

Projekt: ARM/FITkit3: Zabezpečení dat pomocí 16/32-bit. kódu CRC v IMP 2017/2018
Meno a priezvisko: Dávid Bolvanský
Login: xbolva00

1 Cieľ projektu

Projekt má za úlohu demonštrovať možnosti zabezpečenia dát 16/32-bitovým kódom CRC na čipe Kinetis K60 z dosky platformy FITkit 3. Projekt bol vypracovaný pomocou Kinetis Design Studio v3.0 a SDK API pre CRC a UART.

2 Blok dát a jeho zabezpečenie

Za vhodný blok dát bol vybraný reťazec – text Lorem Ipsum, ktorý sa následne zabezpečoval pomocou CRC16/32. Za **CRC16** v tomto projekte berieme CRC-16-ANSI/CRC-16-IBM s polynómom **0x8005** (obrátené 0xA001). **CRC32** pracuje s polynómom **0x04C11DB7** (obrátené 0xEDB88320).

3 Výpočet CRC pomocou HW modulu Cyclic Redundancy Check

Interakcia s CRC modulom je vykonávaná pomocou SDK API určených na prácu s CRC modulom. V štruktúre **crc_config_t** bolo potrebné nastaviť požadovaný polynóm a počiатnú seed hodnotu. Pomocou funkcie **CRC_Init** dôjde k inicializácii CRC modulu. Pomocou funkcie **CRC_WriteData** zapíšeme dáta na zabezpečenie. Následne, pomocou funkcie **CRC_Get16bitResult** / **CRC_Get32bitResult** získame výsledný kontrolný CRC súčet.

4 Výpočet CRC pomocou polynómu z tabuľky

Tabuľka predstavuje optimalizačný mechanizmus výpočtu CRC, keďže obsahuje výpočty, ktoré by sa inak opätovne počítali pre každý bajt reťazca.

Nižšie uvedený pseudokód¹ bol použitý v projekte a prezentuje použitie tabuľky pri výpočte CRC.

Vstup: Blok dát

Výstup: Vypočítaná hodnota CRC32

$\text{crc32} \leftarrow \text{SEED}$

for *bajt v bloku dát* **do**

$\text{table_index} \leftarrow (\text{crc32} \text{ xor } \text{bajt}) \text{ and } 0\text{xFF}$

$\text{crc32} \leftarrow (\text{crc32} \text{ shr } 8) \text{ xor } \text{crc32_table}[\text{table_index}]$

end

$\text{crc32} \leftarrow \text{crc32} \text{ xor } 0\text{xFFFFFFFF}$

return crc32

Algorithm 1: Pseudokód výpočtu CRC32 pomocou polynómu z tabuľky

Zdroj CRC16 tabuľky pre polynóm 0x8005: <https://github.com/torvalds/linux/blob/master/lib/crc16.c>

Zdroj CRC32 tabuľky pre polynóm 0x04C11DB7: http://www.ipgp.fr/~tuchais/earthworm/v6.3/src/data_sources/naqs2ew/crc32.c

¹http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html

5 Výpočet CRC pomocou polynómu (základný algoritmus)

Základný algoritmus výpočtu je popísaný nasledujúcim pseudokódom².

```
Vstup: Blok dát  
Výstup: Vypočítaná hodnota CRC  
crc ← SEED  
for bajt v bloku dát do  
|   crc ← crc xor bajt;  
|   for i = 0; i < 8; i = i + 1 do  
|   |   crc = (crc shr 1) xor (OBRATENY_POLYNOM and -(crc and 1));  
|   end  
end  
return crc32
```

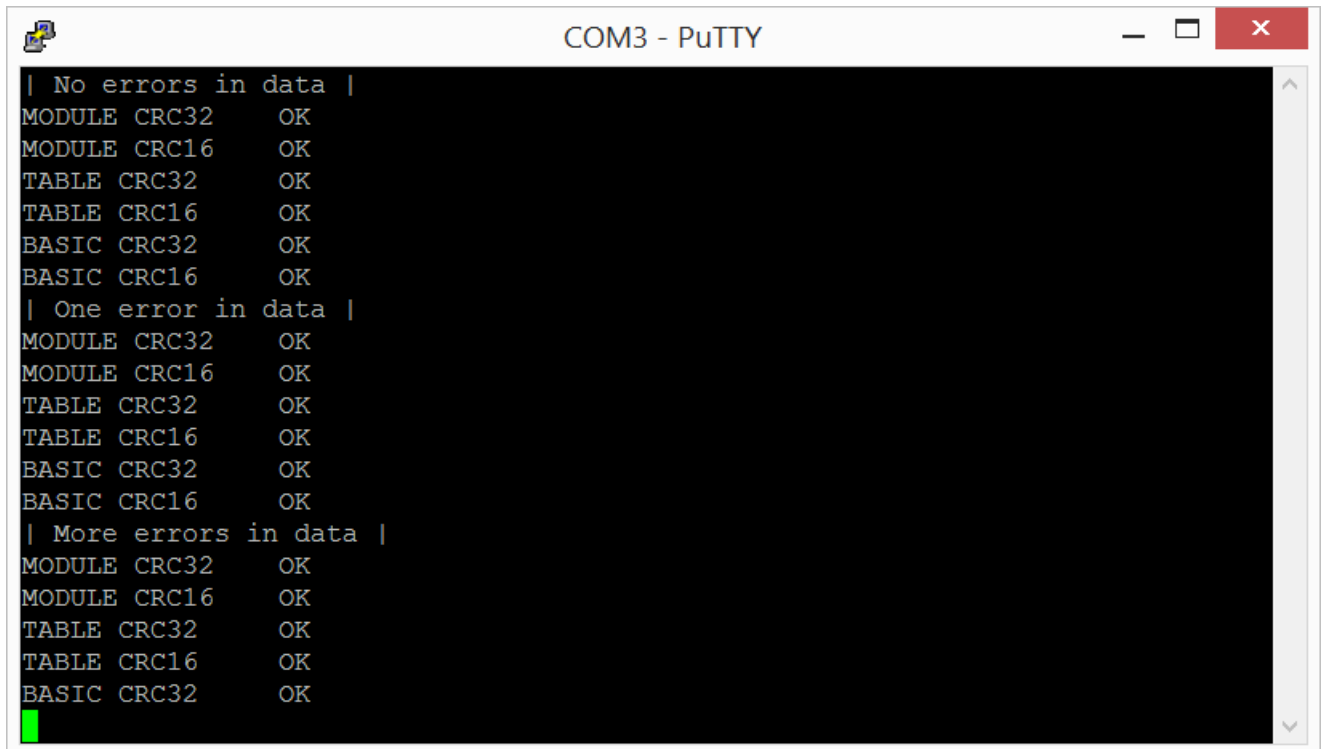
Algorithm 2: Pseudokód výpočtu pomocou polynómu

Tento pseudokód základného algoritmu na výpočet CRC16/32 bol implementovaný v projekte.

6 Overenie správnosti výpočtov CRC

Pre zvolený blok dát sme si predvypočítali CRC16/32 hodnoty, ktoré následne porovnávame s hodnotami získali vyššie spomenutými spôsobmi výpočtu CRC. Následne sme do bloku dát zanesli jednu / viac chýb a vykonali výpočty a porovnania znova.

Informácie o správnosti vypočítaných hodnôt pre jednotlivé spôsoby výpočtu sú prezentované na terminál pomocou UARTu. Práca s UARTom prebieha taktiež pomocou SDK API funkcií, menovite **UART_Init** na inicializáciu UARTu a **UART_WriteBlocking** na zápis dát.

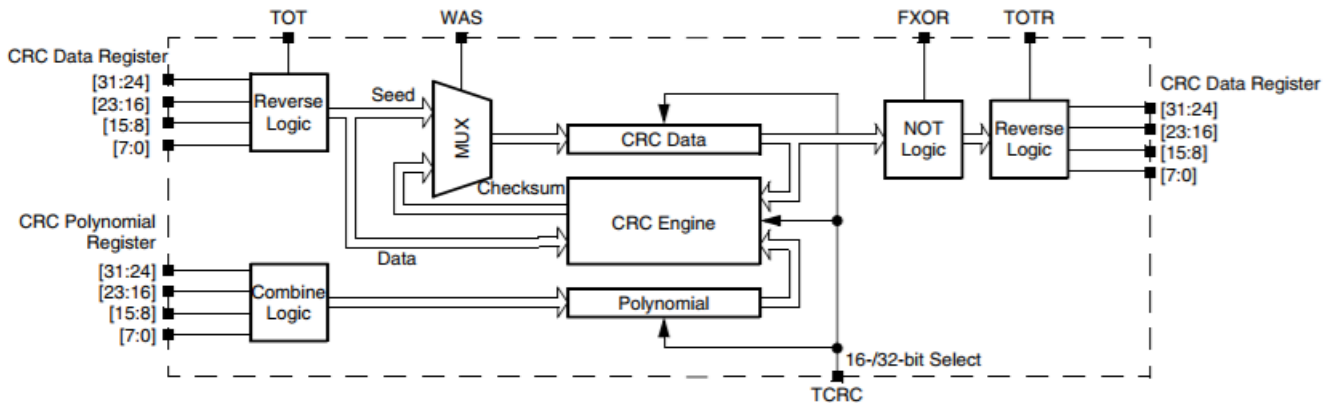


Obr. 1: Výstup aplikácie

²https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks

7 Realizačná a výpočetná režia

Obvod hardvérového CRC generátora používa 16/32-bitový programovateľný posuvný register.³



Obr. 2: Blokový diagram CRC modulu na čipe K60

Podporuje detekciu jednoduchých, dvojchýb a väčšiny viacbitových chýb. Hodnota polynómu, seedu je programovateľná. Ponúka vysokorýchlostný výpočet CRC. Voliteľná funkcionálna transformácia vstupných dát a výsledku cez transformačný register (pre aplikácie pracujúce s LSB formátom). Možnosť obrátenia (reverzácie) výsledného CRC.

Režimy MCU ovplyvňujú funkcionálnosť CRC modulu – pri režime nižšieho výkonu, ktorý vypne hodiny, dochádza k prerušeniu CRC výpočtu. Obnoví sa po znovu spustení hodín alebo po systémovom resete, ktorý spôsobí opustenie tohto režimu.

Softvérovým, ale menej efektívnym, riešením, je použitie základného algoritmu výpočtu CRC. Keďže sa počas výpočtu CRC pre každý bajt opakujú niektoré výpočty, vytvorila sa optimalizovaná metóda výpočtu CRC pomocou tabuľky s predvypočítanými hodnotami. Experimenty poukazujú na 5-násobne menej vykonaných inštrukcií v prípade výpočtu CRC pomocou tabuľky oproti základnému algoritmu⁴.

Softvérové riešenia výpočtu CRC vyžadujú certifikáciu, že spĺňajú požiadavky daného protokolu. Hardvérový CRC modul už tieto podmienky spĺňa. Softvérové riešenie vyžaduje niekoľko stoviek zbernicových cyklov (700) na spracovanie jedného bajtu cez CRC algoritmu. Hardvérový modul dokáže posunúť bajt do CRC generátora v jednom zbernicovom cykle, čím dochádza k znateľnému zvýšeniu rýchlosti výpočtu.

Merania na čipe MC9S08AC128 ukazujú, že softvérové riešenie spĺňajúce protokol CRC16-CCITT bežiacie s rýchlosťou zbernice 20 MHz trvá približne 6,7 sekundy, aby vykonalo kontrolu cyklickým kódom pre 128 KB flash pamäte. Na porovnanie rovnaká kontrola zaberie len 170 ms pri použití hardvérového CRC modulu, čo je výrazne rýchlejšie riešenie⁵.

Vysoká rýchlosť výpočtu CRC pomocou hardvérového modulu nad softvérovými riešeniami preto predurčuje CRC modul ako ideálneho kandidáta na vykonávanie kontrol cyklickým kódom pri prenosoch dát s minimálnou reziou.

³http://cache.freescale.com/files/32bit/doc/ref_manual/K60P144M100SF2V2RM.pdf

⁴<https://barrgroup.com/Embedded-Systems/How-To/CRC-Calculations-C-Code>

⁵<https://www.nxp.com/docs/en/application-note/AN3795.pdf>