FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Implementace TRNG na ARM Cortex-M0
Student:	Tomáš Zach
Vedoucí:	Ing. Filip Kodýtek
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Nastudujte problematiku generátorů skutečně náhodných čísel (TRNG) se zaměřením na TRNG vhodné pro mikrokontroléry. Vyberte cílovou platformu, jejíž součástí je procesor s jádrem ARM Cortex-M0. Implementujte vhodnou variantu TRNG za použití prostředků zvolené platformy. Proveďte vyhodnocení vámi implementovaného řešení a diskutujte své výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Implementace TRNG na ARM Cortex-M0

Tomáš Zach

Katedra počítačových systémů
Vedoucí práce: Ing. Filip Kodýtek

14. května 2018

Poděkování

Především děkuji svému vedoucímu práce Ing. Filipu Kodýtkovi za možnost zabývat se zajímavým tématem a též za vstřícný přístup a čas, který mi věnoval při psaní této práce. Dále děkuji své rodině za podporu v průběhu celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2018 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2018 Tomáš Zach. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Zach, Tomáš. *Implementace TRNG na ARM Cortex-M0.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá generátory skutečně náhodných čísel - zkráceně TRNG (True Random Number Generator). Nejdříve je provedena literární rešerše týkající se problematiky TRNG se zaměřením na mikrokontroléry. Dále je zde představena vlastní implementace TRNG na mikrokontroléru s procesorem ARM-CORTEX-M0. Na závěr je provedeno vyhodnocení měření dané implementace pomocí sady statistických testů.

Klíčová slova Generátor skutečně náhodných čísel, TRNG, mikrokontrolér, ARM-CORTEX-M0, clock jitter, NIST

Abstract

This work deals with true random number generators (TRNG). At first, a literature research concerning TRNG with a focus on microcontrollers is done. Then own implementation of TRNG using microcontroller with ARM-CORTEX-M0 processor is introduced. Finally measurements of the proposed design are evaluated with a set of statistical tests.

Keywords True Random Number Generator, TRNG, microcontroller, ARM-CORTEX-M0, clock jitter, NIST

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Motivace

Chytrá elektronická zařízení jsou již nedílnou součástí běžného života. Ať už se jedná o mobil, notebook nebo televizi. Stále více podobných vynálezů nabízí nové možnosti použití, proto jsou vybavena operačním systémem či podobným softwarem a umožňují připojení k internetu. To s sebou nese bezpečnostní rizika. Proto je potřeba zařízení chránit před útočníky. Téměř každý zabezpečený protokol stojí na základu náhodných čísel a generátoru náhodných čísel.

Cíle práce

Cílem rešeršní části této práce je seznámení se s problematikou generátorů náhodných čísel a jejich klasifikací s důrazem na TRNG vhodné pro mikrokontroléry. Dále pak prozkoumání existujících typů konstrukcí TRNG.

Praktická část této práce se věnuje návrhu a konkrétní implementaci TRNG na mikrokontroléru s procesorem ARM Cortex-M0. Provedení měření a statistickému vyhodnocení dané implementace.

Kapitola 1

Generátory náhodných čísel

Tato kapitola se věnuje představení pojmu generátory náhodných čísel, jejich klasifikaci.

1.1 Generátory náhodných čísel

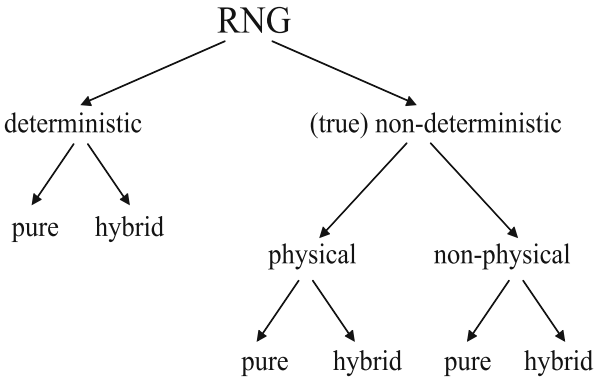
Problematika náhodných generátorů RNG (Random Number Generator) je velmi diskutované a stále aktuální téma. Náhodná čísla mají široké využití ve statistice, simulacích, počítačových hrách a dalších oblastech [1].

Náhodná čísla jsou nedílnou součástí moderní kryptografie. Používají se hlavně k vytváření klíčů v šifrovacích algoritmech jako RSA, Diffie-Hellman nebo AES. Nacházejí využití při generování vyplňujících bitů, které slouží k rozšíření krátkých zpráv na pevný počet bitů bez narušení použité šifry. Z tohoto důvodu jsou RNG součástí mnoha produktů v oblasti IT bezpečnosti a kladou se na ně striktní požadavky.

Generátor náhodných bitů je zařízení nebo algoritmus, jehož výstupem je posloupnost statisticky nezávislých a objektivních binárních číslic. Ten může být použit ke generování náhodných čísel [2].

Především musí vytvářet data použitelná pro kryptografické aplikace. Pokud generovaná data nejsou opravdu náhodná a jistou formou se dají předpovídat, útočník může snadno uhodnout klíč šifry z omezeného počtu možností. Není vyloučeno, že se útočník pokusí aktivně napadnout RNG, aby určil, nebo dokonce manipuloval s generovanými daty. Proto by RNG měl poskytovat schopnost odolat těmto druhům útoků. Dalším uvažovaným požadavkem je rychlost produkce dat pro některé časově náročné aplikace. Špatně navržený RNG může oslabit bezpečnost celého systému.

V roce 2010 došlo k prolomení bezpečnosti Sony Playstation 3. Konzole ověřovala svoje soubory pomocí podpisu. Jeho vytvoření vyžadovalo náhodné číslo. Chybou generátoru se pro každý podpis používala konstantní hodnota. Útočníci tuto slabinu odhalili a díky tomu mohli podepisovat vlastní soubory.



Obrázek 1.1: Klasifikace RNG[1]

Což jim umožnilo nahrát do konzole změněný operační systém a pouštět nelegální software.

1.2 Klasifikace generátorů náhodných čísel

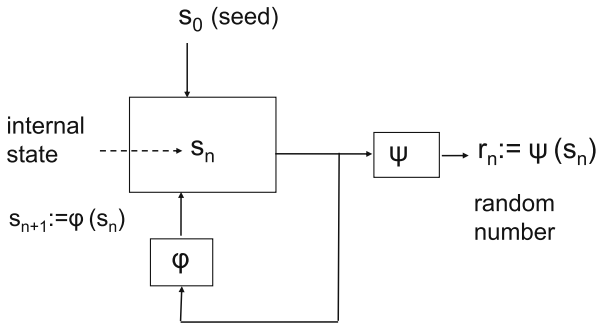
I když existují různá hlediska klasifikace RNG, podle schématu 1.1 rozdělujeme RNG do dvou skupin. První skupina zahrnuje pseudonáhodné generátory čísel PRNG (Pseudo Random Number Generator) nazývané některými autory deterministické RNG. Na vstup dostanou počáteční hodnotu tzv. seed a z ní algoritmicky generují pseudonáhodná čísla.

Druhou skupinu tvoří generátory opravdu náhodných čísel TRNG (True Random Number Generator), které se rozlišují na fyzické a nefyzické. Zdrojem náhodnosti fyzických TRNG jsou jevy elektrických obvodů. Např. šum ze Zenerovy diody, teplotní šum v polovodičích, volně bežící oscilátory apod. Nebo fyzikální jevy jako záření při radioaktivním rozpadu, či jiné kvantové procesy. Nefyzické TRNG využívají náhodné události jako vyhledávací čas pevného disku, obsah paměti RAM, interakce uživatele.

1.3 PRNG

Metody produkující náhodná čísla pomocí deterministického algoritmu nazýváme pseudonáhodnými generátory čísel [1]. Deterministický nazýváme algoritmus, který pro stejné vstupy má stejný průběh. Jinými slovy vstup jeho běh jednoznačně určí. Obrázek 1.2 ilustruje strukturu PRNG. Na začátku PRNG přijme na vstupu řetězec bitů nazývaný *seed*, který určí první vnitřní stav generátoru s_0 a jeho běh. Funkce Ψ na základě vnitřního stavu vypočte náhodnou hodnotu r_n. Potom se z aktuálního vnitřního stavu s_n vypočte nový

[1] Převzato z [3]



Obrázek 1.2: Obecná struktura PRNG[2]

vnitřní stav s_{n+1} pomocí funkce φ a celý cyklus se opakuje. Ačkoliv algoritmicky generovaná čísla nemůžou být opravdu náhodná a s určitou periodou se opakují, má tato skupina generátorů i své výhody. Mohou být mnohem rychlejší a jejich výsledky jsou opakovatelné. Například můžeme provést stejnou simulaci když známe *seed*.

1.3.1 Kryptograficky bezpečné PRNG

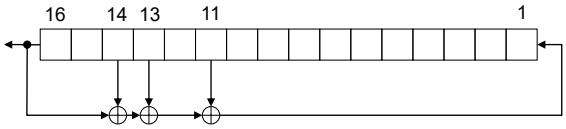
Zajímavou podskupinou jsou *kryptograficky bezpečné* PRNG - CSPRNG (Cryptographically Secure Pseudo-random Number Generator). Pro ně platí, že splňují následující vlastnosti:

- Musí projít statistickými testy náhodnosti.
- Next-bit test: Pokud známe prvních k bitů náhodné sekvence, neexistuje algoritmus, který v polynomiálním čase předpoví (k+1) bit s pravděpodobností větší než 1/2.
- Odhalení stavu: V případě že je odhalen vnitřní stav, není možné zpětně zrekonstruovat generovanou sekvenci do tohoto okamžiku.
- Odolnost proti zpětné rekonstrukci: Pokud generátor při běhu využívá další zdroj entropie, znalost aktuálního vnitřního stavu nestačí k předpovězení vnitřních stavů v dalších iteracích.

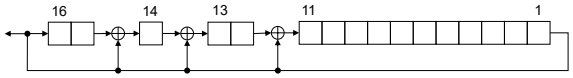
1.3.2 Zástupci PRNG

V této podsekci uvádíme pro úplnost některé typy PRNG.

[2]Převzato z [3]



Obrázek 1.3: Fibonacciho LFSR s vnější zpětnou vazbou[3]



Obrázek 1.4: Galoův LFSR s vnitřní zpětnou vazbou[4]

Lineární kongruenční generátor

Lineární kongruenční generátor LCG je jedním z nejznámějších a široce používaných PRNG zvlaště v simulačních aplikacích. Definuje se pomocí rekurentního předpisu:

$$X_{n+1} = (aX_n + c)\ mod\ m$$

Kde X je posloupnost pseudonáhodných čísel a X_0 je počáteční *seed*. Kvalita generátoru závisí na volbě násobku a, inkrementu c a modulu m, což jsou parametry LCG. Pro kryptografii se nehodí, pokud známe několik hodnot X_i, můžeme postupně odvodit jeho parametry jak popisuje [4].

Posuvný registr s lineární zpětnou vazbou

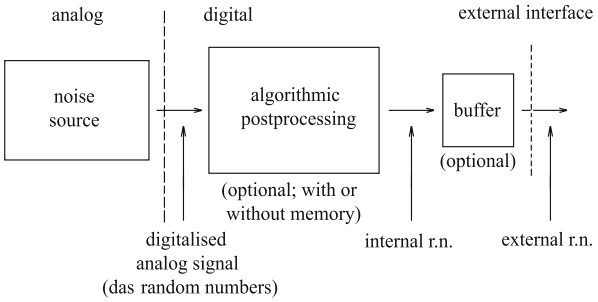
Tento typ se používá ke generování sekvence binárních bitů. Vstupní bit je výsledek lineární funkce dvou nebo více bitů. Díky snadné hardwarové implementaci se hodně používá. Opět není vhodný pro kryptografii, hodnoty můžeme předpovídat pomocí Berlekamp-Massey algoritmu [5].

V této práci se PRNG již více nezabýváme a zvídavého čtenáře odkazujeme na příslušnou literaturu.

1.4 TRNG

Jak zmiňuje [7], TRNG je zařízení, které využívá fyzikální procesy ke generování náhodné posloupnosti bitů. Obrázek 1.5 zobrazuje stavbu TRNG. Základem je zdroj šumu vznikající v elektrických obvodech (např. nestabilita napětí na polovodičových součástkách), nebo způsobený fyzikálními jevy (např. radioaktivní rozpad). Zdroj šumu vytváří v čase nepřetržité analogové signály, které jsou v jisté fázi opakovaně převáděny do digitálních binárních

[3] Převzato z [6]
[4] Převzato z [6]



Obrázek 1.5: Obecná struktura TRNG[5]

hodnot. Ve schématu se nazývají *das random numbers*. Tato čísla můžou být následně algoritmicky zpracována na vnitřní náhodná čísla, aby se snížily jejich potenciální slabiny. Zpracování může probíhat bez paměti, ale to záleží na konkrétním případě. Navíc silný zdroj šumu nutně nevyžaduje následné zpracování. Nakonec při vnějším požadavku jsou interní náhodná čísla předána na výstup.

1.4.1 Stavební bloky TRNG

Ačkoliv existuje mnoho typů TRNG, nejpopulárnější a nejužitečnější se běžně skládají z následujících tří komponent:

Zdroj entropie

Četné konstrukce TRNG jak bylo řečeno v literatuře sbírají náhodnost z fyzikálních procesů jako teplotní nebo výstřelový šum v obvodech, jitter, nestabilita signálu v obvodech, Brownův pohyb, atmosférický šum, nebo dokonce radioaktivní rozpad. Zdroj entropie je pravděpodobně nejdůležitější z komponent, protože určuje výslednou entropii. Na druhou stranu je jasné že zdroje jako atmosférický šum a radioaktivní rozpad jsou požitelné pro omezené aplikace nebo online distribuované služby [8]. Navíc některé zdroje mají sklony, které by měly být eliminovány v krocích sběru nebo následném zpracování. Kvantifikace celkové entropie a jejich statistických vlastností je další úkol konstrukce. Jiným problémem jsou dlouhodobé efekty, které mohou způsobit selhání zdroje entropie. Existují aktivní monitorovací techniky detekující totální selhání. Nicméně jemné chyby se těžko v praxi detekují.

Technika sběru

Zdroj entropie zaznamenáváme použitím techniky sběru, která v ideálním případě nenaruší fyzikální proces a sesbírá nejvyšší možnou entropii. Mnoho konstrukcí realizuje tento krok. Vzhledem k tomu, že neexistuje jiný způsob

[5]Převzato z [3]

analýzy výstupu TRNG než statistickými testy a jednoduchými testy opravdové náhodnosti (Tot a restart testy), mechanismus sběru by měl mít přesné zdůvodnění. Jednoduše řečeno, Tot testy kontrolují celkové selhání zdroje entropie RNG obvykle způsobeného vlivem stárnutí materiálu nebo extrémním kolísáním v provozních podmínkách. Restart testy ověřují generování náhodnosti restartováním RNG za téměř stejných provozních podmínek.

Následné zpracování

Ačkoliv tato součást není vyžadována ve všech případech, dobrá konstrukce přikazuje použít postprocessing. Cílem je zvýšit robustnost konstrukce TRNG použitím následného zpracování výstupních bitů. Postprocessing se využívá ke skrytí nebo eliminaci výkyvů anebo závislostí ve zdroji entropie nebo mechanismu sběru. Druhý cíl, který získal na důležitosti kvůli aktivním útokům na selhání systému a útokům postranními kanály, je poskytnout odolnost proti změnám prostředí a proti manipulaci útočníky. Postprocessing může realizovat jednoduchý von Neumannův korektor nebo komplikovanější extrakční funkce nebo jednosměrná hašovací funkce jako SHA-1. I když jednosměrná hašovací funkce jako SHA-1 nebo MD5 poskytuje bezpečnost při použití v následném zpracování, komplikují analýzu rozdělení výstupu.

1.4.2 Požadavky TRNG

Tím pádem existuje velké množství konstrukcí TRNG. Jednotlivé typy se znatelně liší podle zdrojů entropie nebo použité techniky sběru. Každá konstrukce má své silné a slabé stránky. Některé z těchto vlastností se týkají výkonu a jiné bezpečnosti a robustnosti. Některé požadavky na TRNG shrnuje [7].

- Z praktického hlediska je nezbytné vyrábět TRNG pomocí běžného křemíkového procesu. Navíc je vysoce žádoucí implementovat TRNG použitím čistě digitální techniky. To dovoluje snadnější integraci s digitálními mikroprocesory a možnost implementovat TRNG na populárních konfigurovatelných platformách (např. FPGA a CPLD).

- Kompaktní a efektivní konstrukce s vysokou propustností dané prostorem a využitou energií. Vyvarovat se použití zesilovačů a jiných analogových součástek pokud je to možné. Analogové součástky mají tendenci spotřebovat více energie a ztížit analýzu. Poznamenejme, že protože zakazujeme analogové komponenty, musíme vzorkovat změny v čase jinak než pomocí úrovně napětí. V případě striktního dodržení tohoto kritéria bychom neměli používat komplikovaná schémata následného zpracování (např. SHA-1), nebo je alespoň implementovat v softwaru.

- Je žádoucí mít matematické zdůvodnění mechanismu sběru entropie a všechny předpoklady empiricky ověřit. Návrh by měl být dostatečně

jednoduchý, aby umožnil přesnou analýzu. K ověření výstupu TRNG se používají testovací sady jako DIEHARD [9] nebo NIST [10].

1.4.3 Techniky následného zpracování TRNG

V praxi existují různé možnosti následného zpracování. Zde prezentujeme ty nejobvyklejší, jak je uvedeno v [7].

- Kryptografické hašovací funkce:

  pravděpodobně nejpopulárnější technikou je použít kryptograficky silnou hašovací funkci (SHA-1) na výstup TRNG. Například Intel RNG používá SHA-1. Z hlediska výkonu se implementování plnohodnotné hašovací funkce v TRNG jeví jako přehnané. Avšak pokud je vhodně implementovaná má důležitý přínos z hlediska bezpečnosti - pokud selže zdroj náhodnosti tak TRNG "pouze" degraduje na PRNG.

- Von Neumannův korektor:

  jde o jednu z nejstarších a nejznámějších technik. Používá se k potlačení lokálních výkyvů náhodných hodnot. Přijímá na vstupu dva bity a pokud jsou stejné (oba 0 nebo 1) odstraní je z posloupnosti náhodných bitů, pokud jsou odlišné, použije se jeden z bitů - například první. Rychlost výstupu je pak zpomalena přibližně o jednu čtvrtinu. Jeho velkou výhodou je snadná implementace.

1.5 Praktické použití RNG

CSPRNG

- generování klíčů asymetrické kryptografie
- generování hodnot na jedno použití tzv. *nonce* (number used once)
- solení v digitálních podpisech
- jednorázové šifry (one-time pad)

1.6 Aplikace RNG

V této kapitole zmiňujeme některé konkrétní aplikace RNG.

- CSPRNG na linuxu dostupný jako `/dev/random` a `/dev/urandom`
- CSPRNG implementovaný v OpenSSL používaný ke generování klíčů asymetrické kryptografie (např. RSA SSH klíče)

- TrusZone RNG - skládá se ze dvou částí - hardwarového TRNG, který generuje seed pro softwarový PRNG
- TL200 - hardwarový TRNG s USB rozhraním, vestavěný posprocessing (např. SHA-512)

Kapitola 2

Typy konstrukcí TRNG

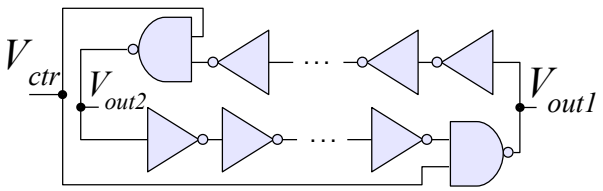
V této sekci se zabýváme přehledem nejznámějších konstrukcí TRNG.

2.1 TERO TRNG

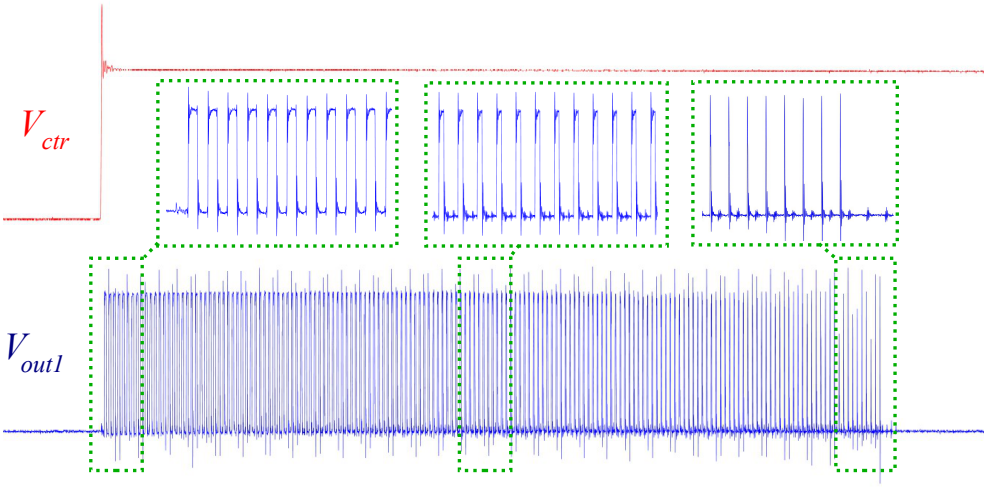
Hlavní součástí tohoto typu TRNG je elektronický obvod TERO (Transient Effect Ring Oscilator), který dočasně osciluje. Skládá se ze sudého počtu invertorů a páru hradel, který restartuje dočasné oscilace (např. dvě hradla NAND nebo XOR). Typické schéma TERO obvodu zobrazuje 2.1. Skládá se ze dvou hradel NAND a dvou větví s invertory. TERO obvod můžeme chápat jako RS klopný obvod se dvěma vstupy stejného napětí V_{ctr} a dvěma rozdílnými výstupy V_{out1} a V_{out2}.

Při náběžné hraně vstupu V_{ctr}, výstupy V_{out1} a V_{out2} začnou oscilovat. Oscilace mají v průměru konstantní frekvenci, ale jejich střída se časem mění. Střída je poměr časů obdélníkového signálu v jednotlivých úrovních. Střída se mění monotónně a po určitém počtu oscilací dosáhne poměru 0%, nebo 100%. V této chvíli výstupy V_{out1} a V_{out2} přestanou oscilovat a ustálí se ve dvou navzájem opačných logických hodnotách. Obrázek 2.2 ukazuje průběh signálů vstupu V_{ctr} a výstupu V_{out1} naměřený osciloskopem. Jak vidíme, po náběžné hraně řídícího signálu V_{ctr} začne výstup V_{out1} oscilovat.

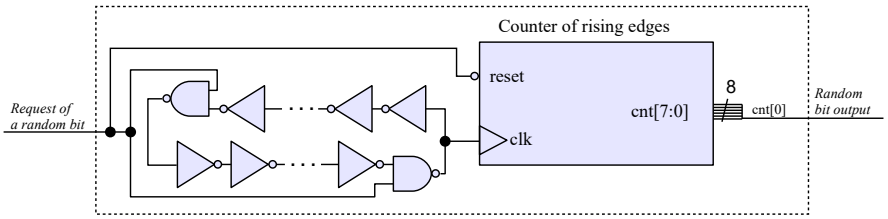
[6] Převzato z [11]



Obrázek 2.1: TERO Obvod[6]



Obrázek 2.2: průběh signálů TERO obvodu[7]



Obrázek 2.3: stavba TERO TRNG[8]

Tři přiblížené pohledy ukazují změnu střídy v čase, která postupně klesá, až se signál V_{out1} ustálí v hodnotě logická 0.

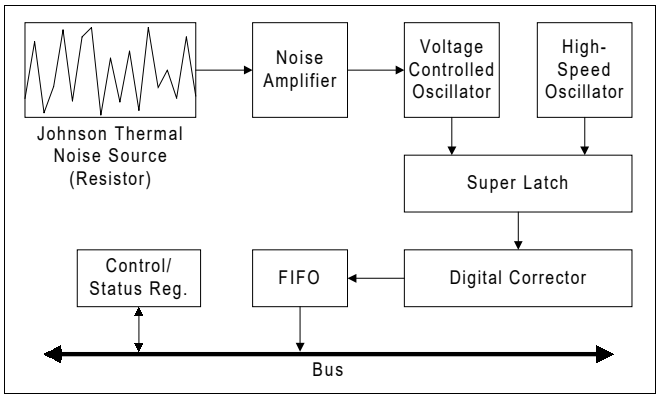
Počet oscilací předtím, než se výstup stabilizuje se vždy liší, protože ho ovlivňuje elektrický šum, který narušuje normální chování tranzistorů TERO obvodu.

Strukturu tohoto typu TRNG zobrazuje 2.4. Již zmíněný TERO obvod je napojen na čítač, který počítá náběžné hrany dočasných oscilací. Náhodná binární posloupnost obvykle vzniká spojením nejnižších bitů čítače.

2.2 Intel TRNG

Tento typ je popsán v [12]. Vzorkuje teplotní šum zesílením napětí naměřeném na nevybuzených rezistorech. Navíc k velké náhodné části jsou tato

[7] Převzato z [11]
[8] Převzato z [11]



Obrázek 2.4: Blokové schéma Intel TRNG[9]

měření ovlivněna pseudonáhodnými vlastnostmi prostředí jako elektromagnetické záření, teplota a kolísání zdroje napájení. Odečtením signálů naměřených na dvou sousedních rezistorech Intel RNG znatelně sníží párové součásti.

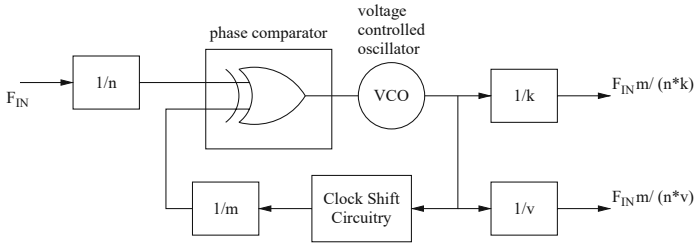
Dále se využívá dvou volně běžících oscilátorů. Jeden je rychlý a druhý mnohem pomalejší. Teplotní šum se používá k modulování frekvence pomalejšího hodinového signálu. Ten je použit k aktivování měření rychlejších hodin. Odchylky těchto dvou hodin poskytují zdroj náhodných binárních číslic.

2.3 Konstrukce Fischer-Drutarovský

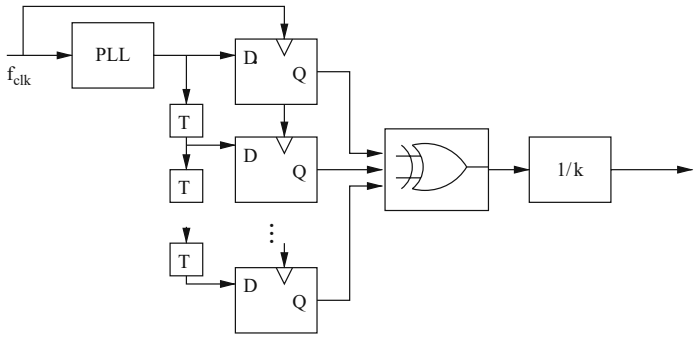
Tento TRNG je založen na analogovém Phase-Locked Loop (PLL). Návrh byl implementován v Altera Field Programmable Logic Device (FPLD). Konstrukce je unikátní v tom, že šlo o první TRNG, který byl cílen na FPGA. Jak zobrazuje 2.5 vzorkuje se jitter hodinového signálu generovaného PLL pomocí zpožděných kaskádových vzorkovačů, sestavených dle schématu na obrázku 2.6.

Klíčovou myšlenkou je použít více vzorkovačů, aby bylo možné vzorkovat přechodovou zónu, která je ovlivněná jitterem, který podle [13] probíhá v řádu desítek pikosekund. Vzorky zaznamenané v pravidelných intervalech jsou zkombinovány dohromady pomocí funkce XOR. Výsledkem je vzorek z oblasti křivky signálu, která má potřebnou náhodnost. Tento návrh je zajímavý, protože osvětluje potřebu stavět TRNG na rekonfigurovatelných platformách.

[9] Převzato z [12]
[10] Převzato z [7]
[11] Převzato z [7]



Obrázek 2.5: Stavba PLL použité jako zdroje entropie konstrukce TRNG Fischer-Drutarovský[10]



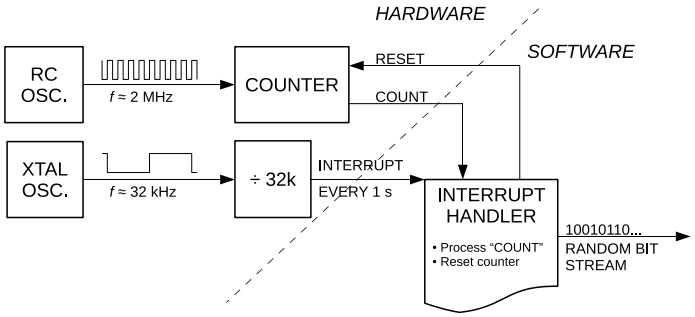
Obrázek 2.6: TRNG konstrukce Fischer-Drutarovský[11]

2.4 TRNG na Atmel AVR

Následující konstrukce je popsána v [14]. I když 8 bitové mikrokontroléry série AVR nemají přímo určený hardware ke generování náhodných čísel, ukázalo se, že je možné použít jako zdroj entropie vestavěný oscilátor bez dalších součástek nebo úprav. Využívá se faktu, že mikrokontrolér Atmega 169 na desce AVR Butterfly může přistupovat k několika oscilátorům. Základní konfigurace používá čipový RC oscilátor jako systémové hodiny. Může využívat i jiný oscilátor s externě připojeným krystalem jako generátorem pulzů pro jednotku časovač / čítač 2.

Tyto oscilátory nejsou nikdy ideálně stabilní. Jejich frekvence ovlivňuje mnoho fyzikálních vlivů jako napájecí napětí, provozní teplota či přirozené vlastnosti jitteru. Dopady těchto vlivů jsou znatelně odlišné pro každý oscilátor. Krystalový oscilátor je mnohem více stabilní v široké škále provozních podmínek. Zatímco RC oscilátor vykazuje patrné odchylky i když je kalibrovaný. I když je mikrokontrolér ve stabilním prostředí a napájen stabilizovaným zdrojem napětí, RC oscilátor vykazuje jednoduše měřitelný jitter.

Princip fungování je na obrázku 2.7. Zdrojem entropie je jitter RC oscilátoru. Krystalový oscilátor byl použit k periodickému časování (1 sekunda), a za tu dobu se naměřil počet pulzů RC oscilátoru. V ideálním světě by byl vždy stejný, avšak reálně byly pozorovány odchylky až 2^{12} pulzů mezi



Obrázek 2.7: Princip Atmel AVR TRNG[12]

následnými měřeními.

Ke generování náhodných bitů se používá následující metoda. Čítač / časovač 1 počítá pulzy systémového RC oscilátoru a běží na 2 MHz. Čítač 2 využívá krystalový oscilátor (32,768 kHz) a odpovídající nastavení předděličky generuje přerušení každou 1 sekundu. V obsluze přerušení se přečte hodnota čítače / časovače 1 a čítač je vynulován. Pak jsou z 16 bitové hodnoty extrahovány bity 1 až 8 a předány na výstup jako část generované posloupnosti náhodných bitů.

[12]Převzato z [14]

Kapitola 3

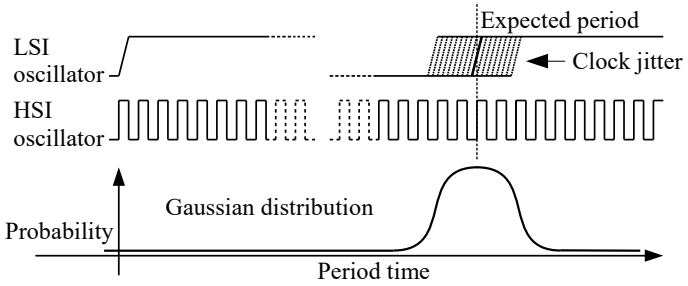
Návrh a implementace

V této kapitole popisujeme návrh a implementaci vlastního TRNG na mikrokontroléru.

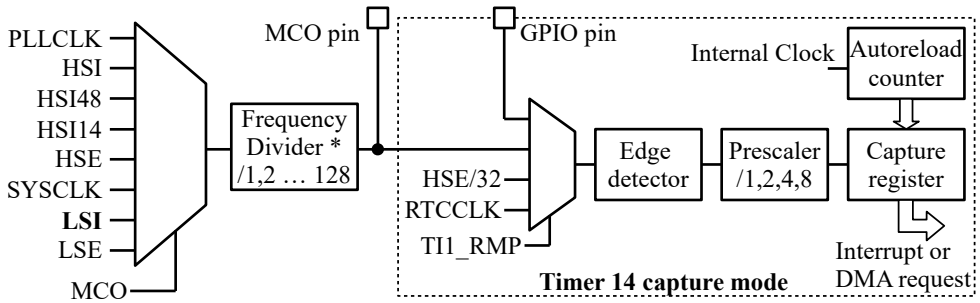
3.1 Popis implementace

Při navrhování TRNG vycházíme z [15]. Na rozdíl od citovaného návrhu používáme 32 bitový mikrokontrolér STM32F030R8T6 [16] se stejným procesorem tj. ARM-CORTEX-M0. Zdrojem náhodnosti je nestabilita hodinového signálu tzv. *clock jitter*. Princip zobrazuje schéma 3.1. Generátor používá dva nezávislé oscilátory. Jedním z nich je vysokorychlostní interní oscilátor HSI (High Speed Internal) kmitající na frekvenci 48 MHz a druhý je pomalejší externí krystalový oscilátor LSE (Low Speed External) s frekvencí 32,768 KHz, který nahrazuje vnitřní 40 KHz LSI (Low Speed Internal) krystal v původním návrhu. Po předem určeném počtu pulzů LSE změříme počet period uplynulých na HSI oscilátoru. V ideálním případě bude hodnota vždy konstantní, ale v realitě způsobí šum odchylky, které se dají použít při generování náhodných čísel. Tyto hodnoty mají gaussovské rozdělení.

[13] Převzato z [15]



Obrázek 3.1: Princip clock jitteru v TRNG[13]



Obrázek 3.2: Schéma čítače 14 v TRNG. *Našemu modelu chybí dělička frekvence MCO pinu.[14]

Čítač 14 (Timer 14) v zachytávacím módu používáme k měření period HSI. Nastavení je znázorněno na obrázku 3.2. Oscilátor LSE je přes první multiplexor vyveden na MCO (microcontroller clock output) pin, který se používá jako zdroj pulzů vstupující do detektoru hran (Edge detector). Ten je nastaven, aby reagoval na každou náběžnou hranu. Dále pomocí předděličky (Prescaler) můžeme zvolit počet pulzů (1, 2, 4, 8) potřebných k aktivování záchytného (capture) registru. Při takové události se aktuální hodnota čítače napojeného na HSI zaznamená v capture registru a případně se vyvolá odpovídající přerušení.

3.2 Hardware

Používáme mikrokontrolér STMF030R8T6 [16] s 32bitovým procesorem ARM-Cortex-M0 s nastavitelným kmitočtem v rozsahu 8 - 48 MHz viz. obrázek 3.3.

Uživatel má k dispozici:

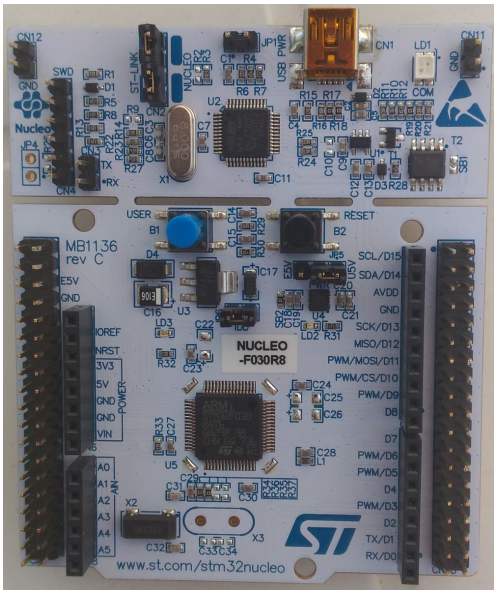
- zelenou LED diodu s označením LD2, kterou může programově ovládat
- modré tlačítko B1, na jehož stisk může program reagovat
- černé tlačítko B2, jenž slouží k resetu mikrokontroléru

Přípravek je napájen mikro USB konektorem, který zároveň slouží k připojení kontroléru k počítači. Tímto způsobem se do mikrokontroléru nahrávají programy a probíhá komunikace s počítačem přes sériovou linku.

Deska je osazena třemi oscilátory. Dva z nich jsou integrální součástí procesoru (HSI, LSI) a třetí je externí krystalový oscilátor (LSE). Deska nabízí možnost dodatečné instalace HSE (High Speed External) oscilátoru.

Mikrokontrolér je vybaven flash pamětí 64KB, která slouží k uložení programu.

[14] Převzato z [15]



Obrázek 3.3: Mikrokontrolér STM32F030R8T6

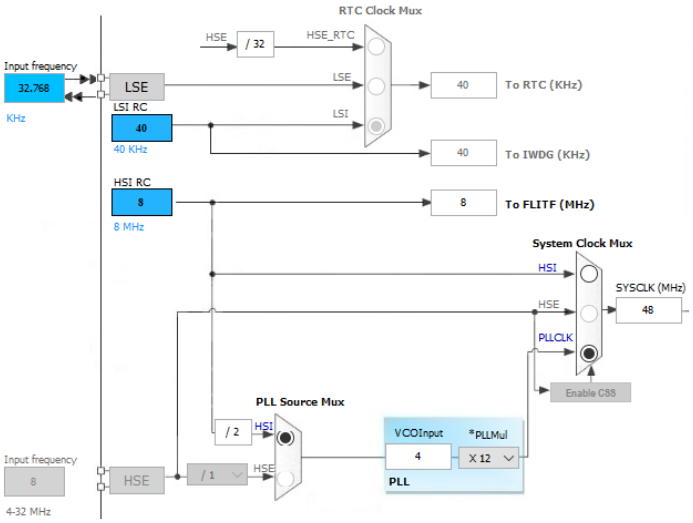
3.3 Program

Program pro mikrokontrolér je vyvinut v jazyku C. K vývoji používáme oficiální vývojové prostředí ARM Keil, běžící na Windows. Jeho úkolem je zdrojový kód v jazyku C přeložit do spustitelné binární podoby, běžící na procesoru ARM a také pomocí sériové linky program nahrát do mikrokontroléru. ARM Keil nabízí obvyklý standard pro vývojářskou práci jako je krokování programu, breakpointy, či zobrazování hodnot proměnných.

Fáze běhu programu v mikrokontroléru:

- nastavení hodin (oscilátorů)
- nastavení sériové linky
- nastavení čítače 14
- nastavení přerušení
- čekání na znak ”s”
- odeslání hodnoty do počítače

Po nahrání programu do mikrokontroléru pomocí stisknutí tlačítka reset dojde k jeho spuštění. Po spuštění programu v mikrokontroléru dojde k inicializaci řídících registrů. Nejdůležitějším krokem této fáze je správné nastavení hodin celého systému.



Obrázek 3.4: Schéma nastavení hodin mikrokontroléru

Zdrojem hodinového signálu je vnitřní HSI RC oscilátor. Signál je přes děličku přiveden do PLL (Phase Locked Loop), kde je jeho poloviční frekvence (4 MHz) vynásobena 12. Multiplexor je nastaven tak, že jako zdroj systémových hodin (SYSCLK) vybere výstup z PLL s frekvencí 48 MHz. SYSCLK se používá dále jako hodinový signál sběrnice, hodiny periferií a čítačů, včetně čítače 14.

Dalším krokem je nastavení parametrů sériové linky, která slouží k přenosu naměřených hodnot z mikrokontroléru do počítače. Používáme 115,2 kbit/s.

Následně je inicializován čítač 14 v záchytném režimu - jak bylo popsáno výše - a nastaveno přerušení. Přerušení je vyvoláno při události zachycení hodnoty čítače v ”capture registru”. V obsluze přerušení dojde k přečtení hodnoty z ”capture registru”, vynulování čítače 14 a uložení hodnoty do kruhové fronty.

Nyní program na sériové lince očekává znak ”s”, čímž dojde ke spuštění čítače a záchytného registru. Program vstoupí do hlavní smyčky - ve chvíli kdy kruhová fronta obsahuje naměřenou hodnotu, odešle se přes sériovou linku do počítače.

V počítači jsou hodnoty načítány pomocným programem (napsaným v jazyce C#) a ukládány do binárního souboru (2 byte unsigned integer).

Prvotní implementace programu pro mikrokontrolér využívala API softwarové vrstvy HAL (Hardware Abstraction Layer) [17]. Její výhodou je snadnější práce s programováním mikrokontroléru. Nabízí pomocné struktury a funkce. V našem konkrétním případě se ukázalo, že vrstva navíc způsobuje zpoždění po přečtení generované hodnoty. Důsledkem je zkreslení generovaných hodnot. Při nastavení předděličky na 1 by se hodnoty měly pohybovat okolo teoretické hodnoty 1464 dané poměrem frekvencí HSI a LSE oscilátorů. S použitím HAL výsledky měření dosahovaly intervalu okolo hodnoty 1370, tedy hod-

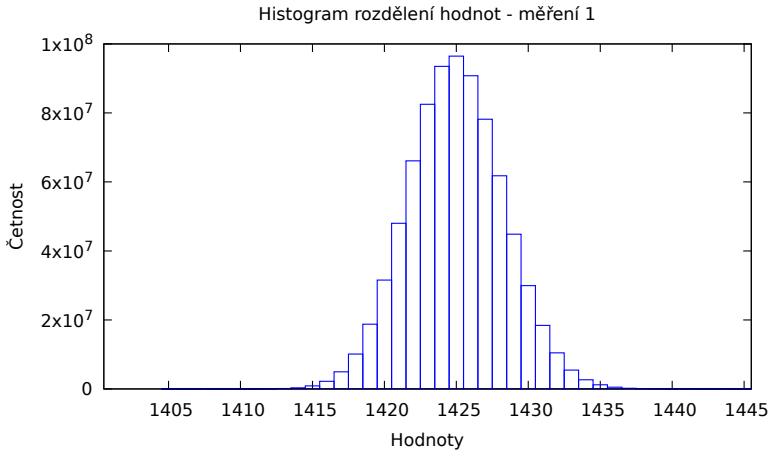
noty mnohem nižší. Proto jsme přepracovali implementaci bez použití zmíněné vrstvy, nyní pracujeme přímo s řídícími registry a tím jsme dosáhli mnohem přesnějších výsledných hodnot okolo 1430.

Kapitola 4

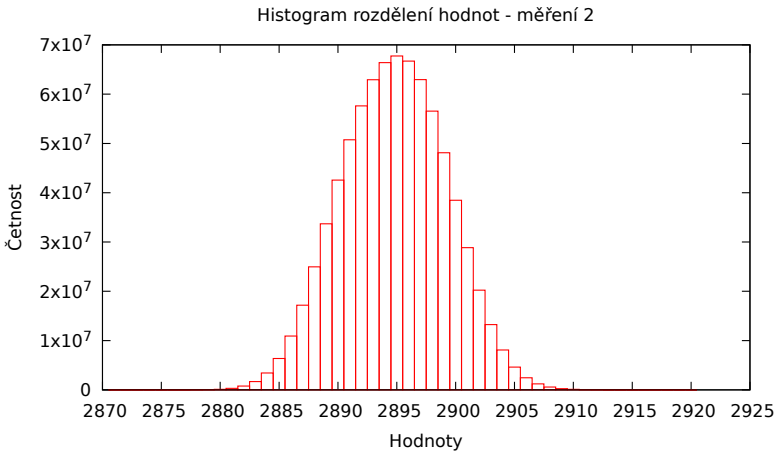
Měření

Před zahájením měření propojíme přípravek s počítačem, nahrajeme program do mikrokontroléru a stiskneme tlačítko reset. Spustíme pomocnou měřící aplikaci v počítači, zadáme konfiguraci a potvrdíme jméno výstupního souboru. Tím dojde k odeslání startovního znaku "s" z počítače do přípravku, který začne generovat náhodné hodnoty. Ty jsou v počítači ukládány do binárního souboru.

Pro následné statistické vyhodnocení necháváme vygenerovat 800 miliónů hodnot, což odpovídá zhruba 40 hodinám běhu programu a objemu 1,4 GB dat. Celkem jsme provedli 4 série měření s různým nastavením předděličky (1, 2, 4, 8 pulzů na zachycení hodnoty).



Obrázek 4.1: Histogram měření 1



Obrázek 4.2: Histogram měření 2

4.1 Měření 1

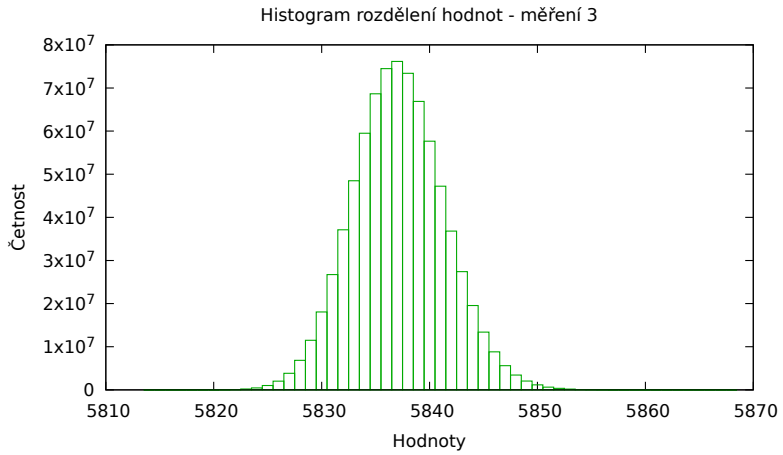
Histogram měření je na obrázku 4.1.

- Nastavení předděličky: 1 (zachycení hodnoty na každý pulz od LSE)
- Rozsah hodnot: 1405 - 1446
- Průměr: 1424.962
- Rozptyl: 10.98

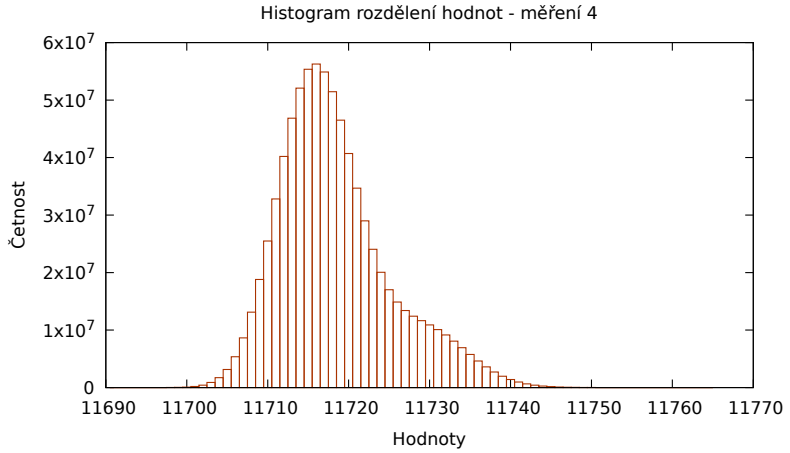
4.2 Měření 2

Histogram měření je na obrázku 4.2.

- Nastavení předděličky: 2 (zachycení hodnoty každé 2 pulzy od LSE)
- Rozsah hodnot: 2871 - 2920
- Průměr: 2894.762
- Rozptyl: 19.786



Obrázek 4.3: Histogram měření 3



Obrázek 4.4: Histogram měření 4

4.3 Měření 3

Histogram měření je na obrázku 4.3.

- Nastavení předděličky: 4 (zachycení hodnoty každé 4 pulzy od LSE)
- Rozsah hodnot: 5814 - 5868
- Průměr: 5837.125
- Rozptyl: 18.0768

4.3.1 Analýza výsledků měření 3

Histogram četností hodnot měření 3 vykazuje nižší hodnotu rozptylu oproti předpokladu pro dané nastavení předděličky. Rozptyl by měl lineárně růst v rámci jednotlivých měření. Proto jsme provedli další analýzu s cílem objasnit tuto anomálii. Vygenerovali jsme 800 histogramů vždy po milionu hodnot a z nich vytvořili video animaci. Histogramy i animaci přikládáme na CD.

Vygenerované histogramy přibližně odpovídají celkovému histogramu. Hodnoty rozptylu nevykazují žádné větší odchylky. Řešením by bylo měření s daným nastavením předděličky zopakovat popř. s jiným fyzickým zařízením.

4.4 Měření 4

Histogram měření je na obrázku 4.4.

- Nastavení předděličky: 8 (zachycení hodnoty každých 8 pulzů od LSE)
- Rozsah hodnot: 11691 - 11764
- Průměr: 11718.327
- Rozptyl: 48.153

4.4.1 Analýza výsledků měření 4

Na histogramu četností hodnot měření 4 je v pravé části grafu patrná odchylka od ideální gaussovské křivky (kolem hodnoty 11730). Proto jsme provedli další analýzu s cílem objasnit tuto anomálii. Vygenerovali jsme 800 histogramů vždy po milionu hodnot a z nich vytvořili video animaci. Histogramy i animaci přikládáme na CD.

Jak je z grafů patrné, k vychýlení celé křivky směrem doprava k vyšším hodnotám došlo celkem 3x okolo 300., 527. a 670. měřeného milionu hodnot. Aby došlo k pozorovanému jevu, muselo by buď dojít ke zrychlení vnitřního oscilátoru (HSI), anebo ke zpomalení vnějšího krystalového oscilátoru (LSE). Vzhledem k tomu, že nešlo o vychýlení trvalé, což by mohlo být způsobeno například zahřátím čipu, ale o tři výchylky během 40 hodinového měření, docházíme k závěru, že muselo jít o nějaký vnější vliv. Pravděpodobnou příčinou se nám jeví nestabilita zdroje napájení nebo změny okolní teploty.

4.5 Shrnutí

Všechna zmíněná měření mají očekávané gaussovské rozdělení charakteristické pro *clock jitter*, jak je zmíněno v kapitole výše. Dále si všimněme závislosti nastavení předděličky a rozptylu. Při vetším počtu pulzů na zachycenou hodnotu se nestabilita signálu hodin akumuluje, což má za následek i vyšší hodnotu rozptylu.

Kapitola 5

Statistické vyhodnocení

Statistické vyhodnocení provádíme pomocí sady testů v aplikaci NIST STS (Statistical Test Suite).

Z naměřených dat prostřednictvím pomocné aplikace vygenerujeme *bit-stream* - bitový proud sestávající například pouze z 1. nejnižšího bitu každé naměřené hodnoty. Takto získaný bitový proud dále vyhodnotíme pomocí sady testů STS.

5.1 Sada testů NIST STS

NIST test suite se skládá z 15 testů, které slouží k testování náhodnosti binárních sekvencí, produkovaných hardwarovými nebo softwarovými TRNG nebo PRNG. Testy se zaměřují na odhalení porušení náhodnosti, která se může v testované vstupní sekvenci objevit.

Příklady některých testů:

- Frequency (Monobit) Test - tento test se zaměřuje na zjištění množství nul a jedniček ve vstupní sekvenci. Účelem testu je zjistit zda nul a jedniček je přibližně stejné množství - t.j. polovina

- Non-overlapping Template Matching Test - tento test se zaměřuje na vyhodnocení počtu výskytů určité posloupnosti bitů. Účelem tohoto testu je detekovat generátory, které produkují mnoho výskytů daného ne-periodického vzoru.

- Linear Complexity Test - tento test se zaměřuje na délku registru s lineární zpětnou vazbou. Účelem testu je rozhodnout zda je sekvence dostatečně komplexní, aby ji bylo možné považovat za náhodnou. Náhodné sekvence charakterizuje delší LFSR (viz. výše)

5.2 Použití NIST STS

STS spustíme s parametrem, který udává délku testované sekvence v bitech. Dále zadáme vstupní soubor se zpracovávanými daty a kterou sadu testů chceme spustit. Následně zadáme počet sekvencí ke zpracování. Dále se zadává typ vstupního souboru - t.j. zda jde o binární či ASCII vstup. Po exekuci testů je vygenerován výstupní soubor (příklad je v příloze). Počet řádků odpovídá počtu provedených testů, dále výstup obsahuje pro každý test hodnotu p-value, která udává do jaké míry se testovaná sekvence podobá normálnímu rozdělení či rozdělení χ^2 a název testu.

5.3 Testovaná Data

V rámci každého měření jsme testovali nejméně významné bity na pozicích 0-3. Délka testované sekvence byla 1 milión a počet těchto sekvencí 800, což odpovídá 800 miliónům naměřených hodnot. Na základě výsledků jednotlivých bitů jsme dále provedli vyhodnocení dat získaných spojením bitů 0-1 a 0-1-2.

5.4 Výsledky

V příloze jsou výstupy testů měření 1 pro bit 0 - všechny testy úspěšně prošly, bit1 - jeden test selhal a bit 2 - vetšina testů selhala. Výsledky statistických testů shrnují tabulky 5.1 a 5.2. Řádek reprezentuje jedno měření charakterizované nastavením předděličky. Ve sloupcích jsou testované bity. Na výsledky lze nahlížet ze dvou pohledů. Zaprvé zkoumat rozdíly v rámci jednotlivých bitů a zadruhé analyzovat výsledky s různým nastavením předděličky.

Testy prokázaly předpokládanou tendenci klesající kvality náhodnosti s rostoucí pozicí bitu. Výsledky s bity 0 nebo 1 vycházejí nejlépe. Podobně dobře vychází i spojení bitů 0-1. V obou těchto případech buď prošly všechny testy úspěšně anebo neprošly pouze jednotky testů. U bitů 2, 3 a rovněž u spojení bitů 0-2 došlo k selhání většiny testů.

Můžeme konstatovat, že spojením bitu 0 a 1 se výsledky statistických testů zlepšily. Neboť s nastavením předděličky na 1, 2 a 8 prošly úspěšně všechny testy na rozdíl od výsledků bitu 0 nebo 1, kde alespoň při dvou nastaveních předděličky neprošel nějaký test. Také lze říci, že i pro bity 2 nebo 3 lze ve výsledcích, s rostoucím číslem předděličky, pozorovat zlepšující se trend - tedy zvyšující se počet úspěšných testů.

5.5 Shrnutí

Výsledky indikují, že se daný návrh dá použít jako TRNG. Abychom si mohli být jistí je třeba provést podrobnější vyhodnocení, použitá sada testů slouží k vyhodnocení PRNG, ale lze ji využít pro základní vyhodnocování TRNG.

předdělička	bit 0	bit 1	bit 2	bit 3
1	ok	ok*	x	x
2	ok	ok	x	x
4	ok*	ok**	x	x
8	ok*	ok*	x	x

* ... neprošel jeden ze statistických testů
** ... neprošly dva ze statistických testů
x ... většina statistických testů selhala

Tabulka 5.1: Výsledky testů NIST STS jednotlivých bitů.

předdělička	bit 0-1	bit 0-2
1	ok	x
2	ok	x
4	ok*	x
8	ok	x

* ... neprošel jeden ze statistických testů
x ... většina statistických testů selhala

Tabulka 5.2: Výsledky testů NIST STS spojených bitů.

Pro reálné nasazení TRNG se jeví jako nejvhodnější použití sekvence tvořené bitem 0 nebo 1 popřípadě sekvence spojených bitů 0 a 1. Jestliže chceme dosáhnout co největší výtěžnosti tak je při daných výsledcích vhodné použít nastavení předděličky na 1 a vybírat spojené bity 0 a 1.

Závěr

Cílem rešeršní části této práce bylo seznámit se s problematikou generátorů náhodných čísel a jejich klasifikací. Dále prozkoumat existující řešení generátorů skutečně náhodných čísel s důrazem na konstrukce vhodné pro mikrokontroléry. V praktické části jsme se zabývali implementací vlastního TRNG na mikrokontroléru. Provedli jsme měření a statistické vyhodnocení jeho výstupů.

V rámci práce jsme splnili všechny požadavky. Provedli jsme literární rešerši, podařilo se nám úspěšně implementovat generátor náhodných čísel na mikrokontroléru STM32F030R8T6 s procesorem ARM Cortex-M0. V kapitole 4 jsme provedli 4 měření s různým nastavením předděličky a následně v kapitole 5 jsme provedli jejich statistické vyhodnocení. Na základě výsledků NIST STS se jeví implementovaný návrh RNG jako vhodný pro použití jako TRNG. Nejlepších výsledků jsme dosáhli při nastavení předděličky na 1 a výběru bitů 0 a 1.

V práci by se dalo pokračovat následovně:

- Provést opakované měření s nastavením předděličky 4 a 8.
- Provést rozsáhlejší měření za různých provozních podmínek, např. rozdílné teploty, zdroje napájení.
- Spustit implementaci TRNG na více fyzických mikrokontrolérech stejného typu a porovnat statistické výsledky jednotlivých zařízení.
- Provést měření s použitím LSI oscilátoru (interní RC oscilátor) a výsledky porovnat.

Literatura

[1] Herrero-Collantes, M.; Garcia-Escartin, J. C.: *Quantum Random Number Generators*, říjen 2016, [cit. 2018-04-25]. Dostupné z: `https://arxiv.org/pdf/1604.03304.pdf`

[2] Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A.: *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997, [cit. 2018-04-27].

[3] Schindler, W.: *Random Number generators for cryptographic applications*. Cryptographic Engineering, Springer-Verlag, 2009, ISBN 978-0-387-71816-3.

[4] Boyar, J.: Inferring Sequences Produced by Pseudo-Random Number Generators. *Journal of the Association for Computing Machinery, Vol. 36, No. 1*, leden 1989, [cit. 2018-04-29]. Dostupné z: `http://asterix.cs.gsu.edu/crypto/p129-boyar.pdf`

[5] Massey, J. L.: *Shift-Register Synthesis and BCH Decoding*, leden 1969: s. 122–127.

[6] Hlaváč, J.; Kodýtek, F.: *Random Number Generators*. Přednáška z předmětu Hardwarová bezpečnost. České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[7] Sunar, B.: *True Random Number Generators for Cryptography*. Cryptographic Engineering, Springer-Verlag, 2009, ISBN 978-0-387-71816-3, 55-73 s.

[8] Random.org: *True Random Number Service*. Dostupné z: `www.random.org`

[9] Marsaglia, G.: *The Marsaglia Random Number CDROM, with The Diehard Battery of Tests of Randomness*. [online] Florida State University, 1995, [cit. 2018-05-06]. Dostupné z: `www.stat.fsu.edu/pub/diehard/`

[10] Rukhin A. et al: *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. National Institute of Standards and Technology, NIST Special Publication 800-22rev1a, Duben 2010, [cit. 2018-05-06]. Dostupné z: `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf`

[11] Haddad, P.; Fischer, V.; Bernard, F.; aj.: *A Physical Approach for Stochastic Modeling of TERO-based TRNG*, červen 2015, [cit. 2018-04-30]. Dostupné z: `https://eprint.iacr.org/2015/593.pdf`

[12] Jun, B.; Kocher, P.: *The Intel® Random Number Generator*. Technical report, Cyptography Research Inc., San Francisco, 1999.

[13] Fischer, V.; Drutarovský, M.: *True Random Number Generator Embedded in Reconfigurable Hardware*. In Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002). Redwood Shores (USA), Springer-Verlag, srpen 2002, ISBN 3-540-00409-2, 415-430 s.

[14] Hlaváč, J.; Hadáček, M.; Lórencz, R.: *True Random Number Generation on an Atmel AVR Microcontroller*. Proceedings of 2nd International Conference on Computer Engineering and Technology – ICCET 2010, vol. 2, 2010, ISBN 978-1-4244-6347-3, 493-495 s.

[15] Laban, M.; Drutarovský, M.: *Low Cost ARM Cortex-M0 based TRNG for IOT Applications*, 2014, ISSN 1335-8243.

[16] STMicroelectronics: *STMF030x8 Datasheet*. [cit. 2018-05-06]. Dostupné z: `http://www.st.com/resource/en/datasheet/stm32f030r8.pdf`

[17] STMicroelectronics: *Description of STM32F0 HAL and low layer drivers*. [online], [cit. 2018-05-09]. Dostupné z: `www.st.com/resource/en/user_manual/dm00122015.pdf`

Příloha **A**

Výstupy NIST STS

Na následujících stránkách přikládáme výstupy statistických testů NIST pro měření 1.

Měření 1 - bit 0

Všechny testy úspěšně prošly.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <Onyx-INT1aa_bit-0>
---------------------------------------
 P-VALUE  PROPORTION  STATISTICAL TEST
---------------------------------------
 0.514124    791/800     Frequency
 0.749884    790/800     BlockFrequency
 0.892399    790/800     CumulativeSums
 0.757297    794/800     CumulativeSums
 0.446556    797/800     Runs
 0.857592    793/800     LongestRun
 0.952387    797/800     Rank
 0.024981    792/800     FFT
 0.275709    792/800     NonOverlappingTemplate
 0.257329    791/800     NonOverlappingTemplate
 0.504219    796/800     NonOverlappingTemplate
 0.694171    795/800     NonOverlappingTemplate
 0.477392    792/800     NonOverlappingTemplate
 0.644928    792/800     NonOverlappingTemplate
 0.582629    795/800     NonOverlappingTemplate
 0.322901    794/800     NonOverlappingTemplate
 0.562080    795/800     NonOverlappingTemplate
```

```
0.717205    792/800     NonOverlappingTemplate
0.524101    789/800     NonOverlappingTemplate
0.453583    784/800     NonOverlappingTemplate
0.914672    795/800     NonOverlappingTemplate
0.482223    791/800     NonOverlappingTemplate
0.670915    793/800     NonOverlappingTemplate
0.951205    792/800     NonOverlappingTemplate
0.969946    797/800     NonOverlappingTemplate
0.293235    788/800     NonOverlappingTemplate
0.375313    791/800     NonOverlappingTemplate
0.230755    791/800     NonOverlappingTemplate
0.340461    794/800     NonOverlappingTemplate
0.865697    794/800     NonOverlappingTemplate
0.582629    789/800     NonOverlappingTemplate
0.739918    792/800     NonOverlappingTemplate
0.941144    793/800     NonOverlappingTemplate
0.881284    789/800     NonOverlappingTemplate
0.921005    789/800     NonOverlappingTemplate
0.499295    794/800     NonOverlappingTemplate
0.273999    791/800     NonOverlappingTemplate
0.330628    791/800     NonOverlappingTemplate
0.714660    797/800     NonOverlappingTemplate
0.960198    791/800     NonOverlappingTemplate
0.587791    788/800     NonOverlappingTemplate
0.150552    789/800     NonOverlappingTemplate
0.704442    791/800     NonOverlappingTemplate
0.242986    793/800     NonOverlappingTemplate
0.083654    787/800     NonOverlappingTemplate
0.524101    794/800     NonOverlappingTemplate
0.539193    790/800     NonOverlappingTemplate
0.354548    792/800     NonOverlappingTemplate
0.379555    793/800     NonOverlappingTemplate
0.618905    788/800     NonOverlappingTemplate
0.709558    792/800     NonOverlappingTemplate
0.678686    792/800     NonOverlappingTemplate
0.899521    793/800     NonOverlappingTemplate
0.759756    790/800     NonOverlappingTemplate
0.247698    794/800     NonOverlappingTemplate
0.204985    790/800     NonOverlappingTemplate
0.362765    794/800     NonOverlappingTemplate
0.057601    793/800     NonOverlappingTemplate
0.119682    790/800     NonOverlappingTemplate
0.489508    793/800     NonOverlappingTemplate
0.151616    794/800     NonOverlappingTemplate
```

0.221898	790/800	NonOverlappingTemplate
0.531629	797/800	NonOverlappingTemplate
0.448892	784/800	NonOverlappingTemplate
0.403403	798/800	NonOverlappingTemplate
0.719747	791/800	NonOverlappingTemplate
0.320988	792/800	NonOverlappingTemplate
0.747401	791/800	NonOverlappingTemplate
0.961247	789/800	NonOverlappingTemplate
0.031323	789/800	NonOverlappingTemplate
0.696743	792/800	NonOverlappingTemplate
0.676097	794/800	NonOverlappingTemplate
0.811993	793/800	NonOverlappingTemplate
0.344448	790/800	NonOverlappingTemplate
0.309677	794/800	NonOverlappingTemplate
0.373203	792/800	NonOverlappingTemplate
0.825505	792/800	NonOverlappingTemplate
0.781584	793/800	NonOverlappingTemplate
0.300464	784/800	NonOverlappingTemplate
0.250878	794/800	NonOverlappingTemplate
0.696743	790/800	NonOverlappingTemplate
0.696743	786/800	NonOverlappingTemplate
0.304126	792/800	NonOverlappingTemplate
0.016573	793/800	NonOverlappingTemplate
0.701879	792/800	NonOverlappingTemplate
0.762209	785/800	NonOverlappingTemplate
0.881284	792/800	NonOverlappingTemplate
0.749884	793/800	NonOverlappingTemplate
0.556970	791/800	NonOverlappingTemplate
0.631914	796/800	NonOverlappingTemplate
0.798139	793/800	NonOverlappingTemplate
0.701879	792/800	NonOverlappingTemplate
0.932904	792/800	NonOverlappingTemplate
0.869673	793/800	NonOverlappingTemplate
0.214722	794/800	NonOverlappingTemplate
0.699313	793/800	NonOverlappingTemplate
0.973388	799/800	NonOverlappingTemplate
0.090252	792/800	NonOverlappingTemplate
0.988867	790/800	NonOverlappingTemplate
0.769527	790/800	NonOverlappingTemplate
0.964295	790/800	NonOverlappingTemplate
0.244549	793/800	NonOverlappingTemplate
0.877468	791/800	NonOverlappingTemplate
0.304126	794/800	NonOverlappingTemplate
0.401199	796/800	NonOverlappingTemplate

```
0.932904    791/800     NonOverlappingTemplate
0.516611    794/800     NonOverlappingTemplate
0.840797    791/800     NonOverlappingTemplate
0.482223    794/800     NonOverlappingTemplate
0.233767    794/800     NonOverlappingTemplate
0.135331    792/800     NonOverlappingTemplate
0.531629    793/800     NonOverlappingTemplate
0.987896    796/800     NonOverlappingTemplate
0.951205    794/800     NonOverlappingTemplate
0.734904    792/800     NonOverlappingTemplate
0.143279    796/800     NonOverlappingTemplate
0.629311    794/800     NonOverlappingTemplate
0.626709    789/800     NonOverlappingTemplate
0.394631    791/800     NonOverlappingTemplate
0.895989    794/800     NonOverlappingTemplate
0.300464    792/800     NonOverlappingTemplate
0.529115    795/800     NonOverlappingTemplate
0.639722    789/800     NonOverlappingTemplate
0.244549    791/800     NonOverlappingTemplate
0.428095    796/800     NonOverlappingTemplate
0.356591    791/800     NonOverlappingTemplate
0.724817    785/800     NonOverlappingTemplate
0.521600    794/800     NonOverlappingTemplate
0.315297    793/800     NonOverlappingTemplate
0.694171    794/800     NonOverlappingTemplate
0.088226    794/800     NonOverlappingTemplate
0.340461    791/800     NonOverlappingTemplate
0.021999    795/800     NonOverlappingTemplate
0.946308    787/800     NonOverlappingTemplate
0.164882    795/800     NonOverlappingTemplate
0.534146    788/800     NonOverlappingTemplate
0.642325    787/800     NonOverlappingTemplate
0.127762    794/800     NonOverlappingTemplate
0.514124    790/800     NonOverlappingTemplate
0.836482    788/800     NonOverlappingTemplate
0.153763    788/800     NonOverlappingTemplate
0.514124    793/800     NonOverlappingTemplate
0.311542    792/800     NonOverlappingTemplate
0.410055    794/800     NonOverlappingTemplate
0.598138    793/800     NonOverlappingTemplate
0.897763    793/800     NonOverlappingTemplate
0.786355    797/800     NonOverlappingTemplate
0.802793    791/800     NonOverlappingTemplate
0.467799    788/800     NonOverlappingTemplate
```

```
 0.840797    793/800     NonOverlappingTemplate
 0.430380    793/800     NonOverlappingTemplate
 0.159242    793/800     NonOverlappingTemplate
 0.595549    789/800     NonOverlappingTemplate
 0.727346    791/800     NonOverlappingTemplate
 0.871642    789/800     NonOverlappingTemplate
 0.639722    786/800     NonOverlappingTemplate
 0.061356    794/800     OverlappingTemplate
 0.375313    787/800     Universal
 0.039137    788/800     ApproximateEntropy
 0.093664    502/514     RandomExcursions
 0.996265    510/514     RandomExcursions
 0.192093    511/514     RandomExcursions
 0.192093    510/514     RandomExcursions
 0.985481    507/514     RandomExcursions
 0.610599    508/514     RandomExcursions
 0.238734    504/514     RandomExcursions
 0.751633    510/514     RandomExcursions
 0.900788    510/514     RandomExcursionsVariant
 0.984580    509/514     RandomExcursionsVariant
 0.963513    504/514     RandomExcursionsVariant
 0.639161    503/514     RandomExcursionsVariant
 0.683958    506/514     RandomExcursionsVariant
 0.263807    511/514     RandomExcursionsVariant
 0.154782    510/514     RandomExcursionsVariant
 0.836014    509/514     RandomExcursionsVariant
 0.626913    511/514     RandomExcursionsVariant
 0.226901    508/514     RandomExcursionsVariant
 0.941662    514/514     RandomExcursionsVariant
 0.424702    512/514     RandomExcursionsVariant
 0.176373    511/514     RandomExcursionsVariant
 0.379806    512/514     RandomExcursionsVariant
 0.461129    512/514     RandomExcursionsVariant
 0.986346    512/514     RandomExcursionsVariant
 0.101691    512/514     RandomExcursionsVariant
 0.916498    510/514     RandomExcursionsVariant
 0.375313    789/800     Serial
 0.849289    793/800     Serial
 0.742418    798/800     LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 783 for a
```

```
sample size = 800 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 502 for a sample size = 514 binary sequences.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Měření 1 - bit 1

Jeden test typu `NonOverlappingTemplate` selhal (Označen *).

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <Onyx-INT1aa_bit-1>
--------------------------------------
 P-VALUE  PROPORTION  STATISTICAL TEST
--------------------------------------
 0.099513    790/800     Frequency
 0.128688    790/800     BlockFrequency
 0.263904    791/800     CumulativeSums
 0.403403    789/800     CumulativeSums
 0.879383    792/800     Runs
 0.136304    788/800     LongestRun
 0.138267    790/800     Rank
 0.001372    788/800     FFT
 0.655334    792/800     NonOverlappingTemplate
 0.509162    791/800     NonOverlappingTemplate
 0.569766    793/800     NonOverlappingTemplate
 0.906407    791/800     NonOverlappingTemplate
 0.375313    796/800     NonOverlappingTemplate
 0.947557    793/800     NonOverlappingTemplate
 0.491947    794/800     NonOverlappingTemplate
 0.373203    791/800     NonOverlappingTemplate
 0.631914    794/800     NonOverlappingTemplate
 0.689019    793/800     NonOverlappingTemplate
 0.901265    796/800     NonOverlappingTemplate
 0.125927    792/800     NonOverlappingTemplate
 0.254088    792/800     NonOverlappingTemplate
 0.275709    791/800     NonOverlappingTemplate
 0.798139    788/800     NonOverlappingTemplate
 0.279152    793/800     NonOverlappingTemplate
 0.268917    795/800     NonOverlappingTemplate
 0.779188    792/800     NonOverlappingTemplate
 0.244549    795/800     NonOverlappingTemplate
```

0.499295	793/800		NonOverlappingTemplate
0.985226	791/800		NonOverlappingTemplate
0.190396	796/800		NonOverlappingTemplate
0.771953	794/800		NonOverlappingTemplate
0.008800	795/800		NonOverlappingTemplate
0.425817	797/800		NonOverlappingTemplate
0.899521	790/800		NonOverlappingTemplate
0.465415	792/800		NonOverlappingTemplate
0.847183	792/800		NonOverlappingTemplate
0.885045	781/800	*	NonOverlappingTemplate
0.129620	795/800		NonOverlappingTemplate
0.410055	795/800		NonOverlappingTemplate
0.463037	791/800		NonOverlappingTemplate
0.362765	794/800		NonOverlappingTemplate
0.719747	794/800		NonOverlappingTemplate
0.811993	793/800		NonOverlappingTemplate
0.082387	794/800		NonOverlappingTemplate
0.759756	794/800		NonOverlappingTemplate
0.747401	789/800		NonOverlappingTemplate
0.058984	791/800		NonOverlappingTemplate
0.650132	794/800		NonOverlappingTemplate
0.934318	792/800		NonOverlappingTemplate
0.401199	795/800		NonOverlappingTemplate
0.249284	791/800		NonOverlappingTemplate
0.865697	790/800		NonOverlappingTemplate
0.827722	792/800		NonOverlappingTemplate
0.948790	793/800		NonOverlappingTemplate
0.017009	788/800		NonOverlappingTemplate
0.534146	791/800		NonOverlappingTemplate
0.539193	793/800		NonOverlappingTemplate
0.064822	794/800		NonOverlappingTemplate
0.286131	797/800		NonOverlappingTemplate
0.016431	795/800		NonOverlappingTemplate
0.567201	792/800		NonOverlappingTemplate
0.392456	796/800		NonOverlappingTemplate
0.220448	795/800		NonOverlappingTemplate
0.058520	793/800		NonOverlappingTemplate
0.809707	795/800		NonOverlappingTemplate
0.689019	789/800		NonOverlappingTemplate
0.358641	794/800		NonOverlappingTemplate
0.569766	796/800		NonOverlappingTemplate
0.394631	792/800		NonOverlappingTemplate
0.712111	795/800		NonOverlappingTemplate
0.732389	793/800		NonOverlappingTemplate

```
0.336505    791/800     NonOverlappingTemplate
0.496841    787/800     NonOverlappingTemplate
0.305968    788/800     NonOverlappingTemplate
0.722284    790/800     NonOverlappingTemplate
0.148443    792/800     NonOverlappingTemplate
0.133404    792/800     NonOverlappingTemplate
0.580051    794/800     NonOverlappingTemplate
0.407831    791/800     NonOverlappingTemplate
0.390288    791/800     NonOverlappingTemplate
0.519103    793/800     NonOverlappingTemplate
0.524101    789/800     NonOverlappingTemplate
0.683857    792/800     NonOverlappingTemplate
0.098036    791/800     NonOverlappingTemplate
0.704442    792/800     NonOverlappingTemplate
0.536668    788/800     NonOverlappingTemplate
0.678686    788/800     NonOverlappingTemplate
0.074560    788/800     NonOverlappingTemplate
0.729870    794/800     NonOverlappingTemplate
0.434970    797/800     NonOverlappingTemplate
0.869673    792/800     NonOverlappingTemplate
0.021444    794/800     NonOverlappingTemplate
0.126842    793/800     NonOverlappingTemplate
0.978072    794/800     NonOverlappingTemplate
0.142264    797/800     NonOverlappingTemplate
0.182799    796/800     NonOverlappingTemplate
0.514124    785/800     NonOverlappingTemplate
0.873597    794/800     NonOverlappingTemplate
0.258961    796/800     NonOverlappingTemplate
0.928563    794/800     NonOverlappingTemplate
0.994250    793/800     NonOverlappingTemplate
0.221898    797/800     NonOverlappingTemplate
0.986336    788/800     NonOverlappingTemplate
0.091625    793/800     NonOverlappingTemplate
0.334538    794/800     NonOverlappingTemplate
0.371101    793/800     NonOverlappingTemplate
0.683857    793/800     NonOverlappingTemplate
0.262249    792/800     NonOverlappingTemplate
0.439585    790/800     NonOverlappingTemplate
0.451234    795/800     NonOverlappingTemplate
0.529115    791/800     NonOverlappingTemplate
0.403403    792/800     NonOverlappingTemplate
0.834308    787/800     NonOverlappingTemplate
0.304126    791/800     NonOverlappingTemplate
0.771953    797/800     NonOverlappingTemplate
```

0.044580	791/800	NonOverlappingTemplate
0.504219	794/800	NonOverlappingTemplate
0.665726	790/800	NonOverlappingTemplate
0.921005	791/800	NonOverlappingTemplate
0.719747	795/800	NonOverlappingTemplate
0.161478	793/800	NonOverlappingTemplate
0.021812	793/800	NonOverlappingTemplate
0.885045	794/800	NonOverlappingTemplate
0.827722	791/800	NonOverlappingTemplate
0.665726	794/800	NonOverlappingTemplate
0.686439	791/800	NonOverlappingTemplate
0.184047	787/800	NonOverlappingTemplate
0.906407	791/800	NonOverlappingTemplate
0.894201	793/800	NonOverlappingTemplate
0.090252	793/800	NonOverlappingTemplate
0.875539	792/800	NonOverlappingTemplate
0.037259	792/800	NonOverlappingTemplate
0.472584	797/800	NonOverlappingTemplate
0.534146	788/800	NonOverlappingTemplate
0.737414	793/800	NonOverlappingTemplate
0.048326	793/800	NonOverlappingTemplate
0.616305	794/800	NonOverlappingTemplate
0.551874	796/800	NonOverlappingTemplate
0.595549	792/800	NonOverlappingTemplate
0.233767	791/800	NonOverlappingTemplate
0.888750	793/800	NonOverlappingTemplate
0.247698	794/800	NonOverlappingTemplate
0.873597	794/800	NonOverlappingTemplate
0.239883	793/800	NonOverlappingTemplate
0.802793	795/800	NonOverlappingTemplate
0.432672	794/800	NonOverlappingTemplate
0.564639	789/800	NonOverlappingTemplate
0.954702	793/800	NonOverlappingTemplate
0.712111	793/800	NonOverlappingTemplate
0.324821	793/800	NonOverlappingTemplate
0.962280	790/800	NonOverlappingTemplate
0.781584	792/800	NonOverlappingTemplate
0.173054	793/800	NonOverlappingTemplate
0.975801	793/800	NonOverlappingTemplate
0.670915	792/800	NonOverlappingTemplate
0.629311	789/800	NonOverlappingTemplate
0.890582	787/800	OverlappingTemplate
0.489508	792/800	Universal
0.034313	794/800	ApproximateEntropy

```
 0.811993    476/482     RandomExcursions
 0.337162    474/482     RandomExcursions
 0.585209    475/482     RandomExcursions
 0.192984    479/482     RandomExcursions
 0.624107    481/482     RandomExcursions
 0.878746    478/482     RandomExcursions
 0.237831    475/482     RandomExcursions
 0.916811    475/482     RandomExcursions
 0.663130    479/482     RandomExcursionsVariant
 0.464622    478/482     RandomExcursionsVariant
 0.641458    479/482     RandomExcursionsVariant
 0.848588    477/482     RandomExcursionsVariant
 0.112322    477/482     RandomExcursionsVariant
 0.606781    479/482     RandomExcursionsVariant
 0.551026    476/482     RandomExcursionsVariant
 0.158872    480/482     RandomExcursionsVariant
 0.999735    478/482     RandomExcursionsVariant
 0.172262    474/482     RandomExcursionsVariant
 0.534146    475/482     RandomExcursionsVariant
 0.357274    475/482     RandomExcursionsVariant
 0.957319    474/482     RandomExcursionsVariant
 0.542566    475/482     RandomExcursionsVariant
 0.106930    476/482     RandomExcursionsVariant
 0.894201    473/482     RandomExcursionsVariant
 0.788728    473/482     RandomExcursionsVariant
 0.938915    471/482     RandomExcursionsVariant
 0.176657    788/800     Serial
 0.287895    792/800     Serial
 0.514124    790/800     LinearComplexity


- - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 783 for a
sample size = 800 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 470 for a sample size = 482 binary sequences.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Měření 1 - bit 2

Většina testů selhala (Označeny *).

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <Onyx-INT1aa_bit-2>
---------------------------------------
 P-VALUE  PROPORTION  STATISTICAL TEST
---------------------------------------
 0.000000 *    0/800  *  Frequency
 0.000000 *    0/800  *  BlockFrequency
 0.000000 *    0/800  *  CumulativeSums
 0.000000 *    0/800  *  CumulativeSums
 0.000000 *    0/800  *  Runs
 0.000000 *  245/800  *  LongestRun
 0.407831    793/800     Rank
 0.000000 *  635/800  *  FFT
 0.000000 *   10/800  *  NonOverlappingTemplate
 0.000000 *  120/800  *  NonOverlappingTemplate
 0.000000 *    6/800  *  NonOverlappingTemplate
 0.000000 *  326/800  *  NonOverlappingTemplate
 0.000000 *    4/800  *  NonOverlappingTemplate
 0.000000 *  149/800  *  NonOverlappingTemplate
 0.000000 *  257/800  *  NonOverlappingTemplate
 0.000000 *  687/800  *  NonOverlappingTemplate
 0.000000 *    1/800  *  NonOverlappingTemplate
 0.000000 *  117/800  *  NonOverlappingTemplate
 0.000000 *  138/800  *  NonOverlappingTemplate
 0.000000 *  757/800  *  NonOverlappingTemplate
 0.000000 *  175/800  *  NonOverlappingTemplate
 0.000000 *  718/800  *  NonOverlappingTemplate
 0.000000 *  757/800  *  NonOverlappingTemplate
 0.000000 *  526/800  *  NonOverlappingTemplate
 0.000000 *  132/800  *  NonOverlappingTemplate
 0.000000 *  197/800  *  NonOverlappingTemplate
 0.000000 *  770/800  *  NonOverlappingTemplate
 0.000000 *  179/800  *  NonOverlappingTemplate
 0.000000 *  703/800  *  NonOverlappingTemplate
 0.000000 *  729/800  *  NonOverlappingTemplate
 0.000000 *  705/800  *  NonOverlappingTemplate
 0.000000 *  753/800  *  NonOverlappingTemplate
 0.000000 *  770/800  *  NonOverlappingTemplate
 0.000000 *  584/800  *  NonOverlappingTemplate
 0.000000 *  749/800  *  NonOverlappingTemplate
 0.000000 *  675/800  *  NonOverlappingTemplate
 0.000000 *  654/800  *  NonOverlappingTemplate
```

```
0.000000 *  150/800  *  NonOverlappingTemplate
0.000000 *   94/800  *  NonOverlappingTemplate
0.000000 *  213/800  *  NonOverlappingTemplate
0.000000 *  739/800  *  NonOverlappingTemplate
0.000000 *  594/800  *  NonOverlappingTemplate
0.000000 *  730/800  *  NonOverlappingTemplate
0.000000 *  743/800  *  NonOverlappingTemplate
0.000000 *  701/800  *  NonOverlappingTemplate
0.000000 *  563/800  *  NonOverlappingTemplate
0.000000 *  741/800  *  NonOverlappingTemplate
0.000000 *  767/800  *  NonOverlappingTemplate
0.000000 *  750/800  *  NonOverlappingTemplate
0.000000 *  115/800  *  NonOverlappingTemplate
0.000000 *  740/800  *  NonOverlappingTemplate
0.000000 *  666/800  *  NonOverlappingTemplate
0.000000 *  772/800  *  NonOverlappingTemplate
0.000000 *  739/800  *  NonOverlappingTemplate
0.000000 *  150/800  *  NonOverlappingTemplate
0.000000 *  774/800  *  NonOverlappingTemplate
0.000000 *  125/800  *  NonOverlappingTemplate
0.000000 *  144/800  *  NonOverlappingTemplate
0.000000 *  131/800  *  NonOverlappingTemplate
0.000000 *   14/800  *  NonOverlappingTemplate
0.000000 *  149/800  *  NonOverlappingTemplate
0.000000 *  746/800  *  NonOverlappingTemplate
0.000000 *  665/800  *  NonOverlappingTemplate
0.000000 *  734/800  *  NonOverlappingTemplate
0.000000 *  715/800  *  NonOverlappingTemplate
0.000000 *  722/800  *  NonOverlappingTemplate
0.000000 *  753/800  *  NonOverlappingTemplate
0.000000 *  103/800  *  NonOverlappingTemplate
0.000000 *  737/800  *  NonOverlappingTemplate
0.000000 *  727/800  *  NonOverlappingTemplate
0.000000 *  717/800  *  NonOverlappingTemplate
0.000000 *  114/800  *  NonOverlappingTemplate
0.000000 *  766/800  *  NonOverlappingTemplate
0.000000 *  142/800  *  NonOverlappingTemplate
0.000000 *  125/800  *  NonOverlappingTemplate
0.000000 *    8/800  *  NonOverlappingTemplate
0.000000 *  690/800  *  NonOverlappingTemplate
0.000000 *  153/800  *  NonOverlappingTemplate
0.000000 *  167/800  *  NonOverlappingTemplate
0.000000 *   11/800  *  NonOverlappingTemplate
0.000000 *   12/800  *  NonOverlappingTemplate
```

```
0.000000 *    1/800  *  NonOverlappingTemplate
0.000000 *   10/800  *  NonOverlappingTemplate
0.000000 *    6/800  *  NonOverlappingTemplate
0.000000 *    6/800  *  NonOverlappingTemplate
0.000000 *  173/800  *  NonOverlappingTemplate
0.000000 *  240/800  *  NonOverlappingTemplate
0.000000 *  759/800  *  NonOverlappingTemplate
0.000000 *    8/800  *  NonOverlappingTemplate
0.000000 *  194/800  *  NonOverlappingTemplate
0.000000 *  167/800  *  NonOverlappingTemplate
0.000000 *  753/800  *  NonOverlappingTemplate
0.000000 *  137/800  *  NonOverlappingTemplate
0.000000 *  572/800  *  NonOverlappingTemplate
0.000000 *  767/800  *  NonOverlappingTemplate
0.000000 *  761/800  *  NonOverlappingTemplate
0.000000 *  227/800  *  NonOverlappingTemplate
0.000000 *  622/800  *  NonOverlappingTemplate
0.000000 *  706/800  *  NonOverlappingTemplate
0.000000 *  767/800  *  NonOverlappingTemplate
0.000000 *  760/800  *  NonOverlappingTemplate
0.000000 *  757/800  *  NonOverlappingTemplate
0.000000 *  668/800  *  NonOverlappingTemplate
0.000000 *  131/800  *  NonOverlappingTemplate
0.000000 *  113/800  *  NonOverlappingTemplate
0.000000 *  109/800  *  NonOverlappingTemplate
0.000000 *  117/800  *  NonOverlappingTemplate
0.000000 *  148/800  *  NonOverlappingTemplate
0.000000 *  717/800  *  NonOverlappingTemplate
0.000000 *  129/800  *  NonOverlappingTemplate
0.000000 *  581/800  *  NonOverlappingTemplate
0.000000 *  695/800  *  NonOverlappingTemplate
0.000000 *  754/800  *  NonOverlappingTemplate
0.000000 *  763/800  *  NonOverlappingTemplate
0.000000 *  172/800  *  NonOverlappingTemplate
0.000000 *  608/800  *  NonOverlappingTemplate
0.000000 *  697/800  *  NonOverlappingTemplate
0.000000 *  730/800  *  NonOverlappingTemplate
0.000000 *  720/800  *  NonOverlappingTemplate
0.000000 *  769/800  *  NonOverlappingTemplate
0.000000 *  712/800  *  NonOverlappingTemplate
0.000000 *  774/800  *  NonOverlappingTemplate
0.000000 *  761/800  *  NonOverlappingTemplate
0.000000 *  639/800  *  NonOverlappingTemplate
0.000000 *  152/800  *  NonOverlappingTemplate
```

```
0.000000 *  154/800  *  NonOverlappingTemplate
0.000000 *  333/800  *  NonOverlappingTemplate
0.000000 *  755/800  *  NonOverlappingTemplate
0.000000 *  761/800  *  NonOverlappingTemplate
0.000000 *  645/800  *  NonOverlappingTemplate
0.000000 *  752/800  *  NonOverlappingTemplate
0.000000 *  743/800  *  NonOverlappingTemplate
0.000000 *  680/800  *  NonOverlappingTemplate
0.000000 *  756/800  *  NonOverlappingTemplate
0.000000 *  726/800  *  NonOverlappingTemplate
0.000000 *  740/800  *  NonOverlappingTemplate
0.000000 *  171/800  *  NonOverlappingTemplate
0.000000 *  599/800  *  NonOverlappingTemplate
0.000000 *  139/800  *  NonOverlappingTemplate
0.000000 *  106/800  *  NonOverlappingTemplate
0.000000 *  685/800  *  NonOverlappingTemplate
0.000000 *  728/800  *  NonOverlappingTemplate
0.000000 *  736/800  *  NonOverlappingTemplate
0.000000 *  136/800  *  NonOverlappingTemplate
0.000000 *  711/800  *  NonOverlappingTemplate
0.000000 *  103/800  *  NonOverlappingTemplate
0.000000 *  146/800  *  NonOverlappingTemplate
0.000000 *    7/800  *  NonOverlappingTemplate
0.000000 *  555/800  *  NonOverlappingTemplate
0.000000 *  109/800  *  NonOverlappingTemplate
0.000000 *  103/800  *  NonOverlappingTemplate
0.000000 *   22/800  *  NonOverlappingTemplate
0.000000 *  147/800  *  NonOverlappingTemplate
0.000000 *    7/800  *  NonOverlappingTemplate
0.000000 *   13/800  *  NonOverlappingTemplate
0.000000 *    1/800  *  NonOverlappingTemplate
0.000000 *   10/800  *  OverlappingTemplate
0.000000 *  417/800  *  Universal
0.000000 *    0/800  *  ApproximateEntropy
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursions
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
```

```
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
   ----     ------     RandomExcursionsVariant
 0.000000 *   31/800  *  Serial
 0.549331    793/800     Serial
 0.559523    797/800     LinearComplexity


- - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 783 for a
sample size = 800 binary sequences.

The minimum pass rate for the random excursion (variant) test is undefined.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Příloha B

Seznam použitých zkratek

AES Advanced Encryption Standard

CSPRNG Cryptographically Secure Pseudo-Random number Generator

HAL Hardware Abstraction Layer

HSI High Speed Internal

LCG Linear Congruential Generator

LFSR Linear Feedback Shift Register

LSI Low Speed Internal

LSE Low Speed External

NIST National Institute of Standards and Technology

PLL Phase Locked Loop

PRNG Pseudo Random Number Generator

RAM Random Access Memory

RNG Random Number Generator

SHA Secure Hash Algorithm

TERO Transient Effect Ring Oscillator

TRNG True Random Number Generator

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `app` ..... programy pro Windows a soubor k nahrání do přípravku
- `measurements` ..... vzorky naměřených dat
- `src` ..... zdrojové kódy
  - `C_programs` ..... zdrojové kódy pomocných programů
  - `Keil_project` ..... projekt pro Keil ARM
  - `MSVS_project` .. projekt pro MSVS programu pro čtení ze sériové linky
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `stats` ..... výsledky statistických testů NIST
- `text` ..... text práce
  - `BP_Zach_Tomas_2018.pdf` ..... text práce ve formátu PDF