卒業論文

ＲＳＡ暗号における冪剰余計算の
ＦＰＧＡボードへの実装

氏名	:	狭間　洋平
学籍番号	:	2260060079-0
指導教員	:	山崎 勝弘　教授
提出日	:	2010 年 2 月 19 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本研究では、RSA 暗号における冪剰余計算アルゴリズムを Verilog-HDL で記述し、FPGA ボードへの実装・検証を行っている。

本論文では、まず冪乗計算アルゴリズムの右向きバイナリ法、剰余計算アルゴリズムの回復法・インターリーブ法・モンゴメリ法のアルゴリズムを説明する。次に、計算に必要な拡張ユークリッドの互除法を説明し、右向きバイナリ法と剰余計算アルゴリズムをそれぞれ組み合わせて、回路規模と処理速度の比較を行う。また、モンゴメリ法に関しては、二通りの基数で処理速度の比較を行う。最後に、実験で得たデータを元に考察を行う。

目次

図目次

表目次

１．はじめに

現在、インターネットを通じて様々な情報がやり取りされている。クレジットカードのデータやパスワード、銀行口座の番号、取引内容など重要なデータが通信されている。書面では改ざんをすれば痕跡が残るため、改ざんを行うことが困難で、本人が押印・サインを行うため、他者がなりすますことも、書面に記載されていることを否認することも困難であった。しかし、デジタルデータは簡単に改ざんができ、顔が見えないので他者になり済ますことも可能である。セキュリティが甘いと、悪意のあるユーザーによって、銀行やクレジット会社になりすまして、パスワードなどを狙うフィッシング詐欺や、取引の信頼性を脅かすデータの改ざんや否認行為が行われてしまう。このようなことを防ぎ、ネットワークを通じた取引の信頼性を守るために電子署名というものがある。電子署名は、公開鍵暗号方式を使用し、秘密鍵をそれぞれのユーザーが管理することによって、個人の特定を行うことができる。または、共通鍵暗号方式の鍵を公開鍵暗号方式で暗号化することで安全性を保つ。共通鍵暗号方式の鍵を公開鍵暗号方式で暗号化するのは、共通鍵暗号の、暗号化と復号化に同じかぎを使うため、鍵を送る際に傍受されてしまうと、データを盗まれてしまうという欠点と､公開鍵暗号の処理の負担が大きいという欠点を補うためである。

また、公開鍵暗号方式が用いられているものに、近年急速に普及してきているICカードがある。銀行が発行するキャッシュカード、クレジット会社が発行するクレジットカード、スーパー・コンビニエンスストアの買い物で使用するEdyや公共交通機関で使用するSuicaやICOCAなどの電子マネー、携帯電話で使用されるSIMカード、地上デジタルテレビを視聴するために必要なB-CASカード、運転免許証や住民基本台帳カードやTASPOなど、行政や民間の多くの分野でICカードが使用されている。今後、ますます多くのICチップを搭載したカードが普及することが考えられる。

ICカードには接触型と非接触型があり、接触型ではリーダー／ライターに接触させることで電力を供給しデータのやり取りを行うが、非接触型ではカードに組み込まれたコイルにより、リーダー／ライターからの磁気を受け、電磁誘導によって発生する電力でデータのやり取りを行うため、接触型よりも省電力である必要がある。

ICカードのセキュリティには公開鍵暗号が用いられており、ICチップは非接触型のように消費電力に制約がある場合や小型化する場合に、回路面積を小さくする必要がある。回路面積が小さくなるとセキュリティが弱くなったり、処理速度が落ちたりする。

本研究では、これからさらに重要となると思われる公開鍵暗号方式で、広く用いられているRSA暗号を扱う。RSA暗号とは大きな素数を掛け合わせることは簡単であるが、掛け合わせた数から元の素数を計算することが困難であることを利用した暗号化アルゴリズムである。RSA暗号は冪乗剰余計算を行うことで暗号化・復号化を行う暗号化方式であり、大きな数の冪乗剰余計算は非常に時間がかかってしまう。ゆえに、剰余計算・べき乗計算に様々な高速アルゴリズムを用いて計算を行っている。

本研究では、RSA暗号に用いられる複数の冪乗剰余計算アルゴリズムを対象として、Verilog-HDLで記述し、FPGAボードへ実装して、回路規模と処理速度の比較・検証を行う。比較・検証を行う剰余アルゴリズムは回復法・インターリーブ法・モンゴメリ法、冪乗計算に用いるアルゴリズムは右向きバイナリ法とする。

本論文では、第２章においてRSA暗号のアルゴリズムを示す。第３章では冪乗計算を行うアルゴリズムの右向きバイナリ法と、剰余計算を行うアルゴリズムの回復法・インターリーブ法・モンゴメリ法のアルゴリズムを示す。第４章では、各アルゴリズムの論理合成の実験結果を示し、考察を行う。

２．ＲＳＡ暗号

ＲＳＡ暗号とは公開鍵暗号のひとつである。公開鍵暗号は、公開鍵と秘密鍵を生成し、公開鍵で暗号化し、秘密鍵で復号化するものである。ＲＳＡ暗号は非常に大きな合成数の素因数分解が困難であることを利用した公開鍵暗号である。
ＲＳＡ暗号のアルゴリズムは次の通りである。

鍵の生成　公開鍵（E，N）　秘密鍵（D，p，q）

STEP1：　２つの素数 p と q を用意する。

STEP2：　$N = p \times q$　とする。

STEP3：　$L = LCM(p-1, q-1)$　とする。

STEP4：　L 未満の整数かつ L と互いに素な数からランダムで選び、E とする。

STEP5：　$D \times E \mod L = 1$ となる D を求める。

【暗号化と復号化】

暗号は　平文^{E} mod N　＝　暗号文

復号は　暗号文^{D} mod N　＝　平文　　となる。

ｐとｑからＮを計算するのは容易だが、Ｎからｐとｑを割り出して（ｐ－１）と（ｑ－１）の最小公倍数を求めるのは非常に困難である。

３．冪乗計算と剰余計算のアルゴリズム

本章では冪乗計算アルゴリズムの右向きバイナリ法、及び剰余計算アルゴリズムの回復法・インターリーブ法・モンゴメリ法の計算法を示す。

回復法は A×B を入力として１０進数で筆算を行う計算を２進数で行うアルゴリズムである。

インターリーブ法は回復法とは違い入力には A と B それぞれをとるため A×B を行う必要がない。また、計算時にも乗算を行うことがない。乗算を行う代わりに、足し算を効率よく使い、計算を行う。

モンゴメリ法は回復法やインターリーブ法とは異なり、計算途中の下位 r ビットが「０」となるように前計算を行っておき、引き算や除算を使用しない代わりにビットシフトを用いて計算する手法である。

３．１　右向きバイナリ法のアルゴリズム

右向きバイナリ法は、指数を２進数表示で上位ビットから１ビットずつ最下位ビットまで見て行き、そのビットの数値にかかわらず２乗し、もし１であればさらに入力値との積を計算するアルゴリズムである。

m bit の入力 A, E が与えられた時、$C = A^E$ を計算する。

- STEP1：　$C = A$
- STEP2：　$i = \lfloor \log_2 E \rfloor$　　($\lfloor x \rfloor$ は x を超えない最大の数)
- STEP3：　$C = C \times C$，$i = i - 1$
- STEP4：　もし $E[i] = 1$ なら　$C = C \times A$
- STEP5：　もし $i \neq 0$ なら STEP3 に戻る
- STEP6：　C を出力

図１に右向きバイナリ法のアルゴリズムのSTEP3・STEP4の１サイクルの回路図を示す。

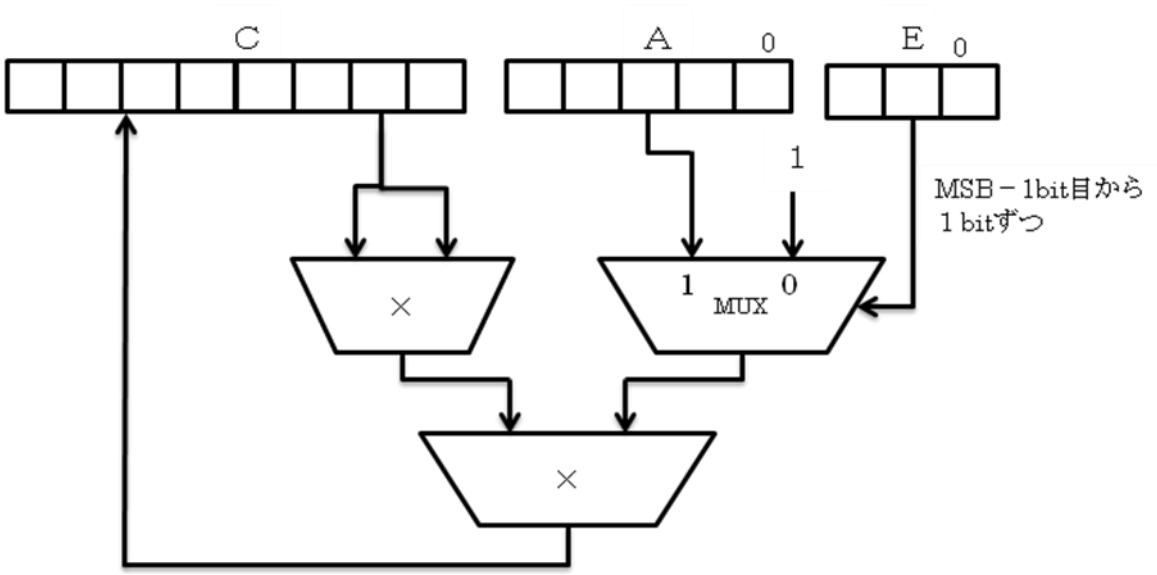

図 1 右向きバイナリ法の回路図

右向きバイナリ法の計算例を表 1 に示す。

入力 $A=3, E=10$ とする $A^E = 3^{10} = 59049$ となる。

表 1 右向きバイナリ法の計算例

	10 = (1010)2
3 × 3 = 9 9 × 1 = 9	0
9 × 9 = 81 81 × 3 = 243	1
243 × 243 = 59049 59049 × 1 = 59049	0

m bitの入力Eが与えられた時、$Y = \lfloor \log_2 E \rfloor$を計算するアルゴリズムを示す。

２進数で表されたEを上位から検索していき、最初に１が現れるのはEの何bit目かを出力する。

- STEP1：　初期化　$i = 0$
- STEP2：　もし$E[m-i]$=1 なら $Y = m - i$　STEP3 へ
  もし$E[m-i]$≠1 なら $i = i + 1$
- STEP3：　Y を出力

図２に、$\lfloor \log_2 E \rfloor$を計算するアルゴリズムのSTEP２の回路図を示す。

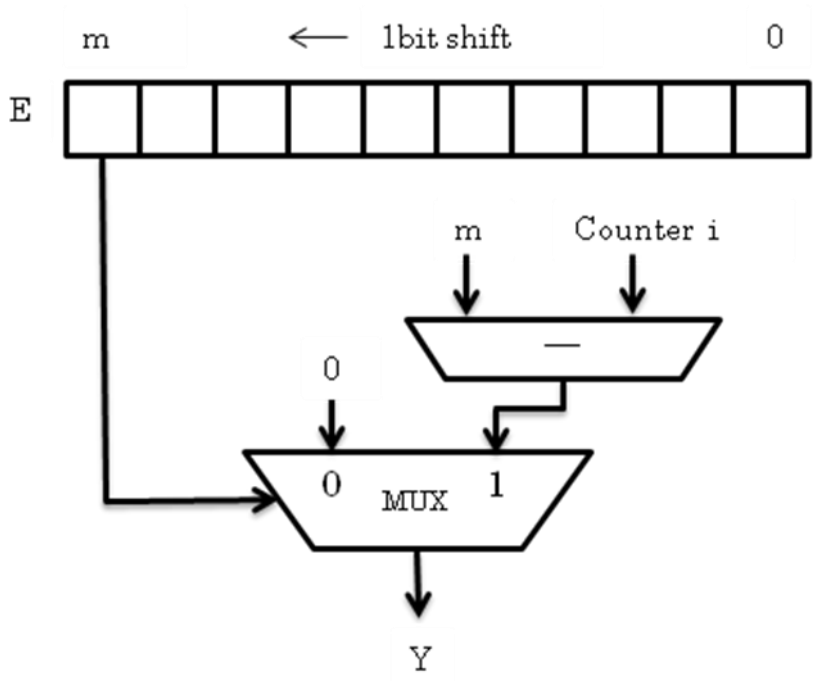

図 2　$\lfloor \log_2 E \rfloor$を計算する回路図

$E = 9 = (01001)_2$ とすると、最下位ビットを０と考えると上位から検索していき最初に１が現れるのは、３ビット目である。$\log_2 9 = 3.1699\ldots$となり、$\lfloor \log_2 9 \rfloor = 3$なので正しい。

３．２ 回復法のアルゴリズム

回復法は割り算を筆算で行うアルゴリズムである。被除数をシフトしていき、除数を超えれば引き算を行い、余りを元に戻すということを被除数の最下位bitまで行う。引くことができた場合に「１」を書き込むと商も得ることができるが、本研究に商は関係がないので無視する。

m bit の入力 N, A, B が与えられた時、$C = A \times B \bmod N$ を計算する。

- STEP1： $C = A \times B$, $i = 2m$ （C は $3m$ bitとする）
- STEP2： C の上位 $m+1$ bit から N を引く
- STEP3： STEP2の結果がマイナスならそのまま,
  プラスならその値を C の上位 $m+1$ bit に代入する。
- STEP4： C を1ビット左シフト, $i = i - 1$
- STEP5： $i \neq 0$ なら STEP2 に戻る
- STEP6： C の上位 m bitを出力

図３に回復法のアルゴリズムのSTEP2からSTEP4の１サイクルの回路図を示す。

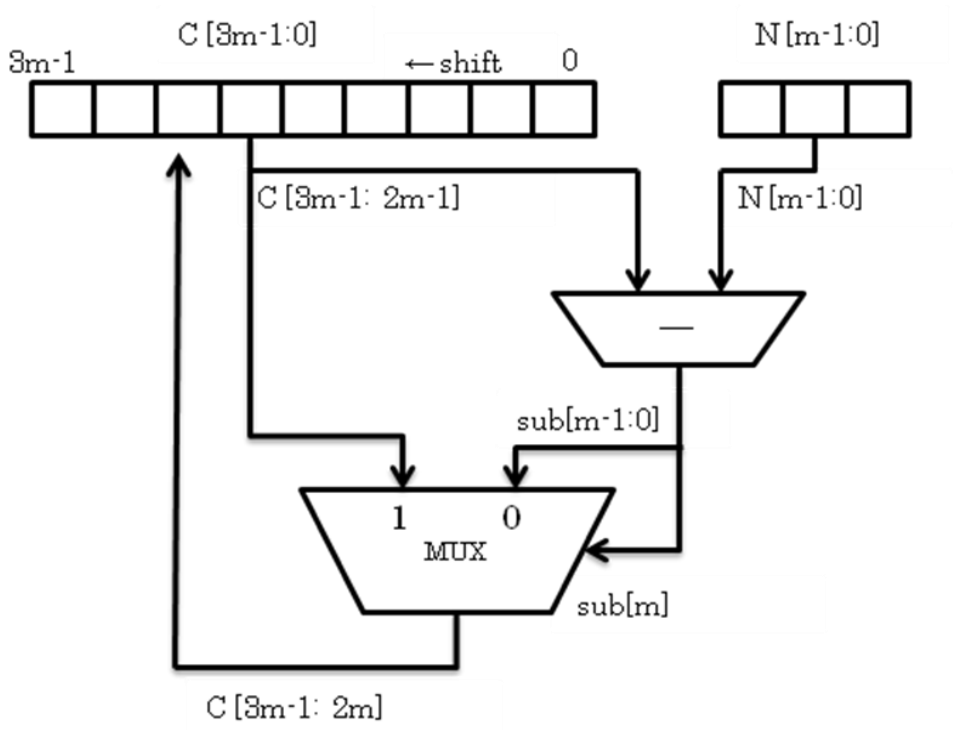

図 3　回復法の回路図

次に、回復法の計算例を表２に示す。

入力 $N=17, A=9, B=13$ とする　$A\times B \bmod N = 9\times 13 \bmod 17 = 15$ となる。

表 2　回復法の計算例

	２進数表記													１０進数		
1						0	1	1	1	0	1	0	1	0		初期値
2					0	1	1	1	0	1	0	1		1		シフト
3				0	1	1	1	0	1	0	1			3		シフト
4			0	1	1	1	0	1	0	1				7		シフト
5	0	1	1	1	0		1	0	1					14		シフト
6	1	1	1	0	1		0	1						29	-17	引き算
7	0	1	1	0	0		0	1						12		シフト
8	1	1	0	0	0		1							24	-17	引き算
9	0	0	1	1	1		1							7		シフト
10	0	1	1	1	1									15		

回復法の $C=A\times B$　を計算するために、ブースの乗算アルゴリズムを使用する。

ブースの乗算アルゴリズムでは、乗数を２進数表記したときに、１のかたまり毎に見ることで少ない回数の加算と減算で乗算を行う。

$(00111110)_2=(2^5+2^4+2^3+2^2+2^1)=62$　を　$62=(2^6-2^1)=(010000-10)_2$　とし、１のかたまりの始まり「01」を見つけると加算を行い、１のかたまりの終わり「10」を見つけると減算を行う。[8]

ブースの乗算アルゴリズムは次の通り。

m bit の符号付整数の入力 A と n bit の符号付整数の入力 B が与えられた時、$C=A\times B$ を計算する。

- STEP1：　$S=-A$ を用意する。　A, S, C を $m+n+1$ に拡張し,
  A, S の先頭 m bitにそれぞれ A, S を入れ、C の先頭 m bitは0を入れる。
  A, S の次の n bitには0を入れ、C の次の n bitに B を入れる。
  A, S, C の最後の1bitには0を入れる。
- STEP2：　C の下位2bitが「00」「11」の場合は何もしない。
  「01」 なら $C=C+A$ 、「10」なら $C=C+S$
- STEP3：C を1bit右算術シフト
  STEP2 を n 回繰り返し、C の LSB 以外を出力する。

図４にブースの乗算アルゴリズムのSTEP1からSTEP3の１サイクルの回路図を示す。

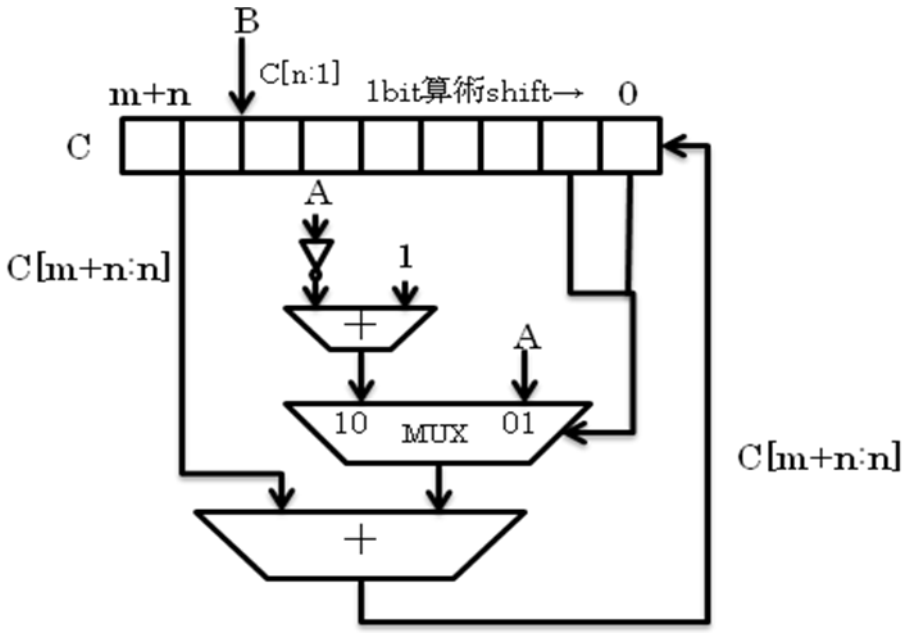

図 4　ブースの乗算アルゴリズムの回路図

ブースの乗算アルゴリズムの計算例を表３に示す。

入力は $A=5,\ B=7$ とする。

表 3　ブースの乗算アルゴリズムの計算例

初期値	0	0	0	0	0	1	1	1	0	下位２bit	動作
１回目	0	0	0	0	0	1	1	1	0	10	減算
	1	0	1	1	0	1	1	1	0		算術シフト
２回目	1	1	0	1	1	0	1	1	1	11	算術シフト
３回目	1	1	1	0	1	1	0	1	1	11	算術シフト
４回目	1	1	1	1	0	1	1	0	1	01	加算
	0	1	0	0	0	1	1	0	1		算術シフト
答え	0	0	1	0	0	0	1	1	0		
	128	64	32	16	8	4	2	1			

結果が３５となっているので正しく動作しているとわかる。

３．３ インターリーブ法のアルゴリズム

m bit の入力 N, A, B が与えられた時、$C = A \times B \bmod N$ を計算する。

- STEP1： 初期化 $C = 0$, $j = 0$, $i = 0$
- STEP2： C を１ビット左シフト
- STEP3： もし $B[m-1-j]$ が１なら $C = C + A$, $j = j + 1$, $i = i + 1$
  もし $B[m-1-j]$ が０なら $j = j + 1$, $i = i + 1$
- STEP4： $C < N$ になるまで $C = C - N$
- STEP5： $i < m$ なら STEP2 へ
- STEP6： C を出力

図５にインターリーブ法のアルゴリズムのSTEP２からSTEP４の１サイクルの回路図を示す。

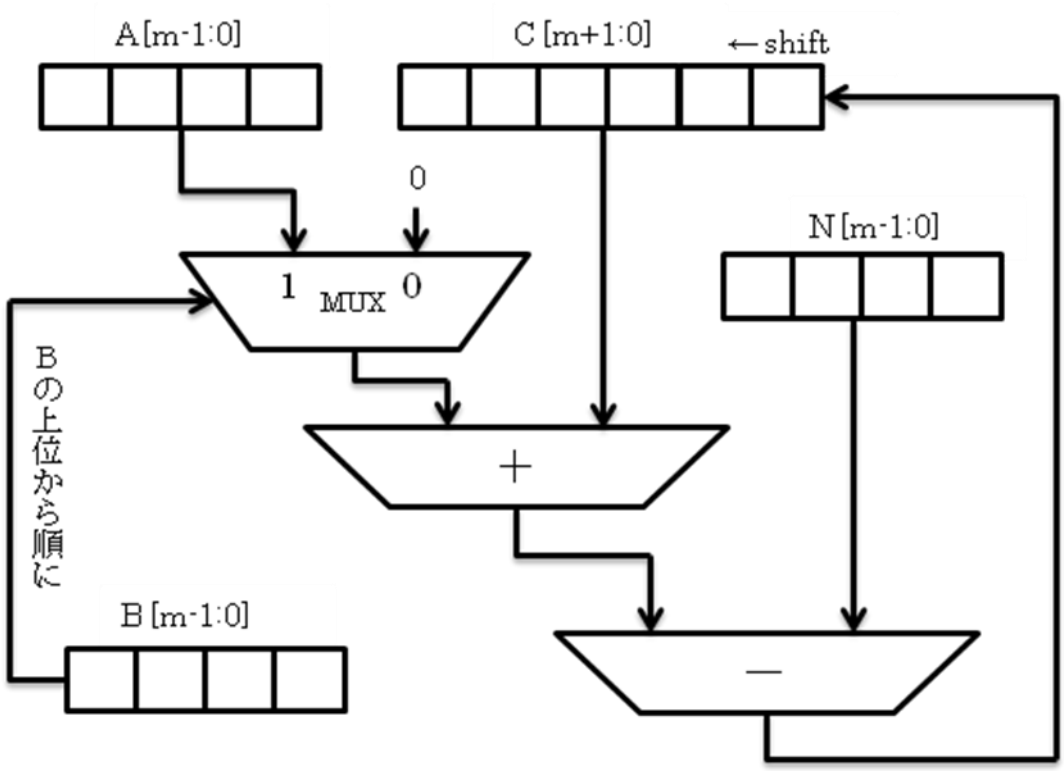

図 5　インターリーブ法の回路図

インターリーブ法の計算例を表４示す。
入力 $N=17, A=9, B=13$ とする　$A\times B \bmod N = 9\times 13 \bmod 17 = 15$ となる。

表 4　インターリーブ法の計算例

１０進数表記					２進数表記									
				0		0	0	0	0	0	0	0	初期値	13 = (1101)2
0	×	2	=	0		0	0	0	0	0	0	0	シフト	
0	＋	9	=	9	+	0	0	0	1	0	0	1	足し算	1
						0	0	0	1	0	0	1	結果	
9	×	2	=	18		0	0	1	0	0	1	0	シフト	
18	＋	9	=	27	+	0	0	0	1	0	0	1	足し算	
						0	0	1	1	0	1	1	結果	1
27	-	17	=	10	-	0	0	1	0	0	0	1	引き算	
						0	0	0	1	0	1	0	結果	
10	×	2	=	20		0	0	1	0	1	0	0	シフト	
20	＋	0	=	20	+	0	0	0	0	0	0	0	足し算	
						0	0	1	0	1	0	0	結果	0
20	-	17	=	3	-	0	0	1	0	0	0	1	引き算	
						0	0	0	0	0	1	1	結果	
3	×	2	=	6		0	0	0	0	1	1	0	シフト	
6	＋	9	=	15	+	0	0	0	1	0	0	1	足し算	1
						0	0	0	1	1	1	1	結果	

３．４ モンゴメリ法

（１） モンゴメリ法のアルゴリズム

モンゴメリ法は割り算の代わりにビットシフトを利用して計算する方法である。

入力 A， B を 2^r bitずつのブロックに区切り、部分積をとりながら順に計算を行う。

A の１を含む最後のブロックを計算したとしても、$\left\lceil \log_2 R \middle/ 2^r \right\rceil$ 回計算を行う必要がある。

m bit　の入力　$N, 0<A<N, 0<B<N$ が与えられた時、$C = A\times B\times R^{-1} \bmod N$ を計算する。

- STEP1: $V = -N^{-1} \bmod 2^r$ （N^{-1} の計算方法は後述）

  $T = \left\lceil \log_2 R / 2^r \right\rceil$ （この計算方法は後述） $(R = 2^{2d} > N)$
- STEP2: $C = 0 \ , i = 0$
- STEP3: $D = (C + A_i \times B_0)V \bmod 2^r$
- STEP4: $C = C + A_i \times B + D \times N$
- STEP5: $C = C / 2^r \ , i = i + 1$
- STEP6: もし $i < T$ なら STEP3 へ
- STEP7: もし $N < C$ なら $C = C - N$
- STEP8: C を出力

図６にモンゴメリ法のアルゴリズムのSTEP3からSTEP5の１サイクルの回路図を示す。

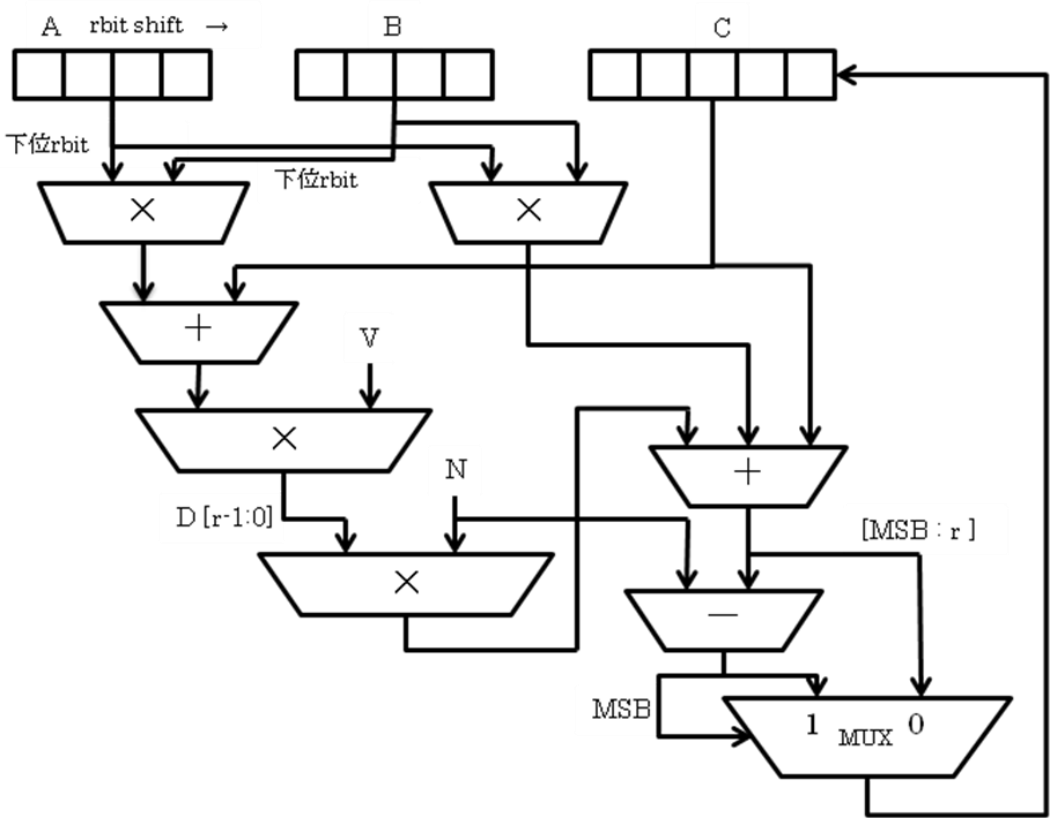

図 6　モンゴメリ法の回路図

モンゴメリ法の計算例を表 5 と表 6 に示す。

入力 $N=53,\ A=23,\ B=37$ とする $A\times B \mod N = 23\times 37 \mod 57 = 3$ となる。

表 5　モンゴメリ法の計算例の入力値

A	B	N	R	r	V	T	A × B mod N
23=(010111)2	37=(100101)2	53	64 = 2^6	2	3	3	23 × 37 mod 53 = 3

表 6　モンゴメリ法の計算例

STEP			
2	C = 0	C = 0	i = 0
3	D = 1	D = (0 + 3 × 1) × 3 mod 4	
4	C = 164	C = 0 + 3 × 37 + 1 × 53	
5	C = 41	C = 164/4	i = 1
3	D = 2	D = (41 + 1 × 1) × 3 mod 4	
4	C = 184	C = 41 + 1 × 37 + 2 × 53	
5	C = 46	C = 184/4	i = 2
3	D = 1	D = (46 + 1 × 1) × 3 mod 4	
4	C = 136	C = 46 + 1 × 37 + 1 × 53	
5	C = 34	C = 136/4	i = 3

$C = A\times B\times R^{-1} \mod N$ なので、$A\times B \mod N = C\times R \mod N$ となるか確認する。

表 7　モンゴメリ法の確認のための計算

A × B mod N	=	C × R mod N
23 × 37 mod 53 = 3	=	34 × 64 mod 53 = 3

左辺と右辺がイコールとなるので、正しく計算されていることが分かる。

（2）$\left\lceil \log_2 R/r \right\rceil$の計算方法

$R = 2^{2d} > N$ なので $\lfloor \log_2 E \rfloor$とは計算方法が異なる。

2^x bit　の入力 N を与えられた時、$Y = \left\lceil \log_2 R/r \right\rceil$ を計算する。

- STEP1：　初期化　$i = 0$
- STEP2：　もし $N[2^x - r \times i]$から $N[2^x - r \times i - (r-1)]$ の中に

  1 のビットがあれば　$Y = m - i$ ，　STEP5　へ
- STEP3：　$i = i + 1$ ，　STEP2 へ戻る
- STEP4：　Y を出力する

図 7 に$\left\lceil \log_2 R/r \right\rceil$を計算するための回路図を示す。

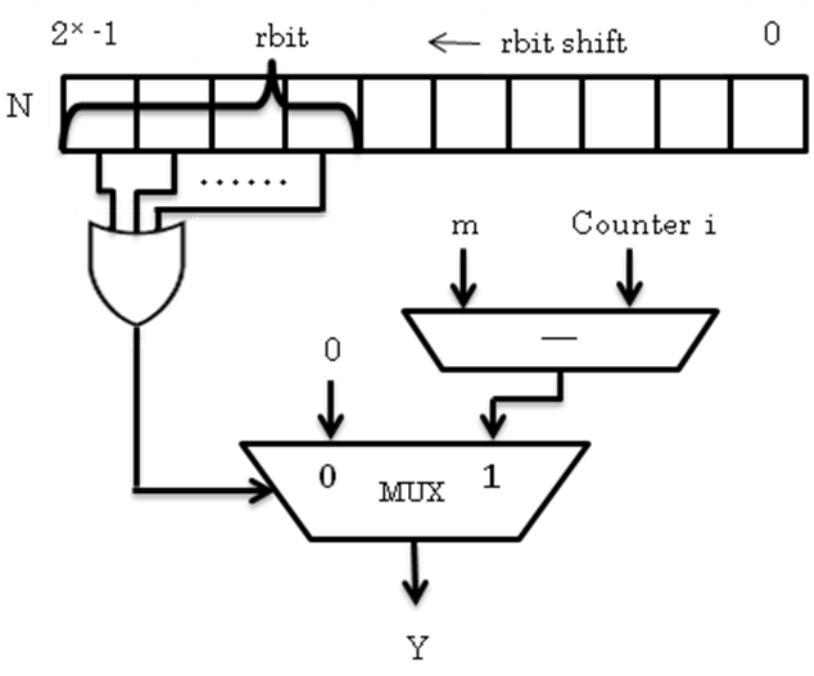

図 7　log2(R)を下回らない数を計算する回路図

$\left\lceil \log_2 R/r \right\rceil$ の計算例を示す。

$N=33=(00100001)_2$ とすると、$R=64=2^6$ となるので、$\log_2 R=6$ となる。

$r=2$、$r=4$、$r=8$ のときを考えてみる。

$r=2$ のときを考える。このときの $m=8/2=4$ である。検索を開始すると、1 がある最初のビットが、最上位から 3 ビット目にあるので、2 サイクル目で見つかることになる。そうすると $m=4$、$i=1$ なので、$Y=m-i=3$ となる。これは

$Y=\left\lceil \log_2 64/2 \right\rceil=\left\lceil 6/2 \right\rceil=\lceil 3 \rceil=3$ であるので正しい。

次に、$r=4$ のときを考える。このときの $m=8/4=2$ である。検索を開始すると、1 がある最初のビットが、最上位から 3 ビット目にあるので、1 サイクル目で見つかることになる。そうすると $m=2$、$i=0$ なので、$Y=m-i=2$ となる。これは

$Y=\left\lceil \log_2 64/4 \right\rceil=\left\lceil 6/4 \right\rceil=\lceil 1.5 \rceil=2$ であるので正しい。

次に、$r=8$ のときを考える。このときの $m=8/8=1$ である。検索を開始すると、1 がある最初のビットが、最上位から 3 ビット目にあるので、1 サイクル目で見つかることになる。そうすると $m=1$、$i=0$ なので、$Y=m-i=1$ となる。これは

$Y=\left\lceil \log_2 64/8 \right\rceil=\left\lceil 6/8 \right\rceil=\lceil 0.75 \rceil=1$ であるので正しい。

（3）V の計算方法

入力　N の下位 $r+1$bit，2^r として　拡張ユークリッドの互除法を利用する。

拡張ユークリッドの互除法のアルゴリズムは以下の通りである。

r+1 bit の 2^r と　N の下位 $r+1bit$ を 与えられた時、N の逆元 $y1 = N^{-1}$ を計算する。

- STEP1:　$x1=1,\ y1=0,\ z1=2^r,$

  $x2=0,\ y2=1,\ z2=N$の下位$r+1bit$

- STEP2:　$q = z1 / z2$
- STEP3:　$(x2, y2, z2) = (x1, y1, z1) - q \times (x2, y2, z2)$

  $(x1, y1, z1) = (x2, y2, z2)$
- STEP4:　もし　$z2 \neq 0$　なら　STEP2　へ
- STEP5:　もし　$y1 < 0$　なら　$y1 = y1 + 2^r$
- STEP6:　$y1$　を出力

この計算で出た $y1$ は N^{-1} なので、V にするためには、N^{-1} を $-N^{-1}$ とするために２の補数表現にし、$\mathrm{mod}\ 2^r$ を計算するために、$-N^{-1}+2^r$ を計算する。

この計算は、N が決まっており r が固定ならば、本計算とは関係なく先に計算しておくことができる。STEP2 の割り算は、r が小さければ引き算の繰り返しでも十分だが、r が大きい場合は別の除算アルゴリズムを私用したほうが良い。
図８に拡張ユークリッドの互除法アルゴリズムの STEP２から STEP３の回路図を示す。

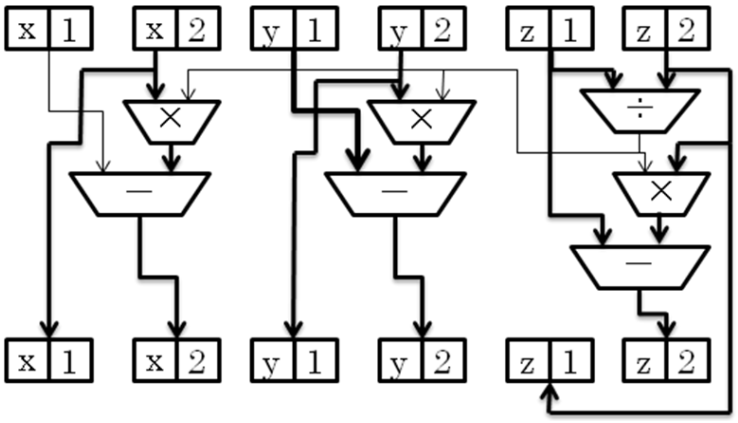

図 8　拡張ユークリッドの互除法の回路図

拡張ユークリッドの互除法の計算例を表８に示す。

r=2 のとき、$N = 43(101011)_2$ とすると N の下位3bit ＝ $3(011)_2$ となる。

表 8　拡張ユークリッドの互除法の計算例

r=2	x1	y1	z1	x2	y2	z2	q = z1/z2
初期値	1	0	4	0	1	3	
１回目	0	1	3	1	-1	1	1
２回目	1	-1	1	-3	4	0	3

z2 ＝ 0 のとき、y1 = −1 なので $y1 = y1 + 2^r$ となり、 y1 = −1 + 4 = 3 となる。

このｙ１は、$V = -N^{-1} \bmod 2^r$ を計算するための N^{-1} であるので、V を計算する。

$V = -N^{-1} \bmod 2^r = -3 \bmod 4 = 1$ となる。$V = 1$ となるか普通の計算で確かめてみる。

$V = -N^{-1} \bmod 2^r = -\frac{1}{43} \bmod 4$ より $43V = -1 \bmod 4$ となり $V = 1$ が確認できる。

拡張ユークリッドの互除法を用いて、V を計算する時、拡張ユークリッドの互除法の入力には N の全てのビットをとらずに、N の下位$r+1bit$ だけをとればよいことがわかる。

モンゴメリ法では計算結果が $A \times B \times R^{-1} \bmod N$ となっているため、A をモンゴメリ法の入力にとり、冪乗剰余計算を行うと

$$(A \times A \times R^{-1} \bmod N) \times \underbrace{(A \times R^{-1} \bmod N) \times (A \times R^{-1} \bmod N) \quad \cdots \quad \times (A \times R^{-1} \bmod N)}_{E-2\text{回}}$$

となり、計算結果が $A^E \times R^{-(E-1)} \bmod N$ となってしまい、誤った計算結果となる。

モンゴメリ法で冪乗剰余計算を行うためには入力を $A \times R \bmod N$ とする。$A \times R \bmod N$ を入力にとると、

$$(AR \times AR \times R^{-1} \bmod N) \underbrace{\times (AR \times R^{-1} \bmod N) \times (AR \times R^{-1} \bmod N) \cdots \times (AR \times R^{-1} \bmod N)}_{E-2\text{回}}$$

となり計算結果が $A^E \times R \bmod N$ となるので、最後に $A^E \times R \bmod N$ と1を入力にとる計算を行えば $(A^E \times R \bmod N) \times 1 \times R^{-1} \bmod N = A^E \bmod N$ となり、正しい結果が得られる。

$AR \bmod N$ を計算するためには R を求める必要がある。R は N より大きい 2^{2d} を満たす数である。R を求めるアルゴリズムは次の通りである。回路図を図9に示す。

（4） R の計算方法

$2m$ bit の N を与えられた時、$Y = R$ （$R = 2^{2d} > N$ を満たす）を計算する。

STEP1： 初期化 $i = 0$

- STEP2： もし $N[2m-2i-1]$=1 又は $N[2m-2i-2]$=1 なら
  $Y[2m-2i] = 1$ とし，STEP5 へ
- STEP4： $i = i+1$ ， STEP2 へ戻る
- STEP5： Y を出力する

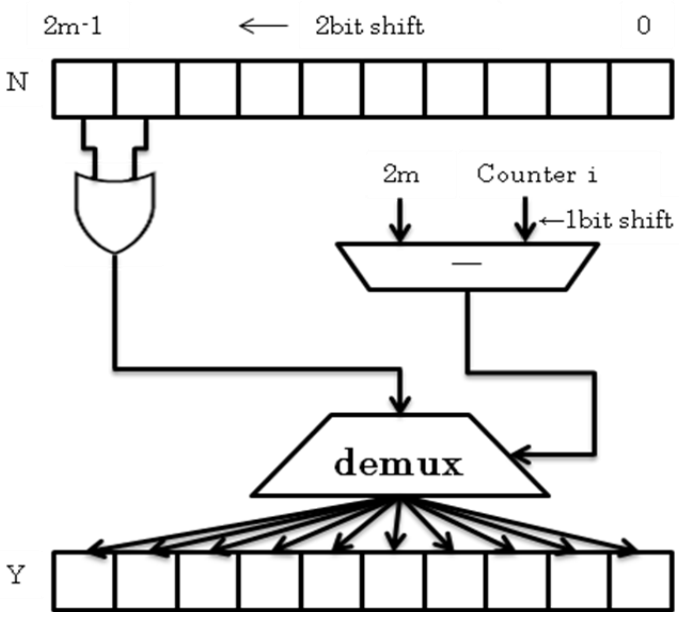

図 9 R を計算する回路図

R の計算例を示す。

$N = 6(0110)_2$ とする。$2m = 4$ なので、$m = 2$ である。1サイクル目で3ビット目の1を見つけるので、$i = 0$ となり、$Y[2m-2i] = Y[4-0] = Y[4] = 1$ とするので、$Y = (10000)_2$ となり、$Y = 2^4 = 16$ となる。これは6以上で、最初に出てくる 2^{2d} を満たす数であるから正しいと言える。

４． 冪乗剰余計算の FPGA ボードへの実装と比較

４．１ 実験内容

アルゴリズム毎に冪乗剰余計算を行い、処理時間とFPGAの使用率を比べる。

入力はA，E，N　とし、出力を $A^E \bmod N$ とする。

なお、実験に使用するボードはXilinx社製「Spartan™-3 Starter Kit」である。

ＦＰＧＡ使用率はXilinx-ISEのSynthesis Reportで確認する。

４．１．１　回復法と右向きバイナリ法

回復法を使用するのは積が法を超えた場合のみにすることで極力計算量を抑える。

A×B　を論理合成にまかせた記述をしたものと、A×B　をブースの乗算アルゴリズムを利用した記述をしたもので比較を行う。

図10に回復法と右向きバイナリ法のブロック図を示す。

４．１．２　インターリーブ法と右向きバイナリ法

STEP3を $P = P + A \times B[m-1-j]$ とすることもできるが、ifの方が掛け算よりも回路規模が小さいので、if文を採用した。

図11にインターリーブ法と右向きバイナリ法のブロック図を示す。

４．１．３　モンゴメリ法と右向きバイナリ法

r を A，B のbit数まで大きくすると１度で計算をすることができるが、$A_i \times B$ が大きくなってしまうので、r を小さくし、部分積を計算するようにする。

$r = 2$ の場合と $r = 4$ の場合の、拡張ユークリッドの互除法を用いて逆元を計算する場合と、拡張ユークリッドの互除法を使わず、Nの下位２ビットが１ならV=３、３ならV=１として計算を行うもので比較を行う。

また、乗算を論理合成にまかせた記述したものと、乗算をANDとビットシフトで記述したもので比較を行う。

図12にモンゴメリ法と右向きバイナリ法のブロック図を示す。

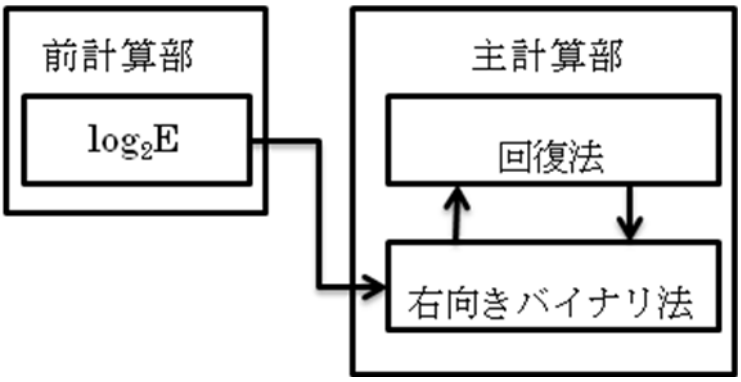

図 10　回復法のブロック図

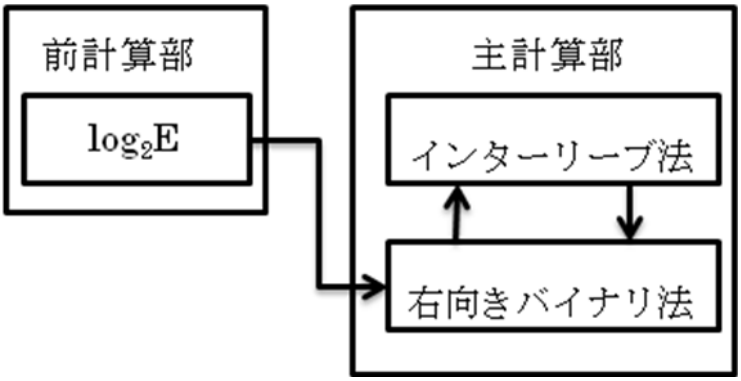

図 11　インターリーブ法のブロック図

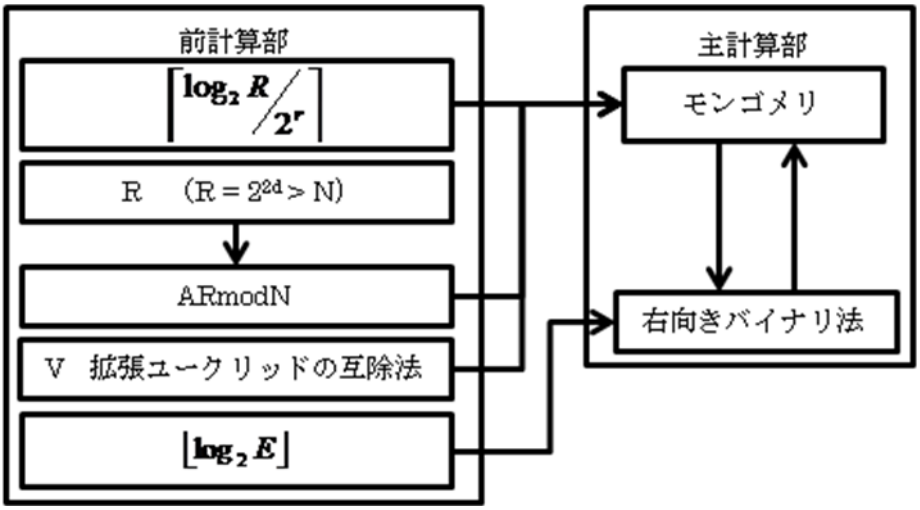

図 12　モンゴメリ法のブロック図

４．２ 実験結果

回復法、インターリーブ法、モンゴメリ法で冪乗剰余計算を行うために用いる全モジュールでの論理合成結果を示す。

Spartan-3 xc3s200 のスライス数が1920となっているので、1920スライス以下のものはFPGAボード上でクロック数を調べ、1920スライスを超えるものについてはModelSim上でクロック数を調べる。

処理時間は $\frac{clock数}{最大動作周波数}$ で求める。

入力する平文のデータは入力ビットを全て１で埋めたものを使用する。

入力する冪指数のデータは65537という17bitのものがよく使用されているので、17bitとする。しかし、65337ではなく、17bitすべてを１で埋めたものとする。

入力する鍵データは、256bit以下のものはそれぞれ使用する２つの素数を最大のものから２つ選び、掛け合わせたものとする。また、512bit以上のものに関しては、「RSA Security 社」のコンテストの「RSA Factoring Challenge」で使用されたものとする。512bitではRSA-155、1024bitではRSA-1024、2048bitではRSA-2048のものを使用する。

表に各アルゴリズムのslice数、最大周波数、あるデータを入力とした時のクロック数、クロック数を最大周波数で割った処理時間を示す。

表9に回復法の乗算を＊で記述し、論理合成に任せた場合の、回路の論理合成結果を示す。

表10に回復法の乗算をブースの乗算アルゴリズムで記述した回路の論理合成結果を示す。

表11にインターリーブ法の回路の論理合成結果を示す。

表12にモンゴメリ法の乗算を＊で記述し、Vの計算を拡張ユークリッドの互除法を使用しない回路の論理合成結果を示す。

表13にモンゴメリ法ｒ＝２とした時の、乗算をANDとシフトで記述し、Vの計算を拡張ユークリッドの互除法を使用しない回路の論理合成結果を示す。

表14にモンゴメリ法ｒ＝４とした時の、乗算をANDとシフトで記述し、Vの計算を拡張ユークリッドの互除法を使用しない回路の論理合成結果を示す。

表15にモンゴメリ法ｒ＝２とした時の、乗算をANDとシフトで記述し、Vの計算を拡張ユークリッドの互除法を使用する回路の論理合成結果を示す。

表16にモンゴメリ法ｒ＝４とした時の、乗算をANDとシフトで記述し、Vの計算を拡張ユークリッドの互除法を使用する回路の論理合成結果を示す。

表 9　回復法　乗算部を＊で記述

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間(ms)
32	1615	44.643	8325	0.186479403
64	10314	32.951	16517	0.501259446
128	40076	22.898	32901	1.43685038
256	－			
512	－			
1024	－			
2048	－			

表 10　回復法　ブースの乗算アルゴリズム使用

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間(ms)
32	724	94.597	14612	0.154465786
64	1425	62.896	28948	0.460251844
128	2793	37.657	57620	1.530127201
256	5530	20.891	114964	5.503039586
512	11016	11.051	230100	20.8216451
1024	21959	5.688	459028	80.70112518
2048	43843	2.888	917780	317.7908587

表 11　インターリーブ法

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間(ms)
32	204	109.25	5018	0.04593135
64	389	91.568	9971	0.108891753
128	710	67.625	19989	0.295585952
256	1354	45.976	39239	0.853467026
512	2626	25.328	81145	3.203766582
1024	5181	14.549	160651	11.04206475
2048	10304	8.175	318889	39.00782875

表 12　モンゴメリ法　ｒ＝２　乗算部を＊で記述

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間(ms)
32	636	98.52	4434	0.04500609
64	1173	83.825	8794	0.104909037
128	2256	63.643	17506	0.2750656
256	4473	43.575	34912	0.801193345
512	8708	26.552	70448	2.653208798
1024	17621	15.245	141034	9.251164316
2048	36991	8.139	281832	34.6273498

表 13　モンゴメリ法　ｒ＝２　乗算部分を AND とシフトで表現

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間(ms)
32	603	107.758	4434	0.041147757
64	1088	90.061	8794	0.09764493
128	2097	66.781	17506	0.262140429
256	4131	43.905	34912	0.795171393
512	8145	27.262	70448	2.58410975
1024	16127	15.34	141034	9.193872229
2048	36026	8.139	281832	34.6273498

表 14 モンゴメリ法　ｒ＝４　乗算部分を AND とシフトで表現

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間(ms)
32	751	83.786	2322	0.02771346
64	1388	72.656	4570	0.062899141
128	2580	57.857	9058	0.156558411
256	5200	40.466	18016	0.445213265
512	12008	24.966	36656	1.468236802
1024	24486	14.449	73450	5.083396775
2048	47701	7.934	146664	18.48550542

表 15　モンゴメリ法 r=2　拡張ユークリッドの互除法を使用

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間 ms
32	687	109.229	4434	0.040593615
64	1216	90.061	8794	0.09764493
128	2297	66.781	17506	0.262140429
256	4467	43.905	34912	0.795171393
512	8748	27.262	70448	2.58410975
1024	17282	15.34	141034	9.193872229
2048	38290	8.139	281832	34.6273498

表 16　モンゴメリ法 r＝4　拡張ユークリッドの互除法を使用

入力 bit 幅	スライス数	最大動作周波数(MHz)	処理クロック	処理時間 ms
32	829	83.786	2322	0.02771346
64	1482	72.84	4570	0.062740253
128	2779	57.857	9058	0.156558411
256	5537	40.466	18016	0.445213265
512	12620	24.966	36656	1.468236802
1024	25649	14.449	73450	5.083396775
2048	49967	7.934	146664	18.48550542

図 13 に各アルゴリズムの bit 幅と slice 数の関係をグラフで示す。

回復法は表 10 のブースの乗算アルゴリズムを使用したものとし、モンゴメリ法は表 13 の r=２の時、乗算を AND と SHIFT で置き換えたものとする。

図 14 に各アルゴリズムの bit 幅と処理時間の関係を示す。

図 15 に各アルゴリズムのスライス数×処理時間を示す。スライス数×処理時間は小さい程、面積使用効率が良いと言える。回復法は表 10 のブースの乗算アルゴリズムを使用したものとし、モンゴメリ法は表 13 の r=２の時、乗算を AND と SHIFT で置き換えたものとする。

図 16 にモンゴメリ法の表 12 から表 16 の結果のスライス数や処理時間の違いをグラフに示す。

図 17 にモンゴメリ法のそれぞれの回路の効率を示す。

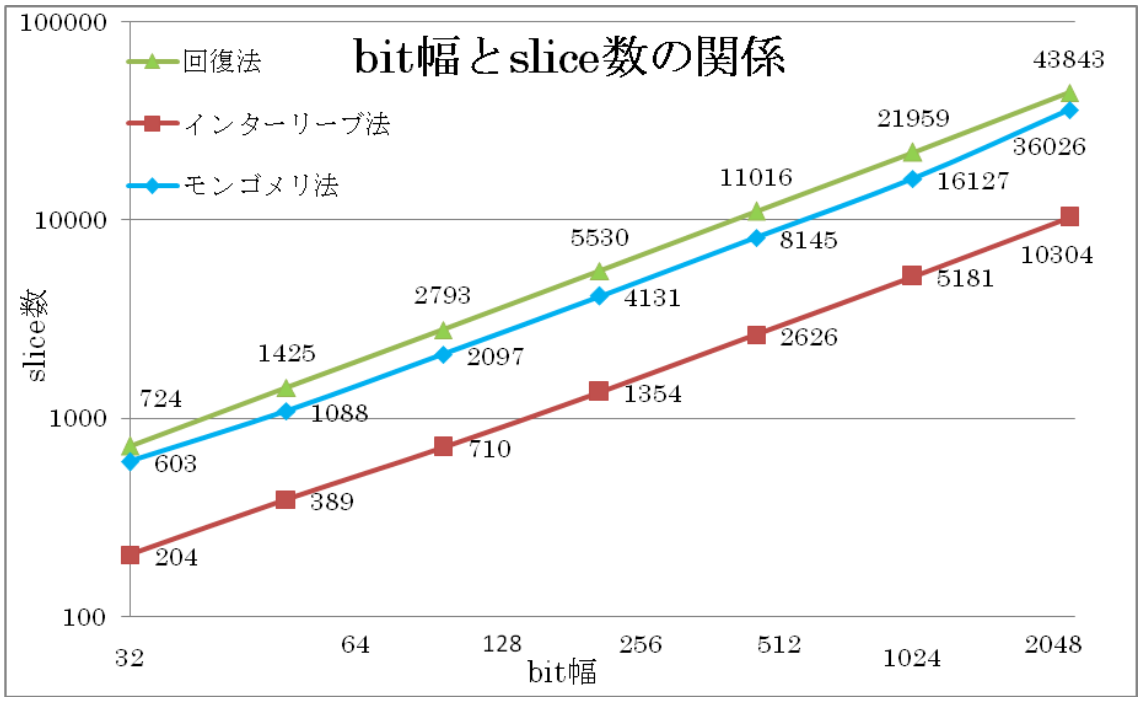

図 13　アルゴリズムの bit 幅と slice 数の関係

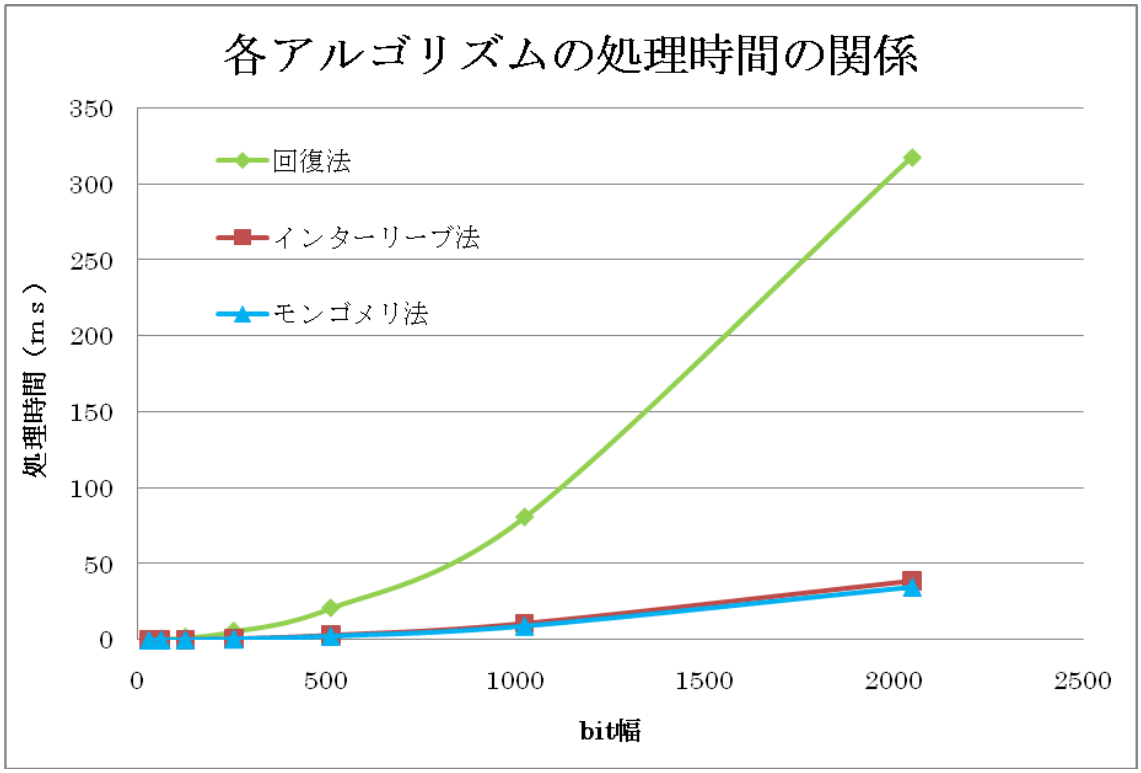

図 14　各アルゴリズムの処理時間の関係

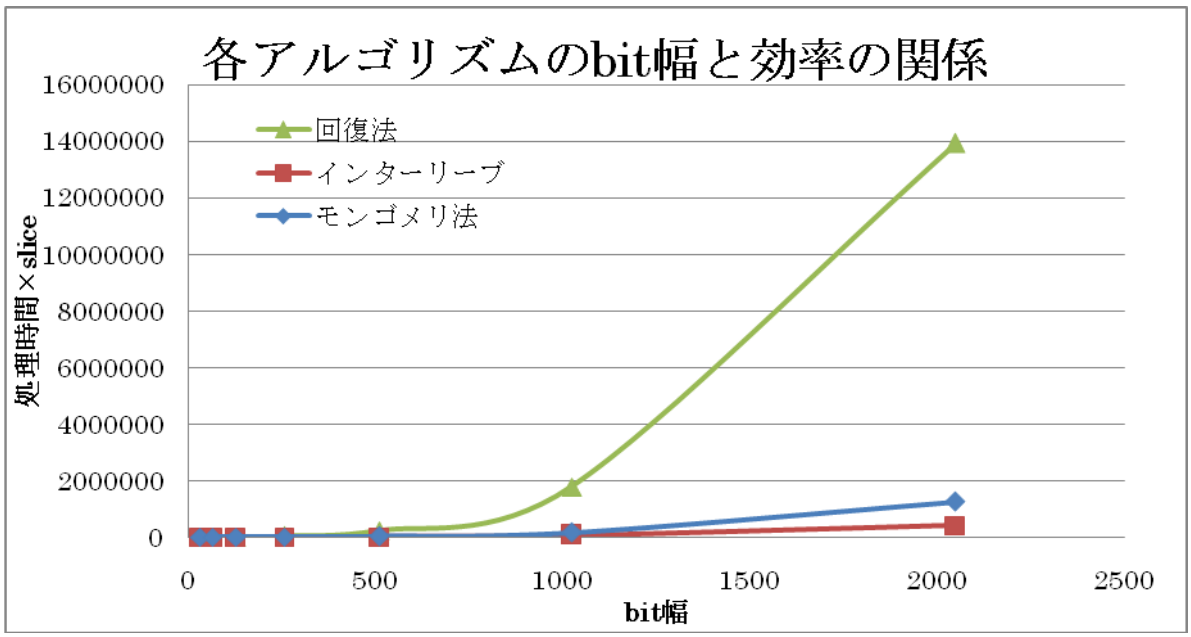

図 15　各アルゴリズムの bit 幅と効率の関係

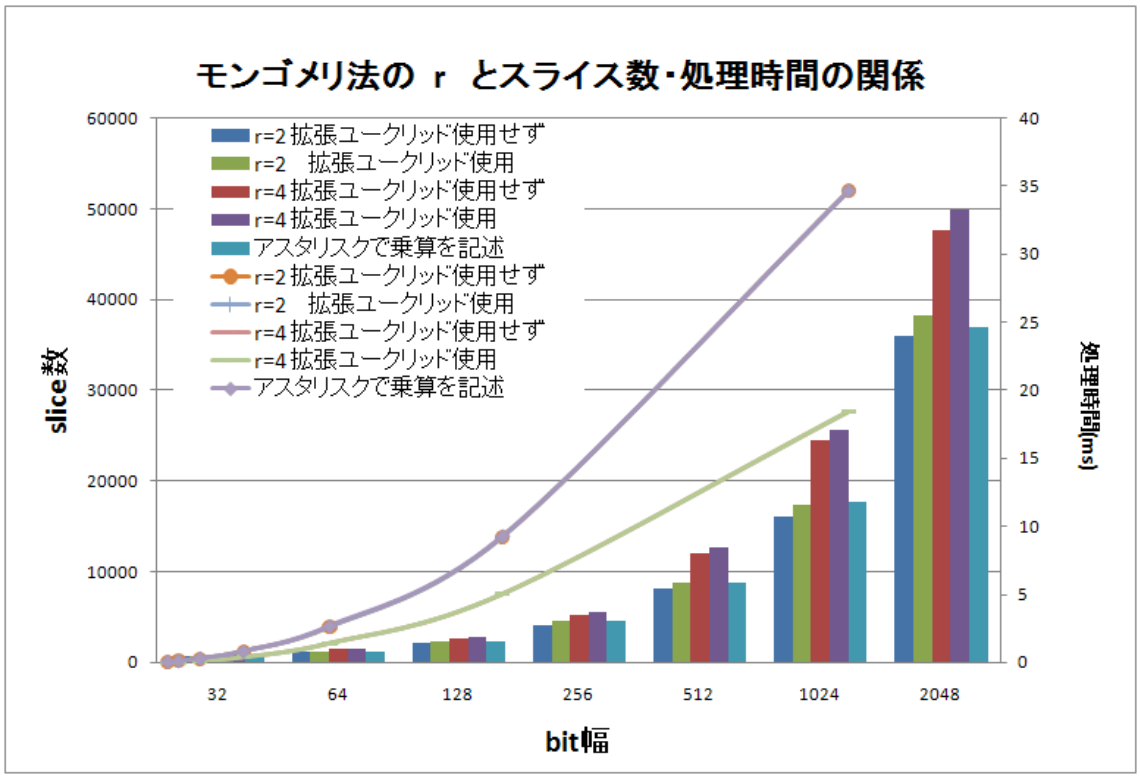

図 16　モンゴメリ法の基数 r とスライス数・処理時間の関係

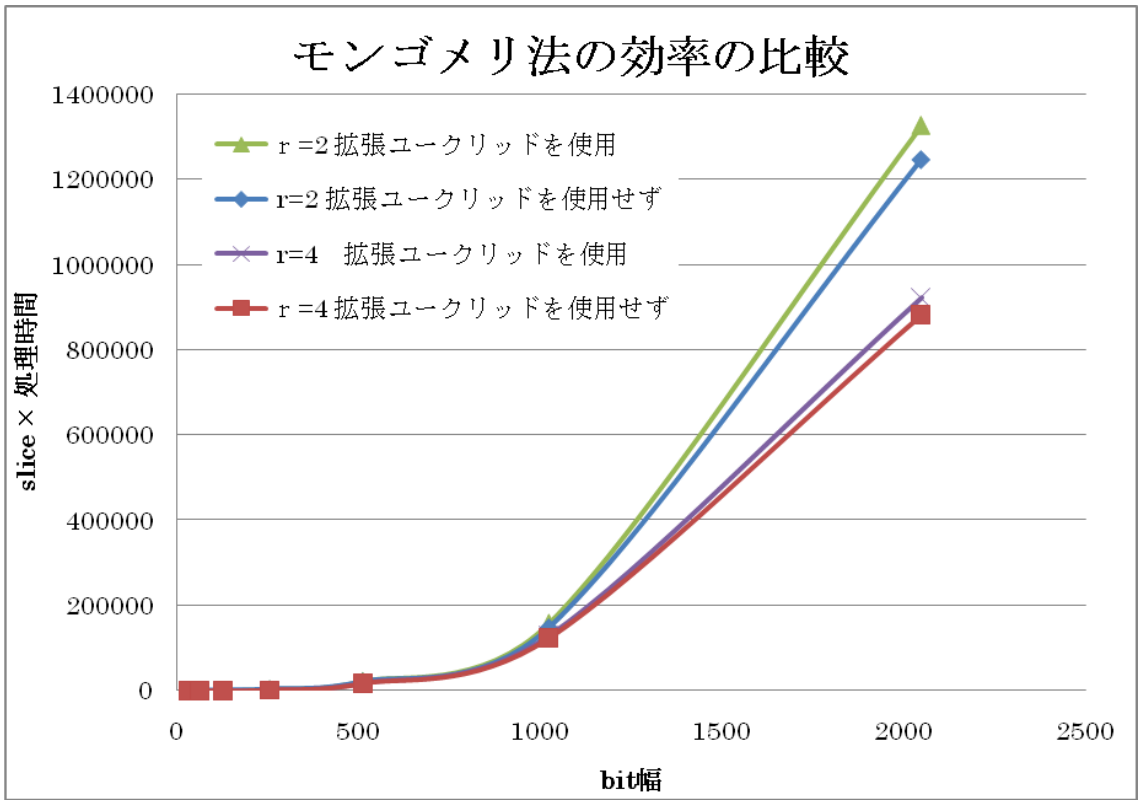

図 17　モンゴメリ法の効率の比較

４．３ 考察

表9と表10を比べると、表9の乗算を論理合成ツールに任せた場合は、論理合成が出来ないほど、回路が大きくなったのに対し、ブースの乗算アルゴリズムを使用した表２の方は表9に比べて非常に回路規模が小さくなった。Spartan-3には乗算器が搭載されているが、なるべく使用しない方が良いことが分かる。

図13でslice数を比較すると、インターリーブ法の回路規模が最小であることが分かる。インターリーブ法は、図15を見ると、一番効率が良いことが分かる。よって、回路規模を重視した回路では、インターリーブ法を用いた回路を設計することが望ましい。

図14で処理時間を比較すると、モンゴメリ法が一番早いことが分かる。モンゴメリ法はインターリーブ法や回復法よりも、事前に計算することが多い。もし、モンゴメリ法で使用する入力値 N や E が、ある程度決まっている場合は、事前に計算し、保持しておくことで、前計算部が入力される平文に依存する $A \times R \bmod N$ だけになり、さらに処理時間を短くすることができる。計算速度を重視した回路では、モンゴメリ法を使用することが望ましい。

図16では、モンゴメリ法のrが２の時と４の時、拡張ユークリッドの互除法を用いて、Vを計算した場合と、拡張ユークリッドの互除法を使用せずに、Nの下位rbitを入力とし、あらかじめ計算しておいたVを出力する場合の比較を行っている。まず、ｒについて検証してみると、ｒが小さい方が回路規模は小さいことが分かる。これは、ｒを大きくすると途中の乗算の桁が大きくなってしまうことからも分かる。拡張ユークリッドの互除法を使用した場合と、使用しない場合のデータを見ると、拡張ユークリッドの互除法を使用せずに、Vを出力する回路の方が回路規模は小さくなっている。

図17では、モンゴメリ法のｒを２または４とし、拡張ユークリッドの互除法を使用した場合と使用しなかった場合の回路の効率を検証する。表を見るとｒ＝４の方が、ｒ＝２よりも効率が良いことが分かる。回路規模は1.2倍程しか大きくはならないが、処理時間が半分程度になっているからである。

５． おわりに

本研究では、回復法、インターリーブ法、モンゴメリ法を用いた剰余計算の実装と検証を行った。回路規模ではインターリーブ法、処理速度ではモンゴメリ法が適しているという結果となった。また、モンゴメリ法では、基数ｒは２よりも４の方が回路面積の効率が良いという結果となった。

今後の課題として、ｒが２、４の場合では拡張ユークリッドの互除法を使用しない場合の方が回路規模は小さいことが分かったが、ｒが８や１６と大きくなっていった場合は、保持しておく回路が拡張ユークリッドの互除法の回路規模を超えてしまう可能性はあるため、検証が必要である。ｒが８以上の時の検証は行っていないが、ｒが一定以上大きくなってしまうと、乗算の関係で効率が落ちていくと予想できるので、一番効率のよい基数ｒは何かを調べる。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導をいただきました山崎勝弘教授に深く感謝いたします。 また、貴重な助言を下さった高性能計算研究室の皆様に心より深く感謝いたします。

参考文献

[1] 神永正博，渡邊高志：情報セキュリティの理論と技術:暗号理論から IC カードの耐タンパー技術まで，森北出版，2005
[2] 山本　聡：高速剰余算アルゴリズムとその実装についての研究，首都大学東京　理工学研究科 修士論文，2008 ，http://tmubdell.math.metro-u.ac.jp/2007/yamamoto2007.pdf

[3] 吉川　浩：モンゴメリ乗算について，北海道大学大学院 情報科学研究科 ，2008
http://assam.cims.hokudai.ac.jp/~josch/workshop/math/montgomery/montgomery.pdf
[4] 山本和彦：はやわかり RSA，2006，http://pgp.iijlab.net/crypt/rsa.html
[5] 拡張ユークリッドの互除法
http://www2.cc.niigata-u.ac.jp/~takeuchi/tbasic/BackGround/ExEuclid.html
[6] 電子マネー・IC カード講座，　http://www.icnavi.com/
[7] 財団法人　日本情報処理開発協会　電子署名センター，電子署名について
http://www.jipdec.or.jp/esac/intro/index.html
[8] Wikipedia ，ブースの乗算アルゴリズム，
http://ja.wikipedia.org/wiki/%E3%83%96%E3%83%BC%E3%82%B9%E3%81%AE%E4%B9%97%E7%AE%97%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0