Multivariate Gaussian Random Number Generator Targeting Specific Resource Utilization in an FPGA

Chalermpol Saiprasert, Christos-Savvas Bouganis and
George A. Constantinides

Department of Electrical & Electronic Engineering,
Imperial College London, Exhibition Road,
London SW7 2BT, United Kingdom.
{cs405,christos-savvas.bouganis,g.constantinides}@imperial.ac.uk

Abstract. *Financial applications are one of many fields where a multivariate Gaussian random number generator plays a key role in performing computationally extensive simulations. Recent technological advances and today's requirements have led to the migration of the traditional software based multivariate Gaussian random number generator to a hardware based model. Field Programmable Gate Arrays (FPGA) are normally used as a target device due to their fine grain parallelism and reconfigurability. As well as the ability to achieve designs with high throughput it is also desirable to produce designs with the flexibility to control the resource usage in order to meet given resource constraints. This paper proposes an algorithm for a multivariate Gaussian random number generator implementation in an FPGA given a set of resources to be utilized. Experiments demonstrate the proposed algorithm's capability of producing a design that meets any given resource constraints.*

Key words: Multivariate Gaussian Distribution; Random Numbers; FPGA; Resource Constraint

1 Introduction

Financial applications are one of the fields that require computationally extensive simulations. Examples of these applications include equity returns modeling and portfolio optimization [1]. Many of these applications involve simulations for predicting the behaviour of stock prices in the stock market. Monte Carlo simulation is a well known technique that is widely used in order to realize these simulations [2]. A key component in a Monte Carlo simulation is a random number generator which generates random samples from a variety of distributions that model certain aspects in equity pricing. One of the most widely used distributions is the multivariate Gaussian distribution which is defined by its mean and covariance matrix. The function of the covariance matrix is to encapsulate correlation information of the random variables.

In certain applications, such as calculating the value-at-risk, it can take up to a day to complete the simulation using a cluster of PCs due to its computationally demanding nature [3]. In order to accelerate this task, one possibility is to improve the speed of the random number generator. One way to achieve that is by dedicating a hardware device which implements the multivariate Gaussian random number generator. Often, this target device is a Field Programmable Gate Array (FPGA) due to its fine grain parallelism and reconfigurability properties.

In most applications, a random number generator module constitutes part of a larger application such as modeling equity returns and portfolio optimization. As the number of resources on a single FPGA is limited, it is essential to keep the number of resources dedicated to such a module as low as possible. Recently, Thomas and Luk [4] have proposed an architecture for generating multivariate samples from a Gaussian distribution. Their algorithm, however, does not have the flexibility to tune the number of resources required to implement the proposed architecture in hardware to a specific resource requirement. Moreover, since the covariance matrix is constructed empirically from historical data it is expected that its elements deviate by certain amount from the underlying values. Therefore, a certain degree of freedom is allowed in the search for a hardware design by allowing some deviation in the approximation of the original covariance matrix. In order to exploit this idea we require a methodology which produces a design that generates multivariate Gaussian random samples by permitting an approximation to the original covariance matrix. The quality of the approximation depends on the number of utilized resources when the design is mapped into hardware. This paper presents a novel architecture for the implementation of a multivariate Gaussian random number generator in hardware, with the ability to tune the resource usage to user's requirement.

The organization of this paper is as follows. The background theory involved in this paper is explained in Section 2, while a description of current related work concerning Gaussian random number generators is given in Section 3. Section 4 focuses on the detailed description of the proposed algorithm and on the hardware implementation. Results regarding the performance evaluation of the proposed architecture are presented in Section 5. Finally, Section 6 concludes the paper.

2 Background Theory

Many existing techniques are available in the literature for the generation of multivariate Gaussian random samples such as the Rotation method [5], the Conditional method [6] and the Triangular Factorization method [7]. One of the current techniques that researchers have heavily focused on is the Triangular Factorization method where multivariate Gaussian samples are generated based on univariate Gaussian random samples. This approach factorizes the covariance matrix $\mathbf{C}$ into a product of a lower triangular matrix $\mathbf{A}$ and its transpose, $\mathbf{C} = \mathbf{A}\mathbf{A}^T$. The required multivariate Gaussian random samples $\mathbf{x}$ are gener-

ated by multiplying a vector containing univariate Gaussian random numbers, $\mathbf{r} \sim N(O, \mathbf{I})$ with the lower triangular matrix $\mathbf{A}$ to achieve the desired correlation structure while a vector $\mathbf{m}$ is added to adjust the mean values as shown in (1).

$$\mathbf{x} = \mathbf{A}\mathbf{r} + \mathbf{m}. \tag{1}$$

3 Related Work

In the literature researchers have focused on various techniques to generate multivariate Gaussian random samples. In [8], three methods for generating multivariate Gaussian random vectors are reviewed. These are the Rotation method, the Conditional method and the Triangular Factorization method which are all based on univariate Gaussian random samples. The authors in [8] implemented these approaches in software and evaluation of results have shown that the Triangular Factorization method is the most preferable method out of the three approaches as it requires less processing time and memory.

Recent technological advances coupled with today's requirements have driven designers towards hardware implementation of these generators on digital circuits. Many architectures exist for the generation of univariate Gaussian random numbers on an FPGA platform. This includes the Ziggurat method [9], the Wallace method [10] and the Box-Muller method [11]. An extensive review of these techniques has been performed in [12] where it has been concluded that the Wallace method has the highest throughput while the Ziggurat method comes second. However, the Wallace method is susceptible to correlation problems. Although these three methods are capable of producing Gaussian random samples using an FPGA platform, their hardware architectures involve the use of embedded multipliers. An alternative approach is established by Thomas and Luk [13] targeting an architecture without multipliers. The piecewise-linear approximation technique enables the design to be pipelined easily in order to perform high speed operations. This is because the design does not require any multipliers but only a lookup table, a subtractor and a comparator are utilized instead.

In the case of a multivariate random number generator in hardware, designs can be classified into two major categories, serial and parallel. If a vector of N samples is to be generated then a serial generator outputs one element of the vector in every clock cycle requiring N clock cycles for creating a complete vector. On the other hand, a parallel generator produces a complete vector of size N every clock cycle. The parallel approach has a larger throughput than the serial approach requiring however many more resources than the serial design. Thus, a trade off between throughput and resource usage exists between the two approaches. The rest of the paper focuses on the class of serial designs.

Thomas and Luk [4] developed a hardware model to generate multivariate Gaussian random numbers based on the Triangular Factorization method. To date, this is the only hardware based approach to generate such samples where the design is mapped onto an FPGA. Their design is capable of producing a

vector of length N which contains multivariate random samples for every N clock cycles. In terms of resource usage their approach requires N Multiply Accumulate (MACC) Units which are directly mapped into DSP blocks on an FPGA.

If we consider a scenario where the number of available DSP blocks on an FPGA is fewer than the size of the output vector N then the number of available MACC units will not be adequate to implement Thomas and Luk approach [4] on to a single FPGA, maintaining the same throughput. In order to further illustrate this point, if a vector of 1000 multivariate samples is to be generated then a 1000x1000 covariance matrix is required resulting in 1000 DSP blocks to be implemented on an FPGA using Thomas and Luk architecture [4]. This amount exceeds the number of available DSP blocks available on a modern high-end FPGA. For example, a high-end FPGA such as a Xilinx Virtex-4 offers up to 512 available DSP blocks [14]. As a consequence, multiple FPGAs would be required to map this architecture. Thus, a drawback of this approach is the lack of flexibility to accommodate designs where the size of the output vector is larger than the available DSP blocks on a single FPGA or the case where the system designer does not want to allocate the required resources to this module. In order to address this problem, this paper proposes an approach that produces multivariate Gaussian random samples by utilizing only a certain number of available DSP blocks specified by the system designer by allowing some error in the approximation of the covariance matrix while maintaining the same throughput as Thomas and Luk approach [4].

4 Proposed Algorithm

The proposed algorithm is based on the Triangular Factorization method. According to this method, the covariance matrix $\mathbf{C}$ is decomposed into a product of a lower triangular matrix $\mathbf{A}$ and its transpose using Cholesky Decomposition [15]. This lower triangular matrix $\mathbf{A}$ is multiplied with a vector $\mathbf{r} \sim N(O, \mathbf{I})$ which contains univariate Gaussian samples in order to produce multivariate samples (2) which have zero mean and covariance matrix $\mathbf{C}$.

$$\mathbf{x} = \mathbf{A}\mathbf{r}. \tag{2}$$

For an $N \times N$ lower triangular matrix $\mathbf{A}$ the number of multiplications required to perform the computation in (2) is $N(N+1)/2$ which corresponds to the number of non-zero elements in matrix $\mathbf{A}$. In the proposed algorithm, the lower triangular matrix $\mathbf{A}$ is approximated by applying the Singular Value Decomposition algorithm. The Singular Value Decomposition algorithm [16] or SVD is a technique where a matrix is decomposed into a product of an orthogonal matrix $\mathbf{U}$, a diagonal matrix $\mathbf{S}$ and the transpose of another orthogonal matrix $\mathbf{V}$. The diagonal matrix $\mathbf{S}$ contains only positive or zero elements in its main diagonal which are sorted in descending order, hence $s_{1,1} \geq s_{2,2} \geq s_{3,3}$ and so on. Essentially, the SVD algorithm expresses an initial matrix $\mathbf{A}$ as a linear com-

bination of separable matrices using the least number of decomposition levels, K, as possible.

The result of applying the SVD algorithm to the lower triangular matrix $\mathbf{A}$ is shown in (3) where the original matrix multiplication can be expressed as vector multiplication. $\mathbf{u}_i$ denotes the $i^{(th)}$ column of matrix $\mathbf{U}$ while $\mathbf{v}_i$ denotes the $i^{(th)}$ column of matrix $\mathbf{V}$. K is the number of decomposition levels used by the algorithm. Using the SVD algorithm the number of general multiplications to achieve the same output is $2KN$ where K is the number of decomposition levels. Therefore, the number of required multiplications can be reduced if K is less than $N/2$ in comparison with Thomas and Luk approach [4].

$$\mathbf{x} = \mathbf{Ar} = \mathbf{USV}^T\mathbf{r} = \left(\sum_{i=1}^{K} \mathbf{u}_i s_i \mathbf{v}_i^T \mathbf{r}\right) = \sum_{i=1}^{K} \mathbf{u}_i s_i \left(\mathbf{v}_i^T \mathbf{r}\right). \tag{3}$$

Due to the fact that most of the large covariance matrices are constructed empirically, it is expected that they deviate from the true underlying matrix. The proposed methodology exploits this fact by approximating the covariance matrix up to the appropriate precision level defined by the user.

The metric which is used to assess the quality of the approximation is the mean square error (MSE) between the original covariance matrix and its approximation using (4).

$$Err_K = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\mathbf{C}_{i,j}^K - \widehat{\mathbf{C}}_{i,j}^K\right)^2. \tag{4}$$

$\mathbf{C}$ represents the original covariance matrix, $\widehat{\mathbf{C}}$ corresponds to the approximated covariance matrix after approximating matrix $\mathbf{A}$ through the SVD algorithm, N is the size of the output vector and K is the decomposition level. i and j are the row and column indices of the matrix respectively.

4.1 Generation of U and V using the SVD algorithm

The proposed algorithm takes a covariance matrix $\mathbf{C}$ as an input and produces matrices $\mathbf{U}$ and $\mathbf{V}$ which contain the decomposition vectors $\mathbf{u}_q^i$ and $\mathbf{v}_q^i$ for each decomposition level i by applying the SVD algorithm. The subscripts q denotes the quantized version of the vectors. The decomposition can be driven either by the required MSE in the covariance matrix approximation or by the number of available resources.

In the proposed algorithm, fixed point number representation is used to store the matrix coefficients since fixed point arithmetic leads to designs which are smaller and have higher operating frequency in an FPGA compared to floating point arithmetic based designs. Thus, quantization error is introduced when the vectors are mapped into hardware. The proposed algorithm minimizes the inserted error in the system due to quantization effect by propagating it to the next level of decomposition [17]. Hence, the added error is taken into account in the remaining decomposition stages.

The pseudo-code that illustrates the outline of this operation is shown in Fig. 1. The first step of the proposed algorithm is to obtain the first level of the decomposition of the lower triangular matrix $\mathbf{A}$ resulting in the generation of vectors $\mathbf{u}$ and $\mathbf{v}$ and a scalar s. As the order in which the scalar s is multiplied to vector $\mathbf{u}$ and $\mathbf{v}$ has no effect on the resulting product, the coefficient $\sqrt{s}$ is multiplied to both $\mathbf{u}$ and $\mathbf{v}$. The two vectors are then transformed so that their coefficients lie in the range [-1,1) using a power of 2 scaling. This is to ensure that the range of numbers representable in hardware implementation is maximized. The vectors $\sqrt{s}\mathbf{u}$ and $\sqrt{s}\mathbf{v}$ are quantized to a user specified number of bits and an initial approximation of the $\mathbf{A}$ matrix is obtained. The two quantized vectors are now represented by $\mathbf{u}_q$ and $\mathbf{v}_q$ respectively. The entire process is repeated, having the remaining matrix $\mathbf{A}_r$ as a starting point, until the termination condition is met.

Algorithm: Approximate a covariance matrix $\mathbf{C}$ given K DSP blocks and p bits precision
Calculate lower triangular matrix $\mathbf{A}$ using Cholesky Decomposition $\mathbf{A} = Chol(\mathbf{C})$
$\mathbf{A}_r = \mathbf{A}$
$N = \lceil \frac{K}{2} \rceil$
FOR $i = 1 : N$
  Calculate the first decomposition level of $\mathbf{A}$ using $[\mathbf{u}_i, s_i, \mathbf{v}_i^T] = \text{SVD}(\mathbf{A}_r, 1)$
  Transform $\sqrt{s}\mathbf{u}_i$ and $\sqrt{s}\mathbf{v}_i^T$ to be in the range [-1,1) using a power of 2 scaling
  Quantize $\sqrt{s}\mathbf{u}_i$: $\mathbf{u}_q^i \leftarrow \sqrt{s}\mathbf{u}_i$ with p bits precision
  Quantize $\sqrt{s}\mathbf{v}_i$: $\mathbf{v}_q^i \leftarrow \sqrt{s}\mathbf{v}_i$ with p bits precision
  $\widehat{\mathbf{A}} = \sum_{K=1}^{i} \mathbf{u}_q^K (\mathbf{v}_q^K)^T$
  $\mathbf{A}_r = \mathbf{A} - \widehat{\mathbf{A}}$
  Store $\mathbf{u}_q^i$ in a 2-dimensional array $\mathbf{U}(:,i)$
  Store $\mathbf{v}_q^i$ in a 2-dimensional array $\mathbf{V}(:,i)$
END LOOP
RETURN $\mathbf{U}$ and $\mathbf{V}$

Fig. 1. Outline of the algorithm.

4.2 Hardware Implementation

Fig. 2 illustrates a high level hardware implementation of the multivariate Gaussian random number generator. In the figure, the circuit comprises of two computational blocks representing the two decomposition levels. In terms of the architecture the vectors $\mathbf{u}_q^i$ and $\mathbf{v}_q^i$ generated from the proposed algorithm are stored in the embedded block RAMs on the FPGA permitting parallel access to the data.

The inner structure of each computational block is illustrated in Fig.3. Both the MACC unit and the multiplier are mapped onto one of the available DSP

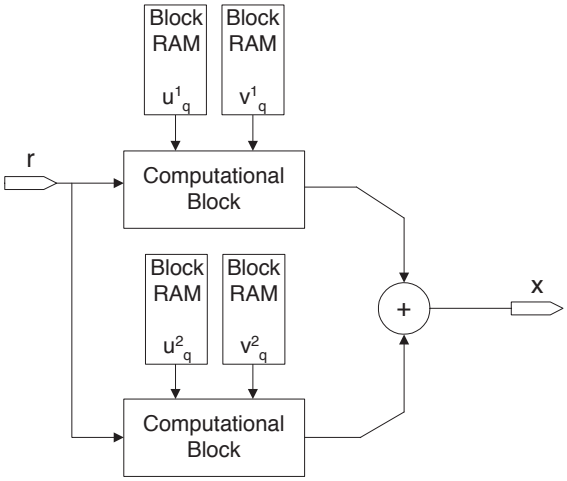


Fig. 2. High level hardware implementation.

blocks on the FPGA. The overall operation can be pipelined into two sections to achieve improved throughput. The first part performs the multiply-accumulate of the vector with univariate samples $\mathbf{r}$ and a vector $\mathbf{v}_q$ to produce a scalar quantity rv which is stored in a register. Note that the whole process of multiply-add requires one clock cycle as this function is realized using the MACC unit. For a vector of size N, it takes N cycles for the output in the register to be valid. The second part of the circuit multiplies rv with the vector $\mathbf{u}_q$ and N clock cycles are required in order to generate a complete output vector of size N. As both stages take N cycles to produce a valid output, the design is optimized by running the two modules in parallel. P denotes the precision of the system which is specified by the user, while the block denoted by T is a truncation block where the data is truncated to a specified precision. The final stage of the implementation is the summing stage where an adder tree structure is utilized for adding the outputs of K computational blocks to obtain an element in the output vector.

5 Performance Evaluation

In the hardware implementation, a Virtex-4 XC4VSX55 FPGA from Xilinx is utilized to map the design. A benefit for selecting the Virtex-4 model is that it provides DSP blocks which have the capability to perform a multiply-accumulate operation (MACC) in one clock cycle [14]. This is desirable since we would like to produce a sample per clock cycle and if the multiply-accumulate was not performed every clock cycle then two clock cycles would be required due to the feedback. In this work we adopt the method in [13] for the implementation of the univariate random number generator as it is the most efficient in terms of resources requirement in comparison to other techniques in the literature.

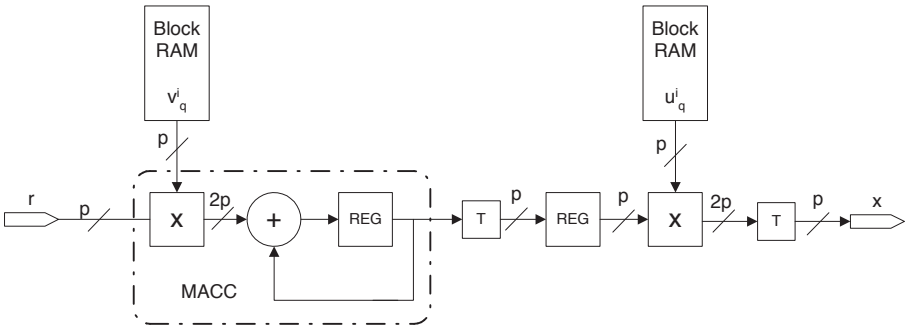


Fig. 3. Inner structure of the computational block.

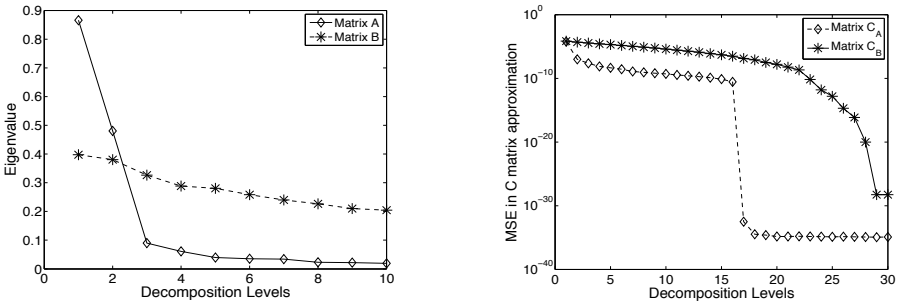
5.1 Impact of the Matrix Structure on the Proposed Algorithm

The eigenvalues of a matrix can provide a lower bound on the error of the approximation for a given number of decomposition levels. In order to demonstrate that we have selected, without the loss of generality, two lower triangular 30×30 square matrices $\mathbf{A}$ and $\mathbf{B}$ with different eigenvalue profiles to be applied with the proposed algorithm. The first ten eigenvalues of both matrices are plotted versus the level of decomposition in Fig. 4(a). From the graph, it can be seen that matrix $\mathbf{A}$ is more separable than matrix $\mathbf{B}$ since fewer decomposition levels are required before the eigenvalue drops to small values. Fig.4(b) illustrates the mean square error (MSE) of covariance matrices $\mathbf{C}_A = \mathbf{A}\mathbf{A}^T$ and $\mathbf{C}_B = \mathbf{B}\mathbf{B}^T$ for different decomposition levels. From the figure it is apparent that matrix $\mathbf{C}_A$ requires approximately 17 levels of decomposition to achieve an MSE of around 10^{-30} while 30 levels of decomposition are required for matrix $\mathbf{C}_B$ to approximately obtain the same MSE. Thus, it can be concluded that matrices with different separability properties require considerably different decomposition levels to achieve a certain error in the approximation. This provides a lower bound for the achieved error for a given decomposition level in the proposed algorithm since the quantization effect and truncation effect in the data path have not been considered. In summary, the above demonstrates that the performance of the proposed algorithm depends on the separability property of the matrix of interest.

5.2 Evaluation of the Proposed Algorithm

The main idea behind this work is that by permitting a certain error in the approximation of the covariance matrix a reduction of the number of required DSP blocks on the FPGA can be achieved.

As illustrated in Section 4.2, the proposed algorithm employs two DSP blocks for every level of decomposition, one for the MACC unit and one for the multiplier. An experiment is performed where matrices $\mathbf{A}$ and $\mathbf{B}$ are used from the previous section to determine the achievable MSE given a range of resources.



(a) Eigenvalue profiles of matrices **A** and **B**.

(b) MSE Comparison of matrices with different eigenvalue profile.

Fig. 4. Impact of different eigenvalue profiles to the approximation of a matrix.

The calculated MSE is the error between the original covariance matrix $\mathbf{C_A}$ and the approximated covariance matrix $\widehat{\mathbf{C_A}}$. The same principle applies for matrix **B**.

The proposed algorithm is applied to matrices $\mathbf{C}_A$ and $\mathbf{C}_B$ with no quantization, 4 bits quantization and 18 bits quantization. In order to compare the performance of the proposed algorithm with the technique proposed by Thomas and Luk [4], the MSE is also obtained from Thomas and Luk algorithm [4] where the algorithm is applied to the same matrices.

Fig. 5 illustrates the obtained results. The results show that regarding matrix **B** there is only a considerable difference in the obtained MSE between 4 bits quantization and 18 bits quantization when 10 or more DSP blocks are utilized using the proposed algorithm. However, the difference in MSE between 4bit and 18 bit precisions can clearly be seen for matrix **A** as shown in the graph. The difference in the patterns of the MSE plots between the two matrices is due to the different eigenvalue profiles. Moreover, for both matrices **A** and **B**, the MSE of the covariance matrix obtained using the proposed algorithm with 18 bits word-length is slightly higher than the optimum MSE obtained using the proposed algorithm without any quantization. The approach by Thomas and Luk [4] is only able to generate samples using a fixed number of resources as can be seen on the graph where three different MSE values are plotted that correspond to different level of precisions.

In addition, the results from the figure demonstrate that the proposed algorithm has the ability to produce designs across the available design space while the approach by Thomas and Luk [4] does not have this flexibility, with both approaches maintaining the same throughput.

In order to fully assess the functionality of the proposed algorithm, it is necessary to investigate the covariance matrix taking into account the quantization effects of the data path. An experiment is performed to calculate the MSE between the original covariance matrix and the empirical covariance matrix. The number of DSP blocks used is set to half of the size of the output vector N,

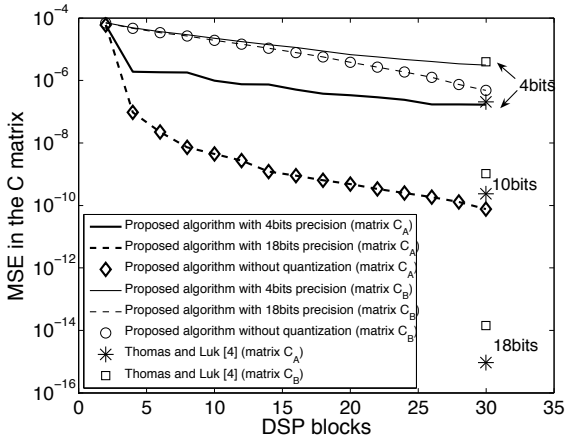


Fig. 5. Comparison of mean square error from two randomly generated matrices.

thus, in all cases, the decomposition level K is equal to $N/2$. Fig. 6 illustrates the result of this experiment. As expected, it is apparent that the MSE obtained with 18bits precision is lower than that of 4bits precision. The graph also shows the lower bound of MSE for the 4 and 18 bits precision and the case where no quantization to the coefficients is performed. Moreover, a trend can be observed where the MSE decreases as the size of the matrix increases. This behaviour depends on the structure of the covariance matrix and it should not be generalized. Matrices with different eigenvalue profiles are expected to have different levels of approximation.

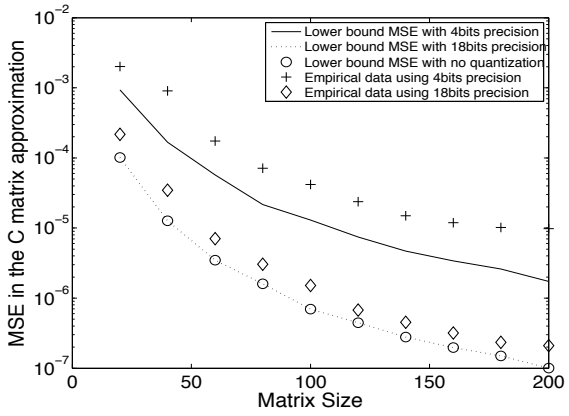


Fig. 6. MSE of covariance matrix approximation using empirical data.

5.3 Hardware Synthesis

Table 1 illustrates the results of implementing the proposed algorithm onto a Virtex-4 FPGA. In this paper the design is synthesized into hardware using Handel-C. N denotes the size of the output vector while K represents the decomposition level. Note that this data does not include the resource usage of a univariate Gaussian random number generator, it solely considers the proposed architecture. It is apparent from the table that the number of slices utilized scales linearly with the levels of decomposition. The designs generated have 101.21MHz operating frequency. Moreover, the percentage of DSP block usage is calculated based on the fact that the Xilinx Virtex-4 XC4VSX55 contains 512 available DSP blocks [14]. The table clearly illustrates the linear dependency between the decomposition levels and the required Block RAMs, DSP blocks and slice usage. On the other hand, the same design mapped onto an FPGA using Thomas and Luk approach [4] requires N number of DSP blocks where N denotes the size of the output vector.

Table 1. Resource usage in the Virtex-4 architecture using the proposed algorithm.

Configuration	Block RAM	DSP Block	Slices	DSP Blocks Usage
N=20, K=5	10	10	185	1.95%
N=50, K=10	20	20	402	3.91%
N=100, K=30	60	60	913	11.72%
N=150, K=50	100	100	1841	19.53%
N=300, K=100	200	200	3678	39.06%

In addition, synthesis results have shown that the proposed algorithm is able to generate 1 sample every 9.88ns. Therefore, the throughput of the proposed architecture is $1.01 \times 10^8/N$ vectors per second where N denotes the size of the output vector.

6 Conclusion

In this paper, a novel hardware architecture to serially generate multivariate Gaussian random samples with the ability to control the resource usage in a modern FPGA is presented. The key idea is the approximation of the lower triangular matrix using the Singular Value Decomposition algorithm for the generation of multivariate Gaussian random samples. The motivation behind this work is the need to produce a design that generates multivariate Gaussian random samples given any number of available resources and the fact that a certain error in the approximation of the covariance matrix can be tolerated. For large covariance matrices, where the size of the output vector is larger than the available DSP blocks in a single FPGA, the proposed algorithm offers the flexibility to implement the design in a single FPGA which is not possible using

the currently available approach [4]. Future work includes the investigation of the appropriate level of quantization precision given the separability of the matrix under consideration.

References

1. M. Verhofen, "Markov chain monte carlo methods in financial econometrics," *Financial Markets and Portfolio Management*, vol. 19, pp. 397–405, 2005.
2. A. Brace, D. Gatarek, and M. Musiela, "The market model of interest rate dynamics," *Mathematical Finance*, vol. 7, pp. 127–155, 1997.
3. P. Glasserman, P. Heidelberger, and P. Shahabuddin, "Variance reduction techniques for value-at-risk with heavy-tailed risk factors," in *Proceedings of the 32nd conference on Winter simulation.* San Diego, CA, USA: Society for Computer Simulation International, 2000, pp. 604–609.
4. D. B. Thomas and W. Luk, "Sampling from the multivariate gaussian distribution using reconfigurable hardware," in *Proceedings IEEE International Symposium on Field-Programmable Custom Computing Machines.*, 2007, pp. 3–12.
5. F. A. Graybill, *An Introduction to Linear Statistical Models.* McGraw Hill, New York, 1961.
6. E. M. Scheuer and D. S. Stoller, "On the generation of normal random vectors," *Technometrics*, vol. 4, no. 2, pp. 278–281, May 1962.
7. F. A. Graybill, *Introduction to Matrices with Applications in Statistics.* Wadsworth, Belmont, CA, 1969.
8. D. R. Barr and N. L. Slezak, "A comparison of multivariate normal generators," *Commun. ACM*, vol. 15, no. 12, pp. 1048–1049, 1972.
9. G. Zhang, P. H. Leong, D.-U. Lee, J. D. Villasenor, R. C. Cheung, and W. Luk, "Ziggurat-based hardware gaussian random number generator," in *Proceedings IEEE International Conference on Field Programmable Logic and Applications*, 2005, pp. 275–280.
10. D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. Leong, "A hardware gaussian noise generator using the wallace method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, pp. 911–920, 2005.
11. D.-U. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong, "A hardware gaussian noise generator using the box-muller method and its error analysis," *IEEE Transactions On Computers.*, vol. 55, no. 6, pp. 659–671, 2006.
12. D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, "Gaussian random number generators," *ACM Comput. Surv.*, vol. 39, no. 4, p. 11, 2007.
13. D. B. Thomas and W. Luk, "Non-uniform random number generation through piecewise linear approximations," *IET Computers & Digital Techniques*, vol. 1, pp. 312–321, 2007.
14. Xilinx, "Virtex-4 family overview," 2007. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf.
15. R. A. Horn and C. R. Johnson, *Matrix Analysis.* Cambridge University Press, Cambridge, 1985.
16. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C.* Cambridge University Press, 1992.
17. C.-S. Bouganis, G. A. Constantinides, and P. Y. K. Cheung, "A novel 2d filter design methodology for heterogeneous devices," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 13–22.