ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Design & Implementation of Well-Method for Embedded Generation of LPR Numbers

Venkateshwarlu Purumala
M.Tech, VLSI & Embedded Systems
DVR College of Engineering and Technology,
Kashipur, Kandi, Sangareddy, Medak Dist.

P.Ramesh Reddy
Associate Professor,
DVR College of Engineering and Technology,
Kashipur, Kandi, Sangareddy, Medak Dist.

Abstract:
The Well Equidistributed Long-period Linear (WELL) algorithm is proven to have better characteristics than the Mersenne Twister (MT), one of the most widely used long-period pseudo-random number generators (PRNGs). In this paper, we propose a hardware architecture for efficient implementation of WELL. Our design achieves a throughput of 1 sample-per-cycle and runs as fast as 449.4 MHz on a Xilinx XC6VLX240T FPGA. This performance is 7.6-fold faster than a dedicated software implementation, and is comparable to a MT hardware generator built on the same device. It takes up 633 LUTs, 537 Flip-Flops and 4 BRAMs, which is only 0.5% of the device. Furthermore, we design a software/hardware framework that is capable of dividing the WELL stream into an arbitrary number of independent parallel sub-streams. With support from software, this framework can obtain speedup roughly proportional to the number of parallel cores. The quality of the random numbers generated by our design is verified by the standard statistical test suites Diehard and TestU01. We also apply our framework to a Monte-Carlo simulation for estimating . Experimental results verify the correctness of our framework as well as the better characteristics of the WELL algorithm.

I. INTRODUCTION:

High quality random numbers are of critical importance to many scientific applications, particularly for Monte-Carlo simulations. Considering the advantages of high performance and reproducibility, Pseudo-Random Number Generators (PRNGs) based on linear recurrences over are widely adopted in such simulations. One prevalent -linear PRNG is the Mersenne Twister (MT) [1], which has a very long period (219937-1 or larger) and good equidistribution. However, it is sensitive to poor initialization and can take a long time to recover from a “zero-excess” initial state [2][11]. To overcome this problem, the Well Equidistributed Longperiod Linear (WELL) algorithm was proposed [2]. Compared to MT, WELL has better equidistribution while retaining an equal period length. As application sizes scale, one emerging trend is to develop parallelized versions of the applications to exploit the available parallel hardware resources, such as in FPGAs, to achieve high speed up in performance.

Being the key component of many scientific applications, designing PRNGs that can rapidly provide independent parallel streams of high quality random numbers is also becoming more and more important in modern systems. The Fast Jump Ahead Technique [5] provides an efficient algorithm to jump ahead by a large number of steps for the long-period -linear PRNGs, thus providing strong theoretical support for parallelizing PRNGs of long-period. A large body of research has been done on -linear PRNGs, most of which focus on algorithms and corresponding software implementations. Only a few hardware implementations can be found in the literature.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

For those hardware implementations, most of them employ the MT method, including straightforward non-parallel [6][8] and parallel [9][10] hardware implementations. Given its advantages over MT, WELL also receives great attention from the software community. However, few hardware implementations can be found. In [4], the Ukalta Engineering Corporation gives a brief introduction to its product that employs the WELL algorithm. However, it only achieves a throughput of 1 sample every 2 cycles and no structural details are revealed. In this paper, we develop a hardware architecture forWELL19937 with throughput of 1 sample per cycle. We also design a software/hardware framework to parallelize its output stream.

More specifically, we make the following contributions: □ □ We design a hardware architecture for the WELL method that can achieve a throughput of 1 sample per cycle. □ □ We deise a dedicated 6R/2W RAM structure for WELL, which helps to achieve a high throughput for the entire design, with little resource overhead. □ □ We design a software/hardware framework to generate parallel random numbers, based on the WELL algorithm and Fast Jump Ahead Technique. □ □ We develop an algorithm to efficiently derive them characteristic polynomial "cp(z)" for WELL, which is universal and can be easily extended to other linear generators. □ □ We evaluate the proposed architectures using different statistical tests, including the Diehard and the TestU01 test suites.

We also apply our framework to a practical application, Monte-Carlo simulation for estimation. □ □ We implement the proposed architectures and application on a Xilinx Virtex 6 device. The rest of the paper is organized as follows. Section II gives brief introductions of the WELL algorithm and Fast Jump Ahead Technique. Section III presents our WELL hardware architecture. Section IV introduces the Software Hardware framework. Section V describes technique specific implementations, discusses evaluations and results. Section VI provides results of statistical testing and the Monte-Carlo application and Section VII gives conclusions.

II. ALGORITHMIC BACKGROUNDS

This section gives brief introductions of the WELL algorithm and Fast Jump Ahead Technique.

A. WELL Algorithm

The state transition process of the WELL algorithm is illustrated in Fig. 1.

The state vector S contains $k = w \times R - p(0 \leq p < w)$ bits, which are decomposed into R blocks with w-bit size. The last p bits of $S[R-1]$ are always zero. In each transition, six blocks in S (i.e. S[0], S[M1], S[M2], S[M3], S[R-2] and S[R-1]), are transformed into two results, Feedback1 and Feedback2, through a series of elementary bit-wise operations. Feedback2 is chosen as the output, directly or after some other bit-wise operations. After that, both Feedback1 and Feedback2 are inserted back into S with the positions of 1 and 0, respectively. In the meantime, the middle R-2 blocks from S[1] to S[R-2] are shifted by one word as shown in Fig. 1. When p=31, R=624 and w=32, Fig. 1 represents the state transition process for 32-bit WELL19937. The other coefficients M1, M2, M3 are 70, 179 and 449, respectively.

B. Fast Jump Ahead Technique

To generate multiple independent sub-streams, we need to efficiently jump from a certain state S_n to a far-away state . Considering the state transition of all F_2 -linear PRNGs can be expressed in the form of matrix multiplication [11], Jump Ahead with step length V is essentially equivalent to computing the equation (1) as follows.

$$S_{n+v} = A^v S_n \qquad (1)$$

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Using the standard square-and-multiply exponentiation method [3] requires $O(k^3 \log v)$ operations and k^2 bits of memory to store A^v. This approach is only suitable for those simple algorithms with a small k, like LFSRs. For WELL19937, the exponentiation operation is slow and the 19937×19937-bit A^v needs about 47.4 MB of memory! Based on the characteristic polynomials, the Fast Jump Ahead Technique [5] solves this problem with a higher speed and less memory consumption. Consider the characteristic polynomial of matrix A.

$$cp(z) = \det(zI + A) = z^k + a_1 z^{k-1} + \cdots + a_{k-1} z + a_k \quad (2)$$

Where I is the identity matrix and $a_i \in F_2$. According to, the fundamental property of the characteristic polynomial,

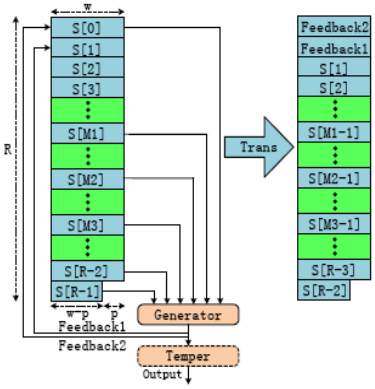


Figure 1. State Transition Process of WELL.

we can conclude.

$$cp(A) = A^k + a_1 A^{k-1} + \cdots + a_{k-1} A + a_k I = 0 \quad (3)$$

Let a polynomial

$$g(z) = z^v \bmod cp(z) = a_1 z^{k-1} + \cdots + a_{k-1} z + a_k \quad (4)$$

Observe that $g(z) = z^v + q(z)cp(z)$

for some polynomial $q(z)$ and $cp(A) = 0$,

we get $g(A) = A^v$, so,

$$\begin{aligned} A^v S_n &= (a_1 A^{k-1} + \cdots + a_{k-1} A + a_k I) S_n \\ &= A(\ldots A(A(A a_1 S_n + a_2 S_n) + a_3 S_n) + \cdots + \\ &\quad a_{k-1} S_n) + a_k S_n \end{aligned} \quad (5)$$

As shown in Eqn.(5), the new state can be computed by a series of AS_n and additions (XORs) of the initial state vector S_n. Notice that at most k-1 such operations are required, thus for any step length, the computation requirement is roughly the same.

III. HARDWARE ARCHITECTURE FOR WELL19937

Fig. 2 illustrates our hardware architecture for WELL19937. It consists of five blocks: the Control Unit, the Address Unit, the Generate Unit, the Temper Unit and a 6R/2W RAM. The core component is the RAM, which stores the 624 32-bit state vectors and is capable of concurrently supporting 6 Read and 2 Write operations. The Address Unit generates appropriate R/W addresses for the RAM. The Generate Unit and the Temper Unit compute the Generate and Temper operations of Fig. 1, and can be fully pipelined. The Control Unit produces the control signals to coordinate the system.This architecture has two advantages: 1) High throughput: Utilizing a BRAM-and-register-hybrid structure, the system achieves a throughput of 1 random number per cycle. 2) Does not require external resources such as off-chip memory. The system can be built on a single FPGA device.

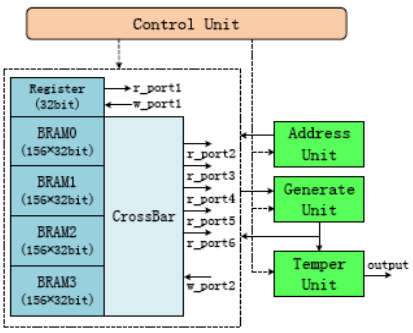


Figure 2. WELL19937 Hardware Architecture Overview.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

```
1. let addr[0..5] = {69,178,448,621,622,624}
2. for  i = 1 to numbers_to_be_generated do
3.     output addr[0..5] as the access addresses
4.     for j = 0 to 5 do
5.        if addr[j] equal to 0 then
6.           addr[j] = 624
7.        endif
8.        addr[j] = addr[j] - 1;
9.     endfor
10. endfor
```

Figure 3. Pseudo-code for generating the access addresses implemented by the Address Unit.

A. Structure of the 6R/2W Multi-port RAM

As shown in Fig. 1, in each transition process, six blocks from the state vector are fetched while two blocks are updated. Therefore, to achieve the expected throughput, the RAM should be able to read six 32-bit operands and store two 32-bit feedbacks concurrently in a single cycle. Such a RAM can be directly implemented using 624 32- bit registers, but this is not area-efficient and is impractical when building multiple parallel PRNGs. It is also not straightforward to provide 8 ports by simply assembling 4 BRAMs together, since we need to guarantee that the read and write operations are distributed across different BRAMs evenly. Instead, we propose a BRAM-and-register-hybrid structure to build the required 6R/2W Multi-port RAM, which is the key component to achieve 1 random number percycle throughput.

As shown in Fig. 1, the state vector S[0] is read and updated in each cycle. We therefore can use a single register to store S[0] and provide the necessary 1R/1W operations. The remaining 5R/1W operations can be provided by 4 dualported 156×32-bit BRAMs. Fig. 3 gives the pseudo-code for generating the access addresses for the Read ports from r_port2 to r_port6 (addr[0..4]) and the Write port w_port2 (addr[5]). The Crossbar implements the mapping rules described in Fig. 4, to forward the 5R/1W ports to the appropriate BRAMs. Where the access address is generated by the Address Unit using the pseudo-code in Fig. 3

```
BRAM ID                = access address mod 4
internal BRAM address  = access address / 4
```

Figure 4. Address mapping rules implemented by the Crossbar.

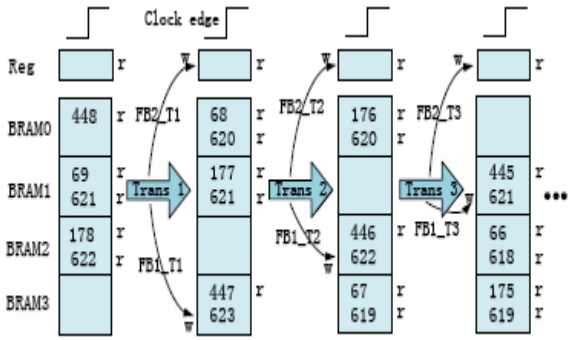


Figure 5. R/W details during the first 3 transition steps.

Based on the mapping rules, the starting access addresses of the 5R/1W ports (as shown in line 1 of Fig. 3) are mapped into BRAM[1, 2, 0, 1, 2, 0], respectively. During runtime, each of these addresses is updated synchronously by the same counter and traverses the BRAMs in exactly the same manner. Therefore, no BRAM will have more than 2 accesses in a single cycle. Fig. 5 illustrates the R/W details during the first 3 transition steps. Where FB2_Tx and FB1_Tx are the two results generated in the xth transition step. The numbers in the cells (448,69,621, etc) correspond to the access addresses generated by the Address Unit, which are mapped to the appropriate BRAMs. We can see that 6 Read and 2 Write operations can be completed in a single cycle. Thus our BRAM-and-register-hybrid structure can successfully satisfy the required memory accesses.

IV.SOFTWARE/HARDWARE FRAMEWORK

Due to the intrinsic limitation of the WELL algorithm, i.e. one operand of a new transition directly depends on a result from the previous iteration (Feedback2), it is difficult to parallelize WELL using the Interleaved Parallelization approach

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

as in [9] and [10]. To tackle this problem, we propose a software/hardware framework using the Fast Jump Ahead Technique [5], as shown in Fig. 6. The Jump Ahead Unit in Software takes the following responsibilities: 1) Generating the initial vector states for each PRNG according to user configurations, i.e. the total random numbers, the number of cores in hardware and the initial seed. 2) Offloading initial state vectors to the hardware. 3) Collecting results from simulation and doing postcomputations. The core component of the hardware is a PRNG Array consisting of a number of parallel WELL PRNGs. It can be constructed by simply replicating the single generator described in Section III. After receiving proper state vectors, an array of size N can produce N random numbers in parallel in every clock cycle. These numbers are denoted as $y_0, y_{v_1} \cdots y_{v_{n-1}}$. The output(s) of the PRNG Array can also be directly connected in parallel via the logic fabric to the destination application on the FPGA to achieve the highest possible throughput.

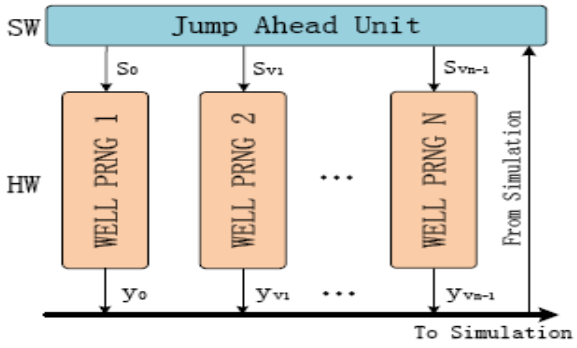


Figure 6. Software/Hardware Framework Overview.

A protocol is introduced to support the communications between software and hardware. It defines a series of instructions that describe all the necessary operations from the software, e.g. WRITE VECTOR STATEs instruction, PAUSE/START instruction and so on. It also contains some instructions to support the collection of simulation results from the hardware. The protocol is platform-independent, and can be implemented over links such as a UART, PCIE or Ethernet.

A. Algorithm to Get the Characteristic Polynomial

A key step of the Fast Jump Ahead Technique is to calculate the characteristic polynomial "cp(z)" of matrix A in Eqn. (2). Considering the size (19937×19937) and complexity of the transition matrix A of WELL19937, it is prohibitive to directly c l ul te "cp (z)". To address this problem, we develop a fast algorithm as described in Fig. 7. In this algorith , "cp(z)" can be obtained using only two processes, no matter how large the jump length is: 1) Invoke the WELL19937 routine 2K times, where K = 19937; 2) Use the Berlekamp-Massey algorithm [12] to derive all the coefficients. Moreover, this algorithm can be applied to find the characteristic polynomial for any other -linear PRNGs, e.g. MT19937 or WELL44497, by modifying the coefficient K and the calling routine.

V. IMPLEMENTATIONS AND EVALUATIONS

This section presents implementations and evaluations of the proposed architecture and framework using FPGA technology.

A. Single WELL Generator

We implement the architecture described in Section III on a Xilinx Virtex-6 XC6VLX240T (hosted on the ML605 evaluation board). Described in Verilog HDL, the design is synthesized and implemented using Xilinx ISE 11.5. The initial design is simulated in Modelsim SE 6.5 to ensure functional correctness. Optimization techniques, such as register-retiming, are applied to improve the clock speed of the design

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

1. let K be a constant integer with value 19937
2. let W be a constant integer with any value between 0 and 31;
3. let RND be a 32-bit unsigned integer;
4. let LinerGenSeq be an array of Boolean Type;
5. let CharPolyCoef be an array of Boolean Type;
6. do some initializations;
7. for i = 0 to 2K-1 do
   1> Invoke the WELL19937 routine once and obtain a 32-bit random number RND;
   2> Extract the Wth bit of RND and assign it to LinerGenSeq[i];
   end for
8. Compute the minimum polynomial of the sequence LinerGenSeq using Berlekamp-Massey algorithm and assign the result to Char-PolyCoef.
9. output: CharPolyCoef contains the coefficents of cp(z)

Figure 7. Algorithm for obtaining "cp(z)" for WELL19937.

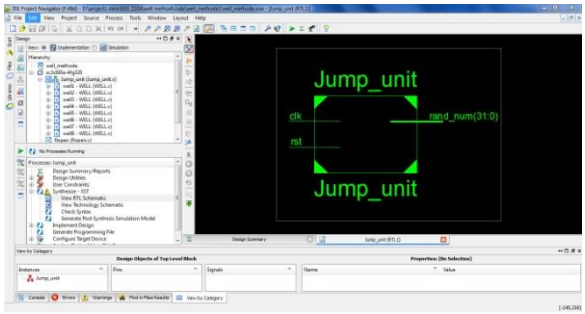
The four BRAMs are configured to enable the embedded output registers. Since one result of previous iteration is a source operand for the next transition process, this value is directly bypassed to the next iteration. The whole system is fully pipelined. Three reference designs are implemented for comparison: a hardware generator based on MT19937, and two software generators of WELL19937 and MT19937. The MT hardware generator utilizes a similar structure to those described in [9][10]. It is built on exactly the same device as our design with completely the same configurations. The software designs are based on the codes provided by [2][1] and run on a 2.93-GHz Intel Core processor with 3GB DDR3 SDRAM.

Table I summarizes the comparisons between different implementations, in terms of resource usage and maximum performance. Our proposed generator achieved a 7.6-fold speed up compared to its optimized software version. Due to the complexity of the algorithm, the WELL generator consumes roughly twice the resources than a MT generator. However, the throughputs of the two generators are comparable. Moreover, it is worth noting that the resource usage of the WELL generator is just about 0.50% of the device, which is a negligible.

B. Framework evaluation

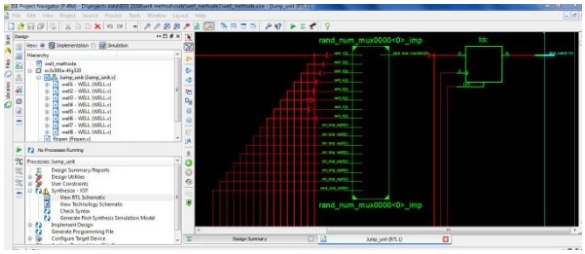
The performance of the framework is evaluated from two aspects. For the software, the characteristic polynomial "cp (z)" in Eqn. (2) is calculated using the algorithm described in Fig. 7. It takes about 51ms. The number of nonzero coefficients of the "cp(z)" generated by our algorithm is 8585, which matches the data given in [2]. $"z^v \bmod cp(z)"$ in Eqn. (4) is pre-computed using a dedicated software library called the *Number Theory Library* [14], which contains efficient algorithms to perform polynomial operations. The average times for the jump process in Eqn. (5) with different steps are presented in Table II. Results show that the jump process can be completed within a few milliseconds regardless how long the distance is. Thus it is possible to quickly provide the new initial states to each PRNG in the hardware as the seed changes. For the hardware, we have implemented the PRNG array with parallel degrees of 1, 2, 4, 8, 12 and 16. Initial states are generated by software and then directly written into each generator. The resources usage and maximal performance statistics are shown in Table III. We also plot corresponding throughput vs. area, along with a linear least-squares fit, as shown in Fig. 8. We can see that the throughput/area efficiency roughly remains constant as the degree of parallelization increases

VI. RESULTS

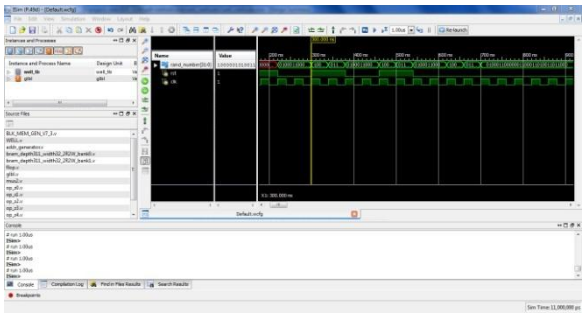


A . RTL BLOCK DIAGRAM

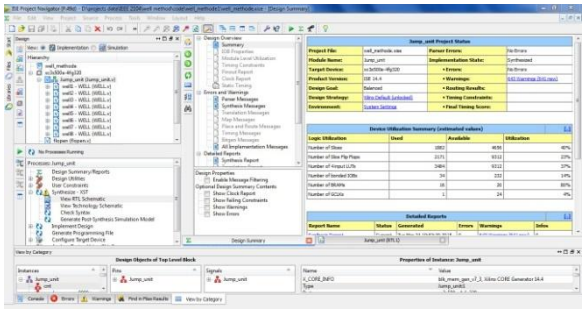
ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal



B. RTL INTERNAL DIAGRAM



C.WAVEFORM



D. AREA REPORT

VII. CONCLUSION

Through our study, we demonstrate that our proposed 1 sample per cycle hardware architecture for the WELL19937 algorithm achieves high performance, low area cost and high quality output at the same time. It runs as fast as 449.4 MHz on a Xilinx XC6VLX240T FPGA, which is 7.6-fold faster than its dedicated software version and is comparable to the MT19937 hardware implementation. At the same time, it is small in area: takes only 633 LUTs, 537 Flip-Flops and 4 BRAMs. Based on the Fast Jump Ahead Technique, we develop a software/hardware framework to parallelize the WELL19937 sequence. With the support of software, the performance and area cost of our hardware design scales linearly with the degree of parallelization. Finally, we successfully pass the quality test of the Diehard and TestU01 test suites, as well as the Monte-Carlo simulation for π estimation. We expect its successful use in various Monte-Carlo simulations and other applications.

REFERENCES

[1] M.Matsumoto and T.Nishimura, "Mersenne twister: a 623- dimensionally equidistributed uniform pseudo-random number generator," ACM Trans. Modeling and Computer Simulation, vol.8, no 1, pp.3-30, Jan. 1998.

[2] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," ACM Trans. Mathematical Software, vol.32, no.1, pp.1-16, Mar. 2006.

[3] D. E. Knuth, "The Art of Computer Programming, Volume 2: Seminumerical Algorithms," Addison Wesley, Reading, Mass., third edition, 1998.

[4] Ukalta Engineering Corporation, "Uncorrelated Pseudo-Random Number Generator IP Cores," Product Brief, 2009.

[5] H. Haramoto, M. Matsumoto, T. Nishimura, F. Panneton, and P. L'Ecuyer, "Efficient jump ahead for F2-linear random number generators," Informs Journal. Computing, vol.20, no.3, pp.385-390, summer 2008.

[6] T.Kurokawa, and H. Kajisaki, "FPGA based implementation of Mersenne Twister," Scientific and Engineering Reports of the National Defense Academy, vol.40, no.2, pp.15-21, Mar. 2003.

[7] G. Marsaglia, "DIEHARD: A Battery of Tests of Randomness", http://stat.fsu.edu/.geo/diehard.html, 1997.

[8] V. Sriram, and D. Kearney, "An area time efficient field programmable Mersenne Twister uniform random number generator," In Proc. Int. Conf. Eng. of Reconfigurable Systems & Algorithms,pp.244-246, 2006.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

[9] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudo-random number generator MT19937," IEICE Trans. Info. and Systems, vol.88, no.12, pp.2876-2879, Dec. 2005.

[10] I. L. Dalal, and D. Stefan, "A Hardware Framework for the Fast Generation of Multiple Long-period Random Number Streams," in Proc. 16th ACM int. symp. FPGAs, Feb. 2008, pp. 245-254.

[11] P L'Ecuyer, and F. Panneton, "Fast random number generators based on linear recurrences modulo 2: overview and comparison," in Proc.37th conf. Winter simulation, Dec. 2005,pp.110-119.

[12] J.L. Massey, "Shift-register synthesis and BCH decoding," IEEE Trans. Inform. Theory, vol.15,no.1, pp.122-127, Jan.1969.

[13] Pierre L'E uyer n Richard Simard. TestU01: A C library for empirical testing of random number generators. ACM Transactions on Mathematical Software (TOMS), 33(4), Aug. 2007.

[14] Victor Shoup, "NTL:A Libr ry f r ing Nu ber The ry,"