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

FPGA Implementation of Metastability-based True Random Number Generator

豊橋技術科学大学 大学院 工学研究科
2008年 修士(工学)

知識情報工学専攻 市川研究室
043732 畑 尚志

2009　年　1 月　9日

知識情報 工学専攻	学籍番号	043732
申請者氏名	畑 尚志	

指導教員氏名	市川 周一

論　文　要　旨(修士)

論文題目	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

多くのセキュリティ技術は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生成器(True Random Number Generator; TRNG)は，セキュリティの基盤技術として重要である．熱雑音など物理乱数を利用した TRNG が実用化されているが，実装にアナログ回路が必要であるため論理 LSI への集積が難しい．デジタル回路だけで TRNG を構成できれば論理 LSI へ集積可能になるため有用である．

デジタル回路で実装可能な TRNG として，メタスタビリティを利用する回路が提案されている．メタスタビリティを利用した TRNG は乱数品質の維持が難しく，カスタム LSI で実装された例がほとんどである．本研究では RS ラッチのメタスタビリティを利用した TRNG (ラッチ型 TRNG)を FPGA (Field Programmable Gate Array)で実装し，その乱数品質と生成速度を報告する．FPGA は RAM (LUT)とスイッチで構成される LSI で，回路構成に制限がある．そのため，FPGA で実装可能であればカスタム LSI でも実装可能である．また，FPGA は近年広く利用されているため，FPGA で実装可能であることはそれ自体が実用的価値が高い．

一般に，ラッチのセットアップ時間やホールドタイム違反が起こると，メタスタビリティが発生する．メタスタビリティが発生したラッチは準安定状態から，1 もしくは 0 の安定状態に移行する．メタスタビリティを利用した TRNG ではこれを利用して乱数を生成する．本研究では RS ラッチのメタスタビリティを利用したラッチ型 TRNG を実現する．FPGA に実装したラッチから高い品質の乱数を得ることは難しいため，複数のラッチの出力を XOR することによって乱数品質を維持する．また，少ない数のラッチで TRNG を実現するためにラッチに実装上の工夫を施す．

作成した TRNG は Xilinx Virtex4 FPGA XC4VFX20 に実装し，NIST テストに後処理なしで通過することを確認した．ラッチ 128 個からなる TRNG で，回路規模 290 Slice，生成速度 8.33 Mbps を実現した．

Department of Knowledge-based Information Engineering	Registration	043732
Name	Hisashi Hata	

Graduate Adviser	Shuichi Ichikawa

Abstract

Title	FPGA Implementation of Metastability-based True Random Number Generator

Metastability of RS latch is utilizable as an entropy source for true random number generators (TRNG). This kind of TRNG is comprised of logic gates, which can be integrated into a logic LSI. Though latch-based TRNG has been mostly implemented with full-custom LSI technology, this study presents an implementation with common FPGA technology.

The RS latch in our TRNG is implemented as a hard-macro to guarantee the quality of randomness, minimizing the clock skew and load imbalance of internal nodes. The quality and throughput are further improved by XOR'ing the output of 32--128 latches. The derived design was implemented with Xilinx Virtex4 FPGA (XC4VFX20), and passed NIST test without post-processing. A TRNG of 128 latches occupies 290 slices, while achieving 8.33 Mbps throughput.

目次

1　はじめに

多くのセキュリティ技術は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生成器 (True Random Number Generator; TRNG) は，セキュリティの基盤技術として重要である．熱雑音など物理現象を利用した TRNG も実用化されているが，実装にアナログ回路が必要であるため論理 LSI への集積が難しい．デジタル回路で TRNG を構成できれば，論理 LSI に集積可能であるため，応用範囲も広く実用的価値が高いと考えられる．

デジタル回路で構成可能な TRNG として，メタスタビリティを利用する回路が提案されている．しかしメタスタビリティを利用した TRNG は実装が難しいため，カスタム LSI で実装した例がほとんどである．本研究では，RS ラッチのメタスタビリティを利用した TRNG (ラッチ型 TRNG) を FPGA で実装し，その乱数品質と生成速度を報告する．FPGA で実装するための回路設計上の工夫と，最適な設計パラメータを決定するための実験方法についても述べる．提案回路は同期式デジタル回路であるため，FPGA に限らず全ての論理 LSI で実装可能である．

本稿では, ラッチ型 TRNG の FPGA への実装と評価について報告を行う．まず 2 章で先行研究についてのべ, 3 章で基本原理について解説する．4 で評価方法について述べ, 5, 6 章で実装と評価を行う．最後に 7 章でまとめと今後の課題について述べる．

2　関連研究

デジタル回路で TRNG を構成する代表的な手法として, 発振器のジッタを利用する方法と, メタスタビリティを利用する方法が挙げられる．

2.1　発振器のジッタを利用する方法

理想的な発振器は, 出力は常に同じ周期の信号を出力すると規定されているが, 現実の発振器出力には時間軸上の微小なずれ (ジッタ) が含まれる．発信機のジッタを利用する方法ではこれを利用して TRNG を実現する．代表的な手法としてリングオシレータ (Ring Oscillator; RO) を利用する手法がある．Sunar ら[2] は, 多数の RO 出力を xor で集約する手法を提案した (Sunar 型) ．Schellekens ら[3] は Sunar 型 TRNG を Xilinx Virtex II Pro FPGA (XC2VP20) で実装し, 2Mbps 以上の乱数生成速度を報告している．しかし, Schellekens らの実装では，330MHz

で動作する free-run の RO が 110 個並列で動作しており，回路規模と消費電力の面で問題がある．Fischer らは[4)5)] は PLL のジッタを利用する TRNG を提案した．Altera APEX EP20K200E と Stratix EP1S25 の PLL を利用して TRNG を構成し, EP20K200E で 0.07 Mbps, EP1S25 で 1M bps の乱数生成速度を報告した．Fischer らの手法は PLL を必要とするため PLL が搭載された FPGA でなければ実装できない．

2.2 メタスタビリティを利用する方法

一般に，ラッチや FF のセットアップ時間やホールドタイム違反が起こると，メタスタビリティが発生する[11)]．メタスタビリティが発生した回路はメタステーブル状態 (準安定状態) になった後, 0 もしくは 1 の安定状態へと移行する．どちらの状態に移行するかは予測することはできない．メタスタビリティを利用する方法ではこれを利用して乱数を生成する．

Belido ら[6)] は RS ラッチのメタスタビリティを利用した TRNG を提案した．ラッチの初期化など回路設計上の工夫を施した TRNG を 2μm CMOS プロセスで試作し一定の乱数品質を確認した．Kinniment ら[8)] は差動増幅器とラッチを組み合わせた TRNG を提案した．これは出力が偏らないようにネガティブフィードバックを用いて, ラッチのバイアス電圧を自動調整するように工夫を施したものである．Epstein ら[7)] はインバータと MUX で構成される, 発信器とラッチを組み合わせた回路を提案した．この回路は発信とラッチ状態を有し, 発信状態からラッチ状態に切り替えることでメタスタビリティが発生する．Epstein らはこれの出力を 247 個 xor した TRNG を 0.18 μm CMOS で実装し, Diehard テスト[12)] に通過することを報告した．Tokunaga らはラッチのメタスタビリティを利用した TRNG を提案した．Tokunaga ら[9)] の回路は動作中にメタステーブルの持続時間を測定し, その平均時間が長くなるように調整される工夫が施されている．これを 0.13μm COMS で実装し, 後処理なしで NIST テストに合格することを報告した．

メタスタビリティを利用した TRNG は乱数品質の維持が難しくカスタム LSI での報告がほとんどである．FPGA での実装例は 1 例だけ報告されている．Danger[10)] らは D ラッチのセットアップホールドタイム違反を起こすことによってメタスタビリティを発生させる TRNG を提案した．FPGA の DFF にはメタスタビリティ耐性があるため LUT で実装した D ラッチを用いて TRNG を構成している．セットアップホールドタイム違反を起こすためにはスキューの小さい

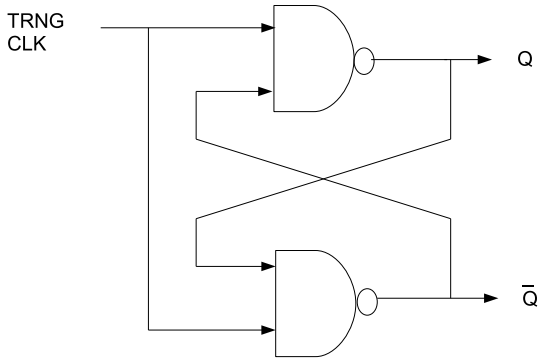


図 1　RS ラッチによる乱数生成

信号を D 端子とクロック端子に入力する必要がある. Danger らはこれをグローバルクロック配線と一般配線の配線遅延の差を利用して実現した. Danger らの実装はアナログ的な遅延に依存するため再現性, 移植性に問題がある.

本研究では RS ラッチのメタスタビリティを利用してアナログ的な遅延に頼らない FPGA に実装可能な TRNG を提案する.

3　FPGA を用いた乱数生成

3.1　ラッチのメタスタビリティ

一般に，ラッチのセットアップ時間やホールドタイム違反が起こると，メタスタビリティが発生する[11)]．例として，RS ラッチの R 入力と S 入力を同時にアサートする回路を図 1 に示す．図 1 の回路は，TRNG CLK=0 では出力 $(Q, \bar{Q}) = (1, 1)$ で安定状態となるが，TRNG CLK=1 では $(Q, \bar{Q}) = (0, 1)$ あるいは $(1, 0)$ で安定となる．実際には，ラッチは TRNG CLK の立ち上がりエッジでメタステーブル状態 (準安定状態) になり，その後，いずれかの安定状態へ遷移する．これは (理想的には) コイントスにより 1 bit の乱数を得ることに相当する．安定状態への遷移時間は一定でなく，確率分布をもつ[11)]．

上に述べた TRNG の動作原理は単純であるが，実際に高品質の乱数を生成することは容易でない．例えば TRNG CLK 信号にスキューがある場合，ラッチの出力が 0 もしくは 1 に偏る可能性がある．2 つの NAND ゲートのドライブ能力に差がある場合も同様である．スキューやドライブ能力差がない場合でも，$Q, \bar{Q}$ 間に電位差がある状態で TRNG CLK=1 になれば，出力に偏りが生じう

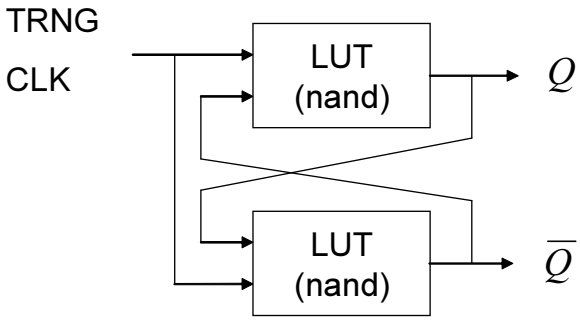


図 2　LUT を使用した RS ラッチ

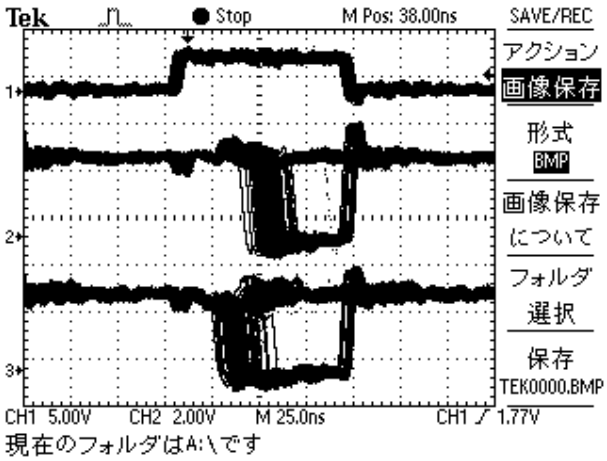


図 3　RS ラッチのメタスタビリティ

る．$Q, \bar{Q}$ の電位は TRNG CLK=0 の期間に初期化されるため，TRNG CLK はアサート期間だけでなくネゲート期間も十分にとる必要がある．

3.2　FPGA のラッチ

このようにラッチ型 TRNG の実現には注意深い回路設計が必要であるため，多くの先行研究ではカスタム LSI で実装評価している．従って，FPGA 上のラッチを利用して良い乱数が生成できるかどうかは，必ずしも自明でない．そこで予備実験として，FPGA 上に図 1 のような回路を実装し，出力を観察した．

観測には Xilinx Virtex4 FX20 FPGA を使用し，NAND ゲート 1 個に対して 4 入力 LUT 1 個を割り当てた．遷移時間の観測を行うため，出力は FF を通さず直接汎用ピンに出力している．IO バッファの出力電圧設定は LVCMOS 3.3V とし，Tektronix の TDS2024B で観察した．ラッチの個体毎にメタスタビリティの発生状況が違うため，多くのラッチを実装した中から，メタスタビリティを

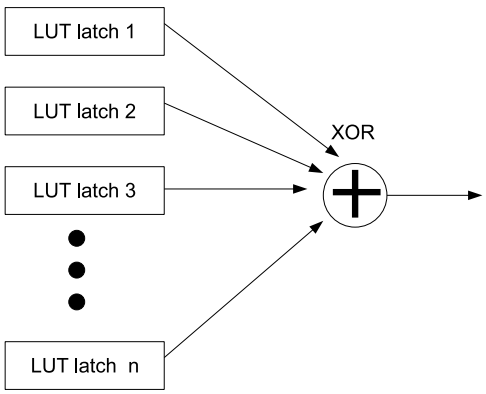


図 4　LUT latch n

頻繁に発生する RS ラッチを選択して観測した.

観測波形を図 3 に示す．波形は上から TRNG CLK，$Q, \bar{Q}$ を表し，5 秒間の波形を重ねて表示している．オシロスコープの掃引時間は 25 ns/div とした．IO バッファを通しているため，遷移中の中間電位の観測はできなかったが，周期毎に Q の値がランダムに変化することや，準安定状態からの遷移時間が一定でないことが観測できた．観測された遷移時間は数 10 ns に達する場合があり，乱数生成速度の制約要因になりうることがわかった.

このように FPGA の RS ラッチでもメタスタビリティを観測できるが，ラッチの個体差は大きく，出力が 0 または 1 に固定されているものも多い．ラッチの生成エントロピーの個体差については，5.5 章で評価結果を示す.

実用的には，複数ラッチの出力を xor で集約し，TRNG の出力を生成すればよい．これは XOR コレクターとして古くから用いられているものである．例えば, 0 もしくは 1 が出現する確率 (生起確率) が μ である乱数列 A, B を xor すると その出力 C の生起確率は $2\mu(1-\mu)$ になる．A, B の生起確率が 90%であるならば C の生起確率は 18%になる．xor する前が 40%の偏りであったのに対し, xor 後は 32%の偏りに改善されている．XOR コレクターでは, これを繰り返すことで乱数品質の改善を行う．以下，n 個の RS ラッチを使用する TRNG を **LUT latch** n と表記する.

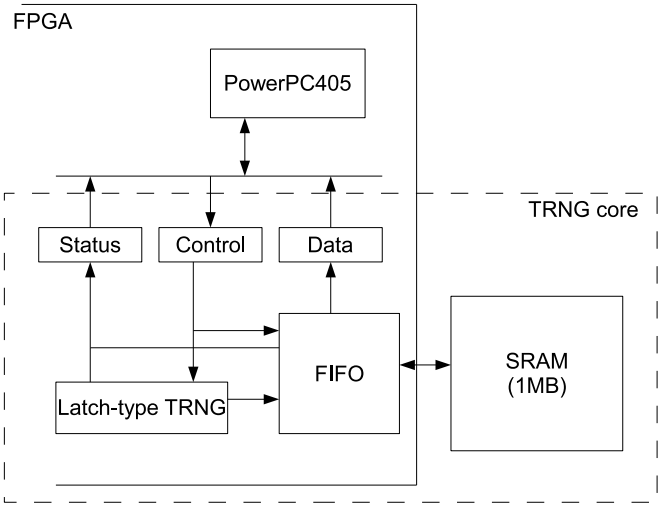


図 5　TRNG のデータ取得

4　評価環境

4.1　評価システム

実装評価には，Xilinx Virtex4 FX20 (XC4VFX20) を搭載する XilinxML405 ボードを使用した．CAD は Xilinx ISE 10.1.03 を使用し，パラメータはデフォルトで使用した．乱数の取得は，FPGA に搭載されている PowerPC 405 (PPC) に TRNG を周辺回路として接続して行う．測定システムは PPC に UART, System ACE[⋆1], Ethernet MAC, DDR SDRAM, TRNG を接続した構成とした．乱数検定に大量 (10^9 bit) のデータが必要になるため，OS に組込み Linux を使用している．Linux のファイルシステムはコンパクトフラッシュに置かれており，取得した乱数データはコンパクトフラッシュに蓄積される．クロックは PPC が 300MHz，バスクロックが 100MHz である．

TRNG コアはバッファ(1MB SRAM) とラッチ型 TRNG で構成される (図 5) ．CPU のインターフェースとしてデータ, ステータス, コントロールレジスタが用意されている．ラッチ型 TRNG はバッファを介して PPC に接続されており, PPC が直接ラッチ型 TRNG にアクセスすることはできない．

バッファは SRAM を利用した簡易 FIFO として実装した．データアクセスは

⋆1 コンパクトフラッシュインターフェース．

TRNG 側からはライト, PPC 側からはリードが許されており, 許可されていないアクセスは無視する．リードとライトを同時に実行することは想定していないため, TRNG が FIFO にアクセスしている間は PPC がアクセスすることはできない. ステータスレジスタはリードアクセスのみ許可されており, バッファの状態とラッチ型 TRNG の動作状況を観察することができる. コントロールレジスタはライトアクセスのみ許可されておりバッファのクリア, TRNG CLK の周期, 測定開始を設定することができる.

TRNG CLK はバスクロックを m 分周して作成する．実験の簡素化のため, 本研究では TRNG CLK のデューティ比を 50%に固定している．従って TRNG CLK の周期は $(20 \times m)$ ns となる．TRNG CLK はクロック専用線を使用せずに一般配線によって配線される.

測定を行う前に PPC はバッファのクリア, TRNG CLK の設定を行う. PPC がコントロールレジスタに測定開始のフラグを書き込むとラッチ型 TRNG はバッファを満たすまで出力を行う. PPC は，ポーリングでバッファの状態を監視し, バッファが満たされるとバッファからデータを読み出しファイルに保存する．ラッチ型 TRNG はバッファが満たされると乱数の生成を停止する. 1 MB 以上の出力を生成する場合は，1 MB の計測を複数回繰り返す.

4.2 テスト方法

本研究では 2 つの代表的テストを用いて，設計段階の比較検討と最終的な品質評価を行う．Diehard テスト[12] は必要なデータ量が小さく短時間でテストできるため，設計上の試行錯誤やパラメータ決定に最適である．一方，NIST テスト[13] は網羅的で大量のデータを必要とするため，TRNG の最終評価に使用する.

4.2.1 Diehard テスト

Diehard テストは Marsaglia が提案した乱数テストセットである．本研究では Diehard テスト v0.2 beta[14] を使用した．テストへ入力する乱数のサイズは 10MB にした. Diehard v0.2 beta では 17 系列のテストが提供されているが，そのうち GCD，Gorilla, Overlapping permutation テストは 10 MB 以上の乱数を必要とするため，除外して用いた.

乱数の評価は p value によって行う．p value は理想的な乱数ならば [0,1] の区間に一様分布するとされる値で，Diehard テストは 10 MB の乱数を入力すると 229 個の p value を出力する．Diehard テストでは p value の評価基準を設けてい

表 1　Diehard テストの構成

name	number of p value
Birthday Spacings	11
Ranks of 31x31 and 32x32 matrices	2
Ranks of 6x8 Matrices	26
Monkey Tests on 20-bit Words	20
Monkey Tests OPSO,OQSO,DNA	82
Count the 1‘s in a Stream of Bytes	1
Count the 1‘s in Specific Bytes	25
Parking Lot Test	11
Minimum Distance Test	11
Random Spheres Test	21
The Sqeeze Test	1
Overlapping Sums Test	11
Runs Up and Down Test	3
The Craps Test	4

ないので，本研究では，0.01 <pvalue< 0.99 の範囲に収まる値がいくつあるかで評価した．理想的な乱数であれば，225 個前後の p value がテストに合格する．入力データのエントロピーが低くすべてのテストが実行できない場合は，そのテストを除外して評価した．テストが出力する p value の数は系列ごとに異なる (表 1) . テストの系列によって p value の数に違いがあるため, 合格する p value の数を比較するだけでは, TRNG の良し悪しを議論することは出来ない. また, 入力する乱数の量が小さいため, 十分な乱数品質の評価も出来ないが，悪い設計を排除することは可能なので，設計時の簡便な比較評価には有用である．

4.2.2　NIST テスト

NIST テスト[13)15)] は，National Institute of Standards and Technology が提案する乱数検定テストで，15 種, 187 項目のテストから構成されている (表 2)．本研究ではバージョン 1.8 を使用した. NIST テストでは 1 回のテストに 10^6 bit 程度の乱数を必要とし, 1 回のテストで p value を 1 つ出力する．NIST は 10^6 bit のテストを 1000 回以上行い, それによって得られた 1000 個以上の p value を評

表 2　NIST テストの構成

name	number of p value
frequency	1
block-frequency	1
cumulative-sums	2
runs	1
longest-run	1
rank	1
fft	1
nonperiodic-templates	148
overlapping-templates	1
universal	1
apen	1
random-excursions	8
random-excursions-variant	18
serial	1
linear-complexity	1

価することを推奨している.

Diehard テストでは複数のテストが出力する p value を区別無く評価したが, NIST テストでは項目ごとに p value を管理する. 同系列のテストであってもテスト方法が違う場合は個別に評価される. 例えば, nonperiodic-templates テストは 10^6 bit の入力に対しパラメータを変えながら 148 回テストを行う. そのため 148 個の p value を出力するがそれらはパラメータごとに別の項目として管理され, 項目ごとに評価される.

乱数の評価は 1000 個以上の p value の分布と合格率を評価することで行われる. 分布と合格率の評価方法は NIST の基準に従った.

本研究ではテストに入力する乱数のサイズを 10^6 bit, テスト回数を 1000 回, テストのパラメータをデフォルトに設定して評価を行い, 187 項目すべてに合格するものを NIST テストに合格したと判定した.

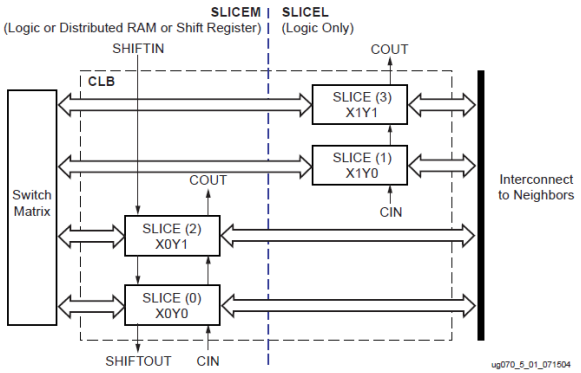


図 6　Virtex4 CLB[16)]

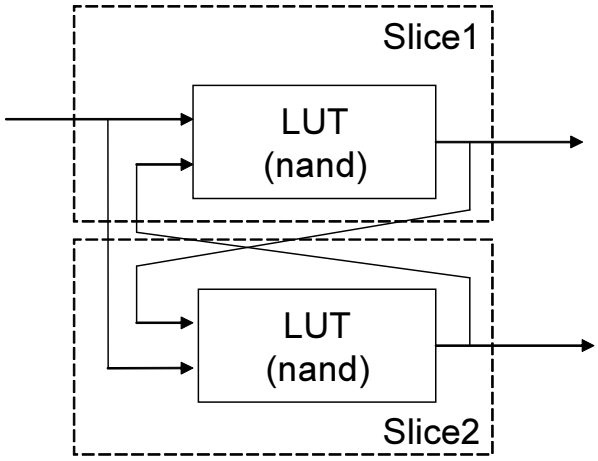


図 7　ラッチの割り当て

5　ラッチの評価

5.1　Virtex4 の構成

FPGA の構造は品種により様々であるが，論理資源と配線資源を基本とする点は共通である．論理資源は階層化されており，論理演算は最終的に LUT (look-up table) で実現される．Xilinx Virtex4 FPGA の場合，4 入力 LUT と FF のペア 2 組で 1 Slice が構成され，4 Slice を 1 組として CLB が構成される (図 6)．CLB 内の Slice のうち，偶数列は Slice L，奇数列は Slice M が配置されている．Slice M は分散メモリ等にも使用可能な構成を持ち，一方 Slice L は論理用途のみに利用できる．

RS ラッチは，図 7 に示す通り，2 Slice に分けて実装される．ラッチを構成する Slice の相対位置や配線経路が異なると再現性のある結果を得られないため，

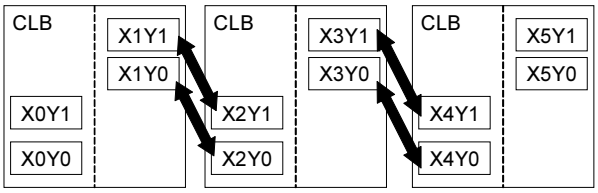


図 8　配線不能エラーが起こるラッチの配置

本研究では RS ラッチをハードマクロで実装した．ハードマクロは Slice の位置関係や配線が固定されたマクロであり, 自動配置配線によって Slice の相対位置や配線経路が変更されることがない. ただしラッチの各インスタンスの配置は, 自動配置配線で行う．本研究では図 7 の Slice 1 が配置される Slice を基点と呼称する. ハードマクロでは基点を配置する Slice の種類を明確に区別する. 基点を Slice L として作成されれたハードマクロは Slice L 以外に基点を置くことはない.

ハードマクロでラッチを実装すると, 同一 CLB の 2 つの Slice を使用した場合, Slice L と Slice M を使用した場合に自動配置配線が配線不能により失敗する．これは配線が固定されたマクロが近傍に複数配置された事による資源不足であると考えられる. 例えば, Slice L と Slice M を使用したラッチが図 8 の様に配置された場合, 中央の CLB には 4 つのインスタンスが配置されることになる. 一方, Slice L だけで構成されるラッチは奇数列の Slice を使用することは無いため, CLB に配置される基点は 2 個に制限される．これは Slice M についても同様である．そのため, 本研究では**異なる CLB の同種の Slice 2 個**を使用して RS ラッチを実装する.

5.2　ラッチの実装

ラッチが生成する乱数の品質は TRNG CLK のスキューや配線付加の影響を受けると考えられる．それらの影響を評価するために以下の 3 種類の実装について評価を行った.

(1)　**no FF**　図 10 の in FF, out FF を取り除いた RS ラッチ.

(2)　**in FF**　図 10 の in FF のみを実装する．図 11 に示すように，in FF は NAND LUT と同一の Slice に実装して，LUT 入力のスキューを最小化する.

(3)　**out FF**　図 10 の out FF のみを実装する. 出力に out FF を追加することで，Slice 1 の後段回路の影響を小さくし，Q と $\bar{Q}$ の配線負荷を均衡化

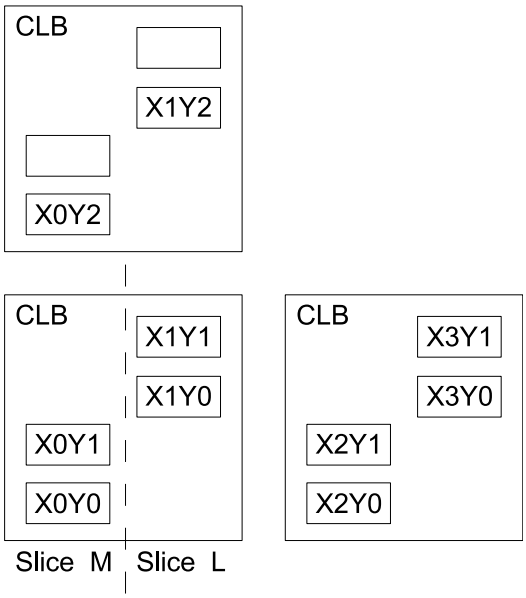


図 9　CLB の並びと座標番号

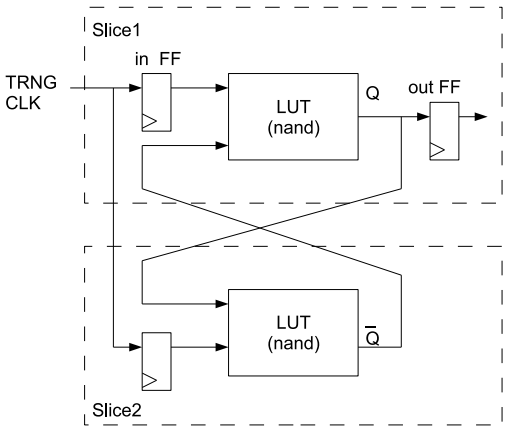


図 10　FF を使用した LUT latch の実装

する．

(4)　**in out FF**　図 10 をそのまま実装する．

RS ラッチは 2 Slice に分けて実装される．ラッチの出力は 2 つの Slice 間の距離の影響を受けると考えられるが, 当面はスキューと配線コストの低減を重視して，X 軸方向に隣り合った CLB の Slice L で RS ラッチを実装するものとす

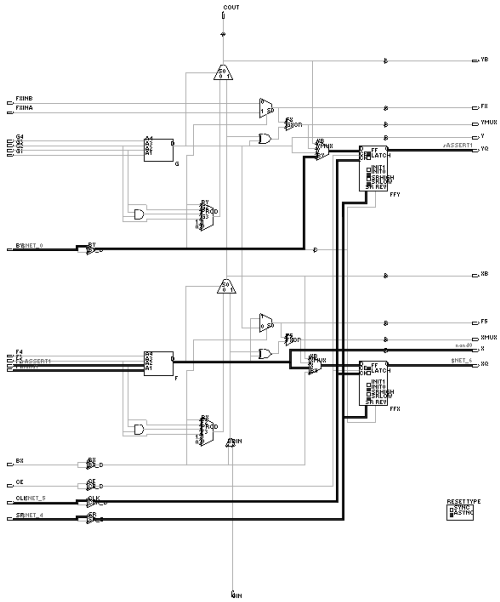


図 11　Slice L の資源割り当て

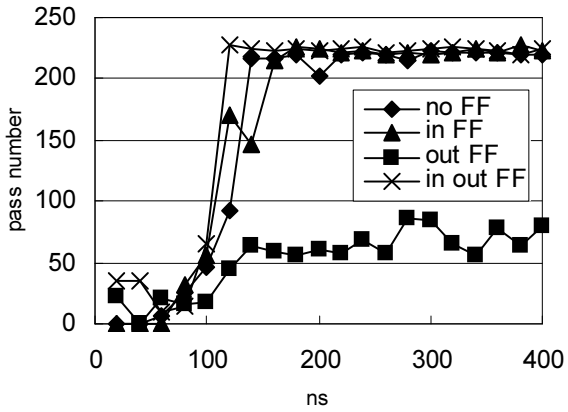


図 12　ラッチの実装とエントロピー LUT latch 64

る．この点は 5.4 章で定量的実験結果を示す．図 9 の例では X1Y0 に基点がおかれた場合，Slice 2 は X3Y0 に置かれることになる．

評価は LUT latch 64, 32 に対して行った (図 12, 13) . 図 12 は測定条件を LUT latch 64, サンプリング周期を 20ns から 400ns まで 20ns 間隔で可変として, in FF, in out FF を評価した結果である. LUT latch 64 では out FF の構成以外では, 結果に有意な差が見られなかった. これは LUT latch の数が多すぎるため差が現れなかったためと考えられる. そのため LUT latch 32 の構成で評価を行っ

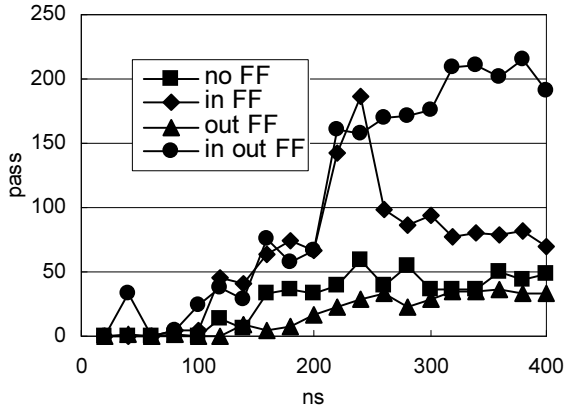


図 13　ラッチの実装とエントロピー LUT latch 32

た (図 13) . 評価条件は LUT latch 32, サンプリング周期を 20ns から 400ns まで 20ns 間隔で可変とし, 評価を no FF, in FF, in out FF に対して行った. 評価では in out FF の実装が最も高いエントロピーを出した. in FF の構成は in out FF よりも高い値を示す部分があるが安定して使用できる取得できるエントロピーは in out FF の方が良かった. 高品質の乱数を得るには対称的な構成をとることが重要であると考えられる. out FF の構成が悪く, in out FF が良い理由は良く分からない. 自動配置配線の影響が悪い方向に出た可能性が考えられる.

in FF の構成は no FF の構成に対して高い値を示した. 通過するテストの系列を比較するとサンプリング周期が 320ns の時に, no FF では Ranks of 31x31 and 32x32 matrices, Minimum Distance Test, Random Spheres Test, Runs Up and Down Test, The Craps Test に合格する. 一方, in FF では no FF で通過したテストに加えて, Birthday Spacings, Ranks of 6x8 Matrices, Parking Lot Test, Overlapping Sums Test が通過している. テストに合格する数は自動配置配線の影響を受けて上下するため安定性の評価が必要だと考えられる.

以後, in out FF の構成を LUT latch と表記する. また, in out FF の評価ではサンプリング周期が 320ns 以上で一定の値を示すことから, サンプリング周期を固定にして評価する場合は 320ns に設定する.

5.3　out FF の評価

out FF を使用すると乱数品質が低下する原因として自動配置配線の影響が考えられる. そのため cost table パラメータを変えた場合について評価を行った. cost table は自動配置配線の重みを指定するパラメータで, これを変更すること

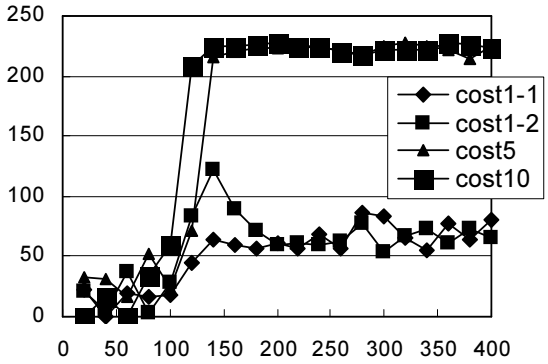


図 14　cost table を変えた場合の評価 LUT latch 64

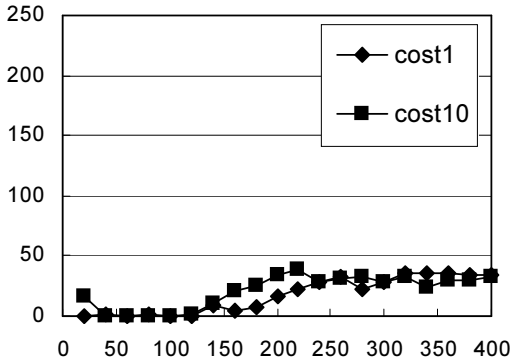


図 15　cost table を変えた場合の評価 LUT latch 32

で自動配置配線の結果を変える事が出来る.

まず最初に LUT latch 64 の構成で cost table を 1, 5, 10 に設定した場合について評価した. cost table を変えた場合の再現性を評価するため cost table が 1 の時については同じ条件で自動配置配線をやり直した場合の評価も行った (図 14) . 評価から分かるように cost table が 1 の時は良い結果が得られないが, それ以外の時は高品質の乱数が得られれた. cost table が 1 で自動配置配線をかけなおした場合 (cost1-2) でも乱数品質の低下が見られるため cost table の再現性は有るようだ. out FF 以外の構成でも cost table による乱数品質の低下が有るかも知れないがすべての構成を試すことは出来ないため, よく分からない.

次に LUT latch 32 の場合の評価を行った (図 15) . cost table を 1, 10 として評価したが, 乱数品質の改善は見られなかった. cost table を変えればよい結果が得られる可能性は有るが, 低品質の乱数を出力する cost table の方が多数であ

表 3 nand LUT 間の距離による影響

	dX2Y0	dX4Y0	dX6Y0	dX8Y0	dX0Y2	dX0Y3	dX0Y4
pass	226	0	224	0	215	224	131

ると予測される.

5.4 LUT latch の Slice 間の距離

ラッチを構成する 2 つの Slice 間の距離が長くなると, スキューが増大して生成エントロピーが小さくなる可能性がある．配線遅延も大きくなるため，遷移時間が増大して生成速度が低下する可能性もある．一方で，配線が長いとノイズレベルが大きくなり，エントロピーが増大する可能性も否定できない．そのため Slice 間の距離の影響を評価した．Slice 間の距離は Slice L を基点として X 軸方向は偶数 Slice, Y 軸方向は基点から 2Slice 以上の距離で 1 Slice ずつ離して評価する．X 軸方向が偶数 Slice なのは, LUT latch を奇数 Slice 離して実装すると異種 Slice を使用した実装になるためである．Y 軸方向が 2 Slice 以上なのは 1 Slice だけ離すと同一 CLB の Slice を使用した実装になるためである．距離を X 軸方向に 9 以上, Y 軸方向に 5 以上はなすと自動配置配線が配線不能のエラーを出力するため, それ未満の距離で評価を行った.

LUT latch 64 でサンプリング周期を 320 ns としたとき，Diehard テストの結果は表 3 の通りであった．ただし, dX4Y0 の実装で自動配置配線が配線不能のエラーを出したため, 配置配線の重みを指定する cost table パラメータを変えてエラーを回避している．表のラベルは基点からの相対距離を表す．例えば, 図 9 で基点座標が X1Y0, 相対距離が dX0Y2 とした場合, Slice 2 は X1Y2 に配置されることになる．Slice 間の距離には最適な値が有ることが分かる．特に dX4Y0, dX8Y0 で乱数品質が下がる結果になった．これは FPGA の配線構造によるものであると考えられるが, これ以上の評価は出来なかった．Y 軸方法に離した場合の評価では配線距離を長くすると乱数品質が下がる傾向が見られた．さらに離して評価する必要があるが, 自動配置配線でエラーが出るため評価できなかった.

表 3 では dX2Y0 の結果が最良であり，また Slice 間の距離は小さい方が配線資源を節約できることから，以降の評価では LUT latch の配線距離を dX2Y0 として実装する．

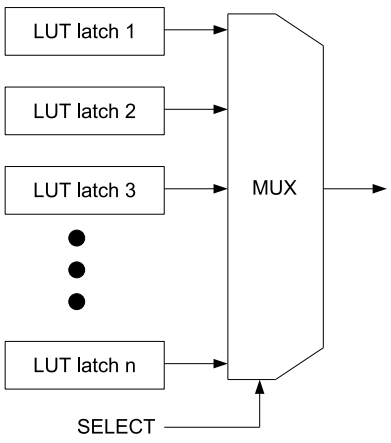


図 16　LUT latch 出力の測定回路

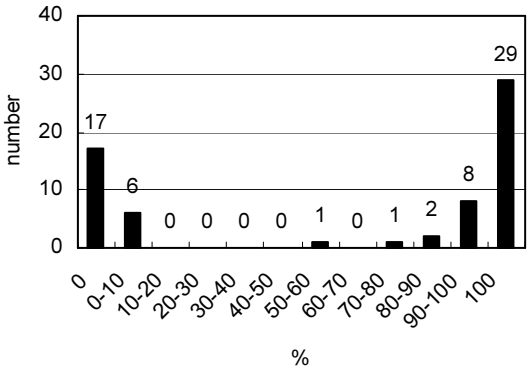


図 17　個体差のヒストグラム

5.5　LUT latch の個体差

個々の LUT latch が出力する乱数の偏りについて評価を行った．64 個の LUT latch を実装し，各ラッチの出力を MUX で選択して 1 MB ずつ取得する．(図 16) LUT latch の選択はソフトウェアで行い，サンプリング周期は 320 ns に固定した．評価は各 LUT latch 出力の 1 の数を調べることで行う．(表 4) 図 17 は表 4 をヒストグラム化した図である．横軸は 1 の割合，縦軸は個数を表す．72%の LUT latch は出力が 0 または 1 に固定されており，エントロピーを生成していない．残る 28%のラッチも，多くは出力に 90%以上の偏りを示している．この原因としては，ラッチを構成する LUT のドライブ能力の不均衡などが考えられ

表 4　LUT latch の偏り

number	%	number	%	number	%
LUT latch1	100.0	LUT latch23	100.0	LUT latch45	100.0
LUT latch2	82.381976	LUT latch24	95.144475	LUT latch46	100.0
LUT latch3	0.0	LUT latch25	100.0	LUT latch47	96.797335
LUT latch4	0.109065	LUT latch26	80.051267	LUT latch48	100.0
LUT latch5	96.995234	LUT latch27	100.0	LUT latch49	5.028307
LUT latch6	0.0	LUT latch28	100.0	LUT latch50	0.000024
LUT latch7	0.0	LUT latch29	0.0	LUT latch51	0.0
LUT latch8	0.0	LUT latch30	100.0	LUT latch52	76.398289
LUT latch9	95.490444	LUT latch31	98.540449	LUT latch53	100.0
LUT latch10	96.296442	LUT latch32	0.0	LUT latch54	100.0
LUT latch11	0.0	LUT latch33	0.0	LUT latch55	100.0
LUT latch12	0.0	LUT latch34	100.0	LUT latch56	100.0
LUT latch13	2.897334	LUT latch35	100.0	LUT latch57	100.0
LUT latch14	100.0	LUT latch36	0.0	LUT latch58	7.86097
LUT latch15	100.0	LUT latch37	93.703234	LUT latch59	100.0
LUT latch16	0.000167	LUT latch38	0.0	LUT latch60	100.0
LUT latch17	0.0	LUT latch39	0.0	LUT latch61	100.0
LUT latch18	100.0	LUT latch40	100.0	LUT latch62	100.0
LUT latch19	100.0	LUT latch41	100.0	LUT latch63	100.0
LUT latch20	0.0	LUT latch42	99.886382	LUT latch64	53.445065
LUT latch21	0.0	LUT latch43	0.0		
LUT latch22	100.0	LUT latch44	100.0		

る．FPGA ではフィードバック回路等で回路を制御することは難しいため，複数の LUT latch からエントロピーを収穫することにより，乱数品質の向上を図る必要がある．表 4 の LUT latch を XOR コレクターに通した場合の生起確率を計算すると 49.71%程度になる．

表 5　CLB に割り当てられる基点の数の影響

	Lxxx	LLxx	LLMM
pass	209	1	28

5.6　CLB 数とエントロピー

これまで実装では，2 CLB (8 Slice) のうち 2 つの Slice L を用いて LUT latch を実装した．未使用の Slice は自動配置で利用されるため無駄にはならないが，ハードマクロで LUTlatch を並べて密に実装することも可能である．そこで，LUT latch の基点が密集した場合の影響を，以下の 3 種類について評価した．

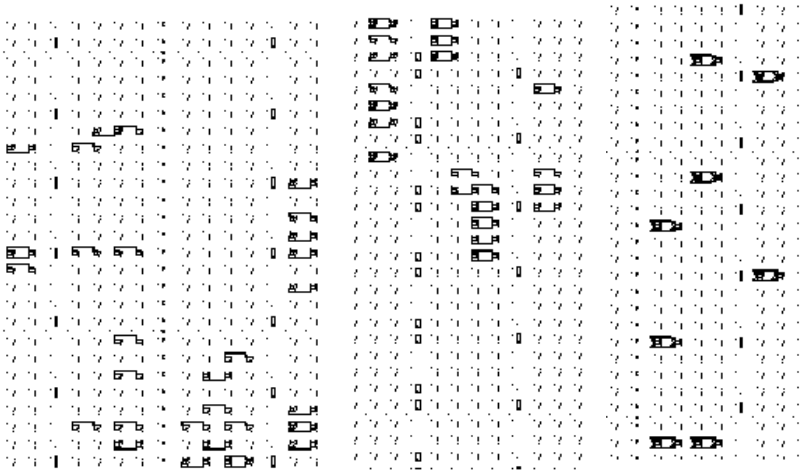
(1)　**Lxxx**　1 組の Slice L で 1 つの LUT latch を実装する．ただし，1 つの CLB に LUT latch が 1 つだけ配置されるとは限らない．自動配置配線で，同じ CLB 内に 2 つの基点が置かれる可能性はある．

(2)　**LLxx**　ハードマクロで，2 つの LUT latch の基点 (Slice L) が Y 軸方向に隣接して配置されるように実装する．ただし 2 組の LUTlatch の配置は自動配置配線が行う．図 9 の例では，X1Y0, X1Y1 のように 1 つの CLB に基点が置かれる場合と X1Y1, X1Y2 のように CLB を超えて配置される場合がある．

(3)　**LLMM**　ハードマクロで，1 つの CLB に基点が必ず 4 つ配置されるように実装する．即ち，一対 (2 個) の CLB に，4 つの LUT latch が実装される．図 9 の例では，X0Y0, X0Y1, X1Y0, X1Y1 の全てに基点が置かれる．4 個 1 組の LUT latch の配置は，自動配置配線が行う．

LUT latch 32 個を自動配置配線した結果を，図 18 に示す．各配置図の濃く見える部分が LUT Latch である．Lxxx が最も広い範囲に渡って配置されており，一方 LLxx や LLMM では基点が置かれる CLB 数が減っていることが確認できる．

これらの LUT latch32 について，サンプリング周期 320 ns で Diehard テストを行った．評価結果 (表 5) から明らかなように，同じ CLB に複数の基点が置かれると乱数品質が顕著に低下する．この原因として，(1) 配置が密集すると配線が難しくなって生成エントロピーが低下する，(2) 近接したラッチでは何らかの理由で出力に相関関係が発生する，等が考えられる．これ以上の原因究明は行っていないが，実験結果に従い，本研究では Lxxx の実装を採用する．



(a) Lxxx (b) LLxx (c) LLMM

図 18 LUT latch の配置

5.7 LUT latch の影響

TRNG の LUT latch 数を変えた場合の影響を評価した (図 19)．サンプリング周期は 320 ns に設定している．この結果では，64 個以上の LUTlatch を使うことが望ましい．

次に LUT latch の数とサンプリング周期の関係について評価した．LUT latch 32, 64, 128 について，サンプリング周期を 20 ns 刻みで変えながら Diehard テストで評価した (図 20) ．Diehard テストには LUT latch 64 で通過するが，さらに LUT latch の数を増やすことにより，サンプリング周期を短縮できることがわかった．LUT latch が増えることにより，遷移時間の短い LUTlatch が増えるためと考えられる．

6 乱数品質と回路規模

最終評価として，LUT latch 64, 128, 256 を NIST テストで評価し，全て合格することを確認した (表 6, 6) ．NIST テストに通過するサンプリング周期は LUT latch 64 が 260ns, LUT latch 128 が 120ns, LUT latch 256 が 80ns だった．TRNG の比較のため Sunar 型の TRNG (RO 110) についても評価した．RO 110 は 110 個の RO で構成される TRNG で, RO は 3 つのインバータを X 軸方向に隣り合った 3 つ Slice に割り当てたハードマクロとして実装される．RO 110 の

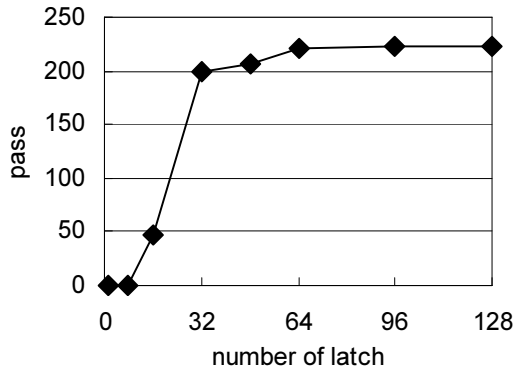


図 19　LUT latch の数

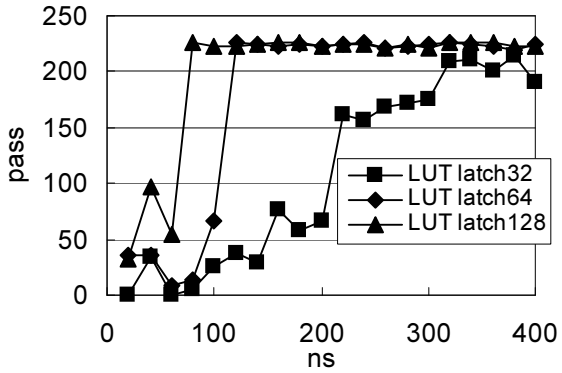


図 20　サンプリング周期

サンプリング周期は 320ns である．表では不合格の項目を太字で表記し, 偏りが大きくテストを行えなかった場合は N/A と表記している．項目数が大きく記載しきれないものについては合格数/項目の総数の形式で表記し, 一つでも不合格の項目が有る場合は太字表記にしている．RO 110 の TRNG では十分なエントロピーが得られないうえに回路規模も大きい結果になった．(表 8)．RO 110 の TRNG は実装上の工夫を行っていないため公平な評価ではないが, Sunar の手法では後処理を行って乱数品質を維持しているのに対し, 本手法は後処理なしで NIST テストに合格していることから本手法の方が優位であると言える．

回路規模は 145〜580 Slice となった．XC4VFX20 の回路規模は 8544Slice なので，TRNG はチップの 1.7〜6.7%程度となる．回路規模が小さいため LUT latch

の数が増えたとしてもシステムに与える影響は小さい (図 21) 乱数生成速度では LUT latch 256 が最も高い値になったが，各 TRNG は論理規模が違うため一律に生成速度を比較することは出来ない．LUT latch128 の生成速度は 8.33 Mbps であるが，2 つ並列に実装すれば LUTlatch 256 と同等の回路規模で 16.66 Mbps の生成速度を達成する．ただし TRNG 間に相関が発生する可能性があるので，実装して検証する必要がある．

表 6　NIST テストレポート 1

	LUT latch 64 260ns		LUT latch 128 120ns	
	P-VALUE	PROPORTION	P-VALUE	PROPORTION
frequency	0.116746	0.9880	0.255705	0.9890
block-frequency	0.745908	0.9910	0.610070	0.9880
cumulative-sums-up	0.266235	0.9890	0.207730	0.9860
cumulative-sums-down	0.136499	0.9850	0.322135	0.9850
runs	0.599693	0.9940	0.480771	0.9930
longest-run	0.339271	0.9870	0.008385	0.9880
rank	0.392456	0.9920	0.486588	0.9900
fft	0.041438	0.9840	0.190654	0.9880
nonperiodic-templates	148/148	148/148	148/148	148/148
overlapping-templates	0.340858	0.9860	0.624627	0.9890
universal	0.419021	0.9900	0.540204	0.9890
apen	0.325206	0.9910	0.969588	0.9880
random-excursions	8/8	8/8	8/8	8/8
random-excursions-variant	18/18	18/18	18/18	18/18
serial1	0.512137	0.9930	0.314544	0.9900
serial2	0.761719	0.9910	0.159910	0.9900
linear-complexity	0.000223	0.9890	0.049984	0.9940

表 7　NIST テストレポート 2

	LUT latch 256 80ns		RO 110 320ns	
	P-VALUE	PROPORTION	P-VALUE	PROPORTION
frequency	0.132640	0.9850	**0.000000**	**0.0000**
block-frequency	0.199045	0.9930	**0.000000**	**0.0030**
cumulative-sums-up	0.146152	0.9870	**0.000000**	**0.0000**
cumulative-sums-down	0.344048	0.9870	**0.000000**	**0.0000**
runs	0.540204	0.9900	**0.000000**	**0.0000**
longest-run	0.668321	0.9920	**0.000000**	**0.5260**
rank	0.883171	0.9850	0.739918	0.9890
fft	0.015490	0.9800	0.006107	0.9840
nonperiodic-templates	148/148	148/148	**38/148**	**41/148**
overlapping-templates	0.387264	0.9900	**0.000000**	**0.0000**
universal	0.581082	0.9820	**0.000000**	**0.9140**
apen	0.390721	0.9910	**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	0.068571	0.9910	**0.000000**	**0.4760**
serial2	0.420827	0.9880	0.038565	0.9900
linear-complexity	0.649612	0.9920	0.484646	0.9920

表 8　回路規模とビットレート

Design	Slice	System	Mbps
LUT latch 64	145	7013	3.8
LUT latch 128	290	7159	8.3
LUT latch 256	580	7447	12.5
RO 110	359	7219	

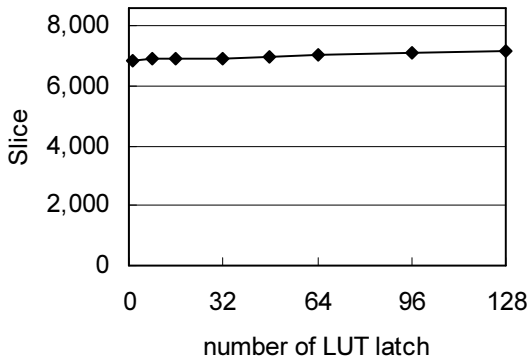


図 21　TRNG の回路規模と System の関係

7　終わりに

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 290 Slice で 8.33 Mbps を達成し，後処理なしで NIST テストに合格する．ハードマクロで実装することにより FPGA でも高品質の乱数を安定して生成することを可能にした．提案回路は同期式デジタル回路で構成されており，チップ外に追加回路が必要ないため容易に実装できる．クロックを停止することで電力消費を抑えることができるため，組込み用途にも適している．

本研究では TRNG CLK のデューティ比を 50%に固定した．しかし本来，TRNG CLK = 1 の期間は準安定状態から安定状態の遷移時間であり，一方 TRNG CLK = 0 の期間は $Q, \bar{Q}$ の初期化時間である．それぞれが別の物理過程であるため，それぞれを最適値に調整することにより，生成速度を改善できる可能性がある．

本研究でラッチ型 TRNG の基本動作は確認したが，実応用には更なる検討が

必要である．まず，電源電圧や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．しかし提案回路はアナログ的な配線遅延等を利用していないため，電源電圧，温度変化の影響は小さいと思われる．

XC4VFX20 以外のデバイスでも評価する必要がある．ラッチ型 TRNG は FF と論理要素で構成されるシンプルな回路で有ること，ラッチ数の変更で乱数品質と生成速度の調整ができることから，別のデバイスに対する実装も可能だと推測される．また，FPGA でも実装できるので，セミカスタムやフルカスタムの論理 LSI にも適用可能であると考えられる．

謝辞

最後まで根気強く御指導してくださった市川 周一先生にこの場を借りて御礼申し上げます．また，修士論文の副査をしていただいた河合 和久先生, 貴重なご意見を下さった和田 和千先生にこの場を借りて御礼申し上げます．最後に，苦楽を共にした研究室の皆さんに感謝致します．

References

1) 市川周一，畑尚志，“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,” Proc. SCIS 2009，2F1-5，2009. (CDROM)

2) 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.

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

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

5) V.Fischer, M.Drutarovský, M.Šimka , and N.Bochard, “High performance true random number generator in Altera Stratix FPLDs,” Proc. FPL 2004, LNCS 3203, pp.555–564, Springer, 2004.

6) M.Bellido, A.Acosta, M.Valencia, A.Barriga, and J.Huertas, “Simple binary random number generator,” Electronics Letters, vol.28, no.7, pp.617–618, 1992.

7) M.Epstein et al., "Design and implementation of a true random numer generator based on digital circuit artifacts," Proc. CHES 2003, LNCS 2779, pp.152–165, Springer, 2003.

8) D.Kinniment, and E.Chester, "Design of an on-chip random number generator using metastability," Proc. ESSCIRC 2002, vol.4, no.6, pp.595–598, 2002.

9) 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.78–84, 2008.

10) J.L. Danger, S.Guilley, and P.Hoogvorst, "Fast true random generator in FPGAs," Proc. IEEE NEWCAS 2007, pp.506–509, 2007.

11) J.R. Marino, "General theory of metastable operation," IEEE Transactions on Computers, vol.30, no.2, pp.107–115, 1981.

12) G. Marsaglia, "The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness," `http://www.stat.fsu.edu/pub/diehard/`, 2008.

13) NIST, "Statistical test suite," `http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html`, 2008.

14) "Diehard Battery of Tests of Randomness v0.2 beta," `http://i.cs.hku.hk/~diehard/`, 2008.

15) 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).

16) Xilinx, "Virtex-4 FPGA user guide," 2008.

付録

A　ハードマクロを使用しなかった場合

本研究ではハードマクロを利用することによって乱数品質の改善を行った. ハードマクロはデバイスに依存する機能であるためハードマクロを使用せずに実装できれば有用であるといえる. そのため, ソフトマクロでの実装 (ソフトマクロ LUT latch) について検討を行った.

本来, ラッチのような組み合わせ回路のループ構造を持つ回路を HDL で記述することは推奨されていない. そのため CAD はラッチ記述を発見するとそれを除外して回路を生成する. これを回避するためにソフトマクロ LUT latch では NAND ゲートを LUT のインスタンシエーションによって実現し, ループ配線に KEEP 制約をつけて最適化を防いでいる. また in FF のような論理的には意味のない FF も最適化によって削除されるため EQUIVALENT_REGISTER_REMOVAL 制約を使用して除外されるのを防いでいる. ソフトマクロ LUT latch を構成する Slice の相対位置を自動配置配線に任せると安定した有意な結果が得られないため, RLOC 制約を使用して相対位置の指定を行う. 実装例を以下に示す.

- ソフトマクロ LUT latch の実装

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity LUT_latch is
        port (
            iCLK               : in  std_logic;
            iRESET             : in  std_logic;
            iASSERT            : in  std_logic;
            oQ                 : out std_logic;
            oNQ                : out std_logic
        );
end entity LUT_latch;

architecture RTL of LUT_latch is
    --=============================================================
    -- signal
    --=============================================================
    signal nand_1 : std_logic;
    signal nand_2 : std_logic;
    signal rIN_FF : std_logic_vector(1 downto 0);
    attribute KEEP : string;
    attribute KEEP of nand_1 : signal is "TRUE";
```

```
    attribute KEEP of nand_2 : signal is "TRUE";
    attribute RLOC : string;
    attribute RLOC of U0_LUT : label is "X0Y0";
    attribute RLOC of U1_LUT : label is "X2Y0";
    attribute EQUIVALENT_REGISTER_REMOVAL : string;
    attribute EQUIVALENT_REGISTER_REMOVAL of rIN_FF : signal is "NO";
    --=============================================================
    -- component
    --=============================================================
    component LUT2
        generic(INIT : bit_vector := "0000");
        port(O    : out std_ulogic;
            i0   : in std_ulogic;
            i1   : in std_ulogic);
    end component;
--=================================================================
begin
    L_rOFF:
    process( iRESET, iCLK ) begin
        if( iRESET = '1' ) then             -- グローバルリセット
            oQ <= '0';
    elsif(iCLK'event and iCLK = '1') then
            oQ <= nand_1;
        end if;
    end process;

    L_rINFF:
    process( iRESET, iCLK ) begin
        if( iRESET = '1' ) then             -- グローバルリセット
            rIN_FF <= (others => '0');
        elsif(iCLK'event and iCLK = '1') then
            rIN_FF(0) <= iASSERT;
            rIN_FF(1) <= iASSERT;
        end if;
    end process;

    oNQ <= '0';
    U0_LUT : LUT2                           --NAND ゲート 1
        generic map(INIT => "0111")
        port map (
            O =>  nand_2,
            I0 => rIN_FF(0),
            i1 => nand_1);

    U1_LUT : LUT2                           --NAND ゲート 2
        generic map(INIT => "0111")
        port map (
            O =>  nand_1,
            I0 => rIN_FF(1),
            i1 => nand_2);
end RTL;
```

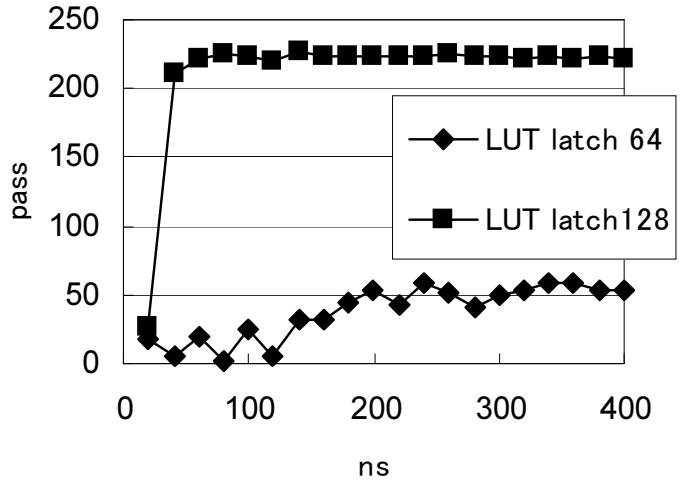


図 22　rloc での Diehard テスト

評価は LUT latch64, 128 の構成で行った (図 22) . Slice 間の距離は X 軸方向に隣り合う (attribute RLOC of U1_LUT : label is "X2Y0";) ように設定し, サンプリング周期を変えながら評価した. 図 22 から分かるように LUT latch 128 の構成であれば Diehard テストに通過する. ハードマクロに対してエントロピーは下がっているが, 有る程度の品質は確保できているようだ. LUT latch 128, サンプリング周期 320ns の時に NIST テストに合格することを確認した. 乱数生成速度については自動配置配線による変動が大きく評価できなかった. ソフトマクロ LUT latch でも RLOC による相対位置指定を行えば有る程度の乱数品質が保持できることが分かった.

B　デューティー比を変更した場合の乱数品質

本研究では TRNG CLK のデューティ比を 50% に固定した. しかし本来, TRNG CLK = 1 の期間は準安定状態から安定状態の遷移時間であり, 一方 TRNG CLK = 0 の期間は $Q, \bar{Q}$ の初期化時間である. これらは別の物理過程であり独立に設定できるものである. そのため, Duty を変えた場合の傾向を観察した (図 23, 24) .

図 23 は LUT latch 64 の構成で TRNG CLK = 1 の期間を 200ns で固定とし, TRNG CLK = 0 の期間を可変にした場合である. 横軸は TRNG CLK = 0 の期間を表している. テストに通過する p value の数に多少の増減はあるが, 全体的に高い値を維持している. 図 24 は LUT latch 64 の構成で TRNG CLK = 0 の

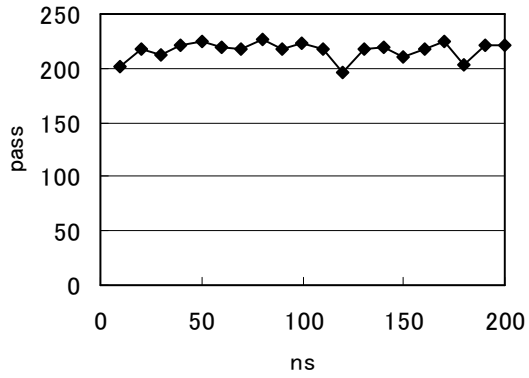


図 23　TRNG CLK = 0 の区間を可変にした場合

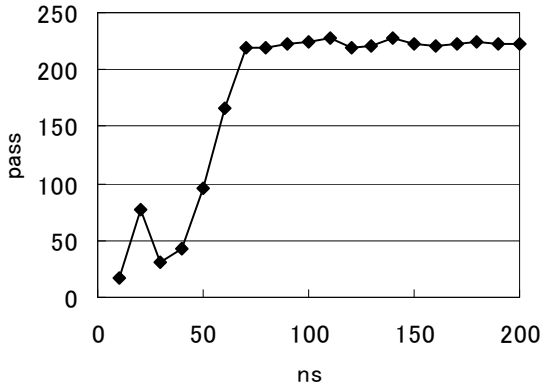


図 24　TRNG CLK = 1 の区間を可変にした場合

期間を 200ns で固定とし, TRNG CLK = 1 の期間を可変にした場合である. 横軸は TRNG CLK = 1 の期間を表している. この評価ではサンプリング周期が 70ns で十分な乱数品質が得られている. デューティー比を 50%に設定したときの評価は TRNG CLK = 1 の期間が 60ns 以上の時に一定の値を示すようになっており, 図 24 の結果とほ一致している. 乱数生成速度のボトルネックとなっているのは TRNG CLK = 1 の期間であり, TRNG CLK = 0 の期間は短く出来る可能性がある.

C　ハードマクロの作り方

C.1　概要

ハードマクロはISEのFPGA Editerを使用して作成する．まずFPGA Editerを起動してFileメニューのnew projectでプロジェクトを作成する．このときデバイス設定とhard macroプロジェクトを選択する．プロジェクト作成後に図25のようなウインドウが開く．図のプロジェクトでは作業領域を白黒反転させて表示している．

図中の1番が作業ウインドウ, 2番はステータスウインドウである．

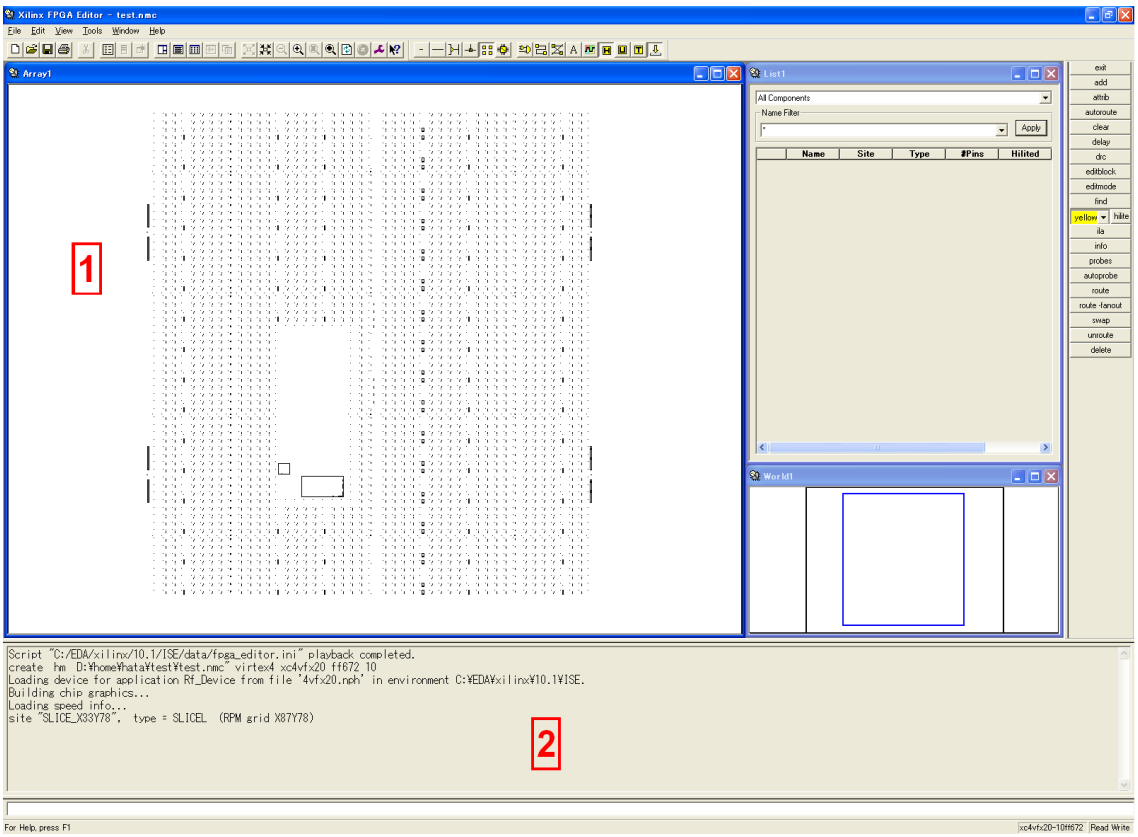


図 25　FPGA Editerのウインドウ

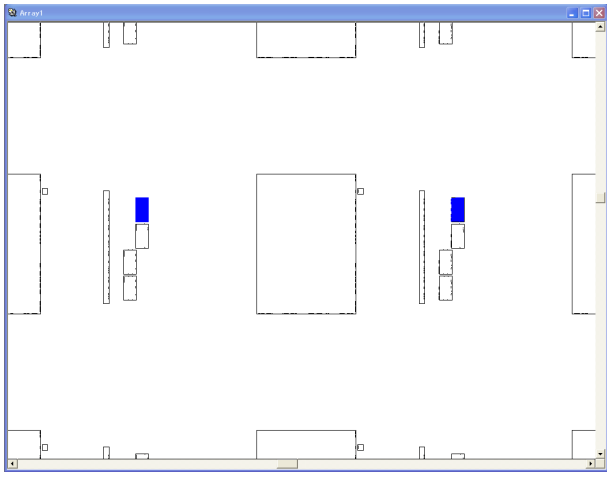


図 26　Slice の追加

C.2　使用する Slice の定義

ハードマクロを作成するために, ロジックとして使用する Slice を定義する. 作業ウインドウで使用するスライスを選択し, Edit メニューの add をクリックする. 追加する Slice のステータスが表示されるので, Slice のインスタンス名を入力する. 追加する Slice が Slice M の場合はロジック (Slice L) もしくは機能ブロック (Slice M) の選択をする. 定義された Slice は青色で表示される (図 26) .

C.3　Slice 内のロジック定義

次に Slice のロジックを定義する. 追加した Slice をダブルクリックして図 27 のウインドウを開く. ロジックを編集するために図 27 の 1 番の鉛筆アイコンと 2 番の F=アイコンを選択して 3 番のウインドウを表示させる. 3 番のウインドウは LUT の論理を編集するための物で, 論理を定義する場合は F と G の項目のみ編集する. F の項目で下側の LUT (LUT F) の定義を行い, G の項目で上側の LUT (LUT G) の定義を行う. LUT F の入力は F1-F4, LUT G の入力は G1-G4

表 9　演算子シンボル

Symbol	Logic
~	NOT
+	OR
*	AND
@	XOR

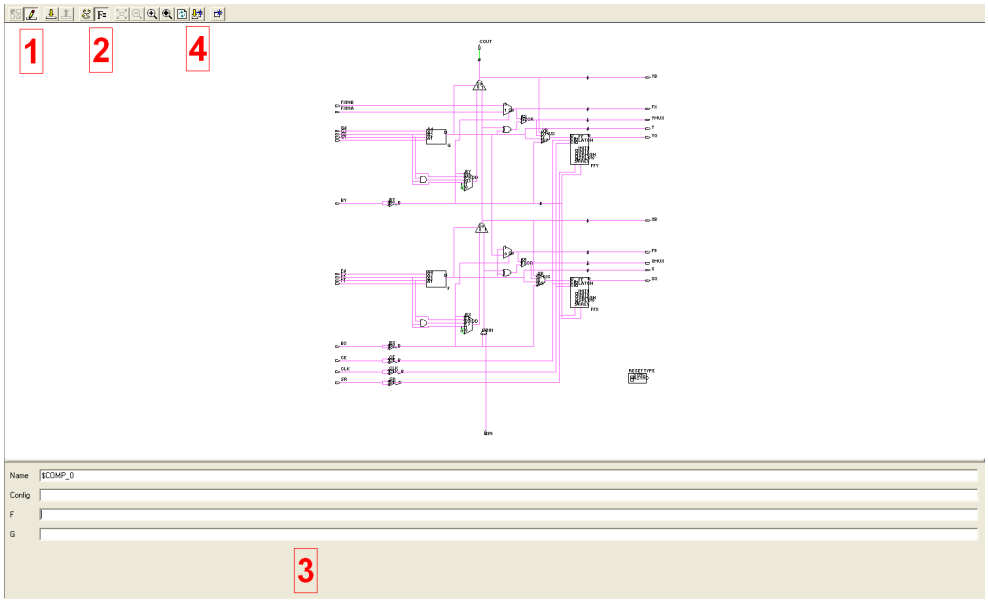


図 27　Slice ロジックの編集

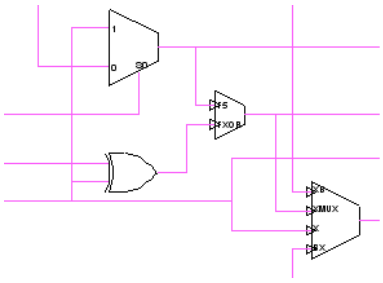


図 28　データパスの設定

であらわされるがロジックを編集する場合は, それぞれ A1-A4 として扱う. たとえば LUT F を F1 or F2 として定義する場合は F の項目に A1+A2 を入力する. LUT の定義に使用できる演算子を表 9 に示す.

次にデータパスを設定する. データパスの定義は MUX の入力を選択することで行う. Slice 内の MUX には図 28 のように入力側に三角形のアイコンが設けられており, これをクリックすることでデータパスが定義される. FF を使用する場合はクロック, リセット配線の選択も行う.

FF を使用するためには FF と LATCH, 初期化時の値 (INIT 1, INIT 0), リセット時の値 (SR HIGH, SR LOW), 同期非同期リセット (SYNC, ASYNC) の選択も行う. これもアイコンをクリックして行う. 編集が終了すると 4 番のアイコンをクリックして保存終了する.

C.4 Slice 間の配線定義

次に Slice 間の配線定義を行う. 配線を定義するには配線のソースとディスティネーションの端子を Ctrl を押しながら選択し, Tools メニュー Route の Manual Route を選択する.

C.5 入出力信号の登録

最後にハードマクロの入出力信号を定義する. 入出力信号の登録は登録したい端子を選択して Edit メニューの Add Hard Macro External Pin を選択する. クロックやリセットのように同じ信号を複数の端子に割り当てたい場合は, 割り当てたい端子を Ctrl を押しながらすべて選択し Edit メニューの Add を選択して net を張る. その後, net を構成するいずれかの端子を入出力信号として登録する. 最後に保存終了して終了する. ハードマクロはマクロ名.nmc ファイルに保存される.

C.6 ハードマクロの使用方法

ハードマクロを使用する場合, ハードマクロファイルを ISE のパスが通っているフォルダ (ISE プロジェクトのカレントディレクトリなど) に置く. HDL からマクロを呼び出す場合はコンポーネント名をマクロのファイル名, 入出力を登録した端子名としてインスタンシエーションする.

D　TRNG IP 仕様

D.1　概要

TRNG IP はPLB インターフェースと SRAM_FIFO_ADAPTER, TRNG_ADAPTER マクロで構成される (図 29) ． SRAM_FIFO_ADAPTER は ML405 ボードの SRAM を使用した簡易 FIFO マクロであり, TRNG_ADAPTER は乱数を 32bit データとして出力するマクロである. DET_UP_DOWN_VEC は立ち上がり検出を行う. PPC 側のインターフェースとして CONTROL, STATUS, DATA レジスタが用意されている. Linux の TRNG IP デバイスドライバは ioremap 関数でアドレス登録する. ioremap 関数で登録されたデバイスはリトルエンディアンでデータアクセスが行われるため, PLB インターフェースはリトルエンディアンで入出力を行う. PPC は本来ビッグエンディアンを採用しているため ioremap 関数以外でアドレス登録する場合はエンディアン変換を行う必要がある.

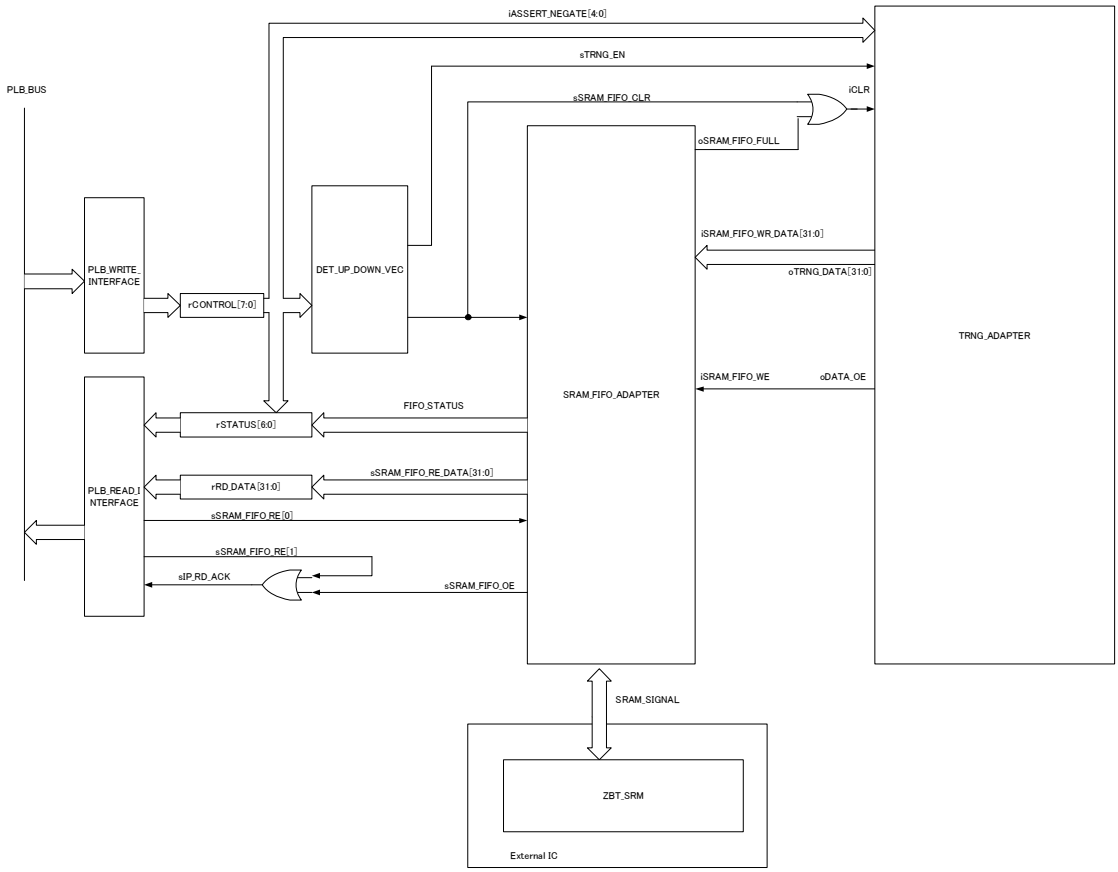


図 29　TRNG コアブロック図

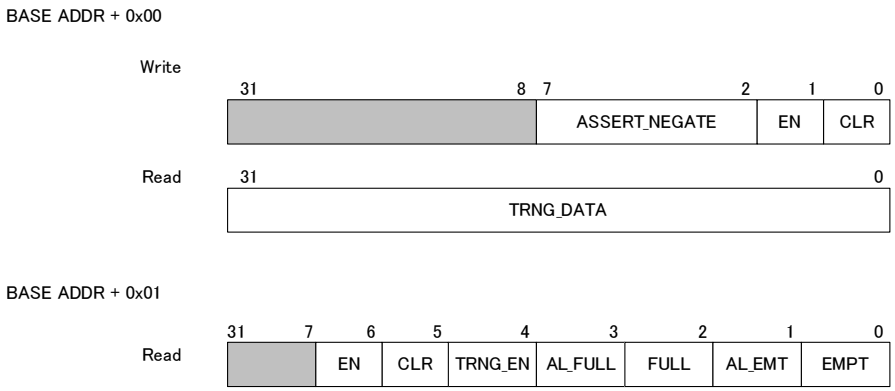


図 30　レジスタマップ

D.2　レジスタマップ

レジスタはベースアドレスに TRNG_DATA レジスタ, CONTROL レジスタ. ベースアドレス+1 に STATUS レジスタが割り当てられている (図 30) . ベースアドレスへのライトは CONTROL レジスタ, リードは TRNG_DATA レジスタへのアクセスになる.

- CONTROL レジスタ

  コントロールレジスタは TRNG IP のクリア, 乱数取得開始, サンプリング周期の設定を行う (表 10) . 乱数の取得前に IP クリア, サンプリング周期の設定を行なう必要がある. IP クリア, サンプリング周期の設定を行わずに EN フラグを立てた場合の動作は保障しない. サンプリング周期は ASSERT_NEGATE の 2 倍のクロック数になる. デューティは 50%で固定である.

表 10　CONTROL レジスタ

NAME	BIT	R/W	FUNCTION
CLR	0	W	0 から 1 の立ち上がりでコアクリア
EN	1	W	0 から 1 の立ち上がりで乱数取得開始
ASSERT_NEGATE	7:2	W	サンプリング周期の設定 デューティー比 50% 周期は ASSERT_NEGATE×2 クロック

- TRNG_DATA レジスタ
  TRNG_DATA は FIFO からデータを取得するリードオンリーのレジスタである．TRNG_DATA レジスタは乱数取得開始後, STATUS レジスタの FULL フラグが H なったのちアクセスできる．その後, SRAM FIFO が空になるまで EMPTY になるまでアクセスできるが, EMPTY の状態でアクセスすることは禁止されている．FULL フラグが H になる前や, EMPTY の状態でリードした場合の動作は保証しない.
- STATUS レジスタ
  STATUS レジスタは TRNG IP の状況を取得するリードオンリーのレジスタである (図 11) ．EMPTY, AL_EMPTY, FULL, AL_FULL フラグはそれぞれ SRAM FIFO の状態を表す．EMPTY フラグが H の時には FIFO にデータがない状態であり, AL_EMPTY は FIFO のデータが 1 つだけある事を表す．FULL は FIFO が満たされていることを表し, AL_FULL フラグは FIFO にあと 1 つだけデータを書き込めることを表す.
  TRNG_EN は TRNG ADAPTER マクロの動作状況を表すフラグで主にデバッグに使用する．TRNG_ADAPTER マクロが乱数生成中ならば TRNG_EN は H になる．EN, CLR は CONTROL レジスタの EN, CLR の設定値を取得する.

表 11 STATUS レジスタ

NAME	BIT	R/W	FUNCTION
EMPTY	0	R	SRAM FIFO EMPTY フラグ
AL_EMPTY	1	R	SRAM FIFO ALMOST EMPY フラグ
FULL	2	R	SRAM FIFO FULL フラグ
AL_FULL	3	R	SRAM FIFO ALMOST FULL フラグ
TRNG_EN	4	R	TRNG ADAPTER 乱数生成中
EN	5	R	CONTROL レジスタ EN
CLR	6	R	CONTROL レジスタ CLR

D.3 入出力信号

TRNG IP の入出力信号は大きく分けて 2 系統に分けられる (表 12). PLB 系統の信号は Xilinx の IPIF コアに接続するための信号である. IPIF コアは Xilinx が提供する PLB バスインターフェスコアで, TRNG IP は IPIF コアを介して PPC に接続される.

SRAM 系統の信号は ML405 の ZBT SRAM IC に接続するための信号である. チップ外の IC にアクセスするための信号であるため入出力バッファと FF が実装されている.

表 12 TRNG IP 入出力信号

TYPE	NAME	BIT	I/O	FUNCTION
PLB	iCLK		I	バスクロック
	iRESET		I	システムリセット
	iADR	[0:31]	I	アドレス入力
	iRNW		I	Read not Write
	iCS		I	チップセレクト
	iWDATA	[0:31]	I	ライトデータ
	oRDATA	[0:31]	O	リードデータ
	iWR_REQ		I	ライトリクエスト
	iRD_REQ		I	リードリクエスト
	oRD_ACK		O	Read acknowledge
	oWR_ACK		O	write acknowledge
SRAM	oFLASH_SRAM_WE_N		O	SRAM WE 負論理
	oFLASH_CE_SRAM_CEN		O	SRAM チップイネーブル 負論理 0 固定
	oSRAM_ADV_LD_N		O	バースト信号 0 固定
	oSRAM_OE_N		O	出力イネーブル 負論理
	oSRAM_BW_N	[3:0]	O	バイトセレクト
	oSRAM_ADDR	[20:0]	O	SRAM アドレス
	ioSRAM_DATA	[31:0]	I/O	SRAM データ 双方向バス

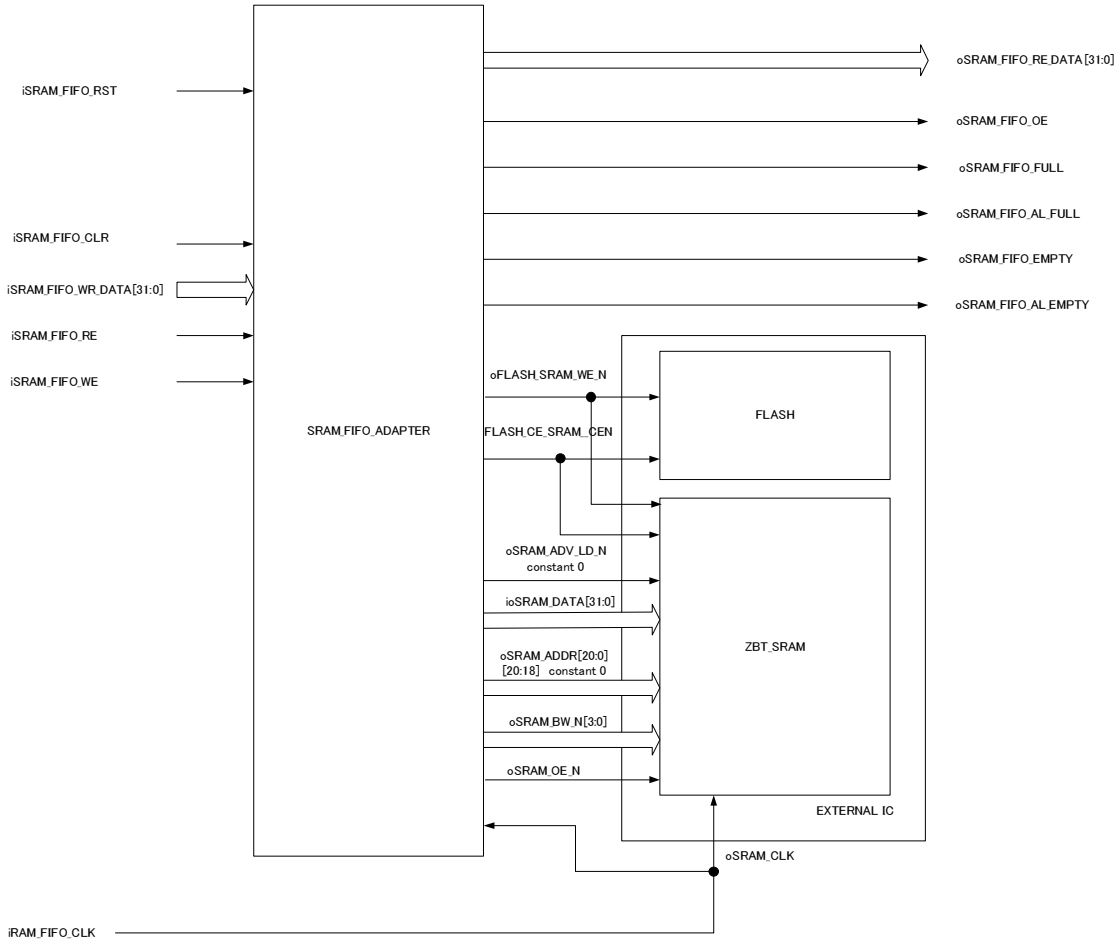


図 31　ML405 ボード SRAM バス

D.4　SRAM_FIFO_ADAPTER マクロ

SRAM_FIFO_ADAPTER は ML405 ボードの SRAM を簡易 FIFO として使用するためのマクロである．マクロは ML405 ボードに依存しているため他のボードで使用する場合は検討が必要である．ML405 の ZMT SRAM はクロックを供給する必要が有るが, ユーザマクロで扱うべき信号ではないため本マクロでは扱わない．マクロを使用する場合は SRAM へのクロックを別に設定する必要がある．ML405 ボードではバスを SRAM と Flash が共有して使用している．そのため SRAM と Flash を調停するための信号が実装されている．しかし, 本研究では Flash を使用しないためそのほとんどは固定値となっている．Flash を使用する場合は調停回路を追加する必要がある．

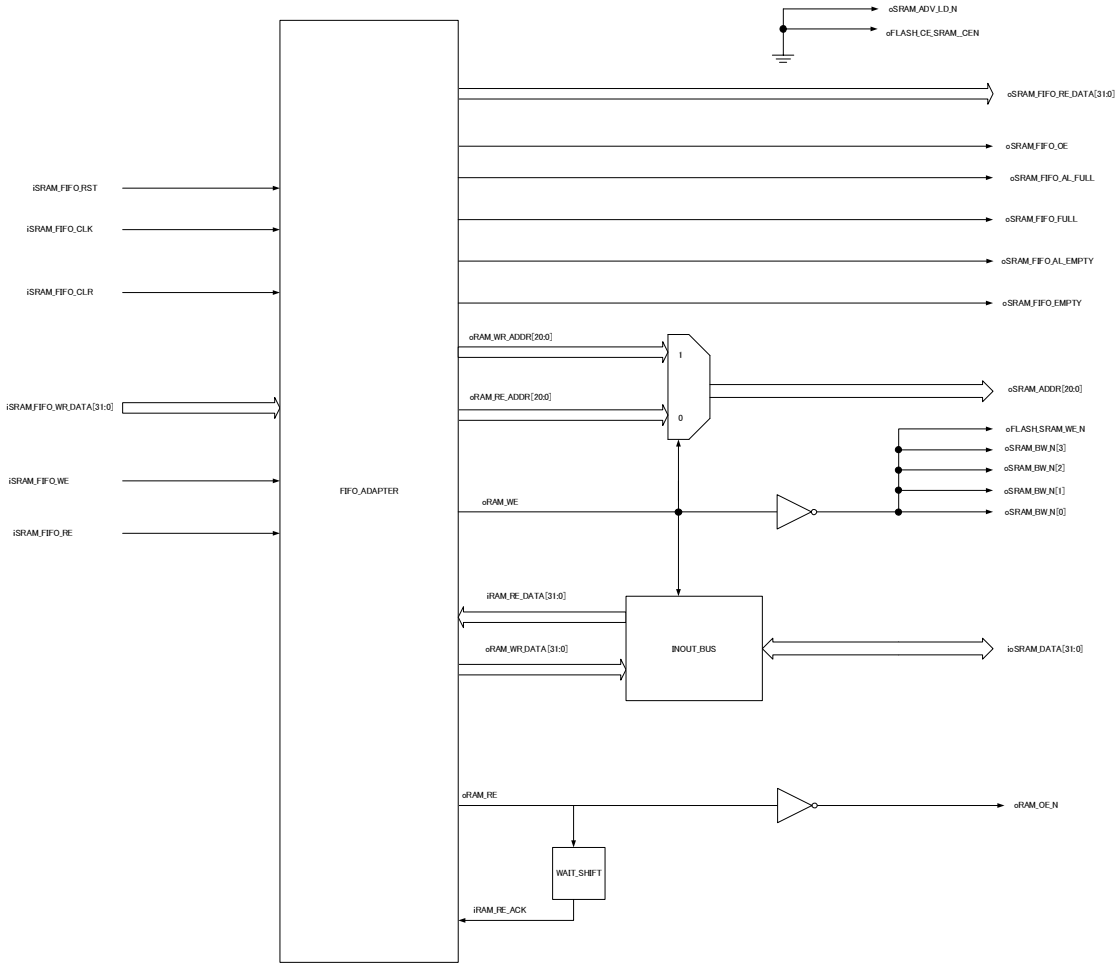


図 32　SRAM_FIFO_ADAPTER ブロック図

D.4.1　ブロック図

SRAM_FIFO_ADAPTER は SRAM を使用して FIFO を構成する．FIFO としての機能は FIFO_ADAPTER マクロによって構成されており, SRAM_FIFO_ADAPTER ばボード依存部の機能だけ実装している．ボード依存部としては, Flash と共有しているバスの調停回路, データバスの双方向バッファ (INOUT_BUS) , リード時間調整回路, 入出力バッファと FF である．ライトは連続書き込みできるが, リードには 5CLK かかる．

D.4.2 入出力信号

SRAM_FIFO_ADAPTERの入出力信号を表13に示す. SRAM_FIFO_ADAPTERの本体はFIFO_ADAPUTEであるためFIFO制御信号の詳細はD.5章を参照

表13 SRAM_FIFO_ADAPTER入出力信号

TYPE	NAME	BIT	I/O	FUNCTION
SYSTEM	iSRAM_FIFO_CLK		I	システムクロック
	iSRAM_FIFO_RST		I	非同期リセット
USER	iSRAM_FIFO_CLR		I	同期クリア
	iSRAM_FIFO_RE		I	リードイネーブル
	iSRAM_FIFO_WE		I	ライトイネーブル
	oSRAM_FIFO_OE		O	データ出力イネーブル
	oSRAM_FIFO_FULL		O	FIFO FULL
	oSRAM_FIFO_AL_FULL		O	FIFO ALMOST FULL
	oSRAM_FIFO_EMPTY		O	FIFO EMPTY
	oSRAM_FIFO_AL_EMPTY		O	FIFO ALMOST EMPTY
	iSRAM_FIFO_WR_DATA	[31:0]	I	ライトデータ
	oSRAM_FIFO_RE_DATA	[31:0]	O	リードデータ
SRAM	oFLASH_SRAM_WE_N		O	SRAM WE 負論理
	oFLASH_CE_SRAM_CEN		O	SRAM CE 負論理 0 固定
	oSRAM_ADV_LD_N		O	バースト信号 0 固定
	oSRAM_OE_N		O	出力イネーブル 負論理
	oSRAM_BW_N	[3:0]	O	バイトセレクト
	oSRAM_ADDR	[20:0]	O	SRAM アドレス
	ioSRAM_DATA	[31:0]	I/O	SRAM データ 双方向バス

D.5 FIFO_ADAPTER

FIFO_ADAPTERマクロはRAMを使用して簡易FIFOを構成する. FIFOのステータスはFULL, AL_FLL, EMPTY, AL_EMPTYの4状態で表される. FULLアサートはライト不可を表し, AL_FULLアサートの時は残り1回だけ書き込み可能を表す. EMPTYがアサートされるときはリード不可を表し, AL_EMPTY

アサートはデータ残量が1つだけ有る状態を表す. FIFOの容量はADDR_WIDTHパラメータによって設定可能で, 容量は2^{ADDR_WIDTH}になる. データのbit幅はDATA_WIDTHパラメータで設定する.

D.5.1 入出力信号

FIFO_ADAPTERの入出力信号を図14に示す. 表ではデータバスのビット幅をDW, アドレスバスのビット幅をAWで表す.

表14 FIFO_ADAPTER入出力信号

TYPE	NAME	BIT	I/O	FUNCTION
SYSTEM	iRAM_FIFO_CLK		I	システムクロック
	iRAM_FIFO_RST		I	非同期リセット
USER	iRAM_FIFO_CLR		I	同期クリア
	iFIFO_RE		I	リードイネーブル
	iFIFO_WE		I	ライトイネーブル
	oFIFO_OE		O	データ出力イネーブル
	oFIFO_FULL		O	FULLステータス
	oFIFO_AL_FULL		O	ALMOST FULLステータス
	oFIFO_EMPTY		O	EMPTYステータス
	oFIFO_AL_EMPTY		O	ALMOST EMPTYステータス
	iFIFO_WR_DATA	[DW-1:0]	I	ライトデータ
	oFIFO_RE_DATA	[DW-1:0]	O	リードデータ
RAM	iRAM_RE_ACK		I	RAM READ ACKNOWLEDGE
	oRAM_RE		O	RAM リードリクエスト
	oRAM_WE		O	RAM ライトリクエスト
	iRAM_RE_DATA	[DW-1:0]	I	RAM リードデータ
	oRAM_WR_DATA	[DW-1:0]	O	RAM ライトデータ
	oRAM_RE_ADDR	[AW-1:0]	O	RAM リードアドレス
	oRAM_WR_ADDR	[AW-1:0]	O	RAM ライトアドレス

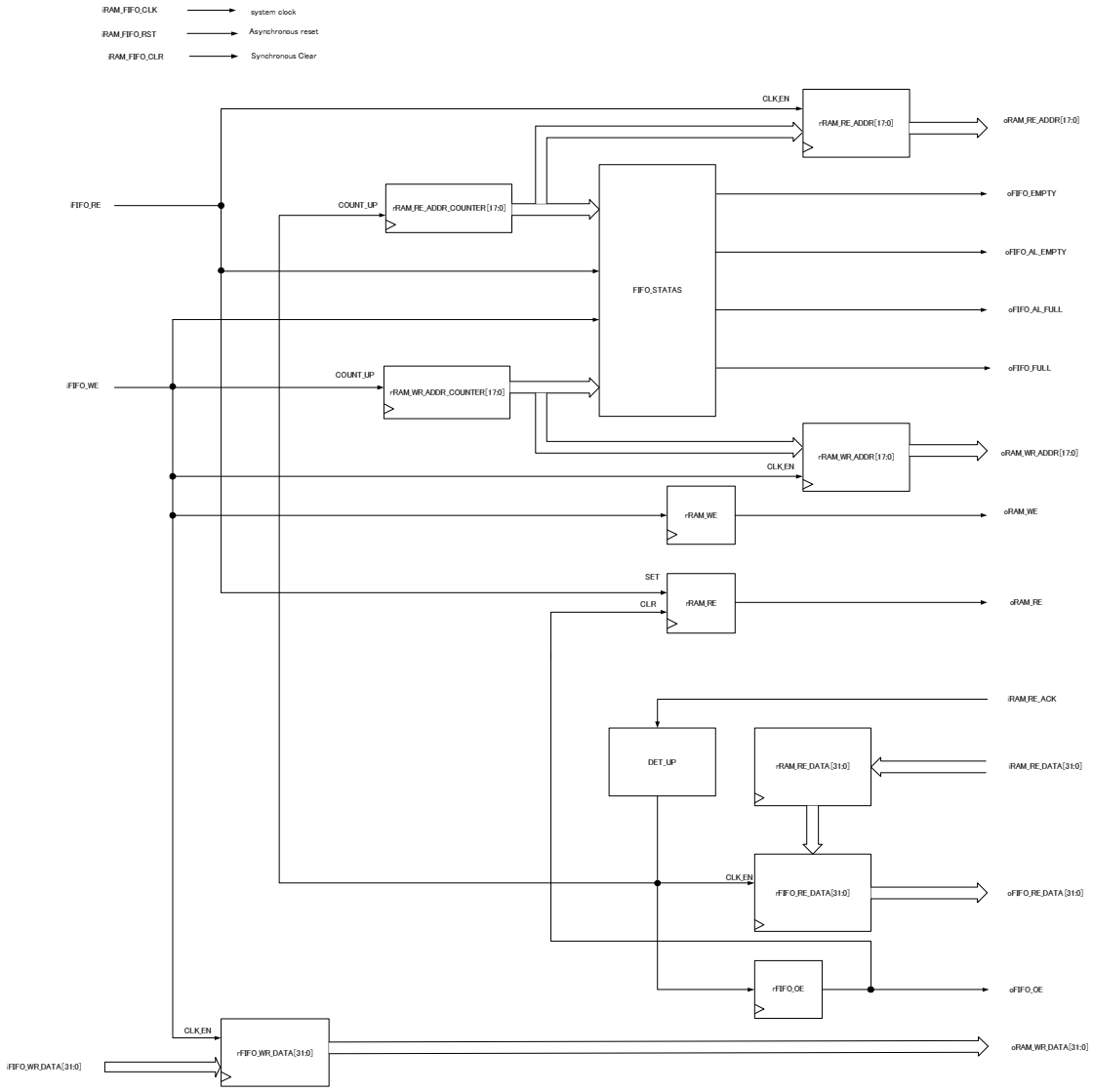


図 33　FIFO_ADAPTER ブロック図

D.5.2　FIFO_ADAPTER ブロック図

FIFO_ADAPTER はアドレスカウンタ (rRAM_RE_ADDR_COUNTER, rRAM_WR_ADDR_COUNTER) とステータス信号生成回路 (FIFO_STATUS) で構成される. アドレスカウンタは FIFO のライト, リードリクエスト (iFIFO_WE, iFIFO_RE) によって更新されるが更新タイミングはライトとリードで変わる.

ライトは FIFO ライトリクエスト がアサートされたクロックの次のクロックでカウンタが更新される. ライトは連続で行えるようになっているため, iFIFO_WE は 1 回のライトにつき 1CLK だけアサートできる. 複数クロックに渡ってアサー

トされた場合は複数回のライトとして処理される.

リードはRAMからリードアクノレッジ (iRAM_RE_ACK) が帰ってきたタイミングで更新される. iRAM_RE_ACK 待ちの状態で複数回 iFIFO_RE がアサートされた場合は無視する. iRAM_RE_ACK 待ち状態の時は oRAM_RE がアサートされる.

図中の DET_UP は立ち上がり検出回路である.

D.5.3 FIFO_STATUS ブロック図

FIFO_STATUS は FIFO のステータス信号を生成する. アドレス比較器と FF によって構成されており, ステータスの更新は FIFO のリクエスト信号がアサートされたクロックで行われる.

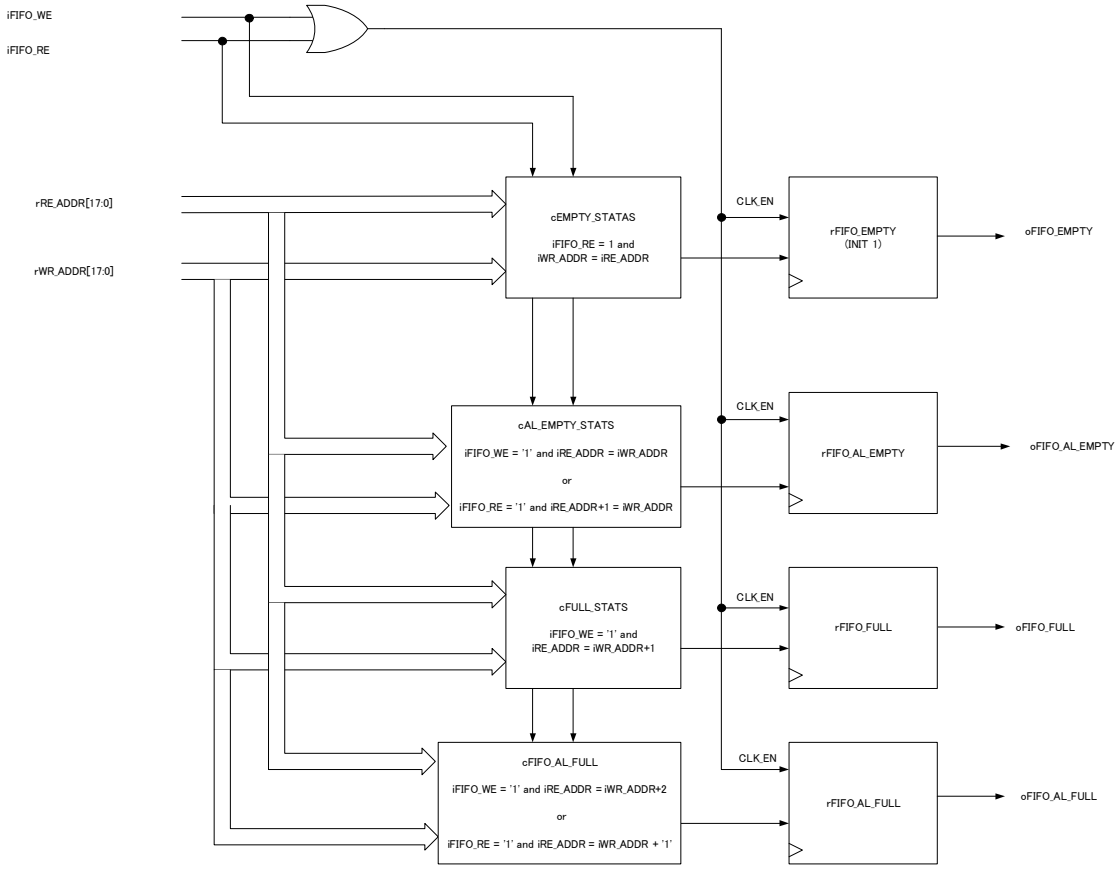


図 34 FIFO_STATUS ブロック図

D.5.4　タイミング図

- リードタイミング

  リードは iFIFO_RE をアサートして開始する．iFIFO_RE がアサートされると FIFO_ADAPTER は RAM にリードリクエストを出力し，RAM から iRAM_RE_ACK が帰ってくるまで待つ．iRAM_RE_ACK がアサートされると iRAM_RE_DATA を取り込み oFIFO_RE_DATA に渡す．oFIFO_RE_DATA は oFIFO_OE がアサートの時有効になる．

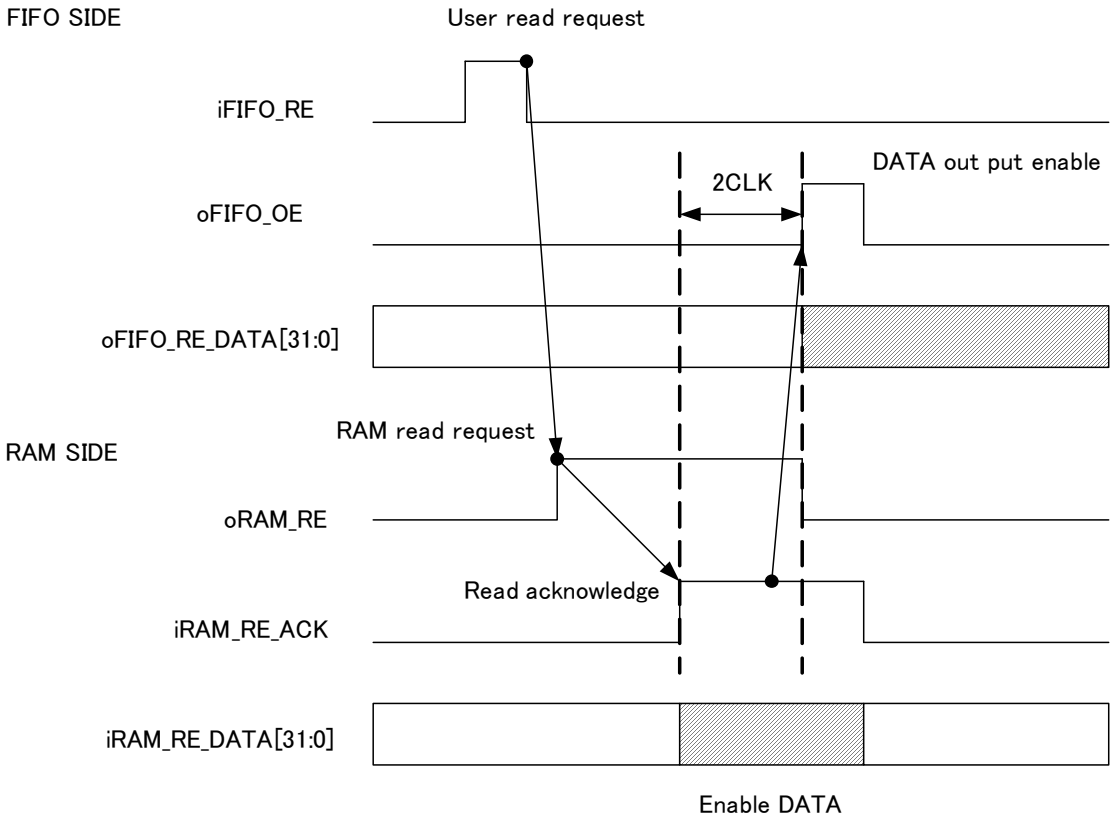


図 35　FIFO_ADAPTER リードタイミング図図

- ライトタイミング
  ライトはライトデータを iFIFO_WR_DATA に入力し iFIFO_WE をアサートして行う．書き込みの確認は行わないため, 連続書き込みを行う場合は RAM の仕様を確認する必要がある．

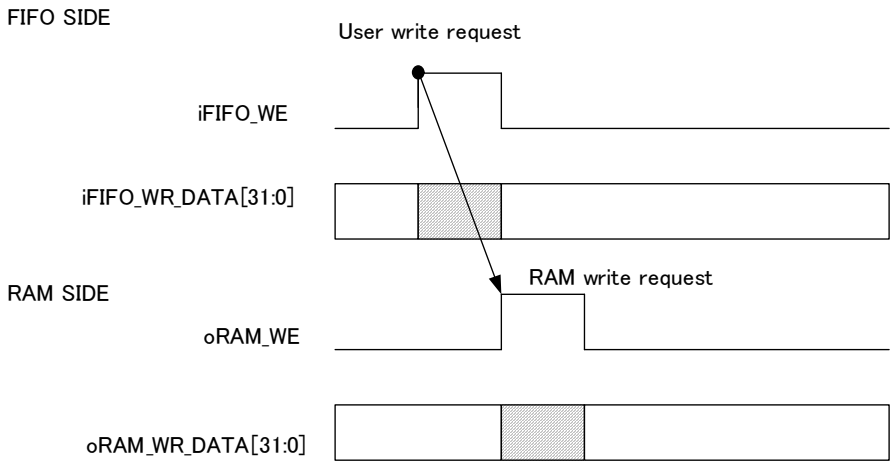


図 36　FIFO_ADAPTER ライトタイミング図

- ステータス更新タイミング リード
  oFIFO_AL_EMPTY は FIFO の保持データが 1 つの時アサートされる．oFIFO_EMPTY は FIFO が空の時アサートされる．リードは複数クロックにわたって行われるが, ステータスの更新は iFIFO_RE をアサートしたクロックで行われる．

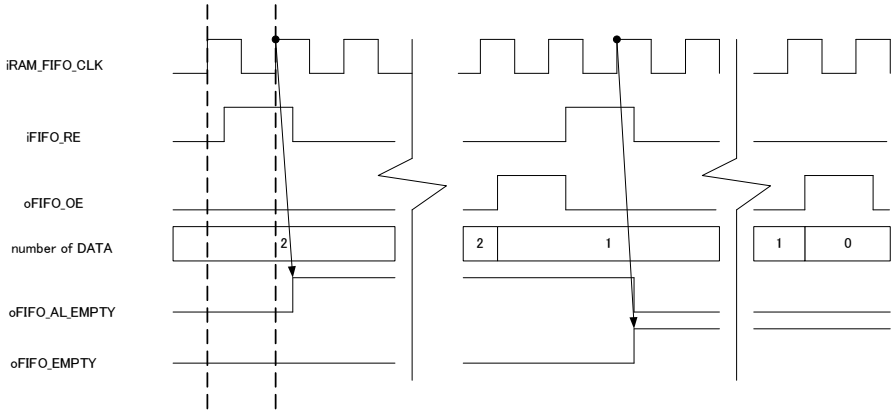


図 37　FIFO_ADAPTER ステータス更新 リードタイミング図

- ステータス更新タイミング ライト
  oFIFO_AL_FULL は FIFO の保持データが MAX-1 の時にアサートされる. oFIFO_FULL は FIFO が満たされておりデータ書き込みできない状態を表す. ステータスの更新は iFIFO_WE アサートの次のクロックで行われる.

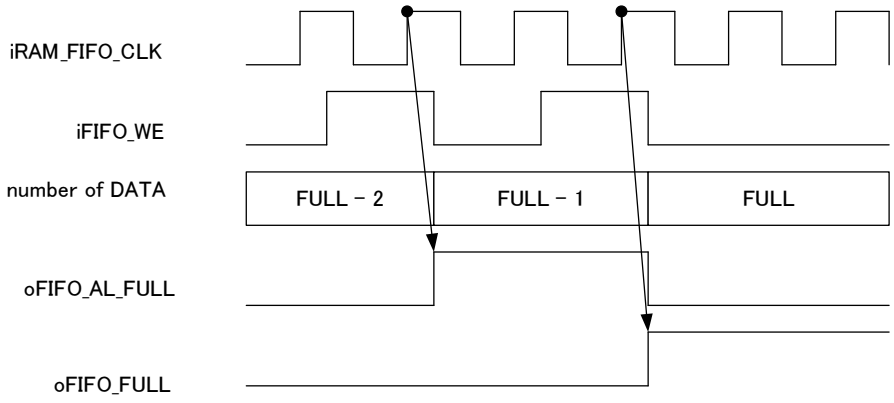


図 38　FIFO_ADAPTER ステータス更新 リードタイミング図

D.6 TRNG_ADAPTER

TRNG_ADAPTERは32bit幅の乱数を生成する. TRNGの構成はTRNG_PKG.vhdで宣言されるパラメータを変えることによって変更可能である (D.6.4章) . iTRNG_EN信号をアサートすると乱数出力が始まり, iCLRがアサートされるまで乱数を出力し続ける. 乱数生成中はoTRNG_ENがアサートされる. TRNG_CLKはiASSERT_NEGATE × 2 CLK周期のデューティー比50%に設定される. iASSERT_NEGATEを乱数生成中に変更することは禁止されており, 変更した場合の動作は保障しない. 出力データ (oTRNG_DATA) はoDATA_OEがアサートされた時に有効になる.

D.6.1 TRNG_ADAPTER 入出力信号

TRNG_ADAPTERの入出力信号を表15に示す.

表15 TRNG_ADAPTER入出力信号

NAME	BIT	I/O	FUNCTION
iCLK		I	システムクロック
iRESET		I	非同期リセット
iCLR		I	同期クリア CONTROLレジスタ
iTRNG_EN		I	乱数出力イネーブル
iASSERT_NEGATE	[4:0]	I	サンプリング周期設定
oTRNG_EN		O	乱数生成中
oDATA_OE		O	データ出力イネーブル
oTRNG_DATA	[31:0]	O	乱数出力

D.6.2　TRNG_ADAPTER ブロック図

TRNG_ADAPTER マクロは TRNG_ASSERTER, TRNG, TRNG_CONV32 マクロとシフトレジスタで構成される. TRNG_ASSERTER マクロは TRNG_CLK を生成する. TRNG マクロは TRNG の本体でラッチと XOR の回路である. TRNG_CONV32 は TRNG の出力を 32bit のパラレルデータに変換するシリアルパラレル変換マクロである. シフトレジスタは rTRNG_OFF1, 2 と rAL_NEGATE_WAIT が実装されている. rTRNG_OFF1, 2 は TRNG マクロのセットアップ/ホールドタイム違反によるメタステーブル対策のための 2 段組シフトレジスタである. TRNG マクロの出力は TRNG_CLK が入力されてから 4CLK (ラッチが 2 CLK, rTRNG_OFF1, 2 が 2 CLK) 遅れて出力される. TRNG_CONV32 のラッチタイミングもこれに合わせる必要があるため, rAL_NEGATE_WAIT シフトレジスタによってラッチタイミングを合わせている. LUT latch のレイテンシが変わる場合は rAL_NEGATE_WAIT のレジスタ長を調整する. rTRNG_EN は TRNG_ADAPTER マクロのイネーブルレジスタで iTRNG_EN がアサートされるとセットされ, iCLR が入力されるとクリアされる.

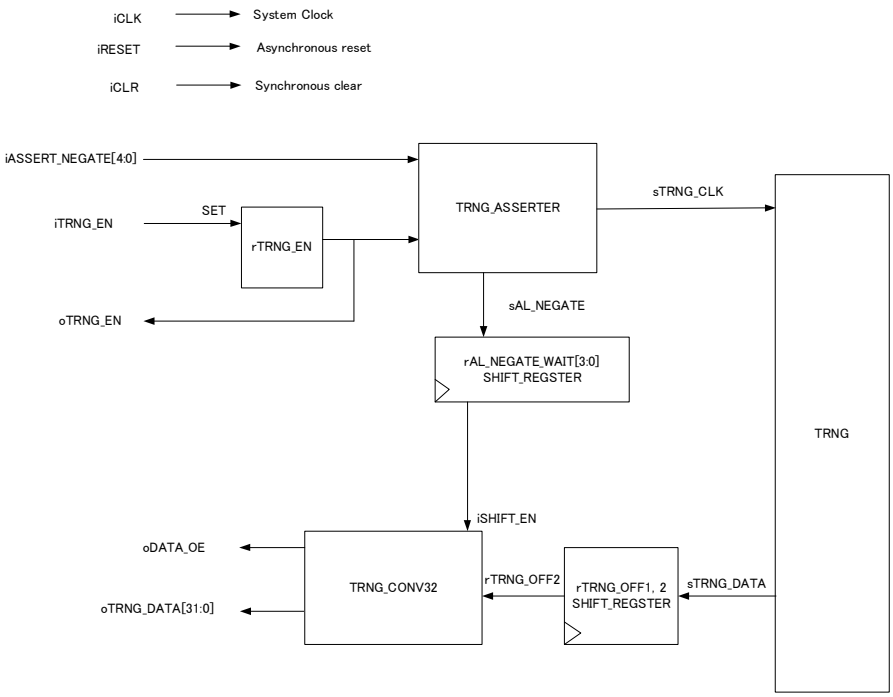


図 39　TRNG_ADAPTER ブロック図

D.6.3 TRNG_ADAPTER タイミング図

- TRNG_ADAPTER
  iTRNG_EN をアサートすると乱数生成が始まる．乱数出力は 32bit 幅で oDATA_OE がアサートされたときに有効になる．乱数出力中は oTRNG_EN がアサートされる．iCLR がアサートされると乱数生成が終了する．

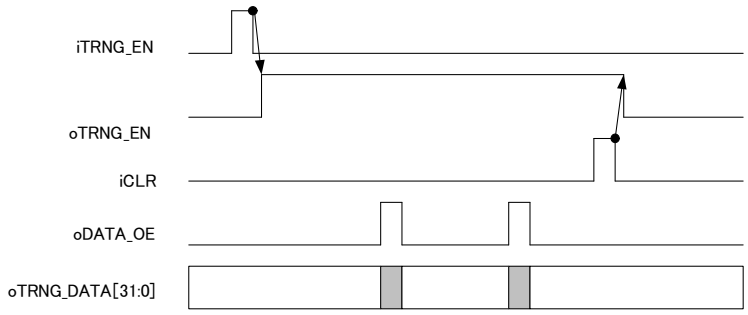


図 40　TRNG_ADAPTER タイミング図

- TRNG_ASSERTER
  TRNG_ASSERTER は TRNG_CLK と oAL_NEGATE 信号を生成する．oAL_NEGATE 信号は TRNG_CLK の立下りタイミングで 1 CLK だけアサートされる．
  TRNG_CLK はイネーブルが入力されて 1 CLK 後に出力が始まる．周期は iASSERT_NEGATE × 2 CLK でデューティー比 50%になる．

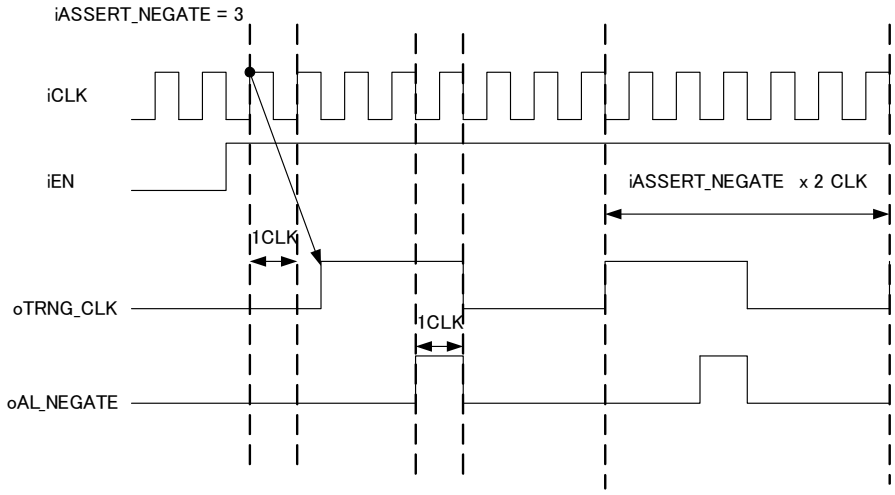


図 41　TRNG_ASSERTER タイミング図

- TRNG_CONV32 入力
  TRNG のラッチが LUT latch で構成される場合, TRNG_CLK が入力されてから出力されるまで 2CLK かかる. これに加えてメタステーブル対策レジスタのレイテンシが 2CLK あるので出力が TRNG_CONV32 に入力されるまでに 4CLK のかかる. そのため, TRNG_CONV32 のラッチタイミング (sAL_NEGATE) も 4CLK 遅らせて使用する (iSHIFT_EN). ラッチのレイテンシが変わる場合は iSHIFT_EN のレイテンシも調整する必要がある.

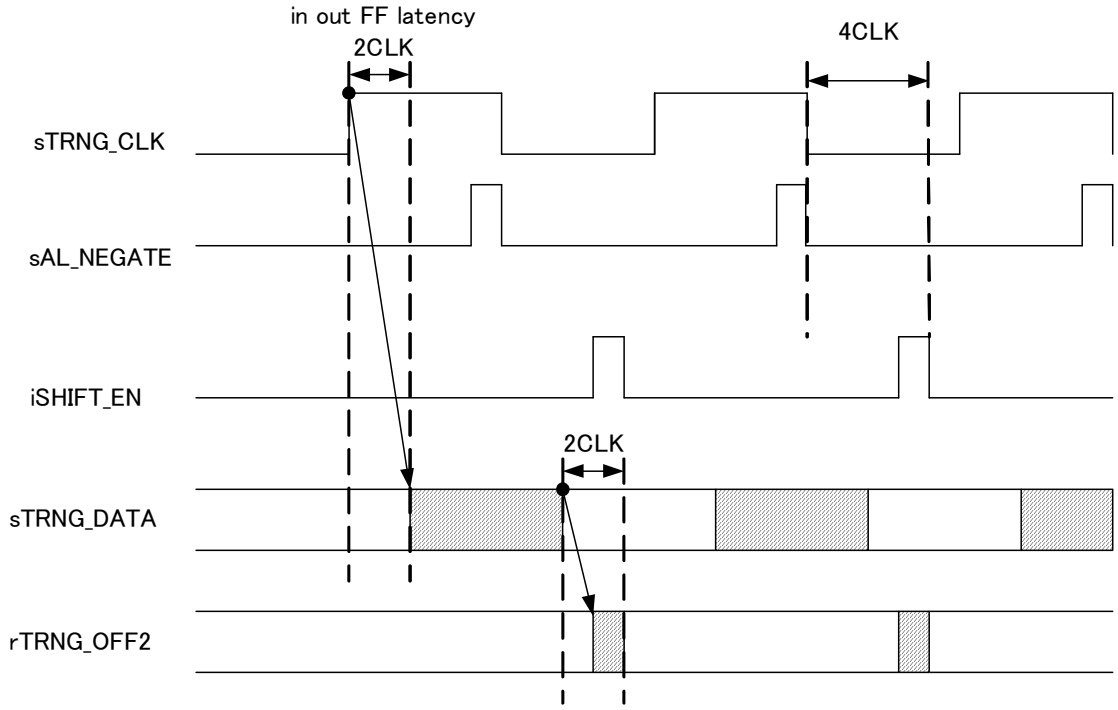


図 42　TRNG_CONV32 入力 タイミング図

D.6.4 TRNG

TRNG の構成は TRNG_PKG.vhd のパラメータで変更する．TRNG_PKG.vhd では以下の 3 つが変更できる．

- LUT_LATHC_NUM
  LUT latch n の n を設定する．LUT_LATCH_NUM パラメータは NUM_IN_HMACRO パラメータで割り切れる値が設定可能である．
- LATCH_DELAY
  ラッチのレイテンシを設定する．0 ならば no FF, 1 ならば in FF の構成のラッチが選択され, 2 を設定すると in out FF (LUT latch) が選択される．LATCH_DELAY を設定すると TRNG 出力のサンプリングタイミングも自動調整される．0 と 1 の設定で使用する場合は NUM_IN_HMACRO を 1 に設定しなければならない．
- NUM_IN_HMACRO
  1 つのハードマクロとして扱うラッチの数を設定する．1 ならば Lxxx, 2 ならば LLxxx が選択され, 4 を設定すると LLMM が選択される．2 と 4 の設定で使用する場合は LATCH_DELAY パラメータを 2 に設定しなければならない．