FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

IMP - Mikroprocesorové a vestavěné systémy
Testovanie mikrokontrolérov
2019 / 2020

15. decembra 2019

Adrián Boros (xboros03)

1 Úvod

Cieľom tohto projektu bolo vytvoriť vstavanú aplikáciu pre samočinné testovanie mikrokontroléru **Kinetis K60** z platformy FITkit 3. Úlohou bolo nájsť alebo vytvoriť mechanizmy vhodné na testovanie zvolených častí mikrokontroléru, napr. hodín, pamätí RAM/flash, registrov, zasobníku, I/O, ADC/DAC a toku programu vhodnými programovými prostriedkami v jazyku `C`.

2 Druh testovania

Rozhodol som sa použiť softwarové testovanie, známe pod skratou SBST (software based self testing). V tomto prípade je testovanie vykonávané za behu a vykonáva ju procesor. Ten spracováva jednotlivé inštrukcie testu, ktoré sú v mojom prípade implementované v jazyku C alebo v jazyku symbolických inštrukcií. Výhodou softwarového testovania je, že nie je závislé na hardware pri úprave alebo rozšírení testu. Znamená to, že ak prebehla aktualizácia alebo úprava hardwarového testu, tak sa musí upraviť aj návrh hardware a musí prebehnúť nová verifikácia.

3 Návrh a popis implementácie

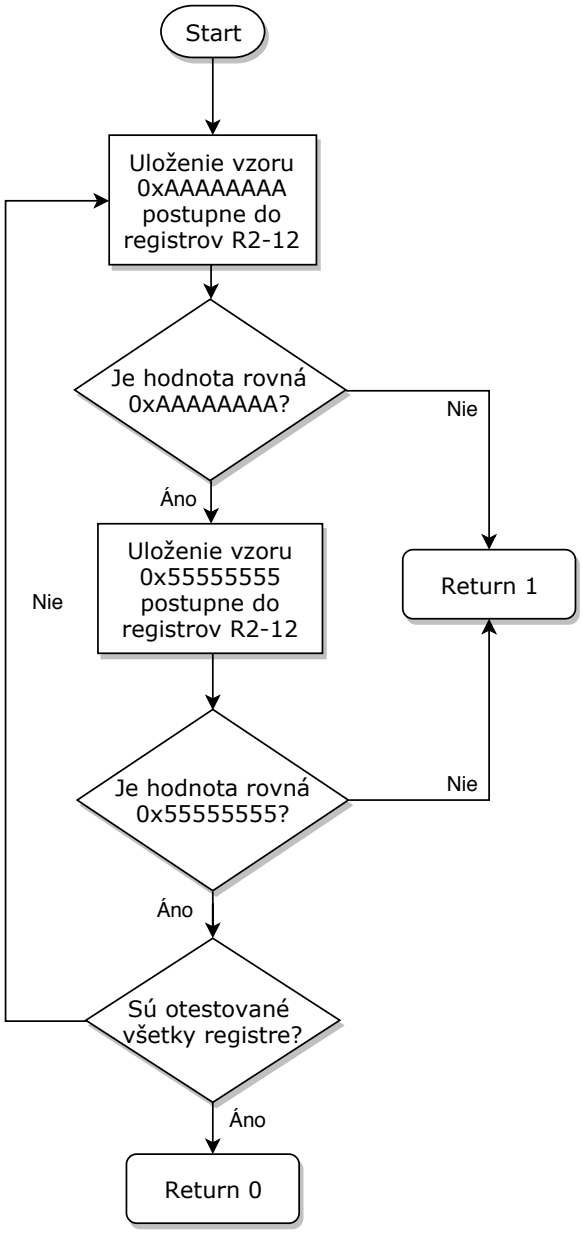
V tejto kapitole je popis riešenia problematiky tohto projektu a popis jednotlivých implementovaných testov. Celá implementácia sa nachádza v súbore `main.c`. K implementácii bol využitý manuál k mikrokontroléru a časti zdrojových kódov z cvičení a democvičení.

3.1 Testovanie CPU registrov

Cieľom testu CPU registrov je odhaliť **stuck-at** poruchu. Znamená to, že ak má aspoň jeden bit registru konštantne hodnotu 0 alebo 1, tak túto skutočnosť musí test odhaliť. Tento test je implementovaný pomocou algoritmu **Checkerboard**. Princíp tohto testu je založený na nahrávaní a následnej kontrole dvoch testovacích vzorov. Postup testu je nasledovný:

1. Uloženie vzoru **0xAAAAAAAA** do registru
2. Kontrola či hodnota v registri je rovná hodnote **0xAAAAAAAA**
3. Uloženie vzoru **0x55555555** do registru
4. Kontrola či hodnota v registri je rovná hodnote **0x55555555**

Vyššie spomínané hodnoty **0xAAAAAAAA** a **0x55555555** nie sú zvolené náhodne, sú si navzájom negáciou. Úlohou testu je odhaliť nielen stuck-at poruchu ale overiť aj to, či hodnota nejakej pamäťovej bunky neovplyvňuje susednú bunku, čo by iné testovacie vzory pravdepodobne neodhalili.

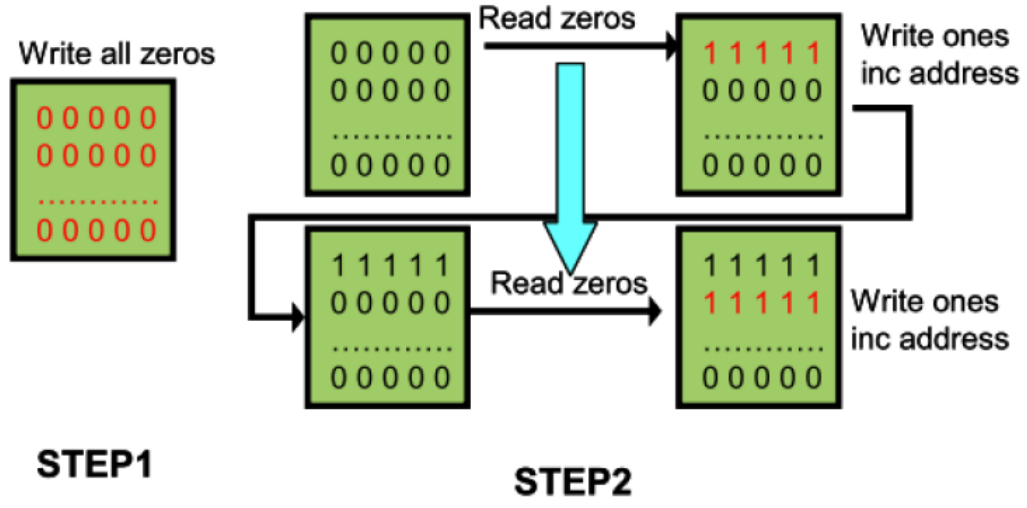


Obr. 1: Vývojový diagram testu CPU registrov

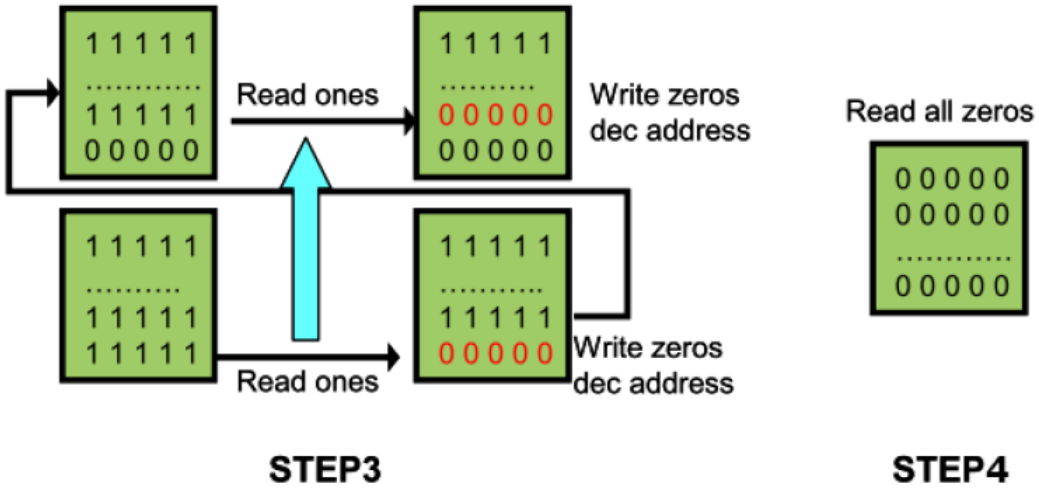
Na obrázku 1 sa nachádza vývojový diagram ktorý zobrazuje postupnosť krokov pri testovaní CPU registrov. Pri implementácii tohto testu bolo využité kľúčové slovo **__asm** ktoré vyvolá inline assembler. Pomocou tohto klúčového slova sa ľahšie pristupuje k jednotlivým registrom a ich nastavenie je jednoduchšie. Výhodou je, že sa dá bezproblémovo kombinovať s ostatnými konštrukciami z jazyka C. Pred začiatkom testu sa do registrov **R0** a **R1** uložia vzory **0xAAAAAAAA** resp. **0x55555555**. Následne sa pomocou assemblerovského príkazu `PUSH` uložia hodnoty v registroch R2 - R12 na zásobník, čím sa zabezpečí že hodnoty sa nestratia počas vykonávania testu. Nasleduje postupné uloženie spomínaných hodnôt do registrov a ich porovnávanie. Po kontrole všetkých registrov sa príkazom `POP` obnovia pôvodné hodnoty v registroch. Chybu v registroch je náročné odhaliť za behu programu, pretože by sa chyba musela zaviesť medzi dve inštrukcie ktoré sú vykonávané za sebou. Chyba sa však dá nasimulovať v debugger móde, kde sa manuálne prepíše hodnota v registri počas kontroly.

3.2 Testovanie RAM pamäte

Najčastejším spôsobom testovania RAM pamäte je periodické testovanie statickej pamäte. Úlohou testu RAM pamäte je detekovať **stuck-at** poruchu alebo detekovať susedné pamäťové bunky ktoré sa navzájom ovplyvňujú. Najčastejšími spôsobmi testovania RAM pamäte sú: **test statickej pamäte** alebo **bitová redundancia**. V projekte bol implementovaný test statickej pamäte a to pomocou **March X** algoritmu. Princíp algoritmu je znázornený na obrázkoch 2 a 3.



Obr. 2: Kroky 1 a 2 algoritmu March X



Obr. 3: Kroky 3 a 4 algoritmu March X

Celé testovanie prebieha v niekoľkých cykloch, kde v každej iterácii cyklu sa otestuje určitá časť RAM pamäte. Jednotlivé kroky testu:

1. Do dvojrozmerného poľa sa zálohujú hodnoty z testovanej oblasti - `STEP1` z obrázku 2
2. Vynulovanie testovanej oblasti - `STEP1` z obrázku 2

3. Vzostupne prebieha zápis hodnoty **0x00** a jeho následná kontrola a po kontrole sa zapisuje hodnota **0x11** - `STEP2` z obrázku 2

4. Zostupne prebieha načítanie hodnoty **0x11** a jeho kontrola a nasleduje zápis hodnoty **0x00** - `STEP13` z obrázku 2

5. Skontroluje sa či je celá oblasť rovná **0x00000000** - `STEP4` z obrázku 2

6. Obnova pôvodných hodnôt z poľa pre zálohu - `STEP4` z obrázku 2

Chybu je možné simulovať rovnakým spôsobom ako pri prvom teste, t.j. pri debugovaní. Princíp simulácie chyby je rovnaký, keď v priebehu testu je program pozastavený a hodnota na testovanej adrese je zámerne prepísaná. Test ihneď túto chybu odhalí.

3.3 Indikácia ne/úspešnosti testov

Na indikovanie úspešnosti alebo neúspešnosti jednotlivých testov je použité pípanie bzučiaka alebo blikanie LED, kde pípanie značí úspešný test a blikanie neúspešný. K implementácii týchto javov bolo potrebné povoliť hodiny pre porty `PORTA` a `PORTB`. Taktiež bolo potrebné inicializovať tie porty, na ktorých sú pripojené LED alebo bzučiak a nastaviť tieto porty ako výstupné. Bola implementovaná funkcia na oneskorenie ktoré bolo použité pri pípaní, blikaní LED ale aj medzi jednotlivými testami aby sa odlíšilo ktorý test práve skončil. Všetky tieto funkcie boli použité z cvičení a upravené pre `FITKIT3`.

4 Záver

Program bol otestovaný, behom týchto testov som nenarazil na žiadne väčšie problémy a všetky testy prebehli úspešne. Program bol vytvorený v prostredí `Kinetis Design Studio` od firmy `NXP`.