Low Discrepancy Sequences for Monte Carlo Simulations on Reconfigurable Platforms

Ishaan L. Dalal, Deian Stefan and Jared Harwayne-Gidansky
The Cooper Union for the Advancement of Science and Art
51 Astor Place, New York, NY 10003
{ishaan, stefan, harway}@cooper.edu

Abstract

Low-discrepancy sequences, also known as "quasi-random" sequences, are numbers that are better equidistributed in a given volume than pseudo-random numbers. Evaluation of high-dimensional integrals is commonly required in scientific fields as well as other areas (such as finance), and is performed by stochastic Monte Carlo simulations. Simulations which use quasi-random numbers can achieve faster convergence and better accuracy than simulations using conventional pseudo-random numbers. Such simulations are called Quasi-Monte Carlo.

Conventional Monte Carlo simulations are increasingly implemented on reconfigurable devices such as FPGAs due to their inherently parallel nature. This has not been possible for Quasi-Monte Carlo simulations because, to our knowledge, no low-discrepancy sequences have been generated in hardware before. We present FPGA-optimized scalable designs to generate three different common low-discrepancy sequences: Sobol, Niederreiter and Halton. We implement these three generators on Virtex-4 FPGAs with varying degrees of fine-grained parallelization, although our ideas can be applied to a far broader class of sequences. We conclude with results from the implementation of an actual Quasi-Monte Carlo simulation for extracting partial inductances from integrated circuits.

1. Introduction

In general, *Monte Carlo* (MC) methods are algorithms which use random sampling to stochastically model systems. A specific MC method is Monte Carlo integration—a numerical integration technique to approximately evaluate a definite integral. While conventional numerical integration evaluates integrands at regularly spaced points in the integrand domain, Monte Carlo integration samples (evaluates) the integrand at multiple random points.

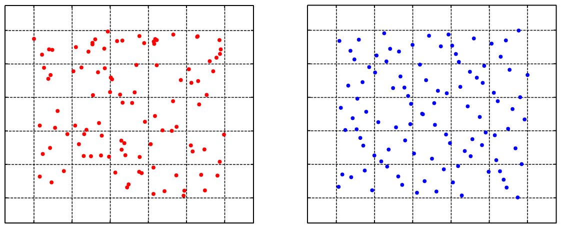
(a) Pseudo-random Points (b) Low-discrepancy Points

Figure 1. "Equidistribution" or Randomness

Monte Carlo integration is used extensively to compute multi-dimensional integrals that occur frequently in physics, finance, etc. since it can be as accurate as and much faster than conventional integration. The random inputs for MC integrations and MC simulations are usually uniform random variates (RVs) provided by a pseudo-random number generator (PRNG). MC simulations are often inherently parallel and, consequently, are increasingly implemented on reconfigurable hardware such as FPGAs.

Real-world MC integration uses a finite quantity of pseudo-random numbers. A critical issue with these numbers is that they may not be perfectly 'equidistributed,' i.e. this finite quantity of pseudo-random numbers actually used are not spread uniformly throughout the domain of the integrand, which leads to poorer results. While equidistribution as well as accuracy improve as more random numbers are used, this requires longer run-times. To solve this problem, *low-discrepancy* sequences that are well-distributed even for small quantities were introduced [1]. Low-discrepancy sequences (LDS) are also called *quasi-random numbers*; therefore, MC methods utilizing low-discrepancy sequences are referred to as *Quasi-Monte Carlo* (QMC) methods.

It is simplest to illustrate the distinction between pseudo-random numbers and low-discrepancy sequences with an example. Figure 1a shows 100 random points from the well-known *Mersenne Twister* PRNG.

Each square subregion does not contain a similar number of points and clumping can be seen. Figure 1b shows 100 points from the Sobol [2] low-discrepancy sequence. Each square subregion contains roughly the same number of points, making the points more even and resulting in a higher degree of equidistribution.

Quasi-Monte Carlo integration is common in software but, to our knowledge, no hardware implementations exist in the literature in spite of the recent trend of performing MC methods in hardware. We attribute this to a corresponding lack of hardware low-discrepancy sequence generators. In this paper, we remedy this issue by presenting the first FPGA-optimized, scalable designs for three common low-discrepancy sequences: Sobol [2], Niederreiter [3] and Halton [4].

We begin with mathematical preliminaries on low-discrepancy sequences, followed by algorithms for generating three low-discrepancy sequences in binary arithmetic. FPGA-optimizations and scalability are discussed next, followed by results of our implementations with various levels of parallelization on the *Xilinx* Virtex-4 FPGA. We conclude by interfacing a real-world Monte Carlo integration problem to our generators and analyzing performance.

2. Low-Discrepancy Sequences

2.1. Defining Discrepancy

Before discussing methods for generating low-discrepancy sequences, let us define *discrepancy* [5]. The numbers we generate belong to the half-open interval $[0, 1)$, and can be scaled if necessary. Consider the s-dimensional half-open unit cube $\mathbb{I}^s = [0, 1)^s$, $s \geq 1$. For N points $x_1, x_2, \dots, x_N \in \mathbb{I}^s$ and a sub-interval J of $\mathbb{I}^s$, if $A(J)$ counts the number of points $x_i \in J$ and $V(J)$ is the volume of J, we define the discrepancy $\mathcal{D}(J, N)$ as

$$\mathcal{D}(J, N) = \left| \frac{A(J)}{N} - V(J) \right|, \qquad (1)$$

Intuitively, the discrepancy is the difference between the proportion of points in J compared to the full unit cube $\mathbb{I}^s$ and the volume of the 'box' J compared to $\mathbb{I}^s$.

Figure 2 shows a 2-dimensional hypercube $\mathbb{I}^s = [0, 1)^2$ (a square) with six points. Three sub-intervals (or boxes) A, B and C are shaded. The discrepancy of box A, which contains 3 points, is calculated from eq. (1) as:

$$\mathcal{D}(A, 6) = \left| \frac{3}{6} - \frac{1}{2} \right| = 0$$

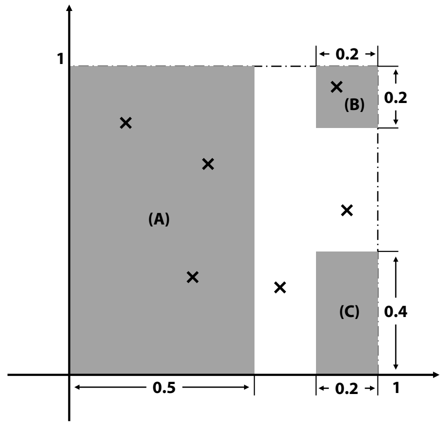

Figure 2. Illustrating discrepancy in 2 dimensions

Similarly, the discrepancies of boxes B and C are ≈ 0.1267 and 0.8 respectively.

The *worst-case* discrepancy, i.e. the worst-case distribution of a set/sequence of points $\{x_1, x_2, \dots, x_N\} \in \mathbb{I}^s$ is called the *star-discrepancy* [5] and is defined as:

$$\mathcal{D}^*(N) = \max_J |\mathcal{D}(J; N)|, \qquad (2)$$

The goal of a *low-discrepancy sequence* is to minimize this star discrepancy.

2.2. The Halton Sequence

Historically, low-discrepancy sequences were not designed with digital arithmetic in mind. The *Van der Corput* sequence [4] was the first such sequence; it takes the natural numbers $\{1, 2, \dots\}$ and reverses their representation in a particular base b. Figure 3 shows the first seven terms of the base-2 (binary) Van der Corput sequence; the fractional values are the reversed binary representations of the term's index in the progression.

The *Halton* sequence [4] generalizes the Van der Corput sequence to higher dimensions. Each dimension is represented in a different prime base b (e.g., $2, 3, 5, 7, \dots$). To generate the n-th point in a sequence, consider the base b-ary expansion of a n: $\sum_{i=0}^{\infty} a_i b^i$, where the b-ary coefficients $a_i \in \{0, \dots, b-1\}$. The n-th Halton point $H(n)$ is the *radical-inverse* function in base b, defined as $H(n) = \sum_{i=0}^{\infty} a_i b^{-i-1}$.

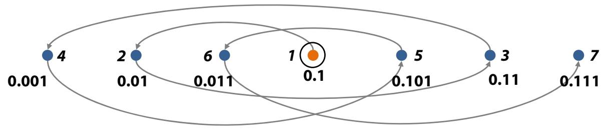

Figure 3. The base-2 *Van der Corput*/Halton Sequence (progression of the first 7 terms)

2.3. The Sobol Low-Discrepancy Sequence

Modern low-discrepancy sequences belong to the general class of *digital sequences* [6]. Essentially, digital sequences are constructed using binary operations on binary expansions and are therefore well-suited to efficient implementations on computers. Further discussion of the mathematical theory behind digital sequences is beyond our scope; the interested reader may consult [6] and [7].

The *Sobol* was the first digital sequence [2]. It operates in base-2 and is still well-regarded for use in quasi Monte-Carlo. We discuss the algorithm for generating a Sobol sequence based on [8]. To generate one sequence (i.e., one dimension) of N-bit low-discrepancy Sobol numbers, we choose odd integers m_i $(0 \leq i < N)$, and define N *direction vectors* c_i:

$$c_i = \frac{m_i}{2^i} = 0.c_{i1}c_{i2}c_{i3}\ldots, \tag{3}$$

where c_{ij} denote the binary expansion of c_i. Now, choose a primitive polynomial $P(x)$ of degree d with co-efficients a_i from the two-element finite (or Galois) field $GF(2)$ (i.e., binary):

$$P(x) = x^d + a_1 x^{d-1} + \cdots + a_{d-1} + 1 \tag{4}$$

These coefficients a_i are used to calculate each direction vector c_i as:

$$\begin{aligned} c_i = {} & a_1 c_{i-1} \oplus a_2 c_{i-2} \oplus a_1 c_{i-1} \oplus \cdots \\ & \cdots \oplus a_{d-1} c_{i-d+1} \oplus c_{i-d} \oplus [c_{i-d} \gg d], \end{aligned} \tag{5}$$

where $\oplus$ is an exclusive-or (XOR), and the last term is c_{i-d} right-shifted by d bits.

A one-dimensional N-bit wide low-discrepancy Sobol sequence $x_1, x_2, \ldots$ can be generated based on this set of direction vectors. Take the n-th term of this sequence, x_n, with $n = b_N b_{N-1} \ldots b_2 b_1$ in binary. Then,

$$x_n = b_1 c_1 \oplus b_2 c_2 \oplus \ldots \oplus b_{N-1} c_{N-1} \oplus b_N c_N \tag{6}$$

If the direction vectors c_i are pre-computed, generating one number requires at most N lookups and $N-1$ XORs. This effort can be drastically reduced by considering a *gray-coded* representation of n [9]—a gray-coded $n+1$ differs from gray-coded n in only one bit. The gray-code representation for n can be obtained by

$$g_N \ldots g_2 g_1 = b_N \ldots b_2 b_1 \oplus b_N \ldots b_3 b_2,$$

and the bit g_r that flips going from $n \to n+1$ is simply the position r of the least-significant zero-bit (LSZ) in $n = b_N \ldots b_1$.

Table 1. Example: Direction Vectors for a Sobol Seq.

i	1	2	3	4	5	6
m_i	1	3	7	5	7	43
v_i	0.1	0.11	0.111	0.0101	0.00111	0.101011

Table 2. Computing the Direction Vectors in Table 1

$$c_i = c_{i-2} \oplus c_{i-3} \oplus (c_{i-3} \gg 3) \quad (i > 3)$$

$$\begin{aligned} c_4 &= c_2 \oplus c_1 \oplus (c_1 \gg 3) \\ &= 0.11 \oplus 0.1 \oplus 0.0001 = 0.0101 \\ c_5 &= c_3 \oplus c_2 \oplus (c_2 \gg 3) \\ &= 0.111 \oplus 0.11 \oplus 0.00011 = 0.00111 \\ c_6 &= c_4 \oplus c_3 \oplus (c_3 \gg 3) \\ &= 0.0101 \oplus 0.111 \oplus 0.000111 = 0.101011 \end{aligned}$$

Now, since $n+1$ differs from n by only one bit, x_{n+1} 'differs' from x_n by only one direction vector c_r. x_{n+1} can therefore be computed based on x_n as

$$x_{n+1} = x_n \oplus c_r, \tag{7}$$

with only one lookup and one XOR; the complexity of finding the least-significant zero-bit r of n can also be decreased from the standard $O(\log n)$ in hardware by using a priority encoder.

2.4. Example: Generating a Sobol Sequence

We start with the degree $d = 3$ primitive polynomial $x^3 + x + 1$. The coefficients, comparing with (5), are $a_1 = 0$, $a_2 = 1$, $a_3 = 1$. The first $d = 3$ direction vectors are arbitrary, depending on choosing odd $m_i < 2^i$.

Let $m_1 = 1$, $m_2 = 3$, $m_3 = 7$, with the corresponding $c_i = m_i/2^i$, i.e. $c_1 = 0.1$, $c_2 = 0.11$, $c_3 = 0.111$. The recurrence for $c_4, c_5, \ldots$ is, from (5),

$$c_i = c_{i-2} \oplus c_{i-3} \oplus (c_{i-3} \gg 3) \quad (i > 3) \tag{8}$$

Once we have computed the direction vectors, the sequence can be generated. Setting the initial conditions for the zeroth term: $x_0 = 0$, $n = 0$, and the position of the least-significant-zero in $n = 0$ is $r = 1$. The first three numbers of this sequence are then calculated as shown in Table 3.

2.5. Niederreiter Sequences

Niederreiter sequences are digital sequences that can be thought of as generalizing the base-2 Sobol to other

Table 3. Generating the Sobol Sequence in Table 2

n	r	x_n
`001`	2	$x_1 = x_0 \oplus v_1 = 0.0 \oplus 0.1 = 0.1$
`010`	1	$x_2 = x_1 \oplus v_2 = 0.1 \oplus 0.11 = 0.01$
`011`	3	$x_3 = x_2 \oplus v_1 = 0.01 \oplus 0.1 = 0.11$

bases. Asymptotically (i,e., in the limit of an infinite number of points), Niederreiter sequences have the lowest star discrepancy among all other low-discrepancy sequences.

They are generated in exactly the same manner as Sobol (section 2.4), although the direction vectors are calculated differently. Just as with Sobol, in practice a multi-dimensional Niederreiter sequence consists of multiple base-2 generators with unique sets of direction vectors.

3. Generating Low-discrepancy Sequences in Hardware

For the Sobol/Niederreiter sequences (and for any digital sequence in general), we pre-compute the N direction vectors and store them in RAM. Generating each term of the actual sequence involves the least-significant zero (LSZ) calculation, a memory lookup and an XOR as in eq. (7). While this process can be made fairly efficient with techniques such as pipelining, we can exploit the LSZ, binary arithmetic, memory structures and even an application's requirements to parallelize and optimize the generation.

3.1. Computing and Exploiting the Least-Significant-Zero (LSZ) Position

From the gray-code recursion in eq. (7), finding the position r of least-significant zero bit (LSZ) in n is critical in choosing the direction vector to compute x_{n+1}. For generating N-bit numbers, finding the LSZ for an index n incurs $O(\log n)$ complexity with a sequential 'shift-and-count' algorithm. However, this is reduced to constant time through the flexibility of programmable logic: an inverted N-line-to-log(N)-line priority encoder. The least-significant zero bit in an input asserts priority over others and its decimal position is returned by the encoder. For the 32-bit numbers generated by our implementations, an input of `110100101`, for example, would produce an output of `2`.

Applications usually require multiple random inputs or multi-dimensional sequences. Since the LSZ for each set of terms is the same, one common LSZ-detector circuit can be used.

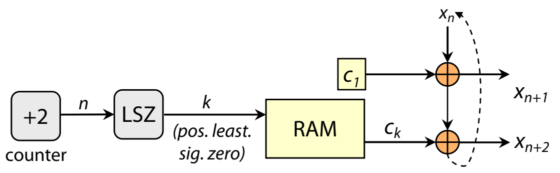

Figure 4. 2-way, 2×parallel generator

LSZ positions r for certain n can also be pre-coded; for $p \in \{1, 2, \ldots, \log_2 N\}$, and

$$n \bmod 2^p \equiv 2^{p-1} - 1, \tag{9}$$

the LSZ position $r \equiv p$ and the required direction vector c_p is a 'constant' that can be pre-coded instead of computing the LSZ and looking the c_r up. This works in practice because, for example, every even n has LSBs $\ldots$`xx0` and therefore LSZ $r = 1$, every fourth $n = 1, 5, \ldots$ has LSBs $\ldots$`xx01` and LSZ $r = 2$, etc. The sequence terms x_n for these n can be computed with their pre-coded direction vectors in parallel with the x_n's that require an LSZ and memory lookup for their direction vector. We call this scheme p-way parallelization. While the user can control the degree of parallelization by choosing the c_p's to pre-code, we note that asymptotically as p increases, the n satisfying (9) decrease geometrically, giving a maximum possible degree of parallelization of $2 + 1 + \sum_{p=1}^{N} \frac{1}{2^p} \approx 4$. Figure 4 shows a 2×parallel generator where the direction vector for every even term c_1 is pre-coded (i.e., $p = 1$).

3.2. Architectural Optimizations for Further Parallelization

Memory (RAM) in contemporary FPGAs, whether constructed from logic/LUTs ('distributed' RAM) or SRAM blocks (BRAM), has dual asynchronous I/O ports. In that case, throughput can be doubled by duplicating the LSZ circuit and computing future terms of the sequence. Since generation only requires reads, the direction vectors c_i can be duplicated across multiple dual-port RAMs for a higher effective number of ports; we call this o-port parallelization.

Figure 5 illustrates a 2-port, 2-way parallelization that generates 4 terms in one cycle. If the index n is generated by a count-by-4 counter, i.e. $n = 0, 4, \ldots$, the four terms are generated as:

$$\begin{aligned} x_{n+1} &= x_n \oplus c_1 \\ x_{n+2} &= x_{n+1} \oplus c_k \\ x_{n+3} &= x_{n+2} \oplus c_1 \\ x_{n+4} &= x_{n+3} \oplus c_l \end{aligned}$$

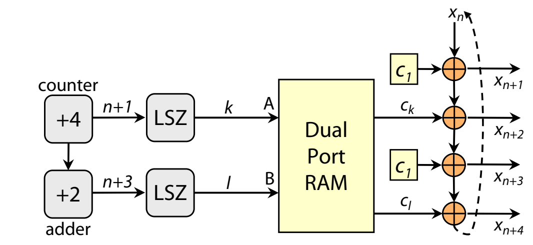

Figure 5. 2-port, 2-way 4×parallel Generator

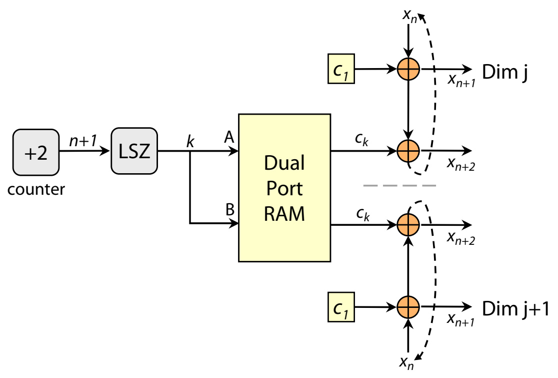

Figure 6. 2-dim., 2-way, 2×parallelized Generator

The optimized circuits outlined so far address the generation of one-dimensional sequences. To generate multiple dimensions, these circuits are replicated. Depending on the balance between throughput and area-efficiency that is desired, especially if physical block RAMs are being used to store the direction vectors instead of distributed RAM, the memories can contain two different sets of direction vectors and simultaneously generate two sequences because of the availability of the dual-ports. Figure 6 shows a 2-dimensional, 2-way parallelized generator.

3.3. Virtex-4 Implementations

We implemented a 2-D Halton generator (in bases 2 and 3) as well as 2×/4× parallelized 2-D and 6-D Sobol/Niederreiter generators on the Xilinx *Virtex 4-SX* FPGA (XC4VSX35), synthesized with Synplify Pro 9.1. 32-bit numbers were generated, with distributed RAM (LUTs) used as memory for the direction vectors ($32 \times 32 = 1024$-bits per dimension). The results, including resource usage, maximum frequency and throughput are shown in Table. 4; the throughput/area-efficiency increases from 2× to 4× parallelization.

Comparison with Pseudo-RNGs. Since no hardware LDS implementations exist, we compare relative complexities with the state-of-the-art parallelized pseudo-random number generators (*Mersenne Twister*) [10]; both the resource usage (area) and throughput of our low-discrepancy sequence generators are comparable to parallelized hardware PRNGs.

Table 4. Virtex-4 Implementation Results

	Sequence	Slices	Clock Freq (MHz)	Throughput (Gbits/s)
2-dimensional	Halton	360	250.1	8.0
	Sobol (2×)	147	349.0	22.3
	Sobol (4×)	194	297.9	38.1
6-dimensional	Sobol (2×)	275	341.0	21.8
	Sobol (4×)	258	261.6	33.5

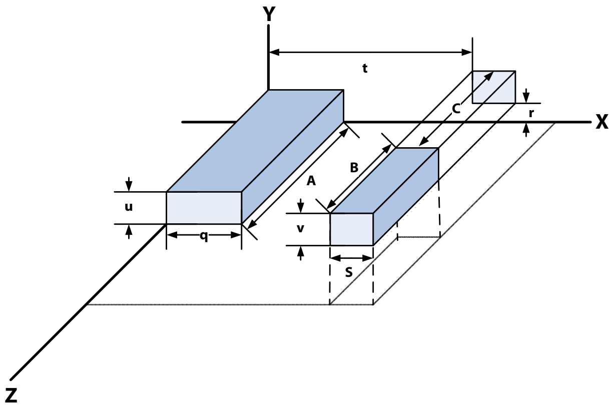

Figure 7. 3-D IC Inductance Problem (Quasi-MC)

4. Application: Quasi-Monte Carlo for 3-D IC Partial Inductance Extraction

Inductances between interconnects (traces) in integrated circuits (IC) are an important limiting factor as designers seek to scale these chips to ever-higher clock frequencies. Modeling these inductances is an indispensable part of high-frequency IC design. Analytically deriving the mutual inductance between a set of interconnects of arbitrary geometry involves a double-volume (i.e., 6-dimensional integral) and precise knowledge of the instantaneous current densities through each of the interconnects. Instead, the mutual inductance can be numerically approximated through Monte Carlo integration [11].

For example, consider deriving the mutual inductance M_6 for two 3-D interconnects placed parallel to each other, with dimensions and separations as shown in Figure 7 [11]. The analytical solution (M_6) for this problem is known, which allows us to benchmark Monte Carlo approximations. For $A = B = C = D = r = t = 5$ μm and $q = s = u = v = 1$ μm, $M_6 =$

Table 5. MC and QMC Computation of Mutual Inductance: Results and Resource Usage

Analytical	**Pseudorandom**	**Sobol**	**Niederreiter**
0.28800	0.28932	0.28800	0.28802

LDS-Slices	**CORDIC-Slices**	**DSP48s**	**Max. Freq.**
990	753	20	104.22 MHz

0.28800. If the points on the surface of each conductor are represented by the position vectors $\mathbf{r_i} = (x_i, y_i, z_i)$ and $\mathbf{r_j} = (x_j, y_j, z_j)$ respectively, the expression for the mutual inductance using η randomly sampled point pairs is

$$M_6 = \mu \cdot \frac{1}{N} \sum_{i,j=0}^{\eta} \frac{1}{|\mathbf{r_i} - \mathbf{r_j}|}, \tag{10}$$

where μ is a constant and we are effectively computing the inverse of the Euclidean distance (i.e., $1/\sqrt{(\mathbf{r_i} - \mathbf{r_j})^2}$) between each point pair.

We implemented the M_6 computation on the Virtex-4, using DSP slices for squaring the distances and a pipelined CORDIC for the square root. Quasi-Monte Carlo integration was performed using 1,000 low-discrepancy Sobol and Niederreiter points (16-bit fractions). Table 5 shows the M_6 values computed and compares them to the analytical results as well as results from a standard MC integration using 1,000 pseudo-random numbers; the Quasi-Monte Carlo results are over an order of magnitude more accurate than those from conventional Monte Carlo.

5. Conclusion

Low-discrepancy sequences are essential for Quasi-Monte Carlo (QMC) methods such as multi-dimensional integration; quasi-Monte Carlo delivers more accurate results in a shorter time than conventional Monte Carlo. While MC simulations are increasingly parallelized and implemented on reconfigurable hardware, this is not true for QMC due to the lack of hardware low-discrepancy sequence generators.

We present the first (to our knowledge) designs for low-discrepancy sequence generation in hardware. Our techniques for optimization and parallelization can be applied to the entire class of digital low-discrepancy sequences; we implemented three specific sequences (Halton, Sobol and Niederreiter) with varying degrees of parallelization on the Virtex-4 FPGA. We also demonstrated the supremacy of QMC over MC with a real-world example involving the extraction of mutual inductances in integrated circuit interconnects. Future work includes an automated GUI-based program that creates synthesizable VHDL for a generator given the parameters for a specific digital sequence.

Acknowledgments

The authors are grateful to Prof. Om Agrawal for his advice, and the Cooper Union Mathematics Department as well as Prof. Fred Fontaine and Center for Signal Processing, Communications and Computer Engineering (*S*ProCom*^{2}) for supporting this research.

References

[1] S. Zaremba, "The mathematical basis of monte carlo and quasi-monte carlo methods," *SIAM Review*, vol. 10, no. 3, pp. 303–314, Jul 1968.

[2] I. M. Sobol, "Uniformly distributed sequences with an additional uniform property," *USSR Comput. Math. Math. Phys.*, vol. 16, pp. 236–242, 1976.

[3] H. Niederreiter, "Point sets and sequences with small discrepancy," *Monatshefte für Mathematik*, vol. 104, no. 4, Dec.

[4] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, no. 1, Dec.

[5] J. Matoušek, "On the l2-discrepancy for anchored boxes," *J. Complex.*, vol. 14, no. 4, pp. 527–556, 1998.

[6] P. L'Ecuyer and C. Lemieux, "Recent advances in randomized quasi-monte carlo methods," in *Modeling Uncertainty*, M. Dror *et al.*, Eds., 2002, pp. 419–474.

[7] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, ser. SIAM CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia: SIAM, 1992, vol. 63.

[8] P. Bratley and B. L. Fox, "Algorithm 659: implementing sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 14, no. 1, pp. 88–100, 1988.

[9] I. A. Antonov and V. Saleev, "An economic method of computing lp$_\tau$-sequences," *USSR Comput. Math. Math. Phys.*, vol. 19, pp. 252–256, 1979.

[10] I. L. Dalal and D. Stefan, "A hardware framework for the fast generation of multiple long-period random number streams," in *Proc. 16th Intl. ACM/SIGDA Symp. FPGAs*. New York, NY, USA: ACM, 2008, pp. 245–254.

[11] K. Chatterjee, "A stochastic algorithm for the extraction of partial inductances in ic interconnect structures," *Applied Computational Electromagnetics Society Journal*, vol. 21, no. 1, pp. 81–89, March 2006.