VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Mikroprocesorové a vestavěné systémy
2018/2019

Technická správa k projektu

ARM/FITkit3: Zabezpečení dat pomocí 16/32-bit. kódu CRC

Vladimír Marcin, xmarci10@stud.fit.vutbr.cz 26. decembra 2018

Obsah

1 Úvod

Práca slúži na demonštráciu zabezpečenia dát pomocou cyklického kódu CRC (Cyclic Redundancy Check). V projekte bola použitá 16 a 32 bitová varianta CRC kódu.

Projekt bol vytvorený vo vývojovom prostredí MCUXpresso IDE v10.3.0 s využitím SDK 2.0 API pre čip K60DN512M10, ktorý sa nachádza na platforme ARM-FITkit3.

1.1 Zadanie projektu

V jazyku C vytvorte projekt demonštrujúci možnosti zabezpečenia dát 16/32 bitovým kódom CRC na čipe Kinetis K60 z dosky platformy FITkit3.

UPRESNENIE:

1. Vytvorte vhodne veľký blok dát, ktoré budete ďalej zabezpečovať CRC kódom.
2. Postupne tieto dáta zabezpečte (vytvorte variantu zabezpečenia pomocou CRC16 a CRC32) tromi spôsobmi:
   (a) pomocou hardwarového modulu Cyclic Redundancy Check (CRC) z čipu K60
   (b) získaním CRC pomocou polynómu z **2(a)** z tabuľky
   (c) výpočtom CRC pomocou polynomu z **2(a)**
3. Zaneste do dát niekoľko jedno aj viacnásobných chýb a overte/vyhodnoťte schopnosť tieto chyby detekovať pomocou CRC z **2(a)** až **2(c)**
4. Diskutujte realizačnú a výpočetnú réžiu a ne/výhody spojené so zabezpečením dát jednotlivými spôsobmi

2 Cyclic Redundancy Check

Nasledujúca kapitola stučne popisuje princípy cyklického kódu CRC.

2.1 Princíp zabezpečenia dát

CRC je cyklický kód slúžiaci na zabezpečenie dát proti ich potenciálnemu poškodeniu. Hlavným účelom tohto kódu je detekcia chýb, avšak existuje mnoho upravených verzií, ktoré za určitých podmienok dané chyby dokážu aj opraviť.

Vypočítaná CRC hodnota je na strane odosielateľa pridaná na koniec datového reťazca, ktorý je následne odoslaný príjemcovi. Ten po prijatí dát vypočíta vlastnú CRC hodnotu pomocou rovnakého polynómu, ktorým boli dáta zakódované na strane odosielateĺa. Hodnota výsledného CRC by sa mala rovnať nule v prípade, že nedošlo pri prenose k poškodeniu dát.

Na zakódovanie dát sa používa vopred špecifikovaný polynóm (odosielateľ a príjemca sa musia vopred dohodnúť). Najbežnejšie používané sú polynomy dĺžky 16 a 32 bitov [1].

3 Návrh a implementácia

Nasledujúca kapitola popisuje implementáciu jednotlivých spôsobov zabezpečovania dát stanovených zadaním. Taktiež si v tejto kapitole stručne popíšeme zvolené varianty CRC kódov použitých v projekte.

3.1 CRC kódy použité v projekte

3.1.1 CRC16 varianta

Pre 16 bitovú variantu bol zvolený kód CRC-16-CCIT-ZERO. V prípade záujmu môže užívateľ zmeniť hodnotu polynómu. V takom prípade sa už samozrejme nejedná o kód CRC-16-CCIT-ZERO.

Vlastnosti:

- polynóm: 0x1021
- seed (počiatčná hodnota): 0x0000
- reflect IN: false
- reflect OUT: false
- final XOR: 0x0000

3.1.2 CRC32 varianta

Pre 32 bitovú variantu bol zvolený kód CRC-32. Ako v predchádzajúcej variante aj tu sme umožnili užívateľovi zadať svoj vlastný polynóm.

Vlastnosti:

- polynóm: 0x04C11DB7
- seed (počiatčná hodnota): 0xFFFFFFFF
- reflect IN: true
- reflect OUT: true
- final XOR: 0xFFFFFFFF

3.2 HW CRC modul

Modul CRC je pred každým výpočtom inicializovaný pre výpočet CRC16 alebo CRC32. Inicializácia zahŕňa nastavenie polynómu, počiatočnej hodnoty v dátovom registri, typu reflektovania a ďalších možností. Inicializáciu má na starosti funkcia `CRC_Init(...)`.

Po inicializácii sa volaním funkcie `CRC_WriteData(...)` do dátového registra zapíšu dáta, ktoré chceme zabezpečiť CRC kódom.

Na záver sa podľa príslušnej varianty volá funkcia `CRC_GetXXbitResult(...)`, ktorá vráti spočítané CRC pre zadané dáta. XX je nahradené 16 alebo 32 podľa použitej varianty.

```
config.reflectIn = false;
config.reflectOut = false;
config.polynomial = polynomial; //default 0x1021
config.seed = 0x0U;
config.complementChecksum = false;
config.crcBits = kCrcBits16;
config.crcResult = kCrcFinalChecksum;
```

Ukážka 1: Konfigurácia 16 bitovej varianty

```
config.reflectIn = true;
config.reflectOut = true;
config.polynomial = polynomial; //default 0x4C11DB7
config.seed = 0xFFFFFFFFU;
config.complementChecksum = true;
config.crcBits = kCrcBits32;
config.crcResult = kCrcFinalChecksum;
```

Ukážka 2: Konfigurácia 32 bitovej varianty

Pri konfigurácii HW CRC modulu som vychádzal z príkladov na oficiálnej stránke MCUXpresso NXP [4].

3.3 Výpočet pomocou polynómu

Zabezpečnie dát pomocou polynómu je implementované vo funkciách `computeCRC16Simple` a `computeCRC32Simple`, ktorých návratová hodnota je výsledný CRC kontrolný súčet.

Obe s funkcií spracovávajú vstupný reťazec po bajtoch. Pre každý bajt následne prechádzame jeho bitmi a počítame novú hodnotu kontrolného súčtu.

```
polynomial = poly;                          /* default poly value 0x1021 */
crc = 0;                                    /* init crc value */

for each (byte b in bytes){
    crc ^= (b << 8);                        /* move byte into MSB of 16bit CRC */

    for (int i = 0; i < 8; i++){
        if (crc & 0x8000)                   /* test for MSB = 1 */
            crc = (crc << 1) ^ polynomial;
        else
            crc <<= 1;
    }
}

return crc;
```

Ukážka 3: Algoritmus pre výpočet CRC16 pomocou polynómu [3]

```
polynomial = reverseBits( poly );           /* default poly value 0x4C11DB7 */
crc = 0xFFFFFFFF;                           /* init crc value */

for each (byte b in bytes){
    crc ^= b;

    for (int i = 0; i < 8; i++){
        if (crc & 0x1)
            crc = (crc >> 1) ^ polynomial;
        else
            crc >>= 1;
    }
}

return invert(crc);
```

Ukážka 4: Algoritmus pre výpočet CRC32 pomocou polynómu [3]

3.4 Výpočet pomocou polynómu z tabuľky

V tejto variante zabezpečnia sa používa rovnaký algoritmus ako v predchádzajúcej s tým rozdielom, že priebežné hodnoty CRC sú dopredu predpočítané pre všetky kombinácie bitov v jednom bajte a sú uložené do tabuľky. Následne sa pri výpočte kontrolného súčtu používajú predpočítané hodnoty, čo v konečnom dôsledku urýchľuje výpočet.

Táto varianta je implementovaná vo funkciách `computeCRC16Table` a `computeCRC32Table`. Pred samotným volaním týchto funkcií, ktoré vrátia výsledný kontrolný súčet je potrebné zavolať funkcie `calculateTableCRC16` resp. `calculateTableCRC32` slúžiace na inicializáciu lookup tabuliek.

4 Porovnanie implementácií

Čo sa týka efektivity jednotlivých implementácií, najrýchlejšou variantou je zabezpečenie dát pomocou HW modulu, ktorý je optimalizovaný práve pre túto činnosť.

Druhou najefektívnejšou variantou je varianta, ktorá používa lookup tabuľku. To však nemusí platiť vždy. V prípade, že sú kodóvané dáta dostatočne krátke môže byť réžia na výpočet lookup tabuľky vyššia ako samotný výpočet bez tabuľky. Naopak ak sú kódované dáta dostatočne dlhé je táto varianta vždy efektívnejšia ako varianta bez tabuľky.

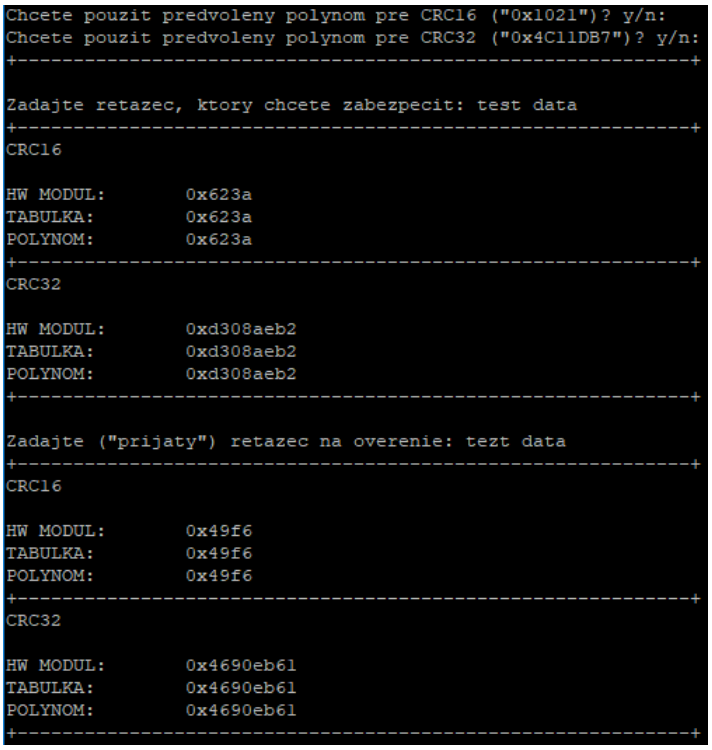
Najpomalšou variantou je počítanie pomocou polynomu bez tabuľky z dôvodu, že vstupné dáta sa spracovávajú bit po bite. Ako sme už hovorili v prípade krátkeho vstupného reťazca môže byť táto varianta lepšia.

5 Demonštrácia funkčnosti programu

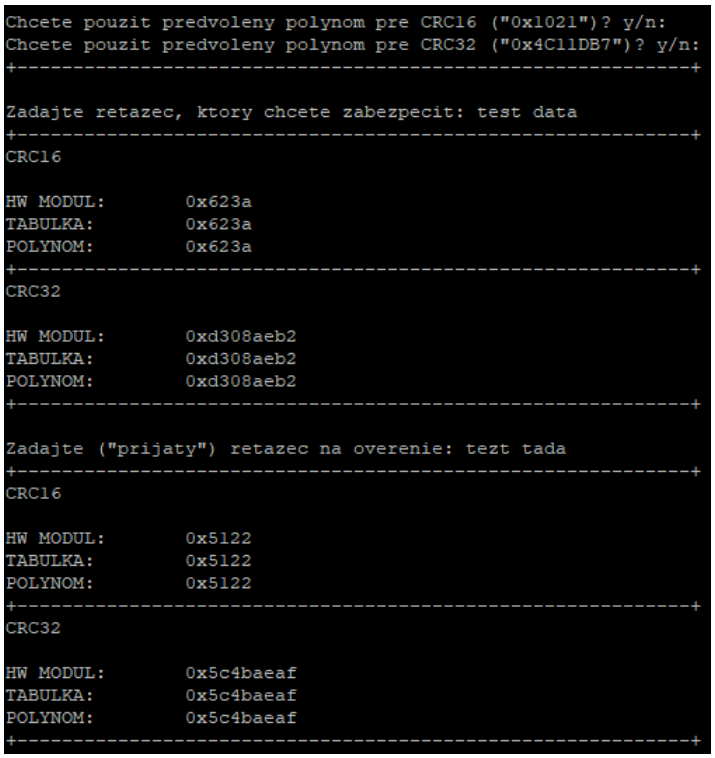
Aplikácia na komunikáciu s užívateľom používa integrovaný modul UART (na prácu s týmto modulom som využil možnosti vývojového prostredia MCUXpresso). Po spustení aplikácia vyzve užívateľa na zadanie polynómu, ktorý chce použiť pre výpočet kontrolného súčtu a následne si vypýta dáta, ktoré majú byť zabezpečené. Následne vypíše výsledné hodnoty CRC16 a CRC32 spočítané pomocou všetkých variánt. Po vypísaní výsledkov je užívateľ vyzvaný pre zadanie reťazca na overenie. Týmto simulujeme použitie CRC zabezpečenia v praxi a to konkrétne overenie prijatého reťazca, ktorý mohol byť napríklad prenášaný po sieti. Konkrétne výsledky sú vidieť na ukážke nižšie.

```
ZABEZPECENIE DAT POMOCOU 16/32-BIT. KODU CRC

Chcete pouzit predvoleny polynom pre CRC16 ("0x1021")? y/n:
Chcete pouzit predvoleny polynom pre CRC32 ("0x4C11DB7")? y/n:
+------------------------------------------------------------+

Zadajte retazec, ktory chcete zabezpecit: test data
+------------------------------------------------------------+
CRC16

HW MODUL:       0x623a
TABULKA:        0x623a
POLYNOM:        0x623a
+------------------------------------------------------------+
CRC32

HW MODUL:       0xd308aeb2
TABULKA:        0xd308aeb2
POLYNOM:        0xd308aeb2
+------------------------------------------------------------+

Zadajte ("prijaty") retazec na overenie: test data
+------------------------------------------------------------+
CRC16

HW MODUL:       0x623a
TABULKA:        0x623a
POLYNOM:        0x623a
+------------------------------------------------------------+
CRC32

HW MODUL:       0xd308aeb2
TABULKA:        0xd308aeb2
POLYNOM:        0xd308aeb2
+------------------------------------------------------------+
```

Obr. 1: Na výstupe je vidieť, že pre výpočet kontolného súčtu boli zvolené predvolené polynomy. Ako vstupné dáta bol zvolený reťazec "test data". Môžme si všimnúť, že overovací reťazec je totožný s pôvodným reťazcom a hodnoty CRC sú podľa očakávania v oboch prípadoch rovnaké.



Obr. 2: V tomto príklade si môžme všimnúť, že do overovacieho reťazca bola zanesená jedna chyba. Aplikácia sa znova správa podľa očakávania a výsledné CRC sa v tomto prípade líšia.



Obr. 3: Posledným testovacím príkladom bolo zanesenie viacerých chýb do overovacieho reťazca a aplikácia znova odhalila, že došlo k chybe.

Správnosť vypočítaných hodnôt CRC kontrolného súčtu bola overená nástrojom [2].

Literatúra

[1] Cyclic redundancy check — Wikipedia, The Free Encyclopedia. online, 2018.
URL https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[2] Molkenthin, B.: CRC Calculator (Javascript). online, 2018.
URL http://www.sunshine2k.de/coding/javascript/crc/crc_js.html

[3] Molkenthin, B.: Understanding and implementing CRC (Cyclic Redundancy Check) calculation. online, 2018.
URL http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html

[4] NXP: CRC Cyclic Redundancy Check Driver. online, 2018.
URL https://mcuxpresso.nxp.com/apidoc/2.0/group__crc__driver.html