|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | **Protokół zmiany schematu oraz projektu płytki obsługującej czujnik US** | | | | | | Data wystawienia: | |
|  |  | |
|  | Doc# |  |
|  | Nr wniosku NCBR: | | POIR.01.01.01-00-0196/19 | | | Nazwa projektu: | | Smart Yacht |
|  | Rozpoczęcie testów: | |  | | Zakończenie testów: | |  | |

#### Założenia

Podczas montażu systemu prototypowego został zaobserwowany szereg problemów:

1. Błędy w komunikacji UART podczas adresowania czujników ultradźwiękowych

2. Problematyczny montaż systemu spowodowany obecnością dużej ilości linii w przewodzie

3. Problematyczny montaż mechaniczny modułu czujnika US

4. Problematyczny i powolny sposób montazu przewodów

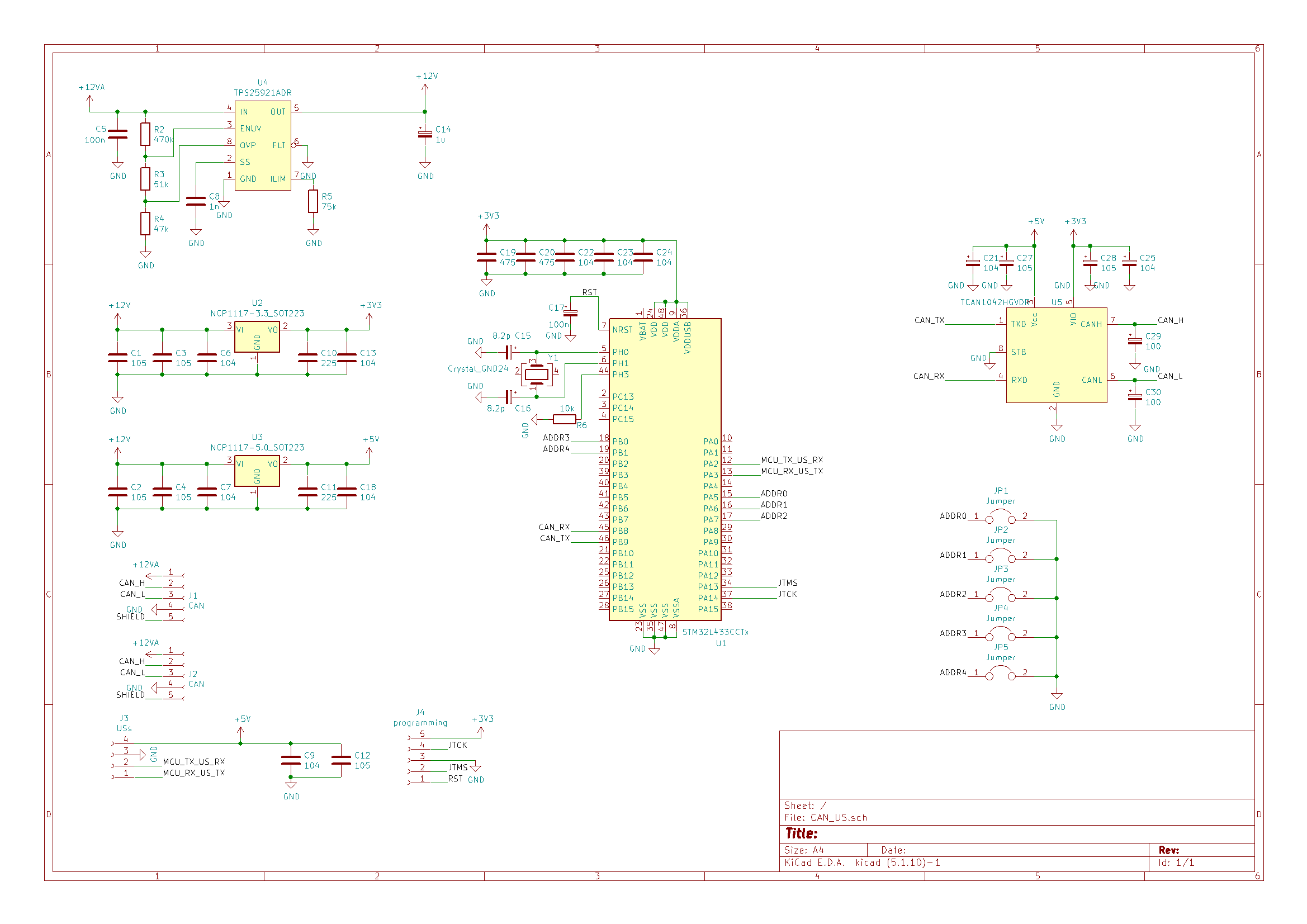
5. Nieustalony standard dotyczący złącz- wymagał zastosowania wielu przejściówek

6. Nieuzasadnione zwiększenie kosztów poprzez projekt obwodu 4 warstwowego.

7. Problematyczne programowanie układu(wymagane zasilanie 12V)

Postanowiono zmodyfikować schemat, a co za tym idzie przeprojektować obwód drukowany.

#### **Zmiany wprowadzone w schemacie**

Figura 1: Nowy schemat sterownika czujnika US

-Zmiana złącza CAN z 8 na 5 pinowe

-Dodane zwory umożliwiające indywidualne przypisanie adresu każdemu z czujników

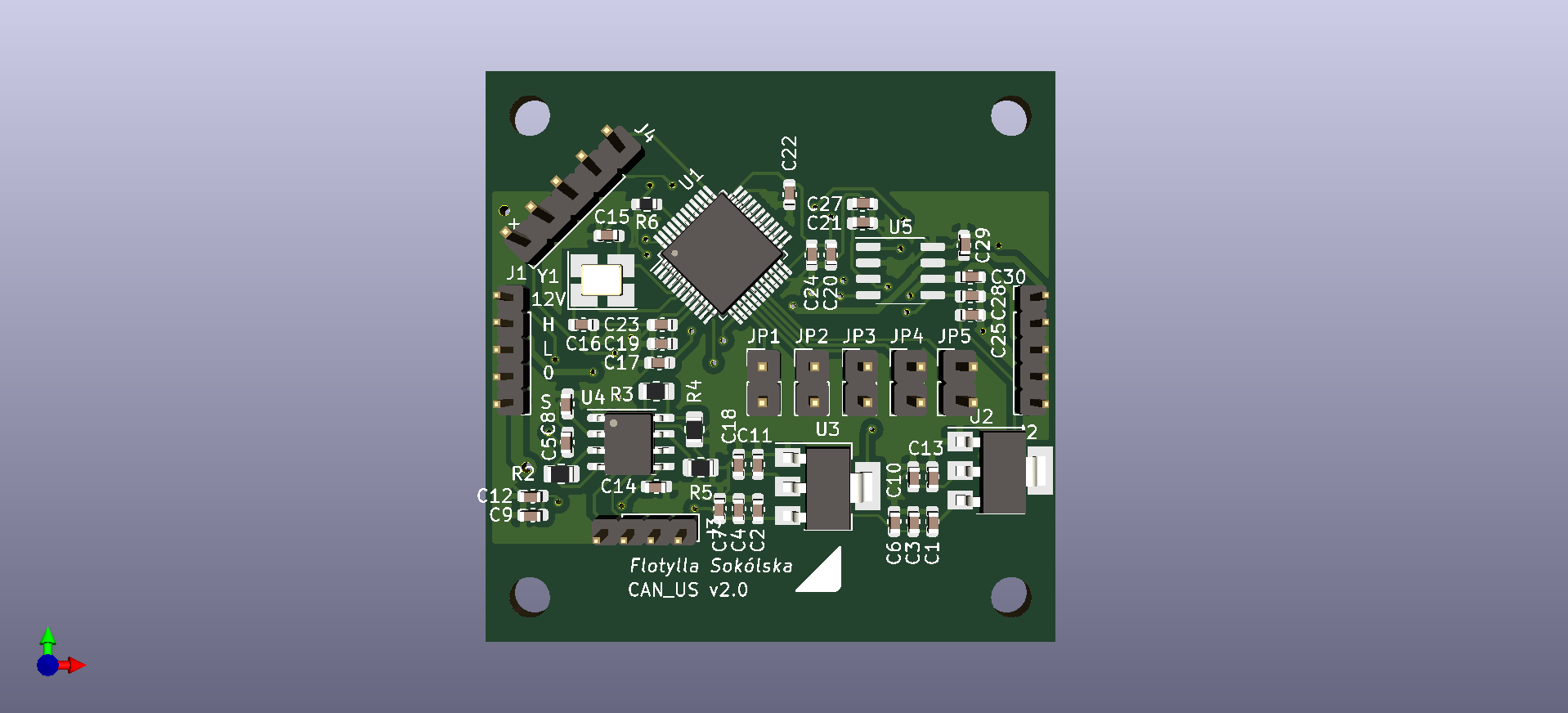
-Kondensatory tantalowe zostały wymieniona na MLCC

-złącze czujnika US zostało wymienione na 4 pinowe umożliwia to podłączenia czujnika oryginalną złączką.

-Dodano oddzielne złącze do programowania układu, wyposażone w pin RESET.

#### Zmiany wprowadzone w projekcie PCB

Płytka PCB została w całości zaprojektowana od nowa, na 2 warstwach miedzi.

Figura 2: Płytka sterownika US, wersja 2.0

Widoczne 5-pinowe złącza can na lewej oraz prawej krawędzi, złącze programowania w lewym górnym rogu oraz złącze czujnika US na dole.

Wymiary płytki i rozstaw otworów pozostają bez zmian.

#### **Zmiany wprowadzone w programie**

Treść pliku głównego:

/\* 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 \*/

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

CAN\_HandleTypeDef hcan1;

TIM\_HandleTypeDef htim6;

UART\_HandleTypeDef huart1;

UART\_HandleTypeDef huart2;

UART\_HandleTypeDef huart3;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

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

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

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

**static** **void** MX\_CAN1\_Init(**void**);

**static** **void** MX\_USART1\_UART\_Init(**void**);

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

**static** **void** MX\_USART3\_UART\_Init(**void**);

**static** **void** MX\_TIM6\_Init(**void**);

/\* USER CODE BEGIN PFP \*/

**void** setCanAdress(**void**);

/\* USER CODE END PFP \*/

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

/\* USER CODE BEGIN 0 \*/

/\* 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\_CAN1\_Init();

MX\_USART1\_UART\_Init();

MX\_USART2\_UART\_Init();

MX\_USART3\_UART\_Init();

MX\_TIM6\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_UART\_Init(&huart2);

HAL\_HalfDuplex\_Init(&huart1);

HAL\_HalfDuplex\_Init(&huart3);

HAL\_HalfDuplex\_EnableReceiver(&huart1);

HAL\_HalfDuplex\_EnableReceiver(&huart3);

HAL\_UART\_Receive\_IT(&huart2, &uartUsRx, 1);

HAL\_UART\_Receive\_IT(&huart3, uart1WireRx, 2);

HAL\_UART\_Receive\_IT(&huart1, uart1WireRx, 2);

HAL\_FLASHEx\_OBGetConfig(&pOB);

HAL\_CAN\_Init(&hcan1);

HAL\_CAN\_Start(&hcan1);

uartState = reciving;

//HAL\_IWDG\_Init(&hiwdg);

/\*if(FLASH\_OPTR\_nBOOT1 & pOB.USERConfig ||FLASH\_OPTR\_nBOOT0 & pOB.USERConfig || FLASH\_OPTR\_nSWBOOT0 & pOB.USERConfig){

setBootFromSRAM(&pOB);

}

setBootFromSRAM(&pOB);\*/

bootloader\_Init(&hcan1, &TxMailbox, 25);

canId = 200;

HAL\_TIM\_Base\_Start\_IT(&htim6);

setCanAdress();

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1) {

/\* 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};

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

\* in the RCC\_OscInitTypeDef structure.

\*/

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

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

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

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

RCC\_OscInitStruct.PLL.PLLM = 1;

RCC\_OscInitStruct.PLL.PLLN = 20;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV7;

RCC\_OscInitStruct.PLL.PLLQ = RCC\_PLLQ\_DIV2;

RCC\_OscInitStruct.PLL.PLLR = RCC\_PLLR\_DIV2;

**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\_4) != HAL\_OK)

{

Error\_Handler();

}

PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_USART1|RCC\_PERIPHCLK\_USART2

|RCC\_PERIPHCLK\_USART3;

PeriphClkInit.Usart1ClockSelection = RCC\_USART1CLKSOURCE\_PCLK2;

PeriphClkInit.Usart2ClockSelection = RCC\_USART2CLKSOURCE\_PCLK1;

PeriphClkInit.Usart3ClockSelection = RCC\_USART3CLKSOURCE\_PCLK1;

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

{

Error\_Handler();

}

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

\*/

**if** (HAL\_PWREx\_ControlVoltageScaling(PWR\_REGULATOR\_VOLTAGE\_SCALE1) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief CAN1 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** MX\_CAN1\_Init(**void**)

{

/\* USER CODE BEGIN CAN1\_Init 0 \*/

/\* USER CODE END CAN1\_Init 0 \*/

/\* USER CODE BEGIN CAN1\_Init 1 \*/

/\* USER CODE END CAN1\_Init 1 \*/

hcan1.Instance = CAN1;

hcan1.Init.Prescaler = 40;

hcan1.Init.Mode = CAN\_MODE\_NORMAL;

hcan1.Init.SyncJumpWidth = CAN\_SJW\_1TQ;

hcan1.Init.TimeSeg1 = CAN\_BS1\_13TQ;

hcan1.Init.TimeSeg2 = CAN\_BS2\_2TQ;

hcan1.Init.TimeTriggeredMode = DISABLE;

hcan1.Init.AutoBusOff = DISABLE;

hcan1.Init.AutoWakeUp = ENABLE;

hcan1.Init.AutoRetransmission = ENABLE;

hcan1.Init.ReceiveFifoLocked = DISABLE;

hcan1.Init.TransmitFifoPriority = DISABLE;

**if** (HAL\_CAN\_Init(&hcan1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN CAN1\_Init 2 \*/

canFilterConfig.FilterMode = CAN\_FILTERMODE\_IDMASK;

canFilterConfig.FilterScale = CAN\_FILTERSCALE\_32BIT;

canFilterConfig.FilterIdHigh = 210;

canFilterConfig.FilterIdLow = 200;

canFilterConfig.FilterMaskIdHigh = 0x0000;

canFilterConfig.FilterMaskIdLow = 0x0000;

canFilterConfig.FilterFIFOAssignment = CAN\_RX\_FIFO0;

canFilterConfig.FilterActivation = ENABLE;

canFilterConfig.FilterBank = 1;

HAL\_CAN\_ConfigFilter(&hcan1, &canFilterConfig);

canFilterConfig.FilterIdHigh = 0;

canFilterConfig.FilterIdLow = 0;

canFilterConfig.FilterBank = 2;

HAL\_CAN\_ConfigFilter(&hcan1, &canFilterConfig);

HAL\_CAN\_Start(&hcan1);

**if** (HAL\_CAN\_ActivateNotification(&hcan1, CAN\_IT\_RX\_FIFO0\_MSG\_PENDING)

!= HAL\_OK) {

/\* Notification Error \*/

Error\_Handler();

}

canTxHeader.DLC = 1; //give message size of 1 byte

canTxHeader.IDE = CAN\_ID\_STD; //set identifier to standard

canTxHeader.RTR = CAN\_RTR\_DATA; //set data type to remote transmission request?

canTxHeader.StdId = 200;

canTxHeader.ExtId = 0x00;

/\* USER CODE END CAN1\_Init 2 \*/

}

/\*\*

\* @brief TIM6 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** MX\_TIM6\_Init(**void**)

{

/\* USER CODE BEGIN TIM6\_Init 0 \*/

/\* USER CODE END TIM6\_Init 0 \*/

TIM\_MasterConfigTypeDef sMasterConfig = {0};

/\* USER CODE BEGIN TIM6\_Init 1 \*/

/\* USER CODE END TIM6\_Init 1 \*/

htim6.Instance = TIM6;

htim6.Init.Prescaler = 2999;

htim6.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim6.Init.Period = 2665;

htim6.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

**if** (HAL\_TIM\_Base\_Init(&htim6) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

**if** (HAL\_TIMEx\_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM6\_Init 2 \*/

/\* USER CODE END TIM6\_Init 2 \*/

}

/\*\*

\* @brief USART1 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** MX\_USART1\_UART\_Init(**void**)

{

/\* USER CODE BEGIN USART1\_Init 0 \*/

/\* USER CODE END USART1\_Init 0 \*/

/\* USER CODE BEGIN USART1\_Init 1 \*/

/\* USER CODE END USART1\_Init 1 \*/

huart1.Instance = USART1;

huart1.Init.BaudRate = 9600;

huart1.Init.WordLength = UART\_WORDLENGTH\_9B;

huart1.Init.StopBits = UART\_STOPBITS\_1;

huart1.Init.Parity = UART\_PARITY\_EVEN;

huart1.Init.Mode = UART\_MODE\_TX\_RX;

huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

**if** (HAL\_HalfDuplex\_Init(&huart1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART1\_Init 2 \*/

/\* USER CODE END USART1\_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 = 9600;

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

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

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

huart2.Init.Mode = UART\_MODE\_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 USART3 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** MX\_USART3\_UART\_Init(**void**)

{

/\* USER CODE BEGIN USART3\_Init 0 \*/

/\* USER CODE END USART3\_Init 0 \*/

/\* USER CODE BEGIN USART3\_Init 1 \*/

/\* USER CODE END USART3\_Init 1 \*/

huart3.Instance = USART3;

huart3.Init.BaudRate = 9600;

huart3.Init.WordLength = UART\_WORDLENGTH\_9B;

huart3.Init.StopBits = UART\_STOPBITS\_1;

huart3.Init.Parity = UART\_PARITY\_EVEN;

huart3.Init.Mode = UART\_MODE\_TX\_RX;

huart3.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart3.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart3.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart3.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

**if** (HAL\_HalfDuplex\_Init(&huart3) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART3\_Init 2 \*/

/\* USER CODE END USART3\_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 pins : ADDR0\_Pin ADDR1\_Pin ADDR2\_Pin \*/

GPIO\_InitStruct.Pin = ADDR0\_Pin|ADDR1\_Pin|ADDR2\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_PULLUP;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\*Configure GPIO pins : ADDR3\_Pin ADDR4\_Pin \*/

GPIO\_InitStruct.Pin = ADDR3\_Pin|ADDR4\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_PULLUP;

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

}

/\* USER CODE BEGIN 4 \*/

**void** setCanAdress(**void**) {

//od 200 co 10

//odczytaj zworki adresowe

uint8\_t bit0 = HAL\_GPIO\_ReadPin(ADDR0\_GPIO\_Port, ADDR0\_Pin);

uint8\_t bit1 = HAL\_GPIO\_ReadPin(ADDR1\_GPIO\_Port, ADDR1\_Pin);

uint8\_t bit2 = HAL\_GPIO\_ReadPin(ADDR2\_GPIO\_Port, ADDR2\_Pin);

uint8\_t bit3 = HAL\_GPIO\_ReadPin(ADDR3\_GPIO\_Port, ADDR3\_Pin);

uint8\_t bit4 = HAL\_GPIO\_ReadPin(ADDR4\_GPIO\_Port, ADDR4\_Pin);

**if** (bit0 == 1 && bit1 == 1 && bit2 == 1 && bit3 == 1 && bit4 == 1)

canId = 1010;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 1 && bit3 == 1 && bit4 == 0)

canId = 1020;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 1 && bit3 == 0 && bit4 == 1)

canId = 1030;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 1 && bit3 == 0 && bit4 == 0)

canId = 1040;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 0 && bit3 == 1 && bit4 == 1)

canId = 1050;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 0 && bit3 == 1 && bit4 == 0)

canId = 1060;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 0 && bit3 == 0 && bit4 == 1)

canId = 1070;

**else** **if**(bit0 == 1 && bit1 == 1 && bit2 == 0 && bit3 == 0 && bit4 == 0)

canId = 1080;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 1 && bit3 == 1 && bit4 == 1)

canId = 1090;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 1 && bit3 == 1 && bit4 == 0)

canId = 1100;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 1 && bit3 == 0 && bit4 == 1)

canId = 1110;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 1 && bit3 == 0 && bit4 == 0)

canId = 1120;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 0 && bit3 == 1 && bit4 == 1)

canId = 1130;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 0 && bit3 == 1 && bit4 == 0)

canId = 1140;

**else** **if**(bit0 == 1 && bit1 == 0 && bit2 == 0 && bit3 == 0 && bit4 == 1)

canId = 1150;

//ustaw adres czujnika

canTxHeader.StdId = canId;

}

//void HAL\_CAN\_RxFifo0MsgPendingCallback(CAN\_HandleTypeDef \*hcan) {

// //HAL\_IWDG\_Refresh(&hiwdg);

// HAL\_CAN\_GetRxMessage(hcan, CAN\_RX\_FIFO0, &canRxHeader, canRxData);

// switch (canRxHeader.StdId) { //komendy do bootloadera

// case 200: // włączenie bootloadera

// if (canRxData[0] == 1) {

// flags.programmingBusy = 1;

// }

// if (canRxData[0] == 0) {

// flags.programmingBusy = 0;

// }

// break;

// case 201: //wysłanie id

// canTxHeader.StdId = 200;

// canTxHeader.DLC = 3;

// canTxData[0] = (canId >> 8) & 0xFF;

// canTxData[1] = canId & 0xFF;

// canTxData[2] = 1;

// HAL\_CAN\_AddTxMessage(&hcan1, &canTxHeader, canTxData, &TxMailbox);

// break;

// case 202: //uruchomienie bootloadera

// if ((canRxData[0] << 8) + canRxData[1] == canId) {

// canTxHeader.DLC = 2;

// canTxHeader.IDE = 0;

// HAL\_CAN\_AddTxMessage(&hcan1, &canTxHeader, canRxData, &TxMailbox);

// startBootloader(&hcan1);

// }

// break;

// }

//}

**void** HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart) {

//HAL\_IWDG\_Refresh(&hiwdg);

// if (huart == &huart1) {

// canAdressingProtocol(&canId, uart1WireRx, &huart1, &huart3, &uartState);

// }

// if (huart == &huart3) {

// canAdressingProtocol(&canId, uart1WireRx, &huart1, &huart3, &uartState);

// }

**if** (huart == &huart2) { //odczyt danych z czujnka

**if** (uartUsRx == 0xff || us.frameStarted == 0) { //&&

us.data[0] = uartUsRx;

us.counter = 0;

us.frameStarted = 1;

}

**if** (us.frameStarted == 1) {

us.data[us.counter] = uartUsRx;

**if** (us.counter == 3) {

us.frameStarted = 0;

**int** sum = (us.data[0] + us.data[1] + us.data[2]) & 0x00FF;

**if** (sum == us.data[3]) {

us.distance = (us.data[1] << 8) + us.data[2];

// if (us.distance < 280) {

// us.distance = 280;

// }

// if (us.distance > 7500) {

// us.distance = 7500;

// }

**if** (us.distance < 280 && us.distance >= 255) {

us.distance = 280;

}

**if** (us.distance > 7500 || us.distance <= 255) {

us.distance = 7500;

}

canTxHeader.StdId = canId;

canTxHeader.DLC = 2;

canTxData[0] = us.distance >> 8;

canTxData[1] = us.distance;

}

} **else** {

us.counter++;

}

}

HAL\_UART\_Receive\_IT(&huart2, &uartUsRx, 1);

}

}

**void** HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim) {

**if** (htim->Instance == TIM6) {

HAL\_CAN\_AddTxMessage(&hcan1, &canTxHeader, canTxData, &TxMailbox);

}

}

**void** setBootFromSysMem(FLASH\_OBProgramInitTypeDef \*pOB) {

**if** (HAL\_FLASH\_Unlock() == HAL\_OK) {

**if** (HAL\_FLASH\_OB\_Unlock() == HAL\_OK) {

pOB->OptionType = OPTIONBYTE\_USER;

pOB->USERType = OB\_USER\_nBOOT1 | OB\_USER\_nSWBOOT0 | OB\_USER\_nBOOT0;

pOB->USERConfig = 0;

pOB->USERConfig = FLASH\_OPTR\_nBOOT1;

HAL\_FLASHEx\_OBProgram(pOB);

HAL\_FLASH\_OB\_Launch();

}

HAL\_FLASH\_OB\_Lock();

HAL\_FLASH\_Lock();

HAL\_FLASH\_OB\_Launch();

}

}

**void** setBootFromSRAM(FLASH\_OBProgramInitTypeDef \*pOB) {

**if** (HAL\_FLASH\_Unlock() == HAL\_OK) {

**if** (HAL\_FLASH\_OB\_Unlock() == HAL\_OK) {

pOB->OptionType = OPTIONBYTE\_USER;

pOB->USERType = OB\_USER\_nBOOT1 | OB\_USER\_nSWBOOT0 | OB\_USER\_nBOOT0;

pOB->USERConfig = 0;

HAL\_FLASHEx\_OBProgram(pOB);

HAL\_FLASH\_OB\_Launch();

}

HAL\_FLASH\_Lock();

HAL\_FLASH\_OB\_Lock();

}

}

/\* 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\*\*\*\*/

Usunięto protokół adresowania kontrolerów poprzez interfejs UART, dodano funkcję setCanAdress() ustawiającą adres CAN kontrolera przy pomocy zworek.

Odwrócono także zamianę pinów RX/TX w UART2. Pozostałe interfejsy UART pozostają nieużywane.

Ostatni dystans jest wysyłany niezależnie od czujnika US, sterowany przerwaniami synchronicznymi od timera 6.

#### **Wnioski**

Układ został przetestowany, programuje się i działa poprawnie.

-

Wykonał: Bartosz Pracz