

# MONTE CARLO LOGARITHMIC NUMBER SYSTEM FOR MODEL PREDICTIVE CONTROL

*Panagiotis D. Vouzis*

Computer Science & Engineering  
Lehigh University  
Bethlehem, PA, 18015, USA

*Mark G. Arnold*

Computer Science & Engineering  
Lehigh University  
Bethlehem, PA, 18015, USA

*Sylvain Collange*

École Normale Supérieure de Lyon  
46 Allée d'Italie  
69364 Lyon Cedex 07, France

*Mayuresh V. Kothare*

Chemical Engineering  
Lehigh University  
Bethlehem, PA, 18015, USA

## ABSTRACT

Simple algorithms can be analytically characterized, but such analysis is questionable or even impossible for more complicated algorithms, such as Model Predictive Control (MPC). Instead, Monte Carlo Arithmetic (MCA) enables statistical experimentation with an algorithm during runtime for detection and mitigation of numerical anomalies. Previous studies of MCA have been limited to software floating point. This paper studies how MCA can be used in an FPGA implementation of the Logarithmic Number System (LNS), forming the Monte Carlo Logarithmic Number System (MCLNS). Simulation studies present how MCLNS affects the accuracy vs. performance of an MPC implementation, and synthesis results give an estimate of the cost of utilizing MCLNS in a Xilinx Virtex-IV FPGA.

## 1. INTRODUCTION

Guaranteeing that a numerical algorithm is stable and gives a result within a desired accuracy interval is a challenging task, especially for elaborate algorithms. Usually the end user of a processor that carries out the arithmetic operations, works at a high level of abstraction; thus he/she is unaware of the underlying limitations of the hardware and it is not possible to determine how accurate the result of a computation is.

For a given input of a problem, the outcome produced by an arithmetic algorithm lies within a certain distance from the exact solution. This distance depends on many factors, such as the precision of the representation of the hardware carrying out the operations, the internal rounding method of the hardware, the input data, and the algorithm itself. Although there are analytic methods to evaluate the stability and accuracy of numerical algorithms, these deal mainly

with round-off and truncation errors and rely on bounding the forward and backward error [1], which sometimes is a pessimistic estimation. For complicated numerical algorithms it is sometimes impossible to carry out this backward/forward error analysis. Although it is important to be aware of the stability properties of an algorithm, when instability occurs there is no means to mitigate its effects, while appearances of catastrophic cancellation (which occurs when almost identical numbers are subtracted resulting in an erroneous result in many significant digits) is unpredictable.

The accumulation of round-off errors is not too damaging since it gives a result that lies in a certain radius around the correct value, while underflows and overflows can be detected and mitigated by a saturation policy. However, catastrophic cancellation is difficult to detect and to lessen its consequences. In [2] Daumas and Matula present an efficient architectural design that guarantees one unit-in-the-last-place (ulp) accuracy for a double-precision IEEE dot product in the absence of catastrophic cancellation. The occurrence of catastrophic cancellation can only be detected but it cannot be resolved in any way.

Forward/backward error analysis [1] has limitations, especially when it comes to complicated algorithms. These limitations can be addressed by a new method, called Monte Carlo Arithmetic (MCA), which is trying to address numerical accuracy from a different perspective—by experimental calculations. MCA is introduced in [3] where it is proposed the use of Monte-Carlo (MC) iterations to detect and mitigate anomalies in Floating-Point (FP) arithmetic caused by catastrophic cancellation and ill-conditioned problems. This method can be used at both the software and the hardware levels, and additionally it can give an approximation of the correct result by avoiding the failure of the computation. The main drawback is the inefficiency of MC iterations, but

this aspect can be included in the trade-off between reliability and efficiency of an implementation.

The effects of round-off errors and catastrophic cancellation become more pronounced as the precision is reduced, such as in custom-designed hardware for specific applications. The reduction of the wordlength is done in order to solve a problem with just sufficient accuracy to reduce the area, the speed, and the power consumption of the circuit. The direct drawback of the reduced accuracy is the increased susceptibility of the circuit to round-off errors that accumulate with consecutive arithmetic operations (especially for ill-conditioned problems), and increased occurrence of overflows and underflows. Another side effect of reduced-precision arithmetic is the increased probability of the appearance of catastrophic cancellation.

In this work we study the effect of utilizing MCA in the context of implementing Model Predictive Control (MPC) [4] by using the Logarithmic Number System (LNS) [5]. MPC is an optimization-based control algorithm that recently has been shown to be advantageous for small-physical-size and portable applications [6, 7]. Utilizing MPC requires a hardware implementation capable of carrying the computational burden of the algorithm in real time on a reduced-area footprint. In [8] an architecture is proposed based on a microprocessor combined with a matrix coprocessor capable of efficient execution of the arithmetic operations of an MPC algorithm in real time. For this implementation the underlying hardware utilizes a 16-bit LNS unit to carry out the arithmetic operations. This choice is based on the work by Garcia et al. [9] where it was shown that a 16-bit LNS unit offers enough accuracy for the particular cases studied, while, without sacrificing performance, LNS occupies 40% less area than the standard choice of a 16-bit FP unit.

The 16-bit wordlength, instead of the ubiquitous 64-bit FP arithmetic, brings up the aforementioned problems of catastrophic cancellation and round-off errors more intensively. Using MCA in the software level with FP to address these issues is covered in [3]. In this paper we study how MCA can be used at the hardware level, and how a reprogrammable FPGA allows us to experiment with the accuracy and the performance of an MPC algorithm that utilizes Monte Carlo Logarithmic Number System (MCLNS).

In the next two sections we introduce the LNS and MPC respectively. In Section 4 we describe how catastrophic cancellation and ill-conditioning are addressed by MCA. Section 5 gives specific arithmetic algorithms for MCLNS multiplication and addition. Section 6 describes the LNS unit that is used for the MPC algorithm, while Section 7 presents the Pseudo-Random-Number Generator (PRNG) added to the LNS unit to implement MCLNS and gives synthesis results for a Xilinx FPGA. Finally, conclusions are drawn in the last section.

## 2. THE LOGARITHMIC NUMBER SYSTEM

MCA has been proposed and studied in conjunction with the FP number system [3]. Since LNS acts as an alternative to FP [5], MCA can be applied to LNS, where a real number,  $X$ , is represented by the quantized logarithm of its absolute value,  $x = Q_f(\log_b |X|)$ , where  $Q_f(\cdot)$  rounds its input to fit in  $f$  bits. The similarity to FP is apparent, since in LNS the two's-complement logarithm,  $x$ , consists of an integer part (the counterpart of the exponent of a FP number), and an  $f$ -bit fractional part (the counterpart of the FP mantissa). In LNS, multiplication and division are reduced to addition and subtraction, which is what makes LNS attractive.

Adding or subtracting in LNS is more elaborate since these operations are calculated via

$$\log_b(X + Y) = \max(x, y) + s_b(z) \quad (1)$$

$$\log_b(X - Y) = \max(x, y) + d_b(z), \quad (2)$$

where  $z = -|x - y|$ ,  $s_b(z) = Q_f(\log_b(1 + b^z))$ , and  $d_b(z) = Q_f(\log_b(1 - b^z))$ . The calculation of sums,  $s_b$ , and differences,  $d_b$ , are the main computational burden of an LNS arithmetic unit. The area to tabulate these functions grows exponentially with respect to  $f$ .  $s_b$  and  $d_b$  converge to zero for  $z \rightarrow -\infty$ , and after a specific point called *essential zero*,  $e_z \simeq -f$  (or  $E_z \simeq 2^{-f}$  in the real domain), the functions need not be tabulated:  $\forall z < e_z, s_b(z) = d_b(z) = 0$ .

## 3. MODEL PREDICTIVE CONTROL

MPC is an advanced-control algorithm that uses a model of the system under control to predict its future behavior and to calculate a number of control moves in order to move the plant to the desired state optimally with respect to a predetermined criterion. MPC gained popularity because it can handle multiple-input-multiple-output systems, and it can take into account constraints and disturbances explicitly.

A typical objective function that has to be optimized has the form:

$$J_P(t) = \sum_{k=0}^P \{ [y(t+k|t) - y_{\text{ref}}(k)]^2 + Ru(t+k|t)^2 \}, \quad (3)$$

$$|u(t+k|t)| \leq b, \quad k \geq 0, \quad (4)$$

where  $y(t+k|t)$  are the predicted future outputs of the system at time  $t+k$  ( $k = 1, 2, \dots, P$ ) ( $P$  is called the *prediction horizon*) calculated at time  $t$ ,  $y_{\text{ref}}(k)$  is the desired trajectory of the output,  $u(t+k|t)$  are the calculated optimal control moves of the system at time  $t+k$  calculated at time  $t$ ,  $R$  is a design parameter used to weight the control moves, and  $b$  is the vector of the constraints that the future inputs have to obey. The number of control moves,  $M$  ( $M$  is called the *control horizon*), can be smaller than  $P$ , and in this case only

the first  $M$  moves are calculated, while the rest,  $P - M$ , are considered zero.

The optimization of the objective function (3) with respect to  $u$ , under the constraints (4) can be calculated numerically by using Newton's optimization algorithm which requires the calculation of the Hessian  $H(J_P)$ , the Gradient  $\nabla(J_P)$ , and a Gauss-Jordan inversion in multiple iterations of

$$u(t+1) = u(t) - H(J_P)^{-1} \cdot \nabla(J_P), \quad (5)$$

where

$$\nabla(J_P) = \Gamma_u u + \Gamma_x x + \Gamma_y y + \mu \Phi \quad (6)$$

$$H(J_P) = \Gamma_u + (2\mu \mathbf{I}_M^T \Psi) \cdot \mathbf{I}, \quad (7)$$

$$\Phi = \left[ \frac{1}{(u_1 - b)^2} - \frac{1}{(u_1 + b)^2} \cdots \frac{1}{(u_M - b)^2} - \frac{1}{(u_M + b)^2} \right]^T$$

$$\Psi = \left[ \frac{1}{(u_1 - b)^3} + \frac{1}{(u_1 + b)^3} \cdots \frac{1}{(u_M - b)^3} + \frac{1}{(u_M + b)^3} \right]^T$$

$\mathbf{I}_M$  is a length- $M$  vector of ones, and  $\mathbf{I}$  is an  $M \times M$  identity matrix. For a high-end workstation the solution of this problem may not be challenging, but for small-physical-scale applications the fact that the controller has to occupy reduced area and meet real-time requirements may render a general-purpose processor computationally inadequate for this problem. To address this issue in [8] a custom-designed architecture is proposed that consists of a general-purpose processor and a matrix coprocessor implemented respectively as a CPU core and an one-hot state machine synthesized on an Xilinx FPGA. The computationally demanding parts of the algorithm are executed on the matrix coprocessor, which has an instruction set tailored to accelerate particular operations of the algorithm. The microprocessor acts as the master on the coprocessor by sending the appropriate commands, and by carrying out the high-level (and with less computational burden) operations of the algorithm. This is the system that has been used to simulate the MPC with MCA for a case study of an antenna-control problem taken from [10].

#### 4. MODEL PREDICTIVE CONTROL WITH MONTE CARLO ARITHMETIC

The appearance of either of the two problems of catastrophic cancellation or ill-conditioned matrices can lead to the complete failure of an MPC algorithm since the error can be orders of magnitude off the correct value. Thus, for cases where the safe operation of the system is extremely critical, like the glucose regulation problem [7], there is a necessity for a way to escape the failure of the algorithm, or at the very least to detect the problem.

There is not a strict definition of MC iterations for arithmetic. In general MC methods refer to the presence of some kind of uncertainty in a simulation of an algorithm which is carried out multiple times. The uncertainty gives different

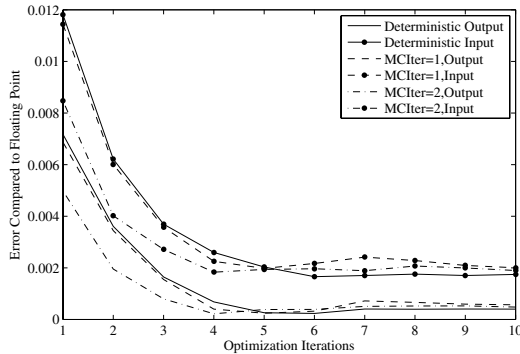
results for different runs, and their average is considered the final outcome of the algorithm.

In the MPC implementation in [8] the uncertainty is introduced in the form of noise in every LNS multiplication. If  $x = \log_b |X|$ ,  $y = \log_b |Y|$  and  $r = x + y$  is the result of the multiplication given by the LNS unit, the MC multiplication is  $\hat{r} = r + \delta$  where  $\delta$  is a variable with uniform distribution in the interval  $[-2^{-t}, 2^{-t}]$ , where  $t$  is a parameter. For example, if we apply these to the LNS unit used in [8] that has  $l = 9$  fractional bits, and we choose  $t = 9$ , then  $\delta = [-2^{-9}, 2^{-9}]$ , which means the MC multiplication affects only the Least-Significant Bit (LSB) of the LNS multiplication.

The algorithm, with the inserted noise, gives slightly different results each time it is executed. However, if the algorithm is executed multiple times, the averaged results of the MC iterations approximate the solution given by the uncertainty-free or deterministic solution. Catastrophic cancellation can appear when using MC simulation the same way it appears in the deterministic case, but since in each MC iteration the operands are slightly different we can safely assume that catastrophic cancellation will not appear in all MC iterations. In the iterations where catastrophic cancellation occurs, the result is very different compared to the other iterations; a discrepancy is used as an alarm indication. This information is already valuable since it is known that the result is wrong and it is better not to apply it to the plant, for it may have unpredictable consequences. Moreover, the results produced in the presence of catastrophic cancellation can be detected and rejected, and the control move to be applied is chosen to be the average of the rest of the MC iterations. Thus, this algorithmic approach offers both detection of catastrophic cancellation and an estimation of the correct control move to be applied to the plant. If catastrophic cancellation occurs in the deterministic simulation of the algorithm; then no matter how many times it is executed, the result is going to be the same. Deterministic arithmetic can offer no indication whether something wrong occurred or not.

The MC iterations can be applied in the same way in cases that ill-conditioned matrices appear. It is arithmetically unstable to calculate the inverse of a matrix when the condition number is large, and the injected uncertainty gives inverses that are different from each other and different from the one calculated by the deterministic algorithm. However, the average of the inverses produced by the MC iterations may be more accurate than the deterministic solution [3].

The immediate result of the introduced noise is the deterioration in the accuracy and the precision of the result. This is depicted in Fig. 1 where the average error is presented on the input and the output of an antenna control problem [10] compared to the solution given by double FP arithmetic. The horizontal axis represents the optimization

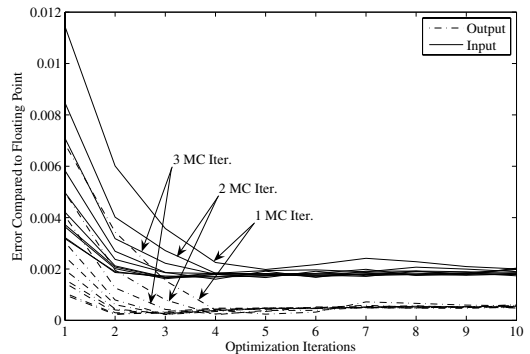


**Fig. 1.** Average error on the antenna control problem by the deterministic LNS and MCLNS ( $P = 5, M = 2$ ).

iterations that Eq. (5) is executed in order to calculate one control move. It is shown that after five iterations the best error performance is achieved at the output, and it remains relatively steady from that point on. In this simulation study  $t = 9$ , i.e., only the LSB of each LNS multiplication is affected. The MC method only makes sense to be applied with two or more iterations—only then it is possible to detect the occurrence of catastrophic cancellation with the assumption that it appears in only one of the two iterations. Additionally, if it is desired to be able to reject the catastrophic cancellation and have a correct estimate of the result at least three MC iterations are required—the two similar solutions are averaged and the third, that is way different compared to the other two, is rejected. The case of one MC iteration is depicted in Fig. 1 just to give a visualization of the effect of the injected MC noise.

The MC method causes a degradation in performance by a factor equal to the number of MC iterations used, since each optimization step is executed multiple times. Consequently, in order to compare the accuracy achieved by the deterministic case and the MC method we have to compare cases equal in computational effort. Thus, the case of two MC iterations with one optimization iteration is computationally equal to the case of two deterministic optimization iterations, or the case of two MC iterations with two optimization iterations is equal to the case of four deterministic optimization iterations. When we compare equivalent points in Fig. 1 it is observed that the MC method degrades the accuracy of the algorithm, and this is the cost to be paid in order to have detection and mitigation of numerical anomalies.

It is obvious that, for a particular number of optimization iterations, the more MC iterations the smaller the error, and the bigger the computational effort. This is depicted in Fig. 2 where the error on the input and the output of the antenna control problem is shown for up to ten MC iterations. The choice of the appropriate number of MC iterations that need to be used by a particular application depends on three factors. First of all is the required accuracy of the solution. The



**Fig. 2.** Average error on the antenna control problem by the MCLNS for MC iterations 1 to 10 ( $P = 5, M = 2$ ).

MC method cannot achieve the accuracy of the deterministic case with the same computational effort, but the simulation study presented helps to make the appropriate choice in order to meet the accuracy requirement. Another factor is the desired performance of the system which again can be calculated by knowing the number of MC iterations required by the particular problem. The performance is further decreased by the extra time required to process the multiple MC results (calculate average, detect and reject catastrophic results) which is only a fraction of the extra time required by the multiple MC iterations, thus not that important. Last but not least, is the probability of appearance of catastrophic cancellations and the level of ill-conditioning of a matrix.

Catastrophic cancellation appears with a low probability, thus it is usually safe to assume that in a certain number of MC iterations it will appear in only one of them. Assuming this, in order to detect catastrophic cancellation three MC iterations are enough, since in only one of the three is the result going to be different, which is easily detectable. However, if it is assumed that catastrophic cancellation can appear more than once in a number of MC iterations, then we have to make the appropriate choice of iterations to guarantee detection. The detection of catastrophic cancellation is not completely guaranteed by using MC iterations since theoretically it can occur in all MC iterations, but the more MC iterations the bigger the probability that catastrophic cancellation will be detected. For ill-conditioned matrices the criterion is different. The higher the condition number of a matrix the more MC iterations are required in order to make a better estimate of the correct inverse. This parameter can be decided after simulation and assessment of the worst-case condition number.

For the case study examined here a reasonable choice for the MC method is to choose the case of 3 MC iterations and 4 optimization iterations, in order to achieve the same error performance with the deterministic case. We can see that the deterministic case achieves the best error performance of  $2.4 \cdot 10^{-4}$  after the threshold of 5 optimization iterations, while the MC method requires  $3 \times 4 = 12$  iterations.

## 5. MCLNS ALGORITHMS

Deterministic arithmetic is the standard implementation for both FP and LNS. Even though there may be an infinite set of bits in the exact value, deterministic arithmetic assumes all the discarded bits (those beyond the  $f$  bits of precision) are zeros. In the case of LNS, an arbitrary real value,  $X$ , is converted to its finite base- $b$  logarithmic representation,  $x = Q_f(\log_b |X|)$ , which only records  $f$  bits of precision.

For multiplication ( $R = XY$ ), deterministic LNS uses a finite addition ( $r = x + y$ ), where the resulting output representation,  $r$ , has the same assumption of an infinite set of discarded bits. For addition ( $R = X + Y$ ), assuming  $x > y$ , from (1) deterministic LNS produces the result as  $r = x + s_b(y - x)$ . There are two possible cases: when  $x$  and  $y$  are close ( $|y - x| < f$ ) versus when  $x$  and  $y$  are far apart: ( $|y - x| > f$ ). In the former case,  $s_b(y - x)$  is nonzero so  $r > x$ , which makes the machine state change to reflect the sum. In contrast, the latter case (the *essential-zero* case),  $s_b(y - x)$  rounds to zero so that  $r = x$ , which means the machine state remains unchanged. The hardware “forgets” that the sum happened. The solution in deterministic arithmetic to overcome this is to increase  $f$ , which causes a corresponding increase in hardware cost.

Monte-Carlo arithmetic takes a different approach. Like deterministic LNS, MCLNS stores  $f$  bits of precision. The difference is that we assume the infinite set of truncated bits form an unbiased random value. For MCLNS multiplication, the random part of the result,  $\xi_{xy}$ , is not recorded, but instead assumed to come from the same distribution as the inputs. Thus, MCLNS requires finding  $\xi_r$  such that  $\xi_{xy} = \xi_x + \xi_y - \xi_r$  can match that distribution, which is equivalent to

$$\begin{aligned} P(\xi_x + \xi_y > 2^{-f-1}) &= P(\xi_r = 2^{-f}) = 0.25 \\ P(|\xi_x + \xi_y| < 2^{-f-1}) &= P(\xi_r = 0) = 0.5 \\ P(\xi_x + \xi_y < -2^{-f-1}) &= P(\xi_r = -2^{-f}) = 0.25. \end{aligned}$$

In other words, the hardware computes  $r = x + y + \xi_r$ , which means the result rounds up one quarter of the time at random; the result rounds down one quarter of the time at random; and the rest of the time the result would be identical to deterministic LNS.

MCLNS addition reduces the hardware cost of overcoming the “forgotten sum” problem in the essential-zero case ( $x - y > f$ ), where  $s_b(z) = 0$  since  $\log_b(1 + b^z)$  has more than  $f$  leading zeros. Roughly speaking, the number of leading bits that are zeros in  $\log_b(1 + b^z)$  is  $\lfloor x - y \rfloor$ . The only case in which this tiny  $\log_b(1 + b^z)$  impacts the bits that are actually stored in MCLNS is when the leading  $\lfloor x - y \rfloor - f$  bits of  $\xi_x$  are all ones. The probability of this is  $P(\xi_r = 2^{-f}) \approx 2^{-(\lfloor x - y \rfloor - f)}$ . Because addition of like signs always produces a larger absolute sum,  $P(\xi_r = -2^{-f}) = 0$ . The MCLNS essential-zero case

causes the machine state to change at random. In the close case ( $|y - x| < f$ ), we simply choose  $\xi_r$  as in the multiplication case. Similar cases apply to  $d_b$ , including one involving catastrophic cancellation.

## 6. THE LNS UNIT

The simulation study presented in Section 4 was carried out in order to estimate the impact of using MCLNS with MPC. The implementation choice of the LNS unit is critical due to the tradeoffs among area, latency, and accuracy offered by alternative techniques. Since MCA causes precision degradation, it is of great benefit to choose an implementation that offers the best accuracy.

The precision of an LNS unit is constrained by the memory size for  $s_b$  and  $d_b$ . For the same accuracy,  $d_b$  requires more memory due to its singularity at zero; thus it is common to relax the accuracy of  $d_b$  close to zero to reduce area. The publicly-offered LNS VHDL libraries [11] (which include the implementation techniques of multipartite-tables [12] and Single-Multiplication-Second-Order (SMSO) interpolation [13]) follow this relaxation approach. Since accuracy is important for MCLNS, we chose to use a more accurate variation of the VHDL libraries in [11] which is described in [14] and is based on the “Improved LNS Cotransformation” [15]. Cotransformation achieves the same accuracy for both  $s_b$  and  $d_b$  with a reduction in the circuit area compared to [11]. With cotransformation the evaluation of the  $d_b$  function is done via the  $s_b$  function and three smaller tables, which overall offer a smaller memory size than the explicit calculation of the  $d_b$  function. The  $s_b$  can be evaluated by any of the multipartite-tables or the SMSO techniques.

## 7. SYNTHESIS WITH PRNG UNIT

Since the objective is to implement MCLNS in hardware, a PRNG is required to provide the random numbers used as noise. We use a triple-Tausworthe 32-bit Uniform-RNG (URNG), described by Lee et al. [16], who claim it has good randomness and throughput/area ratio. Although the 32-bit wordlength of the URNG is large for MCLNS libraries with  $f = 7$  to  $f = 13$ , this size makes it unlikely parallel LNS units will have problems with correlated pseudorandom noise.

The publicly-available VHDL libraries in [11] encompass deterministic LNS operators implemented by the multipartite-table method for  $f = 7$  to  $f = 13$  and with the SMSO method for  $f = 10$ ,  $f = 11$  and  $f = 13$ . The deterministic units were improved to use cotransformation [14]. This deterministic LNS library was synthesized along with a novel version that incorporates the above MCLNS and URNG hardware using Xilinx Webpack 9.1i with a Virtex-

**Table 1.** Synthesis results of the LNS and MCLNS units.

	$f$	LNS	MCLNS	extra %	LNS	MCLNS
		CLBs	CLBs		ns	ns
M u l t i	7	229	266	16.2	16.9	17.0
	8	302	340	12.6	17.8	17.9
	9	407	446	9.6	19.0	18.6
	10	609	640	5.1	20.0	20.0
	11	696	745	7.0	22.2	21.5
	12	918	961	4.7	23.4	22.9
	13	1421	1482	4.3	24.9	23.8
S	10	589	651	10.5	25.0	24.3
M	11	686	721	5.1	24.8	23.5
S	13	1280	1319	3.0	27.6	26.7
O						

IV FPGA for different values of  $f$ . The Configurable-Logic-Unit (CLB) increase, compared to [14], due to integrating this URNG unit in the LNS adder/subtractor, is presented in Table 1 for both the multipartite-table (upper seven lines) and the SMSO (lower three lines) methods. The percentage difference between MCLNS and deterministic LNS decreases as  $f$  increases. This difference would be smaller if the URNG was tailored to the wordlengths of the LNS units. The impact on the Critical Path in ns is practically insignificant.

## 8. CONCLUSIONS

MCLNS is a new arithmetic alternative that combines the easy multiplication and division of LNS with the statistical and experimental benefits of MCA to observe the numerical stability of complex algorithms that need to execute quickly on resource-constrained FPGA systems. MPC is an advanced control algorithm, for which we have implemented an FPGA-based LNS matrix processor [8], which benefits from the analysis possible with MCLNS. The FPGA synthesis results show that for all but the lowest precision, the cost of adding a random number generator to implement MCLNS is small compared to the total FPGA resources required just for LNS. The MCLNS VHDL code, which is an enhancement of [11], is available at <http://www.cse.lehigh.edu/~caar>.

## 9. REFERENCES

- [1] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1996.
- [2] M. Daumas and D. W. Matula, "Validated Rounding of Dot Products by Sticky Accumulation," *IEEE Transactions on Computers*, vol. 46, pp. 623–629, May 1997.
- [3] S. Parker, B. Pierce, and P. R. Eggert, "Monte Carlo Arithmetic: How to Gamble with Floating Point and Win," *Computing in Science & Eng.*, vol. 2, pp. 58–68, Jul./Aug. 2000.
- [4] E. F. Camacho and C. Bordons, *Model Predictive Control*. Springer, New York, 1999.
- [5] E. E. Swartzlander and A. G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE Transactions on Computers*, vol. 24, pp. 1238–1242, Dec. 1975.
- [6] L. G. Bleris, J. G. Garcia, M. G. Arnold, and M. V. Kothare, "Model Predictive Hydrodynamic Regulation of Microflows," *Journal of Micromechanics and Microengineering*, vol. 16, pp. 1792–1799, Sept. 2006.
- [7] R. S. Parker, F. J. Doyle III, and N. A. Peppas, "The Intravenous Route to Blood Glucose Control," *IEEE Engineering in Med. and Biology*, vol. 20, pp. 65–73, Jan./Feb. 2001.
- [8] P. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare, "A Custom-made Algorithm-Specific Processor for Model Predictive Control," in *Int. Symposium of Industrial Electronics*, vol. 1, Montreal, Canada, 9–13 July 2006, pp. 228–233.
- [9] J. Garcia, L. Bleris, M. G. Arnold, and M. V. Kothare, "LNS Architectures for Embedded Model Predictive Control Processors," in *Proceedings of the CASES International Conference*, Washington DC, 22–25 Sept. 2004, pp. 79–84.
- [10] M. V. Kothare, V. Balakrishnan, and M. Morari, "Robust Constrained Model Predictive Control Using Linear Matrix Inequalities," *Automatica*, vol. 32, pp. 1361–1379, Oct. 1996.
- [11] J. Detrey and F. de Dinechin, "A VHDL Library of LNS Operations," in *the Asilomar Conference on Signals, Systems, and Computers*, vol. 2, Pacific Grove, CA, 9–12 Nov. 2003, pp. 2227–2231.
- [12] F. de Dinechin and A. Tisserand, "Some Improvements on Multipartite Table Methods," in *Proc. of the Symposium on Computer Arithmetic*, Vail, Colorado, 2001, pp. 128–135.
- [13] J. Detrey and F. de Dinechin, "Second Order Function Approximation Using a Single Multiplication on FPGAs," in *Field-Programmable Logic and Applications: 14th International Conference, FPL 2004*, LNCS, no. 3203. Antwerp, Belgium: Springer-Verlag, Sep 2004, pp. 221–230.
- [14] P. Vouzis, S. Collange, and M. Arnold, "Cotransformation Provides Area and Accuracy Improvement in an HDL Library for LNS Subtraction," Accepted for *The EuroMicro Conference on Digital Systems and Design*, Lübeck, Germany, 27–31 August 2007.
- [15] M. G. Arnold, "An Improved Cotransformation for Logarithmic Subtraction," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS'02)*, Scottsdale, Arizona, 26–29 May 2002, pp. 752–755.
- [16] 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.