電気学会論文誌D（産業応用部門誌）
IEEJ Transactions on Industry Applications
Vol.141 No.2 pp.86–92　　DOI: 10.1541/ieejias.141.86

論　文

内蔵LFSRとサンプリング間隔の揺らぎを利用した乱数生成手法

非会員　正岡　秀崇*　正　員　市川　周一*a)　非会員　藤枝　直輝**

Random Number Generation from Internal LFSR and Fluctuation of Sampling Interval

Hidetaka Masaoka*, Non-member, Shuichi Ichikawa*a), Member, Naoki Fujieda**, Non-member

（2020年3月6日受付，2020年6月24日再受付）

An unpredictable random number generator (URNG) adopts a deterministic algorithm with volatile internal states of a microprocessor, which makes the output of the URNG practically unpredictable. This study examines the URNG design proposed by Suciu et al., wherein performance counters are considered as entropy sources. Our experiments confirm that the URNG with performance counters requires a relatively long sampling interval with a background task to produce a high-quality random sequence. On this basis, we propose a new URNG design that is suitable for embedded systems. A simple 128-bit LFSR (Linear Feedback Shift Register) is built in a processor, whose lower 32-bit value is used as a random number. If an adequate sampling interval is maintained, the derived values pass the DIEHARD test.

キーワード：URNG, 組込みシステム，LFSR
Keywords: URNG, embedded systems, LFSR

1. はじめに

セキュリティ応用やシミュレーション用途では，乱数が頻繁に使用されている。乱数は生成方法によって大きく以下の2種類に分類される。

真性乱数（True Random Number；TRN）　熱雑音やジッタなど物理現象を利用して生成される。

疑似乱数（Pseudo Random Number；PRN）　確定的なアルゴリズムを利用して生成されるが，乱数の持つ統計的性質を再現しようとするもの。

疑似乱数は内部状態とアルゴリズムから計算できるが，真性乱数は物理現象に由来するので将来値を予測することはできない。真性乱数を生成するには専用ハードウェア（真性乱数生成器；TRNG）が必要で，一般にTRNGのコストは乱数生成速度や乱数品質に応じて増大することが知られている[1]。

疑似乱数生成器（PRNG）では，出力履歴や内部状態から値が予測できる可能性がある。しかしPRNGソフトウェアを実行するプロセッサは複雑な順序機械であり，プロセッサの内部状態はクロック毎に変化するため，それを利用すれば予測困難な乱数が生成できると期待される[2]。

Suciuら[3][4]は，Intel社のCPUがもつパフォーマンスカウンタを内部状態として利用することにより，実質的に予測不可能な乱数生成手法を提案した（Unpredictable Random Number Generator；URNG）。パフォーマンスカウンタはシステム性能のモニタ等で利用されるレジスタで，ソフトウェアから利用可能であり，プロセッサに標準装備されているので追加コスト無しに利用可能である。Suciuら[4]は，複数のパフォーマンスカウンタをXORで集約すること，サンプリング周期を充分長くすること，複数のスレッドを生成すること，などの手法を併用することにより，NIST検定をパスする乱数列が得られたと報告している。重名[5]はパフォーマンスカウンタを用いたURNGをRISC-Vの専用命令として実装したが，DIEHARDテストによる検定に合格することができなかった。

a) Correspondence to: Shuichi Ichikawa. E-mail: ichikawa@ieee.org
* 豊橋技術科学大学 電気・電子情報工学専攻
〒441-8580　愛知県豊橋市天伯町雲雀ケ丘1-1
Department of Electrical and Electronic Information Engineering, Toyohashi University of Technology
1-1, Hibarigaoka, Tempaku-cho, Toyohashi, Aichi 441-8580, Japan
** 愛知工業大学工学部 電気学科
〒470-0392　愛知県豊田市八草町八千草1247
Faculty of Engineering, Aichi Institute of Technology
1247, Yachigusa, Yakusa-cho, Toyota, Aichi 470-0392, Japan

© 2021 The Institute of Electrical Engineers of Japan.

本研究の最終目的は，組込みシステム向けの低コストかつ高品質な乱数生成手法を提供することである。まず 2 章では，Suciu らの URNG を追実験し，実験結果を報告する。次に 3 章では，2 章で得られた知見を踏まえて，低コストなハードウェアに基づく URNG を提案し，その実装・評価結果を報告する。なお本論文は電気学会次世代産業システム研究会で発表した原稿[6]に大幅な加筆修正を施したものである。

2. Suciu らの URNG

〈2･1〉 実験環境 2 章の実験環境は Table 1 に示す通りである。Intel 製プロセッサ，Linux，PAPI という組合せは，先行研究[3][4]に合わせて選択した。ただしプロセッサの品種とソフトウェアのバージョンは先行研究と若干異なる。

乱数品質の評価は統計的検定により行われる[8]。近年では NIST テスト[9]が多く用いられているが，必要なデータ量が大きいため，試行錯誤の多い研究段階には不向きである。そこで本研究では，藤枝ら[10]に準じて，DIEHARD テスト[11]を評価に用いる。

DIEHARD テストは全 18 種のテストからなり，各テストで 1～100 個（合計 313 個）の p 値を出力する（Table 2）。合格判定の基準は定められておらず，利用者の判断に委ねられているので，本研究では以下の基準で乱数品質を評価する。

入力が理想的乱数であれば p 値は区間 [0,1) で均等に分布することが期待されるので，Table 3 に示した基準で各 p 値の成功（PASS）／弱成功（WEAK）／失敗（FAIL）を判定する。FAIL の発生確率（期待値）は 2×10^{-6} なので，入力が乱数であれば（ほぼ）発生しない。WEAK の発生確率（期待値）は 1×10^{-2} なので，WEAK が 1%程度発生することは正常である。

単純に 313 個の p 値について PASS/WEAK/FAIL の個数を示すと，多くの p 値を出力するテストのウエイトが大きく見えてしまう。そこで以下の方法により，各テストの結果を判定する。各テストで出力される p 値の個数が 9 個以上であれば，得られた p 値の分布が一様であるかどうかの判定を Kolmogorov-Smirnov 検定により行い，得られた p 値を Table 3 に示した基準で判定する。テストの出力する p 値が 9 個未満であれば，以下に述べる方法で結果を判定する。各テストで出力される p 値に，ひとつでも FAIL が含まれれば，そのテストは FAIL。出力される p 値に FAIL はなく WEAK が含まれれば，そのテストは WEAK。出力される p 値が全て PASS であれば，そのテストは PASS とする。

以下, 本論文では, 全 18 個のテストの PASS/WEAK/FAIL の内訳により乱数列の品質を評価する。

〈2･2〉 パフォーマンスカウンタの選択 まず，乱数生成に使用するパフォーマンスカウンタを選択した。

先行研究[4]では，乱数品質を改善するため 5 つのパフォーマンスカウンタ（PAPI_TOT_IIS，PAPI_TOT_INS，PAPI_TOT_CYC，PAPI_L1_ICH，PAPI_L1_ICA）を XOR して集約することを提案している。しかし本研究ではプロセッサ品種が異なるため，利用できるパフォーマンスカウンタが異なり，そのまま追実験することができない。

そこで，個別のカウンタ値を取得して DIEHARD で乱数品質を吟味し，テスト結果の比較的良い 5 つのレジスタ（PAPI_BR_PRC，PAPI_BR_UCN，PAPI_LD_INS, PAPI_LST_INS, PAPI_TOT_CYC）を XOR して用いた。（この評価結果の詳細は紙数の関係で省略する。）XOR するカウンタを増やせば乱数品質は向上すると考えられるが，実際には総サイクル数（PAPI_TOT_CYC）の寄与が支配的で，他のカウンタの影響は少なかった。多数のカウンタを使うと読出し時間が増えて乱数生成速度が下がるので, 乱数品質に寄与しないカウンタまで XOR することは避けたい。そこで先行研究と同じ 5 つのカウンタで実験を行った。

また先行研究[4]では，各カウンタの下位何ビットを用いるか, という検討が行われていた。カウンタの性質上, 下位ビットほど値が変わる頻度が高く, 上位ビットでは乱数性が低くなると予想される。Fig. 1 は，DIEHARD テストの結果をまとめたものである。5 つのカウンタを XOR する際，下

Table 1. Experiment environment.

Linux	Ubuntu 18.04.1 LTS
Kernel Version	4.15.0-29-generic
CPU	Intel Core-i5 4590 @ 3.3 GHz
DRAM	16 GB
Library	PAPI 5.7.1.0 (Language: C)[7]
Compiler	GCC 7.3.0
Test Suite	DIEHARD[11]

Table 2. DIEHARD test[11].

Test	Num. p-values
Birthday Spacings	9
Overlapping 5-permutation (OPERM5)	2
Binary Rank (31x31)	1
Binary Rank (32x32)	1
Binary Rank (6x8)	25
Overlapping 20-tuples Bitstream	20
Overlapping Pairs Sparse Occupancy (OPSO)	23
Overlapping Quads Sparse Occupancy (OQSO)	28
DNA	31
Count the 1’s (stream)	1
Count the 1’s (specific)	25
Parking Lot	10
Minimum Distance	100
3-d spheres	20
Squeeze	1
Overlapping Sums	10
Runs Up/Down	4
Craps	2
	313

Table 3. DIEHARD evaluation criteria.

Decision	Condition
PASS	$0.005 \le p < 0.995$
WEAK	$0.000001 \le p < 0.005$, or $0.995 \le p < 0.999999$
FAIL	$p < 0.000001$, or $0.999999 \le p$

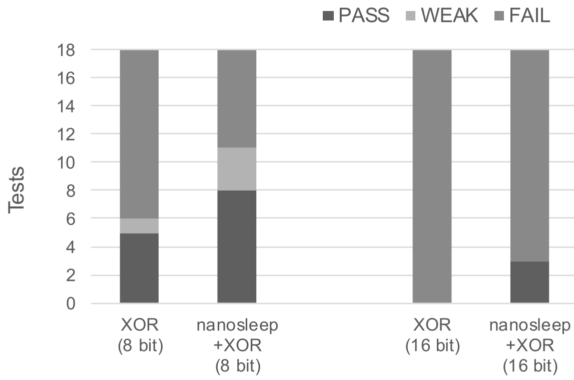

Fig. 1. DIEHARD result for sampling bit-width.

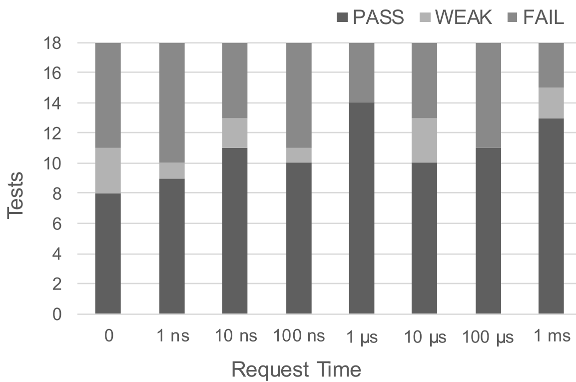

Fig. 2. DIEHARD result vs. sampling interval.

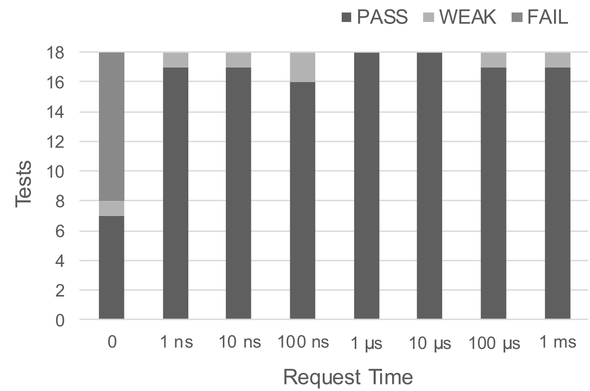

Fig. 3. DIEHARD result with HimenoBMT.

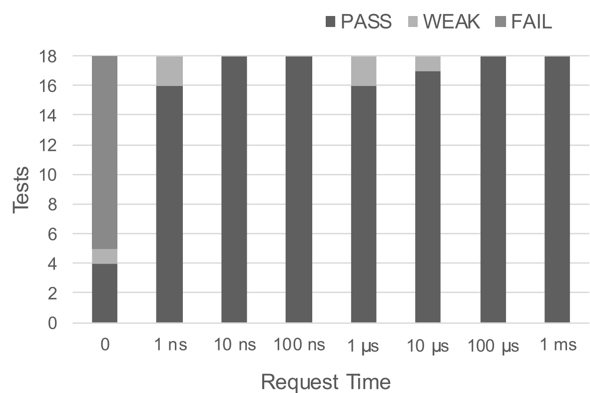

Fig. 4. DIEHARD result with IOZONE.

位 8 ビットだけサンプルする場合（XOR（8 bit）），下位 16 ビットをサンプルする場合（XOR（16 bit））を比較すると，明らかに 8 ビットが優れていた。次項で述べる nanosleep 関数を併用した場合の結果も合わせて示す。nanosleep の要求時間は 1 ns としている。これらの結果から，以下，2 章ではカウンタの下位 8 ビットだけを利用する。

〈2･3〉 サンプリング周期 先行研究[4]は，サンプリングの間隔を伸ばすと乱数品質が向上することを報告している。具体的には，（1）演算処理，（2）スレッド生成，（3）プロセス生成，（4）メモリアクセス，などの実行時間により，サンプリング間隔を増やしている。本研究のターゲットは組込みシステムなので，上記の方法は必ずしも好ましくない。不要な演算やメモリアクセスは，システム性能低下や消費電力増大につながる。また，組込み OS では動的なスレッド／プロセス生成を利用できない場合もある。

本研究では，POSIX 仕様のシステムコール nanosleep を使用して，サンプリング間隔を伸ばした。nanosleep では，引数で要求した時間だけプログラムの実行が遅延される。無駄に CPU 時間を使うことがないので，他のタスクの実行を邪魔することがない。

Fig. 2 に，nanosleep の要求時間ごとのテスト結果を示す。横軸「0」の項は，nanosleep を使わず，遅延なしで繰り返しサンプルした結果を表す。要求時間が長いと PASS 数が増える傾向がみられるが，サンプリング間隔を伸ばすだけでは DIEHARD テストに合格しないことが分かった。

〈2･4〉 バックグラウンド応用 次に，乱数収集のバックグラウンドで別プロセスを実行することにより，乱数品質に影響があるか調べた。パフォーマンスカウンタはシステムの状態や性能を示す指標なので，バックグラウンドのプログラムも乱数品質に影響を及ぼす可能性がある。

バックグラウンドで実行する応用プログラムとしては，それぞれ性質の異なる以下の 3 種を利用した。

HimenoBMT CPU に負荷をかけるベンチマークプログラム。ポアソン方程式をヤコビの反復法で解く[12]。

IOzone ファイルシステムに負荷をかけるベンチマークプログラム。入出力を多く発生する[13]。

STREAM メモリに負荷をかけるベンチマークプログラム。メモリ帯域を多く消費する[14]。

Fig. 3，Fig. 4，Fig. 5 に，各応用の評価結果をまとめる。横軸は nanosleep の要求時間で，横軸「0」の項目は nanosleep を使わない場合を表す。

応用の性質によらず，nanosleep を使用して 1 ns 以上の待ち時間を指定した場合に，DIEHARD テストに合格した。ただし，バックグラウンド応用が実行されていても，nanosleep を使わないとテストに合格できない。また，バックグラウンド応用がなければ，同じ時間 nanosleep してもテストに合格しない（Fig. 2）。つまり，バックグラウンド応用と nanosleep 両方が揃って，初めて乱数品質の向上が得られる。

この現象は以下のように解釈できる。

- nanosleep を使わない場合，乱数収集プロセスが CPU

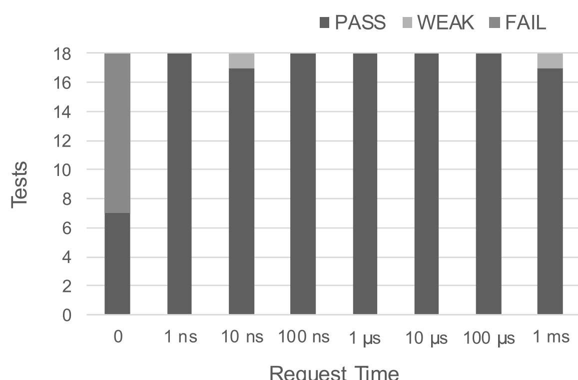

Fig. 5.　DIEHARD result with STREAM.

を利用している期間（タイムスライス），連続でパフォーマンスカウンタを読むため値に大きな変化がなく，良い乱数は得られない。

- nanosleep を実行すると，乱数収集プロセスは指定時間スリープして，他のプロセスが CPU を使用する。
  - このときバックグラウンド応用がないと，実行可能なプロセスがなくなり，CPU は停止状態になる。要求時間が経過し乱数収集プロセスが再開されても，停止によりパフォーマンスカウンタに大きな変化がないので，良い乱数が得られない。
  - nanosleep 中にバックグラウンド応用が実行されると，乱数収集プロセスが再開されるまでにパフォーマンスカウンタの値が大きく変化するため，乱数品質が向上する。

〈2･5〉 本章のまとめ　以上の実験結果，および Suciu ら[(4)]の結果で示された通り，1 個〜数個のパフォーマンスカウンタを XOR するだけでは，充分な品質の乱数は得られない。Suciu ら[(4)]は，サンプリング周期を長くし，OS およびプログラムによる影響を加えることで，乱数検定を通過すると報告した。我々の実験でも，サンプリング周期を大きくすると乱数品質は向上したが，バックグラウンド応用なしでは DIEHARD テストに合格しなかった。バックグラウンド応用を実行して nanosleep することにより，DIEHARD テストに合格した。このように方法は異なるが，Suciu らの結果を概ね再現することができた。

Suciu らの URNG には，幾つかの問題がある。

まず第一に，OS やバックグラウンド応用が乱数品質に影響を及ぼすことは問題である。Suciu らは計算サーバ（Intel + Linux）をターゲットとしたので，OS を含むシステムは複雑であり，多くのバックグラウンドプロセスを仮定できる。しかし組込みシステムは単純なものから複雑なものまで多岐にわたる。単純なシステムではパフォーマンスカウンタに十分な変化が生まれない可能性がある。また，実時間 OS ではバックグラウンド応用が存在しても，優先度やスケジューリングにより充分に実行されないかもしれない。

パフォーマンスカウンタは多くのアーキテクチャで提供されているため（Intel，MIPS，ARM，PowerPC 等），広範囲に適用できる可能性がある。これは Suciu らの方法の利点である。しかしパフォーマンスカウンタの仕様はプロセッサ品種ごとに異なっているため，Suciu らの URNG はプロセッサ品種毎に設計と検証が必要になる。多様性の大きい組込みシステムでは，このような再設計・再検証は大きな負担になる。

URNG のためのハードウェアサポートをプロセッサに付加することによって，容易に高品質な乱数列が得られる可能性がある。組込みシステムでは，プロセッサを含むハードウェアを用途に合わせてカスタマイズすることが多い。実装コストが安ければ，URNG 専用の回路を追加することは充分可能である。

次の章では，本章の知見を踏まえて，新たな URNG 設計について検討する。

3. LFSR を利用した URNG

〈3･1〉 基本的な考え方　2 章の実験では，パフォーマンスカウンタのエントロピー不足のため，URNG の使用条件と生成速度に大きな制約が生じていた。そこで，低コストなハードウェアを追加することにより，URNG の出力する乱数品質を改善することを試みる。2 章で採用したパフォーマンスカウンタは，(1)値が余り変化しない，あるいは(2)カウンタの性質として変化の規則性が強かった。そこで本章では，(1)値が毎サイクル変化し，(2)不規則な値になるように，疑似乱数生成器（PRNG）を利用する。

PRNG には色々な種類があるが，本研究ではハードウェア実装のコストが低いことを重視して，LFSR（Linear Feedback Shift Register）を採用する。プロセッサ内部に LFSR を実装し，その値を毎クロック更新する。ソフトウェアは専用レジスタ（以下 URNG レジスタと呼ぶ）を通じて LFSR の値を読み出す。LFSR を周辺回路として追加することもできるが，その場合はバス経由のアクセスになり，命令列や実行サイクル数がパフォーマンスカウンタの場合と大きく異なるものになる。そこで本研究では，パフォーマンスカウンタと同条件で評価するため URNG レジスタとして実装する。

〈3･2〉 シミュレーション　まずシミュレーションで本提案の妥当性を検討する。以下の評価では 32 ビットと 128 ビットの LFSR を検討する（Table 4）。URNG レジスタの幅は 32 ビットとし，32 ビット LFSR では 32 ビット全体，128 ビット LFSR の場合は最下位 32 ビットだけが読みだされる。128 ビット LFSR の上位 96 ビットはユーザに見えないため，内部状態が隠蔽されて，攻撃者による将来値の予測が困難になる。

Fig. 6 は 32 ビット LFSR の値，Fig. 7 は 128 ビット LFSR の下位 32 ビットの値を DIEHARD テストで検定した結果である。縦軸は全 18 テストの結果を PASS/WEAK/FAIL で集計したものである。横軸は乱数のサンプリング間隔を示しており，間隔 0 は毎サイクルサンプルした場合，間隔 31 は 32 クロックに 1 回（定期的に）サンプルした場合で

Table 4.　LFSR specifications.

Length	Primitive polynomial
32 bit	$X^{32}+X^7+X^5+X^3+X^2+X+1$
128 bit	$X^{128}+X^7+X^2+X+1$

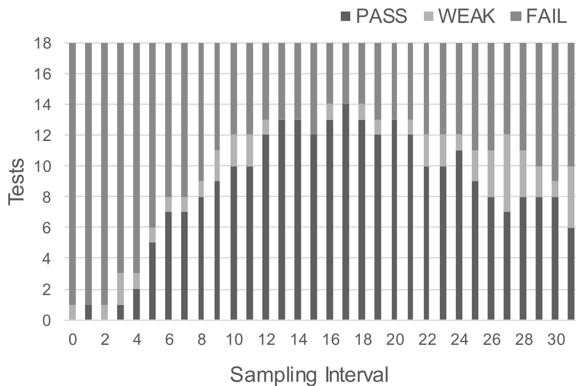

Fig. 6.　DIEHARD result of 32-bit LFSR simulation.

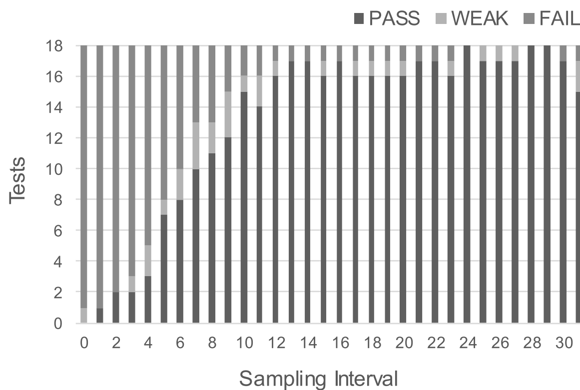

Fig. 7.　DIEHARD result of 128-bit LFSR simulation.

ある。URNG レジスタは 32 ビットなので，32 クロックで全ビットが更新されることから，間隔 31 を横軸の上限とした。

32 ビット LFSR では，間隔を増やしても多くのテストに失敗する。これは内部状態数の不足，あるいは周期が短いことによると解釈される。一方 128 ビット LFSR では，内部状態が多いため，間隔が増えると（ほぼ）失敗はなくなる。サンプル間隔が長いと乱数品質が向上することは，パフォーマンスカウンタの場合と同様で，直観に一致している。

Fig. 6 と Fig. 7 は，サンプリング間隔一定という条件下の結果である。実システムにおいては様々な要因でサンプリング間隔に揺らぎが生じ，それによって乱数品質が向上することが期待される。その意味で，Fig. 7 は揺らぎのない最悪条件での評価であるといえる。サンプリング間隔に揺らぎを入れたシミュレーションも可能であるが，揺らぎの実情は使用条件により様々であるため，この検討は今後の課題とする。

〈3･3〉 実　　装　以下，本研究では，実機上に提案手法を実装して乱数品質の検証を行う。使用する実装環境を Table 5 にまとめる。

Table 5.　Implementation environment.

Processor	PULPino [15] (RI5CY Core)
Evaluation Board	AVNET ZedBoard AES-Z7EV-7Z020-G
Device	Xilinx XC7Z020-CLG484-1
Target Freq.	20 MHz
OS	FreeRTOS [16] v8.2.2

PULPino [15] は，ETH Zurich で開発された RISC-V アーキテクチャのマイクロプロセッサである。オープンソースで公開されており，HDL で記述されているため，変更や拡張が容易である。シミュレーション結果に基づき，本実装では 4 段パイプラインの RI5CY コアに 128 ビット LFSR を組み込んで，下位 32 ビットを URNG レジスタとして読み出せるようにした。論理規模削減のため，RI5CY は FPU なしで実装している。PULPino のクロック周波数は 20 MHz とした。

RI5CY コアで動作する組込み OS は，FreeRTOS [16] である。FreeRTOS はオープンソースの実時間 OS で，多くのプロセッサをサポートしている。

評価ボード（ZedBoard）には Xilinx Zynq-7000 SoC [17] が搭載されている。Zynq-7000 は Processing System（PS）部と Programmable Logic（PL）部からなり，PULPino は PL 部に実現される。PS 部には ARM Cortex-A9 ベースのプロセッサが搭載されており，Linux が動作している。PS と PL 間の通信は SPI（Serial Peripheral Interface）によって行われ，PULPino の監視と通信は Linux 上のプログラムが担当している。

FPGA 実装における PULPino の論理規模を Table 6 に示す。おおまかに，FF（flipflop）は記憶素子，LUT（look-up table）は論理ゲートに対応する。その他の要素，BlockRAM，乗算器，I/O 関連，クロック関連も参考として示す。開発に使用した CAD ソフトウェアは Xilinx Vivado 2015.1 である。

提案手法を追加したことにより，FF は 1.4%増，LUT は 1.5%減となった。128 ビット LFSR だけを単体で評価すると，FF が 128，LUT が 46 になるので，論理規模の増加は PULPino 本体に対して十分小さいといえる。LUT 数が減少するのは直観に反するが，CAD の最適化ではあり得ることである。CAD は論理規模と遅延時間のトレードオフを行うので，クリティカルパス外では遅延時間増大と引き換えに論理規模を削減することがある。変更前後の設計でクリティカルパスを調べると，ALU 部分であり，追加部分ではなかった。また解の探索に乱択が用いられる場合，試行ごとに解の品質に幅が生じることは自然である。

〈3･4〉 乱数品質と生成速度　〈3･3〉節で述べた実装を用いて，実際に URNG レジスタを読みだしてデータを取得した。2 章では，（1）データのサンプリング毎に nanosleep すること，および（2）バックグラウンド応用で内部状態に変化を与えること，が必要とされていた。nanosleep は Linux

Table 6.　Logic Scale.

	FF	LUT	Memory LUT	I/O	BRAM	DSP48	BUFG	MMCM
PULPino	11776	17016	45	32	16	6	5	1
PULPino + 128-bit LFSR	11938	16765	45	32	16	6	5	1
Available	106400	53200	17400	200	140	220	32	4

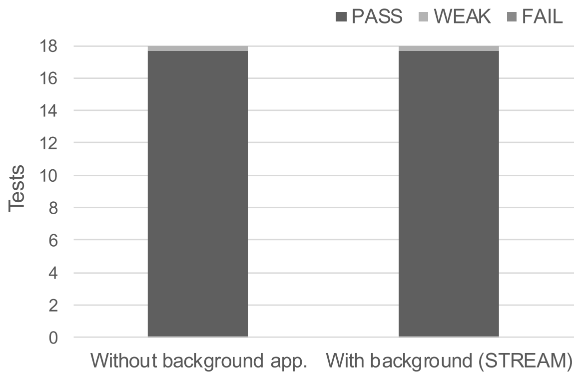

Fig. 8.　DIEHARD result of 128-bit LFSR implementation.

Table 7.　Generation rate.

Without background app.	125.0　kbit/s
With background (STREAM)	2.1　kbit/s

のシステムコールであるため，PULPino の FreeRTOS では taskYIELD を用いた。taskYIELD は FreeRTOS のカーネルに context switch を要求する API である。バックグラウンド応用については，3 種で大きな差がなかったため，FreeRTOS へのポーティングが楽な STREAM だけを利用した。

Fig. 8 に，得られた乱数列の検定結果を示す。3 セットのデータを測定し，DIEHARD にかけた結果の平均を示している。また，Table 7 に生成速度の実測値を示す。

LFSR を用いた URNG では，バックグラウンド応用の有無に関わらず DIEHARD テストに成功した。3 セット（18×3 回のテスト）中，1 回だけ WEAK，残る 53 回は全て PASS であった。〈2·1〉節でも述べた通り，WEAK は 1%程度発生しても正常なので，この結果に問題はないと考えられる。

バックグラウンド応用なしで乱数検定に合格する理由は，以下のように解釈される。URNG レジスタ（32 ビット）から 125 kbit/s の生成速度を得ているということは，サンプル間隔は平均 0.256 ms と計算される。PULPino のコアの動作周波数は 20 MHz なので（Table 5），LFSR はサンプル毎に平均 5.1×10^3 サイクルは変化していることになる。Fig. 7 でも示した通り，これは充分に長いサンプリング間隔である。さらに割込み等の擾乱要因によりサンプリング間隔の揺らぎが生じて，乱数品質を向上させる。このためバックグラウンド応用なしでも乱数検定に通過したと思われる。バックグラウンド応用が存在すれば，サンプリング間隔は更に伸び，生成速度は下がるが乱数品質は維持される。

以上の結果から，LFSR による URNG を実装すれば，ユーザがバックグラウンドで任意の処理を実行する（あるいは何もしない）場合でも良い乱数列を得ることができると期待される。CPU や OS の内部状態を利用する既存手法に対して，提案手法の優位性が示されたといえる。

4.　おわりに

本研究では，システムクロックに同期して動く LFSR をプロセッサに付加することにより，実質的に予測不能な乱数生成器（URNG）を実現することを提案した。128 ビット LFSR はシステム全体から見て極めて小さく，低いコストで URNG を実現することができる。ユーザ（ソフトウェア）は必要に応じて URNG レジスタを読み出すだけであり，得られた乱数列は後処理なしで DIEHARD テストを通過する。

LFSR を追加する代わりに，プロセッサ内部のパイプラインレジスタや信号線を利用して URNG を作成することも考えられる。その場合，プロセッサ内部に新たなクリティカルパスが生じて，動作可能周波数が低下する恐れがある。また URNG 設計がプロセッサの実装に依存するため，異なる実装ごとに再設計が必要になり，乱数品質の再検証が必要になる。さらに乱数出力が実行する命令列に影響されるなど，攻撃の余地が残ることも懸念される。それに対して，LFSR の追加は低コストであり，プロセッサによらず簡単に実装可能である。サンプリング周期を適切にとれば，アーキテクチャによらず乱数品質を維持することができる。これらの点から，提案手法による URNG が優れていると考える。

近年，組込みプロセッサにも真性乱数生成器（TRNG）が実装されているが，一定品質の TRNG には相応の実装コストが必要になる。TRNG はアナログ動作をするため電源電圧や動作温度などの影響を受け，またリングオシレータ型 TRNG に対する Frequency Injection Attack など出力に干渉する攻撃手法も知られている。一方，提案手法は完全なデジタル動作なので，より動作環境に対してロバストである。実装コストも安く，コストセンシティブな組込みシステムに適した方式であるといえる。

本研究では，FPGA ボード上の組込みシステムで提案手法を実装し，動作を検証した。今後は，様々な利用環境における乱数品質を評価し，適用範囲の拡大を検討する必要がある。

本研究の実装では，URNG のサンプル間隔は平均 5.1×10^3 サイクルであった。今後は，実装を改善してサンプル間隔を小さくする（即ち生成速度を大きくする）と共に，サン

プル間隔と乱数品質のトレードオフについて定量的評価を進める。また，本研究では DIEHARD テスト[11]を用いて乱数品質を検証したが，今後は NIST テスト[9]による検証を行いたい。

謝　辞

本研究の一部は JSPS 科研費 20K11733 の支援による。また，本研究の初期段階で文献調査と環境構築に貢献した重名英史氏に感謝いたします。

文　献

（1） H. Hata and S. Ichikawa: "FPGA Implementation of Metastability-based True Random Number Generator", IEICE Trans. Info. Syst., Vol.E95-D, No.2, pp.426–436 (2012)
（2） A. Seznec and N. Sendrier: "Havege: A user-level software heuristic for generating empirically strong random numbers", ACM Trans. Model. Comput. Simul., Vol.13, No.4, pp.334–346 (2003)
（3） A. Suciu, S. Banescu, and K. Marton: "Unpredictable random number generator based on hardware performance counters", Digital Information Processing and Communications, pp.123–137, Springer Berlin Heidelberg (2011)
（4） K. Marton, A. Zaharia, S. Banescu, and A. Suciu: "Randomness Assessment of an Unpredictable Random Number Generator based on Hardware Performance Counters", ROMJIST, Vol.20, No.2, pp.136–160 (2017)
（5） H. Juna: "Random Number Generation Instruction which utilizes internal state of processor", Toyohashi University of Technology, Dept. Electrical and Electronic Information Engineering, Master's thesis (2018) (in Japanese)
重名英史：「プロセッサの内部状態を用いた乱数生成命令」, 豊橋技術科学大学 大学院 電気・電子情報工学専攻, 修士論文 (2018)
（6） H. Masaoka, S. Ichikawa, and N. Fujieda: "Random number generation from internal states and timing fluctuation in microprocessor", The Papers of Technical Meeting on Innovative Industrial System, IEE Japan, IIS-20-018 (2020) (in Japanese)
正岡秀崇・市川周一・藤枝直輝：「マイクロプロセッサの内部状態とタイミング揺らぎを利用した乱数生成手法」, 電学次世代産業システム研究会, IIS-20-018 (2020)
（7） "Downloading and Installing PAPI", Mar. 4 2019. http://icl.utk.edu/papi/software/ (accessed 2019-06-16)
（8） 藤井光昭：「暗号と乱数—乱数の統計的検定—」, 共立出版 (2018)
（9） L. Bassham, et al.: "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", NIST SP 800-22 (Rev.1a) (2010)
（10） N. Fujieda and S. Ichikawa: "A Latch-latch Composition of Metastability-based True Random Number Generator for Xilinx FPGAs", IEICE Electronics Express (ELEX), Vol.15, No.10, pp.1–12 (2018)
（11） G. Marsaglia: "Diehard battery of tests of randomness (Archived)", https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/ (accessed 2020-06-20)
（12） "Himeno Benchmark", http://accc.riken.jp/supercom/documents/himenobmt/ (accessed 2019-10-02)
（13） "IOzone Filesystem Benchmark", http://www.iozone.org/ (accessed 2019-10-18)
（14） J.D. McCalpin: "STREAM: Sustainable Memory Bandwidth in High Performance Computers", http://www.cs.virginia.edu/stream/ (accessed 2019-10-18)
（15） "pulp-platform/pulpino", Aug. 3 2017. https://github.com/pulp-platform/pulpino/tree/master/fpga (accessed 2019-05-22)
（16） "The FreeRTOS Kernel, Market Leading, De-facto Standard and Cross Platform RTOS kernel", https://www.freertos.org/ (accessed 2019-05-22)
（17） Xilinx, Inc.: "Zynq-7000 SoC Data Sheet: Overview", DS190 (v1.11.1) (2018)

正　岡　秀　崇　（非会員）　2018 年豊橋技術科学大学電気・電子情報工学課程卒業。同年，同大学大学院電気・電子情報工学専攻博士前期課程入学。2020 年 3 月，同大学大学院電気・電子情報工学専攻博士前期課程修了。同年 4 月より（株）新来島豊橋造船。

市　川　周　一　（正員）　1985 年東京大学理学部卒業。1987 年同大学大学院理学系研究科修士課程修了。1987 年新技術事業団，1991 年三菱電機（株），1994 年名古屋大学工学部助手。1997 年豊橋技術科学大学工学部講師。同助教授，准教授を経て，2011 年沼津工業高等専門学校制御情報工学科教授。2012 年より豊橋技術科学大学大学院工学研究科教授。現在に至る。理学博士。IEEE（senior member），電子情報通信学会（シニア会員），ACM，情報処理学会，各会員。

藤　枝　直　輝　（非会員）　2013 年東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程修了。博士（工学）。同年より豊橋技術科学大学電気・電子情報工学系助教。2019 年より愛知工業大学工学部電気学科講師。プロセッサアーキテクチャ，FPGA 応用，組み込みシステム，セキュアプロセッサの研究に従事。情報処理学会，電子情報通信学会，IEEE 各会員。