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

株式会社日立製作所

素因数分解専用集積回路等の実現性についての評価

(株) 日立製作所

2004年2月

要旨

本報告では，素因数分解専用の集積回路に関する既存の提案に対して，その実現性に関する評価を行ないその結果を報告する．

評価は主に二つのフェーズから行なった．一つ目は，既存技術において，どの方式が最も評価に値する手法であるかを検討し，かつその方式の問題点を特定する調査作業，二つ目は注目した評価対象に対する詳細な評価結果である．

評価は，アルゴリズム技術者による解析のグループと，装置自体の実装を評価するハードウェア技術者による評価グループとで行ない評価作業をまとめた．

結果として，RSA-1024 型の実用暗号利用で懸念される合成数を分解する実現性は極めて低いと判断した．その判断理由の大きなポイントは以下の通りである:

1. (前提) 現状，最も有力な素因数分解の手法は，一般数体篩法，およびその改良法であり，今後これを脅威的に上回るアルゴリズム的ブレイクスルーはない，と仮定する．数体篩法では全体の計算コストの中で篩処理が最も大きな部分を占める．

2. (アプローチ)1024 ビット合成数を対象とした一般数体篩法における篩処理の実現性を主張する方式として，現在，30 cm ウェハ上で構成したカスタムハードウェアを複数個用いて篩処理を行なう [49] が提案されており，現状，他方式に比べコスト優位性を持つとみられている．これはアイデアレベルであるので，ある程度の改良，アイデアの精密化などにより，計算コストなどの定数オーダーの変更はありうることを考慮に入れた上で，装置設計の規模，現実性という観点からの評価を行う．

考察 1 現状の提案はアイデアのレベルにとどまっており，詳細な装置仕様を記述するに至っていない (第 6.1 節)．詳細な仕様を作成することは自明な作業ではなく，多くの障壁を乗り越えなければならないことが予想される．

考察 2 30 cm ウェハ規模の巨大な論理回路自体が非現実的である．その製造費用を考える以前に，その製造方法が存在しない．また，その代替案も困難であることが予想され，初期費用以前の問題である．仮に製造したとしても (主要な面積部分に対して)1 GHz クロックによる動作を期待することはできない (第 5.2 節, 第 8 章)．

考察 3 故障耐性については原著にて付録として扱っている．しかし，我々の評価では，独立性の低い高並列な回路構成ゆえに，多くの出力に故障が起こることが想定されるなど，原著の検討で故障耐性が十分であるとは考えない．さらに，現状の半導体技術の経験から，今回の評価対象に我々が考える適切な故障耐性を持たせる場合，全体でもかなりの部分を占めるパートについては，場合により大規模な冗長回路構成とする必要がある．すなわち, 30 cm ウェハに原著者が主張する機能を搭載するのは困難と考えるのが妥当である (第 5.3).

考察 4 電力消費からくる発熱に対する冷却，および動作中のメンテナンスなども動作のコストとして計上されるべきであるが，これらは 30 cm ウェハという大きな回路であり，現実的にどう克服してよいか，ノウハウがない (第 7 章).

考察 5 その他，クロックツリーの構築 (第 7.3 節)，冗長回路の埋め込み，試運転段階の動作検査用論理などサポート技術 (第 5.3 節)，製造上のサイズマージンなど (第 8 章)，実世界での実装を試みた場合に回路規模の増加が強く見込まれる.

考察 6 以上の懸念の多くは装置が占めるウェハ上の面積に関わる部分であり，半導体技術が進歩し，集積度があがればこれらは克服できる可能性がある.

目 次

表 目 次

図 目 次

第1章　素因数分解と一般数体篩法

1.1 一般数体篩法

数体篩法 (Number Field Sieve) は合成数の素因数分解を行うアルゴリズムとして 1990 年に A.K.Lenstra, H.W.Lenstra, M.S.Manasse, J.M.Pollard らによって考案された ([33]). 最初は特殊な条件のついた数体を用いる方法 (特殊数体篩法) であり, 対象とする合成数も特殊な形に限定されていたが, 多くの数学的障壁を乗り越えることで, 一般的な数体を用いて, 任意の形の合成数を扱うことができるように改良された. これが一般数体篩法であり, さらなる改良や新しいアイデアが加えられ現在, 素因数分解法として最も強力な方法となっている.

1.1.1 基本方針

一般数体篩法の概略を説明する. 高速化や, より一般的な状況への対策方法など種々の研究結果が知られているが, ここでは最も基本的な方針についてのみ言及することとする.

合成数 n の素因数分解をすることを考える. $f(x) \in \mathbb{Z}[x]$ を 1 変数 monic 既約 d 次整係数多項式とする. $\theta \in \mathbb{C}$ を f の一つの根:

$$f(\theta) = 0 \tag{1.1}$$

とし, $\mathbb{Z}$ 上 θ で生成される環

$$\mathbb{Z}[\theta] := \{a_0 + a_1\theta + \cdots + a_{d-1}\theta^{d-1} \mid a_i \in \mathbb{Z}\}$$

を考える. さらに, $m \in \mathbb{Z}$ を

$$f(m) \equiv 0 \bmod n \tag{1.2}$$

を満たすものとする. このとき, 環準同型

$$\phi : \mathbb{Z}[\theta] \longrightarrow \mathbb{Z}/n\mathbb{Z}, \quad g(\theta) \mapsto g(m) \bmod n \tag{1.3}$$

が自然に定義できる. ここで, 互いに素な整数の組 (a, b) の集合 S であって,

$$\prod_{(a,b)\in S} (a + bm) \quad \text{は } \mathbb{Z} \text{ において平方数}, \tag{1.4}$$

$$\prod_{(a,b)\in S} (a + b\theta) \quad \text{は } \mathbb{Z}[\theta] \text{ において平方数} \tag{1.5}$$

を満たすものを見つけることができたとする. $\prod(a+bm) = x^2$, $x \in \mathbb{Z}$, $\prod(a+b\theta) = \mu^2$, $\mu \in \mathbb{Z}[\theta]$ とおく. このとき上記準同型により, $\phi(a+b\theta) = a+bm \bmod n$ であるから

$$\phi(\mu^2) = x^2 \bmod n \tag{1.6}$$

が成り立つ. $\phi(\mu) = y \bmod n$ $(y \in \mathbb{Z})$ とすれば,

$$y^2 \equiv x^2 \bmod n \tag{1.7}$$

が成り立ち, 従って $\gcd(x-y, n)$ を計算することで n の因数が見つかる可能性がでてくる.

以上が数体篩法の基本方針である. 次に, 多項式 $f(x)$ や法 n での根 m の構成法の一例や, 条件 (1.4), (1.5) を満たす集合 S の求め方について説明する.

1.1.2 多項式 f と整数 m の構成法

ここでは最も素朴な"base m"法と呼ばれる構成法を紹介する.

素数べきでない正整数 n が与えられたとき, 1 変数整係数 monic 多項式 $f(x)$ と, 整数 m であって, $f(m)$ が n の倍数であるものを構成する.

整数 $d > 1$ を $n > 2^{d^2}$ を満たすようにとり, $m = [n^{1/d}]$ (Gauss 記号) とおく. n の m 進展開を

$$n = c_d m^d + c_{d-1} m^{d-1} + \cdots + c_0, \quad c_i \in \mathbb{Z},\ 0 \le c_i < m$$

とする. このとき d, m の取り方から,

$$c_d = 1, \quad c_{d-1} < d$$

であることが容易にわかる.

ここで多項式 $f(x) \in \mathbb{Z}[x]$ を

$$f(x) = x^d + c_{d-1} x^{d-1} + \cdots + c_0 \tag{1.8}$$

とおく. $f(m) = n$ であることに注意 (求める条件より強い). ただし, この段階では $f(x)$ は既約とは限らない. しかし, もし $f(x) = g(x)h(x)$ のように分解されたとす

ると, $n = f(m) = g(m)h(m)$ であるが, $g(m), h(m) \neq \pm 1$ であるならばこれは n の非自明な分解を与えることになる. また, $g(m) = \pm 1$, または $h(m) = \pm 1$ ならば他方を改めて $f(x)$ とし, 再度, 既約性の判定を行えばよい.

1.1.3 集合 S

条件 (1.4), (1.5) を満たす集合 S は, 大まかに次の 2 つのステップに分けて構成される.

(i) 互いに素な整数の組 (a, b) の集合 T で, $a + bm$, $a + b\theta$ がともに smooth であるものを探す (smooth の定義は後述).

(ii) 線型代数を用いて条件を満たす集合 $S \subseteq T$ を求める.

さらに, (i) は, smooth な $a + bm$ を探すこと (これを数体篩法における"**rational sieve**"という) と, smooth な $a + b\theta$ を探すこと (これを"**algebraic sieve**"という) に分けられる.

まず"rational"部分について説明する.

u を (n に依存する) 大きな正整数とし,

$$U := \{(a, b) \mid a, b \in \mathbb{Z}, \gcd(a, b) = 1, |a| \leq u, 0 < b \leq u\}$$

とおく. 次にパラメータ $y = y(n)$(正の実数) を固定し,

$$T_1 := \{(a, b) \in U \mid a + bm : y\text{-smooth}\}$$

とする. ここで, 整数 a が **y-smooth** であるとは, a の全ての素因子が y 以下であることをいう.

T_1 は"**篩**"(sieve) によって求められる:

各 $0 < b \leq u$ に対し, $a + bm$, $-u \leq a \leq a$ が入った配列を考える. 各素数 $p \leq y$ に対し, 各配列の要素 $a + bm$ が $a \equiv -bm \bmod p$ を満たす ($a + bm$ が p で割れる) ならば, 要素を配列からいったん取り出し, p で割れる限り割り続け, 最後に得られた商を同じ配列に戻すという作業を行う. この作業が終了したとき, 最終的に配列に残っている数は, a に対応する配列について, $a + bm$ の, y 以下の全ての素数と素な最大の約数ということになる. もしある a に対応する配列の要素が最終的に ± 1 と

なっているならばこれは $a+bm$ が y-smooth であることを意味し, T_1 の元を見つけたことになる.

y 以下の素数の集合を篩の **factor base** という. y 以下の素数の個数を $\pi(y)$ で表す.

パラメータ y および u は,

$$\#T_1 > \pi(y) + 1 \tag{1.9}$$

を満たすように取られていると仮定する.

$B = \pi(y)$ とおき, p_j $(1 \leq j \leq B)$ を j 番目の素数とする. また $p_0 = -1$ とおく. y-smooth な整数

$$w = \prod_{j=0}^{B} p_j^{e_j}$$

に対し, ベクトル $e(w) \in \mathbb{F}_2^{B+1}$ を次で定義する:

$$e(w) = (e_0 \bmod 2,\ e_1 \bmod 2,\ \ldots,\ e_B \bmod 2)\,.$$

各 $(a,b) \in T_1$ に対し, 2 元体 $\mathbb{F}_2$ 上の $B+1$ 次元ベクトル $e(a+bm)$ を作れば, 仮定 (1.9) よりベクトルの個数がベクトル空間 $\mathbb{F}_2^{B+1}$ の次元 $B+1$ を真に超える. よってこれらのベクトルには ($\mathbb{F}_2$ 上) 非自明な線型関係が存在する (1 次従属). すなわち, 空でない部分集合 $S \subseteq T_1$ が存在して,

$$\sum_{(a,b)\in S} e(a+bm) = 0 \ \in \mathbb{F}_2^{B+1}$$

となる. これは

$$\prod_{(a,b)\in S} (a+bm) = x^2 \text{ in } \mathbb{Z}$$

すなわち, 平方数であることを意味する. 以上により, 線型代数の計算を経て, (1.4) を満たす集合 S を探すことができる.

次に algebraic 部分について説明する.

$\alpha \in \mathbb{Z}[\theta]$ が y-smooth であるとは, ノルム $N(\alpha) \in \mathbb{Z}$ が y-smooth であることと定義する.

素数 p に対し, $f \bmod p$ の ($\mathbb{Z}/p\mathbb{Z}$ での) 根の集合を次のようにおく[1].

$$R(p) := \{r \in \mathbb{Z}/p\mathbb{Z} \mid f(r) \equiv 0 \bmod p\}.$$

整数 $0 < b \leq u$, $b \not\equiv 0 \bmod p$ を固定する.
$a + b\theta \in \mathbb{Z}[\theta] (a, b(\neq 0) \in \mathbb{Z})$ の形の元のノルムは

$$N(a + b\theta) = b^d f(-a/b)$$

により与えられる. このとき次が成り立つ.

$$\begin{aligned} N(a + b\theta) \equiv 0 \bmod p &\Leftrightarrow b^d f(-a/b) \equiv 0 \bmod p \\ &\Leftrightarrow -a/b =: r \in R(p) \\ &\Leftrightarrow a \equiv -br \bmod p \text{ for some } r \in R(p). \end{aligned}$$

集合 U を rational のときと同様, さらに類似として次の集合を考える:

$$T_2 := \{(a, b) \in U \mid a + b\theta : y\text{-smooth}\}.$$

ここで"篩"により, 集合 U の中から T_2 に含まれる元を抽出する:

rational sieve と同様に, b を固定し, $N(a + b\theta)$, $-u \leq a \leq u$ が入った配列を考える. b を割らない各素数 $p \leq y$ と, 各 $r \in R(p)$ に対し, $a \equiv -br \bmod p$ が成り立つ (すなわち $N(a + b\theta) \equiv 0 \bmod p$) 配列の要素 $N(a + b\theta)$ を取り出し, p で割れる限り割り続け, 最後に得られた商を同じ配列に戻すという作業を行う. この作業が終了したとき, ± 1 が入っている配列が y-smooth な $a + b\theta$ に対応している, 従って T_2 の元が得られたということになる.

$a, b \in \mathbb{Z}$, $\gcd(a, b) = 1$, および素数 p と $r \in R(p)$ に対し,

$$e_{p,r}(a + b\theta) = \begin{cases} \mathrm{ord}_p(N(a + b\theta)) & \text{if } a + br \equiv 0 \bmod p, \\ 0 & \text{otherwise,} \end{cases}$$

とおく[2]. このとき

$$N(a + b\theta) = \pm \prod_{p,r} p^{e_{p,r}(a+b\theta)} \tag{1.10}$$

[1] $\mathbb{Z}[\theta]$ の 1 次の素イデアルと, 組 (p, r) $(r \in R(p))$ の間に 1 対 1 対応が存在し, $\alpha \in \mathbb{Z}[\theta]$ が smooth であることの定義は, $\mathbb{Z}[\theta]$ の単項イデアル (α) がノルムが y 以下であるような 1 次の素イデアルの積として表されることと同値である.

[2] ord_p は有理整数上の order 関数をあらわす: $\mathrm{ord}_p(a) = t \overset{\mathrm{def}}{\Leftrightarrow} a = p^t a', \gcd(a', p) = 1$.

が成り立つ (積は素数 p 全体と $r \in R(p)$ 全体にわたる).

ここで $B' := \#\{(p,r) \mid p < y : \text{素数},\ r \in R(p)\}$ とおくとき, $\#T_2 > B'$ ならば, rational sieve で述べた線型代数を用いた方法と同様の方法により, 空でない部分集合 $S \subset T_2$ であって, 任意の (p,r) $(p \le y)$ に対し,

$$\sum_{(a,b)\in S} e_{p,r}(a+b\theta) \equiv 0 \bmod 2 \tag{1.11}$$

が成り立つようなものを見つけることができる. しかし実はこの条件 (1.11) は, 求めたい条件式 (1.5) のためには (すなわち平方数であるためには) 必要であるが, 十分であるには程遠い. この差異を埋めるための手法として, Adleman([1]) による平方剰余記号 (Legendre 記号) の値を補助条件に用いる方法を以下で説明する.

素数 q に対し, $\left(\frac{*}{q}\right)$ を平方剰余記号とする.

(1.3) で定義した全射準同型写像 $\phi : \mathbb{Z}[\theta] \to \mathbb{Z}$ と $\left(\frac{*}{q}\right)$ の合成写像を χ_q と書くこととする. このとき $\mathbb{Z}[\theta]$ のある素イデアル $\mathfrak{q}$ が存在して, $\chi_q(\mathbb{Z}[\theta] \setminus \mathfrak{q}) = \{\pm 1\}$ となる.

$\alpha \in \mathbb{Z}[\theta]$ が平方数であるためには, 素数 q であって, 対応する素イデアル $\mathfrak{q}$ について $\alpha \notin \mathfrak{q}$ を満たすものに対し, $\chi_q(\alpha) = 1$ が成り立つことが必要で, 十分多くの素数についてこれが成り立つことは平方数であることの強い証拠になり得ることが示される. algebraic 部分では条件 (1.11) に加え, このアイデアを用いて条件 (1.5) を満たす S を構成する.

Q をいくつかの素数に対する χ_q からなる有限集合とする[3].

$(a,b) \in T_2$ と $\chi_q \in Q$ に対し,

$$\chi_q(a+b\theta) = (-1)^{e_q(a+b\theta)}, \quad e_q(a+b\theta) \in \{0,1\}$$

とおき, さらに (a,b) に対し, $\mathbb{F}_2$ 上の $B' + \#Q$ 次元ベクトル

$$\left((e_{p,r}(a+b\theta))_{p,r},\ (e_q(a+b\theta))_{\chi_q \in Q} \right)$$

を作る. $\#T_2$ が $B' + \#Q$ を超えているならば, 線型代数により, $\#T_2$ 個のベクトルは一次従属となり, これまでの議論と同様にして, この従属関係式から T_2 の部分集合 S が定まる. この S に対し, 関係式 (1.11) に加え, 全ての $\chi_q \in Q$ に対し,

$$\chi_q\left(\prod_{(a,b)\in S} (a+b\theta)\right) = 1$$

[3] q を factor base の bound よりも大きく選ぶなど, うまく選択する必要がある.

が成り立つこととなる．あとは $\prod_{(a,b)\in S}(a+b\theta)$ の平方根の計算を行えばよい[4]．より詳細な手順については付録 A.2.3 節参照のこと．

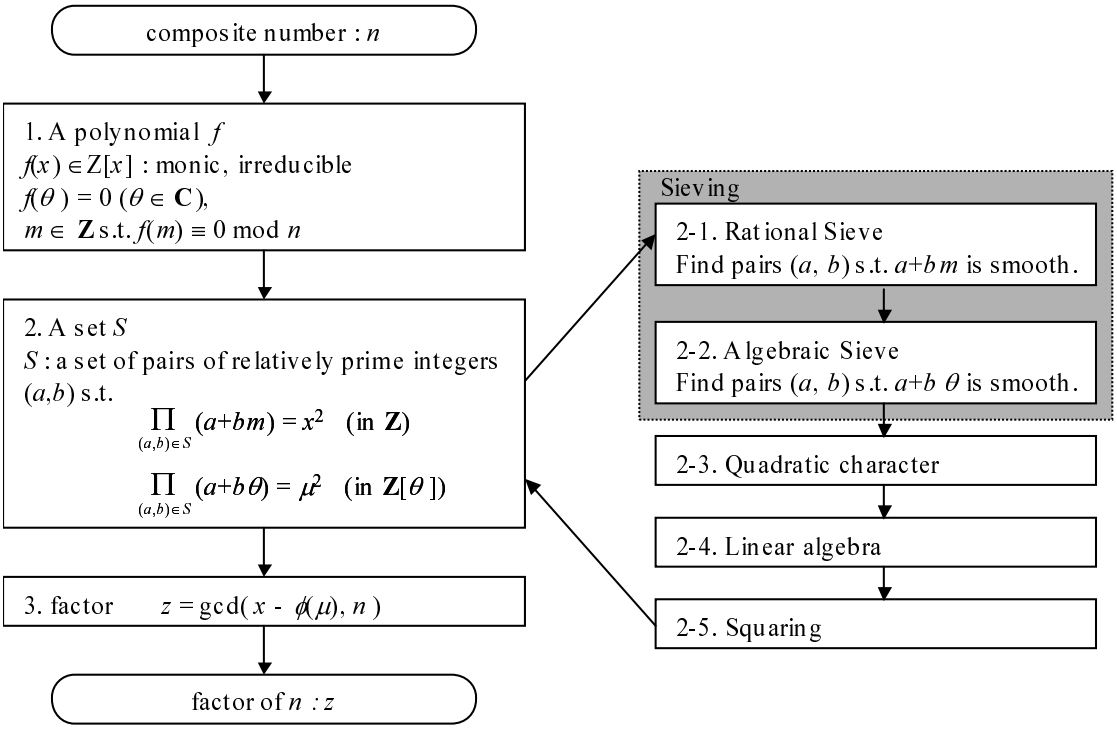


図 1.1: 数体篩法の流れ

1.1.4 篩の計算

前節で述べたように数体篩では rational と algebraic の 2 種の篩を実施する必要がある．

rational sieve ある範囲の a, b に対し, $a+bm$ の形の整数の smooth 性を調べるのが篩である．b を固定して, a を動かしたとき, factor base に含まれる素数 p に対して p で割りきれる $a+bm$ は等差数列として現れる．

そこでまず, 各 $a_{\min} \le a \le a_{\max}$ に対応させた $\max - \min$ 個の配列 $W[i]$ を用意する．初期値は 0 としておき, 各 p に対して, $a+bm$ が p で割れるならば対応する配列

[4]一般には $\mathbb{Z}[\theta] \neq \mathcal{O}_K$(代数的整数環) であり, この場合, 正確には若干の修正が必要になるが, ここでは詳細には立ち入らない．また, 線型代数関係の計算は, rational, algebraic を一度にまとめて行うこともある．

に $\lfloor \log p \rceil$ を加えていく. これは最初に割れる箇所さえ計算すれば, あとは p おきに加えていくことで実施できる.

factor base に含まれるすべての素数に対して上記手順を行った後, 値が, ある閾値を超えている配列に対応する $a + bm$ は, ある程度多くの素数で割り切れたということを意味し, smooth となる可能性の高い候補となる. さらに b の方も動かして多くの候補を集める.

これらの候補に対し, 実際に factor base の素数で素因数分解を実施し, smooth であるかどうかを確かめる[5].

algebraic sieve $N(a+b\theta) \equiv 0 \bmod p \Leftrightarrow a \equiv -br \bmod p$ for some $r \in R(p)$ であった. よって rational の場合と同様, b を固定, a を動かしたとき, (p, r) で割れる箇所は等差数列として現れる. よって手順もまったく rational の場合と同様である.

1.1.5 その他の計算

数体篩法では, 篩作業が終了したのち, 行列の計算を行い一次従属関係を作成しなければならない. 行列のサイズは factor base の個数などに依存して決まるが, 対象となる合成数のサイズが大きい場合, factor base も多く必要で, よって行列のサイズも非常に大きなものになる. 行列計算は基本的には Gauss 消去などで行うが, 扱う行列は疎 (要素が 0 のものが多い) であり, サイズを小さくして計算する方法などを用いることが可能である.

代数体の元の平方根を計算する方法としては, 多項式 ($f(x)$) が monic で次数が低い場合などは Hensel lift を用いて計算する方法, またそれらを改良した方法 ([30]) などがが有力である.

1.1.6 数体篩法の計算量

ここでは数体篩法の全ての処理に必要な計算量見積もりについて, 結果のみ述べることにする.

計算量を次の式を用いて表す.

$$L_x[a, b] = \exp(\ (b + o(1))(\log x)^a (\log \log x)^{1-a}\ ).$$

[5]完全に smooth とならない場合でもうまく組み合わせて平方数をつくる手法もある.

アルゴリズムへの入力を x としたとき, 計算量が $\log x$ の多項式であらわされるものを多項式時間アルゴリズムと呼ぶが, 上記式を用いると $a = 0$ ($L_x[0, b] = \exp((b + o(1))(\log\log x)) = \log x \exp(b + o(1))$) の場合として表現できる. $a = 1$ の場合は指数時間アルゴリズムということになる. よって計算量をこの式であらわした場合, a の値は 0 に近い方が計算量が"少ない", 素因数分解アルゴリズムに関して言えば, より優れた分解法であるということになる.

数体篩法 (平方剰余記号を用いた方法の場合) において, 目的の合成数を n としたときの計算量は, 高々

$$L_n[1/3, (64/9)^{1/3} + o(1)] = L_n[1/3, 1.92299\cdots + o(1)] \tag{1.12}$$

と見積もられている ([30]).

さらに Coppersmith による改良案 ([11]) によれば, 計算量の見積もりは

$$L_n[1/3, 1.901] \tag{1.13}$$

である. ちなみに, 2 次篩法 (MultiPlynomial 版 [50]) の計算量は $L_n[1/2, 1.020]$, 特殊数体篩法は $L_n[1/3, 1.526\ldots]$ となることも知られている (特殊数体篩法が適用できる合成数は極めて特殊なものに限られることに注意).

実際の合成数に対して, 各種パラメータをどの程度の大きさで取ればよいかについて, 様々な試算が行われている. 1024-bit RSA 型合成数 ($n = pq$, p, q は同じビット長の素数) は現在公開鍵暗号の鍵として広く用いられており, このサイズの合成数の素因数分解は多くの研究者の興味を引いている.

rational sieve における smoothness bound(factor base に含まれる素数の上限) を y_r, algebraic sieve に対する bound を y_a (1.1.3 節ではともに y と書いた) とするとき, [49](Crypto2003 版の改訂版) では $y_r \approx 3.5 \cdot 10^9$, $y_a \approx 2.6 \cdot 10^{10}$ を採用している.

また, [37] ではより理論的な考察のもと, 1024-bit に対しては $y_r \approx 1.2 \cdot 10^{10}$, $y_a \approx 5.5 \cdot 10^{10}$ が必要であることを導いている.

現在, 1024-bit 合成数に成功した例は皆無であり, これらのパラメータサイズで成功するか否かの実証は無論行われていない.

第2章　素因数分解の各種実装と実装方法提案

この章では，一般数体篩のこれまでの実装や，実装方法の提案についてまとめる．

2.1 汎用プロセッサによる一般数体篩の実装

これまで大きな合成数を分解することを目的として，一般数体篩法を汎用プロセッサに実装し，素因数分解を目的に動作させた事例がいくつか知られている．これらのうち，近年で主な結果を簡単に紹介する．なお，これらの情報は，[61, 57, 58] などによるものである．一覧にしたものを表 2.1 に示す．

これらのうちいくつかは，分解に用いた計算量やハードウェアについての報告があるが，報告されたものについては，汎用計算機，ワークステーションや PC を複数台用いることにより，計算を実現している．

2.2 汎用プロセッサが最良の実装法なのか

前述の通り，一般数体篩法では，複数の異なる処理ステップをすべて行なうことで，合成数を分解する．そして，当然ながら，異なる処理ステップでは，異なる種類の演算を行なう．

大きな素因数の分解を行なう場合は，これらすべてのステップにおいて，その実施が可能かどうかを検討することが必要である．特に，合成数の大きさが 1024 ビット程度の場合にも，どの処理が実際に計算できるものであって，どの処理が現実的に実装困難であるかを判定することは困難である．この理由として，各処理ステップが，単に (原始的な計算を単位とした) 計算量だけで，実施可能性が議論できるわけではない，という点が挙げられる．具体的には，必要となるメモリ領域の大きさや，メモリを共有するプロセッサ間の情報転送容量といった実装環境の要素も，一般数体篩の実行に大きく影響するからである．

以上の理由から，現状普及しているワークステーションや PC，ならびにそれらを接続するネットワークアーキテクチャが，必ずしも一般数体篩の演算に適している環境であるとは限らず，同じ金銭的コストにおいては，全く異なる計算機モデルがより効率的である可能性は否めない．

近年，特に一般数体篩法の動作において問題と考えられる，篩処理，そして線形行列演算については，その処理の性質から，低コストで膨大な並列処理を実現でき

る計算機モデルを利用した，専用ハードウェアの提案がいくつかある．

以下では，これらについて簡単に説明し，我々の具体的な評価対象となる装置までの落し込みを以下で説明してゆく．

2.3 TWINKLE

Shamir は，光電子工学を用いた篩装置を提案した [48]．これは TWINKLE (The Weizmann INstitute Key Locating Engine の略 [48], ピカピカ [キラキラ] 光る，という意味の動詞) 装置と呼ばれ，二次篩法や，各種数体篩法に必要な篩処理を行なうために用いることができる．

これは，1999 年当時，RSA 剰余として 512 ビットを利用することに対する，現実的な安全性検討のために提案されたものである．よって，この装置の提案におけるパラメータ各種も 512 ビット合成数の分解に合わせて記述されている．さらに，これらのパラメータは Lenstra と Shamir により翌年見直された [34]．この構造と特徴について説明する．

TWINKLE 装置は，直径約 6 インチ ($\approx$ 15.2 cm)，高さ約 10 インチ ($\approx$ 25.4 cm) の円筒系の装置であって，下底は，一面に敷き詰められた LED(発行ダイオード) からなっており，上底には，受光器が一つ下底を向いて設置されている (図 2.1)．各々の LED には簡単な内部状態つき論理回路が附属しており，これをまとめてセルと呼ぶ．各セルと受光器にはクロック信号が入力として与えられる．この装置の内部は空洞で，外部からの光が遮断されており，受光器は，LED の発する光のみを検出するものとする．

具体的な動作の基本的部分を説明すると以下のとおりとなる．TWINKLE デバイスは，各クロック毎に 1 つの篩点を検査する．各セルは論理回路の構成上，一つのプログレッションに相当すると考えることができ，対応する素数の大きさに応じて周期的に LED を発光させる．この発光量は対応するプログレッションの素数のビット数, $\log p_i$ に比例した光量とする．この動作が，ある篩点 (クロック時刻) において，担当するプログレッション (LED) がスコアを計上 (発光) することに対応する．上底の受光器の感知は，これら LED の合計光量であることから，上底の受光器のあるクロック時刻における観測光量が，全プログレッションからのスコアの総量と考えることができる．

観測光量がある閾値を越えた場合には，そのクロック時刻を外部の計算機に報告し，これを篩の結果生き残った篩点としてあとの処理に渡す．

この基本的動作を用いて，Lenstra と Shamir は [34] でパラメータの見直しを行ない，5000 個の TWINKLE 装置と，8 万台の Intel®社製 Penium®[1]II 450 MHz 搭載の PC とにより，768 ビットの合成数を 9 カ月で分解可能と主張している．

しかし，光電子工学に基づく装置は，シリコンウェハのみからなる回路に比較すると開発やコストの観点からなるべく避けたい．次に示す提案方式では，光電子工学に基づかない高並列な回路の提案であり，実現性の検討として十分対象となる提案である．

2.4 Bernstein の行列演算回路とその改良

一般数体篩法を実装するにあたり，篩処理部とならんで大きな問題とされるのが線形行列を解くステップの処理である．Bernstein は，メッシュ型計算機モデルをこれに適用した [4]．ここでは，メッシュ型計算モデルにおける高速なソーティングのアルゴリズム，Schimmler ソート [47]，に着目しこれを使った行列処理の実装方法の提案である．

このアイデアは，[35] でさらに改良され，メッシュ型計算モデルにおいて，特に並列ルーティング処理を用いながら行列演算を行なう手法を提案した．これにより，並列 1024 ビットの合成数に対する素因数分解においては，必要な行列演算の実現は，十分現実的である，との見解まで出ている．

[35] での見積りは，いくつかの楽観的な仮定を置きながらも，数千ドルの装置を製作の上，1 日稼働させることで，1024 ビットの素因数分解に必要な行列演算を終了できると言われている．

これらの解説は CRYPTREC での，別途評価として，公開されているので，ここで詳細な説明は省く [65]．

[1]Pentium は，Intel Corporation のアメリカ合衆国及びその他の国における登録商標です．

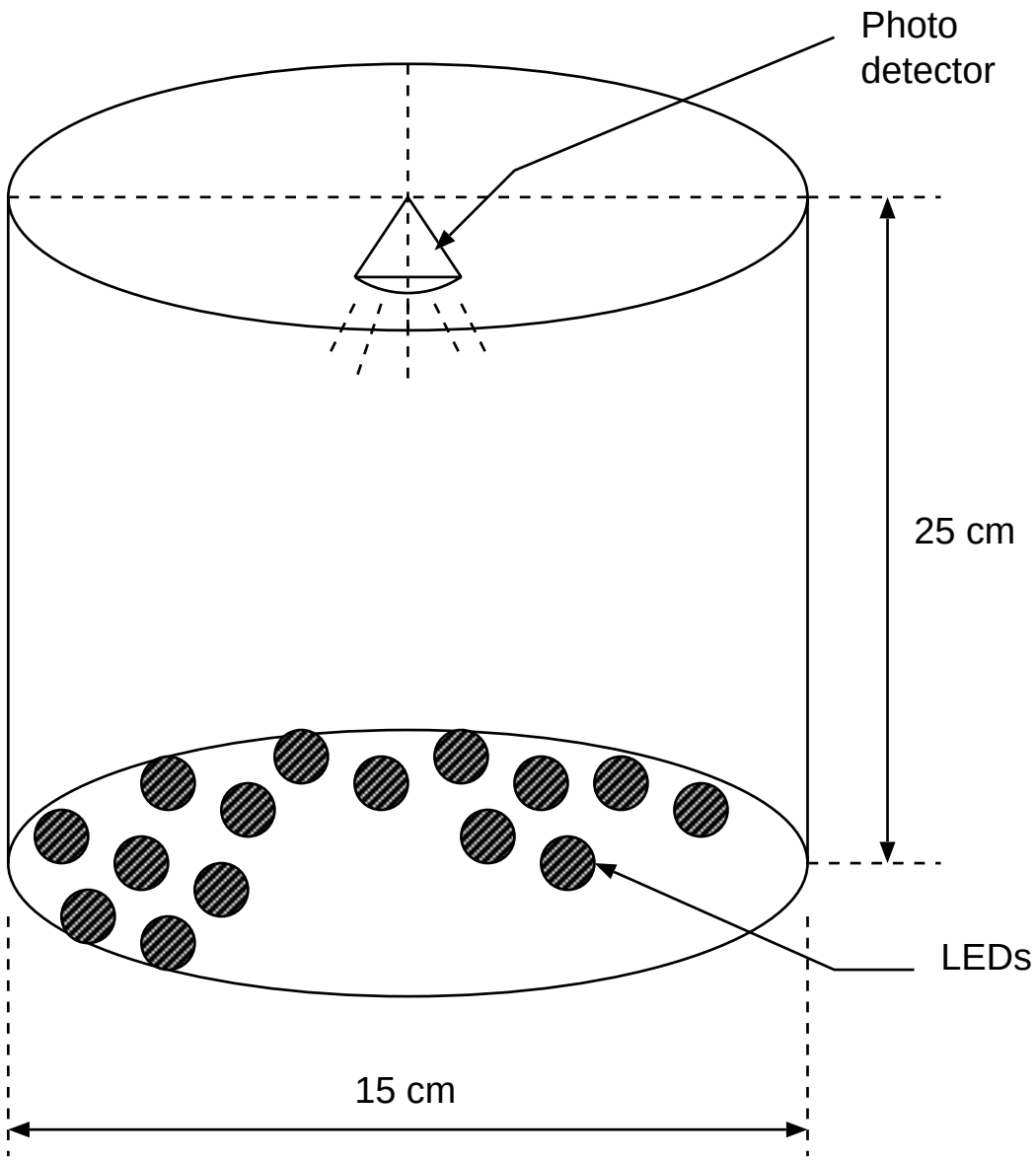


図 2.1: TWINKLE 装置の概要

2.5　篩処理のための専用回路提案

上記のメッシュ型計算モデルによる線形代数処理の結果は，最近になって，篩処理への応用が検討されるようになった．Geiselmann と Steinwandt は，Bernstein が提案 [4] したメッシュ型計算モデルにおける，ソーティングに着目したソーティングに基づく篩処理を実現する装置を提案した [16].

これは，プロセッサ数がプログレッションの数よりも多いことがポイントであるため，その著者らにより提案のあと，合成数サイズが 512 ビットを越えるような場合には，適切でないとの指摘があった [18].

以上の見解から，以下に挙げる TWIRL[49]，ならびにルーティングメッシュ計算モデルに基づく装置 [19] が，一般数体篩法を大きな合成数の分解を目的とした処理実現方法で，現在最も有効であると考えられる提案である．

2.5.1　ルーティングメッシュ計算モデル

Geiselmann と Steinwandt は，並列ルーティング処理可能な，メッシュ計算モデルを篩処理に適用する方法を提案した [19]．この計算モデルは，もともと Lenstra, Shamir, Tomlinson, Tromer らが [35] で線形代数部分に適用したものである．

彼の例示パラメータは，768 ビット合成数の素因数分解を主眼としたものを紹介しており，この中で下記に述べる TWIRL 方式に比較して，より実現性が高いことを主張している．

より具体的には，768 ビット合成数の分解のための装置の一部として，1 チップのサイズが 0.13 μm プロセスのウェハに，4.9 cm 角のチップとして実現可能であるとしている (TWIRL の場合，同じく 768 ビット合成数に対して約 6.7 cm 角のチップが必要)．30 cm シリコンウェハへ実装された装置として考えると TWIRL に比較して 6.3 倍遅いことになるが，チップ間通信の削減を実現できることから，現実性の観点から重要な成果であるという位置付けである．

この結果は，おそらく近い将来 1024 ビットの合成数に対する適用の可能性として拡張され，新しいパラメータの公開が考えうる．しかし，現時点では，パラメータは 768 ビット合成数に対するもののみであり，今回の評価対象として (我々がパラメータを設定した上でそれを我々が評価するというのは) 極めて小さい議論とな

る可能性があるので，本評価の対象とはしない．

2.5.2 TWIRL

TWIRL (The Weizmann Institute Relation Locator, くるくるまわす，振る，という意味の動詞) は，Shamir, Tromer が提案する，一般数体篩法における篩処理を行なうための新しい計算機構造の提案である [49].

その大枠の構造は，シストリックアレイと呼ばれる同じくメッシュ型の計算モデルを考える．しかし，従来とは異なり，セル間をつなぐバスは一方向のみの通信であって，二次元的なシフトレジスタ的な動作を行なうものである．

[49] の提案では，512 ビット，768 ビット，1024 ビット合成数の分解のためのパラメータが詳細に記述しており，その記述は主に 1024 ビット合成数むけの設計について行なっている．

2.5.3 本評価のアプローチ

本評価では，RSA 暗号をはじめとする，素因数分解問題の困難性に基づく暗号系の安全性評価を主たる目的として，具体的な 1024 ビット合成数の素因数分解の現実性について評価を行なう．

その具体的指針として，一般数体篩法，ならびにその改良法を分解アルゴリズムとして選択し，その中でも現状，最も困難であることが予想される篩処理を行なう計算モデルの実現性に焦点をあてる．そして，その有力な手法として，並列ルーティング可能なメッシュ計算モデル [19]，ならびに，TWIRL 装置 [49] を題材として考えるが，1024 ビット合成数という評価対象としてのデータの揃っている TWIRL について詳細な実現性評価を与えることを目的とする．

TWIRL に対する実現性評価の観点からは，第一に，回路仕様にはなるべく忠実とし，可能な限り実現可能性に疑問となる部分の洗い出しを行なう．アプローチとして，は TWIRL の提案者，そして提案論文の議論される分野を理解した上で，暗号学的，あるいは数論的な観点からの考察は限定的とし，本評価の中心とはしない．

本評価はむしろ，TWIRL がこれまであまり評価されなかった，仕様から実際のハードウェア生産への過程における技術専門家の立場からの技術的考察を主眼とした．

これらの考察の一部は，TWIRL 独自のものである部分もある一方，残りの考察については，解読という極めて専門性が高く，規模として巨大なハードウェアの設計という観点から，他の提案モデル，たとえば，並列ルーティング可能なメッシュ計算モデルによる篩処理の実現 [19] や，線形行列処理 [35] についても言及できるものがあると考える．

表 2.1: 大きな合成数の分解を目的にした一般数体篩法の公開実装結果

合成数名称	10 進桁数	ビット長	分解完了日付	関連
RSA-130	130	430?	1996 年 4 月	[27]
RSA-140	140	465	1999 年 2 月	[45]
RSA-155	155	512	1999 年 8 月	[9]
C158	158	524	2002 年 1 月	[12]
RSA-160	160	530	2003 年 3 月	[13]
RSA-576	174	576	2003 年 12 月	[14]
C164	164	545	2003 年 12 月	[59]

注意: RSA-130 については，ビット数表記の情報がみあたらなかったため，10 進数表示の上数桁を用いて，合成数を $c \times 10^e, 1 \leq c < 10$，(ただし e は整数) と表記する (c, e) から，$\lfloor \log_2(c) + e \log_2(10) \rfloor + 1$ よりビット数を換算したもの.

第3章　TWIRLの記述

この章では，TWIRL 装置を記述した技術文書である [49] を紹介する．この章の位置付けは，TWIRL として，Shamir, Tromer が記載したことと，我々が理解の中で補間し，補足した部分とを区別することにある．

3.1 要約

RSA 暗号システムの安全性は大きな整数を素因数分解することが困難であることに依拠している．現在のもっとも優れた素因数分解アルゴリズムは数体篩法 (Number Field Sieve, NFS) であるが，その中では，篩ステップが最も処理困難な部分である．1999 年，数百台のワークステーションからなる大規模分散処理により，数ヶ月を要して，512 ビットの RSA 鍵の素因数分解を行なうことに成功したが，その時点においては，1024 ビット鍵はその後更に 15〜20 年安全であると信じられていた．本論文においては，数体篩法の篩ステップの新たなハードウェア実装法 (スタンダード 0.13μm 半導体プロセス，1GHz 動作による) について述べる．同実装法は，これまでに発表された最もよい設計法 (光電子工学による TWINKLE やメッシュ構造型篩) よりも 3〜4 桁コスト効率が良い．我々は，重要な構成要素全てについての詳細検討を行ない (実際の実装は未実施であるが)，512 ビット RSA 鍵に対しての，数体篩法の篩ステップが$10k の装置により 10 分以下で実現可能であると確信している．1024 ビット鍵については，数体篩法のパラメータを検討したところ (可能な限りの実験データが支持するところでは)，篩ステップは$10M の装置により 1 年以下で実現可能であるとの見通しを得ている．近年の，数体篩法の行列演算ステップの処理コストに関する研究結果と合わせて考えると，1024 ビット鍵長の安全性への懸念が高まった．

3.2 はじめに

整数の素因数分解の困難さは，暗号理論の主たる仮定であり，広く普及している複数の暗号システムの基礎をなしている．既知の最良の素因数分解アルゴリズムは，数体篩法 [30] であり，同アルゴリズムに基づき，512 ビット，530 ビットの RSA 合成数の素因数分解が成功している [9, 2]．しかしながら，PC ベースの数体篩法の実装は，実用上，対象の合成数サイズを大きくすることが困難である上，1024 ビット合

成数の素因数分解に要するコストが膨大である．近年，数体篩法のうちの演算コストが高い部分に対して専用ハードウェアを適用する手法が注目を集めている．メッシュ構造型回路によって，1024 ビット合成数に対する行列演算ステップが極めて現実的になった一方で [4, 35]，篩ステップに関する状況は芳しくない．TWINKLE[34, 48] やメッシュ構造型回路 [16] など，幾つかの篩用デバイスが提案されているが，実用上，1024 ビット合成数を素因数分解できるものがないのは明白である．

行列演算ステップを対象とした，Bernstein のメッシュ構造型回路 [4] から,「単にそこにあるだけで遊んでいる」メモリセルが存在するのは非効率的であるという知見が得られている．つまり，入力値を単に保持するメモリが高価であるならば，適切な並列化により，メモリを効果的に利用すべきである．本稿では，上記の考え方と，スペースを費やして時間を得るという TWINKLE に類似した手法を組み合わせた新しいデバイスを提案する．TWINKLE が篩ポイントを 1 クロックに一点づつ，シリアルに処理するのに対し，提案する新デバイスは数千の篩ポイントを 1 クロックに複数，並列に処理する．更に，新デバイスは，TWINKLE よりも小さく，又製造が容易である．512 ビット合成数の場合で言えば，TWINKLE デバイス 1 個が GaAs ウェハ 1 枚を必要とするのに対し，新デバイスでは直径 30cm のシリコンウェハ上に 79 個の独立動作する篩デバイスを実装可能である．本稿の手法は文献 [16] に関係しているが，本手法の方が対象とする合成数のサイズを拡張することがより簡単であるうえに，幾つかの問題点を回避している．

本手法においては，衝突，又は長い伝播遅延やストレージの効率性の維持管理なしで，一つの (若しくは少数の) 入力値のコピーを用いて，全体処理のための多くの下位問題を並列的に処理する部分が，最も困難な課題である．我々はこの課題に対し，複数種類の経路選択回路を採用し，利用可能な技術の有するトレードオフを有利に活用する混成デザインを用いて対処する．コスト見積もりの結果からは，1024 ビット合成数に対する篩ステップは驚くほどに実現可能性が高いだろうと考えられる．

第 3.3 節で，篩問題と TWINKLE デバイスについて概説する．第 3.4 節では新デバイス (以下，TWIRL[1] と称する) について詳述し，第 3.5 節において仮のコスト見積もりを示す．第 3.7 節では付加的なデザインの詳細と改良について述べ，第 3.8 節ではコスト見積もりの前提とした仮定，第 3.9 節では従来の研究との関係を述べる．

[1] TWIRL とは The Weizmann Institute Relation Locator の略称である．

3.3 研究の背景

3.3.1 数体篩における篩処理

ここで提案する装置実装は，数体篩における関係式 (relation) 収集ステップにおける篩処理である．これは数体篩においてもっとも計算量的に困難とされる部分である [35]．ここでは，篩問題をまず振り返る．しかしここでは詳細についてはかなりの部分を省略し，問題の置き換えなどを適切に行なったものを紹介する[2]．数体篩の背景については [30] を参照のこと．

篩問題の入力は $R \in Z$(篩線幅), $T > 0$(閾値), (p_i, r_i) 組の集合である．ここで，p_i はファクタベース境界 B よりも小さな素数である．1 個の素数あたり，平均 1 個のそのような組が存在する．各々の組 (p_i, r_i) は等差数列 $P_i = \{a : a \equiv r_i (\mathrm{mod} p_i)\}$ に対応する．我々は，次のような性質をもつ篩点 (sieve location) $a \in \{0, .ldots, R-1\}$ を集めることを目的とする．大きな p_i に対する多くのプログレッション P_i のメンバーであって[3]，ある固定の h に対して

$$g(a) > T \quad \text{where} \quad g(a) = \sum_{i:a \in P_i} \log_h p_i$$

となるものである．この閾値検査に多少のエラーがあっても許容するものとし，特に対数値はもっとも近い整数に丸めるとする．

数体篩の関係式収集ステップでは，二種類の篩を考える．rational と algebraic である．これら両方が上記の考え方で扱うことができる．しかし，ファクタベース境界, B_R と B_A, 閾値 T, 対数の底 h には異なるものを用いる．ここでは，H 本の篩線に対して，両方の篩 (rational, algebraic) を処理する必要がある．すなわち，合計 $2H$ 本の篩問題が存在することになる．各々の篩線の中で両方の篩における閾値を通過したもの (a) は candidate(滑らか候補) として扱われる．これらの candidate はさらに追加検査を行なう必要があるが，そこで，$\{i : a \in P_i\}$ なる i を a に対して記憶しておくことが効率の観点からは望ましい．これらのテストでもっとも負荷のかかる部分は cofactor factorization と呼ばれる処理であり，これは，ちょっと大きい

[2] ここでの記述は線篩，格子篩の両方を考えることができる．しかし，格子篩については我々は別のアプローチを取ることにする (第 3.7.8 節参照).

[3] 訳注 本報告では，プログレッション (progression) を，従来の数列という意味からやや脱して，特に TWIRL に特化し，篩ポイントへスコアを加算する (詳細は後述) 各々の素数に対応するものと考える．

(medium-sized) 整数に対する素因数分解からなる処理である[4]．このテストをパスしたものが relation(関係式) と呼ばれ，関係式収集ステップの出力はこれら関係式のリストと，これに対応する集合 $\{i : a \in P_i\}$ からなる．我々の目的はある程度たくさんの関係式を見つけ出すことであり，パラメータはそのように選ばれるものとする．

3.3.2 TWINKLE

TWIRL は従来から考えられてきた数体篩実装に対して，TWINKLE[34, 48] 的な時間-空間のトレードオフを利用している．ここでは，TWINKLE について簡単に復習する (しかし詳細についてはかなり省略する)．TWINKLE 装置は多数の独立したセルを含むウェハにより構成される．各々のセルは単一のプログレッション P_i に対応している．初期化の後，R クロック動作し，これが篩範囲 $\{0 \leq a < R\}$ に相当する．クロック時刻 a において，P_i に対応するセルは，$a \in P_i$ の場合に限って，値 $\log p_i$ を放出する．各クロックで放出された値は合計され，もし，これが閾値 T を越えた場合に，その整数 a が報告される．このことは各セルにもフィードバックされ，P_i に貢献した値 i が報告される．全体加算処理は，アナログな光学メカニズムにより実現されるが，それ以外はデジタルな処理として実現する．電子光学的処理を行なうため，TWINKLE では，ガリウム-ヒソウェハを用いる．一般に GaAs ウェハは，現状多く使われるシリコンウェハに比べて，面積がとれず，高価であり，また製造過程がより困難である，といった欠点がある．

3.4 新しい装置提案

3.4.1 アプローチ

次に TWIRL 装置について説明する．この節での記述は有理数篩についてであるが，これを小変更することで代数篩 (第 3.7.6 節) となる．というのも，代数篩では，有理数篩をパスした篩点 (a) のみを考慮するからである．

ここでは装置の明確化を目的に，1024 ビット合成数のために妥当であろうパラ

[4] ここでは，数体篩の「2+2 large prime」拡張法を用いる [30, 28].

メータ選択の具体的数値例を記述している[5]．その選定については第 3.5 節, 第 3.8.2 節で説明する．これは，最適な値ではなく，大まかな評価である．$\langle\!\langle x \rangle\!\rangle_{\mathrm{R}}$ は有理数篩，$\langle\!\langle x \rangle\!\rangle_{\mathrm{A}}$ は代数篩，$\langle\!\langle x \rangle\!\rangle$ は有理数篩, 代数篩両方に適用するパラメータ即値を示す.

まず，$H\langle\!\langle \approx 2.7 \cdot 10^8 \rangle\!\rangle$ 組の篩問題を解きたいとする．各々の篩問題にについては，篩線幅 $R\langle\!\langle = 1.1 \cdot 10^{15} \rangle\!\rangle$，滑らか境界 $B\langle\!\langle = 3.5 \cdot 10^9 \rangle\!\rangle_{\mathrm{R}}\langle\!\langle = 2.6 \cdot 10^{10} \rangle\!\rangle_{\mathrm{A}}$ である．まず最初の装置として，TWINKLE のようなクロックあたり 1 篩ポイントを扱う装置を考えてみる．そしで，これが電子加算回路からなるパイプライン化されたシストリック連鎖だとする[6]．そのような装置はバス幅 $\log_2 T$ の長い一方向バスからなり，このバスは直列に条件付加算回路を数百万など多数連結しているものとなる．書く条件付加算回路はひとつの 1 つのプログレッション P_i に対応しており，対応するタイマにより指示されると，値[7] $\lfloor \log p_i \rceil$ をバスに加算する．時刻 t では，z 番目の加算は篩ポイント $t - z$ を担当する．最初にパイプライン最後尾列に出力される値は $g(0)$，以降 $g(1), \ldots, g(R)$ と毎クロック出力される．(図 3.1(a) 参照).

さらに，これを $s\langle\!\langle = 4096 \rangle\!\rangle_{\mathrm{R}}\langle\!\langle = 32768 \rangle\!\rangle_{\mathrm{A}}$ 並列にすることで処理時間を $1/s$ に短縮することを考える．全体で $\{0, \ldots, R-1\}$ ある篩ポイントを長さ s の固まり (chunk) に分割し，1 サイクルで s 個の篩ポイントを一度に扱う．このとき，バスが s 倍となり，全体のバス幅は $s \cdot \log_2 T$．時刻 t，z ステージでは，s 個の篩ポイント $(t-z)s+i, i = 0, \ldots s-1$ を処理 (該当するプログレッションに対する加算とその判定を) する．最初にパイプライン最後尾列に出力される値は $\{g(0), \ldots, g(s-1)\}$ であり，同時に出力される．以降，出力は重なることなく篩線を分割しながら出力される．図 3.1(b) 参照.

この設計では主に二つの問題点が発生する．s 並列化したので，ハードウェアは s 倍の処理をこなすことになる．さらに，並列化したすべての line において，p_i を扱うことになる．TWIRL の設計では，単にこれらを line 分複写して，s 個の加算すべてに適用するような素朴なアプローチは適用しない．TWINKLE 的セルを計算ステーション (station) と呼ばれる別のユニットに置き換える．各計算ステーションは複数のプログレッションを担当する．計算ステーションのインターフェースはバス入力，バス出力，クロック，入力を load する回路群からなる．計算ステーション

[5] ここでのパラメータの選択は，本稿最初の原稿で用いられたものとはかなり変更があることに注意する.

[6] このような改良は [34] で検討されたが，当時の設計には不向きであると評価された.

[7] $\lfloor \log p_i \rceil$ はある固定の h について，値 $\log_h p_i$ を最も近い整数に丸めた値を表す.

はパイプライン的に直列に接続され，バスの末端 (最後の計算ステーションの出力) には，閾値チェックユニットがあり，これが装置出力を決定する．

ここで重要なのは，プログレッションが作用するのは正確に p_i 周期であって，かつ p_i が大小多様なサイズで混在することである．また，大小それぞれのプログレッションについて実装のトレードオフが存在する．そこで，TWIRL では，プログレッションを, p_i の大きさに応じて三つのカテゴリに分割する．そして，計算ステーションも三つのカテゴリそれぞれに応じて設計する．p_i が大きい順に，Largish, Smallish, Tiny と呼ぶ[8].

このような特殊な (heterogeneous な) アプローチにより，高度な並列処理を伴うものの，1024 ビット合成数のための素因数分解についても，妥当な装置規模で抑えることができる．標準的な VLSI 技術により，TWIRL の有理数篩を単一の 30 cm シリコンウェハ上に $\langle\!\langle 4 \rangle\!\rangle_{\mathrm{R}}$ 台実装することができる．このための生産コストは量産したと過程して\$5000 程度と見積もられる．ここで，局所欠陥については，第 3.7.9 節にて議論する．また，代数篩についても高並列処理を駆使し，ウェハに $\langle\!\langle 1 \rangle\!\rangle_{\mathrm{A}}$ 台搭載可能である．

以下の節で，各プログレッションに対するハードウェアを説明する．あとで示すコスト評価 (初期評価) については，今回の設計で最も重要である部分すべてに対して行なった詳細な解析の結果に基づくものである．しかし，紙面の都合上，多くの詳細については削除する．いくつかの補足事項については，第 3.7 節にて議論する．

3.4.2 Largish primes (Largish プログレッション)

全体構造 p_i が s よりかなり大きいようなプログレッションについては，加算処理が滅多に起きない．このような，Largishprime，Largish プログレッション ($p_i > \{\langle\!\langle 5.2 \cdot 10^5 \rangle\!\rangle_{\mathrm{R}}, \langle\!\langle 4.2 \cdot 10^6 \rangle\!\rangle_{\mathrm{A}}\}$) については，多少規模が大きくなってでもよいので，多数のプログレッションを扱うことができるような計算ステーションを使うことが有利である．ただし，個々のプログレッションに必要な記憶領域は最小限とする．そこで，考えられたのが，図 3.2 に示す構造である．各々のプログレッションの具体的なものとしては，progression triplet, プログレトリプルデータとして扱われる．こ

[8] これらは，large prime, small prime などと呼ばれる数体篩アルゴリズムにおけるより高いレイヤでの議論の用語との混乱を避けるための呼び方である．ここで扱う素数は，そういう意味ではすべて small である (large でもなければ，特殊 q の範囲でもない).

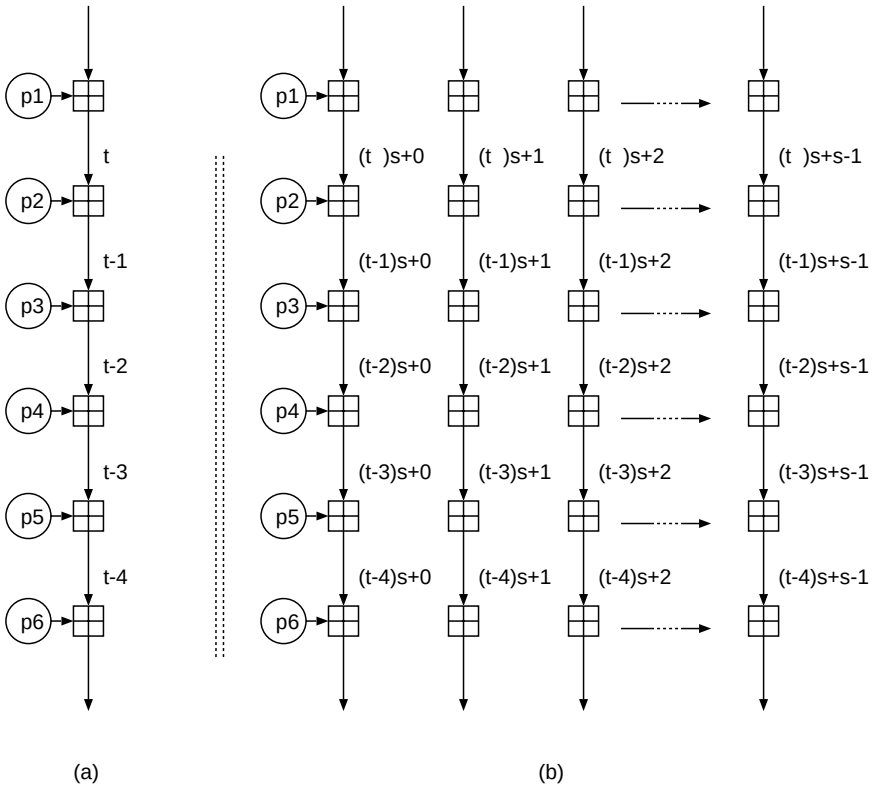


図 3.1: 篩ポイントの流れ: 装置が (a) 加算連鎖の場合, (b)TWIRL の場合

れが小規模の DRAM メモリに保存されている．メモリ中のプログレトリプルデータのリストはある特殊なプロセッサにより，周期的に検査，更新される．この特殊プロセッサは，近い将来のいつ，どのプログレトリプルデータが呼び出されるかを計算し，トリプル発行データを生成する．トリプル発行データはバッファに渡される．バッファは他のプロセッサも含め複数のプロセッサから渡された結果をマージ (結合) し，タイミングを正確に見定めた上で (次の Delivery line, デリバリラインに渡す) Delivery pair, デリバリペアを生成する．デリバリラインはパイプライン化されており，Delivery cell, デリバリセルの連鎖からなる．デリバリセルはデリバリペアを適切なバス線へ配達し，$\lfloor \log p_i \rceil$ を加算するようにする．

メモリ内のプログレトリプルデータのスキャン　プログレッションは多数，$\langle\!\langle 8490 \rangle\!\rangle_{\mathrm{R}}$, $\langle\!\langle 59400 \rangle\!\rangle_{\mathrm{A}}$ 個の DRAM バンクに分割される．各々の bank には d 個のプログレッションが含まれている ($\langle\!\langle 32 \rangle\!\rangle \leq d < \{\langle\!\langle 2.2 \cdot 10^5 \rangle\!\rangle_{\mathrm{R}}, \langle\!\langle 2.0 \cdot 10^5 \rangle\!\rangle_{\mathrm{A}}\}$)．プログレッションはデータの三つ組, (p_i, l_i, τ_i) で表現される．ここで，l_i, τ_i のペアにより，次にどの篩ポイント $a_i \in P_i$ に加算するのかを示す (DRAM 内で明示的に a_i の値が保存されているわけではない)．時刻 $\tau_i = \lfloor a_i/s \rfloor$ は，次にバスに加算されるべきタイミングを示す．$l_i = a_i \bmod s$ は加算の対象となる，篩ポイントがどこの line に相当するか, を示す．プロセッサは以下の処理をパイプライン処理の要領で行なう[9]．

1. (読込み) メモリからある指定されたメモリ番地の (p_i, l_i, τ_i) を読込み，読んだあとは，メモリから消去する．

2. (送信) トリプル発行データ $(\lfloor p_i \rceil, l_i, \tau_i)$ を，プロセッサに接続されているバッファへ送る．

3. (更新)$l'_i = (l_i + p_i) \bmod s$, $\tau'_i = \tau_i + \lfloor p_i/s \rfloor + w$, ここで w は繰り上がりで, $l'_i < l$ なら $w = 1$, そうでないなら $w = 0$ である．

4. メモリに (p_i, l'_i, τ'_i) を書き込む．ただし，そのアドレス番地は τ'_i に依存したものである．

理想的には，トリプル発行データ $(\lfloor \log p_i \rceil, l_i, \tau_i)$ が時刻 τ_i よりもわずか前に生成されたい (早すぎるとバッファが溢れるし，遅いとトリプル発行データが処理される

[9]集合 $\{i : a \in P_i\}$ を報告するための追加の論理回路については，第 3.7.7 で扱う．

前に時刻 τ_i が過ぎてしまう)．よってプロセッサには，imminent な (**差し迫った**) プログレトリプルデータをメモリから随時読み出すことが求められる．ただし，巨大な d(プロセッサが受け持つプログレッションの数) については，単純なアプローチでは処理がうまくいかない (例えば，工夫もせず各々のプログレッションを適当なアドレスに保存したなら，差し迫った τ をスキャンしなければいけないが，とてもメモリ全域スキャンができる時間はない)．よってプログレトリプルデータを，何か優先度つき記憶装置のようなものを考え，τ でソートされているようなものを維持することが理想的である．しかし，篩問題の特徴を使うことにより，上記の理想に対して，ほどよい解決策を以下のように考えることができる．

メモリ内のプログレトリプルデータの保存方法　ここでは，プロセッサが (プログレトリプルデータが保存されている) メモリに対して直列で，読み込み幅が一定の巡回するメモリスキャンで読めるようなメカニズムを考える (ただし，データ (トリプレット) の読み込みは平均 2 クロックで 1 度程度)．もし，読み込みが空要素ならば，そのループでは後の処理を行なわない．もし要素 (プログレトリプルデータ) があったなら，上記にあるとおりプログレトリプルデータを読出し/処理/更新する．ただし，メモリ書き出しの場所が読み出しの場所とは違うかもしれない τ_i' に応じた場所, 具体的には τ_i' が読み出されるであろうアドレスのちょっと前とする．このようにしておけば，初期化直後の動作を除いて，動作が安定すれば定期的にアクセスさえしていれば，それが常に**差し迫った**プログレトリプルデータとなっているはずである．そこで，格納したい番地範囲に空きメモリ空間が (高い確率で) 存在するように，予めメモリ空間は余計にとっておく (今回は $\langle\!\langle 2 \rangle\!\rangle$ 倍)．実際に格納要求がきたら，τ_i' が読み出されるであろう番地から遡った最初の空き番地に保存する (当然巡回アドレス，すなわちメモリ空間の最初まで遡ってしまったら次はメモリ空間の末尾に行く)．

もし，$\langle\!\langle 64 \rangle\!\rangle$ 番地遡っても空番地がみつからない場合，そのプログレトリプルデータは (適切な場所への書き込みを諦めて) 任意の場所，もしくは特別なオーバーフロー領域に書き込んで，いくらか事後になって適切な場所へ移動する．このようなことが起きてしまうと，2〜3 の加算処理ができなくなってしまうかもしれない (実装にもよるが)．しかしこれが起こるのは非常に稀であり[10]，このように多少の加算

[10]例えば，$\langle\!\langle 20000s \rangle\!\rangle_{\mathrm{R}}$ 程度の p_i でシミュレートした場合，$\langle\!\langle 5 \cdot 10^{-6} \rangle\!\rangle_{\mathrm{R}}$ 程度の例外を除いてほ

ミスは許容可能である．

メモリ内部の autonomous 回路は，プロセッサから入力されたプログレトリプルデータを理想番地から手前にある最初の空番地に格納する．これを効率的に実装するために，2段目メモリ階層構造を考える．これは以下の篩装置の性質から発案したものである．例えば，あるプロセッサが d 個の連続した素数, $\{p_{min}, \ldots, p_{max}\}$ を担当しているとする．そこでプロセッサが扱うメモリサイズを p_{max}/s 要素分 (一つの要素で (p_i, l_i, τ_i) を保存する) とする．$p_i = p_{max}$ が現在読みだし番地の直前に書き込まれているとし，これよりも小さい素数 p_i はそれよりも前の番地に保存されている (もちろん，メモリ領域は巡回イメージ)．素数密度の定理から，$p_{max} - p_{min} \approx d \cdot \ln(p_{max})$．よって，プログレトリプルデータは多くとも $d \cdot \ln(p_{max})/s$ 番地程度前，あるいはデータの集中などにより多少前にずれる程度の番地にある．$\ln(p_{max}) \leq \ln(B)$ は s よりもかなり小さい値なので，メモリアクセスは, 2サイクルで1メモリ番地だけ定速度でスライドする小さな窓の中で必ず起こる．そこで，$\langle\!\langle 8490 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 59400 \rangle\!\rangle_{\mathrm{A}}$ 個の DRAM バンクを，いろいろなサイズで閉じた輪の集合に見立てることができる．この輪には，active な窓が輪に沿ってくるくると定速度で回転している (twirling) ようにみえる．

各々のスライド窓は高速アクセス可能な SRAM ベースのキャッシュにより処理される．そして時折，古いキャッシュブロックの書き込みや次のブロックをキャッシュに読み込むなどすることで，窓をシフトする．SRAM と DRAM バンクの間に適切なインターフェース (つまり，full row に対する read-write) を持つことにより，DRAM の欠点である大きなアクセス遅延を事実上無視することができ，高いメモリアクセス速度を達成できる．また，これにより，単純でかつより小さい DRAM を実現できる[11]．注意するのは，キャッシュミスは起こり得ないことである．プロセッサがメモリに対して行なうことができる操作は

1. 次のメモリ番地の内容を読む

ぼすべてが，理想番地とその手前で最初の空き番地との番地距離が $\langle\!\langle 64 \rangle\!\rangle_{\mathrm{R}}$ 個以内である．乱数 $x \in \{1, \ldots, x\}$ が k 個の因子を持つを持つ確率は約 $(\log\log x)^{k-1}/(k-1)!\log x$．篩で扱う値の大きさは，$x \approx \langle\!\langle 10^{64} \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 10^{101} \rangle\!\rangle_{\mathrm{A}}$ 程度であって，これらは確率 $p \approx \langle\!\langle 6.8 \cdot 10^{-5} \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 4.4 \cdot 10^{-9} \rangle\!\rangle_{\mathrm{A}}$ で *good*, すなわち仮滑らか (semi-smooth) だと判定されるので，多くとも *good* な a の $10^{-15}/p$ 程度しか，35個の因数を持たない．よって *good* な a を見逃す確率は無視できるほど小さい．

[11] 多くのペリフェラル DRAM 回路 (例えば，リフレッシュ回路や，コラムデコーダ) を実装する必要がなくなる．row デコーダは小さな状態回路で置き換えることができる．よって DRAM バンクは標準的な設計のものよりも小さくなる．この Largish レンジにおけるより，若干小さい素数を扱う計算ステーションについては，キャッシュサイズを d 程度まで増やし，DRAM 自体を小さくできる．

2. 指定した番地に，データを書き込む (ただし，番地は，指定番地から遡った一番最初の空番地)

のみである．後者のロジックはキャッシュに実装され，補助的なフラグである「前トリプレット空きフラグ」といくつかのローカルパイプライン回路を使う．

バッファ　バッファ装置はトリプル発行データをいくつかのプロセッサから並列に受信し，デリバリペアを生成したら，それをいくつかのデリバリラインに送信する．バッファの作業としては，トリプル発行データをデリバリペアに変換し，必要ならトリプル発行データをマージし，タイミングを厳密にし，そしてそれらをデリバリラインに配布する．受けとった $(\lfloor \log p_i \rceil, l, \tau)$ で表されるトリプル発行データに対して，デリバリペア $(\lfloor \log p_i \rceil, l)$ が，ある特定の (l の値に依存して) デリバリラインに対して，時刻 τ において，送信される．

バッファ装置は以下のように実現される．まず，到着するトリプル発行データは並列化された，優先度つき待ち行列 (キュー) に τ で順序つけされて格納される．これは，小さなメッシュ計算機であって，列 (row) は常にバブルソートが実行され，行 (column) はランダムで局所的なシャッフルが行なわれている．最後の 2〜3 列は時刻マッチングとして tau と比較を行なう．そして，マッチしたものは l(ライン番号) でソートされ，必要であればマージされ，パイプライン処理へ流される．時おり，データの過密度合いなどにより，データが遅れて到着する場合が考えられるがこの場合には，これらデータを無視し，破棄する．しかし，入力は原則ランダムであるので，適切なパラメータを選択してさえいればこのような状況は非常に稀となる．

バッファサイズはトリプル発行データがその発射時間 τ までの間，どれだけのタイムステップだけバッファで待たされるかと，プロセッサががトリプル発行データを生成する頻度 (約 $\langle\!\langle 4 \rangle\!\rangle$ サイクルに 1 発) に依存する．しかし，前者の発射までの時間は，先に説明したプロセッサの動作のおかげでかなり小さい．

デリバリライン　デリバリラインは $(\lfloor \log p_i \rceil, l)$ の形をしたデリバリペアを受けとったら，受けとった直後，バス線 l に，受けとった時刻から $\lfloor l/k \rfloor$ クロック後に，加算する．これは，セルの 1 次元配列として，バスを横切って，実装される．各々のセルでは，1 個のデリバリペアを保持することができる (もちろん，デリバリペアが到着しなければ何も情報を保持しない)．j 番目のセルは到着したデリバリペアの l(ラ

イン番号) と比較し，もし同じなら $\lfloor \log p_i \rceil$ をバスラインに加算し，その組を破棄する．そうでないなら (シフトレジスタのように) 次のセルへ渡す．

全体で $\langle\!\langle 2100 \rangle\!\rangle_{\mathrm{R}}$[12], $\langle\!\langle 14900 \rangle\!\rangle_{\mathrm{A}}$ 個のデリバリラインが Largish プログレッションに実装される．そして，これが装置のかなりの分量を占める．第 3.7.1 節にインターリーブした繰り上がり抑止加算 (carry-save adder) を使ったコスト削減方法を記載する．第 3.7.6 節には代数篩からこれらの数を見積もっている．

注意 3.4.1. プロセッサ, *DRAM,* バッファの説明で，τ をクロックサイクルを示す任意整数として取り扱った．実際にはこれらの値をある剰余，例えば $\langle\!\langle 2048 \rangle\!\rangle$ で保持することも可能である．これの数値は，プログレトリプルデータがメモリから読みこまれ，バッファから放出されるまでのクロック数よりも大きくなければならない．よってプログレッションひとつのプログレトリプルデータあたり，*DRAM* において $\log_2 p_i + \langle\!\langle \log_2 2048 \rangle\!\rangle$ ビット使う．これにさらに $\log_2 p_i$ ビットが再初期化 *(*詳細は第 *3.7.4* 節に記載*)* に必要である．

最終的に必要とする *Largish* プログレッションあたりの回路規模は，$\Theta(s^2(\log s)/p_i + \log s + \log p_i)$ となる[13]．s を固定とすると，$\Theta(1/p_i + \log p_i)$ となり，大きな合成数の素因数分解を行なう場合には，かなり大多数のプログレッション $\langle\!\langle 99.97\% \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 99.98\% \rangle\!\rangle_{\mathrm{A}}$ がこのようにして処理される．

3.4.3 Smallish primes (Smallish プログレッション)

p_i が s 程度のプログレッション，すなわち $\langle\!\langle 256 \rangle\!\rangle < p_i < \{\langle\!\langle 5.2 \cdot 10^5 \rangle\!\rangle_{\mathrm{R}}, \langle\!\langle 4.2 \cdot 10^6 \rangle\!\rangle_{\mathrm{A}}\}$ については，プロセッサがあまりたくさんのプログレッションを処理できない．プロセッサは $\langle\!\langle 2 \rangle\!\rangle$ クロックあたり，せいぜい 1 回の発行しかできないからである．よって，プロセッサ，メモリ，制御回路，バッファにかかるコストがとても大きくなる．さらに，そのようなプログレッションは頻繁に起こってしまうため，発行に関するバス線への通信が巨大なスループットが必要になる (これは，プログレッションの状態が単一の物理装置で管理されている以上避けられない問題である)．そこで，計算ステーションの設計として別のものを考える．これは，Largish プログレッションといくつかの点で大きく異なる (図 3.3)．

[12]訳注 原文の $\langle\!\langle 2, 100120 \rangle\!\rangle_{\mathrm{R}}$ は誤記．$\langle\!\langle 2100 \rangle\!\rangle_{\mathrm{R}}$ が正しい (Tromer 氏に確認済)．

[13]バッファへの放出の frequency は s/p_i，そして，各々の放出はいくつかの配達セルを $\Theta(s)$ クロックの間，使う．式の最後 2 項は DRAM 保存と無視できる定数である．

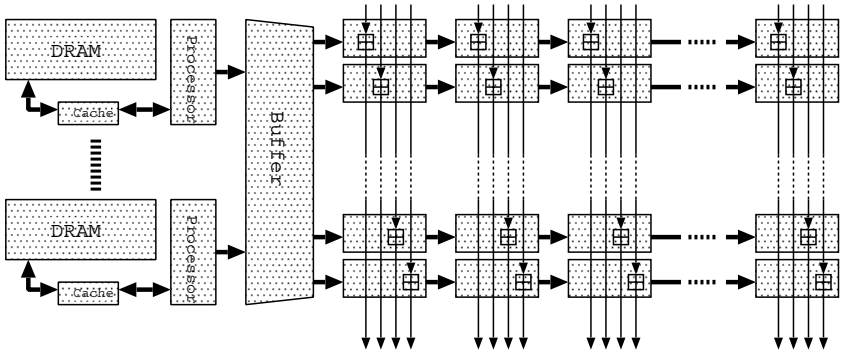


図 3.2: Largish ステーションのブロック図

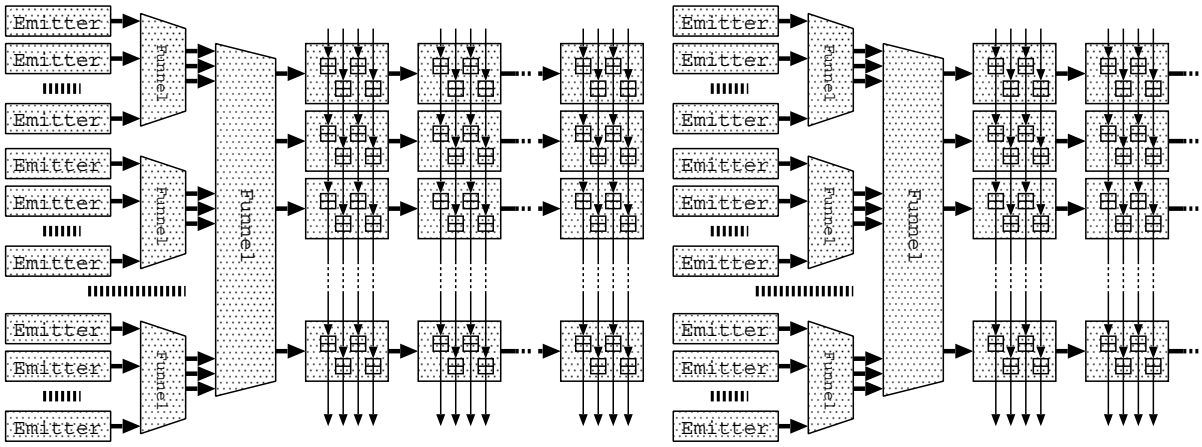


図 3.3: Smallish ステーションのブロック図

エミッタと煙突　まず，プロセッサ，メモリ，バッファを別の装置に置き換える．デリバリペアは直接 emitter, エミッタから放出される．これは，小さな回路であり，単一のプログレッションを担当する (TWINKLE のように)．エミッタは内部レジスタを使ってプログレッションの状態を管理し，時おりデリバリペア $(\lfloor \log p_i \rceil, l)$ を放射する．これは，ある指定のサイクル数が過ぎてから，バス線 l に $\lfloor \log p_i \rceil$ が加算されるべきである，ことを意味する．第 3.7.2 節に小さいエミッタの設計について記述する．

各々のエミッタは内部カウンタを随時更新する．しかし，デリバリペアはせいぜい $\sqrt{p_i}$ サイクル，すなわち $\langle\!\langle 8 \rangle\!\rangle_{\mathrm{R}}$ から $\langle\!\langle 512 \rangle\!\rangle_{\mathrm{R}}$ サイクル，に一度ほどしか放射されない．そこで，各々のエミッタを専用のデリバリラインに接続するのは無駄である．そこで，funnel, 煙突を使うことで，これを回避する．煙突は入力を以下のような処理により圧縮する．煙突は多数の隣接エミッタから来る出力につながっている巨大な入力線を持つ．ここで，入力が，1 次元配列であり，ほとんどの要素が空要素だと考える．煙突は各段に小さい配列を出力し，その非空要素は，さきほど得た巨大な入力配列における僅かな非空要素となっている．煙突の出力はデリバリラインに接続される．第 3.7.3 節は改良シフトレジスタを使った煙突の実装を示す．

Duplication (複製)　もうひとつの大きな変更点として，progression state, プログレッション状態の複製 (duplication) がある．これにより，デリバリペアをより目的のバスに近くなるようにし，バスを交差する回数を減らす．まず，各々のプログレッションが $n_i \approx s/\sqrt{p_i}$ 個の独立したエミッタ[14]により処理されるようにし，これら複数のエミッタが，バスに交差して一定間隔におかれているものとする．よって，全体のデリバリラインを複数の segment, セグメントに分割し，各々のセグメントが $s/n_i \approx \sqrt{p_i}$ 個のバスを担当するようにする．各々のエミッタは (煙突を通じて) ことなるセグメントに接続され，$p_i/sn_i \approx \sqrt{p}$ クロック毎にこのセグメントに放出データを送信する．放出データは目的のセルに早く到達できるので，全体のデリバリラインの本数を削減できる．また同時に，任意の放出 frequency も減らすことができる．これにより，s 程度，場合によってはそれ以下の p_i を扱うことができる．全体で (さまざまなサイズに分割されたセグメントに分割された)$\langle\!\langle 501 \rangle\!\rangle_{\mathrm{R}}$ 個のデリバリ

[14] $\langle\!\langle n_i = s/2\sqrt{p_i} \rangle\!\rangle$ の 2 冪への丸めである (第 3.7.2 節参照)．これは $\langle\!\langle \{2, \ldots, 128\} \rangle\!\rangle_{\mathrm{R}}$ の範囲となる．

ラインが，Smallish プログレッションの計算ステーションに実装されることになる．

注意 3.4.2. 漸近的には，*Smallish* プログレッションに対する消費回路規模は，$\Theta((s/\sqrt{p_i}+1)(\log s+\log p_i))$ である．定数項 *1* は記述するほどのものでもなく，大きな定数を隠すものであって *(*おおよそ，エミッタのサイズと煙突のサイズ程度*)*，大きな p_i に対するコストの主要な部分となる．

3.4.4 Tiny primes (Tiny プログレッション)

極小さな素数の場合，エミッタを複数持つ場合のコスト，特に関連して実装される煙突のコストが極めて高くなる．逆に，このようなプログレッションでは，毎クロック複数の出力が生じるため，複数のプログレッションに渡るデリバリラインのコストを支払うことは，さして重要ではない．以上の議論により，極小さな素数 $\langle\langle p_i < 256 \rangle\rangle$ 用のステーションデザインが導かれる．このようなプログレッションは数少ないが，出力間隔が短いため，その影響は非常に大きい．各々の極小プログレッションは専用の一本のデリバリラインを用いて独立に処理される．デリバリラインは p_i よりも幾分小さいサイズのセグメントに分割され[15]，各セグメントの入力部にエミッタが直接配置される．煙突は配置されない (図 3.4 参照)．このエミッタは Smallish progression 用のエミッタの機能縮小版である (3.7.2 参照)．Largish ステーション，Smallish ステーション用[16] のデリバリセルとは異なり加算器はインターリーブされないが，CSA (carry-save adder) については，小さな定数である $\lfloor \log p_i \rceil$ を条件に応じて加算するだけであるので小さくなる．プログレッション中の占有面積はデリバリラインによるものが支配的であるので，p_i に依存せずに，$\Theta(s)$ となる．

第 3.7 節では追加デザインを考察している．

3.5 コスト見積もり

ここまでで，デザインを概観し，問題の大きさを明示した．次に，仮説上の TWIRL デバイスのコストを，今日の VLSI 技術を用いて見積もってみる．ハードウェアのパラメータは第 3.8.1 節に示した．本見積もりは現実的な数値を見出すことを目的

[15] セグメントの長さは p_i より小さな最大の 2 の冪 (第 3.7.2 節参照)．
[16] 訳注 原文の誤記と思われる．Smallish ステーションでは加算器はインターリーブされない．

としたものであるが，極めて大雑把なものであり，多くの近似と仮定によっていることを強調しておく．本見積もりは真のコストのオーダを示す程度のものである．詳細な VLSI 設計は未実施であり，実際の実装についても言うまでもなく未実施である．

3.5.1 1024 ビット合成数用篩のコスト

数体篩に用いるパラメータとして以下を仮定した．$B_R = 3.5 \cdot 10^9$, $B_A = 2.6 \cdot 10^{10}$, $R = 1.1 \cdot 10^{15}$, $H \approx 2.7 \cdot 10^8$(第 3.8.2 節参照) ここでは，第 3.7.6 節に示すカスケード篩を用いている．有理数篩では，$s_R = 4096$ とした．一つの有理数 TWIRL デバイスは 15960 mm^2 のシリコンウェハ面積 (30 cm ウェハの 1/4 に相当) を必要とする．この面積のうち，76%が Largish プログレッションで占められ (さらに装置全体の 37%は Largish プログレッションの DRAM バンク)，21%が Smallish プログレッション，残り 3%が Tiny プログレッションで占められる．代数篩では，$s_A = 32768$ とした．一つの代数 TWIRL デバイスは 65900 mm^2 のシリコンウェハ面積 (30cm ウェハ一枚に相当) を必要とする．この面積のうち，94%が Largish プログレッションで占められ (装置全体に対する 66%は DRAM バンク)，6%が Smallish プログレッションで占められる．その他のパラメータについては第 3.4 節で説明してある．

8 つの有理数 TWIRL と 1 つの代数 TWIRL とで，1 つのクラスタが構成される．各々の有理数 TWIRL から代数 TWIRL に入力する一方向性の結線があり，結線上をクロック毎に 12 ビットのデータが転送される．1 つのクラスタは 3 枚の 30 cm ウェハからなり，R/s_A クロック (1 GHz クロック時に 33.4 秒) 毎に一本の篩ライン上のすべての篩ポイントを処理する．篩処理全体は H 本の篩ラインからなり，クラスタ一個で処理するには 194 年を要する (第 3.7.5 節に記述する手法による 33%処理削減時)．$2.9 M のコスト (ウェハ一枚$5,000 換算) で，194 個の独立した TWIRL クラスタを作成することが可能であり，これらによる並列処理実施時に，篩処理を 1 年で完了することができる．

パッケージ，電源，冷却装置，データ収集用 PC のコストを加え，更にその他マージンを見込んだ結果[17]，製造コスト$10 M の装置を用いて，1 年間で 1024 ビット合成数を素因数分解するのに必要な全ての篩処理を完了可能であることが明らかとなっ

[17]経験則によれば半導体コストの 2 倍だが，ここでは 3 倍とした．

た．尚，上記のデバイス当たりのコストに加え，初期コストとして$20M 程度の開発費 (設計，検証，マスク作成など) が必要である．

3.5.2 1024 ビット合成数に対する影響

1024 ビット RSA 鍵は今後 15〜20 年は安全であるとしばしば言及されてきたが，それは NFS の篩処理と行列演算が共に実現不可能 (例えば文献 [6, 51] とドラフト版 NIST ガイドライン [41]) であることを根拠にしている．本評価は$10M のコスト (と$20M の初期コスト) を用いて 1 年で篩処理を実現可能であることを示しており，又，近年の研究 [35, 17] は，我々が採用した数体篩パラメータに対して行なう行列演算処理が，同等のコストで実現可能であることを示している．

篩と行列演算を処理する効率的な専用ハードウェアが実現すると，NFS アルゴリズム中の他の処理が新たなボトルネックとして浮かび上がることが考えられる[18]．又，本見積もりは仮説的なものであり数多くの近似によっており，正確なコストを明らかにするには篩処理の実証実験が必要であろう．

本研究結果からは，個人的な攻撃者により 1024 ビット RSA が解読可能である，ということを言えるわけではない．しかしながら,「動機と資力の十分にある何らかの組織がここ 2〜3 年のうちに 1024 ビット合成数向け数体篩処理を適用することが考えられない」，といえるだけの事由を特定するのは困難である．1024 ビット RSA 鍵を利用しようとする際には，上記事情をよく考慮する必要があるであろう．

3.5.3 512 ビット合成数用篩のコスト

512 ビット合成数を対象とした篩ハードウェア設計がすでにいくつか提案されているので [48, 34, 24, 16]，TWIRL のコスト見積もりにそれら先行研究と同一のパラメータを用いることは有益である．ここでは，文献 [34, 16] と同一のパラメータを仮定した．即ち，$B_R = B_A = 2^{24} \approx 1.7 \cdot 10^7$, $R = 1.8 \cdot 10^{10}$, $H = 1.8 \cdot 10^6$ 又，s=1024 とし，1024 ビット合成数向けの場合と同様のコスト見積もり手法を用いた[19]．

TWIRL デバイス一つは，約 800 mm^2 のダイサイズとなり，このうちの 56%が

[18] 我々が用いたパラメータ選択に関しては，cofactor の素因数分解処理をおこなうとき，篩にかかったコストよりも低コストで実現できる (第 3.7.7 節参照)．

[19] 訳注 原文の $2H = 1.8 \cdot 10^6$ は誤記か．

Largish プログレッション用の面積であり，残りのほとんどが Smallish プログレッション用の面積である．一本の篩ラインを 0.018 秒で処理し，篩処理全体を 6 時間で完了する．

同じパラメータを用いた場合，1 ライン処理するのに，TWINKLE では 1.8 秒，文献 [24] の FPGA ベースデザインでは約 10 秒，文献 [16] のメッシュ構造デザインでは 0.36 秒掛かる．TWINKLE や文献 [16] のデザインと公平な比較を行なうため，ウェハ一枚を使用した TWIRL デバイスを考え，これを並列に動作させた場合を考える．その場合，ウェハ一枚には 79 個の TWIRL が載り，1 ライン処理するのに要する時間は 0.00022 秒となる．

以上，512 ビット合成数を対象とした篩処理において，TWIRL デバイスは先行研究中最速である文献 [16] のデザインと比較しても，1600 倍高速であり，TWINKLE よりも 8100 倍高速である．カスケード篩版 (第 3.7.6 節参照) 用に数体篩パラメータを調整すれば，この速度差は更に広がるが，基本設計版であっても，ウェハ一枚一杯に TWIRL デバイスを載せて並列動作させれば，512 ビット合成数の篩処理を 10 分以下で完了可能である．

3.5.4 768 ビット合成数用篩のコスト

ここでは以下の数体篩パラメータを用いた．$B_R = 1{\cdot}10^8$, $B_A = 1{\cdot}10^9$, $R = 3.4{\cdot}10^{13}$, $H \approx 8.9 \cdot 10^6$(第 3.8.2 参照) 又，$s_R = 1024$，$s_A = 4096$ として，第 3.7.6 に示すカスケード篩版の TWIRL を用いた．この場合，有理数篩は 1330 mm^2，代数篩は 4430 mm^2 の面積となる．有理数篩 4 個と代数篩 1 個からなる処理クラスタは篩線 1 本の処理に 8.3 秒を要し，30 cm シリコンウェハ一枚に，独立した 6 つのクラスタが搭載可能である．

このことから，ウェハ一枚構成の TWIRL クラスタは篩処理を 95 日間で完了可能である．このウェハの製造コストは，RSA-768 チャレンジの優勝賞金の 1/10 に当たる$5000 である[20]．

[20] 言うまでもなく，この製造コストは初期費用$20M を無視している．この初期費用は，例えば 0.25 μm プロセスなどの，枯れたプロセスを利用すれば極めて小さくすることができる．但し，篩処理のスループットは幾分犠牲になる．

3.5.5 大きな合成数

Largish プログレッションの場合，プログレッション一つ当たりのコストは，小さな定数を含んで $\Theta(s^2(\log s)/p_i + \log s + \log p_i)$ である (第 3.4.2 節参照)．Smallish プログレッションの場合，ずっと大きな定数を含んで $\Theta((s/\sqrt{p_i}+1)(\log s + \log p_i))$ である (第 3.4.3 節参照)．PC を用いた，又は TWINKLE デバイスによるシリアル実装では，プログレッション一つ当たりのコストは明らかに $\Omega(\log p_i)$ である．このことから，シリアル実装で $\tilde{\Theta}(\sqrt{B})$ レベルの速度優位を得るために，漸近的に s として $\tilde{\Theta}(\sqrt{B})$ を選択することができるが，定数を小さな値に保つ必要がある．実際には，Largish 用のプロセッサ，バッファ，デリバリラインが，プログレトリプルデータを保管する DRAM の面積と同等の面積になるまでは，本質的にはコストなしに，s 値を増加させることが可能である．

入力値のサイズによっては，素数 p_i だけを記憶することにして，Largish プログレッション記憶用の DRAM 容量を減らした方が効率的な場合もある．その際，プログレトリプルデータの他の値は特定用途プロセッサで on-the-fly に生成すればよい (対応する数体篩多項式の法 p_i における根を求める必要がある)．

デバイスがシリコンウェハ一枚の容量を超えてしまう場合でも，ウェハ間を結線するバス線の幅がウェハよりも狭ければ，ウェハ数枚に必要なだけのステーションを搭載することが可能である．この場合，ウェハは，ウェハ間を全て貫通するバスによってシリアルに接続される．

3.6 結論

本稿では，篩処理専用デバイス設計についての提案を行なった．本デバイスは篩ポイントを転送するための密で太いパイプラインで構成され，各々の篩ポイントをスリリングな冒険へと導く．篩ポイントは，様々に変化するスケールで周期的に動作するプログレッションから加算という経験を繰り返される．これには夥しい数の様態からなりこれらは，周期の多様性に最適化している．本デバイスは，512 ビット合成数の篩処理において，これまで知られていた最速の手法よりも 1600 倍高速である．また，$10M のコストで，適切な数体篩パラメータを選択した場合，1024 ビット合成数の篩処理を 1 年で完了可能である．この結果は，1024 ビット RSA 鍵

の安全性に何某かの懸念を引き起こすものである．

謝辞

この研究は，数体篩における行列ステップに関する，Daniel J. Bernstein の先行的な研究，ならびに Willi Geiselmann と Rainer Steinwandt らによる篩処理への適用研究らにより影響されたものである．特に後者については，彼らの設計に対する興味深い議論，そして我々の手法への改良案の提案に対して感謝する．また，Arjen K. Lenstra には先行的議論の数々，Robert D. Silverman と Andrew “bunnie” Huang, Michael Szydlo には，貴重なコメントと提案を頂いたことに感謝する．[29] の準備稿と，Jens Franke と Thorsten Kleinjung らによる多項式選択のプログラムは，数体篩の評価改訂を行なうために，不可欠であったことをここに記す．

3.7　デザインの追加考察

3.7.1　デリバリライン

デリバリラインは，デリバリペアをそのソース (バッファ，煙突，エミッタ) から送り先となるバスラインへと移動させるものであり，全てのステーションが備える．その基本的な構造は既に 3.4.2 で示した．ここでは，効率的実装手法について述べる．

インターリービング　ほとんどの動作時間中，デリバリラインのセルはシフトレジスタとして動作し，セル中の加算器は利用されない．このことから，インターリービングによって，加算器とレジスタを減らすことを考える．隣接する $r\langle\!\langle = 4 \rangle\!\rangle_{\mathrm{R}}$ 本のバスラインをカバーし，且つ同 r 本のバスラインのうちの q 番目のラインを処理する加算器を一個だけ有するデリバリセルを用いる．パラメータ q は，一つのデリバリライン内では固定されており，縦に続く他のデリバリラインでは循環的にインクリメントされた値 (1〜4) を用いる．更に，r 個の隣り合うデリバリラインで一つのバスパイプラインステージを構成するので，バスライン毎に一個の加算器があることになる．以上により，Largish ステーション全体に渡って，バスパイプラインレジスタの数が $1/r$ に減少する．

emission pair は 1 クロック毎に，r ライン分デリバリライン上を横切るので，篩

ポイントの時空間配置をゆがませる必要がある．その配置は，バッファ，バスライン間の距離が増えるほど，篩ポイントの “age” すなわち年齢 $\lfloor a/s \rfloor$ が減るようにすべきであり，より明確には以下のようになる．時刻 t において，篩ポイント a は (ステージ $t - \lfloor a/sr \rfloor - \lfloor (a \bmod s)/r \rfloor$ において，r 本あるデリバリラインの一つにおける) $\lfloor (a \bmod s)/r \rfloor$ 番目のセル[21]が担当する．

Largish ステーションにおいて，バッファは，適切なバスライン上に加算器一個を有するデリバリラインへと，デリバリペアを送出する機能を果たす．各々のデリバリラインのちょうど半分づつが，バッファから外側に向くよう，バスの中間にバッファを配置することで，最悪到達時刻が半分に短縮される．Smallish，及び Tiny ステーションにおいては，インターリービングは用いない．

尚，バス上にパイプラインレジスタを配置する際には，このバッファに接続される全ての下流のデリバリラインを遅延させる必要があるが，これは，デリバリラインの最初のところに，パイプラインステージを加えることで実現可能である．

桁上げ保存加算器　論理的には，全てのバスラインは各々 $log_2 T \langle\!\langle = 10 \rangle\!\rangle$ ビットの整数値を取り扱う．この整数値は冗長表現され，その 2 数の和が，$\lfloor \log p_i \rceil$ の和と等しい $log_2 T$ ビットの整数ペアで表される．デリバリセルに搭載する加算器は，carry-save-adder(桁上げ保存加算器) とする．桁上げ保存加算器とは，a, b, c からなる 3 値の入力に対し，$e + f = a + b + c$ を満たすような e, f の 2 値を出力とする加算器であり，桁をまたぐキャリーの伝播がないことから，非常に小規模且つ高速に動作するという特徴を持つ．桁上げ保存加算器の大きな欠点は，冗長表現された数値を入力として他の演算を行なうのが困難であることであるが，TWIRL デバイスで必要なのは，一続きとなる多数の加算と，最終時点での一回の比較であり，(桁上げ保存加算器の欠点は問題にならない)．数値の冗長表現に伴って追加されるバス配線は，シリコンウェハの多層配線によって搭載可能である[22]．

桁上げ保存加算器に小変更を加え，MSB を固定化することで，$\lfloor \log p_i \rceil$ の和が T よりもかなり大きい時に生じる桁あふれと 0 クリアを防止している．これは，冗長表現値の MSB が複数 1 である場合 (合計が T 以上である場合)，それらの加算時に

[21] 第 3.7.2 節に示す変更を施すと，これは $\lfloor \mathrm{rev}(a \bmod s)/r \rfloor$ となる．ここで，$\mathrm{rev}(\cdot)$ は $\log_2 s$ ビット文字列の逆さ読み (最上位ビットを最下位ビットに置く，など) とし，s, r はそれぞれ 2 冪の数とする．

[22] この手法が疑わしいのであれば，通常の整数表現の数値を用いた carry-lookahead adder (桁上げ先見加算器) を用いればよい．回路規模と処理クロック周波数で若干マイナスに作用するが．

MSB を 1 に固定し 0 に戻さないことで実現している．

3.7.2 エミッタの実装

Smallish，Tiny progression のデザインには，エミッタが必要となる (第 3.4.3, 3.4.4 節参照)．エミッタ一個は，プログレッション P_i 一つを担当し，隣接するラインからなるグループ G を転送先とするデリバリペア $(\lfloor \log p_i \rceil, l)$ を出力する．又，$l \in G$ である．本節では，エミッタの実装方法について提案する．幾分効率的でない手法から，順を追って説明する．

単純実装　単純実装としては，$\lceil \log_2 p_i \rceil$ ビット長のレジスタを搭載し，クロック毎に上記レジスタ値に s を加算する，ただしレジスタ値は p_i を法とする剰余とする，という構成が考えられる．レジスタ値にラップアラウンド (つまり，そのプログレッションを出力すること) が生じる毎に，l 値を算出し，その算出値が $l \in G$ を満たすかどうかを判断する．これには，レジスタ値をクロック毎に更新するために，高コストな桁上げ先見加算器が必要となる．更に，s と $|G|$ を任意に選択した場合，l を算出して $l \in G$ を満たすか否かを評価するコストも高くなる可能性がある．2 の冪となる s と $|G|$ を選べば，幾分コストは低くできる．

別の実装としては，文献 [48] にあるような，次の出力までの時間をカウントダウンするカウンタと l の軌跡を保持するレジスタを持つ構成が考えられる．これには更に 2 種類の改良型がある．出力先バスラインと無関係に次出力までを計数するダウンカウンタを用いるとすると，このような出力はかなり頻繁に生じるため，やはり遅延の短い回路が必要となる (又，$p_i < s$ となる p_i を扱うことができない)．そこで，対応する出力がバスライン G に対するような次出力までを計数するダウンカウンタを用いる場合を考える．これは以下のような別の問題が生じる; 例えば G が隣接するようなバスライン (ライン番号は法 s の剰余で考える) だけに放出しようとすると，どのようなバスライングループ G についても言えることだが，P_i の出力が時刻として規則的でなくなる．よって，その不規則な間隔を計算するための高価な演算回路が必要となる．

ラインアドレスのリバーサル表記　上記問題を解決するために，上記で示したダウンカウンタを用いる 2 手法のうち後者を考える．(1 クロック内での) バスライン

に対する篩ポイントの割り当ては任意とするが，バスライングループ G への結線は，物理的な近接度に基づいて割り当てる．よって，以下に示す手法を用いる．整数 $\alpha\langle\!\langle = 12\rangle\!\rangle_{\mathrm{R}}\langle\!\langle = 15\rangle\!\rangle_{\mathrm{A}}$，$\beta_i$ に対し，$s = 2^{\alpha}$，$|G| = 2^{\beta_i} \approx \sqrt{p_i}$ を選択する．バスライン ID を示す，法 s における剰余の 2 進表記を，リバース (上位-下位反転) したものをインデックスとしてバスラインに割り当てる．即ち，法 s の剰余 w に一致する篩ポイントは，物理位置 $\mathrm{rev}(w)$ のバスラインに割り当てる．ここで，

$$w = \sum_{i=0}^{\alpha-1} c_i 2^i, \quad \mathrm{rev}(w) = \sum_{i=0}^{\alpha-1} c_{\alpha-1-i} 2^1, \quad \text{for some } c_0, ..., c_{\alpha-1} \in \{0, 1\}$$

である．プログレッション P_i，$j \in \{0, ..., 2^{\alpha-\beta_i}\}$ に対応する j 番目のエミッタは，j 番目の 2^{β_i} バスライングループを担当する．この選択の利点を以下に示す．

補題 3.7.1. $p_i > 2$ であるような，任意の固定したプログレッションに対し一定のグループを出力先とするプログレ放出は，常に $T_i = \lfloor 2^{-\beta_i} p_i \rfloor$ 間隔となる．但し，法 s 効果に起因する *1* クロックの遅延が，場合によって生じる．

Proof. j 番目バスグループへ作用するプログレ放出とは，$\lfloor \mathrm{rev}(a \bmod s)/2^{\beta_i} \rfloor = j$ を満たすような篩ポイント $a \in P_i$ であって，これは，ある定数 c_j により $a \equiv c_j \pmod{2^{\alpha-\beta_i}}$ であることと等価である．$a \in P_i$ であるとは，$a \equiv r_i \pmod{p_i}$ かつ，p_i が $2^{\alpha-\beta_i}$ と互いに素であること，であることから，中国人剰余定理 (CRT) によりそのような篩ポイントの集合は正確に $P_{i,j} \equiv \{a : a \equiv c_{i,j} \pmod{2^{\alpha-\beta_i} p_i}\}$．よって対応するプログレ放出の時間差 Δ は，$\Delta = \lfloor a_2/s \rfloor - \lfloor a_1/s \rfloor$ となる．$(a_2 \bmod s) > (a_1 \bmod s)$ ならば $\Delta = \lfloor (a_2 - a_1)/s \rfloor = \lfloor 2^{\alpha-\beta_i} p_i / s \rfloor = T_i$ となる．それ以外の場合，$\Delta = \lceil (a_2 - a_1)/s \rceil = T_i + 1$ □

ここで，β_i の選択から，$T_i \approx \sqrt{p_i}$ である．

エミッタの構造　Smallish ステーションの場合，エミッタは以下の二つのカウンタを備える．

カウンタ A このカウンタは法 $T_i = \lfloor 2^{-\beta_i} p_i \rfloor$ で動作するものとする (一般的には $\langle\!\langle 7\rangle\!\rangle_{\mathrm{R}}\langle\!\langle 5\rangle\!\rangle_{\mathrm{A}}$ ビット)．これにより処理対象としているエミッタの次回プログレ放出までの時間を示し，概ねクロック毎に 1 デクリメントされる．

カウンタ B このカウンタは法 2^{β_i} で動作するものとする (一般的には $\langle\!\langle 10 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 15 \rangle\!\rangle_{\mathrm{A}}$ ビット). このカウンタは，次プログレ放出に対応する篩ポイントの (法 s の) 剰余グループの最上位 β_i ビットを示す．カウンタ A がラップアラウンドする毎に，$2^{\alpha-\beta_i} p_i \bmod 2^{\beta_i}$ だけインクリメントする．カウンタ B がラップアラウンドする毎に，カウンタ A は 1 クロックだけ動作を休止する (これによりモジュロ s 効果をキャンセルする).

$\lfloor \log p_i \rceil$ 値はエミッタ毎に固定されており，デリバリライン $(\lfloor \log p_i \rceil, l)$ が，カウンタ A のラップアラウンド毎に出力される．出力対象となるバスライン l はカウンタ B から β_1 値を得る．ライン l の $\alpha - \beta_i$ LSB はエミッタ毎に固定されており，又，デリバリライン中の対応するセグメントを通して固定されている．従って，この値を明示的に転送する必要はない.

エミッタは，接続されるバスライングループに物理的に近接して (あるいはその下に) 配置される．カウンタ値と定数値は，デバイスの初期化時に適切に設定される．デバイスを特定の篩処理専用とするならば，上記値を固定配線とすることで回路規模を削減することが可能である[23]．カウントのビット長は，二つ合わせて大雑把に $\log_2 p_i$ ビット程度であり，適切な調整を行なえば，文献 [34] 同様に，小論理規模のリップルキャリー加算器[24]での実装が可能である.

Tiny プログレッション用のエミッタ Tiny ステーションでも，上記と同一に近い手法を用いている．バスラインは，同様に，ビット表記逆転の手法で物理位置を割当てる (実際のところ，(Smallish と Tiny で結線の) 順序を変更するにはかなりのコストを要する)．Tiny プログレッション用には，$|G| = 2^{\beta_i}$ が p_i よりも小さい最大の 2 の冪となるように β_i を選択した．これにより，$T_i = 1$ に固定され，1 又は 2 クロック毎のプログレ放出出力となる．エミッタの回路は，Smallish プログレッション用と同一であるが，カウンタ A は有名無実となり (単なる結線)，$\beta_i \approx \log_2 p_i$ ビット長のカウンタ B のみとなる.

[23]数対篩の有理数篩では，滑らか境界を固定しても問題はない．Coppersmith の Factorization Factory[11] の予備処理ステージについても同様である.

[24]リップルキャリー加算器を用いるには，小さな遅延の挿入と定数値の微調整が必要となる.

3.7.3 煙突の実装

Smallish ステーションはエミッタからの疎らな出力を纏めてからデリバリラインに渡すために煙突を用いる (第 3.4.3 節参照)．煙突の実装方法を以下に示す．

n 入力 m 出力 ($n \gg m$) の煙突は，m 行 n 列の行列からなり，個々の行列要素は，プログレトリプルデータを一つ保持するレジスタを備えている．クロック毎に，エミッタから最上段行への入力 (1 列当たり 1 入力) があり，t 番目の入力配列の i 番目の要素は，時刻 $t+i$ に i 番目の列に挿入されるようにスケジューリングがなされる．毎クロック，全ての行列要素は右の列へと水平シフトされる．又，毎クロック，他の値を上書きしない場合に限り，全ての行列要素は下の列へとシフトされる．t 番目の出力配列は，時刻 $t+n$ において，最も右側の列から読み出される[25]．

どんな $m < n$ 値を選択したとしても，煙突にオーバフロー (値が一杯に詰まっていて入力不可能な列へ，値を入力する必要が生じる場合) が起きる可能性はある．任意の入力が，他入力との依存関係なしに，確率 ν で入力値がある (空でない) とすると ($\nu \approx 1/\sqrt{p_i}$, 第 3.4.3 節参照)，オーバフローによって入力値が入力できない確率は，以下の式で表される．

$$\sum_{k=m+1}^{n} \binom{n}{k} \nu^k (1-\nu)^{n-k} (k-m)/k$$

$\langle\!\langle m = 5 \rangle\!\rangle_{\mathrm{R}}$ 行，$\langle\!\langle n \approx 1/\nu \rangle\!\rangle_{\mathrm{R}}$ 列の煙突を用いた場合，Smallish プログレッションの範囲では，上記のオーバフローが起きる確率は 0.00011 未満である．この値は，TWIRL デバイスの性質からして十分低い．

上記煙突の圧縮率は $n/m\langle\!\langle \approx 1/5\nu \rangle\!\rangle_{\mathrm{R}}$ であり最適とは言い難い．即ち，煙突出力に値がある (空でない) 確率は $\nu'\langle\!\langle \approx 1/5 \rangle\!\rangle_{\mathrm{R}}$ であり，やや低いと言える．そこで，上記煙突の出力を更に次段の煙突 $\langle\!\langle m' = 35, n\prime = 14 \rangle\!\rangle_{\mathrm{R}}$ に入力する．次段煙突のオーバフロー確率は 0.00016 未満であり，そのコストは数多くのプログレッションに対応することを考えると無視できる．この 2 段目煙突の出力がデリバリラインへと供給される．上記 2 層煙突の圧縮率については，達成率 $5 \cdot 14/34 = 2$ で準最適であり，デリバリライン数は，単純最適化に対して 2 倍となる．適切なデリバリラインにデリバリペアを割り当てるオーバヘッドの発生を嫌って，Largish ステーションで用

[25]訳注 最下段行においては下方向へのシフト動作は実行できず，右シフト動作のみとなる．最下段行に値が詰まっている場合，その上の行の下シフトも不可能になり，一列に下からデータが詰まっていく．

いたようなデリバリライン上の加算器のインターリーブ (第 3.7.1 節参照) は用いなかった[26].

3.7.4 初期化

デバイスの初期化は，全てのステーションに，プログレッション状態とカウンタの初期値を与えることと，バスバイパス用の経路再決定スイッチに制御命令を与える (デバイス上の欠陥位置特定後) ことからなる．

プログレッションは篩の実行毎に異なるが，デバイスの再設定には長時間を要する (文献 [48] ではこの時間がボトルネックとなった)．文献 [16] の手順により，デバイス再設定を回避可能である．この手順は，数体篩で生じる篩問題間には強い関連があり，各篩実行後に定数値 $\tilde{r}_i$ を r_i に加算することだけが必要であるという特徴から得られる．$\tilde{r}_i$ 値はプログレッション当たり $\log_2 p_i$ ビットの小容量で DRAM 内に保管可能であり (この点はコスト見積もりに織り込み済み)，加算はウェハ上の特定用途プロセサで効率的に実行可能である．この値の更新までの時間，R/s クロックが大変大きいことから，更新を高速に実行するために大きなリソースを専用に割く必要はない．代数篩では，事情は幾分異なっている (第 3.7.8 節参照)．

3.7.5 篩ポイントの除外

数体篩の関係式収集処理では，b 番目の篩線上の篩ポイント a のうち，$\gcd(a', b)$ のものだけが対象となる，ただし，$a' = a - R/2$．それ以外の篩ポイントは関係式を重複させるだけだからである．重複する篩ポイントは候補評価処理で結局除外されるが，とても小さい c に対して，$c|a', b$ となる篩ポイントを予め除外しておくことで，篩処理を削減することが可能である．全てのソフトウェアベースの篩処理では，$2|a', b$ となる場合を考慮している．この場合，25%の篩ポイントを除外することが可能である．TWIRL でも同様の手法を用いる．まず，全ての奇数ライン $b = 1 (\bmod 2)$ について通常の篩処理を施す．次に，偶数ラインの篩処理を行なうが，この際，a' が奇数となるもののみを対象にする．$p_i > 2$ であるプログレッションは，奇数篩ポイントを p_i ステップおきにヒットするので，必要な変更は，メモリとカウンタの初

[26] この場合でも，マルチプレクサを用いて数本のバスラインに加算器一個を割り当てることで加算器の数を削減することは可能である．しかし，上記手法は動作周波数に悪影響を与えると思われる．

期値のみである．以上の偶数ラインに対する篩処理に要する時間は，偶数ラインの処理時間の半分となる．

上記と同様にして $3|a', b$ の場合についても考察し，これらを組み合わせることで，full，half，third，sixth-length からなる 4 種類の篩処理を得る．これら分類は，$b \bmod 6$ として各々$\{1,5\}$，$\{2,4\}$，$\{3\}$，$\{0\}$ を示す．本質的なコストなしに，全体では 33%の実行時間削減効果を得ている．$c > 3$ に対して $c|a', b$ の場合の考察を行なう価値はない．

3.7.6 篩のカスケード構成

篩問題の対象となる数値は有理数篩と代数篩の両方で処理されるが，着目すべきは両方の篩処理をパスした a 値である (第 3.3.1 節参照)．しかし，両者の篩は同等のものではなく，代数篩の処理数 $B_A\langle\!\langle 2.6 \cdot 10^{10} \rangle\!\rangle_{\mathrm{A}}$ は，有理数篩の処理数 $B_R\langle\!\langle 3.5 \cdot 10^{9} \rangle\!\rangle_{\mathrm{R}}$ よりもかなり大きいのである[27][28]．そのため，各々の篩に最適な s 値を選択した場合，代数篩のコストが全体の中で占める割合が高くなる．しかも，1024 ビット合成数に対してパラメータ (第 3.8 節参照) を選択した場合，代数篩用の s 値を理想的に大きくとることは不可能である．理想的な数値を用いた場合，代数篩がシリコンウェハ一枚に収まらない大きさとなるためである．この問題に対する対策を以下に示す．

s_R，s_A は，有理数篩，代数篩それぞれの s 値を表す．まず，s_A を増やし，自由に扱える並列化が不可能な理由として，これを実装すると制御不能になるほどバス幅が広くなり，デリバリラインの数，長さが共に増加する (そのコストは $\Theta(s^2)$ である) ことがあげられるためである．しかし，バスは，パイプラインステージ当たり s_A 個の篩ポイントを処理するよう作られている．これは，最初に有理数篩処理を行なうことで，ほとんどの篩ポイントを前もって除外しまうことが可能であり，極少量 $\langle\!\langle 1.7 \cdot 10^{-4} \rangle\!\rangle$ 個の篩ポイント以外の全ては，有理数篩の閾値をパスせず[29]，これらは，代数篩処理を待たずに候補者とはなりえない．

上記から，代数篩の構成を以下のように変更した．単純に，法 s_A の剰余ひとつひとつに s_A 本のバスを割り当てる幅広バスではなく，たった $u\langle\!\langle = 32 \rangle\!\rangle_{\mathrm{A}}$ 本のラインか

[27] (暗黙的に) 篩処理の対象となる数値が**準滑らか**, semi-smooth となる確率が十分となるよう，B_A と B_R を選択している．約 $\langle\!\langle 10^{64} \rangle\!\rangle_{\mathrm{R}}$ に対して約 $\langle\!\langle 10^{101} \rangle\!\rangle_{\mathrm{A}}$.

[28] 訳注 原文の B_R は B_A の誤記と思われる．

[29] cofactor の因数分解の前の場合．$\lfloor \log p_i \rceil$ による丸めを考慮した場合にはもう少し多くなる．

らなる非常に狭いバスを用いる (ただし，各々のラインがデータ対 (C, L) を扱う)．ここで，$L(= a \bmod s_A)$ は篩ポイントを示し，C は a に対する (これまでの) $\lfloor \log p_i \rceil$ の和である．篩ポイントはクロック毎に s_A ポイントの割合でパイプライン処理される．すべてのデリバリペアはそれぞれのステーション構成に応じ，従来法と同様にして生成される．

デリバリラインの構成も異なっている．長くて「おろかな」構成ではなく，短くて「賢い」構成である．例えばデリバリペア $(\lfloor \log p_i \rceil, l)$ が生成されたとする．デリバリペアがセルに到着する度に，デリバリペア内の l が，篩ポイントの情報 L と比較される．比較の結果が一致すれば，該当ラインの C 値に $\lfloor \log p_i \rceil$ が加算され，不一致の場合 (この場合がほとんどだが) には，デリバリペアは破棄される．

バスの初段入力には，有理数篩をパスした篩ポイント a に対して $(0, a \bmod s_A)$ 値を入力するが，このためには，有理数篩の出力を代数篩の入力に接続し，有理数篩，代数篩の両者を同期させて動作させる (位相シフトを要する)．s 値が一致していないので，s_A/s_R 個の有理数篩を代数篩に接続することになる．よって，$s_A/s_R + 1$ 個の篩デバイスで構成される**クラスタ**が，同期して動作することで，一度に一本の篩ラインを処理する[30]．複数の有理数篩で処理を分割するために，篩ポイントのインターリービングを行なう (第 3.7.2 に示したビット逆転表記手法と類似)．有理数篩，代数篩間の接続は，1 クロック当たり $\log_2 s_R \langle\langle 12 \rangle\rangle$ ビット長の値を高々一つ転送する (平均化して輻輳を無くすためには適切なバッファが必要)．

上記変更により，バスワイヤとデリバリラインが占める回路面積は大幅に削減される．本稿が選択したパラメータでは，これらのコストはひじょうに些細なサイズとなった．また，Smallish プログレッション用のエミッタを複製する必要がない (但し，$p_i < s$ の場合を除く)．これによって，代数篩用に大きな $s\langle\langle = 32768 \rangle\rangle_{\mathrm{A}}$ を用いることが可能となり，コストについても有理数篩未満に抑えることができる (第 3.5.1 節参照)．更に，コスト面に大きな影響を与えずに B_A を更に大きくすることが可能であるが，これは H と R を小さくすることにもなる (数体篩パラメータ選択上のトレードオフによる)．

[30]訳注 原文の s_A/s_B は s_A/s_R の誤記と思われる．

3.7.7 候補の検査

各々の篩ポイント a に対して対数の和 $g(a)$ の近似を計算し終えると，結果として残った候補を認識し，対応するプログレッション集合 $\{i : a \in P_i\}$ を計算し，さらに詳細な検査 (第 3.3.1 節参照) を行なう必要がある．これらは以下のように実装される．

候補の認識　各々の TWIRL 装置には，バスの末端 (すなわち，すべてのステーションの下流) に比較器をバスラインに一つ設置している．これは篩ポイント a に対して，$g(a) > T$ となるものを識別するためである．基本的な TWIRL の設計では，篩装置の組 (すなわち，有理数篩と代数篩) を結合して動作させる．毎クロック，比較器閾値を越えたバスラインについて二つの装置間で通信する．そしてその共有部分 (すなわち候補) が識別される．カスケード改良では，有理数篩を通過した篩ポイントのみが代数篩への処理に進まない．よって，この構成法ならば候補というのは，代数篩の出力に他ならない．ここで最終的に候補となる篩ポイントの割合は，ごくごく小さいものである $\langle\!\langle 2 \cdot 10^{-11} \rangle\!\rangle$.

対応するプログレッションの宣告　各々の候補に対して，有理数篩，代数篩両方についてプログレッションの集合 $\{i : a \in P_i\}$ を計算しなければならない．数体篩法では，比較的小さい p_i 集合の要素については見つけることはそう難しくない[31]．よって，部分集合である $\{i : a \in P_i,\ p_i \text{ is Largish}\}$ のみを見つければ良い．これは，Largish station に最近の p_i を記憶させ，これらを必要に応じて報告させることで実現できる．

これを実現するにあたり，すべての Largish station のプロセッサを貫通する専用パイプラインを 2 チャンネル用意する．まず，第一の**ラインチャンネル**は，$\log_2 s$ ビット幅で閾値比較器よりはじまって上流に向かってデータが転送される (つまり，バスデータの流れと逆である)．第 2 のチャンネルは**因子チャンネル**と呼ばれ，バス幅 $\log_2 B$ のものが下流に向かって流れている．両方のチャンネルは，各々のプロセッサ毎にパイプラインレジスタを保持していて，TWIRL 装置の出力となる．各 Largish プロセッサに対して，我々は**日誌**なるものをつける．これは $\log_2 B$ ビット値

[31] つまり，$F_j(a - R, b)$ の小さな因子を見つけることでよい．ここで，F_j は相当する数体篩多項式であり，b は篩線である．

の循環リストである．毎クロック，プロセッサは値を日記に記入する．その動作は以下の通り; もし，あるクロックでプロセッサがトリプル発行データ $(\lfloor \log p_i \rceil, l_i, \tau_i)$ をバッファに発射したならば，そのデータ $(\lfloor \log p_i \rceil, l_i, \tau_i)$ を日記に記入する．そうでないならば，特別な値の NULL を記入する．もし，あるバスラインで候補が発見されたら，l はラインチャンネルを使って，上流に送信される．プロセッサがラインチャンネルに値 l を検出すれば，日記を確認し，正確に z クロック前にバスライン l にむけて発射したかどうかを確認する．ここで，z はプロセッサ出力からバッファ，バス，閾値比較器，そしてラインチャンネルを通ってプロセッサまで帰ってくるまでの (パイプラインステージとしての) 距離である．この検査は，z クロック前に書かれた日記内容から遡ること，$\langle\!\langle 64 \rangle\!\rangle$ 日記エントリ (ただし，non-NULL 値のみであって $l_i = l$ なる (p_i, l_i)) を検索することで可能である．もし，そのような日記がみつかれば，プロセッサは p_i を因子チャンネルを使って下流へ流す (もし，衝突，すなわち送信のタイミングですでにチャンネルが埋まっているならば，再送信をあとで行なう)．データが混乱し他の異なる候補に属するように認識されたとしても，それは無視できるし，また，適切な割算試行により回復することもできる)．

カスケード篩の改良法では，代数篩では，デリバリラインで破棄されなかったプログレッション加算のみを日記につける．有理数篩の日記は，日記エントリを長い時間保存する必要があるので，かなり大きい ($\langle\!\langle 13530 \rangle\!\rangle_{\mathrm{R}}$ 個のエントリ) 容量が必要である．というのも，遅延時間 z は，すべての有理数篩のバスパイプラインステージ，すべての代数バスラインステージ，そして，すべての有理数ステーションを通る最悪時間である必要がある．しかし，これらの日記は DRAM バンクとして，固定長の周期的アクセス回数による短縮表現 (Largish ステーションのメモリバンクと同様な方法) により効率的に実装できる．

候補の検査　以上の情報が与えられてから，今度はこれまでの篩処理における様々な近似誤差やエラー，そして，数体篩法における large prime (第 3.3.1 節参照) を考慮して処理を進める．最初のステップ (多項式の値を評価し，小さな因子や日記からのレポートを割り除き，サイズを検査し，残った因子の素数判定すること) は専用のプロセッサとパイプライン構造により効率的に処理が可能である．これは，文献 [16] 4 節に記載の因子パイプラインと同様である．ただし我々のモデルでのほうが，より少ない候補を扱う (第 3.9 節参照)．

因子の素因数分解　以上の処理を生き残った (かつ，残った因子が素数判定で失敗，すなわち合成数判定が出て，かつ十分に小さい数字の) 候補は因子の素因数分解を行なう．これは，サイズが $\langle\!\langle 1 \cdot 10^{24} \rangle\!\rangle$ 程度の整数 1 個，もしくは 2 個程度のの素因数分解となる．篩ポイント全体のうち割合として $\langle\!\langle 1 \cdot 10^{-11} \rangle\!\rangle$ 以下の篩ポイントがこのステージに到達する (ここでは，$\lfloor \log p_i \rceil$ 丸め誤差も考慮している)．現在の汎用プロセッサはこの処理を 0.05 秒で処理可能である．よって，専用ハードウェアを使えば，篩コスト全体に比較すれば割合として小さなコストにより処理が可能である．また，アルゴリズム的な拡張をここに適用することも可能である [3].

3.7.8　格子篩

これまで，数体篩法の線篩法をもとに設計を行なった．この線篩はひじょうに大きな篩線の幅 R を持つ．格子篩法 (Lattice sieving, すなわち特殊 q) はより少ない篩ポイントをもつ．しかし，格子篩法はとても短い篩線 ([9] では 8192) を持つ．よって，ここで定義した方法を格子篩の問題へと写す (すなわち線による格子篩) と R が小さくなりすぎてしまう．

TWIRL は以下のようにして効率的な格子篩に適用することができる．s を格子篩の領域の幅に合わせる (これらはだいたい同じ大きさである)．よって，格子線すべてを 1 クロックで処理することを考える．R は篩を行なう格子ブロックにおける篩ポイントの総数である．ただし，この場合 (p_i, r_i) の定義が若干異なる．これはベクトルを使った格子篩法におけるベクトルへと対応づける (ただし，格子に落される前に)．プログレッションの $\bmod s$ ラップアラウンドの処理は，多少厄介になり，プログレ放出を計算するロジックがすべてのタイプのステーションに適用される必要がある．ここで，Largish プロセッサは，原則格子篩をベクトルにより処理することに注意する．つまり，これらプロセッサは値を遠くの将来にポンなげし，次のプログレ放出までは再び見ることもない，という点である．

再初期化は，特殊 q 格子が変化した場合にのみ必要である ([9] では $8192 \cdot 5000$ 篩ポイントおき)．しかし，再初期化自体の処理が重くなる．格子篩法の大きな利得を考慮すれば，高速な再同期のための (やや大きな) 回路を使い，篩う範囲を増やすこと (しかし低い生産率) は得であるが，これらについてはさらなる検討が必要である．

3.7.9 フォールトトレランス 障害許容性

デバイスのサイズが巨大であるため，TWIRL デバイス上には，LSI 製造上の不完全性に起因する局所的な欠陥が複数発生することは避けられない．生産歩留まりを向上させるために，以下に示す変更を加える．

ステーションを構成する，どこかのモジュールに欠陥がある場合，そのステーションは単純に使用されない．各々の種類の予備ステーションを少数設けておくことで，欠陥があるステーションが担当する筈だったプログレッションを処理可能である．

TWIRL デバイスは加算パイプラインを用いるので，バスラインと加算器内の欠陥は大きな問題となる．この問題の対処法としては，バスに沿った予備ラインセグメントを少数設けておき，局所的欠陥をバイパスするために，予備セグメントを用いてバスラインの一部を論理的に置き換えることが可能である．この場合，Largish ステーションの特定用途プロセサによって，バスライン置き換えを目的とする，バスの送出先アドレス (即ち，トリプル発行データの l 値) 変更を容易に実現可能である．Smallish と Tiny ステーションの場合，置き換えを実現するのは困難であるので，本来の $\lfloor \log p_i \rfloor$ 値を加算するのを断念し，置き換えられたバスラインに小さな定数値を加算することで部分的に補償できる可能性がある．篩ステップは極めて粗い (が非常に効率的な) フィルタとして利用されるので，少々のミスジャッジは許容可能である．

3.8 コスト見積もりに用いたパラメータ

3.8.1 ハードウェア

ハードウェアに関するパラメータは文献 [35] を参考にした．これらの数値は文献 [22] にあるものとも矛盾していない．0.13 μm プロセスの 30 cm シリコンウェハは，ウェハ当たり$5,000 のコストであると仮定した．1024 ビットと 768 ビット合成数の処理には，DRAM 向けウェハの利用を想定した．ロジック部では，1 トランジスタ当たり平均 2.8 μm^2 の面積を占有し，DRAM 部では，1 ビット当たり平均 0.2 μm^2 の面積を占有するものとした．512 ビット合成数の処理には，ロジック向けウェハの利用を想定した．ロジック部では，1 トランジスタ当たり平均 2.38 μm^2 の面積を占有し，DRAM 部では，1 ビット当たり平均 0.7 μm^2 の面積を占有するものとした．

クロック周波数は1 GHz としている．これは，よく練られたパイプライン構成のプロセッサでは現実的な数値であることから判断した．

本研究においては，設計に関する全ての主要な構成要素の大まかな見積もりを行なったが，作業過程で，アルゴリズムに関する更なる解析，仮定，シミュレーションが必要であった．第3.4 節で示した選択パラメータによる1024 ビット合成数処理する場合について，主要なものを幾つか示す．Largish 向けの特定用途プロセッサは $\langle\!\langle 96400 \rangle\!\rangle_{\mathrm{R}}$ 個のトランジスタを必要とする．これはバッファ，小容量 (約 14K ビット，p_i に非依存) のキャッシュメモリを含めての値である．Smallish ステーションのエミッタは $\langle\!\langle 2037 \rangle\!\rangle_{\mathrm{R}}$ 個のトランジスタからなり (煙突のコスト含む)，Tiny ステーションのエミッタは $\langle\!\langle 522 \rangle\!\rangle_{\mathrm{R}}$ 個のトランジスタからなる．インターリーブした場合のデリバリセル (Largish ステーションの場合) は $\langle\!\langle 530 \rangle\!\rangle_{\mathrm{R}}$ 個のトランジスタを要し，インターリーブなしの場合 (Smallish，Tiny ステーションの場合) には $\langle\!\langle 1220 \rangle\!\rangle_{\mathrm{R}}$ 個のトランジスタを必要とする．第3.4.2 節に示したメモリシステムは標準 DRAM よりもビット当たり面積が $\langle\!\langle 2.5 \rangle\!\rangle$ 倍必要である．これは，必要となる余裕領域とキャッシュ領域のためである．多層の配線メタル層を用いることにより，バスワイヤ用の配線領域は不要であると仮定している．同様に，多層の配線メタル層を用いることにより，バスワイヤの cross-bus density(交差密度) を $\langle\!\langle 0.5 \rangle\!\rangle$ ビット / μm であるとしている．

半導体デバイスは，多数且つ様々な大きさのモジュールを相互に接続して構成されるものであるので，効率的なレイアウト (本研究では未実施) は重大な課題である．しかしながら，モジュールの種類数は通常の VLSI デザインと較べてとても少なく，また，変更の余地も多々ある．大抵の systolic 設計では，レチクル[32]のサイズよりも大きなデバイスを作ることが可能である．これにはマスク数が一枚，若しくは非常に少ない枚数を用いる．

フォールトトレラント設計 (第3.7.9 節参照) を採用することで，歩留まりはとても高くなり，装置組み立て後の機能テストについても低コスト化が可能になる．しかも，本デバイスは，大抵の VLSI 設計と比較して検出不能故障率がかなり高くても許容可能である．

[32]訳注 reticle: 半導体デバイスの製作工程でパターン露光に使用されるフォトマスクのこと．

3.8.2 篩パラメータ

篩処理のコストを予測するために，対応する数体篩法のパラメータ (R, H, B_R, B_A) を見積もる．我々が採用した値を表 3.1 にまとめる．512 ビット合成数のパラメータは，TWINKLE[34] で用いたものと同じであり，実際の実験 [9] と比較しても無難な設定である．

大きなサイズの合成数に対してそれなりに正確な予測を立てるために，[29] のアプローチに従うものとする．つまり，RSA チャレンジ合成数の RSA-1024, RSA-768[46] 向けの，数体篩法に用いる多項式の具体的ペアを生成する．そしてこれらが生成する関係式を見積もる．数体篩法の多項式の探索は Jens Franke, Thorsten Kleinjun らのプログラムを，多少の調整を行なった上で用いた．我々の 1024 ビット見積りについては，以下の多項式対を選択した．これらは，RSA-1024 合成数を法として共通の整数根を持つ．

$$
\begin{aligned}
f(x) &= 1719304894236345143401011418080x^5 \\
&\quad -69919734888666058610740741860436344 71x^4 \\
&\quad +270860304835695328940509742578513466495213 14x^3 \\
&\quad +46937584052668574502886791835536552277410242359042x^2 \\
&\quad -10107029484257211137178145885069684587770689954539450138 4x \\
&\quad -22666915939490940578617524677045371189128909899716560398434136 \\
g(x) &= 938772308370263069845713674770 27x \\
&\quad -37934895496425027513691045755639637174211483324451628365
\end{aligned}
$$

関係式出現に関する評価は，篩領域に対して，対応する滑らかさ確率関数 [25] を積分することにより求めた．素因数分解が成功するためには，十分なだけ多数の関係式間のサイクルを見つけなければならない．そして，(仮定したとおり) 代数, 有理数の 2 面のうち片側あたりの 2 個 large prime については，関係式の数からサイクル数を予測する手法が知られていない．しかし我々は，従来のより小さな合成数の分解実験と比較しながら，数字が妥当であることを検証した．768 ビット合成数に対するパラメータについても同様に導出した．詳細情報については，専用 web[49] や [29] で入手可能である．

よりよい多項式をみつければ，篩処理のコストを減らすことができる．実際のところ，今回の代数篩の次数は5となっている (これは我々が利用したプログラムの限界によるものである)．しかし，いくぶん高い次数の多項式により，無視できないほどより高い関係式吐き出しが期待されることは，理論的にも，実験的にもさまざまな根拠が知られている．

3.9 先行研究との関連

TWINKLE つづりの見ためからも明らかなように，今回の新しい装置は，従来の篩法に比較して，時間-空間を逆転した性質をTWINKLEと共有している．TWIRLは明らかにTWINKLEよりも高速である．確かに，両者は動作クロック周波数に大差がない．しかし，TWINKLEは1クロックあたり，1篩ポイントを検査するのに対し，TWIRLは数千もの篩ポイントを検査する．しかしながら，TWIRLはTWINKLEに比較して小型である．これは，効率的な並列化とLargishプログレッションのためのコンパクトなDRAMの利用によるものである (TWINKLEはGaAsウェハを用いるが，DRAMはこの上では効率的に実装できない)．もちろん，電子加算パイプラインを用いずに，TWINKLE的な光学式アナログ加算を考えることはできるかもしれない．しかし，法sによる剰余クラス毎に独立な光学加算器を構築することは，事実上困難を伴う．実際，加算するべき値の数が少ないのもあり，検討に値しないように見える．

FPGAに基づく直列篩 KimとMangione-Smithは，現状すでに入手可能なパーツを使う篩装置を提案し，時間-メモリ交換を利用せず，TWINKLEよりもたった6倍遅い程度で処理を行なうことができることを示した[24]．処理速度の向上は，メモリアクセス速度の向上によるものである．いくつかのFPGAチップがあり，各々が複数のSRAMチップに接続しているものである．上記で示したように，この実装手法は処理速度やコストの観点からTWIRLと比較できるものではない．さらに，特定のハードウェアプラットフォームに特化したものであり，より高度な並列度や，より大きな篩問題へのスケーラビリティという点に疑問がある．

低メモリによる篩回路　Bernsteinは，低メモリで実現できる滑らか検査(smoothness test)を完全に篩処理に置き換える方法を提案した[4]．滑らか検査(smoothness test)とは，具体的には，素因数分解における楕円曲線法などである．この手法により，行列演算ステップにおける漸近的な時間 × 空間コストを $y^{3+o(1)}$ から $y^{2+o(1)}$ に削減した．ここで y は分解する合成数の長さに準指数となる値であり，数体篩法のパラメータの選択に依存する．TWIRLと比較すると，TWIRLは $y^{2.5+o(1)}$ のスループットコストがかかる．これは，速度向上が，プログレッション数の平方根にともなって増えるからである(第3.5.5参照)．しかし，これら漸近的評価では，致命的な係数をも消してしまう．現状の実験的経験から，1024ビット合成数については，低メモリによる滑らか検査が従来の篩法と処理速度において，比較となるとは考えられない．より勝っている漸近的な計算量にも関わらず，TWIRLについても同様である．

メッシュ型篩回路　[4]では数体篩法の線形行列処理をメインに扱っているが，一方でSchimmlerのアルゴリズムを使った篩実装法についても触れている．そしてそのコストは $L^{2.5+o(1)}$(つまりTWIRLと同じ)であることが示されている．GeiselmannとSteinwandtはこのアプローチに従い，メッシュ型篩回路の詳細な設計を提案した[16]．従来の篩装置に比較して，[16]もTWIRLも，$\tilde{\Theta}(\sqrt{B})$ 倍の速度向上を達成している[33]．しかしながら，スケーラビリティとコストにおいて，大きな差がでる．TWIRLでは，512ビット合成数については1600倍も，より効率的である．さらにより大きな合成数の場合や，カスケード型篩(第3.5.3, 3.7.6節参照)を採用すれば，それ以上の開きがでる．

理由の一つに以下のことが言える．[16]のメッシュ型計算モデルによるソーティングは，時間遅延の観点からは効率的であり，ゆえに，Bernsteinの行列演算装置[4]として適切だった．ここでは，各々の装置動作開始が前の状態に依存していた．しかし，篩処理では，スループットのみが関係する．時間遅延を無視すれば，小さな回路でより高いクロック周波数が必要となる．例えば，TWIRLの場合，デリバリラインは，ビットサイズ $\langle\!\langle 12+10 \rangle\!\rangle_{\mathrm{R}}$ のデータを単純な一次元一方向にルーティングする．これは，データサイズ $\langle\!\langle 2 \cdot 21.7 \rangle\!\rangle_{\mathrm{R}}$ のプログレッション情報を複雑な二次元メッシュソートするのと対照的である[34]．代数篩については，状況はより極端と

[33]もしくは，[16]についてはこれ以下かもしれない．しかし，漸近的評価は不十分である．得にsmall primeを扱うという観点について．

[34][16]の著者らは，private communicationで，[35]に類似する，プログレッション状態をソート

なる (第 3.7.6 節参照).

[16] の設計では，各々のプログレッションの状態は，$\lceil\tilde{\Theta}(B/p_i)\rceil$ 回，複製される (TWIRL は $\lceil\tilde{\Theta}(\sqrt{B/p_i})\rceil$ 回) か，あるいは別の方法で処理される．これにより，コストが極端に増加する．[16] で示された 512 ビット合成数のための設計パラメータの主な部分については，仮に 2^{17} よりも小さな素数全部がその他の方法で処理されたとしても，メッシュのうち 75% が，複製された値で占有されることになる．別の方法とは，メッシュが認識した本質的な候補を検査するための独立した因子パイプラインであって，12000 以上もの高価な整数割算ユニットを用いる．さらに，ここでは，$p_i > 2^{17}$ のプログレッションから来る $\lfloor\log p_i\rceil$ 加算和がすべてのプログレッションについて滑らかさ (smoothness) と十分な相関があることを仮定している．この仮定が妥当かどうかは不明確である．

DRAM 記憶領域を使いながら Largish prime を処理する TWIRL は現在の VLSI 技術を使って実装しても，その回路規模を十分に小さくできる (90 DRAM ビット vs. 約 2500 トランジスタ [16]).

もし，その装置が複数のウェハにまたがるならば，ウェハ間通信の必要転送速度は，[16] で必要なそれと比べて (バス幅がウェハ面積を越えない限りは) 我々の設計のほうがはるかに低い．また，ウェハ越しのラインに起こる長い遅延を処理することにアルゴリズム的な問題は TWIRL にはない．さらに，ウェハを連鎖的に接続することは，それらを 2 次元的に接続することよりも容易だろう．少なくとも冷却や故障の観点からもそのことが言える．

するのではなく，プログレ放出をルートするような彼らの装置の変形について示唆している．

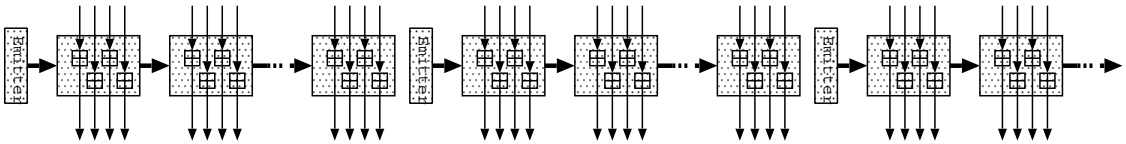


図 3.4: Tiny ステーションのブロック図

表 3.1: 篩パラメータ

	パラメータと意味	1024 ビット	768 ビット	512 ビット
R	篩線幅	$1.1 \cdot 10^{15}$	$3.4 \cdot 10^{13}$	$1.8 \cdot 10^{10}$
H	篩線数	$2.7 \cdot 10^{8}$	$8.9 \cdot 10^{6}$	$9.0 \cdot 10^{5}$
B_R	有理数篩の滑らか境界	$3.5 \cdot 10^{9}$	$1 \cdot 10^{8}$	$1.7 \cdot 10^{7}$
B_A	代数篩の滑らか境界	$2.6 \cdot 10^{10}$	$1 \cdot 10^{9}$	$1.7 \cdot 10^{7}$

第4章　評価対象モデルの詳細

この章では，我々が評価したTWIRL装置についてより詳細に記述する．これは[49]では，扱っていない詳細部分の仕様についてを記載するものである．

4.1 TWIRLの構成

以下，図4.1，及び図4.2を用いて，篩処理に必要な演算/処理装置の構成とTWIRLデバイスの構成を説明し，更に，論理規模低減のためにTWIRLデバイスが採用した構成の利点と，設計上のトレードオフと考えられる点について述べる。

図4.1は，単純な篩処理ハードウェアの全体構成である．篩処理に必要な演算を明らかにすることを目的としており，極素朴な並列/パイプライン実装を想定している．篩処理は，大量の篩ポイントに対して，ある評価演算を行い，その評価値から該篩ポイントが次段処理の候補となるか否かを判断するものである．図中，Rは篩ラインの幅 (=1 篩ライン中の篩ポイント数)，p_i の最大値=(B以下の素数) である．

安直なアルゴリズムの理解では，評価ポイントは，篩ライン一本当たり全部で$R*$(p_iの個数) この評価検査があり，評価方法は a_i 毎に，図中 ◯ のついたポイント (条件に合致したポイント) の $\lfloor \log p_i \rceil$ を加算することで評価値を得ることにある．このような篩ライン (評価ポイントはRポイント) を全部でH種類 (篩ライン毎に相違なる a_0〜a_{R-1} を用いる) 処理することで，篩処理が完了する．即ち，処理対象となる篩ラインの本数はHである．

上記のRポイントの評価ポイントを，極素朴に並列/パイプライン実装することを考えると，ライン方向に評価ポイントをR並列化を行なうことをまず考える．次に，カラム方向には，1ライン/クロック動作，p_i の個数分段のパイプライン化という構成がまず得られる．上記構成が，実際には実現不可能な規模となることは言うまでもない．

図4.1を出発点として，論理規模を削減するためにTWIRLデバイスに盛り込まれたアイデアを説明する．ライン方向の論理規模を削減するために，篩ライン上のR点全点を一度に，ではなく，そのうちのs点のみの並列評価とし，時間軸でこれをシリアルに処理している．これにより，ライン方向の論理規模は単純にsとなるが，同時に処理速度もR/sに減少する．更に，各評価ポイントに $\lfloor \log p_i \rceil$ を供給するために，シフトレジスタを用いたデリバリラインを用いることで，データ供給用のワイヤを大幅に削減している．

次に，カラム方向の論理規模削減については，各評価ポイントにおいて，評価条件が成立する確率 ($\lfloor \log p_i \rfloor$ を加算する確率) が，素数の大きさに応じて異なり，特に大きい素数 pi に対応した評価ポイントでは非常に稀になることを利用する．大多数の p_i に対しては，一本のデリバリラインが $\lfloor \log p_i \rfloor$ を供給する頻度が低いことから，複数の p_i 値でデリバリラインを共有することで，デリバリライン数を削減する．

上記説明から得られる TWIRL の構造を図 4.2 に示す．Tiny プログレッションについては素数の数=デリバリライン数であるが，これより大きい素数については，合計 2601 本のデリバリライン数に削減している．この数は，図 4.1 に示した素朴な実装に要するパイプライン段数 ($= B$ 以下の素数の数) と比較すると極端に小さく，大幅な論理規模削減となっている．原著の選択したパラメータ選択時の 1024 ビット合成数を対象とした篩処理時に，smallish については，501/43007，largish については，$2100/(1.6 \cdot 10^8)$[1]となっている．

また，上記構成は処理時間の短縮にも貢献しており，パイプライン段数についても，わずか 2655 段となっている．上記により，各評価ポイントで必要な演算を行なうための，並列，パイプライン構造は，かなりの小論理規模化，処理段数削減に成功しており，この点は，篩処理ハードウェア現実化のための大きな改良点であると評価できる．

但し，各デリバリラインに素数 π 毎の評価値を適切に出力する仕組み (エミッタ，プロセッサ) は複雑となっており，評価ポイント演算構成を単純化するためのトレードオフになっていると言える．

4.2 個々の構成要素

前の章や，これからのハードウェア評価の章においても，Largish, Smallish, Tiny それぞれの計算ステーションの構成の説明があるので，中枠の説明は大きく省き，局所的な部品に関する説明を行なう．

メモリシステム (DRAM/SRAM) DRAM 及び SRAM は，読みだし書き込みの機能を持った大型のメモリシステムであって，Largish の計算ステーションの重要な構成要素である．サイズは，担当するプログレッションにより異なっているが，プ

[1]素数定理より x 以下の素数の数は $\pi(x) \approx x/\log x$ で近似できる．有理数篩の Largish プログレッションは $5.2 \cdot 10^5 < p_i < 3.5 \cdot 10^9$ であり，$\pi(3.5 \cdot 10^9) - \pi(5.2 \cdot 10^5) \approx 1.6 \cdot 10^8$ となる．

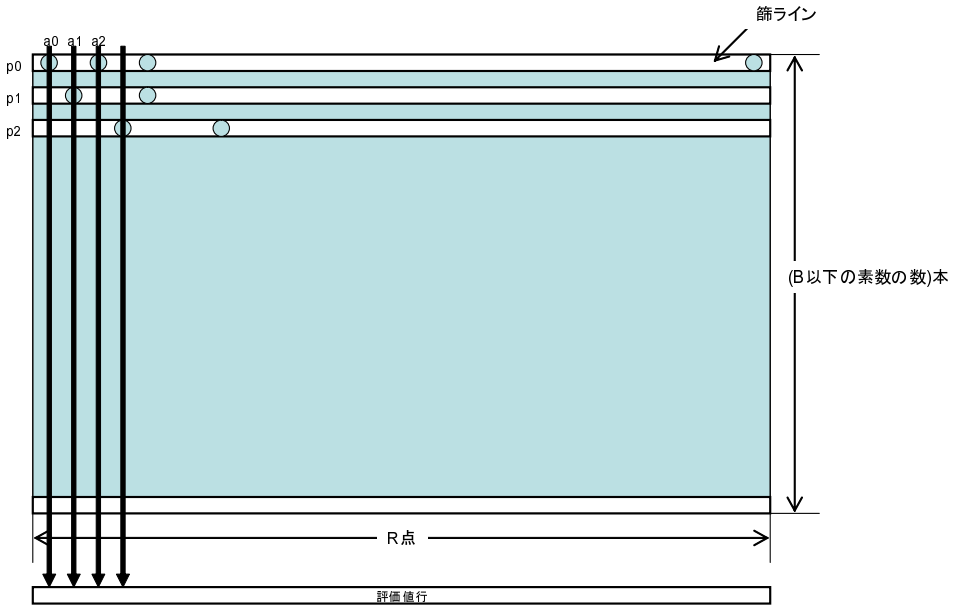


図 4.1: 篩処理の全体構成 (単純な並列/パイプライン実装を想定した場合)

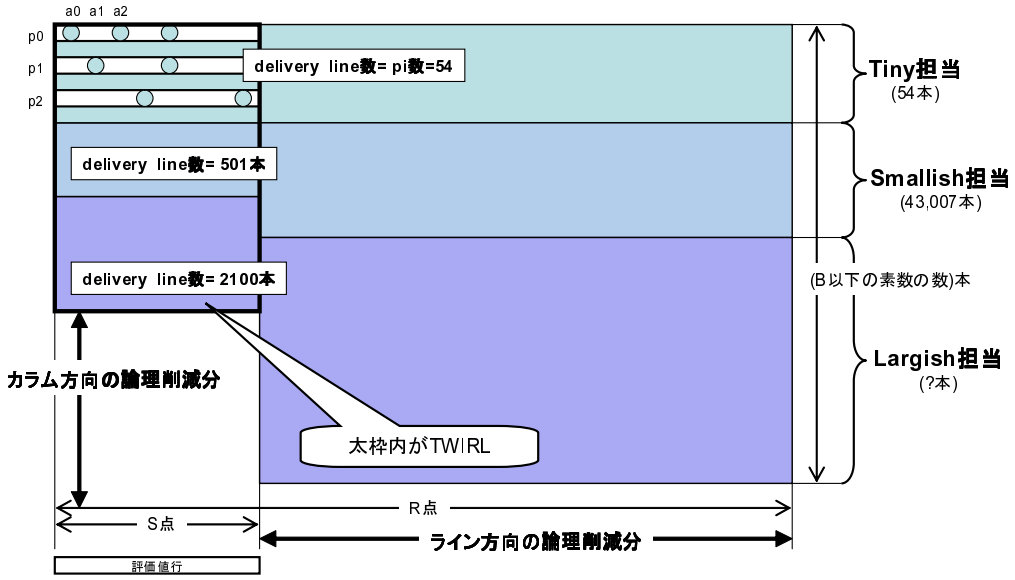


図 4.2: TWIRL の構成

ログレッションに相当する素数が大きいものは，多数のプログレッションを担当し，素数が小さいと，少数のプログレッションを担当する．これにより，プロセッサへの出力 (プログレ放出) の頻度が均等化されることを期待するものである．

DRAM はその性質上，プロセッサが求める動作速度，1 GHz での読み書きを達成できない．そこで，SRAM をキャッシュに見立てた構造を提案している．

読みだしは TWIRL の構成方法から，ひじょうに簡単な動作であり，定義される領域を 1 クロックあたり 1 番地の定速で読み出す番地を動き，そこを読みとることである．書き込みはランダムアクセスの要求に近いが，その格納番地が，その時刻における格納の状態に応じた動的な決定を行なう．これには失敗も伴う．

確かに，SRAM も DRAM も現在の実装の LSI 技術では 1 GHz で動作することはできない．しかし，以上の要求のみを満たすのであればバッファリングなどにより，見かけ上 1 GHz で動作しているのと同等なメモリシステムの構築は可能である．しかし，これと [49] で主張されている SRAM の使い方とが技術的に繋がらないことを理由の一つとし，かつ実現性の観点からは重要性は低いこともあるため，本評価では，詳細な評価は行なわない．これらの動作を示したものを図 4.3 に示す．

プロセッサ　プロセッサの主なる動作は簡単である．受けとったプログレトリプルデータから (1) 簡単な剰余演算を行って格納するのと，(2) トリプル発行データをバッファに渡すこと，のみである．

しかし，第 3.7.7 節にて，$\langle\!\langle 13530 \rangle\!\rangle_{\mathrm{R}}$ サイクル分のトリプル発行データを記録する必要があることを示している．これを日記と読んでいる．このデータは，篩装置をパスしたものが，実際に smooth かどうかを検査するための (cofactor factoring) の手間を大きく軽減するための，重要なレポートである．実際にプロセッサは，シストリックアレイの下流からあがってくる別のパイプライン (ラインチャンネル，と呼ぶ) を常に監視し，それが自分の放出したトリプル発行データのものであれば，それを認識し日記から該当トリプル発行データを発見し，もう一つの下流向けのパイプラインである因子チャンネルに流す．以上の動作を示したものが図 4.4 である．

バッファ　バッファはプロセッサ群から受けとるいくつかのトリプル発行データを放出時刻の τ 要素で簡単なソートをかけながら，うまくタイミングを調整し，厳密なタイミングで デリバリペア を生成し，放出するものである．

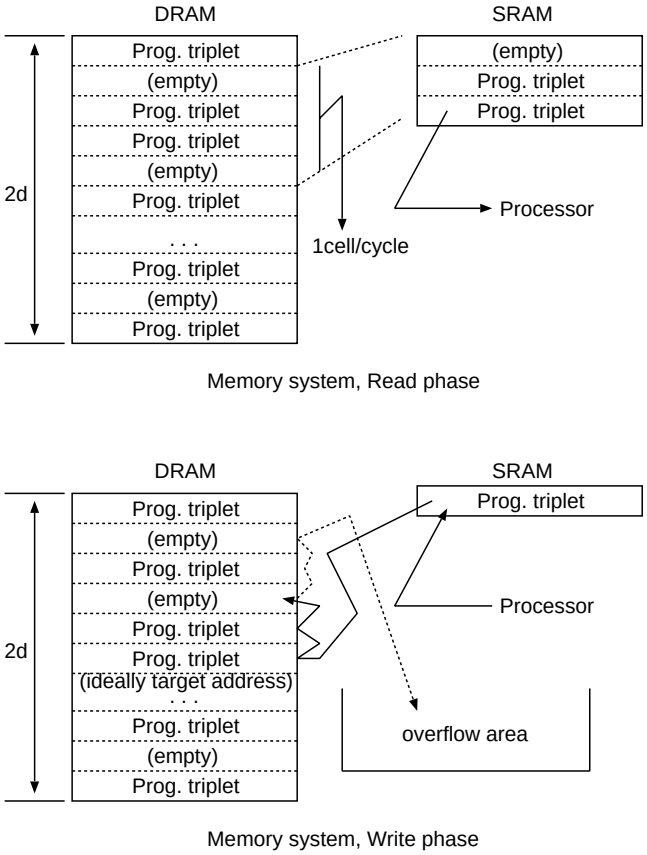


図 4.3: DRAM/SRAM 動作を説明するブロック図

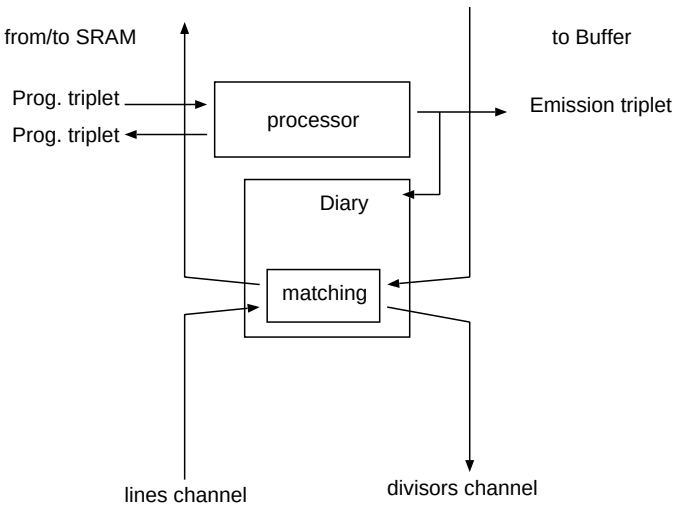


図 4.4: プロセッサの動作を説明するブロック図

バッファは二次元的な構成であり，例えば理解として縦軸で τ ソートを行ない，近い将来放出されるべき トリプル発行データ は下に沈むと考える．局所的な渋滞を避けるために横方向にはランダムシャッフルが適用される．

バッファについての記述は，他の部品に比較しても簡単であり，我々の評価では設計に十分な情報が記載されていないと考えた．よって，詳細は不明なままであるが，概要のみをブロック図，図 4.5 を使いながら説明する．

各サイクル毎に，

1. 最下部 2 列にたまったトリプル発行データを τ マッチングし，時刻が合致したものは，放出する．

2. (最下部 2 列にたまったトリプル発行データのうち?) 時刻がすでに過ぎてしまったものは例外処理として，抹消する．

3. 空きができた場合には，下方向へ各番地に保存されているエントリをスライドする．

4. プロセッサから送信されたトリプル発行データを受信し，適切な空セルに保存する．

5. 縦方向にバブルソート (全部が並びきるまでソートするわけではないはずであり，おそらくバブルソートの 1 ステップ，例えば偶数-奇数番地で大小比較を行ない，結果に応じて swap するなど) をかける．

6. 横方向にランダムシャッフルする．

以上の処理を行なうよう記述があるが，各々の動作の詳細については不明である．

特に放出については，一つのバッファに接続するデリバリラインの数についても言及されておらず，またそのメカニズム (例えば以下のような処理を達成するもの) を考えるとかなり複雑である．

1. 複数のトリプル発行データが τ マッチングをパスしてくることを想定している．

2. τ マッチングの対象のセルは比較的多数，例えば 8 セル分などであり，8 セルのどこからでも来る可能性がある．

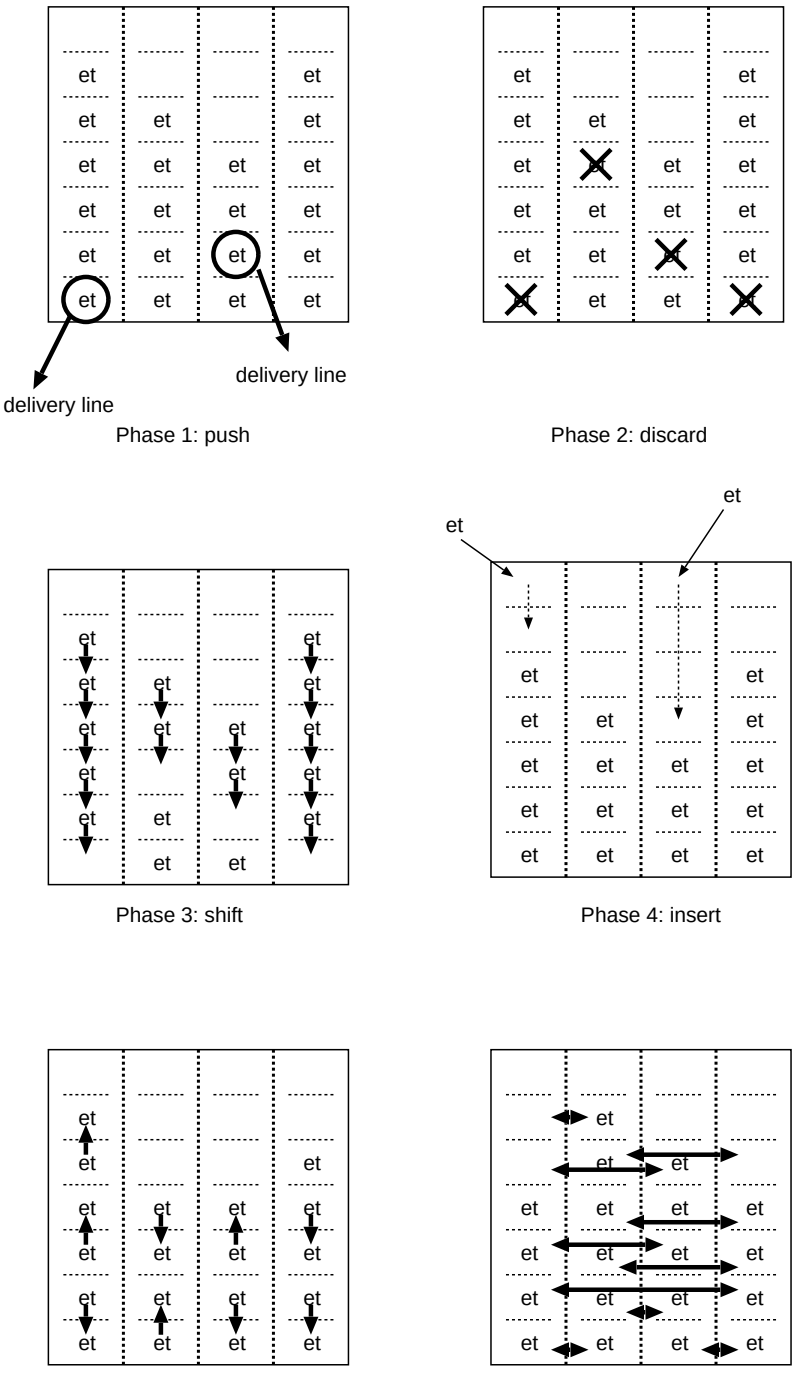


図 4.5: バッファの動作を説明するブロック図

3. デリバリラインはτマッチングのパス数をみながらパスしたトリプル発行データを (デリバリペアに変換して) デリバリラインに割り当てる.

4.3 変数表

表 4.1: TWIRL の変数表

和名	記号	有理数篩	代数篩
篩問題数	H	$2.7 \cdot 10^8$	$2.7 \cdot 10^8$
篩線幅	R	$1.1 \cdot 10^{15}$	$1.1 \cdot 10^{15}$
滑らか境界	B	$3.5 \cdot 10^9$	$2.6 \cdot 10^{10}$
閾 (しきい) 値	T	2^{10}	2^{10}
対数の底	h	> 2	> 2
並列度	s	4096	32768
ウェハ内篩装置数	–	4	1
LP 素数サイズ	p_i	$> 5.2 \cdot 10^5$	$4.2 \cdot 10^6$
LP 内 DRAM バンク数	–	8490	59400
バンク内プログレ数	d	$32 \leq d < 2.2 \cdot 10^5$	$32 \leq d < 2.0 \cdot 10^5$

表 4.2: 派生した変数一覧

素数	$\lceil \log_2 B \rceil$	32	35
配達線 ID	$\lceil \log_2 s \rceil$	12	15
クロック時刻	$\lceil \log_2 (R/s_A) \rceil$	35	
DRAM 希釈度数	–	2	2
DRAM オーバーフロー閾値	–	64	64
丸め素数スコア	$\lceil \log_h B \rceil$	20	22
丸め素数スコアサイズ	$\lceil \log_2 (\log_h B) \rceil$	5	5

第5章　TWIRLデバイスの面積評価

本章においては，原著 4.1 及び B.1(本稿では，第 3.5.1 節, 第 3.8.1 節) に記載された TWIRL デバイスのトランジスタ数及びシリコンウェハ面積見積もり数値に矛盾がないかを検証し，[49] の主張する面積/コストに関する妥当性を評価する．

5.1 面積から見た全体構成の把握

30 cm シリコンウェハの面積は 70,650 mm^2 である．原著では 0.13 μm プロセスでのロジック部 1 トランジスタ当たりのウェハ面積を 2.8 $\mu\mathrm{m}^2$，DRAM 部 1 ビット当たりのウェハ面積を 0.2 $\mu\mathrm{m}^2$ としているが，0.13 μm 標準 CMOS プロセスでの単位面積当たりのゲート規模は，1 mm^2 当たり約 200k ゲート程度と言われており，1 ゲート=4 トランジスタ換算では，1 トランジスタ当たりのウェハ面積は約 1.5 $\mu\mathrm{m}^2$ となる．原著では DRAM プロセスのウェハを用いるとしており厳密な比較は難しいが，オーダとしては合致しており，倍程度大きめな見積もり値を用いていると判断する．以降ではこれらの値を用いて評価を行なう．

表 5.1 に TWIRL デバイスの機能ブロック別面積を示す．この数値は，原著 4.1(第 3.5.1 節) 記載の TWIRL デバイス面積に，デバイス内各機能ブロックの面積比率を乗じて算出したものである．また，第 3.8.1 節記載の 1 トランジスタ当たりのウェハ面積，及び，各ステーション構成要素のトランジスタ数から，同構成要素の面積は，各々表 5.2 に示した数値として得られる．

表 5.2 の数値から，有理数篩の Largish ステーションのデリバリラインの占める面積を積み上げると，

$$0.0015\,\mathrm{mm}^2 \times 1024(=s/4) \times 2100(\text{デリバリライン数}) = 3,226\,\mathrm{mm}^2$$

となり，同様に特定用途プロセッサの占める面積は，

$$0.27\,\mathrm{mm}^2 \times 8490(\text{個数}) = 2,292\,\mathrm{mm}^2$$

であることから，合計で 5,518 mm^2 となる．この数値は，表 5.1 の Largish ステーションロジック面積 6,224 mm^2 と約 1 割の 706 mm^2 の開きがある．

同様に，代数篩の Largish ステーションのデリバリラインの占める面積を積み上げると，

$$0.0015\,\mathrm{mm}^2 \times 32 \times 14,900(\text{デリバリライン数}) = 715\,\mathrm{mm}^2$$

表 5.1: 機能ブロック別面積

機能名称	面積 (mm^2)	備考
有理数 TWIRL デバイス	15,960	
Largish ステーション	12,129	
logic	6,224	
DRAM	5,905	29.5G ビット (3.7 GByte) 分
Smallish ステーション	3,352	
Tiny ステーション	479	
代数 TWIRL デバイス	65,900	
Largish ステーション	61,946	
logic	18,452	
DRAM	43,494	217.5G ビット (27.2GByte) 分
Smallish ステーション	3,954	
Tiny ステーション	?	4.1 には面積比記述なし

表 5.2: 構成要素別面積

機能名称	一個あたりの面積 (mm^2)	備考
delivery cell(Largish 用)	0.0015	
delivery cell(Smallish,Tiny 用)	0.0034	
processor(Largish 用)	0.27	キャッシュ14k ビット分, バッファ含む
emitter(smaillish 用)	0.0057	funnel 含む
emitter(Tiny 用)	0.0015	

となり，特定用途プロセッサの占める面積は,

$$0.27\,\mathrm{mm}^2 \times 59400(\text{個数}) = 16,038\,\mathrm{mm}^2$$

であることから，合計で 16,753 mm^2 となる．この数値は，表 5.1 の Largish ステーションロジック面積 18,452 mm^2 と約 1 割の 1699 mm^2 の開きがある．与えられた情報のみでは，この原因の推定は困難であり，ここでは事実を指摘するにとどめる．尚，有理数篩，代数篩の DRAM 面積比は，約 7.4 倍であり，滑らか境界 (=DRAM に保管すべき素数 p_i の数) B_R，B_A の比と概ね一致する．

5.2 一般的なLSIとの面積比較

次に，TWIRLデバイスの要する面積と現実に量産されているLSIの面積との比較を行なう．現実に量産されている汎用プロセッサとしてインテル社製品のダイサイズを調査した．その結果を表5.3に示す．

一般的に，LSI製品のコストはシリコン面積と比例する関係にあり，リーズナブルな量産を行うためには，おのずと最大面積が定まってくる．シリコンウェハそのものを製造する際のシリコンの格子欠陥，LSI製造時のトランジスタ生成失敗，配線失敗などが生じる確率が零ではないことから，一枚の半導体ウェハ上に製造した複数のLSIには，かなりの確率で不良品が含まれる．同一の半導体製造設備を用いた場合，半導体ウェハ面積当たりの製造不良率は同一であるので，半導体ウェハから切り出すLSI製品の面積が小さい程，生産歩留まり (=良品確率) が大きいことになる．

表5.3を参考にすると，半導体プロセスの世代が進む際に，PC用プロセッサは$100\,\mathrm{mm}^2$超程度のサイズをキープし，サーバ用プロセッサは$400\,\mathrm{mm}^2$程度のサイズをキープしていることが分かる．PC用プロセッサ，サーバ用プロセッサ間の面積の違いは，許容できる製造コストの差とみなすことができる．利用する半導体プロセスに依存せずにリーズナブルな量産を行うためには，$50\,\mathrm{mm}^2$程度の面積が妥当との説もあり，上記プロセッサの面積も，LSI製品全体から見ると，かなり大きめな数値となっている．

ここで，TWIRLデバイスの面積を見てみると，各ステーションを構成する要素の面積は十分に微小であるものの，構成要素を組み合わせたステーションの面積は巨大なものであり，例えば，最も小さい，有理数篩用のTinyステーションであっても，サーバ用プロセッサの面積を超えている．現実的なLSI製品の面積を考えた場合，TWIRLデバイスの要する面積がどれだけかけ離れた数値であるかは以上の比較から理解可能である．原著の「30 cm ウェハに実装可能な論理規模であるので，実現可能性が高い」という主張に対しては，利用可能な半導体技術の観点からは，**「30 cm ウェハにやっと実装可能な論理規模では，実現可能性は極めて低い」**という異議申し立てが必要である．

以下，付加的な議論であるが，半導体プロセスは一世代進む毎に，同一面積に搭載可能な論理量は倍になる．0.13 μm プロセスで65,900 mm^2を占める代数篩TWIRL

表 5.3: 一般的な LSI 面積・トランジスタ数の参考値 (インテル社製品)

製品シリーズ名称	プロセスルール	ダイサイズ (mm^2)	トランジスタ数
Intel®Pentium®4	0.13 μm プロセス	131	5500 万
	90nm プロセス	112	1 億 2500 万
Intel®Itanium®-2	0.18 μm プロセス	421	2 億 2100 万
	0.13 μm プロセス	374	4 億 1000 万

デバイスを，サーバ用プロセッサ並みの 400 mm^2 程度の面積で実現するには，面積を約 1/165 に縮小する必要があり，半導体プロセスで言うと，7〜8 世代分の技術開発が必要になるということを付け加えておく．

5.3 大規模論理回路サポート向け半導体技術について

回路規模の妥当性の観点から，原著の問題点を大雑把に述べると，

- ウェハ一枚に渡る規模の回路を想定していること．
- 大規模回路を 1 GHz という高速で動作させていること．
- ウェハ間を 1 GHz クロックで同期動作させていること．

が挙げられる．

このうち，後者 2 つについては，別章で扱うこととし，ここでは,「ウェハ一枚に渡る規模の回路」の妥当性について論じる．先ず，TWIRL デバイスの唱える処理自体は論理的であると考えられる．即ち，大規模並列処理でデータ転送時間が問題になるのは周知の事実であり，デバイス (TWIRL デバイス) 間のデータ転送時間を低減させるために，やり取りするデータ量が小さいところでシステム (TWIRL クラスタ) を分割するというアイディア自体に異論はない．だが，残念ながら，システムを構成する個々のデバイスが，半導体技術の観点から巨大過ぎて現実的でないと言わざるを得ない．実際のデバイスとして実現するには，超大規模 LSI の実現をサポートする半導体技術が多数必要になると思われる．言い換えると，主張する面積，速度を，現在利用可能な技術で実現するのは極めて困難だが，半導体プロセス

技術の今後の進展と，サポート技術の開発がなされれば，必要な機能を実現することは不可能ではないと考える．

半導体技術の現状としては，ウェハ一枚に渡って無故障の半導体デバイスを製造することは不可能であり，おそらく今後ともこの状況に変化はないと考える．現状の半導体製造装置の故障検出技術は，良品と不良品を峻別するための技術である．100%動作するものが良品であり，不良部分が一箇所でもあれば不良品と判断する．

TWIRL のような大規模論理回路をサポートする技術としては，製造結果の良否だけではなく，製造不良が生じた際に，どこが不良なのかを詳細に検知する技術が必要となる．大規模回路を動かす方法=耐故障設計技術と言ってもよく，同技術は，故障位置特定技術と故障位置回避技術とに大別できる．このうち，前者の技術を実現することの方がより困難ではないかと考える．

現在一般的な故障検出技術としては，標準技術として JTAG/IEEE1149.1 があり，その他半導体製造メーカ独自の故障診断技術を用意している場合もある．技術的には，本来の機能を実現する論理回路に加え，テスト用の論理回路を追加することで，本来の機能を実現する論理回路の正常動作を確認する．テスト用論理回路の追加によって，LSI 装置全体としての論理規模増加は不可避であり，数十%の論理規模増加になるケースもある．

又，近年は電磁波解析という手法も開発されている．正常動作するデバイスと，内部回路の故障により異常動作するデバイスとでは，故障デバイスは，その故障部が動作しないために，同一の動作をさせた時の回路内部の動作が異なっている．従って，動作に伴い周りに漏れる電磁波のパターンも異なっている．この電磁波パターンの違いから故障を検知するものである．現状適用可能なのは，LSI パッケージ内の多層配線の欠陥検出レベル (数 μm) であり，すぐに適用可能な技術ではないが，大規模回路の故障位置を大まかに検知する技術として将来的な可能性はある．

故障を回避して回路を正常動作させる技術としては，第 3.7.9 節にもあるとおり，デリバリラインのバイパス回路作成が考えられる．しかし，スペア回路の搭載は，面積の増大，ルーティングによる遅延量の増大/動作速度低下に繋がる．又，バイパス回路を動作させるためには，経路選択を指定する回路が正確に動作する必要があり，同回路の故障回避の優先度は，デリバリラインのスペア回路の動作よりも高くなる．優先度の高い回路の故障回避策としては，例えば，多数決論理による冗長論理構成が考えられるが，回路規模は増大する方向である．

以上の議論からは，原著の TWIRL デバイスの機能を実現するには，大規模論理回路を正常に動作させるためのサポートロジック用に更に多くの面積が必要であると言うことが分かる．このことから，30 cm ウェハ一枚に搭載可能とされた代数篩の実現は困難であり，**「原著の主張する 3 枚のウェハからなるクラスタ構成を実現するのは困難」**と判断する．

5.4 コストの妥当性について

原著は，TWIRL デバイスの実現費用が$ 30M であると主張しているが，以下，その積み上げを再整理する．30 cm ウェハ一枚のコストは$ 5,000 である．1TWIRL クラスタに 3 枚のウェハが必要であり，194 個の独立した TWIRL クラスタを構成するには，582 枚のウェハが必要である．従い，582 枚分のウェハコストは総額$ 2.9M となる．この金額に，パッケージ，電源，冷却装置などの付随費用をマージンとして見込み，総額$ 10M の製造コストとしている．これに，$ 20M の開発費 (設計，検証，マスク作成) を加え，$ 30M という数字が出ている．原著者も述べているが極めてラフな見積もりであることは確かである．明らかにできるのは，ウェハ一枚$ 5,000 という数字が，何の加工も施されていないシリコンウェハのコストとして妥当であるという点のみである．

先の議論であった通り，582 枚のウェハに原著の主張する機能を盛り込むのは困難であり，大規模論理回路を正常動作させるためのサポートロジックを追加するとすれば，それは回路面積，即ち要するウェハ枚数の増加要因となる．又，現状の半導体製造技術が，限定された面積の LSI 良品を製造するための技術であることから，大規模論理回路サポート向けの製造技術自体の開発が必要になる可能性がある．例えば，製造時の良品確率を向上させる手法として，以下のようなものが考えられる．

LSI の製造には，空気中における浮遊微粒子を厳重な管理下におくクリーンルームが用いられるが，半導体製造プロセスが微細化すると，それまでの製造プロセスでは問題とならなかったサイズの浮遊微粒子が新たに問題化する．そのために，クリーンルームの「クリーン度」は製造プロセスの世代が進む程高くなっている．従って，世代遅れの半導体製造プロセスの製造装置を最新製造プロセス用のクリーンルームに設置することで，半導体製品の故障率が下がることが期待される．世代遅れの半導体製造プロセスを用いて LSI を製造することで，面積当たりの処理能力は

低下し，デバイス全体として，より多くのシリコンウェハを必要とすることになるが，製造良品確率が向上することで製造コストを削減できる可能性がある．が，上記手段を採るための技術的課題は未知であり，その開発コストを考慮すると，製造コストに対するインパクトについては言及困難である．

原著の主張する TWIRL デバイスは，従来技術を組み合わせて実現可能なデバイスではなく，特に，大規模論理回路サポート向けの製造技術の開発が重要となる．本技術開発の課題及び費用を見積もることが困難であることから，全体としての初期費用見積もりは困難であると判断する．

第6章　要素部品に対する回路評価

篩処理を 1 年で終える TWIRL 全体 (図 6.1) は，194 個の TWIRL クラスタが並列に動作する．

TWIRL クラスタ一つ (図 6.2) は，8 個の有理数 TWIRL と 1 個の代数 TWIRL から成る．有理数 TWIRL と代数 TWIRL は，サイズや詳細アーキテクチャは異なるが，共通化して考えることができる部分も多いため，本評価では特に有理数 TWIRL に着目する．

有理数 TWIRL 一つ (図 6.3) は，256 未満の 54 個の素数を扱う Tiny station(図 6.4)，256 超 $5.2 \cdot 10^5$ 未満の 43,007 個の素数を扱う Smallish station(図 6.5)，$5.2 \cdot 10^5$ 超 $B_R = 3.5 \cdot 10^9$ 未満の 167,236,272 個の素数を扱う Largish station(図 6.6) から成る．Tiny station では 54 本，Smallish station では 501 本，Largish station では 2,100 本のデリバリラインが存在する．1 本のデリバリラインは 4,096 個のデリバリセルから成る．2 次元的なデリバリセルの並びが，シストリックアレイ構造として機能する．最終のデリバリラインの 4,096 個の出力 $g(a)$ それぞれに対して，閾値 $T = 1,024$ と比較し，それを超えるものが候補として代数 TWIRL に引き継がれる．

Largish station(図 6.6) は，167,236,272 個の素数を扱う．そのような膨大な個数の各素数について専用回路を個別に持つのは現実的ではないため，DRAM-cache(高機能 SRAM)-processor の組を備えて DRAM に膨大な個数の素数を格納することで，比較的現実的な回路として実現できるようになっている．有理数 TWIRL 一つに付き，DRAM-cache-processor の組は 8,490 組存在する．

本評価では，Tiny station，Smallish station，Largish station のうち，最もクリティカルと考えられる Largish station に着目する．そして，Largish station の構成要素のうち，特に，DRAM-chache-processor の個数とデリバリラインの本数に対するプログレ放出の頻度，および DRAM の容量，およびデリバリセルの動作周波数・回路規模に着目し，原著の妥当性を検証する．

6.1 プロセッサ，バッファは評価できない

なお，cache-processor とバッファの回路構造は，次の不明点により詳細なサイズ・アーキテクチャを原著から読み取ることが困難である．また，回路規模や電力消費の観点からも特別に大きな部分構成要素というわけではない．そこで，我々がわからない部分を独自に考え，原著からは読みとれない仕様とは異なる仕様で見積り，

その結果を原著と照らし合わせて評価したとしても，それ自身，現在の TWIRL の現実性評価，という観点からはあまり意味がない．よって，これらの構成要素については詳細に評価しないこととする．(注意：評価や実装することが不可能と言っているわけではない．)

cache-processor の組の主な不明点

- バッファに予め渡すことのできるプログレ放出の放出時点 τ と，現時点との許容時間差.
- DRAM-cache 間のバス接続ビット数やアクセスサイクル・タイミング.
- processor から送られる triplet を格納すべき DRAM の領域がなかなか見つからない場合の例外処理．(cache はその間にも，DRAM からプログレ放出となるべき triplet を次々と読み出す必要がある上，それに応じて processor から DRAM に格納すべき triplet が次々と書き込まれる．)

特に 3 点目については，ある程度の記載はある．すなわち 64 番地遡ったところで空きがなければ例外処理をする，とある．しかしながら，τ は 12 ビットの領域しかもたないことになっており，例外処理として，しばらく，具体的には 2^{12} クロック以上，の間不正な動きをしたとするとその後も対応するプログレッションは不正なプログレ放出を篩問題が終るまで振舞い続けることになる．これらによる装置の誤動作が篩問題の処理として許容できるかどうかが計上されていない.

バッファの組の主な不明点

- バッファ一つに接続されている上流の processor および下流のデリバリラインのそれぞれの個数.
- processor からプログレッションが届いてから実際に時点 τ で放出するまで，バッファが同プログレ放出を保持しておく (予め processor から受けとれる) 許容時間.
- バッファ内で行われる，バブルソート処理の詳細，およびランダムシャッフル処理の詳細.

- 複数のプログレ放出がバッファから取り出されることを前提に複数のデリバリラインへの放出を考慮した，賢いセレクタの仕様.

- processor からバッファ容量を超えるプログレ放出が (複数サイクルにまたがって) 届いた時の処理や，プログレ放出を放出すべき時点 τ を超えてしまった場合の該プログレ放出の扱い等，例外処理.

6.2 プログレ放出頻度

DRAM-cache-processor の組は，有理数 TWIRL 一つに付き 8,490 個存在し，それぞれが $32 \leq d < 2.2 \cdot 10^5$ 個の素数を扱い，全体では $5.2 \cdot 10^5$ 超 $B_R = 3.5 \cdot 10^9$ 未満の 167,236,272 個の素数を扱う.

一素数 p に関するプログレ放出の頻度は $\frac{4,096}{p}$(/cycle) であるから，Largish station 全てのプログレ放出の頻度は

$$\sum_{\substack{5.2\cdot10^5<p<3.5\cdot10^9 \\ \text{prime } p}} \frac{4,096}{p} \quad = \quad 2099.33103634 \quad (/\text{cycle})$$

となる．一方，Largish station に存在するデリバリラインは 2,100 本であるから，これらデリバリラインが常にほぼ 100%使用されるようにバッファが機能することが理想的である．一つの processor が放出するプログレ放出の平均頻度は，$\frac{2099.33103634}{8,490} = 0.24727103 = \frac{1}{4.04414542}$(/cycle) である．プログレ放出の放出は互いに異なる膨大な個数の素数に基づいているため，各 processor は 4.04414542 サイクルに 1 回の確率でランダムの様相でプログレ放出を放出する．よって，任意のサイクルにおいて，8,490 個存在する processor から総計 n 個以上のプログレ放出が放出される確率 $P(n)$ は

$$P(n) = \sum_{i=n}^{8,490} {}_{8,490}C_i \cdot 0.24727103^i \cdot 0.24727103^i \cdot (1 - 0.24727103)^{8,490-i}$$

となる．n と $P(n)$ の関係をグラフ化したものを図 6.7 および図 6.8 に示す．図 6.8 は，図 6.7 の縦軸 $P(n)$ を対数表示したものである.

図 6.7,6.8 より，プログレ放出の放出個数がデリバリラインの本数 2,100 よりも例えば 50 多い $n = 2,150$ は，およそ 10 サイクルに 1 回発生する．よって，およそ 100

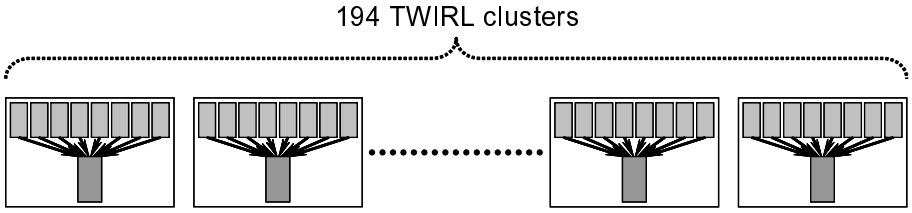


図 6.1: 処理時間 1 年の TWIRL 最上位

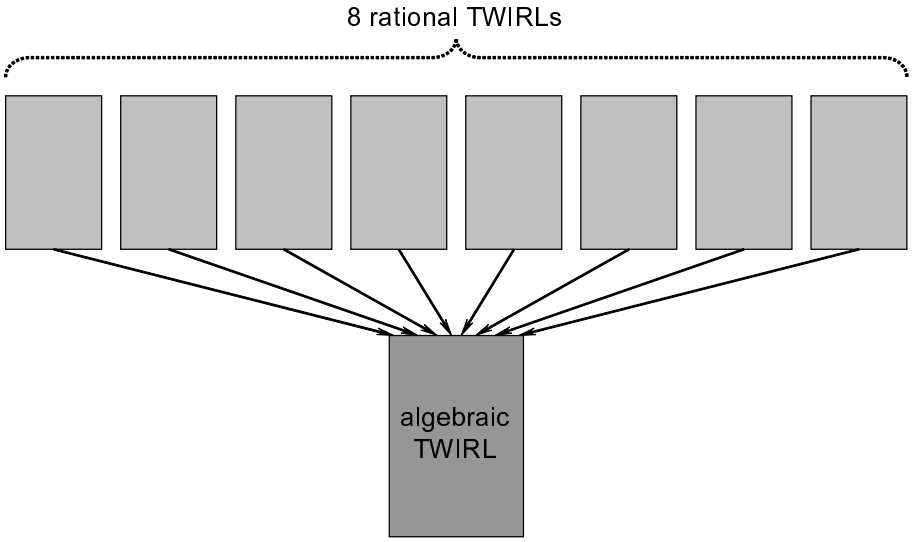


図 6.2: TWIRL クラスタ

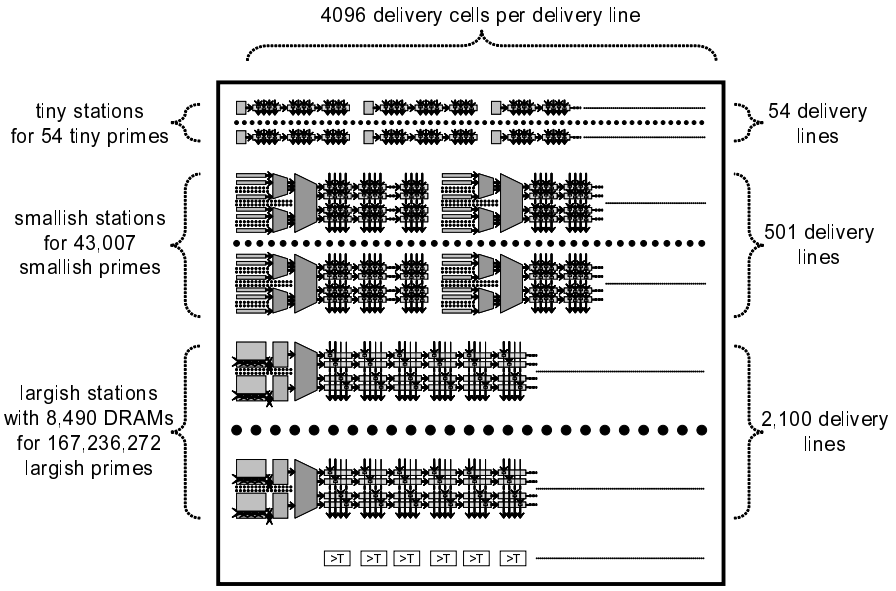


図 6.3: 有理数 TWIRL

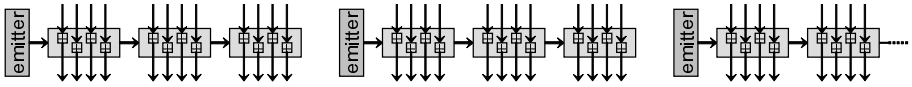


図 6.4: Tiny station

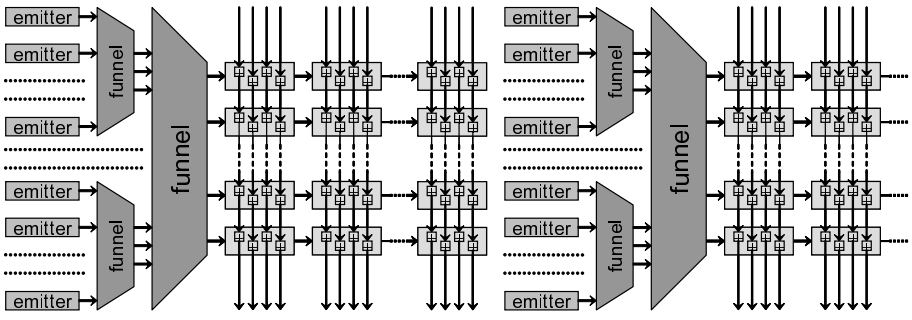


図 6.5: Smallish station

サイクルに 1 回の確率でデリバリラインの本数よりも 50 多いプログレ放出が 2 サイクル連続で発生するし，およそ 1,000 サイクルに 1 回の確率でデリバリラインの本数よりも 50 多いプログレ放出が 3 サイクル連続で発生することとなる．このようにデリバリラインの本数よりも多いプログレ放出が単サイクルだけでなく時間的に連続して発生した場合でも，それらプログレ放出をデリバリラインへ確実に送れるように，バッファは十分な容量を持ち且つ高性能な論理が必要である．言い換えれば，バッファが備えるべき容量を見極める必要があるし，1 GHz 動作する高性能な論理を実装する必要がある．実際は Largish station のデリバリラインは $r = 4$ でインタリーブされているため，それによってバッファの制約がより厳しくなることは述べるまでもない．

あるいは，プログレ放出のうち全てをデリバリラインへ送る必要はないのかもしれない．この場合は，バッファは比較的容易に実装できると考えられるが，無効となるプログレ放出が頻発しても篩問題 (ひいては素因数分解問題) が期待通りに解けるのか疑わしい．篩問題としてはどの程度のプログレ放出を無効としてよいのか，また実際に実装した場合にどの程度発生するのかを，各々数学的に導出して照らし合わせて検証する必要がある．

6.3 DRAM 容量

5.1 節に示すように，原著に示されたウェハの面積から逆算した有理数 TWIRL 一つの DRAM 容量は 29.5G ビットと導出される．本節ではこれを検証する．

DRAM に格納されるデータの最小単位 triplet の内容は (p_i, l_i, τ_i) と，その内容が有効かどうかを示すフラグ用の 1 ビットである．l_i のビット数は $\log_2 s = \log_2 4,096 = 12$ ビット であり，τ_i のビット数は $\log_2 2,048 = 11$ ビット である (原著 3.2 段落 (第 3.4.2 節)Notes. 部分より)．

8,490 組存在する DRAM-cache-processor の組に対して，p_i を格納するためのビット数は可変である．インデックス j $(1 \leq j \leq 8,490)$ の DRAM-cache-processor の組が扱う素数の個数を $n^{(j)}$，その各素数を $p^{(j)} = \{p_1^{(j)}, p_2^{(j)}, \cdots, p_{n^{(j)}}^{(j)}\}$ とおく．このとき，p_i を格納するためのビット数は $\log_2 \max\{p^{(j)}\}$ となる．また，$\sum_{j=1}^{8,490} n^{(j)} = 167,236,272$ である．

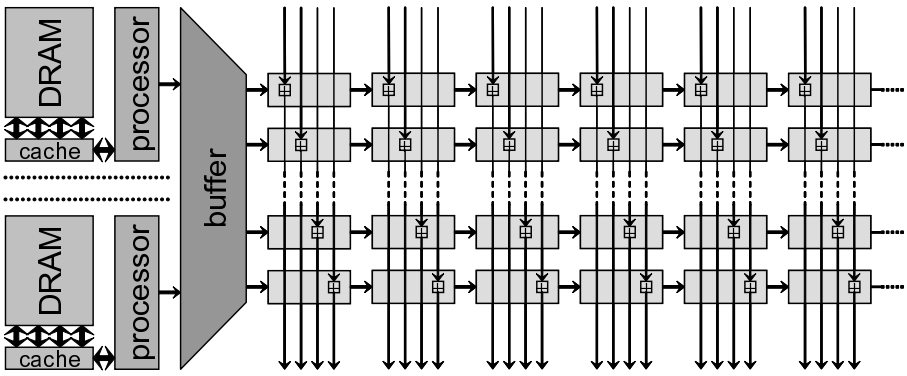


図 6.6: Largish station

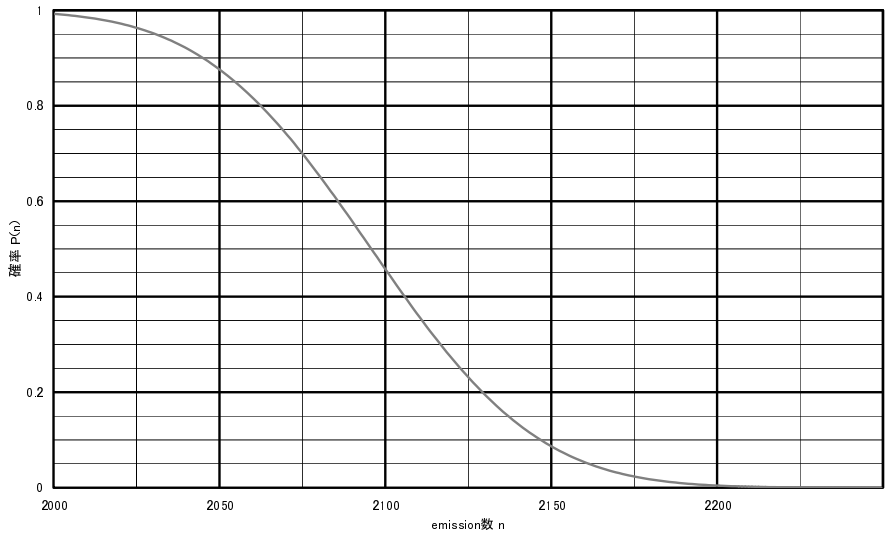


図 6.7: n 個以上のプログレ放出が放出される確率 $P(n)$

さらに，有理数 TWIRL 一つが Largish station で扱う $5.2 \cdot 10^5$ 超 $B_R = 3.5 \cdot 10^9$ 未満の 167,236,272 個の素数を，任意の j に対して $\frac{1}{p_1^{(j)}} + \frac{1}{p_2^{(j)}} + \cdots + \frac{1}{p_{n^{(j)}}^{(j)}} \approx \frac{1}{4.04414542}$ となるように，小さいほうから順に 8,490 組に分割したとき，$\sum_{j=1}^{8,490} n^{(j)} \cdot \log_2 \max\{p^{(j)}\} = 5,134,034,180$ であることを計算機を用いて計算することで確認した．

有理数 TWIRL 一つに付き必要な DRAM 容量を $D_{\min}(\text{bit})$ とおくと，以上より，最低でも

$$\begin{aligned} D_{\min} &= \sum_{j=1}^{8,490} n^{(j)} \cdot \left(\log_2 \max\{p^{(j)}\} + 12 + 11 + 1\right) \\ &= \left(\sum_{j=1}^{8,490} n^{(j)} \cdot \log_2 \max\{p^{(j)}\}\right) + 167,236,272 \cdot 24 \\ &= 5,134,034,180 + 4,013,670,528 \;=\; 9,147,704,708 \text{ (bit)} \end{aligned}$$

となる．原著では，各種のマージンのために必要最低限の情報量に対して 2.5 倍の DRAM 容量を確保していると記述がある．よって $D = 2.5 \cdot D_{\min}$ とおくと，$D = 22,869,261,770(\text{bit})$ であり，原著に示されたウェハの面積は，周辺回路や配線領域を考慮してよりマージンを確保した値と見てとれる．すなわち，**「DRAM 容量・面積について，原著の評価は妥当である」**と言える．但し，DRAM の動作周波数や cache とのインタフェース，また τ のビット数 11 ビットやマージンの 2.5 倍の妥当性等，実装に向けて考慮する必要がある問題はここでは一切考慮しておらず，DRAM 容量を原著に書いてある通りに単純に見積っただけであることを述べておく．

6.4 デリバリセルの動作周波数・回路規模

Tiny station，Smallish station，Largish station の各デリバリセルのうち，最もクリティカルと考えられる Largish station のデリバリセルを以下で評価する．

各デリバリラインで加算されていく $g(a)$ は，最終的に閾値 $T = 1,024$ と比較される．$g(a)$ の中間結果を桁上げ保存表現し，各デリバリセルでは桁上げ保存加算を適用するとき，桁上げ保存表現を成す二つの 2 進数はせいぜい重み 2^9 までを保持すればよい．桁上げ保存加算の論理の最上位ビット付近には特殊な論理を付加すれ

ば，閾値を超えた中間結果が再び閾値以下を示す値となる (オーバーフローする) のを避けることができる．

また，Largish station では，素数の範囲は $5.2 \cdot 10^5 < p < B_R = 3.5 \cdot 10^9$ であり，よってデリバリラインで加算する値の範囲は $\lfloor \log_2 5.2 \cdot 10^5 \rceil = 19 \leq \lfloor \log_2 p \rceil \leq \lfloor \log_2 3.5 \cdot 10^9 \rceil = 32$ である．すなわち，$\lfloor \log_2 p \rceil$ は 14 種類の値をとるため，その信号線は $\lceil \log_2 14 \rceil = 4$ ビットあればよい．

さらに，各デリバリラインには 4,096 個のデリバリセルが存在するため，加算すべきデリバリセルのインデックスを示す l の信号線は $\log_2 4,096 = 12$ ビットあればよい．これらに加えて，デリバリラインを流れる $\lfloor \log_2 p \rceil$ および l が有効であるかどうかを示す信号線が 1 ビットあればよい．

デリバリセルを図 6.9 に示し，Verilog-HDL を用いたその回路記述例を B.1 節に示す．この回路記述を原著と同条件の 0.13 μm プロセス相当のセルベースライブラリを用いて論理合成した結果，クロック周期約 1ns，ゲート数約 388 ゲートであった．一般に，トランジスタ数とゲート数の関係は，3 対 1〜4 対 1 程度と言われている．よって，**「デリバリセルについて，原著の評価 (1 GHz 動作，1,220 トランジスタ) は妥当である」**と言える．

実際には Largish station のデリバリセルは，インタリーブされているため，$r = 4$ 個を一組としてそのうち 1 個のみが $g(a)$ 中間結果を加算する桁上げ保存加算器を持つ．インタリーブされたデリバリセルの組を図 6.10 に示し，Verilog-HDL を用いたその回路記述例を B.2 節に示す．この回路記述を同条件の 0.13 μm で論理合成した結果，クロック周期約 1ns，ゲート数約 704 ゲートであった．すなわちデリバリセル一つあたり，$704 \div 4 = 176$ ゲートである．よって，**「インターリーブしたデリバリセルについて，原著の評価 (1 GHz 動作，530 トランジスタ) は妥当である」**と言える．

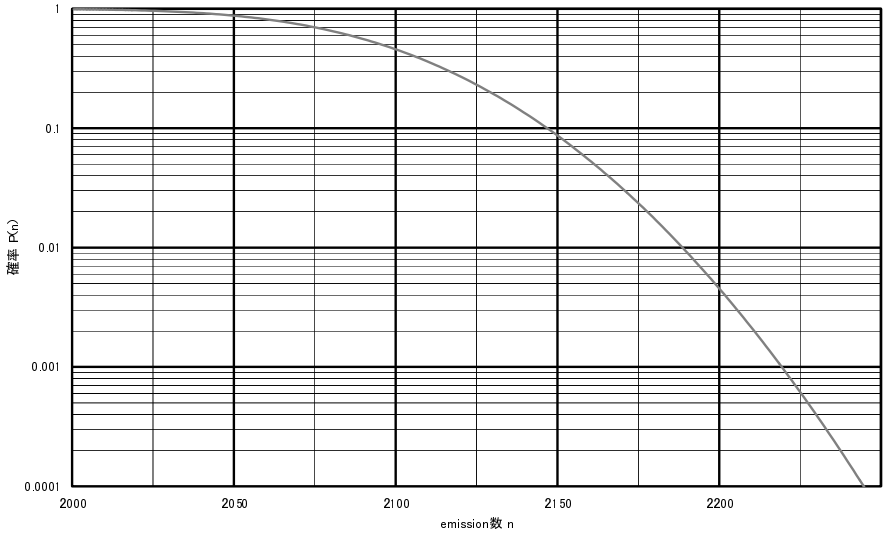


図 6.8: n 個以上のプログレ放出が放出される確率 $P(n)$ (対数表示)

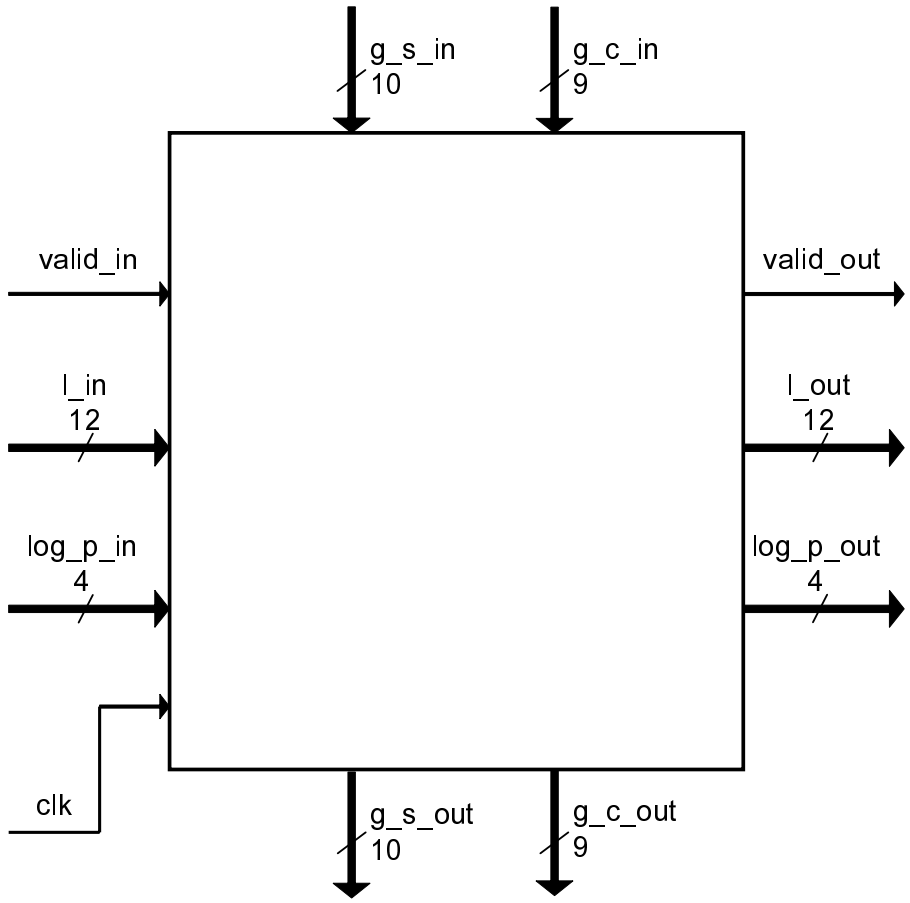


図 6.9: デリバリセル

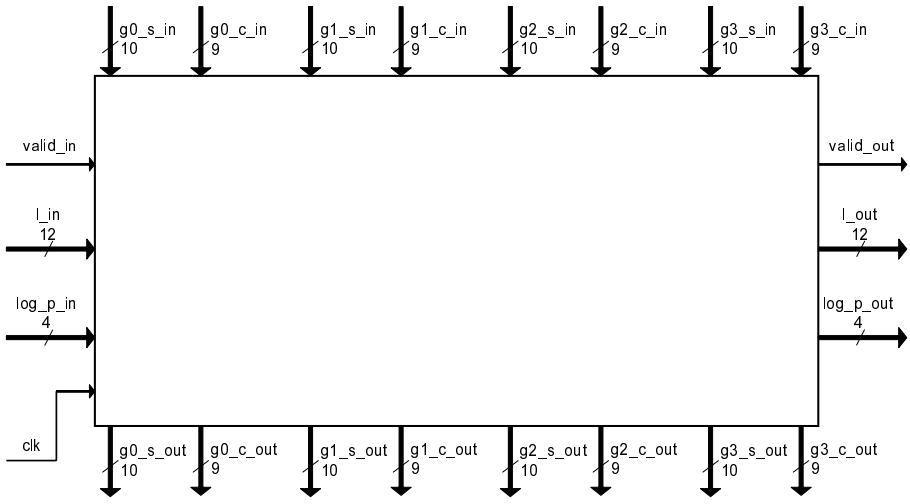


図 6.10: インターリーブしたデリバリセルの組 ($r = 4$ 個分)

第7章　消費電力評価

本節では，動作が比較的明快な，シストリックアレイ部分の消費電力を見積もる．

7.1 半導体製品動向 ('03 年)

現在 (2003 年時点で) 利用可能な半導体プロセスにおいて，ゲート当たりの消費電力で比較的小さいもには，TSMC 社の 0.13 μm プロセスにおけるゲート当たりの消費電力の 4nW / MHz がある [52].

以下，本検討では，ゲート当たりの消費電力基本単位として 4nW/MHz を用いながら評価を行なう．

7.2 消費電力評価

7.2.1 トランジスタ数からの見積もり

ゲート規模，動作率等が推測しやすく，有理数篩の大部分を占めるデリバリセルで構成されるシストリックアレイ部分の消費電力を見積もる．

TWIRL を記述した論文 [49] より，1,024 ビット合成数を分解するためのパラメータ例として，表 7.1 に示す数値データが与えられている．インターリーブを行なわないデリバリセルは，バスライン 4 本分で，1,220 トランジスタを消費する．インターリーブ型のデリバリセルは，縦方向のライン 4 本分をまとめて，530 トランジスタを消費する．以上のことが [49] から読み取れる．

よって，インターリーブ型のデリバリセルの個数は，

$$4,096 \times 2,100 = 8,601,600(\text{個})$$

通常 (インターリーブなし) のデリバリセルの個数は，

$$4,096 \times (501 + 54) = 2,273,280(\text{個})$$

これより，有理数篩におけるシストリックアレイ部分の総トランジスタ数は，

$$8,601,600 \times 530 + 2,273,280 \times 1,220 = 7,332,249,600$$

と見積もることができる．

現在一般的な半導体における低消費電力化のためには，CMOS 構造を採用するのが最良である．CMOS 構造の単位ゲートを，2 入力 NAND 回路と想定すると，ゲート-トランジスタ換算は，1 ゲート = 4 トランジスタ である．これより，シストリックアレイ部分の総ゲート数は，

$$7,332,249,600/4 = 1,833,062,400(\text{ゲート})$$

となる．

一方各デリバリセルへ供給されるクロックは 1 GHz であるが，内部の組み合わせ回路および FF(フリップフロップ) のデータとして入力される信号の周波数は 0.5 GHz である．

各デリバリセルの動作率を 50%と想定すると，シストリックアレイ部分の消費電力 P_{array} は，

$$\begin{aligned} P_{array} &= 1,833,062,400(\text{ゲート}) \times 4(\text{nW/MHz}) \times 0.5(\text{GHz}) \times 0.5 \\ &= 1,833,062,400 \times 4 \times 10^{-15} \times 0.5 \times 10^{9} \times 0.5 \\ &\approx 1,833(\text{W}) \end{aligned}$$

と見積もれる．ただし，これはデリバリセルで構成されるシストリックアレイ部分のみの消費電力であり，バッファ，プロセッサ，DRAM，キャッシュ等の消費電力は含んでいない．最低でもこれだけの電力消費があると言える．

この値より，TWIRL 全体を概算する．上記より有理数篩 1 個当たり，最低 1,833 (W) の消費電力であった．[49] より，有理数篩 4 個で 1 ウェハを占有することから，1 ウェハ当たり

$$1,833 \times 4 = 7,332(\text{W})$$

の消費電力は少なくとも期待される．次に，TWIRL のクラスタを考えた場合，3 枚のウェハで 1 クラスタを構成する．よって，1 クラスタではこの 3 倍の電力を消費すると考えてみる[1]．TWIRL 装置を用いて 1,024 ビット合成数を 1 年間で処理する場合，その想定として，全体で 194 クラスタを使うので，TWIRL 全体の消費電力は，

$$7,332 \times 3 \times 194 = 4,267,224(\text{W}) \approx 4,267(\text{kW})$$

[1] これは代数篩でも同じ対面積比の電力消費があることを想定している．しかしながら，代数篩の装置では DRAM バンクの割合が，有理数篩のそれよりも多いことから，この仮定は若干，消費電力を多めに見積もることになる．

となる．この装置を一時間動作させた場合の消費電力は 8,534 (kWh)．よって一ヶ月では，

$$4,267 \times 24 \times 30 = 3,072,240(\text{kWh})$$

の電力消費となる．

一般家庭の一ヶ月の電力消費が，300kWh であるため，TWIRL は，約 10,000 戸分の生活に使われる電力を消費する．

これらの消費電力は，そのまま熱になるため，その電力消費だけでなく，かなり大掛かりな冷却設備等が必要になると考えられる．

なお，大型の計算機として，地球シミュレータ [53] があり，その消費電力は 8 MW 程度といわれている [67].

7.2.2 デリバリセル実装評価による見積もり

この節では，デリバリセルの実装評価のゲート数より，より正確に消費電力を見積もることを考える．

我々の評価では，表 7.2 のゲート消費を見積もった．これによると，シストリックアレイ部分の総ゲート数は次のように見積もられる．Smallish, Tiny の計算ステーションにおいては，インターリーブを行なわないデリバリセルを用いるので，

$$388 \times 4,096 \times (501 + 54) = 882,032,640(\text{ゲート})$$

となる，一方 Largish 計算ステーションにおいては，インターリーブが適用されるので，

$$176 \times 4,096 \times 2,100 = 1,513,881,600(\text{ゲート})$$

のようになり，合計

$$882,032,640 + 1,513,881,600 = 2,395,914,240(\text{ゲート})$$

となる．

また，シストリックアレイ部分の FF の数についても同様に見積もる．Smallish, Tiny 計算ステーションについては，縦方向の 19 ビットのバスと，横方向のライン番号 (12 ビット)，フラグ (1 ビット)，対数スコア (4 ビット) の合計 36 ビットの FF

を利用する．よって

$$36 \times 4,096 \times (501 + 54) = 81,838,080$$

となる．また，Largish については，パイプライン化を行なっているので 4 ライン分の FF の見積りが，19×4 ビットの縦方向のデータ，ならびに Smallish, Tiny と同様な 17 ビットのデータを扱うことから，$19 \times 4 + 17 = 93$ ビット分の FF が必要．よって Largish では，

$$93 \times 4,096/4 \times 2,100 = 199,987,200$$

だけの FF を用いる．詳細については，第 7.3 節，クロックツリーのゲート数と消費電力を参照頂きたい．

以上から，シストリックアレイ部分の FF の総和は 281,825,280(個) となる．FF は 6 ゲートで構成されるため，FF 部分のゲート数は，

$$6 \times 281,825,280 = 1,690,951,680(\text{ゲート})$$

組み合わせ論理回路部分は，

$$2,395,914,240 - 1,690,951,680 = 704,962,560(\text{ゲート})$$

となる．

よって，組み合わせ論理回路部分の消費電力は，

$$704,962,560 \times 4 \times 10^{-15} \times 0.5 \times 10^{9} \times 0.5 \approx 705(\text{W})$$

ここで，動作周波数が半分になっているのは，データの変化の周期は，回路全体の動作クロック 1 GHz に対してその約 50%，つまり半分と見積もっているためである．

また，FF 部分はクロックに依存して電力消費する部分と，データが変化することにより電力消費する部分に分けられる．クロック依存部分とデータ依存部分の比を 1:2 とすると，クロック依存部分の消費電力は

$$1,690,951,680 \times 4 \times 10^{-15} \times 1 \times 10^{9} \times 1 \times 0.33 \approx 2,232(\text{W})$$

となり，データ依存部分の消費電力は，

$$1,690,951,680 \times 4 \times 10^{-15} \times 0.5 \times 10^{9} \times 0.5 \times 0.67 \approx 1,133(\mathrm{W})$$

データ依存部分の周波数が半分になっているのは，データの変化の周期は1GHzの半分であるためである．

以上より，シストリックアレイ部分の総消費電力は，

$$705 + 2,232 + 1,133 = 4,070(\mathrm{W})$$

となる．

第7.2.1節の結果である1,833 (W)と比較すると，倍以上差が生じてることに注意する．これは，最初のゲート総数の見積もりに差があること．および，デリバリセルの大半が動作周波数0.5 GHzで動作していると想定したが，実際はデリバリセル内のFFの占める割合が多く，1 GHzで動作してる部分がかなり存在することからくると考えられる．よって，約4.1 kWの方が信憑性が高いと思われる．

以上により，TWIRL全体では，一般家庭の約22,000戸分の消費電力量に匹敵する消費電力を継続的に消費すると考えられる．

7.2.3　デリバリセルの動作率

この節では，第7.2.2節，第7.2.1節において，電力消費を換算するために，デリバリセルの動作率を0.5と仮定した．これについての根拠を示す．

電力消費を検討したシストリックアレイの構造を考える．横方向には，4,096本のバスラインが並び，縦方向には，Largish, Smallish, Tinyあわせて2,655本のデリバリラインが並ぶ構造となってる (もちろん，各計算ステーションによりそのデリバリセルの構造は異なる)．便宜上，縦のバスラインのデータの流れを，上から下へ流れるとし，デリバリラインのデータの流れを，左から右へ流れるとする．

縦のバスラインのデータの流れを見ると，上の方のデリバリセルでは加算による値の変化は少ない．一方，下の方 (つまり，比較器に近い側) のデリバリセルでは，上の方のデリバリセルで加算されたデータが各クロック毎に次々と落ちてくるので，毎クロック異なるデータを保持する確率が高い．以上の観察から，縦方向については平均，50%の動作率と見積もることは妥当である．

一方，デリバリラインのデータの流れを見る．左の方 (すなわち，煙突やバッファに近い側) のデリバリセルは，毎クロック新しいデリバリペアが送られてきて，それを処理する確率はかなり高いと考えられるが，反対の右端のデリバリセルまでデリバリペアが届くことは稀である．よって，横方向についても 50%の動作率と見積もることができる．

一方，縦方向の動作と横方向の動作は，完全に独立しているため，左下のデリバリセルは，ほぼ 100%の動作率，左上のデリバリセルは，ほぼ 50%の動作率，右下のデリバリセルは，ほぼ 50%の動作率，右上のデリバリセルは，ほぼ 0%の動作率となる．これを全体で平均化すると，約 50%の動作率となる．

7.3 クロックに関する評価

LSI を設計する上で考慮しなければならない項目にクロックがある．

最近の LSI 設計では，内部論理回路を同期回路として設計し，設計時にはクロックを意識させない設計手法が一般的になってきている．一般の LSI 設計手順では，論理回路設計後に，論理合成を行い，配置配線を行う．配置配線を行うと，実際の信号線の配線長がわかり，信号伝播遅延等が正確にわかる．この配置配線中に論理回路中の FF 同位相のクロックが供給されるよう，クロックツリーを生成し，クロックドライバを配置してゆく．こうしたクロック供給の仕組みを構築することも LSI 製造の上で無視できない過程である．しかし，最近の LSI 開発では，クロックツリーの生成は，自動化されてきている．

TWIRL の製造についても内部論理回路は単一クロックに完全同期して動くように設計するのが最も簡単な設計方法である．

しかし，設計者のゲート数の見積もりとは別に，クロックツリーを構成するクロックドライバのゲート数が追加されてしまい，さらにクロックドライバも電力を消費する．

よって，本節では，想定する装置を動作させるために必要であろうクロックに関するメカニズムを検討し，クロックツリーに因るゲート数の増加と消費電力を評価する．

7.3.1 クロックツリーのゲート数と消費電力

クロックツリーには，論理回路中のFF(フリップフロップ) の数により，必要なクロックドライバの数が決定される．ここでは，第7.2.2節で評価したシストリックアレイ部分に，クロックツリーを付加することを考える．

通常のデリバリセルには，縦方向のデータ転送には19ビット (10ビットが各篩ポイントに対する現時点のスコアであり，その情報をCarry save adderにより実現しているので合計19ビット) が必要なので，19個のFFが必要．横方向のデータ転送には17ビットが必要なので，17個のFFが必要．よって，合計36個のFFが必要．

インターリーブ型デリバリセルでは，縦方向は19個で同じだが，横方向は，1/4になる．デリバリセル4個分で，$19 \times 4 + 17 = 93$ 個のFFが必要になる．

以上よりシストリックアレイ部分のFFの数は，Smallish, Tinyの計算ステーションでは

$$36 \times 4,096 \times (501 + 54) = 81,838,080(\text{個})$$

となり，同様にLargishでは，

$$93 \times 4,096/4 \times 2,100 = 199,987,200(\text{個})$$

よって，FFの総和は281,825,280(個) になる．

一般には，クロックドライバ1個で，最大で16個のFFしか駆動しない．また，クロックドライバ同士は，図7.1のように，配線長が等しくなるよう，H状に階層構造を構成することに，具体的には1個のクロックドライバで4個のクロックドライバを駆動する構成をとる．このような，クロックツリーの一般的構成方法により，合計281,825,280 (個) のFFにクロックを供給しようとする．それに必要なクロックドライバの数を以下に評価してゆく．

クロックドライバの数を図7.1に示した構成の末端，すなわちFFに近い側からカウントして行く．直接FFを駆動する末端のクロックドライバ数は，

$$281,825,280/16 = 17,614,080(\text{個})$$

である．以降の (上の) 階層について考えて行くと，上位のクロックドライバの個数 n_d^{k-1} は，下位のクロックドライバの個数 n_d^k の4分の1，$n_d^{k-1} = \lceil n_d^k/4 \rceil$ であるから，各階層毎に下の階層から順に以下の個数のクロックドライバが必要になる．

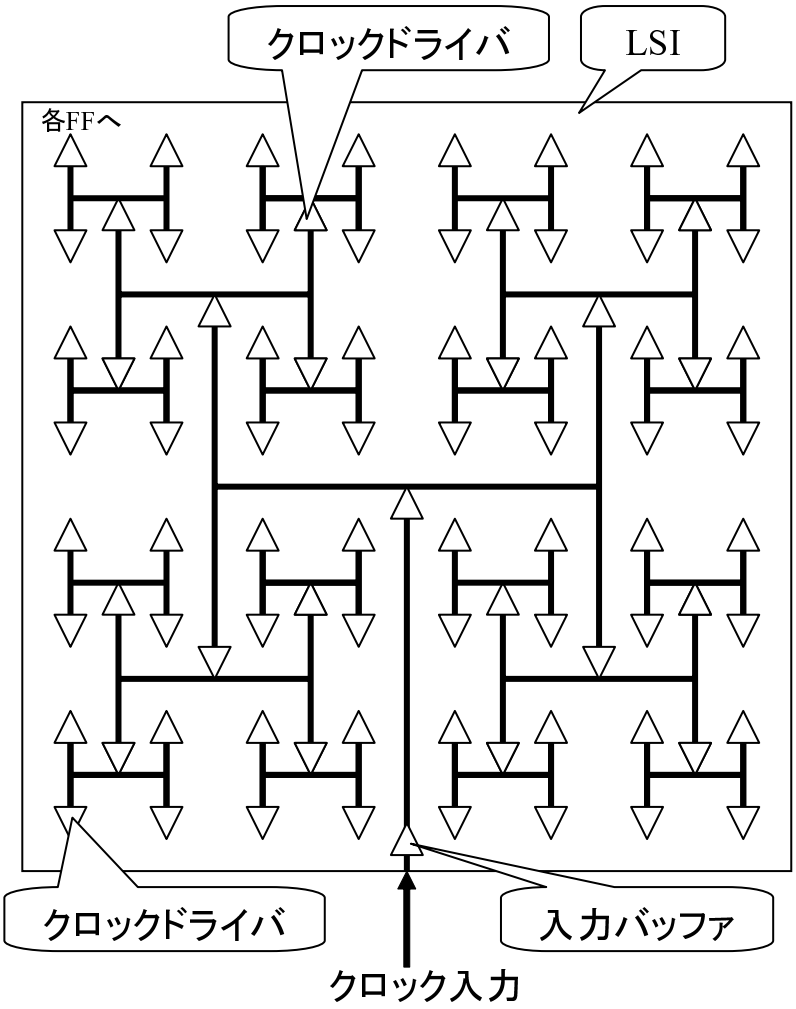


図 7.1: 全回路同期に用いる一般的なクロックツリーの構造

第 14 階層 (最下位階層)	17,614,080
第 13 階層	4,403,520
第 12 階層	1,100,880
第 11 階層	275,220
第 10 階層	68,805
第 9 階層	17,202
第 8 階層	4,301
第 7 階層	1,076
第 6 階層	269
第 5 階層	68
第 4 階層	17
第 3 階層	5
第 2 階層	2
第 1 階層 (最上位階層)	1

これらを合計すると，23,485,446 個になる[2]．

クロックドライバは一般には，1〜3 ゲートで構成されている．駆動する FF の個数が多いクロックドライバや，配線長が長くなる上位階層のクロックドライバほど，クロック信号線の駆動能力が必要なためゲート数が増える．また，高い周波数での駆動にもゲート数が増える．ここでは，簡略化のために，1 クロックドライバを 3 ゲート相当として見積もる．

クロックドライバ数は，23,485,446 個であるため，クロックドライバのゲート数は，

$$3 \times 23,485,446 = 70,456,338(\text{ゲート})$$

となる．

これは，原著トランジスタ数からのゲート見積もりである，1,833,062,400 (ゲート) の 3.8%，我々の実装評価でのゲート見積もりである，2,395,914,240 (ゲート) の 2.9%に相当する．

一方，クロックドライバはクロック周波数で常に動作するため，周波数 1 GHz，動作率 100%となる．よって，消費電力は，

$$70,456,320 \times 4 \times 10^{-15} \times 1 \times 10^{9} \times 1 = 282(\text{W})$$

これは，原著トランジスタ数からの消費電力見積もりである，1,833 (W) の 15.4%，我々の実装評価での消費電力見積もりである，4,070 (W) の 6.7%に相当する．

[2]最上位の階層から 1, 4, 16(= 4^2), 4^3, ... のようにつみあげて，17,614,080 を越えるような構成は上記の構成よりもより多くのクロックドライバ (39,983,701 個) を消費する．

以上より，クロックツリーによる，ゲート数の増加は，全体に与える影響は少ないかもしれないが，消費電力の増加は無視できない．クロックツリーそのものは，計算や演算を行わないため，回路面積，消費電力の観点からはオーバヘッドでしかない．

7.3.2 信号伝播遅延の評価

原著のトランジスタの面積，デリバリセルのトランジスタ数より，シストリックアレイ部分の面積を概算すると，

$$2.8(\,\mu\mathrm{m}^2) \times 7,332,249,600(\text{個}) = 20,530(\,\mathrm{mm}^2)$$

これを，正方形にすると，1 辺 143 mm になる．

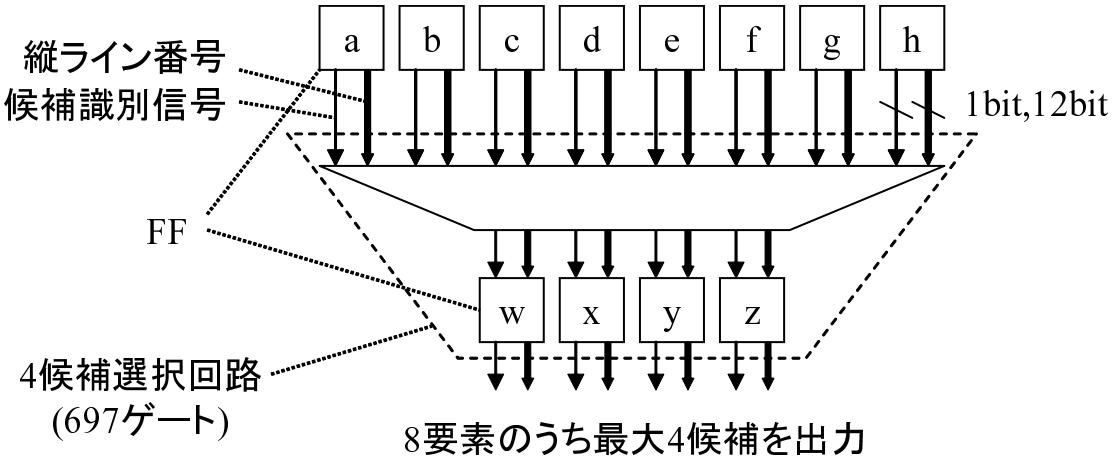
0.13 μm プロセスの LSI の世界では，1 mm の信号伝搬に，約 0.12 ns かかる．よって，143 mm 信号が伝搬するためには，17.2 ns かかることになる．これは，1 GHz では，17 クロックに相当する．

これは，シストリックアレイ部分の出力，4,096 ラインの出力に，閾値を超えた候補が現れても，1 クロックでは選択できないことを示している．

提案のカスケード方式では，有理数篩を通過した出力群をデータとして集約したあと，バッファリングするなどして，転送を行なう．よって，有理数篩から，代数篩へデータを受け渡す部分にも，4,096 個の出力から候補をまとめて扱うために，何らかの工夫が必要になる．例えば，1 クロック毎に候補を 1/2 に絞ってゆく方法などを取る必要がある．

有理数篩をパスする候補は，1 サイクルあたり最大 4 候補であるとしている．よって，最大 4 候補はそれ以上絞らないようにしなければ成らない．例えば，図 7.2 のように，8 候補から 4 候補を選ぶ回路をつくり，それを三角形に 1,023 個並べる．このような構成をとることにより，毎クロック最大 4 候補を出力する回路を作ることが可能になる．

しかし，これには全体で，713,031 ゲートが追加され，4,092 個の FF が追加されるため，クロックツリーも増える．しかしこれらは，シストリックアレイ部分の規模と比較すると小さいため，大勢に影響は無い．



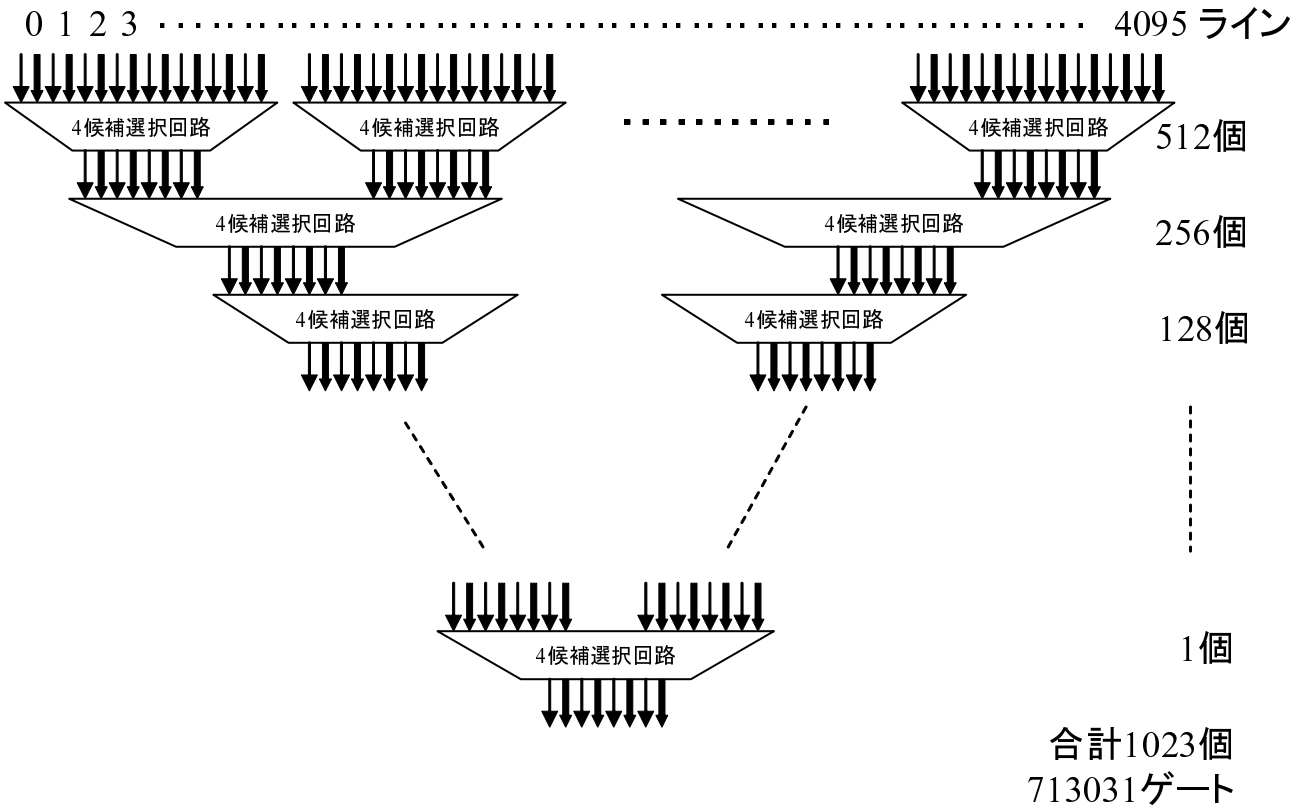


図 7.2: 同期したシストリックアレイ末端部分の 4 候補選択のための回路構成

7.3.3　クロックツリーを用いない設計

[49] で提案された TWIRL デバイスのシストリックアレイ部分では，信号の帰還が存在しないため[3]，完全な同期回路として設計しなくとも，実現できる可能性は存在する．ただし，データの遅延とクロックの遅延を正確に制御しながら設計する必要が出てくるため，設計に必要な労力は桁違いに増える．

各論理素子の遅延のみでなく，信号線とクロック線の配線長等のレイアウト情報も考慮する必要が出てくるため，熟練者が設計しないと誤動作する回路になる．

シストリックアレイは 2 次元の方向を持つため，1 方向を同期クロックを用い，1 方向を非同期クロックを用いる方法も考えられる．

横方向を非同期クロックにすると，図 7.3 のように，4,096 の篩ポイントの最終出力が少しずつずれて出力され，シストリックアレイ部分全体では，17 クロックにも及ぶ．

この 4,096 候補の中から，最大 4 候補を選択するのにも工夫が必要になる．図 7.4 のように，1 個ずつ篩ポイントを減らし，最大 4 候補のみが残るようにする必要がある．この場合，5 個の篩ポイントから最大 4 候補を選択する回路が，4,092 個，13 ビットの FF が 4,091 から 1 までの和の 8,370,186 個必要である．4 候補選択回路 2 は 444 ゲートで実現可能で，13 ビットの FF は，78 ゲートで実現可能なので，合計

$$444 \times 4,092 + 78 \times 8,370,186 = 654,691,356(\text{ゲート})$$

が必要になる．これは，[49] 記載のトランジスタ数から求めた，シストリックアレイ部分の総ゲート数 1,833,062,400 の 35.7% にあたり，我々の実装評価から求めた，シストリックアレイ部分の総ゲート数 2,395,914,240 に対しても 27.3% に相当する．これは無視できない回路規模を占めることになり，30 cm ウェハへの実装を困難とするゲート増加と考えられる．

一方，縦方向のクロックを非同期化すると，図 7.5 のように下に配置されるデリバリラインほど，クロックの到達する時間が遅くなるため，バッファはデリバリラインへ出力する，同クロックに対応するデータを遅らせる必要がある．これらは，配線遅延や半導体素子の遅延を利用するしかないが，遅延時間の制御には慎重な設計が必要である．

[3]なお，TWIRL 装置の補足説明で，パスした篩ポイントに対して貢献した Largish プログレッションを報告する日記や，それを運ぶバスラインについて触れられており，これらは計算ステーショ

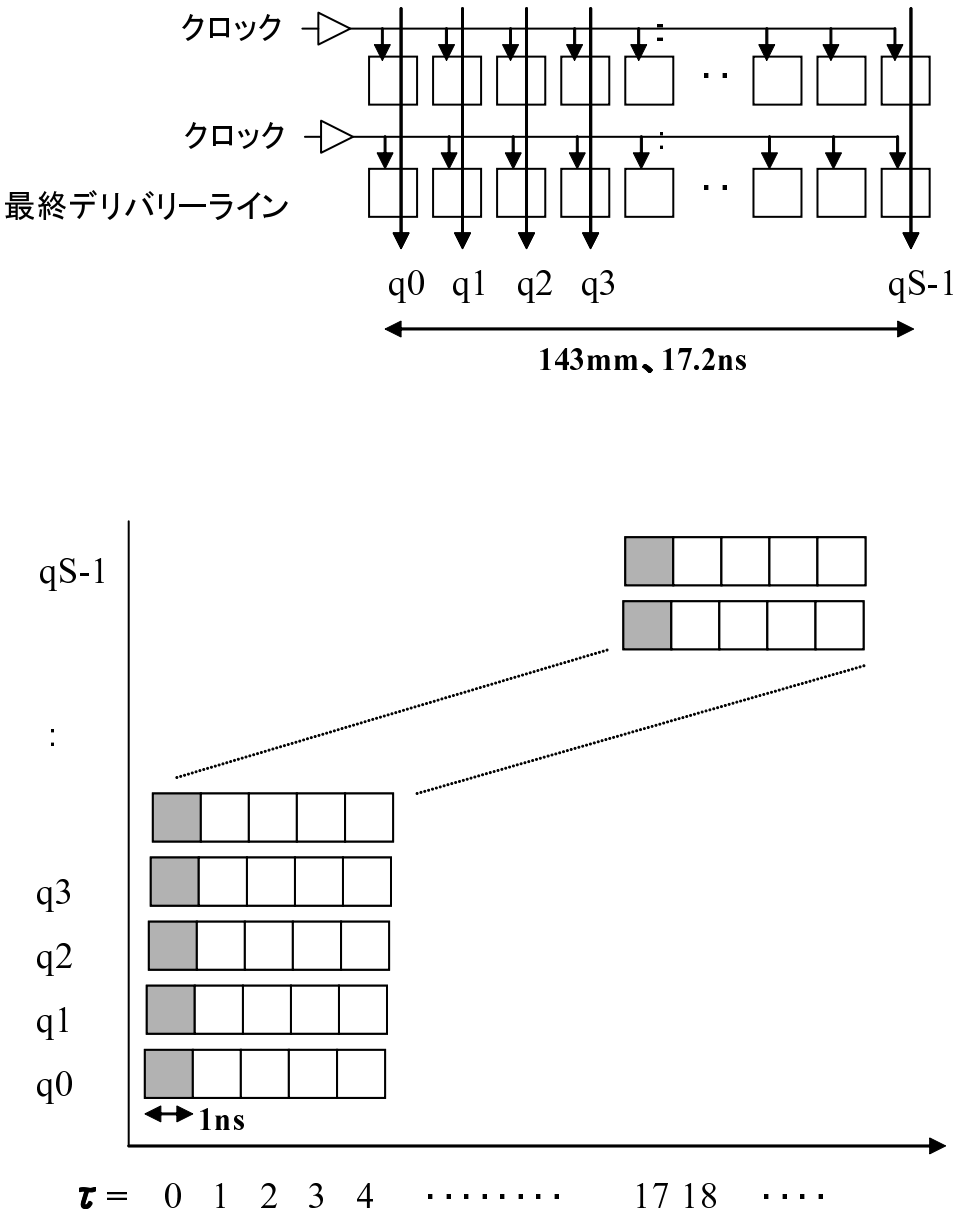
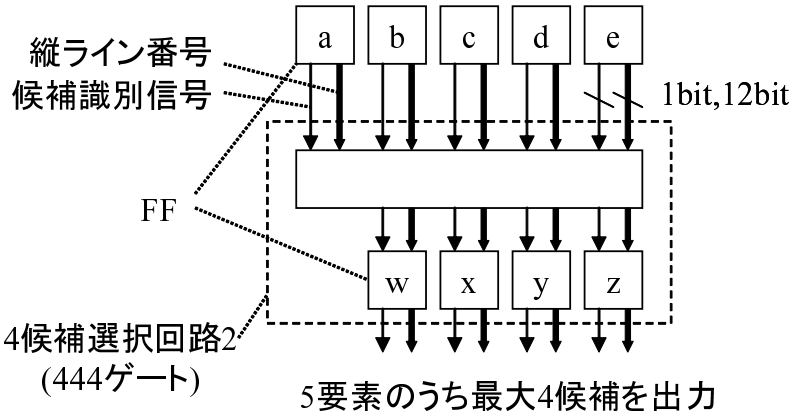


図 7.3: 非同期クロックによる各ライン出力のタイミング遅延を示すブロック図



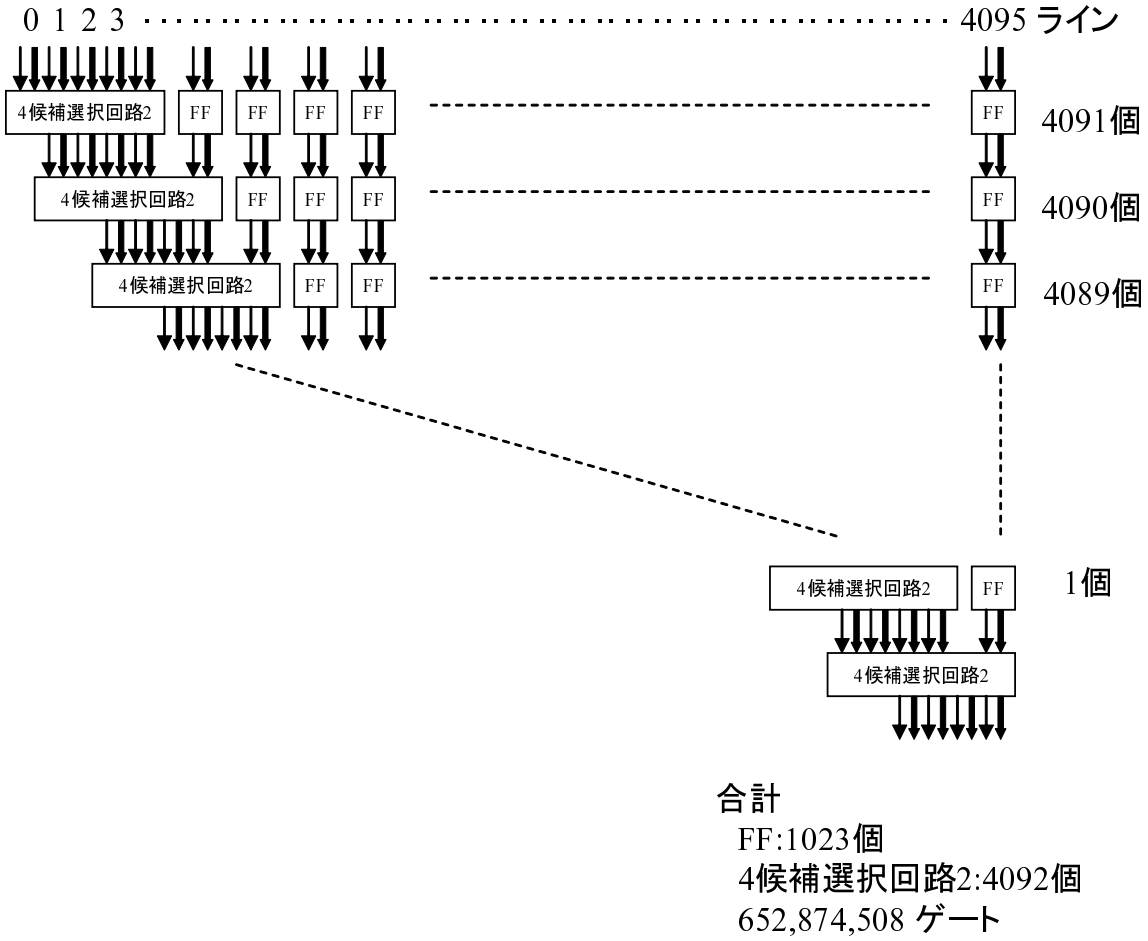


図 7.4: 非同期なシストリックアレイ末端部分の 4 候補選択のための回路構成

表 7.1: 電力消費見積りのためのパラメータ一覧

Largish プログレッション部分のデリバリライン数	:	2,100 本
Smallish プログレッション部分のデリバリライン数	:	501 本
Tiny プログレッション部分のデリバリライン数	:	54 本
並列度	:	4,096
デリバリセルで消費するトランジスタ数 (インターリーブなし)	:	1,220 tr
デリバリセルで消費するトランジスタ数 (インターリーブあり)	:	530 tr

表 7.2: デリバリセルのハードウェアゲート消費見積り

通常のデリバリセル (インターリーブなし)	:	388 ゲート
インターリーブ型のデリバリセル	:	176 ゲート

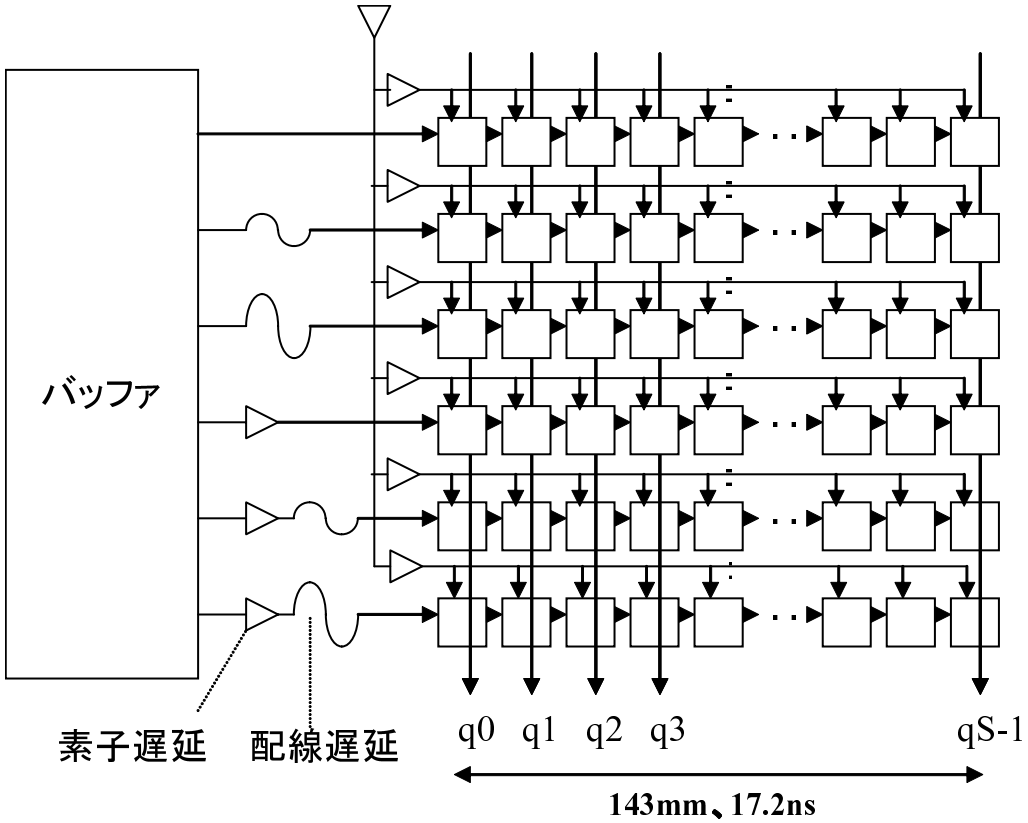


図 7.5: 非同期な回路構成を行なう場合のバッファ出力をデリバリラインへ投入する場合の遅延の挿入

以上の考察から，クロックツリーによる消費電力増加を抑えるためには，非同期クロックを用いた方が一見有利に見えるが，非同期クロックを用いることによるシストリックアレイの拡張回路の複雑化を懸念すると，ゲートの増加や設計の複雑化が伴い，余計なコストがかかってしまう．よって，前述の，同期クロックを用いた方が実現の可能性は高いと思われる．

ン間を渡るシストリックアレイ部分とは逆方向のパイプライニングを実装しなければならない．

第8章　製造に関するコメント

30 cm ウェハで 1 個の LSI を作ることの実現可能性を，マスクの観点から評価する．現在最も一般的な LSI の製造方法は，写真技術の応用で，半導体ウェハ上に感光剤を塗り，マスクを載せ光を当て感光剤を感光させる．この操作により感光した部分とマスクにより光を遮られ，感光しなかった部分ができる．この感光，非感光に応じて半導体ウェハ上の領域を選択することが可能であり，選択領域に対して，不純物をイオン打ち込みしたり，熱拡散したりしてトランジスタを製造し，また，選択的にエッチングすることで，配線層を製造していく．一般的には 25 枚程度のマスクを用いて LSI を製造していく．

現在量産されている半導体の最先端のプロセスは 0.13 μm 程度であるが，このプロセス用の 30 cm 四方のサイズのマスクは現時点存在しない．上層の配線層で 1 μm よりも粗い配線を造るマスクには，30 cm ウェハを一枚で一括露光するものは存在するが，現状では最大でも 30 mm 四方程度を感光させることができる程度である．

よって，30 cm のウェハで 1 個の LSI を製造するには，LSI を複数箇所に分割して，それぞれマスクを製造するしかない．

しかし，隣り合うマスクの配線の接続や絶縁を全て確実にするのは困難と思われる．通常は，隣り合う LSI は，カッターで切り離されるため，削りしろの隙間を開けて配置する．また，マスクの位置合わせ精度は，前回のマスクとの相対誤差を，押さえ込めば良く，実現は比較的楽である．一方，30 cm のウェハでは，相対精度ではなく，絶対精度が求められる為である．

以上により，現状の半導体製造技術をもって 30 cm ウェハ規模の LSI を作ることは，容易ではないと判断する．

第9章　まとめ

本評価では，1024 ビット素因数分解の可能性を検討目標として，近年提案された，専用集積回路についての実現性を中心に評価を行なった．

評価対象の選定　本評価では，第 1 章で議論したとおりの，素因数分解アルゴリズムとして，一般数体篩法，ならびのその改良を考察した．これは，近い将来，アルゴリズム的な劇的進歩がないことを仮定すれば，妥当な選択である．

次に，本評価の第 2 章で議論したとおり，これまでの汎用 CPU の並列実装でのスケーラビリティや，将来への進歩速度については比較的推定可能である．しかし，近年のブレイクスルーとして大きいものにカスタムハードウェアを使った新しい計算機モデルの提案，ならびにそれの数体篩法への適用の検討がある．

本評価ではこれらの動向に注目し，数体篩法の各処理の中でも実現性を議論する上で最も重要な話題を，篩処理に限定した．これは，もう一つ一般的に大きな懸念とされる行列演算の部分については，画期的なハードウェア構成が提案され，篩処理の実現性に比較してかなり高い現実性をもっているからである．

本評価では篩法の模索を行ない，最終的に 1024 ビット合成数を分解することの現実性を議論するための対象として，TWIRL 装置の評価を行なうこととした．その考察に至るまでの技術的，非技術的検討は，第 2 章に記載した．

対象装置に対する実現性評価　第 3 章では，評価対象のクレームの明確化の目的で，提案を忠実に再現した．続く，第 4 章で，さらに詳細部分について，評価可能な部分までの理解の噛み砕きを紹介した．以上の背景から以下のハードウェア評価を行なった．

第 5 章では，装置の提案者らが提示する装置の規模を示すデータを用いてそのような大規模装置の実現，という観点から LSI 技術の知識に基づいて考察した．ここでの主なクレームは，(1) 検討しているサイズの論理回路を動作まで進めることが，その規模の大きさゆえ，不可能であること，(2) 費用について，その初期投資について見積りが不可能であること，である．

第 6 章では，[49] の主要な主張を支える評価を行なった．これは，第 5 章の主張を裏付けるものにもなる．具体的には，部分部品の設計と，装置動作の一部分について検証を行なった．具体的には，致命的となる構造の部品要素については，我々で検証可能なサイズで見積もられており，回路規模についての主張はおおむね正し

いといえる．この具体的設計は付録，第 B 章に記載してある．

第 7 章では，これら，著者らの実装プロファイルが仮に妥当であったとして，それらが現実世界において，動作可能であるか，という観点から考察を行なった．特に，ここでは，消費電力に関する考察と，装置を動作させるのに必要なクロックツリーとそのコストという切口から考察を進めた．この結果，電力消費は，最低見積りにより，オーダーとして，日本に存在する大型計算機，地球シミュレータ，の消費電力程度だった (第 7.2.1 節)．実際に動く計算機が日本に存在することから，これは現実的であるといえる．

以上の解析，評価をもとに，筆頭にあげる要旨に記載した考察を与えた．

関連図書

[1] L.M. Adleman, "Factoring Numbers Using Singular Integers," Proc. of STOC, pp. 64–71, 1991.

[2] F. Bahr, J. Franke, T. Kleinjung, M. Lochter, and M. Böhm, "RSA-160," e-mail anouncement, Apr. 2003, `http://www.loria.fr/~zimmerma/records/rsa160`.

[3] D.J. Bernstein, "How to Find Small Factors of Integers," manuscript, 2000, `http://cr.yp.to/papers.html`.

[4] D.J. Bernstein, "Circuits for Integer Factorization: a Proposal," manuscript, 2001, `http://cr.yp.to/papers.html`.

[5] D. Boneh, "Twenty Years of Attacks on the RSA Cryptosystem," *Notices of the American Mathematical Society*, **46**(2): 203–213, February 1999.

[6] R.P. Brent, "Recent Progress And Prospects for Integer Factorisation Algorithms," *Computing and Combinatorics: Proc. Sixth Annual International Computing and Combinatorics Conference* LNCS vol. **1858**, pp. 3–22, Springer-Verlag, 2000.

[7] P.J. Cameron and A.M. Cohen, "On the Number of Fixed Point Free Elements in a Permutation Group, " Discrete Math. **106/107**, pp. 135–138, 1992.

[8] J.W.S. Cassels and A. Fröhlich (eds.), "Algebraic Number Theory," Proceedings of an Instructional Conference, Academic Press, London, 1967.

[9] S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, P.L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist,

G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann, "Factorization of a 512-bit RSA Modulus," *Advances in Cryptology — EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 2000. Proceedings*, B. Preneel (ed.), pp. 1–17, LNCS vol. **1807**, Springer-Verlag, 2000.

[10] H. Cohen, *A Course in Computational Algebraic Number Theory*, GTM vo. **138**, Springer-Verlag, 1993.

[11] D. Coppersmith, "Modifications to the Number Field Sieve," *Journal of Cryptology*, **6**(3): pp. 169–180, 1993.

[12] J. Franke, "2.953+ c158," `http://www.crypto-world.com/announcements/c158.txt`.

[13] J. Franke, "RSA-160," `http://www.loria.fr/ zimmerma/records/rsa160`.

[14] J. Franke, "RSA576," `http://www.loria.fr/~zimmerma/records/rsa576`.

[15] P.X. Gallagher, "The large sieve and probabilistic Galois theory," Analytic number theory, Proc. Symp. Pure Math., **24**, H.G. Diamond (ed.), Amer. Math. Soc., Providence, 1973, pp. 91–101.

[16] W. Geiselmann and R. Steinwandt, "A Dedicated Sieving Hardware," *Public Key Cryptography — PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6–8, 2003, Proceedings*, Y. Desmedt (ed.), LNCS vol. **2567**, pp. 254–266, Springer-Verlag, 2002.

[17] W. Geiselmann and R. Steinwandt, "Hardware to Solve Sparse Systems of Linear Equations over GF(2)," *Cryptographic Hardware and Embedded Systems — CHES 2003, 5th International Workshop, Cologne, Germany, September 8–10, 2003, Proceedings*, C.D. Walter, Ç.K.Koç, C. Paar, (eds.) LNCS vol. **2779**, pp. 51–61, Springer-Verlag, 2003.

[18] W. Geiselmann and R. Steinwandt, "A Special Purpose Mesh Architecture for Sieving in the Number Field Sieve," Talk Material at Workshop on the State-of-the-art of Factoring Large Numbers, Utrecht, Dec. 12, 2003, available at http://homepages.cwi.nl/~herman/GeiselmannSteinwandt.pdf.

[19] W. Geiselmann and R. Steinwandt, "Yet Another Sieving Device (Extended Version)," Manuscript, available at http://eprint.iacr.org/2003/202/.

[20] M. Huizing, "A Multiple Polynomial General Number Field Sieve," *Algorithmic Number Theory, Second International Symposium, ANTS-II, Talence, France, May 18 – 23, 1996, Proceedings*, H. Cohen (ed.), LNCS, vol. **1122**, pp. 99-114, Springer-Verlag, 1996.

[21] M. Huizing, "An Implementation of the Number Field Sieve," *Experimental Mathematics* vol. **5**, pp. 231–253, 1996.

[22] International Technilogy Roadmap for Semiconductors, 2001, http://public.itrs.net/

[23] B. Kaliski, "TWIRL and RSA Key Size," Technical Notes and Reports, May 1, 2003, http://www.rsasecurity.com/rsalabs/technotes/twirl.html.

[24] H.J. Kim and W.H. Magione-Smith, "Factoring Large Numbers with Programmable Hardware," *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays, Monterey, California, US*, H.J. Kim and W.H.Mangione-Smith (eds.), pp. 41–48, ACM, 2000.

[25] R. Lambert, *Computational Aspects of Discrete Logarithms*, Ph.D. Thesis, University of Waterloo, 1996.

[26] S. Lang, *Algebraic Number Theory*, GTM **110**, Springer-Verlag, 1986.

[27] A. Lenstra, "New Factorization Record," Posted to the sci.crypt.research NNTP News Group, http://www.loria.fr/~zimmerma/records/RSA-130.

[28] A.K. Lenstra and B. Dodson, "NFS with Four Large Primes: an Explosive Experiment," *Advances in Cryptology — CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 1995. Proceedings*, D. Coppersmith (ed.), LNCS vol. **963**, pp. 372–385, Springer-Verlag, 1995.

[29] A.K. Lenstra, B. Dodson, J. Hughes, W. Kortsmit, and P. Leyland, "Factoring Estimates for 1024-bit RSA Modulus," to be published.

[30] A.K. Lentsra and H.W. Lenstra, Jr., (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Math. **1554**, Springer-Verlag, 1993.

[31] A.K. Lenstra, H.W. Lenstra, Jr., and L. Lovász, "Factoring Polynomials with Rational Coefficients, " Math. Ann. **261**, pp. 515–534, 1982.

[32] A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, and J.M. Pollard, "The Factorization of the Ninth Fermat Number," Math. Comp., **61**, pp. 319–349, 1993.

[33] A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, and J.M. Pollard, "The Number Field Sieve," Proc. of STOC, pp. 564–572, 1990.

[34] A.K. Lenstra and A. Shamir, "Analysis and Optimization of the TWINKLE Factoring Device," *Advances in Cryptology — EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 2000. Proceedings*, B. Preneel (ed.), pp. 35–52, LNCS vol. **1807**, Springer-Verlag, 2000.

[35] A.K. Lenstra, A. Shamir, J. Tomlinson, and E. Tromer, "Analysis of Bernstein's factorization circuit," *Advances in Cryptology — ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1–5, 2002, Proceedings*, Y. Zheng (ed.), LNCS vol. **2501**, pp. 1–26, Springer-Verlag, 2002.

[36] H.W. Lenstra, Jr. and R. Tijdeman, "Computational Methods in Number Theory," Mathematical Centre Tracts **154/155**, Mathematisch Centrum, Amsterdam, 1982.

[37] A.K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes, and P. Leyland, "Factoring Estimates for 1024-bit RSA Modulus, " *Advances in Cryptology — ASIACRYPT2003, 9th International Conference on the Theory, and Application of Cryptology and Information Security, Taipei, Taiwan, November/December 2003, Proceedings*, C.S. Laih (ed.), LNCS vol. **2894**, pp. 55–74, Springer-Verlag, 2003.

[38] A.K. Lenstra and E.R. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology* **14**(4): 255–293, 2001, available via `http://citeseer.nj.net.com/lenstra99selecting.html`.

[39] P.L. Montgomery, "A Block Lanczos Algorithm for Finding Dependencies over GF(2)," *Advances in Cryptology — EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995, Proceedings*, L.C. Guillou and J.-J. Quisquater (eds.), LNCS, vol. **921**, pp. 106–120, Springer-Verlag, 1995.

[40] B. Murphy, *Polynomial Selection for the Number Field Sieve Integer Factoriazation Algorithm*, Ph.D. thesis, Australian National University, 1999.

[41] National Institute of Standards and Technology, "Key Management Guidelines, Part 1: General Guidance (Draft)," Jan. 2003, `http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html`.

[42] NESSIE Consortium, *Portfolio of Recommended Cryptolgraphic Primities*, February 27, 2003, available via `http://www.cryptonessie.org/`.

[43] C. Pomerance, "The Quadratic Sieve Algorithm," LNCS vol. **209**, Springer-Verlag, 1985.

[44] C. Pomerance, J.W. Smith, and R. Tuler, "A Pipeline Architecture for Factoring Large Integers with the Quardratic Sieve Algorithm," *SIAM Journal on Computing*, **17**(2): 387–403, April 1988.

[45] H. te Riele, "Factorization of RSA-140," Posted to Number Theory List, `http://listserv.nodak.edu/scripts/wa.exe?A2=ind9902&L=nmbrthry&F=&S=&P=302`.

[46] RSA Security, *The new RSA Factoring challenge*, web page, Jan. 2003, `http://www.rsasecurity.com/rsalabs/challenges/factoring/`.

[47] M. Schimmler, "Fast Sorting on the Instruction Systolic Array," Report 8709, Christian Albrecht University Kiel, 1987.

[48] A. Shamir, "Factoring Large Numbers with the TWINKLE Device (Extended Abstract)," *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99 Worcester, MA, USA, August 12–13, 1999 Proceedings*, Ç.K.Koç and C. Paar (eds.), LNCS, vol. **1717**, pp. 2–12, Springer-Verlag, 1999.

[49] A. Shamir and E. Tromer, "Factoring Large Numbers with the TWIRL Device," `http://www.wisdom.weizmann.ac.il/~tromer/twirl/`.

[50] R.D. Silverman, "The Multiple Polynomial Quadratic Sieve," Math. Comp., **48**, pp. 243–264, 1987.

[51] R.D. Silverman, "A Cost-cased Security Analysis of Symmetric and Asymmetric Key Length," *Bulletin* 13, RSA Security, 2000, `http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.html`.

[52] Taiwan Semiconductor Manufacturing Company Ltd., *0.13-Micro Technology*, `http://www.tsmc.com/download/enliterature/013_bro_2003.pdf`.

[53] The Earth Simulator, `http://www.es.jamstec.go.jp/`.

[54] P.J. Weinberger and L.P. Rothschild, "Factoring Polynomials over Algebraic Number Fields," ACM Trans. Math. Software **2**, pp. 335–350, 1976.

[55] E. Weiss, *Algebraic Number Theory*, McGraw-Hill, New York, 1963; reprinted, Chelsea, New York, 1976.

[56] D. Wiedemann, "Solving Space Linear Equations over Finite Fields," IEEE Trans. Inform. Theory **32**, pp. 54–62, 1986.

[57] *FactorWorld*, http://www.crypto-world.com/FactorWorld.html.

[58] *Some Number Records*, http://www.loria.fr/~zimmerma/records/.

[59] 青木和麻呂, 植田広樹, 木田祐司, 下山武司, 園田裕貴, 「一般数体篩法実装実験 (1)—概要」, 暗号と情報セキュリティシンポジウム SCIS2004 講演予稿集 2B1-3, 仙台市, 2004.

[60] 河田敬義, 数論 — 古典数論から類体論へ, 岩波書店, 1992.

[61] 木田祐司, 「暗号アルゴリズムの詳細評価に関する報告書」, available at http://www.shiba.tao.go.jp/kenkyu/CRYPTREC/fy15/cryptrec20030424_outrep01.html.

[62] 永田雅宜, 可換環論, 紀伊国屋数学叢書 1, 紀伊国屋書店, 1974.

[63] 都筑俊郎, 有限群と有限幾何, 数学選書, 岩波書店, 1976.

[64] 山崎圭次郎, 環と加群, 岩波基礎数学選書, 岩波書店, 1990.

[65] 「Bernstein 及び A.K.Lenstra らの素因数分解 (行列演算ステップ) 回路に関する調査報告書」, 暗号アルゴリズム及び関連技術の評価報告, 情報処理振興事業協会, 通信・放送機構, avaiable at http://www.shiba.tao.go.jp/kenkyu/CRYPTREC/fy15/cryptrec20030424_outrep01.html.

[66] 暗号技術評価報告書 (2002 年度版), CRYPTREC Report 2002, 情報処理振興事業協会, 通信放送機構, 2003 年 3 月, http://www.shiba.tao.go.jp/kenkyu/CRYPTREC/PDF/c02_report.pdf.

[67] 地球シミュレータと Red Storm, スパコンが今年の主役, http://pcweb.mycom.co.jp/news/2003/08/26/13.html.

付 録A　数体篩法

ここでは数体篩法の仕組みについて説明する. 詳細は [33], [30], [1] などを参照されたい.

A.1 2次篩法

2 次篩法は 1983 年に Pomerance([43]) が考案した素因数分解法で, 数体篩法の原点となっている.

A.1.1 素因数分解の基本方針

合成数 n が与えられたとする. n は奇数で, 平方数でないと仮定する. n の一つの (自明でない) 約数 n_1 に対し, $n_2 = n/n_1$ とおく. すなわち $n = n_1 n_2$ であるとする. ここで一般性を失うことなく, $n_1 > n_2$ と仮定してよい. $x = (n_1 + n_2)/2$, $y = (n_1 - n_2)/2$ とおけば,

$$x^2 - y^2 = (x + y)(x - y) = n_1 n_2 = n \tag{A.1}$$

となる. すなわち合成数 n は必ずある整数 x, y $(x > y)$ を用いて式 (A.1) のように表される (無論, 合成数によっては表現方法は一意的とは限らない).

式 (A.1) より次の式も成り立つ.

$$x^2 \equiv y^2 \bmod n. \tag{A.2}$$

式 (A.2) は, ある整数 $\alpha(> 0)$ に対し,

$$x^2 - y^2 = (x + y)(x - y) = \alpha n$$

が成り立つことを意味する (式 (A.1) は $\alpha = 1$ の場合). 2 次篩法は式 (A.2) を満たす x, y $(x \not\equiv \pm y \bmod n)$ を見つけ, $\gcd(x \pm y, n)$ により n の因子を見つけようという方法である.

ただし, 式 (A.2) を満たす x, y が見つかったとしても $\alpha > 1$ の場合には gcd により n の因子が見つかるとは限らない. しかし, 見つかることも高い確率で期待されることから, より効率よく x, y の組を見つけるための様々な工夫がなされているのが 2 次篩法である.

式 (A.2) を満たす x, y を探すには,

$$x_i \equiv y_i \bmod n, \quad x_i > y_i$$

を満たす x_i, y_i をたくさん集めてそれらを組み合わせて平方数を作ればよい.

多数の組から平方数になるものを見つける効率的方法として知られているのが次の節で述べる篩法である.

A.1.2 篩 (sieve) と線型代数 (linear algebra)

一般に, 多数の整数 r_1, r_2, ..., r_s が与えられたとき, これらの中から組み合わせて平方数となるものを見つける方法として次のものが知られている.

Step 1. 適当な個数の素数の集合 $\{p_1, p_2, \ldots, p_B\}$ を決める. これを factor base という.

Step 2. (篩) $r_1, \ldots, r_s$ のうち, factor base により完全に素因数分解できるもの (これを smooth な数という), すなわち

$$r_i = \prod_{j=1}^{B} p_j^{e(i,j)}$$

と表されるものを $B+1$ 個以上探す. これらを改めて $r_0, \ldots r_B$ とおく.

Step 3. 各 r_i に対し, これらのべき指数 $e(i,j)$ を用いて 次のような $\mathbb{Z}/2\mathbb{Z}$ 上の B 次元ベクトルを作る.

$$v_i = (\ e(i,1) \bmod 2,\ e(i,2) \bmod 2, \cdots, e(i,B) \bmod 2\ ).$$

Step 4. (線型代数) B 次元ベクトル $v_0, v_1, \ldots, v_B$ に対し, 一次従属関係式を作る.

$$c_0 v_0 + c_1 v_1 + \cdots + c_B v_B \equiv 0 \bmod 2. \quad (c_i = 0 \text{ or } 1)$$

Step 5. このとき次は平方数となる.

$$r_0^{c_0} r_1^{c_1} \cdots r_B^{c_B}.$$

Step 4 において, B 次元ベクトルが $B+1$ 個あるので必ず一次従属となる. 右辺の 0 は 0 ベクトル $= (0, \cdots, 0)$ を表すことに注意. また, Step 2 で各 r_i が smooth であるか否かを確かめなければならないため, r_i の大きさは小さいほうが好ましいことがわかる.

A.1.3　2 次篩法

2 次篩では x_i, y_i を集めるために次のような方法を用いる.

$$m = \lfloor \sqrt{n} \rfloor$$

とおいて 2 次多項式

$$Q(X) = (X+m)^2 - n$$

を考える. X に比較的小さい数 $a = 0, \pm 1, \pm 2, \ldots$ を代入すると $Q(a) \sim 2am \sim 2a\sqrt{n}$ となり, n と比較すれば小さい数であるといえる.

A.1.2 節による方法で, このような $Q(a)$ のうち, ある factor base に対し smooth な数となるものを集め, 組み合わせで平方数となるもの, すなわち $Q(a_1), \ldots, Q(a_t)$ であって smooth かつ $Q(a_1) \cdots Q(a_t) = Y^2$: 平方数 となるものを見つければ, $X = (a_1+m)(a_2+m)\cdots(a_t+m)$ と置いて

$$X^2 \equiv Y^2 \bmod n$$

が成り立つことになる. あとは gcd により n と $X \pm Y$ の共通因子を見つければよい.

2 次篩法の手順を以下にまとめる.

Step 1. パラメータ t, B (ともに正整数, $t > B$) を定め, factor base $\{p_1, p_2, \ldots, p_B\}$ を決める.

Step 2. $-t \le a \le t$ に対し $r_a = Q(a)$ を計算する.

Step 3. $r_{-t}, \ldots, r_0, r_1, \ldots, r_t$ のうち, factor base により完全に素因数分解できるもの, すなわち

$$r_i = \prod_{j=1}^{B} p_j^{e(i,j)}$$

と表されるものを $B+1$ 個以上探す. これらを改めて $r_0, \ldots r_B$ とおく. また, 各 r_i に対応する a の値を $a_0, \ldots, a_B$ とおく.

Step 4. 各 r_i に対し, これらのべき指数 $e(i,j)$ を用いて 次のような $\mathbb{Z}/2\mathbb{Z}$ 上の B 次元ベクトルを作る.

$$v_i = (\ e(i,1) \bmod 2,\ e(i,2) \bmod 2, \cdots, e(i,B) \bmod 2\ ).$$

Step 4. B 次元ベクトル $v_0, v_1, \ldots, v_B$ に対し, 一次従属関係式を作る.

$$c_0 v_0 + c_1 v_1 + \cdots + c_B v_B \equiv 0 \bmod 2. \quad (c_i = 0 \text{ or } 1)$$

Step 5. このとき次は平方数となる.

$$r_0^{c_0} r_1^{c_1} \cdots r_B^{c_B} = Y^2.$$

一方, $X = (a_0 + m)^{c_0}(a_1 + m)^{c_1} \cdots (a_B + m)^{c_B}$ とおく.

Step 6. $\gcd(X \pm Y, n)$ で n の非自明な因子であるものがあればそれを出力して終了. なければ **fail** を出力して終了.

A.1.4 原理

対象の合成数を n とする. 2 次篩法において, 素因数分解の基本原理となっているのは次のことである.

(0) factor base や検索などの upper bound を定める.
(1) 定めた factor base に対して"smooth"な x_i, y_i $(x_i \neq y_i)$ であって,

$$x_i \equiv y_i \bmod n$$

を満たすものを factor base の数より多く集める.
(2) 各 x_i の factor base による素因数分解におけるべき指数について, 2 を法とした 1 次従属関係を見つける. y_i についても同様に行う.
(3) (2) の従属関係から

$$x^2 \equiv y^2 \bmod n, \quad x \neq y$$

を構成し, $\gcd(x \pm y, n)$ により n の因子を見つける.

2 次篩法では上記ステップ (2) において, 多項式 $Q(X) = (X + m)^2 - n$ を用いて比較的小さい a に対し, $x_i = (a + m)^2$, $y_i = Q(a)$ とした.

数体篩法ではさらに代数的数体の性質を用いて (1) のステップをより効率よく行う方法を提供する.

A.2 数体篩法

A.2.1 数学的準備

ここでは数体篩法を説明するために必要な代数的整数論などの準備を行う.

代数的整数環

有限次代数体 $K/\mathbb{Q}$ を考える[1]. K における $\mathbb{Z}$ の整閉方を $\mathcal{O}_K$ とおく:

$$\mathcal{O}_K = \{a \in K \mid g(a) = 0 \text{ for some } g(x) \in \mathbb{Z}[x],\ \text{monic}\}.$$

$\mathcal{O}_K$ を K の**代数的整数環**という. これは有理数体 $\mathbb{Q}$ における有理整数環 $\mathbb{Z}$ の一般化であり, $\mathcal{O}_\mathbb{Q} = \mathbb{Z}$ が成り立つ. 代数的整数環においては零因子は存在しない. すなわち $ab = 0$ ならば $a = 0$ または $b = 0$ が成り立つ (一般にこのような性質を持つ (可換) 環を**整域**という).

素元分解

定義から $1 \in \mathcal{O}_K$ である. $e \in \mathcal{O}_K$ に対し, $ee' = 1$ となる $e' \in \mathcal{O}_K$ が存在するとき e(よって e' も) は**単元**であるという. $a, b, c \in \mathcal{O}_K$ が $a = bc$ を満たすとき, b, c は a の**約元**といい, $b|a$, $c|a$ とかく. $p \in \mathcal{O}_K$ が**素元**であるとは, p は単元でなく, $a, b \in \mathcal{O}_K$ に対し, $p|ab$ ならば $p|a$, または $p|b$ が成り立つことをいう. 素元は $\mathbb{Z}$ における素数の一般化である.

一般に整域 R において, 0 でない任意の元 $a \in R$ が有限個の素元の積 $a = p_1 p_2 \cdots p_r$

[1]体 K の部分集合 F が 2 つ以上の元を持ち, K の演算に関して体をなすとき, F を K の**部分体**, K は F の**拡大体**といい, K/F と書く. $F \subseteq K' \subseteq K$ なる K の部分体 K' を, K/F の**中間体**という. 真の部分体を持たない体を**素体**という. 素体 K は 有理数体 $\mathbb{Q}$, またはある素数 p による有理整数環 $\mathbb{Z}$ の剰余環 $\mathbb{Z}/p\mathbb{Z}$ と同型であることが知られている. 体 K に含まれる最小の部分体 K_0 は素体であり, $K_0 \cong \mathbb{Q}$ のとき K の**標数は 0**, $K_0 \cong \mathbb{Z}/p\mathbb{Z}$ のとき K の**標数は** p であるという (標数が $p > 0$ のとき, p は $n \circ 1 = 1 + \cdots + 1 = 0$ となる最小の正整数である).

体の拡大 K/F において, K の F 上のベクトル空間としての次元を**拡大次数**といい, $[K : F]$ であらわす. 有限次元のとき, **有限次拡大**という. $a \in K$ に対し, F 上の 0 でない多項式 $f(x) \in F[x]$ が存在して $f(a) = 0$ となるとき, a は F 上**代数的**であるといい, K のすべての元が F 上代数的であるとき, K/F を**代数拡大**という. 有限次拡大はすべて代数拡大となることが知られている.

$\mathbb{Q}$ 係数の 1 変数多項式は複素数体 $\mathbb{C}$ 上で 1 次の多項式の積に分解する. そこで特に断わらない限り, $\mathbb{Q}$ 上の代数拡大は全て $\mathbb{C}$ の部分体として考えることとする.

K/F を体の拡大とする. $a_1, a_2, \ldots \in K$ に対し, a_1, a_2, ... と F を含む K の最小の部分体を $F(a_1, a_2, \ldots)$ と書き, F に a_1, a_2, ... を**付加**して得られる体という. 特に定数でない多項式 $f(x) \in F[x]$ の全ての根を F に付加して得られる体を f の F 上の**最小分解体**という.

であらわされるとき, R を素元分解整域という. このとき $a = p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$ であれば, $r = s$ であり, 適当に番号を付け替えて $q_i = e_i p_i$, $e_i \in R$: 単元 が成り立つ.

有理整数環 $\mathbb{Z}$ は素元分解整域で, 素元分解は整数の素因数分解に他ならない. しかし, $\mathbb{Z}$ の一般化である代数的整数環 $\mathcal{O}_K$ は, 一般には素元分解整域とは限らない.

しかしながら, 次に述べるイデアルの概念を導入すると類似の分解が成立する.

イデアルと素イデアル分解

整域 R において, R の部分集合 I が R の**イデアル** (ideal) であるとは, I は R の部分加群 (R の加法群に関して部分群) でありかつ任意の $a \in R$ に対して $aI \subseteq I$ が成り立つことをいう. 零元のみからなる集合 $\{0\}$ や R 全体はイデアルである. これを自明なイデアルという. $a \in R$ に対し, $aR = \{ab | b \in R\}$ は R のイデアルとなる. このような形のイデアルを**単項イデアル**といい, $(a) = aR$ などとかく.

R のイデアル $\mathfrak{p}$ が**素イデアル**であるとは, $a, b \in R$, $ab \in \mathfrak{p}$ ならば $a \in \mathfrak{p}$ または $b \in \mathfrak{p}$ が成り立つことをいう. さらに $\mathfrak{p} (\neq R)$ が**極大イデアル**であるとは, $\mathfrak{p}$ を含むイデアルは自身または R しかない, すなわち $\mathfrak{p} \subseteq \mathfrak{p}' \subseteq R$ ならば $\mathfrak{p}' = \mathfrak{p}$, または $\mathfrak{p}' = R$ が成り立つことをいう.

R のイデアル $\mathfrak{a}$ に対して, R の $\mathfrak{a}$ を法とする剰余類 $a + \mathfrak{a}$ $(a \in R)$ の全体は演算

$$(a + \mathfrak{a}) + (b + \mathfrak{a}) = (a + b) + \mathfrak{a}, \quad (a + \mathfrak{a})(b + \mathfrak{a}) = (ab) + \mathfrak{a}$$

に関して可換環 $R/\mathfrak{a}$ を作る. これをイデアル $\mathfrak{a}$ に関する**剰余環**という. 例えば整数 $n \in \mathbb{Z}$ に対し, $(n) = n\mathbb{Z}$ は $\mathbb{Z}$ の (単項) イデアルであり, これに関する剰余環を $\mathbb{Z}/n\mathbb{Z}$ などとかく.

剰余環 $R/\mathfrak{a}$ が整域となるための必要十分条件は $\mathfrak{a}$ が素イデアルであること, また $R/\mathfrak{a}$ が体となるための必要十分条件は $\mathfrak{a}$ が極大イデアルであることなどが知られている.

イデアル $\mathfrak{a}$ と $\mathfrak{b}$ の積 $\mathfrak{ab}$ が定義され, $\mathfrak{ab}$ もまた R のイデアルとなる:

$$\mathfrak{ab} = \{ab | a \in \mathfrak{a},\ b \in \mathfrak{b}\}.$$

代数的整数環 $\mathcal{O}_K$ においては, 任意のイデアル $\mathfrak{a}$ は素イデアルの積としてあらわ

される:

$$\mathfrak{a} = \mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_r.$$

また, $\mathfrak{a}$ が 2 通りにあらわされたとする: $\mathfrak{a} = \mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_r = \mathfrak{q}_1\mathfrak{q}_2\cdots\mathfrak{q}_s$. このとき $r = s$ であり, 適当に順序を並べ替えると $\mathfrak{p}_i = \mathfrak{q}_i$ が成り立つ.

代数的整数環 $\mathcal{O}_K$ において, 素イデアルは常に極大イデアルとなる[2].

さらに, 一般に体 k(を環とみなしたとき) は自明なイデアルしか持たない[3].

ノルム

拡大体の元 $a \in K$ に対し, 多項式 $h(x) \in \mathbb{Q}[x]$ であって, $\mathbb{Q}$ 上既約かつ monic, $h(a) = 0$ を満たすものが唯一存在する. これを a の ($\mathbb{Q}$ 上の) **最小多項式**という.

a の最小多項式が $h(x) = x^d + a_{d-1}x^{d-1} + \cdots + a_1x + a_0$ $(a_i \in \mathbb{Q})$ のとき,

$$N(a) := (-1)^d a_0$$

を a の ($\mathbb{Q}$ 上の) **ノルム**という[4]. 元のノルムは乗法的である. すなわち任意の元 $a, b \in K$ に対し, $N(ab) = N(a)N(b)$ が成り立つ.

$a \in K$ の最小多項式が $h(x) = \prod(x - a_i)$, $a_1 = a$ のとき, $s, t \in \mathbb{Z}$ に対し, $s + ta$ の最小多項式は $\prod(x - (s + ta_i))$ である. よって $s + ta$ のノルムは $(-1)^d\prod(s + ta_i) = (-1)^d(-t)^d\prod(-s/t - a_i) = t^d h(-s/t)$ となる:

$$N(s + ta) = t^d h(-s/t). \tag{A.3}$$

K の代数的整数環 $\mathcal{O}_K$ のイデアル $\mathfrak{a}$ に対し, 剰余環 $\mathcal{O}_K/\mathfrak{a}$ は有限環となり, その元の個数

$$N(\mathfrak{a}) = \#\mathcal{O}_K/\mathfrak{a}$$

をイデアル $\mathfrak{a}$ の**ノルム**という. イデアルのノルムも乗法的である. すなわち任意のイデアル $\mathfrak{a}, \mathfrak{b}$ に対し, $N(\mathfrak{ab}) = N(\mathfrak{a})N(\mathfrak{b})$ が成り立つ.

---

[2]代数的整数環は Dedekind 整域である.

[3]$\mathfrak{a}$ を体 k のイデアルとする. $\mathfrak{a} \neq \{0\}$ とすると $a(\neq 0) \in \mathfrak{a}$ が存在し, 任意の $b \in k$ に対し, $b = b \cdot (a/a) = (b/a) \cdot a \in \mathfrak{a}$. 故に $\mathfrak{a} = k$.

[4]元のノルムは $\mathbb{Q}$ 上共役な元の積としても定義される. また拡大 $K/\mathbb{Q}$ の (ある基底に関する) 正則表現 A における $A(a)$ の行列式 $\det A(a)$ で定義することもある. これらは全て一致する.

$\mathfrak{a}$ が元 a で生成される単項イデアル ($\mathfrak{a} = (a)$) であるとき, 元のノルムとイデアルのノルムは次の関係を持つ.

$$N(\mathfrak{a}) = |N(a)|.$$

環準同型

一般に環 R_1, R_2 に対し, 写像 $\phi : R_1 \to R_2$ が環準同型であるとは, 任意の $a, b \in R_1$ に対して,
(1) $\phi(a+b) = \phi(a) + \phi(b)$,
(2) $\phi(ab) = \phi(a)\phi(b)$
が成り立つことをいう. ϕ が全単射であるとき, ϕ は**同型写像**であるといい, (集合としての) 逆写像 ϕ^{-1} も環の同型写像となる. 同型写像が存在するとき R_1, R_2 は**同型**であるという.

準同型 $\phi : R_1 \to R_2$ に対し,

$$\mathrm{Ker}(\phi) = \{a \in R_1 \mid \phi(a) = 0\}$$

とおくと, $\mathrm{Ker}(\phi)$ は R_1 のイデアルとなる. これを ϕ の**核** (kernel) と呼ぶ. ϕ が単射であるための必要十分条件は $\mathrm{Ker}(\phi) = \{0\}$ である.

定理 A.2.1. (**準同型定理**) $\phi : R_1 \to R_2$ は全射であるとする. このとき ϕ は次の環同型を導く.

$$R_1/\mathrm{Ker}(\phi) \xrightarrow{\sim} R_2.$$

体 k_1, k_2 と環準同型 $\phi : k_1 \to k_2$ において, 準同型定理より $k_1/\mathrm{Ker}(\phi) \cong \mathrm{Im}(\phi) \subseteq k_2$ (Im は写像の像) であるが, 体は自明なイデアルしか持たないので $\mathrm{Ker}(\phi) = \{0\}$ or k_1 が成り立つ. したがって ϕ は 0 写像 (全てを 0 にうつす)($\mathrm{Ker}(\phi) = k_1$) か, 単射 ($\mathrm{Ker}(\phi) = \{0\}$) となる.

有限次代数体

K, K' を $\mathbb{Q}$ の有限次拡大体とする. K から K' への (環としての) 準同型写像 ϕ で, $\mathbb{Q}$ の元を動かさない (任意の $x \in \mathbb{Q}$ に対し, $\phi(x) = x$) ものを K から K' への

$\mathbb{Q}$ **上の準同型写像**という. 体の準同型は単射なので, (K' の) 中への同型写像ともいう. ϕ が同型ならば (すなわち全射ならば) $\mathbb{Q}$ **上の同型写像**という.

次の条件が成り立つとき, K と K' は $\mathbb{Q}$ 上, 互いに**共役**である (または, K' は K の共役体) という.
(1) ある体 L が存在して K, K' を部分体として含む.
(2) K と K' は $\mathbb{Q}$ 上同型である.

先に述べたように, $\mathbb{Q}$ の有限次拡大 K は複素数体 $\mathbb{C}$ の部分体と考える. このとき, K の ($\mathbb{C}$ 内の)$\mathbb{Q}$ 上の共役体は $[K:\mathbb{Q}]=n$ とするとき n 個 : $K^{(1)}=K, K^{(2)}, \cdots, K^{(n)}$ 存在する. これらのうち, 実体 (すなわち実数体 $\mathbb{R}$ の部分体) であるものが r_1 個とすると, $n-r_1$ は偶数となり, これを $2r_2$ とおく.

K の代数的整数環 $\mathcal{O}_K$ の元で, 可逆なもののなす乗法群を K の**単数群**といい, $\mathcal{O}_K^\times$ とかく. 単数群の構造に対して次の定理は重要である ([60] 9.2 節など).

定理 A.2.2. (**Dirichlet の単数定理**) K の単数群 $\mathcal{O}_K^\times$ は r_1+r_2 個の生成元

$$\rho, \mu_1, \ldots, \mu_{r_1+r_2-1}$$

を持ち, ρ は乗法的位数が有限, 他は位数無限で一次独立となる.

K を $\mathbb{Q}$ の有限次拡大体とする. K の代数的整数環 $\mathcal{O}_K$ は $\mathbb{Z}$ 加群として自由であり, 階数は拡大次数と一致する. すなわち, ある基底 $\omega_1, \ldots, \omega_n$ ($n=[K:\mathbb{Q}]$) が存在して

$$\mathcal{O}_K = \mathbb{Z}\omega_1 + \cdots + \mathbb{Z}\omega_n \quad (\text{直和})$$

となる.

ω_i の $\mathbb{Q}$ 上の共役元を $\omega_i^{(j)}$ とする (すなわち同型 $K \to K^{(j)}$ による ω_i の像).

$$\Delta(\omega_1, \cdots, \omega_n) = \begin{vmatrix} \omega_1^{(1)} & \omega_2^{(1)} & \cdots & \omega_n^{(1)} \\ \omega_1^{(2)} & \omega_2^{(2)} & \cdots & \omega_n^{(2)} \\ & \cdots & & \\ \omega_1^{(n)} & \omega_2^{(n)} & \cdots & \omega_n^{(n)} \end{vmatrix}$$

とおく. ただし, $|A|$ は行列 A の行列式を表す. このとき,

$$\Delta_K = \Delta(\omega_1, \cdots, \omega_n)^2$$

は基底のとり方によらず, K により一意に定まる. また $\Delta_K(\neq 0) \in \mathbb{Z}$ となる. これを K の**判別式**という.

以下, やや一般に F を標数 0 の体かまたは有限体であるとする.

体の拡大 K/F において, K の F 上の共役体がすべて K と一致するとき, K を F の**正規拡大体**という. 代数拡大 K/F が正規拡大であるとき, K/F は **Galois 拡大**であるという[5]. 有限次か無限次であるかによって, 有限次 Galois 拡大, 無限次 Galois 拡大であるという.

K/F が Galois 拡大のとき,

$$\mathrm{Gal}(K/F) = \{\sigma \mid \sigma : K \text{ の } F \text{ 上の自己同型 }\}$$

とおくと, $\mathrm{Gal}(K/F)$ は写像の合成に関して群をなす (単位元は恒等写像). これを K/F の **Galois 群**という[6]. Galois 群が Abel 群 (= 可換群), 巡回群のとき, K/F をそれぞれ **Abel 拡大**, **巡回拡大**という.

F が q 個の元からなる有限体で, K/F を f 次拡大とすると, K は q^f 個のからなる

[5]一般には, 分離的かつ正規拡大であるとき Galois 拡大であるという. しかし, F の標数が 0 または有限体であるとき, 任意の代数拡大は分離的であるので, ここでは正規拡大だけの条件でよい.

[6]Galois 拡大について, 次が成り立つことが知られている.
(1) K/F が有限次 Galois 拡大のとき, $[K:F] = \#\mathrm{Gal}(K/F)$.
(2) K' を Galois 拡大 K/F の中間体とすると K/K' も Galois 拡大である. (K'/F は必ずしも Galois 拡大とはならない.)
(3) K/F が Galois 拡大のとき, $\Phi(K, \mathrm{Gal}(K/F)) = \{a \in K \mid \sigma(a) = a \text{ for any } \sigma \in \mathrm{Gal}(K/F)\}$ とおくと, $\Phi(K, \mathrm{Gal}(K/F)) = F$.

K/F を有限次 Galois 拡大とする. $\mathrm{Gal}(K/F)$ の部分群 H, K/F の中間体 K' に対して, $\Phi(H)$, $\Gamma(K')$ を次で定義する.

$$\Phi(H) = \{a \in K \mid \sigma(a) = a \text{ for any } \sigma \in H\},$$
$$\Gamma(K') = \{\sigma \in \mathrm{Gal}(K/F) \mid \sigma(a) = a \text{ for any } a \in K'\}.$$

Theorem (Galois の基本定理) $\Phi(H)$ は K/F の中間体, $\Gamma(K')$ は $\mathrm{Gal}(K/F)$ の部分群であり,

$$\Gamma(\Phi(H)) = H, \quad \Phi(\Gamma(K')) = K'$$

が成り立つ. よって, K/F の中間体と, $\mathrm{Gal}(K/F)$ の部分群の間に互いに逆となる, 1 対 1 の対応

$$K' \to \Gamma(K'), \quad H \to \Phi(H)$$

が存在する. さらにこの対応により, $K_i' \leftrightarrow H_i$ $(i = 1, 2)$ であるとき, $K_1' \subseteq K_2' \Leftrightarrow H_1 \supseteq H_2$.

K/F が無限次 Galois 拡大のとき, Galois 群に適当な位相 (Krull 位相) を入れ, 位相群とみたとき, K/F の中間体と, Galois 群の閉部分群とが 1 対 1 対応することが知られているが, ここではこれ以上立ち入らない.

有限体であり, K/F は巡回拡大となる. その Galois 群は $\sigma : K \to K$, $a \mapsto a^q$ で生成される位数 f の巡回群である.

再び K を $\mathbb{Q}$ の n 次拡大とする.

$p \in \mathbb{Z}$ を素数とし, $\mathcal{O}_K$ におけるイデアル $(p) = p\mathcal{O}_K$ が $\mathcal{O}_K$ において

$$(p) = \mathfrak{p}_1^{e_1}\mathfrak{p}_2^{e_2}\cdots\mathfrak{p}_g^{e_g}$$

と素イデアル分解されたとする. このとき次の関係式が成り立つ.

$$n = e_1 f_1 + \cdots + e_g f_g, \quad N(\mathfrak{p}_i) = p^{f_i}.$$

ある e_i が 1 より真に大きいとき ($e_i > 1$), 素数 p は拡大 $K/\mathbb{Q}$ において**分岐**するという. e_i を**分岐指数**, f_i を**相対次数**と呼ぶ. 相対次数は拡大次数 $[(\mathcal{O}_K/\mathfrak{p}_i) : (\mathbb{Z}/p\mathbb{Z})]$ に他ならない.

素数 p が $K/\mathbb{Q}$ で分岐するための必要十分条件は p が判別式 Δ_K を割ることである (**Dedekind の判別定理**). 特に分岐する素数は有限個である.

$K/\mathbb{Q}$ が Galois 拡大のときは,

$$(p) = (\mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_g)^e, \quad N(\mathfrak{p}_i) = p^f, \quad n = efg$$

が成り立つ (すなわち各分岐指数は一致し, また各相対次数も一致する).

G を $K/\mathbb{Q}$ の Galois 群とする. 素数 p とその $\mathcal{O}_K$ での分解に現れる素イデアル $\mathfrak{p}_i$ について, 剰余体の拡大 $(\mathcal{O}_K/\mathfrak{p}_i)/(\mathbb{Z}/p\mathbb{Z})$ は有限体の拡大だから巡回拡大で, $a \mapsto a^p$ ($a \in \mathcal{O}_K/\mathfrak{p}_i$) で生成される. p が $K/\mathbb{Q}$ において分岐しないとき, Galois 群 G の元 $\sigma \in G$ であって, 任意の $\alpha \in \mathcal{O}_K$ に対し,

$$\sigma(\alpha) \equiv \alpha^p \bmod \mathfrak{p}_i$$

が一意に存在することがわかる[7]([26], [60]). これを $\mathfrak{p}_i$ の **Frobenius 自己同型**とよび

$$\sigma = (\mathfrak{p}_i, K/\mathbb{Q})$$

などとかく.

[7] p が分岐する場合には G の, ($\mathfrak{p}_i$ に依存して決まる) ある部分群 (惰性群) の分だけ曖昧さが残る.

$f(x) \in \mathbb{Q}[x]$ を定数でない 1 変数 d 次 monic 既約多項式とし, K を f の最小分解体とする. このとき, $K/\mathbb{Q}$ は Galois 拡大で, その Galois 群 G は f の Galois 群と呼ばれる. G は $f(x) = 0$ の d 個の根の集合上に作用し (d 個の文字の置換を与える), 従って d 次対称群 S_d の部分群とみなすことができる.

素数

ここでは素数についての重要な結果をいくつか紹介する.

正の実数 x を超えない素数の個数を $\pi(x)$ とおく. このとき次が成り立つ.

定理 A.2.3. (**素数定理** J.Hadamard, C.de la Vallée-Poussin, 1896)

$$\pi(x) \ \sim \ \frac{x}{\log x}.$$

正の実数 x を超えず, かつ, 相異なる k 個の素数の積としてあらわされる正整数の個数を $\pi_k(x)$ とおく. このとき, 素数定理の拡張として次が得られる.

定理 A.2.4. (E.Landau, 1911)

$$\pi_k(x) \ \sim \ \frac{1}{(k-1)!}\frac{x(\log\log x)^{k-1}}{\log x}.$$

M を素数全体の集合の部分集合とする. このとき極限

$$\lim_{s\to 1+} \frac{\displaystyle\sum_{p\in M}\frac{1}{p^s}}{\log\dfrac{1}{s-1}}$$

が存在するならば, これを M の (Dirichlet) **密度**という[8]. これについて, 次の重要な定理が知られている.

[8] $\lim_{n\to\infty} \dfrac{\#\{p \in M \mid p \le n\}}{\#\{p : \text{prime} \mid p \le n\}}$ が存在するとき Dirichlet 密度も存在して両者は一致する (逆は必ずしも成り立たない).

定理 A.2.5. (**Čebotarev の密度定理**) $K/\mathbb{Q}$ を n 次 Galois 拡大とし, その Galois 群を G とする. $\sigma \in G$ に対し, G における σ の共役類に属す元の個数を c とおく. M を, 素数 p であって, K において分岐せず, さらに p の上にある素イデアル $\mathfrak{p}$ が存在して

$$\sigma = (\mathfrak{p}, K/\mathbb{Q})$$

となるもののなす集合とする. このとき M の密度は存在して c/n に等しい[9].

整環

$K/\mathbb{Q}$ を有限次拡大とする. $\mathcal{O}$ が K の**整環 (order)** であるとは, $\mathcal{O}$ は K の部分環で, 加群として $\mathbb{Z}$ 上有限生成, かつ $\mathcal{O} \otimes \mathcal{Q} = K$ が成り立つことをいう. この条件から整環は代数的整数環 $\mathcal{O}_K$ の部分環であって, 加法群として指数有限: $[\mathcal{O}_K : \mathcal{O}] < \infty$ であることが導かれる. すべての整環を含む最大の整環という意味で, $\mathcal{O}_K$ は**最大整環 (maximal order)** ともいう.

$x(\neq 0) \in \mathcal{O}$ に対し,

$$\#\mathcal{O}/x\mathcal{O} = |N(x)|(= N((x)) = \#\mathcal{O}_K/x\mathcal{O}_K)$$

が成り立つ[10]. $\mathfrak{a} \subset \mathcal{O}$ を 0 でない任意のイデアルとする. $x(\neq 0) \in \mathfrak{a}$ に対し, $(x) \subseteq \mathfrak{a}$ であり, 自然な写像 $A/(x) \to A/\mathfrak{a}$ は全射となるので, $\#A/\mathfrak{a}$ は有限である. これを整環 $\mathcal{O}$ のイデアル $\mathfrak{a}$ の**ノルム**といい, $\mathcal{N}(\mathfrak{a})$ とかく. 単項イデアル $\mathfrak{a} = (x)$ に対しては定義から明らかに $\mathcal{N}((x)) = |N(x)|$ が成り立つ.

$\mathfrak{p} \subset \mathcal{O}$ を 0 でない素イデアルとする. $\mathcal{O}/\mathfrak{p}$ は有限整域であるから有限体となる. すなわち, $\mathfrak{p}$ は極大イデアルで, 有理素数 p を唯一つ含む. このとき剰余体 $\mathcal{O}/\mathfrak{p}$ は素体 $\mathbb{F}_p$ を含み, その拡大次数 $[\mathcal{O}/\mathfrak{p} : \mathbb{F}_p]$ を $\mathfrak{p}$ の**次数**という.

$x(\neq 0) \in \mathcal{O}$ に対し, $(x) = x\mathcal{O}$ は $\mathcal{O}$ で指数有限であることから, イデアルの有限列

$$\mathcal{O} = \mathfrak{a}_0 \supset \mathfrak{a}_1 \supset \cdots \supset \mathfrak{a}_{t-1} \supset \mathfrak{a}_t = x\mathcal{O}$$

---

[9] この定理はより一般的な状況下で証明されている.

[10] 正則表現における $\mathbb{Q}$ 基底として $\mathcal{O}_K$ の $\mathbb{Z}$ 基底 $\{\omega_i\}$ を取ったものと, $\mathcal{O}$ の $\mathbb{Z}$ 基底 $\{\omega_i'\}$ を取ったものとを比較する.

$x(\neq 0) \in K$ に対し, $|N(x)| = |\det((x\omega_i)^{(j)})/\det(\omega_i^{(j)})|$ ($z^{(j)}$ は z の $\mathbb{Q}$ 上の共役元) であるが, ある整係数行列 B に対し $(\omega_i') = B(\omega_i)$ のとき, $\#\mathcal{O}/x\mathcal{O} = \det((x\omega_i')^{(j)})/\det(\omega_i'^{(j)})| = |\det B \det((x\omega_i)^{(j)})/\det B \det(\omega_i^{(j)})| = |N(x)|$ となる.

であって, $\mathfrak{a}_{i-1}$ と $\mathfrak{a}_i$ $(1 \leq i \leq t)$ の間には (非自明な包含関係のある) イデアルが存在しないようなものが存在する.

0 でない任意の素イデアル $\mathfrak{p} \subset \mathcal{O}$ に対し, $l_{\mathfrak{p}}(x)$ を, $i \in \{1, 2, \ldots, t\}$ のうち, $\mathfrak{a}_{i-1}/\mathfrak{a}_i$ が $\mathcal{O}$-加群として $\mathcal{O}/\mathfrak{p}$ と同型であるものの個数とおく:

$$l_{\mathfrak{p}}(x) := \#\left\{i \in \{1, 2, \ldots, t\} \mid \mathfrak{a}_{i-1}/\mathfrak{a}_i \cong \mathcal{O}/\mathfrak{p}\right\}.$$

$\mathcal{O}$ の $\mathfrak{p}$ での局所化を $\mathcal{O}_{\mathfrak{p}}$[11] とするとき, 明らかに $l_{\mathfrak{p}}(x)$ は $\mathcal{O}_{\mathfrak{p}}$ 加群 $\mathcal{O}_{\mathfrak{p}}/x\mathcal{O}_{\mathfrak{p}}$ の長さ[12] と一致する:

$$l_{\mathfrak{p}}(x) = \mathrm{length}_{\mathcal{O}_{\mathfrak{p}}} \mathcal{O}_{\mathfrak{p}}/x\mathcal{O}_{\mathfrak{p}}. \tag{A.4}$$

また Jordan-Hölder の定理から $l_{\mathfrak{p}}(x)$ はイデアル列の取り方によらず well-defined で, さらに次が成り立つ[13]. ただし, 左辺の積は素イデアル全体にわたる.

$$\prod_{\mathfrak{p}} \mathfrak{p}^{l_{\mathfrak{p}}(x)} \subseteq x\mathcal{O}. \tag{A.5}$$

[11] 積閉集合 $\mathcal{O} \setminus \mathfrak{p}$ による商環. $\mathcal{O}_{\mathfrak{p}} = \{a/b \mid a, b \in \mathcal{O}, b \notin \mathfrak{p}\}$.

[12] 部分加群列の長さの最大値. 組成列の長さ.

[13] 定義より明らかに $x \notin \mathfrak{p}$ ならば $l_{\mathfrak{p}}(x) = 0$ (命題 A.2.1 (2)). よって $\prod_{\mathfrak{p}} \mathfrak{p}^{l_{\mathfrak{p}}(x)} = \prod_{\mathfrak{p} \ni x} \mathfrak{p}^{l_{\mathfrak{p}}(x)}$.
$x \in \mathfrak{p}$ とする. $\mathfrak{P} = \mathfrak{p}\mathcal{O}_{\mathfrak{p}}$ とおき, 列

$$\mathcal{O}_{\mathfrak{p}}/x\mathcal{O}_{\mathfrak{p}} \supset (\mathfrak{P} + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} \supset (\mathfrak{P}^2 + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} \supset \cdots \supset (\mathfrak{P}^i + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} \supset \cdots$$

を考える. ある s に対して $(\mathfrak{P}^s + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} = (\mathfrak{P}^{s+1} + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}}$ であることと, $\mathfrak{P}^s \subseteq \mathfrak{P}^{s+1} + x\mathcal{O}_{\mathfrak{p}}$ であることは同値である.
このとき $\mathfrak{P}^{s+1} \subseteq \mathfrak{P}^{s+2} + x\mathfrak{P} \subset \mathfrak{P}^{s+2} + x\mathcal{O}_{\mathfrak{p}}$, よって $(\mathfrak{P}^s + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} = (\mathfrak{P}^{s+2} + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}}$ となり, 従って $\mathfrak{P}^s \subseteq \mathfrak{P}^{s+2} + x\mathcal{O}_{\mathfrak{p}}$. これを繰り返せば

$$\mathfrak{P}^s \subset \bigcap_{j=s+1}^{\infty} (\mathfrak{P}^j + x\mathcal{O}_{\mathfrak{p}}).$$

右辺は $\mathcal{O}_{\mathfrak{p}}$ が Noether 局所環であることから $x\mathcal{O}_{\mathfrak{p}}$ と一致することがわかる ([62] 定理 5.0.6). すなわち $\mathfrak{P}^s \subset x\mathcal{O}_{\mathfrak{p}}$. 故に $(\mathfrak{P}^s + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} = x\mathcal{O}_{\mathfrak{p}}/x\mathcal{O}_{\mathfrak{p}} = \{0\}$. s を $(\mathfrak{P}^s + x\mathcal{O}_{\mathfrak{p}})/x\mathcal{O}_{\mathfrak{p}} = \{0\}$ となる最小の整数とすれば, $l_{\mathfrak{p}}$ の定義より, $m = l_{\mathfrak{p}}(x) \geq s$ でなければならない. このとき, $\mathfrak{P}^m \subseteq \mathfrak{P}^s \subseteq x\mathcal{O}_{\mathfrak{p}}$. $\mathfrak{P}^m \cap \mathcal{O} = \mathfrak{p}^m$ ([64] 定理 6.20) であるから, x を含む素イデアル $\mathfrak{p}$ 全体について共通部分をとれば,

$$\prod_{\mathfrak{p} \ni x} \mathfrak{p}^{l_{\mathfrak{p}}(x)} \subset \bigcap_{\mathfrak{p} \ni x} (x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O})$$

であることがわかる ($\mathfrak{p}$, $\mathfrak{p}'$ が異なる素イデアルのとき, $\mathfrak{p}^i \cap \mathfrak{p}'^j = \mathfrak{p}^i\mathfrak{p}'^j$ に注意).
素イデアル $\mathfrak{p}$ について, $\mathfrak{p} \not\supset x\mathcal{O}$ ならば, ある $y \in x\mathcal{O} \setminus \mathfrak{p}$ が存在し, y は $\mathcal{O}_{\mathfrak{p}}$ において単元, よって $x\mathcal{O}_{\mathfrak{p}} = \mathcal{O}_{\mathfrak{p}}$. 故に, $\bigcap_{\mathfrak{p} \ni x} (x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O}) = \bigcap_{\mathfrak{p}} (x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O})$ (右辺の共通部分は全ての素イデアルにわたる).
$\bigcap_{\mathfrak{p}} (x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O})$ に $x\mathcal{O}$ が含まれることは自明. 逆を示す. $y \in \bigcap_{\mathfrak{p}} (x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O})$ とし, $\mathcal{O}$ の全商環 $(= K)$ において $z = y/x$ を考える. 任意の素イデアル $\mathfrak{p}$ に対し, $y \in x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O}$ であるからある $z_{\mathfrak{p}} \in \mathcal{O}_{\mathfrak{p}}$ が存在して, $y = xz_{\mathfrak{p}}$ とかける. このとき $z = z_{\mathfrak{p}}$ であり, 従って $z \in \bigcap_{\mathfrak{p}} \mathcal{O}_{\mathfrak{p}} = \mathcal{O}$ ($\mathcal{O}$ が Noether 整域であることより等号が成り立つ ([62] 定理 3.6.1)). 故に $y \in x\mathcal{O}$ となり, $\bigcap_{\mathfrak{p} \ni x} (x\mathcal{O}_{\mathfrak{p}} \cap \mathcal{O}) = x\mathcal{O}$ がわかる. 以上により, $\prod_{\mathfrak{p}} \mathfrak{p}^{l_{\mathfrak{p}}(x)} \subseteq x\mathcal{O}$ が示された.

$\mathcal{O}$-加群として $\mathfrak{a}_{i-1}/\mathfrak{a}_i \cong \mathcal{O}/\mathfrak{p}$ であるとき, $\mathfrak{a}_{i-1}/\mathfrak{a}_i$ の零化域 ($\mathrm{Ann}(\mathfrak{a}_{i-1}/\mathfrak{a}_i) := \{a \in \mathcal{O} \mid a(\mathfrak{a}_{i-1}/\mathfrak{a}_i) = \{0\}\}$) は $\mathfrak{p}$ と一致する.

$x, y \in \mathcal{O}$, $x, y \neq 0$ に対し, $l_\mathfrak{p}(x) = t$, $l_\mathfrak{p}(y) = u$, 対応するイデアルの列を, $\mathcal{O} = \mathfrak{a}_0 \supset \mathfrak{a}_1 \supset \cdots \supset \mathfrak{a}_{t-1} \supset \mathfrak{a}_t = x\mathcal{O}$, $\mathcal{O} = \mathfrak{b}_0 \supset \mathfrak{b}_1 \supset \cdots \supset \mathfrak{b}_{u-1} \supset \mathfrak{b}_u = y\mathcal{O}$ とおくとき, $\mathcal{O} = \mathfrak{a}_0 \supset \mathfrak{a}_1 \supset \cdots \supset \mathfrak{a}_{t-1} \supset \mathfrak{a}_t = x\mathcal{O} = x\mathfrak{b}_0 \supset x\mathfrak{b}_1 \supset \cdots \supset x\mathfrak{b}_{u-1} \supset x\mathfrak{b}_u = xy\mathcal{O}$ は xy に対する列となる. よって,

$$l_\mathfrak{p}(xy) = l_\mathfrak{p}(x) + l_\mathfrak{p}(y)$$

が成り立つ. さらに, 整環の定義から K は $\mathcal{O}$ の商体で, 任意の $z(\neq 0) \in K$ に対し, $x, y \in \mathcal{O}$ が存在し, $z = x/y$ とかくことができる. そこで, $l_\mathfrak{p}(z) := l_\mathfrak{p}(x) - l_\mathfrak{p}(y)$ とすることにより, $l_\mathfrak{p}$ は写像

$$l_\mathfrak{p} : K^\times \to \mathbb{Z}$$

を定める. well-defined であることは, $z = x/y = x'/y'$ であるとき, $l_\mathfrak{p}(x) + l_\mathfrak{p}(y') = l_\mathfrak{p}(xy') = l_\mathfrak{p}(yx') = l_\mathfrak{p}(y) + l_\mathfrak{p}(x')$ であることからわかる. 写像 $l_\mathfrak{p}$ は次のような性質を持つ.

命題 A.2.1. 整環 $\mathcal{O}$ の素イデアル $\mathfrak{p}$ に対し, 写像 $l_\mathfrak{p} : K^\times \to \mathbb{Z}$ は次の性質を持つ.
(i) $l_\mathfrak{p}$ は準同型である.
(ii) 任意の $x(\neq 0) \in \mathcal{O}$ に対し, $l_\mathfrak{p}(x) \geq 0$. さらに, $l_\mathfrak{p}(x) > 0 \ \Leftrightarrow \ x \in \mathfrak{p}$.
(iii) 各 $x \in K^\times$ に対し, 有限個を除きすべての素イデアル $\mathfrak{p}$ に対し, $l_\mathfrak{p}(x) = 0$ が成り立つ. さらに,

$$|N(x)| = \prod_\mathfrak{p} (\mathcal{N}(\mathfrak{p}))^{l_\mathfrak{p}(x)}.$$

ただし, 積は $\mathcal{O}$ の 0 でない素イデアル全体にわたる.

上記性質 (i)〜(iii) を満たす $K^\times$ から $\mathbb{Z}$ への写像は $l_\mathfrak{p}$ に限る.

Proof. (i) および (ii) の前半は写像の定義から自明である.

(ii) の後半について. $x \in \mathfrak{p}$ ならば, イデアルの列として少なくとも $\mathcal{O} \supset \mathfrak{p} \supset x\mathcal{O}$ が存在し, $l_\mathfrak{p}(x) \geq 1$ となる. $x \notin \mathfrak{p}$ とすると, $\mathfrak{p}$ は極大イデアルだから, $\mathcal{O} = x\mathcal{O} + \mathfrak{p}$ が成り立つ. 従って, ある $y \in \mathcal{O}$, $z \in \mathfrak{p}$ が存在して $1 = xy + z$ とかける. このとき, z 乗法は $\mathcal{O}/x\mathcal{O}$ 上の, 従って任意の i に対して $\mathfrak{a}_{i-1}/\mathfrak{a}_i$ 上の恒等写像を導く. $\mathfrak{a}_{i-1}/\mathfrak{a}_i$

が $\mathcal{O}/\mathfrak{p}$ と同型 ($\Leftrightarrow\ l_{\mathfrak{p}}(x) > 0$) ならば, 零化域は $\mathfrak{p}$ であるから z が恒等写像を導くことはありえない. 従って $l_{\mathfrak{p}}(x) = 0$ でなければならない.

(iii) は $x \in \mathcal{O}$ の場合について証明すれば十分である. x に対するイデアル列により,

$$|N(x)| = \#\mathcal{O}/x\mathcal{O} = \prod_{i=1}^{t} (\#\mathfrak{a}_{i-1}/\mathfrak{a}_i)$$

が成り立つ. あとは, 各 i に対し, $\mathfrak{a}_{i-1}/\mathfrak{a}_i \cong \mathcal{O}/\mathfrak{p}$ となる素イデアル $\mathfrak{p}$ が一意に定まることを示せばよい. $y \in \mathfrak{a}_{i-1} \setminus \mathfrak{a}_i$ とすると, $\mathfrak{a}_{i-1}$ と $\mathfrak{a}_i$ の間にイデアルが存在しないことから, $y\mathcal{O} + \mathfrak{a}_i = \mathfrak{a}_{i-1}$ が成り立つ. 従って, 写像 $A \to \mathfrak{a}_{i-1}/\mathfrak{a}_i$, $x \mapsto yx + \mathfrak{a}_i$ は全射準同型である. $\mathfrak{a}_{i-1}/\mathfrak{a}_i$ は可換な単純環 (非自明なイデアルを持たない) であるから体であり, 従って上記全射準同型の核を $\mathfrak{p}$ とおけば $\mathfrak{p}$ は極大イデアル (よって素イデアル) である. また, $\mathfrak{p}$ は $\mathfrak{a}_{i-1}/\mathfrak{a}_i$ の零化域なので一意的に定まる.

最後に $l_{\mathfrak{p}}$ の一意性について示す. 元 $px \in \mathcal{O}$ に対し, (A.5) により,

$$\prod_{\mathfrak{q}} \mathfrak{q}^{l_{\mathfrak{q}}(px)} \subseteq px\mathcal{O} \tag{A.6}$$

が成り立つ. $m = l_{\mathfrak{p}}(px)$, $\mathfrak{b} = \prod_{\mathfrak{q}\neq\mathfrak{p}} \mathfrak{q}^{l_{\mathfrak{q}}(px)}$ とおく. $\mathfrak{p}^m + \mathfrak{b} = \mathcal{O}$[14] 従って中国人剰余定理を適用して,

$$\begin{cases} y \equiv x \bmod \mathfrak{p}^m, \\ y \equiv 1 \bmod \mathfrak{b}, \end{cases} \qquad \begin{cases} z \equiv 1 \bmod \mathfrak{p}^m, \\ z \equiv x \bmod \mathfrak{b} \end{cases}$$

は根 $y, z \in \mathcal{O}$ を持つことがわかる[15]. このとき, $yz \equiv x \bmod \mathfrak{p}^m\mathfrak{b}$, 従って包含関係 (A.6) から

$$yz \equiv x \bmod px$$

すなわち, ある $w \in \mathcal{O}$ が存在して $yz = x + pxw = x(1+pw)$, 従って $l_{\mathfrak{p}}(x(1+pw)) = l_{\mathfrak{p}}(yz)$ が成り立つことがわかる. $z, 1+pw \notin \mathfrak{p}$ であるから (i), (ii) より, $l_{\mathfrak{p}}(x) = l_{\mathfrak{p}}(y)$ が成り立つことがわかる.

定義より $\mathfrak{b}$ には $\mathfrak{p}$ と異なる p の上にある素イデアルは全て現れており, 従って $y \equiv 1 \bmod \mathfrak{b}$ は y を含む p の上にある素イデアルは $\mathfrak{p}$ のみであることを意味する. よって (iii) の式により $\mathcal{N}(\mathfrak{p})^{l_{\mathfrak{p}}(y)}$ は $N(y)$ を割る p べきと一致することがわかる.

[14] $\mathfrak{p}$ は極大イデアルだから, $\mathfrak{p} + \mathfrak{b} = \mathcal{O}$. よって $s \in \mathfrak{p}$, $t \in \mathfrak{b}$ が存在して $s + t = 1$. このとき $1 = (s+t)^{2m} = \sum_{i=0}^{m} a_i s^i t^{2m-i} + \sum_{i=m+1}^{2m} a_i s^i t^{2m-i}$ であり, 第一項は $\mathfrak{b}$ に, 第二項は $\mathfrak{p}^m$ に含まれるので, $\mathfrak{p}^m + \mathfrak{b} \ni 1$ がわかる.

[15] $\mathcal{O}$ のイデアル $\mathfrak{a}$ と $x, y \in \mathcal{O}$ に対し, $x \equiv y \bmod \mathfrak{a} \Leftrightarrow x - y \in \mathfrak{a}$ と定める.

$l'_{\mathfrak{p}} : K^\times \to \mathbb{Z}$ を (i), (ii), (iii) を満たす任意の写像とすれば, 同様にして $l'_{\mathfrak{p}}(x) = l'_{\mathfrak{p}}(y)$, および, $\mathcal{N}(\mathfrak{p})^{l'_{\mathfrak{p}}(y)}$ は $N(y)$ を割る p べきと一致する. 故に $l_{\mathfrak{p}}(x) = l_{\mathfrak{p}}(y) = l'_{\mathfrak{p}}(y) = l'_{\mathfrak{p}}(x)$ を得る.

$\mathcal{O} = \mathcal{O}_K$ のときは, $l_{\mathfrak{p}}$ は一意性により, $\mathcal{O}_K$ の素イデアル $\mathfrak{p}$ に対する正規指数付値と一致することがわかる. すなわち, $l_{\mathfrak{p}}(x)$ は単項イデアル $x\mathcal{O}_K$ の素イデアル分解における $\mathfrak{p}$ のべき指数に一致する.

A, B, $A \subset B$ を K の整環とする. $\mathfrak{q} \subset B$ を素イデアルとするとき, $\mathfrak{p} = \mathfrak{q} \cup A$ は A の素イデアルである. このとき, $\mathfrak{q}$ は $\mathfrak{p}$ の上にあるといい, $\mathfrak{q}|\mathfrak{p}$ とかく.

$\mathfrak{q}$ が $\mathfrak{p}$ の上にあるとき, 明らかに $B/\mathfrak{q}$ は $A/\mathfrak{p}$ の拡大体で, その拡大次数を $f(\mathfrak{q}/\mathfrak{p})$ とかく.

先に定義した写像 $l_{\mathfrak{p}}$ について, ここでは混乱を避けるため, 整環を明記して $l_{\mathfrak{p},A}$ とかく. この状況下では次が成り立つ.

命題 A.2.2. $\mathfrak{p}$ を A の素イデアルとする. このとき任意の $x \in K^\times$ に対し,

$$l_{\mathfrak{p},A}(x) = \sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p}) l_{\mathfrak{q},B}(x)$$

が成り立つ. ただし, 和は $\mathfrak{p}$ の上にある B の素イデアル全体にわたる.

Proof. 一般に, 有限 A 加群 M に対し, $l_{\mathfrak{p},A}(M)$ を, M の組成列の因子[16] のうち, $A/\mathfrak{p}$ と同型なものの個数とおく. 定義により, $x(\neq 0) \in A$ に対し, $l_{\mathfrak{p},A}(x) = l_{\mathfrak{p},A}(A/xA)$ が成り立つ. また M の部分加群 L に対し, 明らかに $l_{\mathfrak{p},A}(L) = l_{\mathfrak{p},A}(M) - l_{\mathfrak{p},A}(M/L)$ が成り立つ.

命題の主張は, A の商体は K であること, および $l_{\mathfrak{p},A}$ の準同型性から, $x \in A$ に対して示せば十分であることがわかる.

x-乗法 (x を掛ける写像) は B/A と xB/xA の同型を与える. よって $l_{\mathfrak{p},A}(B/A) = l_{\mathfrak{p},A}(xB/xA)$ が成り立ち, したがって,

$$\begin{aligned} l_{\mathfrak{p},A}(x) = l_{\mathfrak{p},A}(A/xA) & \\ &= l_{\mathfrak{p},A}(B/xA) - l_{\mathfrak{p},A}((B/xA)/(A/xA)) \\ &= l_{\mathfrak{p},A}(B/xA) - l_{\mathfrak{p},A}(B/A) \\ &= l_{\mathfrak{p},A}(B/xA) - l_{\mathfrak{p},A}(xB/xA) \\ &= l_{\mathfrak{p},A}(B/xB) \end{aligned}$$

[16] $M \subset M_1 \subset M_2 \cdots$ を組成列とするとき, M_{i-1}/M_i を因子という.

となる. よって求める関係式は $M = B/xB$ とおくとき

$$l_{\mathfrak{p},A}(M) = \sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p}) l_{\mathfrak{q},B}(M)$$

と同値である. そこで任意の有限 B 加群に対し, この関係式を示せば十分. さらに, 組成列を考えることにより, M が単純 B 加群, すなわち, 部分加群が $\{0\}$ とそれ自身しかないような加群の場合に示せばよいことがわかる. この場合, M に対し, ある素イデアル $\mathfrak{q}' \subset B$ が存在し, $M \cong B/\mathfrak{q}'$ となり[17],

$$l_{\mathfrak{p},B}(M) = \begin{cases} 1 & \mathfrak{q} = \mathfrak{q}', \\ 0 & \mathfrak{q} \neq \mathfrak{q}'. \end{cases}$$

$\mathfrak{p}' = \mathfrak{q}' \cap A$ とおけば, $f(*/*)$ の定義により, A 加群として, $M = B/\mathfrak{q}'$ は $A/\mathfrak{p}'$ の $f(\mathfrak{q}'/\mathfrak{p}')$ 個の直和となる. よって

$$l_{\mathfrak{p},A}(M) = \begin{cases} f(\mathfrak{q}'/\mathfrak{p}') & \mathfrak{p} = \mathfrak{p}', \\ 0 & \mathfrak{p} \neq \mathfrak{p}' \end{cases}$$

となり, 求める関係式を得る.

さらに次が成り立つ.

命題 A.2.3. 有限個を除き全ての素イデアル $\mathfrak{p} \subset A$ に対し,

$$\sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p}) = 1$$

が成り立つ. さらに, 整数

$$\prod_{\mathfrak{p}} \mathcal{N}(\mathfrak{p})^{-1+\sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p})}$$

は指数 $[B : A]$ を割る. ただし, 積は A の素イデアル $\mathfrak{p}$ 全体にわたる.

Proof. T を A の素イデアルの有限集合とし, U を T に含まれる素イデアルの上にある B の素イデアル全体の集合とする. $\mathfrak{a} = \bigcap_{\mathfrak{p}\in T} \mathfrak{p}$, $\mathfrak{b} = \bigcap_{\mathfrak{q}\in U} \mathfrak{q}$ とおく. このとき, $\mathfrak{a} = \mathfrak{b} \cap A$ であり, よって $A/\mathfrak{a}$ は $B/\mathfrak{b}$ の部分環[18] 指数 $[B : A]$ は $[B/\mathfrak{b} : A/\mathfrak{a}]$ で割り

[17] $\mathrm{Ann}(M) = \mathfrak{q}'$ とおけば $M \cong B/\mathfrak{q}$. M の有限性や単純性より $\mathfrak{q}'$ が素イデアルであることがわかる.

[18] 準同型 $A \hookrightarrow B \to B/\mathfrak{b}$ の核は $\mathfrak{a}$.

切れる[19]. ここで中国人剰余定理により,

$$A/\mathfrak{a} \cong \prod_{\mathfrak{p}\in T} A/\mathfrak{p}$$

となるので,

$$\#A/\mathfrak{a} = \prod_{\mathfrak{p}\in T} \mathcal{N}(\mathfrak{p})$$

を得る. 同様にして (また $f(*/*)$ の定義から),

$$\#B/\mathfrak{b} = \prod_{\mathfrak{q}\in U} \mathcal{N}(\mathfrak{q}) = \prod_{\mathfrak{p}\in T} \mathcal{N}(\mathfrak{p})^{\sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p})}.$$

以上により, $[B:A]$ が

$$(\#B/\mathfrak{b}) \,/\, (\#A/\mathfrak{a}) = \prod_{\mathfrak{p}\in T} \mathcal{N}(\mathfrak{p})^{-1+\sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p})}$$

で割り切れることがわかる. $[B:A]$ は T とは無関係であるから, この関係式より $\sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p}) \neq 1$ であるような $\mathfrak{p}$ が有限個であることがわかる. また 2 番目の主張も上記関係式より明らか.

これらの命題により, ただちに次がわかる.

系 A.2.1. A の素イデアルに対し, その上にある B の素イデアルは存在してかつ有限個である. また, 有限個を除き全ての素イデアル $\mathfrak{p} \subset A$ に対し, $\mathfrak{p}$ の上にある素イデアル $\mathfrak{q}$ はただ一つで, $f(\mathfrak{q}/\mathfrak{p}) = 1$ を満たす.

K の整環 A に対し,

$$V_A = \left\{ x \in K^\times \mid l_{\mathfrak{p},A}(x) \equiv 0 \bmod 2 \text{ for all primes } \mathfrak{p} \subset A \right\}$$

とおく. $A \subset B$ ならば, 命題 A.2.2 により $V_B \subset V_A$ となる. さらに次が成り立つ.

命題 A.2.4. $A \subset B$ を K の整環とするとき

$$[V_A : V_B] \leq [B : A].$$

[19] $A/\mathfrak{a} = A/(\mathfrak{b}\cap A) \cong (A+\mathfrak{b})/\mathfrak{b}$ であり, よって, $[B/\mathfrak{b} : A/\mathfrak{a}] = [B/\mathfrak{b} : (A+\mathfrak{b})/\mathfrak{b}] = [B : A+\mathfrak{b}]$. 一方, $[B:A] = [B : A+\mathfrak{b}][A+\mathfrak{b} : A]$ であるから主張を得る.

Proof. 各素イデアル $\mathfrak{p} \subset A$ に対し, 集合 $S_\mathfrak{p}$ を次のように定める:

$\mathfrak{p}$ の上にある B の全ての素イデアル $\mathfrak{q}$ に対し, $f(\mathfrak{q}/\mathfrak{p})$ が偶数であるならば, $S_\mathfrak{p} = \{\mathfrak{p} \subset B \mid \mathfrak{q}|\mathfrak{p}\}$ (すなわち $\mathfrak{p}$ の上にある素イデアル全体).

$\mathfrak{p}$ の上にある素イデアルのうち, すくなくとも一つの $\mathfrak{q}$ に対し, $f(\mathfrak{q}/\mathfrak{p})$ が奇数である場合, そのような素イデアルを $\mathfrak{q}_0$ とするとき, $S_\mathfrak{p} = \{\mathfrak{p} \subset B \mid \mathfrak{q}|\mathfrak{p}\} \setminus \{\mathfrak{q}_0\}$ (すなわち $\mathfrak{p}$ の上にある素イデアルのうち $\mathfrak{q}_0$ 以外全て) とおく.

このとき明らかに

$$\#S_\mathfrak{p} \leq -1 + \sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p})$$

が成り立つ ($f(\mathfrak{q}/\mathfrak{p})$ が偶数ならば ≥ 2 であるからよい. 奇数のものが存在する場合でも $S_\mathfrak{p}$ は $\mathfrak{p}$ の上にある素イデアル全てからひとつ引かれているので上記不等号は成り立つ).

命題 A.2.3 より, 有限個を除き全ての $\mathfrak{p}$ に対し, $S_\mathfrak{p}$ は空集合である.

$$S = \bigcup_\mathfrak{p} S_\mathfrak{p}$$

とおく ($\mathfrak{p}$ は A の素イデアル全体にわたる). このとき命題 A.2.3 により S は有限集合で, さらに

$$2^{\#S} \leq \prod_\mathfrak{p} \mathcal{N}(\mathfrak{p})^{\#S_\mathfrak{p}} \leq \prod_\mathfrak{p} \mathcal{N}(\mathfrak{p})^{-1+\sum_{\mathfrak{q}|\mathfrak{p}} f(\mathfrak{q}/\mathfrak{p})} \leq [B : A]$$

が成り立つ. よって, V_A/V_B を, $(\mathbb{Z}/2\mathbb{Z})^{\#S}$ に埋め込めることを示せば主張を得る. そこで, 写像

$$V_A/V_B \to (\mathbb{Z}/2\mathbb{Z})^{\#S}, \quad x \mapsto (l_{\mathfrak{q},B}(x) \bmod 2)_{\mathfrak{q}\in S}$$

を考える. x が核の元であるとすると, 全ての $\mathfrak{q} \in S$ に対し $l_{\mathfrak{q},B}(x) \equiv 0 \bmod 2$. $l_{\mathfrak{p},A}(x)$ は全て偶数であり, 命題 A.2.2 と $S_\mathfrak{p}$ の定義により, $l_{\mathfrak{q},B}(x)$ は全ての $\mathfrak{q}$ に対し偶数, すなわち $x \in V_B$ がわかる. よって上記写像は単射で, 埋め込むことができることが示せた.

$f(x) \in \mathbb{Z}[x]$ を整係数 monic 既約多項式とし, $\theta \in \mathbb{C}$ を $f(x) = 0$ の一つの根 : $f(\theta) = 0$ とする. 有限次代数体 $K = \mathbb{Q}(\theta)$ において, $\mathbb{Z}[\theta]$ は K の整環である. しかし, 一般には代数的整数環 $\mathcal{O}_K$ と一致するとは限らないが, 次が成り立つ.

命題 A.2.5. $f'(\theta)\mathcal{O}_K \subseteq \mathbb{Z}[\theta]$. すなわち, 任意の $a \in \mathcal{O}_K$ に対し, $f'(\theta)a \in \mathbb{Z}[\theta]$. ただし, $f'(x)$ は $f(x)$ の 1 階導関数である.

Proof. [55] Proposition 3-7-14.

平方剰余記号

p を奇素数とする. p と素な整数 $a \in \mathbb{Z}$, $\gcd(a,p)=1$ に対し, ある $b \in \mathbb{Z}$ が存在して $a \equiv b^2 \bmod p$ となるとき, a を p を法とした**平方剰余**であるという.

$a \in \mathbb{Z}$, $\gcd(a,p)=1$ が p を法として平方剰余であるとき,

$$\left(\frac{a}{p}\right) = 1$$

とあらわし, そうでないとき

$$\left(\frac{a}{p}\right) = -1$$

とあらわす. これらを**平方剰余記号** (または Legendre 記号) という.

定義から明らかに $a \equiv c \bmod p$ ならば $\left(\frac{a}{p}\right) = \left(\frac{c}{p}\right)$ であり, さらに乗法的: $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$ である. 従って, 平方剰余記号は有限体の乗法群 $(\mathbb{Z}/p\mathbb{Z})^\times$ から $\{\pm 1\}$ への関数とみなすこともできる.

また平方剰余記号は次の **Euler 規準**により計算できる.

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p.$$

異なる素数を法とした平方剰余記号については次の法則が成り立つ.

(i) $\left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{p-1}{2}}$.

(ii) $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$.

(iii) $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$.

(i) を平方剰余記号の**相互法則**, (ii),(iii) を補充法則という. これらを用いて平方剰余記号を高速に計算する方法もある ([10] など).

A.2.2 数体篩法の方針

ここでは数体篩法における素因数分解の方針について, 概略を説明する.

合成数 n の素因数分解をすることを考える. $f(x) \in \mathbb{Z}[x]$ を 1 変数 monic 既約 d 次整係数多項式とする. $\theta \in \mathbb{C}$ を f の一つの根:

$$f(\theta) = 0 \tag{A.7}$$

とし, $\mathbb{Z}$ 上, θ で生成される環

$$\mathbb{Z}[\theta] := \{a_0 + a_1\theta + \cdots + a_{d-1}\theta^{d-1} \mid a_i \in \mathbb{Z}\}$$

を考える. さらに, $m \in \mathbb{Z}$ を

$$f(m) \equiv 0 \bmod n \tag{A.8}$$

を満たすものとする. このとき, 環準同型

$$\phi : \mathbb{Z}[\theta] \longrightarrow \mathbb{Z}/n\mathbb{Z}, \quad g(\theta) \mapsto g(m) \bmod n \tag{A.9}$$

が自然に定義できる. ここで, 互いに素な整数の組 (a, b) の集合 S であって,

$$\prod_{(a,b)\in S} (a + bm) \quad \text{は } \mathbb{Z} \text{ において平方数}, \tag{A.10}$$

$$\prod_{(a,b)\in S} (a + b\theta) \quad \text{は } \mathbb{Z}[\theta] \text{ において平方数} \tag{A.11}$$

を満たすものを見つけることができたとする. $\prod(a + bm) = x^2$, $x \in \mathbb{Z}$, $\prod(a + b\theta) = \mu^2$, $\mu \in \mathbb{Z}[\theta]$ とおく. このとき上記準同型により, $\phi(a + b\theta) = a + bm \bmod n$ であるから

$$\phi(\mu^2) = x^2 \bmod n \tag{A.12}$$

が成り立つ. $\phi(\mu) = y \bmod n$ $(y \in \mathbb{Z})$ とすれば,

$$y^2 \equiv x^2 \bmod n \tag{A.13}$$

が成り立ち, 従って $\gcd(x - y, n)$ を計算することで n の因数が見つかる可能性がでてくる.

以上が数体篩法の基本方針である.

篩系の素因数分解アルゴリズム (2 次篩, 数体篩など) は次のようにまとめることができる.

環 R と環準同型

$$\psi : R \to \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$$

であって, ψ による像が対角成分 $\{(x, x) \mid x \in (\mathbb{Z}/n\mathbb{Z})^\times\} \subset \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ に含まれるような R の元をたくさん生成できるアルゴリズムが与えられたとする.

このような元を乗法的に組み合わせて R における平方数を構成する. その平方根の ψ による像が $\{(x, \pm x) \mid x \in (\mathbb{Z}/n\mathbb{Z})^\times\}$ に含まれていないならば, これらの元から n の素因数分解が可能となる.

2 次篩法では $R = \mathbb{Z} \times \mathbb{Z}$ をとり, $(a_i + m, Q(a_i)) \in R$ を組み合わせて平方数を構成, これらから素因数分解を行った.

数体篩法では $R = \mathbb{Z} \times \mathbb{Z}[\theta]$ とおき, $\psi(a, g(\theta)) = (a \bmod n, \phi(\alpha))$ を考えた.

より一般的に $R = \prod_i \mathbb{Z}[\theta_i]$ を考えることもでき, 数体篩法の高速化手法などで用いられている.

数体篩法において, 上記方針に沿って具体的に実行する場合, 次のような問題が生じる.

NFS-1) 多項式 $f(x)$ や法 n での根 m はどのように構成するか.

NFS-2) 条件 (A.10), (A.11) を満たす集合 S をどのように探すか.

NFS-1) については次節で構成法の 1 つを紹介する. 課題 NFS-2) の集合 S は, 大まかには 2 つのステップに分けて構成される.

(i) 互いに素な整数の組 (a, b) の集合 T で, $a + bm$, $a + b\theta$ がともに smooth であるものを探す (smooth の定義は後述).

(ii) 線型代数を用いて条件を満たす集合 $S \subseteq T$ を求める.

さらに, (i) は, smooth な $a + bm$ を探すこと (これを数体篩法における"rational sieve"という) と, smooth な $a + b\theta$ を探すこと (これを"algebraic sieve"という), さらに同時に smooth となるものを探すことに分けられる.

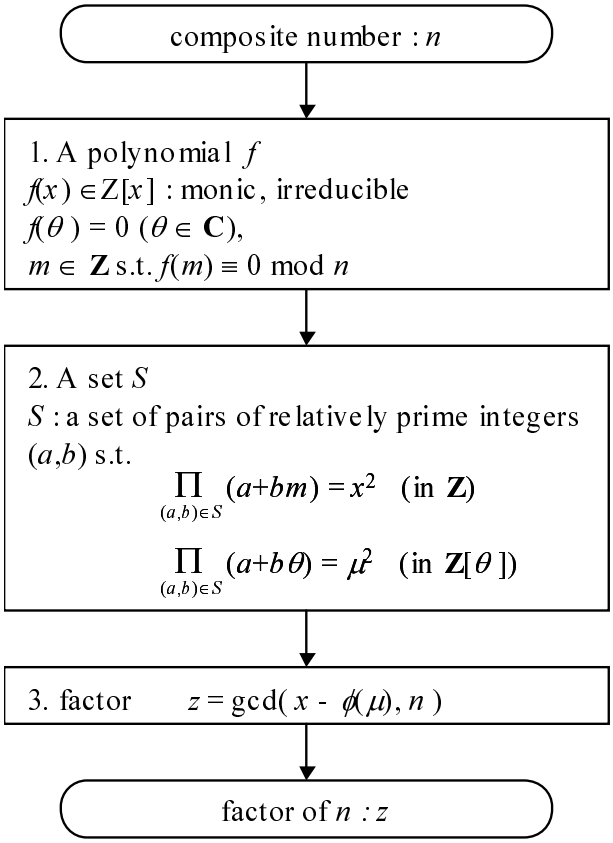

composite number : n
1. A polynomial f
$f(x) \in Z[x]$: monic, irreducible
$f(\theta) = 0$ ($\theta \in \mathbf{C}$),
$m \in \mathbf{Z}$ s.t. $f(m) \equiv 0 \bmod n$
2. A set S
S : a set of pairs of relatively prime integers (a,b) s.t.
$\prod_{(a,b)\in S} (a+bm) = x^2$ (in $\mathbf{Z}$)
$\prod_{(a,b)\in S} (a+b\theta) = \mu^2$ (in $\mathbf{Z}[\theta]$)
3. factor $z = \gcd(x - \phi(\mu), n)$
factor of n : z

図 A.1: 数体篩法の流れ

多項式 f と整数 m の構成法

ここでは最も素朴な"base m"法と呼ばれる構成法を紹介する.

素数べきでない正整数 n が与えられたとき, 1 変数整係数 monic 多項式 $f(x)$ と, 整数 m であって, $f(m)$ が n の倍数であるものを構成する.

整数 $d > 1$ を $n > 2^{d^2}$ を満たすようにとり, $m = [n^{1/d}]$ (Gauss 記号) とおく. このとき n の m 進展開を

$$n = c_d m^d + c_{d-1} m^{d-1} + \cdots + c_0, \quad c_i \in \mathbb{Z},\ 0 \leq c_i < m$$

とする. このとき d, m の取り方から,

$$c_d = 1, \quad c_{d-1} < d$$

であることが容易にわかる.

ここで多項式 $f(x) \in \mathbb{Z}[x]$ を

$$f(x) = x^d + c_{d-1} x^{d-1} + \cdots + c_0 \tag{A.14}$$

とおく. $f(m) = n$ であることに注意 (求める条件より強い). ただし, この段階では $f(x)$ は既約とは限らない. しかし, もし $f(x) = g(x)h(x)$ のように分解されたとすると, $n = f(m) = g(m)h(m)$ であるが, $g(m), h(m) \neq \pm 1$ であるならばこれは n の非自明な分解を与えることになる. また, $g(m) = \pm 1$, または $h(m) = \pm 1$ ならば他方を改めて $f(x)$ とし, 再度, 既約性の判定を行えばよい[20].

"base m"法で構成された多項式 f の判別式 $\Delta = \Delta_f$ は

$$|\Delta| < d^{2d} n^{2-3/d} \tag{A.15}$$

を満たす. 実際, 多項式の判別式は符号の差を除き, 自身とその導関数との終結式により与えられる. 対応する Sylvester 行列の成分を評価することでその行列式の絶対値を評価することができる.

[20] 整係数多項式の分解時間は $(\log n)^{O(1)}$ であることが知られている ([31]).

rational part

ここでは”rational”について説明する.

u を (n に依存する) 大きな正整数とし,

$$U := \{(a,b) \mid a,b \in \mathbb{Z}, \gcd(a,b) = 1, |a| \leq u, 0 < b \leq u\}$$

とおく. 次にパラメータ $y = y(n)$(正の実数) を固定し,

$$T_1 := \{(a,b) \in U \mid a + bm : y\text{-smooth}\}$$

とする. ここで, 整数 a が y-smooth であるとは, a の全ての素因子が y 以下であることをいう.

T_1 は”**篩**”(sieve) によって求められる:

各 $0 < b \leq u$ に対し, $a + bm$, $-u \leq a \leq a$ が入った配列を考える. 各素数 $p \leq y$ に対し, 各配列の要素 $a + bm$ が $a \equiv -bm \bmod p$ を満たす ($a + bm$ が p で割れる) ならば, 要素を配列からいったん取り出し, p で割れる限り割り続け, 最後に得られた商を同じ配列に戻すという作業を行う. この作業が終了したとき, 最終的に配列に残っている数は, a に対応する配列について, $a + bm$ の, y 以下の全ての素数と素な最大の約数ということになる. もしある a に対応する配列の要素が最終的に ± 1 となっているならばこれは $a + bm$ が y-smooth であることを意味し, T_1 の元を見つけたことになる.

y 以下の素数の集合を篩の **factor base** という. y 以下の素数の個数を $\pi(y)$ で表す.

パラメータ y および u は,

$$\#T_1 > \pi(y) + 1 \tag{A.16}$$

を満たすように取られていると仮定する. このとき, 線型代数 (上記 (ii)) を用いて, (A.10) を満たす集合 $S \subset T_1$ を見つけることが出来る:

$B = \pi(y)$ とおき, p_j $(1 \leq j \leq B)$ を j 番目の素数とする. また $p_0 = -1$ とおく. y-smooth な整数

$$w = \prod_{j=0}^{B} p_j^{e_j}$$

に対し, ベクトル $e(w) \in \mathbb{F}_2^{B+1}$ を次で定義する:

$$e(w) = (e_0 \bmod 2,\ e_1 \bmod 2,\ \ldots,\ e_B \bmod 2)\,.$$

各 $(a,b) \in T_1$ に対し, 2 元体 $\mathbb{F}_2$ 上の $B+1$ 次元ベクトル $e(a+bm)$ を作れば, 仮定 (A.16) よりベクトルの個数がベクトル空間 $\mathbb{F}_2^{B+1}$ の次元 $B+1$ を真に超える. よってこれらのベクトルには ($\mathbb{F}_2$ 上) 非自明な線型関係が存在する (1 次従属). すなわち, 空でない部分集合 $S \subseteq T_1$ が存在して,

$$\sum_{(a,b)\in S} e(a+bm) = 0 \ \in \mathbb{F}_2^{B+1}$$

となる. これは

$$\prod_{(a,b)\in S} (a+bm) = x^2 \text{ in } \mathbb{Z}$$

すなわち, 平方数であることを意味する. 以上により, まず $(A.10)$ を満たす集合 S を探すことができる.

algebraic part

次に algebraic 部分について説明する. 記号は前節までと同様とする.

$\alpha \in \mathbb{Z}[\theta]$ が y-smooth であるとは, ノルム $N(\alpha) \in \mathbb{Z}$ が y-smooth であることと定義する.

$a+b\theta \in \mathbb{Z}[\theta]$ $(a, b(\neq 0) \in \mathbb{Z})$ の形の元のノルムは

$$N(a+b\theta) = b^d f(-a/b)$$

により与えられる (A.2.1 節参照).

集合 U を前節と同様とし, 前節の類似として次の集合を考える:

$$T_2 := \{(a,b) \in U \mid a+b\theta : y\text{-smooth}\}\,.$$

素数 p に対し, $f \bmod p$ の ($\mathbb{Z}/p\mathbb{Z}$ での) 根の集合を次のようにおく.

$$R(p) := \{r \in \mathbb{Z}/p\mathbb{Z} \mid f(r) \equiv 0 \bmod p\}.$$

整数 $0 < b \leq u$, $b \not\equiv 0 \bmod p$ を固定するとき,

$$\begin{aligned} N(a+b\theta) \equiv 0 \bmod p &\Leftrightarrow b^d f(-a/b) \equiv 0 \bmod p \\ &\Leftrightarrow -a/b =: r \in R(p) \\ &\Leftrightarrow a \equiv -br \bmod p \text{ for some } r \in R(p) \end{aligned}$$

となる. $b \equiv 0 \bmod p$ ならば, $(a,b) \in U$ において, a, b は互いに素だから, $N(a+b\theta) \equiv 0 \bmod p$ となるものは存在しないことに注意.

さらに, $N(a+b\theta) \equiv 0 \bmod p$ のとき, 対応する $r \in R(p)$ は一意であることに注意する.

前節の篩と同様に, b を固定し, $N(a+b\theta)$, $-u \leq a \leq u$ が入った配列を考える. b を割らない各素数 $p \leq y$ と, 各 $r \in R(p)$ に対し, $a \equiv -br \bmod p$ が成り立つ (すなわち $N(a+b\theta) \equiv 0 \bmod p$) 配列の要素 $N(a+b\theta)$ を取り出し, p で割れる限り割り続け, 最後に得られた商を同じ配列に戻すという作業を行う. この作業が終了したとき, ± 1 が入っている配列が y-smooth な $a+b\theta$ に対応している, 従って T_2 の元が得られたということになる.

ここで T_2 に対し, "rational" のときと同様に, 2 元体上の線型代数を適用したいのだが, その結果得られるのは単に, ノルム $N(a+b\theta)$ の組み合わせで有理整数として平方となるもの (集合 S) であり, 求めるもの (条件 (A.11)) とは異なっている. ノルムが平方となることは, 条件 (A.11) が成り立つためには必要条件であるが, 十分条件にはほど遠い. さらに条件を加える必要がある.

$a, b \in \mathbb{Z}$, $\gcd(a,b) = 1$, および素数 p と $r \in R(p)$ に対し,

$$e_{p,r}(a+b\theta) = \begin{cases} \mathrm{ord}_p(N(a+b\theta)) & \text{if } a+br \equiv 0 \bmod p, \\ 0 & \text{otherwise,} \end{cases}$$

とおく. ただし, ord_p は有理整数上の order 関数 ($\mathrm{ord}_p(a) = t \overset{\text{def}}{\Leftrightarrow} a = p^t a', \gcd(a', p) = 1$) をあらわす.

前述のように, $p | N(a+b\theta)$ ならば, $r \in R(p)$ が条件 $a+br \equiv 0 \bmod p$ によりただ一つ定まり, 従って

$$N(a+b\theta) = \pm \prod_{p,r} p^{e_{p,r}(a+b\theta)} \tag{A.17}$$

が成り立つ (積は素数 p 全体と $r \in R(p)$ 全体にわたる)[21].

$\mathfrak{p} \subset \mathbb{Z}[\theta]$ を p(有理素数) の上にある 1 次の素イデアル (すなわち, $\mathcal{N}(\mathfrak{p}) = p$) とする. 定義より, 同型写像 $\varphi_{\mathfrak{p}} : \mathbb{Z}[\theta]/\mathfrak{p} \xrightarrow{\sim} \mathbb{F}_p$ が存在するが, この同型写像による θ の像を r とおくと, $r \in R(p)$ である ($f(\theta) = 0$ だから $\varphi_{\mathfrak{p}}(f(\theta)) = f(r) = 0$ (in $\mathbb{F}_p$)). よって p の上にある 1 次の素イデアル $\mathfrak{p}$ に対し $r \in R(p)$ が対応することがわかる. 逆に組 (p, r) ($r \in R(p)$) に対して, 写像 $\varphi_{(p,r)} : \mathbb{Z}[\theta] \to \mathbb{F}_p$, $\theta \mapsto r$ は全射準同型で, その核 $\ker(\varphi_{(p,r)})$ は p の上にある 1 次の素イデアル $\mathfrak{p}$ であり, 明らかに p と $\theta - r$ で生成される: $\mathfrak{p} = (p, \theta - r)$.

以上により, $\mathcal{O}$ の 1 次の素イデアルと, 組 (p, r) ($r \in R(p)$) の間に 1 対 1 対応が存在することがわかった.

$$\begin{array}{ccc} \{\mathcal{O} \text{ の 1 次の素イデアル }\} & \overset{1:1}{\longleftrightarrow} & \{(p, r) \mid p : \text{有理素数}, r \in R(p)\} \\ \mathfrak{p} = \ker(\varphi_{(p,r)}) & \leftrightarrow & (p, r) \text{ where } r = \varphi_{\mathfrak{p}}(\theta) \end{array}$$

次の命題は $e_{p,r}$ と $l_{\mathfrak{p}}$(A.2.1 節参照) との関係を与える.

命題 A.2.6. a, b を互いに素な有理整数とし, $\mathfrak{p} \subset \mathbb{Z}[\theta]$ を素イデアルとする. このとき $\mathfrak{p}$ が 1 次の素イデアルならば, (p, r) を対応する組とするとき $l_{\mathfrak{p}}(a + b\theta) = e_{p,r}(a + b\theta)$ が成り立つ. 1 次でなければ $l_{\mathfrak{p}}(a + b\theta) = 0$ となる.

Proof. . $l_{\mathfrak{p}}(a + b\theta) > 0$ とする. $\mathfrak{p}$ が 1 次の素イデアルであることを示せば, その対偶により主張の後半を示すことができる. 命題 A.2.1(ii) より $a + b\theta \in \mathfrak{p}$ であり, 自然な写像 $\psi_{\mathfrak{p}} : \mathbb{Z}[\theta] \to \mathbb{Z}[\theta]/\mathfrak{p}$ において $a + b\theta$ は 0 に写される. p を $\mathfrak{p}$ に含まれる有理素数とする. b が p で割れるならば, $0 = \psi_{\mathfrak{p}}(a + b\theta) = \psi_{\mathfrak{p}}(a)$, すなわち $a \in \mathfrak{p}$ で, 有理整数であることから p は a を割ることがわかる. これは a, b が互いに素であることに矛盾する. 従って b は p で割れず, $\psi_{\mathfrak{p}}(b) = b \bmod p \neq 0$. よって,

$$\psi_{\mathfrak{p}}(\theta) = -\frac{\psi_{\mathfrak{p}}(a)}{\psi_{\mathfrak{p}}(b)} = -\frac{a \bmod p}{b \bmod p} \tag{A.18}$$

であり, かつ $\mathbb{Z}[\theta]/\mathfrak{p}$ の素体 $\mathbb{F}_p$ に含まれる. これは $\mathbb{Z}[\theta]/\mathfrak{p} = \psi_{\mathfrak{p}}(\mathbb{Z}[\theta]) = \mathbb{F}_p$, すなわち, $\mathfrak{p}$ が 1 次の素イデアルであることを意味し, $r = \psi_{\mathfrak{p}}(\theta)$ の $\bmod p$ での代表元

[21] $p|N(a + b\theta)$ のとき一意に定まる $r \in R(p)$ を r_p とかくならば,

$$\prod_{p,r} p^{e_{p,r}(a+b\theta)} = \prod_{p|N(a+b\theta)} p^{e_{p,r_p}(a+b\theta)} = \prod_{p|N(a+b\theta)} p^{\mathrm{ord}_p(N(a+b\theta))} = \pm N(a + b\theta).$$

$(\in \mathbb{Z})$ と置けば, $\mathfrak{p} = (p, \theta - r)$, すなわち, 組 (p, r) に対応することがわかる. 従って, $a + b\theta$ と有理素数 p を含む素イデアルは, 存在するならば一意的であることもわかる. そこで, 命題 A.2.1(iii) の式と, 式 (A.17) を比較すれば, $l_{\mathfrak{p}}(a + b\theta) = e_{p,r}(a + b\theta)$ $(r = -a/b \bmod p)$ が成り立つことがわかる.

系 A.2.2. a, b を互いに素な有理整数, p を有理素数とする. $\mathbb{Z}[\theta]$ において, $a + b\theta$ と p を含む素イデアルが存在するならば一意である.

以上の準備のもと, 次を示すことができる.

命題 A.2.7. S を互いに素な整数の組 (a, b) の集合であって, 条件 (A.11)

$$\prod_{(a,b)\in S} (a + b\theta) \text{ は } K = \mathbb{Q}(\theta) \text{ において平方数}$$

が成り立つものとする. このとき, 各素数 p と各 $r \in R(p)$ に対し,

$$\sum_{(a,b)\in S} e_{p,r}(a + b\theta) \equiv 0 \bmod 2$$

が成り立つ.

Proof. . $\prod_{(a,b)\in S}(a + b\theta) = \gamma^2$, $\gamma \in K$ とおく. 組 (p, r) に対応する素イデアルを $\mathfrak{p}$ と書けば, 命題 A.2.6 より $e_{p,r}(a + b\theta) = l_{\mathfrak{p}}(a + b\theta)$ であり, さらに命題 A.2.1(i) により $l_{\mathfrak{p}}$ は準同型であるから,

$$\begin{aligned} &\textstyle\sum_{(a,b)\in S} e_{p,r}(a + b\theta) = \sum_{(a,b)\in S} l_{\mathfrak{p}}(a + b\theta) = l_{\mathfrak{p}}\left(\prod_{(a,b)\in S}(a + b\theta)\right) \\ &= l_{\mathfrak{p}}(\gamma^2) = 2l_{\mathfrak{p}}(\gamma) \equiv 0 \bmod 2. \end{aligned}$$

一般には逆は成り立たない.

さて, $B' := \#\{\mathfrak{p} \mid \mathfrak{p} \text{ は } 1 \text{ 次の素イデアル}, \mathcal{N}(\mathfrak{p}) \le y\}$ とおく. もし, $\#T_2 > B'$ ならば, A.2.2 節で述べた線型代数を用いた方法と同様の方法により, 空でない部分集合 $S \subset T_2$ であって, 任意の素イデアル $\mathfrak{p}$ に対し,

$$\sum_{(a,b)\in S} l_{\mathfrak{p}}(a + b\theta) \equiv 0 \bmod 2 \tag{A.19}$$

が成り立つようなものを見つけることができる. しかしこの条件 (A.19) は, 求めたい条件式 (A.11) のためには必要であるが, 十分であるにはなお程遠く, 次のような問題が残っている.

AS-1) $\mathcal{O}_K$ のイデアルとして $\prod_{(a,b)\in S}(a+b\theta)\mathcal{O}_K$ は平方イデアルか?

AS-2) イデアル $\mathfrak{a}\subset\mathcal{O}_K$ に対し, $\prod_{(a,b)\in S}(a+b\theta)\mathcal{O}_K=\mathfrak{a}^2$ であったとして, $\mathfrak{a}$ は単項イデアルか?

AS-3) 元 $\gamma\in\mathcal{O}_K$ に対し, $\prod_{(a,b)\in S}(a+b\theta)\mathcal{O}_K=\gamma^2\mathcal{O}_K$ であったとして, $\prod_{(a,b)\in S}(a+b\theta)$ は $\mathcal{O}_K$ での平方数か?

AS-4) 元 $\gamma\in\mathcal{O}_K$ に対し, $\prod_{(a,b)\in S}(a+b\theta)=\gamma^2$ であったとして, $\mathbb{Z}[\theta]$ の元として平方数 (すなわち $\gamma\in\mathbb{Z}[\theta]$) か?

特殊な事情が成り立てば上記のいくつかは考慮する必要はない[22]. しかしながら, ここでは一般的状況下での考察を続ける.

AS-4) については, $\prod_{(a,b)\in S}(a+b\theta)=\gamma^2$, $\gamma\in K$ ならば, $\gamma\in\mathcal{O}_K$ であり, さらに $f'(\theta)\gamma\in\mathbb{Z}[\theta]$ が成り立つ (命題 A.2.5). 従って, 条件 (A.10), (A.11) を

$$f'(m)^2\prod_{(a,b)\in S}(a+bm)\quad \text{は}\ \mathbb{Z}\ \text{において平方数}, \tag{A.20}$$

$$f'(\theta)^2\prod_{(a,b)\in S}(a+b\theta)\quad \text{は}\ \mathbb{Z}[\theta]\ \text{において平方数} \tag{A.21}$$

に置き換えればよい. f, m が base m 法で構成されているならば, $1<f'(m)<n$ であり, $\gcd(f'(m),n)=1$ を仮定することができる (互いに素でないならば n の素因数分解ができる!).

以上により, AS-4) は容易に乗り越えられることがわかった.

従って, あとは, AS-1) ~ AS-3) を乗り越え, 条件 (A.19), (A.21) を満たすような S を見つければよいのだが, これには平方剰余記号を用いて"あたり"をつける方法が有効である ([1]). 次節でそのアイデアを説明する. 本節残りでは, 次節で必要な定理を紹介する. これは AS-1)~AS-4) の数学的構造を述べた定理で, これらの障壁がある意味でそれほど大きなものではないことを主張している.

[22] $\mathcal{O}_K=\mathbb{Z}[\theta]$ ならば, AS-1), AS-4) は考慮する必要はない. さらに, $\mathcal{O}_K$ の類数が 1 ($\Leftrightarrow$ $\mathcal{O}_K$ において, 素元分解が可能) ならば, AS-2) は考慮しなくてよい. 単数群の基底が明示的にわかっているのならば, AS-4) も無用である.

V を次のように定義する.

$$V = \left\{\beta \in K^{\times} \mid l_{\mathfrak{p}}(\beta) \equiv 0 \bmod 2 \text{ for all primes } \mathfrak{p} \subset \mathbb{Z}[\theta]\right\}.$$

$l_{\mathfrak{p}}$ は準同型なので明らかに $K^{\times 2} \subset V$ であり, $V/K^{\times 2}$ は $\mathbb{F}_2$ 上のベクトル空間である. この次元について, 次のことがわかる.

定理 A.2.6. n, d を, $d \geq 2$, $n > d^{2d^2}$ を満たす整数とする. また, f, m を base m 法 (A.2.2 節) で作成した多項式, 整数とする. $K = \mathbb{Q}(\theta)$, V は上記のとおりとするとき, 次が成り立つ.

$$\dim_{\mathbb{F}_2} V/K^{\times 2} < \frac{\log n}{\log 2}.$$

Proof. . 主張は言い換えると $[V : K^{\times 2}] < n$ となる. 以下これを示す.

$$W = \{\gamma \in K \mid \gamma\mathcal{O}_K = \mathfrak{a}^2 \text{ for some fractional } \mathcal{O}_K\text{-ideal } \mathfrak{a}\}$$

とおく[23]. 代数的整数環 $\mathcal{O}_K$ においては

$$W = V_{\mathcal{O}_K} = \{\beta \mid l_{\mathfrak{p},\mathcal{O}_K}(\beta) \equiv 0 \bmod 2 \text{ for all primes } \mathfrak{p} \subset \mathcal{O}_K\}$$

が成り立つから, 命題 A.2.4 により,

$$V \supset W, \quad [V : W] \leq [\mathcal{O}_K : \mathbb{Z}[\theta]]$$

が成り立つ[24]

$\mathcal{O}_K^{\times}$ を単数群とし, $Y = \mathcal{O}_K^{\times} K^{\times 2}$ とおけば, 次のような包含関係となる[25].

$$V \supset W \supset Y \supset K^{\times 2}.$$

W/Y を考える. $\mathcal{C}_K$ を K のイデアル類群[26] とするとき, 次の写像を考える.

$$W \longrightarrow \mathcal{C}_K, \qquad \gamma \text{ where } \gamma\mathcal{O}_K = \mathfrak{a}^2 \ \mapsto \ \text{class of } \mathfrak{a}.$$

[23] K の部分集合 $\mathfrak{a}$ が, $\mathcal{O}_K$-加群であって, ある $\mu(\neq 0) \in \mathcal{O}_K$ に対して $\mu\mathfrak{a} \subset \mathcal{O}_K$ となるとき, $\mathfrak{a}$ を K の分数イデアル (fractional ideal) という. $\mathcal{O}_K$ の通常のイデアルは分数イデアルである.

[24] 命題 A.2.4 において $A = \mathbb{Z}[\theta]$, $B = \mathcal{O}_K$ とすればよい.

[25] この包含関係は AS-1)∼AS-4) に対応している.

[26] $\mathcal{C}_K = \{$ 分数イデアル $\}/\{$ 単項分数イデアル $\}$. 有限群であり, その位数を K の**類数**という.

γがこの写像の核に含まれる元とすると, 対応する分数イデアル$\mathfrak{a}$は単項イデアル(γ')である. このとき, $\mathcal{O}_K^\times$の元ξが存在して$\gamma\xi = \gamma'^2$でなければならない. すなわち$\gamma \in Y$となる. 逆にYが核に含まれることは自明であるから, 埋め込み$W/Y \hookrightarrow \mathcal{C}_K$を得る. 特に$K$の類数を$h = \#\mathcal{C}_K$とすれば,

$$[W : Y] \leq h$$

が成り立つ.

次に, $Y/{K^\times}^2 \cong \mathcal{O}_K^\times/{\mathcal{O}_K^\times}^2$であり, 右辺の$\mathbb{F}_2$上の次元は単数群の (free-part の) 階数 $+1$ である ($+1$ は 1 のべき根の群からの影響). よって Dirichlet の単数定理 (定理 A.2.2) により, sをKの虚共役の個数の半分とするとき

$$[Y : {K^\times}^2] = 2^{d-s}$$

となる ($d = |K : \mathbb{Q}|$).

以上をあわせて次を得る.

$$[V : {K^\times}^2] \leq [\mathcal{O} : \mathbb{Z}[\theta]] \cdot h \cdot 2^{d-s}.$$

Δ_K を K の判別式とする. このとき類数について次が成り立つことが知られている.

$$h \leq M_K \cdot \frac{(d - 1 + \log M_K)^{d-1}}{(d-1)!},$$

ただし, $M_K = (d!/d^d)(4/\pi)^s\sqrt{|\Delta_K|}$ は Minkowski 定数 ([26]) である. Δ を多項式 f の判別式とすると次が成り立つ.

$$M_K \leq \sqrt{|\Delta_K|} \leq \sqrt{|\Delta_K|} \cdot [\mathcal{O}_K : \mathbb{Z}[\theta]] = \sqrt{|\Delta|} < d^d n^{1-3/(2d)}.$$

ただし, 等号は [8], 最後の不等式は式 (A.15) による.

条件 $n > d^{2d^2}$, および $d \geq 2$ により

$$d - 1 + d\log d < (3/2d)\log n, \quad 2d \cdot (2\log n)^{d-1} < n^{3/(2d)}$$

が成り立つことがわかる. 以上を全てまとめると

$$
\begin{aligned}
&[V : K^{\times 2}] \\
&\leq [\mathcal{O}_K : \mathbb{Z}[\theta]] \cdot \frac{d!}{d^d} \left(\frac{4}{\pi}\right)^s \sqrt{|\Delta_K|} \cdot \frac{\left(d-1+\log\sqrt{|\Delta|}\right)^{d-1}}{(d-1)!} \cdot 2^{d-s} \\
&= \frac{\sqrt{|\Delta|}}{d^{d-1}} \cdot 2^d \cdot \left(d-1+\log\sqrt{|\Delta|}\right)^{d-1} \cdot \left(\frac{2}{\pi}\right)^s \\
&< n^{1-3/(2d)} \cdot d \cdot 2^d \cdot \left(d-1+d\log d+\left(1-\frac{3}{2d}\right)\log n\right)^{d-1} \\
&< n^{1-3/(2d)} \cdot 2d \cdot (2\log n)^{d-1} \\
&< n,
\end{aligned}
$$

となり主張が得られる.

平方剰余記号

まず次の補題を示す.

補題 A.2.1. k, r を非負整数とし, E を $\mathbb{F}_2$ 上の k 次元ベクトル空間とする. E から独立, 一様に $k+r$ 個の元を選択したとき, これらの元が E 全体を生成する確率は少なくとも $1-2^{-r}$ である.

Proof. . いくつかのベクトルが E を生成しないための必要十分条件はこれらのベクトルが E のある超平面 ($k-1$ 次元部分空間) に含まれることである. さらに, E の任意の超平面 H に対し, $k+r$ 個のベクトルが H に含まれる確率は 2^{-k-r} である. E に含まれる超平面は 2^k-1 個 (逆に 1 次元部分空間の個数を考えればよい) であるから, $k+r$ 個のベクトルが少なくとも一つの超平面に含まれる確率は高々

$$(2^k-1)2^{-k-r} < 2^{-r}$$

となる. よって主張を得る.

記号は前のとおりとする. K の元が平方数であることを確かめるのに平方剰余記号を利用する方法 ([1]) を紹介する. まず, 平方数であれば, 平方剰余記号の値は 1 となることを確かめておく:

命題 A.2.8. S を互いに素な整数の組 (a,b) の集合であって, $\prod_{(a,b)\in S}(a+b\theta)$ が K において平方であるようなものとする. さらに q を奇素数とし, $s \in R(q)$ を

$$\begin{gathered}\text{各 } (a,b)\in S \text{ に対し, } a+bs \not\equiv 0 \bmod q,\\ f'(s) \not\equiv 0 \bmod q\end{gathered}$$

を満たす数とする. このとき q に対する平方剰余記号 $\left(\frac{*}{q}\right)$ について

$$\prod_{(a,b)\in S}\left(\frac{a+bs}{q}\right) = 1$$

が成り立つ.

Proof. . 環準同型 $\mathbb{Z}[\theta] \to \mathbb{F}_q,\ \theta \mapsto s \bmod q$ を考える. この核を $\mathfrak{q}$ と置くと, $\mathfrak{q}$ は (q,s) に対応する 1 次の素イデアルであった. 写像 $\mathbb{Z}[\theta]\setminus\mathfrak{q} \to \mathbb{F}_q^\times$ と, 平方剰余記号 $\mathbb{F}_q^\times \to \{\pm 1\}$ との合成を $\chi_\mathfrak{q}$ とおく: $\chi_\mathfrak{q}(a+b\theta) = \left(\frac{a+bs}{q}\right)$.

仮定より $\prod_{(a,b)\in S}(a+b\theta)$ は K において平方であるから, (A.21) で見たように, ある $\delta \in \mathbb{Z}[\theta]$ が存在して

$$f'(\theta)^2 \prod_{(a,b)\in S}(a+b\theta) = \delta^2$$

となる. また仮定より左辺は $\mathfrak{q}$ に含まれないから, 両辺に $\chi_\mathfrak{q}$ を適用することで主張を得る.

実は**この命題の逆も成り立つ**: $\beta \in \mathbb{Z}[\theta]\setminus\{0\}$ が $2\beta \notin \mathfrak{q}$ を満たす全ての 1 次の素イデアル $\mathfrak{q}$ に対し, $\chi_\mathfrak{q}(\beta) = 1$ を満たすならば, β は K において平方数である.

"全て"という条件は"有限個を除いて全て"に緩めることもできる.

しかしながら実際の計算には無限個の素イデアルに対して平方剰余性を確かめるわけにはいかない. 有限個の $\chi_\mathfrak{q}$ を" うまく" 選び, これらに対して平方剰余性を確かめれば高い確率で平方であることが言えるようにしなければならない.

実際には, rational, algebraic の"線型代数部分"とまとめて以下のように行う:

rational, algebraic それぞれの factor base とその個数を

$$FB_R := \{p \mid p:\ \text{prime},\ p \le y\},$$
$$B = \pi(y) = \#FB_R,$$

$$FB_A = \{(p,r) \mid p:\ \text{prime},\ p \le y,\ r \in R(p)\},$$

$B' = \#FB_A$,

とおく. さらに

$B'' = [3(\log n)/\log 2]$

とおき, 組 (q,s), q : 素数, $s \in R(q)$ であって, $q > y$, $f'(s) \not\equiv 0 \bmod q$ を満たすものを B'' 個探す[27] : $(q_1, s_1), \ldots, (q_{B''}, s_{B''})$. またこれらがなす集合を Q とおく.

0 でない整数 a に対し, $\mathrm{s}(a) = 0$, if $a > 0$, $= 1$ if $a < 0$ と定める.

集合 U の元 (a,b) であって, $a + bm$ が y-smooth, かつ $a + b\theta$ が y-smooth であるようなものの集合を $T = T_1 \cap T_2$ とおく:

$$T = \{(a,b) \mid \gcd(a,b) = 1,\ |a| \le u,\ 0 < b \le b, (a+bm)\mathcal{N}(a+b\theta) : y\text{-smooth}\}\,.$$

T からベクトル空間 $\mathbb{F}_2^B$ への写像 e_r, および $\mathbb{F}_2^{B'}$ への写像 e_a を

$$\begin{aligned} e_r &: T \longrightarrow \mathbb{F}_2^B, \quad (a,b) \mapsto (\ \mathrm{ord}_p(a+bm) \bmod 2\ )_{p \in FB_R}, \\ e_a &: T \longrightarrow \mathbb{F}_2^{B'}, \quad (a,b) \mapsto (\ e_{p,r}(a+b\theta) \bmod 2)_{(p,r) \in FB_A} \end{aligned}$$

と定義する. $(q,s) \in Q$, $(a,b) \in T$ に対し, $e_{q,s}(a,b)$ を, $\left(\frac{a+bs}{q}\right) = 1$ ならば 0, そうでなければ 1 と定め[28], T から $\mathbb{F}_2^{B''}$ への写像 e_Q を以下のように定義する.

$$e_Q : T \longrightarrow \mathbb{F}_2^{B''}, \quad (a,b) \mapsto (\ e_{q,s}(a,b))_{(q,s) \in Q}.$$

最後にこれらを用いて T からベクトル空間 $\mathbb{F}_2^{1+B+B'+B''}$ への写像 e を次で定義する.

$$\begin{aligned} e &: T \longrightarrow \mathbb{F}_2^{1+B+B'+B''}, \\ (a,b) &\mapsto (\ \mathrm{s}(a+bm),\ e_r(a,b),\ e_a(a,b),\ e_Q(a,b)\ )\,. \end{aligned} \tag{A.22}$$

このとき, $\#T > 1 + B + B' + B''$ であるならば $\{e(a,b)\}_{(a,b)\in T}$ は一次従属となる. よって空でない部分集合 $S \subset T$ が存在して $\sum_{(a,b)\in S} e(a,b) = 0$ (0 ベクトル) となる. この S に対して

$$f'(\theta)^2 \prod_{(a,b)\in S} (a + b\theta)$$

は高い確率で平方になることを以下に示す.

[27] 単純に条件を満たす素数を小さい順に探していけばよい.

[28] $\left(\frac{a+bs}{q}\right) = (-1)^{e_{q,s}(a,b)}$.

V を上述のように定める. $\mathfrak{q}$ を $\mathbb{Z}[\theta]$ の 1 次の素イデアルで $f'(\theta) \notin \mathfrak{q}$ なるものとする. このとき $\chi_{\mathfrak{q}}$ は準同型 $V/K^{\times 2} \to \{\pm 1\}$ を導くが, これを再び $\chi_{\mathfrak{q}}$ と書く.

Čebotarev の密度定理 ([26] Chapter VIII, Section 4) により, $\mathfrak{q}$ が $\mathbb{Z}[\theta]$ の 1 次の素イデアルで $f'(\theta) \notin \mathfrak{q}$ なるもの全体を動くとき, $\chi_{\mathfrak{q}}$ はノルムで順序付けしたとき, $\mathrm{Hom}(V/K^{\times 2}, \{\pm 1\})$ 上, 漸近的に均等に分布することがわかる.

このことは B'' 個の $\chi_{\mathfrak{q}}$ が $V/K^{\times 2} \to \{\pm 1\}$ 上のランダム準同型とみなせることを意味し, よって定理 A.2.6, 補題 A.2.1 により高い確率, 少なくとも確率 $1 - (1/n^2)$ 以上でこれらの関数 $\chi_{\mathfrak{q}}$ が $\mathrm{Hom}(V/K^{\times 2}, \{\pm 1\})$ を生成することを示唆するものとなる.

これらの関数が生成元であった場合, $\beta \in V$ が平方であるための必要十分条件は各 $\chi_{\mathfrak{q}} \in Q$ に対し, $\chi_{\mathfrak{q}}(\beta) = 1$ となることとなる. これがアルゴリズムの根拠である.

平方計算

平方剰余記号の利用により, 平方数であるものの抽出は可能となったが, これはある意味で平方数であることの間接的な証拠でしかなく, 実際に平方を行うにはさらに計算が必要である[29].

$\mathbb{Z}[\theta]$ において平方である可能性が非常に高い元 $\gamma = f'(\theta)^2 \prod_{(a,b)\in S}(a+b\theta)$ が与えられたとする.

q を奇素数であって, $f \bmod q$ が $\mathbb{F}_q[x]$ で既約であるものを見つけることができたとする. このとき図式

$$\begin{array}{ccc} \mathbb{Z}[x]/(f(x)) & \xrightarrow{\sim} & \mathbb{Z}[\theta] \\ {\scriptstyle \bmod q}\downarrow & & \downarrow{\scriptstyle \bmod q} \\ \mathbb{F}_q[x]/(f \bmod q) & \xrightarrow{\sim} & \mathbb{Z}[\theta]/q\mathbb{Z}[\theta] \end{array}$$

は可換で, 仮定から $\mathbb{F}_q[x]/(f \bmod q)$ は q^d 個の元を持つ有限体である. よってイデアル

$$\mathfrak{q} = q\mathbb{Z}[\theta] = \left\{ \sum_{i=0}^{d-1} \alpha_i \theta^i \;\middle|\; \alpha_i \in \mathbb{Z},\ \alpha \equiv 0 \bmod q \right\}$$

は $\mathbb{Z}[\theta]$ の d 次の素イデアルである. さらに $f \bmod q$ の既約性から, $f'(\theta) \notin \mathfrak{q}$ もわかる. また, 各 $(a,b) \in S$ に対し, $\gcd(a,b) = 1$ であるから $a + b\theta \notin \mathfrak{q}$ であり, よってこれらの積である γ も $\mathfrak{q}$ に含まれないことがわかる.

そこで, q を法として γ の (θ の多項式としてみたときの) 係数を考え, 有限体 $\mathbb{Z}/\mathfrak{q}$ における平方根の計算方法 ([36](pp.169-198) など) により, $\delta_0 \bmod \mathfrak{q} \in \mathbb{Z}/\mathfrak{q}$ であって,

$$\delta_0^2 \gamma \equiv 1 \bmod \mathfrak{q}$$

を計算する (δ_0 は符号の差を除き一意に定まる)[30]. δ_0 は γ の ($\mathfrak{q}$ を法とした) 平方の逆元であるが, これは以下に述べる Newton 法 (または Hensel lift) において除算を排除するためである.

---

[29] rational 部分の平方計算 ($f'(m)\prod(a+bm)$ の平方) は, n を法とした計算でよい. また篩の過程で各 $a+bm$ の因子はわかっているので, 容易に実行できる.

[30] もし平方が見つからない, すなわち $\gamma \bmod \mathfrak{q}$ が $\mathbb{Z}/\mathfrak{q}$ において平方でないならば, 平方剰余記号を増やして平方数の根拠を探す部分を強化しなければならない.

δ_0 から始めて, 関係式

$$\delta_j \equiv \frac{\delta_{j-1}(3-\delta_{j-1}^2\gamma)}{2} \bmod \mathfrak{q}^{2^j}$$

により,

$$\delta_j^2\gamma \equiv 1 \bmod \mathfrak{q}^{2^j}$$

を満たす $\delta_1, \delta_2, \ldots$ を求める. γ の $\mathbb{Z}[\theta]$ における真の平方を ν とおき, ν の (θ の多項式と見たときの) 係数の絶対値の上限を Γ とおくとき (a, b の大きさ, S の個数などでおおよそ定めることができる), q^{2^j} が 2Γ を超える程度まで上記手順を繰り返せば, 平方根 ν は

$$\nu = \delta_j\gamma \bmod \mathfrak{q}^{2^j}$$

により計算できることになる.

実際の計算においては, d が奇数の場合, Couveignes([30]) により, 効率的な計算方法が提案されている.

この Newton 法を用いる手順の最初に, q は $f \bmod q$ が既約となるような奇素数と仮定した. このような素数が実際に簡単に見つかるのかどうか, そもそも存在するのかということが最後の問題として浮上する.

まず, 次の命題を示す[31].

命題 A.2.9. $f \in \mathbb{Z}[x]$ を d 次既約 monic 多項式とする ($d > 1$). このとき, 全素数の中で, $f \bmod q$ が $\mathbb{F}_q[x]$ において相異なる 2 次以上の既約因子に分解するような素数 q 全体のなす集合に対し, 密度は存在し, 少なくとも $1/d$ 以上である.

[31] ここで, 必要な群論の言葉を復習しておく.

群 G と集合 X に対し, 写像 $f : X \times G \to X$ が与えられているとし, $f(x, g) = x^g$ とかくことにする (べき乗ではない). これらが次を満たすとき, G は X に**作用**しているという.

- $x^e = x$ (e は G の単位元).
- $x^{gh} = (x^g)^h$ ($g, h \in G$).

$x, y \in X$ に対し, ある $g \in G$ が存在して $y = x^g$ となるとき, x と y は (G-) 同値という. これは同値関係となり, その同値類は**軌道**と呼ばれる. X 自身が一つの軌道となるとき, G は X に推移的に作用するという.

$x \in X$, $g \in G$ に対し, $x^g = x$ となるとき x は g の固定点という.

$g \in G$ に対し, $G \to G$, $h \mapsto ghg^{-1}$ を自己共役写像という. 自己共役写像により, G は G 自身に作用しているとみることができる. このとき, $h \in G$ が含まれる G-軌道を, h を含む G の共役類という. G の部分集合 C が任意の自己共役写像で自分自身に写される, すなわち任意の $g \in G$ に対し, $gCg^{-1} \subseteq C$ となるとき, C は共役について閉じているなどという.

Proof. G を f の $\mathbb{Q}$ 上の Galois 群とし, f の零点の集合 Ω 上の置換群とみなす. f の判別式を割らない素数 q に対し, G における Frobenius 自己同型の共役類が一意に定まり, 一つの代表元を σ_q とおくと, $f \bmod q$ の既約分解における既約因子の次数は σ_q の巡回置換への分解における巡回置換の長さと対応する[32]. この対応から, $f \bmod q$ の既約分解に 1 次因子があらわれることと, σ_q が固定点を持つことが同値となり, 従って, 主張を示すには固定点を持たない σ_q(の共役類) に対応する素数 q の集合の密度を考察すればよい.

Čebotarev の密度定理 (定理 A.2.5) により (やや一般化して), 共役に関して閉じている部分集合 $C \subset G$ に対し, $\sigma_q \in C$ なる素数 q の集合の密度は存在して $\#C/\#G$ と一致することがわかる.

次の補題により, 固定点を持たない G の元は少なくとも $\#G/d$ 個存在することがわかり, よって求める密度は $(\#G/d)/\#G = 1/d$ 以上となることがわかる.

補題 A.2.2. G を有限群とし, 有限集合 Ω $(\#\Omega = d > 1)$ に推移的に作用しているものとする. このとき少なくとも $(\#G)/d$ 個の G の元は Ω への作用において固定点を持たない.

Proof. (P.J.Cameron, A.M.Cohen による ([7]))

一般に, 有限群 G が有限集合 X に作用するとする. $X^\sigma = \{x \in X \mid x^\sigma = x\}$ (σ の固定点の集合) と置くとき, さらに X 上にちょうど i 個の固定点を持つ元 $\sigma \in G$ の個数を f_i^X とする. このとき, X の軌道の個数は,

$$\frac{1}{\#G}\sum_{\sigma\in G}\#X^\sigma = \frac{1}{\#G}\sum_{i=0}^{d}\left(\sum_{\substack{\sigma\in G\\ \#X^\sigma=i}} i\right) = \frac{1}{\#G}\sum_{i=0}^{d} i f_i^X$$

で与えられる ([63] 第 3 章 §2 定理 2.4). $X = \Omega$ のとき, 仮定から軌道の個数は一つなので,

$$\#G = \sum_{i=0}^{d} i f_i^\Omega \tag{A.23}$$

が成り立つ. 次に $X = \Omega \times \Omega$ に適用 (G は成分ごとに作用) すると, $\sigma \in G$ の Ω 上の

[32] 「有限次代数体」節と, σ_q の根の上の作用から示すことができる.

固定点の個数が i のとき, $\Omega \times \Omega$ 上の固定点の個数は i^2 で与えられるから,

$$\sum_{i=0}^{d^2} i f_i^{\Omega\times\Omega} = \sum_{\substack{i=0 \\ i:\text{square}}}^{d^2} i f_i^{\Omega\times\Omega} = \sum_{i=0}^{d} i^2 f_i^{\Omega}$$

となる. 一方, $X = \Omega \times \Omega$ の対角成分の集合は G の作用で閉じており, また, $d > 1$ より X は対角成分以外の元を含むので, 軌道は 2 以上存在する. 従って,

$$2\#G \leq \sum_{i=0}^{d} i^2 f_i^{\Omega} \tag{A.24}$$

が成り立つ. さらに, f_i の定義から, 明らかに

$$\#G = \sum_{i=0}^{d} f_i^{\Omega}. \tag{A.25}$$

これらの式 (A.23), (A.24), (A.25) に対し, $(A.24) - (d+1)(A.23) + d(A.25)$ を考える.

$i^2 - (d+1)i + d = (i-1)(i-d)$ は $1 \leq i \leq d$ に対し, 0 または負の数であり, $i = 0$ に対してはちょうど d と一致する. 故に,

$$\sum_{i=0}^{d} (i^2 - (d+1)i + d) f_i^{\Omega} = d f_0 + \sum_{i=1}^{d} (i^2 - (d+1)i + d) f_i^{\Omega} \leq d f_0^{\Omega},$$

一方,

$$\#G = (2 - (d+1) + d)\#G \leq \sum_{i=0}^{d} (i^2 - (d+1)i + d) f_i^{\Omega}.$$

よって, $\#G/d \leq f_0^{\Omega} = \Omega$上に固定点を持たない G の元の個数 を得る.

さて, d 次 monic 既約多項式 $f \in \mathbb{Z}[x]$ の Galois 群は対称群 S_d(位数 $d!$) の部分群であるが, 高い確率で全体に一致する ([15]). 一方, 命題 A.2.9, 証明により, $f \bmod q$ が既約となる素数 q 全体の集合は, 対応する Frobenius 自己同型 σ_q が長さ d の巡回置換であるような素数全体の集合と一致する. f の Galois 群が S_d である場合, 長さ d の巡回置換は $(d-1)!$ 個存在するので, Čebotarev の密度定理 (定理 A.2.5) (または命題 A.2.9, 証明) により, この集合の密度は $(d-1)!/d! = 1/d$ となる. d は (n に比べ) かなり小さく選択されているので, これはかなり大きいと考えることができ, 従って, ほとんどの n に対し, f を構成したとき, $f \bmod q$ が既約となる素数 q を探すことは困難ではないと考えられる.

しかしながら, 場合によっては全ての奇素数 q に対して $f \bmod q$ が可約, あるいは $f \bmod q$ が既約となるものを見つけることが困難であることも生じ得る. その場合, f を取り替えるなどの対応策が考えられるが, 計算量的に f の係数は小さいほうが都合がよく, 取替えを避けたい場合もある. 以下では必ずしも $f \bmod q$ が既約でない場合の手法を述べる ([54]).

Newton 法と同様の手法であるが, 既約の場合と異なるのは, まず $q\mathbb{Z}[\theta]$ が素イデアルではないことである. よって δ_0 は符号の差を除いても一意ではない. この場合, q を含む各素イデアルに対し平方 (の逆元) を求め, 中国人剰余定理によりあわせることで $q\mathbb{Z}[\theta]$ を法とした平方 (の逆元) を求める. q を含む素イデアルが t 個存在する場合には初期値となる δ_0 は 2^{t-1} 個存在する. これらに対し Newton 法を適用していくのだが, パラメータをうまく選択することで t を十分小さくすることができ ($t \leq d/2$ など), 数体篩全体の計算量からみて, 大きな影響がないことを示すことが出来る.

ただし, 上記手法においても素数 q は任意でよいわけではなく, $\gamma = f'(\theta)^2 \prod (a+b\theta)$ と素であるように選択する必要性は残る (上述のように $f \bmod q$ が既約ならば自動的に γ と q は素となる). この条件は $f \bmod q$ が既約の場合を若干一般化することで満たすようにすることができる.

すなわち, $f \bmod q$ の既約因子が全て相異なり, かつ 2 次以上であればよい. 実際, $f \bmod q$ が squarefree(平方因子を持たない) ならば $f'(\theta)$ は q と素であり, さらに $f \bmod q$ が 1 次因子を持たないならば, q の上にある 1 次の素イデアルは存在せず, よって命題 A.2.6 により $a + b\theta$ は q と素, 従って γ は q と素となる.

さらに, このような q は命題 A.2.9 により, 十分存在することがわかる. 従って実際に数対篩法を実行する場合, $f \bmod q$, $q = 3, 5, 7, \ldots$ と調べていくことで容易に求める性質を持つ q を探すことができると考えられる.

A.2.3　数体篩法の手順

数体篩法の手順をまとめる.

【 数体篩法 】

正整数 n, およびパラメータ d, u, y であって $d > 1$, $n > d^{2d^2}$ を満たすものが与えられたとき, n の非自明な因子を見つけるか, n が素数であることを試みるアルゴリズムである.

Step 1. n が素数べきであるか否かを調べる ([32]). または y 以下の素数で割れるか否かを調べる. いずれかの場合にはその素因子を出力して終了する.

Step 2. base m 法により整数 m と, d 次 monic 多項式 $f(x) \in \mathbb{Z}[x]$ であって, $f(m) \equiv 0 \bmod n$ を満たすものを探す. f を $\mathbb{Z}[x]$ において既約因子に分解し ([31]), f が可約であるならば, 一つの非自明な因子 g に対し $g(m)$ が n の非自明な因子であるか否かを調べ, そうであるなら $g(m)$ を出力して終了する.

以下では f は既約とし, $\theta \in \mathbb{C}$ を f の一つの根とする. $\gcd(f'(m), n)$ を計算し, これが n の非自明な因子であるならばこれを出力して終了する.

Step 3. 篩により, 集合

$$T = \{(a,b) \in \mathbb{Z}^2 \mid \gcd(a,b) = 1,\ |a| \le u,\\ 0 < b \le u,\ (a+bm)N(a+b\theta) \text{ is } y\text{−smooth}\}$$

の要素を全て求める.

Step 4. $(a,b) \in T$ に対し, $\mathbb{F}_2$ 上のベクトル $e(a,b)$(定義は (A.22) 参照) を生成し, これらを列とする行列を構成する. この行列から, 列の非自明な線型従属関係式を見つける (Wiedemann coordinate recurrence algorithm [56] など). 見つからなかった場合はここで終了. 見つかった場合はその従属関係式にあらわれる (a,b) 全体のなす部分集合を $S \subset T$ とおく.

Step 5. 代数的整数 $\gamma = f'(\theta)^2 \prod_{(a,b)\in S}(a+b\theta)$ を θ の d 次未満の多項式であらわし, その平方根 $\nu = \sum_{i=0}^{d-1} \alpha_i \theta^i$ $(\alpha_i \in \mathbb{Z})$ を計算する ([54] 参照). ここで平方根の計算に失敗したら終了する.

Step 6. $\xi^2 = f'(m)^2 \prod_{(a,b)\in S}(a+bm)$ なる $\xi \in \mathbb{Z}$ に対し, $\xi \bmod n$ を計算する.

Step 7. $\gcd(\xi - \sum_{i=0}^{d-1} \alpha_i m^i,\ n)$ を計算し, それが非自明な n の因子ならばそれを出力して終了する. そうでなければ T から S を取り除き, Step 4 に戻る.

ここでは rational, algebraic を同時に行う形で記述したが, 高速化のための最適値を考えると, smoothness bound (y) や, 検索範囲 (u) などのパラメータは rational, algebraic で異なるものを取る方がよい. 具体値などについては A.2.6 節で言及する.

A.2.4 特殊数体篩法

これまでの議論では, 多項式 f の根の一つ θ から生成される整環 $\mathbb{Z}[\theta]$ を一般的に考察してきた. A.2.2「algebraic part」節での条件 AS-1)∼AS-4) のところでも述べたが, $\mathbb{Z}[\theta]$ に特殊な条件が成り立てば, 数体篩法における手順のいくつかは省略することができる. Pollard らによる最初の数体篩法のアイデア ([33]) では,

- $\mathbb{Z}[\theta]$ は素元分解環

なる仮定を置いた. この条件下では代数体上で直接, 素因数 (素元) 分解でき, 単数の情報が必要になること以外は有理数体上での手順とほぼ同様となる.

しかしながら上記条件は一般には成り立ちにくく, $n = c^k \pm s$ (c, s は小さい) の形の合成数で, 扱う代数体も特殊なものに限られる. ただし, このような形の合成数 (円分数など) に対しては最も強力な手法となっている. 上記条件付の分解法を特殊数体篩法, 一般的な状況下での分解法は, 特殊と区別する場合, 一般数体篩法と呼ばれる.

以下に最近の特殊/一般数体篩法による素因数分解の結果を表にまとめる[33]. 特殊な合成数については特殊数体篩法により, 800-bit 超が分解されており, 一般では (一般) 数体篩法により 576-bit の分解が報告されている.

	bit	type	method
1996/04	429	RSA-130$_d$	NFS
1998/09	615	$12^{167}+1$	SNFS
1999/02	462	RSA-140$_d$	NFS
1999/04	698	$(10^{211}-1)/9$	SNFS
1999/08	512	RSA-155$_d$	NFS
2000/11	773	$2^{773}+1$	SNFS
2002/01	522	$c158_d$ of $2^{953}+1$	NFS
2003/01	809	$M809$	SNFS
2003/03	529	RSA-160$_d$	NFS
2003/12	576	RSA-576	NFS

表 A.1: 特殊数体篩法と一般数体篩法の最近の結果

[33] Arjen K. Lenstra, SNFS versus (G)NFS and the feasibility of factoring a 1024-bit number with SNFS, Workshops on the state-of the-art of factoring large numbers (organized by CWI in Utrecht, December 12, 2003) の資料 (`http://db.cwi.nl/projecten/project.php4?prjnr=84`)

A.2.5 計算

数体篩では rational と algebraic の 2 種の篩を実施する必要がある.

rational sieve ある範囲の a, b に対し, $a+bm$ の形の整数の smooth 性を調べるのが篩である. b を固定して, a を動かしたとき, factor base に含まれる素数 p に対して p で割りきれる $a+bm$ は等差数列として現れる.

そこでまず, 各 $a_{\min} \leq a \leq a_{\max}$ に対応させた $\max - \min$ 個の配列 $W[i]$ を用意する. 初期値は 0 としておき, 各 p に対して, $a+bm$ が p で割れるならば対応する配列に $\lfloor \log p \rceil$ を加えていく. これは最初に割れる箇所さえ計算すれば, あとは p おきに加えていくことで実施できる.

factor base に含まれるすべての素数に対して上記手順を行った後, 値が, ある閾値を超えている配列に対応する $a+bm$ は, ある程度多くの素数で割り切れたということを意味し, smooth となる可能性の高い候補となる. さらに b の方も動かして多くの候補を集める.

これらの候補に対し, 実際に factor base の素数で素因数分解を実施し, smooth であるかどうかを確かめる[34].

algebraic sieve $N(a+b\theta) \equiv 0 \bmod p \Leftrightarrow a \equiv -br \bmod p$ for some $r \in R(p)$ であった. よって rational の場合と同様, b を固定, a を動かしたとき, (p, r) で割れる箇所は等差数列として現れる. よって手順もまったく rational の場合と同様である.

数体篩法では, 篩作業が終了したのち, 行列の計算を行い一次従属関係を作成しなければならない. 行列のサイズは factor base の個数などに依存して決まるが, 対象となる合成数のサイズが大きい場合, factor base も多く必要で, よって行列のサイズも非常に大きなものになる. 行列計算は基本的には Gauss 消去などで行うが, 扱う行列は疎 (要素が 0 のものが多い) であり, サイズを小さくして計算する方法などを用いることが可能である. また, 反復系のアルゴリズムとして知られている Lanczos 法を改良した方法も提案されている ([39]).

から抜粋. $M\nu = 2^{\nu} - 1$: Mersenne 数 RSA-$***$: RSA Factoring Challenge([46])

[34] 完全に smooth とならない場合でもうまく組み合わせて平方数をつくる手法もある ([28] など).

A.2.6 数体篩法の計算量

ここでは数体篩法の全ての処理に必要な計算量見積もりについて, 結果のみ述べることにする.

計算量を次の式を用いて表す.

$$L_x[a,b] = \exp(\ (b+o(1))(\log x)^a(\log\log x)^{1-a}\ ).$$

アルゴリズムへの入力を x としたとき, 計算量が $\log x$ の多項式であらわされるものを多項式時間アルゴリズムと呼ぶが, 上記式を用いると $a=0$ ($L_x[0,b] = \exp((b+o(1))(\log\log x)) = \log x \exp(b+o(1))$) の場合として表現できる. $a=1$ の場合は指数時間アルゴリズムということになる. よって計算量をこの式であらわした場合, a の値は 0 に近い方が計算量が"少ない", 素因数分解アルゴリズムに関して言えば, より優れた分解法であるということになる.

A.2.3 節で述べた数体篩法 (Adleman による平方剰余記号を用いた方法の場合) において, 目的の合成数を n とし,

$$\begin{aligned} &d = (3^{1/3}+o(1))(\log n/\log\log n)^{1/3}, \\ &1 < d^{2d^2} < n, \\ &u = y = L_n[1/3, (8/9)^{1/3}+o(1)] \end{aligned}$$

とおくとき, 高々

$$L_n[1/3,(64/9)^{1/3}+o(1)] = L_n[1/3, 1.92299\cdots+o(1)] \tag{A.26}$$

の計算量により, n の非自明な因子を出力するか, もしくは n が素数であると判定することが見積もられている.

さらに Coppersmith による改良案 ([11]) によれば, 計算量は

$$L_n[1/3, 1.901] \tag{A.27}$$

となる. ちなみに, 2 次篩法 (MultiPolynomial 版 [50]) の計算量は $L_n[1/2, 1.020]$, 特殊数体篩法は $L_n[1/3, 1.526\ldots]$ となることも知られている (前節でも述べたが特殊数体篩法が適用できる合成数は極めて特殊なものに限られることに注意).

実際の合成数に対して, 各種パラメータをどの程度の大きさで取ればよいかについて, 様々な試算が行われている. 1024-bit RSA 型合成数 ($n=pq$, p, q は同じビッ

ト長の素数) は現在公開鍵暗号の鍵として広く用いられており, このサイズの合成数の素因数分解は多くの研究者の興味を引いている.

rational sieve における smoothness bound(factor base に含まれる素数の上限) を y_r, algebraic sieve に対する bound を y_a (A.2.2 節ではともに y と書いた) とするとき, [49](Crypto2003 版の改訂版) では $y_r \approx 3.5 \cdot 10^9$, $y_a \approx 2.6 \cdot 10^{10}$ を採用している.

また, [37] ではより理論的な考察のもと, 1024-bit に対しては $y_r \approx 1.2 \cdot 10^{10}$, $y_a \approx 5.5 \cdot 10^{10}$ が必要であることを導いている.

現在, 1024-bit 合成数に成功した例は皆無であり, これらのパラメータサイズで成功するか否かの実証は無論行われていない.

A.2.7 数体篩法の改良

数体篩法の改良や発展形が提案されているが, ここでは主なものについて文献を紹介するのみとする.

Multiple Polynomial NFS ([20]).

対象となる合成数 n に対し, 複数の多項式 $f_j(x)$ $(j = 1, 2, \ldots)$ と共通の m であって, $f_j(m) \equiv 0 \bmod n$ となるものを選ぶ方法. 各多項式に対し, その 1 つの根を付加した体の上で数体篩法と同様の手順を行う.

この方法では, 多項式の係数を小さくとることが可能となる場合があり, 高速化に貢献する.

Lattice Sieve ([30]).

factor base の素数を大きさによって大中小などに分割する. 中ぐらいの大きさの素数を一つ固定し, 篩対象の集合 $\{a + bm\}$ のうち, その素数の倍数であるものからなる部分集合を取り出し, これに対し篩を行うというのが lattice 篩である. その部分集合ははじめから factor base に含まれるひとつの素数で割れることがわかっているため, "あたり"を付けやすいというのがアイデアである.

このアイデアをさらに改良した高速化手法も提案されている ([21]).

付 録B　Verilog記述言語による評価詳細

B.1 デリバリセルの設計

$\lfloor \log_2 p \rfloor$(ソースコード中 log_p) は 4 ビットあればよいことは，6.4 節で既に述べた．以下に $\lfloor \log_2 p \rfloor$ と log_p の対応を示す．

$\lfloor \log_2 p \rfloor$	log_p[3:0] (二進表現)
19	0010
20	0011
21	0100
22	0101
23	0110
24	0111
25	1000
26	1001
27	1010
28	1011
29	1100
30	1101
31	1110
32	1111

すなわち $\lfloor \log_2 p \rfloor =$ log_p $+ 17$ としている．

また，L は delivery line 上のどの delivery cell かを示すインデックスである．

```
module delivery_cell
(
        clk ,
        g_s_in ,
        g_c_in ,
        valid_in ,
        l_in ,
        log_p_in ,
        g_s_out ,
        g_c_out ,
        valid_out ,
        l_out ,
        log_p_out
) ;

input   clk ;
input   [9:0] g_s_in ;
input   [9:1] g_c_in ;
input   valid_in ;
```

```
input   [11:0] l_in ;
input   [3:0] log_p_in ;
output  [9:0] g_s_out ;
output  [9:1] g_c_out ;
output  valid_out ;
output  [11:0] l_out ;
output  [3:0] log_p_out ;

reg     [9:0] g_s_out ;
reg     [9:1] g_c_out ;
reg     valid_out ;
reg     [11:0] l_out ;
reg     [3:0] log_p_out ;

wire    g_can_be_added = valid_in & ( l_in[11:0] == `L ) ;

always  @ ( posedge clk )
begin
        g_s_out[9] <= g_can_be_added ?
                ( g_s_in[9] ? 1'b1 : g_c_in[9] ) :
                g_s_in[9] ;
        g_s_out[8] <= g_can_be_added ?
                ( ( g_s_in[9] & g_c_in[9] & g_s_in[8] ) ?
                        1'b1 : ( g_s_in[8] ^ g_c_in[8] ) ) :
                 g_s_in[8] ;
        g_s_out[7:5] <= g_can_be_added ?
                ( g_s_in[7:5] ^ g_c_in[7:5] ) : g_s_in[7:5] ;
        g_s_out[4] <= g_can_be_added ?
                ( ~ ( g_s_in[4] ^ g_c_in[4] ) ) : g_s_in[4] ;
        g_s_out[3:1] <= g_can_be_added ?
                ( g_s_in[3:1] ^ g_c_in[3:1] ^ log_p_in[3:1] ) :
                g_s_in[3:1] ;
        g_s_out[0] <= g_can_be_added ?
                ( ~ ( g_s_in[0] ^ log_p_in[0] ) ) : g_s_in[0] ;

        g_c_out[9] <= g_can_be_added ?
                ( ( g_s_in[9] & g_c_in[9] ) ?
                        1'b1 : ( g_s_in[8] & g_c_in[8] ) ) :
                g_c_in[9] ;
        g_c_out[8:6] <= g_can_be_added ?
                ( g_s_in[7:5] & g_c_in[7:5] ) : g_c_in[8:6] ;
        g_c_out[5] <= g_can_be_added ?
                ( g_s_in[4] | g_c_in[4] ) : g_c_in[5] ;
        g_c_out[4:2] <= g_can_be_added ?
                ( ( g_s_in[3:1] & g_c_in[3:1] ) |
                        ( log_p_in[3:1] & g_s_in[3:1] ) |
                        ( g_c_in[3:1] & log_p_in[3:1] ) ) :
```

```
                g_c_in[4:2] ;
        g_c_out[1] <= g_can_be_added ?
                ( g_s_in[0] | log_p_in[0] ) :
                g_c_in[1] ;

        valid_out <= valid_in ;
        l_out[11:0] <= l_in[11:0] ;
        log_p_out[3:0] <= log_p_in[3:0] ;
end

endmodule
```

B.2 インターリーブしたデリバリセルの組の設計 ($r = 4$ 個分)

以下のソースコードにおいて，log_p と L の扱いは B.1 節と同じである．

```
module interleaved_4_delivery_cells
(
        clk ,
        g0_s_in ,
        g0_c_in ,
        g1_s_in ,
        g1_c_in ,
        g2_s_in ,
        g2_c_in ,
        g3_s_in ,
        g3_c_in ,
        valid_in ,
        l_in ,
        log_p_in ,
        g0_s_out ,
        g0_c_out ,
        g1_s_out ,
        g1_c_out ,
        g2_s_out ,
        g2_c_out ,
        g3_s_out ,
        g3_c_out ,
        valid_out ,
        l_out ,
        log_p_out
) ;

input   clk ;
```

```
input   [9:0] g0_s_in ;
input   [9:1] g0_c_in ;
input   [9:0] g1_s_in ;
input   [9:1] g1_c_in ;
input   [9:0] g2_s_in ;
input   [9:1] g2_c_in ;
input   [9:0] g3_s_in ;
input   [9:1] g3_c_in ;
input   valid_in ;
input   [11:0] l_in ;
input   [3:0] log_p_in ;
output  [9:0] g0_s_out ;
output  [9:1] g0_c_out ;
output  [9:0] g1_s_out ;
output  [9:1] g1_c_out ;
output  [9:0] g2_s_out ;
output  [9:1] g2_c_out ;
output  [9:0] g3_s_out ;
output  [9:1] g3_c_out ;
output  valid_out ;
output  [11:0] l_out ;
output  [3:0] log_p_out ;

reg     [9:0] g0_s_out ;
reg     [9:1] g0_c_out ;
reg     [9:0] g1_s_out ;
reg     [9:1] g1_c_out ;
reg     [9:0] g2_s_out ;
reg     [9:1] g2_c_out ;
reg     [9:0] g3_s_out ;
reg     [9:1] g3_c_out ;
reg     valid_out ;
reg     [11:0] l_out ;
reg     [3:0] log_p_out ;

wire    g0_can_be_added = valid_in & ( l_in[11:0] == `L ) ;

always  @ ( posedge clk )
begin
        g0_s_out[9] <= g0_can_be_added ?
                ( g0_s_in[9] ? 1'b1 : g0_c_in[9] ) :
                g0_s_in[9] ;
        g0_s_out[8] <= g0_can_be_added ?
                ( ( g0_s_in[9] & g0_c_in[9] & g0_s_in[8] ) ?
                        1'b1 : ( g0_s_in[8] ^ g0_c_in[8] ) ) :
                 g0_s_in[8] ;
        g0_s_out[7:5] <= g0_can_be_added ?
```

```
                ( g0_s_in[7:5] ^ g0_c_in[7:5] ) : g0_s_in[7:5] ;
        g0_s_out[4] <= g0_can_be_added ?
                ( ~ ( g0_s_in[4] ^ g0_c_in[4] ) ) : g0_s_in[4] ;
        g0_s_out[3:1] <= g0_can_be_added ?
                ( g0_s_in[3:1] ^ g0_c_in[3:1] ^ log_p_in[3:1] ) :
                g0_s_in[3:1] ;
        g0_s_out[0] <= g0_can_be_added ?
                ( ~ ( g0_s_in[0] ^ log_p_in[0] ) ) : g0_s_in[0] ;

        g0_c_out[9] <= g0_can_be_added ?
                ( ( g0_s_in[9] & g0_c_in[9] ) ?
                        1'b1 : ( g0_s_in[8] & g0_c_in[8] ) ) :
                g0_c_in[9] ;
        g0_c_out[8:6] <= g0_can_be_added ?
                ( g0_s_in[7:5] & g0_c_in[7:5] ) : g0_c_in[8:6] ;
        g0_c_out[5] <= g0_can_be_added ?
                ( g0_s_in[4] | g0_c_in[4] ) : g0_c_in[5] ;
        g0_c_out[4:2] <= g0_can_be_added ?
                ( ( g0_s_in[3:1] & g0_c_in[3:1] ) |
                        ( log_p_in[3:1] & g0_s_in[3:1] ) |
                        ( g0_c_in[3:1] & log_p_in[3:1] ) ) :
                g0_c_in[4:2] ;
        g0_c_out[1] <= g0_can_be_added ?
                ( g0_s_in[0] | log_p_in[0] ) :
                g0_c_in[1] ;

        g1_s_out[9:0] <= g1_s_in[9:0] ;
        g1_c_out[9:1] <= g1_c_in[9:1] ;
        g2_s_out[9:0] <= g2_s_in[9:0] ;
        g2_c_out[9:1] <= g2_c_in[9:1] ;
        g3_s_out[9:0] <= g3_s_in[9:0] ;
        g3_c_out[9:1] <= g3_c_in[9:1] ;

        valid_out <= valid_in ;
        l_out[11:0] <= l_in[11:0] ;
        log_p_out[3:0] <= log_p_in[3:0] ;
end

endmodule
```