VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zabezpečenie dát pomocou 16/32 - bitového kódu CRC

Projekt IMP, 2017

Autor: Kristián Liščinský(xlisci01)

1 Úvod

Hlavnou úlohou projektu bolo v jazyku C zabezpečiť dáta kontrolným cyklickým kódom CRC(Cyclic Redundancy Check). Je to druh kontrolného súčtu používaného na kontrolu správnosti prenášaných údajov v telekomunikačnej technike a počítačových sieťach, ako aj uložených údajov na pamäťových médiách ako je napríklad pevný disk. Projekt bol tvorený v prostredí Kinetis Design Studio 3.0.0 s využitím SDK API pre čip K60DN512M10, ktorý sa nachádza na platforme FITkit3.

2 Špecifikácia zadania

Projekt pozostával z niekoľkých častí:

- Vytvorte vhodne veľký blok dát, které budete ďalej zabezpečovať CRC kódom.
- Postupne tieto data zabezpečte (vytvorte variantu zabezpečenia 16 - bitového CRC a variantu 32 - bitového CRC) troma spôsobmi:
  - Pomocou hardwarového modulu Cyclic Redundancy Check (CRC) z čipu K60.
  - Získáním CRC pomocou polynómu z tabuľky.
  - Výpočtom CRC pomocou polynómu.
- Zaneste do dat niekoľko jedno i viacnásobných chýb a overte / vyhodnoťte schopnosť tieto chyby detekovať pomocí CRC metód, ktoré ste vytvorili.

3 Návrh a implementácia

3.1 32 - bitová varianta

Pre variantu 32b bol zvolený štandard CRC - 32 s generačným polynómom $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$, teda polynómom s hodnotou 0x04C11DB7.

3.2 16 - bitová varianta

Pre variantu 16b bol zvolený štandard CRC - 16 - CCITT s generačným polynómom $x^{16} + x^{12} + x^5 + 1$, teda polynóm s hodnotou 0x1021.

3.3 Výpočet pomocou hardwarového modulu Cyclic Redudancy Check z čipu K60

Modul CRC sme museli nakonfigurovať podľa typu zabezpečenia. Jednalo sa o konfiguráciu počiatočnej hodnoty, jednotlivých flagov a potrebného polynómu [1]. Následne bol modul inicializovaný pomocou funkcie CRC_Init().

[1] http://bit.ly/2CyFMso

```
void InitCrc16_CCIT(CRC_Type *base, uint32_t seed, bool isLast)
{
    crc_config_t config;

    config.polynomial = 0x1021;
    config.seed = seed;
    config.reflectIn = false;
    config.reflectOut = false;
    config.complementChecksum = false;
    config.crcBits = kCrcBits16;
    config.crcResult = isLast?kCrcFinalChecksum:kCrcIntermediateChecksum

    CRC_Init(base, &config);
}
```

Obrázek 1: Konfigurácia modulu CRC16

```
void InitCrc32(CRC_Type *base, uint32_t seed, bool isLast)
{
    crc_config_t config;

    config.polynomial = 0x04C11DB7U;
    config.seed = seed;
    config.reflectIn = true;
    config.reflectOut = true;
    config.complementChecksum = true;
    config.crcBits = kCrcBits32;
    config.crcResult = isLast?kCrcFinalChecksum:kCrcIntermediateChecksum;

    CRC_Init(base, &config);
}
```

Obrázek 2: Konfigurácia modulu CRC32

3.4 Výpočet pomocou polynómu z tabuľky

U CRC - 16 - CCITT ide o cyklus prechadzajúci všetkými bytami. Vypočítame index tabuľky a následne vykonáme operáciu XOR nad počiatočnou hodnotou posunutou o 8 bitov doľava.

U CRC 32 vypočítame taktiež index tabuľky a hodnotu v tabuľke XORujeme s počiatočnou hodnotou posunutou doprava o 8 bitov. Nakoniec potrebujeme ešte invertovať bity. To zabezpečíme pomocou operácie XOR s hodnotou 0xFFFFFFFF[2].

[2]http://bit.ly/2eyWJYf

```
Initialize crc-32 to starting value
crc32 ← 0xffffffff

for each byte in data do
   nLookupIndex ← (crc32 xor byte) and 0xFF;
   crc32 ← (crc32 shr 8) xor CRCTable[nLookupIndex] //CRCTable is an array of 256 32-bit constants

Finalize the CRC-32 value by inverting all the bits
crc32 ← crc32 xor 0xFFFFFFFF

return crc32
```

Obrázek 3: Varianta algoritmus CRC - 32 s použitím tabuľky

3.5 Výpočet pomocou polynómu

Jedná sa o najpomalšiu metódu zabezpečenia dát kódom CRC. Každý znak musíme prechádzať každým bitom a vždy potrebujeme počítať jeho novú hodnotu. 32 - bitovú variantu počítame pomocou maskovania inverzného polynómu. Nakoniec invertujeme bity a dostávame hodnotu CRC.

4 Vyhodnotenie výsledkov

Na výpis som použil komunikáciu cez UART, čo mi umožnilo vypisovať jednotlivé hodnoty na terminál za pomoci SDK API. Ako blok dát na zabezpečenie CRC kódom som použil reťazec "Glory glory Manchester United". Pomocou webovej aplikácie zo stránky `https://www.lammertbies.nl/comm/info/crc-calculation.` som zistil aké výsledky mám od môjho programu očakávať a následný výstup z môjho programu som porovnal z výsledkami z webovej stránky. Potom som do dát zaniesol jednu chybu a vznikli mi dáta "Hlory glory Manchester United". Podobný proces overovania výsledku som spravil aj tentokrát a aj ďalší, kedy som do pôvodných dát zaniesol viacero chýb. Program na zanesené chyby správne reagoval, čo sa prejavilo zmenou hodnoty CRC - kódu.

5 Záver

Program bol riadne otestovaný, behom tohto testovania nedošlo k žiadným chybám. Program bol tvorený v prostredí Kinetis Design Studio 3.0.0 s využitím SDK API pre čip K60DN512M10, ktorý sa nachádza na platforme FITkite3.