|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | **Protokół kontrukcyjny emulatora manetki silnika** | | | | | | Data wystawienia: | |
|  |  | |
|  | Doc# |  |
|  | Nr wniosku NCBR: | | POIR.01.01.01-00-0196/19 | | | Nazwa projektu: | | Smart Yacht |
|  | Rozpoczęcie testów: | |  | | Zakończenie testów: | |  | |

#### Założenia

System powinien mieć możłiwość sterowania silnikiem elektrycznym zamontowanym w łodzi.

Kontroler silnika nie przewiduje sterowania innego niż manetką, zatem postanowiono zaprojektować emulator manetki o następujących cechach:

- Sterowanie magistralą CAN

- Przekaźnik przyłączający do kontrolera silnika manetkę(domyślnie) bądź sygnał analogowy generowany przez mikrokontroler(na żądanie)

-Wysyłanie do urządzenia master informacji o aktualnym położeniu manetki

-Odbieranie od urządzenia master informacji o włączeniu bądź wyłączeniu sterowania automatycznego

-Odbieranie od urządzenia master informacji o zadanej mocy i kierunku obrotów silnika

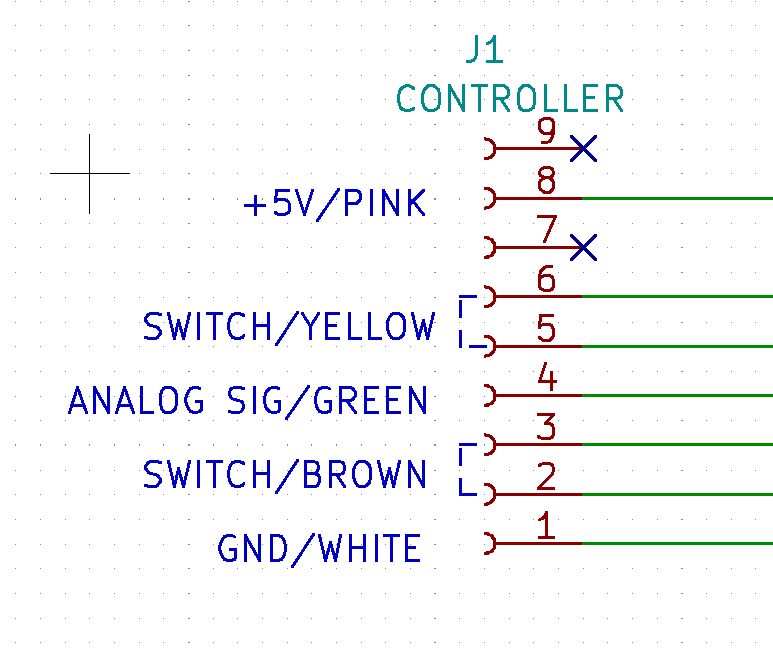
#### **Sposób działania kontrolera silnika**

Kontroler silnika obiera informację o położeniu manetki w sposób analogowy- manetka zasilona zostaje napięciem 5V, odysyła sygnał o napięciu 0-5V, gdzie 2.5V odpowiada silnikowi zatrzymanemu, 0,5V maksymalnym obrotom do przodu, oraz 4,5V maksymalnym obrotom do tyłu.

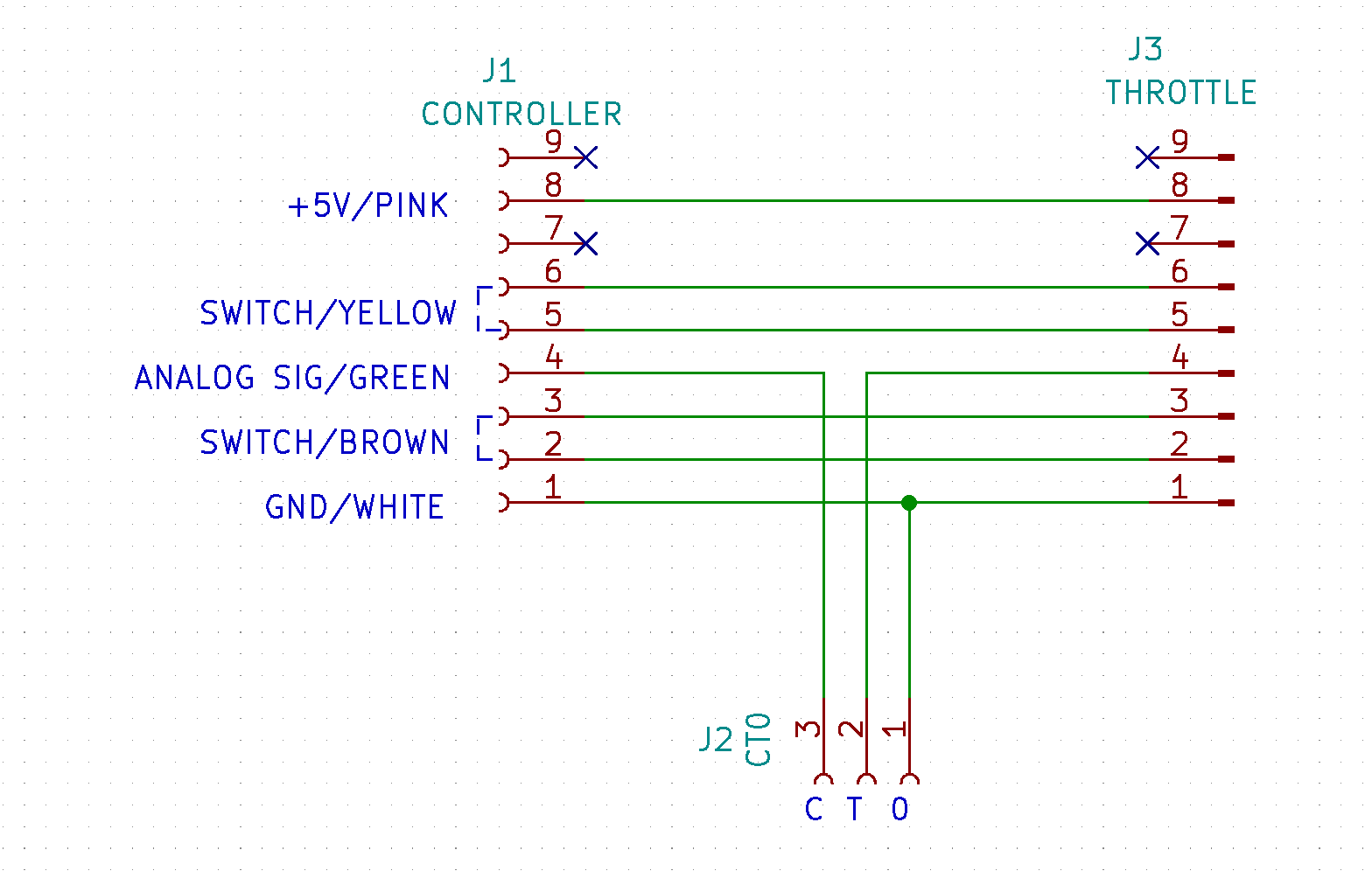
Gdy stacyjka jest wyłączona, napięcie wynosi 0V(manetka jest odłączona od zasilania).

#### **Sposób emulacji hardware’owej**

Metodą inżynierii wstecznej pinout złącza manetki został rozrysowany poniżej:

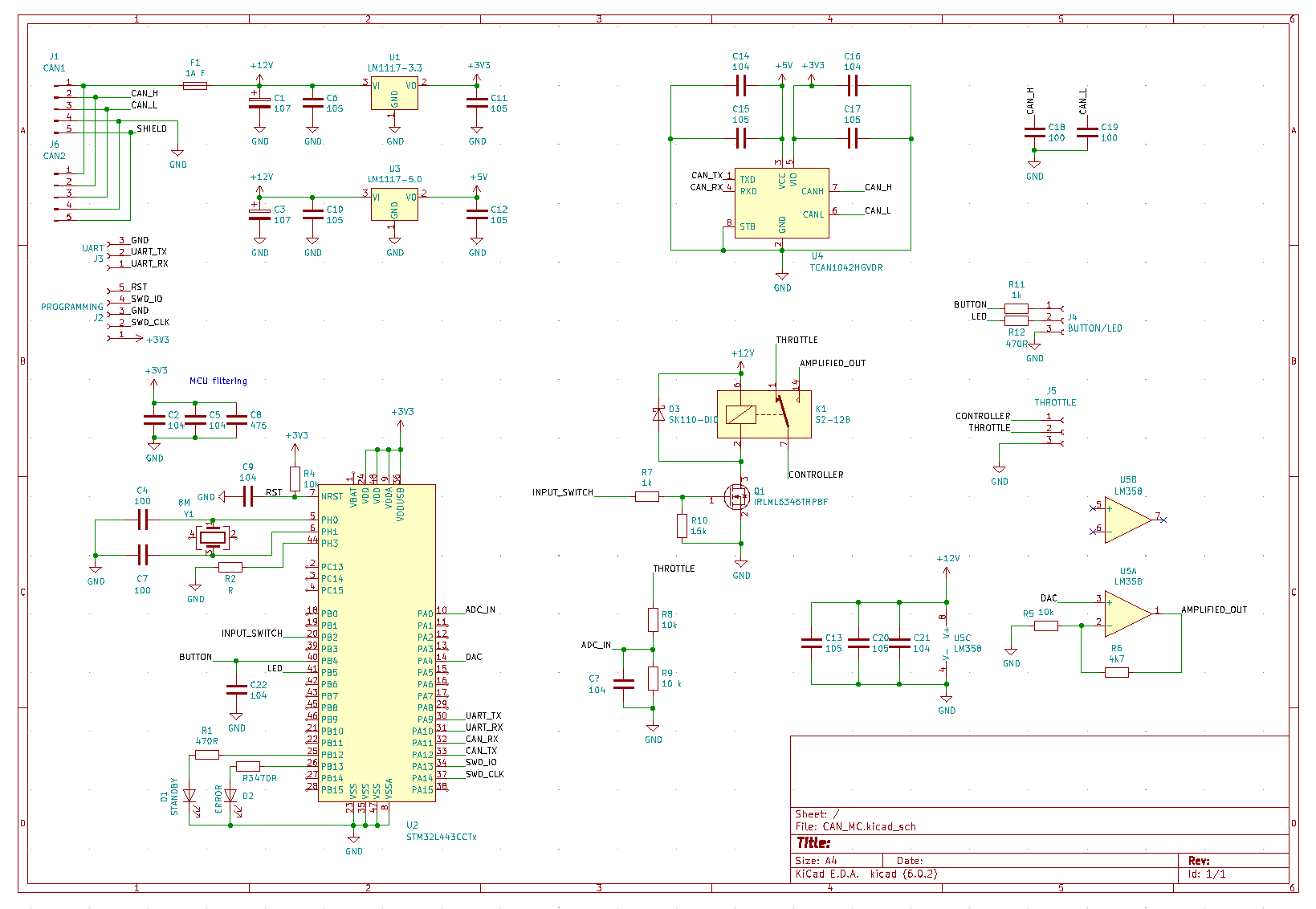
Figura 1: Pinout złącza manetki Kelly Controller

Wpinając układ pomiędzy manetkę a kontroler, ma on możliwość emulacji położenia manetki generując sygnał analogowy. Połączenie takie wygląda następująco:

Figura 2: Podłączenie emulatora

#### Linie C(controller), T(throttle) oraz O(output) zostają podłączone do układu emulatora.

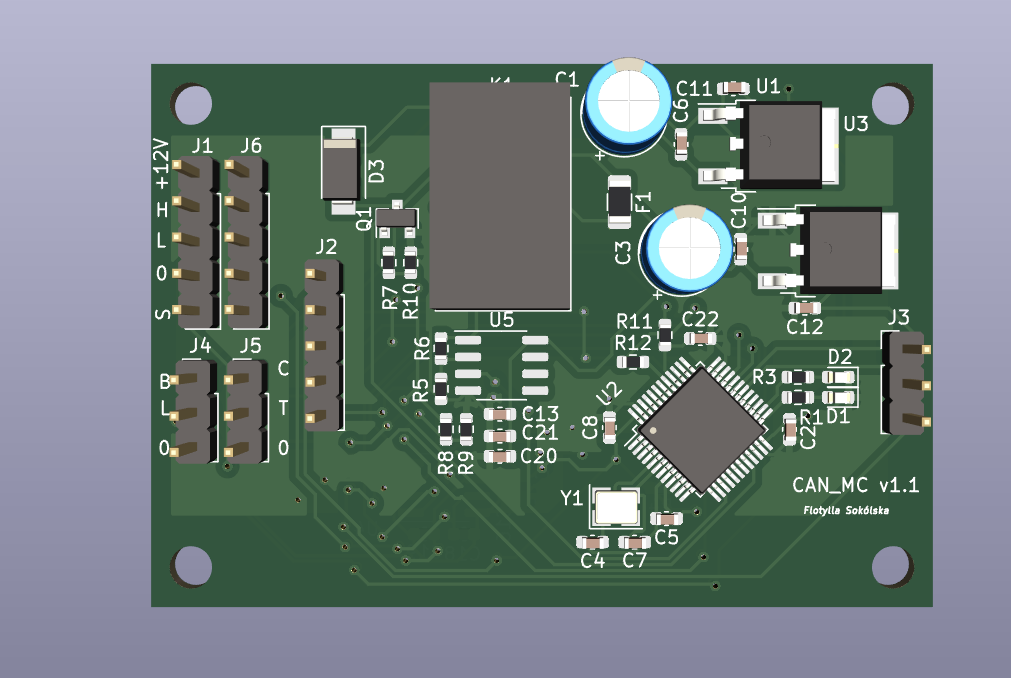
Emulator przy pomocy przekaźnika, dopina do linii Output sygnał z manetki, bądź sygnał który sam wygeneruje.

Figura 3: Schemat emulatora

Jako generator sygnału analogowego został użyty układ DAC wbudowany w mikrokontroler. Do jego napięciowego wzmocnienia został użyty wzmacniacz operacyjny(LM358) w aplikacji wzmacniacza nieodwracającego.

Sygnał z manetki jest odczytywany poprzez wbudowany w mikrokontroler przetwornik ADC, poprzez dzielnik napięcia(R8/R9) z pasywnym filtrem dolnoprzepustowym(R8/C23).

Przekaźnik jest sterowany sygnałem z mikrokontrolera wzmocnionym tranzystorem Q1.

Figura 4: Render płytki PCB emulatora

#### **Firmware**

Firmware został napisany w taki sposób, aby urządzenie master miało możliwość kontrolowania silnika tylko wtedy, kiedy manetka jest ustawiona w środkowej pozycji(silnik wyłączony). Przy sterowaniu ręcznym emulator wysyła informację o jej położeniu.

Tylko jeżeli manetka jest ustawiona w położeniu środkowym, odebrane przez emulator żądanie sterowania automatycznego od urządzenia master skutkuje załączeniem przekaźnika i uruchomieniem sterowania automatycznego. Wraz z żądaniem master wysyła wysokość sygnału analogowego. W przeciwnym wypadku(manetka wychylona przez sternika) żądanie zostaje zignorowane.

Kiedy sterowanie systemowe jest włączone, emulator odsyła z powrotem wartość zadaną.

Ręczne odchylenie manetki podczas sterowania systemowego skutkuje natychmiastowym odłączeniem sygnału zadanego i podłączeniem manetki.

#### Treść programu

/\* USER CODE BEGIN Header \*/

/\*\*

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

\* @file : main.c

\* @brief : Main program body

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

\* @attention

\*

\* Copyright (c) 2022 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

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

\*/

/\* 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** huart2;

/\* USER CODE BEGIN PV \*/

**CAN\_FilterTypeDef** canFilterConfig;

**CAN\_TxHeaderTypeDef** pHeader;

**CAN\_RxHeaderTypeDef** canRxHeader;

**CAN\_TxHeaderTypeDef** canTxHeader;

**uint8\_t** Data;

**uint8\_t** uartUsRx;

**GPIO\_InitTypeDef** gpio;

**uint8\_t** canRxData[8];

**uint8\_t** canTxData[8];

**union** **Data2Bytes**{

**uint8\_t** bytes [2];

**uint16\_t** data;

}data2Bytes;

**uint16\_t** canId;

**uint32\_t** TxMailbox;

**typedef** **struct**{

**uint8\_t** data[4];

**uint16\_t** distance;

**uint8\_t** counter;

**uint8\_t** frameStarted;

}**Us**;

**Us** us;

/\* 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\_USART2\_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\_USART2\_UART\_Init**();

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

/\* USER CODE BEGIN 2 \*/

**HAL\_UART\_Init**(&huart2);

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

**HAL\_CAN\_Start**(&hcan1);

canId = 200;

**setCanAdress**();

//change interval of TIM6 interrupt depending on sensor number. This will take it unsynchronized

htim6.Init.Prescaler = 2900 + (canTxHeader.StdId-1000);

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

**Error\_Handler**();

}

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

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

**setCanAdress**();

/\* 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};

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

\*/

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

{

**Error\_Handler**();

}

/\*\* 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**();

}

}

/\*\*

\* @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 = *DISABLE*;

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 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 GPIO Initialization Function

\* @param None

\* @retval None

\*/

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

{

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

/\* GPIO Ports Clock 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\_UART\_RxCpltCallback**(**UART\_HandleTypeDef** \*huart) {

**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 >= 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);

}

}

/\* 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 \*/

#### **Wnioski**

Układ został przetestowany, programuje się i działa poprawnie.

Wykonał: Bartosz Pracz