Implementasi Ring Oscillator sebagai Seed Pseudo Random Bit Generator Berbasis Chaotic Logistic Map

Dedy Septono
Lembaga Sandi Negara,
Jakarta, Indonesia
dedy.septono@lemsaneg.go.id

Rachmadiar Prima Sastyo
Sekolah Tinggi Sandi Negara
Bogor, Indonesia
rachmadiar.prima@student.stsn-nci.ac.id

***Abstrak*—Keamanan kunci dalam kriptografi harus dijaga karena keamanan algoritma kriptografi modern tidak bergantung pada kerahasiaan algoritma yang digunakan, namun bergantung pada kerahasiaan kunci yang digunakan. *Ring oscillator* adalah salah satu dari banyak cara untuk mendapatkan serangkaian kunci acak karena dapat menghasilkan sinyal yang nonperiodik dan memiliki frekuensi yang berubah-ubah. Salah satu teknik yang bisa diaplikasikan sebagai *random number generator* adalah *chaotic logistic map*. *Chaotic logistic map* merupakan salah satu metode *chaos* yang disebut *butterfly effect* karena metode *chaos* ini memiliki kepekaan terhadap *initial condition*. Tulisan ini menyajikan hasil penelitian tentang implementasi *pseudo random bit generator*, yang diterapkan pada Arduino Uno menggunakan rangkaian *ring oscillator* dengan proses *chaotic logistic map*. Untuk memastikan keacakan rangkaian kunci yang dihasilkan digunakan pengujian NIST (National Institute of Standards and Technology) 800-22 Stastistical Test Suite, yang terbukti merupakan properti kriptografi untuk menentukan kualitas dari pseudo random bit generator, sehingga dapat digunakan untuk pembangkitan kunci.**

Kata Kunci—pseudo random bit generator; ring oscillators; chaotic random bit generator; Arduino Uno

I. PENDAHULUAN

Ada berbagai ancaman dan kerentanan terhadap informasi, seperti penyadapan, modifikasi, dan lain-lain. Enkripsi adalah salah satu teknik kriptografi yang digunakan untuk memenuhi layanan kerahasiaan. Enkripsi adalah proses untuk menyembunyikan informasi sehingga isi informasi tidak dapat diketahui oleh pihak yang tidak berwenang [1]. Kunci diperlukan dalam melakukan proses enkripsi. Keamanan kunci harus diperhatikan karena keamanan algoritma kriptografi modern tidak bergantung pada kerahasiaan algoritma yang digunakan, namun bergantung pada kerahasiaan kunci yang digunakan [2].

Pentingnya kunci dalam algoritma kriptografi menyebabkan cara menghasilkan rangkaian kunci tidak bisa secara sembarangan. Keacakan memiliki peran yang sangat penting dalam proses menghasilkan rangkaian kunci untuk memastikan sistem kriptografi tidak mudah diserang oleh pihak ketiga yang tidak berkepentingan [3]. Ada berbagai cara untuk mendapatkan rangkaian kunci yang acak. Rangkaian kunci yang benar-benar acak atau *truly random number generator* (TRNG) dapat diperoleh dengan sumber fisik seperti *thermal noise*, *voice* dan lain-lain.

Salah satu sumber keacakan pada TRNG adalah osilator. Osilator dapat ditemukan dalam berbagai sistem, terutama elektronika dan optik. Osilator digunakan untuk mentranslasikan frekuensi dari sinyal informasi. Osilator memiliki berbagai tipe [4], seperti osilator Sinuoidal, osilator RC, osilator LC, osilator *Armstrong*, osilator *Hartley*, osilator *Colpits*, osilator kristal, osilator *Pierce*, dan *ring oscillator*. *Ring oscillator* adalah osilator yang dapat menghasilkan sinyal non-periodik dan frekuensinya berubah-ubah. *Ring oscillator* terdiri dari sekumpulan elemen *delay* yang salah satunya dapat terdiri dari komponen *inverter* atau gerbang logika NOT yang berjumlah ganjil dan disusun dalam rantai untuk membentuk *ring* [5]. Dalam *ring oscillator*, keluaran dari masing-masing *inverter* akan berosilasi dari logika NOT ke HIGH logika maupun sebaliknya dan keluaran *inverter* yang terakhir akan kembali ke *inverter* pertama [6].

Ring oscillator adalah jenis osilator yang banyak digunakan untuk menghasilkan rangkaian kunci yang acak [5] [6] [7] [8]. Dalam studi [6], peneliti merancang sebuah TRNG berdasarkan hasil XOR dari keluaran yang berasal dari sampling *phase jitter* pada *ring oscillator*. Dalam studi [8], peneliti merancang sebuah TRNG untuk memperbaiki penelitian dari studi [6] dengan menambahkan D flip-flop pada setiap *ring oscillator*. Dalam penelitian [5], peneliti merancang sebuah TRNG dengan menggabungkan desain yang diusulkan dalam penelitian [6] dan [8], yang diimplementasikan pada *Field Programmable Gate Array* (FPGA).

Salah satu teknik yang bisa diaplikasikan sebagai *random number generator* adalah *chaotic logistic map* [9]. Teknik *chaotic logistic map* adalah salah satu metode *chaos* yang bisa disebut *butterly effect* (Shruthi & Sheela, 2014). Hal ini karena *chaos* memiliki kepekaan terhadap keadaan semula atau *initial condition*. Teknik *chaotic logistic map* telah lulus uji keacakan dari NIST 800-22 Statistical Test Suite yang terdiri dari 15 uji statistik untuk mengidentifikasi dan mendeteksi karakteristik keluaran dari urutan bit yang diharapkan mendekati keacakan [9]. Dalam penelitian [10], peneliti memodifikasi algoritma *chaotic logistic map* dengan menggunakan fungsi XOR dan menggunakan teknik Von Neumann. Hasil pada penelitan telah lulus FIPS-140-2 *suite test* [10].

Berdasarkan latar belakang tersebut, pada penelitian ini akan membuat *pseudo random bit generator* dengan menggunakan frekuensi untuk mendapatkan *seed* berdasarkan

chaotic logistic map yang diterapkan pada Arduino Uno. Berdasarkan hasil penelitian [10] [11] [12] [13] dikatakan bahwa *chaotic logistic map* dapat diimplementasikan pada Arduino Uno. Pemilihan Arduino Uno sebagai mikrokontroler dalam penelitian ini karena memiliki *flash memory* sebesar 32 KB dan SRAM sebesar 2 KB, sehingga dapat mengakomodasi kebutuhan pemrograman untuk pengolahan *seed* dari *ring oscillator* dan *chaotic logistic map*. Arduino Uno dapat menjalankan dua fungsi *logistic map* dalam *chaotic logistic map* dengan beberapa *initial condition* dan parameter sistem yang berbeda sehingga dapat membaca pola dari fungsi *logistic map* yang digunakan [10] [12]. Selain itu Arduino Uno adalah perangkat keras yang murah dan mudah digunakan [11].

Uji keacakan dilakukan dengan menggunakan standar dari *National Institute of Standards and Technology* (NIST) 800-22 *Statistical Test Suite*. Hasil pengujian diharapkan memenuhi standar NIST 800-22 *Stastistical Test* Suite, sehingga dapat digunakan sebagai alternatif pembangkit kunci pada saat ini.

II. BACKGROUND

A. Chaotic Logistic Map

Algoritma *pseudo random bit generator* berdasarkan *chaotic logistic map* merupakan salah satu cara untuk menghasilkan bit acak [9]. *Random Bit Generator* (RBG) adalah algoritma yang digunakan untuk menghasilkan barisan bit yang secara statistik tidak bias dan independen [9]. Algoritma PRBG berdasarkan *chaotic logistic map* telah lulus uji keacakan dari NIST 800-22 *Statistical Test Suite*. Karakteristik PRBG berdasarkan *chaotic logistic map* yaitu didasarkan pada penggunaan dua fungsi *logistic map* dan memiliki kepekaan terhadap *initial condition* pada parameter *seed* maupun sistem atau λ yang digunakan [9]. Algoritma ini dimulai dari *intial condition* yang acak $(X_0, Y_0 \in 0,1)$. Berikut adalah dua fungsi *logistic map* yang digunakan di PRBG.

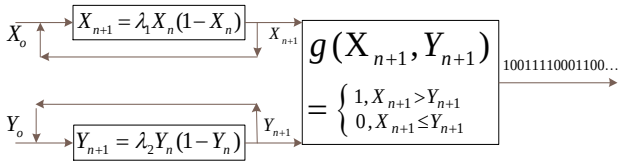
$$x_{n+1} = \lambda_1 x_n(1 - x_n \quad (1)$$

$$Y_{n+1} = \lambda_2 Y_n(1 - Y_n) \quad (2)$$

X dan Y adalah *seed* yang diperoleh dari sumber, sedangkan n adalah indeks data. Barisan bit dihasilkan dengan cara membandingkan output dari dua fungsi *logistic map*.

$$g(X_{n+1}, Y_{n+1}) = \begin{cases} 1 \; if \; X_{n+1} > Y_{n+1} \\ 0 \; if \; X_{n+1} \leq Y_{n+1} \end{cases} \quad (3)$$

PRBG berbasis *chaotic logistic map* akan menghasilkan bit yang tidak memiliki korelasi antar setiap keluaran bit (*deterministic*) ketika parameter sistem atau λ yang digunakan adalah λ = 4 [9]. Berikut ini adalah skema PRBG berbasis *chaotic logistic map*:



Gambar 1. Diagram blok skema PRBG berbasis *chaotic logistic map* [9]

B. Chaotic Random Bit Generator (CRBG)

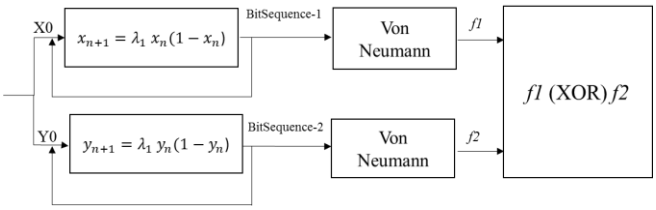
CRBG diciptakan oleh Christos Volos pada tahun 2013 [10]. Blok pertama CRBG akan menerima *intial condition*: (x_0, y_0) dengan $x_0 \neq y_0$ dan didasarkan pada dua fungsi *logistic map* seperti pada Eq.(1):

$$x_{n+1} = \lambda_1 x_n(1 - x_n) \quad (4)$$

$$y_{n+1} = \lambda_2 y_n(1 - y_n) \quad (5)$$

CRBG memiliki parameter $(r1, r2) = (\lambda_1, \lambda_2)$ sedangkan fungsi *logistic map* dimulai dari *initial condition* yang acak dan independen: (x_0, y_0) dengan $x_0 \neq y_0$. Jadi blok pertama CRBG yang diusulkan terdiri dari dua persamaan yaitu Eq.(4) dan Eq.(5). Dengan menggunakan Eq.(4) dan Eq.(5) akan menghasilkan barisan bit yang berbeda (b_1, b_2).

Blok kedua CRBG menggunakan teknik Von Neumann yang bertujuan untuk membuat bit yang tidak bias tanpa korelasi dari setiap keluaran fungsi *chaotic logistic map* [14], meningkatkan kompleksitas dan stabilitas *generator* karena menangani distribusi barisan bit secara merata [15]. Proses itu mengubah pasangan bit yang tidak tumpang tindih menjadi keluaran satu bit dengan mengubah pasangan bit, jika pasangan bit "01" diubah menjadi keluaran "0" dan pasangan bit "10" diubah menjadi keluaran "1", sedangkan pasangan bit "11" dan "00" adalah barisan bit yang akan dihilangkan. Blok ketiga dari CRBG menghasilkan barisan bit dengan menggunakan fungsi XOR dari keluaran bit yang berasal dari blok kedua. Hal ini ditunjukkan pada Gambar. 2.

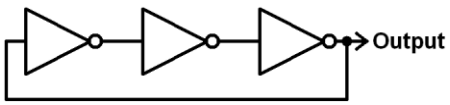


Gambar. 2. Diagram blok skema *chaotic random bit generator* [10]

C. Ring Oscillator

Osilator adalah perangkat yang merupakan kombinasi dari elemen aktif dan pasif untuk menghasilkan bentuk gelombang sinusoidal yang amplitudonya bervariasi secara berkala dengan waktu. Osilator mengubah daya arus (DC) menjadi arus *alternating current* (AC).

Ring oscillator adalah perangkat yang terdiri dari elemen *delay* yang salahtunya dapat dibentuk dengan gerbang NOT atau *inverter* berjumlah ganjil, dimana keluaran dari setiap *inverter* akan berosilasi antara dua tingkat tegangan [16]. Gerbang NOT atau *inverter* yang terletak pada keluaran *inverter* terakhir akan dimasukkan kembali ke *inverter* awal.



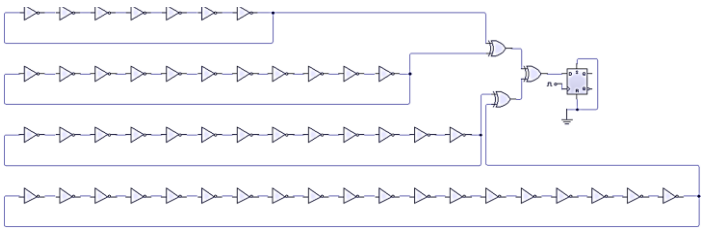
Gambar 3. *Ring Oscillator* yang dibentuk oleh 3 *inverter*

Bila *inverter* menghitung logika NOT dari suatu *input*, maka *output* terakhir dari rangkaian *inverter* berjumlah ganjil adalah logika NOT dari input awal. Hasilnya menjelaskan bahwa waktu *finite amount* setelah *input* pertama akan diproses dan *feeback*

dari keluaran terakhir menjadi input pada *inverter* pertama yang menyebabkan osilasi [17].

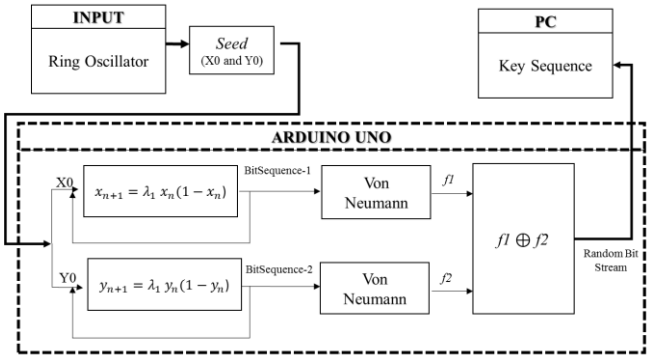
III. PROPOSED DESIGN

Pada bagian ini, dijelaskan mengenai *pseudo random bit generator* dengan *ring oscillator* yang digunakan sebagai *seed*. Pada rangkaian *ring oscillator* digunakan 4 RO dengan 7, 11, 13, 19 inverter pada masing – masing RO (IC 4069), XOR dan D flip-flop untuk *sampling* pada setiap *output* dari *ring oscillator* [18]. Perbedaan dalam jumlah *gate inverter* digunakan untuk menghasilkan perbedaan besar dalam frekuensi antara masing-masing RO [19]. Rangkaian *ring oscillator* ditunjukkan pada Gambar. 4.



Gambar 4. Desain *ring oscillator* yang dibentuk oleh 4 RO

Blok diagram dari rangkaian yang diusulkan untuk membuat *pseudo random bit enerator* ditampilam pada Gambar. 5.



Gambar 5. Diagram blok skematik dari *pseudo random bit generator*

Ring oscillator akan menghasilkan nilai analog yang berbeda. Nilai tegangan akan dibaca menggunakan fitur *analogRead ( )* yang ada pada Arduino. Hasil *analogRead ( )* dari rangkaian *ring osilator* akan digunakan sebagai *initial condition* (x_0, y_0). Nilai analog yang dihasilkan oleh rangkaian *ring oscillator* akan dikonversi ke dalam nilai *float* karena *input initial condition* (x_0, y_0) membutuhkan nilai *float*. Proses konversi nilai analog ke floating point ditunjukkan pada persamaan 6 dan 7.

$$x_0 = \text{analogRead (1)} * 500/1023 \quad (6)$$

$$y_0 = \text{analogRead (2)} * 500/1023 \quad (7)$$

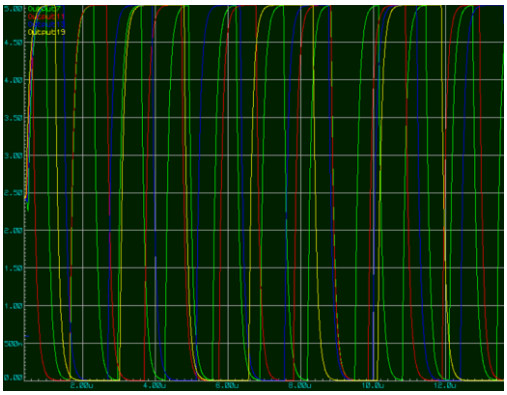
Output dari x_0 dan y_0 diproses dengan teknik CRBG. Setelah proses iterasi selesai maka akan menghasilkan barisan bit yang acak.

IV. EKSPERIMEN

A. Sinyal *Ring Oscilator*

Ring oscillator yang terdiri dari *inverter* berjumlah ganjjil yang disusun secara bertingkat, dengan *output* pada *inverter* terakhir akan kembali ke rantai inverter yang pertama. Frekuensi dari *ring oscillator* tergantung pada *time delay* inverter dan karena itu tidak dapat dikontrol secara eksternal.

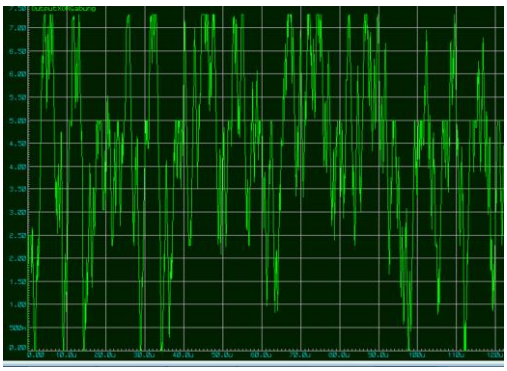
Untuk menjaga logika yang benar dari *ring oscillator*, keluaran dari setiap gerbang logika tidak dapat mendorong gerbang logika lain karena gerbang memiliki kapasitansi dan hambatan. Jika *output* dari sebuah gerbang mempengaruhi banyak komponen lainnya, gerbang tersebut akan menghadapi kapasitansi dengan beban tinggi, yang memperlambat transisi *output*, sehingga meningkatkan *delay* pada propagasi sinyal.



Waktu (Microseconds)

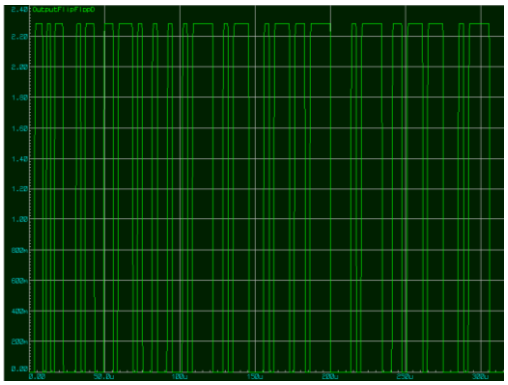
7 Stages Inverter
13 Stages Inverter
11 Stages Inverter
19 Stages Inverter

Gambar 6. Sinyal dari 4 RO dengan *inverter* 7, 11, 13, 19



Waktu (Microseconds)

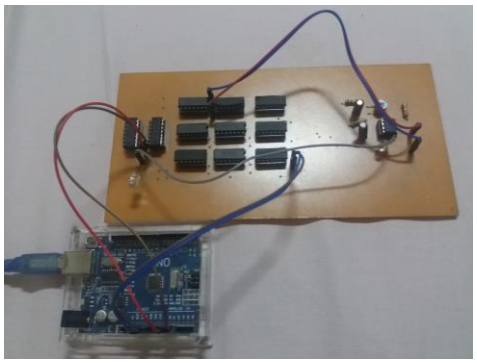
Gambar 7. Sinyal dari operasi XOR pada 4 RO



Waktu (Microseconds)

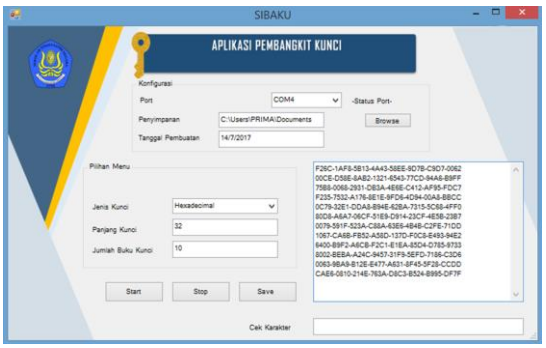
Gambar 8. Sinyal dari D Flip-Flop

Pseudo random bit generator yang diusulkan tersebut diimplementasikan pada Arduino Uno karena fleksibel dan mudah digunakan. Board ini memiliki 14 pin *input/output digital*, 6 *input analog*, osilator kristal 16 MHz, koneksi USB, *header* ICSP, dan tombol reset (Gambar 9.). Implementasi *pseudo random bit generator* pada PCB ditunjukkan pada Fig. 10.



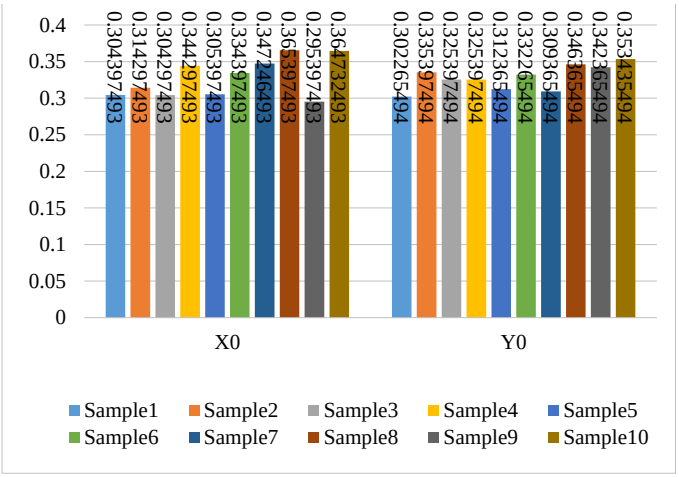
Gambar 9. Implementasi *hardware pseudo random bit generator*

Sketsa algoritma yang di *upload* pada Arduino Uno menggunakan 3.556 byte (11%) ruang penyimpanan program. *Global variable* menggunakan 194 byte (9%) *dynamic memory*, menghasilkan 1.854 byte untuk *local variable*. *Pseudo random bit generator* yang diusulkan untuk pengujian menggunakan tegangan 5 V dan *sampling frequency* 21,2 MHz. Untuk tampilan *interface* dari aplikasi menggunakan bahasa pemograman Visual Studio C#.



Gambar 10. *Interface* dari aplikasi *pseudo random bit generator*

Arduino Uno akan mengambil *seed* dari *ring oscillator*. Dalam penelitian ini, 10 sampel nilai analog yang dihasilkan oleh rangkaian *ring oscillator*, dimana setiap sampel dari nilai analog digunakan untuk membangkitkan 100000000 bit. Hal tersebut dapat lihat Gambar. 11.



Gambar 11. Nilai analog dari rangkaian *ring oscillator*

Nilai analog digunakan sebagai nilai pada *initial condition* dari algoritma *chaotic logistic map*. Keluaran rangkaian bit acak yang diimplementasikan oleh Arduino Uno akan disimpan dengan menggunakan aplikasi *interface*

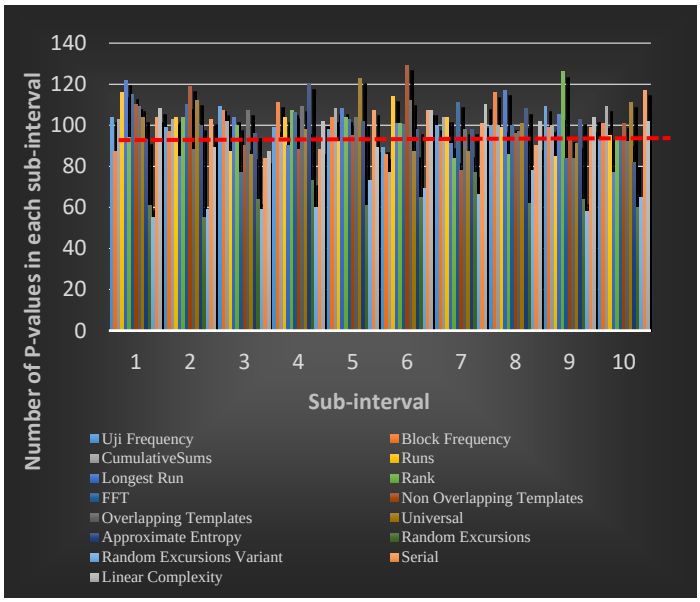
V. HASIL PENGUJIAN

A. *NIST statistical test suite*

Dalam penelitian ini 15 uji statistik dipilih untuk menguji keacakan 1000 sampel dengan urutan 1000000 bit yang dihasilkan oleh *pseudo random bit generator*. Parameter NIST 800-22 *Statistical Suite* ditunjukkan pada tabel berikut:

TABLE I. PARAMETER NIST STATISTICAL SUITE

Item	Value
Block Length of Frequency Test within a Block	128
Block Length of Approximate Entropy Test	10
Block Length of Serial Test	16
α	0.01
Length of Sequences	1000000
Total Number of Sequences	1000



Gambar 12. Diagram Distribution *P-Values*

Distribusi *P-value* pada setiap *sub-interval* yang dihasilkan dari setiap pada barisan biner yang dibangkitkan oleh *pseudo random bit generator* bersifat tidak seragam. Hal tersebut karena banyaknya *P-value* pada setiap *sub-interval* tidak mendekati banyaknya *P-value* yang diharapkan pada setiap *sub-interval,* yaitu 100. Dalam pemeriksaan distribusi *P-value,* barisan bit dianggap memiliki distribusi seragam apabila *P-value* $\geq 0{,}0001$, untuk $\alpha = 0{,}01$.

TABLE II. NIST TEST RESULT

Test Name	Test Items		Result
	P-Value	Proportion	
Frequency	0.939005	986/1000	Success
Block Frequency	0.454053	993/1000	Success
Cumulative Sums	0.587274	984/1000	Success
Runs	0.363593	986/1000	Success
Longest Run	0.033584	985/1000	Success
Rank	0.170922	995/1000	Success
FFT	0.157251	989/1000	Success
Non Overlapping Templates	0.012300	987/1000	Success
Overlapping Templates	0.486588	986/1000	Success
Universal	0.112047	986/1000	Success
Approximate Entropy	0.461612	981/1000	Success
Random Excursions	0.761596	642/642	Success
Random Excursions Variant	0.569124	642/642	Success
Serial	0.424453	988/1000	Success
Linear Complexity	0.686955	989/1000	Success

VI. KESIMPULAN

Dalam penelitian ini kami mempresentasikan tentang *pseudo random bit generator* berbaasis *chaotic logistic map* yang diterapkan pada Arduino Uno. *Pseudo random bit generator* yang diusulkan mampu memberikan bit acak. Untuk 4 jumlah *ring oscillator* dengan *inverter* 7, 11, 13, 19, dan diproses dengan teknik *chaotic random bit generator* dapat melewati semua tes NIST 800-22. Penggunaan teknik Von Neumann membuat bit yang tidak bias tanpa korelasi dari dua fungsi *logistic map* dan meningkatkan kompleksitas serta stabilitas generator karena dapat menangani distribusi rangkaian bit secara merata

DAFTAR PUSTAKA

[1] B. Scheiner, F. William, H. Feistel, and Y. Heights, *Applied Cryptography*, 2nd ed. New York: John Wiley & Sons, Inc, 1996.

[2] D. Arroyo, G. Alvarez, and V. Fernandez, "On The Inadequacy Of The Logistic Map For Cryptographic Applications," no. ArVix, pp. 1–6, 2008.

[3] D. Eastlake, J. Schiller, and S. Crocker, "RFC 4086 : Randomness Requirement for Security." 2005.

[4] S. T. Karris, *Electronic Devices and Amplifier Circuit*. Publications, Orchard, 2005.

[5] N. Bochard, F. Bernard, V. Fischer, and B. Valtchanov, "True-Randomness and Pseudo-Randomness in Ring Oscillator-Based True Random Number Generators," *Intenational J. Reconfigurable Comput.*, vol. 2010, no. Hindawi Publishing Corporation, pp. 13, 2010.

[6] B. Sunar, I. C. Society, W. J. Martin, and D. R. Stinson, "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks," in *IEEE Transactions On Computer*, 2007, vol. 56, no. 1, pp. 109–119.

[7] S. Łoza and L. Matuszeqski, "A True Random Number Generator Using Ring Oscillators and SHA-256 as Post-Processing," no. IEEE, 2014.

[8] K. Wold and C. H. Tan, "Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings," *Intenational Conf. Reconfigurable Coomputing FPGAs*, 2008.

[9] V. Patidar, K. Sud, and N. Pareek, "A Pseudo Random Bit Generator Based On Chaotic Logistic Map And Its Statistical Testing," *J. Informatical*, vol. 33, no. Informatica, pp. 441–452, 2009.

[10] C. K. Volos, "Chaotic Random Bit Generator Realized with a Microcontroller," vol. 3, no. 4, pp. 115–136, 2013.

[11] L. Acho, "A Discrete-Time Chaotic Oscillator Based On The Logistic Map : A Secure Communication Scheme And A Simple Experiment Using Arduino," *J. Franklin Inst.*, 2015.

[12] J. D. Serna and A. Joshi, "Visualizing the logistic map with a microcontroller," *IOP Sci.*, pp. 1–6, 2011.

[13] M. Zapateiro, D. Hoz, L. Acho, and Y. Vidal, "An Experimental Realization of a Chaos-Based Secure Communication Using Arduino Microcontrollers," *Sci. World J.*, no. Hindawi Publishing Corporation, p. 10, 2015.

[14] J. Von Neumann, "Various techniques used in connection with random digits", G.E. Forsythe (eds.), *Applied Mathematics Series*, National Bureau of Standards, no, 12, pp. 36-38, 1951.

[15] M. Andrei, S. Paul, S. Emil, "Optimising Ring Oscillator-based True Random Number Generators Concept on FPGA," *in International Spring Seminar on Electronics Technology (ISSE)*, 2016.

[16] V. Agrawal and C. Bandi, *Ring Oscillator Power and Frequncey Vs Voltage*. 2007.

[17] M. K. Mandal and B. C. Sarkar, "Ring oscillators : Characteristics and applications," *Indian J. Pure Appl. Phys.*, vol. 48, pp. 136–145, 2010.

[18] Wang Liao, M. Wan, Kui Dai, and Xuecheng Zour, "Scalable Truly Random Number Generator", *Proceedings of the World Congress on Engineering*, 2015.

[19] Siva Nishok Dhanuskodi, Arunkumar Vijaykumar, Sandip Kundi, "A Chaotic Ring Oscillator based Random Number Generator", *International Symposium on Hardware-Oriented Security and Trust*, 2010.