SCIS 2009 The 2009 Symposium on Cryptography and Information Security
Otsu, Japan, Jan. 20-23, 2009
The Institute of Electronics, Information and Communication Engineers

RS ラッチのメタスタビリティを利用した真性乱数生成回路
True random number generator based on metastability of RS latch

市川 周一 * 畑 尚志 *
Shuichi Ichikawa Hisashi Hata

あらまし 真性乱数生成回路 (TRNG) はセキュリティの基盤技術として極めて重要である．アナログ回路による TRNG も実用化されているが，論理 LSI に集積するためにはデジタル回路による TRNG 設計が必要である．発振器のジッタを利用する TRNG はデジタル回路で構成できるが，リングオシレータの消費電力が大きい，あるいは付加部品や PLL が必要，という問題がある．メタスタビリティを利用する TRNG も同期式デジタル回路だけで構成できるが，実装が難しく乱数生成速度に問題があるとされてきた．本研究では，単一クロックのラッチ型 TRNG を Xilinx Virtex4 FPGA で実装し，後処理なしで NIST テストに合格することを報告する．この TRNG は，145〜580 slice の論理規模で 3.8〜12.5 Mbps の生成速度を実現する．

キーワード 乱数生成, 同期式デジタル回路, FPGA, 準安定状態

1 はじめに

多くのセキュリティ技術は，その安全性の根拠を乱数に依存している．そのため，ハードウェアによる真性乱数の生成は，実用上極めて重要である．

真性乱数生成回路 (TRNG; true random number generator) については，これまでも多くの研究が行われてきた．例えば熱雑音等の物理現象に基づいた TRNG が実用化されているが，これらはアナログ回路で実装されており，デジタル回路との混在は簡単でない．デジタル回路だけで TRNG を構成することができれば，論理回路と容易に接続でき，論理 LSI 上に集積することも可能になる．

本研究では，RS ラッチのメタスタビリティを利用した TRNG を設計・実装し，その評価結果を報告する．この回路は同期式デジタル回路で構成されているため，基本的に全ての論理 LSI 上で実現可能である．本研究では実装評価に FPGA を使用したが，FPGA はカスタム LSI と比べて実装上の制約が大きいため，FPGA で実装可能であれば他の論理 LSI (カスタム，セミカスタム) でも支障なく実現できると考えられる．また FPGA は近年広く利用されているため，FPGA による TRNG 実装は，それ自体実用的価値が高いといえる．

* 豊橋技術科学大学・知識情報工学系, 441-8580 豊橋市天伯町雲雀ヶ丘 1-1, Dept. Knowledge-based Information Engineering, Toyohashi University of Technology, 1-1 Hibarigaoka, Tempaku, Toyohashi 441-8580, Japan. ichikawa@ieee.org

2 関連研究

デジタル回路による TRNG は，そのエントロピー源によって大きく分類することができる．

Fairfield ら [1] は，クロックより高周波数でクロックと独立な発振波形を DFF (D flip-flop) の D 入力に入力することにより，TRNG が構成できることを示した (発振器サンプル型)．Tsoi ら [2] は，Xilinx Virtex FPGA XCV300E を用いて発振器サンプル型の TRNG を実装・評価し，乱数生成速度 5〜29 kbps で NIST テスト [3][4] および Diehard テスト [5] に合格することを報告した．

発振器サンプル型では 2 系統以上の独立な発振器が必要なので，Tsoi らは FPGA の IOB に可変抵抗とキャパシタを外付けして，低周波発振器を構成した．一方本研究では同期式デジタル回路による TRNG を提案しており，外付け部品や発振器調整は不要である．

チップ内部で 2 系統以上のクロックが発生できれば，Tsoi らの問題点は回避できる．Fischer ら [6] [7] は，Altera 社の FPGA に搭載されている PLL (Phase-Locked Loop) を利用して，PLL のジッタから真性乱数を生成する手法を提案した．回路は APEX EP20K と Stratix で実装・評価され，NIST テスト [3][4] で有効性が確認された．同様に Kwok と Lam [8] は，Xilinx Vertex II Pro FPGA を用いて，チップに搭載された DCM (Digital Clock Manager) のジッタから真性乱数を生成する TRNG を提案した．

もちろんこれらの手法は，PLL や DCM の搭載された FPGA でなければ利用できない．一方本研究では，汎用の

論理回路から TRNG を構成する方法について検討する．

奇数のインバータを用いてリングオシレータ (Ring Oscillator; RO) を構成すれば，システムクロックと独立な発振波形を論理回路から生成することができる．Sunar ら [9] は，Free-run の RO を多数ならべ，それらの出力を xor で集約する TRNG について詳細に検討した (以下 Sunar 型とよぶ)．Schellekens ら [10] は，Xilinx Virtex II Pro FPGA (XC2VP30) で Sunar 型の TRNG を試作・評価し，インバータ 3 段の RO を 110 個用いた設計で 2 Mbps 以上の乱数生成速度を報告している．

Free-run の RO を利用した TRNG は，FPGA での実装に限ってみても，Sunar の他に幾つか提案されている．Kohlbrenner と Gaj [11] は，同じ構成の 2 つの RO の出力間の位相差を検出することで，ジッタに由来する乱数列を生成する TRNG を提案し，Xilinx Virtex XCV1000 で実装した．渡部と阿部 [12] は，Altera Cyclone FPGA を用いて，Sunar 型 TRNG において RO の配線長が乱数品質に及ぼす影響を検討した．Golić [13] は，Fibonacci ring oscillator (FIRO) と Galois ring oscillator (GARO) を用いた TRNG を提案し，Xilinx XC2V3000 で実装評価した．Dichtl と Golić [14] は，この設計を Xilinx XC3S200 で実装し，12.5 Mbps の生成速度を報告している．

Sunar 型 (および類似) の TRNG の問題点は，複数〜多数の RO が free-run で動作するため消費電力が大きいことである．例えば Schellekens の実装では，330 MHz 動作の Free-run RO が 110 個並列に動作しており，その消費電力は無視できない．特に組込み応用などでは，消費電力や放熱の面からこのような設計は避けることが望ましい．消費電力削減のために RO を停止させることも提案されているが [10]，乱数品質の低下が懸念される．逆に Dichtl ら [14] は，内部状態の初期化 (リスタート) により乱数系列を独立にすることを提案しているが，この場合はリスタートにより生成速度が大きく低下する．

本研究では，ラッチのメタスタビリティをエントロピー源とし，システムクロックに同期して動作する TRNG を提案する．乱数を生成しないときはクロックを停止することが可能で，消費電力も小さい．その動作原理については，次の章で詳しく述べる．

3 メタスタビリティを利用した TRNG

3.1 動作原理

一般に，ラッチや FF のセットアップ時間やホールド時間が満足されないと，メタスタビリティが発生して回路が誤動作する [15]．この問題を逆に利用して，メタスタビリティから真性乱数を生成する TRNG が研究されている (メタスタビリティ型)．

例えば RS ラッチの場合，R 入力と S 入力を同時にアサートすると，ラッチはメタステーブル状態 (準安定状態) になり結果は不定となる．図 1 の回路は，$Clk = 0$ のとき出力 $(Q, \bar{Q}) = (1, 1)$ で安定であるが，$Clk = 1$ では $(Q, \bar{Q}) = (0, 1)$ あるいは $(1, 0)$ で安定になる．すなわち Clk の立ち上がりエッジでラッチは準安定状態になり，いずれ $(0, 1)$ あるいは $(1, 0)$ という安定状態に遷移する．これはコイントスにより 1 ビットの情報を得ることに相当する．

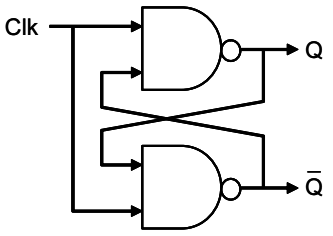

図 1: RS ラッチによる乱数生成.

安定状態への遷移は，平衡状態からの微小なずれ (例えばノイズ) を増幅することによって行われるので，遷移時間は一定でない．メタステーブル状態の持続時間が t_c 以上である確率 $P(t > t_c)$ は，一次モデルによる近似では以下の式で表される [15]．ここで τ は RS ラッチの時定数，A は NAND ゲートのゲインである．

$$P(t > t_c) = e^{-t_c/\tau} = e^{-(A-1)t_c/RC} \tag{1}$$

メタスタビリティ型 TRNG の乱数生成速度は，この遷移時間分布によって制約される．

メタスタビリティ型 TRNG の動作原理は単純であるが，高い乱数品質を得ることは容易ではない．例えば，図 1 の Clk 信号に配線遅延によるタイミングスキューがあると，出力 Q が 0 (あるいは 1) に偏る可能性がある．あるいは製造上のばらつきにより 2 つの NAND ゲートのドライブ能力に差があれば，出力に偏りが生じうる．回路のバランスが完全であったとしても，前回の出力に応じて内部のノード電圧に微小な差が残れば，次回の出力が前回の出力に影響を受けるため，乱数列に相関が生ずる可能性がある．

このような理由により，メタスタビリティ型 TRNG は実装が難しく，乱数生成速度や乱数品質に問題があるとされてきた [13].

3.2 メタスタビリティ型 TRNG の先行研究

メタスタビリティ型 TRNG の実装と評価は，主としてフルカスタム設計で行われてきた．これは回路設計の自由度が大きく，品質の良い TRNG を得やすいためと考えられる．

Bellido ら [16] は，RS ラッチのメタステーブル状態を利用して真性乱数を生成する方法を提案した．初期化

のためのクロックを別に用意し，プリチャージ・ディスチャージを行うなど，回路設計上の工夫を施している．2μm CMOS プロセスで試作した結果，一定品質の乱数が生成されることが確認された．

Kinniment と Chester [17] は，差動増幅器とラッチを組合わせた回路 (R-flop) をエントロピー源とした TRNG を提案し，0.6μm CMOS で実装した．この回路では，出力乱数列の 0 と 1 が偏らないように，ネガティブフィードバックにより R-flop のバイアス電圧を自動調整している．このため回路のばらつきやドリフトに対して強い．

Epstein ら [18] は，インバータ 2 個を MUX で接続し，発振状態からラッチ状態に切り替える際に発生するメタステーブル状態から真性乱数を発生する回路を提案した．回路は 0.18μm CMOS で試作・評価され，247 回路の出力を XOR した回路で Diehard テスト [5] をパスする乱数が得られた．この回路は論理規模が大きく，発振状態を利用しているので消費電力が大きい．

Holleman ら [19] は，ラッチのメタスタビリティを利用した TRNG を提案し，DC-nulling 型と FIR 型の 2 種の回路を 0.35μm CMOS で実装した．後処理後の出力乱数列は，NIST テスト [3][4] をパスした．

Tokunaga ら [20][21] は，ラッチのメタスタビリティを利用して乱数を生成する TRNG 回路を設計し，0.13μm CMOS で実装した．動作中にメタステーブル状態の解消時間 t_d を計測し，t_d の平均値が大きくなるようにラッチの初期状態を自動制御することにより，プロセス変動・温度変化・ノイズによる乱数品質の低下を防いでいる．t_d が一定以上のビットだけを取り出した場合，得られる乱数列は後処理なしで NIST テスト [3][4] に合格した．

本研究では，同期型論理回路によるメタスタビリティ型 TRNG を，FPGA によって実現する．メタスタビリティ型 TRNG の FPGA 実装は，本研究以外には 1 例しか報告されていない．

Danger ら [22] は，DFF の D 入力をクロックと同期して変化させることにより，セットアップ／ホールド時間違反によるメタステーブル状態から真性乱数を生成する TRNG を提案した．実装・評価は Altera Stratix FPGA で行われたが，FPGA に組み込まれた DFF はメタスタビリティに対する耐性があるため [23] [24] [25]，Danger らは LUT を用いて D latch を実装した．メタスタビリティを発生させるには D 入力とクロック入力のスキューを十分小さく保つ必要があるため，Danger らは手動でセルの配置を固定し，ワイヤ遅延を利用してタイミングを調整している．このように非常に微妙な調整を手動で施しているため，結果の再現性，多品種への移植性，汎用性に問題を残している．

本研究はメタスタビリティを利用する点で Danger と共通しているが，RS ラッチのメタスタビリティを用いて，アナログ的遅延に頼らない実装を提案する．

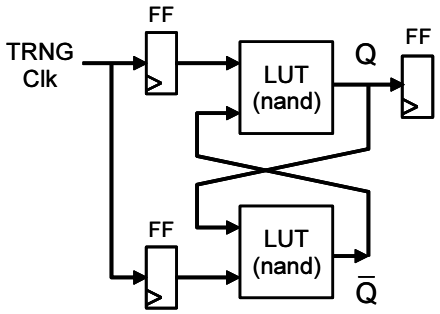

図 2: FPGA を用いた RS ラッチの実現 (LUT latch).

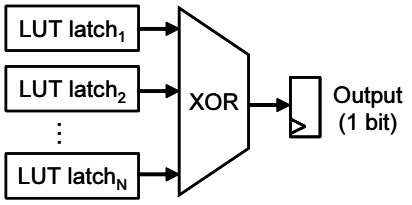

図 3: 提案回路 LUT latch N の全体図.

4 実装

4.1 提案回路の構成

本章では，提案した TRNG を Xilinx Virtex4 FPGA [26] に実装する方法について，その概要を説明する．[1]

今回の実装では，RS ラッチを図 2 のように実装した (以下 LUT latch と呼ぶ)．RS ラッチを構成する 2 つの nand ゲートは，それぞれ LUT (Look-Up Table) で実現されている．3 つの FF は slice に埋め込まれた FF (FFX, FFY) で実装される．TRNG Clk はシステムクロックを m 分周して生成しており，m を変更することにより TRNG の動作周波数を調整できるように設計されている．

現実の LUT latch は出力に偏りがあるため，単体では良い TRNG にならない．そこで複数の LUT latch から出力 (乱数) を生成する回路が必要になる．偏りのある物理乱数系列 (複数) から良い乱数系列を生成することは実用上非常に重要であるため，古くから多数の研究が行われてきた．最も有名で広く使われる補正回路の一つが，XOR corrector [27] である．本研究でも，N 個の LUT latch の出力を XOR で 1 bit に集約して出力とする．この回路を以下 LUT latch N と呼ぶ (図 3).

図 3 の LUT latch を free-run のリングオシレータに置き換えれば，容易に Sunar 型の TRNG を得ることができる．提案回路と Sunar 型 TRNG を同じ実装技術で比較するため，Schellekens ら [10] と同じ構成の TRNG を実装

[1] 本実装の詳細については，RECONF/VLD/CPSY 研究会『FPGA 応用および一般』(2009 年 1 月 29 日) において発表する「メタスタビリティを利用した真性乱数生成回路の FPGA による実装」(畑尚志, 市川周一) を参照されたい.

した．即ち，インバータ 3 段で構成される free-run RO を 110 個並列に動作させ，それらの出力を XOR で集約して FF でサンプルする．この回路を，以下 Ring osc 110 と呼ぶ．

4.2 実装上の工夫と考察

3 章でも述べた通り，メタスタビリティ型 TRNG から乱数を得ることは容易でない．そのため提案回路では，以下のような実装上の工夫を施した．

まず LUT latch (図 2) はハードマクロとして実現している．Virtex4 の 1 slice には LUT 2 つと FF 2 つが含まれており，LUT latch を実装するには 2 slice 必要である．配線の都合上 2 つの LUT (nand) は異なる slice に置かれるが，最適な結果を得るために，2 つの slice の位置関係をハードマクロ化により固定した．さらに Q と $\bar{Q}$ の配線負荷も均等化されるよう考慮している．これを自動配置配線に委ねると，十分なエントロピーが得られず，さらに結果の再現性が失われることが実験的に判明している．

クロックスキューを最小化することも重要である．図 2 において，LUT の入力側 FF (2 個) は論理的には不要である．この 2 つの FF はクロックスキュー防止のために設けられており，ハードマクロ内の FF を利用して，クロックエッジから最小時間で LUT に伝播するよう考慮されている．この FF により生成エントロピーが増加し，評価結果の再現性が高まることが実験で確認されている．

LUT latch 出力側の FF は，XOR 回路への配線負荷を，内部ノード Q の配線負荷と分離するために設けられている．$\bar{Q}$ には FF がないため Q と $\bar{Q}$ の配線負荷に差があるように見えるが，FPGA においては利用しない資源 (FF 等) も存在しているので配線負荷は均等である．

今回の実装では，簡単のため TRNG Clk の Duty 比は 50%に固定した．しかし $Clk = 0$ の期間は Q と $\bar{Q}$ のプリチャージによる初期化時間であり，一方 $Clk = 1$ の期間は準安定状態から安定状態への遷移時間である．それぞれ別々な物理過程に対応しているため，同じ時間にこだわる必然性はない．最適な Duty 比を探すことにより，生成速度が更に改善される可能性がある．

Duty 比を固定した場合でも，TRNG Clk の周期 (サンプリング周期) は，実験的に決める必要がある．3 章で説明した通り，メタスタビリティの解消時間は時定数とゲインに依存する．サンプリング周期を平均解消時間より小さくすると，収穫できるエントロピーが減少し，乱数品質や乱数生成速度が低下する．一方，乱数生成速度はサンプリング周波数に依存するので，サンプリング周期を長くすれば乱数生成速度は低下する．従って実験的に最適なサンプリング周期を求めることが必要になる．

TRNG で生成されるエントロピーを増やすには，LUT latch の個数を増やすか，各 LUT latch の実装を改善する必要がある．既に本節で述べたように，配線負荷の均等化やスキューの低減により，各 LUT latch を一定の範囲で改善することはできる．しかしプロセスのばらつきに由来する回路のアンバランスは，回路設計や実装で回避することができない．例えば LUT 特性の不均衡によりエントロピーが収穫できない LUT latch は実際に存在する．そこで本研究では，複数の LUT latch の出力を XOR することにより安定した乱数出力を生成する．また，各 LUT latch の配置は自動配置配線に任せて，チップ内に分散配置することにより，エントロピーの増大を図っている．[2]

表 1: 論理規模 (Slice).

Design	TRNG	System
LUT latch 64	145	7013
LUT latch 128	290	7159
LUT latch 256	580	7447
Ring osc 110	359	7219

5 評価

TRNG の評価には，Xilinx XC4VFX20 FPGA を搭載する Xilinx ML405 ボードを使用した．CAD には ISE 10.1.03i を使用している．乱数品質の評価には 1 Gbit のデータが必要になるため，XC4VFX20 に搭載されている PowerPC 405 コアを用いて組込み Linux システムを構成した．TRNG は周辺回路としてシステムに接続し，取得した乱数データはコンパクトフラッシュカードに集積している．PowerPC の動作クロックは 300 MHz，周辺回路のバスクロックは 100 MHz である．

各設計の論理規模を表 1 にまとめる．TRNG は乱数生成部のみ，System は (TRNG を含む) 組込み Linux システム全体を示している．XC4VFX20 のスライス数は 8544 なので，LUT latch 64 の TRNG 部はチップの 1.7%，System 全体ではチップの 82%に相当する．LUT latch 64 の TRNG 部は，Ring osc 110 の 40%程度と小型である．

表 2 は，NIST テスト [3][4] の結果である．ここでは LUT latch 64 と Ring osc 110 のいずれも，サンプリング周期を 320 ns として乱数データを収集した．NIST テストのバージョンは 1.8，設定はデフォルトとし，データ長 10^6 bit のテストを 1000 回行っている．表中，Proportion の項は，1000 回のテストにおける合格率を示す．評価項目の多いテストは，合格数／テスト項目数という形で結果をまとめた．不合格項目は太字で，テストが不成立の項目は N/A で示されている．

[2] 簡単な実験を試みたところ，同数の LUT latch であれば，集中配置するより分散配置するほうが乱数品質が向上した．

表 2 から明らかなように，LUT latch 64 は NIST テストに全項目で合格している．一方，Ring osc 110 は多くの項目で不合格となっているが，この結果だけで Sunar 型に問題があるとはいえない．Schellekens ら [10] の設計パラメータが，今回の実装環境に適していなかったと考えられる．リングオシレータの段数，個数，サンプリング周期などを実装環境に合わせて最適に調整すれば，乱数品質が向上する可能性はある．過去の研究で「メタスタビリティ型は再現性や信頼性に乏しい」と批判されてきたが，Sunar 型も必ずしも再現性や信頼性に優れるとはいえないことがわかる．

提案した TRNG の最大生成速度を調べるため，様々なサンプリング周期で乱数列を生成して，NIST テストに合格する最短周期を求めた．LUT latch 64, 128, 256 の最大乱数生成速度は，表 3 に示す通りである．先行研究と比較するため，FPGA で実現された TRNG の乱数生成速度を，設計のタイプごとに表 3 にまとめた．本研究で提案したメタスタビリティ型 TRNG は，近年の先行研究と比べても遜色ない生成速度を実現していることが確認できる．

ただし，各設計は実現技術 (デバイス) や論理規模が異なるため，一律に生成速度を比較することは適切でない．例えば，独立な TRNG を 2 つ並列に動作させれば生成速度は 2 倍になりうるので，論理規模を無視して生成速度だけを比較することは無意味である．我々の実装でも，LUT latch 256 の生成速度が 12.5 Mbps であるのに対し，LUT latch 128 を 2 個並列に使えば同じ論理規模で 16.7 Mbps の生成速度を得ることができる．[3]

6 おわりに

本研究では，RS ラッチのメタスタビリティを利用した TRNG を FPGA で実装・評価した．提案回路は，同期式デジタル回路だけで構成されており，後処理なしで NIST テストに合格する．RS ラッチ 128 個の回路で，論理規模は 290 スライスと小さく，生成速度は 8.3 Mbps と先行研究と比べて遜色ない．提案した回路は随時クロックを停止することが可能なので，低消費電力を求められる組込み応用にも対応可能である．

本研究で提案回路の基本動作は確認されたが，実応用に適用するには，さらに詳細な検討を進める必要がある．

まず，電源電圧や動作温度の変化に対する，乱数品質と生成速度の評価が必要である．しかし提案回路は同期式デジタル回路であり，回路や配線のアナログ的遅延に依存しないことから，電圧や温度の変動による影響は小さいと予想される．

[3] 並列に乱数を生成する場合，実装上の理由で TRNG 間に相関が発生する可能性があるので，実装上の配慮と検証が必要である．

他の実装技術での実装・評価も必要である．提案回路の構成は極めて単純であり，論理ゲート (LUT) と FF だけで構成されているため，FPGA の品種や製造会社によらず実装可能と考えられる．当然，ゲートアレイやフルカスタム LSI で実装できる．乱数品質を確保するには実装環境に合わせた設計パラメータの決定が重要であるが，本研究でサンプリング周期の決定方法など体系的なアプローチを提示したので，比較的容易に実現可能であると考えている．

謝辞

本研究の一部は，科学研究費補助金・基盤研究 (C)19500042 により行われた．

参考文献

[1] R. Fairfield, R. Mortenson, and K. Coulthart, “An LSI random number generator (RNG),” Proc. CRYPTO 1984, pp.203–230, Springer-Verlag, 1985.

[2] K. Tsoi, K. Leung, and P. Leong, “Compact FPGA-based true and pseudo random number generators,” Proc. IEEE FCCM 2003, pp.51–61, IEEE CS, 2003.

[3] NIST, “Statistical test suite,” August 2008. http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html.

[4] A. Rukhin et al., “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” NIST Special Publication 800-22 (with revisions dated May 15, 2001).

[5] G. Marsaglia, “The Marsaglia random number CDROM including the Diehard battery of tests of randomness,” October 2008. http://www.stat.fsu.edu/pub/diehard/.

[6] V. Fischer and M. Drutarovský, “True random number generator embedded in reconfigurable hardware,” Proc. CHES 2002, LNCS 2523, pp.415–430, Springer, 2002.

[7] V. Fischer et al., “High performance true random number generator in Altera Stratix FPLDs,” Proc. FPL 2004, LNCS 3203, pp.555–564, Springer, 2004.

[8] S.H.M. Kwok and E.Y. Lam, “FPGA-based high-speed true random number generator for cryptographic applications,” Proc. IEEE TENCON 2006, pp.1–4, IEEE, 2006.

[9] B. Sunar, W.J. Martin, and D.R. Stinson, “A provably secure true random number generator with built-in tolerance to active attacks,” IEEE Transactions on Computers, vol.56, no.1, pp.109–119, 2007.

[10] D. Schellekens, B. Preneel, and I. Verbauwhede, “FPGA vendor agnostic true random number generator,” Proc. FPL 2006, pp.1–6, IEEE, 2006.

[11] P. Kohlbrenner and K. Gaj, “An embedded true random number generator for FPGAs,” Proc. ACM/SIGDA FPGA 2004, pp.71–78, ACM, 2004.

[12] S. Watanabe and K. Abe, “True random number generation on FPGA,” Proc. SCIS 2007, 3E1-5, IEICE, 2007.

[13] J.D. Golić, “New methods for digital generation and postprocessing of random data,” IEEE Transactions on Computers, vol.55, no.10, pp.1217–1229, 2006.

表 2: NIST テストの結果．太字は不合格項目を意味する．

	LUT latch 64		Ring osc 110	
	P-value	Proportion	P-value	Proportion
frequency	0.907419	0.9930	**0.000000**	**0.0000**
block-frequency	0.576961	0.9870	**0.000000**	**0.0030**
cumulative sums-up	0.532132	0.9930	**0.000000**	**0.0000**
cumulative sums-down	0.175691	0.9930	**0.000000**	**0.0000**
runs	0.779188	0.9900	**0.000000**	**0.0000**
longest-run	0.670396	0.9870	**0.000000**	**0.5260**
rank	0.701366	0.9890	0.739918	0.9890
fft	0.128874	0.9880	0.006107	0.9840
nonperiodic-templates	148/148	148/148	**38/148**	**41/148**
overlapping-templates	0.846338	0.9870	**0.000000**	**0.0000**
universal	0.632955	0.9900	**0.000000**	**0.9140**
apen	0.906069	0.9860	**0.000000**	**0.0000**
random-excursions	8/8	8/8	N/A	N/A
random-excursions variant	18/18	18/18	N/A	N/A
serial1	0.680755	0.9890	**0.000000**	**0.4760**
serial2	0.972382	0.9950	0.038565	0.9900
linear-complexity	0.516113	0.9880	0.484646	0.9920

表 3: FPGA による TRNG の乱数生成速度

Design category	Reference	Device	[Mbps]
Oscillator sampling	Tsoi et al. [2]	Xilinx XCV300E	0.03
PLL/DLL	Fischer, Drutarovský [6]	Altera EP20K200E	0.07
	Fischer et al. [7]	Altera EP1S25	1.
	Kwok, Lam [8]	Xilinx XC2VP20	6.
Free-run oscillator	Kohlbrenner, Gaj [11]	Xilinx XCV1000	0.5
	Schellekens [10]	Xilinx XC2VP30	2.5
	Dichtl, Golić [14]	Xilinx XC3S200	12.5
D latch metastability	Danger et al. [22]	Altera EP1S25	20.
RS latch metastability	(LUT latch 64)	Xilinx XC4VFX20	3.85
	(LUT latch 128)	Xilinx XC4VFX20	8.33
	(LUT latch 256)	Xilinx XC4VFX20	12.5

[14] M. Dichtl and J.D. Golić, “High-speed true random number generation with logic gates only,” Proc. CHES 2007, LNCS 4727, pp.45–62, Springer, 2007.

[15] L. Kleeman and A. Cantoni, “Metastable behavior in digital systems,” IEEE Design and Test of Computers, vol.4, no.6, pp.4–19, 1987.

[16] M. Bellido et al., “Simple binary random number generator,” Electronics Letters, vol.28, no.7, pp.617–618, 1992.

[17] D. Kinniment and E. Chester, “Design of an on-chip random number generator using metastability,” Proc. ESSCIRC 2002, pp.595–598, IEEE SSCS, 2002.

[18] M. Epstein et al., “Design and implementation of a true random number generator based on digital circuit artifacts,” Proc. CHES 2003, LNCS 2779, pp.152–165, Springer, 2003.

[19] J. Holleman et al., “A 2.92μw hardware random number generator,” Proc. ESSCIRC 2006, pp.134–137, IEEE, 2006.

[20] C. Tokunaga, D. Blaauw, and T. Mudge, “True random number generator with a metastability-based quality control,” Proc. IEEE ISSCC 2007, pp.404–405, IEEE, 2007.

[21] C. Tokunaga, D. Blaauw, and T. Mudge, “True random number generator with a metastability-based quality control,” IEEE Journal of Solid-State Circuits, vol.43, no.1, pp.78–85, 2008.

[22] J.L. Danger, S. Guilley, and P. Hoogvorst, “Fast true random generator in FPGAs,” Proc. IEEE NEWCAS 2007, pp.506–509, IEEE, 2007.

[23] Altera, Metastability in Altera Devices, May 1999. Application Note 42.

[24] Xilinx, Metastability Considerations, January 1997. XAPP077.

[25] P. Alfke, Metastable Recovery in Virtex-II Pro FPGAs. Xilinx, February 2005. XAPP094.

[26] Xilinx, Virtex-4 FPGA User Guide, June 2008. UG070 (v2.5).

[27] R. Davies, “Exclusive OR (XOR) and hardware random number generators,” February 2002. `http://www.robertnz.net/pdf/xor2.pdf`.