Hardware Accelerated Scalable Parallel Random Number Generators

JunKyu Lee, Gregory D. Peterson, Robert J. Harrison
The University of Tennessee

Abstract

The Scalable Parallel Random Number Generators (SPRNG) library is widely used for scientific applications. A hardware implementation of SPRNG on Xilinx Virtex II Pro FPGAs introduces 4-15 times speedup over 2.8GHz Pentium 4 processors. The Hardware Accelerated Scalable Parallel Random Number Generators (HASPRNG) library produces identical results with SPRNG. This paper describes the implementation of HASPRNG, the verification platforms, and its performance.

1. Introduction

The SPRNG library was developed to support large numbers of parallel, independent streams of random numbers for supercomputing applications such as Monte Carlo simulations [1]. SPRNG consists of 6 types of random number generators: Modified Lagged Fibonacci Generator (Modified LFG), 48bit Linear Congruential Generator with prime addend (48bit LCG), 64bit Linear Congruential Generator with prime addend (64bit LCG), Combined Multiple Recursive Generator (CMRG), Multiplicative Lagged Fibonacci Generator (Multiplicative LFG), and Prime Modulus Linear Congruential Generator (PMLCG). This paper introduces the Hardware Accelerated Scalable Parallel Random Number Generators (HASPRNG) library which seeks to accelerate each of the SPRNG generators using FPGAs. In order to simplify adoption of HASPRNG for users, the HASPRNG produces identical results to SPRNG for the same generator, parameters, and seeds. The programming interface is also nearly identical to simplify integration with existing codes.

2. HASPRNG Implementation

Each of the SPRNG library random number generators are implemented in HASPRNG. We now give a brief overview of each generator and its hardware implementation. The Modified LFG uses different architectures depending on whether the lag values are even or odd [2]. The generator produces a random number every clock cycle.

The two LCGs have the same architecture. The 48 and 64bit LCGs are characterized by the following equation:

$$Z(n) = a \times Z(n-1) + p \ (\text{Mod } M)$$

where *p* is a prime number, *a* is the multiplier, and Z(n) is the *n*th random number. M is 2^{48} for the 48bit LCG and 2^{64} for the 64bit LCG. The LCGs use 7 stage pipelined multipliers. Thus, the LCGs in HASPRNG transform the equation as follows:

$$Z(n) = a' \times Z(n-8) + p' \ (\text{Mod } M)$$
$$a' = a^8,\ p' = p \times (a^7+a^6+a^5+a^4+a^3+a^2+a^1+1)$$

This equation allows the generator to generate a random number in each clock cycle.

The CMRG satisfies the following equations:

$$Z(n) = X(n) + Y(n) \times 2^{32} \ (\text{Mod } 2^{64})$$
$$Y(n)=107374182\times Y(n-1)+104480\times Y(n-5)(\text{Mod } 2^{31}-1)$$

where X(n) is generated by 64 bit LCG and Z(n) is the resulting random number. The architecture has two parts, each generating a partial result. The first part is a 64bit LCG as above, and the second part is the generator having two lag factors. The second part is composed of 2 multipliers, 4 deep FIFO registers, and combinational logic. The CMRG generates a random number every other clock cycle.

The PMLCG implements the following equation:

$$Z(n) = a \times Z(n-1) \times 2^{32} \ (\text{Mod } 2^{61}-1)$$

where *a* is a multiplier and Z(n) is the resulting random number. The PMLCG uses 4 two-stage pipelined multipliers. The PMLCG generates a random number every other clock cycle.

The multiplicative LFG satisfies the following equation:

$$Z(n) = Z(n-k) * Z(n-l) \ (\text{Mod } 2^{64})$$

where *k* and *l* are time lags and Z(n) is the resulting random number. The MLFG uses dual port RAMs (DPRAMs) to hold the previous results (lags). The two values read from the DPRAMs are fed into a multiplier.

The multiplier output is stored back to the two DPRAMs. The MLFG generator produces a random number every clock cycle.

HASPRNG uses one of the PowerPC processors for initialization, seeding, and interface support to the host. The current implementation executes using Digilent XUP boards containing Xilinx XC2VP30 FPGAs. Work to port the HASPRNG library to the Cray XD1 platform is in progress. Support for other FPGA parts (e.g. Spartan 3, Virtex 4/5, and Altera) is also in progress.

3. HASPRNG Performance

HASPRNG generators provide random numbers faster than processors. Table 1 shows the performance summary for HASPRNG, where MRNS represents millions of random numbers generated per second. HASPRNG achieves 4-15 times speedup over 2.8 GHz Pentium 4 processors for each copy of the random number generators. Table 2 shows the hardware requirements and maximum clock frequency report on the XUP board. Obviously, additional streams can be easily added to the FPGAs providing additional speedup.

Table 1. HASPRNG Performance

	2.8GHz P4	FPGA	SpeedUp
ALFG	19.3 MRNS	100 MRNS	5.2X
LCG48	20.3 MRNS	100 MRNS	4.9X
LCG64	6.6 MRNS	100 MRNS	15.2X
CMRG	3.2 MRNS	50 MRNS	15.6X
MLFG	10.4 MRNS	66.7MRNS	6.4X
PMLCG	7.3 MRNS	50 MRNS	6.8X

Table2. HASPRNG Hardware Usage and Clock Frequency on XC2VP30

	Hardware Usage			Clock Rate (MHz)
	Slices (13696)	BRAM (136)	Multiplier (136)	
ALFG/E	1882(13%)	48(35%)	-	105
ALFG/O	1904(13%)	48(35%)	-	101
LCG48	3231(23%)	33(24%)	12 (8%)	103
LCG64	4072(29%)	9(6%)	20 (14%)	101
CMRG	5482(40%)	17(12%)	20 (14%)	101
MLFG	1710(12%)	41(30%)	10 (7%)	68
PMLCG	1984(14%)	17(12%)	16 (11%)	78

4. HASPRNG Verification

For verification of HASPRNG generators, each of them is compared to its SPRNG counterpart to ensure bit-equivalent behavior. Because we wish to test each configuration of generator and parameter set with several different initial seeds [1], the number of test cases becomes impractical for off-line verification. Hence, the SPRNG were ported to execute on one of the PowerPC processors. Figure 1 illustrates the HASPRNG verification architecture.

For the regression testing, a DPRAM is employed for 128KB data transfer. When data is full in the DPRAM, the local controller (Cont_1) directs the master controller (Cont_2) to pause HASPRNG operation. The PowerPC reads data from both SPRNG and HASPRNG, and verify the HASPRNG data by a bitwise XOR function. When the 32K random numbers are checked, the PowerPC allows the controller (Cont_2) to resume HASPRNG operations. Using this architecture, each HASPRNG generator has been verified with over 100 million random numbers.

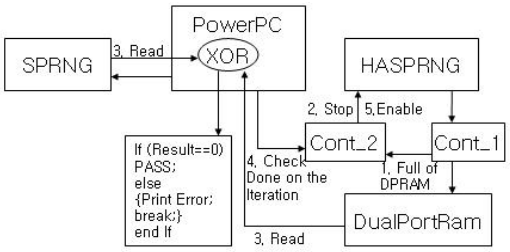

Figure 1. Verification of HASPRNG

5. Conclusions

HASPRNG was verified with a substantial set of dynamically generated random numbers. HASPRNG shows good speedup relative to SPRNG. Based on these reasons, HASPRNG should help contribute to computational science and the practical ability to exploit high performance reconfigurable computing resources. The HASPRNG library is open source.

6. Acknowledgements

This work was supported by the National Science Foundation grant, NSF CHE-0625598, and the authors gratefully acknowledge prior support for related work from the University of Tennessee Science Alliance.

7. References

[1] Scalable Parallel Pseudo Random Number Generators Library, http://sprng.fsu.edu/

[2] Y. Bi, G. D. Peterson, G. L. Warren, and R. J. Harrison, “Hardware acceleration of parallel lagged-Fibonacci pseudo random number generation,” *Proc of Intl Conf on Engineering of Reconfigurable Systems and Algorithms*. CSREA, 2006.