改良モンゴメリ乗算アルゴリズムを用いたビットシリアル RSA 暗号化器の設計

要約 – **Design Wave Magazine 設計コンテスト 2008 の RSA 暗号回路設計結果を報告する．本稿はモンゴメリ乗算アルゴリズムを用いる累乗計算で，RSA 暗号処理の中間演算の法を暗号処理で与えられた法 M の 3 倍にとることでより簡易な回路構成となることを示す．これを回路化するときに，メッセージのビット長 N に回路規模と遅延時間が依存しない回路構成として，冗長表現を用いたビットシリアル演算回路を用いた．Xilinx 社の Sparntan3 をターゲットに合成をおこない，50 入力 XOR に対して回路規模 47 倍，遅延時間 1.8 の結果を得た．さらにスループットを 2 倍にする方法として，本稿の基数 2 の設計に対して基数 4 で設計した場合を取り上げ，そのメリットを回路規模と遅延時間から考察をした．**

Keywords **– RSA 暗号処理，モンゴメリ乗算**

I. RSA 暗号のなかみ

RSA 暗号処理をまとめます．平文 Cp，暗号文 Ce, 暗号化鍵 E，復号鍵 D,および法 M がそれぞれ正整数で与えられているときに，以下の式を用いて暗号化をおこないます．

$$C_e = C_p^{\,E} \bmod M \qquad \text{--- (1)}$$

ここで mod は剰余を求める演算子で，A mod B は A を B で割った余りをあらわします．RSA 暗号は以下の式を用いて復号を行います．

$$C_p = C_e^{\,D} \bmod M \qquad \text{--- (2)}$$

RSA 暗号の暗号化と復号処理はいずれも，累乗の剰余で与えられます．

暗号化鍵 E，復号鍵 D および法 M それぞれは以下の条件を満たすように選びます． まず法 M は十分大きな 2 つの素数 p および q の積で与えます．このとき(p-1)および(q-1)の最小公倍数を L とすれば，

$$C^L = 1 \qquad \text{--- (3)}$$

が成り立ちます．したがって暗号化鍵 E および復号鍵 D を正の整数を H として，

$$E \cdot D = H \cdot L + 1 \qquad \text{--- (4)}$$

を満たすように決めます．つまり，RSA 暗号は剰余演算の累乗の周期を利用した暗号アルゴリズムというわけです．

RSA 暗号では，送信者は受信者から渡された法 M および暗号化鍵 E から式(1)で平文から暗号文を求めます．受信者は法 M および秘密にしている復号鍵 D から式(2)で暗号文から平文を復号します．

II. 回路化に適したアルゴリズムを考える

これで RSA 暗号化器の設計は，式(1)の法 M の剰余で累乗を計算する回路の設計だと分かります．この剰余の四則演算を剰余演算と呼ぶことにします．回路化するために本稿で取った余剰演算のテクニックを述べます．

説明のために暗号化鍵 E，復号鍵 D を正整数 N として 2 進数で表される N ビットの整数(つまり，$(2^N - 1) >= E, D >= 0$)とします．法 M は最上位ビット(Most Significant Bit, 以下 MSb)が'1'の N ビットの奇数(十分大きな 2 つの素数の積は因数 2 を含まないから奇数になる，また $(2^N - 1) >= M > (2^{N-1} - 1)$)，平文 Cp および暗号文 Ce は法 M より小さい正の整数とします．

もしも平文 Cp の剰余計算を，まず通常の四則演算をしてから法 M の剰余を求めたとすればどうでしょうか．通常，ビット長 N には 1024 ビットという大きな値が取られますから，その乗算結果は 2048 ビットになります．この数を除算して剰余を求めるとすれば，乗算 1 回あたり 1024 ビットの乗算と 2048 ビットの除算をすることになります．これを素直に回路化すれば大変な回路面積になってしまいます．

(テクニック 1，改良モンゴメリ・アルゴリズム)

そこで除算をしなくても剰余演算の乗算ができるモンゴメリ・アルゴリズムによるモンゴメリ乗算回路を採用しました．章 III に，以下に述べる冗長 2 進表現に適するようにこのアルゴリズムを改良した点を述べます．

(テクニック 2，冗長 2 進表現をとるビット･シリアル演算回路)

乗算回路には必ず加算回路が必要です．1024-bit の加算回路を何も考えずに設計すると桁上げ信号の伝播遅延時間が大きくなり実用的ではありません．そこで 1 桁の数値を 1 ビットの 0/1 ではなく 2 ビットで表す冗長 2 進表現を採用しました．本稿の冗長 2 進表現は値{0, 1, 2, 3}それぞれに{2'b00, 2'b01, 2'b10, 2'b11}を割りあげます．この冗長表現により桁上げがなくなり高速な加算回路が得られます．加算回路構成は 1 桁づつ演算をするビット･シリアル演算回路を採用しました．この回路はビット長 N に動作速度が依存しないので，ビット長 N が大きい暗号にも動作速度を落とさずに対応 できるという拡張性が得られます． 章 VI に，冗長 2 進表現のビット･シリアル加算，乗算および自乗回路の構成を述べます．

(テクニック 3，入出力はすべて LSb-first)

ビット･シリアル演算回路を縦接続するときに，例えば，前段および次段がそれぞれ LSb-first 出力，MSb-first 入力だと，接続部分に MSb と LSb の順序を入れ替えるためにビット長 N のバッファが余分に必要になってしまいます．このビット長 N の順序入れかえは N クロックかかるので処理時間の無駄にもなります．

本稿は LSB-first 入力を必要とする改良モンゴメリ・アルゴリズムに合わせて，全回路を LSb-first 入力にそろえています．章 IV に全体回路構成を示します．

III. 改良モンゴメリ乗算法

改良モンゴメリ乗算アルゴリズムを C ライクの表記で図 1 に示します．

(入力)：
　法：M,
　乗数：A=(a_{n-1}, a_{n-2}, …, a_1, a_0) (2 進数表記)
　被乗数：B (ビット長 N)
(出力)
　$R \equiv A * B * 2^{-N} \pmod M$),
　　$0 \le R \le 2^N + M < 2^{N+1}$,
(アルゴリズム)：
MM(A, B, M)
{
　P1: $A * B = C = C_1 * 2^N + C_0$;
　　($0 \le C_0, C_1 < 2^N$, $C_0=(c_{n-1}, c_{n-2}, \ldots, c_1, c_0)$)
　　P[0] = 0;
　P2:for(i = 0; i < n; i++) {
　　// P[I]は偶数か奇数か?
　　$q_i = (P[i] + c_i) \bmod 2$;
　　//右に 1-bit シフト
　　$P[i+1] = (P[i] + q_i * M + c_i) \text{ div } 2$;
　　}
　　R = P[n] + C_1;
　　return R;
}

図 1. 改良モンゴメリ乗算アルゴリズム

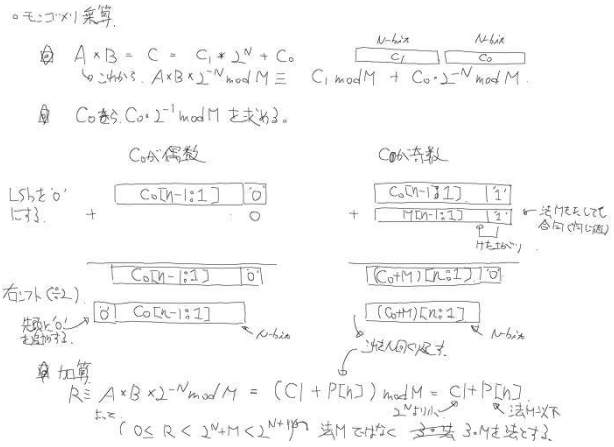

図 2. モンゴメリ乗算の計算例

図 1 のモンゴメリ･アルゴリズムのポイントは，積に定数 2^{-N} をかけることで，除算を使わずに，加算とビットシフトだけで法 M の乗算ができることです．図 1 の計算例を図 2 に示します．図 1 の P1 で A と B はビット長 N なので，その積の C はビット長 2N になります．これに 2^{-N} をかけるので，R は C の上位 N ビット C_1 と，下位 N ビット C_0 と 2^{-N} の積の合計値になります．

この C_0 と 2^{-N} の積を求めるのが図 1 の P2 になります．まずは $C_0 * 2^{-1}$ を求めます(図 2)．C_0 が偶数であれば LSb は'0'なので 1-bit 右シフトするだけです．C_0 が奇数のときは法 M(法 M は 2 つの素数の積なので必ず奇数)を足して LSb を'0'にしてから 1-bit 右シフトします．こうして得られた $C_0 * 2^{-1}$ に同じ処理をあと(N-1)回繰り返せば求める $C_0 * 2^{-N}$ が得られます．最後に C1 と $C_0 * 2^{-N}$ を足せば R が求まります．

こうして得られた R はビット長(N+1)で，法 M より大きいかもしれません．オリジナルのモンゴメリ･アルゴリズムは R が法 M より小さくなるように R から法 M を減算します．これには R と法 M の値比較と減算という計算ステップが必要になります．この改良アルゴリズムはその計算ステップを省略しています．これは法 M ではなくて法 3*M で計算していることになります．この改良アルゴリズムによる累乗アルゴリズムを図 3 に示します．

(入力)：
　法：M,
　指数：E =($1, e_{k-2}, e_{k-3}$ …, e_1, e_0),
　メッセージ：C_p,
　定数：$C=2^{2(N+2)} \bmod M$,
(関数)
　$MM(A, B, M) = A * B * 2^{-(N+2)} \pmod M$),
(出力)
　$R[k - 1] \equiv C_p^E \pmod M$), $0 \le R[k-1] <$ M ,
(アルゴリズム)：
MM_E(C_p, E, M, C)
{
　//C_pの前処理．定数 C をかける．
　C_p' = MM(C_p, C, M);
　R[0] = C_p';
　for(i = 0; i < k-1; i++) {
　　R[i] = MM(R[i], R[i], M); //自乗
　　if(e_{k-i-2} == 1)
　　　R[i+1] = MM(R[i], C_p', M);
　　else
　　　R[i+1] = R[i];
　}
　//定数 2^{N+2} を消す
　R[k-1] = MM(R[k-1], 1, N);
　return R[k-1];

図 3.累乗アルゴリズム

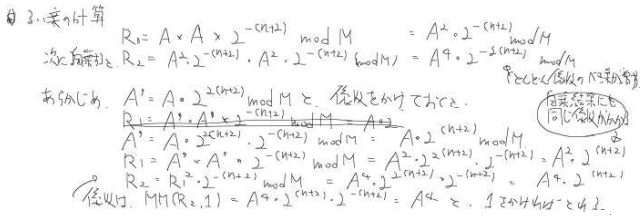

図 4.乗算により発生する係数の扱い

図 3 の累乗アルゴリズムそれ自体はよくある累乗アルゴリズムです．2 進表記した指数の'0'でない最上位ビットを見つけて，そこから自乗と次のビットが'1'であれば元の値をかけることを繰り返して，求める累乗を得ます．

このアルゴリズムのポイントは，メッセージに定数をかける前処理をしてモンゴメリ乗算で生じる定数をうまく処理すること，および図 1 の P2 のループ

回数を(N+2)回にしてモンゴメリ乗算の結果をビット長(N+1)に収めていること，の２つです．

図 3 の乗算例を図 4 に示します．

モンゴメリ乗算で単に値 A を自乗すると $A^2 * 2^{-(N+2)}$，これをさらに自乗すると $A^4 * 2^{-2(N+2)}$ と自乗するたびに異なる係数がかかります．こうなると自乗するたびに係数 $2^{-(N+2)}$を取り除かないと正しい結果が得られなくなります．ここで値 A にあらかじめ係数 $2^{2(N+2)}$ をかけておけば，モンゴメリ乗算で自乗を繰り返しても同じ係数 $2^{(N+2)}$ がかかります．この係数 $2^{(N+2)}$ は最後にモンゴメリ乗算で 1 をかければ取り除けます．

メッセージ C_p を前処理した結果得られる R[0]はビット長(N+1)です．図 1 の P2 のループ回数を(N+2)回にすれば自乗した結果はビット長(N+1)に収まります(図 5).

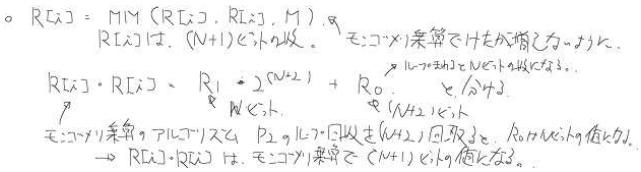

図 5.モンゴメリ乗算による累乗計算の桁丸め込み

IV. 回路構成

図 6 に全体構成を示します． 入出力信号名はコンテスト課題に従います．バス幅の明記がない信号線は全てビット幅 1 です．インスタンス I0，インスタンス I1 は，それぞれ $2^{(N+2)}$ mod CM を求める回路ブロック，モンゴメリ乗算回路を示します．RSA 暗号化処理の手順は以下になります：

1. 入力信号をバッファに取り込む.
   A) B0,B1 および B2 それぞれにブロックサイズ分信号を取り込みます.
2. インスタンス I0 で $2^{(N+2)}$ mod CM を求めます.
3. 図３のメッセージ前処理をおこないます.
   A) モンゴメリ乗算回路 I1 のシリアル入力選択 MUX(I2)で I0 の出力を選択します.
   B) Cp’にはバッファされた平文が取り込まれています.
   C) (N+2)サイクル後に出力される乗算結果をバッファ B1 にバッファします.
4. 累乗計算をおこないます.
   A) 図３の R[i]の自乗計算をおこないます．モンゴメリ乗算回路 I1 を自乗計算に設定してビットシリアル出力を MUX I2 でモンゴメリ乗算回路の入力に戻します.
   B) 指数 CE の MSb の次ビットが’1’ならば，先の乗算結果に前処理した平文 Cp’を乗算します．モンゴメリ乗算回路の乗数には手順 3-C で求めた平文 Cp’が入力されているので，モンゴメリ乗算回路 I1 を乗算計算に設定して，計算結果を MUX I1 で入力に戻せば，これを求められます.
5. 処理結果を B1 に取り込む.
   A) 手順 3 と同じ手順でモンゴメリ乗算回路の出力をバッファ B1 に取り込む.
6. 処理結果の出力.
   A) バッファ B1 から 8 ビットづつ選択出力する.

法 CM が同じならば手順 2 の $2^{(N+2)}$ mod CM の計算を繰り返す必要はありません．ですが設計を簡単にするために毎回これを計算しています.

課題で入出力順序は任意とある入出力信号は，LSB-first, 出力も LSB-first にしました．またブロックサイズ B の値とブロック長の対応は，B が 0, 1, 2, 3 のときブロック長はそれぞれ 1, 2, 3, 4 バイトとしました.

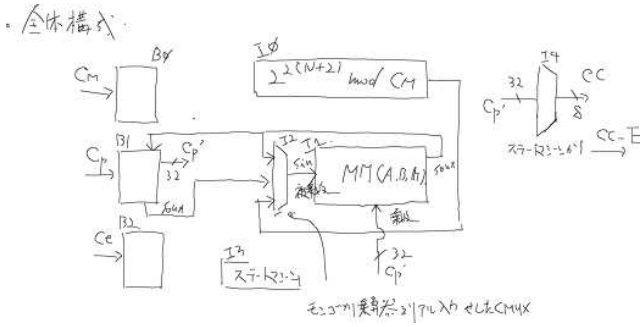

図 6. RSA 暗号化器 全体回路構成

V. $2^{2(N+2)}$計算回路

$2^{(N+2)}$ mod M の計算アルゴリズムを図 7 に示します．簡単のためにビット長 N を 8 とします．求める値は 9’h100 をビットシフトしていくことです．計算結果が法 M 以下にするため，ビットシフトした結果が法 M より大きければ結果から法を引きます.

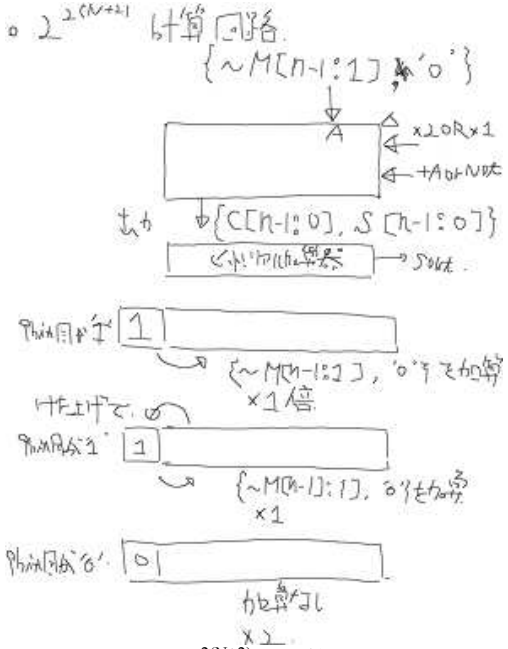

図 7. $2^{2(N+2)}$の計算アルゴリズム

法 M を引く/引かないの判断は 9 ビットの値のみで決めます．法 M は 2^8 未満の値ですから，9 ビット目が’1’であれば必ず法 M が引けます．ただし，計算結果が法 2M より大きい場合があります．このときに

は法 M を引いても桁上げで 9 ビット目が再び'1'になりますから，次のサイクルで再度法 M を引きます．9 ビット目が'0'ならば右に 1 ビットシフトして 2 倍します．これを(N+3)回繰り返せば結果が得られます．

法 M の減算は法 M の 2 の補数を加算しています．法 M は奇数ですからその補数はビット反転したものの LSb を'0'にするだけで求まります．

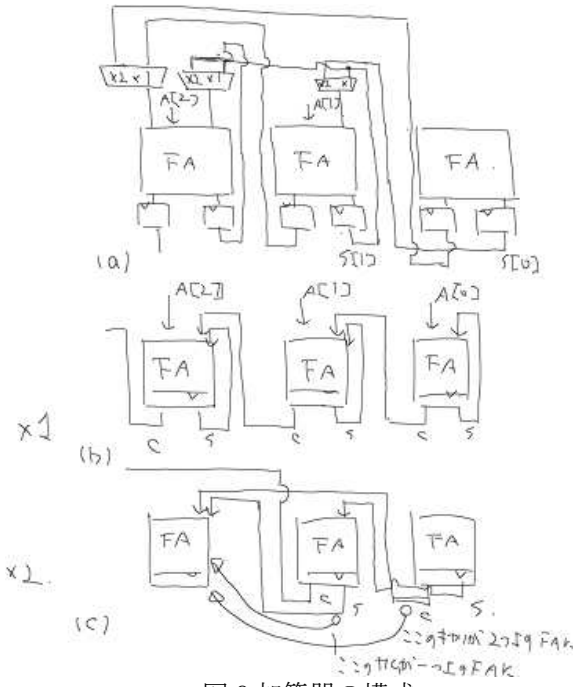

図 8.加算器の構成

図 7 の加算の回路構成を図 8 に示します．FA は 1 ビットの全加算器を，下に線が入った FA は出力がフリップフロップ(以下，F.F.)の FA を示します．桁上げ信号伝播が生じないように，1 桁の値をキャリアとサムを組み合わせた 2 ビットの値(以下，{C, S}と表記します)で表現します．拡張表現{C, S}の (2 * C + S) を求めれば 2 進数に戻せます．図 8(a)の回路は加算と乗算を一度に行います．MUX の働きが分かりにくいので，A を加算するだけのときの信号配線を図 8(b)に示します．キャリアを一段上の FA にサムを同じ FA に戻して A を加算します．加算器出力の 2 倍と A を加算するときの信号配線を図 8(c)に示します．キャリアを 2 段上の FA にサムを 1 段上の FA に入力します．

```
if( C[N-1] == 1) {
    x1 倍，法 M の 2 の補数を 足す
} else if( C[N-2] == 1) {
    x1 倍，0 を足す
} else if( S[N-1] == 1) {
    x2 倍，法 M の 2 の補数を 足す
} else {
    x2 倍，0 を足す
}
```

図 9. 冗長表現の $2^{2(N+2)}$計算アルゴリズム

前述のアルゴリズムを拡張表現{C, S}が使えるようにすると図 9 になります．(N –1)ビット目のキャリア C[N-1]が'1'ならば 9 ビット目が'1'なので法 M を引きます．(N –1)ビット目のサム S[N-1]が'1'ならば，2 倍すると 9 ビット目が'1'になるので法 M を引きます．ここで，図 8 から，2 倍すると C[N-2]を失うので，C[N-2]が'1'のときには単に 1 倍してこれを防止しているのがポイントです．拡張表現{C,S}はキャリアとサムを別々に扱うために，それぞれを個別に丁寧に見てやらねば思わぬ計算違いが生じてしまいます．

VI. モンゴメリ乗算回路

モンゴメリ乗算回路の構成を図 10 に示します．図 1 のアルゴリズムどおりに，求めた 2 数の積の下位ビットに $2^{(N+2)}$を乗算します．乗算回路を図 12 および図 13 に示します．ビットシリアル入力 A と前処理したバッファにあるメッセージ B の積は，A のビットと B の部分積を加算していけば求まります．モンゴメリ乗算のビットシリアル出力を入力に戻して直ちに自乗計算するために，自乗回路の部分積は図 12 に示す順序で入力します．

自乗と乗算を 1 つの回路で行うように図 13 の部分積を求める回路を組みました．またブロックサイズに対応するために，ブロックサイズが 1, 2, 3, 4 それぞれのときに 10, 18, 26, 34 桁目それぞれのキャリとサムが後段に伝わらないように MUX を入れています．

図 1 の C0* $2^{-(N+2)}$を求める回路を図 11 に示します．2 の割り算はビットシフトで実現できます．図 1 の P2 の式 $P[i+1] = (P[i] + q_i * M + c_i)$ div 2 の LSb の加算をみると入力数が 4 になります．計算結果が 3'b100 になるために，結果が冗長表現の 1 桁{C,S}に収まりません．そこで加算を $q_i * M + c_i = q_i * (M - 1) + q_i + c_i$ と分けて別々に加算することで，冗長表現の範囲内に収めます．これを回路にしたのが図 11 です．法 M は奇数ですから法 M の LSb を'0'にすれば(M –1)になります．

ブロックサイズ 0~3 に対応するために，ブロック数が 1, 2, 3, 4 のときにモンゴメリ乗数回路のサイクル数を 10, 18, 26, 34 にそれぞれ設定しています．

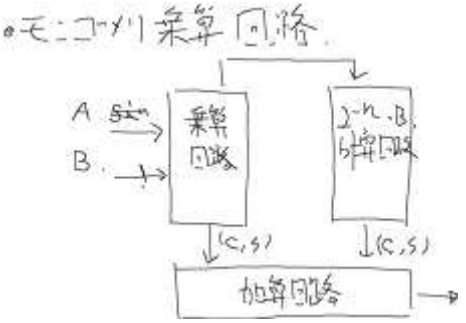

図 10.モンゴメリ乗数回路 構成

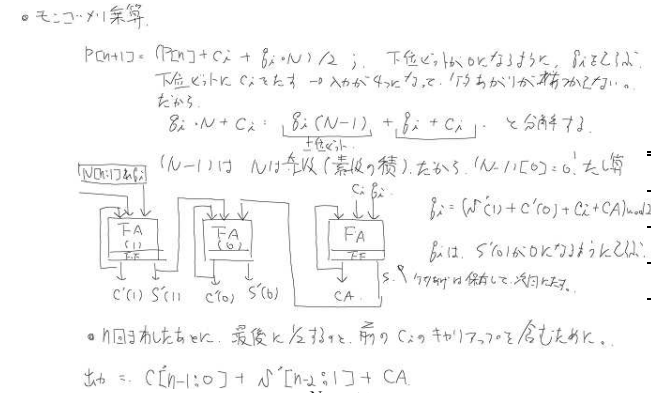

図 11. 2^{-N}乗算回路

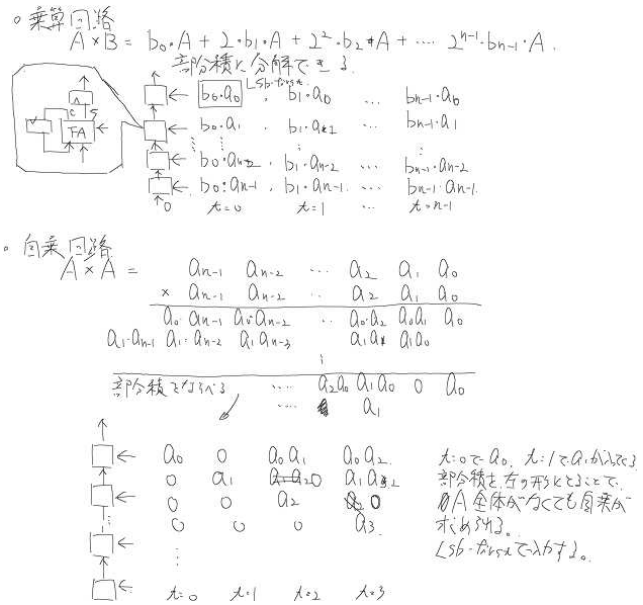

図 12. ビットシリアル乗算/自乗回路

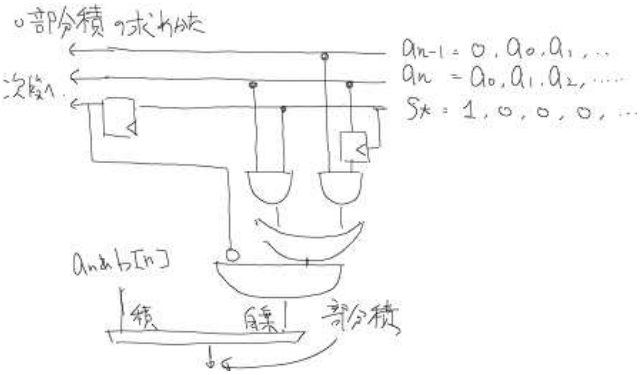

図 13. 部分積回路

VII. 合成結果と考察

Xilinx 社の ISE version 9.2.03i でターゲットを XC3S50-4(Spartan3 スピードグレード 4)として合成した結果を表 1 に示します．50 入力 XOR の使用スライスおよび遅延時間はそれぞれ 10 スライスおよび 12 ns でした．それらを 1 単位とした値を括弧内に示しています．

期待値を別プログラムで作成して検証をかけました．全ての入力値に対する検証までは終了しておらず詰めができていません．

表 1 合成結果

項目	値 (50 入力 XOR 対比)
使用スライス	465 (47)
F.F.の使用数	478
4 入力 LUT の使用数	849
遅延時間	22.1 ns (1.8)
最大動作クロック	45 MHz

入力信号 START の立ち上がりから出力信号 CC_EN 立ち上がりまでのレイテンシは鍵の値によりますが，コンテスト課題の法と復号鍵でのレイテンシを表 2 にまとめます．

表 2 合成結果

法	復号鍵	レイテンシ
253	61	114 クロック
54991	1657	286 クロック
4165512167	612556253	1578 クロック

ターゲットにした Xilinx 社の Spartan3 は 1 スライスあたり F.F.と LUT を 2 つづつ持ちます．合成結果から LUT の数で回路規模(スライス数)が決まったとわかります．今回の設計はビットシリアル演算 で構成しています． したがって LUT の使用数と F.F.の使用数が同数になるように 1 ビット分演算回路を作りこめば，スライス数は 250 程度まで絞り込める可能性があります．

動作速度は，モンゴメリ乗算回路の入出力信号の引き回し方法で決まってしまいました．モンゴメリ乗算回路は(N+2)サイクルで計算結果が得られます．その計算結果を次サイクルで直ちにモンゴメリ乗算回路に戻しています．この信号経路は，図 14 に示すようにモンゴメリ乗算回路の演算回路，LSb 加算回路および MUX を経由するために，遅延時間がかさみました．

これを高速にするには(N+2)サイクルでは加算のみをして，(N+3)サイクル目でモンゴメリ乗算回路に出力結果を戻すようにすればよいです．(N+2)サイクルの最小サイクルで演算することにこだわり設計しましたが，これが(N+3)サイクルになっても，ビット長 N が 1024 ビット程度であれば全体として 0.1%程度処理時間が長くなるに過ぎません．モンゴメリ乗算回路単体は 100MHz で動作しますので，処理クロック数が 0.1%増えても，100MHz の最大動作速度を狙うべきでした．設計の退くべきところと攻めるところを間違えた，良い教訓を得ました．

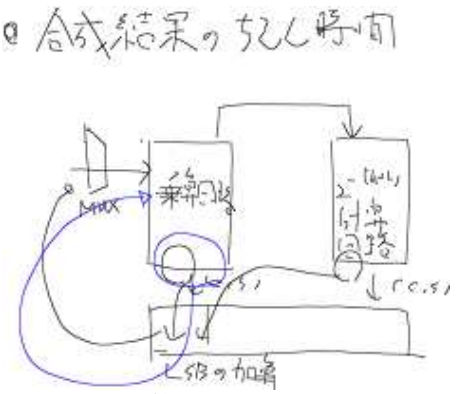

図 14.モンゴメリ乗算回路まわりのクリティカルパス

VIII. ビット長 1024 の回路規模と最大動作クロック

本設計はビットシリアル回路構成をとったのでビット長 N を容易に変更できます．大雑把な見積もりかもしれませんが，ビット長 32 で 465 スライス使用したので，ビット長 1 あたりのスライス数は 15 です．ビット長 1024 の場合回路規模は 15400 スライス程度になると予測されます．ビットシリアル回路は動作速度がビット長 N によりませんから，45MHz の動作速度がえられるはずです．

IX. スループットを 2 倍にする方法，基数を 4 にする

FPGA でも Gbps レベルの高速シリアル通信が一般になってきています．このような高速シリアル通信に対応するには高いスループットが求められるはずです．スループットを高める今回の設計の改良点を述べます．

スループットを 2 倍にする手段は基数を 2 から 4 にすることです．つまり 1 クロックあたり 1 ビット処理していたのを 2 ビット処理することになります．

回路規模はどの程度になるでしょうか．本設計のほとんどはビットシリアル加算回路が占めています．図 15 に示すように基数 2 の加算回路は LUT2 つですが，基数 4 では LUT7 つになります．したがって回路規模は 1.8 倍程度になると予測されます．また図 1 のモンゴメリ乗算アルゴリズムに変更が必要です．基数 2 では最下位桁が 0 にするためには，法 M を足すか足さないかだけでよかったのですが，基数 4 では法 M の 0, 1, 2, 3 倍のいずれかを足すようにしなければなりません．このために法 M の 3 倍をあらかじめ求めておくステップと，計算結果を保持するバッファ，それに法 M の倍数を選択する MUX が新たに必要になります．

動作速度は，本設計がそうであったように，1 桁の回路動作速度で与えられるとしてよいでしょう．ビット幅が 1 ビットから 2 ビットになってもスライス自体が加算回路に最適化されている Spartan3 では，加算回路の動作速度は変わらないと予測されます．基数 2 の最大クロックは同程度でしょう．

まとめると基数を 4 にすると回路規模およびスループットは，それぞれ 1.8 倍，2 倍になり，最大動作クロックは基数 2 とほぼ同じと予測されます．

この結果からレイテンシの短縮が問題になるならば基数を 4 にする，スループットのみが問題であれば基数を 4 にするか，もしくは基数 2 の回路を 2 つ並列に動作させることが有効だと考えられます．

図 15. 基数による加算器の回路規模比較

X. まとめ

改良したモンゴメリ乗算アルゴリズムを用いた RSA 暗号化器の設計をおこない回路規模と動作速度の見積もりを得ました．また本設計をビット長 1024 に適用した場合，および基数を 4 にした場合の，回路規模と速度を予測しました．

モンゴメリ・アルゴリズムを用いた累乗アルゴリズムで，計算途中結果の法を 3M にとることで，MSb1 ビットのみで法 M の減算判定ができて，オリジナルのモンゴメリ乗算アルゴリズムに必要だった法 M の減算判定と減算処理を省略することができました．そのかわりに，オリジナルが乗算 1 回を N サイクルでできたのに対して，改良アルゴリズムは (N+2) サイクルが必要になりました．しかし，ビット長 N が 1024 程度の場合，これは 0.2%程度のサイクル数の増加にすぎません．

ビット長 N に依存しない前述アルゴリズムの回路化を考えて，ビットシリアル演算回路で構成しました．計算結果のバッファリングおよび前段の計算結果完了待ちサイクルなどの無駄がでないように，全ての演算回路を LSb-first 入力，LSb-first 出力にして，演算回路間を直接ワイヤ接続できるようにしました．

合成結果は 45 MHz とモンゴメリ乗算回路単体の動作速度約 100 MHz に及びませんでした．これは(N+2) サイクルで乗算するために演算結果が直ちに演算入力に流れるように設計したパスがクリティカルになったためでした．1 サイクル入れて(N+3)サイクルで動かすように設計すれば，モンゴメリ乗算回路単体と同程度の動作クロックになったと思います．サイクル数を(N+2)にする設計にこだわったばかりに最大動作クロックを半分に落としてしまいました．攻めるところと譲るところの見通しが甘かった，いい教訓になりました．

XI. 引用文献

[1] Ching-Chao Yang et. al., “A New RSA Cryptosystem Hardware Design Based on Montgomery’s Algorithm,” *IEEE Trans. Circuts and Systems II*, vol.45, pp.908-913, July, 1998.

[2] Stephen E. Eldridge and Colin D. Walter, “Hardware Implementation of Montgomery’s Modular Multiplication Algorithm,” *IEEE Trans. Comput.*, vol.42, pp.693-699, June, 1993.