ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

Implementation of a Bi-Variate Gaussian Random Number Generator on FPGA without Using Multipliers

Eldho P Sunny [1], Haripriya. P [2]

M.Tech Student [VLSI & Embedded Systems] , Sree Narayana Gurukulam College of Engineering, Kadayiruppu, Kolenchery, Kerala, India[1]

Asst.Professor, Department of Electronics and Communication, Sree Narayana Gurukulam College of Engineering, Kadayiruppu, Kolenchery, Kerala, India [2]

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. Multivariate Gaussian vectors with length n are usually produced by first generating a vector of n independent Gaussian samples, then multiplying with a correlation inducing matrix requiring Operations(n^2) multiplications. This paper presents a method of generating bivariate vectors directly from the uniform distribution and eliminating the need for any multipliers. The method relies only on small read only memories and adders, and so can be implemented using only logic resources (lookup-tables and registers), saving multipliers, and block-memory resources for the numerical simulation that the multivariate generator is driving. As an optimization, only positive values are considered for the implementation. FPGA-optimized Random Number Generators (RNGs) used for uniform distribution are more resource efficient than software-optimized RNGs, as they can take advantage of bit-wise operations and FPGA-specific features. This paper use a new type of FPGA RNG called a LUT-SR RNG, which takes advantage of bit-wise XOR operations and the ability to turn LUTs into shift-registers of varying lengths. This paper model a Gaussian random number generator where its inputs elements are correlation matrix of standard deviation from previous history.

Keywords: Field-programmable gate array (FPGA), Monte Carlo simulation, bivariate samples, random number generation.

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 business sectors, where the stock prices of companies in the sector tend to rise and fall together. To simulate the behavior 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 are used to estimate how much money the institution might lose over the next day. Such long-running simulations are an ideal target for field programmable gate array (FPGA) acceleration, as they offer abundant parallelism, and are computationally bound [1]–[3]; however, they are reliant on a fast, resource-efficient, and accurate source of random samples from the multivariate Gaussian distribution. This paper is an implementation of a bivariate random number generator was resource usage to standard bit-wise logic elements.

This paper follows

1) An algorithm for generating samples from the multivariate Gaussian distribution using only uniform random bits, table- lookup, and addition;

2)A hardware architecture for implementing an MVGRNG using only lookup-tables (LUTs) and flip-flops (FFs), which allows a regular densely-packed placement strategy and achieves 500 MHz+ clock speeds.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

II. MULTIVARIATE GAUSSIAN RANDOM NUMBERS

Generation of the univariate Gaussian distribution with a specific mean, μ, and variance, σ^2, is achieved by first generating a standard Gaussian variate 'r' with mean zero and variance one, then applying a linear transformation

$$x = \sigma r + \mu \tag{1}$$

Where r is N(0, 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 in this case, each output sample is, a length n vector x. The mean and variance also increase in dimension, so the mean is a length n vector m, and the variance becomes an n × n covariance matrix Σ. 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 similar to the univariate linear transform, except the starting point is a vector r of n independent identically distributed (IID) standard Gaussian random numbers

$$x = Ar + m. \tag{2}$$

An alternative method is to use the singular value decomposition (SVD) algorithm. This decomposes the matrix into an orthogonal matrix U and a diagonal matrix S, such that $\Sigma = USUT$. This decomposition allows the construction of the solution $A = U \sqrt{} S$. The disadvantage of the SVD-based construction is that in general all the elements of the matrix are nonzero, 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. Such difficult covariance matrices frequently occur in practice [7], so this paper focuses on the use of a dense SVD-style decomposition.

III. GENERATION USING LUTS AND ADDERS

The standard generation method uses direct matrix multiplication, forming each output element from a linear combination of r (the vector of n IID Gaussian samples)

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

If there is no advance knowledge about the covariance matrix and the SVD decomposition is used, this requires n2 multiply accumulations. In addition, this method also requires the generation of the n elements of r, which are IID standard Gaussian samples. Both the generation of r and the multiplication with A require significant resources (i.e., DSPs and block-RAMs in an FPGA), so simplifying the process and reducing the resource cost are highly desirable. The method proposed in this paper is that, instead of generating expensive independent Gaussian samples and then inducing the desired covariance structure with n2 multiplications, cheap uniform samples will be converted to correlated Gaussian samples using n^2 table-lookups. Each table contains a pre-calculated discretized Gaussian distribution with the correct SD, so the only operations required are table-lookups and additions. The following text frequently refers to tables, which in this context means an array of read-only elements (a ROM) which will be implemented in the FPGA using LUTs. Unless otherwise specified, each table contains k elements, and is indexed using the syntax L[i] to access table elements L[1], . . . , L[k]. Where arrays of tables are used, sub-scripts identify a table within the array, which can then be indexed as for a standalone table, e.g., L2,3 [4]. Tables can also be interchangeably treated 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

$$x = L\,[seal[uk]] \tag{4}$$

Where u ~ U (0, 1). The central idea of this method is to construct an n × n array of tables G, such that the discrete distribution of each table Gi, j approximates a Gaussian distribution with SD Ai, j

$$G_{i,j} \sim N(0, A_{i,j}). \tag{5}$$

Now instead of starting from a Gaussian vector r, the input is an IID uniform vector u. Generation of each output element uses each element of u as a random index into the table, then sums the elements selected from each table

$$x_i = m_i + \sum_{j=1}^{n} Li,j\ seal\ [u\ j\ k] \qquad i \in 1, \ldots, n. \tag{6}$$

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

In practice k will be selected to be a power of 2, so each element of 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 cumulative distribution function (CDF) inversion. To generate a table L with SD σ, table elements are chosen according to

$$L[i] = \sigma\Phi - 1\ (i\ /(k + 1)) \qquad i \in 1, k \qquad (7)$$

Where Φ^{-1} is Gaussian inverse CDF. The table G corresponding to a given target matrix A can then be specified as

$$G\,i,j\,[z] = A\,i,j\Phi - 1\ (z/(k+1)) \qquad i,j \in 1,\ldots n, z \in 1,\ldots k. \qquad (8)$$

Construction of G allows the direct transformation of uniform samples (random bits) into multivariate Gaussian samples using [6], requiring only table-lookups and addition. Replacing continuous samples with discrete tables means that the output distribution range is no longer continuous; instead, each sample is drawn from within a discrete multidimensional lattice.

IV. HARDWARE ARCHITECTURE

The central idea in this paper, of replacing Gaussian samples and multipliers with uniform samples and tables, allows for many types of possible implementations. 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. This paper focuses on the highest performance mode of the generator, to provide the maximum contrast with previous implementations, while still providing good efficiency and quality. The specific choices made are as follows.

1) Logic Resources Only: tables are implemented using LUTs, so the only resources used are LUTs and FFs (no DSPs or block-RAMs).

2) Parallel Generation: the generator operates in a fully parallel mode, providing one new n-element vector per cycle, unlike previous approaches which generated one vector for every n cycle.

3) Maximum Clock Rate: the generator operates at the maximum realistic clock-rate for the target FPGA. For the Virtex-5 this is effectively 550 MHz, 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.

4) Regular Architecture: simulations typically consume almost all resources in the FPGA (due to replication of simulation cores), and a regular, explicitly placed highly routable, and generator allows fast place-and route while still achieving high overall clock-rate.

5) No Matrix Specialization: the generator is not optimized for a specific correlation matrix, and should support the loading of any correlation structure without structural modification.

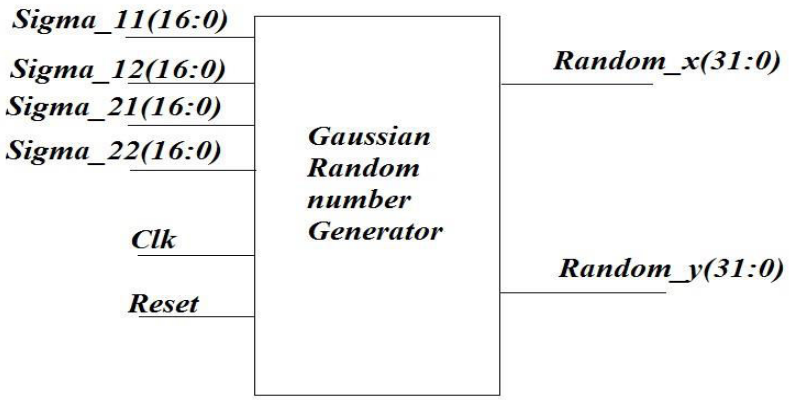

Fig. 1 Gaussian random no generator top module.

V. SIMULATIONS AND EXPERIMENTAL RESULTS

For the simulation purpose the σ values of the covariance matrices, clock and a reset signal are given as inputs. The σ values can be obtained from the expert advice or from the previous experience. The clock can attain the maximum value

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

of the used platform. The reset is made logic high for one clock for loading the sigma values i.e. the covariance matrices values and then the random number generated for corresponding σ values is available in every clock cycle after reset is made logic low. Output is random numbers for the corresponding experiment for which it can be used for fast simulations like Monte Carlo simulation.

For memory optimization and bit representation and evaluation in Xilinx, we in this experiment use σ values represented as bits. The bits can vary from 0.0000 to 9.9999, 105 values can be represented by this method by using 217 bits. This reduces the memory utilization to a great extent.So the real value of σ values is converted into its corresponding bits representation which can be obtained by multiplying the value with 104 converting it into binary and appending zeros to the MSB. So by this method real value 1 is equivalent to binary of 1000 and real value of 2 is equivalent to binary of 2000. So the 1, 2 and 3 can be represented respectively as 00010011100010000, 00100111000100000, and 00111010100110000.

The output random number generated is also in the binary format of 32 bits because of the addition involved in obtaining output and it can be converted into its real values by multiplying with 10-8 and convert to decimal. The obtained σ values are used for updating the LUT values by multiplying each RAM values with the corresponding co-variance matrics values (σ values). Then this updated RAM-LUT is used for random number generation.

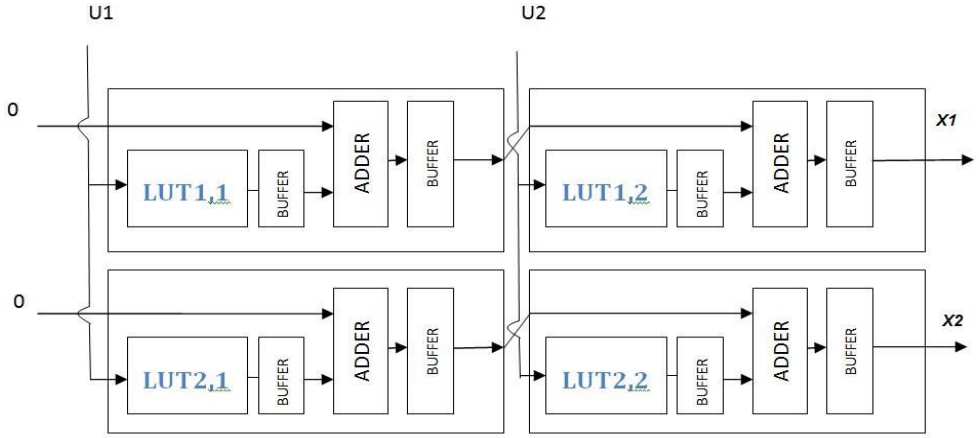

Fig. 2 Architecture of the bivariate Gaussian random number generator.

A bivariate random number generator is implemented in the vertex 5 platform and many optimizations are done for the fast processing and less memory usage. An optimization is on avoiding the negative values understanding that the Gaussian profile maps its mirror image in both positive and negative side. So the adder used here works in the first quadrant. The LUTs used here also avoids negative values and reduce memory in terms of an extra bit needed to indicate it. This also gives more accurate outputs as the number of samples taken exactly doubles. The values stored in the LUT'S have equal probability [5].

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

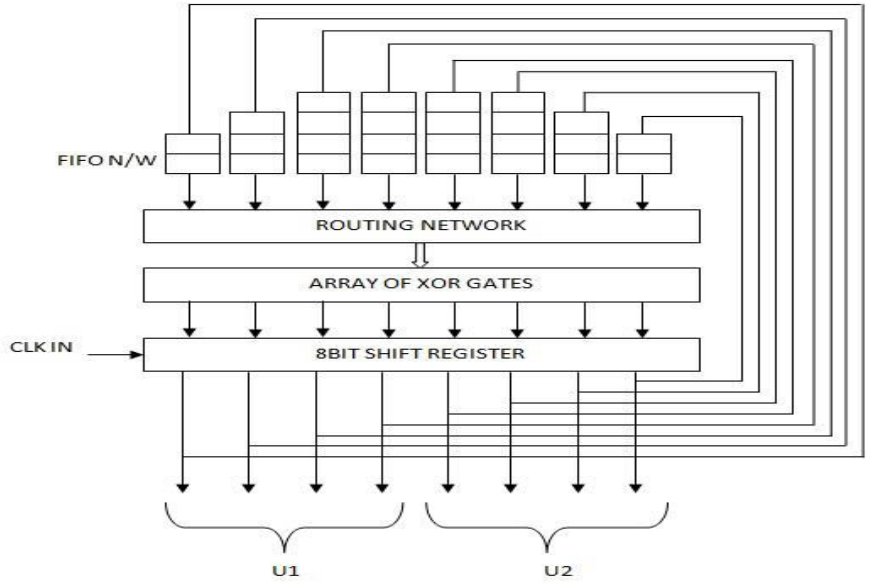

Fig. 3 Architecture of the IID uniform vector generator u1and u2.

This type of FPGA-optimized uniform random number generator, called a LUT-SR RNG. These RNGs takes advantage of the ability to configure LUTs as independent shift-registers, allowing high-quality long period generators to be implemented using only a small amount of logic. In addition the period and quality scale with the number of output bits, unlike generators adapted from software. A key advantage of the LUT-SR generators over previous FPGA-optimized uniform random number generators is that they can be reconstructed using a simple algorithms. In concert with the tables of maximum period generators, this allows FPGA engineers to use the new RNGs without needing to find generator instances themselves [6].

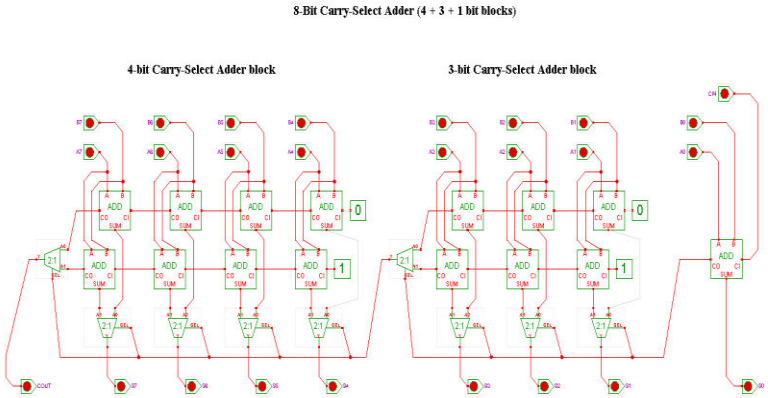

Fig. 4 Architecture of Adder module (Carry select adder)

Carry select adder is used for the fast addition. The basic idea is to add 2 bits using 3 1-bit full adders and a 2-bit multiplexer. One adder adds the least significant bit in the normal fashion. The other two add the most significant bit, one of them assuming that the carry in will be 0 and the other assuming that the carry in will be 1. Then the multiplexer chooses the output from one of these adders based on the carry produced by the adder for the least significant bit [8].

TABLE I

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

THE DEVICE UTILISATION SUMMARY GENERATED USING XILINX ISE 14.1.

gauss Project Status (08/27/2013 - 01:23:39)			
Project File:	gauss2.xise	**Parser Errors:**	No Errors
Module Name:	gauss	**Implementation State:**	Synthesized
Target Device:	xc5vlx20t-2ff323	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	161 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1342	12480	10%
Number of Slice LUTs	466	12480	3%
Number of fully used LUT-FF pairs	200	1608	12%
Number of bonded IOBs	134	172	77%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	4	24	16%

TABLE II

THE TIMING ANALYSIS REPORT.

	Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	Autotimespec constraint for clock net clk_BUFGP	SETUP HOLD	0.297ns	5.811ns	0	0 0

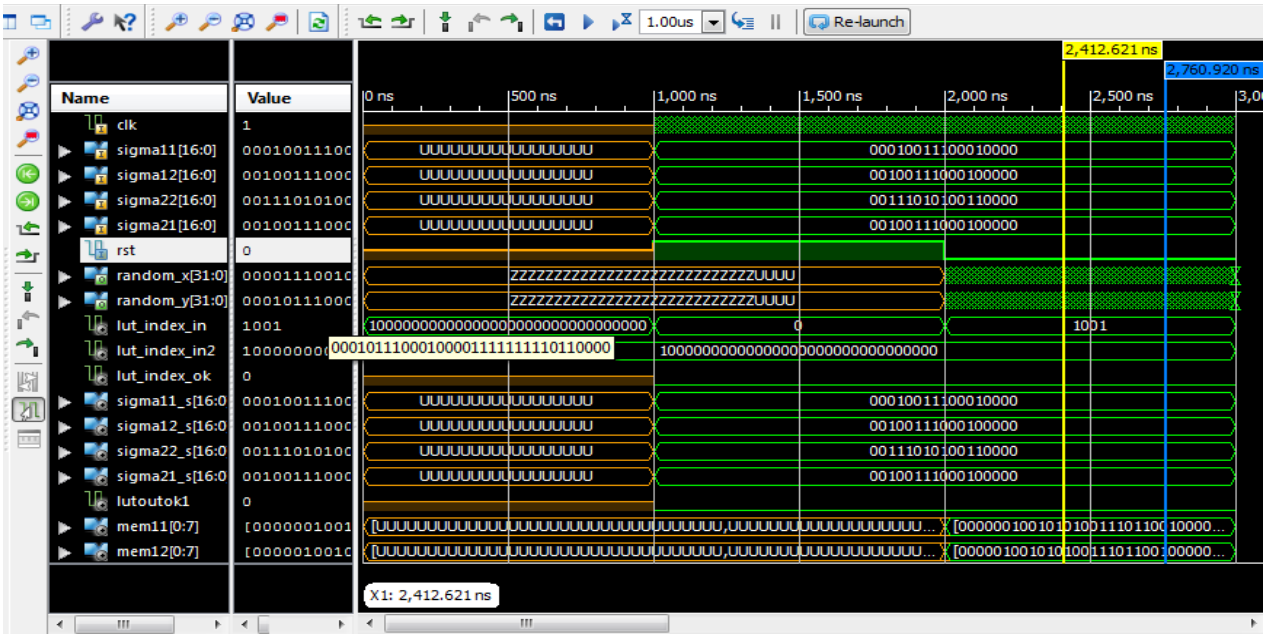

Fig. 5 Output window generated using isim simulator Xilinx 14.1

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 1, December 2013

VI. CONCLUSION

This paper presented a method for bi-variate Gaussian random number generator in FPGAs, which decomposes the generation into a series of table-lookups and additions. This method can be extended to use in numerical Monte Carlo simulations; risk analysis etc 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. This can be extended to multivariate generators as a future work. Adder/subtracter module can be included for getting more accurate and extended values.

ACKNOWLEDGMENT

The authors would like to thank the Department of Electronics and communication, SNGCE Kadayirippu for their support. We would also like to thank friends and resource persons who give technical support, valuable advice and encouragement throughout the project.

REFERENCES

[1] D. B. Thomas and W. Luk, "Credit risk modelling using hardware accelerated monte-carlo simulation," in Proc. ACM Symp. FPGAs Custom Comput. ach., 2008, pp. 229–238.
[2] N. Woods and T. VanCourt, "FPGA acceleration of Quasi-Monte Carlo in finance," in Proc. Int. Conf. Field Programm. Logic Appl., 2008, pp. 335–340.
[3] A. Kaganov, P. Chow, and A. Lakhany, "FPGA acceleration of Monte- Carlo based credit derivative pricing," in Proc. Int. Conf. Field Programm. Logic Appl., 2008, pp. 329–334.
[4] D. B. Thomas and W. Luk, "Multivariate Gaussian random number generation targeting reconfigurable hardware," ACM Trans. Recon. Technol. Syst., vol. no. 12, pp. 22–26, 2008.
[5] Multiplierless Algorithm for Multivariate Gaussian Random Number Generation in FPGAs David B. Thomas, Member, IEEE, and Wayne Luk, Fellow, IEEE
[6] FPGA-Optimised Uniform Random Number Generators using LUTs and Shift Registers David B. Thomas and Wayne Luk Imperial College London
[7] N. J. Higham, "Computing the nearest correlation matrix—a problem from finance," IMA J. Num. Anal., vol. 22, pp. 329–343, Oct. 2002.
[8] Bedrij, O. J., (1962), "Carry-select adder," IRE Trans. Electron. Comput. Pp.340–344.