VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ZABEZPEČENIE DÁT POMOCOU 16/32-BIT. KÓDU CRC

SECURING DATA WITH 16/32-BIT CRC CODE

PROJEKTOVÁ DOKUMENTÁCIA
PROJECT DOCUMENTATION

AUTOR PRÁCE **JOZEF URBANOVSKÝ**
AUTHOR

BRNO 2017

Obsah

Kapitola 1

Úvod

Úlohou tohto projektu bolo napísať program pre výukovú platformu Fitkit3 - Minerva s procesorom rady Kinetis K modelu MK60D10 vo vývojovom prostredí Kinetis Design Studio 3.0. Program zabezpečuje dáta vypočítaním 16 alebo 32 bitového CRC kódu troma rôznymi spôsobami a porovnáva ich rýchlosť a spôsob implementácie na danom prostriedku. Prvý spôsob využíva hardwarový modul Cyclic Redundancy Check (CRC) z čipu K60. Druhý spôsob pracuje s použitím polynómu z predchádzajúceho spôsobu a využitím lookup tabuľky. Posledný spôsob je čisto softwarovo založený a to na použití jednoduchého matematického polynómu pre výpočet CRC kódu.

Aplikácia zabezpečuje blok dát CRC kódom, ktorý by sa pridal do posielaných dát a príjmateľ by si následne rovnakou implementáciou CRC kódu overil, či dáta, ktoré dostal majú správny CRC kód a sú bez chyby.

Kapitola 2

Implementácia

Samotná implementácia projektu je rozdelená do niekoľkých logických častí, z ktorých je každý popísaný detailne nižšie. Súbor s vlastnou implementáciou sa nachádza v zložke `sources`.

2.1 Príprava

Na tento projekt bolo vhodné využiť Kinetis Software Development Kit (KSDK)[5], ktorý obsahoval implementáciu CRC kódu, navrhovanú v zadaní projektu. Inšpiráciou z demo aplikácie z KSDK pre dosku `twrk60d100m` a využitím KSDK bolo možné abstrahovať implementáciu bez nutnosti písania inicializácie samotnej dosky, čítačov a iných registrov potrebných pre výpočet a beh programu. Testovacie dáta majú predvypočítané kontrolné súčty, pre možnosť overenia správnosti počítania naimplementovaných CRC kódov.

2.2 Hardwarový modul

Pre implementáciu 16 bitového cyklického zabezpečovacieho kódu som využil jeden z napoužívanejších CRC kódov s názvom `CRC16 CCITT FALSE`[3], ktorého implementáciu môžeme nájsť aj napríklad u HDLC algoritmu na druhej sieťovej vrstve. Ako základný polynóm v tomto algoritme sa používa kombinácia `0x1021` a incializácia prebieha s polynómom `0xFFFF`. Po inicializácii hodnôt CRC konfigurácie sa do registru zapíšu dáta, nad ktorými sa CRC kód bude počítať. Následne sa vypočíta CRC použitím implementácie z hardwarového modulu K60 a výsledok sa uloží do premennej `checksum16`.

Výsledok tejto premennej sa porovná s obsahom `correct16`, kde je uložená predpočítaná správna hodnota zhodného CRC kódu. Pokiaľ sa zhodujú, vieme určiť, že nami vygenerovaný CRC kód je správny.

Následne ako experiment príjmateľa vyskúšame previesť CRC výpočet na daný kód pre overenie správnosti teoreticky prenesených dát. Pokiaľ je výsledok nulový, vieme určite, že dáta prišli v pôvodnej forme, ako boli vysielané.

Implementácia 32 bitového cyklického zabezpečovacieho kódu bola použitá varianta CRC-32[4], známa aj pod názvom PKZIP. Využitý polynóm pre konfiguráciu CRC-32 kódu bol `0x04C11DB7`. Zbytok funkcionality je takmer zhodný s 16 bitovým CRC kódom s odlišnosťou výpisu a použitých registrov kvôli veľkosti samotného kódu.

2.3 Lookup tabuľka

Nakoľko tabuľka pri 16 bitovom CRC16 CCITT FALSE je nezávislá od dát bolo možné si ju bokom vyrátať, za použitia jednoduchého Python skriptu spolu s formátovaním pre C a pre účel zrýchlenia samotného výpočtu ju staticky definovať s predvypočítanými hodnotami do programu.

Algoritmus na vypočítanie hodnoty CRC kódu pre daný vstup sa prevádza vo funkcii `TableCRC16`[1], ktorá pracuje postupne bit po bite nad vstupnými dátami a prevádza operáciu XOR s príslušným záznomom v lookup tabuľke. Po prejdení celého bloku dát je výsledkom CRC kód vypočítaný pomocou lookup tabuľky.

Podobne ako u hardwarového výpočtu sa vypočítaná hodnota porovná s referenčnou a prevedie sa experiment príjmateľa na verifikovanie správnosti potencionálne prenesených dát. Tento algoritmus bol inšpirovaný kódom a dokumentáciou projektu crccal.com[1]

Implementácia 32 bitovej tabuľky je obdobná 16 bitovej a taktiež platí, že jej hodnoty sú nezávislé od vstupný dát, čo nám opäť umožnilo statickú definíciu. Na rozdiel od 16 bitovej varianty sú dáta na konci výpočtu ešte invertované, nakoľko CRC32 implementácia používa invertovaný polynóm `0xEDB88320`[6].

2.4 Matematický polynóm

Poslednou variantou je softwarová implementácia matematického polynómu vybraného CRC kódu. Pre 16 bitovú variantu CRC16 CRITT FALSE používame vzorec $x^{16} + x^{12} + x^5 + 1$ a pre zložitejšiu CRC32 variantu

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Implementácia týchto polynómov je jemne zjednodušená s použitím bitových posunov a logických operátorov. Dáta sú prejdené bit po bite a na každú hodnotu je aplikovaný matematický polynóm pre výpočet CRC kódu. Pri 16 bitovom výpočte CRC kódu možno vidieť implementáciu polynómu nasledovne:

```
while (len--){
       tmp = result >> 8 ^ *int_data++;
       tmp ^= tmp >> 4;
       result = (result << 8) ^ ((uint16_t)(tmp << 12)) ^
              ((uint16_t)(tmp << 5)) ^ ((uint16_t) tmp);
       }
```

Pri 32 bitovej variante je výpočet inšpirovaný zo stránky Medium[2] a využíva pre zjednodušenie výpočtu invertovaný polynóm `0xEDB88320`. Dáta sú ako zvyčajne bit po bite prejdené a za použitia logických operácii nad danými bitmi sa po vypočítaní jednoho bajtu použije XOR na výsledok a výpočet pokračuje až do konca vstupných dát. Kvôli využitiu invertovaného polynómu je nutné výsledok nakonci znegovať, aby sme dostali správny CRC súčet. Implementácia vnútorného cyklu výpočtu je nasledovná:

```
while(len--) {
        result ^= *int_data++;
        for(i = 0; i < 8; i++) {
                tmp = result & 1;
                tmp--;
                tmp = ~tmp;
                result = (result >> 1) ^ (0xEDB88320 & tmp);
        }
}
```

Kapitola 3

Štatistiky a réžia

Hardwarová implementácia za použitia CRC modulu čipu K60 je jednoznačne najrýchlejšia, ako sa overilo v testoch pri väčšom vzorku vstupných dát potrebujúcich CRC súčet. Rozmýšlaľ som o implementácii vlastného čítača na vypočítanie samotnej operácie jedného CRC súčtu, ale v hardwarovej podobe by bolo nutné upraviť a prenastaviť veľa registrov a v softwarovej podobe používajúcej volania štandardnej C knižnice by to bolo až príliš pomalé. Preto bolo zvolené testovanie na väčšej vzorke a približne meraný čas vykonania. Taktiež pre podporu odhadu rýchlosti vykonania bol použitý GDB debuger a jeho rozbor kódu na inštrukcie pre približný počet inštrukcií, nutných pre vypočítanie CRC kódu. Nakoľko sa jedná len o 16 a 32 bitové CRC kódy rozdiely neboli až také obrovské, nakoľko sa jedná o relatívne malý výpočet.

Hardwarová implementácia za použitia samotnej hardwarovej akcelerácie a paralelizácie bola jednoznačne najrýchlejšia s najnižším počtom inštrukcií potrebných na výpočet jednoho CRC súčtu.

Pri použití tabuľky na výpočet ktorá bola staticky definovaná dochádza k značnému spomaleniu, nakoľko nieje implementovaný žiaden druh paralelizmu a celý výpočet sa prevádza v jedinom vlákne sekvenčne.

Použitie matematického polynómu pre výpočet CRC kódu je najpomalšie z daných možností, nakoľko je nutné celý výpočet previesť sekvenčne bez pomoci tabuľky, len na základe matematických operácii. Počet inštrukcií bol v tomto prípade najvyšší.

Hlavný rozdiel v implementáciach a ich rýchlostiach je možnosť využitia paralelizmu pre hardwarový výpočet. Aj pri použití softwarového paralelizmu by sme neboli schopný dosiahnuť rýchlosť hardwarového riešenia.

Kapitola 4

Záver

Aktuálna implementácia pokrýva funkcionalitu zo zadania. Pre reálne použitie by bolo nutné zrefaktorovať kód pre vyššiu škálovateľnosť a odstrániť nepotrebné drivery a iné súbory vytvorené z KSDK. Kód názorne ukazuje výhody hardwarových riešení pre rátanie CRC kódu, ktorý môže trvať v odhade do 10 taktov procesoru MK60D10. Program podporuje výpis na konzolu, ktorý je formátovný do čitateľnej podoby prezentujúci výsledky jednotlivých implementácii a ich následné overenie.

Literatúra

[1] Isakov, A.: *CRC Calc*. [Online; navštívené 29.12.2017].
URL http://crccalc.com/

[2] Medium: *CRC-32 without lookup tables*. [Online; navštívené 29.12.2017].
URL
https://medium.com/@nxtchg/crc-32-without-lookup-tables-1682405aa048

[3] RevEng, C.: *Catalogue of parametrised CRC algorithms with 16 bits*. [Online; navštívené 29.12.2017].
URL http://reveng.sourceforge.net/crc-catalogue/16.htm#crc.cat-bits.16

[4] RevEng, C.: *Catalogue of parametrised CRC algorithms with 17 or more bits*. [Online; navštívené 29.12.2017].
URL
http://reveng.sourceforge.net/crc-catalogue/17plus.htm#crc.cat-bits.32

[5] Semiconductors, F.: *Getting Started with Kinetis SDK (KSDK) v.2.0*. [Online; navštívené 29.12.2017].
URL https://www.nxp.com/docs/en/user-guide/KSDK20GSUG.pdf

[6] Wikipedia: *Cyclic redundancy check*. [Online; navštívené 29.12.2017].
URL
https://en.wikipedia.org/wiki/Cyclic_redundancy_check#CRC-32_algorithm