respondera. Został on zwiększony z 210 do 1000us.

Dyrektywa RNG\_DELAY\_MS jest interwałem pomiędzy wysyłanymi do respondera żądaniami. Został on zwiększony z 1000 do 2000ms.

Dyrektywa UUS\_TO\_DWT\_TIME jest mnożnikiem konwertującym czas. Ponieważ docelowy MCU jest wolniejszy niż ten w przykładzie, wartość została zmieniona z 65536 na 147456

Dodana została również obsługa wyświetlacza alfanumerycznego.

Pinout został ustawiony zgodnie z przykładem i shieldem DWS1000.

Figura 3: Pinout inicjatora

Kod główny(main.c):

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.

\* All rights reserved.</center></h2>

\*

\* This software component is licensed by ST under BSD 3-Clause license,

\* the "License"; You may not use this file except in compliance with the

\* License. You may obtain a copy of the License at:

\* opensource.org/licenses/BSD-3-Clause

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

**#include** "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

//main ex

**#include** "platform/port.h"

**app\_t** app;

//end of main ex

**#include** <stdio.h>

**#include** <string.h>

**#include** "decadriver/deca\_device\_api.h"

**#include** "decadriver/deca\_regs.h"

**#include** "platform/stdio.h"

**#include** "platform/deca\_spi.h"

**#include** "platform/port.h"

**#include** "../Display/an\_disp.h"

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

**SPI\_HandleTypeDef** hspi1;

**UART\_HandleTypeDef** huart2;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_USART2\_UART\_Init**(**void**);

**static** **void** **MX\_SPI1\_Init**(**void**);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* Example application name and version to display. \*/

**#define** **APP\_NAME** "SS TWR INIT v1.3\r\n"

/\* Inter-ranging delay period, in milliseconds. \*/

**#define** **RNG\_DELAY\_MS** 2000

/\* Default communication configuration. We use here EVK1000's mode 4. See NOTE 1 below. \*/

**static** **dwt\_config\_t** config = {

2, /\* Channel number. \*/

DWT\_PRF\_64M, /\* Pulse repetition frequency. \*/

DWT\_PLEN\_128, /\* Preamble length. Used in TX only. \*/

DWT\_PAC8, /\* Preamble acquisition chunk size. Used in RX only. \*/

9, /\* TX preamble code. Used in TX only. \*/

9, /\* RX preamble code. Used in RX only. \*/

0, /\* 0 to use standard SFD, 1 to use non-standard SFD. \*/

DWT\_BR\_6M8, /\* Data rate. \*/

DWT\_PHRMODE\_STD, /\* PHY header mode. \*/

(129 + 8 - 8) /\* SFD timeout (preamble length + 1 + SFD length - PAC size). Used in RX only. \*/

};

/\* Default antenna delay values for 64 MHz PRF. See NOTE 2 below. \*/

**#define** **TX\_ANT\_DLY** 16505

**#define** **RX\_ANT\_DLY** 16505

/\* Frames used in the ranging process. See NOTE 3 below. \*/

**static** **uint8** tx\_poll\_msg[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'W', 'A', 'V', 'E', 0xE0, 0, 0};

**static** **uint8** rx\_resp\_msg[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'V', 'E', 'W', 'A', 0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

/\* Length of the common part of the message (up to and including the function code, see NOTE 3 below). \*/

**#define** **ALL\_MSG\_COMMON\_LEN** 10

/\* Indexes to access some of the fields in the frames defined above. \*/

**#define** **ALL\_MSG\_SN\_IDX** 2

**#define** **RESP\_MSG\_POLL\_RX\_TS\_IDX** 10

**#define** **RESP\_MSG\_RESP\_TX\_TS\_IDX** 14

**#define** **RESP\_MSG\_TS\_LEN** 4

/\* Frame sequence number, incremented after each transmission. \*/

**static** **uint8** frame\_seq\_nb = 0;

/\* Buffer to store received response message.

\* Its size is adjusted to longest frame that this example code is supposed to handle. \*/

**#define** **RX\_BUF\_LEN** 20

**static** **uint8** rx\_buffer[RX\_BUF\_LEN];

/\* Hold copy of status register state here for reference so that it can be examined at a debug breakpoint. \*/

**static** **uint32** status\_reg = 0;

/\* UWB microsecond (uus) to device time unit (dtu, around 15.65 ps) conversion factor.

\* 1 uus = 512 / 499.2 µs and 1 µs = 499.2 \* 128 dtu. \*/

**#define** **UUS\_TO\_DWT\_TIME** 147456

/\* Delay between frames, in UWB microseconds. See NOTE 1 below. \*/

**#define** **POLL\_TX\_TO\_RESP\_RX\_DLY\_UUS** 140

/\* Receive response timeout. See NOTE 5 below. \*/

**#define** **RESP\_RX\_TIMEOUT\_UUS** 1000

/\* Speed of light in air, in metres per second. \*/

**#define** **SPEED\_OF\_LIGHT** 299702547

/\* Hold copies of computed time of flight and distance here for reference so that it can be examined at a debug breakpoint. \*/

**static** **double** tof;

**static** **double** distance;

/\* String used to display measured distance over UART. \*/

**char** dist\_str[16] = {0};

/\* Declaration of static functions. \*/

**static** **void** **resp\_msg\_get\_ts**(**uint8** \*ts\_field, **uint32** \*ts);

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

**int** **main**(**void**)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

**HAL\_Init**();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

**SystemClock\_Config**();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

**MX\_GPIO\_Init**();

**MX\_USART2\_UART\_Init**();

**MX\_SPI1\_Init**();

/\* USER CODE BEGIN 2 \*/

//main ex

**setup\_DW1000RSTnIRQ**(0);

**stdio\_init**(&huart2);

// HAL\_TIM\_Base\_Init(&htim1);

//end of my ex

/\* Display application name. \*/

**stdio\_write**(APP\_NAME);

**stdio\_write**("\033[s"); // Save cursor position

/\* Reset and initialise DW1000.

\* For initialisation, DW1000 clocks must be temporarily set to crystal speed. After initialisation SPI rate can be increased for optimum

\* performance. \*/

**reset\_DW1000**(); /\* Target specific drive of RSTn line into DW1000 low for a period. \*/

**port\_set\_dw1000\_slowrate**();

**if** (**dwt\_initialise**(DWT\_LOADUCODE) == DWT\_ERROR)

{

**stdio\_write**("INIT FAILED");

**while** (1)

{ };

}

**port\_set\_dw1000\_fastrate**();

/\* Configure DW1000. See NOTE 6 below. \*/

**dwt\_configure**(&config);

/\* Apply default antenna delay value. See NOTE 2 below. \*/

**dwt\_setrxantennadelay**(RX\_ANT\_DLY);

**dwt\_settxantennadelay**(TX\_ANT\_DLY);

/\* Set expected response's delay and timeout. See NOTE 1 and 5 below.

\* As this example only handles one incoming frame with always the same delay and timeout, those values can be set here once for all. \*/

**dwt\_setrxaftertxdelay**(POLL\_TX\_TO\_RESP\_RX\_DLY\_UUS);

**dwt\_setrxtimeout**(RESP\_RX\_TIMEOUT\_UUS);

**lcdInit**();

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* Write frame data to DW1000 and prepare transmission. See NOTE 7 below. \*/

tx\_poll\_msg[ALL\_MSG\_SN\_IDX] = frame\_seq\_nb;

dwt\_write32bitreg(SYS\_STATUS\_ID, SYS\_STATUS\_TXFRS);

**dwt\_writetxdata**(**sizeof**(tx\_poll\_msg), tx\_poll\_msg, 0); /\* Zero offset in TX buffer. \*/

**dwt\_writetxfctrl**(**sizeof**(tx\_poll\_msg), 0, 1); /\* Zero offset in TX buffer, ranging. \*/

/\* Start transmission, indicating that a response is expected so that reception is enabled automatically after the frame is sent and the delay

\* set by dwt\_setrxaftertxdelay() has elapsed. \*/

**dwt\_starttx**(DWT\_START\_TX\_IMMEDIATE | DWT\_RESPONSE\_EXPECTED);

/\* We assume that the transmission is achieved correctly, poll for reception of a frame or error/timeout. See NOTE 8 below. \*/

**while** (!((status\_reg = dwt\_read32bitreg(SYS\_STATUS\_ID)) & (SYS\_STATUS\_RXFCG | SYS\_STATUS\_ALL\_RX\_TO | SYS\_STATUS\_ALL\_RX\_ERR)))

{ };

/\* Increment frame sequence number after transmission of the poll message (modulo 256). \*/

frame\_seq\_nb++;

**if** (status\_reg & SYS\_STATUS\_RXFCG)

{

**uint32** frame\_len;

/\* Clear good RX frame event in the DW1000 status register. \*/

dwt\_write32bitreg(SYS\_STATUS\_ID, SYS\_STATUS\_RXFCG);

/\* A frame has been received, read it into the local buffer. \*/

frame\_len = dwt\_read32bitreg(RX\_FINFO\_ID) & RX\_FINFO\_RXFLEN\_MASK;

**if** (frame\_len <= RX\_BUF\_LEN)

{

**dwt\_readrxdata**(rx\_buffer, frame\_len, 0);

}

/\* Check that the frame is the expected response from the companion "SS TWR responder" example.

\* As the sequence number field of the frame is not relevant, it is cleared to simplify the validation of the frame. \*/

rx\_buffer[ALL\_MSG\_SN\_IDX] = 0;

**if** (**memcmp**(rx\_buffer, rx\_resp\_msg, ALL\_MSG\_COMMON\_LEN) == 0)

{

**uint32** poll\_tx\_ts, resp\_rx\_ts, poll\_rx\_ts, resp\_tx\_ts;

**int32** rtd\_init, rtd\_resp;

**float** clockOffsetRatio ;

/\* Retrieve poll transmission and response reception timestamps. See NOTE 9 below. \*/

poll\_tx\_ts = **dwt\_readtxtimestamplo32**();

resp\_rx\_ts = **dwt\_readrxtimestamplo32**();

/\* Read carrier integrator value and calculate clock offset ratio. See NOTE 11 below. \*/

clockOffsetRatio = **dwt\_readcarrierintegrator**() \* (FREQ\_OFFSET\_MULTIPLIER \* HERTZ\_TO\_PPM\_MULTIPLIER\_CHAN\_2 / 1.0e6) ;

/\* Get timestamps embedded in response message. \*/

**resp\_msg\_get\_ts**(&rx\_buffer[RESP\_MSG\_POLL\_RX\_TS\_IDX], &poll\_rx\_ts);

**resp\_msg\_get\_ts**(&rx\_buffer[RESP\_MSG\_RESP\_TX\_TS\_IDX], &resp\_tx\_ts);

/\* Compute time of flight and distance, using clock offset ratio to correct for differing local and remote clock rates \*/

rtd\_init = resp\_rx\_ts - poll\_tx\_ts;

rtd\_resp = resp\_tx\_ts - poll\_rx\_ts;

tof = ((rtd\_init - rtd\_resp \* (1 - clockOffsetRatio)) / 2.0) \* DWT\_TIME\_UNITS;

distance = tof \* SPEED\_OF\_LIGHT;

//correction

**double** dist\_corr = (1/(distance\*3.7)) +0.27;

// distance = distance + dist\_corr;

/\* Display computed distance. \*/

**sprintf**(dist\_str, "DIST: %3.2f m \r\n", distance);

**stdio\_write**(dist\_str);

**lcdLocate**(0, 0);

**lcdStr**(dist\_str);

}

}

**else**

{

/\* Clear RX error/timeout events in the DW1000 status register. \*/

dwt\_write32bitreg(SYS\_STATUS\_ID, SYS\_STATUS\_ALL\_RX\_TO | SYS\_STATUS\_ALL\_RX\_ERR);

/\* Reset RX to properly reinitialise LDE operation. \*/

**dwt\_rxreset**();

}

/\* Execute a delay between ranging exchanges. \*/

**Sleep**(RNG\_DELAY\_MS);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

**void** **SystemClock\_Config**(**void**)

{

**RCC\_OscInitTypeDef** RCC\_OscInitStruct = {0};

**RCC\_ClkInitTypeDef** RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI\_DIV2;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL16;

**if** (**HAL\_RCC\_OscConfig**(&RCC\_OscInitStruct) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV2;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

**if** (**HAL\_RCC\_ClockConfig**(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != *HAL\_OK*)

{

**Error\_Handler**();

}

}

/\*\*

\* @brief SPI1 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_SPI1\_Init**(**void**)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_4;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 10;

**if** (**HAL\_SPI\_Init**(&hspi1) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_Init 2 \*/

}

/\*\*

\* @brief USART2 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_USART2\_UART\_Init**(**void**)

{

/\* USER CODE BEGIN USART2\_Init 0 \*/

/\* USER CODE END USART2\_Init 0 \*/

/\* USER CODE BEGIN USART2\_Init 1 \*/

/\* USER CODE END USART2\_Init 1 \*/

huart2.Instance = USART2;

huart2.Init.BaudRate = 115200;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B;

huart2.Init.StopBits = UART\_STOPBITS\_1;

huart2.Init.Parity = UART\_PARITY\_NONE;

huart2.Init.Mode = UART\_MODE\_TX\_RX;

huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

**if** (**HAL\_UART\_Init**(&huart2) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN USART2\_Init 2 \*/

/\* USER CODE END USART2\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_GPIO\_Init**(**void**)

{

**GPIO\_InitTypeDef** GPIO\_InitStruct = {0};

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

**HAL\_GPIO\_WritePin**(DISP\_RS\_GPIO\_Port, DISP\_RS\_Pin, *GPIO\_PIN\_RESET*);

/\*Configure GPIO pin Output Level \*/

**HAL\_GPIO\_WritePin**(GPIOB, DISP\_D6\_Pin|DISP\_D7\_Pin|DISP\_E\_Pin|DISP\_D4\_Pin

|DISP\_D5\_Pin, *GPIO\_PIN\_RESET*);

/\*Configure GPIO pin Output Level \*/

**HAL\_GPIO\_WritePin**(DW\_RESET\_GPIO\_Port, DW\_RESET\_Pin, *GPIO\_PIN\_RESET*);

/\*Configure GPIO pin Output Level \*/

**HAL\_GPIO\_WritePin**(DW\_NSS\_GPIO\_Port, DW\_NSS\_Pin, *GPIO\_PIN\_SET*);

/\*Configure GPIO pin : B1\_Pin \*/

GPIO\_InitStruct.Pin = B1\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_RISING;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

**HAL\_GPIO\_Init**(B1\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : DISP\_RS\_Pin \*/

GPIO\_InitStruct.Pin = DISP\_RS\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(DISP\_RS\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pins : DISP\_D6\_Pin DISP\_D7\_Pin DISP\_E\_Pin DISP\_D4\_Pin

DISP\_D5\_Pin DW\_NSS\_Pin \*/

GPIO\_InitStruct.Pin = DISP\_D6\_Pin|DISP\_D7\_Pin|DISP\_E\_Pin|DISP\_D4\_Pin

|DISP\_D5\_Pin|DW\_NSS\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(GPIOB, &GPIO\_InitStruct);

/\*Configure GPIO pin : DW\_IRQn\_Pin \*/

GPIO\_InitStruct.Pin = DW\_IRQn\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_RISING;

GPIO\_InitStruct.Pull = GPIO\_PULLDOWN;

**HAL\_GPIO\_Init**(DW\_IRQn\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : DW\_RESET\_Pin \*/

GPIO\_InitStruct.Pin = DW\_RESET\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_OD;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(DW\_RESET\_GPIO\_Port, &GPIO\_InitStruct);

/\* EXTI interrupt init\*/

**HAL\_NVIC\_SetPriority**(*EXTI15\_10\_IRQn*, 0, 0);

**HAL\_NVIC\_EnableIRQ**(*EXTI15\_10\_IRQn*);

}

/\* USER CODE BEGIN 4 \*/

/\*! ------------------------------------------------------------------------------------------------------------------

\* @fn resp\_msg\_get\_ts()

\*

\* @brief Read a given timestamp value from the response message. In the timestamp fields of the response message, the

\* least significant byte is at the lower address.

\*

\* @param ts\_field pointer on the first byte of the timestamp field to get

\* ts timestamp value

\*

\* @return none

\*/

**static** **void** **resp\_msg\_get\_ts**(**uint8** \*ts\_field, **uint32** \*ts)

{

**int** i;

\*ts = 0;

**for** (i = 0; i < RESP\_MSG\_TS\_LEN; i++)

{

\*ts += ts\_field[i] << (i \* 8);

}

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

**void** **Error\_Handler**(**void**)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

**\_\_disable\_irq**();

**while** (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

**#ifdef** USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

**void** assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

**#endif** /\* USE\_FULL\_ASSERT \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/

Biblioteka wyświetlacza(an\_disp.c):

/\*

\* an\_disp.c

\*

\* Created on: 21.08.2019

\* Author: Bartosz Pracz

\*/

**#include** "../Display/an\_disp.h"

**void** **lcdSendHalf**(**uint8\_t** data) {

LCD\_E\_HIGH;

**HAL\_GPIO\_WritePin**(LCD\_D4\_PORT, LCD\_D4\_PIN, (data & 0x01));

**HAL\_GPIO\_WritePin**(LCD\_D5\_PORT, LCD\_D5\_PIN, (data & 0x02));

**HAL\_GPIO\_WritePin**(LCD\_D6\_PORT, LCD\_D6\_PIN, (data & 0x04));

**HAL\_GPIO\_WritePin**(LCD\_D7\_PORT, LCD\_D7\_PIN, (data & 0x08));

LCD\_E\_LOW;

}

**void** **lcdWriteByte**(**uint8\_t** data) {

**lcdSendHalf**(data >> 4);

**lcdSendHalf**(data);

**HAL\_Delay**(1);

}

**void** **lcdWriteCmd**(**uint8\_t** cmd) {

LCD\_RS\_LOW;

**lcdWriteByte**(cmd);

}

**void** **lcdChar**(**char** data) {

LCD\_RS\_HIGH;

**lcdWriteByte**(data);

}

**void** **lcdInit**(**void**) {

**HAL\_Delay**(15);

LCD\_E\_LOW;

LCD\_RS\_LOW;

**lcdSendHalf**(0x03);

**HAL\_Delay**(5);

**lcdSendHalf**(0x03);

**HAL\_Delay**(5);

**lcdSendHalf**(0x03);

**HAL\_Delay**(5);

**lcdSendHalf**(0x02);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCD\_FUNC | LCD\_4\_BIT | LCDC\_TWO\_LINE | LCDC\_FONT\_5x7);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCD\_ONOFF | LCD\_DISP\_ON);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCD\_CLEAR);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCDC\_ENTRY\_MODE | LCD\_EM\_SHIFT\_CURSOR | LCD\_EM\_RIGHT);

**HAL\_Delay**(5);

}

**void** **lcdLocate**(**uint8\_t** x, **uint8\_t** y) {

**switch** (y) {

**case** 0:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE1 + x));

**break**;

**case** 1:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE2 + x));

**break**;

**case** 2:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE3 + (x - 12)));

**break**;

**case** 3:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE4 + (x - 12)));

**break**;

}

}

**void** **lcdStr**(**char** \*text) {

**while** (\*text)

**lcdChar**(\*text++);

}

**void** **lcdInt**(**int** data){

**char** buffer[20];

**sprintf**(buffer, "%d", data);

**lcdStr**(buffer);

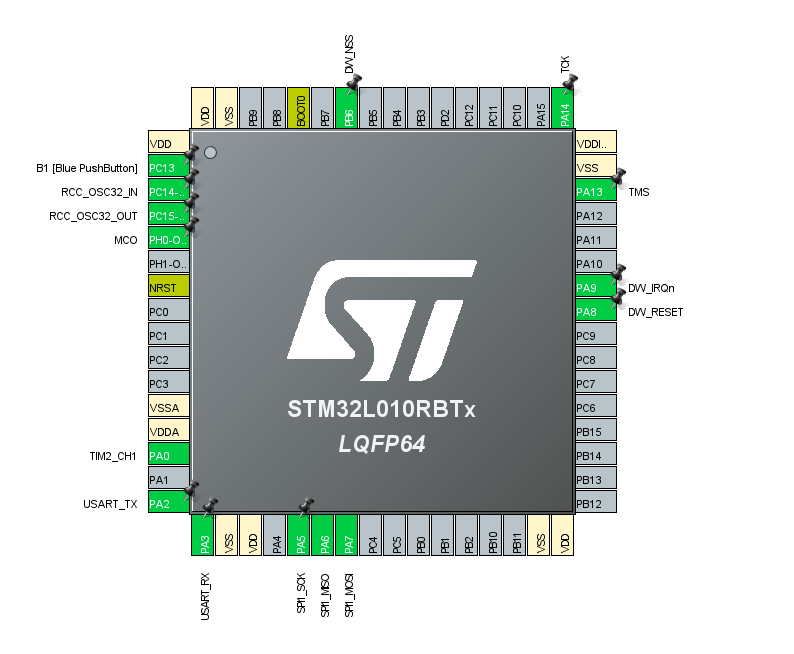
}

* 1. Responder

Kod inicjatora bazuje na przykładzie ex\_06b\_ss\_twr\_resp z wprowadzony mi modyfikacjami.

Dyrektywa UUS\_TO\_DWT\_TIME jest mnożnikiem konwertującym czas. Ponieważ docelowy MCU jest wolniejszy niż ten w przykładzie, wartość została zmieniona z 65536 na 147456

Pinout został ustawiony zgodnie z przykładem.

Figura 4: Pinout respondera

Kod główny(main.c):

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.

\* All rights reserved.</center></h2>

\*

\* This software component is licensed by ST under BSD 3-Clause license,

\* the "License"; You may not use this file except in compliance with the

\* License. You may obtain a copy of the License at:

\* opensource.org/licenses/BSD-3-Clause

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

**#include** "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

//main ex

**#include** "platform/port.h"

**app\_t** app;

//end of main ex

**#include** <string.h>

**#include** "decadriver/deca\_device\_api.h"

**#include** "decadriver/deca\_regs.h"

**#include** "platform/stdio.h"

**#include** "platform/deca\_spi.h"

**#include** "platform/port.h"

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

**SPI\_HandleTypeDef** hspi1;

**TIM\_HandleTypeDef** htim2;

**UART\_HandleTypeDef** huart2;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_SPI1\_Init**(**void**);

**static** **void** **MX\_TIM2\_Init**(**void**);

**static** **void** **MX\_USART2\_UART\_Init**(**void**);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* Example application name and version to display. \*/

**#define** **APP\_NAME** "SS TWR RESP v1.2"

/\* Default communication configuration. We use here EVK1000's mode 4. See NOTE 1 below. \*/

**static** **dwt\_config\_t** config = {

2, /\* Channel number. \*/

DWT\_PRF\_64M, /\* Pulse repetition frequency. \*/

DWT\_PLEN\_128, /\* Preamble length. Used in TX only. \*/

DWT\_PAC8, /\* Preamble acquisition chunk size. Used in RX only. \*/

9, /\* TX preamble code. Used in TX only. \*/

9, /\* RX preamble code. Used in RX only. \*/

0, /\* 0 to use standard SFD, 1 to use non-standard SFD. \*/

DWT\_BR\_6M8, /\* Data rate. \*/

DWT\_PHRMODE\_STD, /\* PHY header mode. \*/

(129 + 8 - 8) /\* SFD timeout (preamble length + 1 + SFD length - PAC size). Used in RX only. \*/

};

/\* Default antenna delay values for 64 MHz PRF. See NOTE 2 below. \*/

**#define** **TX\_ANT\_DLY** 16505

**#define** **RX\_ANT\_DLY** 16505

/\* Frames used in the ranging process. See NOTE 3 below. \*/

**static** **uint8** rx\_poll\_msg[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'W', 'A', 'V', 'E', 0xE0, 0, 0};

**static** **uint8** tx\_resp\_msg[] = {0x41, 0x88, 0, 0xCA, 0xDE, 'V', 'E', 'W', 'A', 0xE1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

/\* Length of the common part of the message (up to and including the function code, see NOTE 3 below). \*/

**#define** **ALL\_MSG\_COMMON\_LEN** 10

/\* Index to access some of the fields in the frames involved in the process. \*/

**#define** **ALL\_MSG\_SN\_IDX** 2

**#define** **RESP\_MSG\_POLL\_RX\_TS\_IDX** 10

**#define** **RESP\_MSG\_RESP\_TX\_TS\_IDX** 14

**#define** **RESP\_MSG\_TS\_LEN** 4

/\* Frame sequence number, incremented after each transmission. \*/

**static** **uint8** frame\_seq\_nb = 0;

/\* Buffer to store received messages.

\* Its size is adjusted to longest frame that this example code is supposed to handle. \*/

**#define** **RX\_BUF\_LEN** 12

**static** **uint8** rx\_buffer[RX\_BUF\_LEN];

/\* Hold copy of status register state here for reference so that it can be examined at a debug breakpoint. \*/

**static** **uint32** status\_reg = 0;

/\* UWB microsecond (uus) to device time unit (dtu, around 15.65 ps) conversion factor.

\* 1 uus = 512 / 499.2 µs and 1 µs = 499.2 \* 128 dtu. \*/

**#define** **UUS\_TO\_DWT\_TIME** 147456

/\* Delay between frames, in UWB microseconds. See NOTE 1 below. \*/

**#define** **POLL\_RX\_TO\_RESP\_TX\_DLY\_UUS** 330

/\* Timestamps of frames transmission/reception.

\* As they are 40-bit wide, we need to define a 64-bit int type to handle them. \*/

**typedef** **unsigned** **long** **long** **uint64**;

**static** **uint64** poll\_rx\_ts;

**static** **uint64** resp\_tx\_ts;

/\* Declaration of static functions. \*/

**static** **uint64** **get\_rx\_timestamp\_u64**(**void**);

**static** **void** **resp\_msg\_set\_ts**(**uint8** \*ts\_field, **const** **uint64** ts);

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

**int** **main**(**void**)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

**HAL\_Init**();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

**SystemClock\_Config**();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

**MX\_GPIO\_Init**();

**MX\_SPI1\_Init**();

**MX\_TIM2\_Init**();

**MX\_USART2\_UART\_Init**();

/\* USER CODE BEGIN 2 \*/

//main ex

**setup\_DW1000RSTnIRQ**(0);

**stdio\_init**(&huart2);

**HAL\_TIM\_Base\_Init**(&htim2);

//end of my ex

/\* Display application name. \*/

**stdio\_write**(APP\_NAME);

/\* Reset and initialise DW1000.

\* For initialisation, DW1000 clocks must be temporarily set to crystal speed. After initialisation SPI rate can be increased for optimum

\* performance. \*/

**reset\_DW1000**(); /\* Target specific drive of RSTn line into DW1000 low for a period. \*/

**port\_set\_dw1000\_slowrate**();

**if** (**dwt\_initialise**(DWT\_LOADUCODE) == DWT\_ERROR)

{

**stdio\_write**("INIT FAILED");

**while** (1)

{ };

}

**port\_set\_dw1000\_fastrate**();

/\* Configure DW1000. See NOTE 5 below. \*/

**dwt\_configure**(&config);

/\* Apply default antenna delay value. See NOTE 2 below. \*/

**dwt\_setrxantennadelay**(RX\_ANT\_DLY);

**dwt\_settxantennadelay**(TX\_ANT\_DLY);

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1) {

/\* Activate reception immediately. \*/

**dwt\_rxenable**(DWT\_START\_RX\_IMMEDIATE);

/\* Poll for reception of a frame or error/timeout. See NOTE 6 below. \*/

**while** (!((status\_reg = dwt\_read32bitreg(SYS\_STATUS\_ID)) & (SYS\_STATUS\_RXFCG | SYS\_STATUS\_ALL\_RX\_ERR)))

{ };

**if** (status\_reg & SYS\_STATUS\_RXFCG)

{

**uint32** frame\_len;

/\* Clear good RX frame event in the DW1000 status register. \*/

dwt\_write32bitreg(SYS\_STATUS\_ID, SYS\_STATUS\_RXFCG);

/\* A frame has been received, read it into the local buffer. \*/

frame\_len = dwt\_read32bitreg(RX\_FINFO\_ID) & RX\_FINFO\_RXFL\_MASK\_1023;

**if** (frame\_len <= RX\_BUFFER\_LEN)

{

**dwt\_readrxdata**(rx\_buffer, frame\_len, 0);

}

/\* Check that the frame is a poll sent by "SS TWR initiator" example.

\* As the sequence number field of the frame is not relevant, it is cleared to simplify the validation of the frame. \*/

rx\_buffer[ALL\_MSG\_SN\_IDX] = 0;

**if** (**memcmp**(rx\_buffer, rx\_poll\_msg, ALL\_MSG\_COMMON\_LEN) == 0)

{

**uint32** resp\_tx\_time;

**int** ret;

/\* Retrieve poll reception timestamp. \*/

poll\_rx\_ts = **get\_rx\_timestamp\_u64**();

/\* Compute final message transmission time. See NOTE 7 below. \*/

resp\_tx\_time = (poll\_rx\_ts + (POLL\_RX\_TO\_RESP\_TX\_DLY\_UUS \* UUS\_TO\_DWT\_TIME)) >> 8;

**dwt\_setdelayedtrxtime**(resp\_tx\_time);

/\* Response TX timestamp is the transmission time we programmed plus the antenna delay. \*/

resp\_tx\_ts = (((**uint64**)(resp\_tx\_time & 0xFFFFFFFEUL)) << 8) + TX\_ANT\_DLY;

/\* Write all timestamps in the final message. See NOTE 8 below. \*/

**resp\_msg\_set\_ts**(&tx\_resp\_msg[RESP\_MSG\_POLL\_RX\_TS\_IDX], poll\_rx\_ts);

**resp\_msg\_set\_ts**(&tx\_resp\_msg[RESP\_MSG\_RESP\_TX\_TS\_IDX], resp\_tx\_ts);

/\* Write and send the response message. See NOTE 9 below. \*/

tx\_resp\_msg[ALL\_MSG\_SN\_IDX] = frame\_seq\_nb;

**dwt\_writetxdata**(**sizeof**(tx\_resp\_msg), tx\_resp\_msg, 0); /\* Zero offset in TX buffer. \*/

**dwt\_writetxfctrl**(**sizeof**(tx\_resp\_msg), 0, 1); /\* Zero offset in TX buffer, ranging. \*/

ret = **dwt\_starttx**(DWT\_START\_TX\_DELAYED);

/\* If dwt\_starttx() returns an error, abandon this ranging exchange and proceed to the next one. See NOTE 10 below. \*/

**if** (ret == DWT\_SUCCESS)

{

/\* Poll DW1000 until TX frame sent event set. See NOTE 6 below. \*/

**while** (!(dwt\_read32bitreg(SYS\_STATUS\_ID) & SYS\_STATUS\_TXFRS))

{ };

/\* Clear TXFRS event. \*/

dwt\_write32bitreg(SYS\_STATUS\_ID, SYS\_STATUS\_TXFRS);

/\* Increment frame sequence number after transmission of the poll message (modulo 256). \*/

frame\_seq\_nb++;

}

}

}

**else**

{

/\* Clear RX error events in the DW1000 status register. \*/

dwt\_write32bitreg(SYS\_STATUS\_ID, SYS\_STATUS\_ALL\_RX\_ERR);

/\* Reset RX to properly reinitialise LDE operation. \*/

**dwt\_rxreset**();

}

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

**void** **SystemClock\_Config**(**void**)

{

**RCC\_OscInitTypeDef** RCC\_OscInitStruct = {0};

**RCC\_ClkInitTypeDef** RCC\_ClkInitStruct = {0};

**RCC\_PeriphCLKInitTypeDef** PeriphClkInit = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLLMUL\_4;

RCC\_OscInitStruct.PLL.PLLDIV = RCC\_PLLDIV\_2;

**if** (**HAL\_RCC\_OscConfig**(&RCC\_OscInitStruct) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

**if** (**HAL\_RCC\_ClockConfig**(&RCC\_ClkInitStruct, FLASH\_LATENCY\_1) != *HAL\_OK*)

{

**Error\_Handler**();

}

PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_USART2;

PeriphClkInit.Usart2ClockSelection = RCC\_USART2CLKSOURCE\_PCLK1;

**if** (**HAL\_RCCEx\_PeriphCLKConfig**(&PeriphClkInit) != *HAL\_OK*)

{

**Error\_Handler**();

}

}

/\*\*

\* @brief SPI1 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_SPI1\_Init**(**void**)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_4;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 7;

**if** (**HAL\_SPI\_Init**(&hspi1) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_Init 2 \*/

}

/\*\*

\* @brief TIM2 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_TIM2\_Init**(**void**)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

**TIM\_SlaveConfigTypeDef** sSlaveConfig = {0};

**TIM\_MasterConfigTypeDef** sMasterConfig = {0};

**TIM\_OC\_InitTypeDef** sConfigOC = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/

htim2.Instance = TIM2;

htim2.Init.Prescaler = 0;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 65535;

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

**if** (**HAL\_TIM\_Base\_Init**(&htim2) != *HAL\_OK*)

{

**Error\_Handler**();

}

**if** (**HAL\_TIM\_OC\_Init**(&htim2) != *HAL\_OK*)

{

**Error\_Handler**();

}

sSlaveConfig.SlaveMode = TIM\_SLAVEMODE\_TRIGGER;

sSlaveConfig.InputTrigger = TIM\_TS\_ITR0;

**if** (**HAL\_TIM\_SlaveConfigSynchro**(&htim2, &sSlaveConfig) != *HAL\_OK*)

{

**Error\_Handler**();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

**if** (**HAL\_TIMEx\_MasterConfigSynchronization**(&htim2, &sMasterConfig) != *HAL\_OK*)

{

**Error\_Handler**();

}

sConfigOC.OCMode = TIM\_OCMODE\_TIMING;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

**if** (**HAL\_TIM\_OC\_ConfigChannel**(&htim2, &sConfigOC, TIM\_CHANNEL\_1) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

/\* USER CODE END TIM2\_Init 2 \*/

**HAL\_TIM\_MspPostInit**(&htim2);

}

/\*\*

\* @brief USART2 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_USART2\_UART\_Init**(**void**)

{

/\* USER CODE BEGIN USART2\_Init 0 \*/

/\* USER CODE END USART2\_Init 0 \*/

/\* USER CODE BEGIN USART2\_Init 1 \*/

/\* USER CODE END USART2\_Init 1 \*/

huart2.Instance = USART2;

huart2.Init.BaudRate = 115200;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B;

huart2.Init.StopBits = UART\_STOPBITS\_1;

huart2.Init.Parity = UART\_PARITY\_NONE;

huart2.Init.Mode = UART\_MODE\_TX\_RX;

huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart2.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart2.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

**if** (**HAL\_UART\_Init**(&huart2) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN USART2\_Init 2 \*/

/\* USER CODE END USART2\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_GPIO\_Init**(**void**)

{

**GPIO\_InitTypeDef** GPIO\_InitStruct = {0};

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

**HAL\_GPIO\_WritePin**(DW\_RESET\_GPIO\_Port, DW\_RESET\_Pin, *GPIO\_PIN\_RESET*);

/\*Configure GPIO pin Output Level \*/

**HAL\_GPIO\_WritePin**(DW\_NSS\_GPIO\_Port, DW\_NSS\_Pin, *GPIO\_PIN\_SET*);

/\*Configure GPIO pin : B1\_Pin \*/

GPIO\_InitStruct.Pin = B1\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_FALLING;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

**HAL\_GPIO\_Init**(B1\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : DW\_RESET\_Pin \*/

GPIO\_InitStruct.Pin = DW\_RESET\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_OD;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(DW\_RESET\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : DW\_IRQn\_Pin \*/

GPIO\_InitStruct.Pin = DW\_IRQn\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_RISING;

GPIO\_InitStruct.Pull = GPIO\_PULLDOWN;

**HAL\_GPIO\_Init**(DW\_IRQn\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : DW\_NSS\_Pin \*/

GPIO\_InitStruct.Pin = DW\_NSS\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(DW\_NSS\_GPIO\_Port, &GPIO\_InitStruct);

/\* EXTI interrupt init\*/

**HAL\_NVIC\_SetPriority**(*EXTI4\_15\_IRQn*, 0, 0);

**HAL\_NVIC\_EnableIRQ**(*EXTI4\_15\_IRQn*);

}

/\* USER CODE BEGIN 4 \*/

/\*! ------------------------------------------------------------------------------------------------------------------

\* @fn get\_rx\_timestamp\_u64()

\*

\* @brief Get the RX time-stamp in a 64-bit variable.

\* /!\ This function assumes that length of time-stamps is 40 bits, for both TX and RX!

\*

\* @param none

\*

\* @return 64-bit value of the read time-stamp.

\*/

**static** **uint64** **get\_rx\_timestamp\_u64**(**void**)

{

**uint8** ts\_tab[5];

**uint64** ts = 0;

**int** i;

**dwt\_readrxtimestamp**(ts\_tab);

**for** (i = 4; i >= 0; i--)

{

ts <<= 8;

ts |= ts\_tab[i];

}

**return** ts;

}

/\*! ------------------------------------------------------------------------------------------------------------------

\* @fn final\_msg\_set\_ts()

\*

\* @brief Fill a given timestamp field in the response message with the given value. In the timestamp fields of the

\* response message, the least significant byte is at the lower address.

\*

\* @param ts\_field pointer on the first byte of the timestamp field to fill

\* ts timestamp value

\*

\* @return none

\*/

**static** **void** **resp\_msg\_set\_ts**(**uint8** \*ts\_field, **const** **uint64** ts)

{

**int** i;

**for** (i = 0; i < RESP\_MSG\_TS\_LEN; i++)

{

ts\_field[i] = (ts >> (i \* 8)) & 0xFF;

}

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

**void** **Error\_Handler**(**void**)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

**\_\_disable\_irq**();

**while** (1) {

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

**#ifdef** USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

**void** assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

**#endif** /\* USE\_FULL\_ASSERT \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/

#### Wnioski

Układ działa poprawnie. Umożliwia prosty odczyt dytansu pomiędzy inicjatorem a responderem. Wynik odczytu jest pokazywany na wyświetlaczu, a każdy z układów może być zasilany z powerbanka, dzięki czemu możliwe będzie testowanie trilateracji UWB na łodzi bez użycia komputera.

Wykonał: Bartosz Pracz