CS-203 Project report

Randomized Encryption - Decryption System

Instructor: Dr. NEERAL GOEL

Contributors:

SANGANWAR SOURABH	- 2020CSB1121
SUKHMEET SINGH	- 2020CSB1129

Contents:

Introduction

Our project is a randomized encryption decryption system, and password generator using LFSR (linear feedback shift register) as a random number generator.

This project is written in verilog. It’s input could be directly given in binary format in my_file.txt and receive output in out_file.txt after running the module. Else we can use python file (main.py) to convert data in characters into binary and convert binary output into characters. Here characters are encoded into 6 bit binary numbers. Python also enables us to encrypt and decrypt files directly.

Features

Our project current has following features:

1) **Encrypt File:**

    Takes file as input and asks path to store encrypted file. Supported characters in input file are:

    a) Uppercase alphabets (A,..Z)
    b) Lowercase alphabets (a,...z)
    c) Numbers (0,...9)
    d) Underscore '_'
    e) Enter '\n'
    f) Space ' '
    g) Comma ','
    h) Fullstop '.'

2) **Decrypt File:**

    Encrypted file is given as input and output is a decrypted file.

3) **Encrypt a short message:**

    Takes data input from console, Encrypts message and prints encrypted message in console. Messages could be upto *10 character*s. Supported characters are:

    a) Uppercase alphabets (A,..Z)
    b) Lowercase alphabets (a,...z)
    c) Numbers (0,...9)
    d) Underscore '_'
    e) Blankspace ' '

4) **Decrypt a short message:**

    Encrypted message is given as input in the console and Decrypted message is received as output in console. Decrypted message consists of *13 characters.*

5) **Password Generator:**

    It generates a random password with the same characters that are supported by the encrypt data feature.

Module Hierarchy

```
TestBench
├── Solver
│   ├── Decrypter
│   │   ├── decrypt_function_1
│   │   ├── decrypt_function_2
│   │   ├── decrypt_function_3
│   │   └── decrypt_function_4
│   ├── Encrypter
│   │   ├── bit_rand_6
│   │   ├── bit_rand_11
│   │   ├── encrypt_function_1
│   │   ├── encrypt_function_2
│   │   ├── encrypt_function_3
│   │   └── encrypt_function_4
│   ├── password_gen
│   │   ├── bit_rand_6
│   │   ├── Gen_1
│   │   │   └── bit_rand_11
│   │   ├── Gen_2
│   │   │   └── bit_rand_11
│   │   ├── Gen_3
│   │   │   └── bit_rand_11
│   └─ └── Gen_4
└──             └── bit_rand_11.
```

Module Functioning

1) Test Bench:

It is the main entry point of the program.It takes the input the type of operation to be performed and the corresponding input is taken from the file *my_file.txt* . The input file contains binary encoded integers of supported characters which are written by *python* and *testbench* reads that *input file*. It writes output in *out_file.txt* which *python* reads and converts them into corresponding characters.

Modules instantiated:

- Solver

Input format from file:

- First line contains a number which is the number of operations.
- From the next line each line contains which operation to be done(*work*) followed by a line with data on which operation to be performed.
- Work equal to 0 represents encryption, the next line contains 60 bit data.
- Work equal to 1 represents decryption,the next line contains 78 bit encrypted data.
- Work equal to 2 represents password generation, there is no next line in this case.

Output file format is:

- Each output generated is written in a new line in the output file.

Test Bench also generates a *Clock signal* for working of the system.

2) Solver:

Solver gets input from the **Test bench** and sends input to **Encryptor**, **Decryptor** and **Password generator**. Based on *work* required output is selected from output received from above 3 modules.

Modules instantiated:

- Encryptor
- Decryptor
- Password generator

Input format:

● Clk (1 bit)	- It is Clock signal
● data_1_80, (60 bit)	- Data for encrypter
● Data_2_96, (78 bit)	- Data for decrypter
● Work_2, (2 bit)	- It contains which operation to be done

Output format:

● Output_1_96 (78 bit)	- Encrypted data
● Output_2_80 (60 bit)	- Decrypted data or password generated

3) Encrypter

Modules instantiated:

In encrypter 6 modules are instantiated

- 4 encryption modules : **encrypt_function_1**,**encrypt_function_2**,**encrypt_function_3** and **encrypt_function_4**.
- and 2 random number generators: **bit_rand_6** and **bit_rand_11**.

4 encryption modules take the *60 bit* data as input and generate *78 bit* encrypted data as output uniquely.
Bit_rand_6 generates a *6 bit* random number and based on this random number output is selected from 4 outputs generated by **4 encryption modules** .
Bit_rand_11 generates a *11 bit* random number and it is sent to 4 encryption modules..

Input format:

● Clk (1 bit)	- It is Clock signal
● Data_to_be_encrpt (60 bit)	- Data input by user (non-encrypted)

Output format:

● Output_encrypted (78 bit)	- Encrypted data

a) Encryption modules

These take *60 bit* input and *11 bit random number* and each encryption module generates a 60 bit number(*say x*) using 11 bit random number uniquely, and adds input and *generated 60 bit number*(x) and makes *61 bit* number (accounting for carry).
This *61 bit number*, *11 bit random number* and *6 bit random number* are concatenated and a *78 bit number* is made uniquely by each of 4 encryption modules. Output is the *78 bit number* generated.
Input format:

● Clk (1 bit)	- It is Clock signal
● Data_1 (60 bit)	- Data input by user (non-encrypted)
● Rand_11 (11 bit)	- 11 bit random number
● Rand_6 (6 bit)	- 6 bit random number

Output format

● outEnc (78 bit)	- Encrypted data

4) Decrypter

Modules instantiated:

In decrypter 4 encryption modules are instantiated

- **decrypt_function_1**,**decrypt_function_2**,**decrypt_function_3** and **decrypt_function_4**.

It first extracts required 6 bits of data which helps us to select the required decryption module's output.

4 decryption modules take the *78 bit encrypted* data as input and decrypts them uniquely.

Input format:

Clk (1 bit)	- It is Clock signal
Data_to_be_decrypt (78 bit)	- Data input by user (encrypted)

Output format:

Output_decrypted (60 bit)	- Decrypted data

a) Decryption modules

These take *78 bit* input and extracts *61 bit number(say y)* and *11 bit random number* and each encryption module regenerates a 60 bit number(*say x*) using 11 bit random number used at the time of encryption, and subtracts *61 bit number(* y) and r*egenerated 60 bit number*(x) and makes *61 bit* number(say *data1)* (accounting for carry).

The *60 bit number* is extracted from the *61 bit number*((*data1)*and is given as output.

The order of addition and concatenation is different for all **4 encryption modules** and is correspondingly known by **4 decryption modules**.

Input format:

Clk (1 bit)	- It is Clock signal
Data_1 (78 bit)	- Data input by user (encrypted)

Output format

outDec (60 bit)	- Decrypted data

5) Password Generator

Modules instantiated:

In Password Generator 5 modules are instantiated

- 4 encryption modules : **Gen_1**,**Gen_2**,**Gen_3** and **Gen_4**.
- and 1 random number generator: **bit_rand_6.**

4 password generation modules take 11 bit random as input and generate a 60 bit password.
Bit_rand_6 generates a *6 bit* random number and based on this random number output is selected from 4 outputs generated by **4 password generation modules** to make them more randomized.

Input format:

● Clk (1 bit)	- It is Clock signal

Output format:

● Password_generated (60 bit)	- Password generated.

a) Password generation modules

A 11 bit random number is generated using a random number generator. Using these *11 bit random number,* each module generates a 60 bit number in a similar way as the encryption module generated.
This 60 bit random number is output

Bit_rand_11 generates a *11 bit* random number and it is sent to 4 encryption modules..

Input format:

● Clk (1 bit)	- It is Clock signal
● Rand_6 (6 bit)	- 6 bit random number

Output format

● out (60 bit)	- Password Generated

6) Bit_rand_6:

This module generates a 6 bit random number using a linear feedback shift register (LFSR).
It has its own clock, hence this makes random number generation more random.
It is called in encrypter and password generator.
Seed is given from the file when we run for the first time. Every time we run module we update the seed value after some time so that we don't get the same set of random numbers as when we start running the module.

Output format

- rand_6 (6 bit) - 6 bit random number

7) Bit_rand_11:

This module generates a 11 bit random number using a linear feedback shift register (LFSR).
It has its own clock, hence this makes random number generation more random.
It is called in encrypter and password generator.
Seed is given from the file when we run for the first time. Every time we run module we update the seed value after some time so that we don't get the same set of random numbers as when we start running the module.

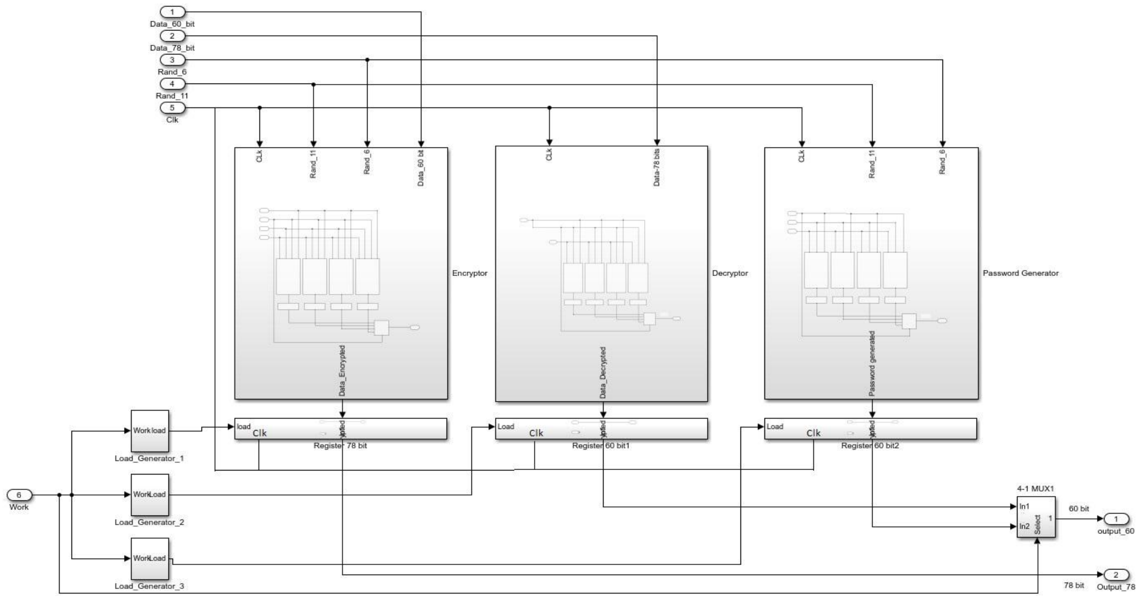
Output format

- rand_11 (11 bit) - 11 bit random number

Design

Design of Solver, Encrypter, Decrypter, Password Generator is as follows:

Solver:

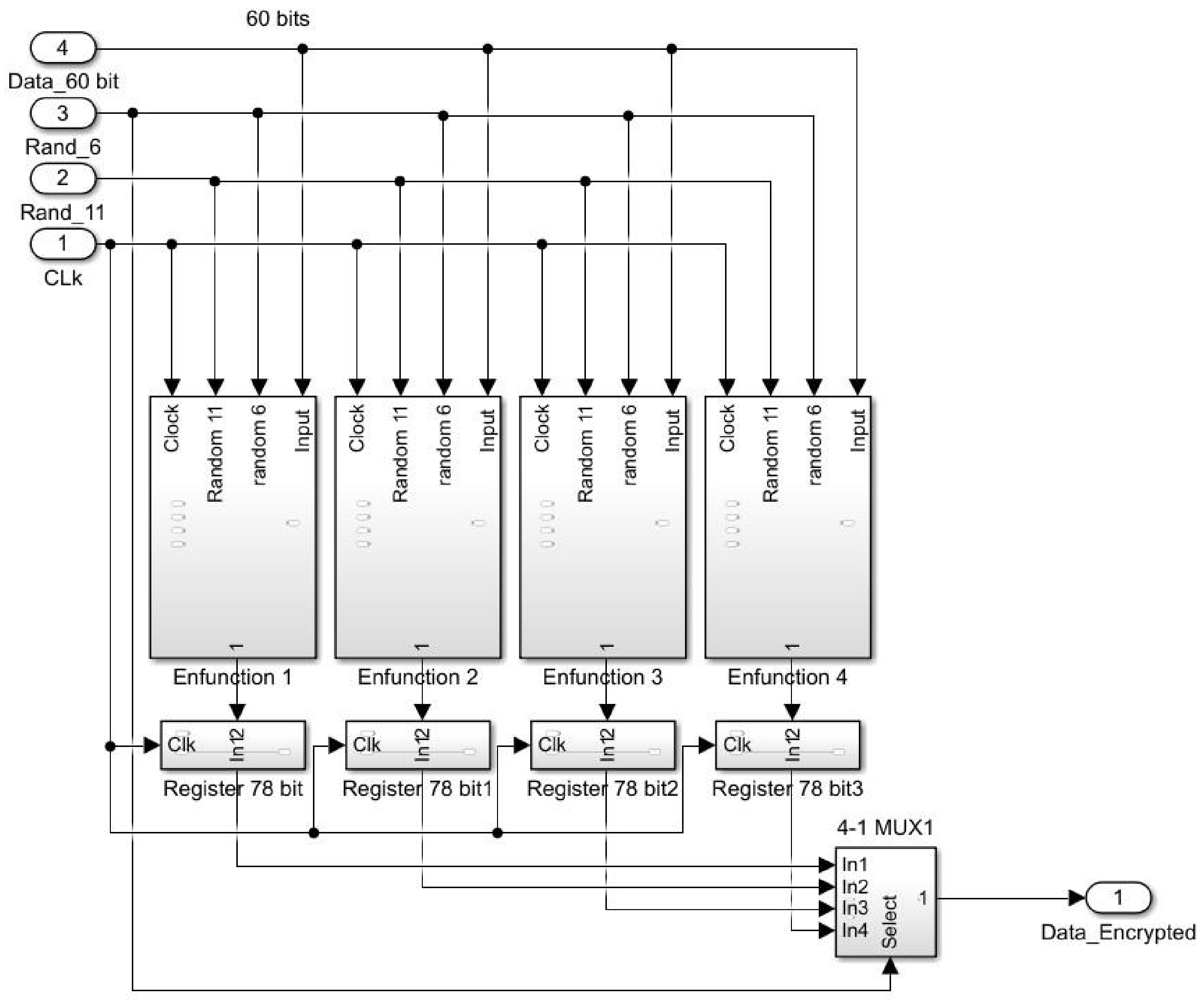


Circuit 1: Main Design

NOTE: Random numbers generators are not instantiated in solver and input is not from solver but as same random blocks used in encrypter and password generator, above design is more appropriate.

Data_60_bit, Data_78_bit, Rand_6, rand_11, Clk, Work are inputs and outputs are output_60, output_78.

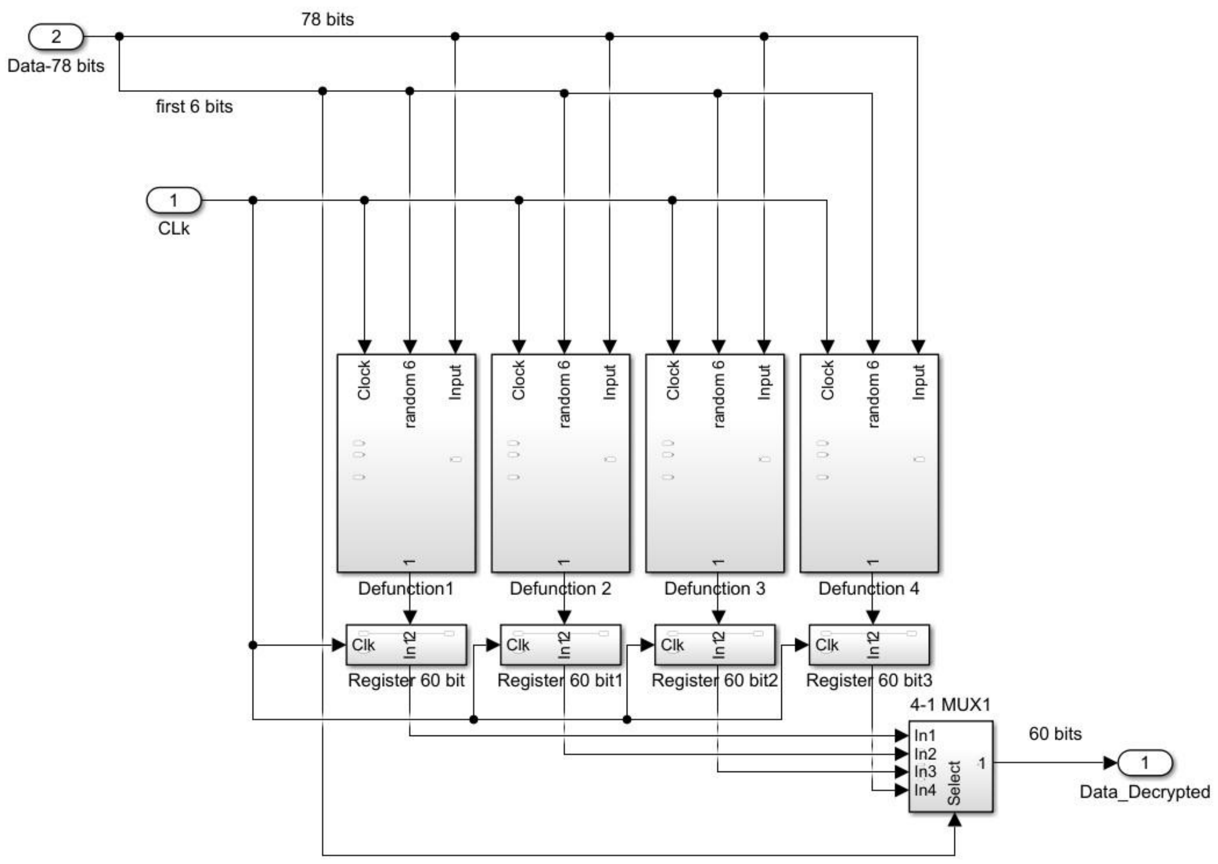
Encrypter



Circuit 2: Encrypter.

Above figure is a circuit diagram of the encrypter. It encrypts Data_60_bit. Data encrypted is output. Registers are used at end of each Enfunction modules to get stable output.

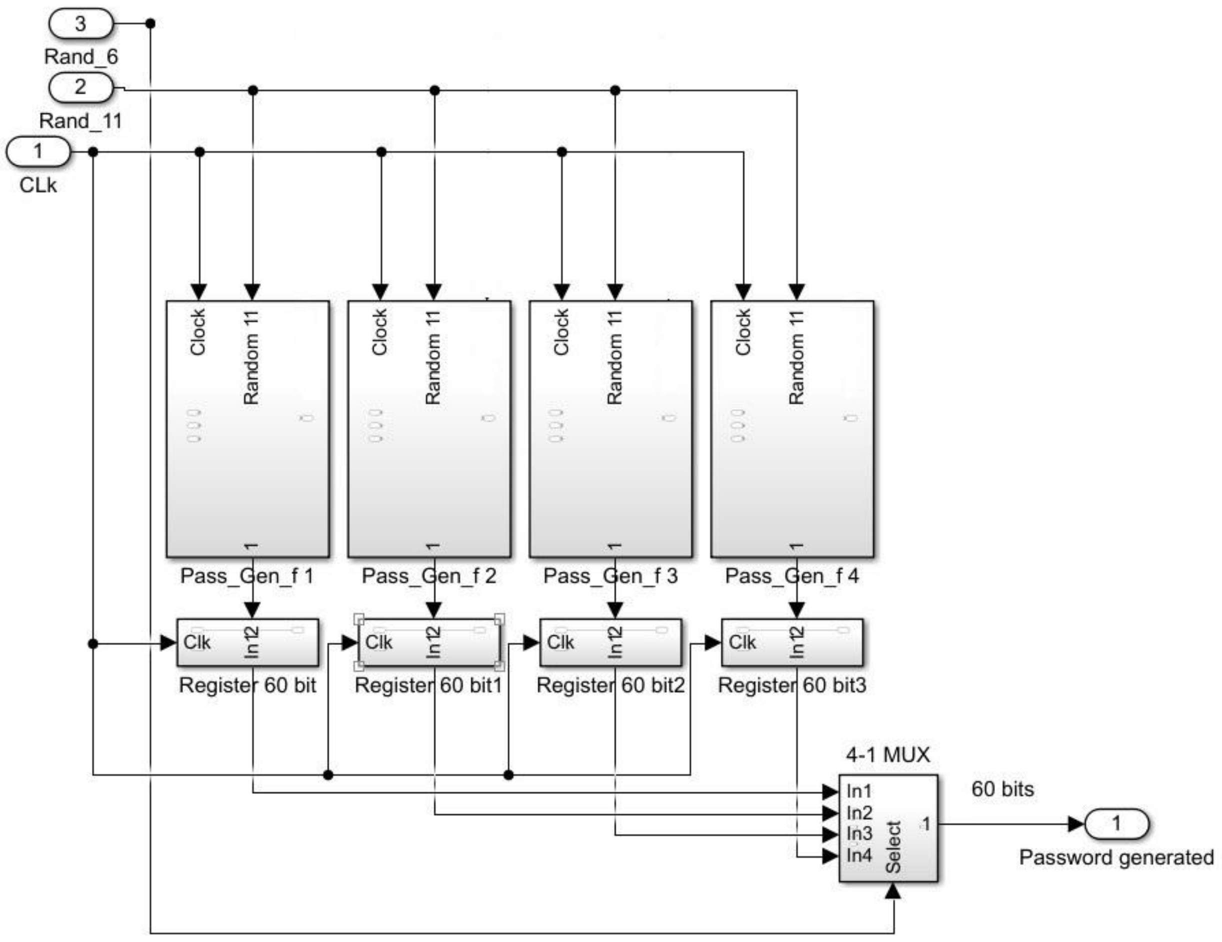
Decrypter



Circuit 3: Decrypter

Above figure is a circuit diagram of the decrypter. It decrypts Data-78 bits. Data_Decrypted is output. Registers are used at the end of each Defunction module to get stable output.

Password Generator



Circuit 4: Password generator

Above figure is a circuit diagram of the password generator. It generates a 60 bit password. Password generated is output. Registers are used at the end of each password generation module to get stable output.

Working of encryption function 1:

As we have 11 bit random number, now we need a 60 bit random number hence let 60 bit random number be as following:

Let the 60 bit number generated be [59:0]b, and rand_11 be the 11 bit random number.

b[10:0]=rand_11

b[21:11]=~rand_11

b[32:22]=~rand_11

b[43:33]=rand_11

b[54:44]=~rand_11

b[59:55]=rand_11[4:0]

~ denotes complement.

As defined above, a 60 bit random number is generated in encryption function 1.

Now the sum of input and b is calculated.

Sum = b+Data_1

Sum is 61 bit.

Let outEnc be a 78 bit output. rand_11, rand_6, Sum are concatenated as follows i encryption function 1:

outEnc[77:17]=Sum

outEnc[16:6]=rand_11

outEnc[5:0]=rand_6

In similar way **encryption functions 2, 3, 4** are defined.

Working of decryption function 1:

The 11 bit random number used for encryption is extracted first from encrypted data.

Rand_11 = Data_78[16:6]

Now the sum is extracted.

Sum = Data_78[77:17]

Value added to data *(b)* is generated using rand_11 as generated in enfunction.

Let the 60 bit number generated be [59:0]b, and rand_11 be the 11 bit random number.As defined above, the required 60 bit random number is generated in decryption function 1.

Now the difference of Sum and b is calculated.

Output = Sum-b

MSB is omitted as it is 0.

Hence Output is 60 - bit decrypted data.

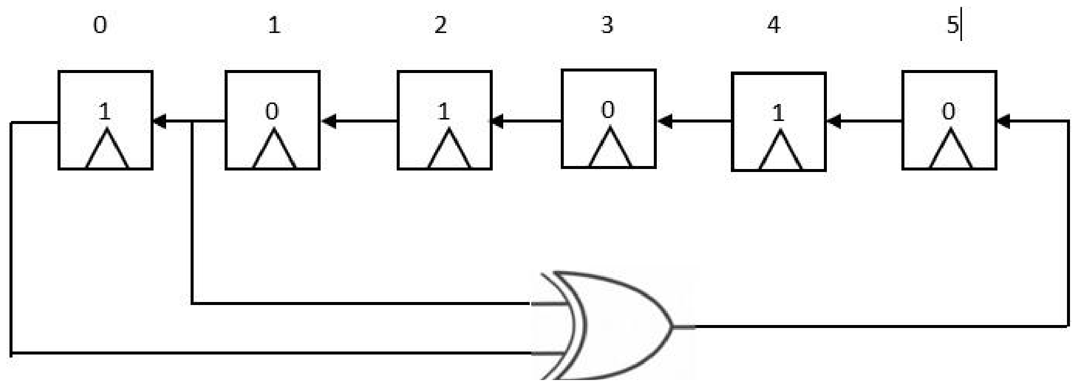
In similar way **decryption functions 2, 3, 4** are defined.

Working of password generation modules:

These modules generate 60 bit random numbers using 11 bit random numbers as encrypter generates.

Random number generator design(6-bit):

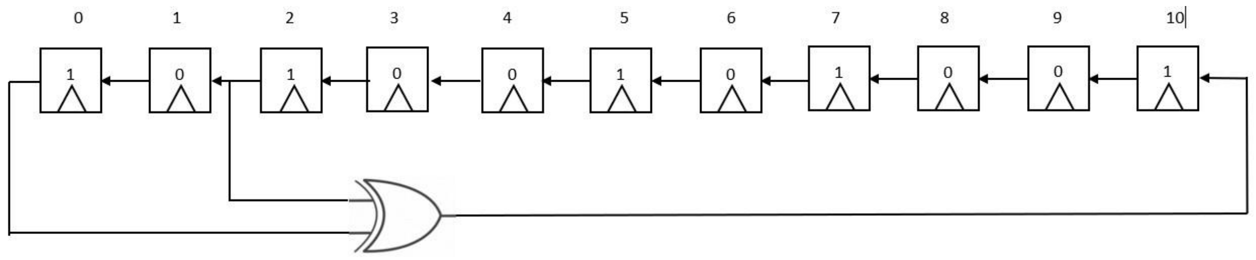
It uses a 6 bit linear feedback shift register.



Circuit 5: 6 bit LFSR

Random number generator design(11-bit):

It uses a 11 bit linear feedback shift register.



Circuit 6: 11 bit LFSR

LFSR:

LFSR (Linear feedback shift register) is basically a shift register but output is a feedback to it.The feedback is usually the *xor* of some bit so *previous state*.The *xor* connection between two flip flops is called *taps* .At each posedge of clock the last bit is dropped and 1st bit is taken as feedback. The 6 bit random number genaraattopsd all possible 63 states and the 11 bit random generator generates all 2047 states.

Approach validation

All features proposed are working perfectly.

Correctness of LFSR:

A maximum length lFSR gives all possible $2^m - 1$ outputs (except 0 otherwise all outputs will always be zero). Not all taps produce the maximum length LFSR. For 6 bit it is between 0 and 1. For 11 bit it is 0 and 2.

Same is verified by printing all the sequences and the number of sequences till one sequence repeats in both random number generators.

Correctness of Encryption:

A 60-bit random number (Let x) is generated using 11 bit and 6 bit random numbers. Now x and initial input data are added. The sum would be of atmost 61 bit. Hence 61 bits are allotted for sum in encrypted data. Hence Encryption works correctly.

Correctness of Decryption:

During encryption the random numbers generated are stored in an encrypted password.

As during decryption, the same random numbers which are generated during encryption are extracted first and the 60 bit random number which was added during encryption is generated. Hence we get the same 60 bit number that was generated. Let it be x.

Now 61 bit sum is extracted.

As sum = actual data + x.

Hence actual data = sum - x.

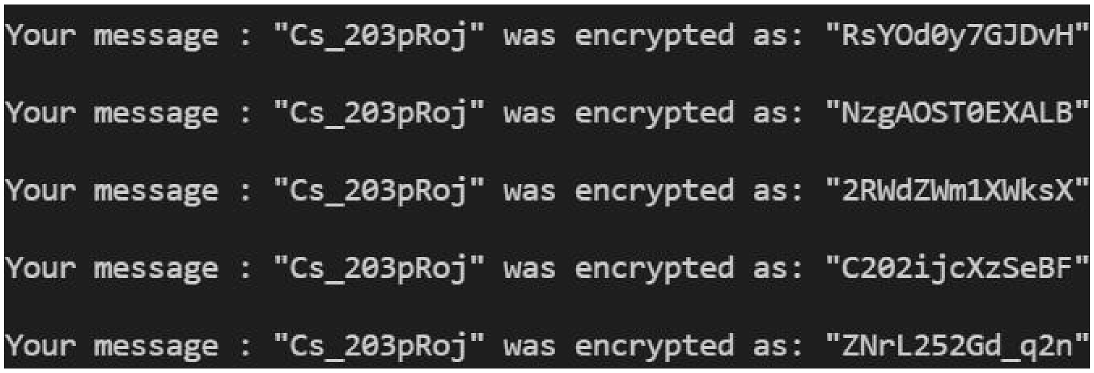
The order of sum is different for all 4 encryption functions.The corresponding decryption function does that in reverse order and correct data is generated.

As actual data is of utmost 60 bit, hence difference is also of utmost 60 bit and difference is required decrypted data.

Results

By testing we observed that the same password when encrypted different times encrypted password generated has changed.

Below is snapshot of output when string “Cs_203pRoj” is encrypted 5 times:

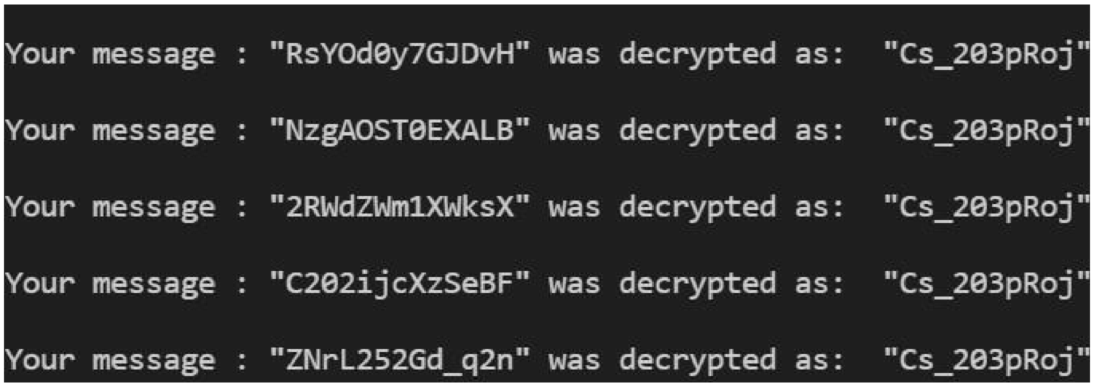


```
Your message : "Cs_203pRoj" was encrypted as: "RsYOd0y7GJDvH"

Your message : "Cs_203pRoj" was encrypted as: "NzgAOST0EXALB"

Your message : "Cs_203pRoj" was encrypted as: "2RWdZWm1XWksX"

Your message : "Cs_203pRoj" was encrypted as: "C202ijcXzSeBF"

Your message : "Cs_203pRoj" was encrypted as: "ZNrL252Gd_q2n"
```

Image 1: Output encrypted

This shows that encryption is randomized.

Even above encrypted passwords are decrypted and result is as follows:

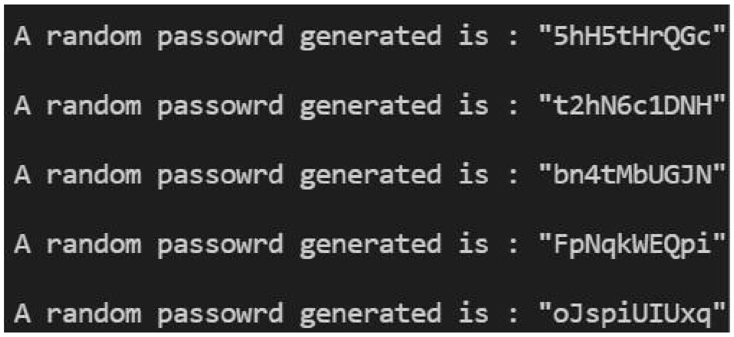


```
Your message : "RsYOd0y7GJDvH" was decrypted as:  "Cs_203pRoj"

Your message : "NzgAOST0EXALB" was decrypted as:  "Cs_203pRoj"

Your message : "2RWdZWm1XWksX" was decrypted as:  "Cs_203pRoj"

Your message : "C202ijcXzSeBF" was decrypted as:  "Cs_203pRoj"

Your message : "ZNrL252Gd_q2n" was decrypted as:  "Cs_203pRoj"
```

Image 2: Output decrypted

This shows that verilog was able to successfully decrypt 5 encrypted passwords.

We also generated some random passwords(data of 10 characters). Below is result:



```
A random passowrd generated is : "5hH5tHrQGc"

A random passowrd generated is : "t2hN6c1DNH"

A random passowrd generated is : "bn4tMbUGJN"

A random passowrd generated is : "FpNqkWEQpi"

A random passowrd generated is : "oJspiUIUxq"
```

Image 3: Random passwords generated

Here every time password generated is different as we use 2 random generators to generate random passwords.

Conclusion

Throughout the course of this project, we learned the implementation of LFSR and found one of the ways to generate random data.We used this random number generator to encrypt the data given by the user. We implemented randomized encryption decryption functions using verilog. Nowadays encryption is one of the most important things for user privacy. Even if data gets leaked then the data is not in readable format. In above encryption we did not use regular encoding(like UTF-8, UTF-16, ASCII, etc), It makes encryption more complex and it helps to reduce redundant bits. It has many such advantages as the same data will be encrypted differently at different times hence it is very difficult to predict the encryption or decryption of a given data.