FPGA Implementation of 16 bit BBS and LFSR PN Sequence Generator: A Comparative Study

Khushboo Sewak, Praveena Rajput and Amit Kumar Panda
Dept. of ECE, IT, Guru Ghasidas Vishwavidyalaya
Bilaspur, Chhattishgarh
amit.eldt@gmail.com

Abstract— The main purpose of this paper is to study the FPGA implementation of two 16 bit PN sequence generator namely Linear Feedback Shift Register (LFSR) and Blum-Blum-Shub (BBS). We have used FPGA to explain how FPGA’s ease the hardware implementation part of communication systems. The logic of PN Sequence Generator presented here can be changed any time by changing the seed in LFSR or by changing the key used in BBS. The analysis is conceded out to find number of gates, memory and speed requirement in FPGA for the two methods. As Recently the field programmable gate arrays have enjoyed wide spread use due to several advantages related to relatively high gate density, short design cycle and low cost. The greatest advantage of FPGA’s are flexibility that we reconfigured the design many times and check the results and verify it on-chip for comparing with others PN sequence generators.

Keywords- BBS, Blum Blum Shub, LFSR, PRNG, TRNG, FPGA

I. INTRODUCTION

The ability to generate pseudorandom numbers is important for simulating events, estimating probabilities and other quantities, making randomized assignments or selections, and numerically testing symbolic results. Such applications may require uniformly distributed numbers, non-uniformly distributed numbers, elements sampled with replacement, or elements sampled without replacement. Random numbers are needed for a wide range of computational problems in Science and Engineering which require statistically random input. The random number generator produces a sequence of number which lack any pattern i.e appear random. Random numbers are generated by various methods [1]; the two types of generators used are True random number generator (TRNG) [1] and Pseudorandom number generator (PRNG) [2]. TRNG are the generators used for generating random data and have existed since ancient times, including dice, coin tossing, playing cards etc. On the other hand computational methods which are based on certain algorithms and can also produced good random sequence but eventually the sequence repeats are known as PRNG.

In this paper we have used two PRNGs methods namely Linear Feedback Shift Register (LFSR) and Blum Blum Shub method (BBS) [6]. LFSR is a periodic PRNG whereas BBS is a non periodic PRNG. It is simulated and synthesized using VHDL on the Xilinx FPGA Spartan 3E500 [9]. The use of feedback shift register permits very fast generation of PN sequence whereas BBS method requires a number of time consuming arithmetic operation as it is based on quadratic Congruential equation . This paper gives an overview about memory required and speed required between the two methods. Also it present how BBS is more secure than LFSR.

The structure of this paper is as follows. Section II contains a brief description of two types of methodology such as LFSR and BBS. Section III shows synthesis and timing simulation result as well as comparison between two by implementing on FPGA [10].

II. METHODOLOGY

A. LFSR PRNG

A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is XOR. Thus, an LFSR is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value. The initial value of the LFSR is called the seed, and by changing the seed we can change the sequence 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 which appears random and which has a very long cycle. That feedback function is called a maximal length polynomial [3]-[5]. In this paper we have chosen maximal length polynomial of 4 bit LFSR which generate 16 random states. After generating the states, LSBs of each state is taken as a PN sequence.

Pseudo Random Number Sequence is generated in VHDL [11] according to the following circuit based on the above concept of shift register. The VHDL code can be extended to Multibit LFSR PRNG.
The bits in the LFSR state which 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. 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 1's or 0's. This is called the feedback polynomial or characteristic polynomial. For example, in a 4 bit LFSR if the taps are at the 4th and 3rd bits (as shown), then the feedback polynomial is
\[ x^4 + x^3 + 1. \]

**1) The Rules For Selecting Feedback Polynomial:**

The rules for selecting feedback polynomial which is given in [3], [4] are as follows:

- 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 will only be maximum-length if the number of taps is even; just 2 or 4 taps can suffice even for extremely long sequences.
- 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 common divisor to all taps.
- 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.

**B. Blum Blum Shub PRNG**

It is a pseudo random number generator proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub in 1986 [6]-[8]. It has been proved that BBS is cryptographically secure method. But one drawback of BBS is that its simulation process is longer. The only aim of implementing BBS on FPGA by VHDL coding is to give on chip verification. Although there are a certain delay and glitch phenomena in the waveform, they don’t affect the normal operation of the whole module. So the test results are correct on the function and timing. Here the simulation clock period is taken as 20 ns.

**1) Algorithm for Blum-Blum-Shub pseudorandom bit generator:**

This algorithm generate a pseudorandom bit sequence \( z_1, z_2, \ldots, z_l \) of length \( l \) [8].

1. Take two large secret random (and distinct) primes \( p \) and \( q \), each congruent to 3 modulo 4.
2. Compute \( n = p^q \).
3. Select a random integer \( s \) (seed) in the interval \([1, n-1]\) such that \( \gcd(s, n) = 1 \).
4. Compute \( x_0 = s^2 \mod n \).
5. For \( i \) from 1 to \( l \) do the following:
   i. \( x_i = x_{i-1} \mod n \).
   ii. \( z_i \) the least significant bit of \( x_i \).
6. The output sequence is \( z_1, z_2, \ldots, z_l \).

**2) Example:** In this paper according to the above algorithm we have selected two prime numbers \( p=47 \) and \( q=67 \)

1. Both are congruent to 3 modulo 4 in the sense that \( 47 \mod 4 \equiv 3 \) and \( 67 \mod 4 \equiv 3 \).
2. Now \( n=47 \times 67 = 3149 \).
3. We have selected seed \( s=7 \) and also \( \gcd(7, 3149) = 1 \).
4. Now \( x_0=7^2 \mod 3149=49 \).
5. Then loop is run from \( i=1 \) to \( i=16 \) and remaining numbers are calculated as per algorithm.
6. We get the sequence \( 111010110110010 \).

**III. SYNTHESIS AND SIMULATION**

In this design, we describe the RTL-level of the LFSR pseudo-random number generator for 4-bit, and 16 bit BBS using VHDL language, and use the Xilinx’s chip XC3S 500 Sparta3E as the target chip. Then we synthesize, place and route on the Xilinx ISE platform. Finally we use ISE Simulator to do a timing simulation [12].

**A. Timing Simulation**

Input the netlist file generated from synthesizing, placing and routing on the Xilinx ISE 10.1 software and the standard delay file, the simulation waveform is shown in Fig. 2, Fig. 3, Fig.4 and Fig.5 under the simulation clock is 371.747MHz. Although there are a certain delay and glitch phenomena in the waveform, but they don’t affect the normal operation of the whole module. So the test results are correct on the function and timing. Here the simulation clock period is taken as 20 ns.

**B. LFSR 4-Bit with seed S= 0001:**

Timing simulation waveform of 4-bit LFSR for maximum length polynomial \((X^4 + X^3 + 1)\) with seed \( S = 0001 \) is shown in Fig. 2. From the figure we can observe 15 random output states (20 ns to 320 ns) are coming for 4 bit LFSR and after that it is repeating. The sequence obtained is “00010011010110010” which are taken from the LSB.

**Figure 2.** Timing Simulation waveform of 4-bit LFSR for maximum length

**C. LFSR 4-Bit with seed S= 0010:**

Timing simulation waveform of 4-bit LFSR for maximum length polynomial \((X^4 + X^3 + 1)\) with seed \( S = 0010 \) is shown in Fig. 3. From the figure we can observe 15 random output states (20 ns to 320 ns) are coming for 4 bit LFSR and after
that it is repeating. The sequence obtained is “0001001101011111” which are taken from the LSB.

Figure 3. Timing Simulation waveform of 4-bit LFSR for maximum length polynomial \((X^4 + X^3 + 1)\) with Seed, \(S = 0010\)

From Fig. 2 and Fig 3 it is clear that if maximum length polynomial is selected than by changing the seed the number of output states doesn’t change.

D. 16-bit BBS:

Timing simulation waveform of 16-bit BBS for the primitive element \(p=47, q=67\) and for the seed \(S = 7\) is shown in Fig. 4. From the figure it can be observed that 16 bit sequence “110101011001010” is generated after 450 ns simulation time.

Figure 4. Timing Simulation waveform of 16-bit BBS for the primitive element \(p= 47, q= 67\) and for the seed \(S = 7\)

IV. COMPARISON OF IMPLEMENTATION RESULT

The synthesis design code is technology independent and was simulated at both RTL and gate level (post place & route netlist) with timing back annotation using Xilinx ISE and mapped onto Xilinx’s Spartan 3E500 FPGA [9] using Foundation ISE 10.1 version. The comparative table between 16-bit BBS and 4-bit LFSR is given on the table 1.

The generated 15 bit sequence of 4-bit LFSR is taking 1260 ns simulation time at 20 ns clock cycle where as as 16-bit BBS generates sequence after 450 ns on the single clock cycle. From the simulation waveform it can be observed that the sequence is repeating after 15 clock periods in LFSR where as in BBS after 15 clock period another random bit is generate.

TABLE I: SIMULATION AND SYNTHESIS RESULT

<table>
<thead>
<tr>
<th>Performance</th>
<th>4 Bit LFSR (15 bit Generated Seq.)</th>
<th>16 Bit BBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to complete the total states</td>
<td>20 ns to 1280 ns = 1260</td>
<td>It takes only 1 ns, After 450 ns</td>
</tr>
<tr>
<td>Total no. of Random States generating</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>Clock</td>
<td>20 ns</td>
<td>20 ns</td>
</tr>
<tr>
<td>Shift Register</td>
<td>04 one bit</td>
<td>One 16 bit Register</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>02</td>
<td>00</td>
</tr>
<tr>
<td>No. of Flip Flops</td>
<td>04</td>
<td>16</td>
</tr>
<tr>
<td>No. of 4 i/p LUT</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>CPU time</td>
<td>0.39 sec</td>
<td>4.11 sec</td>
</tr>
<tr>
<td>Total pin</td>
<td>06</td>
<td>18</td>
</tr>
</tbody>
</table>

V. CONCLUSION

It is clearly found from the synthesis and simulation result that 4 bit LFSR can generate 15-bit sequence with maximum length polynomial and 16-bit BBS can generate 16-bit sequence. Also the performance is better in 4-bit LFSR as compared to 16-bit BBS as the time required is much more in BBS. CPU time is taking for 4-Bit LFSR is 0.39 sec whereas for BBS it is 4.11 sec. Memory utilization is more in BBS than LFSR as the number of FFs used in BBS are more. While as per security concerned, BBS is more secure than LFSR because it is non-periodic where in case of LFSR because of repetition after certain time period any one can detect probability of the random states. It can be observed from the waveform simulation after 15 clock period, LFSR is generating same sequence where as BBS generating another new random bit. So BBS can be used in various cryptographic applications for secure communication where security issue is most important.

REFERENCES