Clock-Gated Pseudo Random Number Generator with Programmable Polynomial Taps using Clock-Gated Linear Feedback Shift Register (LFSR)

[1]N.Sreenivasa Rao,[2] M.Sai Praveena, [3]M.Sravani, [4]O.Renuka Devi, [5]V.Lavanya
[1]Assistant professor and [2345] IV[th] Year students, [12345] Department of ECE,
[12345] Santhiram Engineering College, Nandyal, India.

***Abstract*—** *LFSR have the feature of accepting 32bit programmable polynomial and it has a feature of 32 bit or 16 bit LFSR. It has clock gated flip-flop and can be used in dynamic power reduction. This gating of the clock will depend on the previous input and the present input. This "pseudo random number generator" generates 32-bit random number according to the polynomial which is programmable through the input "poly [31:0]". The top module passes the inputs to the "gated_lfsr" module. The reason for making a top module on top of a "gated_lfsr" is for the flexibility to add multiple "gated_lfsr" in future by just instantiating multiple modules in the top module "pseudo_top". The "gated_lfsr' module, in turn, instantiates total "32" multiplexers and "32" "gated d-flip-flops" to output 32-bit pseudo-random number.The "gated d-flip-flop" is a special type of "d-flip-flop' which has a capability of disabling clock when both the present state and the next state of the d-flip-flop are same, thus saving the dynamic power consumed due to the transitions of the clock signal.*

Keywords*—LFSR, Gated D-FlipFlop, Multiplexer, Polynomial Taps.*

.I. INTRODUCTION

Random numbers are widely used in various applications such as Monte Carlo simulations, cryptography, simulations of wireless communication systems, electronic circuit testing, genetic programming, data encryption, games etc. Usually, random numbers are generated using software algorithms. Although the sequence of numbers they produce seems random, they are not truly random. It is difficult to program a series of logical steps that produce numbers that do not follow some definite sequence. These random numbers are called Pseudo-random numbers. True random numbers can be generated from a physical process, such as measuring thermal noise or noise power level in a radio-frequency receiver, photoelectric effect or other quantum phenomena.

Ideally, the generated random numbers should be uncorrelated and satisfy any statistical test for randomness. A generator can be either "truly random" or "pseudo-random". These use a formula to generate numbers which behave very like genuine random numbers and are widely used for simulations of random processes and statistical methods. In most cases, a good pseudo-random number generator seems to work as you would expect a genuine random generator to work. For a suite of programs for testing pseudo-random number generators and details of some pseudo-random number generators see George Marsaglia's Diehard tests. See also Taygeta Scientific' notes on random number generators and the numerical analysis.

II.LINEAR FEEDBACK SHIFT REGISTER

The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle.

Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of LFSRs are common.

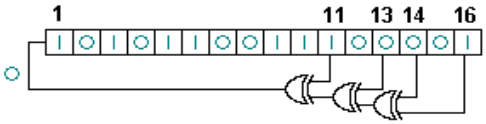


Figure.1. A 16-bit LFSR. The feedback tap numbers shown correspond to a primitive polynomial taps.

The bit positions that affect the next state are called the taps. In the diagram the taps are [16,14,13,11]. The rightmost bit of the LFSR is called the output bit. The taps are XOR'd sequentially with the output bit and then fed back into the leftmost bit. The sequence of bits in the rightmost position is called the output stream.

- The bits in the LFSR state that influence the input are called *taps*.
- A maximum-length LFSR produces an m-sequence (i.e., it cycles through all possible $2^n - 1$ states within the shift register except the state where all bits are zero) unless it contains all zeros, in which case it will never change.
- As an alternative to the XOR-based feedback in an LFSR, one can also use XNOR.[2] This function is an affine map, not strictly a linear map, but it results in an equivalent polynomial counter whose state is the complement of the state of an LFSR. A state with all ones

is illegal when using an XNOR feedback, in the same way as a state with all zeroes is illegal when using XOR. This state is

considered illegal because the counter would remain "locked-up" in this state.

The sequence of numbers generated by an LFSR or its XNOR counterpart can be considered a binary numeral system just as valid as Gray code or the natural binary code.

The arrangement of taps for feedback in an LFSR can be expressed in finite field arithmetic as a polynomial mod 2. This means that the coefficients of the polynomial must be 1s or 0s. This is called the feedback polynomial or reciprocal characteristic polynomial. For example, if the taps are at the 16th, 14th, 13th and 11th bits (as shown), the feedback polynomial is

$$x^{16} + x^{14} + x^{13} + x^{11} + 1$$

The "one" in the polynomial does not correspond to a tap – it corresponds to the input to the first bit (i.e. x^0, which is equivalent to 1). The powers of the terms represent the tapped bits, counting from the left. The first and last bits are always connected as an input and output tap respectively.

The LFSR is maximal-length if and only if the corresponding feedback polynomial is primitive. This means that the following conditions are necessary (but not sufficient):

- The number of taps should be even.
- The set of taps – taken all together, not pairwise (i.e., as pairs of elements) – must be relatively prime. In other words, there must be no divisor other than 1 common to all taps.

Tables of primitive polynomials from which maximum-length LFSRs can be constructed are given below and in the references.

There can be more than one maximum-length tap sequence for a given LFSR length. Also, once one maximum-length tap sequence has been found, another automatically follows. If the tap sequence in an n-bit LFSR is $[n, A, B, C, 0]$, where the 0 corresponds to the $x^0 = 1$ term, then the corresponding "mirror" sequence is $[n, n - C, n - B, n - A, 0]$. So the tap sequence [32, 7, 3, 2, 0] has as its counterpart [32, 30, 29, 25, 0]. Both give a maximum-length sequence.

III. MuLTIPLEXER

A **multiplexer** or mux is a combinational circuit that selects several analogs or digital input signals and forwards the selected input into a single output line. A multiplexer of 2^n inputs has n selected lines, are used to select which input line to send to the output.

Design using transmission gate logic:

A transmission gate is an electronic element and good non-mechanical relay built with CMOS technology. It is made by a parallel combination of NMOS and PMOS transistors with the input at the gate of one transistor (*C*) being complementary to the input at the gate of the other. The symbol of a transmission gate is shown below

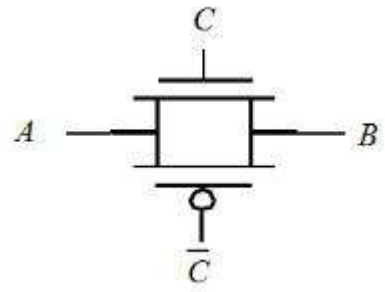


Figure.2: symbol of transmission gate

The transmission gate acts as a bidirectional switch controlled by the gate signal *C*. When *C=1*, both MOSFETs are on, allowing the signal to pass through the gate. In short, *A=B*, if *C=1*. On the other hand, *C=0*, places both transistors in cut-off, creating an open circuit between nodes *A* and *B*. Fig.5 shows the implementation of a 2:1 MUX using transmission gate logic.

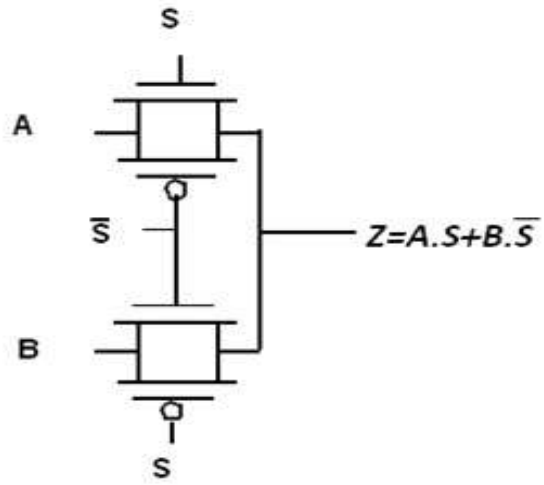


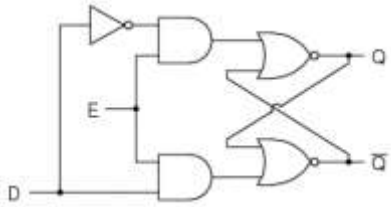
Figure.3: mux using transmission gate

Here, the transmission gates selects input *A* or *B* on the basis of the value of the control signal *S*. When *S=0*, *Z=A* and when *S=1*, *Z=B*

IV. Gated D-Flip Flop

In electronics, a **flip-flop** or **latch** is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. Since the enable input of a gated S-R latch provides a way to latch the Q and not-Q outputs without

regard to the status of S or R, we can eliminate one of those inputs to create a multivibrator latch circuit with no "illegal" input states. Such a circuit is called a D latch, and its internal logic looks like this:



E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

Figure.4:D-Flip Flop and its truth table.

Note that the R input has been replaced with the complement (inversion) of the old S input, and the S input has been renamed to D. As with the gated S-R latch, the D latch will not respond to a signal input if the enable input is 0 it simply stays latched in its last state. When the enable input is 1, however, the Q output follows the D input.

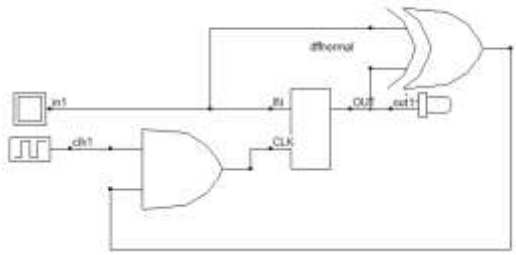


Figure.5: Gated D-Flip Flop

The "gated d-flip-flop" is a special type of "d-flip-flop' which has a capability of disabling clock when both the present state and the next state of the d-flip-flop are same, thus saving the dynamic power consumed due to the transitions of the clock signal.

V.Working Of The Module

This "pseudo random number generator" generates 32-bit random number according to the polynomial which is programmable through the input **"poly [31:0]"**. The top module passes the inputs to the "gated_lfsr" module. The reason for making a top module on top of a "gated_lfsr" is for the flexibility to add multiple "gated_lfsr" in future by just instantiating multiple modules in the top module "pseudo_top".

The "gated_lfsr' module, in turn, instantiates total "32" multiplexers and "32" "gated d-flip-flops" to output 32-bit pseudo-random number.

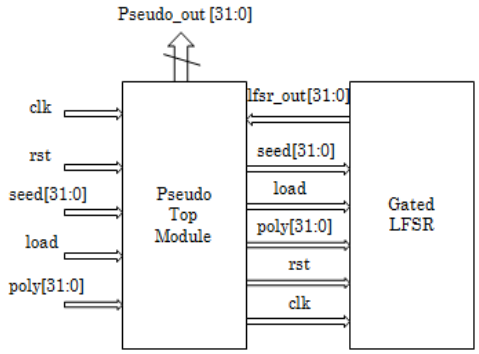


Figure.6 :Block diagram of pseudo random number.

As mentioned in the above there are total five inputs going into the top module "pseudo_top" and one 32-bit output coming out of module, which is as below

Clk - Input clock signal

Rest - Input reset(asynchronous)signal.

Seed[31:0] - Input 32-bit value through which an initial seed value can be programmed into the LFSR.

Load - The input "seed" is loaded on to the flip-flop outputs when this signal is high. When this "load" signal is low, the flip-flop outputs the outputs of previous flip-flops.

Poly[31:0 -The bits in this input denote whether there is tap coming from corresponding flip-flop output as input to the xor gate whose result, in turn, go to the input of the first flip-flop.

Pseudo_out[31:0] -Final 32-bit "pseudo-random number" coming as output

As mentioned in fig.2, there are five inputs going into the module and one output coming out of the "gated_lfsr" output. Those inputs have the same meaning as mentioned for the 'pseudo_top" module. The output "lfsr_out" provide the output "pseudo-random number" from the "gated_lfsr" module.

As mentioned in fig.3, there are three inputs going into the "gated_dff" module and one output coming out of it.

Clk- input clock

D- Input one-bit data to the d-flip-flop

Q- Output one-bit data to the d-flip-flop

You can observe that this d-flip-flop is a special type of flip-flop which gates the input clock to save dynamic power. The gating feature functions as below

When the present output and next input of the d-flip-flop are same, then there is no need for the flip-flop to output the next input because there will be no change in the output as they are same. So, this outputting the next input can be avoided by avoiding the clock to get switched. So the clock will
be gated in this time span.

VI. Gated Pseudo Random Number Generator With Gated LFSR

In this design, the normal "gated LFSR" is used to design "gated pseudo random number generator". In addition to the gating feature, the **new improvement** that is added to this design is the **"programmable feature of the taps coming to the inputs of XOR gates"** whose final input is further given to the input of the first flip-flop.

Normally, the output pseudo-random number from the pseudo random number generator depends on which flip-flop outputs are XORed and given to the input of the first flip-flop. The inputs to the XOR gates is given from the outputs of the flip-flops of LFSR through wires called "taps". Whether there should be a tap from a particular flip-flop output to the XOR gates or not is decided by the **"polynomial"** according to which the designing of the LFSR is done. Normally this polynomial is fixed in most of the designs.

For example, if the polynomial which is decided is

$$x^{31} + x^{20} + x^{5} + 1$$

Then, it means that there should be a tap from the output of the 30^{th} flip-flop, the output of the 19^{th} flip-flop, the output of 4^{th} flip-flop according to the notations followed and these should be XORed and the result of that XOR operation should be given to the input of the first flip-flop. Note that the value "1" in the polynomial does not denote a tap but it denotes the input of the first flip-flop. As the input of the first flip-flop should always be present, "1" should always be present in the polynomial equation

But in this design, we offer a flexibility to the user to use any polynomial through the input "poly[31:0]" according to which the design aligns itself to take only the tabs from the particular flip-flop output number whose corresponding bit in the input "poly[31:0]". **So, by this improvement, we are providing high range of flexibility to the user to use our module in a large number of applications and to program required polynomial to decide the taps which are to be XORed to give to the input of the first flip-flop i.e. as feedback.**

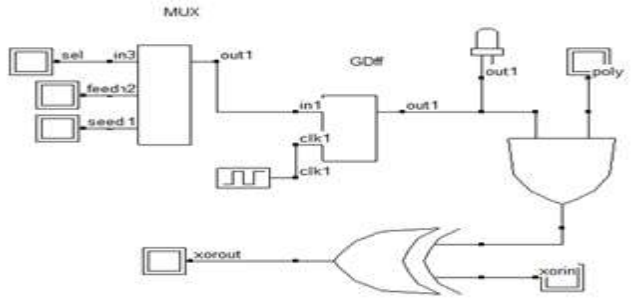


Figure.7: Internal diagram of the LFSR

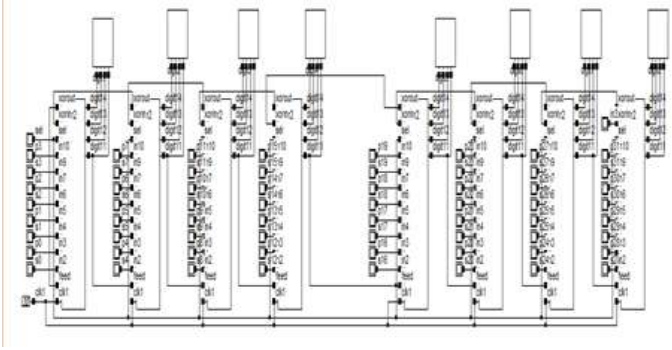


Figure.8: Block diagram of the pseudo random number in the microwind tool.

VII. Result and Discussion

In this, we generated a random number.The poly and seed are the input bits and before that, the selection input should be enabled .In this, we are giving feedback to the each LFSR.The feedback input bit is the previous output.At last LFSR bit should be high to on the circuit.

For different poly input, it produces different random numbers when the clock is on.when the selection input is disabled it shows the seed input at the output. The following are the two different outputs for the different poly and inputs.The outputs are 64666425 and 74362524 32-bit outputs.

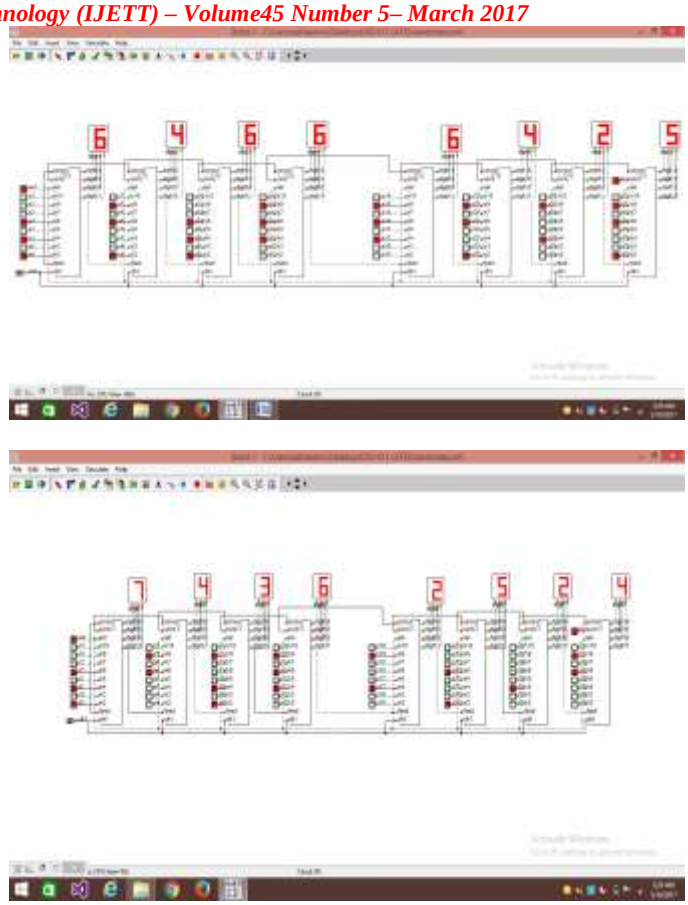


Figure.9: simulation result of the 32-Bit random number in microwind tool.

VII. Applications

- PRNG (Pseudorandom number generator) can be used in cryptographic applications to generate partially random numbers for encryption.
- PRNG for generating OTP's(One Time Password).
- PRNG is used in 'statistical sampling" and also "Monte Carlo" simulations.
- PRNG is used in electronic noise studies.

VIII.CONCLUSION

In this paper, a Pseudo random number is generated by using the clock gated D-Flip Flop and mux. The 32-bit number is generated.We can also modify this by extending by adding BCD converter at the end of the hexadecimal output. By using clock gated we get the less delay and accuracy, thus saving the dynamic power consumed due to the transitions of clock signal.

References

[1] Wikipedia, Pseudorandom Number Generators, http://wikipedia.com. Pseudorandom number generator (2003).
[2] F. James, "A Review of Pseudo-random Number Generators, " Computer physics communication 60,1990.
[3] David B. Thomas, Wayne Luk, ―The LUT-SR Family of Uniform Random Number Generators for FPGA Architectures‖ IEEE transactions on very large scale integration (VLSI) systems, vol. 21, no. 4, April 2013
[4] Carlos Arturo Gayoso, C. González, L. Arnone, M. Rabini, Jorge Castiñeira Moreira, ―Pseudorandom Number Generator Based on the Residue Number System and its FPGA Implementation‖ 2013 Argentine School of Micro-Nanoelectronics, Technology, and Applications.