LETTER

Design and Evaluation of Hardware Pseudo-Random Number Generator MT19937[**]

Shiro KONUMA[†*], *Student Member and* Shuichi ICHIKAWA[†,††a)], *Member*

SUMMARY MT19937 is a kind of Mersenne Twister, which is a pseudo-random number generator. This study presents new designs for a MT19937 circuit suitable for custom computing machinery for high-performance scientific simulations. Our designs can generate multiple random numbers per cycle (multi-port design). The estimated throughput of a 52-port design was 262 Gbps, which is 115 times higher than the software on a Pentium 4 (2.53 GHz) processor. Multi-port designs were proven to be more cost-effective than using multiple single-port designs. The initialization circuit can be included without performance loss in exchange for a slight increase of logic scale.
key words: *custom circuit, simulation, random number, Mersenne Twister, FPGA*

1. Introduction

Random numbers are frequently used in scientific applications, particularly for simulations. One of the typical examples is the Monte Carlo method in computational physics, and it is well known that the result of a simulation is seriously affected by the quality of the random number generator. Pseudo-random number generators (PRNG) are usually adopted in such simulations for performance and reproducibility, but the quality of existing PRNG (e.g., linear congruent algorithms) is not sufficient for precise simulations.

The Mersenne Twister (MT) [1] is a PRNG that was proposed by Matsumoto and Nishimura in 1997. MT is highly suited to simulation because of its various advantages: mathematical background, uniformity, long period, multi-dimensional equidistribution, and high performance.

Recently, custom computing machinery is expected to accelerate simulations. In such machinery, PRNG also have to be implemented with logic circuits. The purpose of this study is to evaluate various designs of MT19937, a type of MT. MT19937 is well suited for scientific simulations for its long period ($2^{19937}-1$) and 623-dimensional equidistribution. The reference code of MT19937 is provided for public use [2].

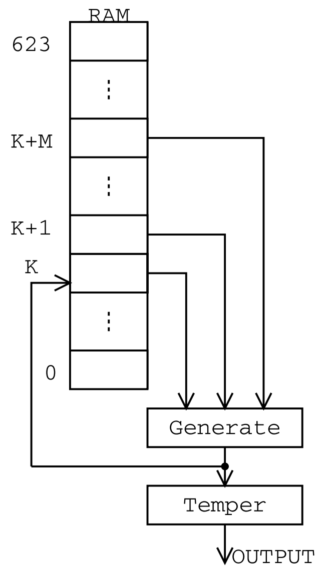

Fig. 1 Block diagram of SMT/PMT.

Kurokawa et al. [3], [4] reported a MT circuit design which was implemented as a peripheral device through a PCI bus. Their circuit [4] requires two cycles to generate one random number, and thus its performance is limited to 0.52 Gbps with a Xilinx XCV50 FPGA. Since their circuit does not include an initialization circuit, it must be externally initialized.

This study presents improved designs of the MT circuit, which are suited for high-performance simulations by custom computing machinery. Our designs generate one or more random numbers per cycle. They are highly suited to fully-pipelined or parallel implementations of simulation circuits. Circuits with initialization circuits are also examined and evaluated.

2. Designs of MT19937 Circuit

A basic MT19937 circuit consists of three modules: RAM, Generate, and Temper. RAM memorizes the internal state of MT19937. Generate calculates the next state of the circuit from the current state. Temper generates a random number from the next state. The block diagram of this basic circuit is shown in Fig. 1, where M is a constant 397. This circuit literally implements the algorithm of the reference code [2].

If Fig. 1 is implemented sequentially, the circuit will

Manuscript received May 30, 2005.
Manuscript revised August 11, 2005.
[†]The authors are with the Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441-8580 Japan.
[††]The author is with the Intelligent Sensing System Research Center, Toyohashi University of Technology, Toyohashi-shi, 441-8580 Japan.
[*]Presently, with Chuo System Corporation.
[**]This work partially appeared as an extended abstract in the 2005 Annual Meeting Record IEE Japan, vol.3, pp.89–90 (March 2005).
a) E-mail: ichikawa@tutkie.tut.ac.jp
DOI: 10.1093/ietisy/e88–d.12.2876

consume four cycles (3 read and 1 write cycles) to generate one random number. Implementing RAM with a dual-port RAM, one random number would be generated every two cycles. The latter design is designated as **SMT** in the following discussion.

SMT is functionally equivalent to the circuit shown by Kurokawa and Kajisaki [4]. Though SMT and Kurokawa's design adopt the same algorithm to generate one random number every two cycles, the implementations are slightly different. In Kurokawa's design, one read port of RAM was omitted by reusing the value of the previous cycle. This could make RAM resource smaller in exchange for a slight increase of logic scale and start-up cycles. In SMT, we did not adopt this technique, because it was possible for us to implement SMT without problem with a small amount of RAM.[†] It should be noted that all these things are the matters of implementation, which are dependent on the implementation platform (particularly on the specifications of RAM primitives). Since our platform (Altera Stratix FPGA) is different from Kurokawa's (Xilinx Virtex FPGA), the resulting implementations would be different, even if the function is the same. We intended to make SMT simple and portable, rather than making it optimized for a specific platform.

SMT can be extended to generate one random number for each cycle by using a multi-port RAM with 3R/1W ports. This design is designated as **PMT**. PMT is further enhanced by using a multi-bank configuration to generate multiple random numbers per cycle. Figure 2 illustrates an example of a 2-port PMT circuit (**PMT2**), which generates two random numbers in a cycle. In PMT2, the RAM of PMT is split into two banks; RAM0 and RAM1. RAM0 contains the elements of even indices, while RAM1 contains the elements of odd indices. RAM0 and RAM1 are multi-port RAMs, which have 3R/1W ports and 2R/1W ports, respectively. The idea of PMT2 corresponds to the unrolling of the innermost loop of software. PMT2 can be naturally extended to n-port PMT (**PMTn**), where n is a factor of 624. In this study, PMT2, PMT4, PMT8, PMT16, and PMT52 were designed and examined.

In a scientific simulation, it is a routine to generate multiple random numbers at the same time; for example, when one wishes to generate a particle in simulation physics. Four random numbers for its mass and three coordinates may be needed, or, six random numbers may be needed for its position vector and velocity vector. PMTn is suited to implement such cases with hardware.

MT is based on the Generalized Feedback Shift Register (GFSR) [1], which can be naturally implemented by chained registers (Fig. 3). This design is designated as **FMT** in the following discussion. FMT is also extensible to n-port implementations (**FMTn**) which generate n random numbers in each cycle ($1 \leq n \leq 227$). Figure 4 illustrates a 2-port version of FMT (**FMT2**).

PMT and its derivatives are based on RAM, which is well structured and suited for high-density implementation. However, the operational frequency of PMT is restricted by the access time of RAM. FMT and its derivatives have simple structures, operate at higher clock frequencies than PMT, and require more logic gates than the PMT family.

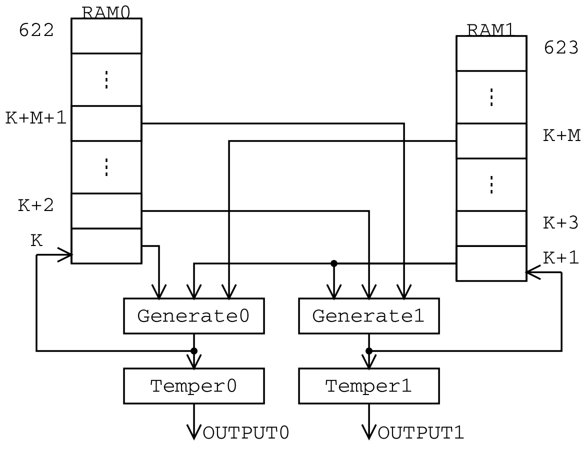

Fig. 2 Block diagram of PMT2.

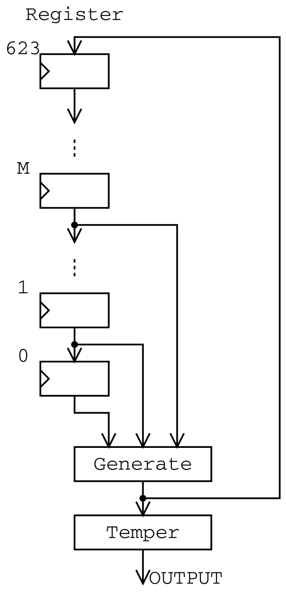

Fig. 3 Block diagram of FMT.

MT19937 requires the initial internal states of 624 words, where each word contains 32 bits. These initialization data can be downloaded from outside the circuit, as in the preceding studies [3], [4]. Or, one may include the initialization circuit which generates initial states from a given seed. In a hardware simulation, many PRNG circuits could be used in parallel. For such cases, it is desirable to include an initialization circuit with PRNG to reduce initialization overhead. Therefore, in this study, we examined the initialization circuit, which includes a multiplier and an adder. This circuit generates 32-bit data each cycle, consequently finishing initialization in 624 cycles. The MT circuits with initialization circuit are designated by the postfix "**_in**" in

[†]As shown in Sect. 3, one SMT unit requires 19 Kbit of RAM, which is as small as 2.2% of the total RAM capacity of an EP1S10 device (920 Kbit).

Table 1 Evaluation results.

Circuit	Logic (LE)	Max. Freq. (MHz)	Memory (bit)	DSP Block (block)	AT product (LE × sec.)	Throughput (Gbps)
mt19937ar.c	–	–	–	–	–	2.28
SMT	298	128.02	19,968	0	$4.656e-6$	2.05
PMT	278	143.04	39,936	0	$1.944e-6$	4.56
FMT	5,570	222.27	14,336	0	$2.506e-5$	7.11
PMT2	405	128.30	39,936	0	$1.578e-6$	8.21
PMT4	620	140.61	39,936	0	$1.102e-6$	18.00
PMT8	1,027	115.05	39,936	0	$1.116e-6$	29.45
PMT16	1,866	84.83	39,936	0	$1.375e-6$	43.43
PMT52	5,700	71.63	39,936	0	$1.530e-6$	119.19
FMT2	20,101	190.22	0	0	$5.284e-5$	12.17
FMT4	20,209	188.82	0	0	$2.676e-5$	24.17
FMT8	20,453	175.56	0	0	$1.456e-5$	44.94
FMT16	20,886	181.26	0	0	$7.202e-6$	92.81
FMT52	22,926	157.60	0	0	$2.797e-6$	262.25
SMT_in	427	63.03	19,968	8	$1.355e-5$	1.01
PMT_in	402	64.17	39,936	8	$6.265e-6$	2.05
PMT2_in	538	63.65	39,936	8	$4.226e-6$	4.07
PMT4_in	736	64.42	39,936	8	$2.856e-6$	8.25
PMT8_in	1,154	63.48	39,936	8	$2.272e-6$	16.25
PMT16_in	1,987	62.70	39,936	8	$1.981e-6$	32.10
PMT52_in	5,827	62.24	39,936	8	$1.800e-6$	103.57
SMT_in4	557	132.29	19,968	8	$8.421e-6$	2.12
PMT_in4	498	148.43	39,936	8	$3.355e-6$	4.75
PMT2_in4	630	128.06	39,936	8	$2.460e-6$	8.20
PMT4_in4	844	110.83	39,936	8	$1.904e-6$	14.19
PMT8_in4	1,252	103.85	39,936	8	$1.507e-6$	26.59
PMT16_in4	2,091	95.76	39,936	8	$1.364e-6$	49.03
PMT52_in4	5,925	74.16	39,936	8	$1.536e-6$	123.40
rand	35	65.10	0	8	$5.376e-7$	2.08
lrand48	72	56.04	0	16	$1.285e-6$	1.79
LFSR	33	390.02	0	0	$8.461e-9$	12.48

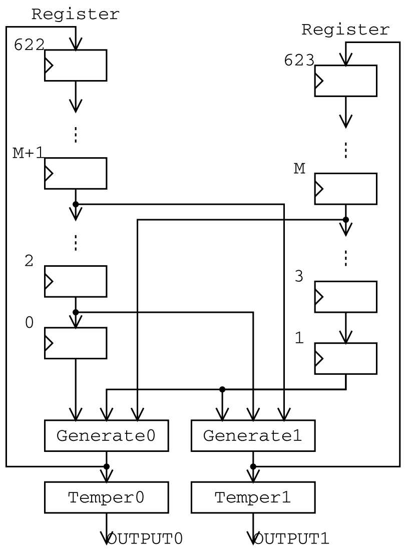

Fig. 4 Block diagram of FMT2.

the following discussion.

3. Evaluation

The abovementioned designs were described in VHDL, then synthesized and implemented for Altera Stratix FPGA [5] using Quartus-II 4.0 software. The target device was set to EP1S10F780C7ES. Design results are summarized in Table 1. For comparison, the performance of the reference software of MT19937 (**mt19937ar.c**) [2] was measured with an Intel Pentium 4 (2.53 GHz) processor, 1.0 GB main memory, gcc 3.2.3 -O, and FreeBSD 4.10.

Logic gates are counted by LE (logic elements) in Stratix architecture. Besides logic gates, some designs require memory elements and DSP blocks. A DSP block is a building block for a multiplier-adder, which is automatically used if the design includes multipliers. Area-Time product (AT product) is the product of the number of LEs and the time required to generate one random number; in other words, the AT product stands for the logic gates required to generate a random number in a unit time. Thus, a smaller AT product means a more efficient design.

The performances of PMT and FMT were superior to that of the software on Pentium 4 (2.53 GHz). Comparing SMT, PMT, and FMT, FMT operates at the highest frequency and shows the largest throughput. However, FMT requires far more LEs than the PMT, because registers (or flipflops) are relatively expensive resources in FPGA. Though the FMT design does not include any RAM explicitly, the derived FMT circuit includes RAM; this is because a simple FIFO structure was automatically implemented using RAM components. On the other hand, PMT inherently includes RAM blocks and naturally fits in FPGA.

Multi-port designs (PMT*n* and FMT*n*) show very high throughput, compared to the original PMT and FMT. Though the performance of FMT*n* is very high, FMT*n* is too large to fit into an EP1S10 device (10570 LE). PMT*n* is relatively small and fast. It should also be noted that PMT*n* is more cost-effective than PMT. For example, replacing a PMT2 with two PMTs, 37% more LE and 100% more memory resources are required for 11% more throughput. Though it is possible to use two or more MTs in parallel, they must be carefully chosen to be mutually independent [6]. Meanwhile, using PMT*n* is easy and simple to migrate from software simulation to hardware simulation. Considering all these facts, PMT*n* and FMT*n* would be better choices in many cases.

The circuits with an initialization circuit were also evaluated (SMT_in, PMT_in, and PMT*n*_in).† Though the increase of logic scale is negligible, the operational frequency is restricted to 63–64 MHz, because the multiplier-adder in the initialization circuit forms a new critical path. To avoid this critical path, we simply split the multiplier into 4 clock cycles. This makes initialization cycles 4 times longer, but the operational frequencies of resulting circuits become almost comparable to the corresponding circuits without initialization circuitry. These circuits are designated by the postfix "**_in4**" in Table 1.

For comparison, we examined three other PRNG circuits: rand, lrand48, and LFSR. Two ANSI C functions, **rand** and **lrand48** of the BSD version, are based on a classic linear congruent algorithm, while **LFSR** is a standard implementation of the Linear Feedback Shift Register. The periods of rand, lrand48, LFSR, MT19937 are 2^{30}, 2^{48}, $2^{32}-1$, and $2^{19937}-1$, respectively. Although rand, lrand48, and LFSR are smaller than ours, their throughputs are less than ours. It is readily seen that our MT circuits are superior in both performance and period.

4. Conclusion

This study presented new designs for a MT19937 circuit, which can generate multiple random numbers per cycle (multi-port designs). The estimated throughput of a 52-port design (FMT52) is 262 Gbps, which is 115 times higher than the software on a Pentium 4 (2.53 GHz) processor. Multi-port designs were shown to be more cost-effective than using multiple single-port designs. The initialization circuit can be included without performance loss in exchange for a slight increase of logic scale.

Two of our designs (SMT_in and PMT_in) were actually implemented on a Stratix EP1S10 evaluation board, and they were proven to be fully operational. Our designs are written in VHDL, built on standard components, and thus very portable and widely applicable. With our designs, PRNG circuits with MT algorithms could be more easily and widely applicable in custom computing hardware for scientific simulations.

†FMT and FMT*n* were excluded here, since they are too large in logic scale.

Acknowledgment

This work was partially supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS). Support for this work was also provided by the 21st Century COE Program "Intelligent Human Sensing" from the Ministry of Education, Culture, Sports, Science and Technology.

References

[1] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," ACM Trans. Model. Comput. Simul., vol.8, no.1, pp.3–30, 1998.

[2] M. Matsumoto, "Mersenne Twister: A random number generator," http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html

[3] T. Kurokawa and S. Fujimoto, "Development of Mersenne Twister using CPLDs," IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J84-D-I, no.5, pp.501–504, May 2001.

[4] T. Kurokawa and H. Kajisaki, "FPGA based implementation of Mersenne Twister," Scientific and Engineering Reports of the National Defense Academy, vol.40, no.2, pp.15–21, March 2003.

[5] Altera Corp., Stratix Device Handbook, July 2003. http://www.altera.com/

[6] M. Matsumoto and T. Nishimura, "Dynamic creation of pseudorandom number generators," in Monte Carlo and Quasi-Monte Carlo Methods 1998, pp.56–69, Springer, 2000.