Using FPGA Resources for Direct Generation of Multivariate Gaussian Random Numbers

David B. Thomas and Wayne Luk

Department of Computing, Imperial College, London, England

{dt10,wl}@doc.ic.ac.uk

***Abstract*—The multivariate Gaussian distribution is used to model random processes with distinct pair-wise correlations, such as stock prices that tend to rise and fall together. Generation from a distribution with dimension n is usually achieved by starting with a vector of n independent Gaussian samples, then multiplying with a correlation inducing matrix, using $O(n^2)$ multiplications. This paper presents a method of generating vectors directly from the uniform distribution, removing the need for any multipliers or a scalar Gaussian generator. The method uses only small ROMs and adders, and so can be implemented using just basic FPGA resources (LUTs and FFs), saving DSP and block-RAM resources for the numerical simulation that the multivariate generator is driving. The method produces a new vector every cycle, unlike existing methods which produce vectors serially over n cycles, with only a modest increase in resource usage. This provides a ten times increase in performance over the fastest existing method, while also providing five times the performance per logic resource of the most efficient method.**

I. Introduction

The multivariate Gaussian distribution is used to capture simple correlations between related stochastic processes, such as the stock prices of companies in similar sectors, where the stock prices of all companies in the sector tend to rise and fall together. To simulate the behaviour of such processes, Multivariate Gaussian Random Number Generators (MVGRNGs) are used to generate random samples, which are then used to drive Monte-Carlo simulations. Such simulations often require a huge number of independent runs in order to provide an accurate answer, such as the Value-at-Risk calculations performed by financial institutions, which estimate how much money the institution might lose over the next day.

Such long-running simulations are an ideal target for FPGA acceleration, as they offer abundant parallelism, and are computationally bound; however they are reliant on a fast, resource-efficient, and accurate source of random samples from the multivariate Gaussian distribution. This paper improves on previous methods [1], [2] for FPGA-based MVGRNG, by developing a method that provides a large increase in performance, while limiting resource usage to standard logic elements (LUTs and FFs). Our key contributions are:

- An algorithm for generating samples from the multivariate Gaussian distribution using only uniform random bits, table-lookup, and addition.
- A hardware architecture for implementing an MVGRNG using only LUTs and FFs, which allows a regular densely-packed placement strategy, and provides operating speeds of 550MHz in Virtex-5 FPGAs.

II. Multivariate Gaussian Random Numbers

Generation of the uni-variate Gaussian distribution with a specific mean, μ, and variance, σ^2, can be achieved by first generating a standard Gaussian variate r with mean zero and variance one. The desired distribution is then generated using a linear transformation:

$$x = \sigma r + \mu, \quad \text{where} \quad r \sim N(0,1) \tag{1}$$

Note that the standard Gaussian variate is multiplied by the Standard Deviation (SD) σ, rather than the variance σ^2.

Generation of a multivariate Gaussian distribution is very similar, except now each output sample is a vector $\mathbf{x}$ of length n. The mean and variance increase in dimension as well, so the mean becomes a length n vector of $\mathbf{m}$, and the variance becomes an $n \times n$ covariance matrix $\mathbf{\Sigma}$. The covariance matrix is a symmetric matrix which captures the variance of each output component on the diagonal, and the correlations between each component on the off-diagonal elements.

Generation is very similar to the uni-variate linear transform, except the starting point is a vector $\mathbf{r}$ of n independent standard normal numbers:

$$\mathbf{x} = \mathbf{A}\mathbf{r} + \mathbf{m} \tag{2}$$

The matrix $\mathbf{A}$ is conceptually similar to the SD: just as the variance is the SD squared, so $\mathbf{A}\mathbf{A}^T = \mathbf{\Sigma}$. However, in the multivariate case there is considerable freedom in the selection of $\mathbf{A}$, as there are many possible choices.

One method is to perform Cholesky decomposition of the correlation matrix, producing a lower-triangular matrix. This choice has computational and storage advantages: only $n(n+1)/2$ of the elements are non-zero and must be stored, and only $n(n+1)/$ multiply accumulates are required to transform the vector $\mathbf{r}$ into $\mathbf{x}$.

An alternative method is to use the Singular Value Decomposition (SVD) algorithm. This decomposes the matrix into an orthonormal matrix $\mathbf{U}$ and a diagonal matrix $\mathbf{S}$, such that $\mathbf{\Sigma} = \mathbf{U}\mathbf{S}\mathbf{U}^T$. This decomposition allows the construction of the solution $\mathbf{A} = \mathbf{U}\sqrt{\mathbf{S}}$. The disadvantage of the SVD-based construction is that in general all the elements of the matrix are non-zero, resulting in an n^2 cost in both the number of stored elements, and in the number of multiply-accumulates

per transformed vector. However, the SVD algorithm is able to handle a wider range of covariance matrices, such as ill-conditioned matrices that are very close to singular, and reduced rank matrices, where the output vector depends on fewer than n random factors.

III. Generation using LUTs and Adders

The standard generation method uses direct matrix multiplication, forming each output element from a linear combination of $\mathbf{r}$, a vector of n IID (Independent Identically Distributed) Gaussian samples:

$$x_i = m_i + \sum_{j=1}^{n} a_{i,j} r_j, \quad i \in 1..n \tag{3}$$

For an arbitrary covariance matrix this requires at most n^2 multiply-accumulations, or least $n(n+1)/2$, depending on the number of non-zero elements of $\mathbf{A}$. However, this method also requires the generation of the n elements of $\mathbf{r}$, which are IID standard Gaussian samples. Both the generation of $\mathbf{r}$ and the multiplication with $\mathbf{A}$ require significant resources (i.e. DSPs and RAMs), so we would like to minimise them if possible.

The method we propose in this paper is that, instead of generating independent Gaussian samples then scaling them to the desired SD with $O(n^2)$ multiplications, we will generate uniform samples and convert them directly to Gaussian samples with the appropriate SD via n^2 tables. Each table contains a pre-calculated discretised Gaussian distribution with the correct SD, so the only operations required are table-lookups and additions.

Each table contains k elements, and is indexed using the syntax $L[i]$ to access table elements $L[1]..L[k]$. We use arrays of tables, with sub-scripts to identify a table within the array, which can then be indexed as for a standalone table, e.g. $L_{2,3}[4]$. We will also treat tables interchangeably as discrete random number generators, where the discrete PDF of each table is given by assigning each element of the table an equal probability of $1/k$. For example, if u is an IID uniform sample between 0 and 1, a random sample x from table L is generated as:

$$x = L\left[\ \lceil uk \rceil\ \right], \quad \text{where} \quad u \sim U(0,1) \tag{4}$$

The central idea of this method is to construct an $n \times n$ array of tables $\mathbf{G}$, such that the discrete distribution of each table $\mathbf{G}_{i,j}$ approximates a Gaussian distribution with SD $\mathbf{A}_{i,j}$:

$$\mathbf{G}_{i,j} \sim N(0, \mathbf{A}_{i,j}) \tag{5}$$

Now instead of starting from a Gaussian vector $\mathbf{r}$, we can use an IID uniform vector $\mathbf{u}$. Generation of each output element uses each element of $\mathbf{u}$ as a random index into a table, then sums the elements selected from each table:

$$x_i = m_i \sum_{j=1}^{n} L_{i,j}\left[\ \lceil u_j k \rceil\ \right], \quad i \in 1..n \tag{6}$$

In practice k should be selected to be a power of 2, so each element of $\mathbf{u}$ is actually a uniform integer constructed from the concatenation of $\log_2(k)$ random bits.

The simplest method of generating a table-based approximation to the Gaussian distribution is direct CDF inversion. To generate a table L with SD σ, we choose table elements:

$$L[i] = \sigma \Phi^{-1}(i/(k+1)), \quad i \in 1..n \tag{7}$$

where $\Phi^{-1}(.)$ is the Gaussian inverse CDF. For small tables the moments of these tables are incorrect - in particular the standard-deviation and kurtosis will be too small. However, a polynomial stretch can be used to make the first four central moments exact. [1]

Given a target matrix $\mathbf{A}$ we can now fully specify $\mathbf{G}$:

$$\mathbf{G}_{i,j}[z] = \mathbf{A}_{i,j} \Phi^{-1}(z/(k+1)), \quad i,j \in 1..n, \quad z \in 1..k \tag{8}$$

Construction of $\mathbf{G}$ allows the direct transformation of uniform samples into multivariate Gaussian samples using Equation 6, requiring only table-lookups and addition.

IV. Hardware Implementation

The central idea of this paper, of replacing Gaussian samples and multipliers with uniform samples and tables, allows for many types of possible implementation. For example, the tables can be implemented using LUTs or block-RAMs, and the generator can vary in throughput from 1 to n cycles per generated vector. In this paper we try to focus on the most interesting mode of the generator, to provide the maximum contrast with previous implementations, while still providing high performance and quality. The specific choices we make are:

- **Logic resources only**: tables are implemented using LUTs, so the only resources used are LUTs and FFs.
- **Parallel generation**: the generator operates in fully parallel mode, providing one new n-element vector per cycle.
- **Maximum clock rate**: the generator operates at the maximum realistic clock-rate for the target FPGA. For the Virtex-5 we use 550MHz, as this is the maximum clock rate of the DSP and RAM blocks that will be used in the simulation that the generator is driving. [2]
- **Regular architecture**: simulations typically consume almost all resources in the FPGA (due to replication of simulation cores), and a regular, explicitly placed, highly routable, generator is useful for reducing stress on the place-and-route tools and maximising overall clock-rate.
- **No matrix specialisation**: the generator is not optimised for a specific correlation matrix, and supports any correlation structure without structural modification or recompilation.

The table-based generator maps naturally into a regular pipelined structure, shown in Figure 1. On the left hand side is a random bit source, which generates a new vector $\mathbf{u}$ every cycle. The elements of vector $\mathbf{u}$ are broadcast horizontally through the cells, and used to select one element from each

[1] The details of the stretch were removed for space reasons, but will be detailed in a later paper.

[2] Note that the MVGRNG can actually be clocked faster than 550MHz according to post place-and-route timing analysis - there is just no practical reason to do so.

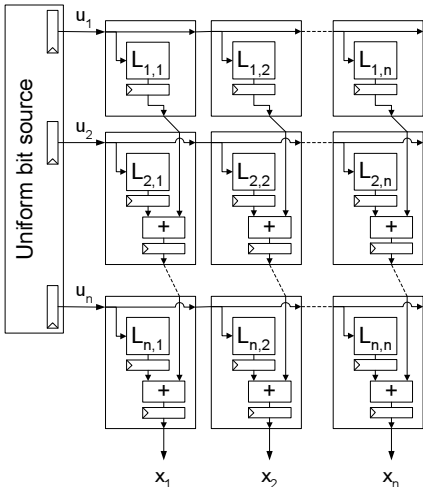

Fig. 1. Regular hardware architecture for table based generation.

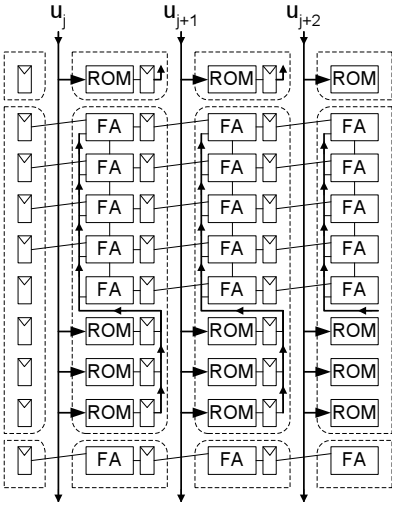

Fig. 2. Practical dense placement for hardware realisation.

table. The selected elements are then accumulated vertically from top to bottom, with one new vector output per cycle.

A useful optimisation for increasing the effective number of elements per table is to take advantage of the symmetry of the tables. Because the tables have a mean of zero, we have the property that $L[i] = -L[k-i+1]$, so it is only necessary to store the elements $L[1]..L[k/2]$. The half-table is now indexed by all but the most-significant bit of the uniform index, while the most significant uniform bit is used to select whether the table value is added or subtracted from the accumulator. Note that the values stored in the table *must* be signed, so that it is possible to encode both positive and negative values from $\mathbf{A}$. This optimisation doubles the effective table size of each LUT; for example, a Virtex-5 6-LUT can support a 128 element table, rather than just 64.

We assume that: table elements have width w_t; accumulators have a width $w_a = w_t + \lceil \log_2 n \rceil$; each LUT can implement a $1 \times k$ bit LUT (using the previous optimisation); and the uniform generator is implemented using a LUT-Optimised RNG [3]. The resource usage of the generator can then be broken down as: Uniform RNG, $n \log_2 k$; tables, $n^2 w_t$; and accumulators, $n^2 w_a = n^2(w_t + \lceil \log_2 n \rceil)$. This results in a total resource utilisation of:

$$n(\log_2 k + n(2w_t + \lceil \log_2 n \rceil)) \tag{9}$$

This describes the number of LUTs, the number of FFs, and also the number of fully-occupied LUT-FF pairs, as all elements use a LUT connected to a FF.

The regularity of the architecture makes it trivial to explicitly place all components in the generator, reducing the load on the place-and-route tools, and making it much easier to achieve high clock-rates for the overall design. In this paper we adopt the simple placement strategy shown in Figure 2, where the accumulator is simply stacked on top of the table. There is a limit to this specific placement strategy, as eventually the generator becomes too tall to fit in a given device, but other wider and shorter arrangements can also be easily described.

This architecture has been described in VHDL, using Virtex-5 primitives, and RPM (Relationally Placed Macro) constraints. When mapped into hardware, the resource utilisation exactly matches the predictions of Equation 9. For $n \leq 16$ and $w_t \leq 16$ this strategy is also able to provide 550MHz operation (this is the post place-and-route timing analysis, and has also been verified in hardware). For larger n it becomes necessary to insert buffering registers on the wires distributing the uniform random bits, as both their length and load scales linearly with n.

V. Evaluation

To test whether a multivariate generator is producing the correct correlation matrix, we use two statistical tests previously used to evaluate hardware MVGRNGs. In [1] Fisher's Z transform is used to examine the hypothesis that each element of the empirical correlation matrix has the same correlation as the target matrix. This results in an $n \times n$ matrix of p-values - p-values should appear as uniformly distributed values between 0 and 1, with values very close to 0 or 1 suggesting failure. The matrix of p-values are then reduced down to a single p-value using the Anderson-Darling method.

Another measure of the statistical quality of a multivariate generator is suggested in [2], where the correlation matrix quality is measured using the Mean Squared Error (MSE) between a correlation matrix empirically recovered from the output stream, and the target correlation matrix.

Figure 3 shows the results of both approaches, using $n = 10$ and a randomly generated matrix (for easier comparison with the results in [2]), with sample sizes from 2^{10} vectors up to 2^{31}. The left axis shows the empirical MSE, with a steady decrease in MSE as the sample size is increased.

The points on vertical stalks show the p-values associated with the sample size at which they were calculated, so for easier interpretation the connected line shows the sorted p-values. If the p-values are truly uniformly distributed they should form a roughly straight line when sorted. The results

TABLE I
CHARACTERISTICS OF THREE ALTERNATIVE METHODS FOR GENERATING MULTIVARIATE GAUSSIAN SAMPLES IN FPGAS.

	Part	Clock rate	Cycles/ Vector	Config. Method	External RNG	LUTs	DSPs	MVec/ sec	KVec/ LUT/sec	MSE
DSP [1]	Virtex 5	550MHz	n	Dedicated logic (1-10 ms)	Gaussian	717	10	55	76.7	1.0×10^{-4}
Custom [2]	Stratix 3	411MHz	n	Place-and-route (10 mins+)	Gaussian	1250	0	41	32.9	2.5×10^{-4}
Table	Virtex 5	550MHz	1	Bitstream manip (1-5 s)	none	3270	0	550	168.2	1.4×10^{-5}

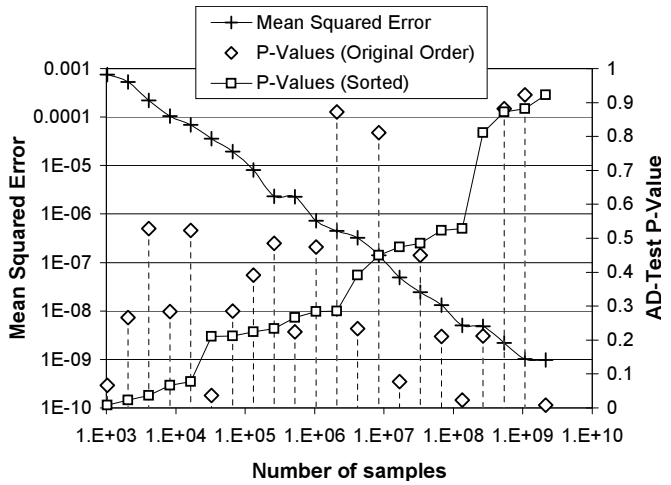

Fig. 3. Empirical results for 10-element matrix as sample size is increased, showing mean square error of the empirical correlation matrix, and p-values for Anderson-Darling test of empirical covariance matrix.

of this test do not show any clear evidence for the hypothesis that the empirical and target correlation matrices are different, so we can conclude that the generator is producing the correct correlation matrix.

VI. RELATED WORK

There are two main previous approaches to the problem of sampling from the Multi-Variate Gaussian distribution. The earliest and simplest approach is the multiplier based approach used in [1]. This uses a set of n DSP blocks (multiply-accumulate units), and splits the generation into two stages. In the first stage, a scalar Gaussian RNG generates the vector of independent variates $\mathbf{r}$ over n cycles, shifting each generated sample down a shift register. In the second phase the vector $\mathbf{r}$ is retrieved in parallel from the shift register, then over the next n cycles each element x_i of the output is calculated using Equation 3. For this paper a new implementation has been developed using optimisations for smaller vectors, and for Virtex-5 rather than the original Virtex-4 primitives. The design is only limited by the DSP components, resulting in a post place-and-route clock rate of 550MHz.

The approach taken in [2], is to build a custom circuit for each covariance matrix $\mathbf{A}$. This allows the circuit to be heavily optimised, taking advantage of word-length optimisation to reduce resource usage, while maintaining the accuracy of the generator's correlation matrix. Multiple possible implementations are also produced, by using a library of building blocks to create multiple candidate generators, then constructing a Pareto frontier of correlation accuracy versus resource usage. Generating a custom circuit for each covariance matrix provides many optimisation possibilities, but also has the disadvantage that each new matrix incurs a full synthesis and place-and-route cycle.

The two existing methods are compared with the table-based method in Table I, comparing both the performance and resource utilisation for a generator of 10-element vectors. In terms of raw clock-rate, the table-based method operates at a similar level to both existing methods, but because it generates a new vector every cycle it offers ten times the performance. When resources are considered, the table-based method requires 2.6 times the resources of the customised generator, but due to the increased performance still offers 5 times the performance per resource.

The table-based method is also the only stand-alone method, as the uniform random number is included in the resource cost. The other methods require an external scalar Gaussian RNG, which increases the practical resource cost, and will require either more DSPs, block-RAMs, or both.

VII. CONCLUSION

This paper has presented a new method for multivariate Gaussian random number generation in FPGAs, which decomposes the generation into a series of table-lookups and additions. This method is ideal for use in numerical Monte-Carlo simulations, as it only uses LUTs and FFs, and so all DSP and block-RAM resources can be used in the simulation core that the generator is driving.

When compared to previous methods for MVGRNG, the table-based method offers greatly improved performance, generating a new random vector every cycle, rather than generating vectors serially over multiple cycles. When generating length ten vectors, the table-based method provides ten times the performance of the fastest DSP-based method. The performance per resource is also five times that of a generator constructed and compiled for a specific correlation matrix, while still supporting loading of arbitrary correlation matrices.

REFERENCES

[1] D. B. Thomas and W. Luk, "Multivariate Gaussian random number generation targeting reconfigurable hardware," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, no. 12, 2008.

[2] C. Saiprasert, C.-S. Bouganis, and G. A. Constantinides, "Word-length optimization and error analysis of a multivariate gaussian random number generator," in *Proc. Int. Conf. Applied Reconfigurable Computing*, 2009, pp. 231–242.

[3] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimised state-transition matrices," *Journal of VLSI Signal Processing*, vol. 47, no. 1, 2007.