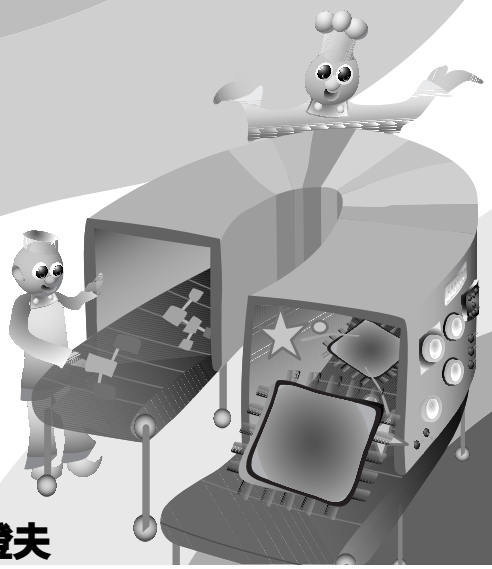
第4章

メモリ・データ処理のハードウェア化

RSA暗号回路の設計事例



森岡澄夫

ここでは，RSA暗号の主要部を例題に，ソフトウェア処理をハードウェア化していく過程を示す．これを例題に選んだ理由は，JPEGデコーダのような「ハードらしいハード」とは対照的に，ソフトウェア処理に近い発想でハードウェアを組むことになるからである．これまでハードウェア化が難しいとされてきた複雑な演算やデータ操作を行うものだが，暗号のみならず画像処理・画像認識でもこの範ちゅうに入る処理が多々あるので，これからハードウェア化のニーズが急に高まっていく中，注目に値する．（筆者）

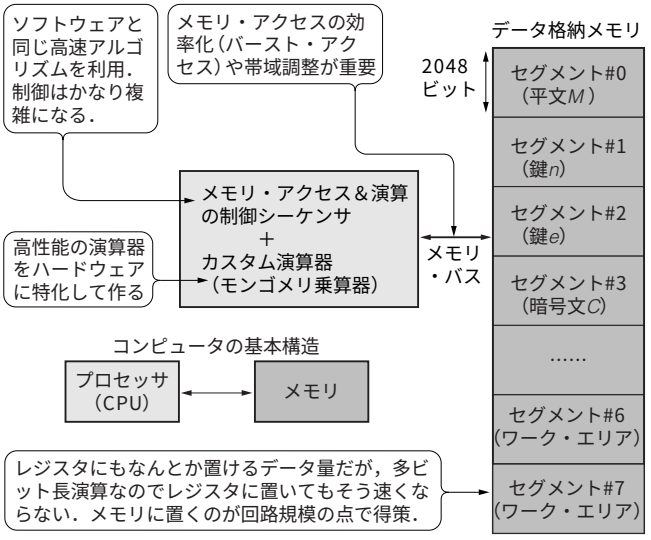


図1　RSA暗号は「コンピュータ型ハード」になる典型例

回路モジュールを処理順に並べてデータを流すのではなく，1個の演算装置がメモリ上のデータに繰り返しアクセスするタイプの回路である．これはコンピュータと同じ基本構造である．繰り返し回数は非常に多い（数十万クロック～それ以上）．同じデータを繰り返し演算をするので，演算の多重化は困難である．

1．回路ブロックを並べる手法が使えない典型例

ストリーム・データ処理であるJPEGデコーダ回路（第3章を参照）とは異なり，RSA暗号は，数千ビットの入力データに対して，繰り返し多量の演算を施す回路です．後述しますが演算手順も複雑です．このため，「モジュールを処理順につなぎ，データをためずに流すことで高性能化する」という回路の組み方はできません．どうしても，メモリ上にデータをまとめて置いておき，そこにアクセスしながら処理を進める回路にせざるを得ません（**図1**）．これはコンピュータと同じ基本構造を持つ回路です．

● ストリーム処理ができないアプリケーションは多い

RSA暗号は決して特殊ではありません．**表1**に，メモリ上のデータ操作が主体となる代表的な処理を示します．実用上重要なものがたくさんあります．

例えば画像処理アルゴリズム中に使うプリミティブ演算には，ストリーム的な処理ができないものが多々あります．単なる3×3フィルタ程度ならば，画像のごく一部（カーネルの部分）だけを見ればよいので，ストリーム処理が可能です．しかし，細線化など画像全体をスキャンするタイプのプリミティブではそうはいきません．また，知的情報処理を施して画質改善，マッチング，画像認識などを行う家電製品が増えています．そこでは，

- データ（画像）に応じて動的に画像走査方向が変わる．

Keyword　RSA暗号，べき乗剰余演算，Binary Method，余剰余算，モンゴメリ乗算，中国人剰余定理，SystemC，動作合成

表1　RSA暗号回路と似た「コンピュータ型」の構成になる回路

回　路	（補足）
画像処理におけるパターン・マッチング，直線・曲線検出，細線化など	画像認識処理では，画像を（その内容に応じて）縦横さまざまな方向に走査したり，画像全体を繰り返し加工したりする場合が数多くある．メモリ上に画像データを置いてアクセスする回路構成にせざるを得ない．
多量データのサーチ（辞書検索など）やソート	少量のデータのサーチやソートならば，並列処理により非常に高速なハードを作れる．しかし多量ならば，ソフトと同様のアルゴリズムを使った高速化をせざるを得ない．
公開鍵暗号（RSA暗号，楕円暗号など）	レジスタに置くにはやや多い量のデータに対し，多数回の繰り返し演算を施す典型的な事例．
その他，複雑なデータ構造を動的に操作するアルゴリズム	ソフトウェアでよくあるように，木構造やその他複雑なデータ構造を動的に組み替えるような処理は，回路を並べる手法では非常に扱いにくい．

●画像全体の情報（ヒストグラムなど）を調べ，それをもとに次の処理内容が決まる．

といった場合が頻繁にあるため，やはり画像全体をメモリ上に置くことになります．

このほか，データの検索やソートなども，データが一定量以上[注1]になるとメモリ上に置いて処理せざるをえなくなります（第2章も参照）．

● 圧倒的な高速化ができなくてもハード化したい

表1に示した処理はいずれも，従来では「ハードウェアではなくてソフトウェアで処理するべき」とされてきたものです．回路の設計が大変で，メモリが必要な点に加え，同じ周波数のプロセッサと比べてせいぜい数倍～数十倍くらいのスループットしか出せないので（ストリーム処理では数百倍～数万倍も可能），ハードウェア化のリスクに対してペイしないと考えられていたからです．組み込みCPUではなくパソコン用CPUと比べた場合，下手な実装をしたASICでは処理速度で負けてしまう可能性すらあります．

最近は状況が変わってきています．速度の向上以外の目的でハードウェア化する必要性が出てきているのです．

例えばセキュリティ分野の場合，システムLSI内でCPUを利用するのは，ファームウェアの解析やファームウェアの脆弱性（バッファ・オーバフロー），サイド・チャネル攻撃(2)などの可能性から敬遠される傾向にあります．また，画像処理回路中の一部だけをオンチップ・バス経由でCPUにつなぎたくない，ボトルネック処理を少しでも速くしたい，少しでも消費電力を下げたい，という多様な要求が存在するようになってきています．

注1：目安としては数K～数十Kバイト以上．そのオーダの量になると，データをレジスタで持つのが回路規模上厳しくなり，メモリに置かざるを得なくなる．データをメモリに置くことにすると，データの並列検索や並列演算が難しくなるので，ハードウェア独特のアルゴリズムを用いることが難しい．

$a = b^e \bmod n$

a,b,e,n はそれぞれ2048ビット（または1024ビットか512ビット）の数

暗号化の時　a：暗号文，b：平文，<n，e>：公開鍵
復号化の時　a：平文，b：暗号文，e（普通はdと表記）：秘密鍵，n：公開鍵

（**a**）RSA暗号の処理

```
tmp=b;
for(i=2;i<e;i++)      ← 最高で2^2048 −1 回まわる
    tmp=tmp×b;        ← 1回まわるたびにtmpのサイズは2048ビット増える
a=tmp mod n;
```

（**b**）数式通りに演算するアルゴリズム

図2　RSA暗復号の入出力データと演算処理の内容
数式が簡単な事は，回路作成が簡単という事を意味しない．何も考えずにこの数式どおりに計算するのは不可能である．

2．RSA暗号処理の内容と典型的ソフト実装

それでは，RSA処理について，回路設計で必要なアウトラインに絞って説明します．数学的な背景やアルゴリズム正当性の説明は省略します（詳しい説明は参考文献(1)にある）．

● 多ビット長のべき乗剰余演算を行う

RSA暗号の演算は，**図2**の通り，

$$a = b^e \bmod n$$

ここで入力はb，e，n，出力はa

と1行の数式で表せます．具体的な数を代入してみると，

$$3 = 2^3 \bmod 5$$

などと一目で分かる内容です．

しかし，数式が簡単であることは，回路の作成が簡単であることを意味しません．どちらかといえば，実用回路の設計では，仕様がこのように簡潔に書いてあった場合ほど，タチが悪いことが多いのです．

実用のRSA暗号では，ソフトウェアでもハードウェアでも作成はなかなか大変です．その理由は，実際に用いられる暗号では上記のa，b，e，nが512～2048ビットと多ビットであるからです．このため，式通りの単純な実装をする