Univerzitet Union

Računarski fakultet

Beograd

Konstrukcija i analiza hardverskog USB generatora slučajnih brojeva sa primenom u kriptografiji

DIPLOMSKI RAD

Kandidat:

Mihailo Jovičić RM 01/2012

Mentor:

Prof. dr Đorđe M. Babić

Beograd, 9.2015.

	УНИВЕРЗИТЕТ „УНИОН“ **РАЧУНАРСКИ ФАКУЛТЕТ** Кнез Михаилова 6/VI 11000 БЕОГРАД	Број: Датум:

UNIVERZITET UNION
RAČUNARSKI FAKULTET
BEOGRAD
Računarske mreže i komunikacije

DIPLOMSKI RAD

Kandidat: **Mihailo Jovičić**
Broj indeksa: RM 01/2012

Tema rada:
Konstrukcija i analiza hardverskog USB generatora slučajnih brojeva sa primenom u kriptografiji

Mentor rada: **Prof. dr Đorđe M. Babić**

Beograd, 9.2015.

SADRŽAJ

1. UVOD

"Podrazumeva se da greši svako ko razmatra korišćenje aritmetičkih metoda za dobijanje slučajnih brojeva."

John Von Neumann (1951)[1]

Značaj generatora slučajnih brojeva je izuzetno veliki u računarstvu, i u mnogim drugim granama nauke. Slučajni brojevi se koriste pri simulacijama prirodnih fenomena gde su neophodni za realističan opis događaja, kao npr. u nuklearnim simulacijama (gde se čestice nasumično sudaraju) ili pri kretanju vozila u saobraćaju. Od velikog značaja su i u numeričkoj analizi gde se traže približni rezultati, u okviru neke dozvoljene margine greške, na probleme koji nisu egzaktno rešivi. Imaju primenu u računarstvu gde je potrebno proveriti rad određenih algoritama unoseći veliki broj kombinacija ulaznih podataka, u industriji pri slučajnom odabiranju proizvoda za uzorkovanje sa proizvodne trake, ili u jednostavnim primenama kao što je simuliranje bacanja kockice.

Za veliki broj primena slučajnih brojeva potrebne su velike količine podataka i brzo generisanje brojeva. Za takve primene se danas najčešće koriste pseudo-slučajni tj. kvazi-slučajni generatori brojeva, koji se baziraju na generisanju pseudoslučajnih sekvenci pomoću određenog algoritma koji se izvršava na računaru. Iako su danas izuzetno brzi i daju vrlo dobre rezultate u statističkim testovima, i dalje svoj rad baziraju na determinističkom algoritmu, i njihov izlaz samo deluje slučajnim. Poznavanjem algoritma generisanja slučajnih brojeva, kao i početnog broja (ili vektora) koji se koristi za inicijalizaciju algoritma, moguće je dobiti potpuno iste sekvence pseudoslučajnih brojeva. Za mnoge primene takvo ponašanje nije problematično, čak je i poželjno nekada ponoviti istu sekvencu generisanih brojeva (npr. za testiranje nekog algoritma). Međutim to predstavlja problem pri primeni u kriptografiji.

Kriptografija je jedna od retkih primena za generatore slučajnih brojeva gde je od izuzetno velikog značaja nedeterminističko ponašanje samog generatora. Pogodnost je što kriptografija ne zahteva velike količine slučajnih brojeva i njihovo brzo generisanje kao npr. računarske simulacije. Za primenu u kriptografiji se i danas preporučuju hardverski generatori slučajnih brojeva koji se baziraju na kvantifikovanju nekog fizičkog procesa koji je slučajne prirode. Fizički procesi koji se najčešće koriste kao izvor slučajnosti su: termički šum, lavinski i Zenerov efekat, atmosferski radio šumovi [2], radioaktivni raspad koji se meri Gajgerovim brojačem [3], fotoni koji prolaze kroz polupropusna ogledala, itd.. Iako slučajni po prirodi, potrebno je pažljivo okarakterisati takve izvore šuma koji nisu uvek ravnomernog spektralnog rasporeda, tj. ne generišu istom učestanošću ceo spektar izlaznih brojnih vrednosti. Kako bi se izbegli takvi problemi, koriste se različite tehnike obrade prikupljenih podataka, od kojih će neke biti korišćene u ovom radu.

Značaj hardverskih generatora slučajnih brojeva u kriptografiji je veliki zbog same prirode generisanja brojeva. Time što je neki generator brojeva hardverski, a ne algoritamski pseudo-generator, garantuje da je nemoguće pronaći zakonitost prema kojoj su brojevi generisani. Time je takođe nemoguće predvideti naredne sekvence brojeva jer se generator ne ponaša deterministički. Međutim, postoje i mane hardverskih generatora slučajnih brojeva, od kojih je najznačajnija brzina generisanja brojeva, koja je mala i ograničava ih na primenu gde se zahtevaju kratke sekvence slučajnih brojeva. Velika mana hardverskih generatora može biti nezaštićen ili loše odabran izvor slučajnosti koji je podložan spoljnim uticajima, preko kojih zlonamerne osobe mogu da utiču na rad generatora i na dobijene izlazne sekvence brojeva. Npr. ukoliko se hardverski generator bazira na

kvantifikovanju okolnog radio šuma, moguće je emitovati radio signal na određenoj frekvenciji koji bi u najgorem slučaju mogao da napravi potpuno deterministički izvor brojeva od hardverskog generatora slučajnih brojeva, i time napravi ozbiljan sigurnosni propust ukoliko je hardverski generator korišćen u kriptografiji.

Zbog sve veće potrebe za smeštanjem podataka na udaljenim serverima kao i povećanjem obima komunikacije i broja čuvanih podataka u digitalnom obliku, značaj kriptografije je sve veći u oblasti zaštite privatnosti. Time je porastao i značaj hardverskih generatora slučajnih brojeva, pa je kompanija Intel od 2012. godine proizvela prve komercijalne mikroprocesore sa ugrađenim hardverskim generatorom slučajnih brojeva [4]. Iako koriste termalni šum kao izvor slučajnosti na koji je nemoguće uticati, sama RDRAND instrukcija koristi hardverski generator šuma unutar procesora samo kao inicijalizator pseudo-slučajnog algoritma. Kasnije je ustanovljeno da postoje potencijalna ugrađena "zadnja vrata" i da je promenom mikrokoda u mikroprocesoru moguće uticati na izlaz samog generatora, zbog čega se više ne koristi ovakav način generisanja slučajnih brojeva kao primarni u FreeBSD kernelu Linux operativnih sistema. Time se uočava još jedan potencijalni problem loših hardverskih generatora slučajnih brojeva, a to je mogućnost promene softvera u delu za obradu podataka koji su prikupljeni sa inače nezavisnog izvora nedeterminističkih slučajnih brojeva.

Imajući u vidu prednosti i mane hardverskih generatora slučajnih brojeva, za potrebe ovog rada je razvijen i napravljen jedan tip hardverskog generatora, koji bi trebalo da sadrži sve prednosti dobrih hardverskih generatora, bez očiglednih sigurnosnih propusta. Generator je osmišljen tako da izbegne poznate probleme koje poseduju određeni hardverski generatori, pre svega mogućnost da se promenom parametara sredine u kojoj se nalazi utiče na njihov izlaz. Zbog toga, generator nije baziran na merenju atmosferskog ili okolnog šuma [2], već se bazira na internom generisanju šuma preko inverzno polarisanog PN spoja. Uređaj čine napajanje, sam generator šuma i višestepeni pojačavač, kao i mikrokontroler koji obrađuje prikupljene podatke, održava temperaturu konstantom (upravljanjem grejačem i merenjem temperature preko digitalnog termometra) i komunicira sa PC računarom. Sam izvor šuma je zaštićen od uticaja spoljne sredine na više načina:

- Oklapanjem u Faradejev kavez kako bi se izbegao uticaj RF talasa koji bi mogli da indukuju neki drugi signal pored osnovnog šuma. Ovo je izuzetno značajno s'obzirom na veoma malu amplitudu napona koja se javlja na PN spoju tranzistora, a koja se kasnije pojačava, obrađuje i koristi kao izvor slučajnosti.
- Temperaturnom stabilizacijom izvora šuma i prvog stepena pojačavača na fiksnu temperaturu od +45°C (+-0,3°C). Zbog samog procesa dobijanja šuma na inverzno polarisanom PN spoju, i zavisnosti u odnosu na temperaturu, potrebno je da jednom podešen hardverski generator slučajnih brojeva radi pouzdano i tačno u različitim sredinama, bez promena u statistici generisanih brojnih vrednosti sa izlaza.
- Filtracijom i DC/DC konverzijom ulaznog napona USB veze preko koje se napaja sam uređaj, kao i naknadom konverzijom i filtracijom. Ulazni napon se sa ~5V podiže na 18V, i naknadno spušta na 15V i 5V napone koji su potrebni za rad uređaja. Time se sprečava uticaj napajanja računara, smetnji na samom napajanju (slučajnih ili namerno indukovanih) ili dužine USB provodnika i pada napona.
- Komunikacija sa personalnim računarom je jednosmerna, i uređaj samo šalje podatke. Time se izbegava mogućnost zlonamernog upada u sam program mikrokontrolera unutar uređaja i povećava bezbednost.

Pored zaštite od uticaja spoljne sredine i zlonamernih napada na generator, izuzetno značajan je i sam kvalitet dobijenih slučajnih brojeva. Da bi se dobio kvalitetan generator slučajnih brojeva potrebno je više korekcija, od analogne kalibracije samog izvora šuma, preko statističke analize dobijenih odbiraka i korekcije unutar softvera mikrokotrolera u samom uređaju, do naknadnih softverskih korekcija dobijenih vrednosti kako bi se korelacija između odbiraka svela na minimalnu. Takođe uvedena je i vremenska zadrška slučajnog trajanja između dva odbirka analogno/digitalne konverzije kako bi se izbegao eventualni uticaj indukovanih periodičnih signala i uvela još jedna dimenzija slučajnosti. Tako pripremljeni podaci se šalju u binarnom ASCII obliku preko USB veze ka personalnom računaru, i mogu se odmah koristiti u određenim aplikacijama u kojima je potreban rad sa slučajnim brojevima.

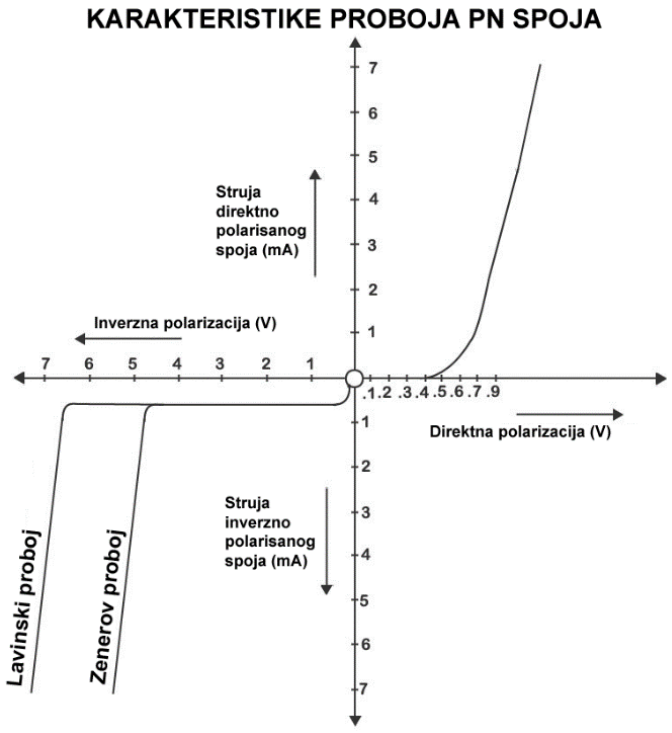
Dobijeni brojevi sa izlaza hardverskog generatora slučajnih brojeva su sakupljani i podvrgnuti statističkim testovima koji se koriste u svrhu provere kvaliteta generatora slučajnih brojeva, kako bi se kvantifikovao kvalitet napravljenog generatora. Urađeno je više testova za kvantifikovanje izlaznih vrednosti sa generatora slučajnih brojeva, i korišćeni su trenutno najmerodavniji paketi za testiranje generatora slučajnih brojeva, pomoću kojih su ocenjeni skoro svi današnji generatori slučajnih brojeva.

Cilj rada je da pokaže kompletan razvoj takvog hardverskog generatora, od opisa prirode slučajnog šuma, njegovog korišćenja i obrade u analognom i digitalnom domenu, različitih statističkih analiza dobijenih slučajnih brojeva, kao i praktične primene u kriptografiji na primeru jedne aplikacije za sigurnu mrežnu komunikaciju.

2. FIZIČKI PROCESI KOJI PROIZVODE ŠUM U INVERZNO POLARISANOM PN SPOJU

Kao izvor šuma za opisani hardverski generator slučajnih brojeva, koji se kasnijom kvantifikacijom i obradom pretvara u niz slučajnih brojeva, je uzet inverzno polarisan PN spoj. Analogni nedeterministički šum na njemu nastaje usled Schottky šuma. Za inverzno polarisan PN spoj može biti korišćena inverzno polarisana dioda, ili u ovom slučaju emiter-baza spoj bipolarnog tranzistora. Za tranzistor je odabran široko rasprostranjeni model 2N2222A koji je tranzistor NPN tipa i planarno epitaksijalne konstrukcije na bazi silicijuma. Značajan podatak pri korišćenju ovog tranzistora kao izvora šuma jeste napon proboja emitor-baza, koji je specifikovan u karakteristikama tranzistora od strane proizvođača [9] i iznosi minimalno 6V pri struji od 10 µA. U praksi taj napon iznosi do ~7.2V. Amplituda šuma inverzno polarisanog spoja ovog tranzistora je velika u poređenju sa drugim tranzistorima i izmerena je na oko 420mVpp, što ga čini pogodnim za korišćenje kao izvora šuma.

Mehanizam nastanka šuma na inverzno polarisanom PN spoju je Schottky (skraćeno Shot) tipa koji je povezan sa Zenerovim probojem (Zenerov efekat), ili lavinskim probojem (lavinski efekat). Zenerov efekat i Shot šum uglavnom ispoljavaju inverzno polarisani PN spojevi kojima je napon proboja do 5-8V, a lavinski efekat svi kojima je napon proboja preko 5-8V. Između 5V i 8V može doći do manifestacije oba efekta. Pošto se napon proboja tranzistora 2N2222A nalazi negde na granici ove podele, pretpostavlja se da oba efekta dovode do stvaranja šuma i biće ukratko opisani



Slika 1: Karakteristike proboja PN spoja

Schottky šum ili skraćeno Shot šum je prvi put opisao Nemački naučnik Walter Hermann Schottky 1918. godine [6]. Prilikom kretanja elektrona preko PN barijere, elektroni pristižu u diskretnim vremenskim trenucima. Takvo diskretno pristizanje elektrona izaziva Schottky šum. Amplituda napona Schottky šuma zavisi od temperature, kao i naponska tačka preko koje inverzno polarisan PN spoj kreće da ispoljava lavinski efekat umesto Zenerovog (slika 1). Zbog toga je u hardverski generator slučajnih brojeva ugrađena temperaturna stabilizacija, kao i zbog otpornika u okolini tranzistora koji proizvode sopstveni Johnson-ov šum čija je amplituda takođe direktno srazmerna temperaturi otpornika.

Zenerov efekat je tip električnog proboja inverzno polarisanog PN spoja koji nastaje kod PN spojeva koji sadrže veliku koncentraciju nečistoća i koji imaju vrlo tanke prelazne regione. Kod njih dolazi do stvaranja jakih električnih polja na prelaznim spojevima pri vrlo malim naponima. Povećavanjem napona inverzne polarizacije dolazi do jačanja električnog polja i pri određenom naponu se dešava Zenerov proboj. Zenerov proboj predstavlja direktno prekidanje kovalentnih veza od strane sile električnog polja i zavisi direktno od maksimalne jačine električnog polja. Sam Schottky šum je povezan sa strujom koja prolazi kroz PN barijeru, i nastaje usled slučajne emisije elektrona koja izaziva fluktuacije struje oko određene srednje vrednosti. Efektivna vrednost Schottky šuma [5, 6] je izražena preko formule :

$$I_{schottky(RMS)} = \sqrt{2\ q\ I_{dc}\ B}$$

gde su: q=naelektrisanje elektrona ($1{,}6 * 10^{-19}$ Kolumba),

I_{dc}=srednja vrednost struje (A), B=širina frekventnog opsega

Kao i Johnson-ov šum, Schottky šum ima karakteristiku belog šuma, što znači da ima konstantnu spektralnu gustinu snage. Ukoliko se prema Omovom zakonu prethodna formula izrazi tako da se dobije napon šuma, efektivna vrednost napona iznosi:

$$U_{schottky(RMS)} = r_d\sqrt{2\ q\ I_{dc}\ B}$$

gde je r_d dinamički otpor PN spoja, izražen preko:

$$r_d = kT/qI_{dc}$$

gde su: k=Boltzmanova konstanta (1,38*e^{-23} Joule/Kelvin), T=temperatura u Kelvinima

Zamenom se dobija:

$$U_{schottky(RMS)} = \sqrt{2\ kT\ Br_d}$$

Dobijena jednačina je vrlo slična jednačini termičkog šuma (Johnson-ov šum) [6, 7] :

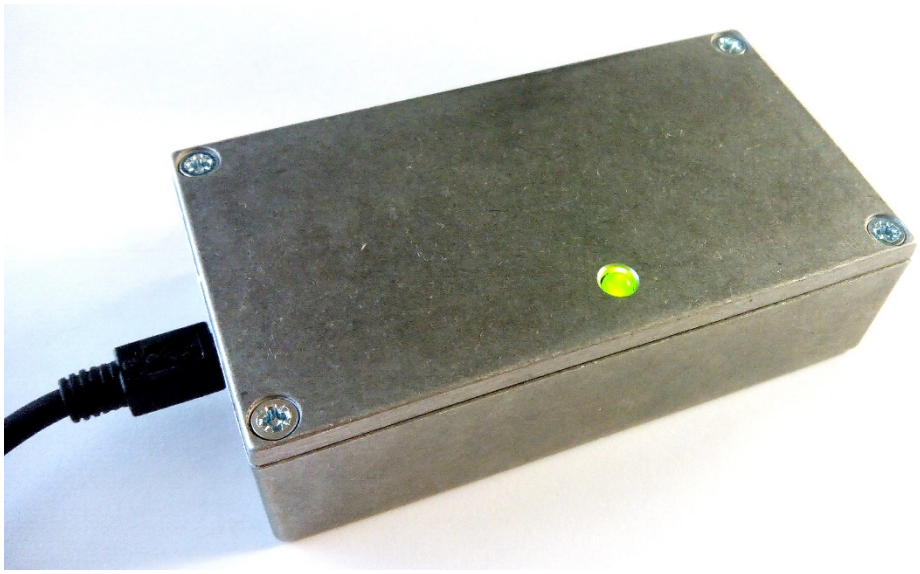
$$U_{Johnson(RMS)} = \sqrt{4\ kT\ Br}$$

Iz obe jednačine se uočava da efektivna vrednost napona šuma direktno zavisi od temperature, zbog čega je i uočena potreba za temperaturnom stabilizacijom tranzistora koji se koristi kao izvor šuma, kao i okolnih otpornika. Samo na taj način je moguće jednom podesiti analognu i digitalnu sekciju hardverskog generatora slučajnih brojeva i očekivati da će se prilikom svakog rada uređaja dobijati isti kvalitet statističkih rezultata sa izlaza generatora.

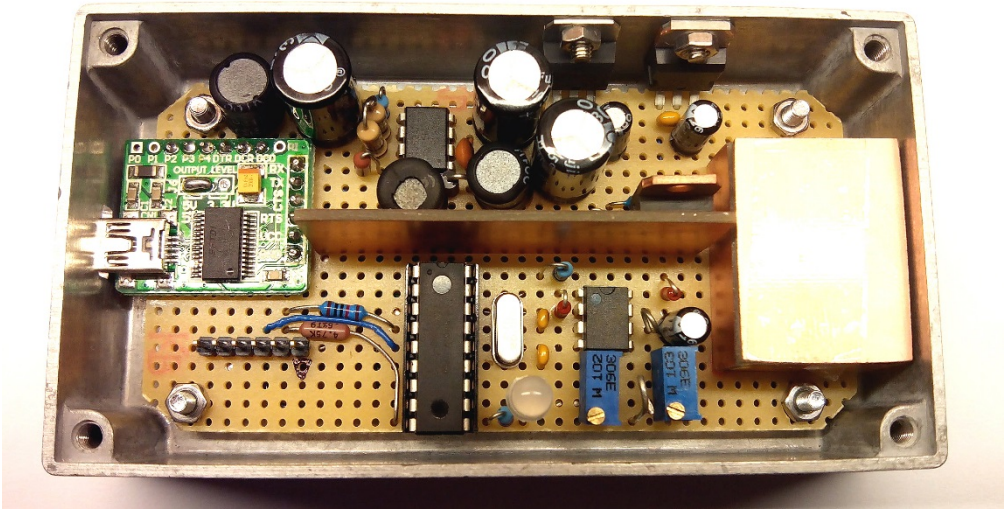
Daljim povećanjem inverznog napona dolazi do lavinskog proboja PN spoja [8] (slika 1). Javlja se pri jačinama električnog polja (većim od 10^6V/cm) u silicijumskim spojevima dopiranim sa oko 10^{18}atoma/cm ^{3} i naponom inverzne polarizacije preko 5V. Lavinski proboj ili lavinsko umnožavanje je vrlo komplikovan proces koji nastaje pri velikim jačinama električnog polja u kome nosioci naelektrisanja poseduju dovoljnu energiju da razbiju kovalentne veze sudarajući se sa kristalnom strukturom. Svaki takav jonizujući sudar proizvodi elektron i rupu, od kojih se oba ubrzavaju od strane električnog polja i potencijalno proizvode još jonizujućih sudara. Svi nosioci naelektrisanja doprinose ukupnoj vrednosti inverzne struje PN spoja. Šum koji se javlja je komplikovaniji od Shottky šuma Zenerovog proboja i poseduje neregularne vrednosti šuma iznad i ispod amplitude takvog Shottky šuma. Daje jako širok frekventni opseg do par desetina Ghz.

3. KONSTRUKCIJA HARDVERSKOG GENERATORA SLUČAJNIH BROJEVA

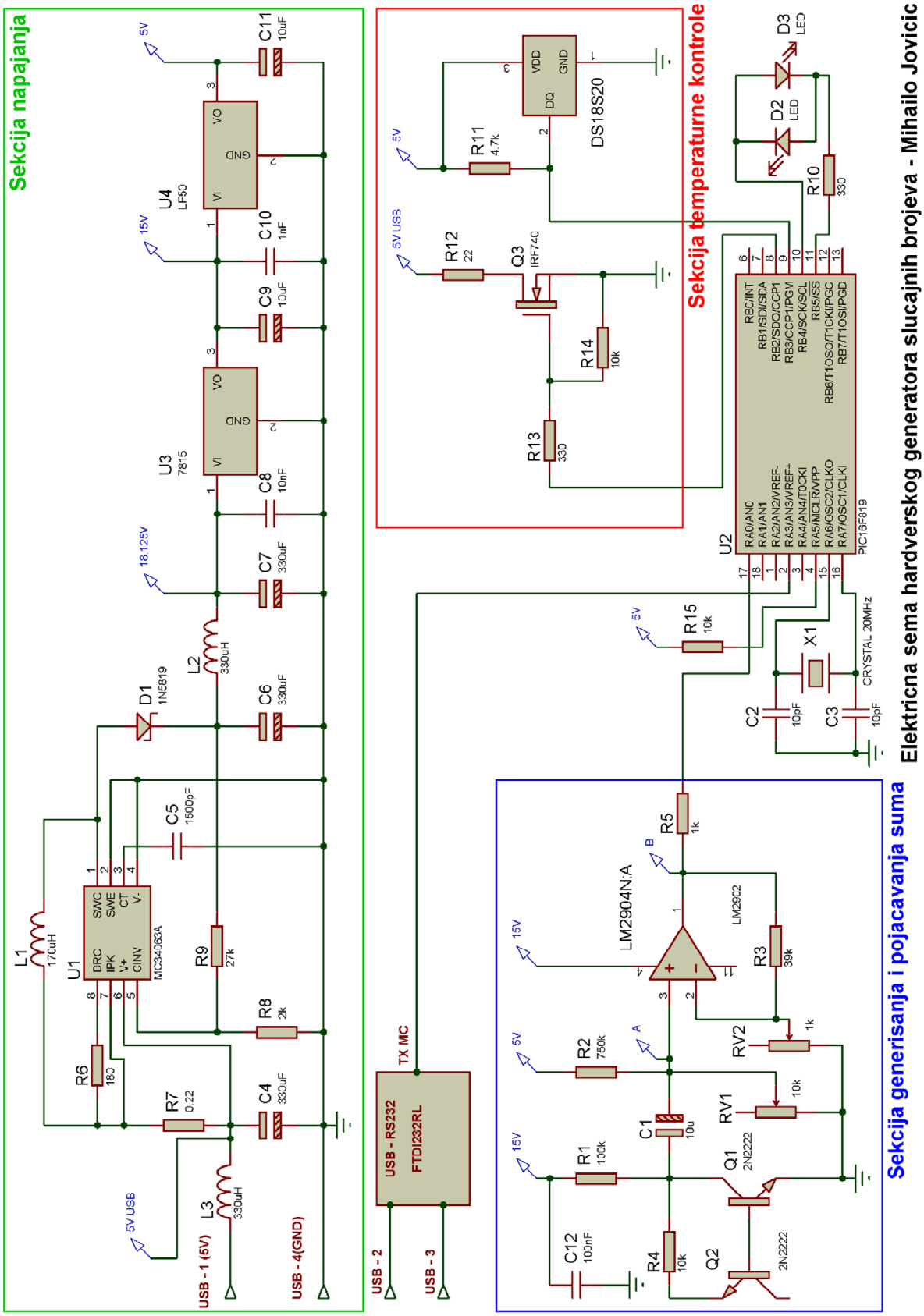
3.1. FOTOGRAFIJE I ELEKTRIČNA ŠEMA UREĐAJA



Slika 2: Spoljašnji izgled generatora slučajnih brojeva



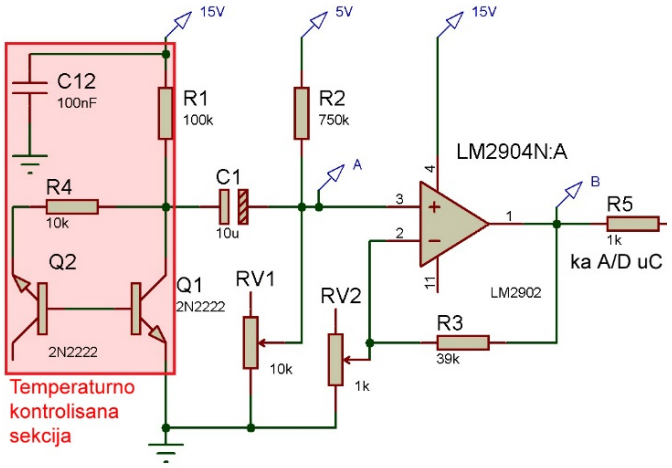
Slika 3: Unutrašnji izgled generatora slučajnih brojeva



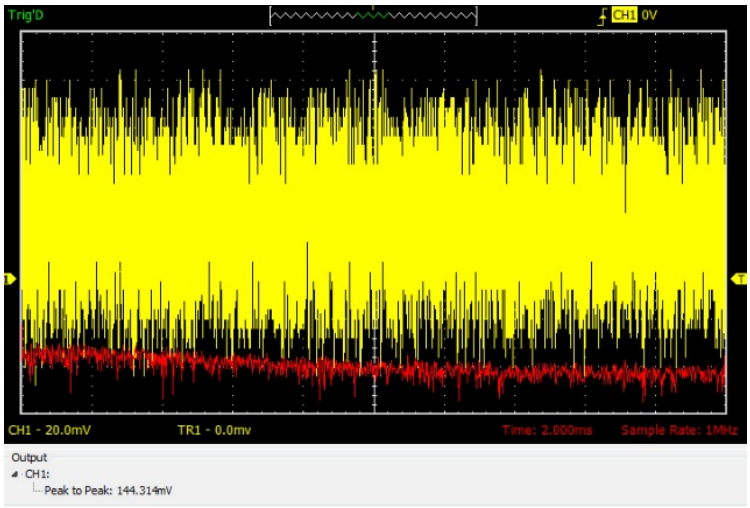
Slika 4: Električna šema hardverskog generatora slučajnih brojeva

3.2. SEKCIJA GENERISANJA I POJAČAVANJA ŠUMA

Sekcija za generisanje šuma (slika 5) je bazirana na prethodno opisanim efektima Schottky šuma (usled Zenerovog i lavinskog efekta) i Johnson-ovog šuma, i realizovana je iz 2 segmenta. Napon na kolektoru tranzistora Q1 treba da ima vrednost iznad zbira napona inverzno polarisanog PN spoja tranzistora Q2 (koji iznosi oko 6V) i napona V_{BE} tranzistora Q1 (za koga se pretpostavlja da iznosi oko 0,6V), tj. iznad 6,6V. Eksperimentalno je utvrđena vrednost napona od 15V kao dovoljno velika da izazove lavinski efekat i da se dobije simetričan analogni oblik šuma na osciloskopu. Na kolektoru tranzistora Q1 će postojati vrednost napona od oko 6,6V, a odatle se izračunava struja kroz emiter tranzistora Q1 definisana otpornikom R1 , $I_E = (15V - 6.6V)/100k\Omega = 84\mu A$. Ova struja se deli na struju kolektora $I_c = \alpha I_E$ i na struju baze $I_B = I_c/\beta$. Prema karakteristikama proizvođača za bipolarni tranzistor 2N2222A [9], parametar pojačanja $\beta(h_{fe})$ iznosi 50. Zamenom vrednosti u formule uz pretpostavku da je parametar α vrlo blizak vrednosti 1, dobija se vrednost struje na bazi tranzistora Q1, odnosno struje kroz inverzno polarisan PN spoj tranzistora Q2 (koji je izvor šuma) od $1{,}68\mu A$. Takođe, postoji termalni šum sa otpornika R4. Maksimalna amplituda ukupnog napona šuma merenog na kolektoru nakon kondenzatora C1 iznosi oko 150mV i dobijeni oblik signala je prikazan na slici 6 (žutom bojom). Na istoj slici crvenom bojom je prikazan frekventni spektar signala u linearnom rasponu od 0-500kHz i vertikalnoj skali izraženoj u dBV. Vidi se da šum ima veoma širok frekventni spektar, ali da zbog konstrukcije uređaja bez štampane pločice i slabe rezolucije osciloskopa na malom naponu, više frekvencije pokazuju manje slabljenje.



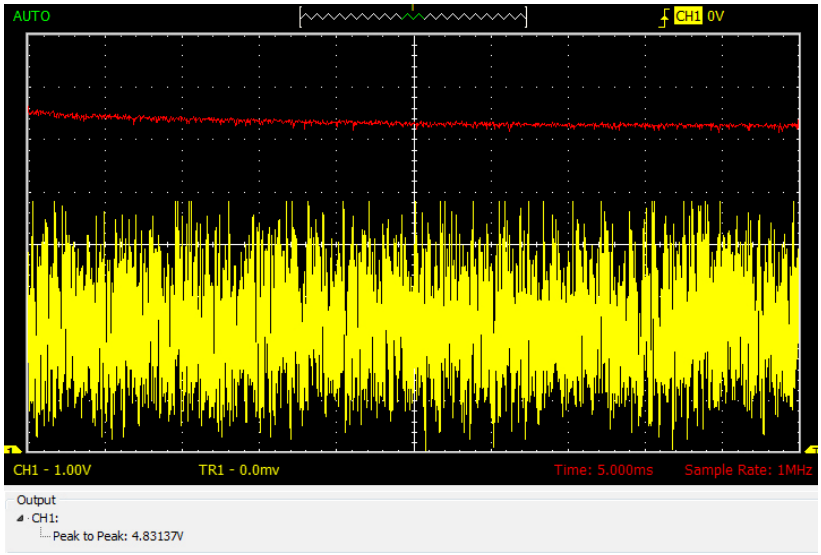
Slika 5: Sekcija generisanja i pojačavanja šuma



Slika 6: Šum meren na kolektoru tranzistora Q1

Sledeći segment ove sekcije radi prilagođenje napona šuma na ulaz A/D konvertora mikrokontrolera koji prihvata napone 0-5V. Potrebno je odstraniti jednosmernu komponentu napona uz pomoć elektrolitskog kondenzatora C1. Nakon njega je postavljen razdelnik napona, koga čine potenciometar RV1 od 10kΩ kao i otpornik od 750kΩ, koji postavlja jednosmernu komponentu napona na oko 75mV, čime se dobija signal šuma sa vrednostima između 0 i 150mV. Potrebno je pojačati takav signal, što je izvršeno pomoću operacionog pojačavača LM2904N [10]. Koristi se neinvertujuća topologija operacionog pojačavača čije je pojačanje opisano jednačinom $P = 1 + R3/RV2$.

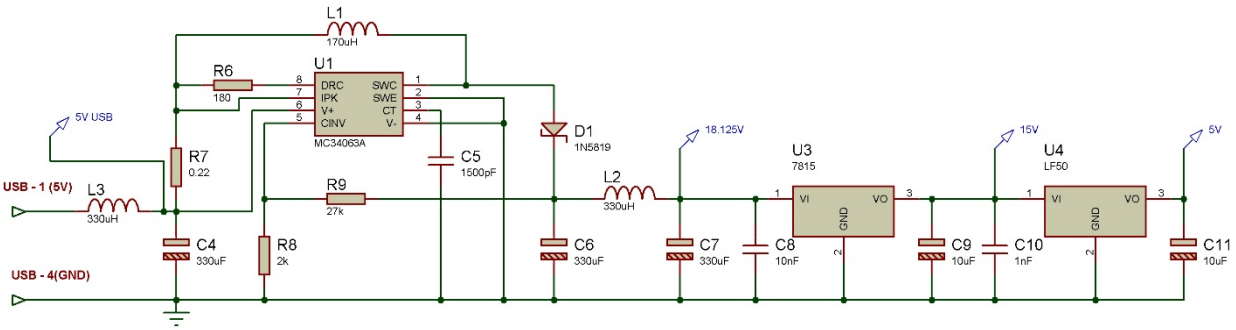
Nakon dostizanja radne temperature, prilikom prvog uključenja uređaja potrebno je podesiti višeobrtne potenciometre RV1 i RV2 kako bi se dobio šum koji će biti dovoljno pojačan i simetričan oko naponske tačke polovine ukupnog merenog napona (2,5V). Simetričnost se podešava dodavanjem ili oduzimanjem jednosmerne komponente napona pomoću potenciometra RV1, a pojačanje potenciometrom RV2. Šum bi trebalo da je dovoljno pojačan kako bi se iskoristila maksimalna rezolucija A/D konvertora mikrokontrolera (10 bita na opsegu 0-5V, tj. 4,88mV po koraku, u slučaju korišćenog PIC16F819 [16]), ali ne previše, kako ne bi došlo do odstranjivanja maksimalnih vrednosti napona šuma. Pravilno podešen signal je prikazan na slici 7 (žutom bojom). Uočava se maksimalna amplituda signala od 4,83V, simetričnost šuma oko srednje vrednosti napona od 2,5V, kao i dalje širok frekventni spektar (crvenom bojom, 0-500kHz, vertikalna skala u dbV). Ovako pripremljen signal se prosleđuje na ulaz analogno/digitalnog konvertora ugrađenog u mikrokontroler, i dalje kvantifikuje i obrađuje pre slanja ka računaru.



Slika 7: Šum meren na otporniku R5, nakon podešavanja potenciometara RV1 i RV2

Posmatrajući oblik napona analognog šuma (slika 7) na osciloskopu (čija se posmatrana karakteristika bolje vidi na analognom osciloskopu zbog osobine remanencije ekrana), jasno se uočava da su analogne vrednosti napona sa generatora šuma mahom grupisane oko srednje vrednosti tj. jednosmerne komponente šuma. Jednostavnom A/D konverzijom ovako dobijenog signala i slanjem takvih vrednosti kao slučajnih ka računaru, dobio bi se vrlo loš generator slučajnih brojeva koji bi veliki procenat generisanih brojeva dobijao oko srednje vrednosti (analogno napon oko 2,5V , digitalno broj vrednosti oko 512). Jedini način dobijanja potpuno nasumičnih vrednosti jeste jednostavno određivanje da li se neka pročitana analogna vrednost nalazi iznad srednje vrednosti, što daje binarno 1, ili ispod što daje binarno 0. Time se izbegava problem grupisanja brojeva, ali i usporava rad hardverskog generatora slučajnih brojeva. Nije dovoljno poslati preko USB veze tako dobijene nule i jedinice, već je potrebno kalibrisati srednju vrednost, tj. prag , ali i smanjiti korelaciju međusobnih odbiraka, o čemu će biti reči u narednim poglavljima. Takođe uvodi se i neravnomerno vremensko odabiranje signala A/D konverzije, kako bi se uvela dodatna slučajnost i izbegla eventualna indukovana smetnja periodičnih signala.

3.3. SEKCIJA NAPAJANJA



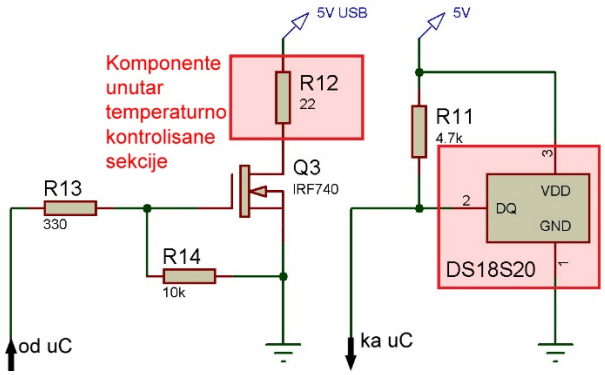
Slika 8: Napajanje hardverskog generatora slučajnih brojeva

Napajanje hardverskog generatora slučajnih brojeva je zahtevalo više izlaznih napona, kao i zaštitu od napada preko napajačkih provodnika. U slučaju direktnog korišćenja napona od 5V sa USB veze, mikrokontroler bi koristio taj napon kao 5V referencu A/D konvertora, pa bi se smanjivanjem napona napajanja dobijale vrednosti slučajnih brojeva koje više ne bi bile balansirane oko podešene srednje vrednosti. Takođe, dodavanjem šuma na linijama napajanja, takav signal bi se propagirao i do same sekcije za generisanje šuma, pa bi bilo moguće uticati na vrednosti generisanih slučajnih brojeva.

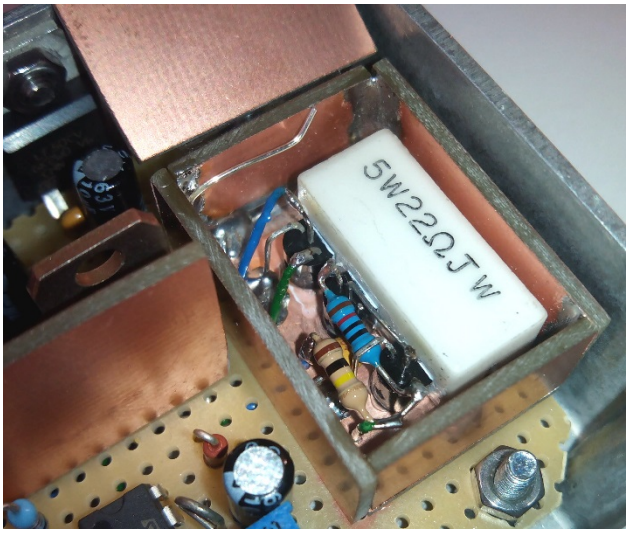
Iz tih razloga, napajanje uređaja (slika 8) na samom ulazu sa USB konektora koristi niskopropusni LC filter, koji prema jednačini $F = 1/2\pi\sqrt{LC}$ daje graničnu frekvenciju od 482Hz, dok se u nepropusnom opsegu slabljenje filtera pojačava 12dB/oktavi. Dobijeni, grubo filtriran, napon od 5V se direktno koristi samo za napajanje grejača u sekciji za održavanje temperature, dok sva ostala elektronika koristi visokofrekventni DC/DC pretvarač koji podiže ulazni napon na 18V. Koristi se integrisano kolo MC34063A [11], koje sadrži većinu neophodnih komponenti za rad DC/DC pretvarača. Radi na 100kHz i ima preciznost održavanja napona od 2%. Sa efikasnošću od oko 87%, malo snage se gubi na grejanje, s'obzirom da ceo uređaj (bez grejača) ima potrošnju od oko 160mA. Dodatnim filterom koga čine L2 i C7 se smanjuje naponsko talasanje sa amplitude od 400mV na oko 40mV. Otpornicima R9 i R8 se zadaje izlazni napon koji se, prema ugrađenoj naponskoj referenci od 1,25V, dobija formulom $U = 1.25(1 + \frac{R9}{R8})$.

Nakon povećanja napona na vrednost od 18V, napon se smanjuje na 15V i 5V korišćenjem linearnih naponskih regulatora, koji naponsku razliku disipiraju u obliku toplote. Zbog male struje koju troše ostale elektronske komponente, disipacija ne predstavlja problem jer je vrlo mala. Prvi linearni regulator tipa LM7815 [12] zahteva minimalno 2,7V veći napon na ulazu nego na izlazu, zbog čega je i odabran napon od 18V kao izlaz sa DC/DC konvertera. Disipacija u obliku toplote iznosi $P_g = \frac{0.16\text{A}*5\text{V}}{18\text{V}} * (18V - 15V) = 133mW$. Bitna karakteristika linearnih regulatora jeste dobro potiskivanje naponskih talasanja tj. šuma, koje na 100kHz iznosi 40dB, pa se dobija 100 puta manja amplituda šuma od 40mV, i iznosi samo 0,4mV. Napon od 15V se dalje spušta na 5V korišćenjem LF50 [13] linearnog regulatora sa malim padom napona. LF50 dalje potiskuje naponska talasanja sa oko 50dB na 100kHz, tako da se dobija 5V napon sa vrlo malo šuma i tačnošću regulacije od 5mV, koji se koristi kao naponska referenca A/D konvertora, kao i za napajanje mikrokontrolera.

3.4. SEKCIJA TEMPERATURNE KONTROLE



Slika 9: Sekcija elektronike namenjena temperaturnoj kontroli



Slika 10: Fotografija otvorene temperaturne/RF komore sa elektronskim komponentama unutar nje

Prethodno je opisan značaj temperaturne kontrole ključnih komponenti koje se nalaze u sekciji za generisanje šuma. Kontrolisanje temperature, tj. grejanje se vrši pomoću žičanog otpornika R12 (slike 9, 10) deklarisanog na 5W maksimalne disipacije snage. Napon za ovaj otpornik se uzima direktno sa filtriranog 5V napajanja sa USB konektora. Kontrola otpornika-grejača R12 se vrši pomoću mikrokontrolera koji na bazu mosfet tranzistora Q3 dovodi napon 0 ili 5V. Pri naponu od 5V na bazi ovog tranzistora koji se koristi kao prekidački element, moguće je ostvariti prekidanje maksimalne struje od 3A. Otpor tranzistora R_{DS} iznosi 0,55Ω , što u rednoj vezi sa otpornikom R12 i maksimalnim teoretskim ulaznim naponom od 5V, daje maksimalnu disipiranu snagu otpornika od $P = (\frac{5V}{0.55\Omega+22\Omega})^2 * 22\Omega = 1.08W$. Prema USB specifikaciji 1.0, dozvoljeno je da povezani uređaj troši maksimalno 2,5W, pa je upravo kompatibilnost sa najrasprostranjenijim USB standardom bila razlog za malu snagu grejača. Nije potrebno da napon sa USB veze bude egzaktne vrednosti od 5V, jer se temperaturna kontrola vrši pomoću digitalnog termometra DS18S20 [15]. DS18S20 digitalni merač temperature meri u opsegu $-10℃ - +55℃$, sa tačnošću $\pm 0{,}5℃$ i rezolucijom od 9 bita (oko 0,127℃).

Dva tranzistora iz sekcije za generisanje šuma sa pratećim otpornicima i kondenzatorom, kao i merač temperature, su fizički smešteni u zatvorenu kutiju, napravljenu od epoksi laminata impregniranog staklenim vlaknima, koji je sa obe spoljne površine prekriven slojem bakra. Zbog svoje strukture dobar je temperaturni izolator. Sve unutrašnje bakarne površine su međusobno povezane i čine radiofrekventni štit od spoljnih smetnji. Q1, Q2 i DS18S20 su zalepljeni temperaturno provodnom pastom direktno za otpornik/grejač R12, koji zbog svoje velike termalne mase ima male varijacije temperature po svojoj površini.

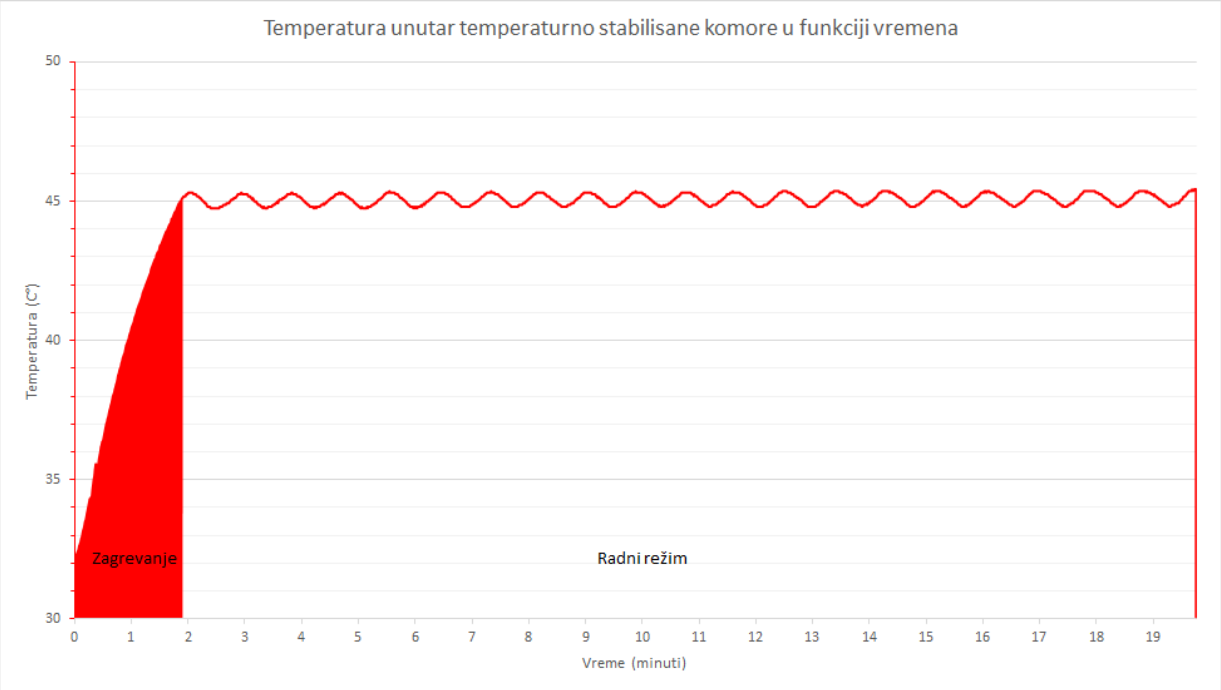
Nakon povezivanja hardverskog generatora slučajnih brojeva sa USB portom računara, uređaj će paliti i gasiti crvenu diodu koja emituje svetlo (LED), time pokazujući da je u procesu zagrevanja temperaturne komore. Kada temperatura u temperaturnoj komori dostigne $+45{,}10℃$, grejač se gasi i uređaj počinje da šalje podatke (slučajne brojeve) prema računaru. Zelenom bojom svetleće diode se indikuje da grejač trenutno nije uključen. Čim temperatura padne ispod $+45{,}00℃$, grejač se ponovo uključuje. Temperatura se drži stabilnom u opsegu $+45{,}00 \pm 0{,}4℃$, što je dovoljno tačna

kontrola temperature za potrebe rada hardverskog generatora slučajnih brojeva. Temperatura od 45℃ je odabrana empirijski, tako što je morala biti veća od sobne temperature kako bi bilo moguće zagrejati temperaturnu komoru, ali u isto vreme ne suviše visoka. U slučaju suviše visoke temperature bi se dugo čekalo na početno zagrevanje, potrošnja struje bi bila veća, i komponente bi radile na suviše visokoj temperaturi što nije preporučljivo s'obzirom da uređaj mora da radi neprekidno i pouzdano tokom dugih vremenskih perioda.

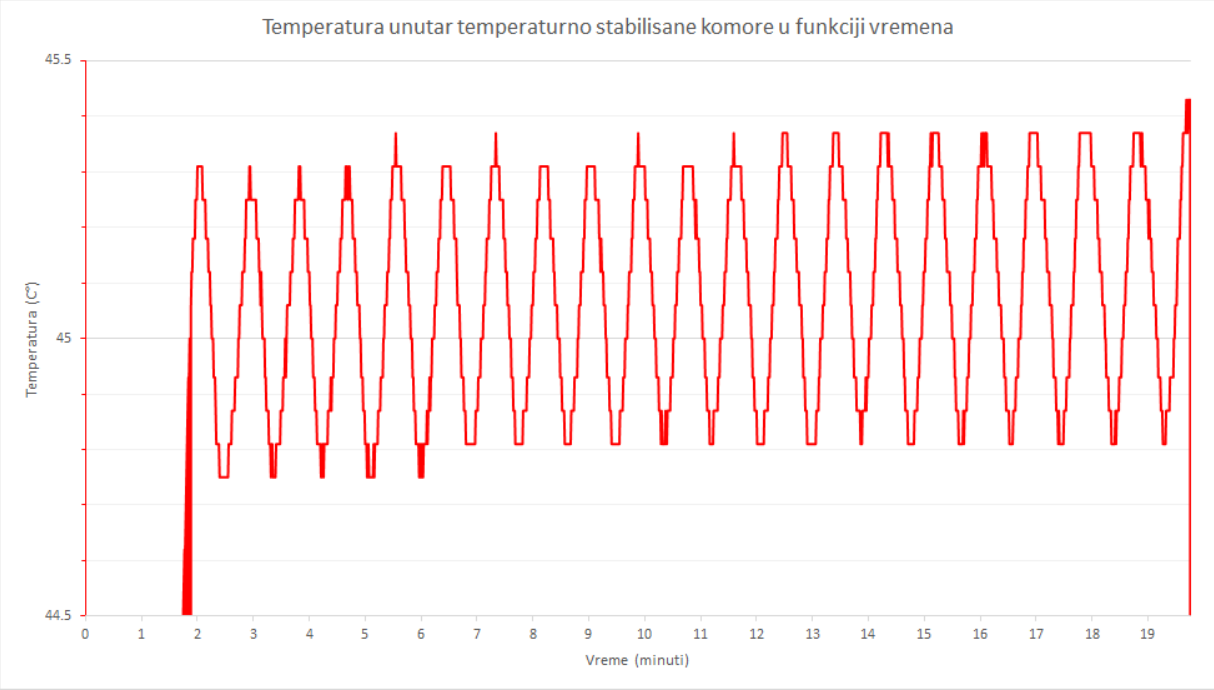
```
Main:                          ' Labela radnog rezima
   for i=0 to 1000             ' Nakon 1000 ciklusa proveri temperaturu
        ADCIN 0,AD             ' Procitaj rezultat A/D konverzije
        gosub slucajnovreme    ' Izracunaj slucajnu vrednost vremena
        pauseus vreme          ' Sacekaj slucajnu vrednost vremena 0-504us
        ADCIN 0,AD2            ' Procitaj rezultat druge A/D konverzije
   next i
   gosub citajds1820           ' Procitaj temperaturu sa senzora
   serout porta.3,T9600,[#TDS,10] ' Posalji ASCII temp ka PC-u, 9600baud, 8N1
   if tds>=(radnatemp+10) and tdsz=1 then ' Ako je pozitivna i veca od zadate+0.1C
      low grejac               ' Ugasi grejac
      gosub ledzelena          ' Ukljuci zelenu LED
   endif
   if tds<=radnatemp and tdsz=1 then     ' Ako je pozitivna i manja od zadate
      high grejac              ' Ukljuci grejac
      gosub ledcrvena          ' Ukljuci crvenu LED
   endif
goto Main
```

Kod 1: Izmenjena Main labela u programu mikrokontrolera za potrebe slanja temperature

Koristeći modifikovanu Main labelu programa (kod 1) koju izvršava mikrokontroler (ostatak programa je identičan programu koji je opisan u poglavlju o softveru unutar mikrokontrolera), dobijen je grafikon (slika 11 i 12), koji pokazuje vrednost temperature unutar temperaturne komore, od trenutka uključivanja hardverskog generatora slučajnih brojeva na USB port računara. Sa početnom sobnom temperaturom od oko 32℃ , potrebno je manje od 2 minuta da hardverski generator slučajnih brojeva dostigne radnu temperaturu i počne da šalje slučajne podatke ka računaru. Uočava se da je dalja kontrola temperature tokom prvih 20 minuta rada stabilna oko vrednosti od 45℃ , sa maksimalnim odstupanjima unutar opsega 0,6℃.



Slika 11: Temperatura unutar temperaturno stabilisane komore u funkciji vremena



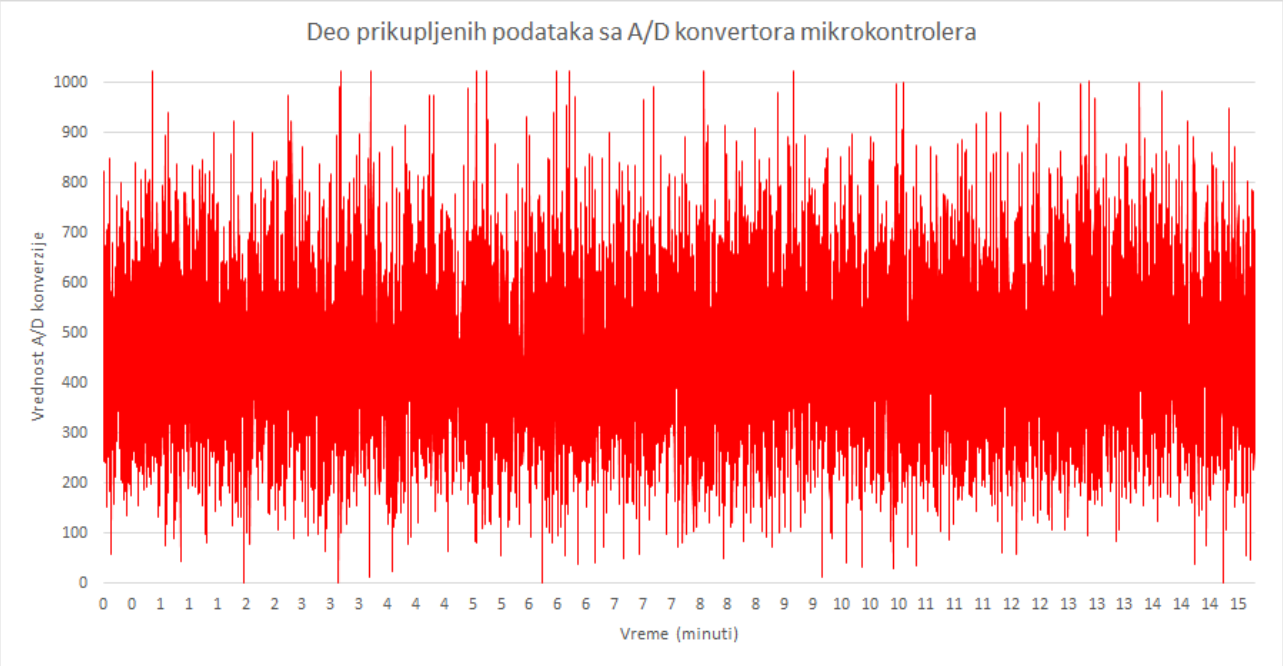
Slika 12: Temperatura unutar temperaturno stabilisane komore u funkciji vremena – uvećan prikaz

3.5. PODEŠAVANJE SREDNJE VREDNOSTI SLUČAJNO GENERISANIH BROJEVA

Neophodno podešavanje amplitude i jednosmerne komponente analognog šuma (poglavlje 3.2.), služi da što bolje prilagodi analogni šum analogno/digitalnom konverteru mikrokontrolera kako bi se iskoristila maksimalna rezolucija A/D konverzije. Međutim, takvo podešavanje je suviše grubo da bi se dobijena digitalna vrednost analogno/digitalne konverzije koristila direktno kao slučajan broj. Neophodno je ustanoviti tačnu srednju vrednost, tj. granicu preko koje će se svaka pročitana analogna vrednost proglasiti jedinicom, ili ispod granice nulom. To je moguće samo uz prikupljanje velike količine podataka sa hardverskog generatora slučajnih brojeva. Podaci sa A/D konvertera su prikupljani u vremenskom periodu od 24h, i na osnovu njih je izračunata srednja vrednost odbiraka koja iznosi 458,6. Teoretska idealna srednja vrednost bi bila 512, da je šum idealno podešen pomoću potenciometara kao što je opisano u poglavlju 3.2. Nakon upisivanja vrednosti granice u program (tj. ROM mikrokonrolera) kao celobrojne vrednosti 459, nije potrebno više menjati vrednost graničnog napona jer generator šuma radi na konstantnoj temperaturi i konstantnim naponima. U kodu 2 je prikazana samo izmenjena glavna labela programa koja šalje direktno pročitane vrednosti sa A/D konvertora, dok ostatak koda nije menjan. Slika 13 pokazuje deo prikupljenih podataka u vremenskom intevalu od 15 minuta (da bi se bolje uočio oblik šuma) koji izgledaju identično signalu sa osciloskopa (poglavlje 3.2. slika 7).

```
Main:                                ' Labela radnog rezima
   for i=0 to 1000                   ' Nakon 1000 ciklusa proveri temperaturu
        ADCIN 0,AD                   ' Procitaj rezultat A/D konverzije
        gosub slucajnovreme          ' Izracunaj slucajnu vrednost vremena
        pauseus vreme                ' Sacekaj slucajnu vrednost vremena 0-504us
        ADCIN 0,AD2                  ' Procitaj rezultat druge A/D konverzije
        Serout porta.3,T9600,[#AD,10,#AD2,10]   ' Posalji ASCII vr. A/D ka PC-u, 9600baud, 8N1
   next i
   gosub citajds1820                  ' Procitaj temperaturu sa senzora
   if tds>=(radnatemp+10) and tdsz=1 then  ' Ako je pozitivna i veca od zadate+0.1C
      low grejac                      ' Ugasi grejac
      gosub ledzelena                 ' Ukljuci zelenu LED
   endif
   if tds<=radnatemp and tdsz=1 then      ' Ako je pozitivna i manja od zadate
      high grejac                     ' Ukljuci grejac
      gosub ledcrvena                 ' Ukljuci crvenu LED
   endif
goto Main
```

Kod 2: Izmenjena Main labela u programu mikrokontrolera za slanje A/D vrednosti



Slika 13: Rezultati A/D konverzije u funkciji vremena na intervalu od 15 minuta

3.6. SEKCIJA OBRADE SIGNALA I KOMUNIKACIJE

Za obradu signala, kontrolu temperature, A/D konverziju i komunikaciju sa računarom je izabran mikrokontroler PIC16F819 proizvođača Microchip [16]. Poseduje 16 ulazno/izlaznih pinova, od kojih se pin RA0 (od maksimalno mogućih 5 analognih ulaza) koristi kao analogni ulaz prema ugrađenom 10-bitnom A/D konvertoru. Dalja obrada signala se vrši u softveru i opisana je u narednom poglavlju. Pinovi definisani kao digitalni izlaz mogu da daju maksimalnu struju od 25mA na 5V. Mikrokontroler radi brzinom od 20Mhz, definisanu spoljnim kristal oscilatorom. Pinovi RB4 i RB5 su definisani kao izlazni i pomoću njih se pali signalna LED koja je dvobojna (crvena-uključen grejač, zelena-isključen grejač). Port RB2 upravlja grejačem preko mosfet tranzistora. Port RB3 je definisan kao ulazni i preko njega se primaju i šalju podaci preko 1-žične magistrale prema temperaturnom senzoru.

Port RA3 je definisan kao izlazni i preko njega se šalju podaci ka UART-USB konvertoru FTDI232RL [17]. UART je serijski komunikacioni interfejs preko koga se podaci šalju brzinom od 9600bauda, bez provere pariteta, veličine bajta od 8 bita i jednim stop bitom. FTDI232RL se prijavljuje na računaru kao virtuelni serijski port, što olakšava komunikaciju sa programima koji će koristiti hardverski generator slučajnih brojeva.

Odabirom nekih od boljih modela mikrokontrolera, postojala bi veća brzina A/D konverzije, mogućnost postojanja hardverskog serijskog modula koji bi omogućavao veće brzine prenosa podataka, ili USB komunikacije čime bi se izbegla potreba za UART-USB konvertorom. Sam rad hardverskog generatora slučajnih brojeva bi i sa boljim mikrokontrolerom bio identičan opisanom, uz manji broj komponenti i veću brzinu generisanja slučajnih brojeva.

3.7. SOFTVER MIKROKONTROLERA

Softver mikrokontrolera je pisan u PicBasic programskom jeziku. Ovaj jezik je izabran zbog razumljivosti i jednostavnosti pisanog koda, kao i zbog velike popularnosti za PIC16 liniju mikrokontrolera koji spadaju u srednju klasu mikrokontrolera firme Microchip. Kompletan kod je prikazan ispod u kodu 3:

```
'*************************************************************
'*  Name        : HWRANDOMGEN.BAS                            *
'*  Author      : Mihailo Jovicic                            *
'*  Notice      : Copyright (c) 2015                         *
'*              : All Rights Reserved                        *
'*  Date        : 2/8/2015                                   *
'*  Version     : 2.0                                        *
'*  Notes       : Program hardverskog generatora slucajnih brojeva *
'*              :                                            *
'*************************************************************
'@ __config _CP_OFF & _CCP1_RB2 & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF & _LVP_OFF &
_BODEN_OFF & _MCLR_ON & _PWRTE_OFF & _WDT_OFF & _HS_OSC
Include "modedefs.bas"
DEFINE OSC 20                   ' Definicija oscilatora
DEFINE ADC_BITS 10              ' Definicija rezolucije A/D konvertora
DEFINE ADC_CLOCK 2              ' Definicija vremena A/D konverzije

TRISA =%00000001                ' Definicija ulaznih i izlaznih pinova porta a
TRISB =%00001000                ' Definicija ulaznih i izlaznih pinova porta b
ADCON1=%10001110                ' Definicija A/D konvertora
CCP1CON=%00000000               ' Definicija capture/compare modula

symbol dq1=portb.3              ' Promenljiva u podprogramu ds1820, definicija pina za SDO simbol
Tempds var word                 ' Promenljiva u podprogramu ds1820, privremena
korakostalo var byte            ' Promenljiva u podprogramu ds1820, iscitana sa senzora
korakpoc var byte               ' Promenljiva u podprogramu ds1820, iscitana sa senzora
t1ima var bit                   ' Promenljiva u podprogramu ds1820, da li je senzor spreman za citanje
TDS var word                    ' Temperatura sa senzora DS1820 (x100 C, 2035 je 20.35c)
TDSZ VAR bit                    ' Znak temperature sa senzora DS1820, 1 za +, 0 za -

symbol grejac=portb.2           ' Portb.2 je grejac
AD VAR WORD                     ' Trenutna vrednost napona ad konvertora
AD2 VAR WORD                    ' Druga trenutna vrednost napona ad konvertora
low portb.2                     ' Ugasi grejac
vreme var word                  ' Slucajno vreme u rasponu 0-504us
broj var word                   ' Slucajna vrednost 0-65535
radnatemp con 4500              ' Radna temperatura koja se odrzava
granica con 459          ' Srednja vrednost AD konverzije dobijena obradom prikupljenih podataka
i var byte                      ' Promenljiva za for petlju
```

```
Priprema:                                   ' Priprema za rad
   high grejac                              ' Ukljuci grejac
   gosub citajds1820                        ' Procitaj temperaturu sa senzora
   gosub Ledcrvena                          ' Ukljuci crvenu LED
   pause 500                                ' Sacekaj 1s ,ujedno trepcuci crvenu LED dok se zagreva
   gosub Ledoff
   pause 500
   if tds>=radnatemp and tdsz=1 then        ' Ukoliko je temp. veca od zadate
      low grejac                            ' Ugasi grejac
      gosub ledzelena                       ' Ukljuci zelenu LED
      goto main                             ' Pocni sa radom
   endif
goto priprema

Main:                                       ' Labela radnog rezima
   for i=0 to 1000                          ' Nakon 1000 ciklusa proveri temperaturu
         ADCIN 0,AD                         ' Procitaj rezultat A/D konverzije
         gosub slucajnovreme                ' Izracunaj slucajnu vrednost vremena
         pauseus vreme                      ' Sacekaj slucajnu vrednost vremena 0-504us
         ADCIN 0,AD2                        ' Procitaj rezultat druge A/D konverzije
' Smanjivanje korelacije izmedju odbiraka koriscenjem John Von Neumann-ovog algoritma
      if ad>=granica and AD2<granica then
         Serout porta.3,T9600,["0"]         ' Posalji ASCII 0 ka PC-u, 9600baud, 8N1
      endif
      if AD2>=granica and AD<granica then
         Serout porta.3,T9600,["1"]         ' Posalji ASCII 1 ka PC-u, 9600baud, 8N1
      endif
   next i
   gosub citajds1820                        ' Procitaj temperaturu sa senzora
   if tds>=(radnatemp+10) and tdsz=1 then   ' Ako je pozitivna i veca od zadate+0.1C
      low grejac                            ' Ugasi grejac
      gosub ledzelena                       ' Ukljuci zelenu LED
   endif
   if tds<=radnatemp and tdsz=1 then        ' Ako je pozitivna i manja od zadate
      high grejac                           ' Ukljuci grejac
      gosub ledcrvena                       ' Ukljuci crvenu LED
   endif
goto Main

Slucajnovreme:                               ' Izracunaj slucajno vreme
   random broj
   vreme=broj/130                            ' Smanji opseg na 0-504us
return

Ledcrvena:                                  ' Ukljuci crvenu LED
   high portb.4
   low portb.5
return
```

```
Ledzelena:                                  ' Ukljuci zelenu LED
   High portb.5
   low portb.4
return

Ledoff:                                     ' Ugasi obe LED
   Low portb.4
   low portb.5
return

CitajDS1820:                                ' Podlabela za citanje temperature sa senzora DS18S20
   owout dq1,1,[$CC,$44]                    ' Adresiraj sve senzore i posalji zahtev za konverziju
T1:
   owin dq1,4,[korakostalo]                 ' Proveri da li je zavrsena konverzija
   if korakostalo=0 then goto T1
T1k:
   if korakostalo=1 then                    ' Ukoliko je zavrsena konverzija
      t1ima=1
      else
      t1ima=0
   endif

if t1ima=1 then
   owout dq1,1,[$CC,$BE]                    ' Posalji zahtev za citanje podataka
   owin dq1,0,[tempds.lowbyte,tempds.highbyte,skip 4,korakostalo,korakpoc]
   pause 50
   tempds=(((tempds>>1)*100)-25)+(((korakpoc-korakostalo)*100)/korakpoc)
endif

if tempds.15=0 then                         ' Ukoliko je temperatura veca od nule
   TDS=Tempds
   TDSZ=1                                   ' Znak +
ENDIF

if tempds.15=1 then                         ' Ukoliko je temperatura manja od nule
   TDS=((Tempds^$ffff)+1)
   TDSZ=0                                   ' Znak -
ENDIF

return

end
```

Kod 3: Kompletan program mikrokontrolera unutar hardverskog generatora slučajnih brojeva

Na početku programa se konfigurišu opcije mikrokontrolera: ugašena zaštita koda, capture/compare port na pinu RB2, debug ugašen, nije dozvoljen upis u flash memoriju, zaštita eeprom memorije ugašena, ugašeno programiranje niskim naponom, ugašen reset u slučaju niskog napona, uključen master pin eksterni reset, ugašena opcija tajmera prilikom uključivanja, ugašena opcija watchdog tajmera i odabran eksterni oscilator visoke brzine.

Uvozi se datoteka "modedefs.bas" koja sadrži definicije softverske serijske komunikacije, koju je potrebno koristiti jer PIC16F819 ne sadrži hardverski UART port. Definiše se brzina upotrebljenog oscilatora na 20Mhz, 10-bitna rezolucija A/D konverzije i vreme svake A/D konverzije prema frekvenciji oscilatora koje iznosi FOSC/32. U nastavku se konfigurišu portovi mikrokontrolera kao ulazni (1) ili izlazni (0). Definiše se A/D konvertor, najniža vrednost bita je krajnja desna, koriste se 0 i 5V napajanja mikrokontrolera kao granice analognog opsega, i konfiguriše se samo pin RA0 kao analogni.

Definišu se promenljive za rad, bit promenljive imaju vrednost 0 ili 1, byte promenljive imaju celobrojnu vrednost 0-255 i word promenljive imaju celobrojnu vrednost i opsegu 0-65535. Definišu se i dve konstante, „radnatemp" koja predstavlja radnu temperaturu u ℃ pomnoženu sa 100 i iznosi 4500 tj. 45,00℃. Takođe u ROM je upisana i vrednost konstante "granica" (čije je određivanje opisano u poglavlju 3.5.), sa vrednošću 459.

Početak programa pod labelom „priprema" uključuje grejač, to signalizira treptanjem crvene LED, i meri temperaturu sa senzora temperature. Ukoliko je dostignuta radna temperatura, grejač se gasi, pali se zelena LED i ulazi se u glavnu programsku petlju Main.

Petlja Main nakon 1000 ciklusa for petlje, očitava vrednost temperature sa senzora i uključuje grejač ukoliko je temperatura manja od zadate radne temperature, ili ga isključuje ukoliko je temperatura veća od zadate radne temperature + 0,1℃. Unutar for petlje se vrši jedno uzorkovanje analogne vrednosti ulaza pomoću A/D konvertora, pravi se pauza koja ima slučajnu vremensku vrednost 0-504us, a zatim se vrši još jedno uzorkovanje pomoću A/D konvertera. Podlabela "slucajnovreme" vrši odredjivanje nasumične vrednosti promenljive tipa word koja se deli kako bi joj se smanjio opseg. Ovaj korak je uveden kao dodatna slučajnost koja sprečava da indukovani spoljni prostoperiodični signali dovedu do smanjenja nedeterminističke prirode šuma.

Analogni signal se uzorkuje dva puta zaredom kako bi se primenila izuzetno značajna tehnika za smanjivanje međusobne korelacije analognih odbiraka kao i dalje smanjenje eventualnog sistematskog pomeraja dobijenih vrednosti u odnosu na srednju vrednost odbiraka. Bez njene primene hardverski generator slučajnih brojeva daje dosta lošije statističke rezultate. Primenjeni algoritam je osmislio poznati naučnik John Von Neumann [18], kako bi pri bacanju pristrasnog novčića dobio iste rezultate kao sa bacanjem fer novčića. Primena algoritma je jednostavna, dve uzastopne identične vrednosti (00 ili 11) se odbacuju, dve različite uzastopne vrednosti daju 0 ili 1 - 10 daje 0 i 01 daje vrednost 1. Ovom tehnikom se dobijaju mnogo bolji statistički rezultati, ali se gubi na brzini rada hardverskog generatora slučajnih brojeva.

Potprogram "CitajDS1820" čita temperaturu sa senzora. Sam način komunikacije je detaljno opisan u uputstvu senzora [15]. Uprošćeno posmatrano, šalje se zahtev za konverziju i čeka se odgovor da li je konverzija završena. Ukoliko jeste, čitaju se primljeni podaci i formira se podatak o temperaturi predstavljen drugim komplementom (tempds). Iz njega se dobijaju apsolutna vrednost temperature TDS i znak TDSZ.

4. ANALIZA PODATAKA GENERISANIH HARDVERSKIM GENERATOROM SLUČAJNIH BROJEVA

4.1. SISTEMATSKO ODSTUPANJE BLOKOVA BROJEVA U ODNOSU NA SREDNJU VREDNOST

Prvi statistički test pokazuje procentualno odstupanje dobijenih blokova slučajnih brojeva u odnosu na stvarnu srednju vrednost. Podaci su sakupljani tokom 228 sati kontinualnog rada hardverskog generatora slučajnih brojeva. Ukupno je sakupljeno 72,6MB slučajnih izlaznih binarnih brojeva u ASCII formatu. Kako bi takvi podaci mogli da budu korišćeni u ostalim programskim paketima, potrebno ih je konvertovati u binarnu datoteku koju čine bajtovi, što je izvedeno pisanjem kratkog uslužnog programa u Javi navedenog ispod (kod 4). Kao izlaz, dobija se binarna datoteka veličine 9,08MB.

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
//Konvertuj izlaznu ASCII *.txt datoteku sa hardverskog generatora slucajnih brojeva
//u binarnu datoteku sa imenom izlaz
public class Konverter {
	Konverter() throws Exception{
		BufferedReader fin=new BufferedReader(new FileReader("d:\\ulaz.txt"));
		DataOutputStream out = new DataOutputStream(new FileOutputStream("d:\\izlaz.bin"));
		int c=0,b;
		int g[]=new int[8];                                  //Niz od 8 int promenljivih
		while(c!=-1){                                        //Ukoliko nije kraj datoteke
			for(int i=0;i<8;i++){                        //Procitaj 1 bajt
				c=fin.read();                        //Procitaj 1 bit
				if(c==-1) break;                     //Ukoliko je kraj datoteke, odustani
				char znak = (char) c;                //Cast
				if(znak=='1') g[i]=1;                //Ukoliko je znak 1, upisi bit 1
				if(znak=='0') g[i]=0;                //Ukoliko je znak 0, upisi bit 0
			}
			b=0;                                 //Inicijalizuj i upisi jedan bajt u promenljivu int
			b=g[0]<<7 | g[1]<<6 | g[2]<<5 | g[3]<<4 | g[4]<<3 | g[5]<<2 | g[6]<<1 | g[7];
			out.write(b);                        //Odstampaj jedan bajt u binarnu datoteku
		}
		fin.close();                                         //Zatvori datoteke
		out.close();
	}
	public static void main(String[] args) {
		try {new Konverter();} catch (Exception e) {
			e.printStackTrace();}}
}
```

Kod 4: Program za konverziju ASCII binarnih vrednosti u binarne datoteke popunjene bajtovima

Ovaj test pokazuje da li hardverski generator slučajnih brojeva ima loše podešene analogne i digitalne srednje vrednosti (opisane u poglavljima 3.2. i 3.5.) , kao i da li se tokom duge upotrebe uređaja u radu (tokom skoro 10 dana neprekidnog rada) javljaju sistematske greške koji bi se manifestovale tendencijom porasta ili smanjivanja prosečnih vrednosti blokova brojeva u odnosu na srednju vrednost. Kako bi se hardverski generator slučajnih brojeva uporedio sa najčešće korišćenim pseudoslučajnim generatorom brojeva u Javi, napisan je kratak program koji generiše binarne fajlove zadate veličine koji su popunjeni bajtovima sa vrednostima u rasponu 0-255 (kod 5). Zadata generisana veličina datoteke bi trebalo da uvek bude identična veličini analiziranih datoteka sa hardverskog generatora slučajnih brojeva.

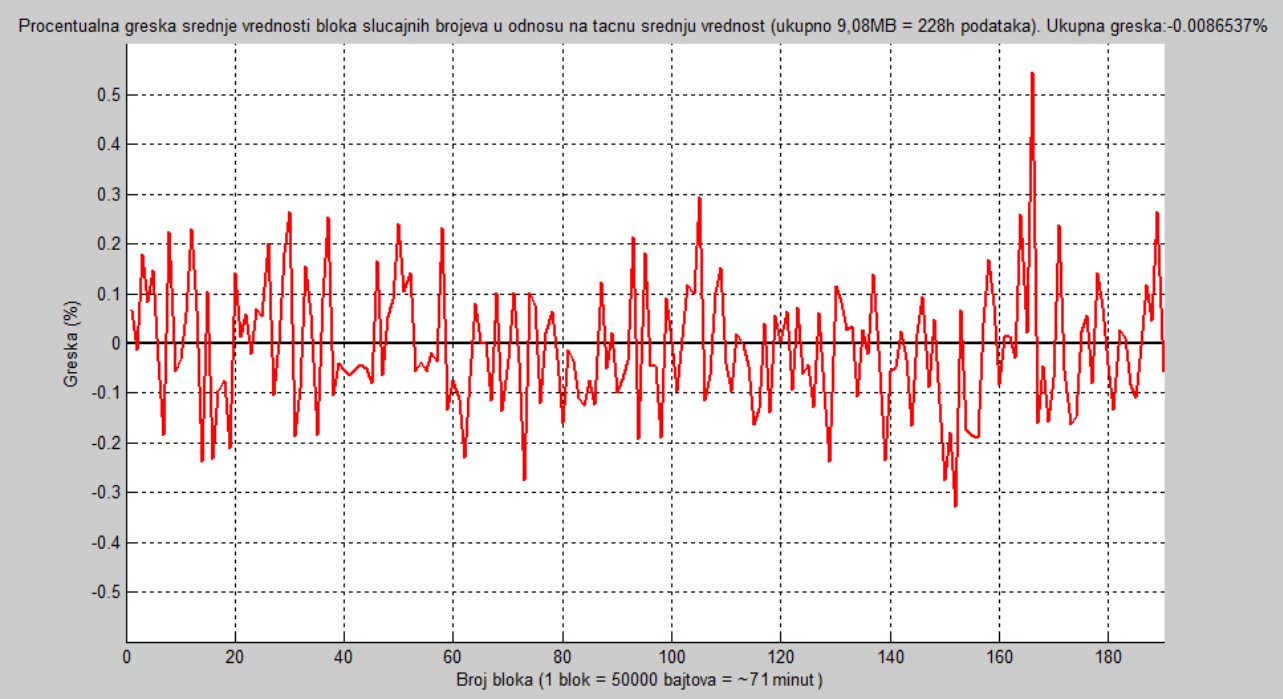
```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;

//Generator slucajnih brojeva i upis u binarnu datoteku koriscenjem Math.random() Java metode

public class Randomgen2 {
	Randomgen2() throws Exception{
		DataOutputStream out = new DataOutputStream(new FileOutputStream("d:\\izlazjava.bin"));
		BufferedReader tin = new BufferedReader(new InputStreamReader(System.in));

		System.out.println("Upisite velicinu random fajla za generisanje u MB:?");
		String ulaz=tin.readLine();
		double velicina=Double.parseDouble(ulaz);		//Velicina fajla u MB
		velicina=velicina*1024*1024;				//Velicina fajla u bajtima

		for (int i=0;i<velicina;i++){				//Upis bajt po bajt
			double r=Math.random();				//Izracunaj slucajni broj 0-1
			int k=(int)(r*256+1)-1;				//Slucajni broj 0-255
			out.write(k);					//Upisi u datoteku
		}
		out.close();						//Zatvori datoteku
	}

	public static void main(String[] args) {
		try {new Randomgen2();} catch (Exception e) {e.printStackTrace();}
}
```

Kod 5: Program za generisanje binarnih datoteka popunjenih pseudoslučajno dobijenim bajtovima

Za veličinu bloka je odabrana dužina od 50000 bajtova, i za svaki blok je potrebno oko 71 minut generisanja brojeva na hardverskom generatoru slučajnih brojeva. U kodu ispod (kod 6) je naveden program napisan u Matlab programskom paketu koji čita prethodno napravljene binarne datoteke i iscrtava procentualnu grešku svakog bloka brojeva u odnosu na stvarnu srednju vrednost (127,5 u slučaju jednog bajta).

```
clear all;
close all;
fajl= fopen('izlaz2.bin');                          %Otvori datoteku
A=fread(fajl);                                      %Procitaj bajtove u kolona matricu A
N=length(A)                                         %Duzina matrice
Duz_blok=50000                                      %Definisana duzina bloka
Br_blokova=floor(length(A)/Duz_blok)                %Ukupan broj blokova
sviblokovi=zeros(Br_blokova,1);                     %Postavi vrednost na 0
for i=[1:Br_blokova]                                %Za svaki blok izracunaj gresku
   blok=A(Duz_blok*(i-1)+1:Duz_blok*i);             %Jedan blok
   sviblokovi(i)=((sum(blok)/Duz_blok)-127.5)*100/255;  %Procentualna greska
end
Ukup_sr=mean(sviblokovi)                            %Ukupna procentualna greska svih blokova
line([0,200],[0,0],'Linewidth',2,'Color','k');      %Iscrtaj liniju na 0%
hold on;
plot(sviblokovi,'r','Linewidth',2);                 %Iscrtaj grafikon procentualne greske
axis([0,190,-0.6,0.6]);                             %Granice grafikona
grid;
xlabel ('Broj bloka (1 blok = 50000 bitova = ~71 minut)');
ylabel ('Greska (%)');
title (['Procentualna greska srednje vrednosti bloka slucajnih brojeva u odnosu na tacnu srednju
vrednost (ukupno 9,08MB = 228h podataka). Ukupna greska:' num2str(Ukup_sr) '%']);
```

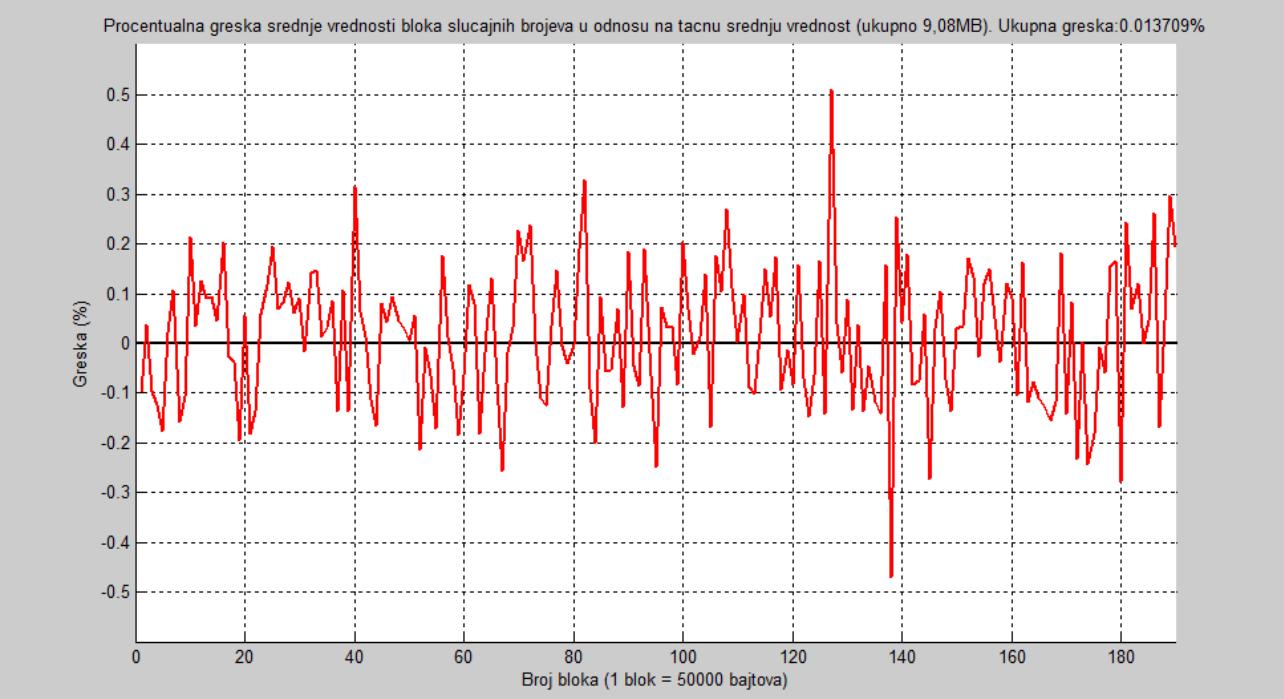
Kod 6: Program napisan u programu Matlab koji računa i iscrtava procentualne greške blokova brojeva



Slika 14: Grafikon procentualne greške blokova brojeva generisanih hardverskim generatorom brojeva u odnosu na srednju vrednost. Ukupna greška -0,0086537%

Sa slike 14 se uočava mala procentualna greška blokova u odnosu na srednju vrednost bajtova, koja je još manja ukoliko se uzimaju duži blokovi brojeva. Ukupna procentualna greška je vrlo mala sa vrednošću od -0,0086537%, što govori da je srednja vrednost generisanih slučajnih brojeva dobro podešena u poglavljima 3.2. i bitnije 3.5. Većom rezolucijom A/D konverzije bi mogla da se dobije još manja apsolutna greška. Posle vrlo dugog vremenskog perioda rada od 228 sati, ne uočava se tendencija porasta ili smanjivanja vrednosti greške što govori da je zbog korišćenja temperaturne stabilizacije uređaj vrlo precizan.

Program je ponovljen za slučajne brojeve generisane pseudoslučajnim algoritmom unutar programskog jezika Java pomoću programa iz koda 5, i dobijen je grafikon na slici 15.



Slika 15: Grafikon procentualne greške blokova brojeva generisanih pseudoslučajnim generatorom slučajnih brojeva u odnosu na srednju vrednost. Ukupna greška 0,013709%

Sa dobijenog grafikona i ukupne procentualne vrednosti greške se uočava da je pseudoslučajni generator koji se najčešće koristi u programskom jeziku Java lošijih karakteristika u odnosu na konstruisani hardverski generator slučajnih brojeva u pogledu procentualne greške u odnosu na srednju vrednost. Pri tome se ne pravi razlika između determinističke i nedeterminističke prirode dobijanja slučajnih brojeva već samo u statistici izlaznih podataka. Sama deterministička priroda pseudoslučajnog generatora ga čini nepreporučljivim za korišćenje u oblasti kriptografije, ali je za manje kritične primene, bar po pitanju greške srednje vrednosti, upotrebljiv.

Važno je skrenuti pažnju da je opšte preporučeni način od strane programera za dobijanje pseudoslučajnog opsega brojeva u programskom jeziku Java u rasponu 0-255, prema formuli “int k=(int)(Math.random()*255+1)” pogrešan. Takav način dobijanja pseudoslučajnih sekvenci brojeva rezultuje greškom od 0,20538% (ista veličina datoteke i isti program prema kodu 6), što dosta pogoršava kvalitet dobijenih sekvenci. Korišćena je formula “int k=(int)(Math.random()*256+1)-1” o čemu će biti reči u narednom poglavlju.

4.2. FUNKCIJA RASPODELE GENERISANIH SLUČAJNIH BROJEVA

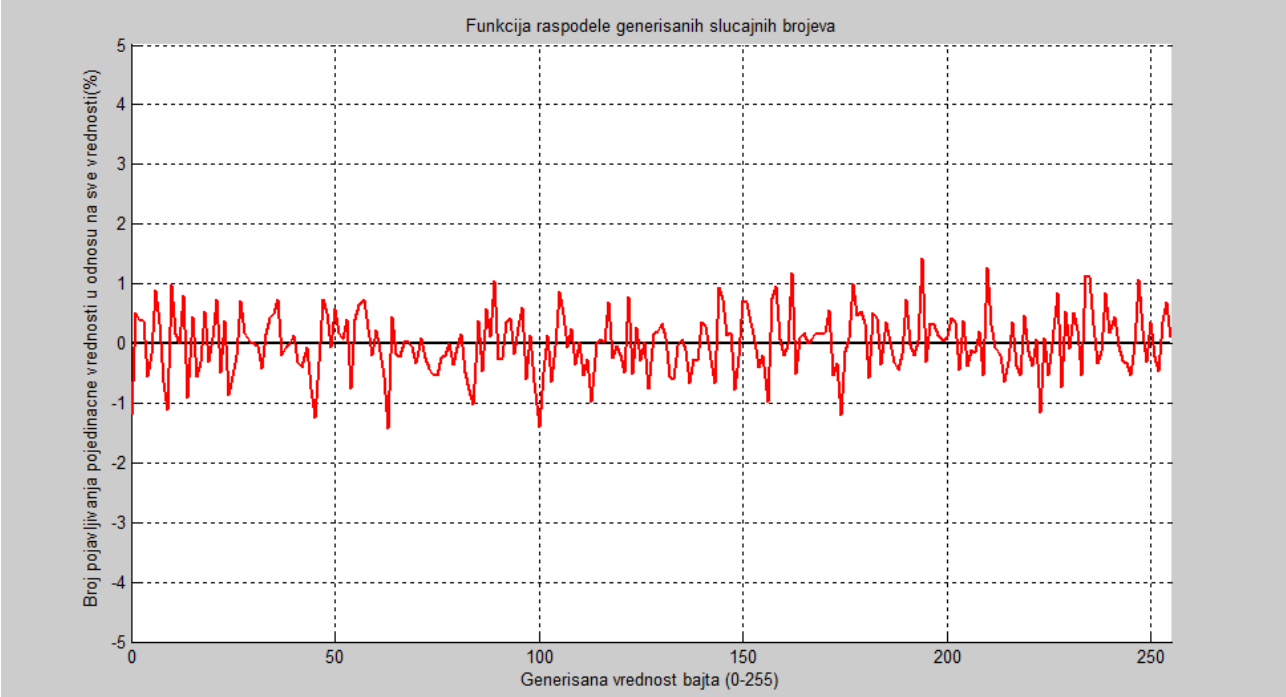
Generatori slučajnih brojeva, bilo da su hardverskog ili pseudoslučajnog tipa, najčešće imaju uniformnu raspodelu izlaznih vrednosti. Pri uniformnoj raspodeli se očekuje da svaka vrednost na izlazu slučajnog generatora ima istu verovatnoću pojavljivanja. Kako bi se proverila ova karakteristika hardverskog generatora slučajnih brojeva i uporedila sa pseudoslučajnim generatorom u programskom jeziku Java, iskorišćeni su sakupljeni i generisani podaci veličine 9,08MB iz prethodnog poglavlja 4.1. Sledeći kod u paketu Matlab računa ukupan broj pojavljivanja svake vrednosti i iscrtava grafikon (kod 7).

```
clear all;
close all;
fajl= fopen('izlaz2.bin');                  %Otvori datoteku
A=fread(fajl);                              %Procitaj bajtove u kolona matricu A
N=length(A)                                 %Duzina matrice
s=zeros(256,1);                             %Postavi vrednosti na 0
for i=[1:N]                                 %Procitaj svaki bajt
        s(A(i)+1)=s(A(i)+1)+1;              %Saberi sa prethodnim pojavljivanjima
end
r=s                                         %Ispisi vrednosti na izlazu
sr=mean(s)                                  %Izracunaj srednju vrednost=100%
line([0,255],[0,0],'Linewidth',2,'Color','k');  %Iscrtaj liniju na 0%
hold on;
plot(0:size(s)-1,s*100/sr-100,'r','Linewidth',2); %Iscrtaj grafikon broja pojavljivanja poj. vr. 0-255
axis([0,255,-5,5]);                         %Granice grafikona +-5%
grid;
xlabel ('Generisana vrednost bajta (0-255)');
ylabel ('Broj pojavljivanja pojedinacne vrednosti u odnosu na sve vrednosti(%)');
title (['Funkcija raspodele generisanih slucajnih brojeva']);
```

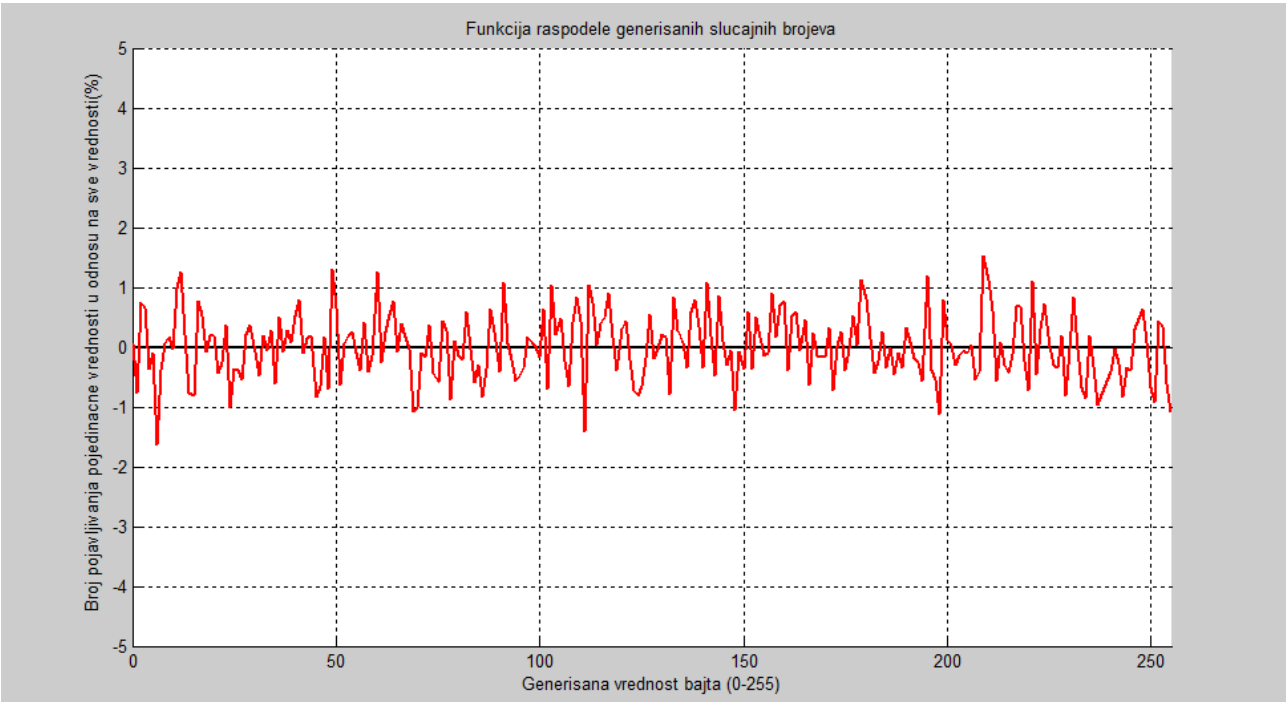
Kod 7: Program napisan u Matlabu koji računa i iscrtava funkciju raspodele

Na slici 16 je prikazan dobijen grafikon funkcije raspodele slučajno generisanih brojeva sa hardverskog generatora slučajnih brojeva. Na slici 17 je prikazan grafikon dobijen iz datoteke koja je generisana Java random funkcijom. Uočava se da oba generatora slučajnih brojeva imaju balansirane vrednosti izlaznih slučajnih brojeva, tj. da grafikon nema nagib koji bi ukazao na nepravilnosti u uniformnoj raspodeli. Maskimalna procentualna razlika u učešću određene vrednosti je oko 1,5% za oba generatora slučajnih brojeva. Ovo je samo gruba grafička predstava koja dobro ilustruje funkciju raspodele, dok će detaljniji testovi biti obrađeni u narednim poglavljima.

Značajno je još jednom spomenuti funkciju random() programskog jezika Java i opštu preporuku za njeno korišćenje pri generisanju opsega brojeva. Iako je random funkcija definisana na intervalu [0,1) , korišćenjem vrednosti 0 se dobija neuniformna raspodela. Tako generisanje brojeva u opsegu 0-255 prema formuli int “k=(int)(Math.random()*255+1)” daje neuniformnu raspodelu gde se broj 0 pojavljuje 18630 puta u odnosu na sve ostale brojeve koji se pojavljuju oko 37200 puta. Zbog toga je dobijena i velika greška srednje vrednosti, opisana u prošlom poglavlju. Potrebno je ne koristiti krajeve opsega 0 i 1, već pomeriti ceo opseg za 1, što se postiže formulom “int k=(int)(Math.random()*256+1)-1” . Tek tada se dobija uniformna funkcija raspodele pseudoslučajnih brojeva (slika 17).



Slika 16: Grafikon procentualne raspodele pojedinačnih generisanih vrednosti u odnosu na sve vrednosti , dobijenih pomoću hardverskog generatora slučajnih brojeva (uzorak od 228 sati, 9,08MB)



Slika 17: Grafikon procentualne raspodele pojedinačnih generisanih vrednosti u odnosu na sve vrednosti dobijenih pomoću Java pseudoslučajnog generatora (uzorak od 9,08MB)

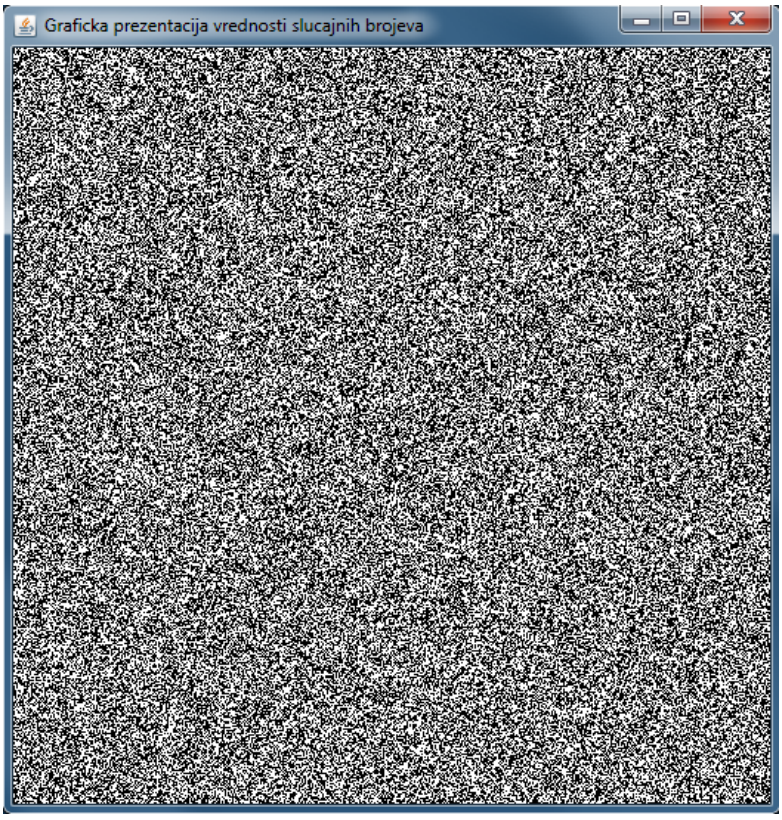
4.3. GRAFIČKI PRIKAZ GENERISANIH SLUČAJNIH BROJEVA

Grafičkim prikazom generisanih slučajnih brojeva, kao crnih ili belih tačaka (1 ili 0, respektivno) na bitmapi, se odokativno utvrđuje da li postoje neki šabloni prilikom generisanja slučajnih brojeva koji bi doveli do pojavljivanja pravilnih šara na slici. Hardverski generatori slučajnih brojeva ne bi smeli da pokazuju nikakve šablone na slici, dok pseudoslučajni generatori, koji rade na bazi loših algoritama, mogu da prikažu vrlo lako uočljive šablone. Za potrebe ovog testa, napisan je program u programskom jeziku Java, priložen u kodu 8.

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.FileReader;

public class Iscrtavanje extends Frame{
	private static final long serialVersionUID = 1L;
	private Platno platno=new Platno();                                  //Objekat klase Platno
	BufferedReader fin=new BufferedReader(new FileReader("d:\\ulaz.txt"));      //Citanje ulazne datoteke
	int[] niz = new int[262144];                                         //512x512 potrebnih bitova

	Iscrtavanje() throws Exception{                                      //Konstruktor klase
		super("Graficka prezentacija vrednosti slucajnih brojeva");      //Naslov
		setSize(528,550);                                                //Rezolucija celog prozora
		add(platno,"Center");                                            //Centralno postavljeno
		setVisible(true);                                                //Vidljivo
		addWindowListener(new WindowAdapter() {                          //Dugme X za gasenje
			public void windowClosing(WindowEvent d){dispose();}
		});
		Procitajniz();                                                   //Procitaj fajl u niz
	}
private void Procitajniz() throws Exception{                             //Metoda za citanje fajla
		for(int i=0;i<262144;i++){                                       //Procitaj dovoljno bitova
			int c=fin.read();                                            //Procitaj 1 bit
			if(c==-1) break;                                         //Ukoliko je kraj datoteke, odustani
			char znak = (char) c;                                        //Cast
			if(znak=='1') niz[i]=1;                                      //Ukoliko je znak 1, upisi bit 1
			if(znak=='0') niz[i]=0;                                      //Ukoliko je znak 0, upisi bit 0
}
		fin.close();                                                     //Zatvori fajl
	}
```

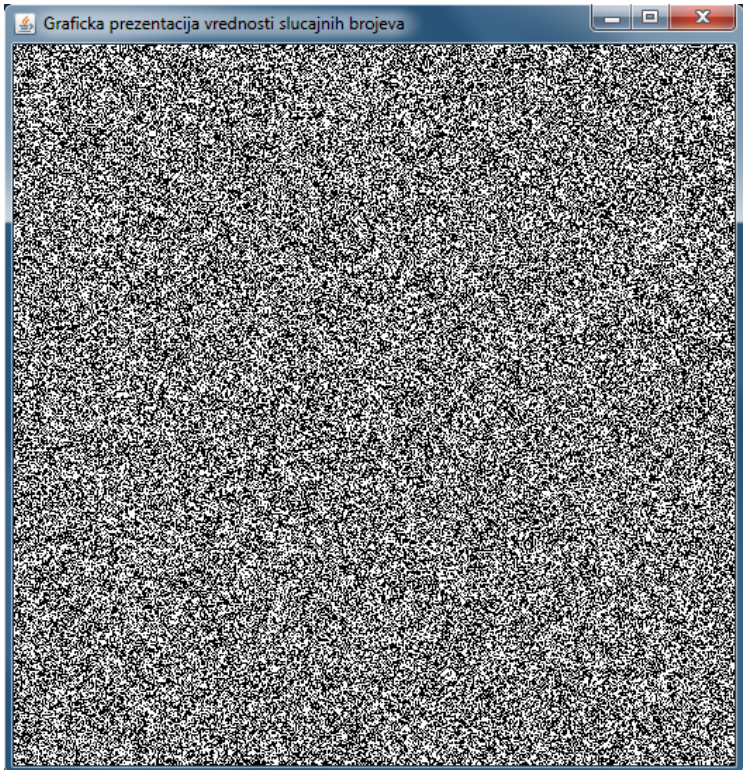
```
private class Platno extends Canvas{                                    //Iscrtavanje
        private static final long serialVersionUID = 1L;
        public void paint(Graphics g){
                int a=0;                                                //x koordinata pocetka
                int b=(getHeight()-1);                                  //y koordinata je obrnuta
                g.translate(a, b);                                      //Namesti koord. pocetak u a,b
                g.setColor(Color.BLACK);                                //Crtaj crnom bojom
                int index=0;                                            //Indeks piksela
                for(int x=0;x<512;x++){                                 //Od 0-512 piksela po x osi
                        for(int y=0;y<512;y++){                         //Od 0-512 piksela po y osi
                                if (niz[index]==1) g.drawLine(x, -y, x, -y);//Crtaj piksel ako je bit=1
                                ++index;                                //Uvecaj indeks
                        }
                }
        }

        public static void main(String[] args) {
                try {
                        new Iscrtavanje();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Kod 8: Program napisan u Javi koji iscrtava bitmapu sastavljenu od pojedinačnih piksela koji predstavljaju bite generisanih slučajnih vrednosti sa generatora slučajnih brojeva



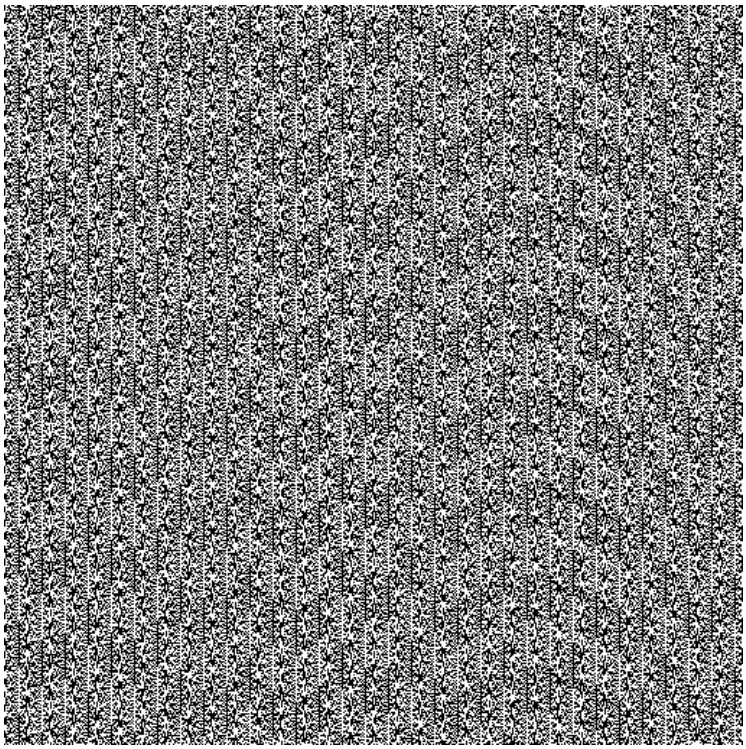
Slika 18: Grafički prikaz slučajnih brojeva dobijenih hardverskim generatorom

Na slici 18 se ne uočavaju nikakve pravilnosti, niti šare koje se ponavljaju, što je očekivano jer je sam generator slučajnih brojeva baziran na nedeterminističkom izvoru šuma.



Izvršavanje istog programa je ponovljeno i za slučajne brojeve generisane pomoću ugrađene Java random() funkcije i dobijena je slika (slika 19), koja takođe ne pokazuje pravilnosti. Za pseudoslučajne algoritme to je prvi dobar pokazatelj ispravnosti algoritma. Srećom malo je pseudoslučajnih generatora brojeva koji su u širokoj upotrebi u današnje vreme i koji koriste loš algoritam koji bi doveo do lako vidljive pravilnosti u slici. Izuzetak je algoritam rand() programskog jezika PHP, koji pod Windows operativnim sistemom daje izuzetno loše rezultate (slika 20) [19].

Slika 19: Grafički prikaz slučajnih brojeva dobijenih pseudoslučajnim generatorom u jeziku Java



Slika 20: Grafički prikaz slučajnih brojeva dobijenih pseudoslučajnim generatorom u jeziku PHP

4.4. IZRAČUNAVANJE BROJA π MONTE KARLO METODOM

Pod Monte Karlo metodom se podrazumeva svaki metod rešavanja problema pomoću korišćenja velikog broja slučajno generisanih brojeva za rešavanje matematičkih problema. Metoda je pogodna za dobijanje numeričkih rešenja za probleme koji su suviše komplikovani da bi se rešili analitički. Od ukupno generisanih slučajnih brojeva, posmatra se njihov deo koji zadovoljava određene uslove i na osnovu njega se dobijaju rešenja. Monte Karlo metoda je izuzetno pogodna za proveru rada hardverskog ili pseudoslučajnog generatora slučajnih brojeva, tačnije njihove uniformne raspodele.

Za korišćenje Monte Karlo metode je izabrano često korišćeno izračunavanje vrednosti broja π [20]. Metoda izračunavanja vrednosti broja π je lako razumljiva i pogodna je za grafičko prikazivanje. Iscrtava se kvadrat poznatih dimenzija i u njemu četvrtina kruga sa poluprečnikom dužine jednakoj jednoj strani kvadrata. Pošto se zna da je površina četvrtine kruga $r^2\pi/4$, a odgovarajućeg kvadrata r^2, vrednost broja π se dobija kao broj tačaka koje se nalaze u četvrtini kruga, pomnožene sa brojem 4 i podeljene sa ukupnim brojem tačaka u kvadratu ($\pi = \frac{r^2\pi}{4} * 4/r^2$). Za izračunavanje broja π Monte Karlo metodom i grafičku predstavu rešenja, napisan je program u programskom jeziku Java priložen u kodu 9.

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.FileReader;

public class MonteCarloPi2 extends Frame{
	private static final long serialVersionUID = 1L;
	private Platno platno=new Platno();                                    //Objekat klase Platno
	BufferedReader fin=new BufferedReader(new FileReader("d:\\ulaz.txt"));//Citanje ulazne datoteke
	int ukupnotacaka=300000;                                               //Ukupno tacaka=ukupnotacaka/2
	int[] niz = new int[ukupnotacaka];                                     //niz x i y koordinata
	int g[]=new int[24];                                                   //Niz od 24 int promenljivih
	int b=0;                                                               //Jedan 24 bitni broj
	int c=0;                                                               //Procitan bit
	int tacakaukrugu;

	MonteCarloPi2() throws Exception{                                      //Konstruktor klase
		super("Racunanje vrednosti broja Pi Monte Carlo metodom");//Naslov
		setSize(512+16,512+38);                                            //Rezolucija 512x512
		add(platno,"Center");                                              //Centralno postavljeno
		setVisible(true);                                                  //Vidljivo
		addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent d){dispose();}});
		Procitajniz();                                                     //Procitaj tacke u niz
		Izracunajpi();
		repaint();
	}
```

```
private void Procitajniz() throws Exception{                    //Metoda za citanje fajla
                for(int k=0;k<ukupnotacaka;k++){                //Procitaj
                        b=0;
                        for(int i=0;i<24;i++){                  //Procitaj 3 bajta
                                c=fin.read();                   //Procitaj 1 bit
                                if(c==-1) break;                //Ukoliko je kraj datoteke, odustani
                                char znak = (char) c;           //Cast
                                if(znak=='1') g[i]=1;           //Ukoliko je znak 1, upisi bit 1
                                if(znak=='0') g[i]=0;           //Ukoliko je znak 0, upisi bit 0
                        }
                        b=0;
                        for(int y=0;y<24;y++) b=b|(g[y]<<(23-y));   //Napravi 24 bitni integer od 24 bita
                        niz[k]=b;
                }
                fin.close();                                    //Zatvori fajl
        }

        void Izracunajpi(){
                for(int index=0;index<ukupnotacaka;index+=2){   //Tacka po tacka (x i y koordinata)
                        if ((Math.pow(niz[index],2)+Math.pow(niz[index+1],2))<Math.pow(2,48)){//Ako je u
krugu
                                ++tacakaukrugu;                 //Broj tacaka u krugu +1
                        }
                }
                        double pi=0;                            //Dobijena vrednost Pi
                        pi=(((double)tacakaukrugu)*4)/(ukupnotacaka/2);     //Izracunaj pi
                        System.out.println("Uzorak velicine:"+ukupnotacaka*3/1024+"kB");
                        System.out.println("Dobijena vrednost broja pi:"+pi);//Ispisi Pi
                        System.out.println("Stvarna vrednost broja pi :"+Math.PI);
                        System.out.println("Procentualna greska:"+(100-((100*pi)/Math.PI))+"%");
                                                //Procentualna greska u odnosu na stvarnu vr.
        }
private class Platno extends Canvas{                            //Iscrtavanje
                private static final long serialVersionUID = 1L;
                public void paint(Graphics g){
                        int a=0;                                //x koordinata pocetka
                        int b=(getHeight()-1);                  //y koordinata je obrnuta
                        g.translate(a, b);                      //Namesti koord. pocetak u a,b
                        g.setColor(Color.BLACK);                //Crtaj crnom bojom
                        for(int index=0;index<ukupnotacaka;index+=2){       //Tacka po tacka (x i y koordinata)
                                if ((Math.pow(niz[index],2)+Math.pow(niz[index+1],2))<Math.pow(2,48)){
                                //Ako je u krugu
                                        g.setColor(Color.RED);  //Crtaj crvenom bojom
                                }else{
                                        g.setColor(Color.BLUE); //Crtaj plavom bojom
                                        }
                                g.drawLine(niz[index]/32768, -niz[index+1]/32768, niz[index]/32768, -
niz[index+1]/32768);                                            //Crtaj 1 piksel
                        }
                        g.setColor(Color.GREEN);                //Crtaj zelenom bojom
                        g.drawOval(-512, -512, 1024, 1024);     //Crtaj granicu kruga
                }
        }
```

```
public static void main(String[] args) {
		try {
			new MonteCarloPi2();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Kod 9: Program napisan u Javi koji izračunava vrednost broja PI korišćenjem Monte Karlo metode i grafički prezentuje način rada

Za izračunavanje broja π je korišćen deo uzorka prikupljenih podataka, kako bi se bolje grafički prezentovao način rada. Za Monte Karlo metodu važi da se uzimanjem većeg ulaznog uzorka slučajnih brojeva dobija tačnije rešenje problema. Program čita tekstualnu datoteku sa binarnim zapisom, dobijenu direktno sa izlaza hardverskog generatora slučajnih brojeva, dok je druga datoteka popunjena pseudoslučajnim binarnim brojevima korišćenjem Java random() funkcije (isti uzorak kao u poglavlju 4.1.). Promenljiva „ukupnotacaka" određuje koliko će biti pročitano koordinata iz ulazne datoteke. Svaka koordinata je napravljena od 24 slučajno generisana bita, što daje opseg 0-2^{24} tj. 0-16777216. U nizu "niz" su smeštene sve koordinate i to redom $x_1, y_{1,} x_2, y_{2,}$ ... U konstruktoru klase se podešavaju parametri prozora i rezolucija grafičkog prikaza. Metoda „Procitajniz()" čita prethodno definisan broj tačaka sa ulaza i pretvara ih u niz 24-bitnih koordinata. Metoda „Izracunajpi()" postavlja uslov koji deo tačaka treba da zadovolji, da kvadrirane vrednosti apcise i ordinate određene tačke ne smeju u zbiru da prekorače vrednost 2^{48} koja predstavlja kvadriranu vrednost jedne stranice kvadrata. Ukoliko određena tačka zadovoljava ovaj uslov, promenljiva "tacakaukrugu" će se uvećati za jedan, i sa njom se opisuje broj svih tačaka koje se nalaze u površini četvrtine kruga. Vrednost π se izračunava prema vrednosti ukupnog broja tačaka u kvadratu i broja tačaka u krugu, i poredi se sa stvarnom vrednošću broja π. Izračunava se i procentualna greška koja daje lakši broj za poređenje kvaliteta rezultata. Privatna klasa „Platno" iscrtava sve tačke koje se nalaze u površini koju opisuje četvrtina kruga, dok su ostale tačke iscrtane plavom bojom. Vrednost apcise i ordinate svake tačke se deli sa brojem 32768, kako bi se dobio opseg 0-512 koji je moguće iscrtati na rezoluciji od 512x512 piksela. Granica između dve oblasti, koja predstavlja četvrtinu kruga, crta se zelenom bojom.

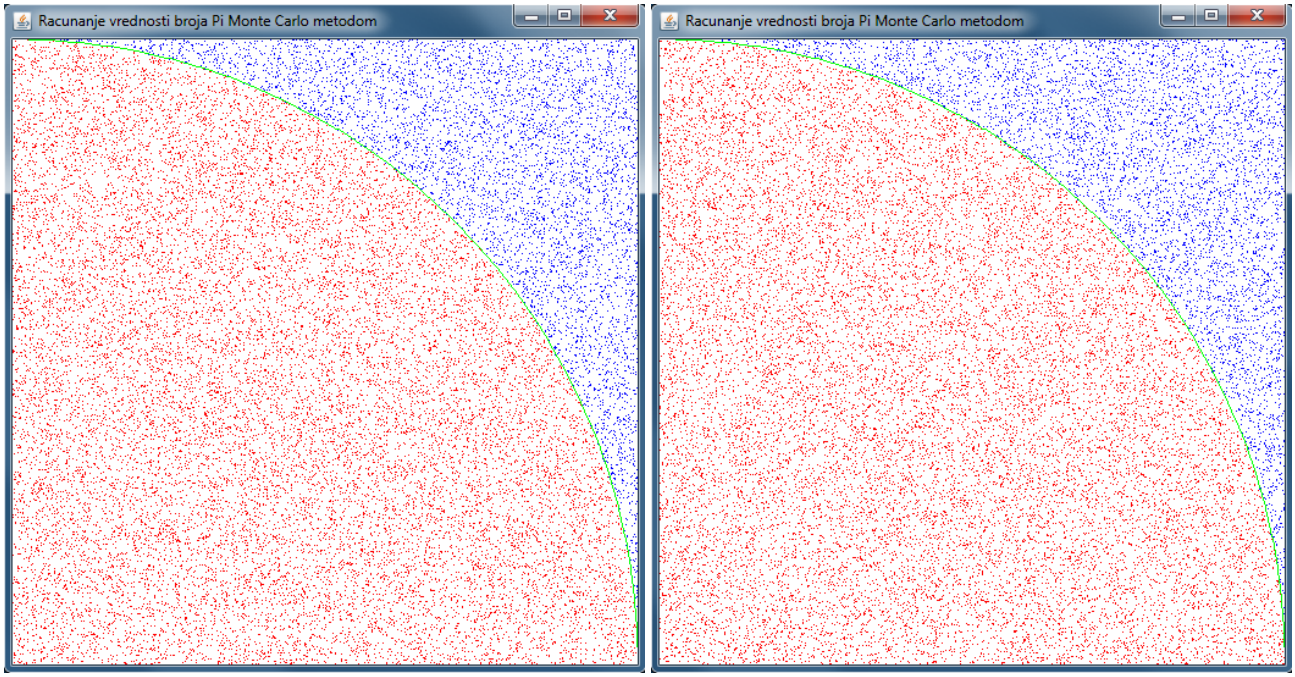
Kako bi se prikazala razlika između dva generatora slučajnih brojeva, hardverskog i pseudoslučajnog, i između malog i velikog broja tačaka, na sledećim slikama su prikazani rezultati 4 testa. Svi testovi koriste iste prethodno korišćene uzorke iz oblasti 4.1. .

Prvi test je koristio uzorak od 50000 koordinata, tj. 25000 tačaka, što predstavlja 146kB prikupljenih slučajno generisanih brojeva. Rezultati hardverskog generatora slučajnih brojeva su:

```
Uzorak velicine:146kB
Dobijena vrednost broja pi:3.13776
Stvarna vrednost broja pi :3.141592653589793
Procentualna greska:0.12199715279488998%
```

Rezultati Java random() pseudoslučajnog generatora su:

```
Uzorak velicine:146kB
Dobijena vrednost broja pi:3.12576
Stvarna vrednost broja pi :3.141592653589793
Procentualna greska:0.5039690162154358%
```



Slika 21: Grafički prikaz izračunavanja vrednosti broja PI Monte Karlo metodom za 25000 tačaka. Sa leve strane je prikazan rezultat raspodele dobijen korišćenjem slučajnih brojeva sa hardverskog generatora, a sa desne sa pseudogeneratora

Gledajući grafički prikaz raspodele tačaka (slika 21), deluje da oba generatora slučajnih brojeva daju uniformnu raspodelu. Međutim gledajući numeričke rezultate Monte Karlo metode i dobijenu vrednost broja π, uočava se prednost hardverskog generatora slučajnih brojeva, sa 4 puta manjim procentualnim odstupanjem od stvarne vrednosti broja π u odnosu na pseudoslučajni generator brojeva ugrađen u Javu. Ipak oba rešenja su relativno loša, jer je već druga decimala broja π pogrešno izračunata.

Za tačnije izračunavanje je potreban veći broj slučajnih vrednosti. Slede rezultati dobijeni računanjem sa 300000 koordinata, tj. 150000 tačaka iz istih, prethodno korišćenih, datoteka sa generisanim slučajnim brojevima. Ukupno je korišćeno 878kB podataka za ovaj test.

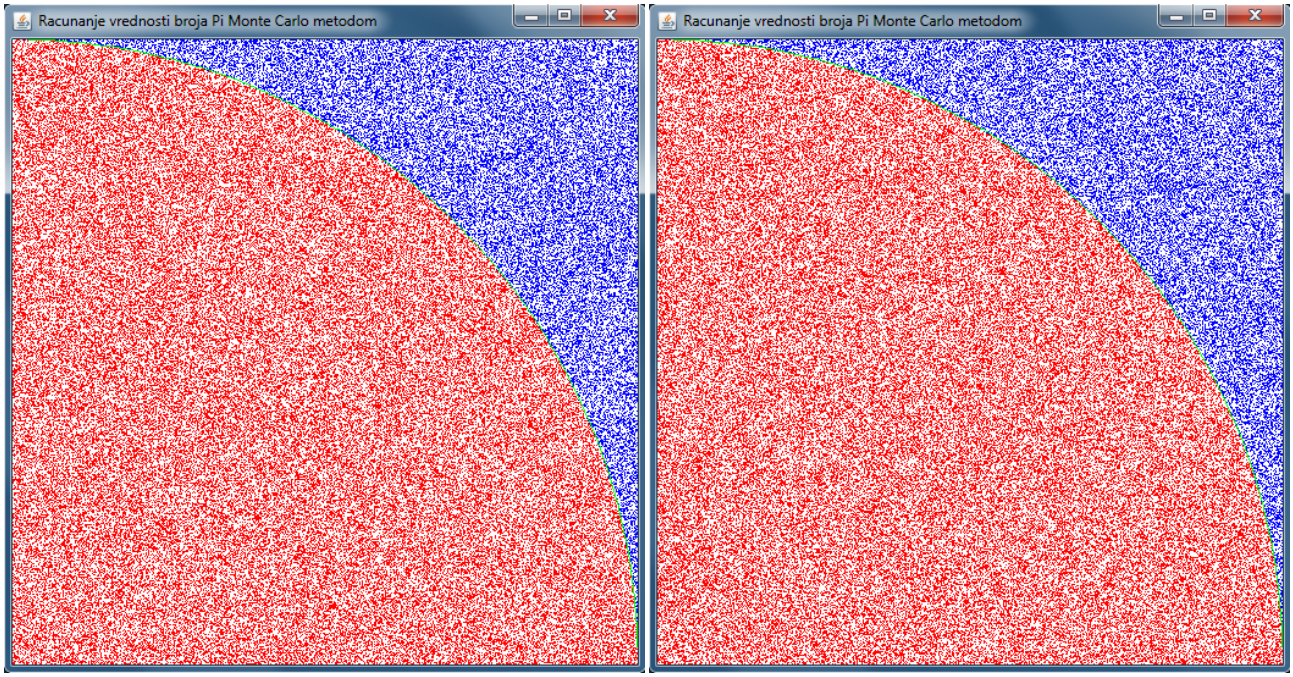
Rezultati dobijeni korišćenjem slučajnih brojeva sa hardverskog generatora su:

```
Uzorak velicine:878kB
Dobijena vrednost broja pi:3.1400266666666665
Stvarna vrednost broja pi :3.141592653589793
Procentualna greska:0.049846911926579196%
```

Rezultati dobijeni korišćenjem pseudoslučajnih brojeva Java random() generatora:

```
Uzorak velicine:878kB
Dobijena vrednost broja pi:3.13368
Stvarna vrednost broja pi :3.141592653589793
Procentualna greska:0.2518675863578892%
```

Grafički prikaz dobijene raspodele slučajno generisanih tačaka dat je na slici 22.



Slika 22: Grafički prikaz izračunavanja vrednosti broja PI Monte Karlo metodom za 150000 tačaka. Sa leve strane je prikazan rezultat raspodele dobijen korišćenjem slučajnih brojeva sa hardverskog generatora, a sa desne sa pseudogeneratora

Sa većim brojem korišćenih slučajno raspoređenih tačaka se dobija i tačnije rešenje Monte Karlo metodom. Grafički prikaz (slika 22) pokazuje dosta veću gustinu tačaka od grafičkih rezultata sa prethodnog testa (slika 21). Sada je broj π izračunat sa tačnim drugim decimalnim mestom, u slučaju rezultata korišćenjem slučajnih brojeva sa hardverskog generatora slučajnih brojeva. Razlika između dva generatora slučajnih brojeva i dalje postoji, i iznosi oko 5 puta, pri poređenju procentualnog odstupanja dobijenih vrednosti od prave vrednosti broja π.

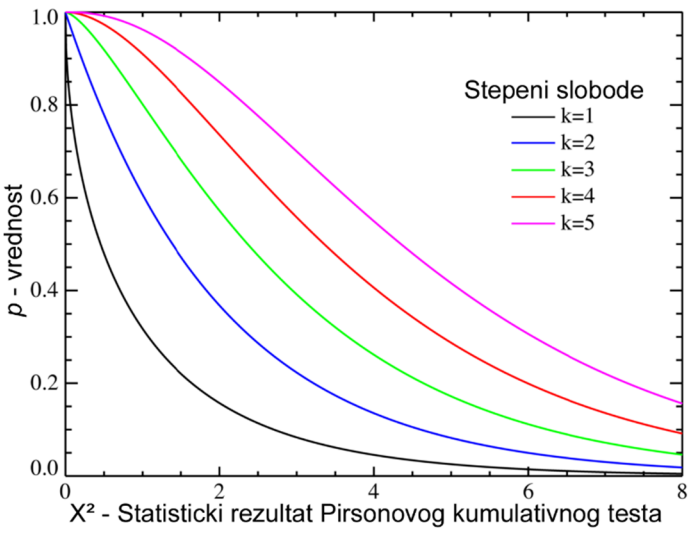
Hardverski generator slučajnih brojeva pokazuje dobre rezultate i u ovom testu. U kasnijim poglavljima će se ponoviti računanje broja π Monte Karlo metodom u komercijalnim programskim paketima za testiranje generatora slučajnih brojeva. Tamo će testiranja biti urađena na mnogo većem uzorku slučajno generisanih brojeva koji ovde nije korišćen zbog nemogućnosti pregledne grafičke prezentacije tolikog broja tačaka.

4.5. PIRSONOV CHI-SQUARE TEST

Pirsonov Chi-square test [21] važi za jedan od najčešće korišćenih testova za proveru kvaliteta generatora slučajnih brojeva. Može biti korišćen za bilo koji tip distribucije brojeva i koristi se u formi sume kvadrata grešaka. Rezultat testa je potvrda ili odbacivanje nulte hipoteze, koja tvrdi da ne postoje velike razlike u broju pojavljivanja pojedinačnih vrednosti, kao i da svaka primećena razlika u broju potiče od greške uzorkovanja podataka ili zbog eksperimentalnih grešaka. Chi-square test poredi posmatrane frekvencije pojavljivanja određenih vrednosti sa teorijskim frekvencijama pojavljivanja, koje se za uniformnu raspodelu računaju preko izraza $\frac{broj\ uzoraka\ (N)}{broj\ mogućih\ vrednosti}$.

Statistički rezultat Pirsonovog Chi-square testa se dobija izračunavanjem sume kvadrata grešaka prema sledećoj formuli: $\chi^2 = \sum_{i=1}^{n} \frac{(P_i - O_i)^2}{O_i}$, gde je n broj različitih generisanih vrednosti, P_i je posmatrani broj pojavljivanja određene vrednosti, a O_i je očekivani broj pojavljivanja određene vrednosti. Nakon toga potrebno je odrediti relativnu tj. kritičnu vrednost na osnovu koje će se nulta teorema prihvatiti ili odbaciti. Taj kritična vrednost se označava *p* vrednošću, koja predstavlja verovatnoću da se odstupanje posmatrane od očekivane vrednosti može pripisati samo slučajnosti. Npr. ukoliko se odabere vrednost *p*>0,05, očekuje se da je svako odstupanje posmatrane od očekivane vrednosti koje nastaje samo zbog slučajnosti, jednako 5% ili manje.

Treći korak izračunavanja Chi-square testa koristi Chi-square raspodelu koja može biti definisana pomoću krivih ili predstavljena tabelarno (slika 23). Chi-square raspodelom se definiše odnos *p* vrednosti u odnosu na χ^2 rezultat kumulativnog statističkog testa, u funkciji od stepena slobode. Broj stepeni slobode za uniformnu raspodelu ulaznih podataka se izračunava kao k=n-1, gde je n broj različitih generisanih vrednosti. Znajući izračunat rezultat statističkog testa χ^2 i broj stepeni slobode k, iz tabele ili grafika je moguće pronaći vrednost verovatnoće *p*. Ukoliko je dobijena p vrednost manja od očekivane kritične *p* vrednosti, nulta hipoteza se odbacuje, a svaka uočena razlika nastaje zbog drugih faktora sem slučajnosti. Ukoliko je dobijena *p* vrednost veća od očekivane *p* vrednosti, nulta hipoteza se prihvata i svaka uočena razlika predstavlja samo grešku koja je nastala zbog slučajnosti. Npr. dobijena vrednost *p*=0,6 označava da postoji 60% verovatnoće da je svako odstupanje od očekivane vrednosti posledica isključivo slučajnosti i da ulazni podaci (slučajno generisani brojevi) imaju uniformnu raspodelu. Iz dobijenog rezultata je moguće proceniti da li je generator slučajnih brojeva loš, ali ne i koliko je dobar. Procentualna vrednost dobijene verovatnoće *p* uz prihvatanje nulte hipoteze, ne bi smela da bude manja od 10 ili veća od 90% za dobre generatore slučajnih brojeva.



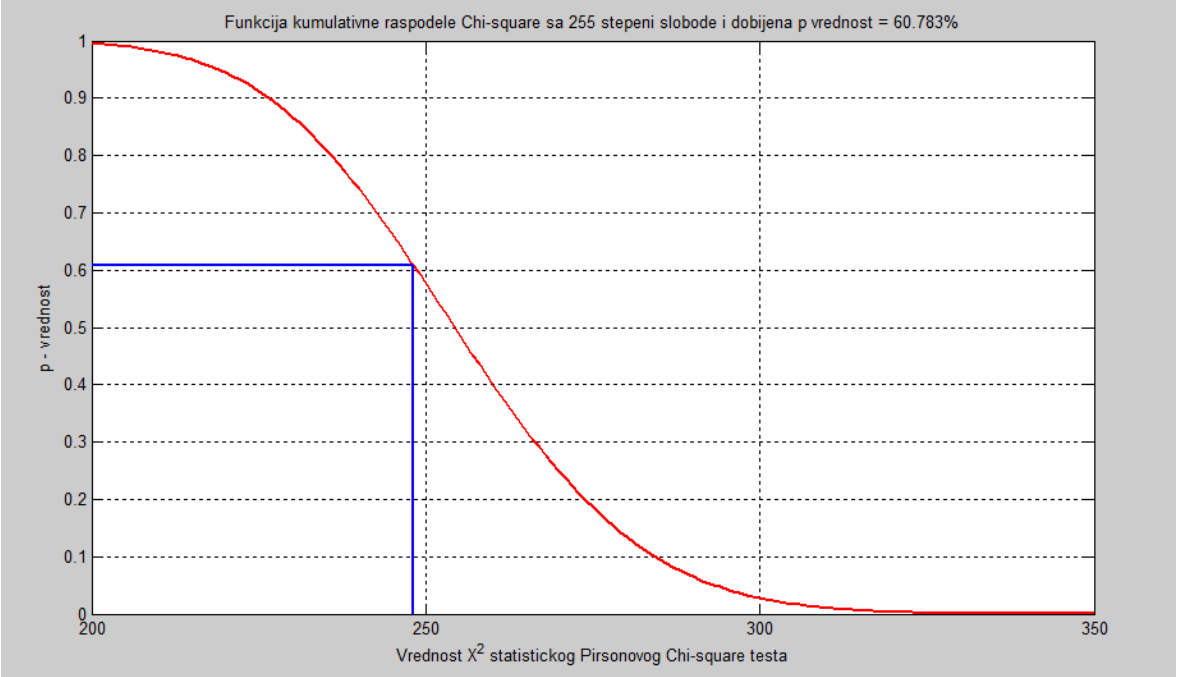
Slika 23: Chi-square funkcija raspodele

U narednom kodu 10 je napisan program u Matlab paketu koji računa vrednost p i iscrtava kumulativnu funkciju raspodele Chi-square. U kodu se izračunava ukupan broj pojavljivanja svake vrednosti u rasponu 0-255, broj stepeni slobode i vrednost statističkog kumulativnog Chi-square testa. Te vrednosti se dodaju kao argumenti funkcije "chi2cdf" koja kao izlaz daje vrednost *p* dobijenu sa kumulativne funkcije raspodele Chi-square. U drugom delu programa ta funkcija se iscrtava za određenu vrednost stepeni slobode (255) i crta se presek izračunate vrednosti χ^2 i dobijene vrednosti verovatnoće p. U naslovu je prikazana izračunata procentualna vrednost verovatnoće *p*.

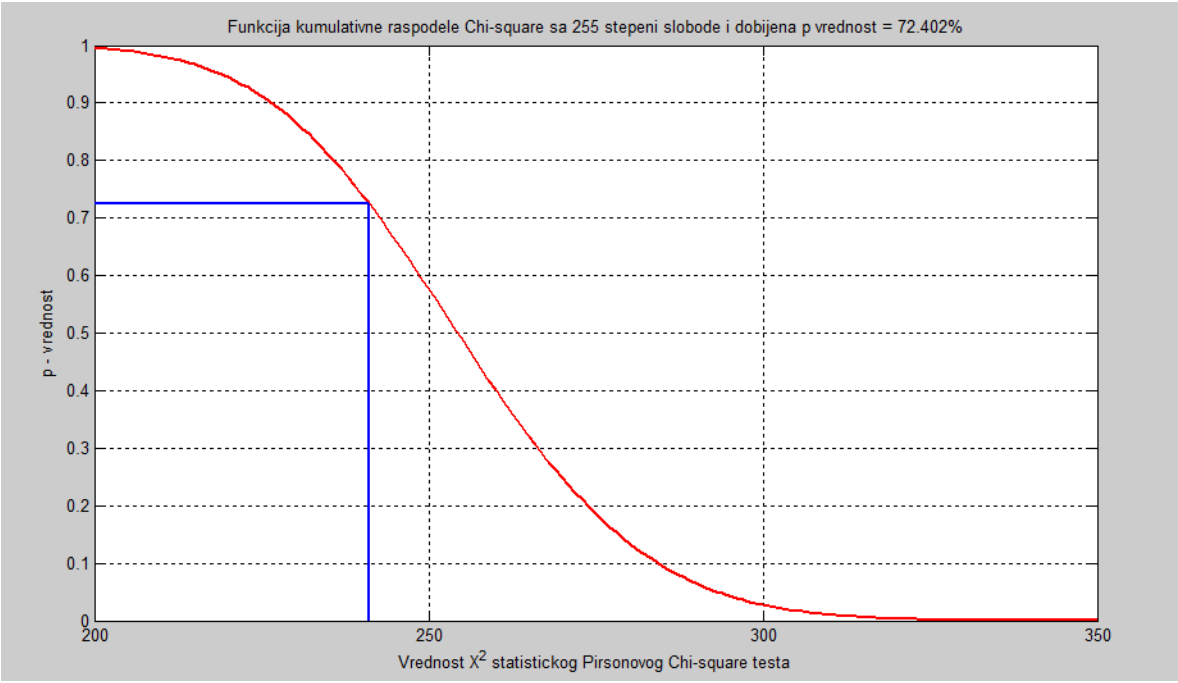
```
clear all;
close all;
fajl= fopen('izlaz3.bin');                  %Otvori datoteku
A=fread(fajl);                              %Procitaj bajtove u kolona matricu A
n=256;                                      %Broj vrednosti
N=length(A);                                %Duzina niza A - broj bajtova
s=zeros(256,1);                             %Postavi sve vrednosti na 0
for i=[1:N]                                 %Procitaj svaki bajt
        s(A(i)+1)=s(A(i)+1)+1;              %Saberi sa prethodnim pojavljivanjima
end
oc=N/256;                                   %Ocekivane vrednosti prema unif. raspodeli
x2=sum(power(s-oc,2)/oc)                    %X^2 - Chi square kumualtivni stat. test
k=N-1;                                      %Stepeni slobode = 255
p=chi2cdf(x2,255,'upper')                   %Izracunavanje verovatnoce p

x=0:1:350;                                  %Iscrtavanje Chi-square kumulativne f-je raspodele
d=chi2cdf(x,255,'upper');
plot(x,d,'r','Linewidth',2);                %Iscrtaj funkciju raspodele za 255 stepeni slobode
axis([200,350,0,1]);                        %Granice grafikona
grid;                                       %Mreza
hold on;
line([x2,x2],[0,p],'Linewidth',2,'Color','b'); %Iscrtaj vert. liniju preseka sa krivom
hold on;
line([0,x2],[p,p],'Linewidth',2,'Color','b');  %Iscrtaj horiz. liniju preseka sa krivom
xlabel ('Vrednost X^2 statistickog Pirsonovog Chi-square testa');
ylabel ('p - vrednost');
title (['Funkcija kumulativne raspodele Chi-square sa 255 stepeni slobode i dobijena p vrednost = '
num2str(p*100,'%3.3f') '%']);
```

Kod 10: Pirsonov Chi-square test

Na narednoj strani su predstavljeni rezultati Pirsonovog Chi-square testa. Prvi ulazni set podataka čine slučajni brojevi generisani pomoću hardverskog generatora slučajnih brojeva (slika 24), dok drugi set ulaznih podataka čini datoteka iste veličine sa slučajnim brojevima generisanim pseudoslučajnim algoritmom random() u programskom jeziku Java (slika 25).



Slika 24: Chi-square funkcija raspodele i vrednost p slučajnih brojeva generisanih hardverskim generatorom



Slika 25: Chi-square funkcija raspodele i vrednost p slučajnih brojeva generisanih Java pseudogeneratorom

Rezultat χ^2 kumulativnog statističkog testa iznosi 248,2128 u slučaju brojeva sa hardverskog generatora slučajnih brojeva, odnosno 241,1559 sa pseudoslučajnog generatora u Javi. Vrednost p je jednaka 0,6078 odnosno 60,78% u slučaju hardversko generatora, odnosno 0,7240 tj. 72,40% u slučaju pseudogeneratora. Oba generatora zadovoljavaju nultu hipotezu, dok brojevi sa hardverskog generatora slučajnih brojeva imaju malo bližu vrednost verovatnoće idealnoj oblasti u okolini 50%.

4.6. 3D NASUMIČNE ŠETNJE

Nasumične šetnje [22] predstavljaju još jedan način provere unifomne raspodele generisanih slučajnih brojeva, kao i prezentacije slučajnosti generisanih brojeva u obliku kretanja tačke. Izabran je trodimenzionalni (3D) način prezentacije. Tačka svoje kretanje započinje u koordinatama (0,0,0) i kreće se kroz prostor u koracima dužine 1. Opseg vrednosti koju može da ima slučajno generisani broj (0-255) je podeljen na 6 ravnomernih opsega. U zavisnosti od opsega kome pripada vrednost trenutno pročitanog slučajnog broja, tačka će napraviti jedan korak u pozitivnom ili negativnom smeru, na jednoj od 3 ose dekartovog koordinatnog sistema.

Trenutna razdaljina koju tačka pređe po x osi je $\Delta X = \Delta x_1 + \Delta x_2 + \Delta x_3 + \cdots$. Isto važi i za z i y osu. Kvadratna vrednost razdaljine tačke od koordinatnog početka iznosi: $R(N)^2 = \Delta X^2 + \Delta Y^2 + \Delta Z^2$, gde je razdaljina od koordinatnog početka R funkcija od broja pređenih koraka N. Ukoliko je raspodela generatora slučajnih brojeva zaista uniformna, tačka ima jednake verovatnoće pravljenja koraka u bilo kom smeru. To rezultuje različitim putanjama za različite šetnje tačke, kao i različitim daljinama koju će tačka preći od koordinatnog početka. Ukoliko postoji mnoštvo različitih šetnji M, i ukoliko u svakoj od njih tačka pređe identičan i vrlo veliki broj koraka N, usrednjavanjem pređenih puteva od koordinatnog početka prema broju šetnji dobiće se dužina pređenog puta u kojoj su uklonjene varijacije dužine puta svake pojedinačne nasumične šetnje.

Jedna šetnja ima pređeni put od koordinatnog početka jednak: $R(N)^2 = (\Delta x_1 + \Delta x_2 + \cdots + \Delta x_N) + (\Delta y_1 + \Delta y_2 + \cdots = (\Delta x_1)^2 + (\Delta x_2)^2 + \cdots + (\Delta x_N)^2 + \Delta x_1 \Delta x_2 + \cdots + \Delta x_1 \Delta x_N + (\Delta y_1)^2 + (\Delta y_2)^2 + \cdots + (\Delta y_N)^2 + \Delta y_1 \Delta y_2 + \cdots + \Delta y_1 \Delta y_N + \cdots + (\Delta z_1)^2 + \cdots$ Ukoliko se računa prosečna vrednost $\overline{R(N)^2}$ za M različitih šetnji, doći će do potiranja članova kao što je $\Delta x_1 \Delta x_2$ jer za veliki broj različitih šetnji, ti koraci imaju jednaku šansu da budu pozitivni ili negativni. Članovi tipa $(\Delta x_1)^2$ ostaju jer imaju kvadrirane vrednosti koje su uvek pozitivne i ne mogu da se skrate. Tako se dolazi do jednostavnijeg oblika ove jednačine: $\overline{R(N)^2} \approx (\Delta x_1)^2 + (\Delta x_2)^2 + (\Delta x_N)^2 + (\Delta y_1)^2 + (\Delta y_2)^2 + \cdots + (\Delta y_N)^2 + (\Delta z_1)^2 + (\Delta z_2)^2 + \cdots + (\Delta z_N)^2 = [(\Delta x_1)^2 + (\Delta y_1)^2 + (\Delta z_1)^2] + [(\Delta x_2)^2 + (\Delta y_2)^2 + (\Delta z_2)^2] + \cdots + [(\Delta x_N)^2 + (\Delta y_N)^2 + (\Delta z_N)^2]$. Svaka od dobijenih suma kao npr. suma $[(\Delta x_1)^2 + (\Delta y_1)^2 + (\Delta z_1)^2]$ ima vrednost 1, jer je pređeni put u jednom koraku uvek jednak 1, po uslovu kretanja tačke kroz prostor. Time se dolazi do jednačine $\overline{R(N)^2} \approx$ N koja je vrlo značajna jer govori da je srednja kvadratna vrednost pređenih puteva od koordinatnog početka za svih M šetnji jednaka broju koraka koju je tačka napravila u prostoru. Za veliki broj tačaka N i različitih pređenih puteva M, odnos $\overline{R(N)^2}$ i N je konstantan i približno jednak broju 1, ukoliko su generisani slučajni brojevi uniformne raspodele.

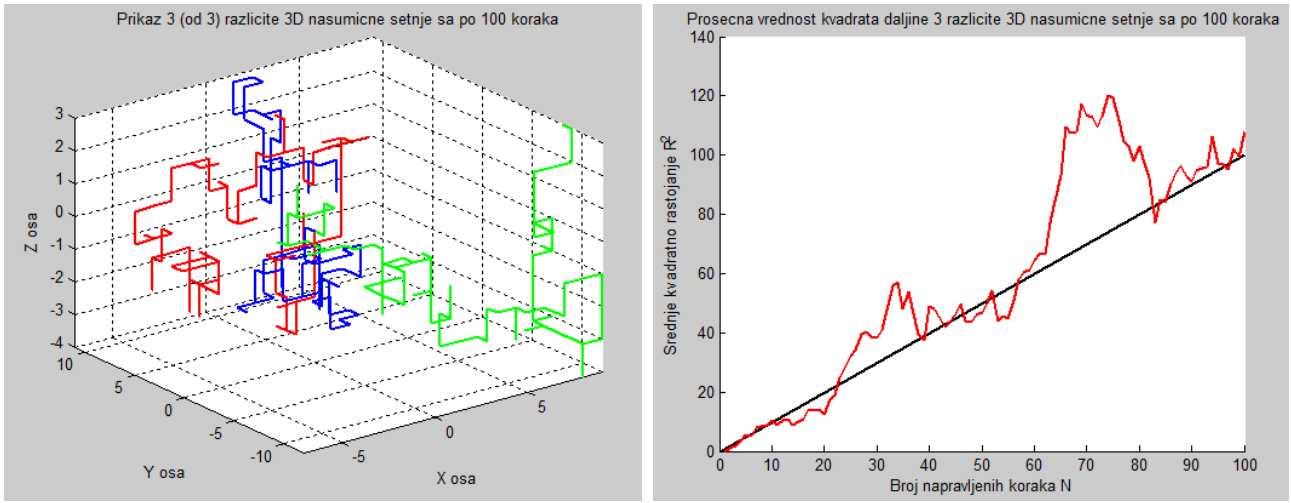
Za izračunavanje i grafički prikaz trodimenzionalnih nasumičnih šetnji, napisan je program u Matlabu (kod 11). Program prihvata određivanje broja šetnji (brs), i ukupnog broja slučajnih brojeva, koji se deli sa brojem šetnji kako bi se dobio broj koraka po jednoj šetnji (N/brs). Spoljni for ciklus određuje broj šetnje, a unutrašnji broj koraka jedne šetnje. Opseg jednog bajta je podeljen na 6 delova iste veličine, tako da svaki pročitani bajt mora da pripada jednom od tih opsega. U zavisnosti od toga kome opsegu broj pripada, tačka se kreće u određenom smeru. Matrica msd sadrži vrednost kvadrata rastojanja. Nakon toga se iscrtavaju 3 šetnje u 3D prostoru (radi bolje preglednosti) i izračunava se srednja vrednost pređenih rastojanja u matricu msdp. Iscrtava se i zavisnost pređenih koraka u odnosu na prosečno rastojanje tačke od koordinatnog početka svih šetnji. Crnom bojom je iscrtana idealna kriva odnosa $\overline{R(N)^2}$ i N, a crvenom dobijena vrednost.

```
clear all;
close all;
fajl= fopen('chijava2.bin');                %Otvori datoteku
A=fread(fajl,300);                          %Procitaj bajtove u kolona matricu A
N=length(A);                                %Duzina niza
brs=3;                                      %Broj setnji
x=zeros(N/brs,brs);                         %Postavi na nulu
y=zeros(N/brs,brs);                         %Postavi na nulu
z=zeros(N/brs,brs);                         %Postavi na nulu
msd=zeros(N/brs,brs);                       %Postavi na nulu
for p=[1:brs]                               %Od 1 do p setnji
   for i=[1:N/brs]                          %Od 1 do N/brs koraka putanje
      if A(i+((p-1)*N/brs))<42.666          %1. oblast
         x(i+1,p)=x(i,p)+1;                 %Uvecaj x, ne menjaj y i z
         y(i+1,p)=y(i,p);
         z(i+1,p)=z(i,p);
      end
      if A(i+((p-1)*N/brs))>=42.666 && A(i+((p-1)*N/brs))<85.333
         y(i+1,p)=y(i,p)+1;                 %Uvecaj y, ne menjaj x i z
         x(i+1,p)=x(i,p);
         z(i+1,p)=z(i,p);
      end
      if A(i+((p-1)*N/brs))>=85.333 && A(i+((p-1)*N/brs))<128
         z(i+1,p)=z(i,p)+1;                 %Uvecaj z, ne menjaj x i y
         x(i+1,p)=x(i,p);
         y(i+1,p)=y(i,p);
      end
      if A(i+((p-1)*N/brs))>=128 && A(i+((p-1)*N/brs))<170.666
         x(i+1,p)=x(i,p)-1;                 %Smanji x, ne menjaj y i z
         y(i+1,p)=y(i,p);
         z(i+1,p)=z(i,p);
      end
      if A(i+((p-1)*N/brs))>=170.666 && A(i+((p-1)*N/brs))<213.333
         y(i+1,p)=y(i,p)-1;                 %Smanji y, ne menjaj x i z
         x(i+1,p)=x(i,p);
         z(i+1,p)=z(i,p);
      end
      if A(i+((p-1)*N/brs))>=213.333
         z(i+1,p)=z(i,p)-1;                 %Smanji z, ne menjaj x i y
         x(i+1,p)=x(i,p);
         y(i+1,p)=y(i,p);
      end
      msd(i,p)=power(x(i,p),2)+power(y(i,p),2)+power(z(i,p),2); %Daljina^2
   end
end
```

```
%----------------------Iscrtavanje 3 3D putanje u prostoru-----------------
maxx=max(max(x));                          %Nadji maksimalnu vrednost x
minx=min(min(x));                          %Nadji minimalnu vrednost x
maxy=max(max(y));                          %Nadji maksimalnu vrednost y
miny=min(min(y));                          %Nadji minimalnu vrednost y
maxz=max(max(z));                          %Nadji maksimalnu vrednost z
minz=min(min(z));                          %Nadji minimalnu vrednost z
plot3(x(:,1),y(:,1),z(:,1),'r','Linewidth',2);          %Iscrtaj u 3D
grid;                                      %Nacrtaj mrezu
axis([minx,maxx,miny,maxy,minz,maxz]);          %Postavi ose
title (['Prikaz 3 (od ' num2str(brs) ') razlicite 3D nasumicne setnje sa po ' num2str(N/brs) ' koraka']);
xlabel ('X osa');
ylabel ('Y osa');
zlabel ('Z osa');
hold on;
plot3(x(:,2),y(:,2),z(:,2),'g','Linewidth',2);          %Iscrtaj drugu setnju
hold on;
plot3(x(:,3),y(:,3),z(:,3),'b','Linewidth',2);          %Iscrtaj trecu setnju

%---------Iscrtavanje prosecne vrednosti kvadrata predjene daljine---------
msdp=zeros(N/brs,1);                       %Izracunaj sr. vred. daljina^2
for e=[1:brs]                              %1 do broj setnji
   msdp(:,1)=msdp(:,1)+msd(:,e)/brs;       %izr. sr. vrednost
end
figure;                                    %Crtaj novi prozor
t=1:N/brs;                                  %1 do broj koraka N/brs
line([0,N/brs],[0,N/brs],'Linewidth',2,'Color','k');  %Iscrtaj horiz. liniju preseka sa krivom
hold on;
plot(t,msdp,'r','Linewidth',2);            %Iscrtaj idealnu vrednost prave
title ([' Prosecna vrednost kvadrata daljine ' num2str(brs) ' razlicite 3D nasumicne setnje sa po '
num2str(N/brs) ' koraka']);
xlabel ('Broj napravljenih koraka N');
ylabel ('Kvadratno rastojanje R^2');
```

Kod 11: Programski kod za vizuelizaciju i izračunavanje puta 3D nasumičnih šetnji, napisan u Matlabu

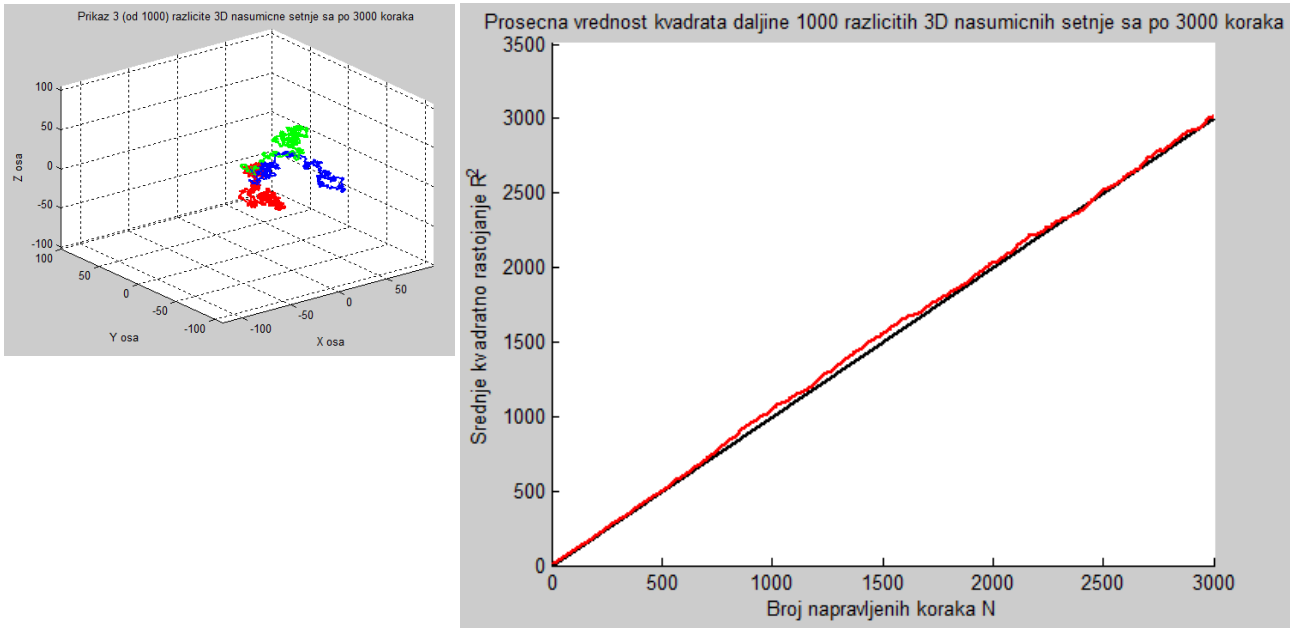
Na narednoj strani su dati rezultati analize slučajnih brojeva generisanih pomoću hardverskog generatora slučajnih brojeva, korišćenjem programa iz koda 11. Korišćeno je ukupno 300 bajtova (300 brojeva) za 3 šetnje, tj. 100 koraka po jednoj šetnji. Ovakva simulacija je isuviše gruba da bi se dobila adekvatna aproksimacija prave zavisnosti srednje vrednosti kvadrata rastojanja i ukupnog broja pređenih koraka, ali je pogodna za grafički prikaz načina kretanja tačke u trodimenzionalnom prostoru (slika 26).



Slika 26: Prikaz 3 3D nasumične šetnje sa po 100 koraka, i kriva srednjeg kvadratnog rastojanja u odnosu na broj koraka N. Slučajni brojevi su dobijeni pomoću hardverskog generatora slučajnih brojeva.

Sa slike 26 se uočava nasumična priroda kretanja tačke, kao i vrlo loše praćenje idealne prave odnosa $\overline{R(N)}^2$ i N zbog malog broja pređenih koraka, kao i malog broja šetnji.

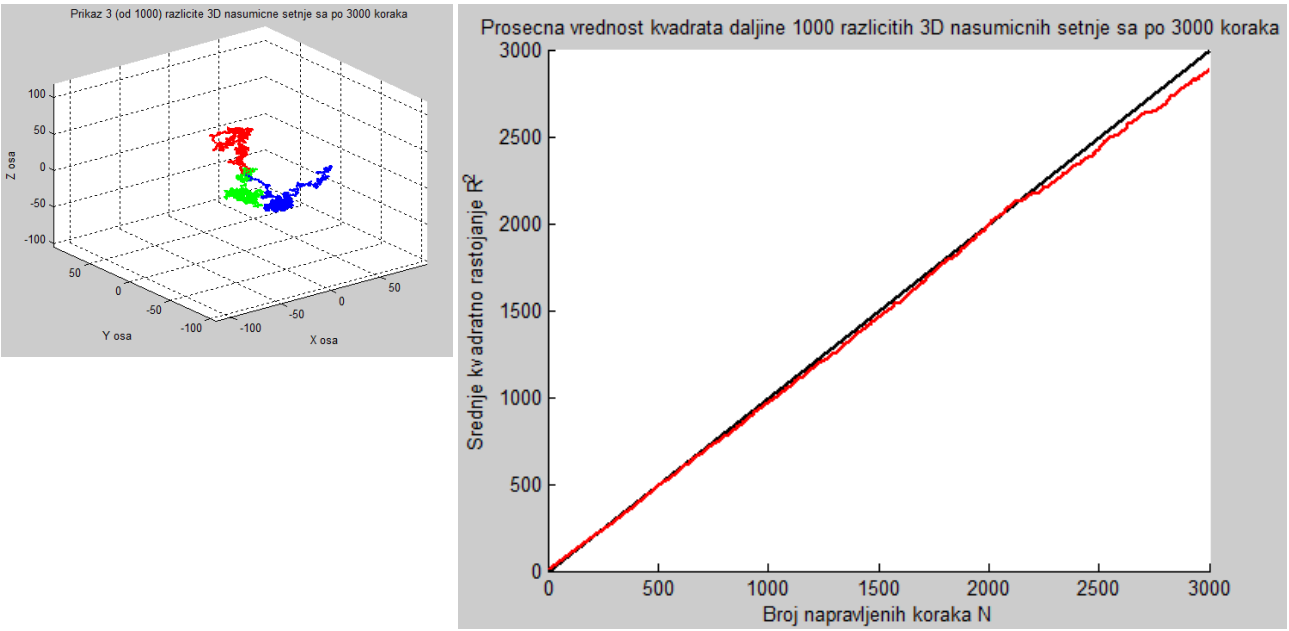
Analiza je ponovljena za ukupno 3000000 brojeva i 1000 različitih šetnji (slika 27).



Slika 27: Prikaz 3 (od 1000) 3D nasumične šetnje sa po 3000 koraka, i kriva R^2 – N. Slučajni brojevi su dobijeni pomoću hardverskog generatora slučajnih brojeva.

Sa prikaza krive zavisnosti $\overline{R(N)}^2$ i N (slika 27) se vidi da su brojevi generisani hardverskim generatorom slučajnih brojeva vrlo uniformne raspodele i da skoro idealno prate očekivanu vrednost prave. Male razlike su posledica malog broja koraka po šetnji (3000) i malog broja šetnji, ali se očekuje da bi odstupanja bila manja za još veći broj ulaznih podataka.

Testiranje je ponovljeno sa slučajnim brojevima dobijenim pseudoslučajnim Java random generatorom za isti broj ulaznih bajtova (3000000) i isti broj šetnji (1000). Rezultati su prezentovani na slici 28.



Slika 28: Prikaz 3 (od 1000) 3D nasumične šetnje sa po 3000 koraka, i kriva R^2 – N. Slučajni brojevi su dobijeni pomoću Java random pseudoslučajnog generatora brojeva.

Sa rezultata na slici 28 se vidi da brojevi, koji su generisani pomoću Java random pseudoslučajnog generatora, imaju veće odstupanje od uniformne raspodele nego brojevi dobijeni pomoću hardverskog generatora slučajnih brojeva. Odstupanje do 2000 koraka je vrlo malo, da bi nakon 2500 napravljenih koraka dobilo blagi trend rasta.

Test 3D nasumičnih šetnji pokazuje da je pogodan način za grafičko predstavljanje slučajnosti. Sama grafička prezentacija nasumičnog kretanja se koristi pri simuliranju različitih fizičkih procesa, npr. kretanja molekula u vazduhu koji prolaze nasumičnim putanjama. Zakonitost srednjeg kvadratnog rastojanja od broja napravljenih koraka daje pogodan način za kvantifikaciju uniformnosti određenog generatora slučajnih brojeva i prezentovanja rezultata na vrlo razumljiv način. Rezultati oba generatora slučajnih brojeva u testu 3d nasumičnih šetnji su vrlo dobri, sa blagom prednošću hardverskog generatora slučajnih brojeva.

4.7. TEST PROGRAM – ENT

Jedan od najčešće korišćenih test programa za proveru kvaliteta izlaza generatora slučajnih brojeva je program ENT [23]. Program ENT vrši više testova nad ulaznim podacima i prezentuje rezultate na jednostavan način. Pogodan je za testiranje pseudoslučajnih brojeva kao i slučajnih brojeva sa hardverskih generatora. Testovi koje vrši ENT su:

- Test entropije – Testira informacionu gustinu sadržaja ulazne datoteke. Rezultat je izražen u broju bita po karakteru. Datoteke koje sadrže veliku gustinu sadržaja ne mogu da budu mnogo komprimovane. Idealna vrednost iznosi 8.0.

- Chi-squared test – Test koji je prethodno opisan i urađen u poglavlju 4.5. Test će se ponoviti na istim ulaznim podacima i uporediće se rezultati sa prethodno dobijenim rezultatima. Očekivana vrednost rezultata je 10%<x<90%, idealna vrednost oko 50%.

- Aritmetička sredina – Određuje zbir svih podataka i deli je sa brojem podataka. Sličan test urađenom u poglavlju 4.1. Očekivana idealna srednja vrednost je 127,5.

- Monte Karlo metoda za izračunavanje broja π. Obrađeno u poglavlju 4.4. Očekivana vrednost rezultata je broj π. Dobija se i procentualna greška od idealne vrednosti, koja bi trebala da bude što bliža vrednosti 0%.

- Koeficijent serijske korelacije – Test određuje stepen korelacije svakog bajta u odnosu na prethodni bajt. Očekivana vrednost bi trebala da bude što bliža broju 0 za prave generatore slučajnih brojeva.

Prvi test je urađen sa podacima dobijenim sa hardverskog generatora slučajnih brojeva koji su već korišćeni za testove u prethodnim poglavljima. Rezultati testa su:

Entropy = 7.999968 bits per byte.

Optimum compression would reduce the size
of this 5562863 byte file by 0 percent.

Chi square distribution for 5562863 samples is 248.21, and randomly
would exceed this value 60.78 percent of the times.

Arithmetic mean value of data bytes is 127.4998 (127.5 = random).
Monte Carlo value for Pi is 3.141582259 (error 0.00 percent).
Serial correlation coefficient is 0.000009 (totally uncorrelated = 0.0).

Svi testovi pokazuju izuzetno dobre rezultate. Entropija je vrlo velika i jako blizu teoretskom maksimumu. Datoteka bi pri kompresiji izgubila podatke, tako da je datoteku moguće smanjiti za 0%, tj. nije moguće kompresovati je.

Chi-square χ^2 kumulativni statistički test ima istu vrednost od 248,21 tj. 60,78% kao i u poglavlju 4.5. To je vrlo dobar rezultat blizak teoretskoj idealnoj vrednosti od 50%.

Aritmetička sredina ima vrednost vrlo blisku idealnoj, sa izuzetno malom greškom.

Monte Karlo metoda dobijanja vrednosti broja π ne pokazuje nikakvu procentualnu grešku na dve decimale. U poglavlju 4.4. je dobijena greška od 0,04% , jer je uzorak slučajnih brojeva bio mnogo manji, zbog bolje preglednosti grafičke prezentacije.

Serijska korelacija pokazuje izuzetno malu vrednost korelacije između susednih bajtova koja je praktično jednaka nuli.

Test je ponovljen za uzorak iste veličine dobijen pomoću Java random pseudogeneratora. Rezultati su priloženi ispod:

Entropy = 7.999969 bits per byte.

Optimum compression would reduce the size
of this 5696914 byte file by 0 percent.

Chi square distribution for 5696914 samples is 241.16, and randomly
would exceed this value 72.40 percent of the times.

Arithmetic mean value of data bytes is 127.5276 (127.5 = random).
Monte Carlo value for Pi is 3.140633080 (error 0.03 percent).
Serial correlation coefficient is 0.000533 (totally uncorrelated = 0.0).

Rezultati Java random pseudogeneratora su vrlo dobri, ali malo lošiji u odnosu na rezultate hardverskog generatora slučajnih brojeva.

Jedini bolji rezultat ENT testova pseudogenerisanih brojeva jeste malo veća informaciona gustina podataka. Greške Chi-square testa, aritmetičke sredine, Monte Karlo metode izračunavanja broja π, kao i serijske korelacije bajtova su veće od grešaka dobijenih pomoću hardverskog generatora slučajnih brojeva.

Dobijeni rezultati ENT testova su potvrda valjanosti prethodnih programa koji su izračunavali iste testove, kao i potvrda kvaliteta slučajno generisanih sekvenci brojeva pomoću hardverskog generatora slučajnih brojeva. Rezultati ENT testa su značajni jer su lako uporedivi sa ostalim generatorima slučajnih brojeva. Pogodnost je što su skoro svi generatori slučajnih brojeva testirani ovim programom, i njihovi rezultati su lako dostupni na internetu.

4.8. TEST PROGRAM – DIEHARDER

Program za testiranje slučajno generisanih sekvenci brojeva Dieharder [24, 25] predstavlja trenutno najstrožiji skup različitih statističkih testova za testiranje izlaza generatora slučajnih brojeva. Poznato je da mnogi, vrlo dobri, generatori slučajnih brojeva ne zadovoljavaju oštre kriterijume pojedinih testova. Za ulaznu datoteku, prikupljeno je 119MB binarnih ASCII podataka (15,2MB binarna datoteka) u periodu od nekoliko dana. To je ujedno i problem, jer je za uspešno testiranje u programu Dieharder potrebno nekoliko GB ulaznih podataka, kako ne bi dolazilo do ponavljanja iste sekvence ulaznih podataka pri testiranju. U priloženim rezultatima se vidi broj ponavljanja sekvence ulaznih podataka pri svakom testu. Za svaki test se dobija potvrda da li je test uspešno završen, ili ne, kao i da li je uspešno završen sa slabim rezultatima. Rezultati su priloženi ispod:

```
#=============================================================================#
#         dieharder version 3.31.1 Copyright 2003 Robert G. Brown         #
#=============================================================================#
  rng_name    |         filename          |rands/second|
 file_input_raw|          /home/mx/izlaz.bin|  3.56e+07  |
#=============================================================================#
      test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
# The file file_input_raw was rewound 3 times
   diehard_birthdays|  0|     100|    100|0.01410536|  PASSED
# The file file_input_raw was rewound 29 times
     diehard_operm5|  0|  1000000|    100|0.00284364|   WEAK
# The file file_input_raw was rewound 62 times
  diehard_rank_32x32|  0|    40000|    100|0.83438125|  PASSED
# The file file_input_raw was rewound 77 times
    diehard_rank_6x8|  0|   100000|    100|0.10339097|  PASSED
# The file file_input_raw was rewound 84 times
   diehard_bitstream|  0|  2097152|    100|0.56710708|  PASSED
# The file file_input_raw was rewound 137 times
       diehard_opso|  0|  2097152|    100|0.00000575|   WEAK
# The file file_input_raw was rewound 173 times
       diehard_oqso|  0|  2097152|    100|0.82667386|  PASSED
# The file file_input_raw was rewound 190 times
        diehard_dna|  0|  2097152|    100|0.08961166|  PASSED
# The file file_input_raw was rewound 192 times
diehard_count_1s_str|  0|   256000|    100|0.80035604|  PASSED
# The file file_input_raw was rewound 225 times
diehard_count_1s_byt|  0|   256000|    100|0.20632949|  PASSED
# The file file_input_raw was rewound 225 times
 diehard_parking_lot|  0|    12000|    100|0.59549008|  PASSED
# The file file_input_raw was rewound 226 times
    diehard_2dsphere|  2|     8000|    100|0.74294642|  PASSED
# The file file_input_raw was rewound 226 times
    diehard_3dsphere|  3|     4000|    100|0.91711360|  PASSED
# The file file_input_raw was rewound 285 times
     diehard_squeeze|  0|   100000|    100|0.00036176|   WEAK
# The file file_input_raw was rewound 285 times
       diehard_sums|  0|      100|    100|0.40141262|  PASSED
# The file file_input_raw was rewound 288 times
       diehard_runs|  0|   100000|    100|0.38300021|  PASSED
```

Priloženi rezultati pokazuju da hardverski generator slučajnih brojeva dobija prolazne ocene na većini testova, ali i poneke slabe i neprolazne. Glavni krivac za takav rezultat je količina sakupljenih podataka, jer se jasno uočava veliki broj ponavljanja iste ulazne sekvence podataka za testove koji daju loše ili neprolazne rezultate. Npr. marsaglia test ponavlja ulaznu sekvencu podataka čak 835 puta što kao rezultat daje neprolaznu ocenu.

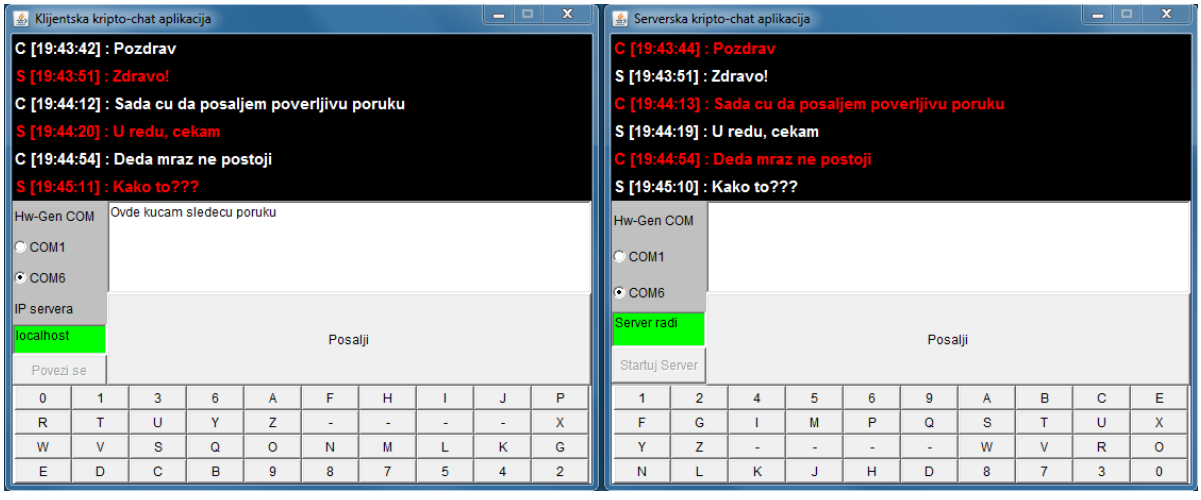
Time se uočava i najveća mana hardverskog generatora slučajnih brojeva, a to je brzina generisanja podataka koja iznosi oko 8MB binarnih ASCII podataka na 24 sata. To je ekvivalentno 1MB binarnih podataka na 24 sata, tj. oko 42kB/h. To je ujedno i mana ostalih tipova hardverskih generatora brojeva koji su dosta sporiji od pseudoslučajnih generatora brojeva. Uniformnim vremenskim odabiranjem signala sa A/D konvertora mikrokontrolera, moguće je postići skoro duplo veću brzinu generisanih slučajnih brojeva. Međutim to nije urađeno zbog prednosti slučajnog vremenskog odabiranja signala o kojima je ranije pisano. Prednost je data boljim statističkim rezultatima i nedeterminističkom načinu generisanja brojeva u odnosu na brzinu, jer je to značajniji faktor za primenu u kriptografiji.

```
# The file file_input_raw was rewound 322 times
     diehard_craps|  0|   200000|    100|0.00049185|  WEAK
     diehard_craps|  0|   200000|    100|0.01878370|  PASSED
# The file file_input_raw was rewound 835 times
 marsaglia_tsang_gcd|  0|  10000000|    100|0.00000000|
FAILED
 marsaglia_tsang_gcd|  0|  10000000|    100|0.00000000|
FAILED
# The file file_input_raw was rewound 838 times
      sts_monobit|  1|   100000|    100|0.00000276|  WEAK
# The file file_input_raw was rewound 840 times
       sts_runs|  2|   100000|    100|0.28944778|  PASSED
# The file file_input_raw was rewound 843 times
      sts_serial|  1|   100000|    100|0.00002372|  WEAK
      sts_serial|  2|   100000|    100|0.28111389|  PASSED
      sts_serial|  3|   100000|    100|0.70661783|  PASSED
      sts_serial|  3|   100000|    100|0.02859715|  PASSED
      sts_serial|  4|   100000|    100|0.27777929|  PASSED
      sts_serial|  4|   100000|    100|0.12279160|  PASSED
      sts_serial|  5|   100000|    100|0.48589892|  PASSED
      sts_serial|  5|   100000|    100|0.15904369|  PASSED
      sts_serial|  6|   100000|    100|0.43931851|  PASSED
      sts_serial|  6|   100000|    100|0.32751671|  PASSED
      sts_serial|  7|   100000|    100|0.15777601|  PASSED
      sts_serial|  7|   100000|    100|0.71785292|  PASSED
      sts_serial|  8|   100000|    100|0.01306696|  PASSED
      sts_serial|  8|   100000|    100|0.00360022|  WEAK
      sts_serial|  9|   100000|    100|0.00003101|  WEAK
      sts_serial|  9|   100000|    100|0.02728893|  PASSED
      sts_serial| 10|   100000|    100|0.00528164|  PASSED
      sts_serial| 10|   100000|    100|0.00001434|  WEAK
      sts_serial| 11|   100000|    100|0.01334389|  PASSED
      sts_serial| 11|   100000|    100|0.00065588|  WEAK
      sts_serial| 12|   100000|    100|0.46671010|  PASSED
      sts_serial| 12|   100000|    100|0.44966843|  PASSED
      sts_serial| 13|   100000|    100|0.15493216|  PASSED
      sts_serial| 13|   100000|    100|0.03848705|  PASSED
      sts_serial| 14|   100000|    100|0.29649401|  PASSED
      sts_serial| 14|   100000|    100|0.02069922|  PASSED
      sts_serial| 15|   100000|    100|0.26485463|  PASSED
      sts_serial| 15|   100000|    100|0.40401723|  PASSED
      sts_serial| 16|   100000|    100|0.06635974|  PASSED
      sts_serial| 16|   100000|    100|0.06985762|  PASSED
# The file file_input_raw was rewound 848 times
     rgb_bitdist|  1|   100000|    100|0.54074818|  PASSED
# The file file_input_raw was rewound 858 times
     rgb_bitdist|  2|   100000|    100|0.33738266|  PASSED
# The file file_input_raw was rewound 874 times
     rgb_bitdist|  3|   100000|    100|0.98758396|  PASSED
# The file file_input_raw was rewound 894 times
     rgb_bitdist|  4|   100000|    100|0.58192441|  PASSED
# The file file_input_raw was rewound 920 times
     rgb_bitdist|  5|   100000|    100|0.85810226|  PASSED
```

5. MREŽNI ČET PROGRAM KOJI KORISTI HARDVERSKI GENERATOR SLUČAJNIH BROJEVA ZA ŠIFROVANJE PRENOŠENIH PORUKA

Za prikaz praktične primene hardverskog generatora slučajnih brojeva, napisan je mrežni čet program koji služi za prenos šifrovanih poruka preko nesigurnog mrežnog kanala. Napisana su dva programa, od kojih jedan predstavlja server, a drugi klijent čet program. Zbog malih razlika između dva programa, biće prikazan celokupan kod klijentskog čet programa i samo razlike u odnosu na njega koje su prisutne u serverskom čet programu. Na slici 29 je prikazan izgled oba programa u radu, prilikom međusobne komunikacije na istom računaru koristeći localhost mrežnu adresu (loopback adresu).

Za šifrovanje podataka je odabran PGP (pretty good privacy – prilično dobra privatnost) protokol. Sem vrlo jake kriptografske zaštite, PGP nudi i zaštitu integriteta poruke i autentikacije izvora poruke. Primarna zaštita poruka koje se šalju se vrši pomoću simetričnog šifarskog algoritma AES (poznatog i pod nazivom Rijndael). AES prihvata ključeve veličine 128, 192 i 256 bita, od kojih je odabrana veličina ključa od 128 bita radi najbržeg slanja podataka. Sam ključ se generiše pomoću hardverskog generatora slučajnih brojeva, čiji se serijski COM port otvara u trenutku čitanja ključa i zatvara odmah nakon čitanja. To omogućuje da klijentski i serverski čet program mogu da se nalaze na istom računaru radi lakše demonstracije, jer ne koriste serijski port u isto vreme. Ujedno, zauzeće serijskog porta prilikom čitanja generisanih brojeva sa hardverskog generatora slučajnih brojeva, onemogućava bilo kom drugom programu da pristupi serijskom portu i da pročita generisane brojeve. Time je postignuta zaštita od zlonamernih programa koji bi mogli da pročitaju generisane šifre. U idealnim uslovima, klijentski i serverski čet program bi trebali da se nalaze na odvojenim računarima, od kojih svaki poseduje opisani hardverski generator slučajnih brojeva. Odabir serijskog COM porta na kome je priključen hardverski generator slučajnih brojeva se postiže odabirom jednog od imena svih prisutnih COM portova na računaru, na panelu sive boje, koji se nalazi na levoj strani programa (slika 29).



Slika 29: Prikaz rada klijentskog i serverskog čet programa koji prenose šifrovane poruke nesigurnim mrežnim kanalom

Kao dodatni vid zaštite od zlonamernih programa, u čet programu je napravljena virtuelna tastatura, koja se nakon svakog slanja poruke ponovo generiše, sa karakterima u slučajnom rasporedu. Time se sprečava način prisluškivanja poruka pri samom ukucavanju na tastaturi, pomoću programa koji pamte pritisnute tastere (keylogger) ili programa koji prate kretanje i pritiskanje tastera kompjuterskog miša na ekranu. Raspored karaktera se takođe određuje pomoću slučajnih brojeva dobijenih sa hardverskog generatora slučajnih brojeva, i to čitanjem 43 bita ulaznih podataka. Na slici 29 se uočava različit raspored karaktera na virtuelnoj tastaturi na klijentskom i serverskom programu.

Nakon startovanja programa, potrebno je prvo odabrati COM port na kome je priključen hardverski generator slučajnih brojeva. Zatim je potrebno pokrenuti server (taster „Startuj server“) koji osluškuje na portu 1234 dolazne zahteve za uspostavljanjem veze. Klijent sem odabira COM porta, treba da upiše i IP adresu servera, ili da ostavi podrazumevanu loopback adresu localhost, ukoliko je i serverski čet program na istom računaru. Nakon što klijent pritisne dugme “Povezi se”, veza između dva programa je uspostavljena. U slučaju neuspešnog povezivanja, odgovarajuća zelena tekstualna polja bi bila predstavljena crvenom bojom. Posle uspešnog povezivanja, moguće je unositi poruke tastaturom ili pritiskanjem dugmića na virtuelnoj tastaturi, što je preporučen način unosa teksta za maksimalnu sigurnost. Poruka može da se pošalje pritiskom na taster “Posalji” ili pritiskom na taster ENTER na tastaturi. Poslednjih 6 poruka je prikazano u crnom prozoru pri vrhu programa, belom bojom su označene poruke poslate sa datog čet programa, a crvenom primljene. Svaka poruka sadrži slovo S ili C, koje označava da je poruka pristigla sa servera ili klijenta, respektivno. Takođe, uz poruku se nalazi i vreme kada je primljena ili poslata.

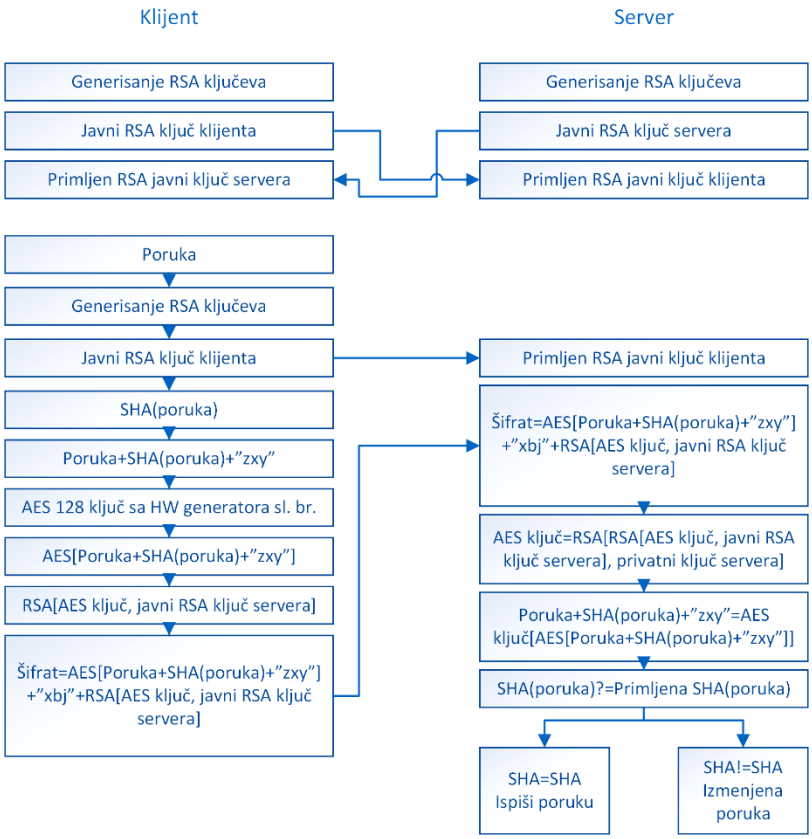
Zaštita poruka PGP protokolom se odvija u više koraka (slika 30) [26]. Šifrovanje podataka se odvija u metodi sifrujPoruku(). Upisanoj poruci se nakon pritiska na dugme za slanje, pridodaje njen heš otisak dobijen pomoću SHA algoritma. Ovaj korak je značajan jer se slanjem heš otiska poruke uz samu poruku, obezbeđuje potvrda integriteta poruke. Kada strana koja prima poruku izračuna heš vrednost dobijene dešifrovane poruke, i uporedi je sa poslatom heš vrednošću, znaće da li je poruka stigla u izmenjenom obliku ili originalnom. Sam SHA algoritam proizvodi heš vrednosti fiksne dužine 160 bita, tj. 40 brojeva u hexadecimalnom zapisu. Uz poruku i njenu heš vrednost, pridodaje se i string “zxy”, kao mera dodatne zaštite od određivanja tačne dužine heš zapisa u poruci.

Tako proširena poruka se šifruje AES simetričnim šifarskim algoritmom za koji se ključ od 128 bita čita sa izlaza hardverskog generatora slučajnih brojeva. Time se obezbeđuje potpuna nedeterministička priroda dobijanja ključa, kao i neponovljivost istog. Ni osoba koja je poslala poruku ne zna koja je šifra upotrebljena. Mala brzina generisanja brojeva pomoću hardverskog generatora slučajnih brojeva ne predstavlja problem zbog male dužine ključa koja zahteva prikupljanje podataka u trajanju manjem od jedne sekunde. AES ključ je potrebno poslati i drugoj strani kako bi mogla da dešifruje poruku. To se postiže šifrovanjem AES ključa RSA algoritmom.

RSA algoritam [26] predstavlja sistem šifrovanja pomoću javnog i privatnog ključa. U slučaju korišćenja u kriptografske svrhe, drugoj strani se šalje javni ključ kojim ona šifruje podatke i šalje ih nazad. Njih je moguće dešifrovati samo privatnim ključem koji nije nikada napustio računar na kome je generisan. Generisanje velikih prostih brojeva se vrši korišćenjem klase BigInteger koja koristi pseudogenerator slučajnih brojeva. Koristi se ključ dužine 512 bita zbog brzog generisanja. Prilikom šifrovanja svake nove poruke, generišu se novi privatni i javni RSA ključevi, i javni ključ se šalje drugoj strani. Tada druga strana može

da šifruje svoj AES ključ primljenim javnim ključem. AES šifrovana poruka i njena heš vrednost, kao i RSA šifrovan AES ključ se šalju kao šifrat drugoj strani, sa dodata tri karaktera između njih (“xbj”), kako bi se lakše odredila granica između dva dela šifrata.

Nakon prvog uspostavljanja veze između serverskog i klijentskog čet programa, šalju se generisani javni RSA ključevi drugoj strani. Nakon toga, svakim novim šifriranjem i slanjem poruke, jedna strana šalje drugoj novogenerisani javni RSA ključ. Time se postiže visok nivo zaštite podataka, jer se u slučaju dešifrovanja RSA i AES algoritama (koje je trenutno izuzetno vremenski zahtevno) postiže samo dešifrovanje jedne poruke. Ne koristi se sesijski ključ čijim dešifrovanjem bi se omogućio pristup svim ostalim porukama.



Slika 30: Prikaz protokola čet programa i PGP načina šifrovanja

Dešifrovanje poruke se odvija obrnutim redosledom u metodi desifrujPoruku() (slika 30). Prvo se razdvaja šifrat poruke od šifrata ključa korišćenjem stringa “xbj”. Zatim se dešifruje AES ključ, korišćenjem privatnog ključa algoritma RSA. Tada je moguće dešifrovati sadržaj poruke i heš funkcije. Vrši se izdvajanje poruke od njene heš vrednosti. Računa se heš vrednost pomoću SHA algoritma primljene poruke i poredi sa primljenom heš vrednošću. Ukoliko su identične, ispisuje se primljena poruka, u suprotnom se ispisuje tekst “poruka je modifikovana”.

Na početku programa su definisane promenljive potrebne za rad (kod 12). Definiše se klasa “poruka” u kojoj se čuva 6 prošlih poruka sa datumima i stranom koja ih je poslala. Poruke se upisuju u FIFO redu, tako da je uvek poslednjih 6 poruka sačuvano. Konstruktor klase Client() definiše izgled i veličinu prozora, kao i panele u njemu. Nakon toga slede definicije panela na kojima se nalaze elementi interfejsa. U nastavku su definisane klase za osluškivanje promena nad elementima interfejsa, koje reaguju na odabrane komande i pozivaju odgovarajuće metode. Klasa “citac” je posebna programska nit koja služi za čitanje pristiglih podataka, i koja je potrebna za sinhrono primanje i slanje podataka. Metode se definišu u nastavku programa. Metoda za povezivanje otvara novi soket na datoj IP adresi i portu, generiše nove RSA ključeve i šalje javni ključ drugoj strani, i generiše novi raspored tastature. Metoda za generisanje rasporeda karaktera na virtuelnoj tastaturi čita samo 43 bita sa hardverskog generatora slučajnih brojeva, i koristeći svaku dobijenu jedinicu postavlja karakter na mesto od početka niza, a svakom nulom ga smešta na prvo slobodno mesto sa

kraja niza. Metode za slanje i primanje poruka pozivaju metode za šifrovanje i dešifrovanje poruka, smeštaju podatke u objekat klase poruka i ispisuju poslednjih 6 poruka na ekranu. Metode za šifrovanje i dešifrovanje su prethodno opisane.

Za program se koristi uslužna bibliteka jssc pomoću koje se vrši pristup i očitavanje serijskog COM porta. Takođe koristi se klasa Base64Coder kojom se vrši prebacivanje binarnih podataka u ASCII stringove i obrnuto. Klasa PGP u sebi sadrži metode za generisanje RSA ključeva, za RSA šifrovanje i dešifrovanje, kao i za AES šifrovanje i dešifrovanje.

U kodu 12 je prikazan kod klijentskog čet programa.

```
package chat_pgp;
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Panel;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.math.BigInteger;
import java.net.Socket;
import java.util.Calendar;

import chat_aplikacija.PGP;
import jssc.SerialPort;
import jssc.SerialPortException;
import jssc.SerialPortList;

public class Client extends Frame{
	private static final long serialVersionUID = 1L;
	private TextField istorija,slanje,adresa;		//Tekstualna polja
	private Button posalji,povezi;					//Dugmici
	private static Label[] tekst=new Label[6];		//Sve prosle poruke
	private static poruka poruke=new poruka();		//Klasa proslih poruka
	private Button dugmici[]=new Button[40];		//Dugmici tastature
	private String combroj,ipadresa;				//Broj COM HW generatora i IP adresa
```

```
	private SerialPort serialPort;				//Serijski port
	private Socket sock;					//Socket
	private static BufferedReader in;			//Ulaz preko socketa
	private static PrintWriter out;				//Izlaz preko socketa
	private static BigInteger es,ns,dc,ec,nc;		//Javni kljuc servera i privatni i javni kljuc klijenta

	private static class poruka{			//Klasa koja sadrzi 6 proslih poruka u FIFO nizu
		private static String[] ulaz=new String[6];	//Poruke
		private Calendar sada ;				//Vreme
		private static int mesto=0;
		poruka(){}					//Konstruktor
		public synchronized void unesiPoruku(String poruka, char k){ //Unosenje poruka
			sada=Calendar.getInstance();			//Uzmi trenutno vreme
			String
vreme="["+sada.get(Calendar.HOUR_OF_DAY)+":"+sada.get(Calendar.MINUTE)+":"+sada.get(Calendar.SECOND)+
"] : ";
			if(mesto==6){				//Ukoliko je zapisano svih 6 poruka, pomeri poruke
				for(int i=0;i<5;i++){
					ulaz[i]=ulaz[i+1];
				}
				if(k=='c') ulaz[5]="C "+vreme+poruka;//Upisi novu poruku (c-klijent, S-server)
				if(k=='s') ulaz[5]="S "+vreme+poruka;
			}else{
				if(k=='c') ulaz[mesto]="C "+vreme+poruka;//Upisi novu poruku (c-klijent, S-
server)
				if(k=='s') ulaz[mesto]="S "+vreme+poruka;
				++mesto;
			}
		}
		public void ispisiPoruke(){		//Ispisi sve poruke
			for(int i=0;i<6;i++){		//Crvenom bojom poruke sa servera, belom sa klijenta
				tekst[i].setText(ulaz[i]);
				if(ulaz[i]!=null && ulaz[i].charAt(0)=='C') tekst[i].setForeground(Color.WHITE);
				if(ulaz[i]!=null && ulaz[i].charAt(0)=='S') tekst[i].setForeground(Color.RED);
			}
		}
	}
	private Client (){					//Konstruktor Klijenta
		super("Klijentska kripto-chat aplikacija");	//Naslov
		setSize(550,450);				//Velicina prozora
		setResizable(false);
		add(plocaCOM(),BorderLayout.WEST);		//Dodavanje ploca
		add(plocaIstorija(),BorderLayout.NORTH);
		add(plocaUnosTeksta(),BorderLayout.CENTER);
		add(plocaDugmici(),BorderLayout.SOUTH);
		addWindowListener(new prozorDogadjaji());//Dodaj osluskivac dogadjaja prozora
		setVisible(true);
		adresa.setText("localhost");			//Podrazumevana IP adresa
	}
```

```
//-------------------------Paneli sa elementima grafickog interfejsa---------------------------------

        private Panel plocaCOM(){           //Panel sa COM dugmicima i dugmetom za povezivanje na socket
                String[] imenaPortova = SerialPortList.getPortNames();//Imena dostupnih serijskih portova
                int brojcom=imenaPortova.length;                    //Broj COM portova na racunaru
                combroj=imenaPortova[brojcom-1];
                Panel ploca=new Panel(new GridLayout(brojcom+4,1)); //Raspored
                ploca.setBackground(Color.LIGHT_GRAY);              //Boja
                ploca.add(new Label("Hw-Gen COM"));                 //Labela
                CheckboxGroup grupa=new CheckboxGroup();            //Checkbox dugmici
                CheckboxPromena promenaCOM=new CheckboxPromena();
                for(int i=0;i<brojcom;i++){                         //Dodaj dugmice
                        Checkbox radio=new Checkbox(imenaPortova[i],grupa,true);
                        ploca.add(radio);
                        radio.addItemListener(promenaCOM);
                }
                ploca.add(new Label("IP servera"));                 //Labela
                ploca.add(adresa=new TextField());                  //Polje za unos IP adrese
                ploca.add(povezi=new Button("Povezi se"));          //Dugme za povezivanje
                povezi.addActionListener(new dugmeIP());            //Dodaj osluskivac dogadjaja
                return ploca;
        }

        private Panel plocaIstorija(){                              //Panel sa proslim porukama
                Panel ploca=new Panel(new GridLayout(6,1));         //Raspored
                ploca.setBackground(Color.BLACK);                   //Crna boja pozadine
                for(int i=0;i<6;i++){                               //Ispisi 6 Labela
                        tekst[i]=new Label ("",Label.LEFT);         //Levo poravnjanje i font
                        tekst[i].setFont(new Font(null,Font.BOLD,15));
                        ploca.add(tekst[i]);
                }
                return ploca;
        }

        private Panel plocaUnosTeksta(){                    //Panel za plocu za unos teksta i dugme za slanje
                Panel ploca=new Panel(new GridLayout(2,1));         //Raspored
                ploca.add(slanje=new TextField());                  //Dodaj polje za tekst
                slanje.setEditable(true);                           //Moguce je menjati tekst
                slanje.addKeyListener(new pritisnutEnter());//Dodaj osluskivac dugmica, za enter dugme
                ploca.add(posalji=new Button("Posalji"));           //Dugme za slanje poruke
                posalji.addActionListener(new dugmeSlanje());
                return ploca;
        }

        private Panel plocaDugmici(){                       //Panel sa dugmicima koji predstavlja tastaturu
                Panel ploca=new Panel(new GridLayout(4,10));//Raspored
                for(int i=0;i<40;i++){                              //40 dugmica
                        ploca.add(dugmici[i]=new Button("-"));      //U pocetku su svi -
                        dugmici[i].addActionListener(new tastaturaAkcija());//Osluckivac dogadjaja
                }
                return ploca;
        }
```

```
//-----------------------------Klase za osluskivanje dogadjaja (akcija) nad interfejsom-------------

private class CheckboxPromena implements ItemListener{      //Osluskivac promena grupe sa COM portovima
                public void itemStateChanged (ItemEvent d){
                        combroj=((Checkbox)(d.getSource())).getLabel();//Vrednost selektovanog COM porta
                }
        }
        private class prozorDogadjaji extends WindowAdapter{          //Osluskivac zatvaranja prozora
                public void windowClosing(WindowEvent d){
                        if(sock!=null) {
                                try {sock.close();} catch (IOException e) {}//Zatvori socket
                                }
                        System.exit(0);                                 //Zatvori prozor
                }
        }

        private class dugmeSlanje implements ActionListener{//Osluskivac pritiska dugmeta za slanje poruke.
                public void actionPerformed(ActionEvent d){
                        slanjePoruke();
                }
        }

        private class dugmeIP implements ActionListener{    //Osluskivac pritiska dugmeta za povezivanje
                public void actionPerformed(ActionEvent d){
                        ipadresa=adresa.getText();                      //Preuzmi unetu IP adresu
                        adresa.setBackground(Color.GREEN);              //Pozadina adrese je zelena
                        try {poveziSe();} catch (Exception e) {         //Probaj da se povezes na server
                                adresa.setBackground(Color.RED); //Ako ne uspe povezivanje, crvena boja
                        }
                        povezi.setEnabled(false);                       //Dugme vise nije moguce koristiti
                }
        }

        private class pritisnutEnter implements KeyListener{ //Pritisnut taster enter, salje poruku
                public void keyTyped(KeyEvent e) {
            }
                public void keyReleased(KeyEvent e) {
            }
                public void keyPressed(KeyEvent e) {
              if(e.getKeyCode() == KeyEvent.VK_ENTER){      //Ukoliko je pritisnut taster enter
                slanjePoruke();                                 //posalji poruku
              }
            }
        }

        private class tastaturaAkcija implements ActionListener{//Ukoliko je pritisnut neki taster na virtuelnoj
tastaturi
                public void actionPerformed(ActionEvent d){
                        String text=slanje.getText();           //Preuzmi tekst iz polja za slanje
                        text=text+(((Button)d.getSource()).getLabel().charAt(0));//Dodaj pritisnut taster
                        slanje.setText(text);                   //Postavi tekst polja za slanje
                        }
        }
```

```
//-------------------------------Posebna programska nit koja cita dolazne podatke-------------------------------
        private static class Citac extends Thread{    //Programska nit koja cita dolazne podatke
                private String text;
                public Citac(){}
                public void run(){
                        try{
                                while(true){
                                        if(in.ready()) primanjePoruke();//Ukoliko je ulaz spreman za citanje
primi poruku
                                        sleep(100);         //Pauza 100ms
                        }}catch (Exception d) {}
                }
        }
//-------------------------------Metode za rad klijentske aplikacije-----------------------------------------------
        private void poveziSe() throws Exception{           //Metoda za povezivanje sa serverom
                sock=new Socket(ipadresa, 1234);            //Port 1234, sa unetom IP adresom servera
                in=new BufferedReader(new InputStreamReader(sock.getInputStream()));//Ulazni tok podataka
                out=new PrintWriter(new OutputStreamWriter(sock.getOutputStream()),true);//Izlazni tok
podataka
                PGP pgp=new PGP();                          //Nov objekat klase PGP
                pgp.generateRSAKeys();                      //Generisi RSA kljuceve
                dc = pgp.getD();                            //Privatni (d,n) i javni (e,n) kljucevi
                ec = pgp.getE();
                nc = pgp.getN();
                es=new BigInteger(in.readLine());           //Procitaj javni kljuc servera za prvu poruku
                ns=new BigInteger(in.readLine());
                out.println(ec);                            //Posalji javni kljuc klijenta serveru
                out.println(nc);
                new Citac().start();                        //Pokreni programsku nit za citanje podataka
                serialPort = new SerialPort(combroj);       //Definisi serijki port hardverskog generatora
slucajnih brojeva
                rasporediDugmice();                         //Slucajan raspored dugmica na tastaturi
        }
        private void rasporediDugmice() throws Exception{  //Metoda za slucajan raspored dugmica na
virtuelnoj tastaturi
                serialPort.openPort();                      //Otvori serijski port
    serialPort.setParams(9600, 8, 1, 0);                    //Podesi parametre porta
    String buffer = serialPort.readString(43);              //Procitaj 43 bita
    serialPort.closePort();                                 //Zatvori serijski port
                int m=0,k=39;
                for (int i=48;i<91;i++){                    //ASCII 48=0 - 57=9, 65=A - 90=Z
                        if(i==58)i=65;                      //Nakon broja 9, preskoci na karakter A
                        if(buffer.charAt(i-48)=='1'){       //Ukoliko je 1 u nizu bitova
                                dugmici[m].setLabel(((char)i)+"");  //Postavi dugme brojeci od pocetka
                                ++m;
                        }else{
                                dugmici[k].setLabel(((char)i)+"");  //Inace postavi dugme brojeci od kraja
                                --k;
                        }
                }
                for(int i=m;i<(m+4);i++){                   //Dodaj 4 nekoriscena dugmeta kao linije
                        dugmici[i].setLabel("-");
                }
        }
```

```
private void slanjePoruke(){                                    //Metoda za slanje poruke
                if(!slanje.getText().equals("")){               //Ukoliko nije trazeno slanje praznog reda
                        poruke.unesiPoruku(slanje.getText(),'c');//Dodaj poruku u prosle poruke, kao klijentsku
                        try {
                                sifrujPoruku(slanje.getText());//Sifruj poruku pre slanja i posalji
                        } catch (Exception e1) {e1.printStackTrace();}
                        slanje.setText("");                     //Isprazni prozor za slanje teksta
                        poruke.ispisiPoruke();                  //Ispisi poruke na prozoru sa proslim porukama
                        try{rasporediDugmice();}catch(Exception e){}//Ponovo slucajno rasporedi dugmice na
tastaturi
                }
        }
        private static void primanjePoruke() throws Exception{          //Metoda za primanje poruke
                String text = desifrujPoruku();                 //Desifruj poruku
                poruke.unesiPoruku(text,'s');                   //Unesi primljenu poruku u prozor sa proslim
porukama
                poruke.ispisiPoruke();                          //Ispisi poruke na prozoru sa proslim porukama
        }
        private void sifrujPoruku (String text) throws Exception{       //Metoda za sifrovanje poruke
                PGP pgp=new PGP();                              //Novi objekat klase PGP
                pgp.generateRSAKeys();                          //Generisi nove RSA kljuceve
                dc = pgp.getD();                                //Privatni i javni kljucevi
                ec = pgp.getE();
                nc = pgp.getN();
                out.println(ec);                                //Posalji javni RSA kljuc klijenta serveru
                out.println(nc);
                String sha = pgp.getSHA(text)+"zxy";            //SHA hash funkcija nad porukom, sa dodatim
tekstom iza
                text=text+sha;                                  //Sjedini poruku, njen hash i dodatak
                String kljuc="";                        //Procitaj kljuc sa harverskog generatora slucajnih brojeva
    try {
      serialPort.openPort();                                    //Otvori serijski port
      serialPort.setParams(9600, 8, 1, 0);                      //Postavi parametre serijskog porta
      kljuc = serialPort.readString(128);       //Procitaj 128 bitova sa harverskog generatora slucajnih brojeva
      serialPort.closePort();                                   //Zatvori serijski port
    }catch (SerialPortException ex) {System.out.println(ex);}
    int g[]=new int[128];                                       //Ceo broj g je niz od 128 celih brojeva

    byte[] ses=new byte[16];                                    //Niz bajtova ses je 128 bitni AES kljuc
                for(int i=0;i<kljuc.length();i++){              //Procitaj 1 bit
                        char znak = kljuc.charAt(i);
                        if(znak=='1') g[i]=1;                   //Ukoliko je znak 1, upisi bit 1
                        if(znak=='0') g[i]=0;                   //Ukoliko je znak 0, upisi bit 0
                }
                for (int i=0;i<16;i++){                         //Napravi 16 bajtova
                        ses[i]=0;                               //Inicijalizuj i upisi jedan bajt u promenljivu int
                        ses[i]=(byte)(g[i*8]<<7 | g[i*8+1]<<6 | g[i*8+2]<<5 | g[i*8+3]<<4 | g[i*8+4]<<3 |
g[i*8+5]<<2 | g[i*8+6]<<1 | g[i*8+7]);
                }
                String cipher = pgp.kodirajAES(text.getBytes(), ses); //Sifrovana poruka AES algoritmom
                String kljucjavni = pgp.kodirajRSA(ses, ns, es);//Sifrovani AES kljuc javnim RSA kljucem servera
                cipher=cipher+"×bj"+kljucjavni;                 //Sifrat=sifrat poruke+string xbj+sifrovan aes kljuc
                out.println(cipher);                            //Posalji zasticenu poruku serveru
        }
```

```
private static String desifrujPoruku() throws Exception{        //Metoda za desifrovanje primljenih podataka
                PGP pgp=new PGP();                              //Novi objekat klase PGP
                es=new BigInteger(in.readLine());               //Procitaj javni kljuc servera koji se koristi za
naredno sifrovanje
                ns=new BigInteger(in.readLine());

                String cipher=in.readLine();                    //Procitaj sifriranu poruku
                String[] split = cipher.split("×bj");           //Podeli poruku oko odatog stringa xbj
                String msg=split[0];                            //Prvi deo je sifrirana poruka
                String kljuc=split[1];                          //Drugi deo je njen sifriran AES kljuc

                byte[] ses = pgp.dekodirajRSA(kljuc.getBytes(), nc, dc);//Desifruj kljuc privatnim kljucem klijenta
                String poruka=new String(pgp.dekodirajAES(msg.getBytes(), ses));//Desifruj poruku primljenim
AES kljucem

                String sha=poruka.substring(poruka.length()-43,poruka.length()-3); //Izdvoji SHA hash od
desifrovane poruke
                poruka=poruka.substring(0,poruka.length()-43);//Izdvoji samu izvornu poruku

                String sha2 = pgp.getSHA(poruka);               //Izracunaj SHA hash od primljene poruke
                if (sha.equals(sha2)){                          //Ukoliko su primljeni i izracunati hash identicni
                        return poruka;                          //Ispisi poruku
                }else {
                        return "Poruka je modifikovana!";  //U suprotnom ispisi da je poruka modifikovana
                }
        }


        public static void main(String[] args) {//Main metoda startuje program (poziva konstruktor klase Client)
                try {
                        new Client();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Kod 12: Programski kod klijentskog čet programa napisan u programskom jeziku Java

Ključna razlika između klijentskog i serverskog čet programa je u delu metode “poveziSe()”, gde je prvi red metode zamenjen sa sledećim kodom (kod 13)

```
                ServerSocket ss=new  ServerSocket(1234);
                Socket sock = ss.accept();
```

Kod 13: Razlika serverskog u odnosu na klijentski čet program

Postoji još par sitnijih izmena u pogledu interfejsa i boja primljenih poruka, koje nisu od velikog značaja i neće biti dodatno opisane.

6. ZAKLJUČAK

U radu je prikazan način konstrukcije jednog tipa hardverskog generatora slučajnih brojeva, kvantifikovana je prednost takvog generatora u odnosu na tipičan pseudoslučajni generator korišćenjem različitih statističkih testova, i pokazana je praktična primena i značaj ovakvog uređaja na primeru često korišćenog tipa komunikacije preko interneta. Prednosti ovakvog hardverskog generatora slučajnih brojeva su brojne. Dobijaju se odlični rezultati na statističkim testovima (koji opisuju kvalitet izlaza generatora slučajnih brojeva), uporedivi sa trenutno najboljim hardverskim generatorima slučajnih brojeva. Takvi rezultati su samo deo prednosti hardverskog generatora, dok je najznačajnija prednost u potpuno nedeterminističkom načinu dobijanja slučajnih brojeva. Tako dobijene sekvence slučajnih brojeva nije moguće ponoviti za razliku od sekvenci dobijenih sa pseudoslučajnih generatora. Još jedna prednost, u odnosu na pseudoslučajne, i neke hardverske generatore slučajnih brojeva, je nemogućnost promene rada i načina generisanja slučajnih brojeva, bilo modifikovanjem koda preko USB porta, ili spoljnim uticajima. Uređaj samo šalje podatke USB vezom, temperaturno je kompenzovan, i zaštićen je od spoljnih uticaja modulacije ulaznog napona ili indukovanja smetnji radio talasima. Samo fizički pristup uređaju bi obezbedio napadaču uvid u generisane sekvence slučajnih brojeva, što zahteva prisustvo napadača za istim računarom. Programi koji koriste ovakav uređaj u svom radu imaju prednost ekskluzivnog pristupa serijskom portu računara, tako da ostalim programima nije omogućen istovremeni uvid u generisane brojeve.

U mane opisanog hardverskog generatora slučajnih brojeva spadaju osobine koje nisu značajne za sam kvalitet dobijenih slučajnih sekvenci. Brzina generisanja slučajnih brojeva je dovoljna za kriptografske primene, međutim vrlo je mala za određene primene koje zahtevaju velike nizove slučajnih brojeva. Ujedno bi to bilo najveće poboljšanje uređaja koji bi sa bržom A/D konverzijom imao višestruko veću brzinu generisanja slučajnih brojeva. Drugu manu predstavlja sam dizajn uređaja u DIP tehnoligiji, koja ga čini krupnim. To bi se lako rešilo prelaskom na SMD tehnologiju izrade pomoću koje bi ceo uređaj bio dimenzija tipične USB memorije, a samim tim pogodniji za transport i često korišćenje.

Sa stanovištva kriptografije, posedovanje opisanog hardverskog generatora slučajnih brojeva predstavlja značajan alat za generisanje ključeva, lozinki, ili nekih ređe korišćenih tehnika kao što je opisani slučajni raspored tastera na virtuelnoj tastaturi. Sem direktnog generisanja ključeva, primena ovakvog generatora je značajna i u generisanju inicijalizacionih vektora kod simetričnih algoritama, dodavanju slučajnih sekvenci na šifre, slučajnim k vrednostima koje se koriste u digitalnom potpisivanju, itd... Zaštita lokalnih podataka, šifrovan prenos datoteka, poruka, glasa ili slike preko interneta, su samo neke od najčešće korišćenih primena gde bi ovakav tip generatora slučajnih brojeva bio pogodan i poželjan za maksimalnu zaštitu.

Opisani hardverski generator slučajnih brojeva pokazuje da je moguće dobiti kvalitetan generator slučajnih brojeva korišćenjem široko dostupnih elektronskih komponenti, kao i da takav uređaj ima širok spektar primene. Razlike u odnosu na druge hardverske generatore predstavljaju korišćenje temperaturne kontrole, korišćenje neuniformnog vremenskog odabiranja vrednosti sa A/D konvertora, kao i precizno podešavanje srednje vrednosti pre korišćenja softverske tehnike smanjenja međusobne korelacije odbiraka. Detaljnijim opisivanjem najčešće korišćenih statističkih metoda za analizu sekvenci slučajnih brojeva, kao i vizuelnom prezentacijom rezultata, pokazan je način kvantifikovanja kvaliteta dobijenih slučajnih sekvenci u popularnim programskim paketima. Praktična primena šifrovanja podataka prikazana na primeru čet programa, pokazuje mogućnost jake zaštite svake poruke, kao i tehniku zaštite slučajno generisanom virtuelnom tastaturom.

7. REFERENCE

[1] - Donald E. Knuth (1968): „The Art of Computer Programming“, Addison-Wesley, Boston.

[2] – Generator slučajnih brojeva na bazi atmosferskog radio šuma - https://www.random.org/

[3] – Generator slučajnih brojeva na bazi radioaktivnog raspada- https://www.fourmilab.ch/hotbits/

[4] – Intel hardverski generator slučajnih brojeva - http://electronicdesign.com/learning-resources/understanding-intels-ivy-bridge-random-number-generator

[5] - Vergers C. (1987): „Handbook of Electrical Noise“, TAB Books, Blue Ridge Summit, PA.

[6] – Ott H. (1976): „Noise Reduction Techniques in Electronic Systems“, John Wiley & Sons

[7] - Gray P., D. DeWitt, A Boothroyd, J. Gibbons (1964) – „Physical Electronics and Circuit Models of Transistors“, John Wiley and Sons.

[8] – Dublin Institute of technology – Noise Sources - http://educypedia.karadimov.info/library/1NoiseSources.pdf

[9] – 2N2222A - http://www.onsemi.com/pub_link/Collateral/2N2222A-D.PDF

[10] – LM2904N - http://www.ti.com/lit/ds/symlink/lm2904.pdf

[11] – MC34063A - http://www.onsemi.com/pub_link/Collateral/MC34063A-D.PDF

[12] – LM7815 - http://www.ti.com/lit/ds/symlink/lm7805c.pdf

[13] – LF50 - http://www.st.com/web/en/resource/technical/document/datasheet/CD00000546.pdf

[14] – IRF740 - http://www.vishay.com/docs/91054/91054.pdf

[15] – DS18S20 - http://datasheets.maximintegrated.com/en/ds/DS18S20.pdf

[16] – PIC16F819 - http://ww1.microchip.com/downloads/en/DeviceDoc/39598F.pdf

[17] – FTDI232RL - http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf

[18] – John Von Neumann – Whitening Algorithm - http://www.futurebeacon.com/encrypt0.htm

[19] – PHP rand() - http://boallen.com/random-numbers.html

[20] – Monte Carlo Pi - http://mathfaculty.fullerton.edu/mathews/n2003/MonteCarloPiMod.html

[21] – Chi-square test - http://www2.lv.psu.edu/jxm57/irp/chisquar.html

[22] – Random walks - http://www.science.oregonstate.edu/~rubin/INSTANCES/WebModules/5_RandomWalk/RandomWalkFiles/Pdfs/RandomWalk.pdf

[23] – ENT – http://www.fourmilab.ch/random/

[24] – Dieharder – https://www.phy.duke.edu/~rgb/General/dieharder.php

[25] – Dieharder Test Descriptions - https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions#T000

[26] – William Stallings (2014): „Osnove Bezbednosti Mreža”, CET, Beograd.