Vysoké učení technické v Brně

Fakulta Informačních Technologií

IMP – Mikroprocesorové a vstavané systémy

Zabezpečení dat pomocí 16/32-bit. kódu CRC

Michal Šajdík (xsajdi00) 19.12.2018

Obsah

1. Úvod 3
2. Návod na použitie 3
3. Implementácia 3
   3.1 Implementácia pomocou hardvérového modulu 3
   3.2 Implementácia pomocou polynómu z i) a tabuľky 4
   3.3 Implementácia pomocou polynómu z i) bez tabuľky 4
4. Záver 4
   4.1 Detekcia chýb 4
   4.2 Realizačné a výpočtové réžie 4
5. Zdroje 5

1. Úvod

V rámci nášho projektu bolo našou úlohou demonštrovať možnosti zabezpečenia dát 16bit/32/bit. kódom CRC, pro čip Kinetis K60 z dosky platformy FITkit 3. Projekt bol vypracovaný v **Kinetis Design Studio (KDS)**. Kvôli všemožným spôsobom demonštrácie zabezpečenia dát sme sa rozhodli neimplementovať ani jeden z nich, ale nechať voľný priestor pre záujemcov vo funkcii *„main“*.

2. Návod na použitie

Ako bolo spomenuté vyššie, možnú demonštráciu je možné uskutočniť vo funkcii *„main“*. Pre výpočet CRC kódu v 16 a 32 bitovej podobe, máme implementované funkcie, tromi rozdielnymi metódami: **i)** Pomocou Hardwaru na čipu Kinetis K60, pre prácu s danou metodou slúžia funkcie `CRC32bit(char*, int)` a `CRC16bit(char*, int)` , **ii)** Pomocou polynómu z i) a za použitím tabuľky, pre prácu s danou metódou slúžia funkcie: `iiCRC32bit(char*, int)` , `iiCRC16bit(char*, int)` , `createCRC32bitTable()` a `createCRC16bitTable()` a **iii)** Pomocou polynómu z i) bez použitia tabuľky, pre prácu s danou metódou nám slúžia funkcie: `iiiMsbCRC32bit(char*, int)`, `iiiMsbCRC16bit(char*, int)`, `iiiLsbCRC32bit(char*, int)` a `iiiLsbCRC16bit(char*, int)`. O všetkých vymenovaných funkciách s 2 parametrami môžeme tvrdiť, že prvý argument je pole „charov“ s očakávanou veľkosťou 8 bitov, z toho si môžeme odvodiť, že najmenšia jednotka vkladaných údajov môže byť 1 bajt, druhý parameter je celé číslo reprezentujúce počet bajtov, a všetky tieto funkcie vracajú príslušný CRC kód pre danú bitovú verziu. Dve vyššie uvedené funkcie bez parametrov slúžia na nastavenie tabuliek pre ii). V prípade potreby zmenenia polynómu, alebo „seed“ hodnoty je možné dané hodnoty zmeniť pomocou globálnych premenných `seed16bit`, `seed32bit`, `poly16bit` a `poly32bit`.

3. Implementácia

3.1 Implementácia pomocou hardvérového modulu

Pre začiatok bolo potrebné nastaviť flag TCRC z CRC na „0“ pre 16bit verziu a na „1“ pre 32 bitovú verziu. Ďalej sa nastavil polynóm do `CRC_GPOLY` v prípade 16bit verzie sa nastavil polynóm len do `CRC_GPOLYL`. Nasledovalo nastavenie „seedu“ (počiatočnej hodnoty). V oboch prípadoch sa musel nastaviť flag WAS z CRC na „1“, aby sme mohli zapísať „seed“ do `CRC_CRC` v 32 bitovej verzii a do `CRC_CRCL` v 16 bitovej verzii. Posledná podstatná vec je zaviesť naše dáta do vzniku CRC kódu, ktorý bude výsledkom našej funkcie. Aby sme toho dosiahli, tak musíme nastaviť flag WAS z CRC na „0“, potom môžeme zapisovať 1 bajtové hodnoty do `CRC_CRCLL`. Po dokončení zapísania všetkých našich dát máme v `CRC_CRC` náš CRC kód, ktorý je posunutý ako návratová hodnota použitej funkcie.

3.2 Implementácia pomocou polynómu z i) a tabuľky

Pomocou tabuľky sme si pred počítali hodnoty, ku ktorým sme pristupovali pomocou vždy na novo vypočítaného kľúča z aktuálneho CRC kódu a nasledovného 1 bajtu, ktorý je potrebné zakódovať. Nový CRC kód vznikne použitím XOR operácie na starý CRC kód a na pred počítanú hodnotu z tabuľky, ktorá odpovedá novo vygenerovanému kľúču. Po prejdení všetkých bajtov, ktoré sme chceli zakódovať, nám funkcia vráti CRC kód. Pre vytvorenie tabuľky sme použili podobný princíp „xorovania“ ako v druhom cykle v iii). Každú hodnotu indexu sme posunuli na najvyšší bit a postupne sme ju prešli cyklom, vrchný bajt a celú hodnotu sme „xorovali“ s daným polynómom v prípade že najvyšší bit bol „1“. Po ukončení cyklu pre „xorovanie“ sme výslednú hodnotu zapísali do tabuľky a pokračovali pre ďalší index, dokiaľ sme nenaplnili tabuľku.

3.3 Implementácia pomocou polynómu z i) bez tabuľky

V podstate sa jednalo 2 vnorené cykle, v prvom sme striedali bajty, ktoré sme chceli použiť k vytvoreniu CRC kódu a zároveň „xorovanie“ CRC kódu s novým bajtom. Následne sme sa v druhom cykle posúvali po bitoch a ak bol najvyšší bit „1“ pre variantu s MSB, tak sme náš CRC kód „xorovali“ s nami predurčeným polynómom. Po prejdení všetkých bajtov, ktoré sme chceli zakódovať CRC kódom. Nám naša funkcia vráti CRC kód.

4. Záver

4.1 Detekcia chýb

Detekcia chýb sa u všetkých metód prejavila inou hodnotou CRC kód, v prípade že dáta budú príliš veľké, tak môže prísť ku zdvojenému CRC kódu (dva druhy dát majú rovnaký CRC kód aj napriek tomu že sú rozdielne) v takom prípade by nastalo zlyhanie detekcie chyby v rámci našich dát, ale šanca na daný jav je mimoriadne nízka a pri viac bitovom CRC kóde ešte viac klesá.

4.2 Realizačné a výpočtové réžie

Z hľadiska efektivity je hardvérový modul bezpochyby najvhodnejší, lebo je priamo optimalizovaný pre daný hardware a preto jak realizačná tak i výpočtová réžia sú najnižšie ako len môžu byť. Preto nám zostáva porovnať CRC kód s tabuľkou a CRC kód bez tabuľky. Pre CRC kód s tabuľkou môžeme povedať, že jeho pamäťová náročnosť je väčšia kvôli potrebe si zapamätať danú tabuľku, ale to sa nám vráti pri rýchlosti tvorenia CRC kódu, lebo už máme niektoré hodnoty vypočítané popredu. Z toho môžeme tvrdiť že CRC kód s tabuľkou je výhodnejší pre väčšie množstvo dát, lebo tam viac využijeme už raz vypočítanú hodnotu. Pre malý počet dát je výhodnejší CRC kód bez tabuľky, ak sa posudzujeme podľa rýchlosti.

5. Zdroje

- https://cs.wikipedia.org/wiki/Cyklick%C3%BD_redundantn%C3%AD_sou%C4%8Det
- https://crccalc.com/?fbclid=IwAR2CcePLslrSGwy0kti0MF_yrnc6lWVcZ7FPpLHHgmMUClGhOVo6zF_E3kk
- https://www.nxp.com/docs/en/reference-manual/K60P120M100SF2RM.pdf#d317e3a1310_d1314e16