Flotylla Sokólska	**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	

1. 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.

2. Hardware

*2.1 **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.

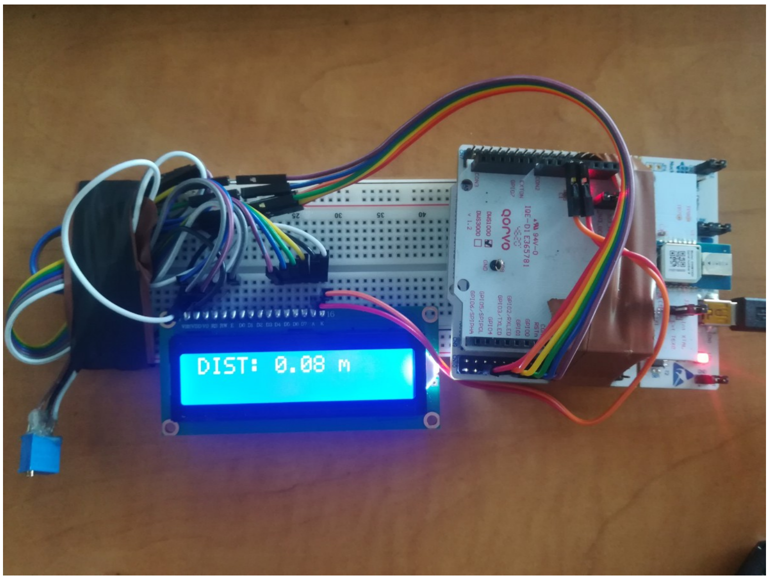

Figura 1: Układ inicjatora

2.2 ***Responder***

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

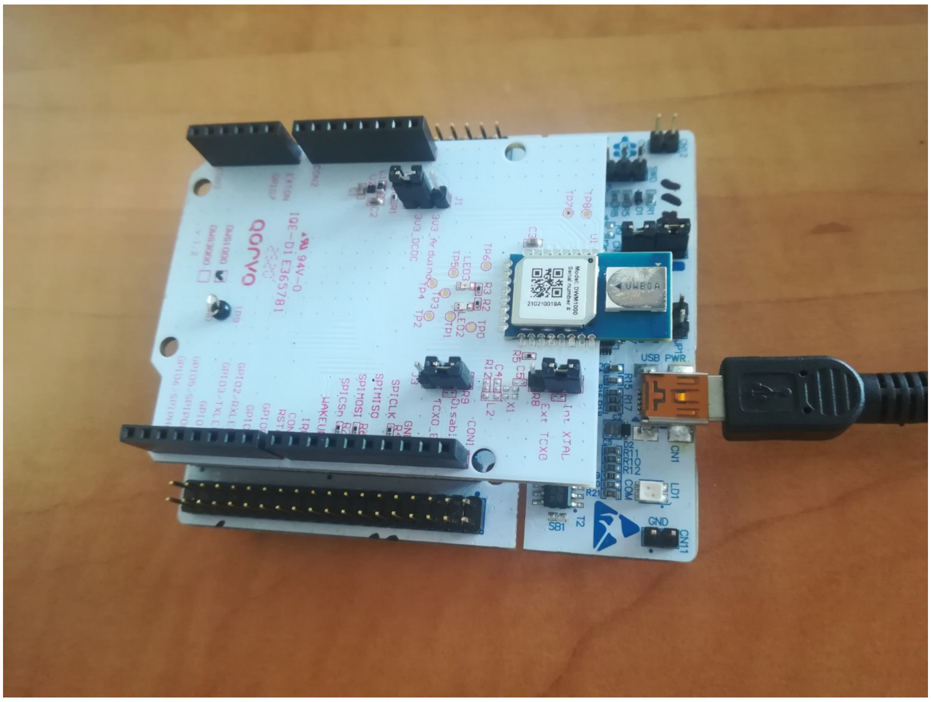

Figura 2: Układ respondera

3. 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.

3.1 ***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.

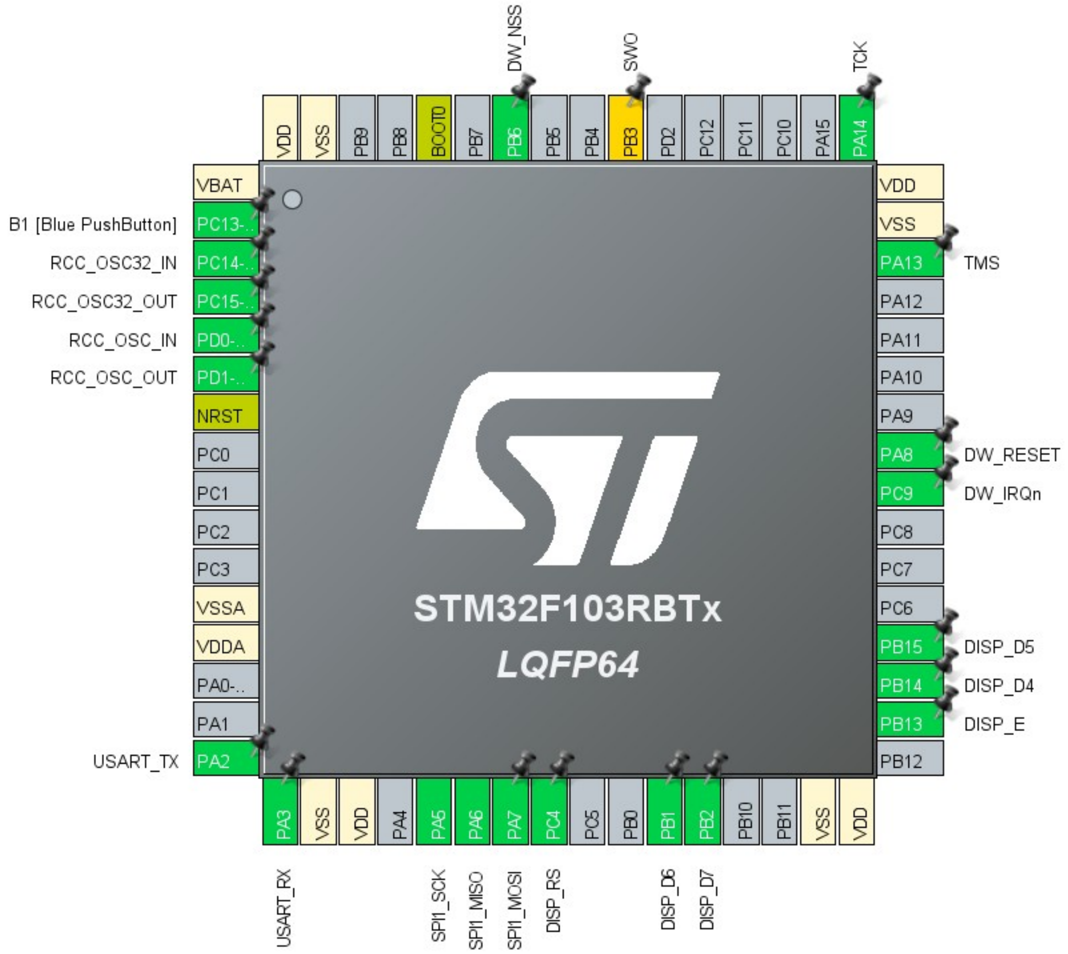

Figura 3: Pinout inicjatora

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 -----------------------------------------------------------*/
```

```c
/* 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
```

```c
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)
  {
```

```c
    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);

}
```

3.2 ***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.

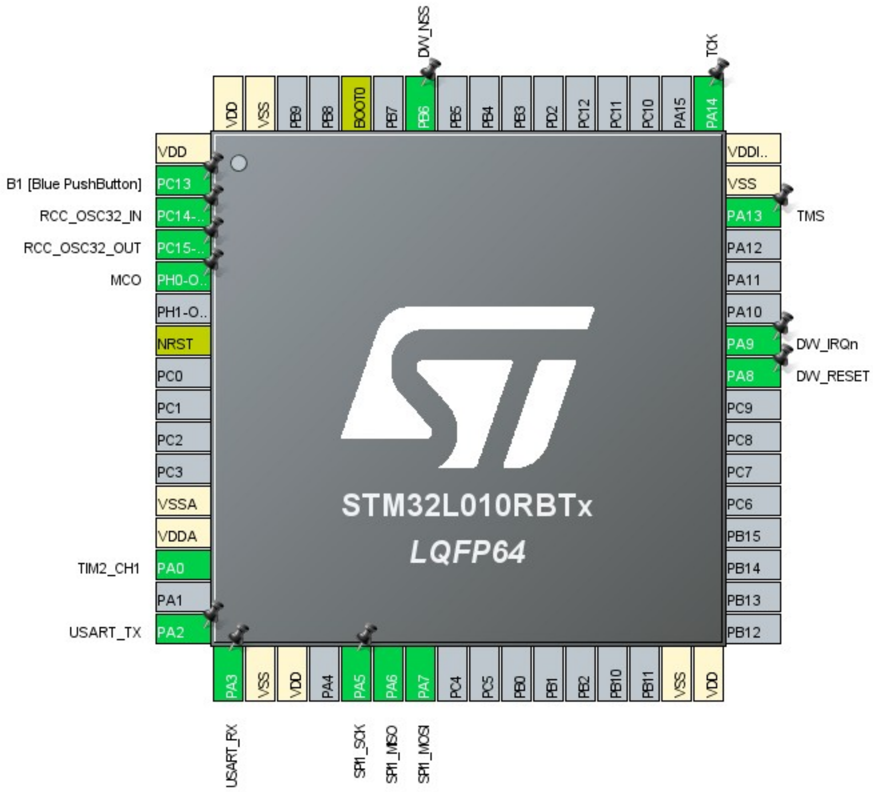

Figura 4: Pinout respondera

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 */
```

```c
/* 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. */
```

```c
#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);
```

```c
            /* 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)
{
```

```c
    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 */
```

```c
    /* 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****/
```

4. 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