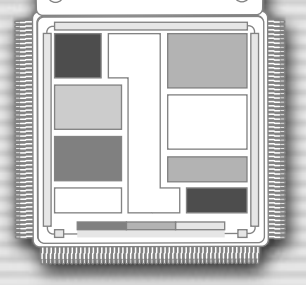


第3回

C++を用いた公開かぎ暗号回路の細分化・階層化設計事例

高野光司，大庭信之，森岡澄夫

時とともに，論理回路設計では大規模かつ複雑なものを扱うようになりました．そして，C/C++やSystemCなどを利用して，高い抽象度でシステムやビヘイビアのモデリングを行うことが多くなっています．しかし，現時点におけるIPコアなどの論理回路の設計では，人手でHDLによるRTL記述を作成しなければならないため，その過程でミスを少なくし，モデルとRTLの等価性の検証を容易にするための手段が重要となります．C++のクラスとカプセル化の概念を用いることで，複雑なモデルであっても論理回路化が可能な単位に細分化でき，設計が楽になります．今回は，筆者らが設計したモンゴメリ型RSA/だ円暗号回路を題材にして，C++を用いた細分化・階層化設計の例を紹介します．（筆者）

公開かぎ暗号であるRSAとだ円暗号[注]は，電子署名や認証，共通かぎ暗号のかぎ交換など，安全性が求められるさまざまなアプリケーションで使われています．RSAとだ円暗号は，暗号化・復号化の計算量が非常に多いという特徴があります．ソフトウェアによる暗号実装も数多く行われていますが，サーバ用途のように高い暗号処理性能が要求される場合や，セキュリティ・レベルを向上させるために専用ハードウェアが必要な場合などでは，論理回路によるハードウェア実装が必要になります．

RSA暗号で用いられる演算量の大きな演算

$y = x^e \bmod n$ （x，y，e，nは512ビット～2,048ビット，もしくはそれ以上）

だ円暗号で用いられる演算量の大きな演算

$y^2 = x^3 + ax + b \bmod p$ 上のk倍算（x，y）＝k（x_0，y_0） （x，y，a，b，p，kは160ビット～256ビット，もしくはそれ以上）

〔図1〕RSAとだ円暗号の代表的な演算

RSAとだ円暗号の処理に必要な演算の中で演算量が大きいものは剰余演算である．

C++を用いた細分化・階層化設計

RSAおよびだ円暗号で演算量の多い代表的な演算を**図1**に示します．RSA暗号の場合，べき乗剰余演算（べき乗演算＋剰余演算）を用います．べき乗剰余演算は内部的には乗剰余演算（乗算＋剰余演算）の繰り返しとなり，暗号演算に必要な演算のほとんどは剰余演算となります．だ円暗号の場合，ガロア体での演算が必要となりますが，基本的には剰余演算の繰り返しとなります．

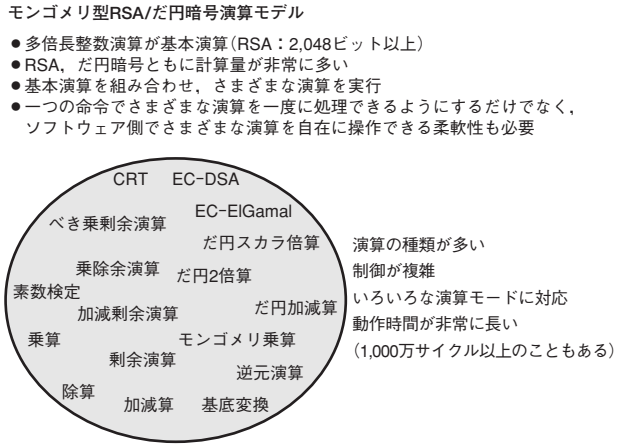
RSAとだ円暗号で必要な基本演算を**図2**にまとめます．整数加減算に始まり，剰余演算，だ円演算，そのほかいろいろな演算が必要になります．筆者らは整数剰余算をより高速に実行するためにモンゴメリ法という演算アルゴリズムを用いているため，そのための演算処理も必要になります．

図3はべき乗剰余演算のフローチャートを表したものです．**図2**で示した多くの演算を組み合わせて処理を行っています．また，このフローチャートからさらに別の演算を呼び出してさまざまな処理を行います．

●複雑なモデルを論理回路化する難しさ

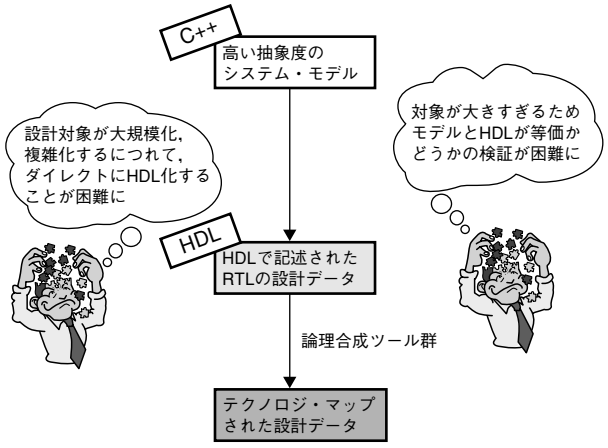
このように，公開かぎ暗号回路は複雑な制御構造を持つため，C/C++やSystemCなどを用いて，高い設計抽象度でモデル化し，演算アルゴリズムや制御に関する検証を行います．このモデルをもとに論理回路を設計するのですが，

注：RSAは現在もっとも広く使用されている公開かぎ暗号方式である．この方式を発明した3人の数学者の頭文字をとって「RSA」と名づけられた．だ円暗号は，だ円曲線が持つ特性を利用した暗号方式であり，一般に処理が高速である．



〔図2〕モンゴメリ型RSA/だ円暗号演算モデル

公開かぎ暗号であるRSAやだ円暗号を行う論理回路は，多くの演算が行えなくてはならない．演算を高速にするためにさまざまな演算アルゴリズムを組み入れるので，制御も複雑になる．

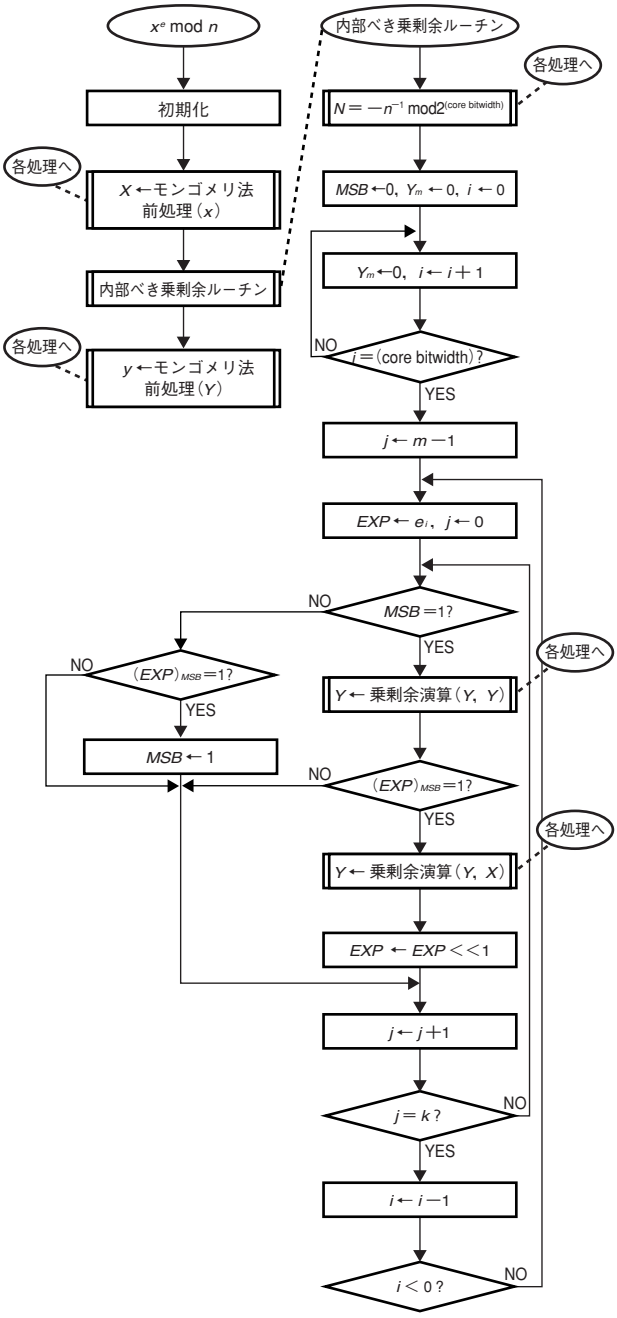


〔図4〕高い抽象度のシステム・モデルから論理回路を設計

モデルが複雑になればなるほど，論理回路設計は難しくなる．同時に論理回路の検証も難しくなる．

これらのモデルからHDL記述を作成する作業は，現時点では人手による設計がほとんどです．そして設計者は，最終的にモデルと同じように動作するHDL記述を作成します．しかし，HDLで書かれた回路が，もととなるモデルと「絶対に同じ」であることを証明することは，非常に難しいというのが現実です（図4）．

ちょっとした演算回路や制御回路を設計するだけであれば，その仕様を決めて一気に論理回路を作ることも可能です．しかし，設計する論理回路の規模が大きくなれば，それだけ設計が複雑になり，一気に論理回路にすることが難しくなります．たとえ論理回路になったとしても，そのような複雑な論理回路がほんとうにモデルと同じように動作するかどうかをどうやって検証すればよいのでしょう．回路のすべての入力とすべての状態に対してシミュレーションして，すべての出力が合っているかどうかを検証できればよいのですが，このような総当たりの検証を行うと，たいていの場合は計算量が爆発します．



〔図3〕べき乗剰余演算のフローチャート

べき乗剰余演算は，乗剰余演算やそのほかの演算アルゴリズムなど，さまざまな処理で構成されている．