FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Implementace TRNG založeného na SRAM na mikrokontroléru
Student:	Daniel Jantošovič
Vedoucí:	Ing. Filip Kodýtek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Nastudujte problematiku generátorů skutečně náhodných čísel (TRNG) se zaměřením na TRNG vhodné pro mikrokontroléry. Věnujte pozornost především TRNG využívajících obsah SRAM po zapnutí napájení jako zdroj entropie. Na zvolené cílové platformě s ARM procesorem implementujte vhodnou variantu TRNG za použití prostředků zvolené platformy. Proveďte vyhodnocení implementovaného řešení a diskutujte své výsledky a možná vylepšení.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 8. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Implementace TRNG založeného na SRAM na mikrokontroléru

Daniel Jantošovič

Katedra informační bezpečnosti
Vedúci práce: Ing. Filip Kodýtek, Ph.D.

11. mája 2022

Poďakovanie

Chcel by som poďakovať vedúcemu práce Ing. Filipovi Kodýtkovi, Ph.D za venovaný čas a poskytnuté rady pri tvorbe tejto bakalárskej práce. Taktiež ďakujem svojej rodine, priateľke a priatelom za podporu počas celého štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 11. mája 2022 . . . . . .

Odkaz na túto prácu

Jantošovič, Daniel. *Implementace TRNG založeného na SRAM na mikrokontroléru*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Táto bakalárska práca sa zaoberá generátormi náhodných čísel. Na začiatku sa venuje ich rozdeleniu z hľadiska determinizmu či konštrukčným princípom a uvádza príklady využiteľnosti v praxi. Následne sa hlbšie venuje generátorom skutočne náhodných čísel implementovaným na mikrokontróleroch. Nasleduje návrh a implementácia generátora náhodných čísel s využítím stavu statickej RAM pamäte po zapnutí napájania na mikrokontroléri s procesorom architektúry ARM Cortex-M4. V závere je navrhnutý generátor podrobený štatistickým testom.

Klíčová slova generátor náhodných čísel, entropia, mikrokontróler, SRAM, NIST, TRNG, PRNG

Abstract

This bachelor thesis deals with random number generators. At the beginning, it deals with their classification in terms of determinism or design principles and gives examples of usability in practice. Subsequently, it delves deeper into the generators of true random numbers implemented on microcontrollers. The following is the design and implementation of a random number generator with the power-up state of the static RAM cells as the source of its entropy on a microcontroller with a processor of ARM Cortex-M4 architecture. Finally, the implemented generator is subjected to a statistical tests

Keywords random number generator, entropy, microcontroller, SRAM, NIST, TRNG, PRNG

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Náhoda. Jej prejavy nás ovplyvňujú na dennej báze, či už si to uvedomujeme alebo nie. Náhodu ale takisto dokážeme využiť a to naprieč širokým spektrom oblastí, či už ide o gambling, výber poroty v anglosaskom právnom systéme alebo aj rôzne vedecké metódy využívajúce náhodné čísla ako napríklad metóda Monte Carlo. Avšak jedným z dnes najbežnejších a zároveň najdôležitejších využití náhody je v kryptografii.

V dnešnom svete je kladený silný dôraz na bezpečnosť elektronickej komunikácie. Posielanie emailov či instantných správ, využívanie webových služieb, pripájanie sa na vzdialené pracoviská, to všetko a mnoho ďalšieho štandardne podlieha zabezpečeniu. To je dosiahnuté predovšetkým využívaním šifrovania dát pomocou známych algoritmov. Jedným zo základných kameňov bezpečnosti takýchto algoritmov je kryptograficky kľúč, na ktorý je kladená netriviálna požiadavka a to nepredvídateľnosť. Efektívnym spôsobom ako to dosiahnuť je ako kľúč využiť náhodné číslo dostatočnej veľkosti.

S rozvíjajúcim sa segmentom Internetu vecí a vstavaných systémov je v našom okolí čoraz viac elektronických zariadení rôzneho druhu s potrebou zabezpečiť svoje dáta alebo komunikáciu na internete. To, okrem iného, znamená potrebu schopnosti generovania náhodných čísel.

Táto práca sa zaoberá problematikou generovania náhodných čísel a možnosti implementácie na bežne dostupných mikrokontroléroch. Výstup tejto bakalárskej práce môže pomôcť zoznámiť sa s princípmi a využívanými metódami generovania skutočne náhodných čísel a priblížiť postup pri implementácii generátoru na mikrokontroléroch.

Štruktúra práce

- Prvá kapitola sa venuje generátorom náhodných čísel, ich klasifikáciou a rozdeleniu podľa zdroja náhody. Ďalej sú predstavené rôzne typy generátorov náhodných čísel.
- Druhá kapitola je venovaná generátorom náhodných čísel implementovaným na mikrokontroléroch.
- V tretej kapitole je predstavený návrh generátora náhodných čísel pre zvolenú platformu.
- Štvrtá kapitola obsahuje analýzu meraní entropie obsiahnutej v statickej RAM pamäti mikrokontroléra.
- Piata kapitola predstavuje štatistické testy NIST a analyzuje výsledky týchto testov vykonaných na implementácii.

Ciele práce

- Zvoliť vhodný typ a navrhnúť generátor náhodných čísel pre mikrokontrolér STM32F446RE.
- Implementovať navrhnutý generátor na danom mikrokontroléri.
- Vyhodnotiť generované postupnosti čísel štatistickými testami.

Kapitola 1

Generovanie náhodnych čísel

Táto kapitola sa venuje náhode, jej využívaniu v praxi a zaoberá sa problémom generovania náhodných čísel.

1.1 Náhodné čísla

Náhodné čísla nachádzajú vo svete široké využitie. Príkladmi môžu byť ich použitie v digitálnych kasínach, pri počítačových simuláciách komplexných fyzikálnych javov či pri výbere štatistickej vzorky z väčšieho dátového setu. Menej výrazným ale rozhodne dôležitým využitím takýchto čísel je v oblasti zabezpečenia informačných systémov, dát či komunikácie a to pri tvorbe kryptografických kľúčov používaných pre šifrovacie algoritmy alebo aj pri tvorbe čísel „nonce“ (Number used once), teda čísel s jednorazovým použitím, využívaných napríklad autentizačnými protokolmi alebo pri nadväzovaní zabezpečeného sieťového spojenia pomocou protokolu TLS.

Náhodným číslom môžeme rozumieť jednotlivé číslo vybrané z množiny čísel, pre ktoré platí rovnaká pravdepodobnosť výberu. V prípade náhodných čísel ako postupnosti čísel musia byť tieto jednotlivé čísla navzájom štatisticky nezávislé [1].

1.1.1 Entropia

Pre potreby vyhodnotenia zdroja náhody pri generovaní náhodných čísel sa využíva pojem entropie. Tento pojem z mnoha vedeckých či technických odborov má širokú škálu definícii, prevažne sa však chápe ako miera neusporiadanosti systému. Pre potreby tohto textu bude použitá informačná entropia H(X), tiež nazývaná shannonovská entropia [2], ktorá je zadefinovaná nasledovne:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log(P(x_i)),$$

kde náhodná veličina X reprezentuje informačný zdroj a $P(x_i)$ značí pravdepodobnosť výskytu vzoru x_i. Ďalej budeme využívať pojem minimálnej entropie, ktorá odpovedá entropii najpravdepodobnejšieho stavu zdroja entropie [3]:

$$H_{min}(X) = -log(P_{max})$$

1.1.2 Štatistické testy

Na vyhodnotenie či nejakú postupnosť čísel môžeme považovať za náhodnú slúžia štatistické testy. Sadu takýchto testov pripravil americký Národný inštitút štandardov a technológie (NIST) [4]. Tieto testy berú na vstup postupnosť bitov a kontrolujú napríklad rovnomerné rozloženie nulových a jednotkových bitov naprieč postupnosťou, frekvenciu ich výskytu, či už samostatne alebo v rámci blokov rôznej veľkosti, alebo vyhľadávajú opakujúce sa vzory. Jednotlivé testy sú predstavené v kapitole 5.

1.2 Generátory náhodnych čísel

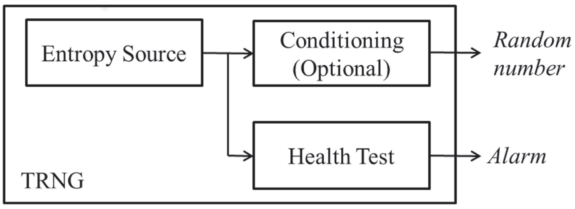
Pojem generátor náhodných čísel (RNG) je pomerne široký a rozumieme pod ním proces, ktorého výstupom je náhodná postupnosť čísel [5]. Princíp tvorby náhodných čísel sa výrazne odlišuje v závislosti na type generátora.

1.2.1 Generátor skutočne náhodných čísel

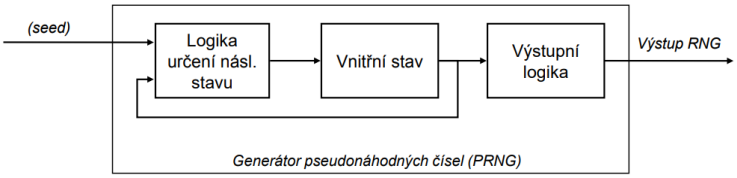
Generátor skutočne náhodných čísel (TRNG) je nedeterministický. Na ilustrácii 1.1 môžeme vidieť základnú štruktúru TRNG [6]. Základným prvkom je zdroj entropie. Jeho zdigitalizovaný výstup je možné pokladať za náhodný, ale takto priamo generovaná postupnosť bitov nemusí mať dostatočne dobré štatistické vlastnosti ako napríklad náklonnosť k jednej z možných hodnôt bitu. Preto je často využívaný postprocessing, na 1.1 značený ako „conditioning“. Ten má za úlohu vyextrahovať zo zdroja číslo s maximálnou entropiou a dobrými štatistickými vlastnosťami. Súčasťou TRNG bývajú aj zabudované testy, ktoré kontrolujú zlyhanie zdroja entropie. Pre postprocessing sa zvyčajne využíva Von Neumannov korektor alebo hashovacie funkcie ako SHA-1. Zdrojom entropie bývajú fyzikálne javy ako napríklad kvantové procesy v polovodičoch, kozmické žiarenie, tepelný šum v elektrických obvodoch alebo aj správanie užívateľa, ako sú pohyby myšou či frekvencia stláčania kláves [6] [7].

1.2.2 Generátor pseudonáhodných čísel

Generátor pseudonáhodných čísel (PRNG) používa deterministický algoritmus. Tento algoritmus využíva vstup, tzv. „seed“, z ktorého generátor určí svoj počiatočný stav. Postupne pomocou matematických operácii prechádza do ďalších stavov a generuje sekvenciu čísel, čo demonštruje 1.2. Tento vstup



Obr. 1.1: Základný princíp fungovania TRNG [6]



Obr. 1.2: Základná štruktúra PRNG [5]

jednoznačne určuje výslednú postupnosť generovaných čísel a pomocou tohto vstupu je možné generovanú sekvenciu opakovane zreprodukovať. Zdrojom entropie pre PRNG je práve hodnota „seed“. Množstvo entropie tohto vstupu je zároveň maximálnou entropiou generovaných postupností. Tieto postupnosti pseudonáhodných čísel majú vhodné štatistické vlastnosti a spĺňajú štatistické testy, ale z princípu tvorby týchto čísel ich nemožno považovať za skutočne náhodné [7].

1.2.2.1 Kryptograficky bezpečné generátory

V rámci PRNG rozlišujeme podtriedu Kryptograficky bezpečných generátorov pseudonáhodných čísel (CSPRNG). Výstupy z takýchto generátorov sa považujú za dostatočne bezpečné pre ich využívanie v kryptografii. Pre zaradenie generátoru medzi CSPRNG musí spĺňať nasledovné dve podmienky.

Prvou je splnenie next-bit testu. Ten je splnený za predpokladu, že ak je k dispozícii predošlá vygenerovaná sekvencia bitov, neexistuje algoritmus, ktorý by bol v polynomiálnom čase schopný predpovedať nasledujúci bit s presnosťou väčšou ako 50%.

Druhou podmienkou je odolnosť voči útokom na odhalený stavu generátora. V prípade ak útočník odhalí aktuálny stav PRNG, nesmie byť možné z neho zrekonštruovať predošlé vygenerované sekvencie čísel [8].

1.2.3 Porovnanie PRNG a TRNG

Determinizmus PRNG so sebou prináša výhody aj nevýhody pri porovnaní s TRNG. Hlavnou výhodou generovania čísel pomocou deterministického algoritmu je rýchlosť. Vzhľadom na absenciu potreby merať externé javy a optimalizáciu využívaných matematických funkcií je rýchlosť generovania nových čísel výrazne vyššia oproti TRNG. Za ďalšiu výhodu môžeme do istej miery považovať aj reprodukovateľnosť výstupu. To sa využíva pri generovaní kľúčov za použitia prúdových šifier.

Na druhej strane, nevýhody deterministických PRNG sú takisto spôsobené reprodukovateľnosťou, a síce ak by sa útočníkovi podarilo odhaliť vstupnú hodnotu „seed“, dokáže si vytvoriť identickú sekvenciu pseudonáhodných čísel.

V praxi sa môžeme stretnúť s kombináciou týchto dvoch typov generátorov. Pomocou TRNG sa vytvorí skutočne náhodná hodnota, ktorá sa používa ako vstupná hodnota „seed“ pre PRNG. Takto získame rýchly a bezpečný spôsob generovania čísel [9].

1.2.4 Príklady PRNG

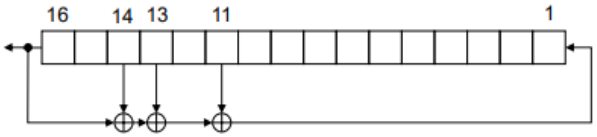
V tejto sekcii uvádzam príklady generátorov pseudonáhodných čísel.

Posuvný register s lineárnou spätnou väzbou

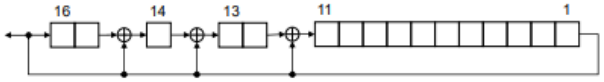
Posuvný register s lineárnou spätnou väzbou(LFSR) je posuvný register, ktorého vstupný bit je výsledkom lineárnej funkcie predošlého stavu registra. LFSR pozostáva z masky registra a inicializačného vektoru, ktorý predstavuje vstupnú hodnotu „seed“. Maska registra je polynomiálna funkcia namapovaná na bity registra podľa stupňa polynómu. Poznáme dva typy LFSR:

- S vonkajšou spätnou väzbou (Fibonacci LFSR)
- S vnútornou spätnou väzbou (Galois LFSR)

Oba tieto typy využívajú operáciu XOR pre výpočet nasledujúceho stavu. Maska registra určí bity, s ktorými sa má operácia XOR vykonať. Výstupom takéhoto generátoru je bit na určenej zafixovanej pozícií registra [9]. Rozdielny prístup týchto typov LFSR je znázornení na 1.3 a 1.4. LFSR je pre svoju jednoduchosť implementácie rozšírený, avšak nie je vhodný pre kryptografické využitie.



Obr. 1.3: Fibonacciho LFSR s vonkajšou spätnou väzbou [5]



Obr. 1.4: Galoisov LFSR s vnútornou spätnou väzbou [5]

Blum Blum Shub

Medzi kryptograficky bezpečné PRNG radíme algoritmus Blum-Blum-Shub. Výstup tohto PRNG je určený rekurzívnym vzťahom

$$x_i = (x_{i-1}^2)\ mod\ M,$$

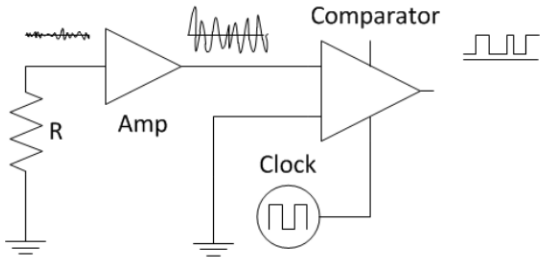
kde M je súčin dvoch prvočísel p a q, ktoré sú rovnakej dĺžky a sú kongruentné 3 modulo 4. Číslo M a hodnota x_0, považovaná za „seed“, sú vstupnými parametrami pre tento PRNG. Hodnotu x_i je možné vypočítať aj priamo z počiatočnej hodnoty x_0 pomocou vzorca

$$x_i = x_0^{2^i\ mod\ \lambda(M)}\ mod\ M,$$

kde λ je Carmichaelova funkcia. Výstupný bitstream $B = \{b_0, b_1, ..., b_i\}$ je tvorený paritou príslušných hodnôt x_i [10].

Iné

Ďalšími využívanými príkladmi PRNG sú generátory založené na hashovacích funkciách, MAC funkciách či blokových šifrách v režime čítača.



Obr. 1.5: Konštrukcia TRNG využívajúca zosilnený tepelný šum rezistora. [14]

1.2.5 Typy TRNG

V tejto sekcii sú uvedené princípy a konštrukcie využívané pre generovanie skutočne náhodných čísel.

Fyzikálne procesy pre TRNG

Ako zdroj entropie pre TRNG sa často využívajú komplexné, nepredvídateľné, ale zároveň pozorovateľné a merateľné fyzikálne javy.

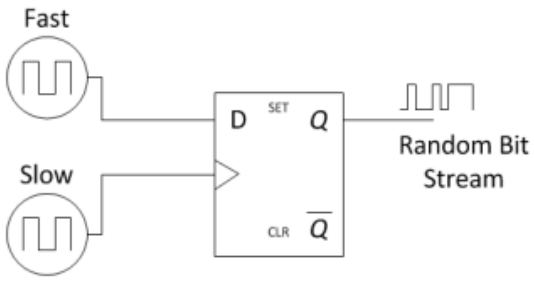
- Atmosférický šum [1].
- Tepelný šum [11] [12].
- Opozdenie alebo nestabilita hodín [13].

1.2.5.1 Zosilnený tepelný šum

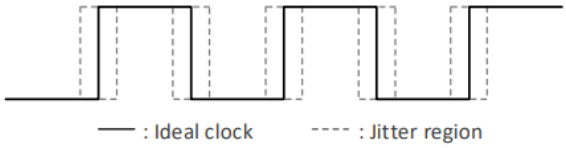
Tepelný šum ako zdroj entropie je využívaný vo viacerých konštrukciách TRNG [14]. Jedna zo základných konštrukcii využíva zosilnený tepelný šum rezistora, spôsobený napr. Brownovým pohybom elektrónov. Zosilnený signál následne spracovaný komparátorom a jeho výstup je využitý ako bitstream. Táto konštrukcia je zobrazená na 1.5.

1.2.5.2 Nestabilita hodinového signálu

Ďalšou možnosťou je využitie nestability hodinového signálu. Konštrukcia zobrazená na 1.6 využíva dva hodinové signály a klopný obvod. Rýchlejšie z hodín sú privedené do obvodu na vstup D, zatiaľ čo pomalšie sú zvolené ako vstup hodinového signálu pre klopný obvod. V prípade, že „jitter“ pomalších hodín je rovnako veľký ako perióda rýchlejších hodín, pravdepodobnosti výstupných hodnôt klopného obvodu sú $\frac{1}{2}$ [14]. Pod pojmom „jitter“



Obr. 1.6: Konštrukcia TRNG využívajúca zosilnený tepelný šum rezistora. [14]



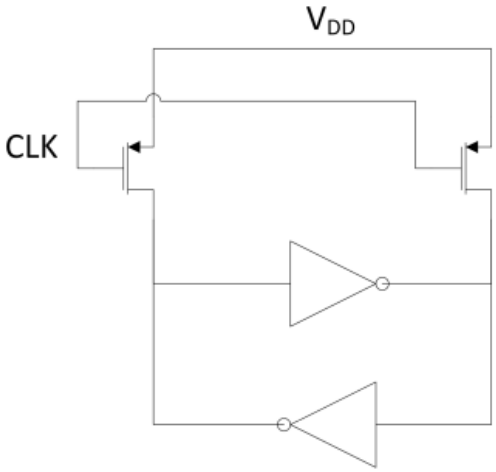
Obr. 1.7: Jitter. [15]

rozumieme odchýlku periodického signálu. V tomto prípade ide o nestabilitu oscilátora generujúceho hodinový signál. Na 1.7 je zobrazený hodinový signál s vyznačenými možnými oblasťami prechodov signálu medzi stavmi 0 a 1 z dôvodu nestability signálu.

1.2.5.3 Metastabilita obvodu

Ďalším prístupom je vytvorenie metastabilného obvodu. Príklad takého obvodu je zobrazený na 1.8. Tento obvod obsahuje dva invertory zapojené tak, že výstup jedného je zároveň vstupom druhého a opačne. Do tohto obvodu je zapojené napätie V_{DD} pomocou hodinovým cyklom ovládanými tranzistormi.

V momente odpojenia obvodu od zdroja napätia dochádza k zníženiu napätia na oboch stranách invertorov na polovicu V_{DD}. V tomto stave obvod ostáva až kým z dôvodu tepelného šumu v niektorom z invertorov dochádza k zvýšeniu napätia na jednej strane obvodu a tým sa výstup jedného z invertorov zmení na hodnotu 1 a výstup druhého invertoru na hodnotu 0 [14].



Obr. 1.8: Konštrukcia metastabilného obvodu z 2 invertorov [14]

Kvantové procesy pre TRNG

Kvantová mechanika ponúka škálu procesov, ktoré sú vhodné pre využitie pri generovaní skutočne náhodných čísel. Ich vhodnosť pre TRNG nespočíva len v komplexnosti, ktorá znemožňuje kvalitný odhad ale aj v princípe týchto javov ako takých. Fyzikálne teórie a modely, ktoré tieto javy opisujú, pracujú s pravdepodobnosťami a náhodou. Kvantové javy tak ako ich dnes poznáme považujeme za skutočne náhodné a to má praktické využitie v generovaní náhodných čísel. Medzi využívané javy patria:

- Kvantové fluktuácie vákua [16].
- Rádioaktívny rozpad materiálu [17].
- Fotón prechádzajúci rozdeľovač lúčov [18].

Iné náhodné pozorovateľné javy

Náhodné dáta môžeme získať aj pozorovaním a spracovaním:

- Správania užívateľa (pohyb myšou, stáčanie klávesnice) [19].
- Prerušení systému(HDD I/0 operácie, sieťová aktivita) [20].

Kapitola 2

TRNG na mikrokontroléroch

Internet vecí nepochybne patrí medzi jednu z najrozvíjajúcejších sa technologických oblastí. V našom okolí neustále pribúda veľké množstvo vstavaných systémov, inteligentných spotrebičov či iných jednoduchších, často jednoúčelových, zariadení. Tie pri plnení svojich funkcii musia spĺňať bezpečnostné nároky a štandardy dnešného sveta. Od schopnosti systému generovať náhodné čísla sú závislé kryptografické aplikácie potrebné pre čenie dát a komunikácie takýchto systémov.

Jadrom týchto systémov sú z veľkej časti mikrokontroléry(MCU). Konštrukčne robustnejšie MCU môžu, okrem iných periférii, obsahovať aj špecializovaný RNG modul. Tie, ktoré ním nedisponujú môžu RNG implementovať za použitia iných periférií a súčastí. Väčšina MCU obsahuje rôzne analógové a digitálne senzory, ktoré je možné využiť ako zdroj entropie. Analógovo-digitálny prevodník(ADC) je bežnou súčasťou MCU [21].

2.1 STM32 RNG Periféria

STMicroelectronics patrí medzi dominantné spoločnosti na poli výrobcov MCU pre prevažne vnorené systémy. Ponúka širokú škálu mikrokontrolérov od jednoduchších 8-bitových až po komplexnejšie 32-bitové mikrokontroléry architektúry ARM, pre ktoré je dostupná široká škála periférií.

Jedným z týchto periférii dostupných pre niektoré rady mikrokontrólerov je aj modul RNG. Ten ponúka generovanie 32-bitových skutočne náhodných čísel vytvorených pomocou analógového zdroja entropie a využitia post-processingu. Štruktúra tohto modulu je zobrazená na 2.1. Tento modul bol otestovaný sadou štatistických testov NIST [22].

Zdroj entropie

Tento model využíva dva analógové zdroje, každý založený na troch kruhových oscilátoroch, na ktorých výstup je aplikovaná operácia XOR. Tieto analógové zdroje ponúkajú kontinuálny 2-bitový výstup, ktorý je zdigitalizovaný pomocou vzorkovania. K taktovaniu tohto procesu je využitý samostatný hodinový vstup `RNG Clock`, nezávislý od hodinového signálu`AHB Clock` výstupnej 32-bitovej zbernice `AHB` [23].

Post processing

Výstup zo zdroja entropie je náchylný k sklonu k jednej hodnote a preto musí byť spracovaný. V prvom kroku dochádza k rozdeleniu výstupu vzorkovacieho procesu na polovicu a následnému invertovaniu bitov jednej polovice. V prípade vyššej frekvencie výskytu bitov s niektorou z hodnôt je tento výkyv mierne odfiltrovaný. V druhej fáze dochádza k využitiu dvoch posuvných registrov s lineárnou spätnou väzbou(LFSR), ktoré následne produkuje 8-bitové reťazce.

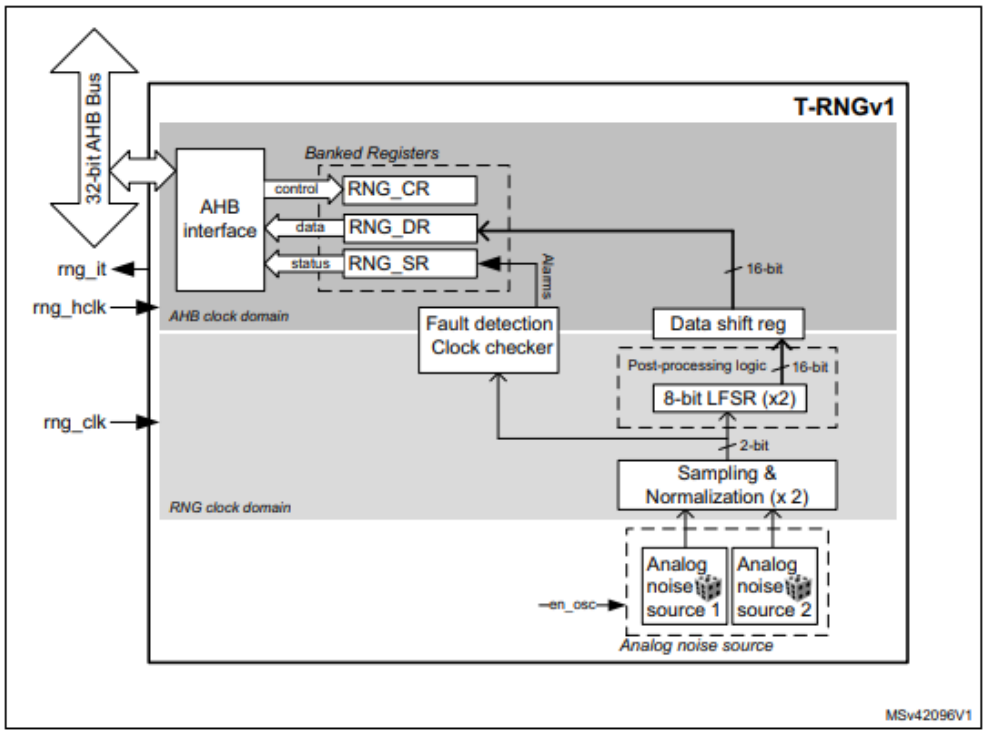
Výstupný buffer

RNG modul obsahuje dátový register o veľkosti 32 bitov. Jeho obsah tvoria dva 16-bitové výstupy z LFSR. Pre naplnenie registra 32-bitovým náhodným číslom je potrebné počkať 42 hodinových cyklov `RNG Clock`. Keď je vygenerované číslo dostupné, príznak „Data ready flag“ prejde do stavu 1. Po vyprázdnení registra je príznak opäť nastavený na hodnotu 0.

Kontrolné testy

RNG modul implementuje testy, tzv. „Health checks“, ktoré nepretržite kontrolujú stavy analógových zdrojov entropie. V prípade nevhodných vlastností výstupov niektorého zo zdrojov je nastavený chybový príznak „Seed error current status“ (SECS) na hodnotu 1. Za chybu zdroja entropie sa považuje, ak na výstup poskytne viac ako 64 bitov s konštantnou hodnotou alebo ak poskytne 32 po sebe idúcich dvojíc bitov v tvare „01“ alebo „10“.

Ďalší test kontroluje rozdiely medzi taktom využívaných hodín. Za chybu sa považuje ak je jeden cyklus `RNG Clock` kratší ako cyklus hodín zbernice `AHB Clock` delený číslom 16. V takomto prípade je nastavený príznak „Clock error current status“ (CECS). V prípade detekcie chyby je RNG modul schopný generovať prerušenie [23].



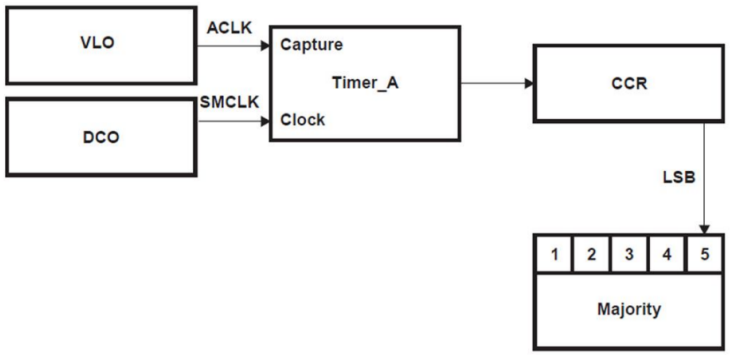
Obr. 2.1: Konštrukcia TRNG periférie na STM32 mikrokontroléroch [23]

2.2 Texas Instruments MSP430

Americká spoločnosť Texas Instruments(TI) je jedným z najväčších hráčov na trhu s polovodičovými súčiastkami. Medzi širokou škálou produktov nájdeme aj 16 a 32 bitové mikrokontroléry. Pre svoju univerzálnosť a cenovú dostupnosť patrí medzi najrozšírenejšie rodina 16 bitových nízkoenergetických mikrokontrolérov MSP430. Tá však nedisponuje špecializovaným hardwarovým komponentom pre generovanie náhodných čísel. Spoločnosť Texas Instruments pre túto rodinu mikrokontrolérov vydala správu [24], v ktorej opisuje možnosť využiť súčasti tohto mikrokontroléra pre generovanie náhodných čísel.

Konštrukcia

Pre generovanie náhodných čísel sa ako zdroj entropie využíva nestabilita hodinového signálu. Návrh využíva rýchlejší hodinový signál „Subsystem Master Clock“(SMCLK) a pomocný hodinový signál(ACLK). SMCLK je generovaný pomocou digitálne ovládaného oscilátora `DCO` so základnou frekvenciou kolísajúcou okolo 1 MHz. Pre signál ACLK je možné použiť dostupný nízko frekvenčný oscilátor `VLO` alebo kryštálový oscilátor `LFXT1` s frekvenciou 32kHz.



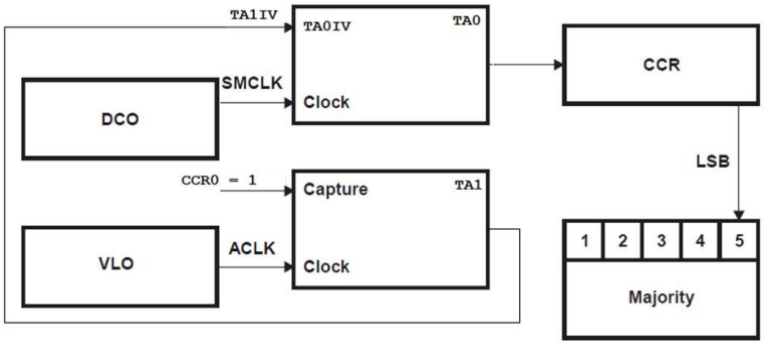
Obr. 2.2: Návrh implementácie TRNG na TI MSP430 mikrokontroléroch [25]

Tieto signály sú privedené do časovača `TIMER_A`. Ten ráta pulzy rýchlejšieho SMCLK a pri nábežnej hrane pomalších hodín ACLK vyvolá uloženie napočítanej hodnoty do registra `CCR` (Capture/Compare register). Keďže tieto hodinové signály pochádzajú z navzájom nezávislých zdrojov, ktoré podliehajú jemným výchylkám vo svojej frekvencii, bude toto číslo za každý cyklus pomalších hodín jemne odlišné. Pre generovanie čísel sa využije najmenej významný bit(LSB) hodnoty registra CCR indikujúci párnosť čísla, ktorý je následne ľavým posunom vložený do registra CPU. Po 16 opakovaniach tohto procesu dostávame 16 bitové náhodné číslo [24] [25].

Vylepšenia základného návrhu

Texas Instruments v prílohe spomínanej správy priložil software realizujúci návrh aj s miernymi vylepšeniami zameranými na zvýšení miery náhody tohto systému. Výsledný bit posunutý do registra CPU je výsledkom 5 cyklov. Každý z týchto cyklov poskytne LSB z registra CCR pričom najviac opakujúca sa hodnota je následne uložená do registra CPU. Pri každom posunutí LSB z registra `CCR` je register `BCSCTL1`(basic clock system control registers) pozmenený za účelom zmeny frekvencie DCO. Zároveň sa vezmú posledné dva bity z registra CPU s uloženou hodnotou a vykoná sa operácia XOR. Výstup tejto operácie je takisto použitý pre zmenu hodnoty registra `BCSCTL1` pre úpravu hodnoty deličky VLO. Tieto úpravy zvyšujú mieru náhody, avšak zmeny frekvencii hodinových signálov môžu spôsobiť problémy s inými bežiacimi procesmi [24].

Jednou z hlavných funkcií MSP430 je schopnosť prepnúť sa do režimu nízkej spotreby. V pôvodnom návrhu RNG sa tento režim nevyužíva. Implementácia v [25] sa zameriava na nízku spotrebu energie. Mikrokontrolér je počas počítania pulzov SMCLK v režime nízkej spotreby. Oproti pôvodnému návrhu využíva časovač aj pre pomalší hodinový signál ACLK. Ten je nastavený tak, aby vyvolal prerušenie po každom cykle ACLK. Toto prerušenie zastaví oba časovače a spôsobí ukončenie režimu nízkej spotreby. Následne je



Obr. 2.3: Návrh implementácie TRNG na TI MSP430 s nízkou spotrebou [25]

prečítaný počet cyklov SMCLK z registra prvého časovača a ďalej spracovaný podľa pôvodného návrhu. Po spracovaní je mikrokontrolér opäť privedený do režimu nízkej spotreby a celý proces sa opakuje.

2.3 TRNG s využitím ADC

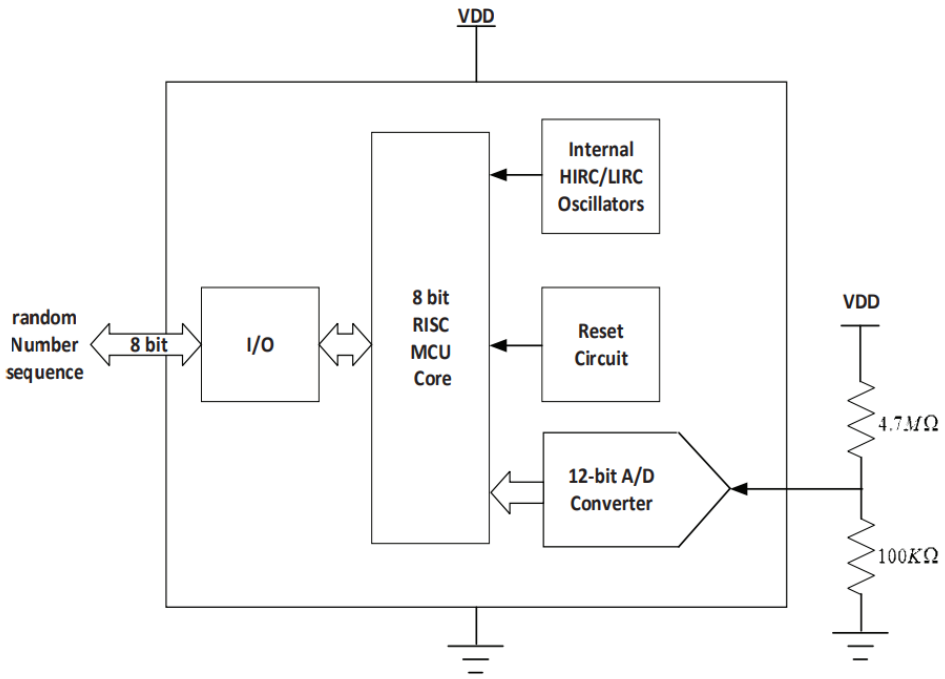
Ďalšou z možností ako generovať náhodné čísla na mikrokontroléroch je využitím analógovo-digitálneho prevodníka(ADC). Na 11. konferencii International Conference on Computer Science & Education (ICCSE) bol uverejnení príspevok [26] popisujúci konštrukciu TRNG za využitia ADC na 8 bitovom mikrokontroléri HT66F185.

Zdroj entropie

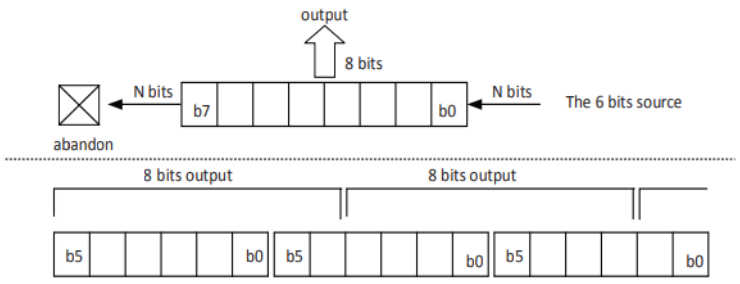
V návrhu 2.4 je do tohto prevodníka pripojený jednoduchý obvod s dvoma rezistormi. Ten je napájaný pomocou zdroja napätia VDD o veľkosti 5V. Odpor v tejto schéme generuje tepelný šum spôsobený Brownovým pohybom elektrónov. To má za následok náhodné výkyvy elektrického prúdu a napätia v obvode. Tento efekt sa zvýrazňuje so stúpajúcou teplotou. Šum vytvorený medzi dvoma rezistormi je privedený do ADC ako vstupný signál.

Postprocessing

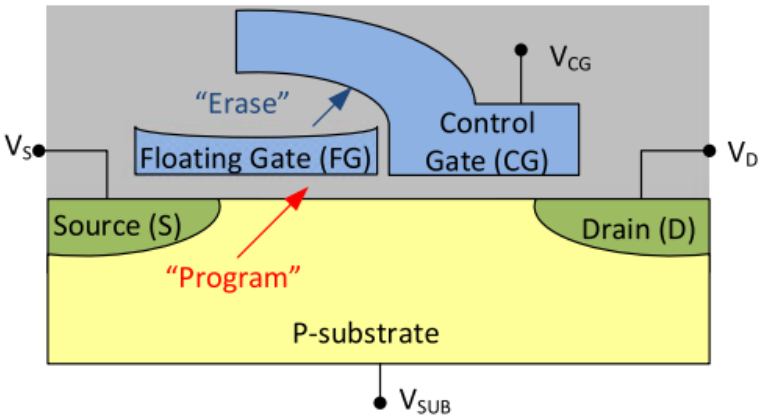
Po analýze konvertovaných hodnôt došli k autori k záveru, že pre generovanie náhodných čísel je vhodné použiť dolných 6 bitov výstupu 12 bitového prevodníka. Pre vytvorenie čísla je potrebných 8 bitov a teda je potrebný mechanizmus tvorby takýchto čísel, ktorý zaručí požadované štatistické vlastnosti. V príspevku sú zobrazené viaceré prístupy, pričom výstupy konštrukcie zobrazenej na 2.5 sú vyhodnotené ako dostatočne náhodné. Na rozdiel od ostatných konštrukcii dochádza k posunu zdrojových 6 bitov vždy o toľko miest aby sa nikdy nevyužívali rovnaké bity viackrát pre viacero výstupov.



Obr. 2.4: Návrh TRNG na mikrokontroléri za využitia ADC [26]



Obr. 2.5: Tvorba 8 bitových čísel z 6 bitového výstupu ADC [26]



Obr. 2.6: Bunka Flash pamäte [27]

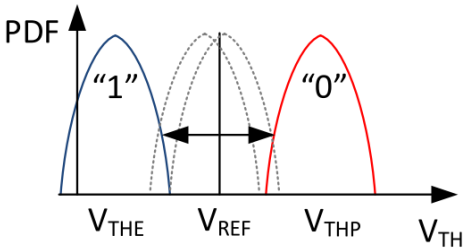
2.4 TRNG s využitím Flash pamäte

Článok [27] z žurnálu IEEE Transactions on Computers predstavuje možnosť generovania náhodných čísel na mikrokontroléroch využitím FLASH pamäte. Tento článok využíva bunky NOR FLASH pamäte, ktoré sa bežne používajú pre pamäte mikrokontrolérov. O využití NAND Flash buniek pre generovanie náhodných čísel pojednáva článok [28].

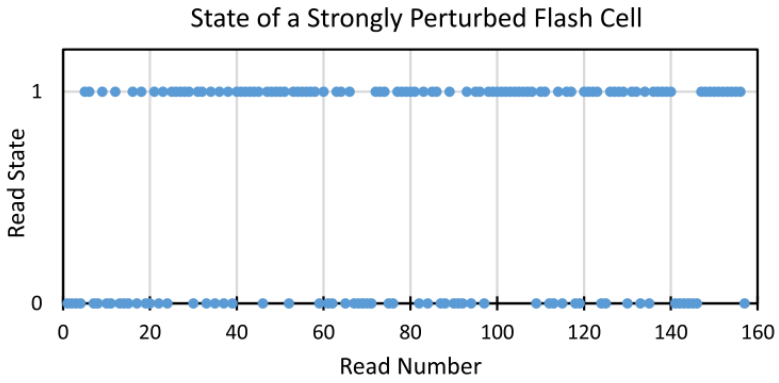
Bunky NOR Flash pamäte

Bunka Flash pamäte, schopná udržiavať informáciu o veľkosti jedného bitu, je zobrazená na 2.6. Predstavuje ju unipolárny tranzistor s rozdeleným hradlom na „Floating Gate“ (`FG`) a „Control Gate“ (`CG`). Táto bunka sa môže nachádzať v dvoch stavoch, a to „erased“, čo predstavuje hodnotu 1 alebo v stave „programmed“ čo predstavuje hodnotu 0.

Pre zmenu tohto stavu sa využívajú operácie „Program“ a „Erase“. Operácia „Program“ pomocou privedeného vysokého napätia na vývod `Source` nabije `FG` elektrickým nábojom čím zvýši prahové napätie. Operácia „Erase“ naopak privedie vysoké napätie na `CG` čím odstráni náboj v `FG` a zníži tak prahové napätie bunky. Pri vykonávaní operácie čítania sa privedie vhodné napätie na `CG` a `Drain` a skúma prahové napätie. Bunka v stave „erased“ prepúšťa prúd, jej hodnota je teda 1, naopak za stavu „programmed“ bunka neprepúšťa žiaden prúd a jej hodnota je prezentovaná ako 0.



Obr. 2.7: Rozdelenie prahového napätia [27]



Obr. 2.8: Hodnoty narušenej bunky po opakovanom čítaní [27]

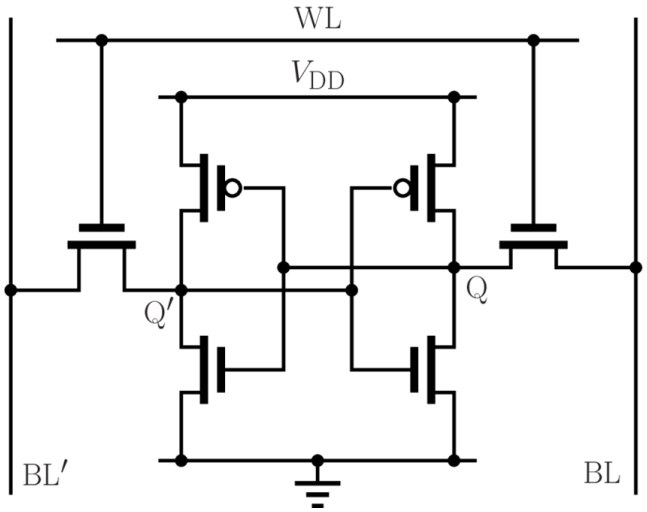
Narušenie stavu

Za bežných podmienok je bunka stabilizovaná v jednom z dvoch spomínaných stavov. Operácia čítania odhaľuje tento stav pomocou referenčného napätia s ktorým porovnáva získané prahové napätie bunky. Pre využitie buniek Flash pamäte je potrebné priviesť bunku do narušeného stavu, kedy jej prahové napätie je príliš blízke referenčnému napätiu. V takomto stave sú pre určenie hodnoty bunky rozhodujúce fluktuácie napätia spôsobené tepelným šumom.

Dosiahnutie tohto stavu je možné pomocou programu, ktorý vykoná operáciu „erase“ na požadovaný segment Flash pamäte a následne na tieto bunky použije operáciu „program“. Počas vykonávania tejto operácie je zavolaná inštrukcia pre okamžité ukončenie operácie. Po návrate z funkcie sú bunky v nepredvídateľnom stave. Na 2.8 sú znázornené vyčítané hodnoty jednej bunky po prevedení do narušeného stavu.

Generovanie čísel

Prvým krokom pre generovanie náhodných čísel je uvedenie dostatočne veľkej časti Flash pamäte do narušeného stavu. Čítanie hodnôt v FLASH pamäte je realizované po skupinách bitov, typicky po bytoch. Skupiny obsahujúce bunky sú opakovane vyčítané a pre každú narušenú bunku je vytvorený vek-



Obr. 2.9: Schéma bunky SRAM pamäte [31]

tor prečítaných hodnôt. Pre každý takýto vektor sa kontroluje počet zmien hodnoty danej bunky a ten je porovnaný s určeným minimálnym počtom zmien. Ak je tento limit splnený, na tieto vektory je aplikovaná operácia XOR s výstupným vektorom `V.RNG` určujúcim výslednú náhodnú postupnosť bitov.

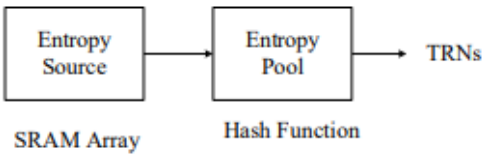
2.5 TRNG s využitím SRAM pamäte

Ďalším spôsobom ako implementovať TRNG na mikrokontroléroch bez potreby špecializovaného hardwaru je pomocou vstavanej pamäte SRAM. Stav buniek po zapnutí napájania závisí od viacerých faktorov a niektoré z buniek sú dostatočne nepredvídateľné pre potreby generovania náhodných čísel.

SRAM Bunky

Bunky pamäte SRAM pozostávajú z do kríža zapojených invertorov a dvoch prístupových tranzistorov Q a Q'. V prípade, že bunka nie je pod napätím sú stavy oboch tranzistorov $Q = 0$, $Q' = 0$. Po zapnutí napájania sa tento stav naruší a dôjde k prechodu do jedného zo stavov $Q = 0$, $Q' = 1$ kedy považujeme hodnotu bitu bunky za 0 alebo do stavu $Q = 1$, $Q' = 0$ čo považujeme za hodnotu bitu 1. Stav $Q = 1$, $Q' = 1$ je nedosiahnuteľný [29].

Stav do ktorého bunka po zapnutí napájania prejde závisí od prahových napätí invertorov. V prípade, že je rozdiel medzi prahovými napätiami výrazný, bunka je vychýlená a po zapnutí bude preferovať jeden zo stavov. Tento rozdiel však môže byť natoľko malý, že sa vzhľadom na fluktuácie zdrojového napätia, spôsobené prevažne tepelným šumom, stáva zanedbateľným a bunka sa správa nepredvídateľne, výsledný stav určujú aktuálne výkyvy napätia [30].



Obr. 2.10: Základná schéma TRNG pomocou SRAM [32]

Zdroj entropie

Nevychýlené bunky predstavujú generátory náhodných bitov. Tieto bunky sú typicky náhodne a rovnomerne rozmiestnené v rámci celého poľa buniek. Praktickým dôsledkom toho je ukrytie týchto jednotlivých zdrojov. Na druhú stranu je potrebné entropiu týchto buniek vyextrahovať z väčšieho bloku pamäte.

Veľkosť tohto bloku závisí od veľkosti požadovaného vygenerovaného čísla a od celkového množstva entropie v celej pamäti. Tú je potrebné pred realizovaním TRNG odmerať. Blok pamäte by mal obsahovať aspoň toľko entropie, koľko bitov má mať výsledné generované číslo [32]. Množstvo entropie v pamäti sa za rôznych, najmä tepelných, podmienok môže líšiť. Je teda vhodné využiť blok s dostatočne veľkou rezervou nadbytočnej entropie.

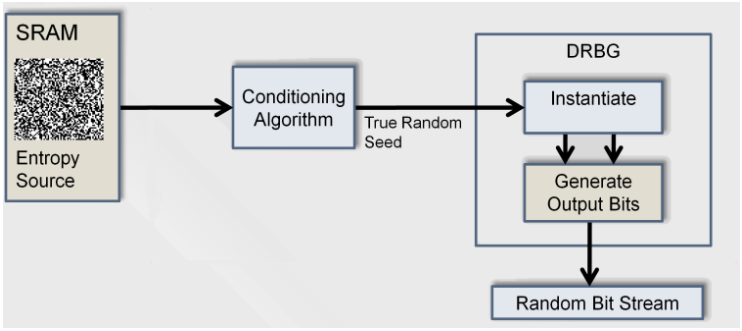
Postprocessing

Bloky pamäte s dostatočnou entropiou sú následne spracované. Za účelom extrakcie entropie sú bežne využívané hashovacie funkcie ako napríklad SHA-256 [3], QUARK[33], alebo aj šifrovací algoritmus AES v móde CBC-MAC [34]. Výstup týchto operácii môžeme použiť ako výstup takto konštruovaného TRNG.

Nevýhody a vylepšenia SRAM TRNG

Medzi najvýraznejšie nevýhody tohto riešenia patrí potreba odpájať SRAM pamäť od napätia pri každom generovaní náhodných čísel. Pri implementáciách na mikrokontroléri, ktoré to podporujú, sa využívajú nízko energetické režimy pri ktorých dochádza k odpojeniu SRAM pamäte od napájania. Prípadne je možné pre generovanie využiť externé pamäťové obvody SRAM [29]. Ďalšou využívanou konfiguráciou je využitie takto vygenerovaného čísla ako „seed“ pre PRNG implementovaný na mikrokontroléri [3].

Ďalšou nevýhodou takéhoto TRNG je, že zvyčajne v pamäti prevládajú vychýlené bunky a pre požadované množstvo entropie je teda potrebné relatívne veľký blok pamäte. Navyše je možné pozorovať, že množstvo vhodných buniek klesá postupným využívaním pamäte. V článku [30] je predstavený



Obr. 2.11: Návrh PRNG využívajúca entropiu SRAM pre seed[3]

spôsob zvýšenia výskytu nevychýlených buniek, a tým zvýšenie celkovej entropie, pomocou ožarovania buniek SRAM. Článok [32] sa zaoberá vplyvom starnutia a ukazuje možnosť kontrolovaného starnutia pre udržanie vysokého množstva entropie v pamäti.

Kapitola 3

Návrh TRNG na mikrokontroléri

V tejto kapitole popisujem návrh implementácie a jej realizáciu na mikrokontroléri .

3.1 Hardware

Zvolenou platformou je vývojová doska NUCLEO-F446RE obsahujúca mikrokontrolér STM32F446RE s jadrom ARM Cortex-M4. Mikrokontrolér obsahuje 128KB SRAM, ktorá v adresnom priestore začína na adrese 0x20000000.

Prípravok podporuje rôzne režimy napájania, medzi ktorými je aj mód „stand-by“, počas ktorého dochádza, okrem iného, k odpojeniu SRAM pamäte od napájania. To je možné využiť pre generovanie náhodných čísel bez potreby úplného vypnutia systému.

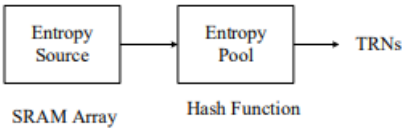
Prípravok ďalej obsahuje nezávislé hodiny Real-Time Clock (RTC), ktoré majú samostatný zdroj napájania a teda sa v móde „stand-by“ nevypínajú. Pomocou nich je možné nastaviť RTC WakeUpTimer, časovač, ktorý po nastavenom počte tiknutí $WakeUpCounter$ odošle budiaci signál a tým ukončí „stand-by“ mód [35]. Výpočet požadovanej hodnoty čítača:

$$WakeUpCounter = \frac{WakeUpTime}{TimeBaseClock}$$

kde $WakeUpTime$ je požadovaná doba než sa odošle signál a $TimeBaseClock$ je zvolená perióda hodín používaných pre časovač.

3.2 Software

Pre potreby práce s mikrokontrolérom som používal software od spoločnosti STMicroelectronics. Pre tvorbu programu navrhovaného TRNG ako aj pomocných programov som využil vývojové prostredie STM32CubeIDE a pre prístup k funkciám jednotlivých častí mikrokontroléru som využil



Obr. 3.1: Základná štruktúra TRNG využívajúceho SRAM [32]

Hardware Abstraction Layer(HAL) driver, ktorý ponúka API pre prácu s perifériami. Pre nahranie výsledného programu na prípravok a inšpekciu obsahu pamäte som využíval aplikáciu STM32CubeProgrammer. Pre komunikáciu s PC po sériovej linke som použil program PuTTY so zapnutým záznamom.

3.3 Návrh

Základný princíp návrhu je zobrazený na obrázku 3.1. Zdrojom entropie je stav statickej RAM pamäte po zapnutí napájania. Podľa množstva prítomnej entropie v pamäti je zvolený blok dostatočnej veľkosti, tak aby entropia tohto bloku bola aspoň taká veľká ako je veľkosť generovaného čísla, teda 256 bitov.

Vzhľadom na analýzu meraní v kapitole 4 je ako zdroj entropie zvolený blok o veľkosti 512B na adrese 0x20015800. Z neho je potrebné vyextrahovať číslo s maximálnou entropiou o veľkosti 256 bitov. K tomu je v tomto návrhu využitá hashovacia funkcia Secure Hash Algorithm 256 (SHA-256).

Program pre generovanie náhodných čísel

Mikrokontrolér obsahuje program, ktorý po zapnutí čaká na užívateľov pokyn pomocou užívateľského tlačidla. Stlačenie vyvolá proces generovania čísel. Počet vygenerovaných čísel po jednom stlačení je zadefinovaný v kóde programu. Proces generovania čísel sa skladá z nasledujúcich krokov:

- Uvedenie mikrokontroléra do „stand-by“ režimu na dobu 80ms.
- Prebudenie mikrokontroléra pomocou RTC časovača.
- Prečítanie 512B bloku dát z pamäte na adrese 0x20015800.
- Zahashovanie 512B bloku dát pomocou implementácie SHA-256 [36].
- Zaslanie 256 bitového čísla pomocou sériovej linky.

Tento proces sa opakuje, kým nie je vygenerované preddefinované množstvo čísel. Následne je uživateľské tlačidlo k dispozícii pre ďalšie generovanie.

Kapitola 4

Entropia SRAM pamäte

Pre vyhodnotenie množstva entropie v bunkách pamäti SRAM tohto prípravku je potrebné vyhodnotiť pravdepodobnosti počiatočnej hodnoty jednotlivých buniek. Tieto pravdepodobnosti aproximujem pomocou sérii meraní. Z nich je následné vypočítané dostupné množstvo entropie v SRAM pamäti.

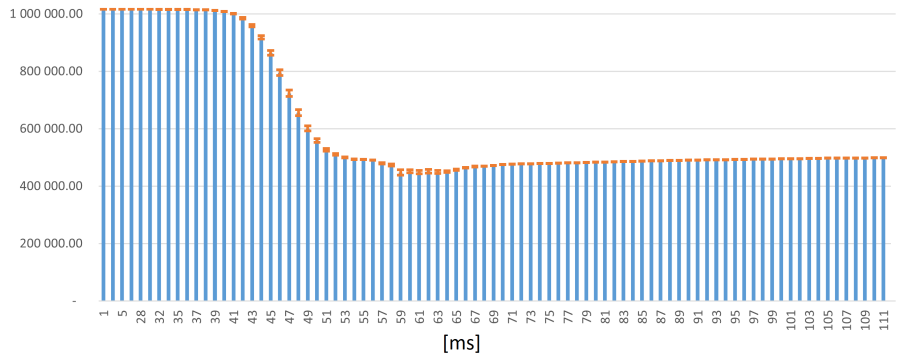
4.1 Obnovenie stavu SRAM buniek

Pred meraním stavu buniek po zapnutí napájania bolo potrebné zistiť čas potrebný pre SRAM pamäť na stratu informácie o predošlej hodnote. Tento čas som získal pomocou sérii meraní.

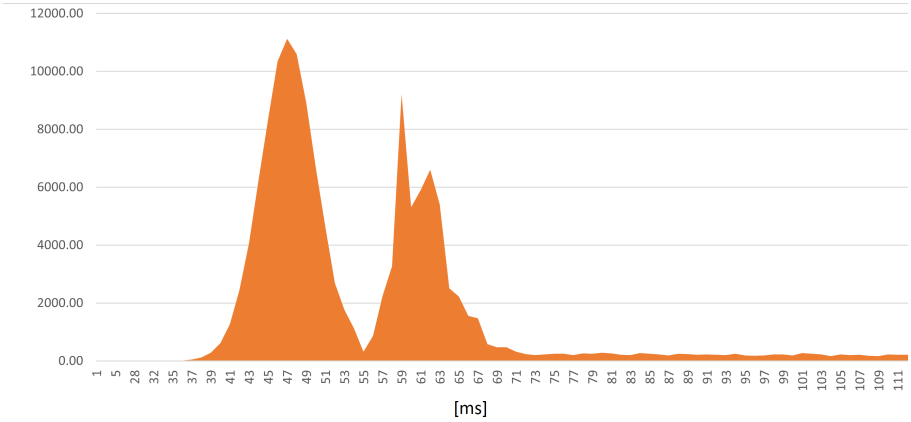
Mikrokontrolér som opakovane prenášal do stavu „stand-by“. Pred každým vypnutím napájania SRAM došlo k prepisu celej pamäte na hodnoty 0xFF, teda tak aby každá bunka bola v stave 1. Po každom prebudení z „stand-by“ módu pomocou časovača $WakeUpTimer$ som zaznamenal počet bitov v stave 1. Čas v móde „stand-by“ som pre každých 50 meraní zmenil od hodnôt 1ms, 2ms, 5ms, 26ms, 28ms a ďalej postupne o 1ms navýšenia až po hodnotu 120ms. Z nameraných dát som si vytvoril graf 4.1, ktorý zobrazuje priemerný počet buniek s hodnotou 1 z 50 meraní s daným časom stráveným v „stand-by“ móde.

Z grafu možno odčítať, že pri nízkom čase bez napätia bunky vedia udržať informáciu o svojom stave. Množstvo buniek s hodnotou 1 sa pre zvyšujúci interval bez napätia znižuje až sa ustáli na približne polovici počtu buniek, čo je očakávaný jav. Za pozornosť stoja aj smerodatné odchýlky meraní jednotlivých časov na grafe 4.2. K výkyvom smerodatnej odchýlky dochádza až dvakrát. Následne dochádza k stabilizácii nameraných hodnôt.

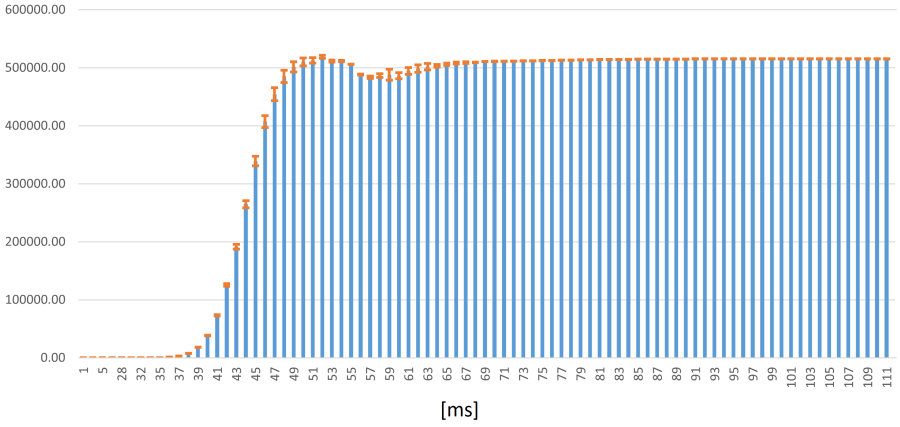
Merania som zopakoval s tým, že sa pred prechodom do „stand-by“ módu pamäť prepísala na 0x00. Výsledky týchto meraní sú zobrazené na grafoch 4.3 a 4.4. Po preskúmaní výsledkov meraní som určil čas 80ms ako dostatočný pre stratu informácie uloženej v SRAM bunke.



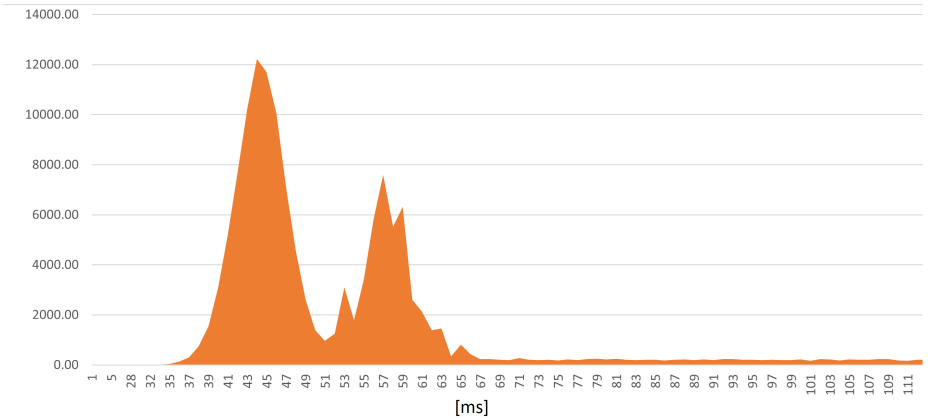
Obr. 4.1: Priemerný počet buniek s hodnotou 1 po prepise pamäti 0xFF pre daný čas strávený v standby móde



Obr. 4.2: Smerodatná odchýlka hodnôt meraní po prepise pamäti 0xFF pre daný čas strávený v standby móde



Obr. 4.3: Priemerný počet buniek s hodnotou 1 po prepise pamäti 0x00 pre daný čas strávený v standby móde



Obr. 4.4: Smerodatná odchýlka hodnôt meraní po prepise pamäti 0x00 pre daný čas strávený v standby móde

4.2 Odhad pravdepodobností hodnôt buniek SRAM

Pre aproximáciu pravdepodobností hodnôt jednotlivých buniek po zapnutí napájania som urobil 1024 meraní. Prípravok som opakovane prenášal do stavu „stand-by“ na dobu 80ms a po prebudení z tohto stavu som prečítal a poslal obsah celej pamäte SRAM do pripojeného počítača pomocou sériovej linky. Obsah pamäte sa po 128KB blokoch pripisoval do konečného súboru `fullsram1024.bin`. Na základe týchto dát som aproximoval pre každú bunku SRAM jej pravdepodobnosť nadobudnutia hodnoty 1 po zapnutí napájania ako priemernú hodnotu meraní danej bunky. Tieto hodnoty som uložil do súboru `means1024.csv`.

4.3 Výpočet entropie

Po získaní pravdepodobností pre bunky SRAM som vypočítal entropiu a minimálnu entropiu SRAM pamäte. Za predpokladu, že jednotlivé hodnoty buniek sú na sebe nezávislé [29] [30], vypočítam entropiu pamäte ako súčet entropií jednotlivých buniek. Pravdepodobnosť, že bunka nadobudne hodnotu 0 značíme $P(bit_0)$, pravdepodobnosť hodnoty 1 značíme $P(bit_1)$.

Pre výpočet entropie binárneho zdroja, jedného bitu, využijem vzorec:

$$H^{bit} = -(P(bit_0)log(P(bit_0)) + P(bit_1)log(P(bit_1)))$$

a pre výpočet minimálnej entropie :

$$H^{bit}_{min} = -log(max(P(bit_0), P(bit_1)).$$

```
Calculation of entropy and minEntropy of sram cells.
Load sample means from file.
1048576 values loaded.
Calculating...
Entropy of SRAM cells : 194324.95763155015
Average entropy per bit : 0.18532272112994208
Min entropy of SRAM cells : 109743.09630991264
Average Min entropy per bit: 0.10465917235366119
```

Obr. 4.5: Výstup zo skriptu entropy.ipynb, ktorý vypočíta celkovú a minimálnu entropiu.

Pre výpočet celkovej entropie som využil skript `entropy.ipynb`. Z výstupu tohto skriptu 4.5 môžeme odčítať, že priemerné množstvo entropie na bit je približne 0,185 a priemerné množstvo minimálnej entropie je 0,105 na bit.

Rozloženie entropie v pamäti

Okrem množstva entropie je potrebné aj preskúmať jej rozloženie naprieč bunkami pamäte. Pomocou skriptu `entropyBlocks.ipynb` som rozdelil pamäť na bloky o veľkostiach:

- 8192 x 128b
- 1024 x 1024b
- 256 x 4096b
- 128 x 8192b

Tento skript ďalej vypočítal pre každé rozdelenie pamäte entropiu a minimálnu entropiu daných blokov a uložil tieto hodnoty do súborov `blocks`[n]`b.csv`, kde n je veľkosť bloku.

	Entropia			Min Entropia			Entropia na bit		
Blok	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
128b	10.7	37.7	23.8	4.8	24.6	13.4	0.083	0.294	0.186
1024b	151.2	222.5	190.4	79.9	132.2	107.5	0.147	0.217	0.186
4096b	694.1	834.3	761.9	388.8	478.7	430.3	0.169	0.203	0.186
8192b	1400	1613	1524	791.6	926.4	860.6	0.170	0.196	0.186

Tabuľka 4.1: Prehľad entropie po blokoch pamäte

Vyhodnotenie entropie v blokoch pamäti

Vzhľadom na priemernú minimálnu entropiu na bit 0.10 je pre generovanie 256 bitového čísla je potrebných približne 2560 bitov (320B). Zvolil som veľkosť bloku 4096b(512B). K preskúmaniu hodnôt entropie v blokoch som si pomocou histogramu znázornil rozloženie hodnôt entropie.

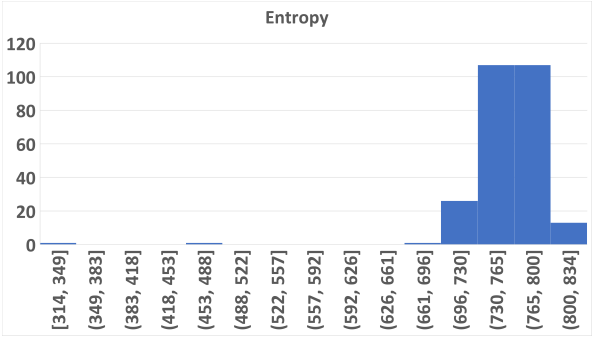
Z histogramu 4.6 vidíme, že distribúcia hodnôt entropie jednotlivých blokov javí známky normálneho rozdelenia, s výnimkou dvoch blokov ktoré majú entropiu výrazne nižšiu. Graf 4.7 zobrazuje množstvo entropie v jednotlivých blokoch od začiatku pamäte. Z neho je možné odvodiť, že hodnoty entropií sú rovnomerne rozdelené naprieč pamäťou a že dva bloky s nízkou entropiou sú prvý a posledný. Po odobratí prvého a posledného bloku histogram 4.8 naznačuje, že množstvo entropie vo zvyšnej časti pamäte zodpovedá normálnemu rozdeleniu.

Rozloženie entropie a minimálnej entropie v blokoch o všetkých veľkostiach je možné porovnať na 4.9 až 4.12. Tieto grafy potvrdzujú, že v pamäti nie je miesto s výrazne nižším množstvom entropie. Taktiež je z nich možné odčítať, že množstvo minimálnej entropie sa pohybuje približne v polovičných hodnotách celkovej entropie. Množstvo minimálnej entropie je rovnako rovnomerne rozložené naprieč pamäťou.

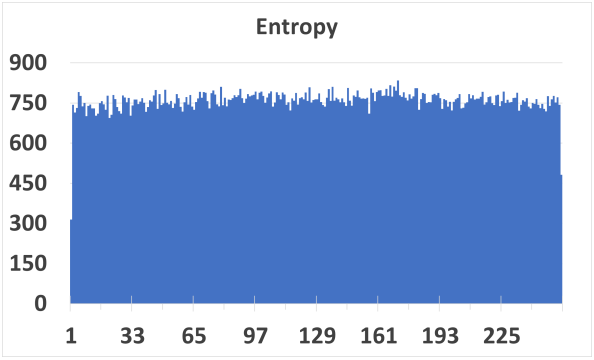
Histogramy hodnôt množstva entropie pre jednotlivé veľkosti blokov sú takisto k dispozícií na 4.13 až 4.16. Na týchto grafoch môžeme pozorovať, že počty hodnôt entropii aj minimálnych entropii sa približujú normálnemu rozdeleniu pre všetky veľkosti blokov.

Žiadna časť pamäte sa výrazne neodlišuje od ostatných. Rozpätie a priemerné hodnoty pre jednotlivé veľkosti blokov sú zobrazené v tabuľke 4.1.

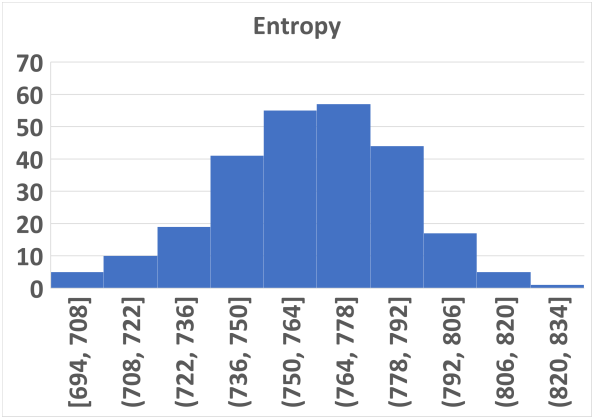
Ďalej som pomocou skriptu `meansVisual.ipynb` vytvoril vizualizácie blokov pamäti 4.17 a 4.18. Na začiatku prvého bloku pamäte a na konci posledného bloku pamäte vidíme oblasť s jednoznačne určeným stavom v každom meraní. Je to oblasť, ktorú pre svoje potreby využíva bežiaci program. Z dát je možné určiť bloky s najvyššou, respektíve najnižšou entropiou. Sú to bloky na adrese `0x20015400`, respektíve `0x20002800`.



Obr. 4.6: Histogram hodnôt entropie v 4096 bitových blokoch.

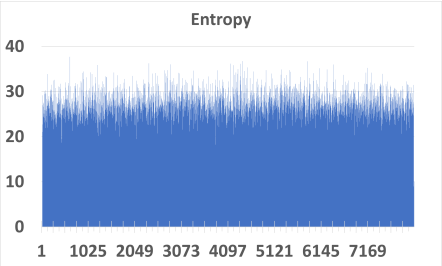


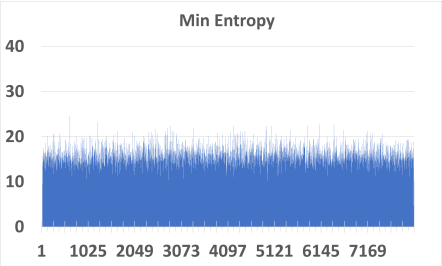
Obr. 4.7: Množstvo entropie na í-ty 4096 bitový blok.



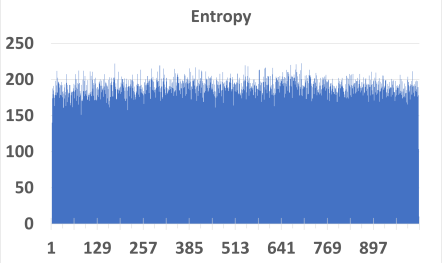
Obr. 4.8: Rozloženie hodnôt entropie v 4096 bitových blokoch bez okrajových blokov.

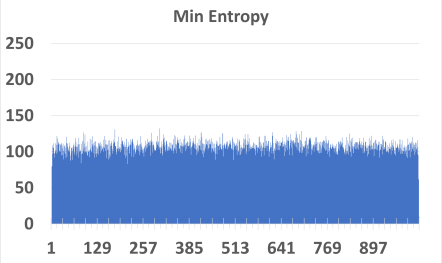
Rozloženie množstva entropie v blokoch



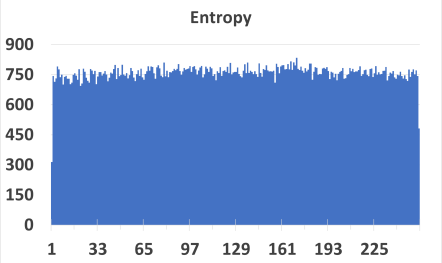


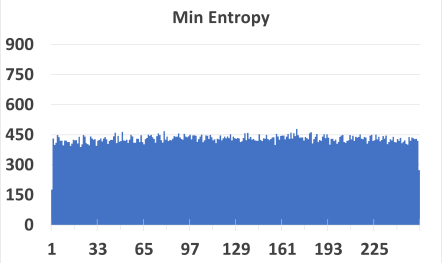
Obr. 4.9: Rozloženie entropie a minimálnej entropie v blokoch, 128b



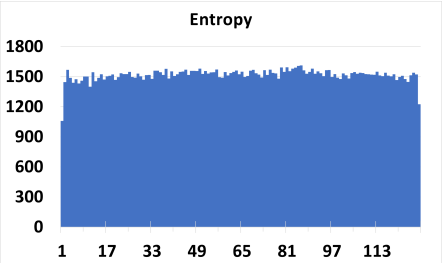


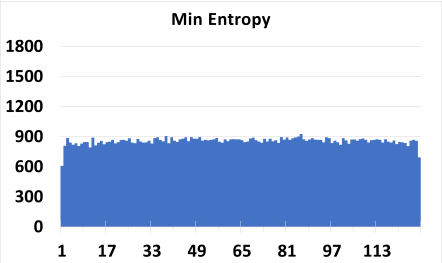
Obr. 4.10: Rozloženie entropie a minimálnej entropie v blokoch, 1024b





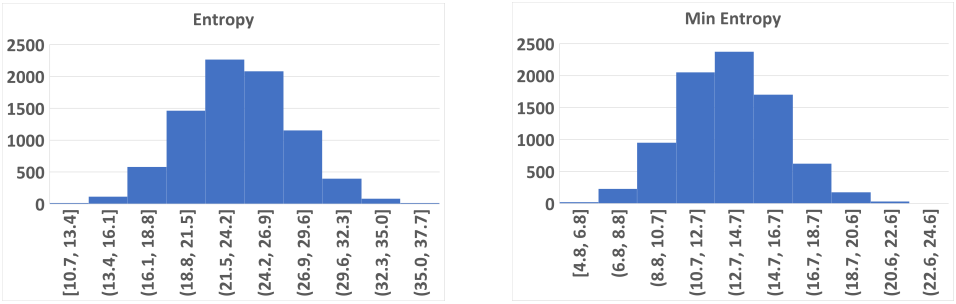
Obr. 4.11: Rozloženie entropie a minimálnej entropie v blokoch, 4096b



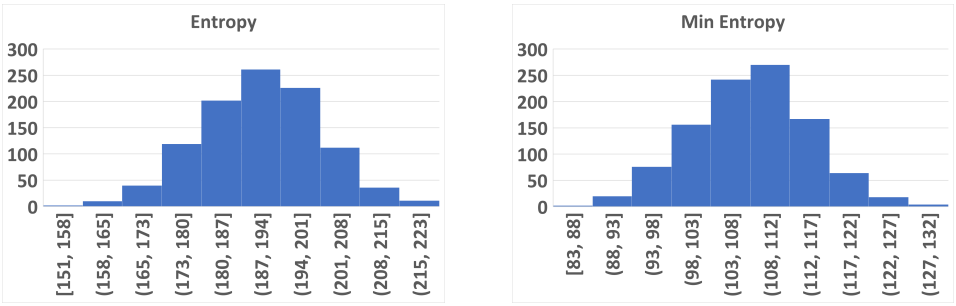


Obr. 4.12: Rozloženie entropie a minimálnej entropie v blokoch, 8192b

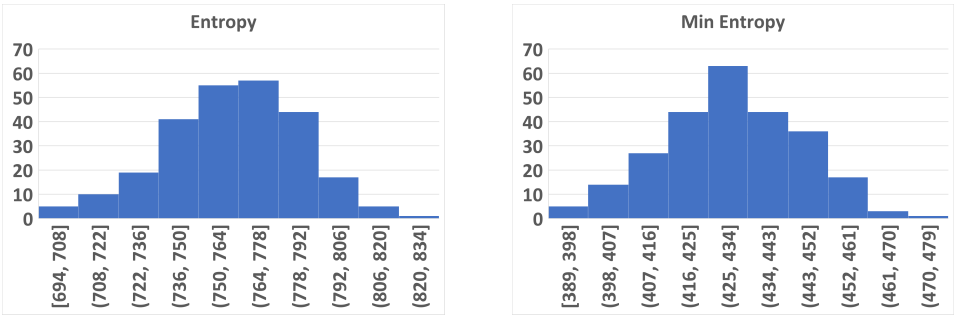
Histogramy množstva entropie v blokoch



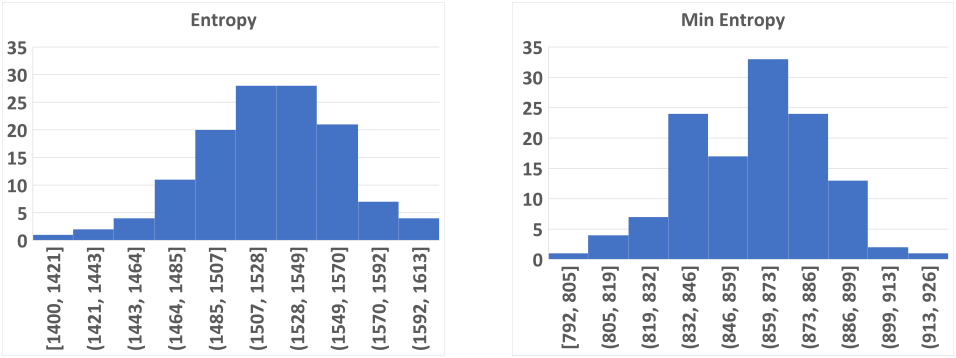
Obr. 4.13: Rozloženie entropie a minimálnej entropie v bloku, 128b



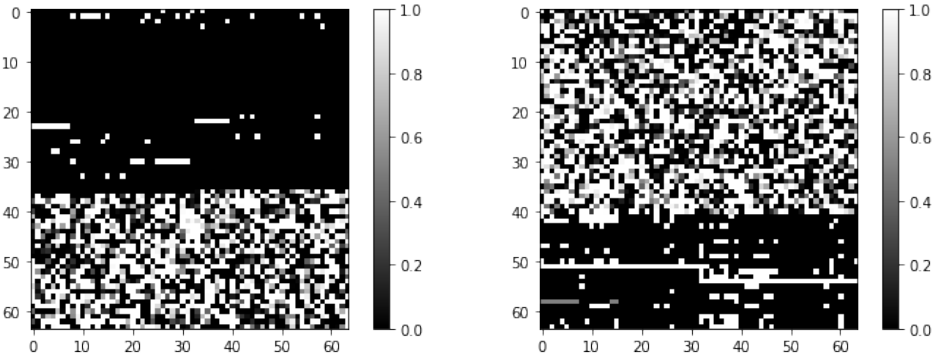
Obr. 4.14: Rozloženie entropie a minimálnej entropie v bloku, 1024b



Obr. 4.15: Rozloženie entropie a minimálnej entropie v bloku, 4096b

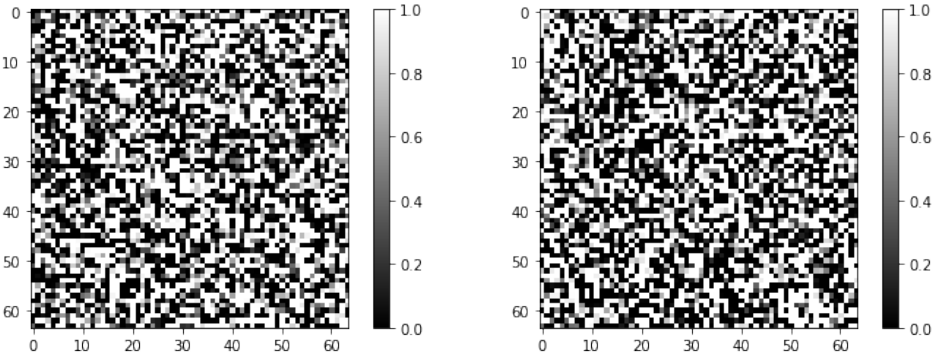


Obr. 4.16: Rozloženie entropie a minimálnej entropie v bloku, 8192b



(a) 0x20000000 (b) 0x2001FE00

Obr. 4.17: Rozloženie pravdepodobností pamäťových buniek na začiatku a konci pamäte.



(a) 0x20015400 (b) 0x20002800

Obr. 4.18: Rozloženie pravdepodobností pamäťových buniek s najvyššou a najnižšou entropiou

Kapitola 5

Štatistické testy

Po implementovaní návrhu na mikrokontroléri je žiadúce otestovať štatistické vlastnosti generovaných čísel. K tomu je využitá sada štatistických testov NIST Test Suite [4]. Táto sada obsahuje 15 testov pre vyhodnotenie štatistických vlastností bitových postupností. Tieto testy stanovujú nulovú hypotézu $H0$, že testovaná sekvencia je náhodná. Tieto testy vracajú P-hodnotu, pomocou ktorej môžeme túto hypotézu zamietnuť alebo nezamietnuť porovnaním s hladinou významnosti daných testov.

5.1 Predstavenie testov

V tejto kapitole sú stručne predstavené jednotlivé testy a opísané požiadavky na vstupné dáta a parametre.

5.1.1 Frequency Monobit Test

Tento test kontroluje frekvenciu výskytov hodnôt 0 a 1 v celej sekvencii čísel. Hodnoty 0 a 1 by sa mali vyskytovať približne rovnako často, podiel ich počtu výskytov by sa mal blížiť k $\frac{1}{2}$.

Vstupné dáta a parametre

Tento test vyžaduje jeden parameter n, ktorý značí dĺžku bitovej sekvencie. Odporúčaná hodnota je $n \geq 100$.

5.1.2 Block Frequency Test

Tento test kontroluje počty výskytov hodnôt 0 a 1 v M bitových blokoch vstupnej sekvencie čísel. Hodnoty 0 a 1 by sa mali vyskytovať približne rovnako často v rámci blokov, podiel ich počtu výskytov by sa mal blížiť k $\frac{1}{2}$.

Vstupné dáta a parametre

Tento test vyžaduje parametre n, ktorý značí dĺžku bitovej sekvencie a M veľkosť jedného bloku. Hodnoty M a n je odporúčané voliť tak aby spĺňali $M > 0.01 \times n$, pričom $M \geq 20$ a $n \geq 100$.

5.1.3 Cumulative Sums Test

Účelom tohto testu je vypočítať kumulatívny súčet hodnôt v sekvencii pričom je za výskyt nulového bitu zo súčtu odčítaná jednotka a za jednotkový bit pričítaná. Účelom testu je zistiť, či nie je pre niektoré čiastkové sekvencie kumulatívny súčet príliš vysoký alebo nízky. Pre náhodné sekvencie by sa odchýlky hodnôt takéhoto kumulatívneho súčtu mali pohybovať okolo nuly.

Vstupné dáta a parametre

Tento test vyžaduje jeden parameter n, ktorý značí dĺžku bitovej sekvencie. Odporúčaná hodnota je $n \geq 100$.

5.1.4 Runs Test

Tento test spočíta počet neprerušených postupností bitov s rovnakou hodnotou o rôznych dĺžkach v zdrojovej sekvencii. Následne skontroluje či tieto počty zodpovedajú očakávaným hodnotám náhodnej sekvencie. Tento test takto kontroluje frekvenciu striedania hodnôt bitov.

Vstupné dáta a parametre

Tento test vyžaduje jeden parameter n, ktorý značí dĺžku bitovej sekvencie. Odporúčaná hodnota je $n \geq 100$.

5.1.5 Longest Run of Ones Test

Tento test spočíta dĺžky najdlhších postupností jednotkových bitov v rámci M bitových blokov zdrojovej sekvencie. Následne skontroluje či tieto počty zodpovedajú očakávaným hodnotám pre náhodne sekvencie. Vzhľadom na to, že bit môže nadobudnúť len dve rôzne hodnoty, odchýlenie od očakávaných hodnôt pre postupností jednotkových bitov znamená aj odchýlenie postupností nulových bitov. Test pre dĺžky sekvencii nulových bitov teda nie je potrebný.

Vstupné dáta a parametre

Tento test vyžaduje parametre n, ktorý značí dĺžku bitovej sekvencie a M veľkosť jedného bloku. Hodnota M je odvodená od hodnoty n, dĺžky vstupnej sekvencie. Minimálna dĺžka sekvencie je 128 bitov. Hodnota M je odvodená nasledovne: $M = 8$ pre $n \geq 128$, $M = 128$ pre $n \geq 6272$ a $M = 10^4$ pre $n \geq 750000$.

5.1.6 Rank Test

Vstupná sekvencia je týmto testom rozdelená do matíc o rozmeroch $M \times Q$. Týmto maticiam sú vypočítané ich hodnosti. Účelom tohto testu je skontrolovať lineárnu závislosť medzi podreťazcami pôvodnej sekvencie.

Vstupné dáta a parametre

Parameter M značí počet riadkov každej z matíc, Q značí počet stĺpcov. Oba tieto parametre majú v zdrojovom kóde testovacej sady prednastavenú hodnotu rovnú 32. Minimálna dĺžka sekvencie n je určená vzťahom

$$n \geq 38 \times M \times Q.$$

To zaručí vytvorenie minimálne 38 matíc. Pre prednastavené hodnoty M, Q je teda minimálna dĺžka sekvencie 38 912 bitov.

5.1.7 Discrete Fourier Transform Test

Tento test sa zameriava na detekciu periodických vlastností skúmanej bitovej sekvencie pomocou aplikovania Diskrétnej Fourierovej transformácie na zdrojovej sekvencii. Zámerom je zistiť, či sa počet vrcholov prekračujúcich stanovenú prahovú hodnotu 95% výrazne líši od 5%.

Vstupné dáta a parametre

Tento test vyžaduje jeden parameter n, ktorý značí dĺžku bitovej sekvencie. Odporúčaná hodnota je $n \geq 1000$.

5.1.8 Non-overlapping Template Matching Test

Tento test skúma výskyt špecifických neperiodických vzorov v generovanej bitovej sekvencii. K tomu je využité m bitové okno, ktoré je umiestnené na začiatok sekvencie. V prípade zhody s m bitovým vzorom je začiatok okna posunutý na prvý bit za koncom tohto vzoru. V prípade, že sa obsah okna nezhoduje so žiadnym vzorom, je toto okno posunuté o jeden bit.

Vstupné dáta a parametre

Zdrojová sekvencia je rozdelená na N blokov o dĺžke M. Vstupným parametrom m je dĺžka okna, resp. dĺžka hľadaných vzorových podreťazcov. Odporúčané hodnoty sú 9 a 10. Hodnota N je nastavená v zdrojovom kóde na hodnotu 8. Pre validné výsledky je potrebné zabezpečiť aby $M > 0.01 \times n$.

5.1.9 Overlapping Template Matching Test

Tento test skúma výskyt špecifických neperiodických vzorov v generovanej bitovej sekvencii. Rovnako ako v predošlom teste je využité m bitové okno, ktoré je umiestnené na začiatok sekvencie. V prípade, že sa obsah okna nezhoduje so žiadnym vzorom, je toto okno posunuté o jeden bit. V prípade zhody s m bitovým vzorom je v tomto teste okno opäť posunuté o jeden bit.

Vstupné dáta a parametre

Vstupným parametrom m je dĺžka okna, resp. dĺžka hľadaných podreťazcov. Odporúčané hodnoty sú 9 a 10. Minimálna dĺžka sekvencie je $n \geq 10^6$.

5.1.10 Universal Statistical Test

Tento test sa zameriava na počet bitov medzi opakujúci vzormi v skúmanej sekvencii. Cieľom tohto testu je detegovať možnosť výraznej bezstratovej kompresie. Sekvencie s touto vlastnosťou nie sú považované za náhodné.

Vstupné dáta a parametre

Minimálna dĺžka sekvencie je $n \geq 387840$.

5.1.11 Approximate Entropy Test

Účelom tohto testu je porovnať frekvenciu prekrývajúcich sa vzorov dvoch po sebe nasledujúcich dĺžok (m a $m + 1$) s očakávaným výsledkom pre náhodnú postupnosť.

Vstupné dáta a parametre

Parameter m určuje veľkosť blokov. Zvolená veľkosť by mala spĺňať:

$$m < \lfloor \log_2 n \rfloor - 5$$

5.1.12 Random Excursions Test

Tento test pracuje s kumulatívnym súčtom vytvoreným rovnakým spôsobom ako v teste 5.1.3. Hodnoty kumulatívneho súčtu sú interpretované ako náhodná prechádzka. Tá je rozdelená do cyklov. Každý cyklus začína a končí keď je hodnota kumulatívneho súčtu rovná 0. Test skúma počet návštev stavov (-4,-3,-2,-1,1,2,3,4) v rámci cyklu a porovnáva to s predpokladanými počtami pre náhodnú sekvenciu.

Vstupné dáta a parametre

Minimálna dĺžka sekvencie je $n \geq 10^6$.

5.1.13 Random Excursions Variant Test

Tento test pracuje s kumulatívnym súčtom vytvoreným rovnakým spôsobom ako v teste 5.1.3. Hodnoty kumulatívneho súčtu sú interpretované ako náhodná prechádzka. Test skúma počet prechodov do konkrétneho stavu (-9,-8,-7,...,-2,-1,1,2,..,8,9) a porovnáva to s predpokladanými počtami pre náhodnú sekvenciu.

Vstupné dáta a parametre

Minimálna dĺžka sekvencie je $n \geq 10^6$.

5.1.14 Serial Test

Účelom tohto testu je porovnať frekvenciu všetkých možných prekrývajúcich sa vzorov. Test porovná počet výskytov 2^m m bitových vzorov s predpokladanými počtami pre náhodnú sekvenciu.

Vstupné dáta a parametre

Parameter m určuje veľkosť blokov. Zvolená veľkosť by mala spĺňať:

$$m < \lfloor \log_2 n \rfloor - 2$$

5.1.15 Linear Complexity Test

Účelom tohto testu je určiť či je sekvencia dostatočné komplexná aby mohla byť považovaná za náhodnú. Pomocou Berlekamp-Massey algoritmu je určený LFSR schopný generovať danú sekvenciu. Test skúma veľkosť týchto LFSR. Pre náhodné čísla sú typické dlhé LFSR.

Vstupné dáta a parametre

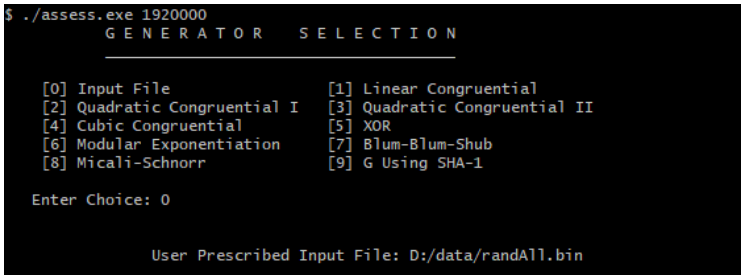
Minimálna dĺžka sekvencie je $n \geq 10^6$. Tá je rozdelená do blokov. Pre veľkosť bloku platí $500 \leq M \leq 5000$.

5.2 Vyhodnotenie testov

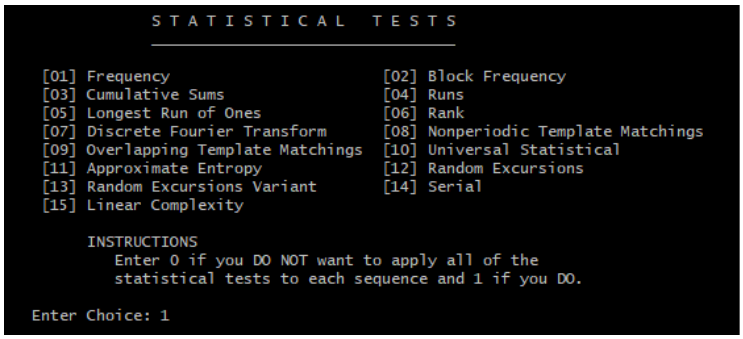
Výstupom každého testu sekvencie je P-hodnota. Tá sa porovnáva s hodnotou hladiny významnosti daného testu. Jej hodnota je predvolená na $\alpha = 0.01$. Ak je P-hodnota testu $< \alpha$, testovanú sekvenciu vyhodnocujeme ako nie náhodnú, sekvencia neprešla testom. V opačnom prípade sekvencia testom prešla a považujeme ju za náhodnú.

Pre validné výsledky je potrebné každým testom otestovať väčšie množstvo sekvencii. Pre každý test vznikne množina P-hodnôt obsahujúca jednu P-hodnotu za každú sekvenciu. Testovacia sada vyhodnotí pomer sekvencií spĺňajúcich testy. Aby bolo možné dáta považovať za náhodné je potrebné aby tento pomer patril do konfidenčného intervalu testu.

Ďalšou požiadavkou na výsledky testov jednotlivých sekvencií je rovnomerné rozloženie výsledných P-hodnôt. Pre ilustráciu je možné rozdeliť interval $[0; 1]$ na 10 menších intervalov. P-hodnoty testov sekvencii sú následne rozdelené to týchto intervalov a podľa počtu sekvencii v každom intervale môžeme odhadnúť rozloženie hodnôt. Testovacia sada navyše pre vyhodnotenie uniformity vykoná „Goodness-of-Fit Distributional Test“. Výsledkom testu je „P-hodnota P-hodnôt“. Pre splnenie požiadavky rovnomerného rozdelenia má byť táto P-hodnota ≥ 0.0001.Pre zmysluplné výsledky je vyžadované testovanie aspoň 55 sekvencii.



Obr. 5.1: Zvolenie vstupného súboru.



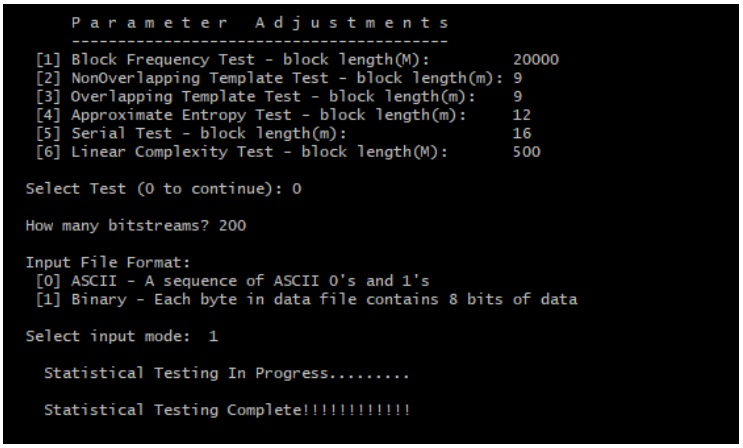
Obr. 5.2: Zvolenie všetkych testov.

Výstupom testovania je súbor `finalAnalysisReport.txt`. V ňom sa nachádza tabuľka ktorej riadky predstavujú jednotlivé testy. Stĺpec `PROPORTION` udáva pomer sekvencii ktoré splnili daný test. Počty P-hodnôt v jednotlivých intervaloch sú zobrazené v stĺpcoch `C1 - C10`. „P-hodnota P-hodnôt“ zobrazená v stĺpci `P-VALUE` a indikuje rovnomerné rozloženie P-hodnôt v intervaloch.

5.3 Testovanie vygenerovaných dát

Testy sú vykonané na binárnom súbore `randAll.bin` ktorý obsahuje 1 500 000 vygenerovaných 256 bitových čísel. To tvorí postupnosť 384 000 000 bitov. Tie sú pre účely testovania rozdelené na 200 bitstreamov o veľkosti 1 920 000 bitov.

Po skompilovaní testovacej sady je pomocou príkazu `./assess.exe 1920000` spustený program a určená dĺžka sekvencie. Na 5.1 je zobrazené zapnutie programu a zvolenie súboru `randAll.bin` ako zdroja náhodných čísel. Na 5.2 je zobrazená ponuka testov a zvolenie všetkých testov. Na 5.3 sú zobrazené zvolené parametre tak aby spĺňali podmienky opísané vyššie. Počet testovaných sekvencii je 200 a zvolený binárny režim vstupu. Testy sú následne vykonané.



Obr. 5.3: Zvolenie parametrov.

5.3.1 Analýza výsledkov testov

Výstupný súbor `finalAnalysisReport.txt` zobrazený na 5.4, dostupný v plnej forme v priloženom médiu, ukazuje, že pre všetky typy testov dostatočné množstvo sekvencii splnilo dané testy. Výsledne P-hodnoty týchto testov sú taktiež rovnomerne rozložené do intervalov. Výsledky štatistických testov NIST teda indikujú, že implementovaný TRNG generuje čísla spĺňajúce požiadavky na náhodné čísla.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <D:/data/randAll.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 14  19  22  25  24  21  14  21  21  19  0.729870    199/200     Frequency
 19  28  17  20  19  18  22  19  12  26  0.419021    198/200     BlockFrequency
 16  18  22  26  25  22  19  18  13  21  0.616305    199/200     CumulativeSums
 18  16  22  27  19  19  22  11  24  22  0.437274    199/200     CumulativeSums
 17  16  16  23  14  22  33  23  19  17  0.125927    196/200     Runs
 30  25  10  12  14  17  24  26  22  20  0.021262    199/200     LongestRun
 17  18  20  24  29  14  21  17  15  25  0.326749    199/200     Rank
 20  23  18  28  20  19  22  20  11  19  0.514124    199/200     FFT
 18  22  15  15  22  26  19  14  21  28  0.350485    199/200     NonOverlappingTemplate
                                           ...
 15  16  21  19  16  24  20  27  23  19  0.668321    195/200     NonOverlappingTemplate
 29  17  13  16  27  18  14  23  19  24  0.141256    197/200     OverlappingTemplate
 22  20  22  19  27  16  25  17  15  17  0.626709    196/200     Universal
 22  23  26  20  13  14  19  21  21  21  0.647530    197/200     ApproximateEntropy
 19   8  11  12  14  13   9  24  13  16  0.059271    138/139     RandomExcursions
                                           ...
 10  19  16  17  19   9  10  11  15  13  0.297391    139/139     RandomExcursions
 10  16  11  13  12  16   9  21  18  13  0.308675    139/139     RandomExcursionsVariant
                                           ...
 11  18  14  12  14  17  14  12  18   9  0.637119    138/139     RandomExcursionsVariant
 22  23  23  27  15  20   9  21  20  20  0.282626    196/200     Serial
 16  27  23  22  15  18  17  26  19  17  0.524101    199/200     Serial
 17  19  17  21  21  15  19  26  27  18  0.657933    196/200     LinearComplexity
```

Obr. 5.4: Ukážka výstupného súboru `finalAnalysisReport.txt`

Záver

Cieľom práce bolo zvoliť typ a navrhnúť generátor skutočne náhodných čísel na cieľovej platforme. Zvolil som TRNG založený na stave SRAM pamäte po zapnutí napájania a vytvoril návrh jeho implementácie na mikrokontroléri STM32F446 s procesorom architektúry ARM Cortex-M4 na vývojovej doske NUCLEO-F446RE.

Tento TRNG som následne implementoval. Pomocou sérii meraní stavu pamäte po zapnutí napájania som odhadol množstvo entropie SRAM buniek a využil som primerane veľkú časť pamäte ako zdroj entropie. Pomocou hashovacej funkcie som z tohto bloku vyextrahoval náhodné číslo o veľkosti 256 bitov, ktoré je vhodné pre použitie v kryptografických aplikáciách.

Výsledné generované čísla som otestoval pomocou sady štatistických testov od NIST a dospel k záveru, že implementovaný TRNG je vhodný pre použitie na generovanie náhodných čísel. Výstupom práce je program schopný generovať skutočne náhodné čísla na mikrokontroléri.

Vzhľadom na potrebu vypnutia napájania SRAM pamäte pre generovanie náhodného čísla by v budúcnosti bolo vhodné tento generátor rozšíriť tak, aby využíval číslo vygenerované TRNG po zapnutí napájania ako seed pre vhodne zvolený deterministický PRNG.

Literatúra

[1] Haahr, M.: Introduction to Randomness and Random Numbers [online]. 2009. Dostupné z: `http://random.org/randomness/`

[2] Shannon, C. E.: Prediction and entropy of printed English. *The Bell System Technical Journal*, 1951.

[3] van der Leest, V.; van der Sluis, E.; Schrijen, G.-J.; aj.: *Efficient Implementation of True Random Number Generator Based on SRAM PUFs*. Cryptography and Security: From Theory to Applications, Berlin, Heidelberg: Springer Berlin Heidelberg, ISBN 0302-9743, s. 300–318.

[4] National Institute of Standards and Technology: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technická Zpráva SP800-22, Rev. 1a, U.S. Department of Commerce, Washington, D.C., 2010, doi:10.6028/NIST.SP.800-22r1a.

[5] Hlaváč, J.; Kodýtek, F.: Prednáška z predmetu Hardwarová Bezpečnosť. 2019.

[6] Torii, N.; Kokubo, H.; Yamamoto, D.; aj.: ASIC implementation of random number generators using SR latches and its evaluation. *EURASIP Journal on Multimedia and Information Security*, ročník 2016, č. 1, 2016: s. 1–12.

[7] Kumar, K. M.; Sunitha, N.: Hybrid cryptographically secure pseudo-random bit generator. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, 2016, s. 296–301, doi: 10.1109/IC3I.2016.7917978.

[8] Williams, B.; Hiromoto, R. E.; Carlson, A.: A Design for a Cryptographically Secure Pseudo Random Number Generator. In *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced*

Computing Systems: Technology and Applications (IDAACS), ročník 2, 2019, s. 864–869, doi:10.1109/IDAACS.2019.8924431.

[9] Marghescu, A.; Svasta, P.; Simion, E.: High speed and secure variable probability Pseudo/True Random Number Generator using FPGA. In *2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2015, s. 323–328, doi:10.1109/SIITME.2015.7342347.

[10] Blum, L.; Blum, M.; Shub, M.: A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing*, ročník 15, č. 2, 1986, doi:10.1137/0215025. Dostupné z: https://doi.org/10.1137/0215025

[11] Wang, Y.-H.; Zhang, H.-G.; Shen, Z.-D.; aj.: Thermal noise random number generator based on SHA-2 (512). In *2005 International Conference on Machine Learning and Cybernetics*, ročník 7, 2005, s. 3970–3974 Vol. 7, doi:10.1109/ICMLC.2005.1527631.

[12] Yamanashi, Y.; Yoshikawa, N.: Superconductive Random Number Generator Using Thermal Noises in SFQ Circuits. *IEEE Transactions on Applied Superconductivity*, ročník 19, č. 3, 2009: s. 630–633, doi:10.1109/TASC.2009.2019294.

[13] Guo, C.; Zhou, Y.; Liu, H.; aj.: On the jitter and entropy of the oscillator-based random source. In *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2015, s. 1–5, doi:10.1109/ICCCNT.2015.7395169.

[14] Robson, S.: A Ring Oscillator Based Truly Random Number Generator. 2013.

[15] Choi, S.; Shin, Y.; Yoo, H.: Analysis of Ring-Oscillator-based True Random Number Generator on FPGAs. In *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, 2021, s. 1–3, doi:10.1109/ICEIC51217.2021.9369714.

[16] Symul, T.; Assad, S. M.; Lam, P. K.: Real time demonstration of high bitrate quantum random number generation with coherent laser light. *Applied Physics Letters*, ročník 98, č. 23, jun 2011: str. 231103, doi:10.1063/1.3597793. Dostupné z: https://doi.org/10.1063%2F1.3597793

[17] Haw, J. Y.; Assad, S. M.; Lance, A. M.; aj.: Maximization of Extractable Randomness in a Quantum Random-Number Generator. *Phys. Rev. Applied*, ročník 3, May 2015: str. 054004, doi:10.1103/PhysRevApplied.3.054004. Dostupné z: https://link.aps.org/doi/10.1103/PhysRevApplied.3.054004

[18] Herrero-Collantes, M.; Garcia-Escartin, J. C.: Quantum random number generators. *Reviews of modern physics*, ročník 89, č. 1, 2017;2016;.

[19] Suciu, A.; Marton, K.; Antal, Z.: Data Flow Entropy Collector. In *2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2008, s. 445–448, doi:10.1109/SYNASC.2008.25.

[20] Castejon-Amenedo, J.; Mccue, R.; Simov, B. H.: Extracting Randomness from External Interrupts. In *In The IASTED International Conference on Communication, Network, and Information Security*, 2003, s. 141–146.

[21] Souaki, G.; Halim, K.: Random number generation based on MCU sources for IoT application. In *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, 2017, s. 1–6, doi: 10.1109/ATSIP.2017.8075524.

[22] STMicroelectronics: STM32 microcontroller random number generation validation using the NIST statistical test suite. Technická Zpráva AN4230, 2020. Dostupné z: `https://www.st.com/resource/en/application_note/dm00073853-stm32-microcontroller-random-number-generation-validation-using-the-nist-statistical-test-suite-stmicroelectronics.pdf`

[23] STMicroelectronics: *Reference manual, STM32F410 advanced Arm®-based 32-bit MCUs*. 2018. Dostupné z: `https://www.st.com/resource/en/reference_manual/rm0401-stm32f410-advanced-armbased-32bit-mcus-stmicroelectronics.pdf`

[24] Texas Instruments : Random Number Generation Using MSP430™ MCUs. Technická Zpráva SLAA338A, Revised, Texas Instruments Incorporated, 2018.

[25] Fujdiak, R.; Mlynek, P.; Misurec, J.; aj.: Cryptography in ultra-low power Microcontroller MSP430. *International Journal of Engineering Trends and Technology*, ročník 6, 12 2013.

[26] Jinming, L.; Jian, M.; Peiguo, L.: Design and implement of a MCU based random number generater. In *2016 11th International Conference on Computer Science Education (ICCSE)*, 2016, s. 945–948, doi: 10.1109/ICCSE.2016.7581710.

[27] Poudel, P.; Ray, B.; Milenkovic, A.: Microcontroller TRNGs Using Perturbed States of NOR Flash Memory Cells. *IEEE Transactions on Computers*, ročník 68, č. 2, 2019: s. 307–313, doi:10.1109/TC.2018.2866459.

[28] Ray, B.; Milenković, A.: True Random Number Generation Using Read Noise of Flash Memory Cells. *IEEE Transactions on Electron Devices*, ročník 65, č. 3, 2018: s. 963–969, doi:10.1109/TED.2018.2792436.

[29] Holcomb, D. E.; Burleson, W. P.; Fu, K.: Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers*, ročník 58, č. 9, 2009: s. 1198–1210, doi: 10.1109/TC.2008.212.

[30] Zhang, X.; Jiang, C.; Dai, G.; aj.: Improved Performance of SRAM-Based True Random Number Generator by Leveraging Irradiation Exposure. *Sensors*, ročník 20, č. 21, 2020, ISSN 1424-8220, doi:10.3390/s20216132. Dostupné z: https://www.mdpi.com/1424-8220/20/21/6132

[31] Bai, Y.; Alawad, M.; DeMara, R. F.; aj.: Optimally Fortifying Logic Reliability through Criticality Ranking. *Electronics*, ročník 4, č. 1, 2015: s. 150–172, ISSN 2079-9292, doi:10.3390/electronics4010150. Dostupné z: https://www.mdpi.com/2079-9292/4/1/150

[32] Wang, W.; Guin, U.; Singh, A.: Aging-Resilient SRAM-based True Random Number Generator for Lightweight Devices. *Journal of Electronic Testing*, ročník 36, 06 2020, doi:10.1007/s10836-020-05881-6.

[33] Li, D.; Lu, Z.; Zou, X.; aj.: PUFKEY: A High-Security and High-Throughput Hardware True Random Number Generator for Sensor Networks. *Sensors*, ročník 15, č. 10, 2015: s. 26251–26266, ISSN 1424-8220, doi:10.3390/s151026251. Dostupné z: https://www.mdpi.com/1424-8220/15/10/26251

[34] Mechalas, J. P.: Intel® Digital Random Number Generator (DRNG) Software Implementation Guide [online]. 2018. Dostupné z: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide

[35] STMicroelectronics: *STM32F446xC/E - Arm® Cortex®-M4 32-bit MCU+FPU, 225 DMIPS, up to 512 KB Flash/128+4 KB RAM, USB OTG HS/FS, seventeen TIMs, three ADCs and twenty communication interfaces*. 2021. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f446re.pdf

[36] Conte, B.: crypto-algorithms - sha256 [Source code]. 2012. Dostupné z: https://github.com/B-Con/crypto-algorithms

Dodatok A

Zoznam použitých skratiek

ADC Analog-Digital Convertor

API Application programming interface

CSPRNG Cryptographically-Secure Pseudorandom Number Generator

LFSR Linear Feedback Shift Register

HAL Hardware abstraction layer

NIST National Institute of Standards and Technology

PRNG Pseudorandom Number Generator

RAM Random Access Memory

RNG Random Number Generator

RTC Real-Time Clock

SRAM Static Random Access Memory

TLS Transport Layer Security

TRNG True Random Number Generator

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `data`
  - `blocks128b.csv` ..... entropie 128b blokov pamäte
  - `blocks1024b.csv` ..... entropie 1024b blokov pamäte
  - `blocks4096b.csv` ..... entropie 4096b blokov pamäte
  - `blocks8192b.csv` ..... entropie 8192b blokov pamäte
  - `finalAnalysisReport.txt` ..... Výsledky testov NIST
  - `fullsram1024.bin` ..... 1024 kópii SRAM pamäte
  - `means1024.csv` ..... priemery hodnôt buniek SRAM z meraní
- `scripts`
  - `entropy.ipynb` ..... výpočet entropie SRAM pamäte
  - `entropyBlocks.ipynb` ..... vyhodnotenie entropie v blokoch pamäte
  - `probabilities.ipynb` ..... výpočet priemerných hodnôt
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `FullSramRead` ..... Program pre zber dát z celej SRAM
    - `ImplTRNG` ..... Program implementujúci TRNG
    - `IntervalEval` ..... Program pre odhad dĺžky stand-by intervalu
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text`
  - `BP_Jantosovic_Daniel_2022.pdf` ..... text práce vo formáte PDF