VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

EFEKTIVNÍ GENERÁTOR NÁHODNÝCH ČÍSEL V NÍZKO-VÝKONOVÝCH ZAŘÍZENÍCH

EFFECTIVE RANDOM NUMBER GENERATOR FOR LIMITED DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Tomáš Michálek
AUTHOR

VEDOUCÍ PRÁCE — Ing. Radek Fujdiak
SUPERVISOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Tomáš Michálek ***ID:*** 134559
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Efektivní generátor náhodných čísel v nízko-výkonových zařízení

POKYNY PRO VYPRACOVÁNÍ:

Student vypracuje teoretický úvod do problematiky náhodných čísel a jejich generování pro kryptografické účely. Následně se zaměří na oblast paměťově a výkonově limitovaných zařízení s bližším popisem vývojového kitu TI MSP430f5438A. Pro tento vývojový kit bude zpracován přehled veškerých dostupných náhodných generátorů a možnosti generování náhodných čísel. Z dostupných generátorů budou implementovány vybrané čtyři. Poté student navrhne a vytvoří jeden vlastní náhodných generátor založený na vhodně vybraném jevu tak, aby byl rychlostně i paměťově efektivní. Všech pět generátorů student následně otestuje z pohledu časové i paměťové náročnosti a vzájemně je porovná.

DOPORUČENÁ LITERATURA:

[1] Texas Instruments. „MSP430F543xA, MSP430F541xA Mixed-Signal Microcontrollers“. SLAS655E, 2015.

[2] Randomness and Integrity Services Ltd. „True Random Number Service“. [ONLINE] Dostupné z: www.random.org

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: Ing. Radek Fujdiak
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato práce řeší problém generování náhodných čísel na nízko-výkonových zařízeních. Autor popisuje možné způsoby generování a implementuje vybrané generátory (pseudo)náhodných čísel na zařízení MSP430F5438A. 4 generátory byly doplněny o vylepšení jednoho z nich a dále byl vytvořen generátor nový, využívající jevu změny teploty v okolí. Pro každý generátor byla vygenerována testovací sekvence a tyto sekvence byly otestovány sadou testů Dieharder, STS-NIST a vizuálním testem. Výstupem práce je funkční implementace generátorů, jejich otestování statistickými metodami a srovnání mezi sebou.

ABSTRACT

This thesis solves the problem of generating random numbers on low-power devices. Author describes possible ways of generating and implements selected generators of (pseudo)random numbers on MSP430F5438A. 4 generators were added by the enhancement of one of them and a new generator was created, using the phenomenon of temperature change in the surroundings. For each generator, test sequences were generated and these sequences were tested by the Dieharder, STS-NIST, and Visual Test. The output of the thesis is the functional implementation of the generators, their testing by statistical methods and their comparison between each other.

Klíčová slova:

náhodná čísla, pravděpodobnost, testovací baterie, NIST, Dieharder, A/D převodník, nízko-výkonová zařízení, MSP430F5438A, generátor náhodných čísel

Keywords:

random numbers, probability, test suite, NIST, Dieharder, A/D converter, limited devices, MSP430F5438A, random number generator

MICHÁLEK, T. *Efektivní generátor náhodných čísel v nízko-výkonových zařízeních.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 66 s. Vedoucí diplomové práce Ing. Radek Fujdiak.

Prohlašuji, že svou diplomovou práci na téma „Efektivní generátor náhodných čísel v nízko-výkonových zařízeních “ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …… .....

podpis autora

Poděkování

Nejprve bych rád srdečně poděkoval svému vedoucímu práce, Ing. Radku Fujdiakovi, za neustálý pozitivní přístup a užitečné podněty. Dále si velmi cením konzultací, poskytnutí studijních materiálů a společné probírání tématu. Taktéž tímto děkuji rodině za shovívavost a své přítelkyni za ohromnou podporu.

V Brně dne …….. .....

podpis autora

OBSAH

ÚVOD

Náhodná čísla jsou výsledkem náhodného pokusu, jehož výsledek není možné dopředu určit. Pokud náhodná čísla poskládáme za sebe do sekvence, nebudou se v této sekvenci objevovat žádné vzory, ani se jednotlivé úseky této sekvence nebudou opakovat. Náhodná čísla lze generovat pomocí zařízení, založených na fyzikálním náhodném jevu, ve kterých se daná událost stane s nějakou pravděpodobností, anebo pomocí matematických algoritmů. Z fyzikálních jevů je možné ke generování náhodných čísel využít například ruletu, hod kostkou, rozpad radioaktivní látky, atmosférický šum a další. Algoritmy, které generují náhodná čísla, mohou být založeny na jednocestných matematických funkcích, na logických operacích nebo bitových posunech. Tyto matematické algoritmy však negenerují skutečně náhodné číslo, ale číslo tzv. pseudonáhodné. Náhodná a pseudonáhodná čísla dále popisuje první kapitola. Tato práce se bude hlouběji zabývat generováním náhodných čísel a jejich testováním.

Věda, která studuje různé způsoby utajení přenášené zprávy či uchovávaných dat, se nazývá kryptografie neboli šifrování. Kryptografie se zabývá ochranou těchto dat před pozměněním či zneužitím a také potvrzením identity komunikující entity Zejména v dnešní době informačních technologií je stále důležitější data ochránit před cizí entitou zamezením přístupu nebo jejich pozměněním do takového tvaru, aby jej mohla přečíst jen povolaná osoba. Algoritmus, který tímto způsobem data pozměňuje, se nazývá šifra. Některé algoritmy používají ke své funkci náhodně generovaná čísla (takovým algoritmem je například protokol pro ustanovení šifrovacích klíčů Diffie-Hellman). Pokud bychom chtěli zašifrovanou zprávu rozluštit bez znalosti šifrovacího klíče, jednalo by se o kryptoanalýzu, která je tedy opakem kryptografie. Kryptografie a kryptoanalýza se dohromady nazývají kryptologie.

Součástí diplomové práce je implementace a analýza vybraných generátorů náhodných čísel a návrh a popis vlastního generátoru. Vzhledem k dnešnímu rozmachu IoT (internet of things = internet věcí) a nutnosti zabezpečit přenos dat, jsou generátory implementovány na výkonově omezeném zařízení, které má v praxi bateriový zdroj energie, a proto je důležité mít nejen kvalitní generátor náhodných čísel, ale i energeticky efektivní generátor. Součástí práce je rovněž návrh a analýza vlastního generátoru a porovnání tohoto generátoru s ostatními generátory, které budou v rámci práce implementovány na zapůjčeném výkonově omezeném zařízení.

1 NÁHODNÉ JEVY A ČÍSLA

Náhodný jev je výsledkem náhodného pokusu a jeho charakteristickým rysem je, že může, ale nemusí nastat. Pokud míru jeho výskytu vyjádříme číselně, dostaneme pravděpodobnost výskytu onoho jevu, přičemž 0 je případ, kdy jev nenastane, a naopak při pravděpodobnosti 1 jev nastane. Pravděpodobnost náhodného jevu P tedy nabývá hodnot z intervalu <0,1>, nebo též 0-100 %. Mnohé náhodné jevy se dají využít ke generování náhodných čísel a příkladem takového náhodného jevu může být hod kostkou, hod mincí, údery na klávesnici, pohyb myší, atmosférický šum či změna teploty. [1]

1.1 Náhodná čísla a pravděpodobnost

Náhodné číslo vznikne jako důsledek náhodného fyzikálního procesu. Poskládáme-li výsledky takovýchto pokusů do posloupnosti, dostaneme náhodnou sekvenci, přičemž tato sekvence postrádá jakýkoliv vzor nebo předvídatelnost. Pro aplikaci v praxi, například v kryptografii, se v dnešní době ke generování náhodných čísel využívá různých generátorů, které generují náhodná čísla pomocí fyzikálních náhodných jevů, jako jsou například měření elektromagnetického šumu nebo rozpad radioaktivní látky. Náhodná čísla hrají svou roli v komunikaci, v kryptografii, v hazardních hrách, simulacích a ve spoustě dalších oblastí.

Ke generování náhodných čísel jsou zvlášť vhodné jevy z kvantové fyziky, jelikož už z podstaty kvantové fyziky vyplývá, že nemůžeme předpovědět výsledek takového jevu. Pokud jako náhodný jev určíme rozpad radioaktivní látky, nemůžeme dopředu předpovědět, kdy přesně k rozpadu dojde. V čase t provedeme měření, a buďto k rozpadu látky došlo nebo ne. Dalším příkladem z kvantové fyziky je průchod elektronu dvěma štěrbinami, kdy je pravděpodobnost 50 %, že elektron projde jednou z děr a není možné zjistit, jakou z děr elektron příště projde. [2][3][4][5][6]

1.2 Entropie aneb měření náhodnosti

S šifrovacími klíči a náhodnými čísly se velmi často setkáváme v oblasti počítačové bezpečnosti, přičemž na kvalitě jejich náhodnosti záleží stejně, jako na kvalitě vlastních šifer. Znát míru náhodnosti používaného zdroje je nutné zejména při generování šifrovacích klíčů. Jestliže generátor náhodných bitů nemá dostatečnou kvalitu, může se stát, že vygenerovaných 128 bitů šifrovacího klíče má pouze 40bitovou informační hodnotu (neurčitost), čímž se degraduje kvalita šifrování. Takovou míru neurčitosti vyjadřuje entropie S. Entropie určuje skutečné množství obsažené informace a měří se v bitech. [7]

Čím je entropie vyšší, tím více bude systém neuspořádán a bude se jevit jako chaotický. Entropie vyjadřuje míru neurčitosti systému. V případě určitých stavů, které mohou nastat, je definována následovně:

$$S = -k \sum P_i ln P_i$$

Rovnice 1 - Míra entropie

kde P_i je pravděpodobnost i-tého stavu. Konstanta k odpovídá volbě jednotek, ve kterých entropii S měříme. V jednotkách bit bude konstanta k=1/ln(2), rovnice pak bude vypadat takto:

$$S = -\sum P_i log_2 P_i$$

Rovnice 2 - Informační entropie

1.3 Pseudonáhodná čísla

Častokrát se v problematice náhodných čísel setkáme s pojmem pseudonáhodná čísla. To jsou čísla, která nejsou opravdu náhodná, ale jen jako náhodná vypadají. Po určité době se však začne jejich sekvence opakovat, či lze nalézt podobnosti v jejich výskytu. Pseudonáhodná čísla vznikají snahou generovat náhodné číslo na počítači, což je deterministický automat. Z teorie determinismu vyplývá, že stejný vstup na stejném stroji bude mít stejný výsledek. Pokud tedy vygenerujeme náhodné číslo za určitých podmínek, můžeme totéž číslo vygenerovat opakovaně za dodržení oněch podmínek. Pseudonáhodná čísla se používají například pro generování testovacích dat při ladění programů nebo v počítačových hrách.

Pseudonáhodná sekvence vznikne z počátečního čísla a algoritmus z tohoto počátku generuje pomocí nějakého vzorce další čísla. Dokud tedy neznáme počáteční číslo (v odborné literatuře označováno jako „seed“) a použitý algoritmus, nezjistíme další generované čísla. Naopak podle toho, že známe posloupnosti vygenerovaných čísel, by nemělo být zjistitelné počáteční číslo. [8][9]

1.4 Generátory náhodných čísel

Kvalitní generátor náhodných čísel je pro dnešní využití výpočetní techniky a zejména kryptografie zásadní. Generátor náhodných čísel je výpočetní zařízení, kde vstupem je nějaké zadání (například délka výstupních dat a případně zdroj dat) a výstupem je sekvence náhodných čísel. Generování náhodných čísel je jev, který je možné libovolně opakovat, přičemž výsledky nejsou jednoznačně určeny vstupními podmínkami, a jednotlivé výsledky není možné předpovědět a ani nejsou závislé na předchozích pokusech.

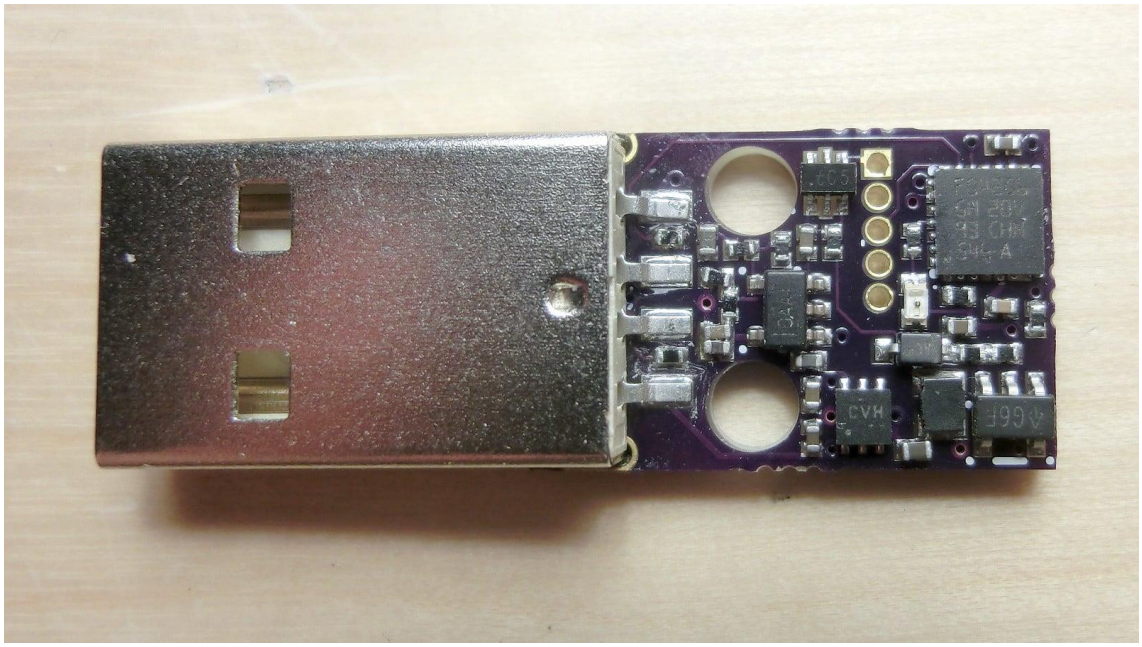
Existují dva typy generátorů náhodných čísel. Hardwarový generátor, který využívá různé fyzikální jevy a často může používat připojené periferie (Geigerův počítač, A/D převodník, oscilátor a jiné) a softwarový generátor, který používá ke generování náhodných čísel matematické algoritmy.

1.4.1 Hardwarový generátor

Tam, kde je potřeba „opravdová náhoda“ nebo kde podmínky umožňují použití samostatného zařízení, tam se používá hardwarový generátor náhodných čísel, který se v praxi označuje jako TRGN (True Random Generator). Hardwarový generátor využívá ke své funkci různé fyzikální jevy, které nastávají náhodně. Ke generování náhodných čísel se dají využít různé druhy náhodných jevů. Náhodným jevem vhodným pro hardwarový generátor může být již zmiňovaný rozpad radioaktivních látek či šum. Kvantově založený hardwarový generátor náhodných čísel se typicky skládá z převodníku převádějící některé aspekty fyzikálních jevů na elektrický signál, zesilovače a dalších elektronických obvodů, aby byl výstup snímače přenesen do makroskopické oblasti a nějakého A/D převodníku pro konverzi analogového výstupu do digitální formy. Samotný A/D převodník lze použít jako generátor náhodných čísel, pokud jej necháme snímat elektromagnetický šum z okolí. [10]

Náhodný jev je i pohyb lávy v lávové lampě. Toho využili výzkumníci z firmy Silicon Graphics, aby nasnímali pohyb lávy a ze snímku extrahovali data, která pak použili jako vstup pseudonáhodného generátoru. Princip je popsaný v US Patent 5,732,138.

Hardwarové generátory náhodných čísel se tak liší od generátorů pseudonáhodných čísel, které se běžné používají ve většině počítačů. Hardwarové generátory se využívají například v bezpečnostních aplikacích, jako jsou produkce náhodných klíčů pro vojenské a obchodní šifrovací systémy, se používají hardwarové generátory. Výstup z hardwarového generátoru může být použit tak jak je, anebo může být použit pro pseudonáhodný generátor. [11][12]



Obrázek 1 - Vyobrazen hardwarový generátor náhodných čísel ChaosKey [13]

Externí hardwarové generátory

Hardwarové generátory využívající periferie připojené k výpočetnímu zařízení se nazývají externí generátory. Může se jednat o samostatný obvod, kde se měří napětí na nějakém prvku nebo Geigerův počítač, který snímá úroveň záření při rozpadu

radioaktivního prvku. Nevýhodou těchto generátorů je vyšší spotřeba energie, která je daná potřebou napájet externí obvody. Taktéž, pokud k výpočetnímu zařízení připojíme další periferie za účelem získávání entropie a vytvořením kvalitního generátoru náhodných čísel, musíme uvážit cenu výsledného zařízení. Proto je někdy vhodné použít jiný způsob generování a využít jen interní součásti našeho zařízení. [14]

Interní hardwarové generátory

Interní hardwarové generátory náhodných čísel mají oproti externím generátorům výhodu v menší spotřebě energie, ale za to jsou omezené jen na vnitřní obvody výpočetního zařízení. Mohou tedy použít jen takový způsob generování čísel, jaký jim dovolí komponenty, které obsahují.

1.4.2 Softwarový generátor

Druhým typem generátoru je softwarový generátor označovaný jako PRGN (pseudorandom number generator). Jak už anglický název napovídá, bývají tyto generátory označovány také jako generátory pseudonáhodných čísel. Softwarové generátory používají jako vstupní hodnotu inicializační vektor seed, který je algoritmem následně použitý a vytváří se z něj sekvence pseudonahodných čísel. Jako seed lze použít vystup TRGN a pak bude vytvořena sekvence kvalitnější. Pokud je nutné, aby měl výstup tohoto generátoru vlastnosti náhodného generátoru, musí být seed náhodné číslo. PRGN tedy ke své funkčnosti potřebuje takový generátor náhodných čísel, který mu bude dodávat náhodná čísla jakožto vstupní hodnoty.

Při provádění některých testů náhodnosti se stává, ze PRGN vykazuje větší náhodnost než opravdu náhodný generátor. Pokud je pseudonáhodná sekvence správně sestavena, každá hodnota je vstupem pro algoritmus a výstupem je následující hodnota, která opět slouží jako vstup pro další kolo algoritmu atd. Série těchto transformací může eliminovat statistické auto-korelace mezi vstupem a výstupem. Může se tedy stát, že výstup PRGN bude vykazovat větší náhodnost než výstup TRGN. [8][15]

Softwarové generátory jsou vhodné zejména pro případy, kdy potřebujeme v poměrně krátké době získat mnoho náhodných hodnot. Jejich typické využití je v simulačních programech. Příkladem softwarového generátoru náhodných čísel je lineární kongruentní generátor.

$$x_{i+1} = (ax_i + c) mod\ m$$

Rovnice 3 - Rovnice lineárního kongruentního generátoru

2 TESTOVÁNÍ NÁHODNOSTI

Abychom zjistili, zdali je vygenerovaná sekvence náhodná nebo ne, používáme testy náhodnosti. Testy náhodnosti analyzují sekvence náhodných čísel (např. o velikosti deset milionů bitů), zjišťují, zdali se v testované sekvenci objevují určité vzory, a nakonec vyhodnotí náhodnost sekvence. Pokud testujeme TRGN, můžeme dopředu odhadnout výsledek testování s nějakou pravděpodobností, a posléze tuto hypotézu ověřit testováním.

Statistických testů může být nekonečné množství a každý zjišťuje přítomnost nebo absenci jiného vzoru, který, pokud je detekován, značí, že je sekvence nenáhodná. Díky existenci několika různých testů se nepovažuje žádný set testu za konečný a výsledky těchto testů se musí vykládat s obezřetností, a ne se 100 % jistotou. Doporučuje se dělat různorodé testy i opakovaně.

2.1 Testování nulovou hypotézou

Statistické testování hypotézou je proces, na jehož konci dojdeme k určité hypotéze, vyhodnocení. Při testování nulovou hypotézou přijmeme nulovou hypotézu H0, která pro účely testu bude znamenat, že testovaná sekvence je náhodná. Po následném testování se může stát, že testy nulovou hypotézu vyvrátí a my tak zamítneme hypotézu H0 a přijmeme její opak, alternativní hypotézu Ha, která pro účely testování bude znamenat, že je sekvence nenáhodná. Následující tabulka zobrazuje možné závěry testování.

Situace	Závěr	
	Přijetí H0	Přijetí Ha (odmítnutí H0)
Sekvence je náhodná (H0 je pravda)	správně	chyba 1
Sekvence není náhodná (Ha je pravda)	chyba 2	správně

Tabulka 1 - Možné závěry testování nulovou hypotézou

S velmi malou pravděpodobností se může stát, že testovaná sekvence je náhodná, ale na základě vyhodnocení testu se rozhodneme odmítnout hypotézu H0 a vznikne stav označený „chyba 1“. Pokud přijmeme H0 (odmítneme Ha) ale ve skutečnosti je sekvence nenáhodná, vznikne stav „chyba 2“. Pokud bude sekvence náhodná a přijmeme H0 nebo bude sekvence nenáhodná a přijmeme Ha, bude to správně.

Pravděpodobnost chyby 1 je často nazývána hladinou významnosti testu (level of significance). Tato hladina může být nastavena před samotným testováním a je značena jako α-. Pro testování tedy platí, že α je pravděpodobnost vyhodnocení náhodné sekvence jako nenáhodné. Pravděpodobnost chyby 2 je značena jako β. Pro účely

testování β znamená, že sekvence je vyhodnocena jako náhodná, ale ve skutečnosti tomu tak není (sekvence je nenáhodná). Tedy generátor generuje sekvenci, která se pouze jeví jako náhodná a má některé vlastnosti, které se očekávají od náhodné sekvence, ale náhodná není. [8]

Při testování generátoru bude nulová hypotéza taková, že „tento generátor je perfektním generátorem náhodných čísel a sekvence, kterou vygeneruje je skutečně náhodná a má vlastnosti, které se očekávají od náhodné sekvence“.

Při testování nulovou hypotézou používáme tzv. p-value (probability value). P-value a testování nulovou hypotézou se uplatňuje v několika oborech, například ve finančnictví, psychologii, sociologii a statistice. Můžeme se setkat s definicí, že p-value je vyjádření síly důkazů působících proti nulové hypotéze. Pokud bude p-value menší než hodnota α, znamená to odmítnutí nulové hypotézy. Nicméně rozložení hodnot p-value v intervalu <0,1> by mělo mít uniformní rozdělení.[17][18]

2.2 Statistické testy NIST

Význam statistických testů náhodnosti je natolik důležitý, že tyto testy byly standardizovány v normě amerického úřadu pro standardy a technologie NIST (Národní institut standardů a technologie - National Institute of Standards and Technology). Jedná se o laboratoř měřících standardů ministerstva obchodu USA, která byla založena roku 1901 za účelem podpory a inovaci průmyslové konkurenceschopnosti USA zlepšováním vědeckých měření, standardů a technologií. To tedy znamená, že NIST vydává standardy a doporučení týkajících se nových technologií od nano zařízení, až po konstrukce mrakodrapů. Mimo jiné vydává i standardy a doporučení pro šifrování nebo testovací metody pro ověření náhodnosti čísel.[8]

Sada testů NIST	
1	Frekvenční test (test rozdělení nul a jedniček)
2	Rozdělení jedniček a nul uvnitř bloků a stejné dané délky
3	Test runů (tj. test počtů a délek úseků typu 1111..11 nebo 0..000)
4	Test nejdelšího runu v bloku (délka nejdelšího řetězce 1..11 při rozdělení posloupnosti do bloků stejné dané délky)
5	Test na hodnost binární matice
6	Spektrální test (diskrétní Fourierova transformace)
7	Nepřekrývající se vzorové řetězce
8	Překrývající se vzorové řetězce
9	Mauerův univerzální statistický test

10	Test lineární složitosti
11	Test sérií
12	Test přibližné entropie
13	Test kumulativních součtů
14	Test náhodné procházky
15	Variantní test náhodné procházky

Tabulka 2 - Seznam sady testů NIST

[19]

2.3 ENT

Statistický test ENT (pseudorandom number sequence test) používá pro kontrolu náhodnosti 6 samostatných testů a jeho použití je především ve statistice, kryptografii, kompresních algoritmech a jiných aplikacích. Testováním metodou ENT jsme schopni se dovědět míru entropie, možnost komprese, chí kvadrát test, aritmetický průměr, hodnotu Monte Carlo pro π a koeficient seriální korelace. [20][21][22]

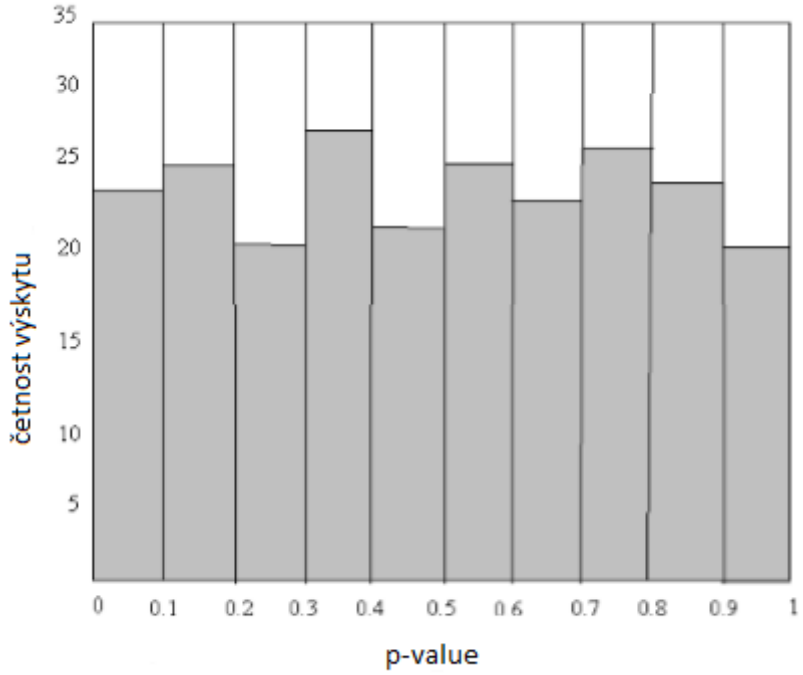
ENT tests	
	Entropy = 7.999825 bits per byte.
	Optimum compression would reduce the size of this 1036003 byte file by 0 percent.
	Chi square distribution for 1036003 samples is 251.25, and randomly would exceed this value 55.46 percent of the times.
	Arithmetic mean value of data bytes is 127.4824 (127.5 = random).
	Monte Carlo value for Pi is 3.141283511 (error 0.01 percent).
	Serial correlation coefficient is 0.000858 (totally uncorrelated = 0.0).

Tabulka 3 - Příklad výsledků sady testů ENT [23]

2.4 Diehard

Jedná se o sérii patnácti statistických testů pro měření kvality náhodných generátorů. Testy byly vytvořeny G. Marsagliem a poprvé publikovány v roce 1995 a uvolněny k akademickému a výzkumnému použití pro veřejnost. Série testů zjišťuje vlastnosti vygenerovaných čísel a výstupem těchto testů je hodnota p-value. Pokud sekvence projde Diehard testem, jsou hodnoty p-value rovnoměrně rozděleny v intervalu <0,1>,

viz obrázek níže. Pro nejpřesnější výsledky se doporučuje použít všech 15 testů.



Obrázek 2 - Histogram hodnot p-value

[25]

Sada testů DIEHARD	
1	Birthday Spacings
2	Overlapping Permutations
3	Ranks of 31x31 and 32x32 Matrices
4	Ranks of 6x8 Matrices
5	Monkey Tests on 20-bit Words
6	Monkey Tests OPSO, OQSO, DNA
7	Count the 1's in a Stream of Bytes
8	Count the 1's in Specific Bytes
9	Parking Lot Test
10	Minimum Distance Test
11	Random Sphere Test
12	The Squeeze Test
13	Overlapping Sums Test
14	Runs Test

15	The Craps Test

Tabulka 4 - Seznam sady testů DIEHARD

2.5 Dieharder

Dieharder sada testů rozšiřující původní sadu Diehard. Za projektem stojí Robert G. Brown z Duke University Physics Department. Kromě původních testů z této sady dále obsahuje Monobit test, Runs test a Seriál test ze sady Statistical Test Suite, která byla vyvinuta organizací NIST a další doplňující testy.[17]

#=======================#		
dieharder version 3.31.1		
Copyright 2003 Robert G. Brow		
Installed dieharder tests:		
Test Number	Test Name	Test Reliability
-d 0	Diehard Birthdays Test	Good
-d 1	Diehard OPERM5 Test	Good
-d 2	Diehard 32x32 Binary Rank Test	Good
-d 3	Diehard 6x8 Binary Rank Test	Good
-d 4	Diehard Bitstream Test	Good
-d 5	Diehard OPSO	Suspect
-d 6	Diehard OQSO Test	Suspect
-d 7	Diehard DNA Test	Suspect
-d 8	Diehard Count the 1s (stream) Test	Good
-d 9	Diehard Count the 1s Test (byte)	Good
-d 10	Diehard Parking Lot Test	Good
-d 11	Diehard Minimum Distance (2d Circle) Test	Good
-d 12	Diehard 3d Sphere (Minimum Distance) Test	Good
-d 13	Diehard Squeeze Test	Good
-d 14	Diehard Sums Test	Do Not Use
-d 15	Diehard Runs Test	Good
-d 16	Diehard Craps Test	Good
-d 17	Marsaglia and Tsang GCD Test	Good
-d 100	STS Monobit Test	Good
-d 101	STS Runs Test	Good
-d 102	STS Serial Test (Generalized)	Good
-d 200	RGB Bit Distribution Test	Good
-d 201	RGB Generalized Minimum Distance Test	Good
-d 202	RGB Permutations Test	Good
-d 203	RGB Lagged Sum Test	Good

-d 204	RGB Kolmogorov-Smirnov Test Test	Good
-d 205	Byte Distribution	Good
-d 206	DAB DCT	Good
-d 207	DAB Fill Tree Test	Good
-d 208	DAB Fill Tree 2 Test	Good
-d 209	DAB Monobit 2 Test	Good

Tabulka 5 - Seznam sady testů Dieharder

Popis jednotlivých testů přesahuje rozsah této práce a vydal by na celou samostatnou práci, a proto je zde nebudu rozebírat. Každý test vygeneruje výslednou hodnotu p-value. [25][17]

2.6 Statistické testy CSE(C)

Dříve Communications Security Establishment Canada, dnes už jen CSE, je Kanadská vládní agentura, zabývající se kryptologií a je odpovědná za ochranu kanadských vládních informací a komunikační sítě. CSE se zabývá šifrováním zahraničního zpravodajství, které může kanadská vláda využít pro strategické varování, formulaci politiky, k rozhodování a každodennímu hodnocení zahraničních vlivů a záměrů.[26]

2.7 Popis vybraných statistických testů

Existuje velké množství statistických testů, přičemž každý testuje sekvenci čísel různými způsoby. Sekvence může selhat u některých testů, a přitom být opravdu náhodná, a naopak. Proto se pro co nejpřesnější analýzu testovaného generátoru používají různé sady (baterie) testů. Níže uvádím vybrané testy z baterie STS od organizace NIST.[20]

2.7.1 Frekvenční (monobit) test

Tento test se zaměřuje na poměr jedniček a nul v celé sekvenci a zjišťuje, zdali je počet jedniček a nul přibližně shodný, tedy poměrově k celkovému počtu bitů by se jedničky a nuly blížily hodnotě 0,5. Tato hodnota se očekává pro opravdu náhodnou sekvenci.

Vstupní sekvence se nejprve převede z nul a jedniček na sekvenci (-1, 1):

$$S_n = X_1 + X_2 + ... + X_n$$

Rovnice 4 - Výpočet Sn

Kde $X_i = 2\varepsilon - 1 = \pm 1$

Spočítá se S_{obs}, což je absolutní hodnota sumy X_i.

Následně se spočítá součet všech hodnot v nové sekvenci a vyhodnotí se P-value.

$$P - value = \boldsymbol{erfc}\left(\frac{S_{obs}}{\sqrt{2}}\right)$$

Rovnice 5 - Výpočet p-value frekvenčního testu

Vyhodnocení: pokud je spočítaná hodnota P-value menší než 0,01, sekvence je označena jako nenáhodná. V opačném případě, kdy je P-value > 0,01 je sekvence vyhodnocena jako náhodná.

2.7.2 Frekvenční blokový test

Tento test vyhodnocuje náhodnost testované sekvence pomocí zkoumání poměru jedniček a nul uvnitř M-bitových bloků. Předpokládá se, že u náhodné sekvence bude počet jedniček a nul uvnitř M-bitového bloku přibližně roven hodnotě M/2. Pro hodnotu M=1 je tento test shodný jako předešlý frekvenční test. Vstupní sekvence se rozdělí do M-bitových bloků a zbylé bity na konci sekvence se nevyužijí. Pro každý blok M se spočítá poměr jedniček. Následně se spočítá hodnota P-value dle vzorce:

$$P - value = \boldsymbol{igamc}(N/2, \chi^2(obs)/2)$$

Rovnice 6 - Výpočet p-value frekvenčního blokového testu

2.7.3 Test shodných bitů

Test se zaměřuje na hledání počtu shodných bitů jdoucích po sobě v testované sekvenci. Účelem testu je zjistit, zda počet skupin se shodnými bity odpovídá počtu, který se očekává pro náhodnou sekvenci. Test počítá celkový počet posloupností shodných bitů.

$$P - value = \boldsymbol{erfc}\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right)$$

Rovnice 7 - Výpočet p-value testu shodných bitů

2.7.4 Test nejdelší posloupnosti jedniček uvnitř bloku

Test se zaměřuje na nalezení nejdelší posloupnosti jedniček uvnitř M-bitového bloku a poté rozhodne, jestli nejdelší posloupnosti jedniček uvnitř bloku testované sekvence odpovídá předpokládané délce, která se očekává pro náhodnou sekvenci. Příliš velká délka posloupnosti jedniček značí příliš velkou délku posloupnosti nul, proto není potřeba dělat test zvlášť pro posloupnost nul. Vstupní sekvence je rozdělena na M-bitové bloky a pro každý blok se spočítá nejdelší posloupnost jedniček načež se z jednotlivých testů spočítá hodnota P-value.

$$P - value = \boldsymbol{igamc}\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right)$$

Rovnice 8 - Výpočet p-value testu nejdelší posloupnosti jedniček uvnitř bloku

2.7.5 Test hodnosti binární matice

Test zjišťuje hodnost disjunktních sub-matic testované sekvence a prověřuje lineární závislost mezi subřetězci pevně dané délky z testované sekvence. Minimální počet testovacích matic je 38 o velikosti matice 32x32. Pro tuto velikost binární matice byly spočítány pravděpodobnosti, které se očekávají od náhodné sekvence.

$$P - value = e^{-x^2(obs)/2}$$

Rovnice 9 - Výpočet p-value testu hodnosti binární matice

2.7.6 Test diskrétní Fourierovy transformace

Test se zaměřuje na vrcholy sekvence v diskrétní Fourierově transformaci. Hledá opakující se vzory v testovací sekvenci, které by znamenaly, že sekvence není náhodná. Zkoumá se, jestli je počet vrcholků frekvenční složky v grafu transformací přesahujících hranici 95% rozdílný než počet vrcholků přesahující hranici 5%.

Nuly a jedničky v sekvenci jsou převedeny na hodnoty -1,1 stejným způsobem, jako při Frekvenčním testu. Vznikne sekvence X = x1, x2, …, xn, kde xi = 2E-1. Následně se provede diskrétní Fourierova transformace (DFT) na sekvenci X a vznikne sekvence komplexních proměnných, které znázorňují periodické komponenty sekvence bitů na různých frekvencích.

Následně se spočítá mez T (treshold). Předpokládá se, že u náhodných sekvencí nebude překročena hranice této meze, což je 95%. Dále hodnota N0, což je očekávaný počet peaku nižších než treshold T. N1 je pak skutečně zjištěný počet peaku nižších než T. Nakonec se spočítá hodnota P-value.

$$P - value = \boldsymbol{erfc}\left(\frac{|d|}{\sqrt{2}}\right)$$

Rovnice 10 - Výpočet p-value testu DFT

2.7.7 Nepřekrývající se test shodných vzorů

Test hledá generátory sekvencí, které produkují mnoho opakujících se vzorů. Pro tento test je použito M-bitové okno pro hledání specifických M-bitových vzorů. Pokud vzor není nalezen, okno se posune o jeden bit. Pokud je nalezen vzor, okno se přesune k bitu následujícím po nalezeném vzoru a pokračuje se v hledání.

Test rozdělí vstupní testovanou sekvenci do shodných bloků velikosti M, dále definuje počet bloků N a pomocí těchto dvou parametrů snadno určíme minimální délku testované sekvence. Pro jednotlivé bloky se spočítají hodnoty a poté se vyhodnotí P-value.

$$P - value = \boldsymbol{igamc}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right)$$

Rovnice 11 - Výpočet p-value testu nepřekrývajících se shodných vzorů

2.7.8 Překrývající se test shodných vzorů

Test se stejně jako předchozí zaměřuje na počet opakování předdefinované předlohy v sekvenci a tím se snaží nalézt generátory, které generují velké množství opakujících se vzorů. Podobně jako předchozí test, i tento používá m-bitové okno pro hledání definovaného m-bitového vzoru. Pokud vzor není nalezen, okno se posune o jeden bit. Rozdíl mezi tímto testem a testem předchozím, je v tom, co se stane, pokud je nalezen vzor. U tohoto testu, pokud je nalezen vzor, se okno posune pouze o jeden bit a poté se algoritmus navrátí k hledání (u předchozího testu se okno přesune až za nalezený vzor).

$$P - value = \boldsymbol{igamc}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right)$$

Rovnice 12 - Výpočet p-value testu překrývajících se shodných vzorů

2.7.9 Maurerův univerzální statistický test

Test zjišťuje náhodnost sekvence pomocí hledání bitů mezi dvěma shodnými vzory. Pokud algoritmus nalezne tyto bity a určí, že může být sekvence významně komprimována, označí tuto sekvenci za nenáhodnou. Stejně jako u všech testů, i zde je spočítána hodnota P-value, která pokud je nižší nebo rovna hodnotě 0,01, je sekvence označena za náhodnou.

$$P - value = \boldsymbol{erfc}\left(\left|\frac{f_n - expectedValue(L)}{\sqrt{2\sigma}}\right|\right)$$

Rovnice 13 - Výpočet p-value Maurerova univerzálního statistického testu

2.7.10 Lineární komplexní Test

Tento test se zaměřuje na délku posuvného registru s lineární zpětnou vazbou (LFSR). Test rozhoduje, zdali je testovaná sekvence dostatečně komplexní, aby byla považována za náhodnou. Předpokládá se přitom, že náhodné sekvence jsou charakterizovány delším LFSR. Příliš krátký LFSR značí nenáhodnost. Vstupní sekvence se rozdělí na N bloků o velikosti M bitů a uvnitř každého bloku se hledá nejkratší bitová sekvence LFSR, která generuje zbylé bloky v bloku i. Ze získaných hodnot je vypočítaná P-value.

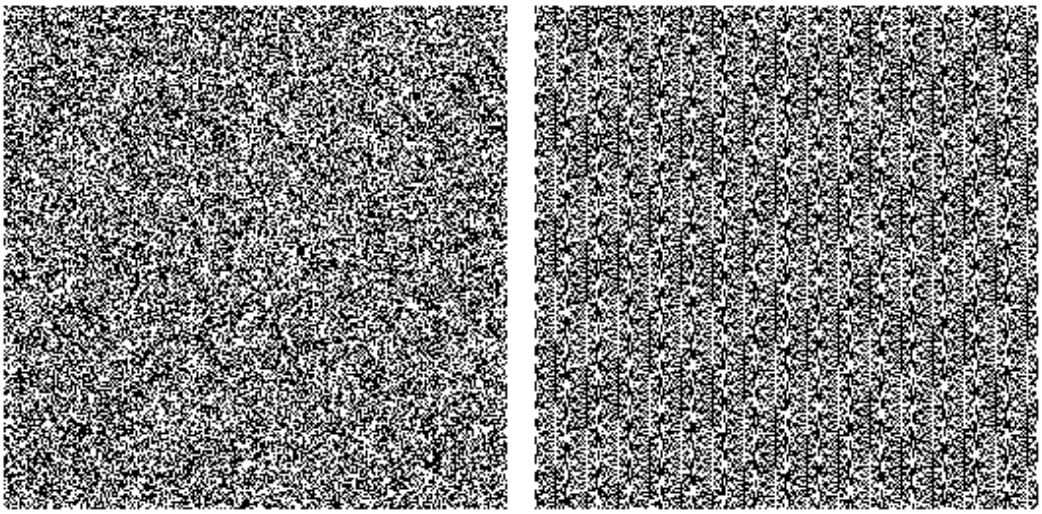
$$P - value = \boldsymbol{igamc}\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right)$$

Rovnice 14 - Výpočet p-value lineárního komplexního testu

2.8 Vizuální test

Další možnost, jak zkoumat náhodnost testované sekvence spočívá ve využití lidské schopnosti vidět tvary a vzory i ve zdánlivé neuspořádanosti. V tom jsou lidé opravdu

dobří. I když se tento druh testu nedá určit jako dostačující pro rozhodnutí, zdali je testovaná sekvence, a tedy i generátor opravdu náhodný, otestování generátoru tímto způsobem je rychlé a člověk ihned vidí hrubý odhad náhodnosti generátoru. Jak vidíte na obrázku vlevo, rozložení jedniček a nul, na bitmapovém obrázku znázorněné jako bílé a černé pixely, je bez viditelných vzorů nebo opakování. Na obrázku vpravo jdou jasně vidět opakující se čáry se stejnou vzdáleností od sebe. Navíc se na obrázku objevují další vzory, připomínající vlnky.



Obrázek 3 - Vlevo výstup z generátoru random.org, vpravo php rand() funkce

[27]

Pro stejné otestování generátorů jsem použil funkci v programu Matlab, kdy jsem převedl matici o velikosti 1024x1024bitů na bitmapový obrázek:

```
A = [];
I = mat2gray(A, [0 1]);
I = mat2gray(A);
imshow(I)
```

Funkce *mat2gray(A, [amin, amax])* převede matici *A* na černobílý obrázek nuly na černé pixely a jedničky na bílé pixely. [28] Výsledný obrázek dá hrubou představu o testovaném generátoru. Velikost 1024x1024 jsem použil pro dostatečnou kvalitu výsledného obrazce.

3 GENEROVÁNÍ NÁHODNÝCH ČÍSEL NA NÍZKO-VÝKONOVÝCH ZAŘÍZENÍCH

3.1 Nízko-výkonová zařízení

Pojmem omezená zařízení budeme rozumět mikrokontrolery s omezenou pamětí flash i RAM a s nízkým výpočetním výkonem. Výkon je omezen nízkou proudovou spotřebou a cenou, čímž se tyto zařízení vyznačují. Tato zařízení jsou používána například v senzorových systémech, kde je nutné komunikaci šifrovat a zároveň chceme co nejmenší výrobek, a tedy malý mikrokontrolér. Takový nebude mít velký výpočetní výkon ani paměť. Níže je uvedeno několik často používaných mikrokontrolérů.

Označení	Velikost flash	Velikost RAM	Frekvence procesoru	Spotřeba
ATtiny85 (Atmel)	8KB	512B	20MHz	300 µA při 1MHz, 1,8V
MSP430F2272 (Texas Instrument)	32kB	1024B	16MHz	270 µA při 1 MHz, 2,2 V
MC9s08ac128 (Freescale Semiconductor)	128kB	8kB	40MHz	3,7mA při 16MHz
MSP4305438a (Texas Instrument)	256kB	18kB	25MHz	230 µA/MHz při 8 MHz, 3,0 V

Tabulka 6 - Přehled mikrokontrolérů různých výrobců[29][30][31][32]

3.2 Nízko-výkonové zařízení MSP430F5438A

Řada velmi malých mikroprocesorů řady TI MSP430 se skládá z několika zařízení s různými sadami periferií určených pro různé aplikace. Šestnácti bitová RISC architektura CPU kombinovaná s pokročilými režimy s nízkým výkonem je optimalizována tak, aby dosáhla prodloužené životnosti baterie v přenosných měřicích aplikacích. Digitálně řízený oscilátor (DCO) umožňuje zařízení probudit z režimu nízké spotřeby do aktivního režimu v čase 3,5µs. [32]

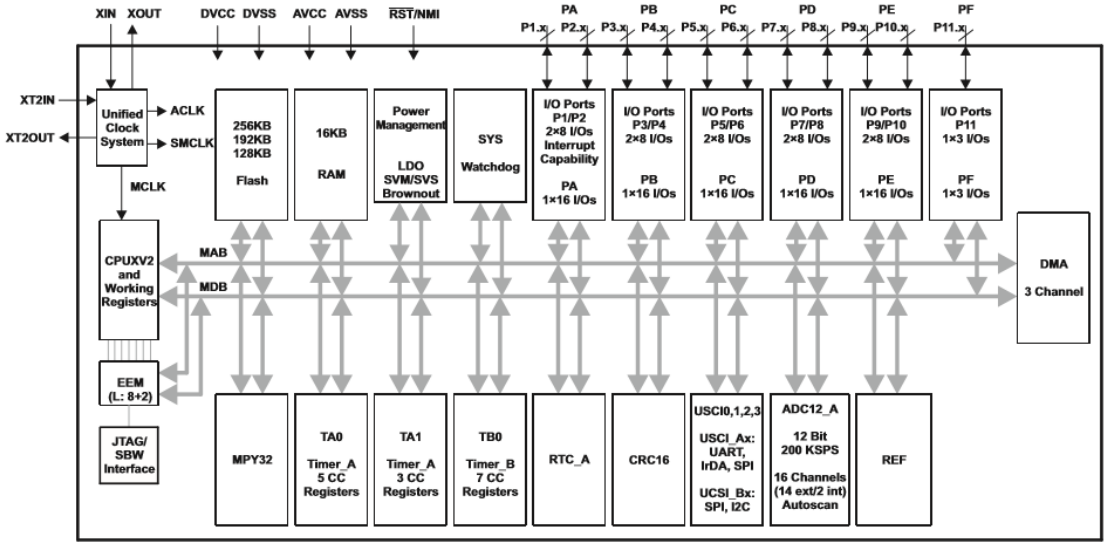
Pro naše účely bylo poskytnuto zařízení MSP430F5438A. Toto zařízení dokáže pracovat až do 25MHz, má 256kB flash paměti a 18kB RAM paměti. Pokud

nezměníme pracovní frekvenci, bude CPU pracovat o frekvenci 1MHz.

ACLK = REFO = ~32768Hz, MCLK = SMCLK = default DCO = 32 x ACLK = 1048576Hz = 1MHz

Obrázek 4 - Pohled na mikrokontrolér MSP430F5438A

Díky velkému počtu I/O pinů je možné připojit prakticky jakékoliv periferie, což nám poskytuje velké možnosti, jak na MSP430 generovat náhodná čísla. Každé připojené zařízení ale má nějaký proudový odběr, a tedy ubírá bateriím na životnosti. Proto je vhodné využít ke generování náhodných čísel interní moduly obsažené v zařízení a vyobrazené na obrázku 4.



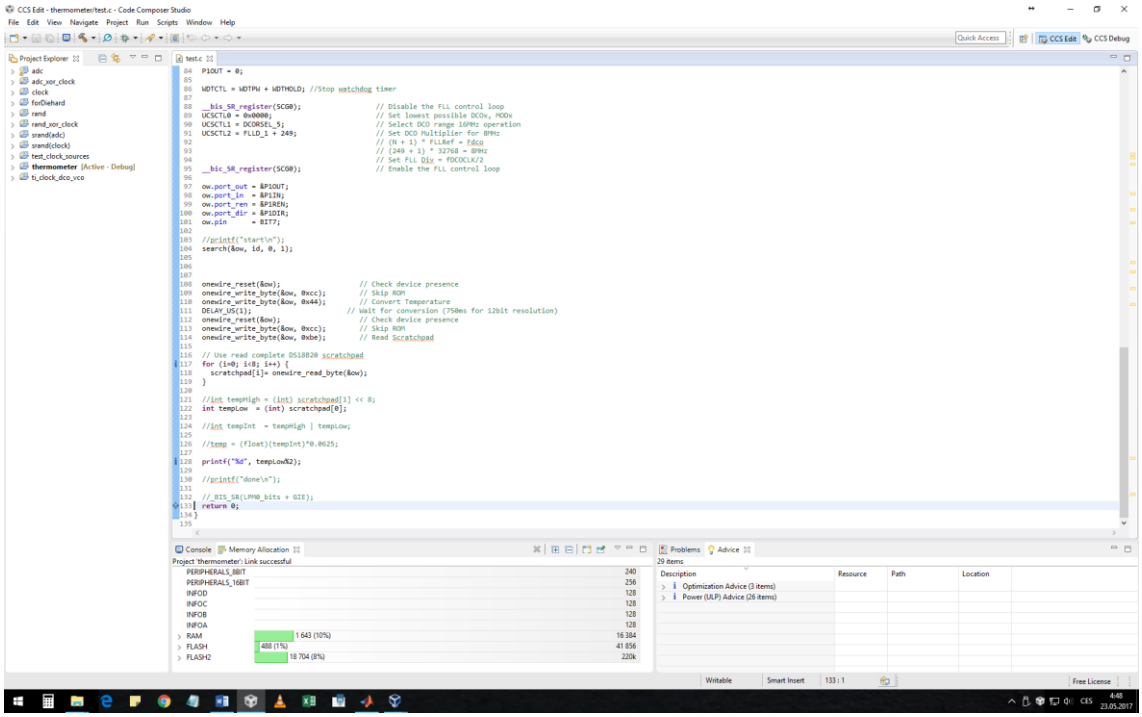
Obrázek 5 - Blokový diagram mikrokontroléru MSP430F5438A

Zařízení obsahuje několik modulů, které lze použít ke generování náhodných čísel. Předně se jedná o 12bitový AD převodník SAR, který je možné využít ke vzorkování šumu z okolí, pokud necháme vstupní piny nezapojeny. Dále se nabízí využití oscilátorů a náhodného jevu, kdy u dvou nezávislých oscilátorů dochází k náhodným odchylkám. [33][34]

Dále mi bylo zapůjčeno zařízení sloužící k jednoduchému ovládání mikrokontroléru (k nahrání software). Jedná se o nízko-výkonové zařízení MSP-FET430UIF.

Pracoval jsem v Code Composer Studio (verze 6.2.0.00050) od firmy Texas

Instruments.[35]



Obrázek 6 - Náhled na použité vývojové prostředí

Vlevo je průzkumník projekty, uprostřed okno s vyvíjeným programem, dole vlevo ukazatel paměti a dole vpravo chybové hlášení.

3.3 Implementace vybraných generátorů

Možných generátorů náhodných čísel je opravdu nepřeberné množství. Ne každý se ale dá použít na nízko-výkonových zařízeních. Z dostupných generátorů jsem vybral k otestování tyto 4:

- Generátor snímající šum na nezapojeném vstupu, dále „ADC“
- Generátor využívající rozdílu mezi dvěma oscilátory, pracovně nazvaný „clock“
- Velmi dobře známá funkce rand
- Doplnění funkce rand o parametr seed, „srand“

3.3.1 ADC generátor

Ke generování náhodných čísel je vhodné použít dobré zdroje entropie, například šum, který je ze své podstaty náhodným. Šum můžeme naměřit při přenosu signálu rádiovým prostředím nebo přes metalické vedení, anebo i na nezapojeném vstupu. S využitím A/D převodníku tak můžeme dostat generátor náhodných čísel bez připojení dalších periferií.[36]

```
    #include  <msp430.h>
    #include  <stdio.h>

    void main(void)
    {
      WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

      ADC12CTL0 =  ADC12SHT0_2 | ADC12ON;       // Set sampling time,
                                                   turn on ADC12
      ADC12CTL1 = ADC12SHP;                     // Use sampling timer
      ADC12IE = 0x01;                           // Enable interrupt
      ADC12CTL0 |= ADC12ENC;                    // Conversion enabled
      P6DIR &= 0x01;                            // P6.0, i/p
      P6SEL |= 0x01;                            // P6.0-ADC option select
      ADC12CTL0 |= ADC12SC;                         // Start convn, software
controlled
      __bis_SR_register(CPUOFF + GIE);              // LPM0, ADC12_ISR will
force exit

    }
    // ADC12 interrupt service routine
    #pragma vector=ADC12_VECTOR
    __interrupt void ADC12_ISR (void)
    {
      printf("%d ", ADC12MEM0%2);
       __bic_SR_register_on_exit(CPUOFF);      // Clear CPUOFF bit from 0(SR)
```

Po spuštění programu se nastaví vzorkovací interval, zapne se funkce přerušení a nastaví se PIN P6 jako aktivní. Samotná konverze se započne po zapsání příkazu ADC12SC do registru ADC12CTL0. Při konverzi se nevyužívá CPU, tudíž se může uspat nastavením bitů (bis = bit is set) do příslušného registru. Po skončení konverze se ADC12 modul ohlásí a program skočí do přerušení, kde se vytiskne zbytek po dělení výsledku konverze dvěma. Tím dostaneme jen LSB, který se mění nejčastěji a nejnáhodněji. CPU se opět probudí a program může pokračovat.

Návrh na vylepšení ADC generátoru

Jelikož tento generátor pracuje s náhodným šumem na nezapojených pinech, nebylo by obtížné tento šum podvrhnout, například na piny připojit pseudonáhodný/periodický šum a tím ovlivnit výsledná generovaná čísla. V kryptografii je běžné použití redundance jako protiopatření proti útokům. Tento postup použijeme i zde. Výstup ADC generátoru budeme xorovat s výstupem Clock generátoru. Pracovně jsem tento nový generátor nazval adc_xor_clock. Poté jsem zkusil nasimulovat podvržený vstupní signál, tedy jak by to vypadalo, kdyby útočník podvrhnul náhodný šum jiným, pseudonáhodným signálem. Tento scénář už jsem aplikoval na upravený generátor a provedl jsem xorování výstupu z funkce rand() (tedy pseudonáhodný signál) s výstupem z generátoru clock. Výsledný generátor jsem pojmenoval rand_xor_clock.

```
    #include <msp430.h>
    #include <stdio.h>
    #include <stdlib.h>

    //#include <timer.h>
    //#include <adc.h>

    unsigned int t,ad,x;
```

```
//*******************************
void set_timers(void) {
//-----------------------------------------------------
TA0R = 0;
UCSCTL4 |= SELS__DCOCLK;
TA0CTL   = TASSEL_2 | MC_2;
TA0CCTL0 = CAP | CM_1 | CCIS_1;
TA1R = 0;
UCSCTL4 |= SELA__VLOCLK;
TA1CCR0 = 1;
TA1CTL   = TASSEL_1 | MC_1;
TA1CCTL0 = CCIE;
__bis_SR_register(LPM3_bits | GIE);
}
#pragma vector=TIMER1_A0_VECTOR
__interrupt void Timer1_A0 (void) {
TA0CTL = MC_0;
TA1CTL = MC_0;
__bic_SR_register_on_exit(LPM3_bits);
}
//*************************************
void adc(void) {
ADC12CTL0 =  ADC12SHT0_2 | ADC12ON;
ADC12CTL1 = ADC12SHP;
ADC12IE = 0x01;
ADC12CTL0 |= ADC12ENC;
P6DIR &= 0x01;
P6SEL |= 0x01;
ADC12CTL0 |= ADC12SC;
__bis_SR_register(CPUOFF + GIE);
}
#pragma vector=ADC12_VECTOR
__interrupt void ADC12_ISR (void)
{
ad=ADC12MEM0%2;
__bic_SR_register_on_exit(CPUOFF);
}
//*****************************************
int main(void) {
WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
x=0;

set_timers();
t=TA0R%2;
adc();
x=t^ad;
printf("%d ",x);

return 0;
}
```

Provedl jsem tedy úpravu původního ADC generátoru tím, že jsem jeho výstup sloučil s výstupem Clock generátoru. Pro otestování, zda toto zkombinování pomohlo při podvržení šumu a ovlivnění ADC generátoru jsem tento nahradil funkcí rand().

3.3.2 Clock generátor

Tento generátor využívá náhodného jevu posuvu dvou nezávislých oscilátorů. Tato myšlenka už byla popsána dříve, ale algoritmus znovu popsal Texas Instrument jako demonstraci vytvoření náhodného generátoru na MSP430x2. Zdrojový kód

pro generátor je volně ke stažení, avšak nenašel jsem verzi pro MSP430x5. Jelikož se druhá modelová řada zařízení liší oproti páté řadě, která mi byla zapůjčena, upravil jsem kód a implementoval na MSP430F5438A. Obě modelové řady se liší velkou měrou v provedení čítačů a hodinového systému. Jelikož je právě tohle principem generátoru, v podstatě jsem napsal kód pro generátor od nuly.

Principem tedy je posuv mezi dvěma nezávislými oscilátory, konkrétně DCO (digitally controlled oscilator) a VLO (very-low-frequency oscillator). DCO je rychlý oscilátor, který počítá počet tiků za jeden tik VLO. Jelikož se mezi nimi náhodně objevuje odchylka, nemůžeme předpovědět, jestli DCO tikne xkrát nebox+1krát za dobu trvání tiku VLO.

Oba oscilátory nastavím na hodinové signály. SMCLK (Submain clock) přiřadím k DCO a ACLK (Auxilliary clock) nastavím na VLO. Dále v kódu nastavím dva čítače TimerA0 a TimerA1.

```
    TA0R = 0;       //reset counter
      UCSCTL4 |= SELS__DCOCLK;   //select oscillator DCO for SMCLK
      TA0CTL   = TASSEL_2 | MC_2;
      // * TASSEL_2 => Clock source for timer is SMCLK
      // * MC_2     => Contiuous mode, count up to 0FFFFh
      TA0CCTL0 = CAP | CM_1 | CCIS_1;
      // Register TA0CCTL0
      // * CAP    => Enable capture mode (record events)
      // * CM_1   => Capture on rising edge (pos. edge)
      // * CCIS_1 => Capture input is signal CCI0B
```

Nejprve je potřeba vynulovat registr TA0R, kam se ukládá počet tiků. Dále nastavím TA0 na hodinový signál SMCLK, který je připojen na oscilátor DCO, a tedy tiká velkou rychlostí. TA0 dále nastavím na „continuous mode“, což znamená, že bude tikat až do maximální hodnoty 0FFFFh. Nakonec nastavit snímání (capture) na náběžnou hranu (rising edge).

```
    TA1R = 0;       //reset counter
      UCSCTL4 |= SELA__VLOCLK;   //select oscillator VLO for ACLK
      //In compare mode, the timer value is compared with value
      TA1CCR0 = 1;
      TA1CTL   = TASSEL_1 | MC_1;
      // * TASSEL_1 => Clock source for timer is ACLK (auxiliary clock)
      // * MC_1     => Up mode, timer counts up to value stored in TA1CCR0
      TA1CCTL0 = CCIE; //enable interrupt
```

Stejně jako u Timeru0 i zde nejprve vynulujeme čítací registr TA0R. Nastavím TA1 na hodinový signál ACLK a zvolím „up mode“. To znamená, že čítač bude tikat do nastavené hodnoty a pak vyhodí příznak přerušení. Tato hodnota je uložena v registru TA1CCR0. Jako poslední krok zapnu přerušení a TimerA1 je nastaven.

Celé toto nastavení může být přímo v hlavní funkci programu main(). Při spuštění programu dojde k nastavení čítačů, které se tím zapnou a nakonec (po dosažení hodnoty „1“) program skočí do obslužné funkce. Zde se oba čítače zastaví příkazem zapsáním hodnoty MC_0 do registru TAxCTL. Poté je v registru TA0R uchována hodnota, počet tiků, které provedl TimerA0. Tuto hodnotu vydělím dvěma a výsledek je vygenerované náhodné číslo

```
#pragma vector=TIMER1_A0_VECTOR
  // Timer1 A0 interrupt service routine
  __interrupt void Timer1_A0 (void) {
    // Pause timers 0, 1
    TA0CTL = MC_0;
    TA1CTL = MC_0;
    // Reactivate CPU from LOW-PWR mode
    // * it will jump back to place where the was the interruption
    __bic_SR_register_on_exit(LPM3_bits);
  }
```

3.3.3 Rand generátor

Funkce Rand() obsažená v knihovně stdlib.h vrací pseudonáhodné číslo od nuly do hodnoty rand_max. Funkce se dá vyjádřit následujícím vztahem:

$$Next = next*1103515245 + 12345$$

Rovnice 15 - Výpočet funkce rand()

Pak bude funkce vracet hodnotu (next/65536)/rand_max.

Implementace byla záležitostí pár řádků:

```
#include <stdio.h>
#include <msp430.h>
#include <stdlib.h>

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer

printf("%d ", rand()%2);

return 0;
}
```

Rand_max jsem nechal beze změny, zůstává na hodnotě 32767.

3.3.4 Srand generátor

Pokud při volání funkce rand() vstupní parametr srand, můžeme dostat náhodné výsledky. Pokud by byl vstupní parametr náhodný, a s každou iterací generování čísla bychom použili jiný vstup, funkce už nemusí být jen pseudonáhodná.

Jako vstup jsem použil výstup z Clock generátoru a z ADC generátoru.

4 NÁVRH VLASTNÍHO GENERÁTORU

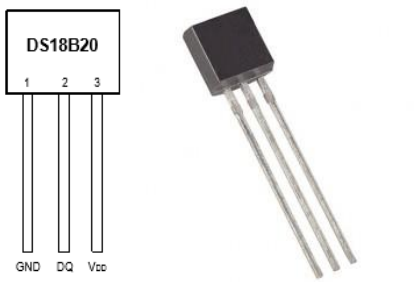
Mnoho dnešních aplikací nízko-výkonových zařízení jsou navrženy k použití v případech, kdy jsou umístěny do těžko přístupných míst, kde snímají a zaznamenávají údaje z okolí (např. ve vinném sklípku). Zařízení snímá okolní teplotu, vlhkost a další vlastnosti a posílá tyto údaje centrálnímu řídícímu uzlu. Může vyvstat potřeba tyto informace posílat zabezpečeně s použitím šifrovacích algoritmů (např. RSA, Diffie-Hellman). Oba zmíněné algoritmy používají ke své funkci generování náhodného čísla.[37]

4.1 Popis generátoru

V této kapitole navrhuji generátor náhodných čísel využívající prvek, který senzorová jednotka už má k plnění svého účelu – teplotní čidlo.

Studie **[1]** ukazuje, že použití změny teploty ke generování náhodných čísel se dá použít jako zdroj entropie. Můžeme tedy využít faktu, že výsledné zařízení bude mít teplotní čidlo k měření teploty i ke generování náhodných čísel.

Použil jsem čidlo s DS18B20 firmy Maxim Integrated s digitálním výstupem. Čidlo komunikuje pouze přes jeden vodič, odtud i pojmenování komunikačního protokolu OneWire. Každé čidlo má vlastní unikátní 64bitový sériový kód, což umožňuje jednomu zařízení komunikovat s více čidly najednou (dokonce po jedné sběrnici).



Obrázek 7 - Nákres čidla DS18B20

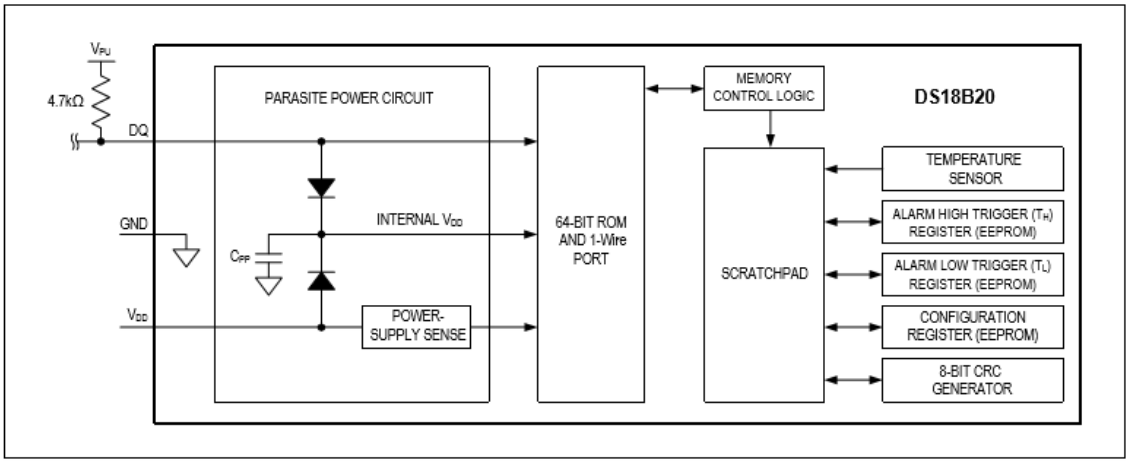
Klíčové vlastnosti:

- Napájecí napětí 3,3-5V
- Odběr ve spánkovém režimu 750nA
- Odběr v aktivním režimu 1mA
- Teplotní rozsah od -55°C po +125°C (-67°F až +257°F)
- Přesnost měření ±0,5°C pro teplotní rozsah -10°C až +85°C
- Programovatelné rozlišení vestavěného A/D převodníku, 9-12 bitů
- Nejsou potřeba žádné další součástky nebo pomocné obvody

- Redukce potřebných komponentů díky vestavěnému teplotnímu čidlu a paměti EEPROM
- Čidlo lze naprogramovat na alarm při dosažení spodní nebo horní teplotní hranice předem nastavené uživatelem.

4.2 Funkce pro komunikaci s čidlem

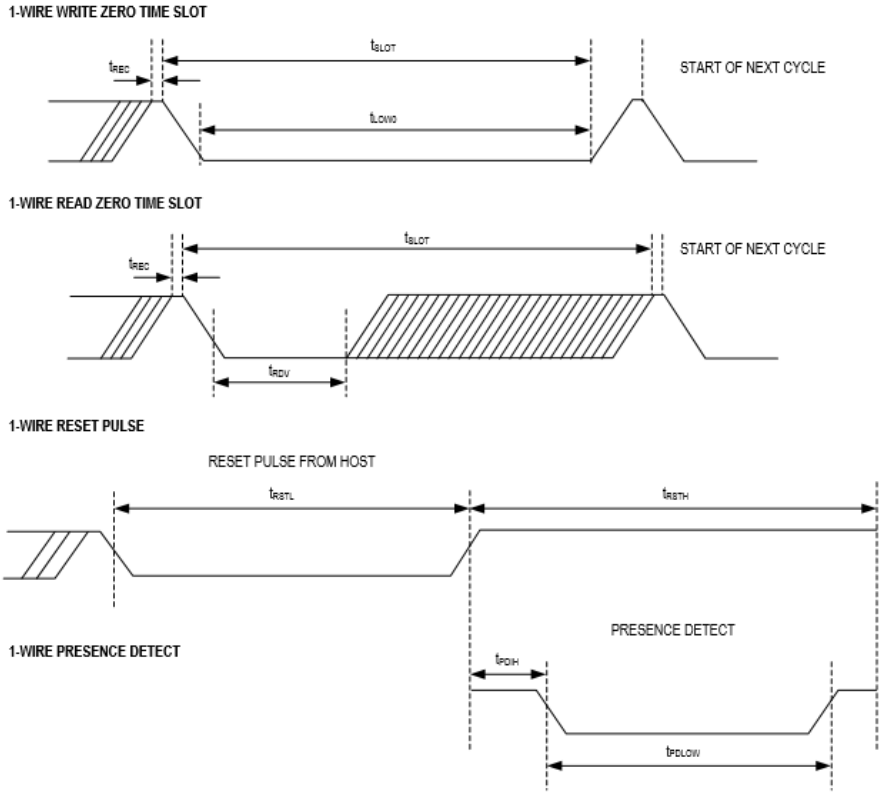
- **Search Rom** – funkce k identifikaci všech připojených čidel
- **Read Rom** – funkce ke zjištění 64bitové adresy jednoho připojeného zařízení
- **Match Rom** – funkce k adresování určitého čidla pomoci 64bitového čísla
- **Skip Rom** – master použije tuto funkci, pokud chce adresovat všechny připojené čidla
- **Alarm Search** – stejná funkce jako Search ROM, jen odpoví pouze čidlo s nastaveným alarmem
- **Convert T** – tento příkaz zahájí konverzi teploty
- **Write Scratchpad** – funkce umožňující masterovi zapsat 3 byty do Scratchpadu
- **Read Scratchpad** – funkce umožňující masterovi přečíst obsah Scratchpadu
- **Copy Scratchpad** – zkopíruje obsah Scratchpadu (T_H, T_L a konfigurační byty) do EEPROM paměti
- **Recall** – slouží k volání uložených dat pro alarm z EEPROM do Scratchpadu
- **Read Power Supply** – příkaz ke zjištění, jestli nějaké připojené čidlo používá parasitní napájení



Obrázek 8 - Vnitřní zapojení DS18B20

4.3 Princip fungování generátoru:

Pracovní frekvenci čipu nastavíme na 8MHz, jelikož protokol komunikuje skrze časové rámce a vyžaduje přesnou synchronizaci. Po spuštění programu proběhne inicializace a spojení s čidlem funkcí search() a funkcí reset(). První funkce hledá připojená zařízení pomocí funkce reset(). Strana MSP430 (master) nakrátko nastaví log“0“ na minimálně 480µs, pak obnoví log“1“. Pokud se na sběrnici nachází zařízení DS18B20 (slave) a zachytí náběžnou hranu, čeká 80µs a shodí hodnotu na log“0“. Po 300µs hodnotu vrací zpět na log“1“, což zachytí master a dostává tak informaci o přítomnosti slave.



Obrázek 9 - Časový diagram komunikace

```
    int onewire_reset(onewire_t *ow)
    {
      onewire_line_low(ow);
      DELAY_US(500); // 480us minimum
      onewire_line_release(ow);
      DELAY_US(80); // slave waits 15-60us
      if (*(ow->port_in) & ow->pin) return 1; // line should be pulled down by
slave
      DELAY_US(300); // slave TX presence pulse 60-240us
      if (!(*(ow->port_in) & ow->pin)) return 2; // line should be "released"
by slave
      return 0;
    }
```

Funkce reset vrací tři možné stavy: return1 a return2 pokud dojde ke špatné synchronizaci mezi master-slave, anebo pokud zařízení není přítomno. Return0 pro bezchybné provedeni funkce, tedy ohlášení zařízení.

```
    void search(onewire_t *ow, uint8_t *id, int depth, int reset)
    {
      int i, b1, b2;

      if (depth == 64)
      {
        // we have all 64 bit in this search branch
        printf("found: ");
        for (i = 0; i < 8; i++) printf("%d", id[i]);
        printf("\n");
        return;
      }

      if (reset)
      {
       stat = onewire_reset(ow);

        if (stat != 0) {
          printf("reset failed\n"); return;
        }
        onewire_write_byte(ow, 0xF0); // search ROM command

        // send currently recognized bits
        for (i = 0; i < depth; i++)
        {
          b1 = onewire_read_bit(ow);
          b2 = onewire_read_bit(ow);
          onewire_write_bit(ow, id[i / 8] & (1 << (i % 8)));
        }
      }
      // check another bit
      b1 = onewire_read_bit(ow);
      b2 = onewire_read_bit(ow);
      if (b1 && b2) return; // no response to search
      if (!b1 && !b2) // two devices with different bits on this position
      {
        // check devices with this bit = 0
        onewire_write_bit(ow, 0);
        id[depth / 8] &= ~(1 << (depth % 8));
        search(ow, id, depth + 1, 0);
        // check devices with this bit = 1
        id[depth / 8] |= 1 << (depth % 8);
        search(ow, id, depth + 1, 1); // different branch, reset must be
issued
      } else if (b1) {
        // devices have 1 on this position
        onewire_write_bit(ow, 1);
        id[depth / 8] |= 1 << (depth % 8);
        search(ow, id, depth + 1, 0);
      } else if (b2) {
        // devices have 0 on this position
        onewire_write_bit(ow, 0);
        id[depth / 8] &= ~(1 << (depth % 8));
        search(ow, id, depth + 1, 0);
      }
    }
```

Funkce search() tedy využívá funkce reset() k rozpoznání, zdali je zařízení přítomno. Pokud funkce reset() vrátí chybové stavy return1 nebo return2 tak funkce search() vypíše „reset failed“. Pokud se zařízení v pořádku ohlásí, tak se vypíše sériové číslo přítomného zařízení.

```
 int main()
{
  onewire_t ow;
  uint8_t id[8];

  P1REN = 0;
  P1DIR = 0;
  P1OUT = 0;

  WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer

  //nastaveni dco na 8mhz

  __bis_SR_register(SCG0);             // Disable the FLL control loop
  UCSCTL0 = 0x0000;                    // Set lowest possible DCOx, MODx
  UCSCTL1 = DCORSEL_5;                 // Select DCO range 16MHz operation
  UCSCTL2 = FLLD_1 + 249;              // Set DCO Multiplier for 8MHz
                                       // (N + 1) * FLLRef = Fdco
                                       // (249 + 1) * 32768 = 8MHz
                                       // Set FLL Div = fDCOCLK/2
  __bic_SR_register(SCG0);             // Enable the FLL control loop

  ow.port_out = &P1OUT;
  ow.port_in  = &P1IN;
  ow.port_ren = &P1REN;
  ow.port_dir = &P1DIR;
  ow.pin      = BIT7;

  printf("start\n");
  search(&ow, id, 0, 1);


  while (1) {

  onewire_reset(&ow);                   // Check device presence
  onewire_write_byte(&ow, 0xcc);        // Skip ROM
  onewire_write_byte(&ow, 0x44);        // Convert Temperature
  DELAY_US(1);                          // Wait for conversion
  onewire_reset(&ow);                   // Check device presence
  onewire_write_byte(&ow, 0xcc);        // Skip ROM
  onewire_write_byte(&ow, 0xbe);        // Read Scratchpad

  // Use read complete DS18B20 scratchpad
  for (i=0; i<8; i++) {
    scratchpad[i]= onewire_read_byte(&ow);
  }


  int tempHigh = (int) scratchpad[1] << 8;
  int tempLow  = (int) scratchpad[0];

  int tempInt  = tempHigh | tempLow;

  temp = (float)(tempInt)*0.0625;
```

```
    printf("Temperature: %f °C\n", temp);

    //DELAY_US(200);
    }
    printf("done\n");

    _BIS_SR(LPM0_bits + GIE);
    return 0;
}
```

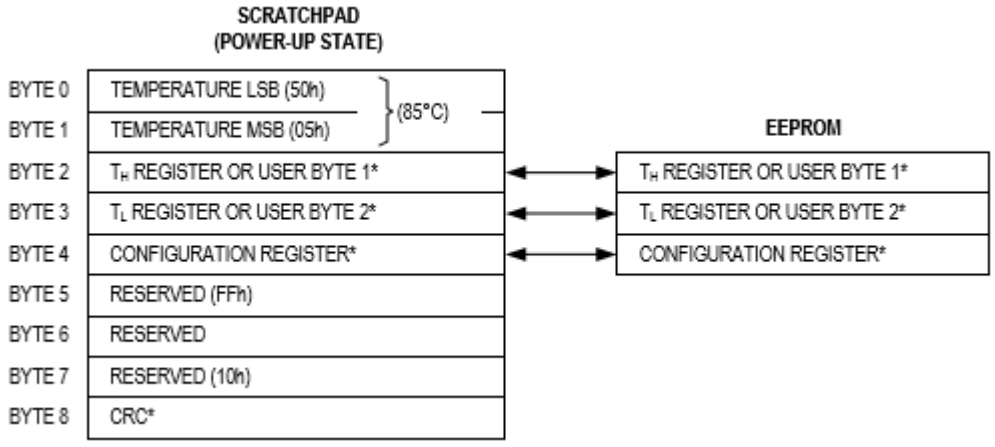
Hlavní funkce main() nastaví aktivní PIN, který bude použit pro komunikaci se zařízením. PIN musí být nastaven jako výstupní a musí se na něm nastavit pullup resistor, díky němuž je možné komunikovat po sběrnici. Dále je nutné nastavit oscilátor DCO na pracovní frekvenci 8MHz, protože takt CPU se řídí dle zdroje MCLK a ten je navázán na DCO. Potřebujeme přesnou hodnotu kvůli synchronizaci s čidlem. Pokud by byl takt jiný, nepodařilo by se navázat inicializační spojení. Dále určíme protokolu OneWire vstupní a výstupní piny a zavoláme funkci search() vysvětlenou výše.

Po inicializaci a ohlášení zapíšeme bity pro „Skip ROM“ následované bity pro „Convert temperature“. Prvním nastavením master adresuje všechna zařízení na sběrnici, poté následuje zapsání bitů „44h“ což na zařízení spustí konverzi teploty. Doba konverze se liší dle nastavené citlivosti vnitřního A/D převodníku. Pokud vyžádáme od zařízení teplotu dříve, než se dokončila konverze, dostaneme předchozí hodnotu. Zkonvertovaná hodnota se uloží do dvou osmibitových registrů LSB a MSB, viz níže na obrázku.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	**BIT 14**	**BIT 13**	**BIT 12**	**BIT 11**	**BIT 10**	**BIT 9**	**BIT 8**
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Obrázek 10 - Registry pro uložení teploty



Obrázek 11 - Zobrazení paměti v teplotním čidle

Z registrů LSB a MSB teplotu nelze odečíst napřímo. Registr MSB totiž obsahuje 5 „sign“ bitů, které označují kladnou nebo zápornou hodnotu a dále 4 horní bity čísla před desetinnou čárkou. LSB registr obsahuje pak spodní 4 bity čísla před desetinnou čárkou a 4 bity čísla za desetinnou čárkou.

```
// Use read complete DS18B20 scratchpad
  for (i=0; i<8; i++) {
    scratchpad[i]= onewire_read_byte(&ow);
  }


  int tempHigh = (int) scratchpad[1] << 8;
  int tempLow  = (int) scratchpad[0];

  int tempInt  = tempHigh | tempLow;

  temp = (float)(tempInt)*0.0625;

  printf("Temperature: %f °C\n", temp);
```

Čtení teploty probíhá následovně:

- Načteme hodnoty z paměti zařízení do proměnné „scratchpad“
- Vložíme hodnoty z registru MSB (scratchpad[1]) do vytvořené proměnné tempHigh a bitově posuneme o 8 pozic, tím se nám uvolní bity 0 až 7
- Vložíme hodnoty z registru LSB (scratchpad[0]) do vytvořené proměnné tempLow
- Sečteme obě proměnné bitovým součtem a dostaneme proměnnou o velikosti 16bitů
- Přetypujeme tuto proměnnou na datový typ “float”, který umožní vyjádřit číslo s desetinnou čárkou a toto číslo vynásobíme jednou šestnáctinou (bitový posun doprava o čtyři pozice).

Při maximálním rozlišení A/D převodníku (12bit) zůstává toto nastavení neměnné, a tedy nejmenší krok, po kterém se může měnit hodnota je jedna šestnáctina = 0,0625. Pokud snížíme rozlišení, nejmenší skok se zvýší na 0,125, jelikož se ubere bit0. Při rozlišení 9bit pak bude nejmenší skok 0,5°C což znamená, že měřená teplota by se musela změnit o půl stupně Celsia, aby zařízení ukázalo jinou teplotu.

Pro generování jedniček a nul nám postačí nultý bit z registru LSB:

```
  int tempLow  = (int) scratchpad[0];
  printf("%d", tempLow%2);
```

V kódu jsem použil tisk do konzole kvůli jednoduššímu generování velkého množství bitů pro následné testování.

5 TESTOVÁNÍ IMPLEMENTOVANÝCH GENERÁTORŮ

Pro testování statistickými testy je důležité mít dostatečně dlouhou testovací sekvenci, jelikož některé testy pracují s bloky o velikosti např. až 100 bitů. Implementované generátory jsem nechal vygenerovat sekvenci o velikosti 10 653 696 bitů a následně tyto sekvence otestoval.

5.1 Parametry implementovaných generátorů

Dobu běhu programu můžeme spočítat, když známe pracovní frekvenci CPU – 1048576 Hz.

$$\text{Čas (s)} = \text{počet cyklů} / \text{frekvence CPU}$$

Rovnice 16 - Výpočet doby trvání programu

	Počet cyklů	Doba vygenerování 1bit[ms]	Flash [kB]	RAM [kB]
ADC	1869	1,78	4556	1628
Adc_xor_clock	2150	2,0	4678	1634
Rand_xor_clock	2205	2,10	4748	1638
Clock	1830	1,75	4608	1628
Rand	1920	1,83	4658	1632
Srand_adc	2216	2,11	4728	1634
Srand_clock	2040	1,95	4750	1632
DS18B20	96266	91,81	18704	1643

Tabulka 7 - Porovnání parametrů generátorů

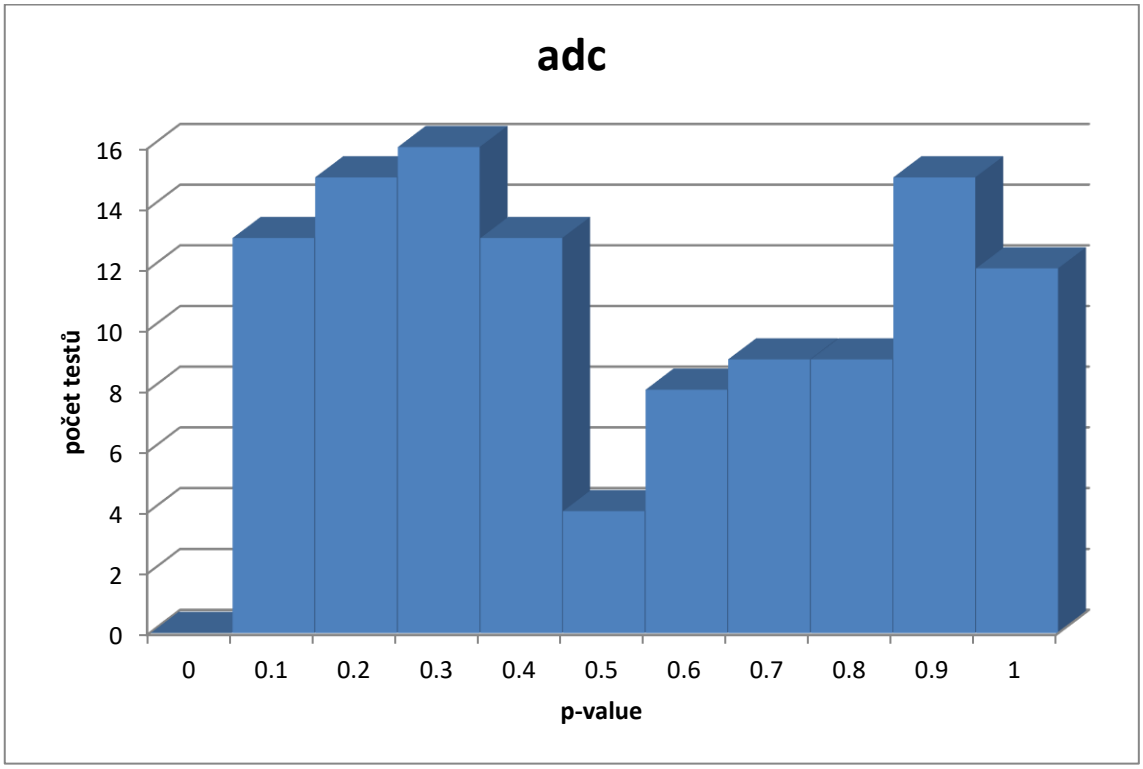
5.2 Dieharder

Pro testování sadou Dieharder jsem použil volně dostupnou verzi na linuxové distribuci lubuntu. V terminálu se zadá příkaz *sudo apt-get install dieharder* a poté je balíček stažen. Samotný test se spustí příkazem *dieharder -a -f „jméno_souboru.txt“.*

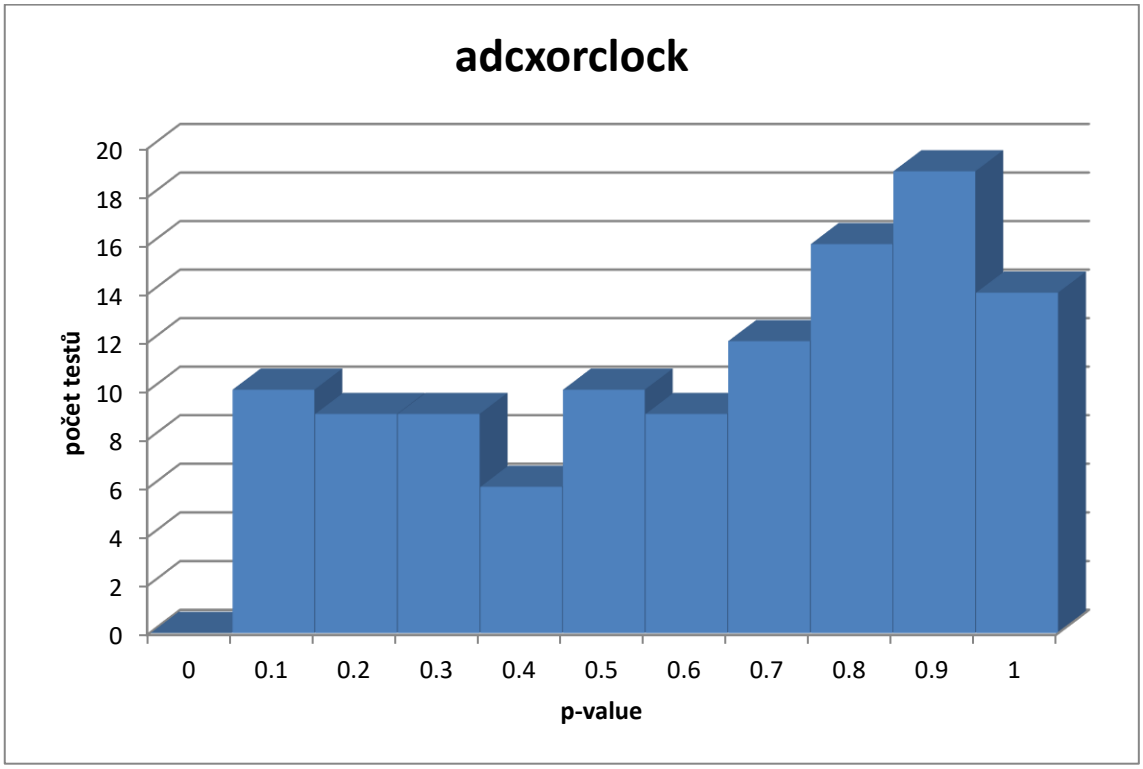
Parametr *a* určuje, že se mají provést všechny dostupné testy. Po skončení nám program vyhodnotí všechny výsledky. Slovně *PASS* nebo *WEAK* a uvede hodnotu p-value.

Náhodný generátor by měl mít výsledné hodnoty p-value rovnoměrně rozdělené

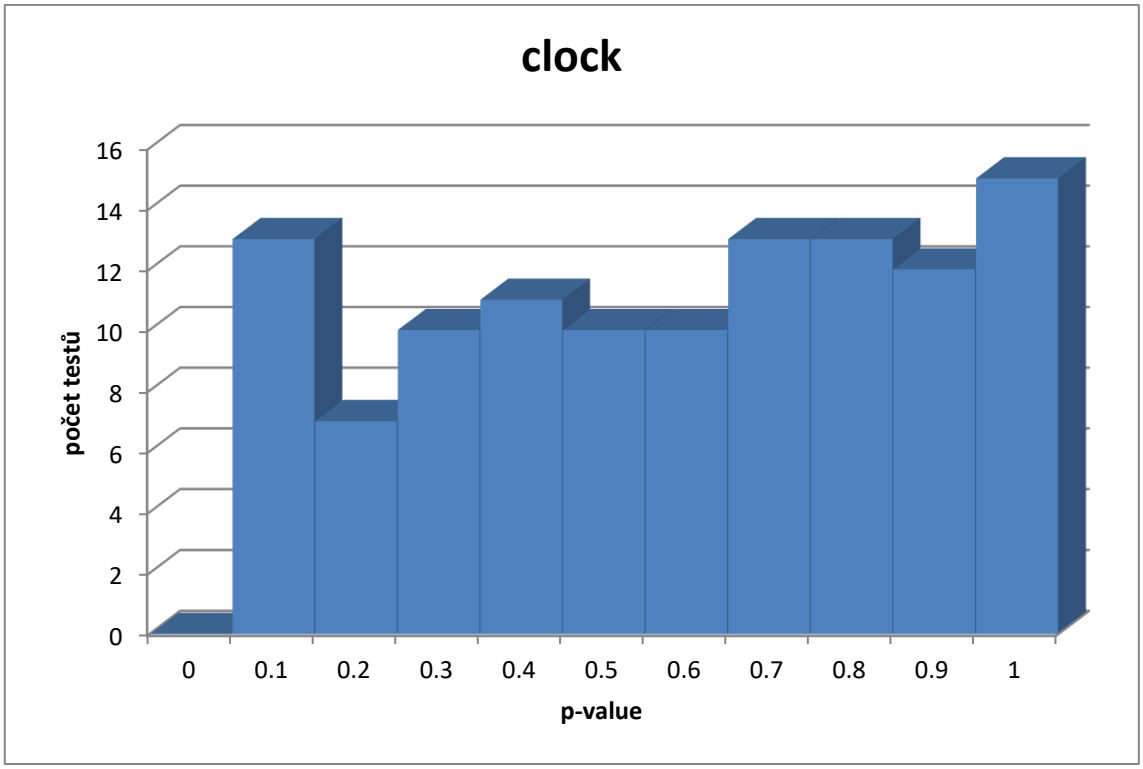
v intervalu <0,1>. Následující histogramy zobrazují toto rozložení jednotlivých generátorů.



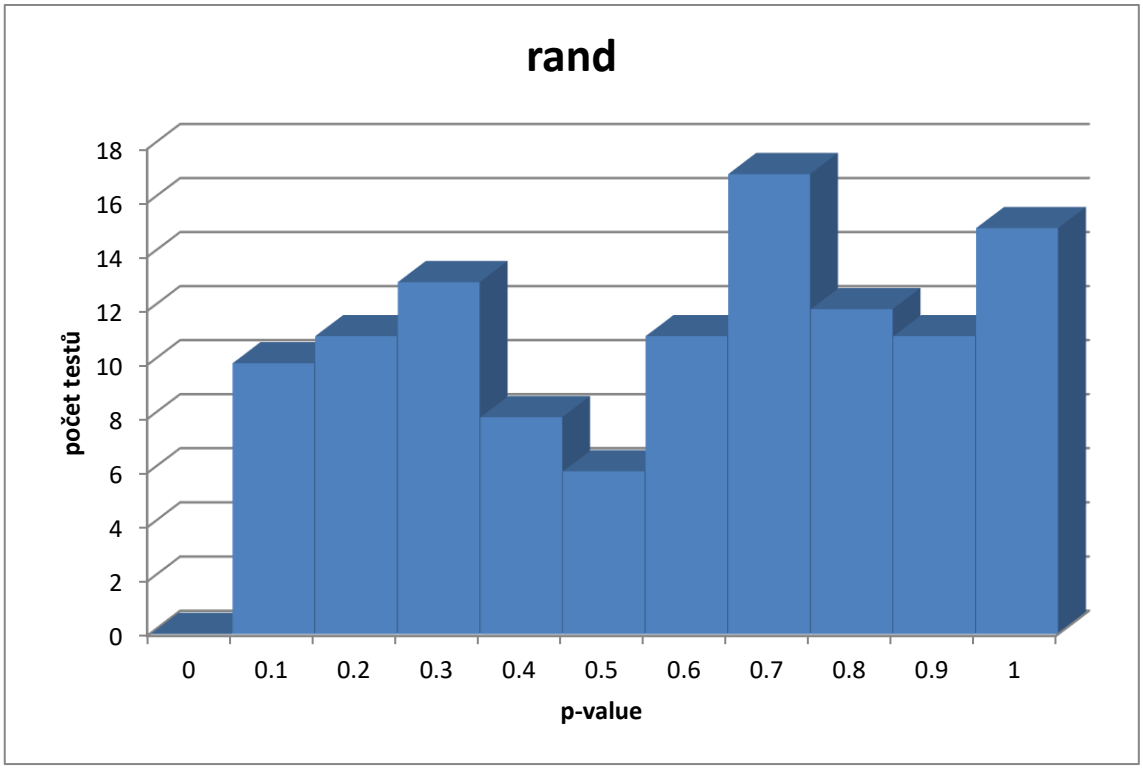
Obrázek 12 - Histogram testů Dieharder, adc



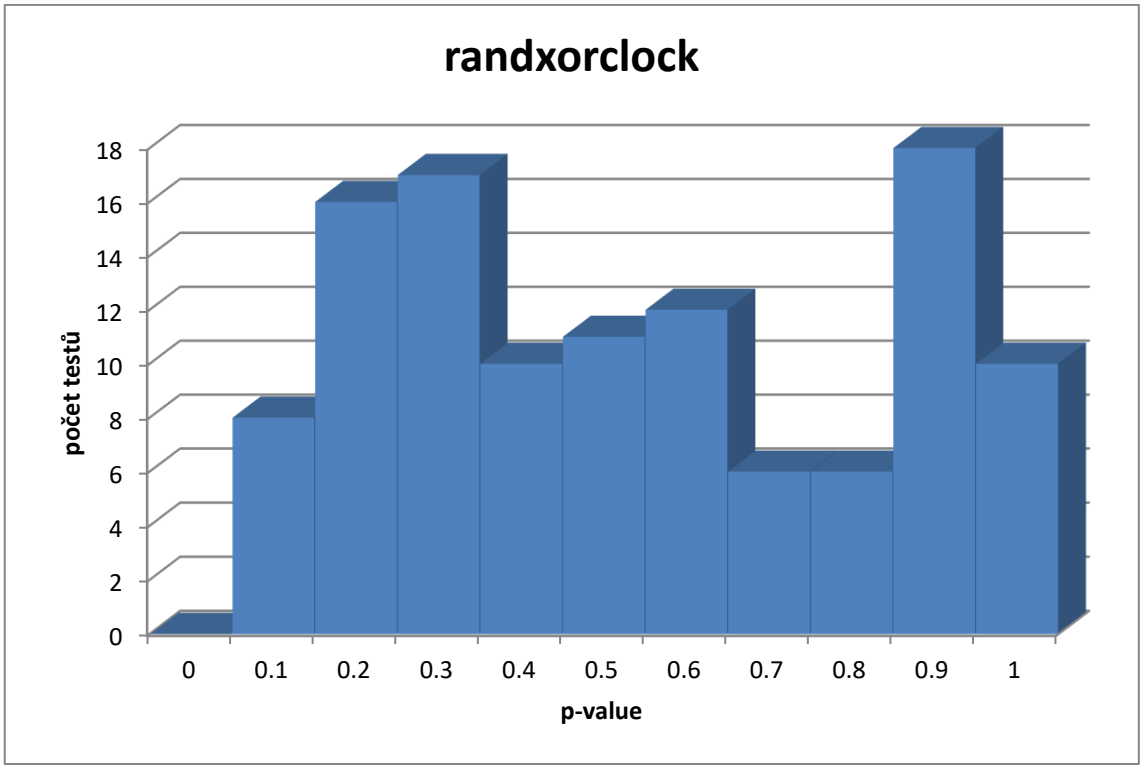
Obrázek 13 - Histogram testů Dieharder, adcxorclock



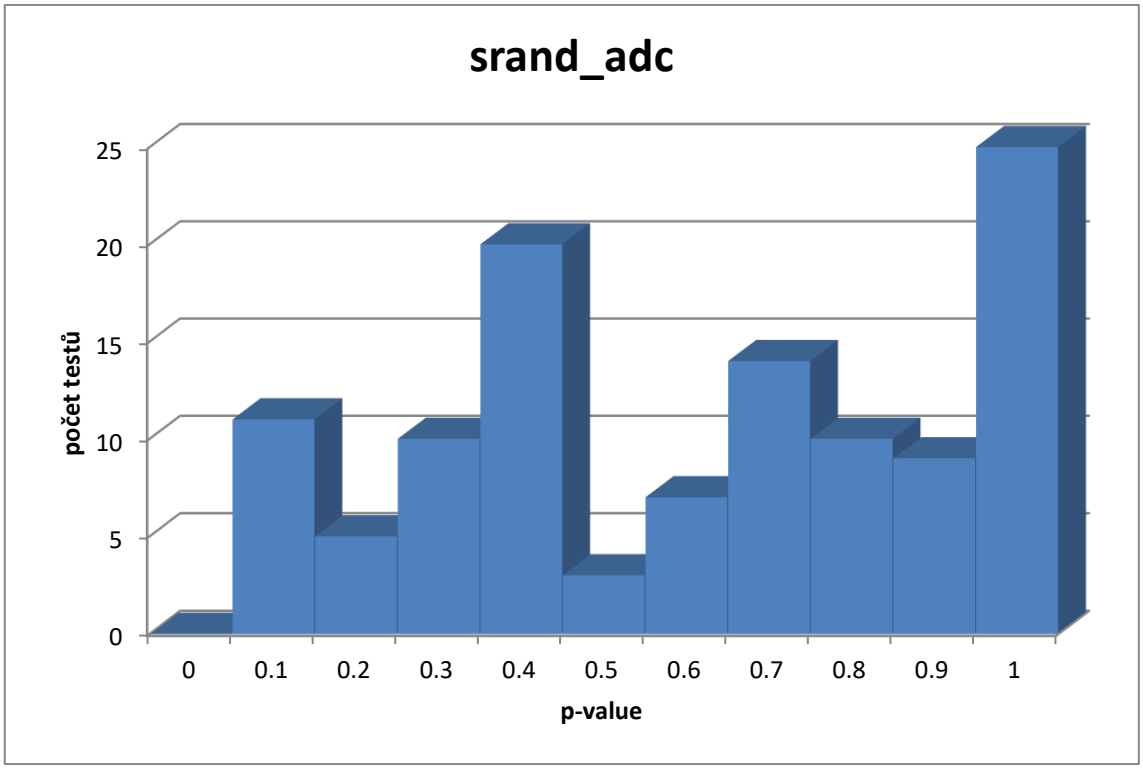
Obrázek 14 - Histogram testů Dieharder, clock



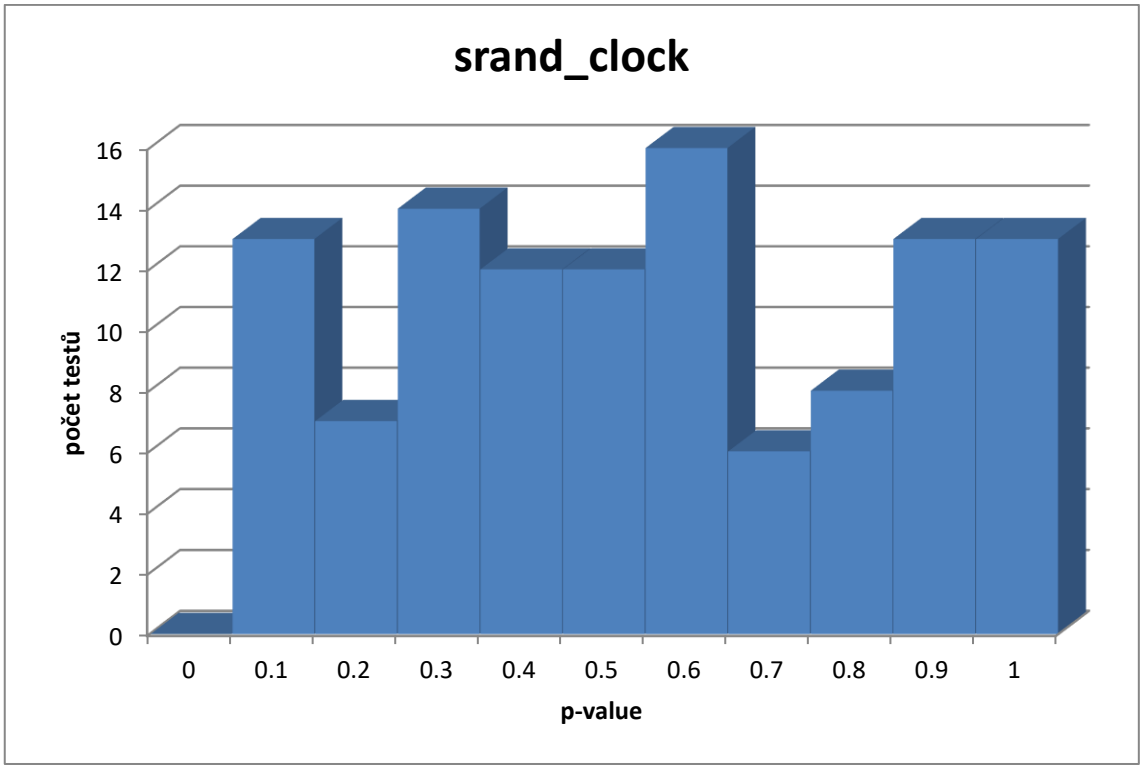
Obrázek 15 - Histogram testů Dieharder, rand



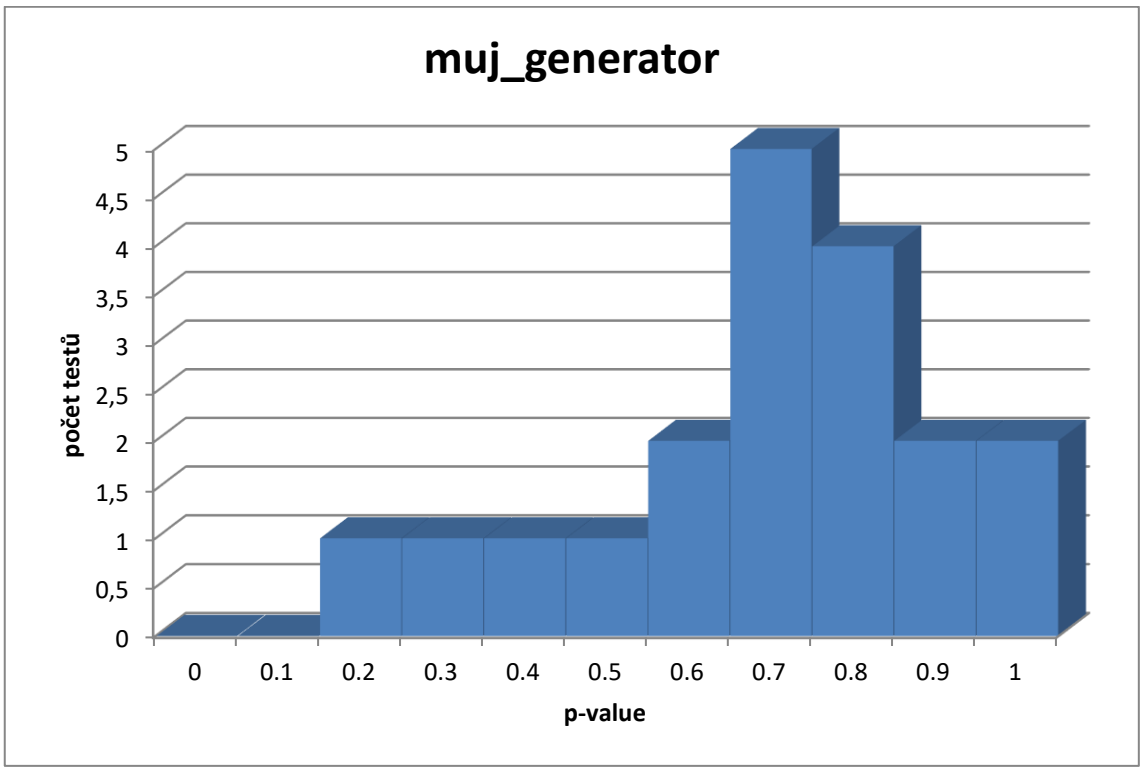
Obrázek 16 - Histogram testů Dieharder, randxorclock



Obrázek 17 - Histogram testů Dieharder, srand_adc



Obrázek 18 - Histogram testů Dieharder, srand_clock



Obrázek 19 - Histogram testů Dieharder, můj generátor

5.3 Sada testů Statistical Test Suite - NIST

Stejně jako sadu testů Dieharder, jsem i tuto sadu zkompiloval v linuxové distribuci lubuntu. Po otevření aplikace příkazem *./assess* je uživatel vyzván k zadání délky sekvence a poté je na výběr možnost otestovat různé generátory (např. Blum-Blum-Blum, Lineární kongruentní generátor apod.) anebo zvolit vlastní soubor k otestování. Aplikace podporuje vstup ve formátu ASCII, kdy ze souboru načítá znaky „0“ a „1“ a tyto považuje za bity a taktéž podporuje binární formát, kdy každý bajt v datovém souboru obsahuje 8 bitů dat. Po zadání všech parametrů testů se spustí testování a po chvíli se zobrazí zpráva informující o úspěšném dokončení. Samotné výsledky nalezneme v adresáři /experiments/algorithmTesting, soubor finalAnalysisReport.txt. Jednotlivé testy nalezneme v příslušných adresářích.

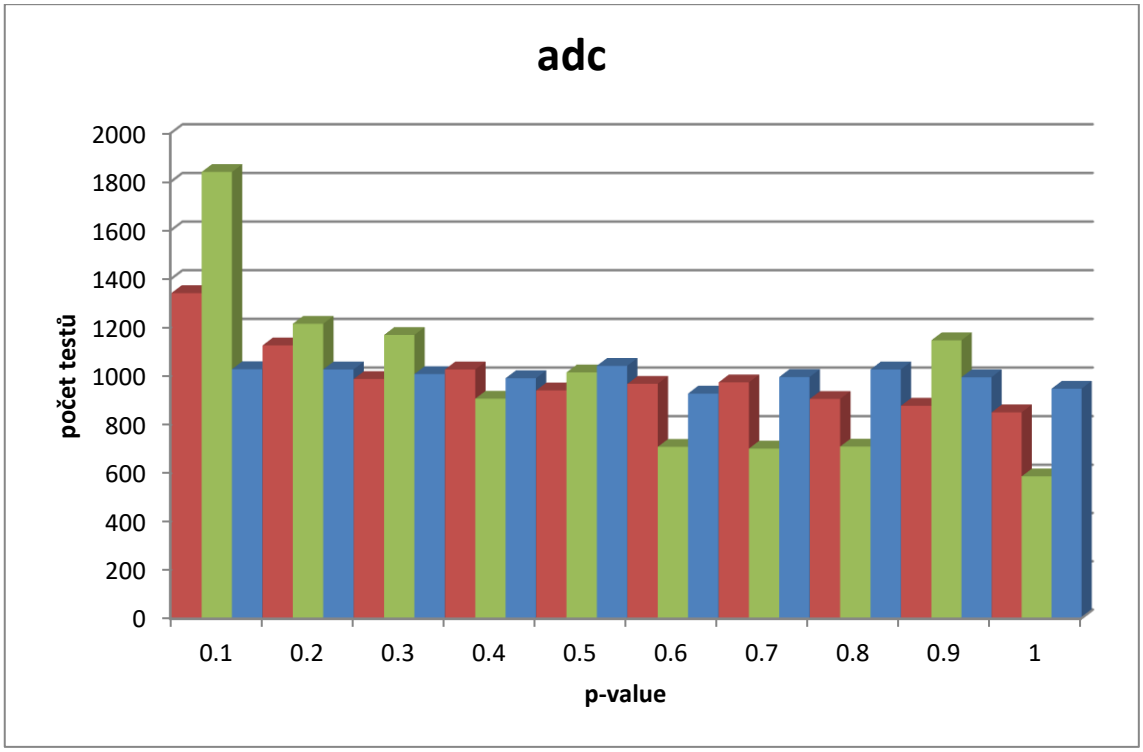
Obrázek 20 - Uživatelské rozhraní testovací baterie NISTu

Pokud výstupní sekvenci generátoru otestujeme jakýmkoliv testem pouze jednou, výsledek může vyjít potvrzující náhodnost nebo nenáhodnost. I funkce rand(), která je bezesporu pseudonáhodná, může úspěšně projít sadou testů. Přesnost výsledku se zvyšuje s počtem opakování, a proto se v praxi sekvence testují několikrát až několiktisíckrát.

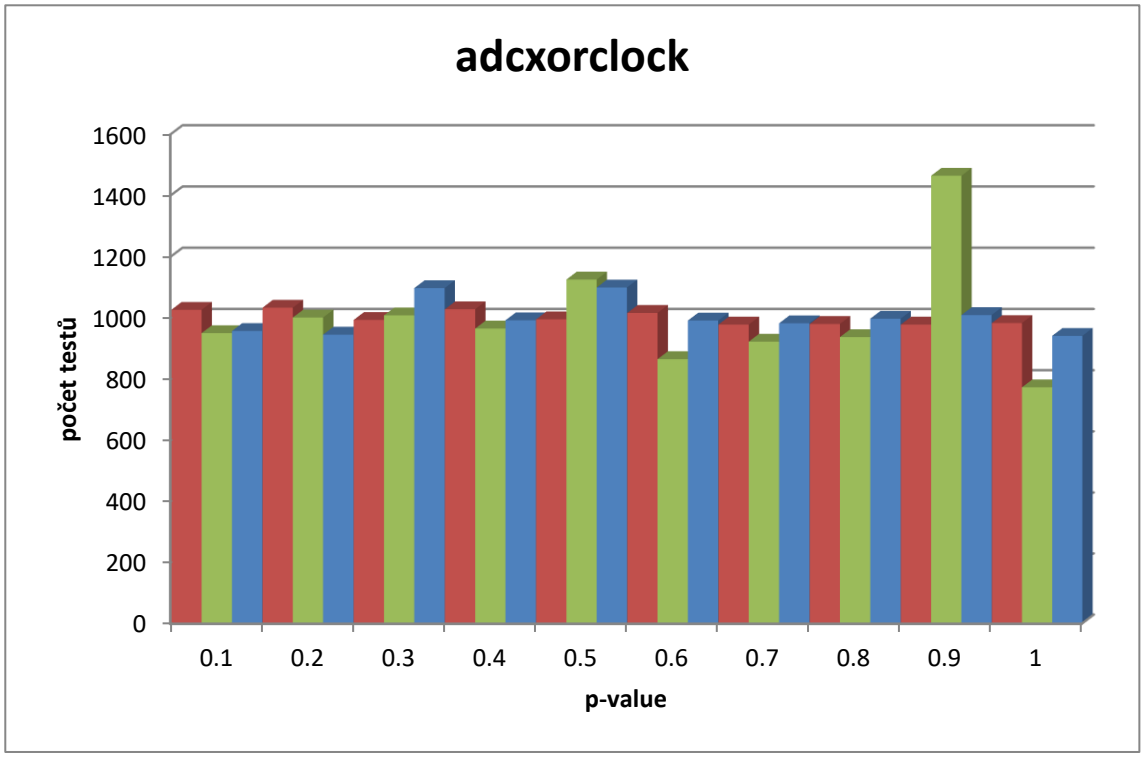
Implementované generátory jsem otestoval i tímto způsobem. Vygenerovaná sekvenci se rozdělí na bloky o velikosti 1000bitů a každá sekvence se otestuje zvlášť. Provede se tak 10000 testů a dostaneme 10000 hodnot p-value. Tyto hodnoty se pak zanesou do grafu a vidíme tak grafické znázornění různých výsledků stejného testu.

Provedl jsem následující testy:

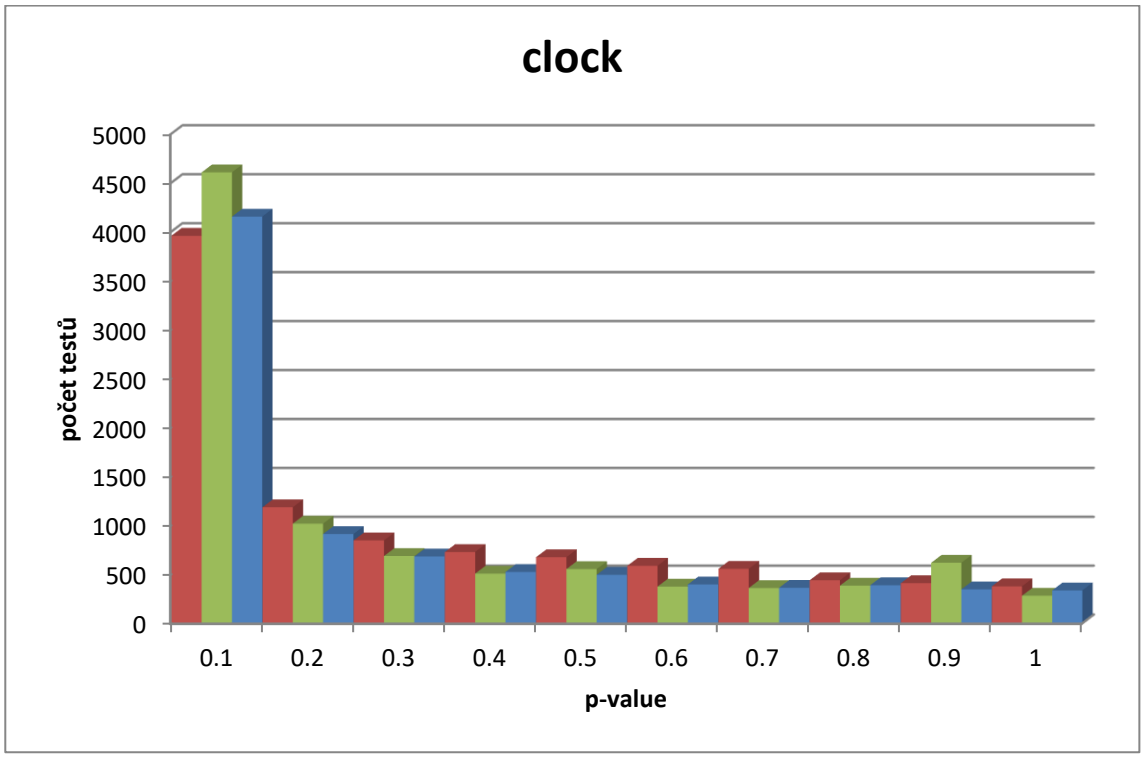
- Blokový frekvenční test (v grafu červená)
- Frekvenční monobit test (v grafu zelená)
- Test nejdelší posloupnosti jedniček (v grafu modrá)



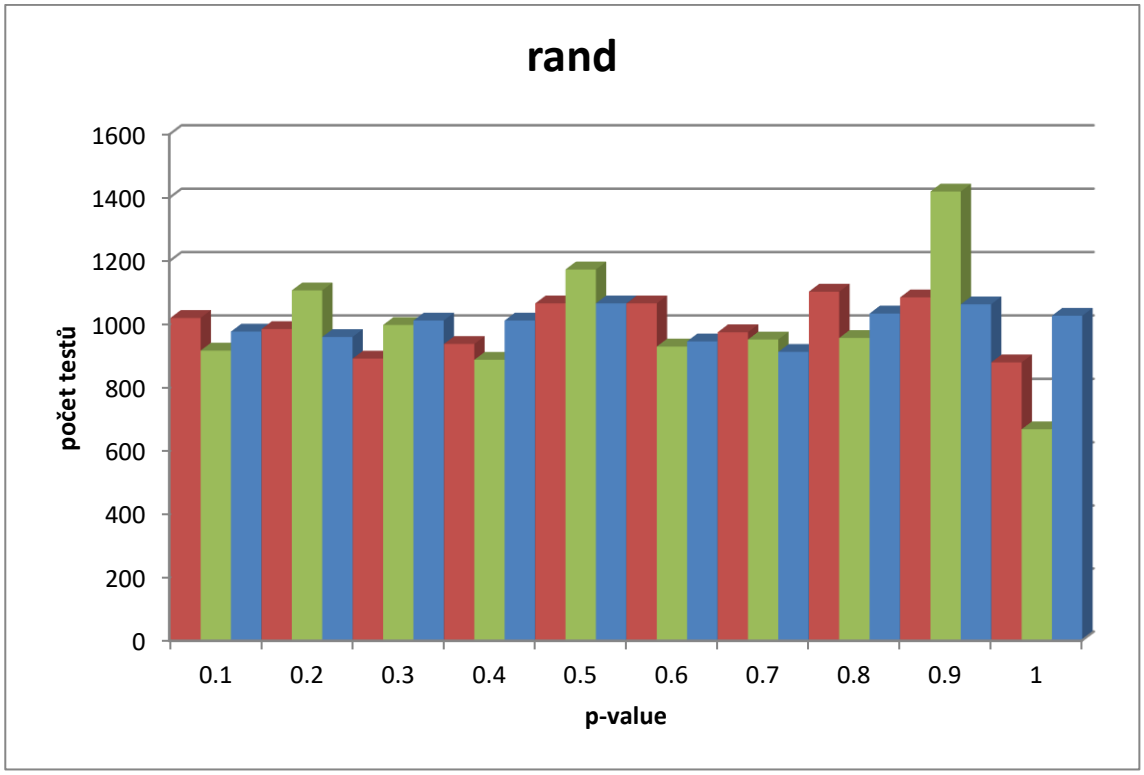
Obrázek 21 - Histogram testů NIST, adc



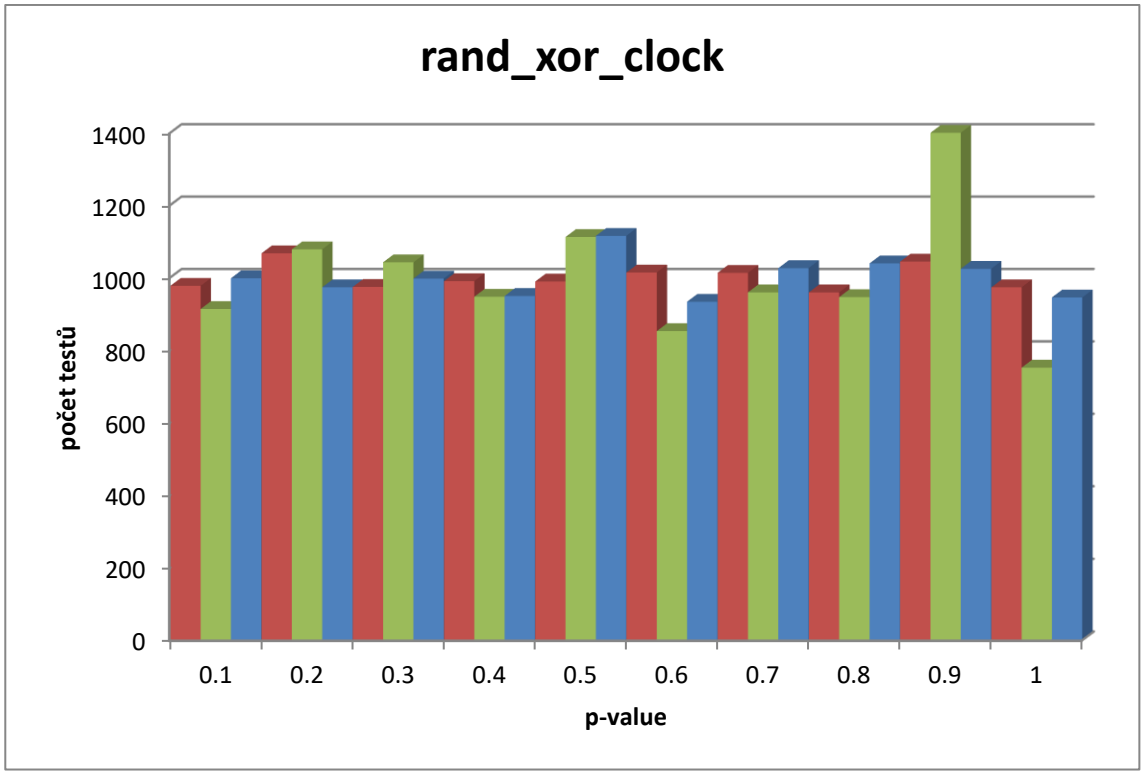
Obrázek 22 - Histogram testů NIST, adcxorclock



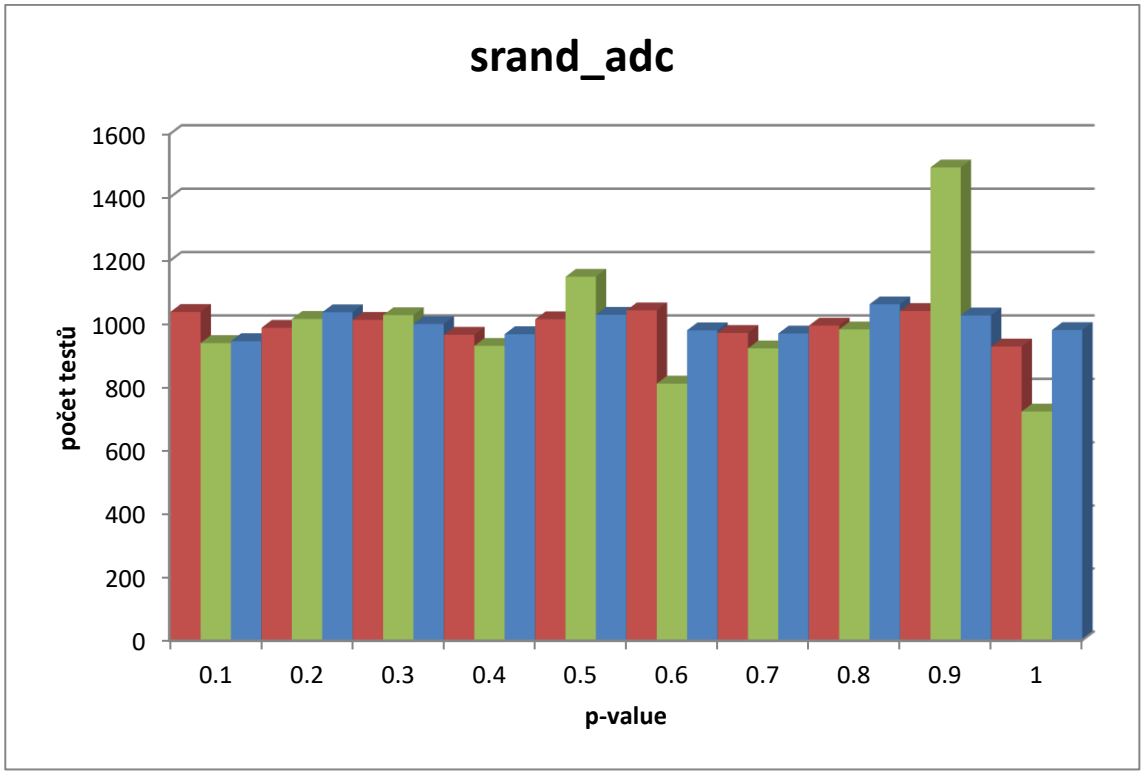
Obrázek 23 - Histogram testů NIST, clock



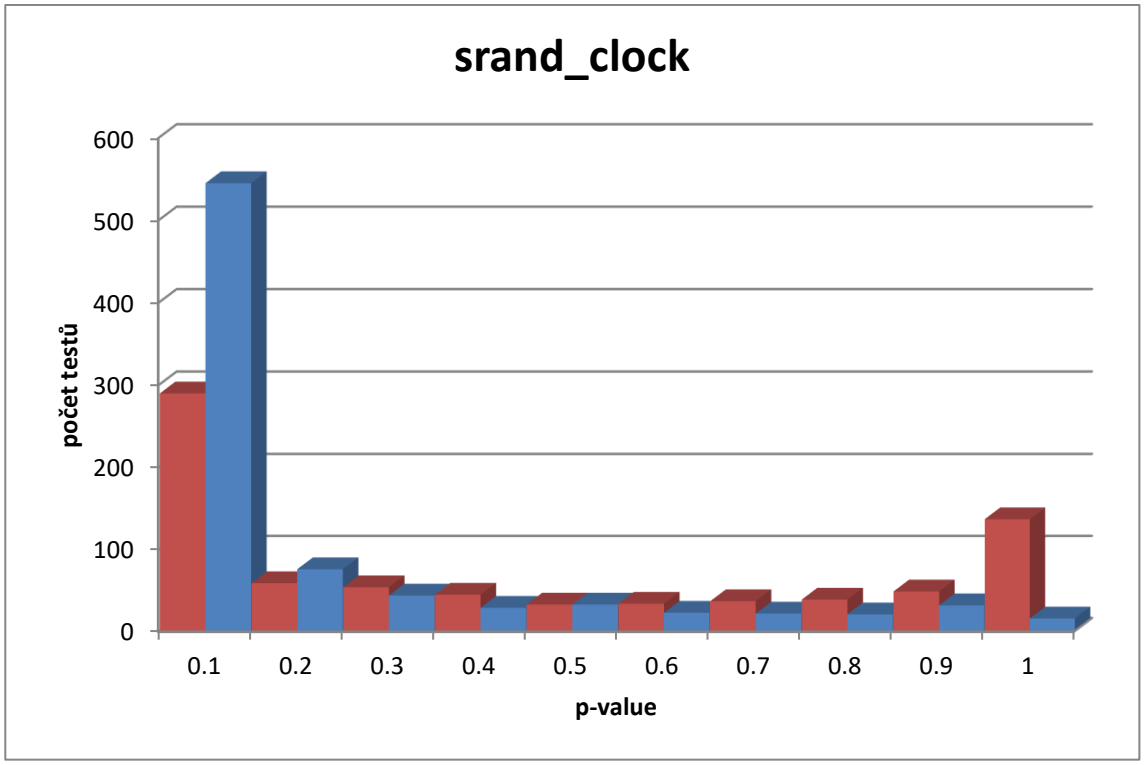
Obrázek 24 - Histogram testů NIST, rand



Obrázek 25 - Histogram testů NIST, randxorclock

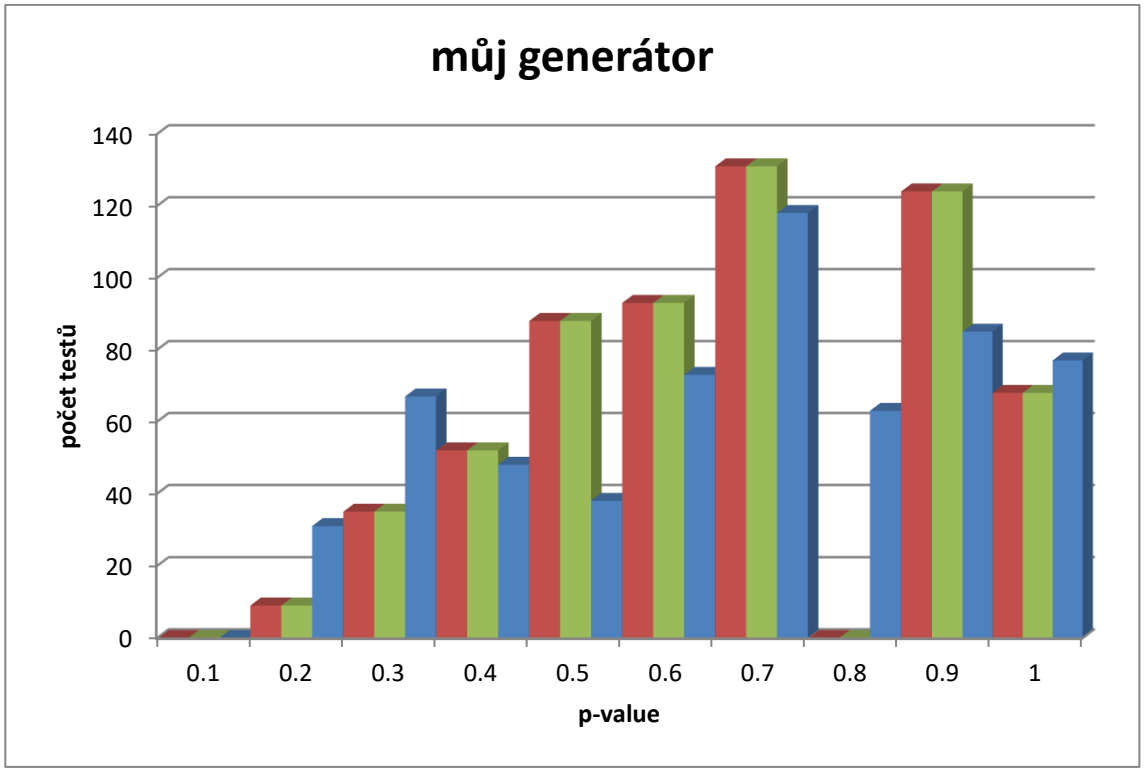


Obrázek 26 - Histogram testů NIST, srand_adc



Obrázek 27 - Histogram testů NIST, srand_clock

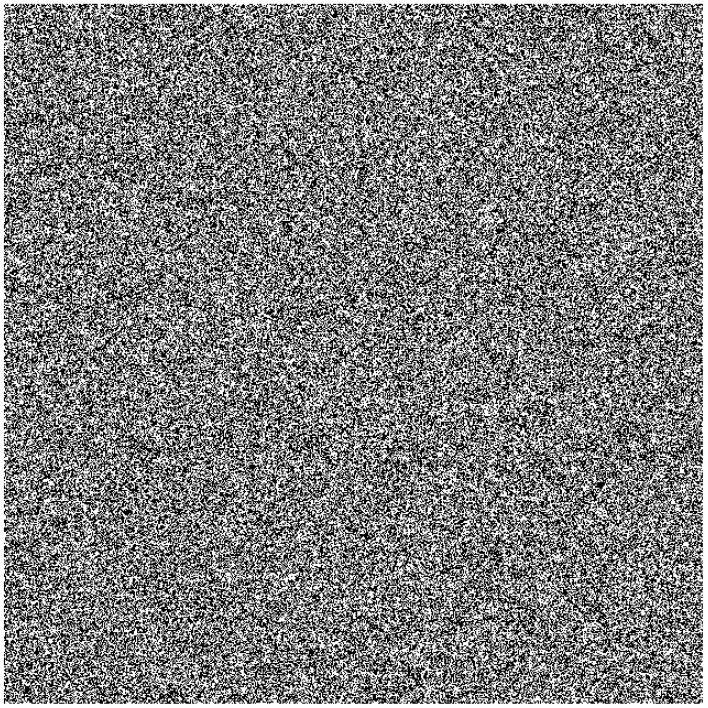
(Test nejdelších shodných bitů skončil nulami a zkreslil by výsledný graf)



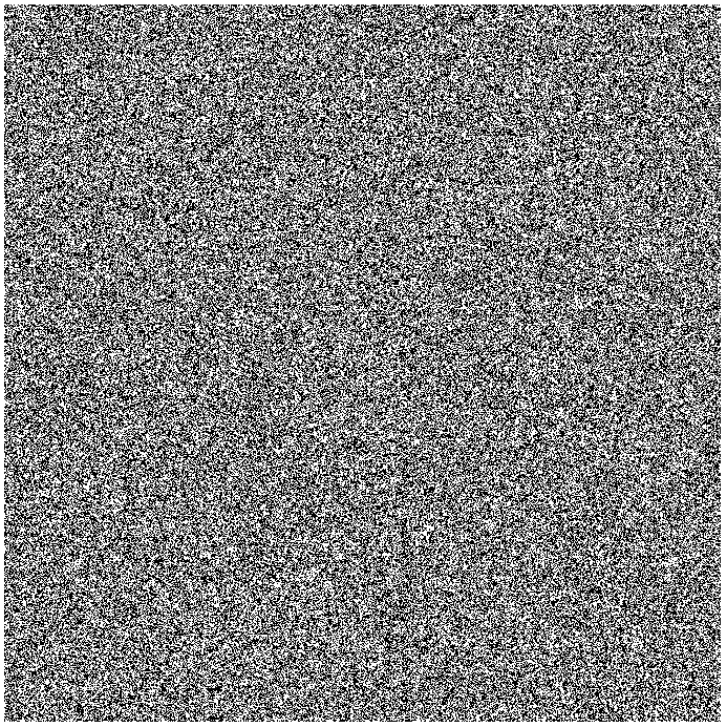
Obrázek 28 - Histogram testů NIST, můj generátor

5.4 Vizuální testy

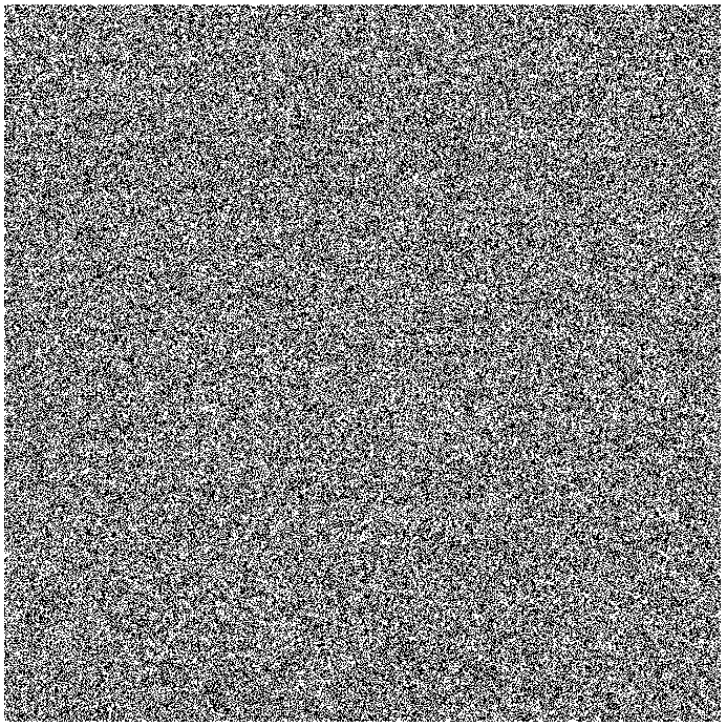
Poslední aplikovanou testovací metodou je vizuální test, který již byl popsán v předchozí kapitole. Test nám dává možnost okamžitě vidět vzory a tvary, které by naznačovaly nenáhodnost generátoru.



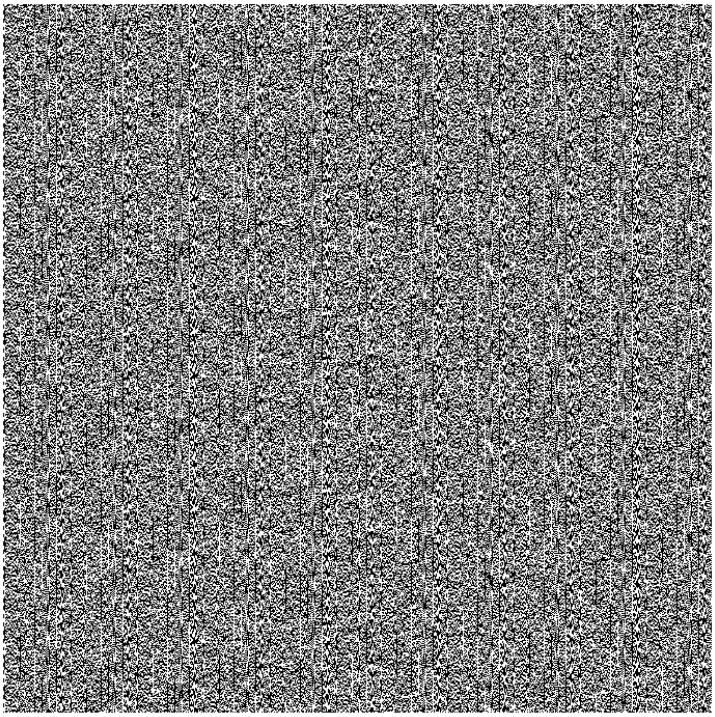
Obrázek 29 - Vizuální test, adc



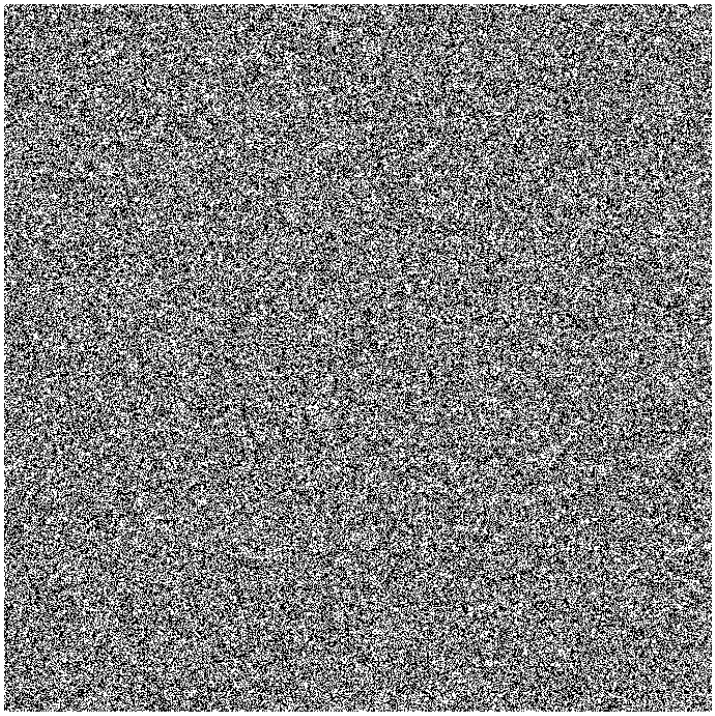
Obrázek 30 - Vizuální test, adcxorclock



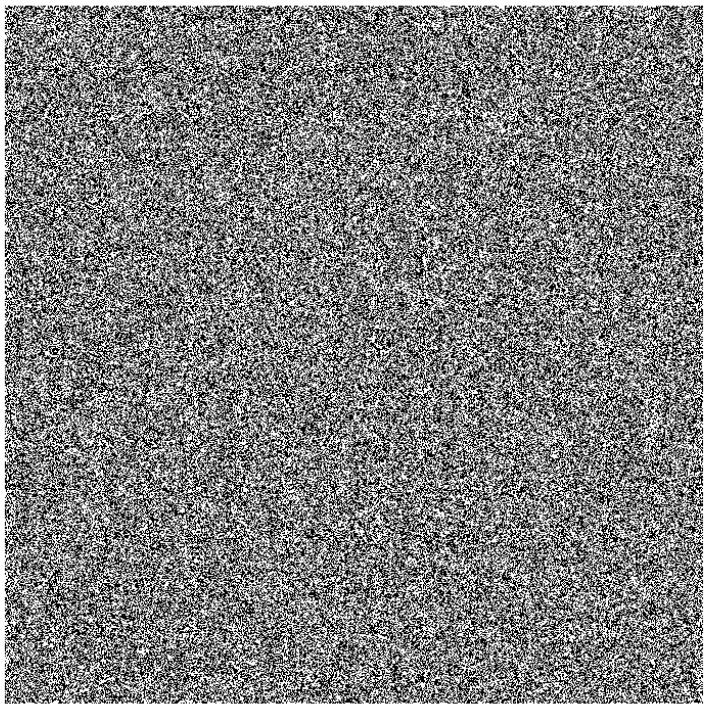
Obrázek 31 - Vizuální test, clock



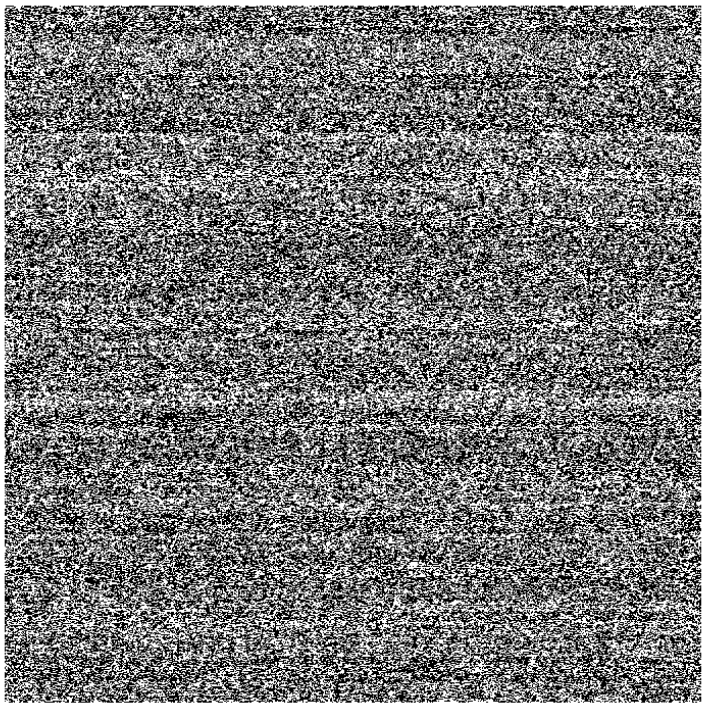
Obrázek 32 - Vizuální test, rand



Obrázek 33 - Vizuální test, randxorclock



Obrázek 34 - Vizuální test, srand_adc



Obrázek 35 - Vizuální test, srand_clock

5.5 Srovnání generátorů

Podíváme-li se na časovou a paměťovou náročnost tak zjistíme, že se generátory o mnoho neliší. Například ADC generátor je jen o 0,22ms rychlejší ve vygenerování jednoho bitu, než Adc_xor_Clock. Při velikosti klíče 256bitů to ale už je 56,32ms, což je nezanedbatelný čas.

5.5.1 Dieharder

Jako první jsem otestoval generátory sadou testů Dieharder. Posuzovat a porovnávat generátory mezi sebou na základě jednoho testu není vhodné (jelikož jeden test ještě nemusí nic říkat o skutečném výsledku), ale můžeme generátory srovnat na základě pravděpodobnostního rozdělení výsledné hodnoty p-value napříč všemi testy. Od generátoru náhodných čísel se očekává právě rovnoměrné rozdělení p-value v intervalu <0,1> takže pokud má dle histogramu jeden generátor rozdělení rovnoměrnější, řekneme, že může být více náhodný.

Dle histogramů je patrné, že nejrovnoměrnější rozdělení hodnot p-value mají generátory **Clock** a **Adc_xor_clock**. Ostatní se nedrží v rovině a mají několik prudkých propadů, což značí prudké výkyvy mezi výsledky a to není chování, které bychom čekali od generátoru náhodných čísel.

5.5.2 Statistical Test Suite

Druhé testy byly z baterie Statistical Test Suite (NIST), kde jsem testoval všech 15 dostupných testů najednou na délku sekvence 1Mbit s počtem bitstreamů 10 (dohromady tedy délka celé sekvence 10Mbit):

- ADC: 159 testů dopadlo s výsledkem p-value
- Clock: 59 testů proběhlo s výslednou p-value
- ADC_xor_clock: 186 testů dopadlo s hodnotou p-value
- Rand: 78 testů dopadlo s hodnotou p-value
- Rand_xor_clock: všech 187 testů dopadlo s hodnotou p-value
- Srand_adc: všech 187 test dopadlo s hodnotou p-value
- Srand_clock: 26 testů dopadlo s hodnotou p-value
- Můj generátor: žádný test nedopadl s hodnotou p-value

Pro většinu testů (kromě random excursion) byla podmínka 8 úspěšných průchodů z desíti. Pokud bylo průchodů méně, celý test o desíti opakováních je označen za nula a nevygeneruje se tedy hodnota p–value. Předpokládáme, že čím kvalitnější generátor, tím více testů by prošlo vyhodnocením hodnoty p-value.

Dle výsledků by nejlépe vycházely generátory **Srand_adc**, **Rand_xor_clock** a **ADC_xor_Clock**.

Dále jsem testoval 3 testy, vždy po délce bloku 1000bit celkovým počtem 10000 testů. Těmi testy byly frekvenční blokový test, frekvenční monobit test a test nejdelší posloupnosti jedniček. Od opravdu náhodné sekvence opět očekáváme rovnoměrné rozdělení hodnot p-value v intervalu <0,1>. Pokud se v některém subintervalu objeví velký propad nebo impuls, znamená to velký výkyv hodnot oproti rovnoměrnému rozdělení.

První generátor, **ADC**, má větší impuls jen v subintervalu <0-0,1>. Ve zbytku histogramu se jednotlivé testy (barvy) drží přibližně ve stejné rovině.

Generátor **ADC_xor_Clock** si stojí dle histogramů lépe. Jednotlivé testy se od sebe příliš neodchylují a celkově se drží kolem jedné úrovně (1000). Avšak také je tu jeden výrazný impuls.

Clock generátor ukázal velké množství p-value v subintervalu <0-0,1> a tím celý histogram dostal logaritmický tvar. Takový histogram není ukázkovým grafem pro náhodný generátor.

Rand i rand_xor_clock ukazují velmi podobné, velké odchylky od hodnoty 1000 a stejný impuls.

Srand_adc vypadá uhlazeněji než rand, jsou zde patrné vlnky, které tvoří maxima hodnoty osy Y.

Srand_clock i mnou vytvořený generátor nemají průběh, jaký bychom očekávali

od náhodného generátoru.

5.5.3 Vizuální testy

Pohledem na vygenerované bitmapové obrázky zjišťujeme, že generátor rand a srand_clock vykazují opakující se vzory, a tudíž pravděpodobně nejsou produktem náhodného generátoru.

ZÁVĚR

Tato práce se zabývala náhodnými jevy a čísly, pravděpodobností a vytvářením náhodných čísel. Tyto náhodná čísla mají velké uplatnění zejména v oboru kryptografie, který nabývá na stále větší významnosti. Náhodná čísla se vytváří pomocí generátorů náhodných čísel, které jsou popsány v první kapitole. Dále je uveden rozdíl, mezi pseudonáhodnými čísly a skutečně náhodnými a dále jsou popsány s tím spojené pseudonáhodné generátory a skutečně náhodné generátory čísel.

Druhá kapitola pojednává o nutnosti ověřování, zdali jsou náhodné čísla skutečné náhodná. V kapitole je čtenář seznámen se statistickou teorií testování pomocí nulové hypotézy a s tím spojeným pojmem „p-value“. V současnosti existuje několik druhů testů, které by dohromady vydaly na celou akademickou práci a některé z těchto testů jsou uvedeny a popsány pro bližší pochopení složitosti testování náhodnosti. Některé testovací sady (nebo též baterie) přejímají testy od jiných sad. Například testovací baterie Dieharder obsahuje některé testy ze sady FIPS140-2, ze sady Statistical Test Suite organizace NIST i testy svého předchůdce Diehard. Z důvodu takovéhoto překrytí testů napříč testovacími sadami, jsou v práci popsány ty testy, jež autor považoval za nejznámější. Velmi zajímavým druhem testu je tzv. vizuální test, který k vyhodnocení nepoužívá procesor, ale lidský mozek. Člověk je totiž zvlášť citlivý na vnímání informací pomocí zraku. Až 80% toho co se dozvíme, se tak stane prostřednictvím našich očí. Vizuální test tedy využívá naší schopnosti vidět a hledat opakující se vzory a tvary. Posloupnost bitů se převede na bitmapový obrázek a o něm pak můžeme rozhodnout, zdali sekvence je, nebo není náhodná. Metoda není tolik spolehlivá jako statistické a matematické testy, ale dává nám rychlou informaci o náhodnosti testované sekvence.

Po popsání testovacích funkcí se dostáváme k jádru diplomové práce, generování náhodných čísel na nízko-výkonových zařízeních. Zatímco kapitola 3.1 nám přiblížila pojem nízko-výkonová zařízení a uvedla některé zástupce, další kapitola už popisuje jen čip společnosti Texas Instrument – MSP430F5438A. Na tento mikrokontrolér jsem implementoval 4 generátory (pseudo)náhodných čísel. Princip fungování všech čtyř generátorů už je znám, avšak tyto generátory nebyly implementovány na MSP430x5 (pátá modelová řada). Po implementaci jsem vytvořil generátor s použitím teplotního digitálního čidla DS18B20 firmy Maxim Integrated. Čidlo je snadno připojitelné k digitálnímu I/O pinu a je napájeno samotným MSP430 při pracovním napětí od 3,3V do 5V. Jako zdroj entropie sloužila změna teploty. Čidlo měří s přesností na 0,0625°C, avšak doba konverze je při maximálním rozlišení velmi vysoká. Spotřeba energie na vygenerování jednoho bitu je cca 0,092mA (spočteno podle doby generování jednoho bitu a doby čekání).

Poslední kapitola této práce se zabývala testováním a porovnáváním všech generátorů. Otestoval jsem generátory testovací sadou Dieharder, STS-NIST a navíc třemi testy ze sady STS-NIST pro vytvoření histogramů.

Při provádění testů náhodnosti a jejich vyhodnocování jsem zjistil, že si pravděpodobně nejlépe vedl generátor ADC_xor_Clock. Při prováděném trojtestu (frekvenční blokový, monobit a test nejdelší posloupnosti jedniček) si vedl stejně anebo lépe, než ADC

generátor. Při celkovém testu STS-NIST si vedl společné s Clock generátorem nejlépe a při testu Diehard dopadl velmi podobně jako ADC generátor.

Generátor ADC_xor_Clock jsem vytvořil jako odpověď na otázku „Mohl by útočník podvrhnout výsledná generovaná čísla připojením pseudonáhodného šumu na vstupní pin A/D převodníku?“. Pokud by útočník pouštěl šum na vstupní pin u ADC generátoru, výsledné hodnoty by náhodné nebyly a útočník by dokázal nastavit například šifrovací klíč, nebo ovlivnit funkci RSA a poté ji prolomit. Pokud by ale útočník takovýmto způsobem útočil na upravený generátor, na ADC_xor_Clock, tak by se jeho pseudonáhodný šum (v této práci simulovaný funkcí rand) stále míchal operací XOR s výstupem z generátoru Clock. Výstup takového generátoru pod útokem jsem nasimuloval vytvořením generátoru Rand_xor_Clock, který si v celkové STS-NIST analýze vedl dobře.

Tato úprava ADC generátoru by mohla být v další práci rozvedena, prověřena a optimalizována. V práci navazující na tento princip by mohly být srovnány dva scénáře. V prvním scénáři se připojí pseudonáhodný šum na vstupní piny ADC generátoru. Ve druhém scénáři se připojí pseudonáhodný šum na vstupní piny ADC_xor_Clock generátoru, a následně se porovnají statistické testy sekvencí od obou generátorů. Tímto testováním by se zjistila využitelnost vylepšeného generátoru pro kryptografické účely.

Výstupem práce je tedy rozbor a implementace stávajících generátorů, vylepšení jednoho vybraného generátoru a dále vývoj nového generátoru založeném na změně teploty.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ROVNIC

SEZNAM LITERATURY

[1] HENNEBERT, Christine, Hicham HOSSAYNI a Cédric LAURADOUX. Entropy harvesting from physical sensors. In: Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks - WiSec '13 [online]. New York, New York, USA: ACM Press, 2013, s. 149- [cit. 2017-05-22]. DOI: 10.1145/2462096.2462122. ISBN 9781450319980. Dostupné z: http://dl.acm.org/citation.cfm?doid=2462096.2462122

[2] Úvod do základních pojmů teorie pravděpodobnosti [online]. s. 10 [cit. 2017-05-22]. Dostupné z: http://labe.felk.cvut.cz/~tkrajnik/kui2/cviceni/sem01/pravdepodobnost.pdf

[3] How To Generate Truly Random Bits [online]. Rick van Rein, 2007 [cit. 2017-05-22]. Dostupné z: http://openfortress.org/cryptodoc/random/

[4] What's this fuss about true randomness? [online]. 2017 [cit. 2017-05-22]. Dostupné z: https://www.random.org/

[5] Kryptografie [online]. [cit. 2017-05-22]. Dostupné z: https://cs.wikipedia.org/wiki/Kryptografie

[6] Generování náhodných čísel [online]. s. 1 [cit. 2017-05-22]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7024

[7] Dá se náhoda měřit?. Počítačová bezpečnost [online]. 2000, s. 2 [cit. 2017-05-22]. Dostupné z: http://crypto-world.info/klima/2000/chip-2000-01-38-39.pdf

[8] RUKHIN, Andrew, Juan SOTO a James NECHVATAL et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [online]. National Institute of Standards and Technology, 2010, , 131 [cit. 2017-05-22]. Dostupné z: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[9] EASTLAKE, Donald E., Stephen D. CROCKER a Jeffrey I. SCHILLER. Randomness Recommendations for Security [online]. , 30 [cit. 2017-05-22]. Dostupné z: ftp://ftp.ietf.org/rfc/rfc1750.txt

[10] CALLEGARI, S., R. ROVATTI a G. SETTI. Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. IEEE Transactions on Signal Processing [online]. 2005, 53(2), 793-805 [cit. 2017-05-22]. DOI: 10.1109/TSP.2004.839924. ISSN 1053-587x. Dostupné z: http://ieeexplore.ieee.org/document/1381779/

[11] Hardwarový generátor náhodných čísel aneb náhoda z atomů [online]. 2010 [cit. 2017-05-22]. Dostupné z: https://www.root.cz/clanky/hardwarovy-generator-nahodnych-cisel-aneb-nahoda-z-atomu/

[12] Hardwarový generátor náhodných čísel [online]. [cit. 2017-05-22]. Dostupné z: http://cs.dbpedia.org/page/Hardwarový_generátor_náhodných_čísel

[13] ChaosKey: skutečný generátor náhodných čísel do USB [online]. 2016 [cit. 2017-05-22]. Dostupné z: https://www.root.cz/clanky/chaoskey-skutecny-generator-

nahodnych-cisel-do-usb/

[14] TKACIK, Thomas E. A Hardware Random Number Generator [online]. s. 450 [cit. 2017-05-22]. DOI: 10.1007/3-540-36400-5_32. Dostupné z: http://link.springer.com/10.1007/3-540-36400-5_32

[15] RUBIN, Jason M. Can a computer generate a truly random number? [online]. [cit. 2017-05-22]. Dostupné z: http://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

[16] About NIST [online]. 2015 [cit. 2017-05-22]. Dostupné z: https://www.nist.gov/about-nist

[17] Dieharder - A testing and benchmarking tool for random number generators [online]. [cit. 2017-05-22]. Dostupné z: http://manpages.ubuntu.com/manpages/precise/man1/dieharder.1.html

[18] Dieharder test descriptions [online]. [cit. 2017-05-22]. Dostupné z: https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions

[19] KLÍMA, Vlastimil. Statistické testy NIST. Kryptologie pro praxi [online]. 2010, , 1 [cit. 2017-05-22]. Dostupné z: http://cryptography.hyperlink.cz/2011/ST_2011_02_23_23.pdf

[20] Guide to the Statistical Tests [online]. [cit. 2017-05-22]. Dostupné z: http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html

[21] ENT - Program pro testování sekvencí pseudonáhodných čísel [online]. 2005 [cit. 2017-05-23]. Dostupné z: https://www.root.cz/clanky/ent-program-pro-testovani-sekvenci-pseudonahodnych-cisel/

[22] Entropy and Random Number Generators [online]. 2017 [cit. 2017-05-23]. Dostupné z: https://calomel.org/entropy_random_number_generators.html

[23] WALKER, John. ENT A Pseudorandom Number Sequence Test Program [online]. 2008, , 1 [cit. 2017-05-23]. Dostupné z: http://fourmilab.ch/random/

[24] TKACIK, Thomas E. A Hardware Random Number Generator [online]. s. 450 [cit. 2017-05-22]. DOI: 10.1007/3-540-36400-5_32. Dostupné z: http://link.springer.com/10.1007/3-540-36400-5_32

[25] BROWN, Robert G., Dirk EDDELBUETTEL a David BAUER. Dieharder: A Random Number Test Suite [online]. 2017 [cit. 2017-05-23]. Dostupné z: https://www.phy.duke.edu/~rgb/General/dieharder.php

[26] CSE Information Kit [online]. [cit. 2017-05-23]. Dostupné z: https://www.cse-cst.gc.ca/en/media/information

[27] Are the Numbers Really Random? [online]. 2017 [cit. 2017-05-23]. Dostupné z: https://www.random.org/analysis/

[28] Mat2gray [online]. [cit. 2017-05-23]. Dostupné z: https://www.mathworks.com/help/images/ref/mat2gray.html?requestedDomain=www.mathworks.com&requestedDomain=www.mathworks.com

[29] Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable

Flash [online]. Atmel Corporation, 2013 [cit. 2017-05-23]. Dostupné z: http://www.atmel.com/images/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf

[30] MSP430F2272: 16-bit Ultra-Low-Power Microcontroller, 32KB Flash, 1K RAM [online]. [cit. 2017-05-23]. Dostupné z: http://www.ti.com/product/MSP430F2272

[31] MC9S08AC128 8-Bit Microcontroller Data Sheet [online]. Freescale Semiconductor, 2008 [cit. 2017-05-23]. Dostupné z: http://pdf1.alldatasheet.com/datasheet-pdf/view/246019/FREESCALE/MC9S08AC128.html

[32] MSP430F5438A: 16-Bit Ultra-Low-Power Microcontroller, 256KB Flash, 16KB RAM, 12 Bit ADC, 4 USCIs, 32-bit HW Multi [online]. 2010 [cit. 2017-05-23]. Dostupné z: http://www.ti.com/product/MSP430F5438A

[33] JUN, Benjamin; KOCHER, Paul. The Intel random number generator. Cryptography Research Inc. white paper, 1999.

[34] WESTLUND, Lane. Random Number Generation Using the MSP430 [online]. Texas Instruments Incorporated, 2006, , 4 [cit. 2017-05-23]. Dostupné z: http://www.ti.com/lit/an/slaa338/slaa338.pdf

[35] Code Composer Studio (CCS) Integrated Development Environment (IDE) [online]. [cit. 2017-05-23]. Dostupné z: http://www.ti.com/tool/ccstudio

[36] CALLEGARI, S., R. ROVATTI a G. SETTI. Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. IEEE Transactions on Signal Processing [online]. 2005, 53(2), 793-805 [cit. 2017-05-22]. DOI: 10.1109/TSP.2004.839924. ISSN 1053-587x. Dostupné z: http://ieeexplore.ieee.org/document/1381779/

[37] Menezes, Alfred; van Oorschot, Paul C.; Vanstone, Scott A. (October 1996). Handbook of Applied Cryptography. CRC Press. ISBN 0-8493-8523-7

SEZNAM POUŽITÝCH ZKRATEK, VELIČIN A SYMBOLŮ

A/D (AD) analog digital
ACLK auxiliary clock
ADC analog to digital converter
CPU central processing unit
CSE Security Establishment Canada
DCO Digitálně řízený oscilátor
ENT název testovaciho programu
erfc chybová funkce
H0 nulová hypotéza
Ha alternativní hypotéza
I/O input output
igamc nekompletní gamma funkce
IoT internet of things
LSB least significant bit
MCLK master clock
MSB most significant bit
NIST National Institute of Standards and Technology
PRNG pseudorandom number generator
RAM random-access memory
REFO referenční hodiny
RSA Rivest, Shamir, Adleman
SAR Successive approximation
SMCLK submain clock
TI Texas Instruments
TRNG true random number generator
VLO very-low-frequency oscillator

SEZNAM PŘÍLOH

Přílohy jsou umístěny na přiloženém nosiči

OBSAH PŘILOŽENÉHO NOSIČE

Dieharder_vysledky

Sts_vysledky

Vizualni_testy

Vygenerovane_sekvence:10M

Zdrojove_kody