Resource-Efficient FPGA Pseudorandom Number Generation

Hüsrev Cılasun†, Ivy Peng*, Maya Gokhale*
*Lawrence Livermore National Laboratory, Email: {peng8,gokhale2}@llnl.gov
†University of Minnesota, Twin Cities, Email: cilas001@umn.edu

I. Introduction

Probability distributions play a critical role in diverse application domains. In simulations, probability distributions model phenomena such as physical properties of materials, of processes, or of behaviors. For instance, molecular dynamics codes often utilize the Maxwell-Boltzmann distribution for modeling temperature. Software-based random number generation usually employs rejection-based approaches. Thus, sample generation is not guaranteed at each iteration. Many applications can be parallelized in hardware for acceleration purposes, yet rejection-based software methods do not suit well into FPGA fabric due to the complexity of exception circuitry [1]. Random number generation in FPGAs, GPUs, and parallel processors is summarized in [2].

We introduce a resource-efficient hardware pseudo-random number generator (RNG) and two optimizations – *alias table partitioning* and *adaptive threshold resolution*. The first technique separates a target distribution into multiple sub-ranges and facilitates local optimizations in each sub-range to improve overall resource utilization. The second technique adjusts bitsize for representing threshold values to the precision of underlying partition. Our main contributions are as follows:

- Analytic study driven by dual considerations of improving accuracy and hardware mapping optimization
- Automated HDL generation of both simulation and synthesis scripts
- Diverse use cases: emulating Gaussian delay profile in FPGA-based memory system emulator; random number server for HPC applications

II. Methodology

Walker's Alias Method [3] is an efficient algorithm for FPGA hardware implementation. It generates arbitrary discrete distributions from uniformly generated random numbers. For a target distribution $E(\cdot)$, this method generates and uses a table of real threshold values $F(\cdot)$ and alternative index values $A(\cdot)$, where $F(\cdot)$, $A(\cdot)$, and $E(\cdot)$ are of the same length. Each output sample Y is generated as

$$Y = \begin{cases} X & U \leq F(X) \\ A(X) & U > F(X), \end{cases}$$

where U is a real uniform random number and X is a uniform random integer. The output quality is a function of the precision of U, i.e., increasing the bit size improves the quality. However, alias table methods are limited in scalability because for output of N bit size, 2^N entries are required in the alias table.

To address this limitation for the Gaussian case, our RNG leverages a resource-efficient approach called PwCLT [1] that exploits Central Limit Theorem (CLT). PwCLT combines alias table method with CLT components by numerically optimizing the alias table. PwCLT uses floating-point thresholds to represent small numbers accurately. This allows increased fixed-point range that results in high tail accuracy even in the range of 9.1σ – 13.2σ.

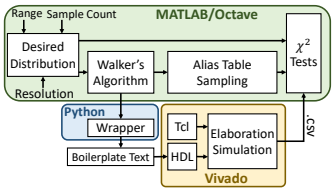
We leverage the generic property of the alias method to support arbitrary distributions in our RNG. For a target distribution, the RNG is customized by the parameters of the distribution to be implemented. For example, the Maxwell-Boltzmann distribution (Eq.1) has its PDF as a function of temperature T while the Planck distribution (Eq.2) is parameterized by the factor a. These distributions are constrained in boundaries and sampled in given resolution, and then the resulting probability distribution is provided as an input to Walker's alias table generation algorithm.

$$f(x) = \frac{2hc^2}{x^5} \exp - \frac{hc}{xkT} \tag{1}$$

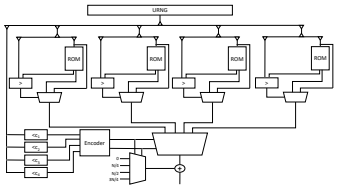
$$f(x) = \sqrt{\frac{2}{\pi}} x^2 \frac{\exp \frac{-x^2}{2a^2}}{a^3} \tag{2}$$

Alias Table Partitioning We improve the resource utilization for alias tables by separating the target distribution into multiple subranges (four subranges are exemplified in Fig. 1b). In each subrange, the standard alias table implementation is performed. This separation allows each table to be optimized locally, i.e., alias tables whose target distribution is smoother can be configured to have fewer threshold bits in $F(\cdot)$ table per entry. Consequently, the alias tables can be selected based on their relative probability range and lifted accordingly. This separation optimization requires one uniform random number r from the Uniform Random Number generator (URNG in Fig. 1b). r is compared against the cutoff values c_is, which are the CDF values in partition boundaries. Empirical χ^2 tests based on alias table sampling can be used to determine a sufficient amount of bits in each subrange for a desired quality metric.

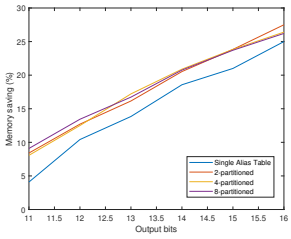
Adaptive Threshold Bitsize We propose adaptive threshold resolution to adjust the threshold bitsize while maintaining statistical accuracy. The quality of the generated samples is determined by the threshold resolution. Instead of using the same bitsize as the alternative indices, we use the χ^2 test to guide the selection of threshold bitsize. For a target distribution interval, the threshold bitsize is selected to be the minimum that passes χ^2 tests. When alias table partitioning is employed, partitions with higher variance yield larger bitsize while smaller bitsize is required for those partitions with lower variance.



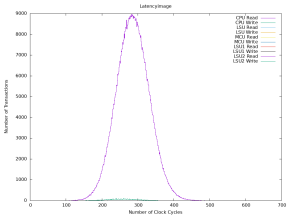
(a) An automated flow of customization and testing



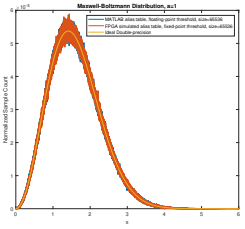
(b) Alias table partitioning with four partitions



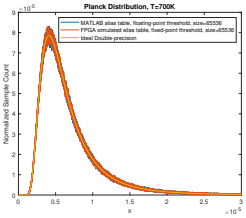
(c) Memory savings from three partitioning schemes.



(d) Gaussian Latency Histogram in LiME



(e) Maxwell-Boltzman distribution



(f) Planck distribution

Automated Customization and Testing We develop an automated flow of customization and testing for hardware pseudorandom number generation (Fig. 1a). The MATLAB code generates the desired probability distribution with given parameters. It uses Walker's alias table generation algorithm to compute the threshold values $F(\cdot)$ and alternate indices $A(\cdot)$. This table is used for both software sampling in MATLAB code as well as for hardware. A Python-based substitution script customizes the boilerplate text and embeds the MATLAB generated table into VHDL in the desired resolution. Using a Tcl script, generated HDL code is elaborated into a behavioral simulation snapshot, which is run to generate the desired amount of output samples, which are used for validation later.

III. Validation and Evaluation

We evaluate the hardware implementation on ZCU102 platform that features a Xilinx Zynq UltraScale+ MPSoC FPGA. We validate the accuracy of generated distributions using the standard χ^2 goodness-of-fit test that empirically measures whether a desired histogram matches a target distribution.

Memory Saving We evaluate the memory saving from the proposed optimizations in the Planck distribution. Fig. 1c reports the averaged results for 10 runs from three threshold bit optimizations, i.e., 2,4,8-partition, as well as the result from a single alias table. For each partition scheme, we increase the number of threshold bits until each subrange until both partitioned the combined distributions pass the χ^2-test. For the given range, the partitioned implementation achieves up to 5% memory saving compared to the single alias table implementation for 2^{24} samples.

A. Gaussian Memory Latency Distribution

We integrate our RNG into the variable delay unit in the LiME [4] memory emulator to emulate memory latency of Gaussian distributions. The implementation is based on PwCLT-8 Gaussian RNG, and allows the user to choose from 14 scaling options. The PwCLT-8 integrated LiME can emulate memory access latency ranging from 64 to 512K cycles. The configuration can be performed at runtime by a makefile parameter `PWCLT`. We test the integrated LiME using several LiME apps. Fig. 1d reports the memory latency histogram obtained using `make D=CLOCKS,STATS,TRACE=_TALL_,PWCLT=_DMAX512_ run` for the *image* benchmark.

B. Maxwell-Boltzmann and Planck Distributions

We implement two distributions common in scientific simulations – the Maxwell-Boltzmann and the Planck distribution. The implementation uses the standard single alias table approach in Sec. II. We target Maxwell-Boltzmann and Planck distributions with $a = 1$ and $K = 700K$, respectively. Fig. 1e and 1f report 2^{25} 16-bit samples generated from MATLAB and FPGA and the ideal reference in double precision. Both MATLAB and FPGA samples pass χ^2-tests for $\alpha = 0.05$.

IV. Conclusion

We introduced a resource-efficient hardware RNG whose accuracy is validated by χ^2 test. We proposed an alias table partitioning technique for optimizing resource utilization. Our RNG is evaluated in three use cases for memory emulations and scientific simulations.

Acknowledgment

This work was supported by LLNL LDRD 19-ERD-004. LLNL-ABS-813772.

References

[1] D. B. Thomas, "FPGA gaussian random number generators with guaranteed statistical accuracy," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 149–156.

[2] D. B. Thomas, L. Howes, and W. Luk, "A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2009, pp. 63–72.

[3] A. J. Walker, "An efficient method for generating discrete random variables with general distributions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 253–256, 1977.

[4] A. K. Jain, S. Lloyd, and M. Gokhale, "Microscope on memory: Mpsoc-enabled computer memory system assessments," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 173–180.