VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

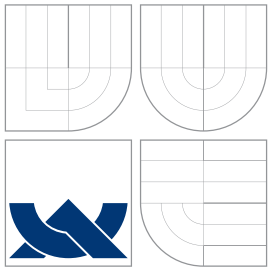
GENERÁTOR NÁHODNÝCH ČÍSEL SO ZVOLENÝM ROZLOŽENÍM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

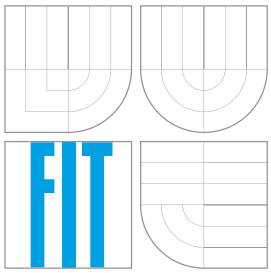
AUTOR PRÁCE
AUTHOR

MICHAL KAJAN

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GENERÁTOR NÁHODNÝCH ČÍSEL SE ZVOLENÝM ROZLOŽENÍM

RANDOM NUMBERS GENERATOR WITH SELECTED DISTRIBUTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL KAJAN
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK
SUPERVISOR

BRNO 2007

Zadanie bakalárskej práce

1. Zoznámte sa s kartou COMBO6 a technológiou Virtex-II od firmy Xilinx.
2. Naštudujte problematiku generovania náhodných čísel.
3. Navrhnite generátory náhodných čísel pre vybrané rozdelenie pravdepodobnosti.
4. Vykonajte implementáciu generátorov v jazyku VHDL a overte ich funkciu.
5. V závere uveďte hardwarové zdroje nutné pre realizáciu generátorov a maximálnu dosiahnuteľnú frekvenciu.

Licenčná zmluva

Licenčná zmluva je uložená v archíve Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Táto práca popisuje problematiku generovania pseudonáhodných čísel. V prvej časti sú priblížené spôsoby získavania pseudonáhodných postupností a uvedené typické príklady generátorov. Táto časť takisto obsahuje popis transformačných metód rozložení pseudonáhodných čísel a stručne pojednáva o testovaní vlastností generátorov náhodných čísel. V nasledujúcej časti je detailne rozobratý generátor typu LFSR, ktorý je najpoužívanejším typom generátoru pre hardvérové aplikácie. Nasleduje popis transformácie a implementácia obvodu, ktorý realizuje transformáciu do exponenciálneho rozloženia. Na záver sú uvedené hardvérové požiadavky pre realizáciu navrhnutých obvodov v FPGA.

Kľúčové slová

generátor pseudonáhodných čísel, transformácia rozložení, LFSR, FPGA, VHDL, FITkit

Abstract

This thesis describes random numbers generating techniques. First part focuses on methods of obtaining pseudorandom numbers and presents typical examples of random numbers generators. This part also contains description of distribution transformation methods of random numbers and briefly deals with testing of statistical properties of random numbers generators. Following part describes LFSR generator in detail as one of most widely used generators for hardware applications. In addition, description of transformation process and implementation of circuit calculating transformation to the exponential distribution is included. Last part contains resources requierements of designed circuits for implementation in FPGA.

Keywords

pseudorandom numbers generator, distribution transformation, linear feedback shift register, FPGA, VHDL, FITkit

Citácia

Michal Kajan: Generátor náhodných čísel se zvoleným rozložením, bakalářská práce, Brno, FIT VUT v Brně, 2007

Generátor náhodných čísel so zvoleným rozložením

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jana Kořenka a uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Kajan
14. mája 2007

Poďakovanie

Rád by som sa poďakoval vedúcemu práce Ing. Janu Kořenkovi za odbornú pomoc a neoceniteľné rady, ktoré mi poskytol pri tvorbe tejto práce. Moje poďakovanie patrí aj mojim rodičom za ich podporu v mojom doterajšom štúdiu.

Obsah

1 Úvod **2**

2 Teoretický rozbor **4**
2.1 Zdroje náhodných čísel 4
2.2 Generovanie skutočne náhodných čísel 5
2.3 Generovanie pseudonáhodných čísel 5
2.3.1 Lineárny kongruentný generátor 6
2.3.2 LFSR — Lineárny spätnoväzbový posuvný register 7
2.3.3 Mersenne Twister 10
2.4 Metódy transformácie rozložení 11
2.5 Testovanie vlastností generátorov 13

3 Architektúra generátoru náhodných čísel **16**
3.1 LFSR register s paralelným vstupom a výstupom 16
3.2 LFSR register so sériovým vstupom a výstupom 17
3.3 Viacnásobný LFSR 19
3.4 Generátor VHDL kódu pre LFSR registre 19

4 Transformácie rozložení **21**

5 Implementácia do prípravku FITkit **25**

6 Dosiahnuté výsledky **27**
6.1 Test dobrej zhody 27
6.2 Spotreba zdrojov na čipe 28

7 Záver **29**

Zoznam príloh **34**

A Prehľad primitívnych polynómov **35**

B Použitie core generátoru **36**

C Ukážky priebehu činnosti **37**

D Použitie aplikácie pre FITkit **38**

Kapitola 1

Úvod

Vývoj výpočtovej a komunikačnej techniky priniesol v posledných desaťročiach úplne nové možnosti vo všetkých oblastiach technickej praxe. Výpočtová technika sa stala účinným nástrojom pre realizáciu a overovanie množstva teoretických úvah. Prudký nárast výkonnosti počítačov a predovšetkým ich dostupnosť umožnili výrazný vývoj v oblasti modelovania a simulácií, počítačovej komunikácie, spracovania dát a ich ochrany, a množstva ďalších. Dostupný výkon umožnil realizáciu náročných numerických výpočtov a rozvoj nových techník v modelovaní a simuláciách. Zároveň však pri modelovaní reálnych procesov vznikla potreba dodania určitého prvku náhodnosti do výpočtov. Podobná situácia nastala v oblasti komunikácií, spracovania signálov, prenosu a zabezpečenia dát a stále dokonalejších techník šifrovania, kde sú prvky náhodnosti neoddeliteľnou a veľmi dôležitou súčasťou týchto procesov.

Z uvedených dôvodov sa začala rozvíjať teória zameraná na návrh generátorov náhodných čísel. V princípe existujú dva základné spôsoby generovania náhodných hodnôt. Jedným sú generátory skutočne náhodných čísel, ktoré však majú veľmi špecifické vlastnosti a sú použiteľné pre veľmi úzky okruh aplikácií, kde je práve samotná náhodnosť jednou z najdôležitejších vlastností v porovnaní s inými požiadavkami, ktoré sú na generátory kladené. Ďalšou kategóriou sú deterministické generátory pseudonáhodných čísel založené na algoritmickom princípe. Vlastnosti tejto skupiny generátorov ich predurčujú pre výrazne väčšie množstvo aplikácií, kde sa kladie dôraz na rýchlosť generovania a jednoduchosť implementácie.

Návrh a realizácia generátorov na softvérovej aplikačnej úrovni sa s rozvojom programovacích jazykov a ich možností stala typickým príkladom využitia. Ako najznámejších zástupcov možno uviesť lineárny kongruentný generátor a jeho rôzne varianty, ktorého princíp je známy desaťročia či Mersenne Twister ako jeden z novších, pričom pre oba je dostupné veľké množstvo ukážkových zdrojových kódov v najpoužívanejších programovacích jazykoch.

Mnohé aplikácie vyžadujúce pre svoju činnosť postupnosť náhodných čísel však môžu mať veľké požiadavky na rýchlosť a kvalitu takýchto generátorov. Aj napriek vysokému výkonu súčasných počítačov v porovnaní s minulosťou, však môže byť použitie generátoru pracujúceho na programovej úrovni limitujúcim faktorom činnosti týchto aplikácií. Preto sa zaujímavou alternatívou v dnešnej dobe stáva oblasť programovateľných hradlových polí FPGA, ktoré umožňujú hardvérovú implementáciu navrhnutých algoritmov a tým aj dosiahnutie zvýšenia výkonu takto realizovanej aplikácie. Naviac je možné pri návrhu využiť špeciálne programovacie jazyky, čím sa stáva popis obvodu jednoduchší a rýchlejší.

Pre obvodové implementácie generátorov možno ako jeden z najznámejších príkladov

uviesť lineárny spätnoväzbový register LFSR (*angl. Linear Feedback Shift Register*). Je dobre teoreticky spracovaný a má veľmi jednoduchú štruktúru. Register tohto typu je zároveň výhodnou voľbou v prípade realizácie pre čip FPGA, preto sa táto práca zameriava na generovanie náhodných čísel týmto typom generátoru a jeho rôznymi variantami tak, aby bol použiteľný v čo najširšom spektre aplikácií.

Druhá kapitola tejto práce obsahuje teoretický rozbor problematiky generovania náhodných čísel, prezentuje najznámejších predstaviteľov generátorov náhodných čísel pre programovú a obvodovú implementáciu a ich vlastnosti. Výstupom generátorov náhodných čísel je obvykle postupnosť hodnôt v rovnomernom rozložení. Pretože pre niektoré aplikácie je potrebné dodať hodnoty, ktoré spĺňajú vlastnosti určitého rozloženia, sú v tejto kapitole rozobraté mechanizmy transformácie rovnomerného rozloženia na iné typy rozložení.

Pri popisovaní spôsobov generovania náhodných čísel nemožno vynechať ani metódy overovania vlastností použitých generátorov. Keďže overovanie štatistických vlastností je veľmi rozsiahlou oblasťou, táto časť obsahuje iba stručné informácie o jej rôznych možnostiach. Detailnejšie je popísané teoretické pozadie testu dobrej zhody, tzv. chí-kvadrát (χ^2), ktorý sa používa na overenie faktu, že generovaná postupnosť náleží určitému rozloženiu.

V tretej kapitole na základe teórie uvedenej v predchádzajúcej časti je detailne popísaná architektúra generátoru náhodných čísel a jeho rôznych možných alternatív. Z teoretických východísk pre transformáciu jedného typu rozloženia na iný bola navrhnutá architektúra obvodu, ktorý túto transformáciu realizuje. Štvrtá kapitola detailne pojednáva o architektúre navrhnutého obvodu. Samotná architektúra tranformačného bloku má zreťazenú štruktúru (*pipeline*), čo umožňuje veľmi efektívny spôsob generovania transformovaných hodnôt prakticky v každom cykle hodinového signálu. Z existujúcich rozložení bolo vybraté exponenciálne rozloženie, nič však nebráni tomu, aby pre daný generátor boli realizované aj iné typy rozložení.

Z dôvodu overenia funkčnosti bola vykonaná implementácia generátoru do prípravku FITkit, ktorá je popísaná v piatej kapitole.

V poslednej kapitole sú zhrnuté dosiahnuté výsledky. Jednak sú v nej uvedené príklady požiadaviek navrhnutých obvodov na hardvérové zdroje, ktoré sú potrebné pre ich praktickú realizáciu a zároveň maximálna dosiahnuteľná frekvencia. V rámci overenia vlastností generovanej postupnosti v exponenciálnom rozložení bol vypočítaný chí-kvadrát.

Kapitola 2

Teoretický rozbor

Problematika generovania náhodných čísel zasahuje do mnohých oblastí využitia v technickej praxi. Medzi najznámejšie príklady patrí oblasť modelovania a simulácií, riešenia úloh v numerickej analýze pomocou metódy Monte Carlo, počítačovej grafiky, dátovej komunikácie, rozhodovania a takisto aj bezpečnosti a kryptografie.

Každý z týchto spôsobov má odlišné požiadavky na vlastnosti použitého generátoru, všeobecne však možno vlastnosti zhrnúť do niekoľkých bodov. Perióda generátoru by mala byť čo najdlhšia, krátka perióda môže spôsobovať problémy napríklad pri realizovaní simulačných experimentov. Ďalšou dôležitou vlastnosťou je rýchlosť generovania, ktorá vplýva na rýchlosť realizovaných výpočtov v konkrétnej aplikácii — čas generovania by mal byť zanedbateľný v porovnaní s dobou trvania operácií, ktoré generovanú postupnosť využívajú. Generovanie postupnosti čísel by malo byť reprodukovateľné, takže pri zadaní rovnakých vstupných parametrov získame rovnakú postupnosť čísel pri opakovanom použití daného generátoru. Generovaná postupnosť čísel by mala vykazovať dobré štatistické vlastnosti ako sú rovnomernosť rozloženia a nezávislosť medzi generovanými hodnotami. V neposlednom rade by mal byť použitý generátor rýchly a nenáročný na výpočtové zdroje a pamäť.

V nasledujúcej časti si priblížime rôzne možnosti získavania postupnosti náhodných hodnôt.

2.1 Zdroje náhodných čísel

Pri potrebe použitia náhodných čísel je nutné zaoberať sa otázkou získavania postupnosti takýchto hodnôt, predovšetkým s ohľadom na uvedené vlastnosti generátorov a typ aplikácie, pre ktorú sa náhodné čísla použijú. Za najjednoduchší generátor možno považovať obyčajnú hraciu kocku, ktorá pri generovaní poskytuje náhodné číslice v rozpätí od 1 do 6. Je však zrejmé, že takýto generátor je prakticky nepoužiteľný pre väčšinu aplikácií hlavne pre jeho nízku rýchlosť a malý počet generovaných číslic.

Ďalším príkladom jednoduchého zdroja náhodných čísel sú tabuľky. Použitie takejto tabuľky je však vhodné iba v prípade potreby získania malého súboru náhodných hodnôt. Je síce možné neustále zväčšovanie obsahu takejto tabuľky, to však má za následok enormné požiadavky na pamäťový priestor. Tento spôsob je teda pre praktické potreby takisto nevhodný.

2.2 Generovanie skutočne náhodných čísel

Generovanie skutočne náhodných čísel je realizované pomocou fyzikálnych generátorov. Takýto generátor predstavuje určité zariadenie pracujúce na princípe snímania hodnôt nejakého fyzikálneho procesu. Medzi tieto procesy možno zaradiť rozpad rádioaktívnych častíc (všeobecne sa generátory tohto typu považujú za najkvalitnejšie), atmosférický šum[1], zmeny v úrovni napätí a prúdu na prechodoch polovodičových prvkov, zmeny teploty na takýchto prvkoch, vzorkovanie signálu nestabilného oscilátoru[2]...

Je zrejmé, že sa jedná o nedeterministické procesy, pri ktorých je prakticky nemožné odhadnúť ďalšie správanie či rekonštruovať predchádzajúci stav. Z prístupnejších foriem zdrojov neurčitosti možno spomenúť sledovanie výstupu jedného (či viacerých) mikrofónov, snímanie obrazového a zvukového signálu kamerou a zvukovou kartou (možno však využiť aj šum spracovávajúceho zariadenia), časovanie operácií pri práci s pevným diskom, prevádzka na sieťovom rozhraní, štatistika bežiacich procesov, intervaly medzi stlačenými klávesmi, prípadne hodnoty týchto kláves a doba stlačenia týchto kláves, zmeny polohy kurzoru pri pohybe myšou a iné. Predovšetkým prvé zo spomenutých spôsobov vyžadujú použitie špeciálneho technického vybavenia či externé prídavné zariadenia, preto ich praktická použiteľnosť v bežných aplikáciách je pomerne obmedzená. Tieto typy generátorov sa vyznačujú dobrými štatistickými vlastnosťami, je však pre ne typická nízka rýchlosť generovania, ktorá dosahuje hodnoty rádovo v bitoch/s.

Zaujímavým príkladom z tejto kategórie generátorov, ktorý nevyžaduje žiadne prídavné zariadenie, je napríklad generátor implementovaný v jadre operačného systému Linux. Je založený na sledovaní systémových udalostí a jeho výstup sa používa pri vytváraní hesiel, generovaní privátnych SSL kľúčov, určovaní sekvenčných čísel pri TCP komunikácii, či jednoduchom generovaní náhodných celočíselných hodnôt. Prístup ku generátoru je možný prostredníctvom zariadení `/dev/random` a `/dev/urandom`. Prvé zariadenie je blokujúce, takže môže dôjsť k zablokovaniu programu pri čítaní z tohto zariadenia, ak sa vyskytne nedostatok entropie. Druhé zariadenie je neblokujúce, takže prostredníctvom neho možno bez prerušenia realizovať generovanie pseudonáhodných dát. Tento výstup je však považovaný za menej bezpečný. Ďalšie zaujímavé informácie o tomto generátore je možné získať v [29].

2.3 Generovanie pseudonáhodných čísel

Okrem svojich nesporných výhod majú generátory skutočne náhodných čísel aj niekoľko nedostatkov, ktoré obmedzujú ich použitie na úzky rozsah aplikácií. Skutočnosť, že generovaná postupnosť dát je naozaj náhodná, môže byť zároveň nevýhodou. Takáto postupnosť je neopakovateľná, čo znemožňuje použitie takéhoto generátoru napríklad pre simulačné experimenty. Ladenie programov či porovnávanie výsledkov testovania je potom veľmi náročné. Takisto stabilita takéhoto generátoru závisí na vonkajších vplyvoch a z dlhodobého hľadiska je ťažko udržateľná. Nakoniec, skutočné správanie generátoru sa môže líšiť od očakávaného v dôsledku výrobných tolerancií a pod.

Generátor pseudonáhodných čísel je deterministický algoritmus, ktorý na základe počiatočnej hodnoty produkuje postupnosť hodnôt. Takto generovaný výstup má pritom vlastnosti podobné skutočne náhodným hodnotám. Fakt, že generovanie náhodných čísel možno popísať ako určitý výpočet vo forme algoritmu, potom umožňuje jeho bezproblémovú

[1] Týmto spôsobom funguje služba na serveri www.random.org, ktorá je voľne prístupným zdrojom skutočne náhodných čísel.

[2] Takýto generátor je možné vytvoriť aj v FPGA čipe, ako je popísané v [7]

softvérovú či hardvérovú implementáciu a dobre navrhnuté generátory spĺňajú požiadavky uvedené v úvode tejto kapitoly. V ďalšom texte si uvedieme príklady generátorov tohto typu a tiež, ktoré sa typicky používajú pre softvérové implementácie a ktoré sú vhodné pre obvodovú implementáciu.

2.3.1 Lineárny kongruentný generátor

Tento typ generátoru (angl. *Linear Congruential Generator — LCG*) patrí medzi najznámejšie, je dobre teoreticky zmapovaný[3] a bol navrhnutý D. H. Lehmerom už v roku 1948.

Je založený na myšlienke, že pseudonáhodné hodnoty je možné generovať rekurentným vzťahom typu

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

Operácia *mod* znamená zvyšok po celočíselnom delení. Označenie generátoru *kongruentný* vychádza z poznatku, že ak m je prirodzené číslo, potom čísla a a b sú kongruentné modulo m, ak majú po delení číslom m ten istý zvyšok. Označenie *lineárny* je zrejmé. Formálne ho možno označiť ako $LCG(m, a, b, x_0)$, kde:

- m predstavuje modul
- a je multiplikatívna konštanta
- b reprezentuje aditívnu konštantu
- x_0 je počiatočná hodnota (semienko, angl. *seed*)

Uvedené konštanty nie je možné voliť ľubovoľne, pretože ich výber priamo ovplyvňuje vlastnosti generovanej postupnosti ako sú perióda a rozloženie.

Voľba konštanty m vplýva na dĺžku periódy a rýchlosť generovania postupnosti náhodných čísel. Samozrejme, cieľom je dosiahnuť čo možno najdlhšiu periódu a takisto aj rýchlosť. Obvykle sa teda pre modul m volí čo možno najväčšia hodnota — typicky je to maximálna hodnota typu unsigned integer (všeobecne 2^n, kde n je počet bitov čísla tohto typu). Voľba takejto hodnoty potom pozitívne ovplyvní rýchlosť, pretože takto nie je nutné operáciu modulo na počítači vykonávať, keďže je v podstate implicitnou vlastnosťou celočíselných výpočtov. Inak by túto operáciu bolo nutné vykonávať celočíselným delením, čo je však časovo náročná operácia. Z dôvodu dĺžky periódy je nutné starostlivo voliť aj konštanty a a b. Maximálnu periódu je potom možné dosiahnuť, ak:

- b a m sú nesúdeliteľné čísla
- $a - 1$ je násobkom každého prvočiniteľa m
- $a - 1$ je násobkom 4, ak m je násobkom 4

Pre výber počiatočnej hodnoty x_0 v podstate nie sú definované presné pravidlá, obvykle sa však dodržuje postup, že za x_0 sa použije dostatočne veľké nepárne číslo, ktoré by nemalo byť súdeliteľné s modulom m. Ak $b = 0$, takýto generátor sa nazýva *multiplikatívny*, ak $b \neq 0$, takýto generátor nesie označenie *zmiešaný*.

[3] Viac teoretických informácií je možné získať v [5]

Ako ukážkový príklad nevhodne zvolených hodnôt prevzatý z [18], kde $a = 3, b = 5, m = 31, x_0 = 13$, alebo zjednodušene *LCG(31, 3, 5, 13)*. Výslednú postupnosť pri takejto konfigurácii budú tvoriť samé 13.

Lineárny kongruentný generátor patrí takisto medzi najpoužívanejšie typy generátorov. Svedčí o tom napríklad fakt, že ním (pod označením RANDU) už v 60-tych rokoch firma IBM vybavovala svoje počítače typu System/360. Takisto je implementovaný v množstve prekladačov (napr. jazyka C) vo forme štandardných knižničných funkcií[4].

Nasledujúca tabuľka obsahuje prehľad konštánt najznámejších LCG:

Názov	a	b	m
Numerical Recipes	1 664 525	1 013 904 223	2^{32}
RAND	69 069	1	2^{32}
ANSI C – rand()	1 103 515 245	12 345	2^{31}
ANSI C – drand48()	25 214 903 917	11	2^{48}
Park & Miller	16 807	16 807	$2^{31} - 1$
Program Maple	427419669081	0	$10^{12} - 11$
Program Derive	3141592653	1	2^{32}
Program Mathematica	2^{31}	0	$a^{48} - a^8 + 1$

Tabuľka 2.1: Príklady konštánt z rôznych aplikácií

Aj napriek tomu, že sa jedná o jeden z najrozšírenejších generátorov, nemožno nespomenúť ani jeho negatívne vlastnosti. Jedným z problémov, ako už bolo spomenuté je jeho perióda, ktorej dĺžka závisí na voľbe príslušných konštánt. Výraznejším problémom lineárneho kongruentného generátoru je závislosť po sebe nasledujúcich generovaných hodnôt. To je možné pozorovať, ak tento generátor použijeme na generovanie náhodných bodov v n-rozmernom priestore. V trojrozmernom priestore takto generované body nebudú rozdelené rovnomerne, ale zoskupujú sa do rovnobežných nadrovín — rovín v trojrozmernom priestore. V n-rozmernom priestore potom body ležia na najviac $n! * m^{1/n}$ nadrovinách, pričom ostatný priestor, ktorý by mal byť rovnomerne vyplnený, ostáva prázdny [19] [10] [18]. Tento nedostatok bol objavený aj u generátoru RANDU. Ďalším nedostatkom je malá náhodnosť menej významných bitov.

Pre svoje nedostatky nie je vhodný pre náročné aplikácie, ako šifrovanie či pre metódu Monte Carlo v simuláciách. Napriek tomu je však predovšetkým pre svoju jednoduchosť a nenáročnosť na výpočtové zdroje vhodný napríklad pre oblasť počítačových hier, kde je takisto často nutné využívať náhodné hodnoty, alebo vo vstavaných systémoch, kde je typicky iba veľmi obmedzené množstvo pamäti a výpočtového výkonu, či iné aplikácie.

2.3.2 LFSR — Lineárny spätnoväzbový posuvný register

Lineárny spätnoväzbový register je ďalším príkladom jedného z najpoužívanejších generátorov. Jeho jednoduchá štruktúra umožnuje veľmi efektívnu implementáciu v hardvéri (ale aj programovo, ale nie je to typické). Navyše je veľmi dobre teoreticky spracovaný, avšak matematická teória je veľmi komplexná.

Základom tohto generátoru je posuvný register so spätnoväzbovou funkciou. Na vstup spätnoväzbovej funkcie je privedených 2 alebo viac bitov a výstup je potom vedený na vstup celého registru alebo pri alternatívnej implementácii priamo na nasledujúce bity tých, ktoré

[4]Príklady zdrojových kódov pre implementáciu tohto typu generátoru je možné nájsť v [6] [22] [23]

slúžia ako vstup funkcie. Spätnoväzbovú funkciu predstavuje operácia exclusive-OR. Výber príslušných bitov sa realizuje na základe tzv. *primitívneho polynómu*.

Formálne [15] možno jeho činnosť popísať rekurentným vzťahom:

$$x_n = a_1 x_{n-1} + \cdots + a_k x_{n-k} \ mod\ 2$$

kde $k > 1$ je stupeň rekurentnosti, $a_k = 1$, a $\forall j : a_j \in \{0, 1\}$. Činnosť je periodická a dĺžka periódy je $2^k - 1$ práve vtedy, keď jeho charakteristický polynóm tvaru:

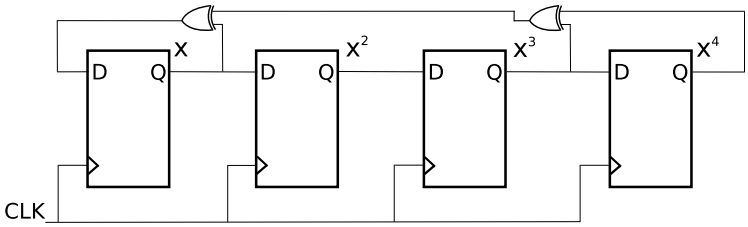
$$P(z) = -\sum_{i=0}^{k} a_i z^{k-i}$$

kde $a_0 = -1$, je *primitívny polynóm* nad Galoisovým telesom s dvoma prvkami, GF(2) (takisto $\mathbf{F_2}$). V anglickej literatúre sa príslušné koeficienty, ktoré tvoria primitívny polynóm a určujú výber príslušných bitov z posuvného registru nazývajú „taps“. Nájdenie takéhoto polynómu je náročné a jednoduchšie je určiť polynóm, ktorý dané vlastnosti nemá. Platí, že primitívny polynóm má vždy párny počet daných členov (číslo 1, ktoré je vždy súčasťou polynómu, sa neuvažuje). Pre exponenty platí, že sú to čísla, ktoré sú navzájom nesúdeliteľné. Teda, polynóm v tvare $x^{12} + x^9 + x^6 + x^3 + 1$ nie je primitívny, pretože všetky exponenty sú deliteľné 3. Použitie takéhoto polynómu takisto nezaručuje, že počet stavov, ktorými by mal generátor pre polynóm tohto stupňa prechádzať, nemusí byť $2^{12} - 1$. Nájdenie jedného primitívneho polynómu vedie k jednoduchému odvodeniu jedného ďalšieho. Ak máme primitívny polynóm so 4 nenulovými koeficientmi zapísaný všeobecne (v zjednodušenej forme a v tomto prípade bez hodnoty 1, ktoré je vždy súčasťou polynómu) v tvare $[n,\ A,\ B,\ C]$, potom ďalší primitívny polynóm je možné odvodiť nasledovne: $[n,\ n - C,\ n - B,\ n - A]$. Je možné samozrejme doplniť do takéhoto zápisu aj príslušnú hodnotu 1, v tom prípade by bola posledná hodnota v skrátenom zápise 0 ($x^0 = 1$). Ako príklad odvodenia možno uviesť polynóm $x^{32} + x^3 + x^2 + x + 1$, ktorý skrátene zapíšeme ako [32, 3, 2, 1]. Z tohto zápisu jednoducho odvodíme [32, 31, 30, 29]. Zvyčajne sa však polynómy pre daný stupeň vyhľadávajú v tabuľkách A.

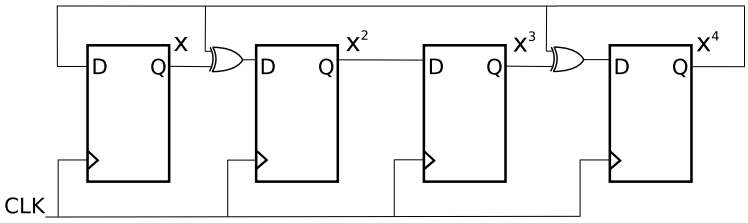
Charakteristický polynóm, pomocou ktorého je možné zostaviť samotný register je možné reprezentovať aj v binárnej podobe. Pri tejto forme zápisu však je nutné dbať na to, ktorý polynóm reprezentujeme. V predchádzajúcom odseku je popísaný spôsob skráteného zápisu, pri ktorom nezáleží, či príslušné členy polynómu zapíšeme so vzrastajúcou alebo klesajúcou hodnotou exponentu, pretože v oboch prípadoch bude hneď zrejmé, o ktorý polynóm sa jedná. Uvažujme 3-bitový LFSR register. Pre tento existuje primitívny polynóm $1 + x^2 + x^3$, z ktorého na základe vyššie popísaného postupu môžeme odvodiť ďalší — $1 + x + x^3$. V skrátenej podobe môžeme prvý polynóm zapísať ako [3,2], či [2,3]. Odvodenie zápisu v binárnej podobe je jednoduché, ale je potrebné vziať do úvahy spôsob zápisu v pôvodnej forme. Teda, pre prvý polynóm môžeme písať 1011 ($\mathbf{1} + \mathbf{0}x + \mathbf{1}x^2 + \mathbf{1}x^3$) a pre druhý 1101 ($\mathbf{1} + \mathbf{1}x + \mathbf{0}x^2 + \mathbf{1}x^3$).

Uveďme si príklad primitívneho polynómu a jemu zodpovedajúcich variant registru LFSR. Na obrázku 1 a 2 sú zobrazené príklady, ako možno realizovať register pre polynóm x^4+x^3+x+1. Na obrázku 2.1 je zobrazená tzv. *Fibonacciho* schéma, v literatúre označovaná aj ako *externá*, či angl. *many-to-one*. Výstupy vybraných klopných obvodov sú zapojené do hradla XOR (v tomto prípade dvojvstupový prvok), všeobecne vstup najnižšieho je funkciou výstupu príslušných klopných obvodov.

Obrázok 2.2 obsahuje alternatívnu *Galoisovu* schému, v literatúre označovanú aj ako *interná*, alebo anglický názov *one-to-many*. V tomto prípade je vstup vybraných klopných



Obrázok 2.1: Fibonacciho schéma



Obrázok 2.2: Galoisova schéma

obvodov funkciou výstupu predchádzajúceho a posledného klopného obvodu. Tento spôsob býva častejšie používaný, keďže obsahuje kratšiu kombinačnú cestu v porovnaní s externou schémou, čo v konečnom dôsledku môže ovplyvniť aj rýchlosť s akou sa náhodná sekvencia generuje.

V tabuľke 2.2 je zobrazený prehľad stavov pre LFSR register, ktorý bol vytvorený podľa primitívneho polynómu $1 + x^2 + x^3$ pre oba možné varianty — internú schému a externú schému. Stupeň polynómu je v tomto prípade 3, preto register bude prechádzať $2^3 - 1$ rôznymi stavmi, táto hodnota zároveň reprezentuje dĺžku periódy. V tabuľke je zobrazený aj výstup z registru v podobe postupnosti bitov v prípade realizácie sériového výstupu. Obsah registru sa v tomto prípade posúva smerom doprava, počiatočná hodnota je 111, na každom nasledujúcom riadku je obsah registru po prechode do ďalšieho stavu.

Externá schéma				Interná schéma			
Q_0	Q_1	Q_2	výstup	Q_0	Q_1	Q_2	výstup
1	1	1	-	1	1	1	-
0	1	1	1	1	1	0	1
0	0	1	1	0	1	1	0
1	0	0	1	1	0	0	1
0	1	0	0	0	1	0	0
1	0	1	0	0	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	1	1	1

Tabuľka 2.2: Príklad prechodu medzi stavmi LFSR registru

Generátory typu LFSR môžu byť jednak použité na generovanie náhodných sekvencií, či už pseudonáhodných čísel, ale takisto aj pseudonáhodného šumu. Ďalej sa široko vyžívajú v overovaní kontrolných súčtov, kompresii dát, tiež veľmi výrazne v oblasti vstavaných

testov v hardvéri (tzv. BIST — *Built-in Self Test*), pri konštrukcii čítačov, v jednoduchej kryptografii, kódovaní signálu (*scrambling, descrambling*), bezdrôtovej komunikácii, spracovaní číslicového signálu... Nepoužíva sa však v náročných aplikáciách v oblasti bezpečnosti, podobne ako ostatné jednoduché generátory. Často však pre svoje výhodné vlastnosti slúži ako stavebný blok pre konštrukciu zložitejších generátorov. Jeho výhodou je vysoká rýchlosť, pretože pracuje na jednoduchom princípe, takisto v prípade hardvérovej implementácie nepredstavuje rozšírenie šírky registru problém, jednoducho sa využije viac voľných prostriedkov či plochy na čipe bez negatívneho dopadu na rýchlosť jeho činnosti.

Ako bolo spomenuté na začiatku, LFSR register pre danú dĺžku n prechádza medzi $2^n - 1$ rôznymi stavmi. Stav, kedy celý register obsahuje samé nuly je stav, z ktorého nie je možné sa dostať do nijakého ďalšieho. Do tohto stavu je možné LFSR register uviesť iba manuálnym prednastavením, pri svojej činnosti tento stav nemôže dosiahnuť. V niektorých prípadoch môže byť užitočné generovať postupnosť s dĺžkou 2^n, kde n je dĺžka registru. Takto sa teda dosiahne stav, ktorý by pôvodne dosiahnuteľný nikdy nebol. Toto je možné docieliť pridaním ďalšej logiky, či čítača, ktorý by umožnil prechod do a z tohto stavu.

Podobne ako je možné použiť hradlá XOR pre realizáciu spätnej väzby, je takisto možné použiť hradlá XNOR. Pri využití tejto alternatívy bude zakázaným stavom prípad, kedy register bude obsahovať samé 1.

2.3.3 Mersenne Twister

Pri popise generátorov náhodných čísel nemožno vynechať v súčasnosti jeden z najlepších generátorov — Mersenne Twister [20] [24]. Bol navrhnutý v roku 1997 japonskými výskumníkmi Makoto Matsumoto a Takuji Nishimura [8]. Pomenovanie tohto generátoru vychádza z toho, že dĺžka periódy je určená Mersennovým prvočíslom.

Mersenne Twister generuje postupnosť hodnôt s periódou $2^{19937} - 1$. Je založený na jednom zo zdokonalených variantov spätnoväzbových posuvných registrov. Okrem dlhej periódy sa vyznačuje veľmi vysokým stupňom rovnomerného rozloženia v n-rozmernom priestore, čo znamená, že netrpí podobným problémom ako lineárny kongruentný generátor — závislosť po sebe nasledujúcich generovaných hodnôt je zanedbateľná. V neposlednom rade má generovaná postupnosť výborné štatistické vlastnosti, tento generátor úspešne obstál aj v testoch známej sady Diehard a je rýchlejší (nevyužíva operácie násobenia a delenia) aj v porovnaní s ostatnými generátormi, ktoré nemajú priaznivé štatistické vlastnosti.

Existuje aj novšia verzia tohto typu generátoru s názvom *SIMD-oriented Fast Mersenne Twister*, ktorá je približne dvakrát rýchlejšia ako pôvodný generátor, produkované hodnoty sú ešte lepšie rovnomerne rozložené a umožňuje meniť dĺžku periódy v rozsahu $2^{607} - 1$ až $2^{132049} - 1$.

Jeho vlastnosti ho predurčujú na použitie v náročných simuláciách, metóde Monte Carlo... Podobne, keďže sa jedná o deterministický algoritmus, sa obvykle nepoužíva v kryptografii a oblasti bezpečnosti.

Tento generátor je najčastejšie implementovaný softvérovo, je však možné ho implementovať aj v hardvéri, napr. v FGPA, avšak dostupnej literatúry alebo praktických príkladov pre takúto realizáciu je výrazný nedostatok v porovnaní s prvou oblasťou. Oproti iným algoritmom, napr. LFSR, sa však dá predpokladať, že v prípade hardvérovej implementácie sa spotrebuje viac zdrojov — obsadenosť plochy na čipe a pod. Môže sa však jednať o zaujímavú alternatívu voči programovej realizácii, ktorá by iste našla využitie v určitých aplikáciách.

2.4 Metódy transformácie rozložení

Doteraz boli popísané spôsoby generovania pseudonáhodných čísel v rovnomernom rozložení $R(0,1)$, resp. $R(0,M)$. Avšak pre veľké množstvo aplikácií potrebujeme mať k dispozícii náhodné veličiny aj s iným než rovnomerným rozdelením.

Pri generovaní pseudonáhodných čísel často daný generátor generuje hodnoty v intervale $(0,M)$. Vo veľkom množstve prípadov však potrebujeme získavať hodnoty v intervale $(0,1)$. Náhodné čísla z tohto intervalu dostaneme jednoduchým výpočtom:

$$y_i = \frac{x_i}{M}$$

kde x_i je pôvodná generovaná hodnota z intervalu $(0,M)$ v $i-$tom kroku výpočtu pseudonáhodného čísla a y_i je nová normovaná hodnota.

Pri generovaní hodnôt v požadovanom rozložení postačí mať generátor, ktorý produkuje čísla s rovnomerným rozložením a tieto sa následne vhodnými metódami transformujú[5] na požadovaný typ rozdelenia. Tieto metódy využívame podľa toho, o aké cieľové rozloženie sa usilujeme a je nutné vziať do úvahy aj presnosť a efektívnosť procesu samotného výpočtu. Preto pri výbere konkrétnej metódy je potrebné voliť určitý kompromis medzi jej efektívnosťou, pamäťovými nárokmi a výsledkami, ktoré dosiahneme zvolenou metódou. Najčastejšie používanými metódami sú:

- metóda inverznej transformácie
- kompozičná metóda
- vylučovacia metóda

Metóda inverznej transformácie

Pri použití tejto metódy využívame inverznú podobu distribučnej funkcie cieľového rozloženia. Našou snahou je vypočítať hodnotu náhodnej veličiny s hustotou pravdepodobnosti $f(x)$. Táto funkcia musí spĺňať dve základné vlastnosti [6]:

$$f(x) > 0, \quad \text{pre všetky } x$$

a

$$\int_{-\infty}^{\infty} f(x)\,\mathrm{d}x = 1$$

Distribučná funkcia má tvar:

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t \quad \text{jej inverzná podoba je} \quad F^{-1}(s)$$

a platí

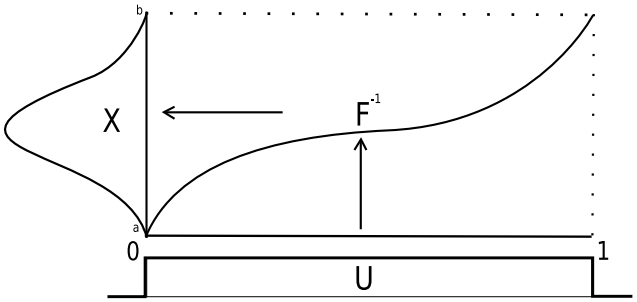
$$y = F(x) \Leftrightarrow x = F^{-1}(y)$$

Distribučná funkcia $F(x)$ musí byť rastúca, čo je nutným predpokladom pre existenciu jej inverznej podoby. Tento spôsob pre generovanie náhodných čísel je možné použiť

[5] Existujú však aj metódy, ktorými je možné generovať veličiny priamo v danom rozložení.

vtedy, keď je možné inverznú podobu distribučnej funkcie $F^{-1}(Y)$ cieľového rozloženia jednoducho a rýchlo vypočítať. Často však distribučná funkcia požadovaného rozloženia nemá svoju inverznú podobu, kedy ju nemožno vyjadriť elementárnymi funkciami. Vtedy je možné pristúpiť k aproximácii distribučnej funkcie inou vhodnou funkciou, ktorej inverzná podoba je už známa. Tento spôsob výpočtu však patrí medzi približné metódy. Keď vieme určiť inverznú podobu distribučnej funkcie, môžeme presne generovať pseudonáhodné čísla v danom rozložení.

Na obrázku môžeme vidieť princíp transformácie pri použití tejto metódy.



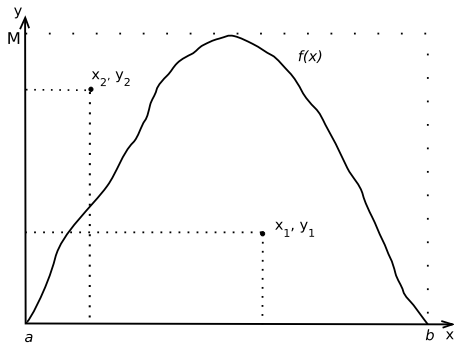
Obrázok 2.3: Princíp metódy inverznej transformácie

Postup pri transformácii môžeme zhrnúť do 3 bodov:

- určíme inverznú podobu distribučnej funkcie F^{-1}
- vygenerujeme náhodné číslo v rovnomernom rozložení $x = R(0, 1)$
- a vypočítame novú hodnotu $y = F^{-1}(x)$

Vylučovacia metóda

Ďalší spôsob generovania náhodných čísel predstavuje vylučovacia metóda [11] [1] [6], v anglickej literatúre označovaná ako *rejection method* alebo *hit-and-miss method*. V princípe sa postupuje tak, že určitú funkciu $f(x)$ ohraničíme na intervale $x \in (a, b)$ a zároveň určíme hornú hranicu funkčných hodnôt M, pre ktorú platí $M \geq f(x)$ pre ľubovoľné x na danom intervale. Na obrázku 2.4 je znázornený princíp tejto metódy.



Obrázok 2.4: Princíp vylučovacej metódy

Pri tomto spôsobe generovania náhodných čísel v podstate generujeme dve čísla, ktoré slúžia ako súradnice bodu v rovine ohraničenej daným obdĺžnikom a hľadáme taký bod, ktorý leží pod krivkou funkcie hustoty cieľového rozloženia. Postup pri využití tejto transformačnej metódy je nasledujúci:

- vygenerujeme náhodné číslo v rovnomernom rozložení $x = R(a, b)$
- vygenerujeme ďalšie náhodné číslo v rovnomernom rozložení $y = R(0, M)$
- ak platí $y \leq f(x)$, potom x prijmeme za hodnotu náhodnej veličiny danej funkcie hustoty
- v opačnom prípade sa vraciame k prvému kroku a danú hodnotu x neprijímame

Na obrázku 2.4 je možné vidieť dva body, pre ktoré boli postupne vygenerované súradnice x_1, y_1 a x_2, y_2. Pre prvý bola splnená daná podmienka, takže hodnota x_1 je prijatá, druhý bod sa však nachádzal nad krivkou funkcie, čo znamená, že nespĺňa podmienku v bode 3, takže hodnota x_2 musí byť zamietnutá. Hodnota y v podstate plní iba úlohu kontrolnej premennej a pri generovaní sa nevracia.

Túto metódu nie je možné použiť pre rozloženia s neobmedzeným rozsahom a efektívnosť jej použitia úzko súvisí s pomerom plochy ohraničenej osou x, funkciou hustoty $f(x)$ a plochou obdĺžnika $M(b-a)$. Pri malom pomere plochy pod funkciou f vzhľadom na plochu celého obdĺžnika výrazne klesá efektívnosť tejto metódy, pretože mnoho generovaných bodov musí byť následne odmietnutých. Takisto väčšina funkcií hustoty určitých rozložení nemá zľava alebo sprava, prípadne z oboch strán ohraničený definičný obor, takže s ohraničením funkcie vykonaným pred samotnou transformáciou dochádza k istému zanedbaniu presnosti pri výpočtoch. Preto je potrebné aj v tomto prípade prijať kompromis medzi voľbou dĺžky intervalu (a, b) a veľkosťou plochy pod krivkou funkcie vzhľadom na celkovú ohraničenú plochu, ktorá je touto voľbou ovplyvnená.

Pravdepodobnosť P generovania hodnoty pod krivkou funkcie $f(x)$ je možné vyjadriť vzťahom

$$P = \frac{\int_a^b f(x)\,\mathrm{d}x}{M(b-a)} = \frac{1}{M(b-a)}$$

Pre niektoré funkcie to bude úplne dostačujúce, pri iné môže nastať situácia, kedy zamietnutých bude väčšina hodnôt. V tom prípade je výhodnejšie uvažovať nad použitím inej transformačnej metódy.

Kompozičná metóda

Nie všetky funkcie je možné použiť pre priamy analytický výpočet, ale na druhej strane sú k dispozícii prostriedky, ktorými je možné dosiahnuť lepšie výsledky než použitím jednoduchej vylučovacej metódy. Kompozičná metóda umožňuje rozložiť zadanú funkciu hustoty na niekoľko jednoduchších funkcií, ktorými sa generovanie zjednoduší.

2.5 Testovanie vlastností generátorov

Cieľom testovania generátorov náhodných čísel je overovanie štatistických vlastností generovanej postupnosti. Môže sa jednať o potvrdenie nejakej konkrétnej vlastnosti, či overenie, že postupnosť generovaných čísel možno považovať za náhodný výber zo súboru

náhodných čísel s určitým pravdepodobnostným rozložením. Existuje však nekonečne mnoho vlastností, ktoré je možné testovať. Pre každý generátor je možné navrhnúť test, ktorý skončí neúspešne. Avšak pri stúpajúcom počte úspešne zdolaných testov rastie aj spokojnosť s vlastnosťami daného generátoru a je možné sa aj na základe porovnania výsledkov takýchto testov rozhodovať o použití určitého generátoru pre nejakú aplikáciu.

V dnešnej dobe existuje niekoľko sád štatistických testov, ktoré patria medzi najpoužívanejšie pri overovaní vlastností generátorov pseudonáhodných čísel (napr. Diehard, Dieharder, Crush, sada NIST...). Overovať vlastnosti je možné aj teoreticky, v tejto práci sa na overenie použil χ^2 test dobrej zhody. Tento test sa používa na overenie, že generovaná postupnosť zodpovedá danému rozloženiu [11] [1] [16].

Postup je možné popísať v nasledovných krokoch:

- Vygenerujeme súbor hodnôt v danom rozložení, ktorý budeme kontrolovať
- Vytvoríme súbor hodnôt so správnymi parametrami a tvarom
- Porovnáme oba súbory hodnôt, a to výpočtom hodnoty testu χ^2
- Vyhodnotíme výsledok testu na základe zvolenej hladiny významnosti

Pri generovaní sa vytvorí veľký súbor n hodnôt (napr. 10 000) a rozdelíme ho do m rovnakých, navzájom disjunktných tried. Počty v jednotlivých triedach označíme N_j. Obvykle sa n volí tak, aby početnosť v každej triede bola $N_j > 5$ (nie vždy sa to však dá dosiahnuť).

Kontrolný súbor môžeme získať napr. využitím distribučnej funkcie daného rozloženia $F(x)$. Početnosti n_j, ktoré zodpovedajú triedam delenia kontrolovaného súboru hodnôt, určíme ako súčin počtu generovaných hodnôt n a rozdielu hodnôt distribučnej funkcie $F(x)$ v krajných bodoch j-tej triedy oboru hodnôt náhodnej veličiny, takže:

$$n_j = n[F(x_1 + j\Delta x) - F(x_1 + (j-1)\Delta x)]$$

$\Delta x = \frac{x_2 - x_1}{m}$ predstavuje veľkosť tried. V prípade, že nepoznáme distribučnú funkciu, môžeme využiť funkciu hustoty daného rozloženia, kde početnosť v j-tej triede získame:

$$n_j = \int_{x_1+j\Delta x}^{x_1+(j-1)\Delta x} f(x)\,\mathrm{d}x$$

Vypočítame hodnotu testu dobrej zhody, čím porovnáme oba súbory hodnôt:

$$\chi^2 = \sum_{j=1}^{m} \frac{(N_j - n_j)^2}{n_j}, \quad \text{pre } n_j > 5$$

kde N_j sú početnosti overovaného súboru hodnôt a n_j početnosti kontrolného súboru. Výpočet realizujeme pre hodnoty $n_j > 5$.
Ďalším dôležitým pojmom je stupeň voľnosti, ktorý je definovaný ako $\nu = m-1$, čiže stupeň voľnosti je o 1 menší ako počet tried rozdelenia. Po vypočítaní výsledku χ^2 a určení stupňov voľnosti (do výpočtov však nezahŕňame tie kategórie, pri ktorých je teoretický počet čísel, ktoré do nej „padnú“ menší ako 5; takto je potrebné znížiť aj počet stupňov voľnosti o počet vyradených kategórií) môžeme v štatistických tabuľkách alebo výpočtom [21] určiť hladinu významnosti, ktorej táto vypočítaná hodnota zodpovedá. Na začiatku máme hypotézu,

ktorú testujeme na určitej hladine významnosti (obvykle 0.01-0.05). Pri určení hladiny významnosti hodnoty χ^2 môžeme teda generátor označiť za vyhovujúci, ak určená hladina významnosti je väčšia ako 0.95.
Alebo je možné postupovať tak, že opäť testujeme hypotézu na zvolenej hladine významnosti (napr. p), ktorá tvrdí, že generovaná postupnosť má dané rozloženie. Určíme kvantil x_p pre túto hladinu významnosti a počet stupňov voľnosti (znova z tabuliek, alebo si pomôžeme dostupnými nástrojmi na jeho výpočet) a porovnáme tieto dve hodnoty. Ak $\chi^2 \leq x_p$ danú hypotézu nezamietame, v opačnom prípade ju zamietame na danej hladine významnosti.

Kapitola 3

Architektúra generátoru náhodných čísel

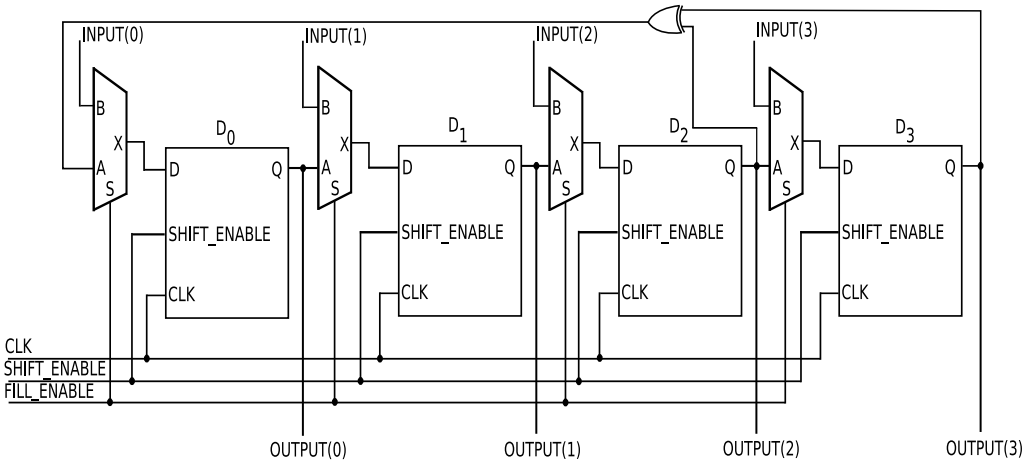
Návrh architektúry samotného generátoru vychádza z informácií uvedených v predošlej kapitole a pre samotnú implementáciu bol zvolený LFSR register. Implementácia prebehla v jazyku VHDL. V tejto časti si priblížime rôzne varianty, ktorými je možné implementáciu realizovať.

3.1 LFSR register s paralelným vstupom a výstupom

Pre implementáciu tohto typu registru bolo potrebné si pripraviť ako základný stavebný prvok klopný obvod typu D. Tento komponent obsahuje štandardne vstupný a výstupný dátový port, vstup hodinového signálu a naviac povoľovací vstup. Tento prvok pracuje s nástupnou hranou hodinového signálu.

Z tohto klopného obvodu je potom možné vytvoriť samotný register. Medzi jeho vstupné porty patria INPUT, ktorého šírka je daná šírkou registru, čiže n je počet bitov registru a zároveň predstavuje stupeň primitívneho polynómu, podľa ktorého bol daný LFSR register vytvorený. Signály FILL_ENABLE a SHIFT_ENABLE sú riadiacimi signálmi. Prvý z nich je použitý pri inicializácii registru a druhý povoľuje posun registru a tým aj zmenu stavu celého LFSR. Pri inicializácii musia byť oba signály aktívne. Činnosť je synchronizovaná hodinovým signálom. Výstupné porty sú OUTPUT a VALID, ktorý určuje platnosť výstupných dát. Po inicializácii, t.j. po zmene úrovne signálu FILL_ENABLE z logickej 1 do logickej 0 sa signál platnosti aktivuje a ostáva aktívny natrvalo. Využitie signálu VALID je potom popísané v časti 4.

Na obrázku 3.1 je zobrazená vnútorná štruktúra 4-bitového LFSR registru, ktorý je vytvorený podľa primitívneho polynómu $1 + x^3 + x^4$. Keďže v tomto prípade sa jedná o paralelný spôsob zapojenia, pre inicializáciu, prípadne ďalšie vloženie hodnoty do registru, je pred každým klopným obvodom zaradený multiplexor, ktorý na základe signálu FILL_ENABLE prepína hodnoty z portu INPUT a výstupu príslušného klopného obvodu. Hodnota signálu VALID je ovládaná procesom vnútri definície architektúry, takže nie je v obrázku zobrazený.



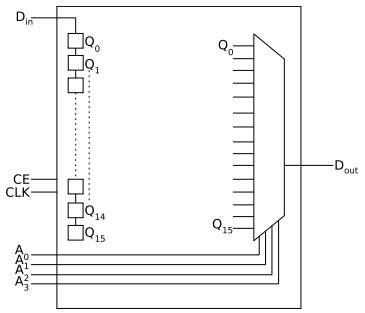
Obrázok 3.1: Štruktúra paralelného LFSR (externé zapojenie)

3.2 LFSR register so sériovým vstupom a výstupom

Variant so sériovým vstupom a výstupom je možné realizovať podobne ako paralelnú verziu. V tomto prípade odpadne nutnosť zaradenia multiplexorov pred každým klopným obvodom — použije sa jediný pred prvým z nich, aby bolo možné sériovo inicializovať obsah registru a zároveň viesť spätnú väzbu z posledného členu do prvého. V rozhraní samotného komponentu teda došlo iba k zmenám signálov INPUT a OUTPUT, ktoré sú v tomto prípade jednobitové. Vnútorná štruktúra je v princípe zhodná s obrázkom 3.1 až na zmeny popísané v tomto odseku.

Omnoho zaujímavejšou alternatívou voči použitiu klopných obvodov typu D môže byť pre realizáciu na čipe Virtex, resp. Spartan, využitie priamo stavebných prvkov dostupných na čipe [27] [28] [9]. Základným konfigurovateľným prvkom na týchto čipoch sú tzv. CLB bloky (*Configurable Logic Blocks*). Počet blokov závisí od konkrétnej verzie čipu, napr. na čipe Spartan-3 umiestnenom na platforme FITkit sa ich nachádza 192.

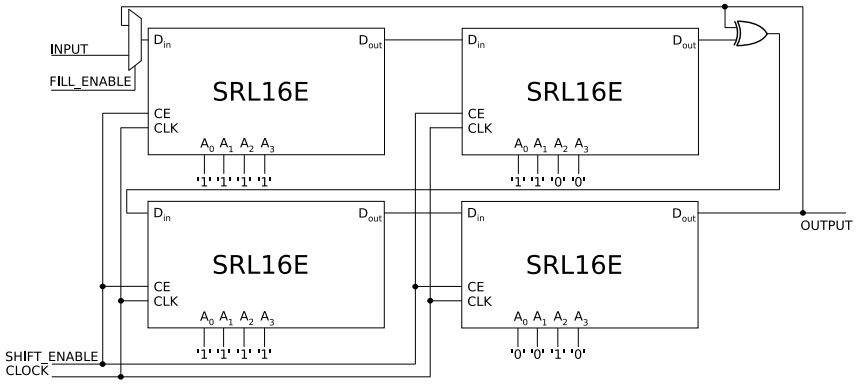
Každý z týchto konfigurovateľných blokov pozostáva zo 4 tzv. *slices*. Slice je tvorený dvoma 4-vstupovými logickými funkčnými generátormi, multiplexormi, dvoma úložnými elementmi a carry logikou. Funkčný generátor potom môže byť naprogramovaný ako 4-vstupová Look-up tabuľka, ako 16-bitová distribuovaná SelectRAM+ pamäť či ako 16-bitový posuvný register s možnosťou zmeny jeho dĺžky v rámci týchto 16 bitov.



Obrázok 3.2: Schéma SRL16E registru

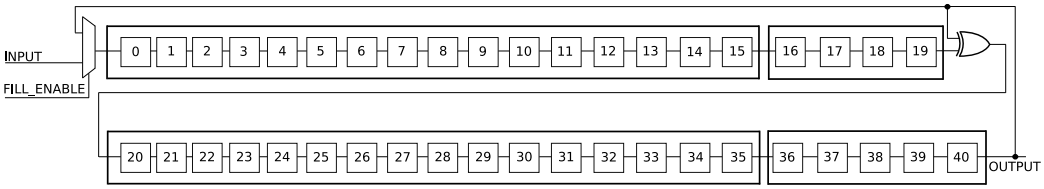
Pre stavbu LFSR registru je teda možné využiť práve tieto posuvné registre (SRL — *Shift Register LUT*). Tento register je 4-bitový a je možné získať hodnotu ľubovoľného bitu na základe zadanej 4-bitovej adresy. Konfiguráciou adresy je možné nastaviť dĺžku registru. Hodnota adresy môže byť nastavená staticky, ale môže byť aj dynamicky zmenená. Posun registru sa uskutočňuje v každom cykle hodinového signálu. K dispozícii sú dve primitíva — SRL16 a SRL16E. Obe obsahujú jednobitový dátový vstup a výstup, vstup hodinového signálu a 4-bitový adresový vstup. SRL16E obsahuje naviac povoľovací vstup hodinového signálu (ClockEnable).

Na obrázku 3.2 je zobrazená zjednodušená schéma SRL16E posuvného registru. Q_0 – Q_{15} je 16 SRAM buniek zapojených ako 16-bitový posuvný register. Príslušný multiplexor slúži na určenie bunky, hodnota ktorej sa bude prenášať na výstup na základe definovanej adresy.



Obrázok 3.3: Príklad zapojenia s využitím SRL16E registrov

Obrázok 3.3 znázorňuje príklad realizácie LFSR registru definovaného polynómom $1 + x^{20} + x^{41}$. Tentokrát je použitá Galoisova schéma. Je dobré si uvedomiť nastavenie adresových bitov na jednotlivých registroch — pre zjednodušenie je na obrázku 3.4 uvedená štandardná schéma s využitím klopných obvodov a vyznačený spôsob, akým je možné tento návrh realizovať pomocou SRL registrov. Z dôvodu udržania prehľadnosti je schéma zjednodušená a sú vynechané niektoré signály, podstatný je spôsob vyznačenia častí celého registru, ktoré je možné zahrnúť do jedného SRL registru.



Obrázok 3.4: Pôvodný register

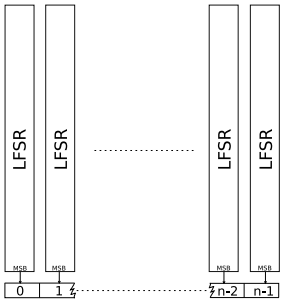
V literatúre [13] [2] [3] je možné nájsť mierne odlišné varianty zapojenia, v tejto práci však naďalej budeme uvažovať doteraz prezentované spôsoby, ktoré sú pomerne priamočiare a jednoduché.

Využitie SRL registrov však so sebou prináša aj nevýhodu v podobe straty prístupu k jednotlivým bitom vytvoreného registru. To je však vyvážené efektívnou implementáciou

a v prípade implementácie sériového variantu LFSR registru sa táto nevýhoda stráca úplne. Preto je dobré dopredu zvažovať požiadavky a možnosti, ktoré sú dostupné v danej situácii.

3.3 Viacnásobný LFSR

V tejto časti si priblížime možnosti zlepšenia vlastností generovanej postupnosti pomocou LFSR registra dosiahnutým pomocou jednoduchého vylepšenia v architektúre. Výhodou je, že nie je potrebné meniť návrhový prístup pri konštrukcii LFSR registra popísaného v predchádzajúcich častiach — základom je stále ten istý register.



Obrázok 3.5: Schéma MLFSR

Inšpirácia pre tento spôsob pochádza z práce Petra Martina [14], ktorý porovnával štatistické vlastnosti rôznych typov generátorov, medzi ktorými bol aj LFSR. Ako ukázal, tak štandardný LFSR register trpí, podobne ako lineárny kongruentný generátor, závislosťou po sebe nasledujúcich generovaných hodnotách. Jednoduchým spôsobom je však možné dosiahnuť výrazné zlepšenie štatistických vlastností. Jeho výsledky ukázali, že v testoch Diehard, ktoré vykonal, patril medzi jeden z najlepších.

Princíp je založený na tom, že sa využije n LFSR registrov dĺžky m, ktoré sa zapoja paralelne, pričom najvyšší bit každého z nich potom slúži pre vytvorenie pseudonáhodného čísla, ktoré má dĺžku práve n bitov. Princíp prehľadne demonštruje obrázok 3.5. Základom je v tomto prípade LFSR register so sériovým vstupom a výstupom, keďže na výstup slúži iba jeden najvyšší bit z každého registru. Takýmto spôsobom sa v každom takte hodinového signálu vytvára jedno pseudonáhodné číslo zložené z hodnôt výstupu jednotlivých LFSR registrov, pričom generované hodnoty majú lepšie štatistické vlastnosti v porovnaní s použitím jediného LFSR registru. Tento generátor sa označuje v literatúre *Multiple LFSR*, skrátene MLFSR.
Pre implementáciu bola zvolená štruktúra MLFSR $n \times n$, tzn. počet bitov generovanej hodnoty je rovné počtu (stupňu) všetkých použitých LFSR registrov. Jednotlivé LFSR registre sú vytvorené podľa jedného polynómu a musia byť inicializované odlišnými hodnotami.

3.4 Generátor VHDL kódu pre LFSR registre

Z praktickej stránky práce bol navrhnutý a implementovaný core generátor pre generovanie VHDL kódu pre LFSR. Generátor je naprogramovaný v jazyku C a umožňuje generovanie všetkých typov LFSR registrov uvedených v predchádzajúcich častiach. Vstupom programu sú jednotlivé parametre, ktoré určujú typ požadovaného LFSR. Vytvorenie kódu pre register je dané uvedením primitívneho polynómu v binárnej podobe. Tento program umožňuje

generovanie LFSR registru ľubovoľnej dĺžky a zvoleného typu daného voľbou príslušných parametrov.
Výsledok činnosti programu je produkovaný na štandardný výstup. Výstup je vo forme VHDL kódu, ktorý obsahuje všetky potrebné komponenty pre zostavenie LFSR registru. Z dôvodu jednoduchosti pri riešení výstupu bola zvolená alternatíva, kedy sú všetky potrebné komponenty umiestnené do jedného súboru. Ďalšou vlastnosťou tohto programu je možnosť generovania obvodu testbench pre zvolený LFSR register. Výstup programu je možné presmerovať do samostatných súborov a následne sa takto dajú vytvorené LFSR registre jednoducho a rýchlo testovať. Pred samotným testovaním je však potrebné vložiť počiatočné hodnoty pre inicializáciu registrov na vymedzené miesto v generovanom kóde.

Výstup programu bol riadne otestovaný simulačnými experimentmi v ModelSime (porovnávali sa požadované hodnoty s hodnotami generovanými ako výstupy simulácie) pre každý typ LFSR registru a jeho varianty. Generovaný kód je syntetizovateľný, avšak je nutné poznamenať, že v kóde sa využívajú priamo inštancie primitív, ktoré sú závislé na použitom čipe, čo v konečnom dôsledku môže mať dopad na prenositeľnosť takéhoto kódu. V kóde sa využíva štrukturálny prístup k prepojeniu základných komponent, preto bol zvolený tento variant inštancovania komponent, ktorý umožňuje jednoduchý návrh. Syntéza kódu bola vykonaná v aplikácii ISE WebPack. Ukážka výstupu — použitých hardwarových zdrojov je dostupná v časti 6.2.

Na záver je dobré si všimnúť, že pri návrhu architektúry sa obišlo použitie resetu, či už asynchrónneho alebo synchrónneho. Použitie asynchrónneho resetu vedie k horšej optimalizácii pri syntéze obvodu a takisto môžu vznikať problémy so správnym časovaním operácií [25]. Program v zdrojovom tvare je umiestnený na priloženom CD médiu spolu so súborom Makefile. Spôsob použitia programu je podrobne popísaný v prílohe B.

Kapitola 4

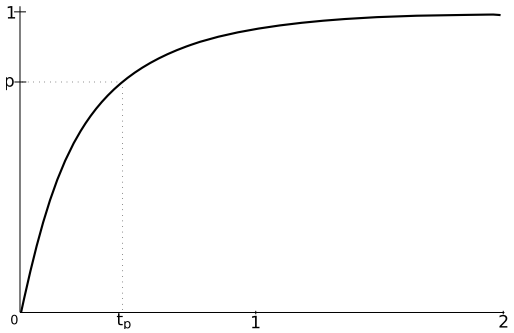
Transformácie rozložení

V rámci tejto práce bola zvolená transformácia do exponenciálneho rozloženia. Použitou metódou je metóda inverznej transformácie, pomocou ktorej bolo možné dosiahnuť jednoduchú implementáciu. Teoretické východiská sú zhruba popísané v časti 2.4, táto kapitola obsahuje podrobnejší popis transformácie do exponenciálneho rozloženia a jej hardvérovú implementáciu.

Pri využití metódy inverznej transformácie [4] vychádzame zo znalosti distribučnej funkcie. Pre distribučnú funkciu exponenciálneho rozloženia platí

$$F(t) = \begin{cases} 0 & t < 0 \\ 1 - e^{-\lambda t} & t \geq 0 \end{cases}$$

Všeobecne sa pri transformácii najprv postupuje tak, že sa vygeneruje číslo p z intervalu $(0, 1)$. Túto hodnotu p nadobúda distribučná funkcia v bode t_p. Tento bod sa nazýva p-kvantil. Takže distribučná funkcia predstavuje prostriedok, pomocou ktorého je možné prejsť medzi hodnotami pravdepodobnosti z intervalu (0,1) a hodnotami, ktoré nadobúda náhodná veličina X v konkrétnom rozložení. Na obrázku 4.1 je znázornený tento princíp.



Obrázok 4.1: Distribučná funkcia exponenciálneho rozloženia

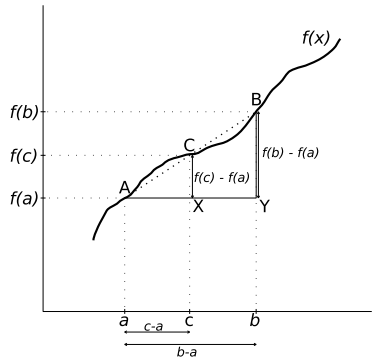
Keďže platí, že $p = F(t_p)$, našou úlohou je vypočítať hodnotu t_p. Postupuje sa klasicky — vyjadríme inverznú podobu funkčného predpisu, čiže:

$$p = 1 - e^{-\lambda t} \quad \Rightarrow \quad t_p = -\frac{1}{\lambda} \cdot \ln(1 - p)$$

Praktická realizácia vyplýva z uvedeného postupu. Pre implementáciu bol zvolený postup, pri ktorom sa definičný obor rozdelí sa určitý počet intervalov a pre hraničné body

intervalov, medzi ktorými je rovnaká vzdialenosť sa určia funkčné hodnoty. Tieto význačné hodnoty sa následne uložia do pamäti a ostatné hodnoty, ktoré „padnú“ do vnútra niektorého z intervalov, sa dopočítajú lineárnou interpoláciou.

Postup tohto výpočtu je všeobecne znázornený na obrázku 4.2, kde máme interval (a, b) poznáme funkčné hodnoty $f(a),\ f(b)$ a potrebujeme vypočítať $f(c)$.



Obrázok 4.2: Lineárna interpolácia

Platí, že $\triangle AYB \sim \triangle AXC$, a teda:

$$\frac{f(b) - f(a)}{b - a} = \frac{f(c) - f(a)}{c - a}$$

Jednoduchou úpravou dostaneme hľadanú hodnotu $f(c)$:

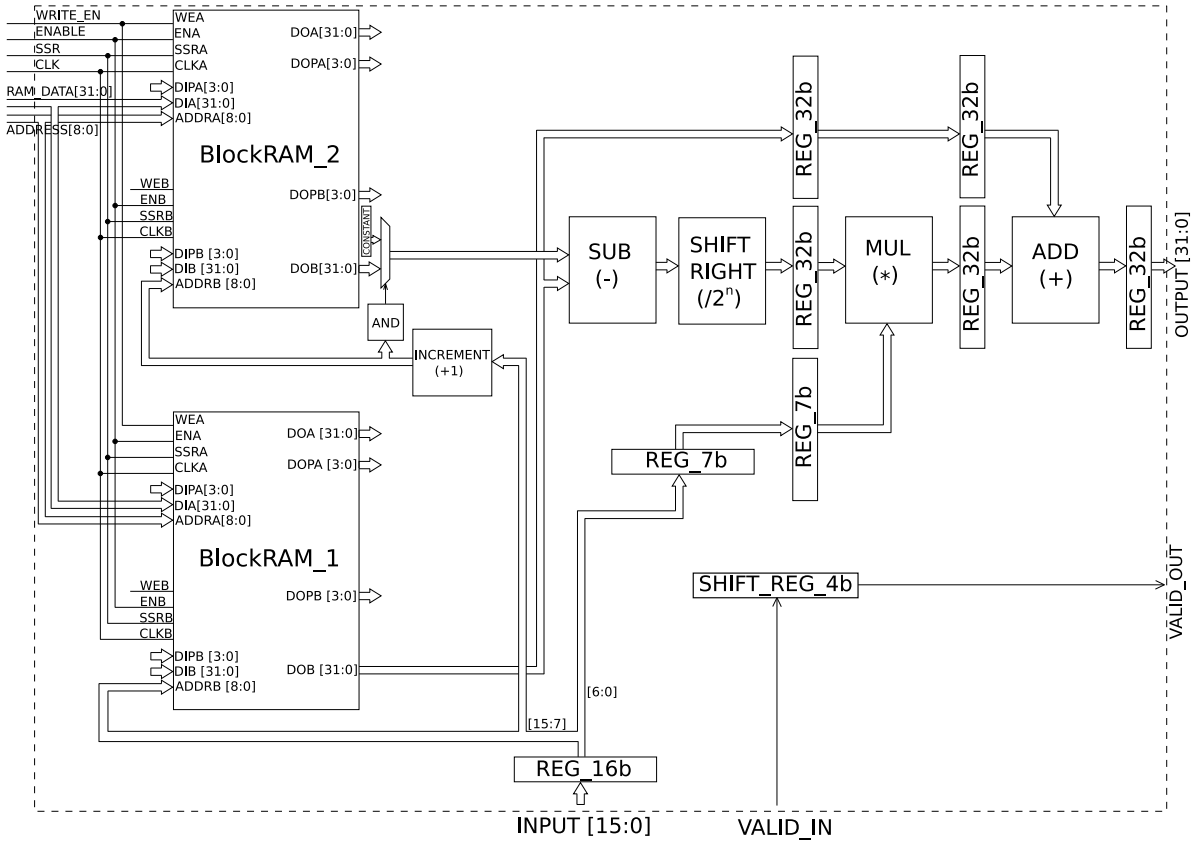
$$f(c) = \frac{f(b) - f(a)}{b - a}(c - a) + f(a) \tag{4.1}$$

Keďže pri tomto postupe je potrebné uchovávať istý počet hodnôt v pamäti, pri implementácii boli využité tzv. BlockRAM pamäte [26], ktoré sú dostupné na čipe FPGA. Ich výhodou je predovšetkým to, že sa dajú vhodne konfigurovať pre danú aplikáciu. Je možné si zvoliť šírku ukladaných dát, ktorá ovplyvňuje množstvo položiek uložiteľných v pamäti, je možné využiť fakt, že dovoľujú konfiguráciu v podobe dvojportových pamätí a takto umožňujú prístup k dátam cez dvojicu nezávislých portov a iné. Počet a umiestnenie tohto druhu pamäte je opäť závislé na použitom čipe a jeho verzii.

Pri konečnej realizácii bola využitá konfigurácia, pri ktorej sa použilo 9 bitov na adresáciu, čím možno adresovať 512 32-bitových hodnôt. Pri spôsobe generovania pseudonáhodných hodnôt v tejto práci bolo nutné sa zaoberať ich reprezentáciou. LFSR generátorom vieme získať $2^n - 1$ celočíselných hodnôt pre daný stupeň n. Preto je potrebné pri transformácii tieto hodnoty normalizovať, následne transformovať a opäť previesť do želaného intervalu celočíselných hodnôt. Použitý generátor je 16-bitový MLFSR. Z generovanej hodnoty sa použije horných 9 bitov na adresovanie hodnôt z BlockRAM pamäti, dolných 7 bitov sa využije pri spresňovaní výpočtu pri lineárnej interpolácii. Dĺžka kroku v tomto prípade je 128 (hodnôt). Pri prevode a výpočte transformovaných hodnôt je nevyhnutné pracovať s číslami s pohyblivou desatinnou čiarkou. Preto pri určovaní hodnôt, ktoré sa uložia do pamäti, bol vytvorený krátky program v jazyku C, ktorý tento výpočet zrealizuje a následne na štandardný výstup vypíše časť VHDL kódu, ktorým sa vykoná počiatočná inicializácia BlockRAM pamäti. To je ďalšia z jej výhod — pamäť je možné inicializovať už pri jej inštancovaní v zdrojovom kóde pri vytváraní cieľovej aplikácie.

Obor funkčných hodnôt inverznej podoby distribučnej funkcie exponenciálneho rozloženia leží v intervale $(0, \infty)$, preto je nutné stanoviť určitú hornú hranicu pre transformované hodnoty. Pri transformácii je opäť vhodné si zvoliť nejaký celočíselný interval, na ktorý sa vypočítané desatinné hodnoty prevedú. V tomto prípade bol zvolený pre jednoduchosť interval 0-65535, v ktorom bude generátor vypočítané hodnoty produkovať. Pri výpočtoch sa teda získané desatinné hodnoty prepočítajú na celočíselné hodnoty v intervale $(0, 65535)$. Je zrejmé, že pri týchto výpočtoch dochádza k nepresnostiam, ktoré je nutné brať do úvahy.

V súhrne teda generovanie hodnôt spočíva najprv v rozdelení intervalu na 512 častí, hraničné hodnoty týchto intervalov sa normalizujú a transformujú pomocou inverznej podoby distribučnej funkcie exponenciálneho rozloženia a následne sa prepočítajú na hodnoty vo zvolenom celočíselnom intervale. Tieto hodnoty sa uložia do pamäti a pomocou nich sa počítajú ostatné hodnoty. Na adresovanie týchto hodnôt sa použije horných 9 bitov generovaného čísla z MLFSR generátoru a dolných 7 bitov pri výpočte konečnej hodnoty.



Obrázok 4.3: Schéma transformačného bloku

Obrázok 4.3 zobrazuje schému obvodu, ktorý transformuje generované hodnoty na vstupe INPUT[15:0] na hodnoty v exponenciálnom rozložení (v tomto konkrétnom prípade). Na tento vstup sú privedené výstupy generátoru produkujúceho čísla v rovnomernom rozložení. Hodnota sa zo vstupu uloží do prvého registru. Pri nástupnej hrane hodinového signálu sa hornými 9 bitmi z tejto hodnoty zaadresuje pamäť č. 1, pričom sa zároveň táto hodnota inkrementuje o hodnotu 1 a ňou sa zaadresuje pamäť č. 2. Týmto spôsobom je dosiahnuté získanie krajných hodnôt príslušného intervalu, do ktorého pripadá generovaná hodnota tak, aby bolo možné realizovať výpočet uvedený v predchádzajúcich odsekoch.

Pri generovaní pseudonáhodných hodnôt môže nastať situácia, kedy obsah horných 9 bitov z celkových 16 budú tvoriť samé jednotky. V tom prípade by mohol nastať problém pri inkrementácii tejto hodnoty, pretože by po tejto operácii pretiekla a z pamäti č. 2 by sa získala nesprávna hodnota, ktorá by negatívne ovplyvnila hodnotu konečného výsledku. Preto je za výstup tejto pamäti zaradený multiplexor ovládaný výsledkom operácie logického súčinu jednotlivých bitov adresy. Ďalší vstup multiplexoru tvorí dopredu vypočítaná konštanta, ktorá obsahuje maximálnu celočíselnú funkčnú hodnotu inverznej podoby distribučnej funkcie (pri rozdelení intervalu sa počítajú funkčné hodnoty pre jednotlivé hraničné body, začína sa 0, spolu s maximálnou hodnotou je týchto hodnôt 513 a týmto spôsobom je možné získať aj poslednú — maximálnu hodnotu). Získané hodnoty z oboch pamätí sa privedú na vstup bloku, ktorý vypočíta ich rozdiel (čitateľ zlomku v rovnici 4.1). Výsledok rozdielu je privedený na vstup bloku, ktorý realizuje operáciu podielu. Táto operácia je s výhodou vykonaná ako operácia bitového posunu, keďže dĺžka intervalu je zvolená ako mocnina čísla 2. Výsledok po tomto kroku sa uloží do registru, z ktorého sa v nasledujúcom takte hodinového signálu privedie na vstup bloku realizujúceho násobenie (viď rovnica 4.1) Ďalšou vstupnou hodnotou je dolných 7 bitov z výstupu generátoru. Pretože hodnoty, s ktorými sa počíta, sú 16-bitové (ale v tomto prípade uložené na 32 bitoch) a navyše po operácii podielu, nie je nutné meniť dátovú šírku výstupu z tohto komponentu. Výsledok násobenia sa opäť uloží do nasledujúceho registru.

V ďalšom takte hodinového signálu sa hodnoty uložené v registroch privedú na vstup sčítačky, ktorá reprezentuje poslednú fázu výpočtu. Pred výstupom z celého transformačného bloku je zaradený posledný register.

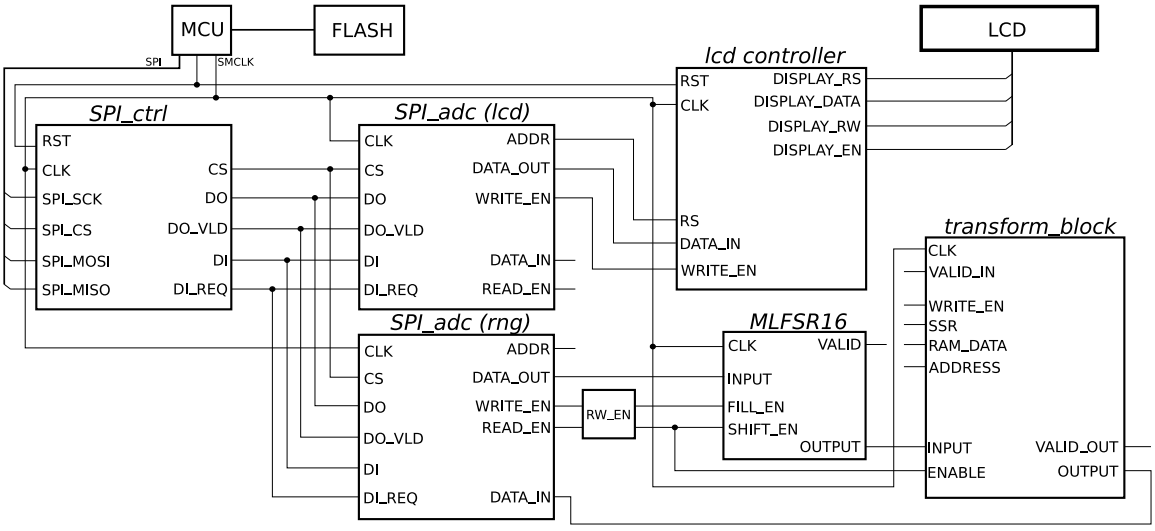
Použitie všetkých registrov na uchovanie čiastkových hodnôt celkového výsledku pred rôznymi fázami výpočtu umožnilo zreťazené spracovanie výsledku bez nutnosti použitia nejakého riadiaceho prvku, ktorý by ovládal jednotlivé komponenty. Takto je možné v každom takte hodinového signálu získavať na výstup vypočítané hodnoty. Keďže získanie prvých správnych výsledkov trvá určitú dobu (je potrebné zahrnúť dobu na inicializáciu generátoru a dobu prechodu medzi jednotlivými stupňami spracovania), je privedený na vstup transformačného bloku VALID_IN signál z generátoru indikujúci platnosť generovaných hodnôt. Tento signál je potom privedený na vstup posuvného registru, kde počet bitov sa zhoduje s počtom registrov medzi jednotlivými stupňami spracovania konečného výsledku. V okamihu, kedy sa platná hodnota tohto signálu dostane na výstup transformačného bloku, bude zároveň na výstupe prvá platná transformovaná hodnota.

Na výstup celého tohto komponentu sú navyše vyvedené vstupy BlockRAM pamätí. Vstup CLK slúži ako vstup hodinového signálu a sú ním ovládané pamäte a registre v jednotlivých stupňoch spracovania výsledku. Obe pamäte sú nakonfigurované ako dvojportové, jeden port je možné využiť na prípadný zápis nových hodnôt, druhý port slúži len na získavanie hodnôt vnútri komponentu. Niektoré výstupy pamätí nie sú využité — napr. výstup paritných bitov, či dátový výstup z jedného z dvojice portov. Port ENABLE slúži ako vstup signálu povoľujúceho činnosť pamätí a jednotlivých registrov. Pri jeho neaktívnej úrovni nie je možné pracovať s pamäťou a takisto jednotlivé registre uchovávajú svoj aktuálny stav. Na zápis do pamätí a privedenie externej adresy slúžia porty WRITE_ENABLE, ADDRESS[8:0] a RAM_DATA[31:0]. Posledným signálom je SSR (*Synchronous Set Reset*), ktorý uvedie záchytné registre výstupov pamätí do stavu uvedeného parametrom SRVAL pri inštancovaní.

Kapitola 5

Implementácia do prípravku FITkit

V rámci praktického overenia funkčnosti bola vykonaná implementácia navrhnutého generátoru a bloku realizujúceho transformáciu do prípravku FITkit. Pri implementácii sa využili už hotové dostupné komponenty, pomocou ktorých je možné vytvoriť funkčný celok. Prepojovací systém [17] tvoria komponenty, ktoré umožňujú vytvoriť vnútri FPGA čipu zbernicu dovoľujúcu komunikáciu medzi mikrokontrolérom a zariadením pripojeným k tejto zbernici. Z hotových komponentov ide predovšetkým o radič SPI, ktorého úlohou je realizovať prevod SPI protokolu na internú sériovú zbernicu. Ďalšou je SPI dekodér, ktorý dekóduje adresu cieľového zariadenia a poskytuje paralelné rozhranie pre pripojenie nejakého zariadenia a prevádza ho na internú sériovú komunikáciu, čím umožňuje obojsmernú komunikáciu. Počet takýchto SPI dekodérov najčastejšie zodpovedá počtu použitých jednotlivých zariadení, ale zároveň aj počtu jednotlivých adresových priestorov vyhradených pre tieto zariadenia. Schéma aplikácie je na obr. 5.1.



Obrázok 5.1: Bloková schéma aplikácie

V demonštračnej aplikácii boli použité 2 SPI dekodéry, jeden pre radič LCD displeja a druhý pre samotný generátor. Keďže komunikácia s MCU prebieha cez SPI, je potrebné

zvoliť adresový priestor pre použité SPI dekodéry. Okrem iného je potrebná voľba ďalších generických parametrov pre tieto komponenty. SPI dekodér pre radič LCD displeja má nasledujúce parametre:

- DATA_WIDTH = 8 bitov
- ADDR_WIDTH = 8 bitov
- ADDR_OUT_WIDTH = 1 bit
- BASE_ADDR = 16#00#

Ďalší SPI dekodér slúži na obsluhu samotného generátoru náhodných čísel a má tieto parametre:

- DATA_WIDTH = 16 bitov
- ADDR_WIDTH = 8 bitov
- ADDR_OUT_WIDTH = 1 bit
- BASE_ADDR = 16#90#

Na vnútornú SPI zbernicu môže byť pripojený ľubovoľný počet SPI dekodérov, preto pre správne adresovanie cieľového zariadenia je potrebné vhodne voliť šírku adresovej časti zbernice (ADDR_WIDTH) a bázovú adresu. Parameter ADDR_OUT_WIDTH musí byť väčší ako nula a udáva šírku adresnej zbernice, ktorú má pripojené zariadenie k dispozícii. Hodnoty tohto a zároveň ďalšieho parametru DATA_WIDTH môžu byť zvolené podľa potreby pripojeného zariadenia. Keďže generátor použitý v tejto aplikácii používa 16 bitovú šírku dát, bola zvolená príslušná hodnota. Pri adresovaní sa využíva hodnota bázovej adresy, pričom do úvahy sa pri porovnávaní berie najvyšších ADDR_WIDTH - ADDR_OUT_WIDTH bitov. Uvedené parametre umožňujú jednoznačnú adresáciu oboch pripojených zariadení.

V tejto aplikácii je použité ovládanie pripojeného generátoru prostredníctvom MCU. MCU prečíta aktuálnu generovanú hodnotu na výstupe bloku počítajúceho transformácie a túto hodnotu prevedie na reťazec, ktorý sa následne zobrazí na displeji. Čítanie a zobrazovanie hodnôt je realizované v cykle a nie je nutný zásah užívateľa. Počiatočná inicializácia registru prebehne po nahraní konfigurácie FPGA prostredníctvom MCU. Keďže zvolený generátor je 16-bitový MLFSR, je tento spôsob rýchlejší ako manuálna inicializácia napr. cez terminál. Návod na použitie tejto aplikácie je uvedený v prílohe D.

Kapitola 6

Dosiahnuté výsledky

6.1 Test dobrej zhody

Pri vykonaní tohto testu vychádzame z teoretických východísk uvedených v časti 2.5. Pri simulácii činnosti transformačného obvodu bolo zaznamenaných 10000 hodnôt, ktoré boli rozdelené do 20 tried. Podobne sa určil súbor kontrolných dát využitím výpočtu pomocou distribučnej funkcie exponenciálneho rozloženia. Všetky hodnoty sú zaznamenané v tabuľke 6.1.

j	n_j	N_j	i	n_i	N_j
1	8236	8173	11	1	0
2	1436	1493	12	2	0
3	265	273	13	2	0
4	41	50	14	1	0
5	0	9	15	1	0
6	2	2	16	0	0
7	1	0	17	2	0
8	2	0	18	2	0
9	0	0	19	3	0
10	1	0	20	2	0

Tabuľka 6.1: Prehľad početností pre oba kontrolované súbory hodnôt

Hodnoty z kontrolného súboru boli zaokrúhlené na celé čísla. Na začiatku stanovíme hypotézu H_0: *generovaná postupnosť hodnôt má exponenciálne rozloženie.* Na základe uvedených hodnôt vypočítame χ^2.

$$\sum_{j=1}^{m} \frac{(N_j - n_j)^2}{n_j} = 4,962$$

Horná hranica m predstavuje všeobecne počet tried, ale z výpočtu boli vynechané všetky členy, kde $n_j < 5$. Počet stupňov voľnosti je teda 3 ($\nu = 4 - 1$). Pri voľbe hladiny významnosti 0,05 dostávame zodpovedajúci kvantil $\chi^2_{0,95}(3) = 7,815$. Na základe porovnania týchto dvoch výsledkov nezamietame hypotézu H_0.

6.2 Spotreba zdrojov na čipe

V tabuľke 6.2 sú zobrazené informácie o nutných hardvérových zdrojoch pre implementáciu generátoru a bloku, ktorý realizuje transformáciu rozloženia. Použitý typ generátoru je 16-bitový MLFSR vytvorený použitím SRL16E registrov. Údaje sú získané z aplikácie ISE WebPack a ako cieľový čip bol v tomto prípade zvolený Spartan-3E.

	Transformačný obvod	MLFSR generátor
Slices	91	70
Slice Flip Flops	139	1
4-vstupové LUT	149	129
max. frekvencia	105,445 MHz	187,793 MHz

Tabuľka 6.2: Hardvérové zdroje nutné pre realizáciu

V nasledujúcich tabuľkách je uvedený prehľad nutných zdrojov na realizáciu MLFSR registru pri využití D-klopných obvodov a SRL16E registrov. Všetky MLFSR generátory majú Galoisovu podobu a príslušné dvojice sú vytvorené podľa rovnakého polynómu.

	MLFSR16_D	MLFSR32_D	MLFSR64_D
Slices	60	120	311
Slice Flip Flops	113	197	405
4-vstupové LUT	97	227	526
max. frekvencia	275,179 MHz	275.179MHz	275.179 MHz

Tabuľka 6.3: Požiadavky na zdroje pre varianty MLFSR generátorov s využitím D-klopných obvodov

	MLFSR16_SRL	MLFSR32_SRL	MLFSR64_SRL
Slices	69	154	371
Slice Flip Flops	1	1	1
4-vstupové LUT	129	289	705
max. frekvencia	187,793 MHz	187,793 MHz	187,793 MHz

Tabuľka 6.4: Požiadavky na zdroje pre varianty MLFSR generátorov s využitím SRL16E registrov

Kapitola 7

Záver

V rámci tejto práce bola naštudovaná problematika generovania náhodných čísel. Boli priblížené vlastnosti najpoužívanejších generátorov, popísané metódy transformácie rozložení a overovania vlastností generátorov pseudonáhodných čísel. Získané informácie sú zhrnuté v kapitole Teoretický rozbor.

Na základe naštudovaných informácií bol navrhnutý obvod pre generovanie náhodných čísel založený na generátore typu LFSR. Keďže je možné realizovať viacero variantov tohto typu generátoru, bola vykonaná implementácia core generátoru v jazyku C. Vytvorený core generátor je konfigurovateľný pomocou sady parametrov a umožňuje vytvoriť generátor pseudonáhodných čísel presne podľa požiadaviek užívateľa. Je možné zadať primitívny polynóm v binárnej reprezentácii, podľa ktorého sa vytvorí cieľový LFSR register. Ďalšou možnosťou je voľba medzi externou a internou schémou generátoru, paralelným či sériovým rozhraním. Takisto je možné vygenerovať násobný LFSR register, ktorý produkuje postupnosť náhodných čísel s lepšími štatistickými vlastnosťami v porovnaní s použitím samostatného LFSR registru. V neposlednom rade umožňuje generovanie testbench obvodu pre LFSR generátor určený ostatnými zadanými parametrami. Výstupom programu je VHDL kód popisujúci štruktúru daného obvodu. Praktická implementácia je popísaná v kapitole Architektúra generátoru náhodných čísel.

Následne bol navrhnutý obvod realizujúci transformáciu do exponenciálneho rozloženia, ktorý má zreťazenú architektúru. Činnosť tohto obvodu spolu so samotným generátorom bola overená simuláciou v ModelSime. Zároveň bola vykonaná implementácia tohto obvodu pre platformu FITkit vo forme jednoduchej demonštračnej aplikácie. Popis tohto obvodu a aplikácie do prípravku FITkit obsahujú kapitoly Transformácie rozložení a Implementácia pre platformu FITkit. Z výsledkov získaných syntézou bol vypočítaný test dobrej zhody pre vygenerovanú postupnosť, pričom na základe výsledku tohto testu sa nezamietla hypotéza s tvrdením, že generovaná postupnosť má exponenciálne rozloženie.

Spolu so syntézou obvodu boli zaznamenané i požiadavky na zdroje nutné pre hardvérovú implementáciu. Maximálna frekvencia, ktorú je možné dosiahnuť obvodom realizujúcim transformáciu je približne 105 MHz a pre tento obvod sa spotrebuje 91 slices na cieľovom čipe. Pre porovnanie sa vykonala syntéza násobných LFSR registrov pre rôzne dátové šírky. Všetky získané údaje sú prehľadne zhrnuté v kapitole Dosiahnuté výsledky.

Možným nadviazaním na výsledky dosiahnuté v tejto práci môže byť návrh zložitejších generátorov vychádzajúcich z princípov podobných činnosti LFSR registru, či implementácia komplexnejších generátorov, ktoré nie sú bežne navrhované pre FPGA (napr. Mersenne Twister), pričom zriedka sa variant tohto spôsobu realizácie objaví, avšak aj podľa zistených informácií sa najčastejšie jedná o komerčné riešenia, ktoré nie sú voľne dostupné.

Pritom by hardvérová implementácia takéhoto generátoru mohla byť veľmi zaujímavou voľbou pre niektoré praktické použitia. Takisto je možné zamerať sa na odlišné metódy transformácie rozložení, pre ktoré existuje množstvo pokročilých algoritmov, či zamerať sa na postupy, ktorými je možné generovať postupnosť náhodných čísel priamo v požadovanom rozložení a prispôsobiť tieto metódy pre implementáciu do čipu FPGA.

Vypracovanie tejto práce znamenalo výrazné prehĺbenie poznatkov z oblasti generovania pseudonáhodných čísel, oboznámenia sa s využitím FPGA pri obvodovom návrhu týchto generátorov a zdokonalenia znalostí jazyka VHDL.

Literatúra

[1] Rábová Zdena a kol. *Modelování a simulace*. Fakulta elektrotechniky a informatiky, VUT v Brně, 1992.

[2] Andy Miller, Michael Gulotta. XAPP 211 (v1.2) PN Generators Using the SRL Macro. `http://www.xilinx.com/bvdocs/appnotes/xapp211.pdf`, 2004. (platnosť odkazu — máj 2007).

[3] Andy Miller, Stephen Lim. XAPP 220 (v1.1) LFSRs as Functional Blocks in Wireless Applications. `http://www.xilinx.com/bvdocs/appnotes/xapp220.pdf`, 2001. (platnosť odkazu — máj 2007).

[4] Irena Růžičková, Břetislav Fajmon. *Matematika 3*. Fakulta elektrotechniky a komunikačních technologií, VUT v Brně, 2003.

[5] Jiří Dřímal, David Trunec, Antonín Brablec. *Úvod do metody Monte Carlo*. Přírodovědecká fakulta, katedra fyzikální elektroniky, Masarykova univerzita Brno, 2006.

[6] Kai Nordlund. Monte Carlo simulations, lecture 4. Generating random numbers. `http://beam.acclab.helsinki.fi/~knordlun/mc/mc4nc.pdf`, 2006. (platnosť odkazu — máj 2007).

[7] K.H. Tsoi, K.H. Leung and P.H.W. Leong. Compact FPGA-based True and Pseudo Random Number Generators. `http://csdl.computer.org/dl/proceedings/fccm/2003/1979/00/19790051.pdf`, 2003. (platnosť odkazu — máj 2007).

[8] Makoto Matsumoto, Takuji Nishimura. Papers on random number generation. `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/earticles.html`. (platnosť odkazu — máj 2007).

[9] Mike Gulotta. Xilinx XCell Journal 35, HDL Coding for Pseudo-random noise Generator. `ftp://ftp.xilinx.com/pub/documentation/xcell/xcell35.pdf`, 2000. (platnosť odkazu — máj 2007).

[10] Miroslav Virius. Test generátoru pseudonáhodných čísel RANDU. `http://tjn.fjfi.cvut.cz/~virius/prednes/mc-prikl/Randu.html`. (platnosť odkazu — máj 2007).

[11] Petr Peringer. *Modelování a simulace, Studijní opora*. Fakulta informačních technologií, VUT v Brně, 2006.

[12] Peter Alfke. XAPP052, Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators. http://www.xilinx.com/bvdocs/appnotes/xapp052.pdf, 1996. (platnosť odkazu — máj 2007).

[13] Peter Alfke, Maria George. XAPP 210 (v1.2) Linear Feedback Shift Registers in Virtex Devices. http://www.xilinx.com/bvdocs/appnotes/xapp210.pdf, 2001. (platnosť odkazu — máj 2007).

[14] Peter Martin. An Analysis of Random Number Generators for a Hardware Implementation of Genetic Programming using FPGAs and Handel-C, Technical Report CSM-358. http://www.celoxica.com/techlib/files/CEL-W0307171J2F-23.pdf, 2002. (platnosť odkazu — máj 2007).

[15] Pierre L'Ecuyer, Francois Panneton. A New Class of Linear Feedback Shift Register Generators. http://www.informs-cs.org/wsc00papers/091.PDF. (platnosť odkazu — máj 2007).

[16] Radoslav Vargic. Generovanie náhodných čísel. www.ktl.elf.stuba.sk/~vargic/mas/mas2004/nahodne_cisla/PREDNASKA.pdf. (platnosť odkazu — máj 2007).

[17] WWW stránky. FITkit. http://www.fit.vutbr.cz/kit/. (platnosť odkazu — máj 2007).

[18] Vlastimil Klíma. Generátory náhodných čísel - II: Jak losuje počítač, Chip 4/1998. http://crypto-world.info/klima/1998/chip-1998-04-50-50.jpg http://crypto-world.info/klima/1998/chip-1998-04-52-52.jpg, 1998. (platnosť odkazu — máj 2007).

[19] Wikipedia. Linear congruential generator — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.phptitle, 2007. (platnosť odkazu — máj 2007).

[20] Wikipedia. Mersenne twister — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Mersenne_twister&oldid=126474969, 2007. (platnosť odkazu — máj 2007).

[21] WWW stránky. Chi-Square Calculator. http://www.fourmilab.ch/rpkp/experiments/analysis/chiCalc.html. (platnosť odkazu — máj 2007).

[22] WWW stránky. Dan's coding practice area, Standard Library Random Numbers. http://cer.freeshell.org/renma/LibraryRandomNumber/. (platnosť odkazu — máj 2007).

[23] WWW stránky. Fast Random Number Generator on the Intel Pentium 4 Processor. http://www.intel.com/cd/ids/developer/asmo-na/eng/microprocessors/ia32/pentium4/optimization/43797.htm?page=1. (platnosť odkazu — máj 2007).

[24] WWW stránky. Mersenne Twister: A Study on Random Number Generators. (platnosť odkazu — máj 2007). http://student.vub.ac.be/~nkaraogl/mt/mt.html#_ftnref2.

[25] Xilinx, Inc. Xilinx Synthesis and Simulation Design Guide 8.2i. http://toolbox.xilinx.com/docsan/xilinx82/books/docs/sim/sim.pdf, 2006. (platnosť odkazu — máj 2007).

[26] Xilinx, Inc., San Jose, CA . XAPP 463 (v2.0) Using Block RAM in Spartan-3 Generation FPGAs. http://www.xilinx.com/bvdocs/appnotes/xapp463.pdf, 2005. (platnosť odkazu — máj 2007).

[27] Xilinx, Inc., San Jose, CA . Spartan-3 FPGA Family: Complete Data Sheet, DS099-3 (v2.1) Product Specification. http://direct.xilinx.com/bvdocs/publications/ds099.pdf, 2006. (platnosť odkazu — máj 2007).

[28] Xilinx, Inc., San Jose, CA. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Datasheet, DS083 (v4.6) Product Specification. http://direct.xilinx.com/bvdocs/publications/ds083.pdf, 2007. (platnosť odkazu — máj 2007).

[29] Z. Gutterman, B. Pinkas, T. Reinman. Analysis of the Linux Random Number Generator. http://eprint.iacr.org/2006/086.pdf, 2006. (platnosť odkazu — máj 2007).

Zoznam príloh

A Prehľad primitívnych polynómov

B Použitie core generátoru

C Ukážky priebehu činnosti generátoru

D Použitie aplikácie pre FITkit

Príloha A

Prehľad primitívnych polynómov

n	výstup z	n	výstup z	n	výstup z	n	výstup z
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,95	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,12
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66, 65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,87	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

Tabuľka A.1: Prehľad členov primitívnych polynómov, ktorými je možné dosiahnuť maximálnu periódu $2^n - 1$ pre stupeň polynómu n. Ako spätnoväzbovú funkciu možno použiť XOR alebo XNOR. Prevzaté z [12].

Príloha B

Použitie core generátoru

Použitie programu pre generovanie VHDL kódu pre jednotlivé varianty LFSR generátoru je dané nasledujúcimi parametrami:

```
lfsr -p XX...XX -F|-G [-S|-SRL [-M]] [-T]
```

Povinné parametre sú `-p`, za ktorým nasleduje binárna reprezentácia polynómu, podľa ktorého sa vytvorí požadovaný LFSR register. Spôsob prevodu do binárnej podoby je uvedený v časti 2.3.2. Je nutné poznamenať, že takto sa vytvorí LFSR register, v ktorom budú vstupy do spätnoväzbovej funkcie dané jednotkovými bitmi v tejto binárnej reprezentácii, pričom prvý a posledný bit je vždy nutne 1. Najviac významný bit vytvoreného registru je najpravejší bit.

Ďalší povinný parameter je jeden z dvojice `-F` alebo `-G`, ktorý určuje typ schémy generovaného registru. Parameter `-F` reprezentuje Fibonacciho (externú) schému a parameter `-G` Galoisovu (internú schému).

Implicitne sa generuje LFSR register s paralelným vstupom a výstupom, parametrami `-S` a `-SRL` je možné generovať sériový variant. Použitie parametru `-SRL` vedie k použitiu SRL16E registrov pri vytváraní vnútornej štruktúry LFSR registru.

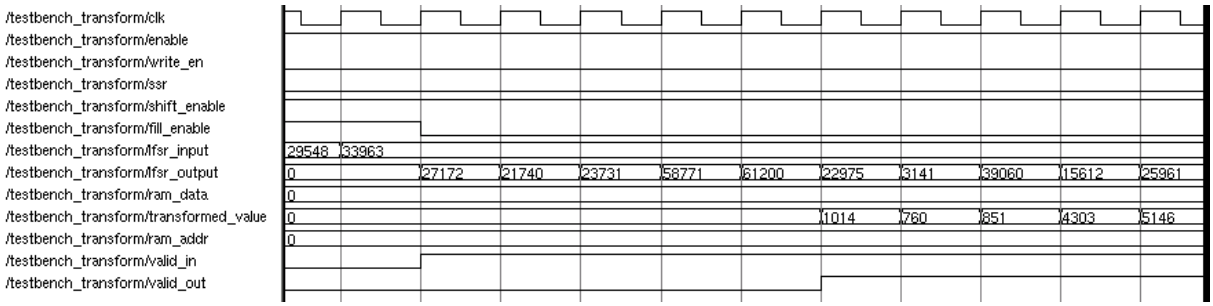
Voliteľný parameter `-M` je možné použiť iba v prípade použitia jedného z parametrov `-S` alebo `-SRL`. Tento parameter dovoľuje generovať násobný LFSR register (MLFSR). Rozhranie takéhoto generátoru je paralelné, ale interne je zostavený z n jednoduchých sériových LFSR registrov stupňa n (dané polynomiálnou reprezentáciou). Štruktúra jednotlivých registrov so sériovým vstupom a výstupom je určená voľbou parametrov `-S` a `-SRL` a všetky reprezentujú jeden a ten istý primitívny polynóm. Vstup a výstup každého z nich je pripojený na jeden bit paralelného vstupu a výstupu celého MLFSR. Inicializácia sa realizuje násobným privedeným počiatočnej hodnoty na vstup, pričom jednotlivé bity počiatočnej hodnoty sa postupne vkladajú do jednotlivých registrov.

Parametrom `-T` je možné vygenerovať testbench pre register daný ostatnými parametrami. Do vytvoreného kódu je potrebné na zvýraznené miesto doplniť počiatočné hodnoty, ktorým sa register inicializuje. Následne je možné činnosť vytvoreného obvodu simulovať napr. v programe ModelSim. Uvedené parametre je možné použiť v ľubovoľnom poradí.

Príloha C

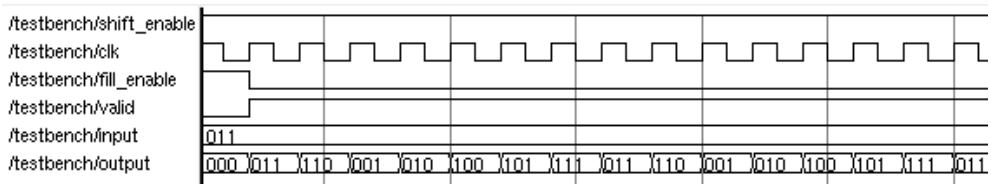
Ukážky priebehu činnosti

Na obrázku C.1 je zobrazený úsek priebehu činnosti navrhnutého obvodu generujúceho čísla v exponenciálnom rozložení. Na tomto obrázku sú viditeľné časové okamžiky, kedy skončila inicializácia generátoru MLFSR a dobou, ktorá uplynula od vygenerovania prvého platného transformovaného čísla.



Obrázok C.1: Priebeh činnosti obvodu realizujúceho transformáciu

Obrázok C.2 zobrazuje ukážku prechodu medzi stavmi jedného 3-bitového LFSR registru s internou schémou. Tento register bol vytvorený podľa primitívneho polynómu $1 + x^2 + x^3$ (binárne 1011) a má podľa očakávaní periódu $2^3 - 1 = 7$. Oba obrázky boli získané simuláciou v programe ModelSim.



Obrázok C.2: Priebeh činnosti jedného LFSR registru

Príloha D

Použitie aplikácie pre FITkit

V prvom rade je nutné mať už nainštalované ovládače a ostatné aplikácie potrebné pre úspešnú prevádzku prípravku FITkit. Postup inštalácie je dostupný na internetových stránkach venovaným FITkitu [17]. Potom stačí z týchto stránok stačí stiahnuť obsah najnovšieho SVN, kde sú nadefinované všetky potrebné základné komponenty a ostatné zdrojové súbory pre prácu s mikrokontrolérom. V hlavnom adresári SVN je potrebné vytvoriť nový adresár, do ktorého sa rozbalí obsah komprimovaného súboru rng.tar.gz umiestneného na CD prílohe tejto práce.

Preklad aplikácie

Je potrebné sa presunúť do adresára `sw` a zadať príkaz `make`. Dôjde k automatickému prekladu programu, keďže v adresári je pripravený Makefile. Nahranie preloženého programu do MCU sa vykoná príkazom `make load`.

Ďalším krokom je syntéza obvodu a vytvorenie konfiguračného súboru pre FPGA. Je potrebné sa vrátiť o adresár o úroveň vyššie a prejsť do adresára `top`. Podobne ako v predchádzajúcom prípade, využije sa prítomný súbor Makefile a príkazom `make` sa automaticky vykoná syntéza obvodu.

Nahranie konfigurácie do FPGA

Postačí spustiť obľúbený správne nakonfigurovaný terminál a vytvoriť spojenie. Po vytvorení spojenia zadať príkaz `prog FPGA` a vybrať v závislosti na konkrétnom termináli možnosť zaslania súboru. Všobecne sa tento krok vykoná vybratím voľby „File transfer“, potom sa potvrdí výber príslušného protokolu, v tomto prípade sa zvolí „XMODEM“ a vyberie sa konkrétny súbor, ktorý sa následne odošle. Je potrebné zvoliť súbor `output.bin`, ktorý sa vytvoril po úspešnej syntéze v adresári `top` našej aplikácie. Po úspešnom prenose MCU automaticky vloží počiatočné hodnoty do generátoru a na displeji sa pravidelných časových okamžikoch budú objavovať generované hodnoty v exponenciálnom rozložení.