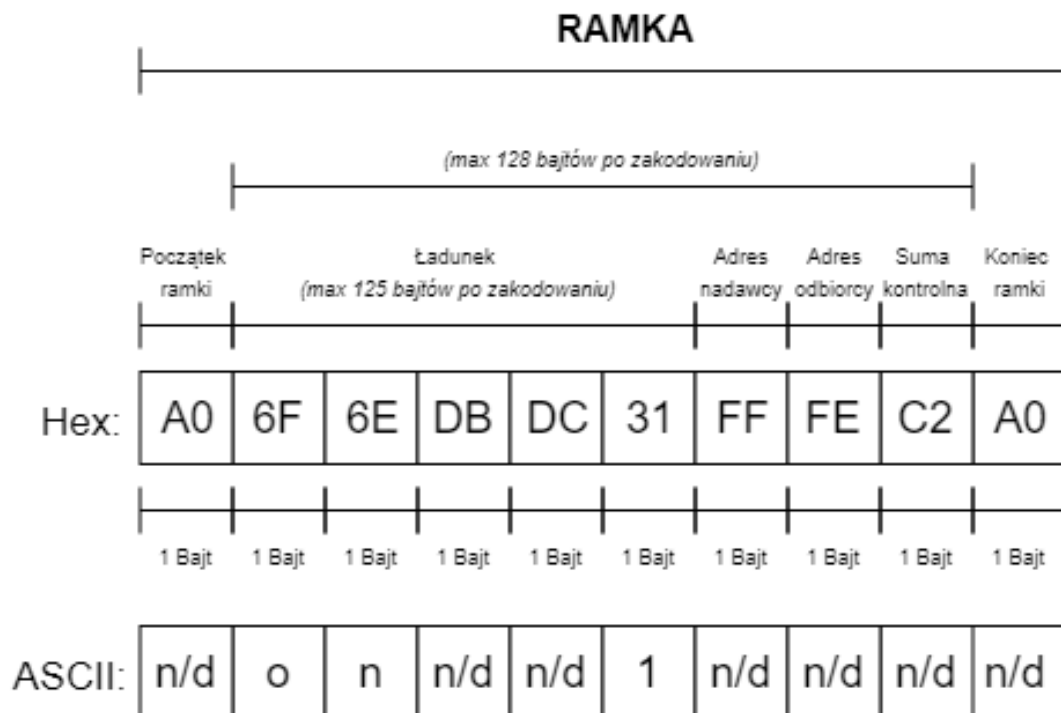
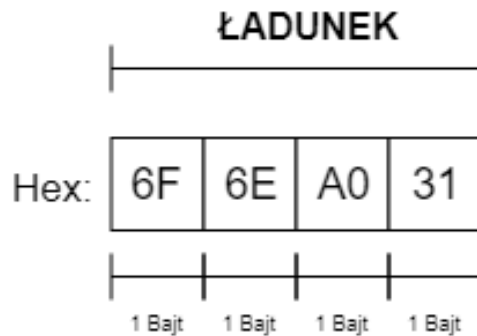


## Protokół komunikacyjny

Nazwisko, imię, indeks	Sulecki Przemysław (111010)
Założenia	<ol style="list-style-type: none"> <li>1. Jednostką informacji jest <b>bajt</b>.</li> <li>2. Bajt początku i końca ramki: <b>0xA0</b>.</li> <li>3. Jeżeli w ładunku wystąpi znak <b>0xA0</b>, należy go zmodyfikować na <b>0xDB 0xDC</b>.</li> <li>4. Jeżeli w ładunku wystąpi <b>0xDB</b>, należy go zmodyfikować na <b>0xDB 0xDD</b>.</li> <li>5. Przedostatni bajt to suma kontrolna: <b>suma kontrolna = (ładunek + adres nadawcy + adres odbiorcy) % 256</b>.</li> <li>6. Szybkość transmisji: <b>9600 bitów na sekundę</b>.</li> <li>7. Czas wysłania jednego bitu: <math>1/9600 = \sim 104 \mu s</math>.</li> <li>8. Czas wysyłania jednego bajta: <math>104 \mu s * 10 \text{ bitów} = 1040 \mu s</math>.</li> <li>9. Czas sprawdzania ciszy na linii: <math>5 \text{ bajtów} * 1040 \mu s = 5,2 \text{ ms}</math>.</li> </ol>

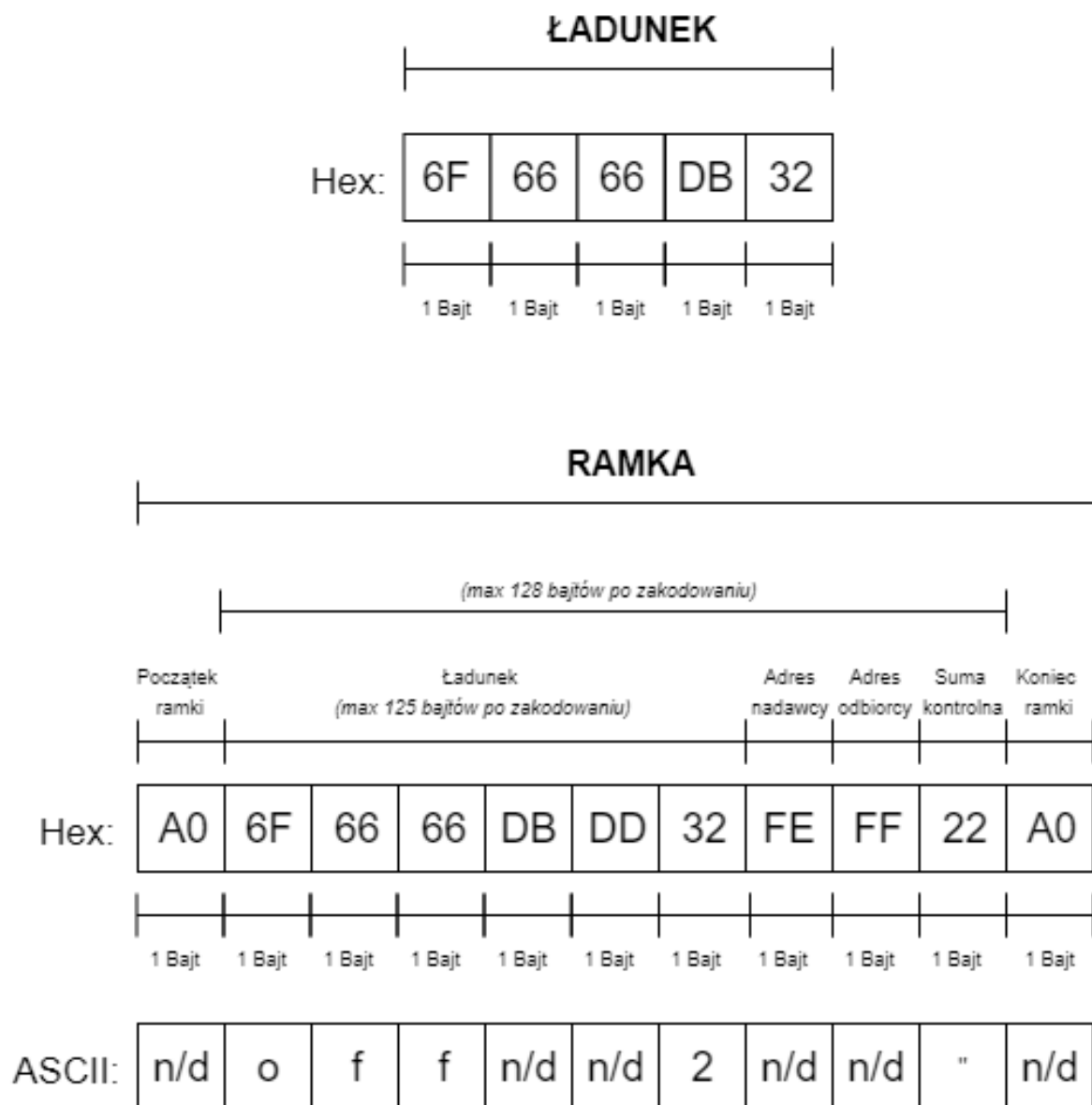
### 1. Przykład 1:

Wysyłanie ramki z terminala PC do MCU.



## 2. Przykład 2:

Wysyłanie ramki z MCU do PC.



### 3. Implementacja w języku C:

#### 3.1. Plik nagłówkowy: *UART.h*

```
#ifndef UART_H_
#define UART_H_

#include "main.h"
#include <stdarg.h>
#include <math.h>
#include <string.h>

/* Function prototypes */
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *);
void UART_AddChar(char);
char UART_GetChar();
void UART_Send_Tx(char *, ...);
void DecodeFrame();
int SendFrame(char *, ...);
void AnalyzeFrame();
void AppendChar(char *, char);
uint8_t CheckSum();
void DoCommand(char *);
void ClearPayload();
void ClearChecksumAndAddresses();

#define BUFF_SIZE      130
#define WAIT_HEADER    0
#define IN_MSG         1
#define AFTER_ESC      2
#define END             0xA0
#define ESC             0xDB
#define ESC_END         0xDC
#define ESC_ESC         0xDD

#endif /* UART_H_ */
```

#### 3.2. Plik źródłowy: *UART.c*

Procedura dodająca znak do bufora. Wykorzystywana przy tworzeniu ramki wysyłanej z MCU do PC:

```
void AppendChar(char *frame, char c) {
    int frame_len = strlen(frame);
    if(frame_len < 130)
        frame[frame_len] = c;
}
```

Funkcja tworząca ramkę, która jest wysyłana z MCU do PC:

```
int SendFrame(char *text, ...) {
    char payload[130] = {0};
    va_list valist;
    va_start(valist, text);
    vsprintf(payload, (char *)text, valist);
    va_end(valist);

    uint32_t payload_len = strlen(payload);
    uint32_t sum_byte = 0;
    char frame[130] = {0};

    AppendChar(frame, END);
```

```

if(payload_len <= 125) {
    for(int i = 0; payload[i] != '\0'; i++) {
        switch(payload[i]) {
            case END:
                sum_byte += (ESC + ESC_END);
                AppendChar(frame, ESC);
                AppendChar(frame, ESC_END);
                break;
            case ESC:
                sum_byte += (ESC + ESC_ESC);
                AppendChar(frame, ESC);
                AppendChar(frame, ESC_ESC);
                break;
            default:
                sum_byte += payload[i];
                AppendChar(frame, payload[i]);
        }
    }
} else {
    SendFrame("Error: Buffer overflow");
    return -1;
}

AppendChar(frame, 0xFE);
AppendChar(frame, 0xFF);

sum_byte += 0xFE + 0xFF;

AppendChar(frame, (char)(sum_byte % 256));
AppendChar(frame, END);

UART_Send_Tx("%s", frame);

return 0;
}

```

Funkcja sprawdzająca sumę kontrolną dołączoną do danych wysłanych z PC:

```

uint8_t CheckSum() {
    int sum_byte = payload[received-1];
    uint32_t result = 0;

    for(int i = 0; i < received-1; i++) {
        if(payload[i] == END)
            result += ESC + ESC_END;
        else if(payload[i] == ESC)
            result += ESC + ESC_ESC;
        else
            result += payload[i];
    }

    result %= 256;

    return (sum_byte == result ? 1 : 0);
}

```

Procedura, która czyści ładunek w przypadku zakończenia przetwarzania ramki lub jej odrzucenia:

```

void ClearPayload() {
    for(int i = 0; i < received; i++)
        payload[i] = '\0';

    received = 0;
}

```

Procedura, która czyści bufor z adresów nadawcy i odbiorcy oraz sumy kontrolnej:

```
void ClearChecksumAndAddresses() {
    for(int i = received - 1; i > received - 4; i--)
        payload[i] = '\0';
}
```

Procedura, która analizuje ramkę. Sprawdza czy ramka nie jest pusta, wywołuje procedurę sprawdzającą sumę kontrolną, zapisuje adresy nadawcy i odbiorcy ramki oraz wywołuje wykonanie komendy:

```
void AnalyzeFrame() {
    if(received > 0) {
        if(CheckSum()) {
            sender_byte = payload[received - 3];
            receiver_byte = payload[received - 2];
            ClearChecksumAndAddresses();
            DoCommand(payload);
        } else
            SendFrame("Error: Invalid checksum");
    } else
        SendFrame("Error: empty frame");
}
```

Procedura dekodująca ramkę:

```
void DecodeFrame() {
    char c = 0;

    if(EMPTY_RX != BUSY_RX) {

        c = UART_GetChar();
        __HAL_TIM_SET_COUNTER(&htim4, 0);

        switch(receiver_state) {
            case WAIT_HEADER:
                if(c == END) {
                    ClearPayload();
                    receiver_state = IN_MSG;
                    HAL_TIM_Base_Start_IT(&htim4);
                }
                break;
            case IN_MSG:
                if(c == ESC) {
                    receiver_state = AFTER_ESC;
                } else if(c == END) {
                    AnalyzeFrame();
                    ClearPayload();
                    receiver_state = IN_MSG;
                } else
                    if(received < 128)
                        payload[received++] = c;
                    else {
                        SendFrame("Error: Buffer overflow");
                        ClearPayload();
                        receiver_state = WAIT_HEADER;
                        HAL_TIM_Base_Stop_IT(&htim4);
                    }
                break;
            case AFTER_ESC:
                switch(c) {
                    case ESC_END:
                        c = END;
                        break;
                    case ESC_ESC:
                        c = ESC;
                        break;
                }
                break;
        }
    }
}
```

```

        default:
            ClearPayload();
            receiver_state = WAIT_HEADER;
            HAL_TIM_Base_Stop_IT(&htim4);
        }
        if(received < 128) {
            payload[received++] = c;
            receiver_state = IN_MSG;
        }
        else {
            SendFrame("Error: Buffer overflow");
            ClearPayload();
            receiver_state = WAIT_HEADER;
            HAL_TIM_Base_Stop_IT(&htim4);
        }
        break;
    }
}
}

```

### 3.3. Plik źródłowy: *main.c*

Sprawdzanie ciszy na linii:

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {

    // USART
    static uint8_t period = 0;
    if(htim->Instance == TIM4) {
        if(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_IDLE) == SET) {
            period++;
            if(period > 5) {
                period = 0;
                receiver_state = WAIT_HEADER;
                HAL_TIM_Base_Stop_IT(&htim4);
            }
        }
    }
}

```

Procedura sprawdzająca czy ładunek odpowiada odpowiedniej komendzie:

```

void DoCommand(char *payload) {
    if(sscanf(payload, "freq:%d", &measure_frequency) == 1) {
        MeasureFrequency();
        SendFrame("%s", payload);
    } else if(sscanf(payload, "sound:%d:%d", &sound_frequency, &sound_level) == 2) {
        SoundLevelFreq();
        SendFrame("%s", payload);
    } else
        SendFrame("Error: Command not found");
}

```

## 4. Program do generowania ramki:

W celu szybkiego przetestowania ramki, napisałem program w języku Python, który generuje i kopiuje ramkę do schowka systemowego, dzięki czemu łatwo można wkleić bajty do terminala. Program należy wywołać z parametrem, którym jest ładunek, który chcemy wysłać:

```

import sys, pyperclip

cmd = str(sys.argv[1])

result = "a0 "
checksum = 0

```

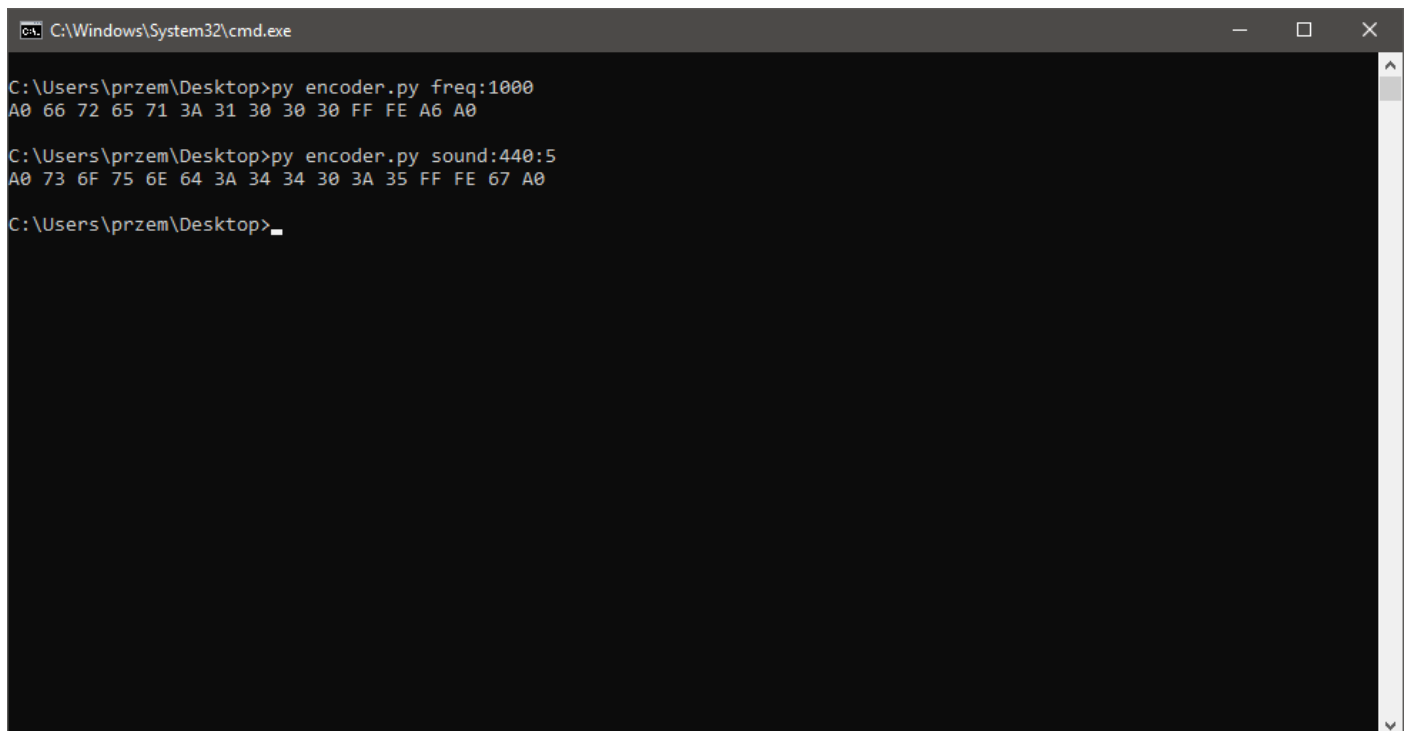
```
for c in cmd:
    result += hex(ord(c))[2:] + " "
    checksum += ord(c)

result += "ff fe "
checksum += 0xFF + 0xFE
result += hex(checksum % 256)[2:]
result += " a0"

pyperclip.copy(result)

print(result.upper())
```

Działanie:



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The prompt is at "C:\Users\przem\Desktop>". The user has entered two commands: "py encoder.py freq:1000" and "py encoder.py sound:440:5". The first command outputs the hexadecimal string "A0 66 72 65 71 3A 31 30 30 30 FF FE A6 A0". The second command outputs "A0 73 6F 75 6E 64 3A 34 34 30 3A 35 FF FE 67 A0". The prompt is now at "C:\Users\przem\Desktop>\_".

```
C:\Windows\System32\cmd.exe
C:\Users\przem\Desktop>py encoder.py freq:1000
A0 66 72 65 71 3A 31 30 30 30 FF FE A6 A0

C:\Users\przem\Desktop>py encoder.py sound:440:5
A0 73 6F 75 6E 64 3A 34 34 30 3A 35 FF FE 67 A0

C:\Users\przem\Desktop>_
```