# High performance FPGA-oriented Mersenne Twister Uniform Random Number Generator

**Pedro Echeverría · Marisa López-Vallejo**

**Abstract** Mersenne Twister uniform random number generators are key cores for hardware acceleration of Monte Carlo simulations. In this work, two different architectures are studied: besides the classical table-based architecture, a new architecture based on a circular buffer and especially targeting FPGAs is proposed. A 30% performance improvement has been obtained when compared to the fastest previous work. The applicability of the proposed MT architectures have been proven in a high performance Gaussian RNG.

## 1 Introduction

The outstanding integration densities of current technologies allows configurable logic to be used as powerful platforms for hardware acceleration. This is the case of numerous Monte Carlo simulations where the models of the system under study can imply great complexity. Parallelism is intrinsic to Monte Carlo as it is based on the replication of the same model with different underlying variables, the random numbers, and making them ideal for parallel architectures.

In this context, uniform random number generators (URNG) play a key role when developing hardware accelerators for Monte Carlo simulations. URNGs are the base element for any random number generation, as it is the case of Gaussian RNGs (GRNGs) which are widely used in this type of simulations. Any URNG used as basic core in a hardware accelerator must provide random samples with a throughput that does not limit the frequency of the whole Monte Carlo simulation. Furthermore, it should require as less resources as possible because resources may be required by the model to simulate or for replicating the whole Monte Carlo system. Finally, it should provide high-quality random samples as the quality of the random numbers directly impacts on the result accuracy.

Previous work on URNGs using FPGAs can be split into two fields, the development of specific URNGs for FPGAs, and the adaptation of software URNGs to FPGAs. In the first field we can outstand several works from D. B. Thomas *et al* as [1,2], where different URNGs are proposed, studied and developed featuring the specific resources and architecture of FPGAs.

Here we concentrate on the second field, the adaptation of software URNGs, due to practical issues like compatibility and application debugging. When developing a hardware accelerator a desired feature is the complete compatibility between the original software application and the accelerated one. Furthermore, the complete compatibility easies the hardware application debugging. Compatibility implies obtaining the same sequence of random numbers. In this work we focus on a very well known generator, the Mersenne Twister [3] which is broadly used in computational science based on Monte Carlo simulations due to its far above the ground quality, superb period and high performance [3].

Previous works in this field [4–6] are based on the most used Mersenne Twister configuration for 32 bits samples, the MT19937, due to its high quality and the simplifications introduced by its set of parameters. However, none of these previous implementations fulfills the desired characteristics. [4] and [5] present a slow clock rate, while in [6] the first part of the algorithm, the initialization, is not implemented in hardware, being thus an incomplete generator. It is an important issue because if the initialization phase has to be carried out in software it complicates the interface with the hardware

Department of Electronic Engineering. Universidad Politécnica de Madrid, UPM (Spain).
E-mail: {petxebe, marisa}@die.upm.es

accelerator (the whole initialization table is transferred to hardware instead of just the seed).

In this work we present two complete MT URNGs based on the same set of parameters, MT19937, but now designed to fulfill the above mentioned features:

1. All in hardware.
2. Able of generating one sample per cycle.
3. Highly efficient in area and performance.

The proposed architectures take advantage of the FPGA structure and resources. Following this idea, in addition to the classical memory table implementation a new architecture is proposed, specifically designed to avoid the use of FPGA internal RAMs as these resources become essential for some simulation models.

Finally, the proposed MT architectures have been used in a high performance GRNG to validate their applicability and study the area cost they imply.

The paper structure is as follows: Section 2 briefly summarizes the Mersenne Twister algorithm. In section 3 the proposed hardware architectures are exposed, while their experimental results and integration within a GRNG are analyzed in section 4. Finally some conclusions are drawn.

## 2 Mersenne Twister Algorithm

The MT generator [3] is a general and highly parameterizable URNG based on a linear recurrence between vectors of $w$ bits. The algorithm is split into three different tasks:

1. Initialization: generation of the first $n$ vectors of the recurrence from a seed.
2. Obtaining the linear recurrence.
3. The *tempering* of the generated variables from the linear recurrence.

First, an initialization from the seed is needed as the linear recurrence requires a work area of $n$ variables of $w$ bits. This initialization takes the seed as the first element of the recurrence, $x_0$, while the other $n-1$ variables are generated following:

$$x_i = 1812433253 \times (x_{i-1} \oplus (x_{i-1} >> 30)) + i$$

Once the first work area is obtained with the initial $n$ variables, the random numbers are calculated following equation:

$$x_{k+n} = x_{k+m} \oplus (x_k^u | x_{k+1}^l)A \quad (1)$$

where $A$ is a matrix while $x_k^u$ stands for the $(w-r)$ most significant bits of $x_k$ and $x_{k+1}^l$ means the $r$ less significant bits of $x_{k+1}$.

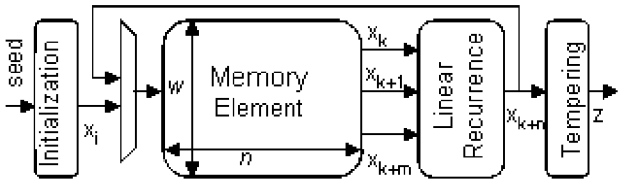



**Fig. 1** Mersenne Twister general Architecture

Finally, to improve the statistical properties of the generator the numbers generated in the recurrence are modified, *tempered*, with a bitwise multiplication by a $w \times w$ binary matrix ($T$):

$$z = x_{k+n}T \quad (2)$$

being $z$ the output of the generator.

### 2.1 MT19937 Algorithm Simplifications

The complexity of the algorithm and its computational charge is mainly due to the two matrix multiplications in equations 1 and 2. However, both multiplications are greatly simplified with a correct selection of the elements of the matrixes. This is what happens with the MT19937 set of parameters.

For the linear recurrence multiplication, when the matrix $A$ follows the form:

$$A = \begin{pmatrix} 0, & I_{w_1} \\ a_{w_1}, & (a_{w_2}, ..., a_0) \end{pmatrix}$$

its multiplication $(x_k^u | x_{k+1}^l)A$ is reduced to:

$$(x_k^u | x_{k+1}^l) >> 1 \;\; when \;\; x_{k+1}(0) = 0$$

$$((x_k^u | x_{k+1}^l) >> 1) \oplus a \;\; when \;\; x_{k+1}(0) = 1$$

where $a$ is the $w^{th}$ row of matrix A.

For the *tempering* matrix multiplication, again the matrix T is selected in such a way that this multiplication is simplified into several logical bitwise operations:

$$\begin{aligned} y &= x_{k+n} \oplus (x_{k+n} >> u) \\ y_1 &= y \oplus ((y << s) \;\&\&\; b) \\ y_2 &= y_1 \oplus ((y << t) \;\&\&\; c) \\ z &= \quad y_2 \oplus (y >> l) \end{aligned}$$

As just seen, MT URNG depends on multiple parameters (w, n, m, r, a, u, s, b, t, c, l) corresponding MT19937 to the set (32, 624, 397, 31, 9908BODF, 11, 7, 9D2C5680, 15, EFC60000, 18).

**Table 1** Hardware Mersenne Twister Results.

| | [4] | [5] | [6] | This Work | | | |
|---|---|---|---|---|---|---|---|
| | | | | CB | | 3P_Table | |
| **FPGA** | N/A | Virtex-E | V4Fx100 | V4Fx100 | V5Fx200 | V4Fx100 | V5Fx200 |
| **MHZ** | 38.4 | 24.2 | 265 | 339.3 | 418.6 | 345.9 | 417.3 |
| **DSPs** | N/A | N/A | 0 | 3 | 3 | 3 | 3 |
| **B-RAMs** | N/A | N/A | 4 | 0 | 0 | 4 | 2 |
| **Slices** | 420 | 330 | 128 | 810 | 272 | 183 | 96 |

## 3 Hardware Architecture

Figure 1 shows the general hardware architecture of the MT URNG. In addition to the logic devoted to the previously described tasks, it requires a memory element for storing the samples composing the $n \times w$ work area of the linear recurrence. Not depicted in the figure, a small control logic is also needed to handle the two possible scenarios, the initialization and the linear recurrence with the tempering.

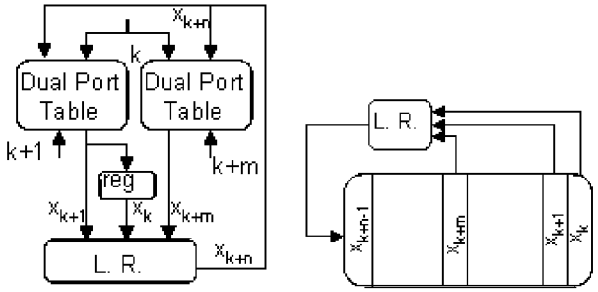
The hardware must be able of generating one sample per cycle at a high clock rate. If we focus on the linear recurrence and the tempering both tasks can easily fulfill the criteria of obtaining one sample per cycle while achieving a high clock rate. The matrix multiplications reduction to bitwise operations ensures fast datapaths as these operations perfectly suit FPGA technology. Furthermore, due to the depth of the work area and the dependencies among samples, the logic of both tasks can be pipelined to increase the clock rate.

The initialization task, besides its bitwise operations, also requires a 32 bit multiplication and a 32 bit addition. These two operations compose a slow datapath with much more logic than the other two tasks. Although initialization stops working once the first $n$ samples are generated, the clock rate of the MT generator is also determined by this logic and therefore, pipelining this task is a must.

The other key element in the hardware architecture is the storage element needed for the $n \times w$ work area of the linear recurrence. There are two suitable options: a storage table, figure 2(a), which is the solution adopted in previous works, and a circular buffer, the new solution proposed in this work, see figure 2(b).

In the first case, a three port storage table with two read and one write ports is needed (3P_Table from now on). In an FPGA, this three port table has a direct translation into two dual port tables implemented by embedded Block-RAMs plus the logic required for updating the indexes for the table addresses.

A second option is the use of a circular buffer (CB) of registers taking advantage of the fixed relationship between the indexes of the words and considering that each step of the recurrence $x_k$ is replaced by $x_{k+n}$ in the work area. This way, the linear recurrence (L. R. in figure 2(b)) and the buffer of registers can be considered as a circular buffer where the linear recurrence is carried out by some combinational logic between the input and the output of the buffer. Hence the architecture is simplified as no logic for the table indexes is needed.




(a) 3 port table. (b) Circular Buffer MT.

**Fig. 2** MT architectures.

## 4 Implementation Results

Table 1 summarizes the results for the implemented MT and two Xilinx FPGAs (Virtex 4-Fx100 and Virtex 5-Fx200) as well as the results published in previous works. The results for our work are Post Place & Route using ISE 11 environment.

Outstanding maximum speeds have been achieved, 418.6 MHz for Virtex 5, and 345.9 for Virtex 4. More in detail, and for the same FPGA (Virtex 4 Fx100), the 3P_Table architecture outperforms the fastest previous work clock rate [6] in a 30.5%, even though that work did not implement the initialization task. It can be also seen that this performance does not come at the cost of resources. The increase of slices (183 to 128) and DSPs (3 to 0) are mainly due to the initialization stage and not to the performance improvement.

### 4.1 Architectures Comparison

The main reason to consider to select between storage table or circular buffer of registers is the resource usage, as the Block RAMs required in the 3P_Table archi-

**Table 2** Virtex 5 FX200 - GRNG Results

| | Slices | BRAM | DSP | MHz |
|---|---|---|---|---|
| $CDF^{-1}$ | 946 (3.1%) | 5 (0.5%) | 10 (2.6%) | 280.5 |
| **Taus88** | 1024 | 5 | 10 | 280.9 |
| **MT 3P_T** | 966 | 7 | 13 | 281.1 |
| **MT CB** | 1125 | 5 | 13 | 281.4 |

tecture become logic slices for the CB implementation. However, this effect is highly related to the FPGA family and the model selected, as will be analyzed next.

Regarding Virtex 4 devices, four BRAMs are required (1.06% of the total BRAMs), whose replacement increases in 682 the required slices (1.61% of the total). However, for Virtex 5, on the one hand the CB architecture benefits from the fact that the number of LUT inputs increases to six with respect to the four inputs LUTs in Virtex 4. Thereby, the implementation of the work area in logic is drastically more compact in Virtex 5. On the other hand, 3P_Table implementation also benefits from the increase of Block RAM capacity, now requiring just two BRAMs, representing a 0.44% of the total BRAMs. In this case the increase of slices (87) just represents a 0.28% of the total. This slice increase represents just a 0.28% of the total slices, being this percentage smaller than the 0.44% of the total BRAMs represented by the two BRAMs.

### 4.2 Using an MT URNG in a Gaussian RNG

As URNGs are base elements for obtaining generators from other distributions, it is desirable to study the impact of the designed MT URNGs when used on other hardware generators providing commonly used distributions in Monte Carlo simulations. This is the case of the Gaussian RNGs (GRNG), so we have included our MT URNGs in a GRNG [7] that previously used a Tausworthe combined generator (Taus88) [8]. The Taus88 URNG presents high performance and very low use of resources, although its quality is not very good [1].

The gaussian generation method of [7] is the inversion method. This method ensures that the transformed distribution, the gaussian, inherits the statistical properties of the base RNG, the uniform, and therefore it is necessary a high quality URNG to achieve a high quality GRNG.

Table 2 summarizes the results obtained in terms of slices, BRAM DSPs and clock frequency. To better analyze the impact of using the MT URNG, the first row presents the requirements of the inversion ($CDF^{-1}$) method without URNG. Next rows collect information for the whole GRNGs including the three different URNGs. As seen in the table, the impact of using a much better URNG (MT) in the GRNG is not very significant taking into account that most GRNG logic is devoted to the inverse CDF function, which requires a small percentage of the resources of the FPGA. Furthermore, the GRNG working frequency is not limited by the URNG.

## 5 Conclusions

Due to its combination of features (high quality, superb period and high performance) the Mersenne Twister URNG is an ideal core for Monte Carlo simulation.

This work provides two efficient implementations of the MT URNG specifically designed for FPGAs differing both implementations in the storage element selected for the linear recurrence work area. With a careful design of the logic, and a pipelined implementation of the initialization task, our architectures present a performance improvements of a 30% with respect to the previous fastest work. The proposed MT architectures have been used in a high performance GRNG to prove their applicability.


## Acknowledgements

This work has been funded by BBVA contract P060920579 and Cicyt project TEC2009-08589.


## References


1. D. B. Thomas and W. Luk, "High quality uniform random number generation using lut optimised state-transition matrices," *Journal of VLSI Signal Proccessing*, vol. 47, pp. 77–92, 2007.
2. ——, ""fpga-optimised uniform random number generator using luts and shift registers"," in *Intl. Conf. on Field Programmable Logic and Applications*, 2010, pp. 77–82.
3. M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.
4. V. Sriram and D. Kearney, "An area time efficient field programmable mersenne twister uniform random generator," in *Intl. Conference on Engineering of Reconfigurable System and Algorithms*, 2006, pp. 244–246.
5. S. Chandrasekaran and A. Amira, "High performance FPGA implementation of the mersenne twister," in *International Symposium on Electronic Design, Test & Applications*, 2008, pp. 482–485.
6. T. Xiang and K. Benkrid, "Mersenne twister random number generation on FPGA, CPU and GPU," in *NASA/ESA Conference on Adaptative Hardware and Systems*, 2009, pp. 460–463.
7. P. Echeverría and M. López-Vallejo, "FPGA gaussian random number generator based on quintic hermite interpolation inversion," in *IEEE International Midwest Symposium on Circuits and Systems*, 2007, pp. 871–974.
8. P. L'Ecuyer, "Maximally equidistributed combined tausworthe generators," *Mathematics of Computation*, vol. 65, no. 213, pp. 203–213, January 1996.