Ahmedabad University

ENR107 Digital Electronics and Microprocessors

Pseudo-Random Number Generator

Date of Submission: 24/11/2024

Course Information

Name of Project	Pseudo-Random Number Generator
Course faculty	Prof. Sanket Patel
Group Number	7

Student Details

Roll No.	Name of the Student	Name of the Program	Contact	Photograph
AU2340014	Manushree Patel	B.Tech (CSE)	8488078586	
AU2340015	Hetvi Raval	B.Tech (CSE)	9879427624	
AU2340059	Renee Vora	B.Tech (CSE)	9327455234	

Roll No.	Name of the Student	Name of the Program	Contact	Photograph
AU2340082	Pushti Sonak	B.Tech (CSE)	8780645461	
AU234116	Yashvi Modi	B.Tech (CSE)	7622076390	

<u>Table of Contents:</u>

ACKNOWLEDGEMENT

We, Group-7, would like to extend our heartfelt gratitude to Prof. Sanket Patel for his invaluable guidance, mentorship, and for granting us the opportunity to undertake this project on the Pseudo Random Number Generator Using LCG. His constructive feedback and insightful suggestions were instrumental in the successful implementation and refinement of our project ideas.

We are also deeply appreciative of the Teaching Assistant of the course in the laboratory session for their unwavering support in addressing our queries, providing valuable insights, and assisting us in conceptualizing and designing the project efficiently within the stipulated timeline.

Thank you,

Group-7

❖ INTRODUCTION:

This project report presents the concept of Pseudo Random Number Generator, commonly referred to as PRNGs. They are algorithms used to produce sequences of numbers that majorly satisfy properties of random numbers. Mathematically, the output can be obtained, as the algorithms can be expressed as equations that contain an initial value known as seed. Their effectiveness depends on factors such as periodicity and reproducibility factor mindful selection of parameters like seed, modulus, multiplier and the algorithmic formula. These algorithms are pseudo as they are not completely random and are predictable after a certain extent. Due to this reason, they are not so efficient for high-security based applications. They are classified into three main genres which are Linear Congruential Generator based (LCGs), Linear Feedback Shift Register based (LFSRs) and Cellular Automata based (CAs). Other than these three types, they are categorized as Non-linear algorithms or cryptographic PRNGs. Here in, we have tried to implement a version of LCG that is a model having operations that generate the most random numbers (according to our research). It comprises 8 LCGs, computed with different combinations like scaling, XOR operation, series and parallel connections along with the usage of computed RAM's GB value, current time etc. This model is implemented in MATLAB.

❖ MOTIVATION:

- Pseudo-random generators have extensive real-time applications across various domains like statistical modeling, cryptography and simulations. Their hardware implementation is suitable for embedded systems that are required to be energy efficient and faster.
- This concept revolves around digital logic design, microelectronics, thus, giving us a hands-on understanding of those basics and their hardware level integration on generation of random numbers.
- Its implementation aims to provide flexibility to observe variations in the generated sequences with changing parameters (modulus, seed, multiplier) and thus, understanding their impact on the randomness and predictability.

- This project involves exploration of circuit design, deterministic algorithms, efficient memory management, handling of large numbers and obtaining the expected output accurately, giving us experience in hardware software integration.
- Overall, this project would lay the foundation for advanced techniques of generating random numbers that have to be integrated in broader domains.

❖ OBJECTIVE:

To generate randomness by leveraging real-time data from a microprocessor's internal memory and system parameters. Using our device, such as a laptop, as the microprocessor, we aim to access real-time data including memory usage, CPU load, and date-time values to serve as seed values and scaling factors. The objective is to implement Pseudo-Random Number Generation (PRNG) by integrating MATLAB with real-time system data. Additionally, we aim to arrange Linear Congruential Generators (LCGs) in specific configurations and analyze the resulting data to evaluate randomness produced under varying conditions.

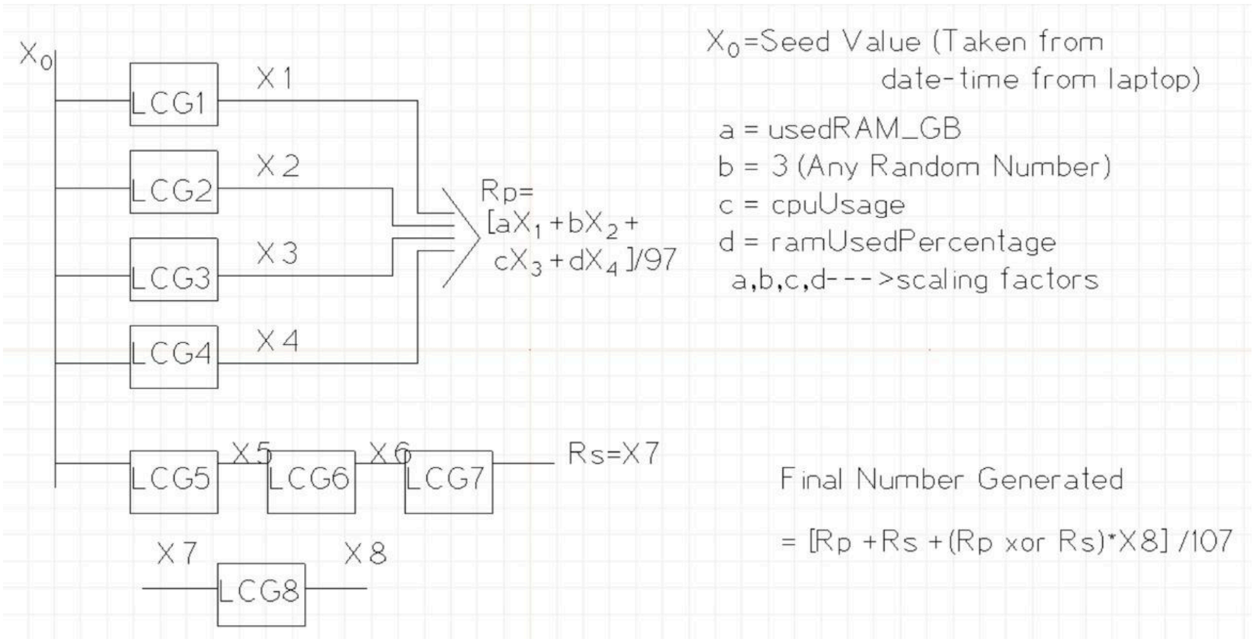
❖ COMPONENTS:

No physical components have been used as we have used our laptops as the microprocessor unit.

❖ TOTAL COST:

The total cost of making our project is zero. We have implemented it in MATLAB. It was more of a research oriented project.

❖ CIRCUIT SCHEMATIC:


X_0
LCG1
X1
LCG2
X2
LCG3
X3
LCG4
X4
Rp= $[aX_1 + bX_2 + cX_3 + dX_4]/97$
X_0=Seed Value (Taken from date-time from laptop)
a = usedRAM_GB
b = 3 (Any Random Number)
c = cpuUsage
d = ramUsedPercentage
a,b,c,d--->scaling factors
LCG5
X5
LCG6
X6
LCG7
Rs=X7
Final Number Generated
= [Rp +Rs +(Rp xor Rs)*X8] /107
X7
LCG8
X8

```
%RAM

[~, sysInfo] = memory;

totalRAM_GB = sysInfo.PhysicalMemory.Total / 1e9;

usedRAM_GB = (sysInfo.PhysicalMemory.Total - sysInfo.PhysicalMemory.Available) / 1e9;

ramUsedPercentage = (usedRAM_GB / totalRAM_GB) * 100; % As a percentage

fprintf('RAM memory %% used: %.2f%%\n', ramUsedPercentage);

fprintf('RAM Used (GB): %.2f GB\n', usedRAM_GB);




%CPU

% Get the system CPU usage

if isunix

    % On Unix-like systems, use the uptime command to fetch load averages

    [~, loadavg] = system('uptime');
    % Parse the load averages from the uptime command output
    loadValues = regexp(loadavg, 'load averages?:\s*([\d.]+),\s*([\d.]+),\s*([\d.]+)', 'tokens');

    if isempty(loadValues)
        error('Unable to retrieve load averages. Ensure the uptime command works on your
system.');
    end

    % Extract the 1-minute, 5-minute, and 15-minute load averages
    load1 = str2double(loadValues{1}{1});
    load5 = str2double(loadValues{1}{2});
    load15 = str2double(loadValues{1}{3});

    % Get the number of CPU cores
    cpuCount = feature('numcores');

    % Calculate CPU usage based on the 15-minute load average
    cpuUsage = (load15 / cpuCount) * 100;

elseif ispc
    % On Windows, use WMIC to fetch CPU usage
    [~, output] = system('wmic cpu get loadpercentage');
    % Clean up the output
    output = strtrim(output);  % Remove leading/trailing whitespace
    lines = strsplit(output, '\n');  % Split output by lines

    % Check if there are valid data lines
    if length(lines) >= 2
        % Extract the CPU load percentage from the second line (after the header)
        cpuUsage = str2double(strtrim(lines{2}));
    else
        error('Unable to extract CPU usage from WMIC output.');
    end

else
    error('Unsupported operating system. This script supports Unix-like systems and Windows.');
end

% Display CPU usage
fprintf('The CPU usage is: %.2f%%\n', cpuUsage);

% Generate random numbers using the enhanced LCG model

function R_sequence = enhanced_lcg_model(N,cpuUsage, usedRAM_GB, ramUsedPercentage)
    % Generate N random numbers
    % Dynamic seeding based on the current time
    X0 = mod(floor(now * 1e7), 100); % Initial seed
    R_sequence = zeros(1, N); % Preallocate array for speed

    for i = 1:N
        % Generate a random number using the enhanced model
        R_sequence(i) = enhanced_lcg(X0,cpuUsage, usedRAM_GB, ramUsedPercentage);
        % Update the seed for the next iteration
        X0 = R_sequence(i);
    end

    % Display the generated sequence
    disp('Generated Random Numbers:');
    disp(R_sequence);
end




% LCG Function
function X_next = lcg(X, a, c, m)
    X_next = mod(a * X + c, m);
end

% Enhanced LCG Model
```

```
function R_final = enhanced_lcg(X0,cpuUsage, usedRAM_GB, ramUsedPercentage)

  % Use cpuUsage, usedRAM_GB, and ramUsedPercentage to generate random numbers


  % Ensure all inputs are integers within a reasonable range

  % Keep within modulus range for LCG


  % Independent LCGs (LCG1 - LCG4 for parallel outputs)

  X1 = lcg(X0, 13, 7, 97);

  X2 = lcg(X0, 17, 11, 89);

  X3 = lcg(X0, 19, 3, 83);

  X4 = lcg(X0, 23, 5, 79);


  % Parallel combination (weighted by RAM usage)

  R_parallel = mod(usedRAM_GB*X1 + 3*X2 + cpuUsage*X3 + ramUsedPercentage*X4, 97);


  % Series Chain (LCG5 → LCG6 → LCG7)

  X5 = lcg(X0, 29, 2, 73);

  X6 = lcg(X5, 31, 9, 67);

  X7 = lcg(X6, 37, 13, 61);

  R_series = X7;


  % Feedback loop: LCG8

  X8 = lcg(X7, 41, 15, 59);
```

```
    % Convert to integer for bitwise operations

    R_parallel_int = uint32(R_parallel);

    R_series_int = uint32(R_series);

    X8_int = uint32(X8);


    % Final Mixing (combining all outputs with bitxor)

    R_final = mod(R_parallel_int + R_series_int + bitxor(R_parallel_int, R_series_int) * X8_int,
107);

end


R = enhanced_lcg_model(5,cpuUsage, usedRAM_GB, ramUsedPercentage); % Generate 10
random numbers
```

- ❖ <u>WORKING AND EXPLANATION:</u>

  <u>Linear Congruential Generator:</u>

  It is one of the widely used algorithms, for the generation of pseudo-random numbers, where the numbers are generated using thee formula: $Xn + 1 = (aXn + c)\ mod\ m$

  Variables and their meaning:

  - Xn : the current number in the sequence; if it is the first number than it is often referred to as 'seed'
  - a : a multiplier, that scales the current number
  - c : increment added to the result obtained after multiplication
  - Xn+1: The next number in the sequence
  - m : It is the modulus operator that gives the remainder. (This makes sure that the sequence result falls in the range of 0-(m-1))

Function of modulus: $(aXn + c)$ -> this might be a large number. Its operation with mod m restricts its range from 0 to (m-1).
This function is very important as it keeps the generated numbers manageable and it does not let it grow uncontrollably after every iteration.
This algorithm is like a finite state machine with a finite number of states, i.e., the sequence repeats after a certain number of numbers have been generated.

Thus, the sequence numbers generated largely depends upon the selection of parameters a, c, m.
The maximum possible period of this generator is m, and is achieved on the basis of the following conditions:

- ‘c’ is co-prime with ‘m’
- If (a-1) is the multiple of 4, then ‘m’ is also a multiple of 4.
- (a-1) should be divisible by all prime factors of ‘m’.

Special case-
If c=0, then numbers are generated faster and the algorithm is called Multiplicative Congruential Generator.

Working Explanation of Project:

We have used our laptop as the microprocessor unit to extract the random initial seed values directly from the CPU. The working of the project is divided into various modules for the better access for the feedback path and also for the easier understanding for us. The various modules are as follows:

1. The RAM module: This module is fetching the RAM data for both the total and available. The used RAM data is calculated by subtracting the available from the total and then it is converted to gigabytes (for generating upto two decimals.) This data is extracted from the system info function in matlab.

2. The CPU load module: This module is further divided into two parts for calculating the average for 3 different load times at 1,5 and 15 minutes respectively and also for the overall load percentage. This is done to get two values as the scaling factors which are later used during the calculation of Rparallel. CPU load is known as the number of processes being carried out in the CPU at that time along with the processes that are waiting to be executed in line. This is calculated for an average of 3 times(1,5 and 15 minutes) as mentioned above which is then divided by the total number of CPU blocks in the computer. For the percentage calculation, the answer of this division is converted into percentage by using only the average of 15 minutes of load time from the CPU.

3. The Initial seed value: The initial seed value is calculated from the "now" function of the time input whose mod is taken out of 100 to take the last two digits which are of seconds to increase the randomness. This is stored as the initial seed value X0 which is further fed into various streams of parallel and series calculation.

4. The LCG function value: The LCG function is used 8 times, 4 of which are used for parallel then the answers of each of those will be scaled and added then again the mod is taken with 97 (a prime number to increase randomness, to cover the whole range and also to get two digit random number). Similarly, the other 4 LCGs were used in calculating Rseries and then the answer given out is again feedback with the scaled function to get X8.

5. The Final Random Number: Calculation of the final random number is a function of sum of Rparallel, Rseries and the XOR of Rparallel and Rseries which was scaled with X8 (XOR function is used because it increases randomness as first the number is converted into bits then the XOR function is carried out and then again it is converted into decimal.)

❖ <u>REAL-LIFE APPLICATION:</u>

Real life applications:

Pseudo random number generator applies in real life for posterior inference along with prediction and posterior predictive checks. Pure data simulation can also be done which is similar to posterior predictive check without any conditions.

Some other applications are as follows:

- Cryptography: The digital encryption of passwords from different browsers and sites uses random numbers for storing their digital data.
- Cryptocurrency wallets: The random seed values calculated by encryption keys are used for cryptocurrency wallets generated by BIO39 standard algorithms.
- Simulations: These sequences generated by the random number generator are used to test the Monte Carlo Simulations for unknown ratio and area sampling.
- Machine Learning and Computing: The learning frameworks in real life applications like domain randomization includes random numbers in various areas such as game development, graphics, etc.
- Mathematical and scientific studies: The random probability samples from different statistical models like population and various non probabilistic models using direct analysis makes the use of inputs from random number generators.

❖ <u>SUMMARY:</u>

The project we have developed is linear congruential based (aXn +c , mod m) pseudo random number generator that can generate 1 to n random numbers based on the input you want, along with usage of concepts like feedback, Hull Dobell theorem for LCG values. The specifications of the project includes generation of our own model by scaling, connection in parallel series of LCGs with different values, using XOR to optimize randomness. Also, the input seed for the very first number is not pre assigned we have fetched the current time and used the seconds of the time as the seed. About the complexity of making the project real-time and not through the pre-assigned values we have scaled the inputs with random values. The complexity was in the key objective of the project where input to scale was done from the current Data of the CPU(Central Processing Unit) of the laptop that is executing the code for random number generation.

The data extracted from CPU was average of CPU usage (load) at 1, 5 and 15 minutes and the RAM access memory both in percentage and also the one with precision up to 5 decimal places fetching the last 2 decimal places from the operating system which is also a random data that can be provided. The process of the generation starts with the initial seed generated using now() function for the first number generation then the first number generated using the model is fed back as the seed for the second number if needed to be generated. The configuration of the model is such that the first four LCG function called is connected in parallel (having same seed as now() function X0) and then scaled and added it, the next four LCGs are connected in series and then the final random number of parallel is connected with XOR to series (till the first 3 series LCGs) random number output. Then the last series LCG output is multiplied by the XOR output of the series and parallel LCG which gives the final output. That final output is given as the seed for the next number generation. The challenges faced were in fulfilling the key objectives of generating random seed and values to scale as the software Open Hardware Monitor was providing the real time data but we were not able to fetch it and was not supported by Matlab. Other than that we also searched for other softwares such as BIOS for CPU's temperature, and acknowledged about intel software for the data but were not able to access it due to security reasons. Overall we were able to successfully implement the project by using an inbuilt function using the operating system itself that we were able to acknowledge from a coding website for python by converting it for matlab code.

❖ CONCLUSION:

This project demonstrates the generation of random numbers using Linear Congruential Generator (LCG) algorithm in MATLAB. Using LCG's simplicity and efficiency, we were able to generate a sequence of random numbers using real time system parameters as dynamic inputs. Typically, LCG is used for generating pseudo random number generation, but it can generate predictable outputs unless designed carefully and keeping in mind conditions for LCG's parameters like multiplier, increment and modulus. In this project the LCG is improved by adjusting the seed and input values using real time data of RAM usage, CPU load and clock time seconds. This introduced a good amount of variability, making the output dependent on the system's state at current time and making the random number more unpredictable. By collecting CPU load average and RAM usage of the system, resource consumption of the system becomes an important factor in influencing randomness. This project capitalizes dynamic seeding, ensuring each run produces a unique starting point, ensuring least predictability. Moreover, the model uses multiple LCGs with half connected in parallel, with the outputs scaled with CPU load and RAM usage, and half in series. Further, refining this LCG output using feedback loops and bitwise XOR operations, enhanced the randomness of generated numbers, making it more unpredictable. This model shows how it is possible to link the random number generation to the microprocessor's functionality and the system load, thus showing an example of a model that has proved useful in repetitional domains such as cryptography, simulation as well as statistical modeling where random numbers are very central. The approach described here offers a possible way to make a random number generation depend on the current state of affairs in the system and become much less likely to repeatedly provide predictable numbers, thus giving a more efficient solution for a number of computer computations.

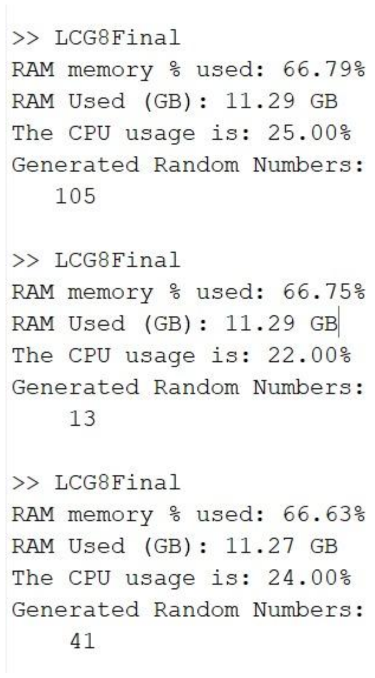
❖ FUTURE PLANS:

- Our future plans include integrating different models other than the LCG for our modules of series and parallel to increase the randomness.
- Another additional part that we would like to explore more about is how to take input from the exterior hardware microprocessor such as arduino or raspberry pi.
- We are also planning to write a research paper extending our knowledge in this domain by integrating the above mentioned parts and also adding our own parts like in this project we explored about the XOR function to increase the randomness.

❖ CONTRIBUTION:

1. Manushree Patel: Research, Randomness Analysis, Documentation
2. Hetvi Raval: Research, Algorithm ideation, Testing and validation
3. Renee Vora: Research, Algorithm ideation, MATLAB implementation
4. Pushti Sonak: Research, Parameters Research, Documentation
5. Yashvi Modi: Research, Testing and validation, Documentation

❖ PROJECT PHOTOS:



```
>> LCG8Final
RAM memory % used: 66.79%
RAM Used (GB): 11.29 GB
The CPU usage is: 25.00%
Generated Random Numbers:
   105

>> LCG8Final
RAM memory % used: 66.75%
RAM Used (GB): 11.29 GB
The CPU usage is: 22.00%
Generated Random Numbers:
    13

>> LCG8Final
RAM memory % used: 66.63%
RAM Used (GB): 11.27 GB
The CPU usage is: 24.00%
Generated Random Numbers:
    41
```

❖ GROUP PHOTO:

❖ PROJECT DEMONSTRATION VIDEO:

https://drive.google.com/drive/folders/15LVbSmbn1KxYRCFCFIN9OBNADOEc8w2y

❖ REFERENCES:

https://mc-stan.org/docs/2_21/stan-users-guide/applications-of-pseudorandom-number-generation.html

https://arxiv.org/pdf/1811.04035

https://link.springer.com/chapter/10.1007/978-1-4757-0602-4_6

https://www.researchgate.net/post/Pseudo_random_number_generator_in_Matlab_code

https://stackoverflow.com/questions/7602919/how-do-i-generate-random-numbers-without-rand-function