**A black and white logo

Description automatically generated**

**Projektowanie sieci typu Fieldbus**

**Zadanie projektowe nr 2: CRC ramki Modbus RTU**

Szustkiewicz Krzysztof

Ivan Kaliankovich

Sprawozdanie

Spis treści

[1) Wybrany algorytm 2](#_Toc150094960)

[2) Sposób pomiaru czasu 2](#_Toc150094961)

[3) Kod żródłowy 2](#_Toc150094962)

[4) Przykłady wyniku 2](#_Toc150094963)

[5) Wnioski 2](#_Toc150094964)

# Wybrany algorytm

Na początku wybrany przez nas został algorytm iteracyjny CRC-16, ponieważ standardowy algorytm CRC-16 zgodny z protokołem MODBUS jest już dość efektywny i nie ma powszechnie stosowanych algorytmów, które byłyby znacznie szybsze przy zachowaniu dokładności. Obliczenie CRC może zostać zaimplementowane w sposób rekurencyjny, natomiast zwykle takie podejście jest mniej wydajne oraz bardziej skomplikowane od metody iteracyjnej.   
W trakcie testowania zauważyliśmy jednak, że gdy chcemy obliczyć wiele sum CRC w krótkim czasie, to na większą wydajność pozwala algorytm tablicowy, dlatego postanowiliśmy pozostać przy **algorytmie tablicowym**.

# Sposób pomiaru czasu

Do pomiaru czasu została użyta biblioteka *chrono* z poniższych powodów:

a) Wysoka precyzja: chrono dostarcza bardzo precyzyjne narzędzia pomiaru czasu, co pozwala na dokładne określenie czasu wykonania algorytmów nawet w mikrosekundach lub nanosekundach. Biblioteka ta oferuje znacznie większą precyzję od biblioteki *time.*  
b) Kros-platformowość: Biblioteka chrono jest częścią standardu C++ i jest dostępna na wielu platformach, co oznacza, że możliwe jest korzystanie z tych samych metod pomiaru czasu niezależnie od systemu operacyjnego.  
c) Uniwersalność: chrono oferuje różne API do różnych zegarów czasu, takie jak **system\_clock(), steady\_clock() i high\_resolution\_clock(),** które mogą być dostosowane do konkretnych potrzeb pomiaru czasu.  
d) Prostota użycia: Dla pomiaru czasu wystarczy wywołać dwie funkcję: uruchomienia zegara oraz jego zatrzymania.  
e) Zabezpieczenia przed różnicami czasu systemowego: chrono dostarcza narzędzia do kontrolowania różnic czasu systemowego, co jest ważne w przypadku precyzyjnych pomiarów czasu na wielu platformach jednocześnie.

# Kod źródłowy

Poniżej przedstawiam kod inicjalizacji tablic oraz funkcji **getTableCRC()** liczącej sumę CRC metodą tablicową. Cały kod źródłowy można znaleźć tu: [[github](https://github.com/kalinkawf/Fieldbus/tree/master/proj2)](https://github.com/kalinkawf/Fieldbus/blob/master/proj2/Modbus.cpp)

// Wielomian generujący dla CRC-16 (Reversed polynomial: 0xA001)

const uint16\_t crc16\_polynomial = 0xA001;

// Tablice LowByte i HiByte CRC-16

uint8\_t crc16\_table\_low[256];

uint8\_t crc16\_table\_high[256];

// Inicjalizacja tablic CRC-16

void initializeCRCTables() {

    for (int i = 0; i < 256; i++) {

        uint16\_t crc = i;

        for (int j = 0; j < 8; j++) {

            if (crc & 1) {

                crc = (crc >> 1) ^ crc16\_polynomial;

            } else {

                crc >>= 1;

            }

        }

        crc16\_table\_low[i] = static\_cast<uint8\_t>(crc & 0xFF);

        crc16\_table\_high[i] = static\_cast<uint8\_t>((crc >> 8) & 0xFF);

    }

}

// Funkcja do obliczania CRC-16

uint16\_t getTableCRC(const std::vector<uint8\_t>& data) {

    uint8\_t crc\_low = 0xFF;

    uint8\_t crc\_high = 0xFF;

    for (uint8\_t byte : data) {

        uint8\_t index = crc\_low ^ byte;

        crc\_low = crc16\_table\_low[index] ^ crc\_high;

        crc\_high = crc16\_table\_high[index];

    }

    return (static\_cast<uint16\_t>(crc\_high) << 8) | static\_cast<uint16\_t>(crc\_low);

}

# Przykłady wyniku

Przykład wyniku obliczenia CRC dla sekwencji: efefefefe1231323323198 przy n = 1490323 powtórzeń algorytmu.   
a) Metoda iteracyjna:   
Obraz zawierający tekst, zrzut ekranu, Czcionka, czarne

Opis wygenerowany automatycznie

1.17274s

b) Metoda tablicowa przy identycznej sekwencji oraz ilości powtórzeń:   
Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne

Opis wygenerowany automatycznie

# 5) Wnioski

Metoda tablicowa pozwala na znaczące przyśpieszenie procesu w przypadku, gdy jest dużo powtórzeń. W sytuacji, gdzie nie ma dużo sum CRC do obliczenia algorytm iteracyjny jest preferowany, ponieważ jest bardzo prosty, nie wymaga dużo pamięci i jest wystarczająco szybki dla większości przypadków.