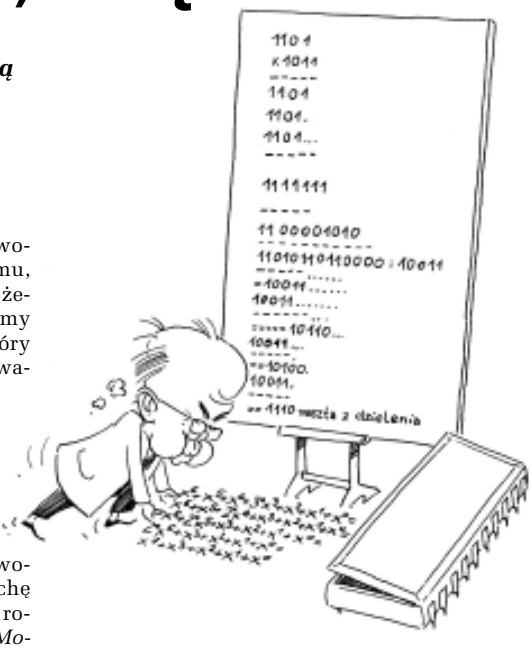
CRC doda Ci pewności, część 4

Ufff... Zbliżamy się do końca. W zasadzie całą teorię dotyczącą algorytmów generowania CRC mamy już za sobą. Czas na ćwiczenia praktyczne.



Wartości początkowe i końcowe

Poszczególne algorytmy generowania CRC różnią się między sobą nie tylko zastosowanymi „chwytami" programowymi, ale również przyjmowanymi wartościami początkowymi rejestru oraz wartością, która będzie XOR-owana z końcową jego zawartością. Przykładowo: w kodzie CRC32 rejestr jest inicjowany wartością FFFFFFFFh, końcowa zawartość rejestru jest XOR-owana wartością FFFFFFFFh. Wiele algorytmów CRC inicjuje rejestr wartością zero, ale nie musi to być regułą. Niektóre, jak widać choćby z powyższego przykładu, wpisują do rejestru niezerową wartość początkową. Teoretycznie nie ma ona wpływu na wydajność ani efekt pracy algorytmu. W praktyce niektóre wiadomości (ciągi danych, z których będzie wyliczane CRC) są bardziej prawdopodobne od innych. Rozsądne wydaje się inicjowanie rejestru taką wartością, która nie będzie zawierała „martwych punktów" mogących wystąpić w rzeczywistości. Określenie „martwy punkt" oznacza w tym przypadku taką sekwencję danych (odbieranych bajtów), która wzięta do obliczeń nie spowoduje zmiany stanu rejestru. Algorytmy, które inicjują rejestr wartością zerową, mogą zawierać „martwy punkt". Zdarza się bowiem często, że sekwencja „rozbiegowa" transmisji zawiera taki właśnie ciąg. Z tego powodu bezpieczniej jest przyjmować niezerową wartość początkową rejestru.

Algorytm absolutny

Jak mogliśmy się przekonać z wcześniejszych rozważań, w teorii tablicowych algorytmów obliczania CRC pojawiło się kilka ważnych aspektów. W ich wyniku powstało wiele metod obliczania CRC, a ogarnięcie całości zagadnienia stało się dość trudne. Spróbujmy teraz zebrać nabytą wiedzę. Widzimy, że poszczególne algorytmy zależą od:

- długości wielomianu generującego (generatora),
- wartości generatora,
- początkowego stanu rejestru,
- tego, czy bity w każdym odebranym bajcie są odwrócone przed wykonaniem obliczeń,
- tego, czy algorytm pobiera bajty wejściowe poprzez rejestr, czy XOR-uje je z bajtem tablicy,
- tego, czy końcowa wartość rejestru powinna być odwrócona,
- tego, czy XOR-ować daną z końcową wartością rejestru.

Gdybyśmy zdecydowali się na stworzenie jednego, uniwersalnego algorytmu, należałoby precyzyjniej określić założenia. Spróbujemy to zrobić. Otrzymamy pewien sparametryzowany model, który później będzie mógł być wykorzystywany w sposób uniwersalny.

Model parametryczny dla algorytmów obliczania CRC

Dojrzeliśmy już do tego, żeby stworzyć praktyczny model obliczania CRC. No dobrze, powiedzmy szczerze, to nie my będziemy go tworzyć, gdyż zrobiono to już za nas trochę wcześniej. My prześledzimy tylko tok rozumowania. Będzie to tzw. *Rocksoft Model CRC Algorithm*. W modelu tym, skupimy się wyłącznie na funkcjonalności algorytmu, nie zważając na detale implementacyjne. Nie będziemy więc, przynajmniej na razie, przejmować się ewentualnymi problemami, jakie mogą wystąpić na etapie kodowania algorytmu w konkretnym języku programowania, chociaż finalnym produktem rozważań będzie program w języku C. Tworzony sparametryzowany model musi być maksymalnie prosty i precyzyjny, a co za tym idzie uporządkowany. Będzie on bazował zasadniczo na bezpośrednim algorytmie tablicowym (patrz część 3). Pamiętając jednak, że powinien spełniać wszystkie powyższe kryteria, dołożymy do niego możliwość ustalania, czy ma (tak jak w algorytmie odwróconym) dokonywać odwracania bajtu wejściowego oraz czy odwrócona ma być również końcowa wartość wyliczonej sumy kontrolnej. Jeden z parametrów pozwoli inicjować rejestr obliczeniowy algorytmu odpowiednią wartością, inny zaś będzie argumentem operacji XOR na końcowej wartości sumy kontrolnej, zanim zostanie zwrócona jako ostateczny wynik obliczeń do procedury nadrzędnej. Mając powyższe założenia, spróbujmy sporządzić konkretną listę parametrów (przyjmiemy oryginalne oznaczenia):

NAME - jest to nazwa algorytmu - zmienna łańcuchowa;

WIDTH - jest to szerokość słowa obliczeniowego algorytmu (rejestru) - liczba bitów. Parametr „SZEROKOŚĆ" jest liczbą o jeden mniejszą niż szerokość generatora.

POLY - ten parametr to po prostu nasz wielomian generujący (generator, *poly*). Jest to liczba binarna, którą dla wygody będziemy zapisywać w postaci szesnastkowej, ale uwaga! W zapisie będziemy omijać najstarszy bit generatora, pamiętając, że zawsze jest on równy „1". Jeśli więc zastosujemy generator np. 10110, to jako parametr będziemy podawać go w postaci 0x06. Ważne jest, że parametr ten przedstawia generator w postaci nieodwróconej. Dolny bit tego parametru będzie zawsze najmniej znaczącym bitem dzielnika, niezależnie od tego, czy algorytm będzie prosty, czy odwrócony.

INIT - parametr, określający stan początkowy rejestru. Jest to wartość wpisywana do rejestru w bezpośredniej metodzie tablicowej. W algorytmie tablicowym rejestr jest zawsze zerowany na początku, a INIT będzie wartością, z którą zostanie XOR-owany rejestr po N-tej iteracji. Parametr ten powinien być podawany w postaci liczby szesnastkowej.

REFIN - jest to parametr logiczny. Jeśli będzie on miał wartość FALSE, to bit 7 bajtów wejściowych będzie traktowany jako najbardziej znaczący (MSB), a bit 0 jako najmniej znaczący (LSB). W tym przypadku każdy bajt powinien być odwrócony przed wykonaniem obliczeń.

REFOUT - jest też parametrem logicznym. Jeśli będzie miał wartość FALSE, to końcowa wartość rejestru będzie bezpośrednio przenoszona do pola XOROUT, w przeciwnym przypadku przed przeniesieniem zawartości rejestru do pola XOROUT rejestr musi być najpierw odwrócony.

XOROUT - jest to W-bitowa liczba, którą będziemy podawać w postaci szesnastkowej. Będzie ona XOR-owana z końcową zawartością rejestru (po REFOUT), przed umieszczeniem wartości zwracanej przez procedurę jako końcowa wartość wyliczonej sumy kontrolnej.

Bezpieczna wymiana danych w systemach mikroprocesorowych

List. 1. Plik nagłówkowy crcmodel.h

```
/*****************************************************************************/
/*                              Początek pliku crcmodel.h                    */
/*****************************************************************************/
/*                                                                           */
/* Autor: Ross Williams (ross@guest.adelaide.edu.au.).                       */
/* Data:  3 czerwca 1993.                                                    */
/* Status: Public domain.                                                    */
/*                                                                           */
/* Opis: To jest plik nagłówkowy (.h), dla programu obliczjącego CRC zgodnie */
/* z Rocksoft^tm Model CRC Algorithm.                                        */
/*Uwaga: Rocksoft jest znakiem handlowym Rocksoft Pty Ltd, Adelaide, Australia*/
/*                                                                           */
/*****************************************************************************/

#ifndef CM_DONE
#define CM_DONE
#ifndef DONE_STYLE

typedef unsigned long   ulong;
typedef unsigned        bool;
typedef unsigned char * p_ubyte_;

#ifndef TRUE
#define FALSE 0
#define TRUE  1
#endif

/* Zamienić na drugą definicję, jeśli nie ma prototypów */
#define P_(A) A
/* #define P_(A) () */

/* Zdjąć symbol komentarza w poniższej definicji, jeśli nie ma void. */
/* typedef int void; */
#endif

/*****************************************************************************/
/* Krótki opis typów w Modelu CRC */
/* Poniższe typy związane są z wykorzystywanym modelem omawianym w artykule   */
/* Większość z nich powinna być ustawiona przed pierwszym wywołaniem cm_ini   */
typedef struct
 {
  int   cm_width;   /* Parametr: Width w bitach [8,32].      */
  ulong cm_poly;    /* Parametr: Generator (poly) algorytmu */
  ulong cm_init;    /* Parametr: Wartość początkowa rejestru */
  bool  cm_refin;   /* Parametr: czy odwracać bajty?         */
  bool  cm_refot;   /* Parametr: czy odwracać wyjściowe CRC?*/
  ulong cm_xorot;   /* Parametr: wartość do XOR-owania wyjściowego CRC. */
  ulong cm_reg;     /* Kontekst: Kontekst podczas obliczeń  */
 } cm_t;
typedef cm_t *p_cm_t;

/*****************************************************************************/
/* Funkcje implementujące Model      */

void cm_ini P_((p_cm_t p_cm));
/* Inicjowanie modelu          */
/* Wszystkie parametry powinny być ustawione przed wywpłaniem */

void cm_nxt P_((p_cm_t p_cm,int ch));
/* Pobranie kolejnego bajtu do obliczeń [0,255] */

void cm_blk P_((p_cm_t p_cm,p_ubyte_ blk_adr,ulong blk_len));
/* Pobranie bloku bajtów komunikatu */

ulong cm_crc P_((p_cm_t p_cm));
/* Zwraca wartość CRC dla wiadomości */

/*****************************************************************************/

/* Funkcje dla obliczeń tablicowych */
/* ---------------- */
/* Poniższe funkcje mogą być wykorzystywane do obliczania tablic (lookup table)*/
/* dla metod tablicowych                                                      */
/* Mogą być również stosowane "w biegu" do tworzenia lub sprawdzania          */
/* tablic statycznych                                                         */

ulong cm_tab P_((p_cm_t p_cm,int index));
/* Zwraca i-ty element tablicy (lookup table) dla określonego algorytmu       */
/* Funkcja sprawdza pola cm_width, cm_poly, cm_refin i argument tablicy       */
/* indeksowany w zakresie [0,255] i zwraca jako wartość funkcji element tablicy*/
/* określony przez cm_width młodszych bajtów                                  */

/*****************************************************************************/

#endif

/*****************************************************************************/
/*                            Koniec pliku crcmodel.h                        */
/*****************************************************************************/
```

CHECK - to pole jest niezwiązane z definicją w dosłownym znaczeniu i w przypadku konfliktu pomiędzy nim a innymi polami ustępuje im pierwszeństwa. Będzie służyło do kontrolowania poprawności implementacji algorytmu. Pole CHECK będzie zawierało sumę kontrolną otrzymaną w wyniku „przepuszczenia" przez algorytm łańcucha znakowego (ASCII) o wartości „123456789" (0x31,0x32,0x33, itd.).

Z tak zdefiniowanymi parametrami model może być wykorzystany do dokładnego opisywania algorytmów. Przykładem niech będzie popularny algorytm CRC-16. Odpowiednie dla niego parametry będą następujące:

Name:	„CRC-16"	
Width:	16	
Poly:	8005	(pamiętamy, że jest to zapis szesnastkowy)
Init:	0000	
RefIn:	True	
RefOut:	True	
XorOut:	0000	
Check:	BB3D	

Przykładowe zestawienie parametrów dla wybranych algorytmów

Poniższa lista zawiera wielomiany generujące dla kilku standardowych algorytmów. Niestety, ze względu na spotykane rozbieżności, określenie kompletu parametrów okazało się dość kłopotliwe.

X25 standardowy:	1021	(CRC-CCITT, ADCCP, SDLC/HDLC)
X25 odwrócony:	0811	
CRC16 standardowy:	8005	
CRC16 odwrócony:	4003	(LHA)

W poniższym zestawie uwzględniono pełniejszą listę parametrów:

Name:	„CRC-16/CITT"
Width:	16
Poly:	1021
Init:	FFFF
RefIn:	False
RefOut:	False
XorOut:	0000
Check:	?

Name:	„XMODEM"
Width:	16
Poly:	8408
Init:	0000
RefIn:	True
RefOut:	True
XorOut:	0000
Check:	?

Name:	„ARC"
Width:	16
Poly:	8005
Init:	0000
RefIn:	True
RefOut:	True
XorOut:	0000
Check:	?

Na zakończenie zestawienie parametrów algorytmu CRC-32 (PKZIP, AUTODIN II, Ethernet, FDDI)

List. 2. Plik implementacyjny crcmodel.c

```
/**********************************************************************************/
/*                              Początek pliku crcmodel.c                         */
/**********************************************************************************/
/*                                                                                */
/* Autor: Ross Williams (ross@guest.adelaide.edu.au.).                            */
/* Data:  3 czerwca 1993.                                                         */
/* Status: Public domain.                                                        */
/* Note: Rocksoft is a trademark of Rocksoft Pty Ltd, Adelaide, Australia.        */
/*                                                                                */
/**********************************************************************************/
/*                                                                                */
#include "crcmodel.h"
/**********************************************************************************/

/* Poniższe definicje zamieszczono w celu zwiększenia czytelności kodu.          */

#define BITMASK(X) (1L << (X))
#define MASK32 0xFFFFFFFFL
#define LOCAL static

/**********************************************************************************/

LOCAL ulong reflect P_((ulong v,int b));
LOCAL ulong reflect (v,b)
/* Zwraca wartość v, w której b młodszych bitów [0,32] jest odwróconych          */
/* Przykład: reflect(0x3e23L,3) == 0x3e26                                         */
ulong v;
int   b;
{
int   i;
ulong t = v;
for (i=0; i<b; i++)
  {
   if (t & 1L)
      v|=  BITMASK((b-1)-i);
   else
      v&= ~BITMASK((b-1)-i);
   t>>=1;
  }
return v;
}

/**********************************************************************************/

LOCAL ulong widmask P_((p_cm_t));
LOCAL ulong widmask (p_cm)
/* Zwraca wartość longword, równą (2^p_cm->cm_width)-1                            */
/* Zastosowano trik pozwalający uniknąć przesunięcia <<32)                        */
p_cm_t p_cm;
{
return (((1L<<(p_cm->cm_width-1))-1L)<<1)|1L;
}

/**********************************************************************************/

void cm_ini (p_cm)
p_cm_t p_cm;
{
p_cm->cm_reg = p_cm->cm_init;
}

/**********************************************************************************/

void cm_nxt (p_cm,ch)
p_cm_t p_cm;
int    ch;
{
int   i;
ulong uch  = (ulong) ch;
ulong topbit = BITMASK(p_cm->cm_width-1);

if (p_cm->cm_refin) uch = reflect(uch,8);
p_cm->cm_reg ^= (uch << (p_cm->cm_width-8));
for (i=0; i<8; i++)
  {
   if (p_cm->cm_reg & topbit)
      p_cm->cm_reg = (p_cm->cm_reg << 1) ^ p_cm->cm_poly;
   else
      p_cm->cm_reg <<= 1;
   p_cm->cm_reg &= widmask(p_cm);
  }
}

/**********************************************************************************/

void cm_blk (p_cm,blk_adr,blk_len)
p_cm_t   p_cm;
p_ubyte_ blk_adr;
ulong    blk_len;
{
while (blk_len-) cm_nxt(p_cm,*blk_adr++);
}
```

Name:	„CRC-32“
Width:	32
Poly:	04C11DB7
Init:	FFFFFFFF
RefIn:	True
RefOut:	True
XorOut:	FFFFFFFF
Check:	CBF43926

Implementacja modelu w języku C

Ci, którym udało się przebrnąć przez dotychczasowe odcinki artykułu, z pewnością czekali na ten właśnie moment. Cała zdobyta wiedza będzie za chwilę wykorzystana do napisania konkretnego programu. Nie zdziwi chyba nikogo, że zostanie do tego celu wykorzystany język C, chyba najbardziej popularny w profesjonalnych zastosowaniach. Nasz program będzie zawierał plik nagłówkowy (.h) i plik implementacyjny (.c). Chociaż suma kontrolna będzie obliczana w sposób jak najbardziej prawidłowy, to program ze względu na jego uniwersalność będzie miał raczej niewielką przydatność praktyczną (jako całość). Jest zaprezentowany ze względu na walory dydaktyczne. Wprawny programista będzie potrafił wyfiltrować ewentualnie tylko te jego fragmenty, które okażą się potrzebne w konkretnych przypadkach. Dla sprawdzenia, czy poniższy kod działa prawidłowo, można go będzie skonfigurować dla algorytmu CRC-16 lub CRC-32, zgodnie z informacjami zamieszczonymi powyżej i sprawdzić wynik na przykładowym, wejściowym łańcuchu tekstowym „123456789“ o znanej sumie kontrolnej. Autorem programu jest Ross Williams. Program został zamieszczony w Internecie ze statusem public domain. Aby zapewnić jak największą niezależność od komputerów, na których może być uruchamiany, zastosowano możliwie proste sposoby kodowania. Na przykład nazwy wszystkich zewnętrznych identyfikatorów ograniczono do 6 znaków, a wewnętrznych do 8. W celu uniknięcia ewentualnych kolizji w nazwach zmiennych zastosowano prefiks cm (od CRC Model). Założona uniwersalność programu niestety niekorzystnie wpływa na jego efektywność. Warto pamiętać o przewadze pod względem szybkości działania metod tablicowych nad metodami algorytmicznymi.

Jak korzystać z programu?

Krok 1: Zadeklarować zmienną typu cm_t. Zadeklarować inną zmienną (p_cm) typu p_cm_t i zainicjować ją jako wskazanie na pierwszą (np. p_cm_t p_cm = &cm_t)

Krok 2: Przypisać wartości poszczególnym polom struktury (patrz uwagi w tekście)

Przykład:

```
p_cm->cm_width = 16;
p_cm->cm_poly = 0x8005L;
p_cm->cm_init = 0L;
p_cm->cm_refin = TRUE;
p_cm->cm_refot = TRUE;
p_cm->cm_xorot = 0L;
```

Uwaga: Wartość Poly jest określana bez najstarszego bitu (18005 to 8005).

List. 2. cd.

```
/***************************************************************************/

ulong cm_crc (p_cm)
p_cm_t p_cm;
{
if (p_cm->cm_refot)
   return p_cm->cm_xorot ^ reflect(p_cm->cm_reg,p_cm->cm_width);
else
   return p_cm->cm_xorot ^ p_cm->cm_reg;
}

/***************************************************************************/

ulong cm_tab (p_cm,index)
p_cm_t p_cm;
int    index;
{
int   i;
ulong r;
ulong topbit = BITMASK(p_cm->cm_width-1);
ulong inbyte = (ulong) index;

if (p_cm->cm_refin) inbyte = reflect(inbyte,8);
r = inbyte << (p_cm->cm_width-8);
for (i=0; i<8; i++)
   if (r & topbit)
      r = (r << 1) ^ p_cm->cm_poly;
   else
      r<<=1;
if (p_cm->cm_refin) r = reflect(r,p_cm->cm_width);
return r & widmask(p_cm);
}

/***************************************************************************/
/*                            Koniec pliku crcmodel.c
*/
/***************************************************************************/
```

Wartość Width jest o jeden mniejsza niż faktyczna szerokość poly.

Krok 3: Zainicjować przykład z wywołaniem cm_ini(p_cm);

Krok 4: Wykonać obliczenia dla zerowej lub niezerowej liczby bajtów komunikatu przez umieszczenie odpowiedniej liczby wywołań cm_nxt. Np.: cm_nxt(p_cm,ch);

Krok 5: Obliczyć CRC, stosując wywołanie crc = cm_crc(p_cm);

Jeśli CRC jest wartością 16-bitową, wynikiem będzie 16 najmłodszych bitów.

Mam nadzieję, że tym razem Czytelnicy zostali usatysfakcjonowani możliwością zetknięcia się z konkretnym programem obliczającym CRC. Właściwie to już wszystko. Czym jest jednak obiad bez deseru? W ostatnim już odcinku (za miesiąc) pokażemy, jak są generowane tablice do metod tablicowych. Pokażemy również kilka innych przykładów oraz wspomnimy coś o metodach sprzętowych obliczania CRC.

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.