社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＣＳＥＣ－26　(11)
2004／7／20

素因数分解ハードウェアTWIRLの実現可能性に関する検討報告 (I)

伊豆　哲也†　伊藤　孝一†　小暮　淳†　小檜山清之†　下山　武司†
武仲　正彦†　鳥居　直哉†　桝井　昇一†　向田　健二†

†(株)富士通研究所/富士通株式会社

あらまし　TWIRLとはAdi ShamirとEran Tromerによって2003年8月に提案された素因数分解専用ハードウェアデザインの名称で,数体篩法における篩ステップを実現している.彼らの試算では,1024-bit合成数を素因数分解する(のに必要な篩ステップを処理する)のに,1000万ドル(約10億円)の費用と約1年の計算時間が必要であるとされている.筆者らはTWIRLの動作仕様の詳細調査と基本回路設計を行い,提案者の主張するハードウェアデザインと回路規模の見積もりを行ったので,その検証結果の概要を報告する.詳細に関しては,稿を改めて報告する予定である.
キーワード　素因数分解ハードウェア,TWIRL,評価,CRYPTREC

A Feasibility Analysis of the Factoring Device TWIRL (Part I)

Tetsuya IZU†, Kouichi ITOH†, Jun KOGURE†, Kiyoshi KOHIYAMA†, Takeshi SHIMOYAMA†,
Masahiko TAKENAKA†, Naoya TORII†, Shoichi MASUI†, and Kenji MUKAIDA†

† FUJITSU LABORATORIES LTD./FUJITSU LIMITED

Abstract　This document reports an overview of our evaluation results of the dedicated factoring device TWIRL by reviewing the circuit design assuming the current state-of-the-art-tecnologies.
Key words　Dedicated hardware for integer factorization, TWIRL, evaluation, CRYPTREC

1. はじめに

公開鍵暗号技術のいくつかは素因数分解問題の難しさに安全性の根拠をおいている.電子政府推奨暗号リスト[CRY02]では,合成数サイズが1024-bit以上の場合,素因数分解は向こう10年にわたり現実的に困難であろうことを想定している.従って,現実的に素因数分解問題が困難であるという事実は,安全な電子政府を構築する上で欠かせない前提となっている.

数体篩(ふるい)法は素因数分解アルゴリズムの中で最も優れた方法である[LL93].従来はソフトウェア実装による実験例しか報告されていなかった(注1)が,最近になって専用ハードウェア,特にLSIを用いたデザインが提案されている.TWIRLはイスラエルの暗号研究者Adi ShamirとEran Tromerによって2003年8月に提案されたハードウェアデザインであり[ST03],篩と呼ばれる処理を大規模に並列計算することを大きな特徴とする.1024-bit合成数をターゲットとしたTWIRLによる素因数分解のコストは,1000万ドル(約10億円)の費用(注2)と約1年の計算時間と試算されており,もしも試算通りに製造・動作が可能であれば,素因数分解問題を利用した公開鍵暗号技術の安全性が損なわれることになる.しかしこの見積もりは,提案者自身も認める通り概算値でしかなく,実現可能性の可否も含めた再検討の必要性が指摘されていた[ST03],[RSA03].

本報告書の目的は,現在電子政府推奨暗号リストを採用した際に想定されているパラメータ(合成数サイズ1024-bit)に対応するTWIRLの仕様を調査・検討し,実現可能性・実現時期・費用等の詳細な評価を行うことである.本稿では評価結果の概要を報告する.著者らの詳細な評価内容については,別の報告書を参照されたい[F04a],[F04b].

TWIRLの特徴:　TWIRL (The Weizmann Institute Relation Locator)とは,2003年8月にAdi ShamirとEran Tromerによって提案された素因数分解専用ハードウェアデザインの名称であり[ST03],最良の素因数分解アルゴリズムである数体篩法[LL93]の篩ステップをハードウェア実装したものである.このため篩ステップで使用する2種類の篩(有理数的篩(rational sieve)と代数的篩(algebraic sieve))に対応して,TWIRLはRational TWIRLとAlgebraic TWIRLの2つの構成要素からなる.TWIRLはLSIを効果的に利用することで,篩処理を高い並列度で計算することを特徴とする(1024-bit合成数をターゲットとしたRational

(注0):本研究は通信・放送機構の委託研究「暗号の技術評価に関する研究開発」の一環として行ったものである.
(注1):数体篩法のソフトウェア実装による分解記録は576-bitである[F+03].
(注2):1ドル=100円として換算,以下同様.

表1 TWIRL の回路規模見積もり

プロセスルール	算出者	Rational TWIRL	Algebraic TWIRL
0.13 μm	提案者	15960 mm^2	65960 mm^2
0.09 μm	提案者	7650 mm^2	31620 mm^2
	報告者	23892 mm^2	—

TWIRL の場合で 4096 並列). さらにこの並列処理は, 篩に伴う対数値の加算処理のみを高い並列度で実行し, これらの加算処理に対する制御を一つの回路にて集中管理することで, 典型的な篩の実装デザインと比較して大幅な回路規模削減を実現している. その結果, 1024-bit 合成数をターゲットとした TWIRL の回路規模は, 0.13 μm プロセスルールを使用した場合, Rational TWIRL の場合で 15960 mm^2, Algebraic TWIRL の場合で 65960 mm^2 になると提案者は試算している (表 1 参照). なお実際の分解にあたっては, Rational TWIRL 8 個と Algebraic TWIRL 1 個を組み合わせた TWIRL クラスタとして分解を行う. 1 年で分解を行うには 194 組の TWIRL クラスタが必要とされている.

調査・検討内容: 今回の評価においては, まず TWIRL の提案論文 [ST03] に従い, 具体的な仕様を調査・検討した. いくつかのパラメータについては見直しが必要と考え, パラメータの理論的な最適値を数値実験を通して定めた. これら仕様・パラメータについては, 概要を 3 節に示す.

次に具体化させた仕様に基づき, 1024-bit 合成数をターゲットとした TWIRL の基本回路設計を行った. 設計方針としては, TWIRL の性能を最も左右するパラメータである Delivery Line と Sieve Location の移動速度が, 提案者の想定通り, 内部周波数 1 GHz のときに 1 clock cycle/cell となるように設計した(注3). そしてこの回路設計を元に回路面積を算出し, TWIRL を複数 LSI に機能分割した場合の LSI 数を概算した. ここで回路設計で用いたパラメータとしては, 基本的には提案論文から導出される理論的な値を用いたが, 実装に伴うさまざまな制約から, 一部については修正した値を用いた. なお本報告書の目的は提案論文の詳細検討であるため, 論文の内容を越える改良は行っていない. しかし論文で前提となっているが詳細が記されていない項目については, 適当な補完を行った. 概要については 3 節を, 詳細については [F04a], [F04b] を参照されたい.

検討結果: 報告者が作成した基本回路設計を元に TWIRL の実現可能性を検討した結果, TWIRL は 2004 年 1 月時点のテクノロジーでは, 実際に製造することは極めて困難であるとの結論を得た. その主な理由は以下の 3 点に要約される.

(i) 提案者の主張通り TWIRL を単一の LSI として製造するには巨大なウェハー (直径 100 mm 以上) を製造する必要があるが, このようなウェハーを欠損なく製造することは不可能である.

(ii) 製造可能な大きさの複数の LSI に分割して TWIRL を実現する方法が考えられるが, 検討結果では, LSI 間の IO 数が多数必要なことから, Rational TWIRL だけでも多数の LSI (40 mm 角の LSI で 3362 個) を必要とする(注4) ため, 単一のボードに搭載不可能な規模(注5) となり, やはり製造不可能との結論に至った.

(iii) 次に分割した LSI を複数のボードに実装する方法が考えられる. しかし報告者が検討したところ, Rational TWIRL の場合, 確かに 54 枚程度のボードに理論的には実装可能(注6) であるものの, 提案者の主張する動作周波数 (1 GHz) を前提とした場合, ボード間の IO 数が多数 (最大 1500-bit) 必要なことから, やはり実現不可能である(注7).

これら考察結果により, 2004 年 1 月時点のテクノロジーでは, TWIRL を実際に実現することは極めて困難であると報告者は考える. なお本検討にあたっては, 結線の困難さは考慮していないため, 実際の実現可能性はさらに低いと思われる.

前述の通り, 報告者の回路設計は 1024-bit をターゲットとした TWIRL に対する提案者の仕様を忠実に反映させている. 従って半導体テクノロジーが劇的に向上した場合や, 仕様を越えた改良を施した場合の実現可能性は検討していない. また 512-bit 合成数や 768-bit 合成数をターゲットとした場合の TWIRL の実現可能性についても報告者は検討を行っていない.

以下, 各項目について詳細を述べるが, ことわりのない限り 1024-bit 合成数をターゲットとしたパラメータを用いる.

a) ウェハーの製造可能性

TWIRL を単一の LSI として製造するためには, 巨大な 1 枚のウェハーを製造する必要がある. 提案者の主張 (0.13 μm プロセスルールを使用) では, Rational TWIRL を製造するためには直径 150 mm ウェハー 1 枚, Algebraic TWIRL を製造するためには直径 300 mm ウェハー 1 枚を必要とする. また 0.09 μm プロセスルールを使用したとしても, 提案者の主張では Rational TWIRL に直径 104 mm ウェハー 1 枚(注8), Algebraic TWIRL に直径 208 mm ウェハー 1 枚を必要とし, 報告者による回路規模見積もりでは Rational TWIRL に直径 174 mm ウェハー 1 枚を必要とする (表 1 参照). しかしこれの大きさのウェハーを欠損なく製造することは, 現在の半導体技術では不可能である. 実際, 歩留まり率の関係から実用上製造可能な LSI は 40 mm 角が限界であろう.

b) 報告者による回路規模見積もり

次に TWIRL の複数 LSI への分割実装を検討するために, Rational TWIRL の回路規模の再評価を行った. その結果, 0.09 μm プロセスルールを用いた場合の回路規模は 23892 mm^2 となった. 同じプロセスルールを用いた場合の提案者による回路規模見積もりは 7640 mm^2 となることから, 約 3.1 倍の回路規模となっている (表 1 参照).

このように報告者の検討結果が提案者の見積もり結果よりも大規模になる最大の理由は, 配送機構, 特に Largish Station の Buffer と呼ばれるユニットと Smallish Station の Funnel と呼ばれるユニットにおいて, 4096 並列された加算器に対するデータの配送機能の回路規模を提案者が考慮していないことが原因である. Buffer/Funnel の回路デザインは慎重に行う必要がある. なぜなら, TWIRL が想定するパイプライン的な加算処理を実現するためには, 例えば Buffer の場合, 4096 通りの配送先を持つ対

(注3) : 従って移動速度や内部周波数を変更させた場合については考察していない. これは変更させた場合に回路設計方針が大きく変わってしまうからである.

(注4) : 0.09 μm プロセスルールを使用. また配送機構の補正を追加している.

(注5) : LSI 間の間隔を 10mm として正方形に配置した場合, (40mm+10mm) × 58 ≈ 3 m 角のボードを必要とする.

(注6) : 0.09 μm プロセスルールを使用. また配送機構の補正を追加している.

(注7) : 動作周波数 1 GHz の場合, ボード間 IO は最大 400 bit が限界であろう.

(注8) : ウェハーの直径は面積による単なる換算値である. 以下も同様.

数値データのソート処理を 1 cycle に 2000 個以上処理する必要があるからである. このソート処理を典型的なバブルソートを用いて行った場合, 非常に巨大な回路 (5.6G gates 以上) を必要とする. この課題に対し報告者が検討を行った結果, 小さい回路 (329M gates) による実現が可能であるとの結論を得た(注9).

c) 複数 LSI を用いた実現可能性

TWIRL を単一の LSI に実装することは不可能であることから, 前述の回路規模見積もりを用いて複数 LSI への分割実装を検討した. 検討にあたっては, 個々の LSI は 0.09 μm プロセスルールを用いて製造された 40 mm 角の大きさであり, 2004 年 1 月時点での標準的な仕様として, 回路規模 400M gates/LSI, IO ピン数 3000 pins/LSI (1500 bit を IO として使用可能(注10)) を仮定した. これら条件を元に Rational TWIRL が必要とする LSI 数を評価したところ, 3362 個の LSI が必要であるという結論を得た. LSI 間の IO ピン数を無視して回路規模のみの制約条件を考慮するならば 15 個の LSI で実現可能となることから, TWIRL の実装では IO ピン数がボトルネックとなっていることがわかる. 3362 個の LSI を 1 枚のボードに実装することを考えると, 正方形に並べた場合, 58×58 で配置する必要ある. LSI の間隔を 10mm として実装するならば, およそ 3 m×3 m のボードが必要となる. このように膨大な数の LSI を 1 枚のボードを実装・動作させることは, 現在の技術を遥かに超える範囲である. 以上により, 複数の LSI を用いて TWIRL を実現させた場合でも, LSI の IO ピン数がネックとなり, 必要となる LSI 数が膨大になり, 単一のボードに搭載して実現することは不可能であると考える.

d) 複数ボードを用いた実現可能性

複数 LSI を用いた分割実装のさらなる可能性として, 複数ボードを用いた実装についても検討を行った. ボードあたりの LSI 数を 100 個と仮定した場合, Rational TWIRL の見積もり面積から算出した理論的なボード数は 54 枚となった. しかし TWIRL の実装にあたってはボード間に 1500 bit の IO を必要とし, 提案者の想定する周波数 (1 GHz) を考慮すると, これは現在の技術レベルを遥かに越えており, 実現は不可能であると考える(注11).

e) TWIRL 実現に必要なブレークスルー

既存技術だけを用いた場合, TWIRL を実際に実装することは極めて困難であることをこれまでに述べた. しかし将来の技術進歩によって実現できる可能性は残されている. 前述の議論から, 単一の LSI を用いるよりも複数の LSI に分割する方が実現性は高いであろう. その場合 LSI 間およびボード間の IO 数がボトルネックとなっていたことから, TWIRL を実現するためには IO を高速かつ多ビット化する技術が必須であると考える(注12).

2. 数体篩法における篩処理

数体篩法 (NFS; Number Field Sieve method, [LL93]) は整数の素因数分解問題に対する最良のアルゴリズムとして知られている [IK03]. 数体篩法は, 前処理 (多項式生成) ステップ, 関係式探索 (篩) ステップ, 線型代数ステップ, 後処理 (平方根計算) ステップの 4 つのステップから構成される. このうち素因数分解計算の大部分を占めるのは, 関係式探索ステップと線型代数ステップである. TWIRL は関係式探索ステップを処理するデバイスであるので, 以下では関係式探索ステップの処理内容だけを説明していく. 数体篩法の動作原理, 他のステップの処理内容, 関係式探索ステップと他のステップとの関係などに関しては, 数体篩法に関する既存の文書 ([LL93], [IK03] など) を参照されたい.

整数の集合を $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots,\}$ で, 自然数の集合を $\mathbb{N} = \{1, 2, \ldots,\}$ であらわす.

整数の組 (a', b) が以下の 3 条件を満たすとき, 組 (a', b) を関係 (relation) と呼ぶ.

(1) $\gcd(a', b) = 1$

ただし $\gcd(a', b)$ は整数 a', b の最大公約数を表す.

(2) $f_R(a', b)$ は B_R-smooth

ただし 1 次の整数係数 2 変数多項式 $f_R(x, y) = mx + y$, 自然数 B_R は与えられているとする.

(3) $f_A(a', b)$ は B_A-smooth

ただし d 次の整数係数 2 変数多項式 $f_A(x, y) = (-x)^d F(-y/x)$, d 次の既約な整数係数 1 変数多項式 $F(z) = c_d z^d + \cdots + c_0$ ($c_i \in \mathbb{Z}$), 自然数 B_A は与えられているとする.

ここで自然数 N が自然数 B に対して B-smooth であるとは, N の最大素因数が B 以下のこと, あるいは N が B 以下の素因数の積に分解できること, つまり N が $N = \prod_{p_i \leq B} p_i^{e_i}$ と表せることをいう. このとき自然数 B を smoothness bound と呼ぶ.

数体篩法における関係式探索ステップの目標は, 与えられた 2 次元領域 $\{ (a', b) \in \mathbb{Z} \times \mathbb{Z} \mid -R/2 \leq a' \leq R/2,\ 1 \leq b \leq H \}$ から, 上の 3 条件を満たす関係 (a', b) を規定の個数以上見つけることである. 定数 $R \in \mathbb{N}$ を篩幅 (sieve line width), $H \in \mathbb{N}$ を篩の本数 (sieve line number) と呼ぶ. 例えば 1024-bit 合成数に対する TWIRL では $R = 1.1 \times 10^{15}$, $H = 2.7 \times 10^8$ を使用する.

以下では多項式 $f_R(a', b)$, $f_A(a', b)$ および自然数 R, H, B_R, B_A は関係式探索ステップへの入力として与えられていると仮定する. また関係の条件 2. と条件 3. は類似しているため, 以下では簡単のため, 多項式 $f(a', b)$ は $f_R(a', b)$ または $f_A(a'b)$ のいずれかを, B は B_R または B_A のいずれかを表すこととする.

関係式探索ステップの基本的な動作は, 固定された b に対し, $f(a', b)$ が B-smooth となる a' を $\{-R/2, \ldots, R/2\}$ の中から見つけ出すことである. ここで $f(a', b)$ が B-smooth であるとは

$$f(a', b) = \prod_{p_i \leq B} p_i^{e_i} \tag{1}$$

となることであったから, $f(a', b)$ を B 以下の素数 p_i で順番に割っていくことで, $f(a', b)$ が B-smooth であるかを判定することはできる. しかし除算は大量の計算リソースを必要とするため, 効率的に処理するには好ましくない.

そこで TWIRL は, 以下で紹介する篩 (sieve) を用いることで, 関係の効率的な探索を可能にしている. まず式 (1) は

(注9) : ただし報告者のソート処理のデザインは, 入力データに偏りが生じた場合非常に低い確率でデータ破棄が発生する. しかしこの確率は 0.000737 と非常に低く, 現実的には問題がない範囲であると考える.

(注10) : 3000 pins のうち, IO として 1500 pins が使用可能であると仮定した (残りの pins は Vcc, GND 用である).

(注11) : 1 GHz で動作する IO 数はおよそ 400 bit が限界と報告者は考える.

(注12) : あくまでも必要条件であって, 十分条件ではない.

$$\log f(a',b) = e_i \sum_{p_i \mid f(a',b),\ p_i \le B} \log p_i \approx \sum_{p_i \mid f(a',b),\ p_i \le B} \lfloor \log p_i \rceil \quad (2)$$

と近似できることに注意する．ここで $\lfloor x \rceil$ は実数 x の四捨五入，すなわち x に最も近い整数を表す．式 (2) のように近似できるのは，ほとんどの大きな素因数 p_i に対しては $e_i = 1$ となることと，メモリ効率の観点から対数の値を整数に丸めた方が好ましく，誤差は後で調整可能なことが理由である．実際の処理は以下のようにして行う：

> あらかじめ $f(a',b)$ に対応する変数 v (初期値 0) を用意し，整数 $f(a',b)$ が素数 p_i で割り切れる場合，v に $\lfloor \log p_i \rceil$ を加える．この操作を B 以下の素数全てに対して行い，最終的に v の値が $\log f(a',b)$ に近い場合，(a',b) を関係の候補として扱う．候補となった $(a'b)$ に対しては，正しい素因数分解を通じて，本当に関係になっているかを判定する．

なお v の値が $\log f(a',b)$ に近いかどうかの判定は，v の値が近似値であることを考慮して，b によって定まる値 $T(b)$ に対し，$v > T(b)$ かどうかで判定する．$T(b)$ としては $\log f(a',b)$ の 0.8 倍程度の値を使用する．$T(b)$ を閾値 (threshold) と呼ぶ．一般に閾値を緩くすれば，たくさんの a' が候補として得られる．

篩は上記の処理を同時に行うことが特徴である．いま多項式の選択方法から，$f(a',b)$ が p_i で割り切れるならば，$f(a'+p_i,b)$ も p_i で割り切れる，という性質が成立する．そこで上記の処理をまとめて行うために，始めに $f(-R/2,b), f(-R/2+1,b), \ldots, f(R/2,b)$ に対応する $R+1$ 個の変数 $v_{-R/2}, v_{-R/2+1}, \ldots, v_{R/2}$ (初期値は全て 0) を用意する．次に，素数 p_i に対し $f(-R/2,b)$ から順に p_i で割り切れるかを調べていく．$f(a'_0,b)$ で初めて p_i で割り切れるとすると，$f(a'_0+p_i,b)$ も p_i で割り切れることになるので，結局 $f(a'_0,b), f(a'_0+p_i,b), f(a'_0+2p_i,b), \ldots$ は p_i で割り切れることが分かる．よって素数 p_i に対しては，変数 $v_{a'_0}$, $v_{a'_0+p_i}$, $v_{a'_0+2p_i}, \ldots$ に $\lfloor \log p_i \rceil$ を加えれば良い．この操作を B 以下の全ての素数 p_i について繰り返していけば，$f(a',b)$ が B-smooth となる a' を $-R/2 \le a' \le R/2$ の中からまとめて見つけることができる．このように a' をまとめて見つける操作が，篩 (sieve) と呼ばれる理由であり，数体篩法の名前の由来でもある．a'_0 が見つかってしまえば，除算は必要ないことに注意が必要である．

数体篩法では，各 line に対して 2 種類の篩処理が必要で，1 つ目の篩を有理数的篩 (rational sieve)，2 つ目の篩を代数的篩 (algebraic sieve) と呼んで区別する．そして 2 つの篩のどちらにも残った (a',b) を候補 (candidate) と呼び，候補は篩の後に以下の最終チェックを受ける．最終チェックでは 2 種類の関数値 $f_R(a',b)$, $f_A(a',b)$ を実際に分解し，本当に B-smooth であるかを確かめる(注13)．そしてこのチェックにパスした (a',b) を関係 (relation) と呼ぶ．篩ステップの出力は関係の集合である．

以下では簡単のため，また文献 [ST03] の記述に合わせるため，$a = a' + R/2$ ($a \in \{0, \ldots, R\}$) とする．また a', b の多項式 $f(a',b)$ と a, b の多項式 $f(a,b)$ を上の変数変換の下で同一視する．

(注13)：実際には 2+2 large prime variation を用いる [LL93]．

3. TWIRL の基本構造

TWIRL は，篩ステップの有理数的ふるいと代数的ふるいに対応して，有理数的 TWIRL (Rational TWIRL) と代数的 TWIRL (Algebraic TWIRL) から構成されているが，パラメータ値以外の構造はほとんど同じであるため，以下の説明では主に Rational TWIRL に絞って説明していく．ただし Algebraic TWIRL は Rational TWILRL に合格した a だけを処理するカスケード構造になっている．以下では 1024-bit 合成数の素因数分解を目的としたパラメータを用いて説明する．具体的なパラメータは 2 重括弧 《·》 をつけて表す(注14)．そのパラメータが Rational TWIRL に関するならば $《x》_R$, Algebraic TWIRL に関するならば $《x》_A$, 両者に関すならば単に $《x》$ のように記す．

3.1 篩の並列処理

関係式探索ステップが処理するラインの本数を H 《$= 2.7\times10^8$》本，各ラインの幅を R 《$= 1.1\times10^{15}$》とする．また smoothness bound を B 《$= 3.5\times10^9$》$_R$, 《$= 2.6\times10^{10}$》$_A$ と定める．

TWIRL の大きな特徴は，関係式探索ステップのふるい計算を並列に処理する点である．まず TWINKLE のように 1 clock cycle あたり 1 つのふるい位置 a を処理するデバイスを考える．閾値を $T = T(b)$ とするとき，このデバイスは $\log_2 T$ 《$= 10$》bit 幅の一方向性なデータパスを持ち，何百万もの条件判断機能付き加算器 (conditional adder) を連続して持つ．それぞれの条件判断機能付き加算機は 1 つの Progression $P_i = \{ a = a_0^{(i)} + kp_i \mid k = 0, 1, \ldots \}$ を担当し，動作時刻になるとパスに $\lfloor \log p_i \rceil$ を加える(注15)．データ $\lfloor \log p_i \rceil$ を p_i に対応する Contribution と呼ぶ．時刻 t の時，z 番目の加算器は $t-z$ 番目のふるい位置を処理する．パイプラインには 1 clock cycle あたり 1 つの速さでふるい位置のデータが流れ，0 を先頭に $1, \ldots, R$ が流れていく．

次にこのデバイスの並列化を行う．先ほどと同じ $\log_2 T$ bit 幅のパスを s 《$= 4,096$》$_R$, 《$= 32,768$》$_A$ 本用意し，時刻 t の時には z 番目の加算器の i 本目のパスは $(t-z)s+i$ 番目のふるい位置を処理するように変更する．ここで各パイプラインの先頭になるふるい位置を $0, 1, \ldots, s-1$ とする．このように変更することで，1 clock cycle あたりに処理するデータを s 個にすることが可能となる．ただし 2 つの問題が生じてしまう．1 つは時刻の進みが $1/s$ になるため，それに応じた処理が必要になることである．もう 1 つは s 本のパスを同時に並列処理するために，同じ Progression P_i に対応する加算 $\lfloor \log p_i \rceil$ を同時に行う必要が生じることである．そこで同じ Progression P_i を担当する送信器を複数使用するのだが，素数によって頻度が異なるため，素数の大きさによって 3 種類の Station と呼ばれるユニットを併用する．具体的には，大きな素数 (Largish Prime) に対しては Largish Station，小さな素数 (Smallish Prime) に対しては Smallish Station，とても小さな素数 (Tiny Prime) に対しては Tiny Station を用いる(注16)．各 Station は決められた範囲の素数を担当し，パイプライン状に連結されてい

(注14)：これらパラメータは提案者の推奨値である．

(注15)：ここで $\lfloor \log p_i \rceil$ は対数値 $\log p_i$ に最も近い整数 (つまり四捨五入) を表す．

(注16)：数体ふるい法では Large/Small Prime という概念もあるが，Largish/Smallish/Tiny Primes とは別の概念であり，注意が必要である．

る．最後の Station の出力後に閾値判定を行うユニットを配置し，その出力をデバイスの出力とする．Rational/Algebraic TWIRL の概念図を図 1 に示す．ただし Station によって Progression の送信器の構造が異なるため，図では省略した．

素因数分解 (の関係式探索) 装置としての TWIRL は，複数個の Rational TWIRL が 1 個の Algebraic TWIRL に連結され，クラスタとして機能する．1024-bit 合成数に対しては，《8》個の Rational TWIRL が 1 個の Algebraic TWIRL に連結される (図 2 参照)．クラスタの出力は関係 (a, b) の集合である．なお提案者によれば，1024-bit 合成数の分解に必要なクラスタ数は《194》組と試算されている (4 節参照)．

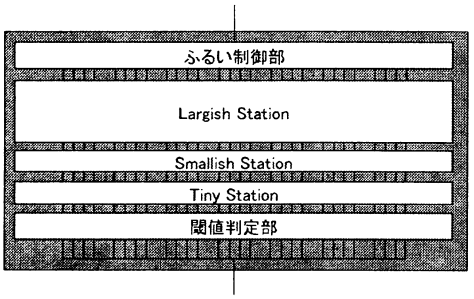

図 1　Rational/Algebraic TWIRL の概念図

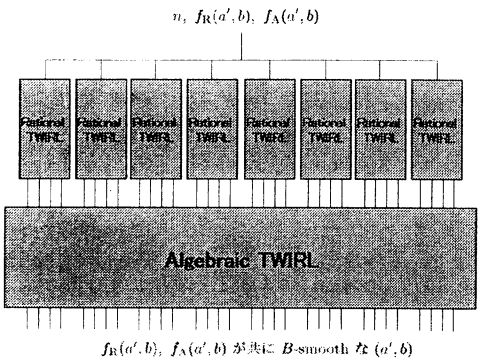

図 2　TWIRL クラスタ

以下，各 Station の構造を簡単に説明する．

3.2 Largish Station

素数 p_i に対応する Progression P_i の送信器は，Contribution $\lfloor \log p_i \rfloor$ を周期 p_i/s で発行する．従って $p_i/s \gg 1$ の時の発行頻度は非常に小さいため，1 個の送信器が 1 組の Progression を担当する構造は非常に効率が悪い．そこで送信器は複数の Progression を担当し，Progression 情報の通り道となる Delivery Line を複数の素数で共有すれば，大幅な回路削減に直結する．これが Largish Station の基本的なアイディアである．実際には素数 p_i が $《5.2\times10^5 < p_i < 3.5\times10^9》_R$, $《4.2\times10^6 < p_i < 2.6\times10^{10}》_A$ の場合を Largish Prime と呼び (このとき $127 < p_i/s_R < 854493$, $127 < p_i/s_A < 793458$ となる)，以下の構造を用いる．

Largish Station は，Progression 送信器 (DRAM Memory + Cache + Processor)，Buffer，Delivery Line から構成される．$C《= 8,490》_R$, $《= 59,400》_A$ 個の送信器は唯一の Buffer に接続され，Progression 情報は送信部から Buffer を経て Delivery Line に送られる．Progression は Progression Triplet として表現され，Memory に保存されている．Progression Triplet は Processor によって周期的にチェックを受け，該当する場合には情報が Emission Triplet として Buffer に送信されるとともに，新しい状態情報に更新される．Buffer は複数のプロセッサから送信された Emission Triplet を受け取り，時刻情報の順に保持し，適当なタイミングで Delivery Pair を発行する．Delivery Pair はパイプライン化された Delivery Line (Cell が鎖状に連結したもので，各 Cell はデータの転送と加算器の機能を持つ) に送信される．

3.2.1 Progression 情報の更新

Progression 情報は送信器内のメモリに保管されている．ここで Largish Prime に対する Progression の集合 $\{P_i\}$ は分割して送信器に保管され，重複はないものとする．j 番目の送信器が保持する Progression の個数を d_j 個とする (ただし $《32 \leq d_j \leq 2.2\times10^5》_R$, $《32 \leq d_j \leq 2.0\times10^5》_A$ で，個数は送信器によって異なる)．Progression Triplet は $(p_i, \ell^{(i)}, \tau^{(i)})$ で表されるデータの組で，p_i は素数の値，次の発行情報 $a^{(i)} \in P_i$ に対して $\tau^{(i)} = \lfloor a^{(i)}/s \rfloor$ は Contribution が足される時刻，$\ell^{(i)} = a^{(i)} \bmod s$ は Contribution が足される Bus Line 番号を表す．

Processor は以下の操作をパイプライン方式で実行する：

(1)　Progression Triplet $(p_i, \ell^{(i)}, \tau^{(i)})$ を読み込み，このデータを Memory から消去する．

(2)　Emission Triplet $(\lfloor \log p_i \rfloor, \ell^{(i)}, \tau^{(i)})$ を Buffer に送信する．

(3)　$\tilde{\ell}^{(i)} \leftarrow (\ell^{(i)} + p_i) \bmod s$, $\tilde{\tau}^{(i)} \leftarrow \tau^{(i)} + \lfloor p_i/s \rfloor + w$ を計算する．ただし $w = 1$ if $\tilde{\ell}^{(i)} < \ell^{(i)}$, $w = 0$ otherwise.

(4)　Progression Triplet $(p_i, \tilde{\ell}^{(i)}, \tilde{\tau}^{(i)})$ を Memory に書き込む．

Emission Triplet $(\lfloor \log p_i \rfloor, \ell^{(i)}, \tau^{(i)})$ が Buffer に送信されるのは，時刻 τ_i の直前であって欲しい (早すぎると Buffer のメモリがあふれてしまうし，遅すぎると Delivery Line への送信に間に合わない)．従って Processor が読み込む Progression Triplet は発行直前のものを読み込むようにしたい．各 Progression Triplet の読み書き位置を固定し，必要な情報をスキャンする方式では，今のように Progression Triplet の個数 d_j が大きいときには非効率的である．次節のように時刻 $\tau^{(i)}$ によってインデックス付けされた位置に保管することで，メモリ上の効率的な配置が可能となる．

3.2.2 Progression の保管方法

Processor はメモリ (DRAM) 上に記憶された Progression Triplet を一定の速度で連続的・巡回的に読み込む (《 1 triplet あたり 2 clock cycle 》)．読み込んだデータが空の場合には何もせずに次のデータを読み込む．そうでない場合，Processor は前述のようにして新しいデータを計算し，適当な位置，つまり時刻 $\tau^{(i)}$ の少し前に読み込まれる位置に更新されたデータを書き込む．このようにして Processor は発行直前の Progression Triplet を読み込んでいく．Progression Triplet は最初の空の位置に書き込まれる．書き込み位置が空である確率を高めるために，Memory は必要量の《2》倍を確保する．

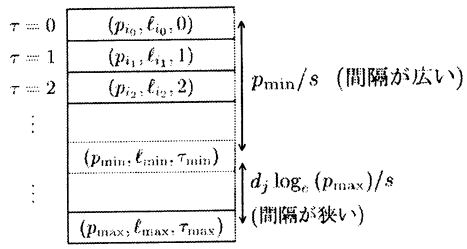

図3 Progression の保管

もしも適切な場所から《64》個以内に空の場所がない場合(注17), Progression Triplet は任意の場所 (または専用のオーバーフロー領域) に書き込まれ, 後に適切な位置に移動するようにするが, 場合によってはこれらの Emission が失われる可能性もある. しかしこのようなことが起きる確率は希であるし, 例え失われたとしても十分に許容範囲内である (つまり関係式探索ステップで関係を規定個数以上集められる) と提案者は主張している.

j 番目の送信器は d_j 個の連続した素数 $\{p_{\min}, \ldots, p_{\max}\}$ を担当するとしよう. この情報を記録するのに必要な Memory 量は $p_{\max}/s$ ワード (ただし 1 ワードは 1 個の Progression Triplet を記録するのに必要なデータ量) である. Progression Triplet を読み込んだ際, $p_i = p_{\max}$ の場合には, 更新後の Progression Triplet は読み込み位置から最も離れている. p_i がそれより小さい場合には, だんだんと読み込み位置に近づくことになり, $p_i = p_{\min}$ の場合が最も近い. 素数定理によると $p_{\max}/s - p_{\min}/s \approx d_j \times \log_e(p_{\max})/s$ となるから, 更新後の書き込み位置の幅は $d_j \times \log_e(p_{\max})/s < d_j \times \log_e(B)/s \ll d_j$ となり, その幅は非常に狭い (図3参照). したがってメモリをシリンダ状に配置し, 小さなウィンドウが一定の速度でメモリの周りを動くようにすれば, 読み書きに伴うメモリへのアクセスはウィンドウを通して行われる(注18). シリンダ状のメモリは素数の個数に応じてさまざまな大きさで各送信器に配置される.

ウィンドウは SRAM ベースの高速なキャッシュによって実現できる. 最も古いデータを DRAM に書き込むことと, 次の DRAM のデータを読み込むことによって, キャッシュのデータはシフトされていく. よって SRAM と DRAM の間に適切なインターフェイスを用いれば, DRAM の高レイテンシ(注19) を緩和することが可能となると同時に, DRAM の構造が簡単に, さらには DRAM は少量で済むことになる.

3.2.3 Buffer

Buffer は複数の Processor から Emission Triplet を受け取り, Delivery Line に対して Delivery Pair を送信する. 従って Buffer の中心となる動作は, 受け取った Emission Triplet を時刻情報を基に適切に配置し, 適切なタイミングで変換した Delivery Pair を Delivery Line に送信することである. つまり Emmision Triplet $(\lfloor \log p_i \rfloor, \ell^{(i)}, \tau^{(i)})$ は, Delivery Pair $(\lfloor \log p_i \rfloor, \ell^{(i)})$ に変換され, $\ell^{(i)}$ に応じて適切な Delivery Line に時刻 $\tau^{(i)}$ の時に送信される.

これらの条件を満足させるために, TWIRL では以下のように Buffer を設計する. Buffer は複数のキューとパイプライン状のネットワークから構成される. まず入力される全ての Emission Triplet は時刻情報をインデックスとしたキューに入力される. ここでキューはメッシュ状になっていて, 行方向では常にバブルソートが, 列方向では常にランダムシャッフルが行われている. 最後の数行のデータの時刻情報は現在の時刻情報と比較され, マッチしたデータはパイプライン状のネットワークにおいて Bus Line 番号をインデックスとして並べ換えられた後, 適当な Delivery Line に対して送信される. 輻輳の影響により Delivery Line への送信が遅れることもあり得るが, その場合にはデータを破棄することにする (しかし適切なパラメータの下では, このようなことが起きるのは極めて希であると主張されている).

Buffer のサイズは, 送信前の Emission Triplet をどれだけ保持するかということ (これは Processor の構造から小さい値となる) と, 各プロセッサが発行する Progression Triplet の速度 (およそ《4 clock cycle / triplet》) によって決まる.

3.2.4 Delivery Line

Delivery Line は Buffer から Delivery Pair $(\lfloor \log p_i, \ell^{(i)} \rfloor)$ を受け取り, $\lfloor \ell^{(i)}/k \rfloor$ clock cycle 後に加算を行う (ここで Delivery Pair は 1 clock cycle あたり k《= 4》本の Bus Line を進むとする). Delivery Line は Bus Line と直交する Cell の 1 次元配列として実装され, Cell は 1 組の Delivery Pair を保持する. Cell は自分の Bus Line 番号 $\ell^{(i)}$ を比較し, 等しい場合には Bus Line に $\lfloor \log p_i \rfloor$ を加えてから Deliver Pair を破棄する. 等しくない場合には, シフトレジスタのように, 隣の Cell に Delivery Pair を送る.

各送信器から発行される Progression Triplet は, 平均的には 4 clock cycle につき 1 個であるから, Delivery Line は送信器の 1/4 程度の割合で用意すれば良い. 提案者による Delivery Line の本数は $\langle\langle 2, 100 \rangle\rangle_{\mathrm{R}}$, $\langle\langle 14, 900 \rangle\rangle_{\mathrm{A}}$ 本であり, 装置の大部分を占める.

3.2.5 Progression Triplet のサイズ

Progression Triplet (p_i, ℓ, τ) を保持するには, データの有無を表すフラグに 1 bit, 素数 p_i を保持するのに $|p_i|$《= 32》$_{\mathrm{R}}$ bit, Bus Line 番号 ℓ を保持するのに $|s_{\mathrm{R}}|$《= 12》$_{\mathrm{R}}$ bit が必要である. また時刻情報 τ については, 前述の Processor, Memory, Buffer の説明では clock cycle を表す任意の整数であることを仮定していたが, τ は Progression Triplet が Memory から読み込まれてから, Buffer から送信されるまでの時間を表せれば良いので, 適当な整数《2048》による剰余で現実的には十分である. よって τ を保持するには $\log_2 2048 = 11$ bit で良い. さらに初期化で必要となる $|p_i|$《= 32》$_{\mathrm{R}}$ bit を考慮すると, Progression Triplet のサイズは $12 + 2|p_i| + \log_2 s$《= 88》$_{\mathrm{R}}$ bit となる.

3.3 Smallish Station

送信器が担当する素数 p_i が s と同じくらいの大きさの時 ($\langle\langle 256 < p_i < 5.2 \times 10^5 \rangle\rangle_{\mathrm{R}}$, $\langle\langle 256 < p_i < 4.2 \times 10^6 \rangle\rangle_{\mathrm{A}}$ すなわち $\langle\langle 0 < p_i/s < 127 \rangle\rangle_{\mathrm{R}}$, $\langle\langle 0 < p_i/s < 129 \rangle\rangle_{\mathrm{A}}$), Progression の発行周期は Largish Station の時に比べて短くなる. 送信器は《2》clock cycle 毎に高々 1 個の Progression しか発行できないので, Small-

(注17) : 1 つの Progression Triplet が必要とするメモリ量を 1 個と表現している.
(注18) : これがデバイス名 "twirl"(回転する) の本当の由来であろう.
(注19) : データの読み出し要求を行ってから, 実際にデータが転送されるまでにかかる遅延時間.

1つの素数を複数の emitter が担当する
- 素数 p_i を n_i 個の emitter が担当
- 各 emitter が担当する Delivery Line のグループ $G_1, \ldots, G_{n_i}$
- 素数 p_i に対応する emission の周期 $T_i = p_i(s/n_i)$

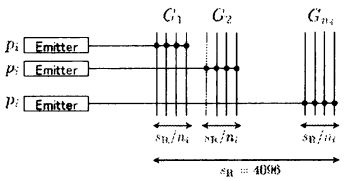

図 4 Segment 分割

ish Prime に対しては各送信器が担当する素数をあまり多くすることができない. つまり Largish Station のような構造はコスト的に不利になってしまう. また Progression が頻繁に発行されることから, 各 Progression が一ヶ所から発行される場合, 遠くの Bus Line まで送信するコストは大きい. 従ってこれらの Smallish Prime に対しては, Largish Station とは異なる別のデザインが必要であり, これが Smallish Station を提案する理由となる.

3.3.1 Emitter

Smallish Station における Progression の送信器を Emitter と呼ぶ. Emitter は単一の Progression を担当し, Delivery Pair を生成する小さな回路である. Emitter は Progression の状態を内部に持つレジスタに保持し, 適宜 Delivery Pair を発行する. ここで Delivery Pair $([\log p_i], \ell^{(i)})$ は, 一定時刻後に $\ell^{(i)}$ 番目の Bus Line に対して $[\log p_i]$ が加算されることを表す.

Smallish Station では発行周期が短いため, 同じ素数 p_i を担当する Emitter を複数用意して, 各 Emitter の周期を増加させる. これを Emitter の重複 (duplication) と呼ぶ. ただし素数 p_i に対応する Emitter の個数を $n_i = \lfloor s/2\sqrt{p_i} \rfloor_2$ 個とする ($n_i = \langle\!\langle 2, 4, 8, 16, 32, 64 \rangle\!\rangle_R$) (注20). 各 Emitter は内部のカウンター情報によって随時更新され, Delivery Pair を T_i ($= \lfloor \sqrt{p_i}/2 \rfloor_2$) clock cycle に 1 回発行する ($T_i = \langle\!\langle 8, 16, 32, 64, 128 \rangle\!\rangle_R$).

Emitter の重複によって, Delivery Pair の生成源から目的の Bus Line までの距離を短くすることができる (図 4 参照). 各素数に対応する Progression は n_i 個の Emitter によって保持され, これら Emitter はバス上に一定間隔で配置される. これに呼応して Delivery Line を s/n_i ($\approx 2\sqrt{p_i}$) 本ずつの Segment に分割する. 各 Emitter は異なる Segment に接続され, この Segment に対して Progression を T_i ($= n_i p_i / s \approx \sqrt{p_i}/2$) clock cycle ごとに送信する. Progression が到着するまでの時間は短縮されるので, Delivery Line の本数を減らすことができる. またいずれの Emitter においても Emission の周期は短くなるので, Smallish Station では s 程度の (あるいはそれよりも小さい) 素数 p_i を扱うことができる. Smallish Station 全体で必要な Delivery Line の本数はおよそ $\langle\!\langle 501 \rangle\!\rangle_R$ 本で, これら Delivery Line は異なるサイズの Segment に分割される.

各素数 p_i に対応する Progression の送信周期は $T_i \approx \sqrt{p_i}/2$ であるので, 1 clock cycle あたりに送信される Contribution は, 平均で $\sum 1/T_i$ 個となる. Funnel の構造を考慮すると, Delivery Line としてこの 2 倍程度の本数を用意すれば良い.

3.3.2 Delivery Line の本数

3.3.3 Funnel

Smallish Station でも Largish Station と同様に複数の Progression で Bus Line を共有するが, 各 Emitter を全ての Bus Line と接続するのは無駄が多い. そこで Funnel と呼ばれる 2 段のユニットを用いて, まばらな入力を以下のようにして圧縮する. 各 Funnel の入力は多数の Emitter の出力に接続される (各 Emitter の接続先となる Funnel は 1 個). Funnel の入力は 1 次元配列 (ただしそのほとんどの成分は空) と考えることもできる. Funnel の出力も 1 次元配列だが, その長さはずっと短く, 出力の配列の空でない成分の個数は, 何 clock cycle か前の入力となった配列の空でない成分の個数と同一になっている. Funnel の出力は Delivery Line に接続される. なお Funnel は Largish Station における Buffer に相当するユニットであるが, 時間情報によるソート機能が省略されている.

3.4 Tiny Station

とても小さな素数 $\langle\!\langle p_i < 256 \rangle\!\rangle$ に対しては, Buffer や Funnel のコストは不利である. またこのような素数は各 clock cycle ごとに複数個の Emission を必要とするので, いくつかの Progression に Delivery Line を割り当てるのは得策でない. 従ってこれら Tiny Primes に対しては 3 つめのデザインが必要となる. Tiny Station では Smallish Station と同様に, 同じ素数に対して複数の Emitter を用意する.

4. 提案者によるコスト見積もり

提案者等の試算によると, 1 つの Rational TWIRL が必要とするシリコンの面積は 15,960 mm^2 (30 cm シリコンウェハー (約 66,000 mm^2) の 1/4) となる. このうち Largish Station は 76 % (DRAM は全体の 37 %), Smallish Station は 21 %, Timy Station は残りである 3 % を占めている. 同様に, 1 つの Algebraic TWIRL が必要とするシリコンの面積は 65,900 mm^2 (30 cm シリコンウェハー 1 枚) であり, このうち Largish Station は 94 % (DRAM は全体の 66 %), Smallish Station は 6 % を占めている.

1 組の TWIRL クラスタは, 8 個の Rational TWIRL と 1 個の Algebraic TWIRL から構成され, 全体で必要なシリコンの面積はウェハー 3 枚 ($1/4 \times 8 + 1 = 3$) となる. 各 Rational TWIRL は Algebraic TWIRL に対して単方向的接続をしていて, 13 bit/clock cycle でデータを転送する. クラスタは 1 本のラインをふるうのに, スループットで R/s_A clock cycle (周波数 1GHz で約 33.4 秒) を必要とするので, 全領域 (H 本のライン) をふるうには, 約 286 年 ($33.4 \times H = 9.02 \times 10^9$ 秒) が必要である. さらに篩位置のインターリーブを用いると約 33 % の高速化が可能であり, 約 194 年 ($= 286 \times 0.67$) に改善できる. 従って 194 組のクラスタを用いれば, 約 1 年でふるいが終了することになる. ウェハー 1 枚の製造コストを 5,000 ドル (約 50 万円) と仮定した場合, このときの製造コストは合計で約 290 万ドル (約 2 億 9000 万円) となる.

パッケージ, 電力供給, 冷却装置などの費用を考慮し, エラーマージンを加味すると, 1024-bit 合成数の素因数分解に必要な

(注20): $\lfloor x \rfloor_2$ は x を 2 のべきに丸めた値. すなわち $\lfloor x \rfloor_2 = 2^{\lfloor \log_2 x \rfloor}$ を表す.

表2 提案者による TWIRL の概算コスト (1024-bit)

	Rational TWIRL	Algebraic TWIRL
プロセスルール	0.13 μm	
周波数	1 GHz	
集積度	論理回路: 2.8×10^{-6} mm^2/transistor, DRAM: 0.2×10^{-6} mm^2/bit	
Largish Station の比率 (DRAM の比率)	76 % (37%)	94 % (66%)
Smallish Station の比率	21 %	6 %
Tiny Station の比率	3 %	0 %
チップ面積	15,960 mm^2/1 個 (30 cm ウェハー 1/4 枚)	66,000 mm^2/1 個 (30 cm ウェハー 1 枚)
	合計 30 cm ウェハー 3 枚/クラスタ 1 組	
計算時間 × クラスタ数	1 年 × 194 組	
コスト	製造コスト: 1000 万ドル (約 10 億円)/一式 NRE: 2000 万ドル (約 20 億円)	

ふるいを 1 年で終了させるのに必要な製造コストは, およそ 1000 万ドル (約 10 億円) となる(注21). この他に設計, シミュレーション, マスク製造などに対し, 初期開発費 (NRE; Non-Recurring Expenses) として 2000 万ドル (20 億円) 程度が必要となる. 以上をまとめると表 2 のようになる.

a) 詳細な回路規模見積もり

回路規模の詳細見積もりを以下に示す.

表3 提案者による Rationl TWIRL の詳細コスト (1024-bit)

Largish Station	12,130 mm^2	76 %
DRAM	5,905 mm^2	37 %
	167268917 個 × 2 倍 × 88 bit/triplet × $(0.2\times10^{-6}$ mm^2/bit)	
CPU	2,292 mm^2	14 %
	96400 trans/個 × 8490 個 × $(2.8\times10^{-6}$ mm^2/trans)	
Cell	3,191 mm^2	20 %
	530 trans/個 × 2100 本 × 1024 本 × $(2.8\times10^{-6}$ mm^2/trans)	
Buffer など	742 mm^2	5 %
Smallish Station	3,352 mm^2	21 %
Emitter	1,053 mm^2	7 %
	2037 trans/個 × 184562 個 × $(2.8\times10^{-6}$ mm^2/bit)	
Cell	1,752 mm^2	11 %
	1220 trans/個 × 501 本 × 1024 本 × $(2.8\times10^{-6}$ mm^2/bit)	
Funnel など	584 mm^2	3 %
Tiny Station	478 mm^2	3 %
合計	15,960 mm^2	100 %

表4 提案者による Algebraic TWIRL の詳細コスト (1024-bit)

Largish Station	61,946 mm^2	94 %
DRAM	43,494 mm^2	66 %
	1133598328 個 × 2 倍 × 94 bit/triplet × $(0.2\times10^{-6}$ mm^2/bit)	
CPU	16,033 mm^2	24 %
	96400 trans/個 × 59400 個 × $(2.8\times10^{-6}$ mm^2/trans)	
Cell	708 mm^2	1 %
	530 trans/個 × 14900 本 × 32 本 × $(2.8\times10^{-6}$ mm^2/trans)	
Buffer など	1,711 mm^2	3 %
Smallish Station	3,954 mm^2	6 %
Emitter	2,719 mm^2	4 %
	2037 trans/個 × 476779 個 × $(2.8\times10^{-6}$ mm^2/bit)	
Cell	137 mm^2	0 %
	1220 trans/個 × 1250 本 × 32 本 × $(2.8\times10^{-6}$ mm^2/bit)	
Funnel など	1,098 mm^2	2 %
Tiny Station	— mm^2	0 %
合計	65,900 mm^2	100 %

b) 0.09 μm での見積もり

Shamir 等はその後の論文において, プロセスルールを 0.09 μm に変更した場合の見積もりを示している [LTS+03]. その論文によると, 変更に伴って回路面積は約 1/2 倍, 速度は約 2 倍となり, パラメータを改良することで, 全体として約 9 倍のコスト改良が可能であるとしており, 0.13 μm の時と同じクラスタ数が 110 万ドル (約 1 億 1000 万円) で製造可能と結論づけている (必要な計算時間は 1 年のまま変わらない).

4.1 ハードウェアに関するパラメータ詳細

コスト算出に用いたハードウェアに関するパラメータは以下の通りである [LSTT02]: シリコンウェハーは 0.13 μm テクノロジーで考え, 300 mm の標準的なウェハー 1 枚の製造コストは 5,000 ドル (約 50 万円) とする. 1024-bit 合成数と 768-bit 合成数に対しては DRAM タイプのウェハーを使用し, トランジスタ密度は 2.8×10^{-6} mm^2/transistor (論理領域での平均値), DRAM 密度は 0.2×10^{-6} mm^2/bit (DRAM バンク領域での平均値) とする. 512-bit 合成数に対してはロジックタイプのウェハーを使用し, トランジスタ密度は 2.38×10^{-6} mm^2/transistor, DRAM 密度は 0.7×10^{-6} mm^2/bit とする. いずれの場合でもパイプラインを使用することから, 周波数は 1 GHz として考える.

以下パラメータの詳細を説明する. Largish Station で使用する特殊なプロセッサが必要とする transistor 数は $\langle\!\langle 96,400 \rangle\!\rangle_R$ であると仮定する (ただしバッファーと《14Kbit》のキャッシュメモリを含む. キャッシュの大きさは p_i とは独立である). Smallish Station で使用する Emitter が必要とする transistor 数は $\langle\!\langle 2,037 \rangle\!\rangle_R$ (Funnel も含む), Tiny Station で使用する Emitter が必要とする transistor 数は $\langle\!\langle 522 \rangle\!\rangle_R$ であると仮定する. Delivery Line 上の cell は, interleaving あり (Largish Station 用) で $\langle\!\langle 530 \rangle\!\rangle_R$ transistor, interleaving なし (Smallish/Tiny Station 用) で $\langle\!\langle 1,220 \rangle\!\rangle_R$ transistor が必要であるとする. 3.2 節で説明したメモリ構造では, 通常の DRAM に比べてスラック領域とキャッシュ領域が多めに必要となるので, 通常よりも $\langle\!\langle 2.5 \rangle\!\rangle_R$ 倍の領域を占めるとする. Bus Line では, ワイヤの重層実装が可能なことから, ラインは領域を必要とせず, レジスタのみ領域を使用するとする. ワイヤのクロスには $\langle\!\langle 0.5 \rangle\!\rangle$ bit/μm を使用するとする.

5. ま と め

素因数分解専用ハードウェア TWIRL の評価報告を行った. 著者らの検討の結果, 2004 年 1 月時点のテクノロジーでは, TWIRL を実際に製造することは極めて困難であるとの結論を得た. 詳細な評価内容については [F04a], [F04b] を参照されたい.

(注21) : このような場合, 通常はシリコンの製造コストの 2 倍を見積もるのに対し, 提案者は 3 倍として見積もっている.

文 献

[CRY02] 情報処理振興事業協会 (IPA), 通信・放送機構 (TAO), "暗号技術評価報告書 (2002 年度版)", March, 2002.

[F+03] J. Franke and others, "RSA-576", Email announcement, December 2003.

[F04a] (株) 富士通研究所/富士通株式会社, "素因数分解装置の調査・検討に関する報告書" 2004 年 2 月.

[F04b] 伊豆, 伊藤, 小暮, 小檜山, 下山, 武仲, 鳥居, 桝井, 向田, "素因数分解ハードウェア TWIRL の実現可能性に関する検討報告 (II)" 電子情報通信学会技術報告書, ISEC 2004 (2004-07), July 2004.

[H04] (株) 日立製作所, "素因数分解専用集積回路等の実現性についての評価" 2004 年 2 月.

[IK03] 伊豆 哲也, 木田 祐司, "素因数分解の現状について", 日本応用数理学会論文誌 Vol. 13, No. 2, pp.289-304, 2003.

[LL93] Arjen K. Lenstra and H.W. Lenstra (eds.), "The development of the number field sieve", Vol. 1554 in Lecture Notes in Mathematics (LNM), Springer-Verlag, 1993.

[LSTT02] Arjen K. Lenstra, Adi Shamir, Jim Tomlinson and Eran Tromer, "Analysis of Bernstein's circuit", *ASIACRYPT 2002*, LNCS 2501, pp.1-26, Springer-Verlag, 2002.

[LTS+03] Arjen K. Lenstra, Eran Tromer, Adi Shamir, Wil Kortsmit, Bruce Dodson, James Hughes and Paul Leyland, "Factoring estimates for a 1024-bit RSA modulus", *ASIACRYPT 2003*, LNCS 2894, pp.55-74, Springer-Verlag, 2003.

[RSA03] RSA Security, "TWIRL and RSA Key Size", May 2003.

[ST03] Adi Shamir and Eran Tromer, "Factoring large numbers with the TWIRL device", *CRYPTO 2003*, LNCS 2729, pp.1-26, Springer-Verlag, 2003.