Elektrotehnički fakultet Univerziteta u Beogradu



Implementacija parametrizovanog generatora pseudoslučajne sekvence

–Diplomski rad–

|  |  |
| --- | --- |
| Kandidat: | Mentor: |
| Nevena Bošković 2015/0623 | doc. dr Zoran Čiča |

Beograd, Septembar 2019.

Sadržaj

[Sadržaj 2](#_Toc20470739)

[1. Uvod 3](#_Toc20470740)

[2. Pomerački registar sa linearnom povratnom spregom 4](#_Toc20470741)

[2.1. Polinom povratne sprege 4](#_Toc20470742)

[2.2. Fibonacci LFSR 4](#_Toc20470743)

[2.3. Galois LFSR 5](#_Toc20470744)

[3. Implementacija LFSR 6](#_Toc20470745)

[3.1. Entitet LFSR 6](#_Toc20470746)

[3.1.1. Interfejsi entiteta 6](#_Toc20470747)

[3.1.2. Generic deo entiteta 7](#_Toc20470748)

[3.2. Arhitektura 7](#_Toc20470749)

[3.2.1. Arhitektura za opis Fibonacci metode 8](#_Toc20470750)

[3.3. Prikaz rada entiteta 8](#_Toc20470751)

[3.3.1. Primer za primenu Fibonacci metode 8](#_Toc20470752)

[3.3.2. Primer za primenu Galois metode 8](#_Toc20470753)

[4. Simulacija 10](#_Toc20470754)

[4.1. Primer 1. 10](#_Toc20470755)

[4.2. Primer 2. 11](#_Toc20470756)

[4.3. Primer 3. 11](#_Toc20470757)

[4.4. Primer 4. 12](#_Toc20470758)

[5. Performanse LFSR 13](#_Toc20470759)

[6. Zaključak 15](#_Toc20470760)

[Literatura 16](#_Toc20470761)

[B. Prilog 17](#_Toc20470762)

[B.1. VHDL kod za lfsr za dužinu *n*=4, primitivni polinom *pol*=0011 i broj pomeraja *sh\_size*=2 17](#_Toc20470763)

[B.2. Testbench za dužinu registra *n*=4, inicijalnu vrednost *seed*=1100 18](#_Toc20470764)

1. Uvod

RAND kompanija je 1955. godine generisala i objavila knjigu od milion slučajnih brojeva, što predstavlja značajan napredak na polju statistike i kriptografije.[1] Utvrđeno je da determinističke sekvence, potpuno određene početnim vrednostima zadovoljavaju kriterijume slučajnosti i počela je upotreba pseudoslučajnih sekvenci. Ovakva sekvenca nastala je generisanjem slučajnih brojeva u realnom vremenu korišćenjem računarskih algoritama.

Kao prateća pojava, proizašla je potreba za generatorima pseudoslučajnih sekvenci. Preteča ovakvih generatora je generator baziran na [metodi sa srednjim kvadratom](https://sr.wikipedia.org/w/index.php?title=Middle-square_method&action=edit&redlink=1) čiji je inovator [Džon](https://sr.wikipedia.org/wiki/%D0%8F%D0%BE%D0%BD_%D1%84%D0%BE%D0%BD_%D0%9D%D0%BE%D1%98%D0%BC%D0%B0%D0%BD) fon Nojman.[2] Najoptimalniji i najčešći metod realizacije pseudoslučajnog binarnog generatora je onaj koji se zaniva na njegovoj implementaciji pomoću pomeračkog registra sa linearnom povratnom spregom (Linear feedback shift register- LFSR).Upravo ovaj pomerački registar biće predmet analize i ispitivanja kroz ovu tezu.

U okviru teme postoje četiri zasebne celine. Prva celina bavi se načinom funkcionisanja pomeračkog registra sa linearnom povratnom spregom, karakterističnim polinomom i načinima implementacije i predstavlja teorijsku osnovu. Druga celina pruža uvid u strukturu VHDL (VHSIC Hardware Description Language) koda koji opisuje LFSR, kao i primere koji prikazuju njegovu funkcionalnost. Treće poglavlje predstavlja testiranje dizajna i poređenje sa teorijskim proračunom. Poslednji deo odnosi se na performanse sistema, upotrebljene resurse i zaključke izvedene analizom.

1. Pomerački registar sa linearnom povratnom spregom

Pomerački registar sadrži skup međusobno povezanih flip-flopova sa zajedničkim taktom. Izlaz prethodnog flip-flopa povezan je sa ulazom za podatak narednog, tako da se memorisani podatak pod dejstvom zajedničkog taktnog signala pomera bit po bit od jednog do drugog flip-flopa. Linearna funkcija primenjena je na određenim bitima iz registra. Postojanje povratne sprege ukazuje da se rezultat funkcije vraća na ulaz nekog od flip-flopova. Najčešće korišćene linearne funkcije su ekskluzivno ili (XOR) i ekskluzivno nili (XNOR).[3]

Pomerački registar dužine n, može imati -1 različitih početnih stanja. U slučaju XOR linearne funkcije nedozvoljeni slučaj su sve nule, dok su u slučaju XNOR funkcije to sve jedinice. Za ova stanja registar bi ostao zaključan tj. nepromenjen, što nema nikakvu primenu, pa se ona smatraju zabranjenim.

Zbog konačnog broja stanja u nekom trenutku dolazi do ponavljanja ciklusa tj. sekvenca je periodična. Tačna sekvenca i period zavise od početnog stanja i od aktivnih povratnih petlji. Maksimalna dužina sekvence dobija se ukoliko ona prođe kroz sva stanja, odnosno ukoliko je njena perioda jednaka broju stanja.

(2.1.)

* 1. Polinom povratne sprege

Odabir n+1 mogućih članova povratne petlje je opisan karakterističnim polinomom LFSR. Za ovaj polinom za LFSR sa n flip-flopova se kaže da je polinom stepena n, i on poseduje sledeći oblik:

(2.1.1*)*

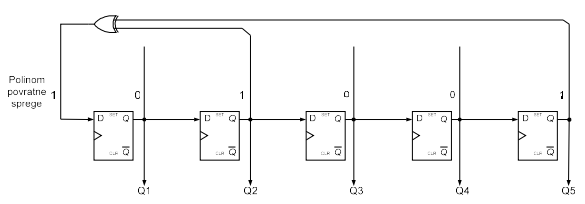
gde su koeficijenti binarni, i mogu biti jedinica ili nula (jedinica znači da je odgovarajuća povratna petlja uspostavljena).

Maksimalna dužina sekvence dobija se ukoliko je karakteristični polinom primitivan.

Postoje dva načina implementacije LFSR u hardveru čija je glavna razlika u načinu realizacije povratne sprege.

* 1. Fibonacci LFSR

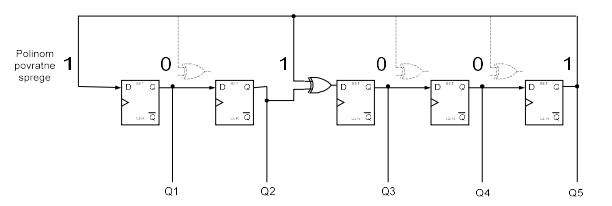
Mehanizam povratne sprege kod Fibonacci-jeve implementacije linearnog pomeračkog registra je realizovan na taj način što se izlazi D flip-flopa sprovode na ulaze XNOR (XOR) kola, a izlaz XNOR (XOR) kola vraća na ulaz jednog od flip-flpova (najčešće prvog/poslednjeg).



Slika 2.2.1. Primer Fibonacci-jeve implementacije polinoma petog stepena

* 1. Galois LFSR

Kod Galois implementacije logičko kolo je umetnuto izmedju izlaza jednog i ulaza njemu susednog flip-flopa. Izlaz poslednjeg flip-flopa u nizu dovodi se na jedan ulaz XNOR (XOR) kola, dok je drugi ulaz izlaz flip-flopa koji prethodi kolu. Rezultat logičke funkcije predstavlja ulazni bit narednog flip-flopa.



Slika 2.3.1. Primer Galois implementacije polinoma petog stepena

Brzina rada je posebno kritična kada u lancu povratne sprege postoji veći broj XNOR(XOR) kola. Kod Fibonacci pristupa, XNOR(XOR) kola su kaskadno povezana, pa je propagaciono kasnjenje signala kroz LFSR veće. Kod Galois izlazi XNOR(XOR) kola se vezuju paraleno, pa je zbog toga propagacija signala manja, tj brzina rada veća.[4]

1. Implementacija LFSR

Realizacija linearnog pomeračkog registra sa povratnom spregom ostvarena je u Xilinx razvojnom okruzenju i VHDL programskom jeziku.

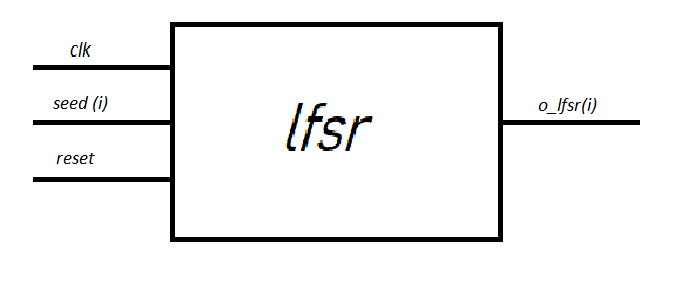
Jedan VHDL kod odgovara opisu dizajna jednog hardverskog bloka (modula) koji izvršava određenu logičku funkciju (ili skup logičkih funkcija). Pri opisu jednog hardverskog modula važne su dve stvari. Prva stvar su interfejsi modula, preko kojih je povezan i omogućena mu je komunikacija sa drugim modulima. Ovaj deo odnosi se na upotrebu modula i njegovu interakciju sa ostalim hardverom u složenim sistemima, tj. bitna je samo funkcija modula, ne i njegova interna struktura.

Druga stvar je interna struktura modula koja definiše princip rada modula, a samim tim i funkciju (ili skup funkcija) koje modul obavlja. Ovaj deo je važan dizajnerima modula i u njemu se definiše način realizacije same funkcije. [5]

* 1. Entitet LFSR

U VHDL kodu u entitetu definiše se ime kola koje se projektuje i imena i osnovne karakteristike njegovih ulaza i izlaza, odnosno portova.

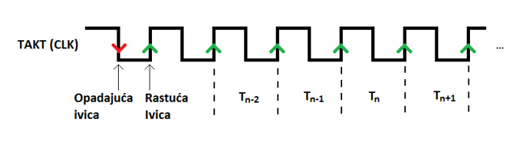
* + 1. Interfejsi entiteta



3.1.1. Slika Entitet LFSR

* + - 1. CLK

Neophodna je vremenska sinhronizacija svih delova modula, koju omogućava taktni signal (clk). Ovaj signal predstavlja povorku pravougaonih impulsa, odnosno vrednosti 1 i 0 koje se međusobno smenjuju sa određenom frekvencijom. Promena stanja može se vršiti na uzlaznu ili silaznu ivicu takta.



Slika 3.1.2. Prikaz signala takta

* + - 1. Reset signal

Razlikujemo dva tipa reset signala:

Asinhroni reset je reset koji nije sinhron sa takt signalom, odnosno kada god se postavi na aktivnu vrednost, flip-flopovi se vraćaju u početno stanje, bez obzira u kom vremenskom trenutku se to desilo.

Sinhroni reset je sinhron sa taktom, tako da se vraćanje flip-flopova u početno stanje ne dešava do prve ivice takt signala nakon što je reset postao aktivan.

* + - 1. Seed signal

Signal seed je niz koji odgovara duzini registra i na aktivnu vrednost signala reseta elementi niza postavljaju se kao inicijalne vrednosti u registru. U okviru ove implementacije korišćena je XNOR logička funkcija, pa su vrednosti svih jedinica u seed signalu zabranjene.

* + - 1. O\_lfsr signal

Izlazni signal o\_lfsr je niz koji predstavlja izlaznu sekvencu iz registra i odgovara dužini registra.

* + 1. Generic deo entiteta

U okviru ovog dela definišu se promenljive. Njihovim uvođenjem omogućavamo parametrizaciju i promenu ovih vrednosti u bilo kom trenutku.

* + - 1. Dužina registra

Promenljivom n određen je broj bita u našem registru.

* + - 1. Broj pomeraja

Vrednost sh\_size predstavlja broj ponovljenih ciklusa, odnosno prikazuje za koliko mesta su pomereni biti u okviru registra.

* + - 1. Primitivni polinom

Parametar pol predstavlja niz bita od kojih jedinice ukazuju na one bite koji učestvuju u XNOR funkciji, pri čemu je bit najveće težine prvi bit sa desne strane. Jednake je dužine kao i registar.

* 1. Arhitektura

U VHDL kodu, u arhitekturi se definiše interna struktura i opisuje dizajn. Povezana je sa definisanim entitetom, odnosno opisuje šta je unutar entiteta. Arhitektura je uvek pridružena jednom entitetu, dok entitet može imati više arhitektura.

* + 1. Arhitektura za opis Fibonacci metode

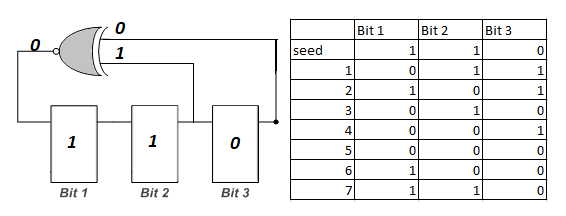
U okviru arhitekture koja opisuje entitet lfsr uveden je proces čiju listu osetljivosti čini taktni i reset signal. Uveli smo promenljivi niz *r\_lfsr* koji odgovara dužini registra. Njegova inicijalna vrednost su sve nule, a na aktivan signal *reset* dodeljuje mu se vrednost *seed* signala. U procesu se nalaze dve while petlje. Unutrašnja petlja ponavlja se n puta, odnosno odgovara dužini niza pol. Uvodimo promenljivu i koja definiše broj iteracija, odnosno položaj u nizu pol. Kako binarna jedinica ukazuje da bit na datoj poziciji ulazi u XNOR funkciju postavljamo if konstrukciju sa ovim uslovom u kojoj postoje dve varijante. Prva ukazuje da je data jedinica prva u nizu *pol* i podatak iz niza r\_lfsr sa pozicije *i* se samo upisuje u promenljivu *r1*. Druga varijanta se koristi ukoliko se u promenljivoj *r1* već nalazi neka vrednost, pa se ona XNOR-uje sa podatkom iz *r\_lfsr* sa pozicije *i*, pa se čuva u promenljivoj *r1*. Inkrementiranjem promenljive *i* omogućava se rad petlje. Nakon toga se vrednosti niza *r\_lfsr* od prve do pretposlednje pozicije postavljaju na mesto druge do n-1. Na mesto prve pozicije niza *r\_lfsr* postavlja se promenljiva *r1*. Spoljašnja petlja zavisi od *sh\_size* i definiše koliko puta se obavlja unutrašnja petlja. Na kraju se vrednosti niza *r\_lfsr* dodeljuju izlaznom signalu *o\_lfsr*.

* 1. Prikaz rada entiteta

Radi verodostojnijeg objašnjenja rada našeg entiteta dat je ručni proračun i prikaz rezultata za date parametre, kako bi prikazali funkciju koju registar izvršava za oba tipa implementacije.

* + 1. Primer za primenu Fibonacci metode

Za registar duzine n=3, primitivni polinom je , tj. 2. i 3. bit se dovode na ulaz XNOR funkcije, pa se rezultat vodi na mesto prvog bita u registru. Ukoliko za inicijalne vrednosti registra postavimo seed=110 dobijamo sledeći proračun:

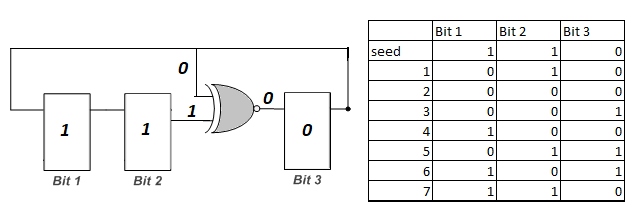


Slika 3.2.1.1. Linearni registar sa povratnom spregom dužine n=3 bita i izlazne sekvence pri primeni primitivnog polinoma..

Odakle zaključujemo da je period ponavljanja T=2^n-1=7, čime potvrđujemo da je sekvenca maksimalne dužine za odgovarajući primitivni polinom.

* + 1. Primer za primenu Galois metode

Za registar duzine n=3, primitivni polinom je , tj. 2. i 3. bit se dovode na ulaz XNOR funkcije, čiji se rezultat dovodi na mesto trećeg bita u registru.



Slika 3.2.2.1. Izlazne sekvence pri primeni registra dužine n=3 i primitivnog polinoma, pri Galois implementaciji

Kao i u slučaju Fibonacci metode perioda ponavljanja je T=7, sa izmenjenim redosledom sekvenci sa izlaza iz registra.

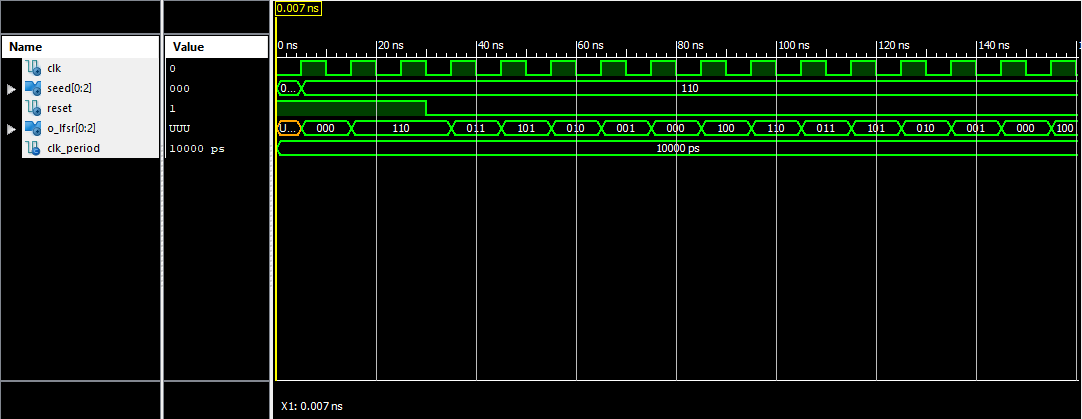
1. Simulacija

U okviru ISE softverskog paketa kompanije Xilinx koristi se ISim (ISE Simulator) simulator.  Simulacioni program se koristi za testiranje dizajna logike korišćenjem simulacionih modela za predstavljanje logičkih kola koja su u skladu sa dizajnom. Svrha simulacije je da se na osnovu zadatih vrednosti ulaznih signala proveri da li dizajn radi onako kako je predviđeno, odnosno na ovaj način se verifikuje njegova funkcija. Kao rezultat izvršenja, simulator generiše vremenske dijagrame koji pokazuju kako se signali iz VHDL koda menjaju u vremenu.

Kreiranjem testbench fajla izazivamo specijalne uslove u kojima se dizajn može naći. On poseduje entitet i arhitekturu, pri čemu njegov entitet nema spoljne ulaze tj. portove. U okviru arhitekture definiše se instanciranje komponente, proces generisanja stimulansa, proces generisanja reseta i proces utiskivanja inicijalne vrednosti u registar. Ovi procesi prikazani su kao ulazni signali na slici simulacije. Promene se dešavaju na uzlaznu ivicu takta, kao i aktivacija sinhronog reseta. Kao rezultat testa prikazuje se izlazni signal o\_lfsr, a tako i naša izlazna sekvenca.

* 1. Primer 1.

Za parametre odabrane u prethodnom poglavlju, registar duzine n=3, primitivni polinom , seed=110 i za Fibonacci metodu dobijamo sledeće rezultate:



Slika 4.1.1. Simulacija za Fibonacci model, za dužinu registra n=3 i odgovarajući primitivni polinom

Upoređivanjem izlazne sekvence ručno i softverski proračunate verzije LFSR za date parametre utvrđujemo da su one jednake. Simulacijom je utvrđeno da kod predstavlja verodostojan prikaz rada pomeračkog registra.

* 1. Primer 2.

Dodatak koji smo uveli je parametar sh\_size koji definiše broj pomeraja. Za parametre duzina registra, n=5, čiji je primitivni polinom, sa početnim vrednostima seed=11011 i broj pomeraja sh\_size=3, pokretanjem simulacije dobijamo sledeći proračun:

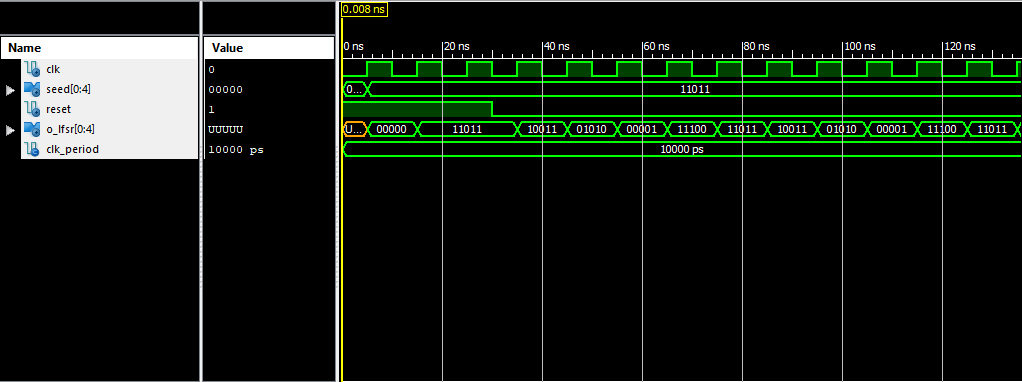


Slika 4.2.1. Simulacija za Fibonacci model, za parametre sh\_size=3,n=5, seed=11011 i primitivni polinom.

Uočavamo da je period ponavljanja sekvence T=31, čime se potvrđuje da je zadati polinom primitivan i da broj pomeraja ne utiče na period.

* 1. Primer 3.

Ako koristimo iste parametre kao iz primera 3, uz izmenjeni polinom povratne sprege, postavimo da su biti koji učestvuju u XNOR funkciji na pozicijama 3 i 4 dobijamo sledeći proračun:

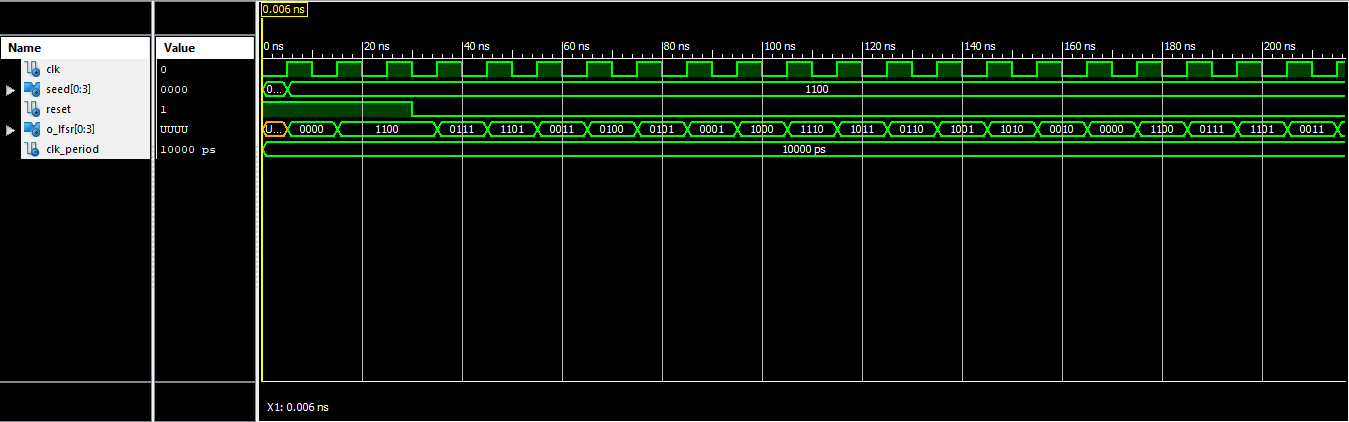


Slika 4.3.1. Simulacija za Fibonacci model, za parametre sh\_size=3,n=5, seed=11011 i polinom x^4+x^3+1

Na osnovu simulacije uočava se da je period ponavljanja T=5, što nije najoptimalnije rešenje u poređenju sa prethodnim primerom.

* 1. Primer 4.

Postavljanjem parametara n=4, sh\_size=2, primitivni polinom , seed=1100, pokretanjem simulacije dobijamo sledeće rezultate:



Slika 4.4.1. Simulacija za Fibonacci model, za parametre sh\_size=2,n=4, seed=1100 i polinom

Period ponavljanja T=15, kao što je i očekivano.

1. Performanse LFSR

Prevođenje jezičkih konstrukcija VHDL koda na hardverske elemente obavlja se u procesu sinteze. Za pojedine konstrukcije iz VHDL jezika postoje direktni hardverski ekvivalenti, pred neke druge se postavljaju posebni zahtevi i ograničenja kako bi mogle da se sintetišu, dok za mnoge jezičke konstrukcije hardverska interpretacija ne postoji. U procesu sinteze i analize dobijamo osnovne informacije o projektu, pregled upotrebljenih resursa, izveštaje o implementaciji dizajna, o koracima u procesu kompajliranja, vremenskom kašnjenju, izveštaje o elementima upotrebljenim za konstrukciju svih delova dizajna i sl. Dobija se uvid o performansama datog dizajna. U okviru ove teme ne radimo proces postavljanja i rutiranja, odnosno raspoređivanje dizajna na čip, pa rezultati u nastavku predstavljaju samo dobru procenu performansi.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Dužina registra | n=4 | | | n=30 | | | n=60 | | |
| Biti koji definišu prim.polinom | 3. i 4. | | | 2., 4., 6. i 30. | | | 59. i 60. | | |
| Pomeraj | sh\_size  =1 | sh\_size  =2 | sh\_size  =3 | sh\_size  =1 | sh\_size  =10 | sh\_size  =20 | sh\_size  =1 | sh\_size  =20 | sh\_size  =40 |
| Maksimalna frekvencija  [MHz] | 1027.908 | 1022.547 | 1022.547 | 916.632 | 521.485 | 498.455 | 1027.908 | 1022.547 | 1022.547 |
| Broj slajs registara | 4/  19200 | 4/  19200 | 4/  19200 | 30/  19200 | 30/  19200 | 30/  19200 | 60/  19200 | 60/  19200 | 60/  19200 |
| Broj slajs LUTova | 4/  19200 | 4/  19200 | 4/  19200 | 31/  19200 | 44/  19200 | 74/  19200 | 60/  19200 | 60/  19200 | 60/  19200 |

Tabela 5.1 Podaci iz izveštaja dobijeni u procesu analize i sinteze

Arhitektura FPGA((Field Programmable Gate Array) čipa zasnovana je na rešetkastoj mreži, pri čemu se u svakom polju ovakve mreže nalazi CLB (Configurable Logic Block) blok. CLB blok koristi LUT (LookUp Table) tabele za formiranje kombinacione logike. LUT tabela predstavlja malu memoriju u čijim se lokacijama čuva rezultat kombinacione logike, a adresa lokacije predstavlja vrednosti ulaznih signala - praktično LUT tabela predstavlja emulaciju tablice istinitosti. LUT ima ograničenja u vidu broja ulaznih podataka, pa se mora kombinovati više LUT.

U konkretnom primeru za dužinu registra n=30, veći broj bita gradi povratnu spregu, pa tako iziskuje veći broj XNOR funkcija.Usled povećenja broja pomeraja veći broj bita utiče na rezultat logičke funkcije, pa broj ulaznih podataka prelazi vrednost dozvoljenih šest i dolazi do povećanja broja LUT. Kod druge dve vrednosti dužine registra, njihovi primitivni polinomi uključuju samo dva bita u povratnu spregu, pa i usled ponavljanja ciklusa nije potrebno kombinovanje LUT.

Povećanje dužine registra dovodi do povećanja broja slajs registara.

Maksimalna frekvencija opada sa povećanjem broja ponavljanja, odnosno usled povećanja broja operacija i kompleksnosti sistema. Kod primera sa dužinom registra n=30 sa povećanjem broja pomeraja znatno se usložnjava sistem, pa je pad frekvencije uočljiviji.

1. Zaključak

Osobina pseudoslučajnih sekvenci da su u potpunosti determinističke, sa poznatim karakteristikama, pri čemu izgledaju kao skupovi slučajnih podataka našla im je primenu u raznim sferama modernog društva.

Ove sekvence liče na slučajne signale što je dovoljno da kodirani signal bude neprepoznatljiv za korisnika kome nije namenjen, pa su tako pronašle primenu u brojnim kriptografskim aplikacijama i kod skremblovanja podataka.

Mogućnost ponavljanja pseudoslučajne sekvence koristi se kod višestrukog pristupa na bazi kodne raspodele i u sistemima sa proširenim spektrom.

Obe osobine neophodne su kod kompresije podataka, kod kodova za detekciju i ispravljanje grešaka, kao i u testiranju veza u realnim uslovima.

Kroz datu temu smo uočili da je LFSR jednostavan za realizaciju, zahteva malu količinu resursa i velikom brzinom generiše pseudoslučajne sekvence, što ga čini idealnim kandidatom za funkciju generatora pseudoslučajnih sekvenci.

Literatura

1. A Million Random Digits with 100,000 Normal Deviates [Online]. Available: <https://en.wikipedia.org/wiki/A_Million_Random_Digits_with_100,000_Normal_Deviates>
2. Middle-square method [Online]. Available: <https://en.wikipedia.org/wiki/Middle-square_method>
3. Linear-feedback shift register [Online]. Available: <https://en.wikipedia.org/wiki/Linear-feedback_shift_register>
4. Pseudoslučajne binarne sekvence(prbs) [Online]. Available: <http://es.elfak.ni.ac.rs/Papers/ASavic%20-%20PMitic%20-%20tehdoc.pdf>
5. Zoran Čiča, *Programiranje komunikacionog hardvera,* 2013.
6. Prilog
   1. VHDL kod za lfsr za dužinu *n*=4, primitivni polinom *pol*=0011 i broj pomeraja *sh\_size*=2

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity lfsr is

generic(

sh\_size: integer :=2;

n: integer :=4;

pol :std\_logic\_vector :="0011");

port

(

clk: in std\_logic;

seed: in std\_logic\_vector(0 to n-1);

reset: in std\_logic;

o\_lfsr :out std\_logic\_vector(0 to n-1)

);

end lfsr;

architecture Behavioral of lfsr IS

begin

process(clk,reset)

variable r\_lfsr: std\_logic\_vector(1 to n);

variable r1 :std\_logic;

variable c :std\_logic;

variable k :integer;

variable i :integer;

begin

IF(clk'EVENT AND clk='1')THEN

IF(reset='1')THEN

r\_lfsr:=seed;

ELSE

k:=0;

while(k<sh\_size) LOOP

c:='0';

i:=1;

while(i<n+1) LOOP

if(pol(i-1)='1') THEN

if(c='1') THEN

r1:= r1 XNOR r\_lfsr(i);

else r1:=r\_lfsr(i);

END IF;

c:='1';

END IF;

i:=i+1;

END LOOP;

r\_lfsr(2 to n):=r\_lfsr(1 to n-1);

r\_lfsr(1):=r1;

k:=k+1;

end loop;

end if;

end if;

o\_lfsr<=r\_lfsr(1 to n);

end process;

end architecture Behavioral;

* 1. Testbench za dužinu registra *n*=4, inicijalnu vrednost *seed*=1100

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY test IS

END test;

ARCHITECTURE behavior OF test IS

COMPONENT lfsr

PORT(

clk : IN std\_logic;

seed : IN std\_logic\_vector(0 to 3);

reset : IN std\_logic;

o\_lfsr : OUT std\_logic\_vector(0 to 3)

);

END COMPONENT;

--Inputs

signal clk : std\_logic := '0';

signal seed : std\_logic\_vector(0 to 3) := (others => '0');

signal reset : std\_logic := '0';

--Outputs

signal o\_lfsr : std\_logic\_vector(0 to 3);

-- Clock period definitions

constant clk\_period : time := 10 ns;

BEGIN

uut: lfsr PORT MAP (

clk => clk,

seed => seed,

reset => reset,

o\_lfsr => o\_lfsr

);

PROCESS

BEGIN

--generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'

loop

clk<='0';

wait for 5 ns;

clk<='1';

wait for 5 ns;

end loop;

END PROCESS;

stim\_proc: process

begin

reset<='1';

wait for 30 ns;

loop

reset<='0';

wait for 500 ns;

end loop;

end process;

process(clk,reset)

begin

if (clk'event and clk='1') then

if(reset='1') then

seed<="1100";

end if;

end if;

end process;

END;