CRC doda Ci pewności, część 5

To już ostatni odcinek kursu. Jak się przekonamy cały trud związany z przedzieraniem się przez meandry wiedzy, o której nie można powiedzieć żeby była „lekkostrawna“, bardzo się teraz przyda. Procedury obliczeniowe wydają się prostsze niż można by początkowo przypuszczać. Potwierdzają to zamieszczone przykłady.

W poprzednim odcinku mieliśmy okazję prześledzić program napisany w języku C, realizujący parametryczną metodę obliczania CRC. Wszystko wyglądało dość poważnie. Sama procedura obliczająca CRC może być - przy wprowadzeniu pewnych założeń wstępnych - znacznie uproszczona. Oczywiście usprawiedliwieniem dla poprzednich rozważań był zamiar stworzenia programu uniwersalnego. Tym razem zdecydujmy się na jedną (konkretną) wersję metody: normalną albo odwróconą. Jeśli przyjmiemy metodę normalną, będzie to odpowiadało nadaniu zmiennym *refin* i *refot* wartości FALSE. Dla metody odwróconej obydwie zmienne mają wartość TRUE. Założenia takie pozwalają zrezygnować z powyższych zmiennych kosztem napisania dwóch niezależnych procedur. Nasz generator będzie zaszyty w tablicy. Pozostałe parametry: INIT i XOROT mogą być zdefiniowane jako makra. Procedura 32-bitowa, normalna będzie więc wyglądała jak na **list. 3**, odwrócona zaś jest przedstawiona na **list. 4**.

Generowanie tablic wykorzystywanych w opisywanych algorytmach (*Lookup Table*)

My tu sobie gadu-gadu, rozpatrujemy różne warianty, raz coś skomplikujemy, raz coś uprościmy, a tym czasem nie mamy najważniejszego elementu naszego programu, czyli tablic służących do obliczeń. Tablice takie mogą być obliczane w biegu przez funkcję *cm_tab*, mogą być też obliczane wstępnie i wstawiane do kodu programu pisanego w języku C. Zwróćmy uwagę na to, że w każdym przypadku będą one zależały jedynie od parametrów POLY oraz REFIN (i REFOUT). Uściślając: jeśli zdecydujemy się na konkretną metodę, to będziemy mogli wygenerować jedynie odpowiednią wersję tablicy (normalną lub odwróconą). Na **list. 5** przedstawiono program generowania 16- lub 32-bitowej tablicy CRC (*CRC Lookup Table*).

List. 3. Procedura normalna obliczania CRC

```
unsigned long crc_normal ();
unsigned long crc_normal (blk_adr,blk_len)
unsigned char *blk_adr;
unsigned long blk_len;
{
 unsigned long crc = INIT;
 while (blk_len--)
  crc = crctable[((crc>>24) ^ *blk_adr++) & 0xFFL] ^ (crc << 8);
 return crc ^ XOROT;
}
```

List. 4. Procedura odwrócona obliczania CRC

```
unsigned long crc_reflected ();
unsigned long crc_reflected (blk_adr,blk_len)
unsigned char *blk_adr;
unsigned long  blk_len;
{
 unsigned long crc = INIT_REFLECTED;
 while (blk_len--)
   crc = crctable[(crc ^ *blk_adr++) & 0xFFL] ^ (crc >> 8));
 return crc ^ XOROT;
}
```

Krótkie podsumowanie poznanych metod

Jako uzupełnienie dotychczasowych przykładów na **list. 6** i **7** prezentujemy (w wersji oryginalnej) jeszcze dwie wersje gotowych procedur służących do obliczeń CRC. W tym momencie właściwie możemy uznać, że dobrnęliśmy do końca kursu. Zaczynaliśmy od metod najprostszych, wykorzystujących niemal naturalne przełożenie teorii na język programowania, by w kolejnych krokach coraz bardziej je komplikować. Nie była to jednak sztuka dla sztuki. Zamierzeniem było uzyskanie jak największej wydajności algorytmów, a więc i szybkości działania końcowych programów. Doszliśmy do wniosku, że najlepsze rezultaty osiągniemy metodami tablicowymi. Ich wadą jest niestety zajmowanie sporej części pamięci programu. Wyboru należy dokonywać w zależności od potrzeb i możliwości w konkretnych przypadkach.

Metod sprawdzania poprawności transmisji obliczających CRC nie będziemy zapewne stosować w prostych aplikacjach, w których ewentualna utrata danych nie będzie stanowiła wielkiego problemu. Decyzja o rezygnacji będzie tym bardziej zasadna, jeśli będziemy mieli do czynienia

Słowniczek

Suma kontrolna - liczba wyliczona na podstawie treści pewnego komunikatu (rozumianego tutaj jako ciąg bajtów, niekoniecznie reprezentującego informacje tekstowe) i dołączana do niego w celu późniejszej kontroli prawidłowości odczytu. Przykładowe zastosowania: transmisja danych, rejestracja danych na nośnikach magnetycznych itp. Słowo „suma" niekoniecznie musi być rozumiane w dosłownym znaczeniu. Najczęściej będą to bardziej wyrafinowane metody obliczeniowe.

CRC - „Cyclic Redundancy Code" - akronim określający metody kontrolowania poprawności odczytu danych bazujące na dzieleniu wielomianów.

Komunikat, wiadomość (*message*) - dane wejściowe, których poprawność transmisji, zapisu itp. będzie sprawdzana w podczas odczytywania. Nie koniecznie rozumiane w dosłownym znaczeniu - jako informacja tekstowa. Najczęściej będą to struktury zorganizowane w sekwencje bajtów.

Generator (*poly*) - robocza nazwa „wielomianu generującego", będącego podstawowym parametrem algorytmów obliczania CRC.

Wielomian generujący (*polynomial*) - wielomianem generującym nazywamy dzielnik używany podczas obliczeń CRC. Jak już wiadomo metody stosowane do tego celu bazują na operacji dzielenia wielomianów.

Liczba odwrócona (odbita) - liczba binarna, której wszystkie bity zostały zamienione miejscami dookoła punku centralnego, np. liczba 1101 jest odbiciem liczby 1011.

ROCKSOFT™ MODEL CRC ALGORITHM - sparametryzowany algorytm obliczania CRC. Poprzez podanie odpowiednich parametrów wejściowych pozwala prowadzić obliczenia różnymi metodami.

Szerokość algorytmu (*width*) - szerokość algorytmu odpowiada szerokości wielomianu generującego minus jeden (czyli jego stopniowi). Np. jeśli generatorem jest 11010, szerokość będzie równa 4. Szerokość algorytmu jest najczęściej wielokrotnością liczby 8.

List. 5. Program generowania 16- lub 32-bitowej tablicy CRC

```
/**************************************************************************/
/*                            Program crctable.c                          */
/**************************************************************************/
/*                                                                        */
/* Autor: Ross Williams (ross@guest.adelaide.edu.au.).                    */
/* Data:  3 czerwca 1993.                                                 */
/* Wersja: 1.0.                                                           */
/* Status : Public domain.                                               */
/*                                                                        */
/* Opis: Program generuje tablicę CRC (lookup table), która może być dołączana */
/*       do programów w języku C, obliczających CRC metodą tablicową.     */
/*       Wygenerowana tablica jest zapisywana w pliku wyjściowym.         */
/*       Tablice mogą być wykorzystywane w obliczeniach wykorzystujących  */
/*       "Rocksoft^tm Model CRC Algorithm" opisany w poprzednich           */
/*       odcinkach kursu. Materiał źródłowy nosi tytuł "A Painless Guide to CRC*/
/*       Error Detection Algorithms"                                      */
/*       ross Williams (ross@guest.adelaide.edu.au.)                      */
/*   Dokument ten jest do pobrania z adresu:                          */
/*       Błąd! Nie określono zakładki.                                    */
/*                                                                        */
/*Uwaga: Rocksoft jest znakiem handlowym Rocksoft Pty Ltd, Adelaide, Australia */
/**************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "crcmodel.h"
/**************************************************************************/
/* PARAMETRY TABLICY                                                      */
/* =================                                                      */
/* Poniższe parametry określają sposób generowania tablicy. W zależności od    */
/* potrzeb należy je zmienić.                                             */
/*                                                                        */
/*    TB_FILE  - nazwa pliku wyjściowego                                  */
/*    TB_WIDTH - szerokość tablic w bajtach (2 or 4)                      */
/*    TB_POLY  - wielomian generujący, musi mieć szerokość taką jak TB_WIDTH   */
/*    TB_REVER - zmienna określająca, czy ma być użyty algorytm prosty, czy    */
/*               odwrócony                                                */
/*                                                                        */
/* Przykład:                                                              */
/*    #define TB_FILE   "crctable.out"                                    */
/*    #define TB_WIDTH  2                                                 */
/*    #define TB_POLY   0x8005L                                           */
/*    #define TB_REVER  TRUE                                              */

#define TB_FILE   "crctable.out"
#define TB_WIDTH  4
#define TB_POLY   0x04C11DB7L
#define TB_REVER  TRUE

/**************************************************************************/
/* Definicje                                                              */
#define LOCAL static
FILE *outfile;
#define WR(X) fprintf(outfile,(X))
#define WP(X,Y) fprintf(outfile,(X),(Y))
/**************************************************************************/
LOCAL void chk_err P_((char *));
LOCAL void chk_err (mess)
char *mess;
{
 if (mess[0] != 0   ) {printf("%s\n",mess); exit(EXIT_FAILURE);}
 if (ferror(outfile)) {perror("chk_err");   exit(EXIT_FAILURE);}
}
/**************************************************************************/
LOCAL void chkparam P_((void));
LOCAL void chkparam ()
{
 if ((TB_WIDTH != 2) && (TB_WIDTH != 4))
    chk_err("chkparam: Width parameter is illegal.");
 if ((TB_WIDTH == 2) && (TB_POLY & 0xFFFF0000L))
    chk_err("chkparam: Poly parameter is too wide.");
 if ((TB_REVER != FALSE) && (TB_REVER != TRUE))
    chk_err("chkparam: Reverse parameter is not boolean.");
}
/**************************************************************************/
LOCAL void gentable P_((void));
LOCAL void gentable ()
{
 WR("/**************************************************************/\n");
 WR("/*                                                            */\n");
 WR("/* CRC LOOKUP TABLE                                           */\n");
 WR("/* ================                                           */\n");
 WR("/* The following CRC lookup table was generated automagically */\n");
 WR("/* by the Rocksoft^tm Model CRC Algorithm Table Generation    */\n");
 WR("/* Program V1.0 using the following model parameters:         */\n");
 WR("/*                                                            */\n");
 WP("/*    Width  : %11u bytes.                                      */\n",
    (ulong) TB_WIDTH);
 if (TB_WIDTH == 2)
 WP("/*    Poly   : 0x%04lX                                        */\n",
    (ulong) TB_POLY);
 else
```

List. 5. cd.

```
 WP("/*    Poly   : 0x%08lXL                                          */\n",
    (ulong) TB_POLY);
 if (TB_REVER)
 WR("/*    Reverse: TRUE.                                             */\n");
 else
 WR("/*    Reverse: FALSE.                                            */\n");
 WR("/*                                                               */\n");
 WR("/* For more information on the Rocksoft^tm Model CRC Algorithm,  */\n");
 WR("/* see the document titled \"A Painless Guide to CRC Error       */\n");
 WR("/* Detection Algorithms\" by Ross Williams                       */\n");
 WR("/* (ross@guest.adelaide.edu.au.). This document is likely to be  */\n");
 WR("/* in the FTP archive \"ftp.adelaide.edu.au/pub/rocksoft\".       */\n");
 WR("/*                                                               */\n");
 WR("/*********************************************************************/\n");
 WR("\n");
 switch (TB_WIDTH)
   {
    case 2: WR("unsigned short crctable[256] =\n{\n"); break;
    case 4: WR("unsigned long  crctable[256] =\n{\n"); break;
    default: chk_err("gentable: TB_WIDTH is invalid.");
   }
 chk_err("");
 {
  int i;
  cm_t cm;
  char *form    = (TB_WIDTH==2) ? "0x%04lX": "0x%08lXL";
  int   perline = (TB_WIDTH==2) ? 8: 4;
  cm.cm_width = TB_WIDTH*8;
  cm.cm_poly  = TB_POLY;
  cm.cm_refin = TB_REVER;
  for (i=0; i<256; i++)
    {
     WR(" ");
     WP(form,(ulong) cm_tab(&cm,i));
     if (i != 255) WR(",");
     if (((i+1) % perline) == 0) WR("\n");
     chk_err("");
    }
 WR("};\n");
 WR("\n");
 WR("/*********************************************************************/\n");
 WR("/*                    End of CRC Lookup Table                        */\n");
 WR("/*********************************************************************/\n");
 WR("");
 chk_err("");
}
}
/*****************************************************************************/
main ()
{
 printf("\n");
 printf("Rocksoft^tm Model CRC Algorithm Table Generation Program V1.0\n");
 printf("------------------------------------------------------------\n");
 printf("Output file is \"%s\".\n",TB_FILE);
 chkparam();
 outfile = fopen(TB_FILE,"w"); chk_err("");
 gentable();
 if (fclose(outfile) != 0)
    chk_err("main: Couldn't close output file.");
 printf("\nSUCCESS: The table has been successfully written.\n");
}
/*****************************************************************************/
/*                            Koniec crctable.c                              */
/*****************************************************************************/
```

List. 6. Zestaw procedur do obliczania CRC

```
/**********************************
crc.c - straightforward 16 bit CRC
by Alberto Ricci Bitti
released to public domain
compatibility notes:
works on little-endian machines only,
assumes 32 bit long integers,
16 bit integers and 8 byte characters
**********************************/
/*crc-16 standard root*/
#define POLYNOMIAL 0x8005

/*place your own here*/
#define INITIAL_VALUE 0x0000

union
   {
    unsigned long Whole;
    struct
          {
           unsigned char Data;
           unsigned int Remainder;
           unsigned char Head;
          } Part;
   } CRC_buffer;

/*internal use only - puts a byte*/
static void PutCRC(unsigned char b)
{
 unsigned char i;
 CRC_buffer.Part.Data = b;
 for (i=0; i<8; i++)
 {
CRC_buffer.Whole = CRC_buffer.Whole << 1;
if (CRC_buffer.Part.Head & 0x01)
CRC_buffer.Part.Remainder ^= POLYNOMIAL;
 };
}

/*call this routine with your
  own data buffer
  yes! it's really that simple!*/

unsigned int CRC (unsigned char *
Data, unsigned int Length)
{
 CRC.Part.Remainder = INITIAL_VALUE;
 while (Length-- > 0)
 PutCRC(*Data++);
 PutCRC(0);
 PutCRC(0);
 return CRC_buffer.Part.Remainder;
}
```

z dobrym, niezaszumionym i niezakłóconym kanałem transmisyjnym. Przykładem może być choćby telemetryczna akwizycja danych o temperaturze z czujnika oddalonego od stacji zbierającej dane z wielu punktów. Utrata jednego pomiaru nie będzie wielką stratą, gdyż już w chwilę później nadejdzie kolejna dana. Dla bezpieczeństwa, zobrazowanie (lub dalsza obróbka) wyników może być poprzedzone uśrednieniem danych z kilku transmisji. W ostateczności można zastosować dużo prostszą metodę, do tego bardzo łatwą w praktycznej realizacji, jaką jest kontrola parzystości na poziomie transmitowanych bajtów. Jeśli jednak staniemy przed problemem np. przesłania dużej liczby danych z jednego urządzenia do drugiego lub zależy nam na jak najmniejszym prawdopodobieństwie błędu choćby ze względu na znaczenie danych, metody CRC mogą być nieodzowne. Czynnikiem wpływającym na podjęcie decyzji o stosowaniu kontroli transmisji może być również prowadzenie transmisji danych z urządzeń w czasie normalnego ich funkcjonowania (on-line). W krytycznej sytuacji może się nawet okazać, że zabraknie nam mocy obliczeniowej mikrokontrolera. Trzeba wówczas sięgać po metody sprzętowe. Przykładem takiego rozwiązania może być układ 74F401 (*Fairchild*), będący generatorem/kontrolerem CRC. W zależności od ustawienia bitów sterujących, umożliwia on pracę w trybach CRC-16 ($X^{16}+X^{15}+X^2+1$), CRC-16 REVERSE ($X^{16}+X^{14}+X+1$), $X^{16}+X^{15}+X^{13}+X^7+X^4+X^2+X^1+1$, CRC-12 ($X^{12}+X^{11}+X^3+X^2+X+1$), $X^8+X^7+X^5+X^4+X+1$, LRC-8 (X^8+1), CRC-CCITT ($X^{16}+X^{12}+X^5+1$) oraz CRC-CCITT REVERSE ($X^{16}+X^{11}+X^4+1$). Pełna nota katalogowa układu jest zamieszczona na płytce CD-ROM dołączonej do tego numeru EPooL.

Literatura:

1. Boudreau, Steen, „Cyclic Redundancy Checking by Program“ AFIPS Proceedings, Vol. 39, 1971.
2. Davies, Barber, „Computer Networks and Their Protocols“ J. Wiley &Sons, 1979.

List. 7. Zestaw procedur do obliczania CRC stosowanych w protokole Modbus

```
/*>>>>>>>>>>>>>>>>>>>>>>>>>> FUNCTION: crc_calc() <<<<<<<<<<<<<<<<<<<<<<<<<< */
/*                                                                           */
/* Purpose: Generate the CRC checksum's used by the Modbus Protocol.        */
/*                                                                           */
/* Notes: This routine will simulate a "reverse" CRC Hardware circuit.      */
/* (Used in the Modbus Protocol).                                            */
/*                                                                           */
/* This function uses the CRC-16 16 bit CCITT polynomial.                    */
/* CRC Polynomial Used: x16+x15+x2+x1                                        */
/*                                                                           */
/* Entry: The calling routine must pass an unsigned char (byte) value        */
/* to perform the CRC calculations on, and a pointer to an                   */
/* unsigned integer location for the storage of the generated                */
/* CRC word (2 bytes). It is the calling functions                           */
/* responsibility to initialize the "crc_accum" location to 0xffff           */
/* prior to calling this routine for the first time. Only init               */
/* the location after the sequence of characters used in the                 */
/* CRC generation are complete.                                              */
/*                                                                           */
/* Example: If the character string is: 06 03 0b b9 00 01                   */
/* The calculated CRC will be: 7c 56                                         */
/* (Note: Swap bytes for serial transmission - the                           */
/* complete transmission sequence will be:                                   */
/*                                                                           */
/* 06 03 0b b9 00 01 56 7c                                                   */
/* <-------- +---+                                                           */
/* Data Flow CRC                                                             */
/*                                                                           */
/* Exit: Location crc_accum will have a 16 bit (two bytes) value of          */
/* the new calculated CRC.                                                   */
/*                                                                           */
/* Programmer: Rick Vaughn Eagle Systems, Santa Fe, Texas                    */
/*                                                                           */
/* Revisions: 8/26/94 (Orig) ---Rick Vaughn---                               */
/*                                                                           */
/*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< */
void crc_calc(uchar work_data)
{
 code uint genpoly = 0xa001; /* Reversed polynomial */
 uchar i;
/* Convert the received byte to an integer */
 crc_accum = crc_accum ^ (uint)work_data;
 for(i=8; i>0; i--)
 {
 /* Loop 8 times to test each bit of the new character */
  if ((crc_accum) & 0x0001)
    crc_accum = ((crc_accum) >> 1) ^ genpoly;
   else
   (crc_accum) >>= 1;
 }
}
```


3. Higginson, Kirstein, „On the Computation of Cyclic Redundancy Checks by Program“, The Computer Journal (British), Vol. 16, No. 1, Feb 1973.
4. McNamara, J. E., „Technical Aspects of Data Communication“ 2nd Edition, Digital Press, Bedford, Massachusetts, 1982.
5. Marton and Frambs, „A Cyclic Redundancy Checking (CRC) Algorithm“, Honeywell Computer Journal, Vol. 5, No. 3, 1971.
6. Nelson M., „File verification using CRC“, Dr Dobbs Journal, May 1992, str. 64-67.
7. Ramabadran T.V., Gaitonde S.S., „A tutorial on CRC computations“, IEEE Micro, Aug 1988.
8. Schwaderer W.D., „CRC Calculation“, April 85 PC Tech Journal, str. 118-133.
9. Ward R.K, Tabandeh M., „Error Correction and Detection, A Geometric Approach“, The Computer Journal, Vol. 27, No. 3, 1984, str. 246-253.
10. Wecker, S., „A Table-Lookup Algorithm for Software Computation of Cyclic Redundancy Check (CRC)“, Digital Equipment Corporation memorandum, 1974.

Jarosław Doliński, AVT
jaroslaw.dolinski@ep.com.pl

Artykuł powstał na podstawie publikacji „A painless guide to CRC error detection algorithms“. Autor Ross N. Williams. Można go znaleźć pod adresem http://www.riccibitti.com/crcguide.htm.