High Performance FPGA implementation of the Mersenne Twister

Shrutisagar Chandrasekaran and Abbes Amira *
Electronic and Computer Engineering
Brunel University, West London, UK
* abbes.amira@brunel.ac.uk

Abstract

Efficient generation of random and pseudorandom sequences is of great importance to a number of applications [4]. In this paper, an efficient implementation of the Mersenne Twister is presented. The proposed architecture has the smallest footprint of all published architectures to date and occupies only 330 FPGA slices. Partial pipelining and sub-expression simplification has been used to improve throughput per clock cycle. The proposed architecture is implemented on an RC1000 FPGA Development platform equipped with a Xilinx XCV2000E FPGA, and can generate 20 million 32 bit random numbers per second at a clock rate of 24.234 MHz. A through performance analysis has been performed, and it is observed that the proposed architecture clearly outperforms other existing implementations in key comparable performance metrics.

1 Introduction

Monte Carlo (MC) methods [8] are very important in a number of application areas including aerospace engineering, bioinfomatics, molecular modelling and financial engineering. These methods are useful for obtaining numerical solutions to problems which are too complicated to solve analytically. Monte Carlo programs, due to their nature of consuming vast computing resources, have historically had to be executed upon the fastest computers available at the time, and employ the most advanced algorithms, implemented with effort invested in algorithmic optimisations and program efficiency. Large-scale Monte Carlo simulation typically require millions or even billions of random numbers for accurate simulations. The most expensive function in a typical Monte Carlo simulation is the Pseudo-Random Number Generator (PRNG). For example, in a typical options pricing scenario using Black Scholes in the domain of financial engineering takes up about 40% of the resources [5]. Hence it is essential that the PRNG be of long period, have good statistical properties and be vectorisable.

The nature of MC simulations also place upon us mathematical restraints upon the sequence of the random numbers used for simulation. Key among these include: uncorrelated sequences, long period, uniformity and most importantly efficiency. The Mersenne twister is a is a twisted generalised feedback shift register pseudorandom number generator that address all these requirements. It is based on a matrix linear recurrence over a finite binary field F_2 and provides for fast generation of very high quality pseudorandom numbers. The Mersenne Twister [6] is called so because it's maximum period is related to the Mersenne Prime sequence. Specifically, the MT19937 PRNG has the same period as the 24^{th} Mersenne Prime (i.e. $2^{19973} - 1$). However, despite its algorithmic simplicity in comparison to other PRNGs, the sheer number of random numbers required for MC methods necessitate the use of efficient methods of generating random vectors.

Reconfigurable hardware in the form of FPGAs is an extremely powerful implementation approach for several reasons. First and foremost, it allows for truly parallel computations to take place in a circuit. With soft cores, dedicated logic, block multipliers and specialised versions available in modern FPGAs, they are being increasingly deployed in computationally intensive application areas involving the processing of massive volumes of data. The regular nature of the complex computations performed repeatedly within some application areas are well suited to a hardware based implementation using FPGAs. It is the aim of this paper to develop high throughput area efficient FPGA implementation of the MT19937 PRNG.

The composition of the rest of the paper is as follows. Related work is described in Section II. Mathematical and algorithmic details of the MT19937 are provided in Section III. A description of the proposed architecture is presented in Section IV. Implementation results and comparison with existing architectures and software implementations of the MT19937 are presented in section V. Concluding remarks are given in section VI.

0-7695-3110-5/08 $25.00 © 2008 IEEE
DOI 10.1109/DELTA.2008.113

2 Related Work

The MT was first published in [6]. MT is a variant of TGFSR modifies in order to allow a Mersenne prime period. The characteristic polynomial has many terms, and has good distribution upto v bits of accuracy for $1 \leq v \leq 32$. The need for high performance implementations of the MT has been well recognised and a attempts at speeding up the MT in software have been made [3]. A limited number of hardware implementations of the MT have been made. In [9] an FPGA based implementation of the MT19937 has been presented using a three staged architecture. The three stages are seed generator, seed value modulator and output generation. However, there are no additional architectural details available. Additionally, the hardware platform used has not been specified; which renders it impossible to make full comparison of all performance metrics. In [7] a Mersenne Twister random number generator has been implemented using a Xilinx Virtex-5 LX50 device. This random number generator, described using Impulse C, uses ten parallel C-language processes that act as independent Mersenne Twister generators. After the initial testing had been performed, a series of iterative optimizations were performed, including an analysis of the generated pipeline structures. Recoding of the C-language statements to achieve a faster clock rate and more efficient hardware.

3 Mathematical and Algorithmic Background

In this section, the underpinnings of the MT19937 algorithm are described in mathematical terms, and a suitable algorithmic presentation in the form of a pseudocode.

3.1 Mathematical Basis of the MT19937

The primitivity test and k-distribution test that are needed in the parameter search for the MT19937 for a word x with bit width of w is expressed as the recurrence relation which is described as follows:

$$x_{k+n} = x_{k+m} \oplus (x_k^u | x_{k+1}^l)A \quad k = 0, 1, 2 \qquad (1)$$

where:
n is the degree of recurrence
m is the middle word, or the number of parallel sequences, $1 \leq m \leq n$
u, l are Mersenne Twister tempering bit shifts
and A is the twist transformation which is defined in rational normal form:

$$A = \begin{pmatrix} 0 & I_{n-1} \\ a_n & (a_{n-1,\ldots,a_0}) \end{pmatrix} \qquad (2)$$

where:
a is a coefficient of the rational normal form twist matrix; and

where I_{n-1} as the $(n-1) \times (n-1)$ identity matrix where the bitwise XOR ($\oplus$) operation replaces addition in the sum of products while implementing matrix multiplication. The rational normal form can be expressed in a more mathematically compact form as follows:

$$xA = \begin{cases} x >> 1; x_0 = 0 \\ (x >> 1) \oplus a; x_0 = 1 \end{cases} \qquad (3)$$

where
$x = \left(x_k^u | x_{k+1}^l\right) \quad k = 0, 1, ..$

The maximum theoretical period of $2^{nw-r} - 1$ can be achieved when $\Phi_B(t)$ is a primitive polynomial and is a characteristic polynomial of:

$$B = \left(\begin{array}{ccccc} 0 & I_w & \cdots & 0 & 0 \\ \vdots & & & & \\ I_w & \vdots & \ddots & \vdots & \vdots \\ \vdots & & & & \\ 0 & 0 & \cdots & I_w & 0 \\ 0 & 0 & \cdots & 0 & I_{w-1} \\ S & 0 & \cdots & 0 & 0 \end{array}\right) \qquad (4)$$

and

$$S = \begin{pmatrix} 0 & I_r \\ I_{w-r} & 0 \end{pmatrix} \qquad (5)$$

where:
r is the separation point of one word, or the number of bits of the lower bitmask, $0 \leq r \leq w - 1$

The MT achieves equidistribution in n dimensions. In order to compensate for reduced dimensionality of equidistribution, tempering is performed as follows:

$$\begin{aligned} y &= x \oplus (x >> u) \\ y &= y \oplus ((y << s)\&b) \\ y &= y \oplus ((y << t)\&c) \\ z &= y \oplus (y >> l) \end{aligned} \qquad (6)$$

where:
b, c are tempering bitmasks
s, t are tempering bit shifts

The corresponding coefficients of MT19937 in line with the above and satisfying the restriction that $2^{nw-r} - 1$ is a Mersenne prime are as follows:
$(w, n, m, r) = (32, 624, 397, 31)$
$a = 9908B0DF16$
$u = 11$
$(s, b) = (7, 9D2C568016)$
$(t, c) = (15, EFC600016)$
$l = 18$

3.2 MT19937 Pseudocode

The pseudocode of the MT19937 PRNG suitable for software implementation is presented in Codeblock 1. An implementation of this pseudocode using a .NET development platform on a PC equipped with a Core Duo processor and 1GB of system memory is also carried out to ensure algorithmic validity and to provide a point of reference with respect to the hardware implementation.

Algorithm 1 Pseudocode for the MT19937 algorithm

declare 32 bit integer $MT[624]$
declare 32 bit variable y
//The following function initialises the generator given a 32 bit seed
function initialiseGenerator (32 bit integer $seed$)
$MT[0] = seed$
for $i = 1 : 623$ **do**
  $MT[i]$=last_32bits_of(1812433253$\times$($MT[i-1]\oplus$ right_shift_by_30_bits($MT[i-1]$)) + i)
end for

// The following function generates an array of 624 untempered numbers
function generateNumbers()
for $i = 0 : 623$ **do**
  $y = 32^{nd}$_bit_of($MT[i]$) + last_31_bits_of($MT[(i+1)\%624]$)
  if y_is_even **then**
    $MT[i] = MT[(i+397)\%624]\oplus$ (right_shift_by_1_bit(y))
  else
    $MT[i] = MT[(i+397)\%624]\oplus$ (right_shift_by_1_bit(y))$\oplus$(2567483615)
  end if
end for

// The following function extracts a tempered pseudorandom number based on the i^{th} value
function extractNumber(integer i)
$y = MT[i]$
$y = y\oplus$(right_shift_by_11_bits(y))
$y = y\oplus$(left_shift_by_7_bits(y)&(2636928640))
$y = y\oplus$(left_shift_by_15_bits(y)&(4022730752))
$y = y\oplus$(right_shift_by_18_bits(y))
return y

4 Proposed Architecture for the MT19937 - Design and Evaluation

In this section, the high throughput architecture designed for the implementation of MT19937 is described.

4.1 Architecture Description

The three key functional blocks in the MT19937 architecture are the Initialise Generator (IG), Generate Numbers (GN) and Extract Number (EN). All three blocks are executed in parallel. To enable parallelisation without increasing hardware overhead, multiported block RAMs are used rather than arrays built from distributed logic. Additionally, the EN sub-block of the design is fully pipelined, allowing the combinational logic within each iteration of all three sub-blocks to execute concurrently. Common subexpression elimination in the design and the intelligent use of logic in place of the more expensive modulo operations enables us to further minimise the hardware complexity of the design.

The overall functional diagram of the proposed MT19937 architecture is as shown in Figure 1.

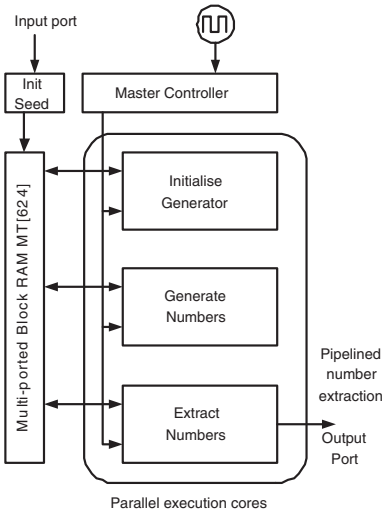

Figure 1. Overall hardware architecture for MT19937

5 Implementation Details

5.1 Design Methodology and Hardware Platform

In order to verify the performance of the proposed architecture for MT19937, the design has beenperformed using Handel C and prototyped on the Celoxica RC1000 board containing the Xilinx XCV2000E FPGA. Available on chip

logic resource include: 19200 Slices, 80 x 120 CLB Array, 655,360 bits Block RAM (BRAM) and 614,400 bits distributed RAM [2]. The RC1000 also has 4 memory banks which communicate with the host by means of DMA transfers [1].

5.2 Performance Metrics

The proposed architecture operates at a maximum frequency of 22MHz. The design is the most compact FPGA implementation to date and occupies a minimal core footprint. Full details of all relevant performance metrics can be obtained form Table 1.

Table 1. Key performance metrics of the proposed architecture for MT19937

Key Metric	Performance
Maximum Frequency	24.234MHz
Number of Slices	330
Total LUTs used	539
Number of RAMB16s	2
Throughput	22M numbers/second

5.3 Comparison with Existing Architectures

Design parameters such as Time Complexity (TC), Area Complexity (AC) and I/O type of the proposed design with existing architectures are presented in Table 2.

From the table, it can be clearly seen that the proposed architecture is much smaller than other existing implementations. Additionally, this compactness does not come at a performance cost. Through intelligent use of FPGA resources and design principles, high performance has been ensured. Since it is difficult to make meaningful direct comparisons of different platforms for throughput due to different device architectures and generational gaps, a term Throughput/MHz term has been introduced in order to normalise the amount of useful work done per system clock. It is clear from the table that the proposed architecture is the most efficient one in this regard. Additionally, it is worth mentioning that although the architecture in [7] has been implemented on the Virtex-5 which is four generations ahead of the Virtex-E FPGA that has been used in this paper, the work in [7] achieves only a 52% speedup which is primarily due to the increased clock speed achieved by the Virtex-5 due to architectural improvements. However, the inefficiency of this architecture is clearly demonstrated in the last row of Table 2.

6 Conclusion

In this paper a high performance FPGA implementation of the MT19937 algorithm has been implemented. Block level parallelism and word level pipelining have been exploited to improve the performance. The proposed architecture has the smallest core footprint of all architectures in print and provides the maximum amount of performance per MHz of the platform used. This architecture is fully modularised and can be directly used as a PRNG for Monte Carlo methods.

Table 2. Comparison of Design Parameters with Existing Architectures

Design	Proposed	[9]	[7]	Reference	[3]
Platform	Virtex-E	N/A	Virtex-5	S/W	S/W
Area (slices)	**330**	420	2028*	N/A	N/A
Freq. (MHz)	24.234	N/A	111	2×1833	N/A
Thru'put (T)	24.16	38.41	37	5.26	10.75
T/MHz	**0.998**	N/A	0.333	0.14	N/A

* The actual number of slices occupied on the Virtex-5 is 1600. However, the Virtex-5 has a 5 LUT per slice architectures unlike all previous generations of XIlinx FPGAs. Hence, a conversion factor is introduced to make the area occupied comparable. Throughput (T) is in Million numbers/second. T/MHz represents the Throughput per million clock cycles.

References

[1] RC1000 development platform product brief. Datasheet v1.1, Celoxica Ltd., August 2002.

[2] Virtex-E 1.8 V Field Programmable Gate Arrays. Datasheet DS022-1 (v2.3), Xilinx Inc., July 2002.

[3] N. Karaoglu. Mersenne twister - a study on random number generators.

[4] D. E. Knuth. *Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading PA, 2 edition, 1981.

[5] M. Mascagni, S. Cuccaro, D. Pryor, and M. Robinson. A fast, high quality, reproducible, parallel, lagged-fibonacci pseudo-random number generator. Technical Report SRC-TR-94-115, Supercomputing Research Center, 17100 Science Drive, Bowie, MD 20715, 1994.

[6] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3 – 30, January 1998.

[7] D. Pellerin, E. Trexel, and M. Xu. Fpga-based hardware acceleration of c/c++ based applications. Impulse Accelerated Technologies, August 2007.

[8] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, 1 edition, 1999.

[9] V. Sriram and D. Kearney. An area time efficient field programmable mersenne twister uniform random number generator. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, USA, June 2006. CSREA Press.