

I. Treść zadania.

Zadanie 5.1

Napisać aplikację wyznaczającą sumę kontrolną dla sieci CAN. Aplikacja ta ma być optymalizowana ze względu na minimalizację czasu realizacji algorytmu wyznaczania CRC.

Założenia:

- a) Aplikacja ma umożliwiać wprowadzenie z klawiatury ciągu do 96 bitów
- b) Aplikacja powinna wyznaczyć sumę kontrolną z wprowadzonego ciągu bitowego i wyświetlić ją w postaci heksadecymalnej
- c) Aplikacja powinna mieć możliwość deklarowania liczby powtórzeń obliczenia sumy kontrolnej od 1 do 10^9 .
- d) Aplikacja powinna mieć możliwość pomiaru i wyświetlania łącznego czasu realizacji zadeklarowanej liczby powtórzeń obliczenia sumy kontrolnej oraz średniego czasu jednokrotnej realizacji obliczenia CRC.
- e) Aplikacja powinna być dostarczona w postaci kodu maszynowego wykonywalnego.
- f) Aplikacja będzie testowana w środowisku Windows XP.

Zadanie 5.2

Jak wiadomo, węzły CAN są generatorami wiadomości. Każda wiadomość ma swój własny identyfikator. Zależnie od specyfikacji sieci CAN występują w specyfikacji standardowej identyfikatory 11 bitowe, a w specyfikacji rozszerzonej identyfikatory 29 bitowe.

W związku z tym teoretycznie możliwe jest zaadresowanie maksymalnie odpowiednio: 2^{11} lub 2^{29} wiadomości.

- Czy tak jest w rzeczywistości? Proszę podać istniejące ograniczenia w specyfikacji CAN na wartości identyfikatorów (o ile występują).

Zadanie 5.3

Dokładna analiza czasu transmisji ramki CAN wskazuje, że przy transmisji tej samej liczby bajtów pola danych, czas trwania transmisji ramki CAN może być różny. Czas ten zależy od kontekstu. Przyczyna leży w sposobie kodowania informacji, a w szczególności w zastosowanej technice synchronizacji transferów sieciowych wspieranych przez dodatkowe bity synchronizujące (technika *bit stuffing*). Technika bit stuffing jest przezroczysta dla użytkownika sieci CAN. Ma ona jednak bardzo istotny wpływ na czas trwania transmisji ramki. Trzeba podkreślić, że świadomość tego faktu jest wśród użytkowników sieci CAN znikoma.

Założenia:

- *prędkość transmisji w sieci CAN została ustalona na 1Mb/s,*
- *przesyłamy w polu danych 8 bajtów informacji,*
- *identyfikator wiadomości podlega swobodnemu wyborowi z ewentualnymi ograniczeniami (por. zadanie 5.2),*
- *stosujemy ramkę o identyfikatorze 29-bitowym,*
- *współczynnik efektywności transmisji zdefiniujemy jako stosunek liczby bitów wysłanej informacji (64 bity) do całkowitej liczby bitów ramki- łącznie z bitami przerwy między ramkami i bitami dodatkowymi dodanymi przez operację bit stuffing.*

Polecenia:

- a) należy wyznaczyć najkrótszy czas transmisji całej ramki CAN. Proszę podać przykład dla jakiej zawartości pola danych ma to miejsce. Proszę podać wartość współczynnika efektywności transmisji.
- b) należy wyznaczyć najdłuższy czas transmisji całej ramki CAN. Proszę podać przykład dla jakiej zawartości pola danych ma to miejsce. Proszę podać wartość współczynnika efektywności transmisji.

II. Zadanie 5.1

1. Algorytm obliczania sumy kontrolnej CRC.

W przypadku sieci CAN suma kontrolna CRC jest słowem piętnastobitowym, a wyznaczanie nowej wartości CRC następuje bit po bicie wiadomości (po destuffingu), poczynawszy od bitu początku ramki, poprzez bity pola arbitrażu, pola kontrolnego i skończywszy na ostatnim bicie pola danych. Różnicą w porównaniu do sieci Modbus jest właśnie wyznaczanie CRC bit po bicie (w sieci Modbus jest to bajt po bajcie). Inny jest też wielomian generacyjny, który dla sieci CAN jest równy $0x4599$. W przypadku sieci CAN stosowane są przesunięcia bitów w lewo (a nie prawo jak w sieci Modbus).

Niech $CRC_RG(14:0)$ oznacza rejestr przechowujący wartość sumy kontrolnej CRC, natomiast $NXTBIT$ oznacza kolejny bit transmitowanego ciągu bitów (po destuffingu) poczynawszy od bitu SOF, poprzez bity pola arbitrażu, pola kontrolnego i skończywszy na ostatnim bicie pola danych.

Dodatkowo ciąg bitów do wyznaczania CRC jest uzupełniany przez 15 bitów o wartości 0. Suma CRC obliczana jest następująco:

```
CRC_RG = 0;
REPEAT
    CRCNXT = NXTBIT EXOR CRC_RG(14)
    CRC_RG(14:1) = CRC_RG(13:0);
    CRC_RG(0) = 0;
    IF CRCNXT THEN
        CRC_RG(14:0) = CRC_RG(14:0) EXOR (4599hex);
    ENDIF
UNTIL (zostanie osiągnięte CRC SEQUENCE lub wystąpi jeden z błędów transmisji)
```

Co opisując słowami można opisać następująco:

- a) Utworzenie 15-bitowego rejestru o zawartości równej: $\langle 0 \rangle$.
- b) Ustalenie wartości operacji XOR bitu wiadomości z najstarszym bitem rejestru.
- c) Przesunięcie rejestru o 1 bit w lewo, wstawienie 0 w miejsce LSB.
- d) Jeśli XOR z pkt. b dał wynik „prawda”, to wykonaj XOR rejestru z wartością 4599 hex (0100 0101 1001 1001). Jeśli nie, to przejdź do kolejnego punktu.
- e) Powtórz punkty b:d do wykorzystania ostatniego bitu transmitowanego ciągu.
- f) Otrzymana wartość to CRC dla wiadomości w sieci CAN.

Należy pamiętać o tym, że w zależności od środowiska algorytm trzeba zmodyfikować. Np. w przypadku komputera PC z systemem Windows 7 najmniejszą jednostką w jakiej można zapisać informację jest rejestr 8 bitowy. Dlatego nie będzie dało się tworzyć rejestru 15 bitowego (bezpośrednio), a po punkcie e trzeba dodatkowo wartość MSB rejestru 16-bitowego ustawić jako 0.

2. Opis zrealizowanej aplikacji.

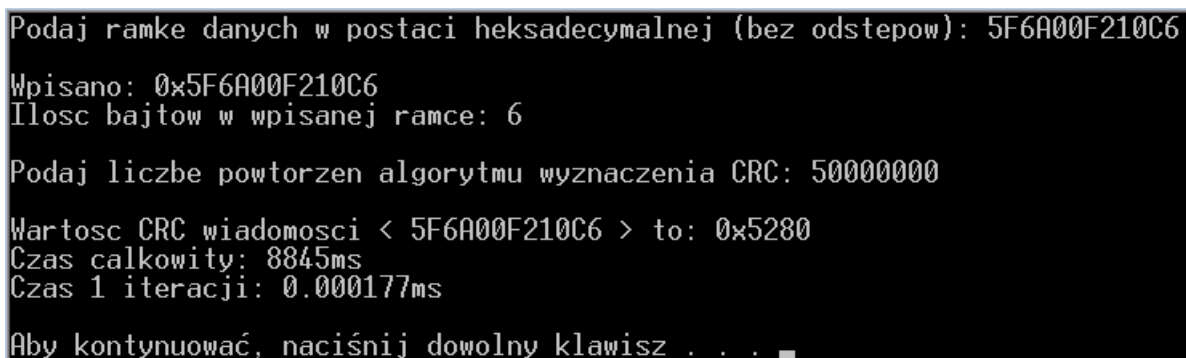
Aplikacja została wykonana w języku C++, w środowisku Code::Blocks ver 17.12. Ma ona postać wiersza poleceń.

W celu podania ramki danych należy wpisać jej wartość w postaci heksadecymalnej (bez spacji/odstępów), co będzie odpowiadało ciągowi bitowemu. Aplikacja zwraca wpisaną wartość i jej rozmiar w bajtach. Jeśli wpisany ciąg przekroczy 96 bitów (czyli 12 bajtów) lub będzie zawierał on znaki nie odpowiadające zapisowi heksadecymalnemu, to wyświetli się informacja o błędnym wpisaniu ramki danych.

Następnie należy podać liczbę powtórzeń algorytmu z przedziału $1 - 10^9$, po czym zostaje zrealizowane wyznaczenie CRC i jego pomiar czasu. Otrzymane wyniki są wyświetlane.

3. Uzyskane wyniki.

Za pomocą ciągu bajtów o wartości 0x5F 6A 00 F2 10 C6, przeprowadzono pomiar czasu realizacji wyznaczenia CRC. W tym celu przyjęto liczbę powtórzeń obliczenia CRC równą $n = 50\,000\,000$.



```
Podaj ramke danych w postaci heksadecymalnej (bez odstępów): 5F6A00F210C6
Wpisano: 0x5F6A00F210C6
Ilość bajtów w wpisanej ramce: 6

Podaj liczbę powtórzeń algorytmu wyznaczenia CRC: 50000000

Wartość CRC wiadomości < 5F6A00F210C6 > to: 0x5280
Czas całkowity: 8845ms
Czas 1 iteracji: 0.000177ms

Aby kontynuować, naciśnij dowolny klawisz . . . █
```

Rysunek 1 – Wynik testowania zrealizowanej aplikacji do wyznaczenia CRC

Aby mieć pewność co do poprawności wyznaczonej wartości CRC, ten sam ciąg bajtów został sprawdzony za pomocą kalkulatora CRC, dostępnego na stronie internetowej pod adresem: <http://www.ghsi.de/pages/subpages/Online%20CRC%20Calculation/>.

W celu zapewnienia algorytmu zgodnego z siecią CAN, algorytm musi wykorzystywać wielomian generacyjny w postaci:

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$$

(źródło: https://en.wikipedia.org/wiki/Cyclic_redundancy_check)

Co dla 15-bitowego CRC da heksadecymalną wartość: 0x4599. Jednak aby kalkulator poprawnie wyliczyć CRC wielomian musi mieć wartość 0xc599. Jest tak ponieważ w wielomianie występuje składnik x^{15} , który „znika” dla 15-bitowego rejestru, ale dla pamięci komputera i algorytmu jest konieczny.

Pod następującym adresem internetowym:
<http://www.ghsi.de/pages/subpages/Online%20CRC%20Calculation/index.php?Polynom=1100010110011001&Message=5F6a00f210c6>, dostępny jest efekt wyliczenia CRC - CAN dla ciągu bitowego równego 0x5F6A00F210C6.

Poniżej przedstawiono wynik wyznaczenia CRC za pomocą kalkulatora:

Be careful: there are several ways to realize a CRC. They differ (at least) in the way which bit is shifted in first and also in the initialization of the flipflops.

Enter your CRC polynomial as bit sequence ("100110001") here:

This gives the following CRC polynomial (press RETURN to update):

$$P(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + x^0$$

Enter your message as sequence of hex bytes here. Don't care about whitespaces since they will be ignored.

Press RETURN or the Calculate button below to see the CRC checksum here:

\$ 5280 (hexadecimal)
% 1010010100000000 (binary, see [calculation details here](#))
! 21120 (decimal)

Rysunek 2 – Wynik wyznaczenia CRC za pomocą internetowego kalkulatora

Jak widać wartość CRC jest identyczna, jak uzyskana za pomocą aplikacji, co świadczy o poprawnym wykonaniu algorytmu wyznaczania sumy kontrolnej CRC dla sieci CAN.

Średni czas wyznaczenia wartości CRC dla przyjętego ciągu bajtów wyniósł 0,177 μ s.

Jednak wartość CRC może być aktualizowana po każdym bicie wiadomości (a nie wyliczana po otrzymaniu całego ciągu bitów). Aby oszacować średni czas aktualizacji wartości CRC dla pojedynczego bitu, można podzielić uzyskany średni czas wyznaczenia CRC dla przyjętego ciągu bitów przez ilość bitów w ciągu. Da to następujący wynik:

$$0,177 [\mu\text{s}] / [6 * 8] \approx 0,0037 [\mu\text{s}]$$

Jeśli odniesiemy się do sieci CAN o prędkości 1 Mb/s (najwyższa dostępna prędkość), gdzie czas między transmitowanymi bitami będzie równy: $1 [\text{s}] / 10^6 = 1 [\mu\text{s}]$, to wyznaczenie CRC nawet całego ciągu zmieści się w czasie transmisji pojedynczego bitu. Świadczy to o poprawnej realizacji algorytmu wyznaczenia CRC dla sieci CAN.

III. Zadanie 5.2

Nie jest możliwe zaadresowanie 2^{11} lub 2^{29} wiadomości, ponieważ identyfikator nie może przyjmować wartości recesywnych jednocześnie na siedmiu najbardziej znaczących bitach (ID-28 ... ID-22). Wynika z tego, że któryś z pierwszych siedmiu bitów musi zawierać wartość dominującą (logiczne 0). Wtedy otrzymuje się $2^{11} - 2^4 = 2032$ możliwych identyfikatorów dla ramki podstawowej. Dla ramki rozszerzonej będzie to 532 676 608 możliwych identyfikatorów.

IV. Zadanie 5.3

Założenia:

- *prędkość transmisji w sieci CAN została ustalona na 1Mb/s,*
- *przesyłamy w polu danych 8 bajtów informacji,*
- *identyfikator wiadomości podlega swobodnemu wyborowi z ewentualnymi ograniczeniami (por. zadanie 5.2),*
- *stosujemy ramkę o identyfikatorze 29-bitowym,*
- *współczynnik efektywności transmisji zdefiniujemy jako stosunek liczby bitów wysłanej informacji (64 bity) do całkowitej liczby bitów ramki- łącznie z bitami przerwy między ramkami i bitami dodatkowymi dodanymi przez operację bit stuffing.*

a) Wyznaczenie najkrótszego czasu transmisji ramki CAN.

Najkrótszy czas transmisji całej ramki CAN będzie wtedy, gdy nie zostaną dodane żadne bity przez bit stuffing (bit stuffing zmienia stan na przeciwny gdy wystąpi sekwencja 5 kolejnych bitów o takiej samej wartości logicznej). Bity w ramce muszą więc zmieniać swoją wartość przynajmniej co 4 bit. Pola CRC DELIMITER, ACK FIELD i END OF FRAME nie podlegają procedurze bit stuffingu.

Dla sieci CAN o prędkości transmisji 1 Mb/s, czas transmisji pojedynczego bitu jest równy **1 μ s**.

Przyjęta do rozważań ramka CAN jest rozszerzona i składa się z:

- Początku ramki: **1 bit** – logiczne „0”
- Pole arbitrażu: składa się z **32 bitów**: 29-bitowy identyfikator i 3 bity zwane: SRR-BIT, IDE-BIT i RTR-BIT
- Pole kontrolne: składa się z **6 bitów**: 2 rezerwowe bity r1 i r0 (logiczne „0”) i 4-bitowy kod długości pola danych DLC (wartość: „1000”)
- Pole danych: składa się z **64 bitów**: 8 bajtów informacji,
- Pole sumy kontrolnej: składa się z **16 bitów** : 15-bitowe CRC i znacznik końca pola CRC (1 bit o wartości recesywnej – logiczna 1)
- Pole potwierdzenia (ACK FIELD): składa się z **2 bitów** : ACK SLOT (wartość zależy od tego czy nadajnik lub odbiornik) i ACK DELIMITER (zawsze recesywny)
- Koniec ramki: **7 bitów** - wszystkie to logiczne „1” – stan recesywny

Łącznie przyjęta ramka CAN składa się z 128 bitów. Jej przykładowa zawartość (zgodna z rozważaniami) jest pokazana poniżej (nie sprawdzono zgodności pola danych i CRC):

SOF	Pole arbitrażu	Pole kontrolne	Pole danych	Pole sumy kontrolnej	Pole potwierdzenia	EOF
0	0100 1000 100 11 1000 1000 1001 0101 01 0	00 1000	1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000	0100 1001 0001 0001	00	111 1111

Skoro ramka składa się z 128 bitów i przyjęto, że nie zostanie dodany żaden bit w wyniku bit stuffingu, wtedy czas transmisji takiej ramki wyniesie: **128 μ s.**

Współczynnik efektywności transmisji wyniesie: $64 / 128 = 0,5$

b) Wyznaczenie najdłuższego czasu transmisji ramki CAN.

Najdłuższy czas transmisji wystąpi wtedy, gdy poprzez bit stuffing zostanie przesłana największa możliwa ilość dodatkowych bitów (powtórzy się sekwencja 5 takich samych bitów). Trzeba też pamiętać o dodatkowych ograniczeniach, jak np. identyfikator nie może przyjmować wartości recesywnych jednocześnie na siedmiu najbardziej znaczących bitach.

Poniżej pokazano jak teoretycznie może wyglądać taka ramka (w nawiasach podano dodatkowe bity bit-stuffingu)

SOF	Pole arbitrażu	Pole kontrolne	Pole danych	Pole sumy kontrolnej	Pole potwierdzenia	EOF
0	0000(1) 1111 1(0)00 11 111(0)0 0000 (1)0000 0(1)111 01 0	00 1000	00(1)00 000(1)1 1111(0) 0000 0(1)000 00(1)00 000(1)0 0000(1) 0000 0(1)000 00(1)00 000(1)0 0000(1) 0000 0(1)000 00(1)00	1100 1101 1110 0011	00	111 1111

Pojawiło się 18 dodatkowych bitów, więc łącznie do przesłania ramki jest potrzebny czas transmisji 146 bitów, czyli **146 μ s.**

Współczynnik efektywności transmisji wyniesie: $64 / 146 \approx 0,438$.

V. Wnioski.

- Zrealizowana aplikacja wyznacza CRC w czasie nie przekraczającym czasu wysłania 1 bitu w najszybszym wariantcie sieci CAN (1Mb/s). Aplikacja ta powinna też być w stanie rozróżnić bity dodatkowe, pojawiające się w wyniku bit stuffingu (o ile będzie obrabiała dane bezpośrednio z magistrali CAN) – jednak to nie było poleceniem zadania.
- Algorytm wyznaczenia CRC w sieci CAN jest bardzo podobny do tego z sieci Modbus.
- Czas transmisji ramki w sieci CAN nie jest stały. Może się on zmieniać w zależności od ilości bajtów w polu danych, a także od tego jak często zostanie wykorzystany bit stuffing. Wpływ ma też rodzaj ramki: podstawowa czy rozszerzona.
- W porównaniu do sieci Modbus RTU, sieć CAN ma mniejszy współczynnik efektywności transmisji.