International Journal of Innovative Technologies
WWW.IJITECH.ORG
ISSN 2321-8665
Vol.04,Issue.01,
January-2016,
Pages:0072-0076

Adaptive Shared Hardware and Software Parallel Random Number Well Based Framework Generation

VANGA SRINIVAS[1], B. RANJITH[2]
[1]PG Scholar, Dept of ECE, Vaagdevi College of Engineering, Bollikunta, Sangem, Warangal, TS, India,
E-mail: vanga.srinivas91@gmail.com.
[2]Assistant Professor, Dept of ECE, Vaagdevi College of Engineering, Bollikunta, Sangem, Warangal, TS, India.

Abstract: This paper presents hardware architecture for efficient implementation of the well undistributed long-period linear (WELL) algorithm. Our design achieves a throughput of one sample-per-cycle and runs as fast as 423 MHz on a XilinxXC5VFX130T field-programmable gate array (FPGA) device. This performance is 7.1-fold faster than a dedicated software implementation. The proposed architecture is also implemented on targeting different devices for the comparison of other types of true random number generators. In addition, 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 sequences produced by the single design are verified to be consistent with the standard software generator. In addition, the statistical tests of interleaved sequences are also performed to check for correlations between different sub streams of the parallel framework. We apply our frame work to two applications. Experimental results verify the correctness of our framework as well as the better characteristics of the WELL algorithm compared with the Messene Twister method.

Keywords: Well Undistributed Long-Period Linear (WELL), BRAM, Random Number Generators (RNGs).

I. INTRODUCTION

Random numbers are of paramount importance. Not only are they needed for gambling, they find applications in cryptography, statistical data sampling, as well as computer simulation (e.g., onte Carlo simulations). In principle, they are needed in any application where unpredictable results are required. For most applications it is desirable to have fast random number generators (RNGs) that produce numbers that are as random as possible. However, these two properties are often inversely proportional to each other: excellent RNGs are often slow, whereas poor RNGs are Typically fast. The true random number generator is fairly involved since it needs to preserve and amplify the thermal noise, and at the same time shield all external disturbances. It consists of mainly analog components and cannot be implemented by pure digital circuitry. The mixed-signal implementation significantly increases the system complexity. This implementation is also relatively slow and cannot match the high-speed digital circuit. One major application of the true random number is to generate the initial seed for pseudo random number generator. Compared with MT,WELL has better Equi-distribution while retaining an equal period length.

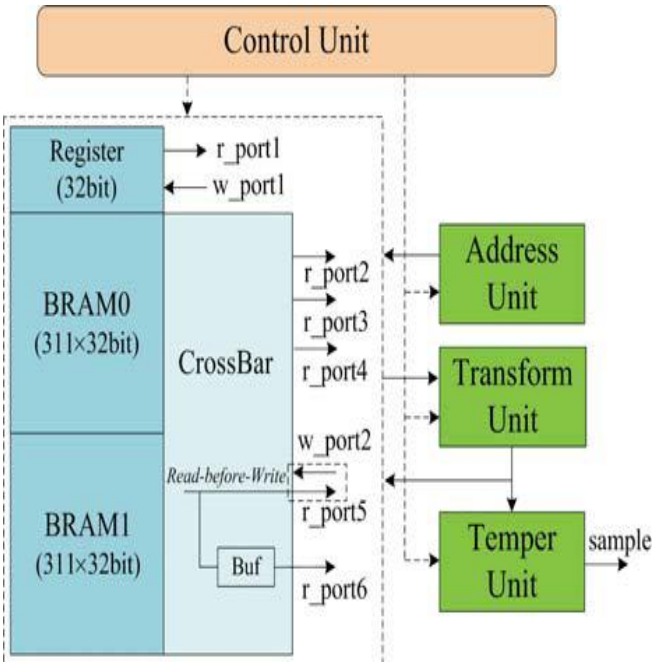

Fig.1. WELL19937 hardware architecture overview86 Int. J. Adv. Eng., 2015, 1(3), 84-89.

We propose a more resource-efficient structure that reduces the usage of BRAMs from four to woo, while retaining the same throughput. The total resource used is also reduced as much as 50% compared with the original structure. We also design a software/hardware framework to parallelize its output stream based on the new structure. More specifically, we make the following contributions.

- A resource-efficient hardware architecture for WELL with a throughput of one sample per cycle.

- A dedicated 6R/2W RAM structure for WELL, which is capable of providing six Reads and two Writes concurrently in a single cycle, with little resource overhead.
- A software/hardware framework to generate parallel random numbers.

We make the following improvements.

- Section II is the main contribution of this paper and is totally rewritten to describe the improved TWOBRAMs- based WELL architecture.
- In Section IV, the implementation of the WELL structure is discussed more detailed and the comparison is enhanced by introducing more types of FPGA-based PRNGs. The evaluation results are also analyzed in more depth.
- In Section V, the statistical testing is enhanced by adding parallel testing to check for correlations between different sub streams.85 Int. J. Adv. Eng., 2015, 1(3), 84-89.

II. SYSTEM ANALYSIS

A. Existing system

Pseudorandom - is defined as having the appearance of randomness, but nevertheless exhibiting a specific, repeatable pattern. Numbers calculated by a computer through a deterministic process, cannot, by definition, berandom. In existing system, pseudo-random sequence generators are defined and their properties discussed. These are called"(1) The 1 /P generator,(2) The x2 mod N generator. The two generators are closely related. For example: From short seeds, each quickly generates long well-distributed sequences. Both generators contain hard problems at their core (the discrete logarithm problem and the quadraticresidua city problem, respectively). But only the second is "unpredictable"--assuming a certain intractability Hypothesis. Any sequence produced by the 1/P generator is completely predictable; that is, given a small segment of the sequence, one cart quickly infer the "seed" and efficiently extend the given segment backwards and forwards output. The digital low pass filter will probably be realized by a digital circuit or by an algorithm within a signal processor. The delta sigma modulator is the core of delta sigma converters. Additional advantages of such an approach include higher reliability, increased functionality, and reduced chip cost. Those characteristics are commonly required in the digital signal processing environment of today. Consequently, the development of digital signal processing technology in general has been an important force in the development of high precision A/D converters which can be integrated on the same die as the digital signal processor itself.

B. Proposed System

Truly random - is defined as exhibiting ``true" randomness. A true random number generator uses entropy sources that already exist instead of inverting them. Entropy refers to the amount of uncertainty about an outcome. Real world events such as coin flips have a high degree of entropy, because it is almost impossible to accurately predict what the end result will be. It is the source of entropy that makes a true random number generator unpredictable. Flipping coins and rolling dice are two ways entropy could be obtained for a generator, although the rate at which random numbers could be produced would be restricted. Since they use real world phenomena, some physical device capable of recording the event is needed. This can make true random generators a lot more expensive to implement, especially if the necessary device is not commonly used. It also means that the generators are vulnerable to physical attacks that can bias the number sequences. A good source of random numbers, where any structure of any kind would be difficult to obtain, is typically needed when the observer is a human, he can profit from any small deviation, and he can gain access to large amounts of random numbers to analyze on a computer.

III. HARDWARE ARCHITECTURE FOR WELL19937

We presented a BRAM-and-register-hybrid architecture for WELL19937 with a throughput of one sample per-cycle. We propose a more resource-efficient structure that reduces the usage of BRAMs from four to two, while retaining the same throughput. The total resource used is also reduced as much as 50% compared with the original structure. We also design a software/hardware framework to parallelize its output stream based on the new structure. More specifically, we make the following contributions.

- A resource-efficient hardware architecture for WELL with a throughput of one sample per cycle.
- A dedicated 6R/2W RAM structure for WELL, which is capable of providing six Reads and two Writes concurrently in a single cycle, with little resource overhead.
- A software/hardware framework to generate true random numbers.

Fig.1 shows our hardware architecture for WELL19937. It consists of five blocks:

- Control Unit
- Address Unit
- Transform Unit
- Temper Unit
- 6R/2W RAM

The core component is the RAM, which stores the 624 32-bit state vectors and is capable of concurrently supporting six Reads and two Writes. The Address Unit generates appropriate R/W addresses for the RAM. The Transform Unit and the Temper Unit perform the Transform and Temper operations of the WELL algorithm, and can be fully pipelined. The Control Unit produces the control signals to coordinate the system. Structure of the 6R/2W Multiport RAM Based on the transformation process of WELL algorithm in each generation process, six blocks from the state vector are fetched while two blocks are updated. Therefore, to achieve the expected throughput, the RAM should read six operands and store two results 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 parallel PRNGs. It is also not straight forward to provide eight ports by simply assembling four BRAMs together, as 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 multiport RAM, which is the key component to achieve one sample per cycle throughput. 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 BRAMs of the FPGA allow a Read-before-Write operation, i.e., when writing anew data into an address, the data previously stored at this address can be fetched concurrently in the same clock cycle. Using this feature, the w_port2 and the r_port5 can be provided by a single port of a BRAM thus saving one R/W operation. Since the Read addresses of the r_port5 and r_port6 are always adjacent, the data from r_port5 can be buffered and reused by r_port6 in the next clock cycle. This saves one more Read operation. By utilizing these two optimizations, the remaining six R/W operations are decreased to only four operations. This can be provided by two dual-ported 311 × 32-bit BRAMs. Assume the Transform Unit is divided into a 1-stage pipeline. Based on the mapping rules, in the first three clock cycles, the addresses of the R/W ports are mapped into BRAM[1, 0, 0, 1], RAM[0, 1, 1, 0], and BRAM [1, 0, 0, 1],Respectively. During runtime, each of these addresses is updated synchronously by the same outer and traverses the BRAMs in exactly the same manner. The write address for w_port2 is always the same as the Read address of r_port5and can be accessed concurrently through the Read before- Write operation. Therefore, no BRAM will need more than two ports in a single cycle. The R/W details during the first four clock cycles, where FB1 and FB2 are the two results Generated in the transform step. The numbers in the cells (178, 448, 69, 621, and so on) correspond to the access addresses generated by the Address Unit, which are mapped to the appropriate BRAMs. The R/W accesses to a same address are performed using the Read-before-Write operation and a buffer is used to cache the data shared by r_port5and r_port6. We can see that 6R/2W operations can be finished in a single cycle. Thus, our BRAM-and-register-hybrid Structure can successfully satisfy the required memory accesses.

IV. SOFTWARE/HARDWARE FRAMEWORK

Fig2 shows the structure of the framework.The jump ahead unit in Software is responsible for: generating the initial vector states for each PRNG according to user configurations, i.e., the total random numbers, the number of simulation cores and the initial seed; 2) offloading initial state vectors to the hardware; and 3) dealing with the simulation results. The core component of the hardware is a PRNG Array consisting a number of parallel WELLPRNGs. It can be constructed by simply replicating the single generator. After receiving proper state vectors, an array of size n can produce n random numbers in parallel in every clock cycle, which is denoted as $y_0, y_{v1}. . . y_{vn-1}$

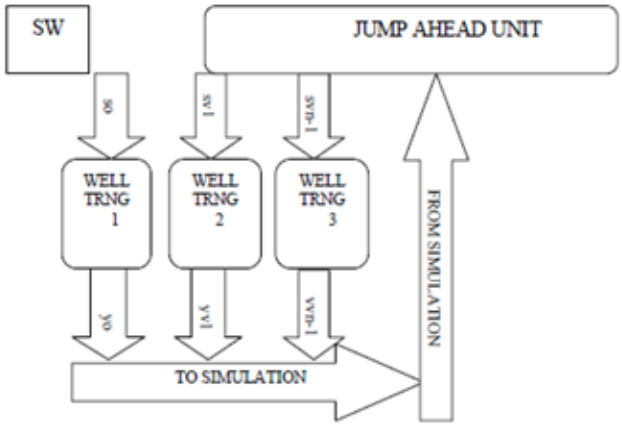

Fig.2. Software/hardware framework overview.

V. IMPLEMENTATIONS AND EVALUATIONS

A. Single Well Generator

Fig. 4 shows the detailed hardware implementation of the WELL architecture. The crossbar is implemented using eight multiplexers. Based on the address mapping rule shown in Fig. 2, the least significant bit of the access addresses are used as the selector while the other nine bits are used as the internal addresses. The embedded output registers of both BRAMs are enabled to improve the clock speed. As one result of the previous iteration is a source operand for the next Generation process, this value is directly passed to the next iteration.The Verilog source code of the implementation is available at http://sdrv.ms/VomTfS. As expected, the improved WELL generator was more resource-efficient (it saves as much as 50% FPGA resource) compared with the original structure.

TABLE I: Comparison of Period, Quality, Resource Usage and Throughput

	Platform	Slices	FFs	LUTs	Multipliers	BRAM 36k	BRAM 18k	Period (log2)	Frequency (Mhz)	Throughput (Gb/s)
This Paper	V5	112	259	317	-	2	-	19937	423	13.5
WELL 19937 [10]	V5	168	535	571	-	4	-	19937	424	13.6
WELL 19937 [2]	SW	-	-	-	-	-	-	19937	-	1.90
MT 19937 [8]	SW	-	-	-	-	-	-	19937	-	6.03
MLFG [12]	V2-Pro	540	-	-	10	-	24	1340	65.3	2.1
LFSR 160 [11]	V2	-	451	481	-	-	-	160	252	8.1
Taus 113 [11]	V2	-	164	161	-	-	-	113	287	9.1

Table I shows the comparison of a number of FPGA-based PRNGs, in terms of quality metric, resource usage, and performance. For statistical testing, both WELL and MT fail only two linear complexity tests. However, the characteristic polynomial of MT19937 has only 135 nonzero coefficients out of 19937. In this condition, when the state vector is initialized with a relatively large fraction of zeros, then only a small part of the state will be changed at every generation process and the change to the state will also be very small.

This tendency likely continues for many steps and causes the MT generator taking a very long time to recover from zero-excess states. WE L corrects this weakness by keeping its characteristic polynomial with the number of nonzero coefficients close to half the degree (e.g., the nonzero number is 8585 out of 19937 for WELL19937). In addition to this, WELL also has better equi-distribution compared with MT. The LFSR-160 and Tauss-113 behave particularly badly and fail multiple tests, including those do not depend on the linear structure of the generator. The integer generators MLFG and CMRG from HSPRNG (that are based on recurrence modulo a positive integer) pass all tests, but the decimal computations significantly slow them down, thus their performance/cost ratio is not attractive. The LUT-SR- 19937 also fails the two linear tests and is of about the same quality as WELL. Although consuming more resource, the throughput of LUT-SR greatly surpass WELL. This is normal because LUT-SR is optimized specifically for FPGA.

B. Framework Evaluation

The framework is evaluated from two aspects. For the software, we evaluate the average time for the jump process with different steps. Simulation results show that the jump process can be completed within a few milliseconds regardless ofthe jump distance. For the hardware, we implemented the PRNG array with 1, 4, 8, 12, and 16 simulation cores. Initial states are generated by software and then directly written into each generator. The resources usage and performance are shown in Table (2). We can see that the throughput/area efficiency roughly remains constant as the number of simulation cores increases.

TABLE II: Resources Usage and Performance for Different Parallel Degrees

No Of Cores	1	4	8	12	16
Flip-Flops	259	1036	2072	3108	4144
LUT s	317	1268	2536	3804	5072
BRAM s	2	8	16	24	32
Frequency(MHz)	423	413	404	397	396
Throughput(M/s)	423	1652	3232	4764	6336
Speed	1.0	3.9	7.6	11.3	15.0

VI. STATICAL TESTING

The statistical testing is two-fold: 1) the sequential testing to check for correlations within a stream and 2) the parallel testing to check for correlations between different sub streams. For the sequential testing, we just have verified that the outputs of the hardware and the standard software version are exactly the same when starting with the same seed. The parallel testing is performed by interleaving different sub streams into a single stream. For example, if the parallel degree is n and the stream i is given by $x_{i,0}$, $x_{i,1}$, $x_{i,2}$, . . ., then the new stream is in the form of $x_{0,0}$, $x_{1,0}$, . .. , $x_{n-1,0}$, $x_{0,1}$, $x_{1,1}$, . . ., $x_{n-1,1}$, ... We apply the new stream to the standard statistical test suites, Diehard and Crush from Test U01. Testing results show that the parallel generators pass all tests, which verify the independence of different parallel streams generated by our framework. We apply the WELL framework to construct a framework for producing parallel Gaussian variables, based on the Box–Muller Method. The table-polynomial-hybrid method was adopted for the approximation of the elementary functions (sin, square root, and so on). Floating-point operations were converted into fixed-point operations and the word-length optimization model was used to maximize the performance/cost efficiency.

VII. RESULT

The true random number is generated with one sample per cycle using the WELL algorithm is shown below Fig.3.

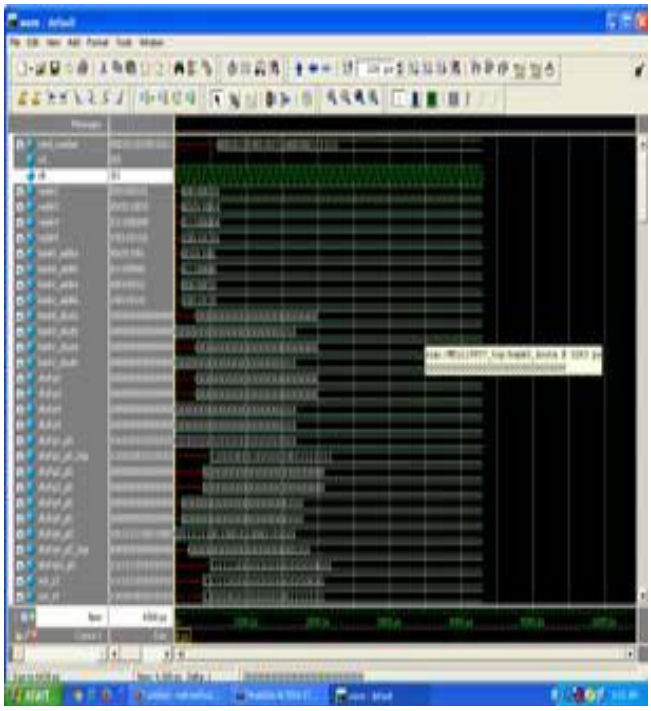

Fig.3. Output waveform.

VIII. CONCLUSION

Through our study, we demonstrated that our proposed one-sample-per-cycle hardware architecture for the WELL algorithm achieved high performance, low area cost, and high quality output at the same time. The SW/HW framework we develop could parallelize the WELL sequence into arbitrary number of independent parallel sub streams and was successfully applied to two applications. We expect its successful use in various Monte Carlo simulations and other Applications.

IX. REFERENCES

[1] M. Matsumoto and T. Nishimura, “Mersenne twister: A 623- dimensionally equidistributed uniform pseudo-random number generator,” ACMTrans. Model. Comput.Simul., 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. Math.Softw., vol. 32, no. 1, pp. 1–16, Mar. 2006.

[3] P. L'Ecuyer and F. Panneton, "Fast random number generators based on linear recurrences modulo 2: Overview and comparison," in Proc. 37thConf. Winter Simul., 2005, pp. 110–119.
[4] H. Haramoto, M. Matsumoto, T. Nishimura, and P. L'Ecuyer, "Efficient jump ahead for F2-linear random number generators," Inf. J. Comput.,vol. 20, no. 3, pp. 385–390, 2008.
[5] V. Sriram and D. Kearney, "An area time efficient field programmable Mersenne Twister uniform random number generator," in Proc. Int. Conf.Eng. Reconfigur. Syst. Algorithms, 2006, pp. 244–246.
[6] C. Shrutisagar and A. Abbes, "High performance FPGA implementation of the mersenne twister," in Proc. 4th IEEE Int. Symp.Electron. Design,Test Appl., Jan. 2008, pp. 482–485.
[7] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudorandom number generator MT19937," IEICE Trans. Inf. Syst., vol. 88,no. 12, pp. 2876–2879, Dec. 2005.
[8] I. L. Dalal and D. Stefan, "A hardware framework for the fast generation of multiple long-period random number streams," in Proc. 16th ACMInt. Symp. FPGAs, Feb. 2008, pp. 245–254.
[9]Uncorrelated Pseudo-Random Number Generator IP Cores, Ukalta Engineering Corporation, Edmonton, AB, Canada, 2009.
[10] Y. Li, P. Chow, J. Jiang, and M. Zhang, "Software/ hardware framework for generating parallel long-period random numbers using the WELLmethod," in Proc. 21st Int. Conf. Field Program. Logic Appl., Sep. 2011, pp. 110–115.
[11] D. B. Thomas and W. Luk, "High quality uniform random number generation through LUT optimised linear recurrences," in Proc. IEEE Int.Conf. Field Program. Technol., Dec. 2005, pp. 61–68.
[12] J. Lee, G. D. Peterson, and R. J. Hinde, "Hardware accelerated scalable parallel random number generators for Monte Carlo methods," in Proc.51st Midwest Symp. Circuits Syst., 2008, pp. 177–180.
[13] D. B. Thomas and W. Luk, "The LUT-SR family of uniform random number generators for FPGA architectures," IEEE Trans. Very LargeScale Integrat. (VLSI) Syst., vol. 21, no. 4, pp. 761–770, Apr. 2013.
[14] G. Marsaglia. (1997). DIEHARD: A Battery of Tests of Randomness [Online]. Available: http://stat.fsu.edu/.geo/ diehard.html
[15] P. L'Ecuyer and R. Simard, "TestU01: AC library for empirical testing of random number generators," ACM Trans. Math. Softw. (TOMS), vol.33, no. 4, p. 22, Aug. 2007.
[16] G. Box and M. Muller, "A note on the generation of random normal deviates," Ann. Math. Stat., vol. 29, no. 2, pp. 610–611, 1958.

Author's Profile:

Vanga Srinivas completed his B tech in Vaagdevi College of Engineering at Bollikunta in Warangal and pursuing M-Tech in Vaagdevi College of Engineering at Bollikunta in Warangal. vanga.srinivas91@gmail.com

Mr. B.Ranjith is working as Asst. prof in Dept of ECE in Vaagdevi College of Engineering at Bollikunta in Warangal.