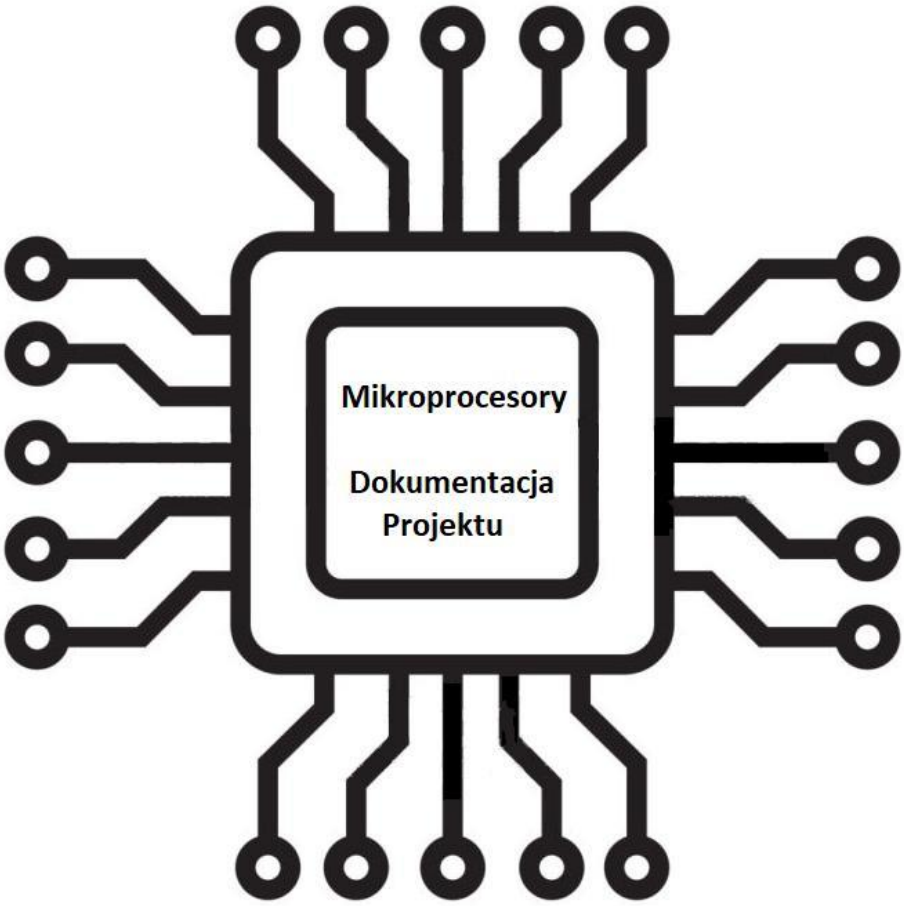
POLITECHNIKA BYDGOSKA
im. Jana i Jędrzeja Śniadeckich

POLITECHNIKA BYDGOSKA
Wydział Telekomunikacji, Informatyki i Elektrotechniki



Adrian Gidaszewski

INFORMATYKA STOSOWANA

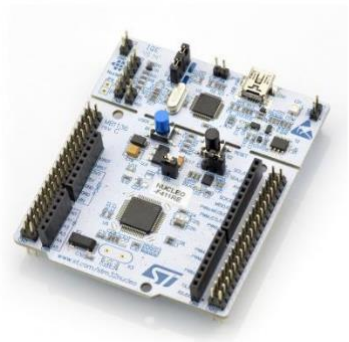
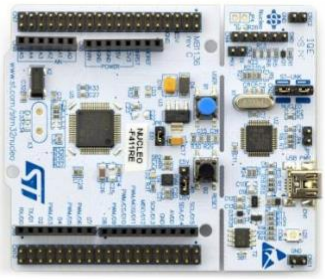
GRUPA 2

2021/2022

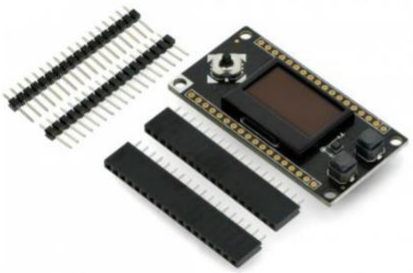
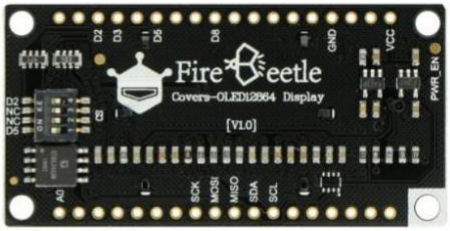
Spis treści i dokumentacji

1. Narzędzia

- STM32 NUCLEO – F411RE – STM32F411RE ARM Cortex M4

- DFRobot wyświetlacz OLED niebieski 0,96” 128x64px – I2C – dla FireBeetle

- Ultradźwiękowy czujnik odległości 2-400cm – HY – SRF05


- Przewody połączeniowe żeńsko – żeńskie 10cm (TYP GOLDPIN)

Specyfikacja projektu

1. Komunikacja z PC za pomocą interfejsu USART z buforem kołowym
2. Protokół komunikacyjny – zaprojektowanie, zaimplementowanie
3. Wyświetlacz OLED niebieski graficzny 0,96” 128x64px I2C
   - podłączenie (fizyczne i logiczne)
   - konfiguracja w CubeMX
4. SSD1306 – możliwość wypisywania tekstu
   - wypisywanie tekstu z zadanej pozycji
   - przewijanie tekstu
   - wybór czcionki (min.3)
5. Obsługa czujnika z wykorzystaniem timera pracującego w trybie Input Capture ze wsparciem DMA

Protokół komunikacyjny

Znaki będą odbierane przez urządzenie, które będzie je przechowywać w formacie ASCII. Możliwe jest wysyłanie dowolnej wartość parametru do ramki, ponieważ RealTerm emuluje terminal tekstowy, lecz wysyłamy jako ASCII. Następnie będą zapisane w systemie decymalnym lub jako tekst. Odebrane znaki program jest w stanie konwertować na dowolny parametr. 1 znak będzie zajmował 1 bajt w pamięci. Minimalna długość ramki to 9 znaków. Maksymalna długość ramki to 108 znaków.

Ramka:

Znak rozpoczynający ramkę	Komenda	Checksuma	Dane	Znak końca
1 znak ($)	3 znaki	3 znaki	1 – 100 znaków	1 znak (#)
1 B	3 B	3 B	1 – 100 B	1 B

Opis komunikacji:

- Komunikacja pomiędzy użytkownikiem, a urządzeniem poprzez program terminalu RealTerm.
- Znaki będą odbierane przez urządzenie, które będzie je przechowywać w formacie ASCII. 1 znak będzie zajmował 1 bajt w pamięci.
- Użytkownik poprzez terminal będzie wysyłał odpowiednie komunikaty do urządzenia STM32.
- Odbieranie poleceń użytkownika poprzez protokół, który będzie zwracał informacje zwrotną.
- Użytkownik przed wysłaniem ramki będzie musiał obliczyć checksumę dla wprowadzonej komendy
- Użytkownik będzie wprowadzał odpowiednią komendę poprzedzając ją znakiem rozpoczęcia ramki ‘$’ i kończąc ją znakiem zakończenia ramki ‘#’

Obsługa błędów:

- Skutkiem wielokrotnego wprowadzenia znaku początku ramki będzie rozpoczęcie się jej od ostatniego wysłania ramki. Reszta zostanie zignorowana.
- Skutkiem wprowadzenia znaku końca ramki w dowolnym miejscu komendy będzie wyświetlenie informacji zwrotnej o błędzie
- Skutkiem wprowadzenia znaku końca ramki w polu pierwszego miejsca danych będzie wyświetlenie informacji zwrotnej o błędzie
- Skutkiem braku jakiegokolwiek znaku w polu komendy będzie wyświetlenie informacji zwrotnej o błędzie
- Skutkiem braku pierwszego znaku w polu danych będzie wyświetlenie informacji zwrotnej o błędzie
- Skutkiem wielokrotnego wprowadzenia znaku końca ramki będzie zakończenie się jej od ostatniego wysłania ramki, a reszta zostanie zignorowana
- Skutkiem wprowadzenia złej komendy będzie wyświetlenie informacji zwrotnej o błędzie.
- Skutkiem przekroczenia maksymalnej długości ramki będzie wyświetlenie informacji zwrotnej o błędzie.
- Skutkiem podania złej checksumy będzie wyświetlenie informacji zwrotnej o błędnej checksumie

Dokumentacja

Adrian Gidaszewski Informatyka Stosowana NST 22/23

Komendy

Konstrukcja ze znaków odpowiadających za konkretne rozkazy.

Znaki komend będą odbierane przez urządzenie, które będzie je przechowywać w formacie ASCII zapisane w systemie decymalnym lub jako tekst (poza znakiem początku i końca ramki).

Komendy są ograniczone, wszystkie komendy mogą składać się jedynie z liter lub literek.

Zakres możliwych znaków do wprowadzenia:

65 – 90 (dec.)

97 – 122 (dec.)

Program zwróci błąd w wypadku wprowadzenia innego znaku niż litera lub literka, ponieważ jest to niedozwolone.

Program zwróci komunikat o nieznalezieniu obsługiwanej komendy w wypadku wprowadzenia nieobsługiwanej komendy.

Tabela komend:

Nazwa komendy	Rozmiar	Czynność
BFF	**3B**	**Zwracanie pierwszego przechowywanego dystansu przez czujnik z bufora**
BFL	**3B**	**Zwracanie ostatniego przechowywanego dystansu przez czujnik z bufora**
IDX	**3B**	**Zwracanie indeksu bufora**
CLR	**3B**	**Czyszczenie bufora**
(inne komendy składające się wyłącznie z liter lub literek)	**3B**	**…**

Checksum’a

Checksuma jest odbierana w formacie ASCII zapisane w systemie decymalnym lub jako tekst. Odebrane znaki z tego pola będą zapisywane w systemie decymalnym lub jako tekst. Urządzenie STM będzie sumować wartość binarną każdego bajtu znaku komendy podanej w ramce. Będzie to robił dla wszystkich znaków danej komendy. Sumowane bajty będą trafiać do zmiennej l_checksum, która w środowisku STM jest typu uint8_t, więc jej zakres wartości to 0 – 255. Kiedy wartość l_checksum przekroczy 255, będzie się zmieniać na 0, a następnie zwiększać aż do 255, a następnie znów przejdzie do 0, tworząc cykl. Wartość ta nie będzie już odpowiadała rzeczywistej sumie bajtów z tablicy, więc przed wysłaniem ramki należy wziąć ten fakt pod uwagę, aby prawidłowo obliczyć wartość. W polu checksum’y niedozwolone są innego typu znaki niż liczby, więc zakres możliwych znaków do wpisania to: 48 – 57 (dec.).

Dane

W polu dane znaki będą przechowywane w formacie ASCII zapisane w systemie decymalnym lub jako tekst.
1 znak będzie zajmował 1 bajt w pamięci. W polu dane nie można umieścić znaku rozpoczynającego/kończącego ramke.
Zakres możliwych znaków do wpisania to:

0 – 34 (dec.)
37 – 127 (dec.)

Parametry komunikacji

Konfiguracja UART

- **Baud Rate** – 9600 Bits/s
- **Word Length** – 8 bitów
- **Parity** – None
- **Stop Bits** – 1
- **Data Direction** – Receive and Transmit
- **Over Sampling** – 16 Sampling

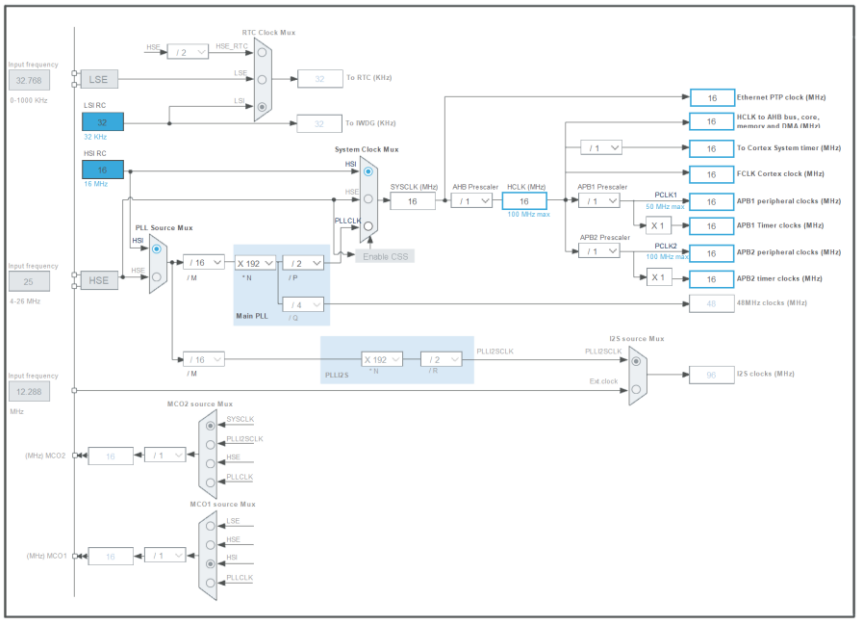
Konfiguracja I2C1

- **I2C Speed Mode** - Fast mode
- **I2C Clock Speed (Hz)** - MSB First
- **Fast Mode Duty Circle** - Duty cycle Tlow / Thigh = 2
- **Clock No Stretch Mode** - Disabled
- **Primary Slave Address** - 0
- **General Call Address Detection** – Disabled

Dokumentacja

Adrian Gidaszewski Informatyka Stosowana NST 22/23

Konfiguracja zegarów



Projekt – kod

Komunikacja za pomocą USART'a pomiędzy PC a mikrokontrolerem

Zmienne

```
volatile buffer_t Tx = {0}, Rx = {0}; // Bufory kołowe
char frameData[100]; // Dane ramki z pola payload (dane)
char command[3]; // Komenda ramki
int badVal[] = {0x00, 0x23};
bool processing = true;
uint8_t frame[sizeOfBuffer]; // Ramka
uint8_t tempstring[CHECKSUM_LEN] = "";
uint8_t frameChar = '\0'; // Pojedynczy znak ramki
uint8_t frameLength = 0; // Długość ramki
uint8_t frameReceiving = 0;
uint8_t checksum = 0;
uint8_t l_checksum = 0;
```

Funkcje odpowiedzialne za USART

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == USART2) {

        Rx.RXbuffIdx++;
        if (Rx.RXbuffIdx >= sizeOfBuffer) {
            Rx.RXbuffIdx = 0;
        }
        HAL_UART_Receive_IT(&huart2, &Rx.array[Rx.RXbuffIdx], 1);
    }
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
    if(Tx.TXbuffIdx != Tx.RXbuffIdx){
        uint8_t tempChar = Tx.array[Tx.TXbuffIdx];
        Tx.TXbuffIdx++;

        if(Tx.TXbuffIdx >= sizeOfBuffer){
            Tx.TXbuffIdx = 0;
        }
        HAL_UART_Transmit_IT(&huart2, &tempChar, 1);
    }
}
```

Funkcja Send

```
void USART_Send(char* message, ...){
    char tempMsg[105];
    int i;
    volatile int send_idx = Tx.empty;

    va_list arglist;
    va_start(arglist, message);
    vsprintf(tempMsg, message, arglist);
    va_end(arglist);

    for (i = 0; i < strlen(tempMsg); i++) {
        Tx.array[send_idx] = tempMsg[i];
        send_idx++;
        if (send_idx >= sizeOfBuffor) {
            send_idx = 0;
        }
    }

    __disable_irq();
    if ((Tx.empty == Tx.busy) && (__HAL_UART_GET_FLAG(&huart2, UART_FLAG_TXE)==SET)) {
        Tx.empty = send_idx;
        uint8_t tmp = Tx.array[Tx.busy];
        Tx.busy++;
        if (Tx.busy >= sizeOfBuffor)
            Tx.busy = 0;

        HAL_UART_Transmit_IT(&huart2, &tmp, 1);
        }else{
            Tx.empty = send_idx;
        }
    __enable_irq();

}
```

Dokumentacja

Adrian Gidaszewski Informatyka Stosowana NST 22/23

ANALIZA RAMKI PROTOKOŁU KOMUNIKACYJNEGO

Odbieramy znak i umieszczamy go w buforze. Następnie sprawdzamy czy podany znak to początek ramki, jeżeli tak to rozpoczyna się proces jej odbierania.
W trakcie procesu odbierania sprawdzamy czy ramka nie jest zbyt długa oraz czy Rx.ReceivedCharIdx nie wykracza poza rozmiar bufora. Jeżeli wykracza to należy ustawić Rx.ReceivedCharIdx na 0, tak aby bufor robił kółko po tablicy. Jeżeli w trakcie odbierania zostanie wysłany kolejny znak rozpoczęcia ramki to cały cykl komplementowania ramki rozpoczyna się od nowa. Proces odbierania ramki zostanie zakończony na pierwszym znaku końca ramki, który otrzyma. Po odebraniu znaku końca program sprawdza czy nasza ramka posiada minimalna długość jaką jest 9 znaków. Następnie program sprawdza czy w naszej ramce nie znajduje się inny znak końca ramki lub pusty znak. Następnie program sprawdza czy na pewno znakami checksumy są znaki reprezentujące liczby. Jeśli spełniły się wszystkie warunki które podałem wyżej to rozpoczyna się proces liczenia i sprawdzania checksumy. Jeśli checksuma się zgadza to program kopiuje dane, a następnie wysyła komunikat zwrotny użytkownikowi.

```
while (1)
{
    if(Rx.ReceivedCharIdx != Rx.RXbuffIdx)                  // Sprawdzamy czy jest jakiś znak do odebrania.
      {
        frameChar = Rx.array[Rx.ReceivedCharIdx];           // Pobieramy znak z bufora.
        Rx.ReceivedCharIdx++;                               // Przechodzimy na kolejne miejsce.

        if(Rx.ReceivedCharIdx >= sizeOfBuffer)              // Sprawdzamy czy wskaźnik nie wykracza poza bufor.
            {
               Rx.ReceivedCharIdx = 0;                      // Jeśli tak, to ustawiamy go na początek bufora.
            }

        if(frameChar == 0x24)                               // Sprawdzamy czy znak początku ramki to $
            {
               frameReceiving = 1;                          // Jeśli tak, to odbieranie na 1
               frameLength = 0;                             // Jeśli tak to długość ramki na 0
            }
        if(frameReceiving == 1)                             // Jeśli wykryliśmy znak początku
            {
               frame[frameLength] = frameChar;              // To pobieramy znak do ramki
               frameLength++;                               // I przechodzimy dalej
               if (frameLength > 108)                       // Sprawdzamy czy ramka nie jest za długa
                   {
                      tooLongFrame();                       // Reagujemy na zbyt długa ramke
                      frameReceiving = 0;                   // Odbieranie na 0
                      frameLength = 0;                      // Długosc ramki na 0
                   }
        if(frameChar == 0x23)                               // Sprawdzamy znak konca ramki #
           {
            if(frameLength > 8)                             // Sprawdzamy minimalna długosc ramki
               {
                   for (int i = 1; i <= 7; i++) {
                      if ((!frame[i]) || frame[i] == 0x23)  // Sprawdzamy czy dane miejsce nie jest puste lub nie wystepuje znak # w tym miejscu
                      {                                     // Jesli wykryjemy brak znaku lub znak konca ramki
                         processing = false;                // to nie bedziemy wysylac ramki
                      }
                   }
                   for (int i = 4; i < 7; i++) {            // Sprawdzamy czy na pozycji
                      if (!isdigit(frame[i])) {             // znakow checksumy znajduja sie znaki liczb
                        processing = false;                 // Jesli nie to nie bedziemy wysylac ramki
                      }
                   }
                   if (processing)
                      {
                         memcpy(command, &frame[1], 3);     // Kopiujemy komende do zmiennej command
                         memcpy(tempstring, &frame[4], 3);  // Kopiujemy checksume do zmiennej tempstring
                         checksum = atoi((const char *)tempstring);  // Konwertujemy checksume na wartosc decymalna
                         for (int offset = 0; offset < 3; ++offset)
                         {
                             l_checksum += command[offset]; // Konwertujemy pobrane znaki z command do zmiennej l_checksum
                         }
                         if (l_checksum == checksum)
                         {
                             memcpy(frameData, &frame[7], 100);  // Kopiujemy znaki z ramki do zmiennej frameData
                             commandsReaction();            // Wywolujemy funkcje odpowiedzialna za wykonanie komendy
                             USART_Send("\r\n");
                         }
                         else
                         {
                             badChecksum();                 // Reagujemy na bledna checksume
                         }
                      }
                         else
                         {
                             badFrame();                    // Reagujemy na bledna komende
                         }
                         memset(frameChar, 0, 100);         // Czyscimy dane ramki
                         tempstring[CHECKSUM_LEN] = 0;      // Czyscimy zapasowej tablicy na 0
                         l_checksum = 0;                    // Długosc policzonej checksumy na 0
                         checksum = 0;                      // Długosc checksumy na 0
                         frameLength = 0;                   // Długosc ramki na 0
                         frameReceiving = 0;                // Zmienna wykrywajaca znak poczatku ramki na 0
                         processing = true;                 // Zmienna odpowiedzialna za proces na 0

               } else {
                   tooShortFrame();
               }
           }
        }
      }
}
```

Funkcje odpowiedzialne za odesłanie ramki

Funkcja broadcastingFrame odpowiadająca za stworzenie ramki zwrotnej do użytkownika, wraz z danymi, które otrzyma w wyniku wywołania innej funkcji.

```
void broadcastingFrame(char data[]) { // Ramka, nadawanie
    char message[105];
    strcat(message,data);
    strcat(message,";");
    USART_Send(message);
}
```

Funkcja command odpowiadająca za analizę komendy ramki w następstwie wywołania odpowiedniej funkcji reagującej na daną komendę.

```
void commandsReaction(){ // Ramka, analiza i wykonanie polecenia
        if (command[0] == 'B'  && command[1] == 'F' && command[2] == 'F'){ infoFrame(); }
        else if (command[0] == 'B' && command[1] == 'F' && command[2] == 'L'){ infoFrame(); }
        else if (command[0] == 'I' && command[1] == 'D' && command[2] == 'X'){ infoFrame(); }
        else if (command[0] == 'C' && command[1] == 'L' && command[2] == 'R'){ infoFrame(); }
        else { badFrame(); USART_Send("\r\n"); }
}
```

Funkcja infoFrame reagująca na komendę ramki.

```
void infoFrame(){ // Ramka, wywolywanie komendy
    USART_Send("Komenda: ");
    USART_Send(command);
    USART_Send("\r\n");
}
```

Funkcja badFrame reagująca na niepoprawną komendę ramki.

```
void badFrame(){ // Ramka, wywolywanie bledu
    char message[105] = "Error! Znaleziono niedozwolone znaki.";
    USART_Send(message);
    USART_Send("\r\n");
}
```

Funkcja badChecksum reagująca na niepoprawną checksumę.

```
void badChecksum(){ // Ramka, wywolywanie bledu
    char message[105] = "Error! Nieprawidlowa checksuma.";
    USART_Send(message);
    USART_Send("\r\n");
}
```

Funkcja tooLongFrame reagująca na zbyt długą komendę ramki.

```
void tooLongFrame(){ // Ramka, reagowanie na przekroczenie limitu znakow
    char message[105] = "Error! Przekroczono limit znakow ramki.";
    USART_Send(message);
    USART_Send("\r\n");
}
```

Funkcja tooShortFrame reagująca na zbyt krótką ramkę.

```
void tooShortFrame(){ // Ramka, reagowanie na przekroczenie limitu znakow
    char message[105] = "Error! Minimalna dlugosc ramki to 7 znakow.";
    USART_Send(message);
    USART_Send("\r\n");
}
```