卒 業 研 究 報 告

題　　目

VHDL による RSA 暗号器の設計

指 導 教 員

矢野　政顕　教授

報 告 者

松村　暢也

平成 14 年　2 月　　7 日

高知工科大学 電子･光システム工学科

目次

第 1 章　　はじめに

1.1　　情報化社会と暗号

暗号という語から、たいていの人は各種の秘密文書を想像すると思う。また、その性格上、暗号には軍事研究のベールと黒い影がつきまとっていた。しかし、電気通信が普及した昨今の情報化社会では、情報の管理やプライバシーの保護等いろいろな問題が山積しており、それらの問題を解決する方法として日常の普通の通信にも暗号が普及するようになった。

現在の暗号に対する社会の位置づけは、コンピュータとネットワークで構築されるサイバー・スペースでの偽造、改ざんなどの不正を防止するための、情報セキュリティの中核をなす技術とされている。つまり、サイバー・スペースの安全性を高めるために暗号があるのではなく、暗号なしにサイバー・スペースあるいは電子社会は築けないといっても過言ではないのである。

これらのセキュリティに使われる暗号は、1970 年代に入って考えられた公開鍵暗号が主流となっている。その主な理由は、鍵を共有し、なおかつ第三者に秘密にしなければならないこれまでの暗号と違い、公開鍵暗号の概念から、秘密の鍵を共有しなくてもよいところにある。この仕組みについては第 2 章以降で記述する。

1.2　　研究目的

暗号には 2 つの大きな機能がある。1 つは情報の秘匿であり、もう 1 つは認証である。古来からの暗号の主な用途は秘匿であり、その重要性は今でも高い。

一方現在、国際的なスケールで実印・印鑑登録証明システムのサイバー世界版が構築され、その制度が進んでいる。これは、電子署名とその電子認証システムを公開鍵暗号によって行うというものである。これは公開鍵暗号を使うものの一例であるが、この他に、交通や物流の効率化として自動料金収受システム(ＥＴＣ:Electronic Toll Collection System)や、医療関係では電子カルテなどがある。この場合の暗号は認証に使われている。

このように、暗号は我々の生活に深く関っているにもかかわらず学ぶ機会の少ない技術である。本研究は、暗号の歴史的背景と仕組み、また、主として公開鍵暗号の 1 つである RSA 暗号について調査し、この技術をデジタル回路で実現することを目的とする。

本報告の、第 2 章ではこれまでの代表的な暗号の歴史について解説し、第 3 章以降では RSA 暗号にスポットを当て、第 3 章ではその演算方法について、さらに第 4 章では RSA 暗号器の VHDL を用いた設計について記述する。

第 2 章　暗号の歴史

2.1　暗号の分類

暗号を作成しそれを元に戻す操作をそれぞれ、暗号化、復号化という。

暗号化とは、元の文(平文 plain text)をある法則を用いて操作し、第三者には意味不明な文(暗号文 cipher text)を作り出すことであり、復号化とは、暗号文を平文に戻すことである。

暗号はその操作の方法によっていくつかの種類に分類できる(図 2.1 参照)。

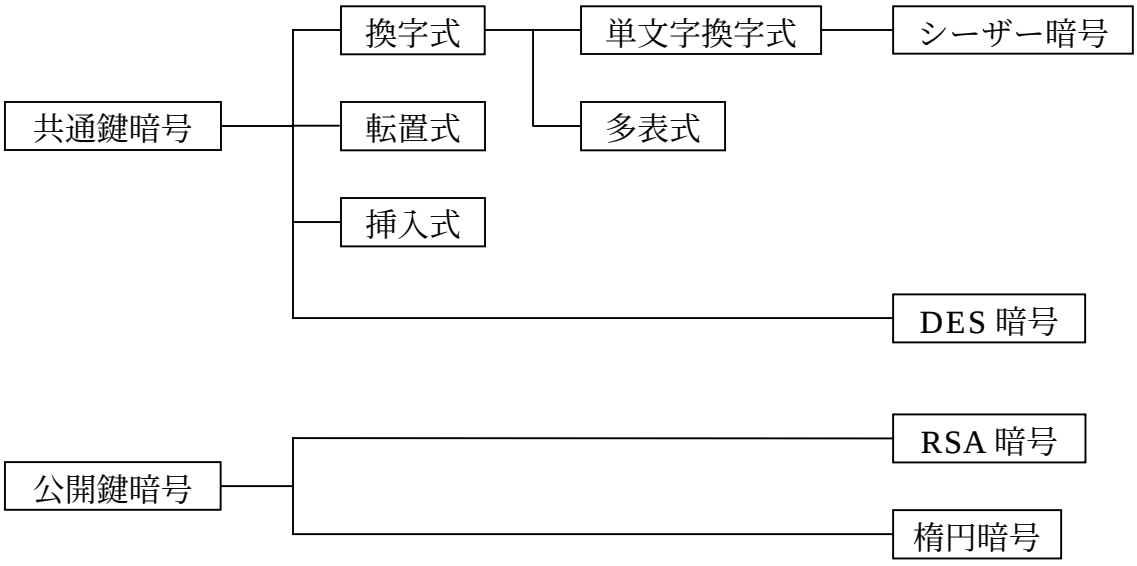

図 2.1　暗号の分類

図 2.1 で示した暗号の名前、方式は、これより後で説明するものに限定してい

る。これは、上記以外にも暗号方式は各種存在するが、一般的な暗号方式と、主として述べる RSA 暗号のみを研究課題として取り上げることを前提としているためである。

この章では暗号の歴史を追うとともに、代表的な暗号化ならびに復号化の方法を説明する。

2.2 共通鍵暗号 ~シーザー暗号から DES 暗号まで~

2.2.1 初期の頃の暗号 ～シーザー暗号～

暗号の歴史は古く、最も古いものでは、紀元前のシーザー暗号が有名である。この暗号は換字式と呼ばれ、平文の文字を、他の文字や、記号、数字等で表す手法である。具体的にシーザー暗号は、アルファベットの文字を‘3’ずらしたもので、例えば、a は D、b は E といった具合である(図 2.2 参照)[1]。このように一文字ずつ順次変換してゆく方式は逐次暗号とも言われる。

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

図 2.2 シーザー暗号の換字表
(上段の小文字が平文、下段の大文字が暗号文)

また、暗号化には換字式以外に、転置式、挿入式がある。この三つが共通鍵暗号の昔からの分類である。

転置式とは、文字の構成単位の順序を変える手法で、文章を逆に書いたり、横書きした文章の縦の列を並べ替えるといった方法がある。

しかし、この手法で作成した暗号は一文字でも誤りや脱落、重複が生じると、復号化した時に文章全体が意味不明なものになる恐れがある。それを防ぐために、一定の間隔に一定の文字(検字)を入れる方式があるが、下手に使用すると解読の手がかりになってしまいかねない。

挿入式とは、文の構成単位(主に単語)間に余分な語を挿入する手法である。しかしこの手法では、解読に少しなれた者ならすぐに解読できるほど強度が弱く、余分な語を挿入するため通信文が長くなってしまう。

以上のような理由から、現在では、転置式ならびに挿入式はほとんど用いられていない。

これらの三つの暗号化方式の中でも換字式は重要な要素を二つ含んでいる。

一つめは、規則(暗号の作成手順)であり、この場合はアルファベットの特定の文字を、特定の数だけ後ろにずらすということである。専門用語でこの規則のことを「アルゴリズム」という。

二つめは、ずらす字数のことである。シーザー暗号では‘3’という数字がそれであり、これが解読の「鍵」といわれるものである。シーザー暗号のように単純な暗号の場合「鍵」は数字である。

しかし、これらの暗号方式では強度が弱い(強度とは、暗号文の解読の難しさを表す指標である)。なぜなら、シーザー暗号であればアルファベットを‘3’ずらしただけなので、25 回の試行錯誤を行うことで平文が求められる。そこで考えられたのが、換字式、転置式、挿入式を組み合わせた暗号である。だが、この方法では暗号化が複雑になり、その結果誤信の可能性が高まってしまい、大事な情報が誤りだらけになってしまいかねない。そこで新たに考えられたのが多表式暗号である。

2.2.2　多表式暗号　～フレデリックの暗号～

多表式暗号は、18 世紀、プロシアのフレデリック大王の頃に使われ始めたといわれている。この暗号方式の大きな特徴は、単文字換字暗号といわれるこれまでの換字式暗号の弱点である、平文に現れる文字の頻度のムラをなくしている点である。頻度のムラとは、単文字換字式暗号では同じ文字を暗号化すると同じ暗号化された文字が出現するため、文字の出現確率、つまり、頻度を調べることで解読の手がかりを与えてしまう。

その例を挙げる。

HOW ARE YOU という平文を前述のシーザー暗号と、多表式暗号で暗号化してみる。なお、多表式暗号では鍵字を使用する(図 2.3 参照)[2]。

図 2.3　単文字換字式暗号(左図)と多表式暗号(右図)

この 2 つの暗号化方式の大きな違いは、鍵の違いである。単文字換字式暗号の場合、鍵は 3 であった。これは、アルファベットを 3 つずらすということである。この場合、平文の 2 つの O は共に R として暗号化される。一方、多表式暗号では、平文の 2 つの O は B と S のように違う文字に暗号化される。なぜこのようになったかというと、多表式暗号では鍵字を使用したからである。暗号化は以下のようにして行われる。

Ⅰ．暗号化するための鍵字を考える(ある程度の長さが必要。ここではENGLANDを使用。)。

Ⅱ．図2.3の右図の平文と同じ文字数だけ鍵字を繰り返して記入し、平文に上下対応する鍵字の数(アルファベットの番号Aを0としてZを25とする)だけ平文の文字をずらす。

このようにして暗号化することで平文の同じ文字は違う暗号文字として、同じ暗号文字は違う平文として表すことができる。そうすると、平文の文字の頻度と暗号文の文字の頻度は一致しないので文字の統計をとっても意味がなくなる。つまり、単文字換字式暗号と比べても強度が高いといえる。

2.2.3　DES暗号　～Date Encryption Standard～

DES 暗号とは、1973 年に暗号の先進国である米国の商務省標準局(NBS：National Bureau of Standards)が、情報化時代の到来に対応して公募したもので、IBM がシャノンの論文を参考にして以前より開発していた、金星暗号に手を加えて応募したものを検討し、修正した暗号方式である。

情報化時代の暗号は、これまでの軍事・外交用の暗号と異なり、不特定多数の間で利用される。例えば、市役所や区役所などが、多数の住民を相手に、一人一人のプライバシーを守るために暗号通信を行う。また、最近のインターネット上でのやりとりでは交信相手はまさに不特定である。このような場合、暗号方式を標準化しておくと、各人が多種類の暗号装置を用意して、相手によって使い分ける必要がなく、暗号装置の製造コストを下げることができる。

次ページに DES 暗号の構成を図 2.4 に示す。

DES 暗号の特徴は、不特定多数の人が使用できるように、そのアルゴリズムを公開している点にある。その為、DES 暗号は手順公開型共通鍵暗号といわれる。

DES は図 2.4 のように 16 段から成る構造を有している。格段は S ボックスと呼ばれる変換表を中心に構成されており、格段ごとに異なる 48 ビットの鍵が加えられる。この鍵は 56 ビットの鍵を元に生成される[3]。

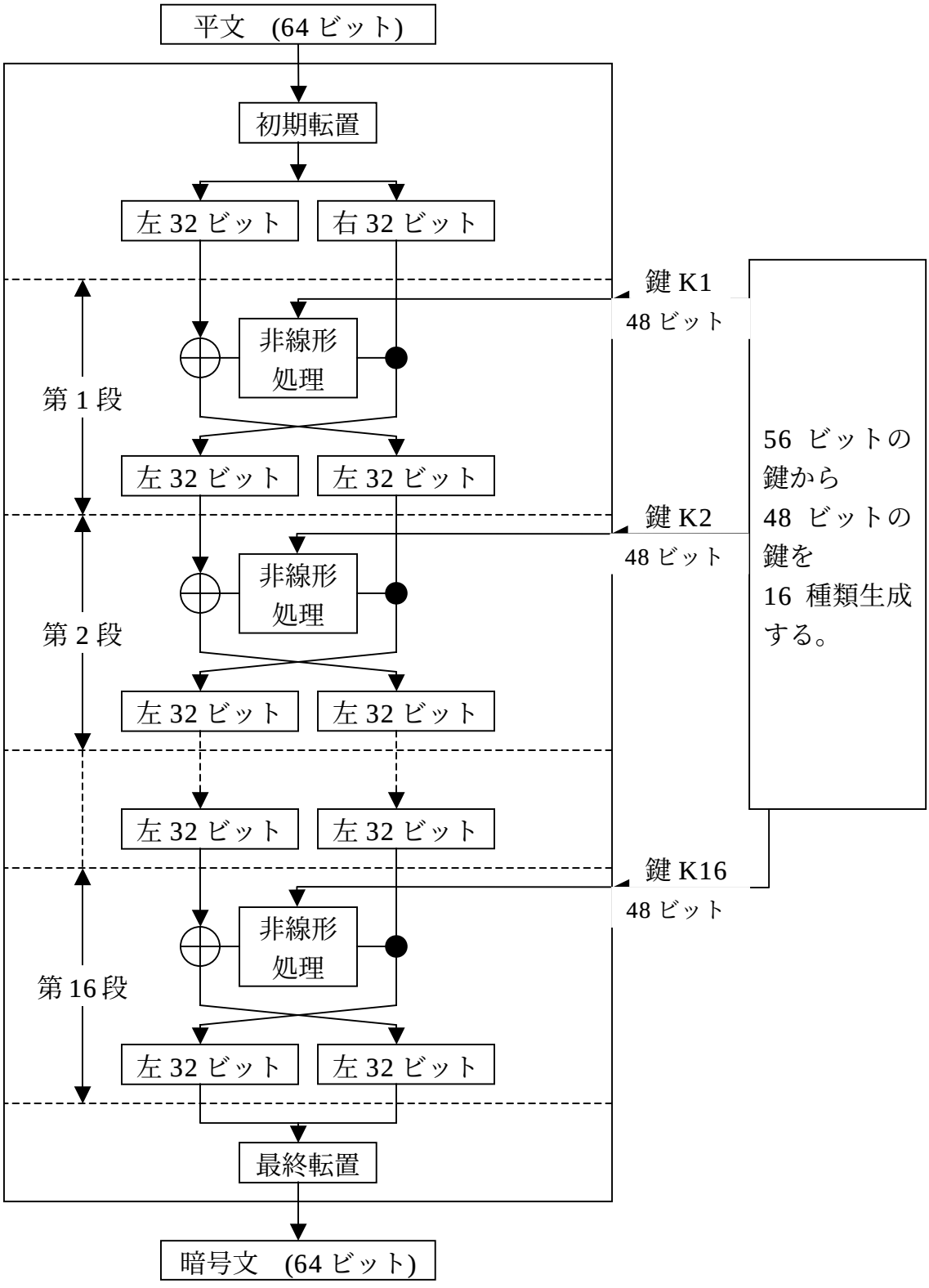

図 2.4　DES 暗号の構成

2.3　公開鍵暗号

前述の共通鍵暗号というのは、ある一定の規則性に従い暗号化し、その逆の操作を行い復号化するというパターンであった。しかし、この方法では、新しく暗号化技術が考えられたとしても簡単に復号化されてしまいやすい。なぜなら、秘密通信を行う両者がその規則性を知っていなければいけないため、暗号化するのに使われる方法というのは、複雑な工程のない、例えば、違う文字や記号に置き換えるというものでしかないからである。

そこで考えられたのが公開鍵暗号である。

公開鍵暗号とは、public key cryptography の直訳で、その名の通り鍵を公開するという、これまでの逆転の発想を生かした暗号化技術である。しかし、鍵を公開するといっても全ての鍵を公開するのではなく、暗号化するための鍵(公開鍵)を公開し、復号化のための鍵(秘密鍵)は隠している。この公開鍵暗号の代表格ともいえるものが、楕円暗号や、RSA 暗号といったものである。

公開鍵暗号の強みは、鍵の数である。これまでの暗号は、使う人数がある程度限られていた。その為、共通鍵暗号であってもその鍵の数はたかが知れている。N 人の人数であれば、送信者と受信者の一対で一つの鍵があればよいので、鍵は N 種あればよい。だが、 N 人相互の連絡を全て必要とすれば、鍵の種類は $N(N-1)/2$ に増える。これは、N が大きくなれば膨大な数になる。例えば、N が 100 万なら、鍵の数は 5000 億個に近い数になる。

これに対して、公開鍵暗号なら、公開鍵と秘密鍵の二種類が必要なので、全体で $2N$ 個となる。つまり、 N が 100 万なら、鍵の数は 200 万ですむ。

このように、現在の加入者相互の通信では公開鍵暗号のほうがコストが下がる。

2.3.1　楕円暗号

この暗号は 1980 年代中頃にミラーやコブリッツによって、公開鍵暗号の安全性を一段と向上させるべく提案されたものである。

楕円暗号とは、楕円曲線と呼ばれる曲線を利用する暗号の総称である。

楕円暗号はRSA暗号のように素因数分解の難しさを利用する公開鍵暗号より格段に安全性が高いとされており、21 世紀初頭から中期にかけて大いに実用に供されるものと思われている[3]。

2.3.2　RSA暗号

RSA 暗号とは、1977 年に MIT(マサチューセッツ工科大学)のリベスト(Rivest)、シャーミル(Shamir)、アドルマン(Adleman)のグループが発表した方式である。前記したようにこの暗号は素因数分解の難しさを利用したもので、素数の性質をうまく活用している。

この暗号については次章から数式を用いて詳しく説明するが、この章では、RSA暗号の仕組みについて簡単に説明する。

RSA 暗号には二種類の鍵が存在していることは前述した。一つは、平文を暗号化する際に用いる公開鍵である。もう一つは、暗号文を復号化するために使用する秘密鍵である。その関係を簡単に図 2.5 に示す。

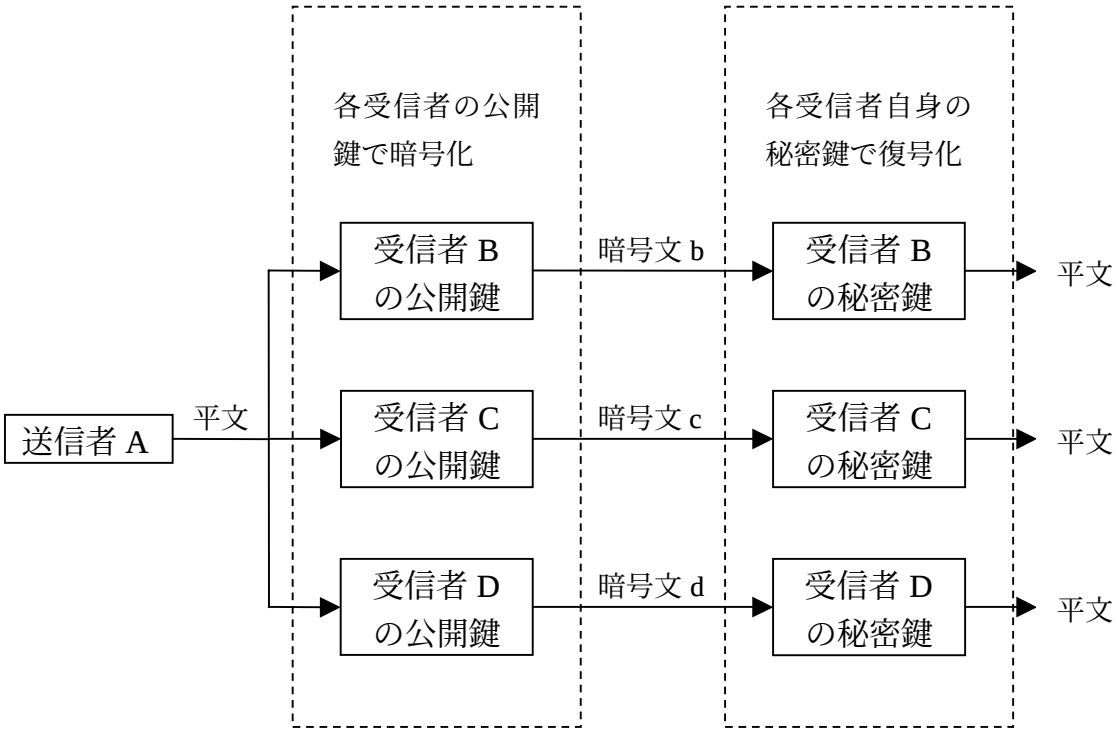

図 2.5　公開鍵暗号の仕組み

この公開鍵はインターネットのホームページ上などに公開しておき、秘密鍵は自分以外に知られないようにしておく必要がある。

次に、どのようにして暗号化、ならびに復号化するかを簡単に説明する。

RSA 暗号は二種類の計算が行われている。一つは、累乗の計算であり、もう一つは、合同である。合同とは mod の演算であり、除算の余りを求める。例えば、時計も mod の演算であるといえる。午後の 5 時と 17 時が同じだというのがそれである。17 を 12 で割ると余りは 5 になる。この場合は、時計代数ともいわれる。

これらの演算を行うことで暗号化し、また、同じ方法で復号化する。つまり、これまでの共通鍵暗号で行っていた、暗号化と逆の操作で復号化するということがないのである。

このように、公開鍵暗号は送信側も受信側も同じ装置で通信を行うことができるので、装置のコストを下げやすい。その為、公開鍵暗号はこれからの通信の分野で、最も使われる可能性の高いものであると考えられる。

第 3 章　RSA 暗号のからくり

3.1　RSA 暗号の数学的概念

前章で RSA 暗号の簡単な仕組みを説明したが、この章ではもう少し深く論じる。具体的には、この暗号に用いられる演算方式である「合同」について説明し、簡単な例を用いて RSA 暗号の流れを示す。

「合同」について学ぶにあたり、H.スターク著の「初等整数論」を用いた[4]。この本で学んだ合同の定義は次の通りである。

(定義)

a,b を整数とし、n を正の整数とする。

$$n \mid (a - b) \tag{3－1}$$

ならば a は b に n を法として合同、あるいは $(\mathrm{mod}\, n)$ で合同といい、

$$a \equiv b \pmod{n} \tag{3－2}$$

と書く。

また、この定義を踏まえて、RSA 暗号では以下の定理が重要になる。

●オイラーの定理

$$\text{ならば} \quad a^{\phi_n} \equiv 1 \pmod{n} \tag{3－3}$$

ここで使われた ϕ_n はオイラー関数と呼ばれ、

$$\phi_n = (p - \quad) \quad - \quad) \qquad (\ p\ ,\ q \quad \text{は異なる素数、} n$$

(3－4)

と表される。

●フェルマーの小定理

P を素数とすれば、すべての a に対して

$$a^{\,P} \equiv a \ (\mathrm{mod} \quad P\ ) \qquad (3－5)$$

$P \mid a$ でなければ、

$$a^{\,(P-1)} \equiv 1 \ (\mathrm{mod} \quad P\ ) \qquad (3－6)$$

となる。

これらの定理を利用して RSA 暗号は成り立っている。これより、実際に数字を使って RSA 暗号の暗号化と復号化の過程を説明する。

暗号化したい情報は文字であれ、映像であれ数字化し、これを暗号化する。

ここで暗号化する情報、つまり平文を 5 とする。まず 5 を 3 乗し、11 で割った余りを求めると、

$$5^3 = 125 = 11 \times 11 + 4 \qquad (3－7)$$

したがって、

$$5^3 \equiv 4 \ (\mathrm{mod} \ \ 11\ ) \qquad (3－8)$$

である。この過程が暗号化であり、4 という数字がこの場合暗号文となる。

次に、受け取った人が元に戻したい(復号したい)場合は、以下の計算を行う。

まず、5 を 10 回掛け合わせると、11 で割った世界では余りが 1 になる(素数 11 を法とする場合、式(3－６)より、$P=11-1=10$ 回掛け合わせると必ず 1 になる)。つまり、

$$5^{10} = 5^{2^5} = 3^5 = 243 \equiv 1 \pmod{11} \tag{3－9}$$

である。

このように、暗号化では平文 5 を 3 乗して暗号文 4 になったので、これを何回かかけると平文に戻すことができる。そこで、暗号文 4 を 7 乗してみる。

$$4^7 = 5^{3^7} = 5^{21} = 5^{10} \times 5^{10} \times 5 \equiv 1 \times 1 \times 5 = 5 \pmod{11} \tag{3－10}$$

すると平文 5 に戻った。平文 5 を 3 乗して得た数を、更に 7 乗するということは、平文を 21 乗したことになる。ここで、5 を 10 乗すると 11 を法とする世界では 1 となるので、20 乗しても 1 となる。したがって 21 乗するということは 1 回掛けること、つまり何もしないこと(平文のまま)と同じであり、復号できたことになる。

このとき、公開鍵暗号では通常、暗号文の作り方を公開するので、上記の例では、11(法とする世界)と 3(平文を掛け合わせる回数)を公開する。そして送られてきた暗号文を 7 乗して平文に戻すのだが、この 7 という数字は受信者だけの秘密にしなければいけない。

ここで、7 という数字を秘密にできるかというと、この規模の数字ならば割り出されてしまう。

式(3－6)より、11 を公開しているので掛け合わせると 1 になる回数 10 がわかる$(P=11-1=10)$。そして、3 を何倍かして 10 で割った余りが 1 になる数が秘密の鍵 K となる。

$$3 \times K = 10 \times \alpha + 1 \tag{3－11}$$

つまり $3K$ の 1 桁目が 1 になるには、$K=7$ であればよい。

このように 7 は 11 と 3 から割り出される。数の世界が大きければ秘密の鍵は簡単には分からない。しかし、ユークリッドの互除法によって秘密にしたい数が計算されてしまう。つまり、上記の暗号方式では受信者だけの秘密が存在せず、暗号としては失敗作である。

ここでオイラーの定理を考える。

まず例として、100 万円を 3 人で分ける場面を想定する。

実数の世界では 1 人当たり 33 万余りになる。しかし、7 を法とする世界では、

$$\frac{1}{3} \tag{3－12}$$

また、11 を法とする世界では、

$$3 \times 4 = 12 \equiv 1 \pmod{11} \qquad \therefore \frac{1}{3} = 4 \tag{3－13}$$

このように法とする世界が違えば同じ 1/3 も 5 や 4 のように違う数字になる。そこで数の世界を隠せば同じ 1/3 でも 5 か 4 か分からなくなる。この数の世界を秘密にした公開鍵暗号が RSA 暗号である。

今、A さんが 2 つの素数 p と q を自分の秘密に決めたとする。そして p と q の積 X を公開鍵とする。

平文の大きさは X より小さければいくつでもよいが、ここでは P とする。

P を何度か掛け合わせると X を法とする世界では 1 となるが、その値は、式(3－4)より、

$$\phi_X = (p-1)(q-1) \qquad (p, q \text{ は異なる素数、} X = pq) \tag{3－14}$$

のような、ϕ_X で求められる(ただし、P と X は互いに素)。

X の他にもう一つ公開鍵が必要だが、ここでは ϕ_X と公約数を持たず、ϕ_X より小さい数として M を使用する。

すると暗号文 C は、

(3－15)

と表せる。この暗号文は X と M を公開しているので、誰でもＡさんだけに届けたい秘密を送ることができる。

次に、Ａさんが復号化するにはどうしたらいいかというと、P をφX乗したら 1 になるということは、2φX乗、3φX乗、4φX乗しても 1 である。つまり、P をφXの倍数(α)回掛け合わせても 1 である。今、φXを法とする世界で M の逆数 r は、

$$M \times r = \phi_X \times \alpha + 1 \equiv 1 \pmod{\phi_X} \quad (\alpha \text{は整数}) \qquad (3－16)$$

より求められる。この r はＡさんの秘密鍵となる。

ここで素因数分解が困難な程度に素数を大きくすれば、Ａさん以外は秘密鍵を X が p と q に素因数分解できることを知らなければ、オイラーの定理を用いてφXという数字を得られないので、公開鍵から割り出せない。これが RSA 暗号の概念である。
(ここまで、アルファベットの小文字は秘密鍵に関するもの、大文字は公開鍵に関するものとして表している。)

(参考)　M の逆数 r の求め方

r はφXを法とする世界では M と掛け合わせたものが 1 になるので、ユークリッド互除法の応用で求められる。その方法をこれより示す。

正の整数 m,n の最大公約数を d とすると、適当な整数(負も含む) u,v をとって、

$$m \times u + n \times v = d \qquad (3－17)$$

とできる。特に、m,n が互いに素(最大公約数が 1)であれば、

$$m \times u + n \times v = 1 \qquad (3－18)$$

とできる。このとき、上式は m を法として n の逆数があることが分かる。つまり、

$$n \times v = 1 \pmod{m} \qquad (3-19)$$

となり、

$$\diagup n$$

となる。ここで、実際に上の文字を使って式を作ってみると r は、

$$\phi_X \times u + M \times v = 1 \qquad (3-21)$$

$$M \times v \equiv 1 \pmod{\phi_X} \qquad (3-22)$$

$$v = 1/M = r \qquad (3-23)$$

となる。

次に、u, v を求める。その算法は次のようになる。

$m > n$、$m = n_0, n = n_1$ とする。n_{i-1} を n_i で割り、商を q_i、余りを n_{i+1} とすると、

$$n_{i-1} = n_i \times q_i + n_{i+1} \quad (i = 1, 2, \cdots) \qquad (3-24)$$

となる。

$n_{l+1} = 0$ となり、n_l が最大公約数 d であると仮定する。

ここで、$u_0 = 0, v_0 = 1, u_1 = 1, v_1 = -q_1$ とおき、以下順次漸化式、

$$u_{i+1} = -q_{i+1} \times u_i + u_{i-1}$$

$$v_{i+1} = -q_{i+1} \times v_i + v_{i-1} \quad (i = 1,2,\cdots,l-2) \qquad (3-26)$$

でu_i, v_iを求める。これを反復し、$u = u_{l-1}, v = v_{l-1}$を求めれば、

$$u_{l-1} \times n_0 + v_{l-1} \times n_1 = n_l \qquad (3-27)$$

すなわち、式(3－17)の形になる。この場合、一般に(iに関する帰納法で)

$$u_{i-1} \times m + v_{i-1} \times n = n_i \qquad (3-28)$$

となっていることが示される。そこで、$m = n_0 = \phi_X, n = n_1 = M$で計算すると、$M$の逆数$r$は$v$の値として算出される。

3.2　論理回路実践に基づく数式の見直し

3.1 節で述べたような数学的手法で RSA 暗号は作成、復号されるが、この過程を論理回路で実現するには式(3－15)のP^Mが問題になってくる。つまり、P^Mを計算してからmod Xをとると、P^Mの値が大きくなるため非効率だということである。

そこでmod演算を少し工夫してみる。

mod演算では次のような性質が成り立つ。

$$(x \times y)(\mathrm{mod}\ p) = (x\ (\mathrm{mod}\ p) \times y\ (\mathrm{mod}\ p))(\mathrm{mod}\ p) \qquad (3-29)$$

この性質を利用すると、

$$C \equiv P^{\phi_X} \pmod{M}$$
$$= \underbrace{(\cdots(P \pmod{M}) \times (P \pmod{M}) \pmod{M} \pmod{M} \times \cdots) \pmod{M}}_{\text{φX回の乗算}}$$

(3－30)

となる。このようにすることで計算の途中結果がP^2を超えることはない。

しかし、この方法ではφX回の乗算を繰り返さなければならず、φXの値も非常に大きくなるため効率が悪い。そこでもう一工夫する必要がある。

φXの値を次のように表現する。

$$\phi_X = (a_k a_{k-1} \cdots a_1 a_0)_2 \qquad (3－31)$$

すなわち、

$$\phi_X = a_k \times 2^k + a_{k-1} \times 2^{k-1} + \cdots + a_1 \times 2^1 + a_0 \times 2^0 = \sum_{i=0}^{k} a \qquad (3－32)$$

である。この関係を用いると、

$$P^{X} = P^{(a_k a_{k-1} \cdots \quad 2}$$
$$= P^{\left(\sum_{i=0}^{k} a_i 2^i\right)}$$
$$= P^{a_k 2^k} \times P^{a_{k-1} 2^{k-1}} \times \cdots \times P^{a_1 2^1} \times P^{a_0 2^0} \qquad (3－33)$$

と表わせる。式(3－30)と式(3－33)から、

$$P^{\phi_X} \pmod M$$
$$= \left(\left(P^{a_k 2^k} \pmod M\right) \times \cdots \times \left(P^{a_1 2^1} \pmod M\right) \times \left(P^{a_0 2^0} \pmod M\right)\right) \pmod M \quad (3-34)$$

と表わせる。さらに、$P^{a_i 2^i} \pmod M \quad (i = 0,1,\cdots,k)$の計算は、

$$P^{a_i 2^i} \pmod M = \underbrace{\left(\cdots\left(\left(P^2 \pmod M\right)^2 \pmod M\right)^2 \cdots\right)^2}_{i\text{回 2 乗する}} \pmod M \quad (3-35)$$

のように行える。

いま、乗算のみに着目すれば、式(3－30)は、最悪の場合、$2^{k+1} = \phi_X$ 回の乗算を行う必要がある。これに対して、式(3－34)と式(3－35)を用いた場合、たかだか k^2 回の乗算を行えばよい。この乗算回数を比較すると、k の値が増えるにしたがって後者のほうが計算回数が少なくなる。

3.3　RSA 暗号の暗号化アルゴリズム

RSA 暗号の回路を考える前に、mod 演算と、暗号化のアルゴリズムを説明する[5]。

●mod 演算のアルゴリズム

mod 演算とは前述したとおり、筆算による 2 進数の除算で得た余りを求めることで、これは被除数の桁数回の大小比較と減算で求まる。

被除数の 2 進数表現を、

$$DD = \left(D_{DL-1}, D_{DL-2}, \cdots, D_1, D_0\right)$$

除数の 2 進数表現を、

$$DS = \left(S_{SL-1}, S_{SL-2}, \cdots, S_1, S_0\right)$$

とし、$DL \geq SL$ とする。

k ビットの配列

$$A = \left(A_{k-1}, A_{k-2}, \cdots, A_1, A_0\right)$$

の、A_i から A_j $\left(k \geq i \geq j \geq 0\right)$ を $A(i:j)$ と表す。

このとき、$DD \left(\mathrm{mod}\, DS\right)$ は以下のアルゴリズムで求まる。

Step1. 長さ $DL + SL - 1$ の配列 D に右詰め(LSB 側)で被除数 DD を格納し、左側(MSB 側)の SL ビットは全て 0 にする。

Step2. 長さ $SL + 1$ の配列 S に右詰めで除数 DS を格納し、MSB を 0 とする。

Step3. $i = DL-1, DL-2, \cdots, 1, 0$ について、$D(SL+i:i) \geq S$ なら、

$$+$$

とする。

例として、$DD = (1001100)_2$、$DS = (111)_2$ の演算過程を図 3.1 に示す。

● 暗号化アルゴリズム

RSA 暗号の暗号化には 3.2 節で述べた、式(3－34)と式(3－35)を使用する。

いま、基底、指数、法の２進数表現をそれぞれBS, EX, Mと表す。また、BS, EXの長さをそれぞれ、BL, EL(ビット)とする。さらに、EXの各ビットを(E_{EL-1}, …, E_1, E_0)と表す。

このとき、$BS^{EX} \pmod{M}$の値は以下のアルゴリズムで求まる(図 3.2 参照)。

Step1. 長さ$2BL$の配列Cを用意し、E_0=1 なら、配列Cに右詰め(LSB 側)でBSを格納する。E_0=0 なら、配列Cを 1 とする。

Step2. 長さ$2BL$の配列Bに右詰めでBSを格納し、残りを 0 にする。

Step3. $i = 1,2,\cdots,EL-1$について、以下の計算を行う。

① $B = (B \times B) \pmod{M}$

とする。

② $E_i = 1$なら、

$C = (C \times B) \pmod{M}$

とする。

上記のアルゴリズムで配列Cに残った値が、求める値$BS^{EX} \pmod{M}$である。

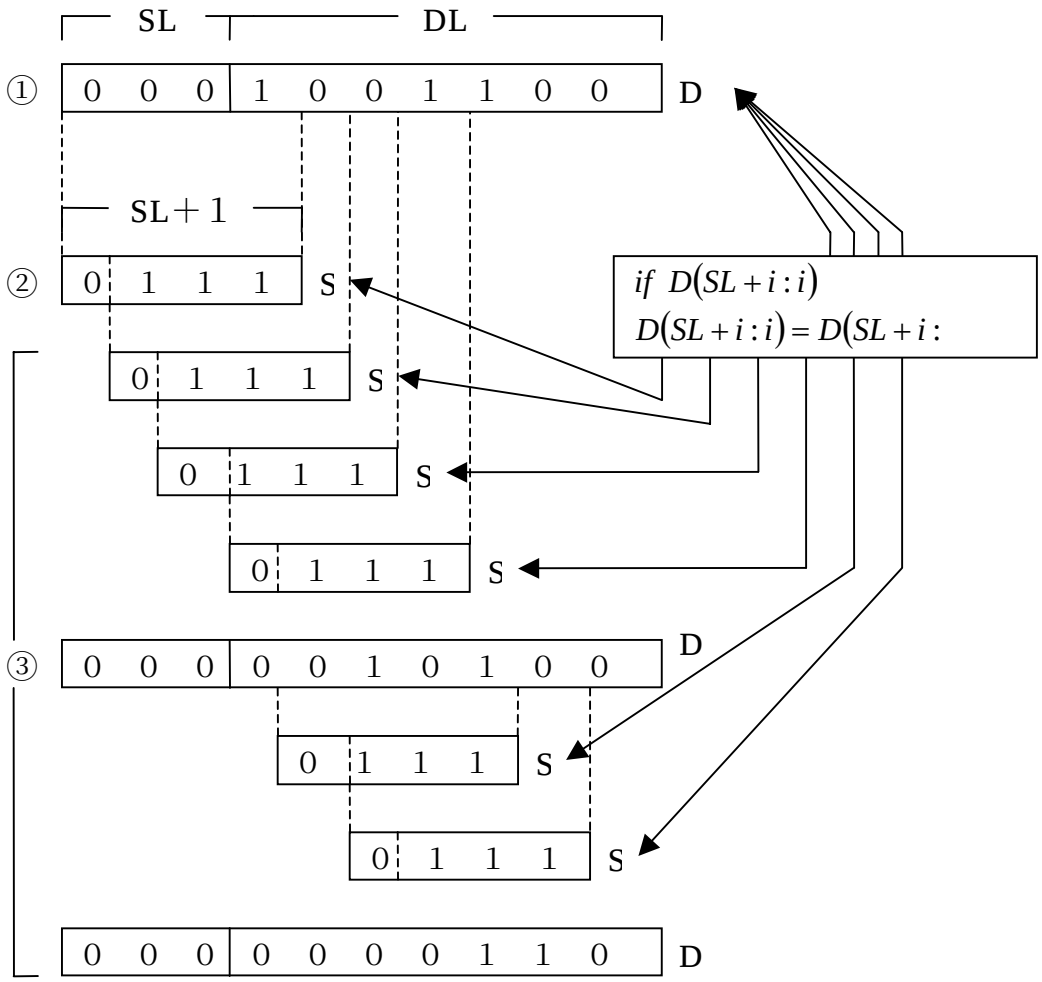

図 3.1　mod 演算のアルゴリズム

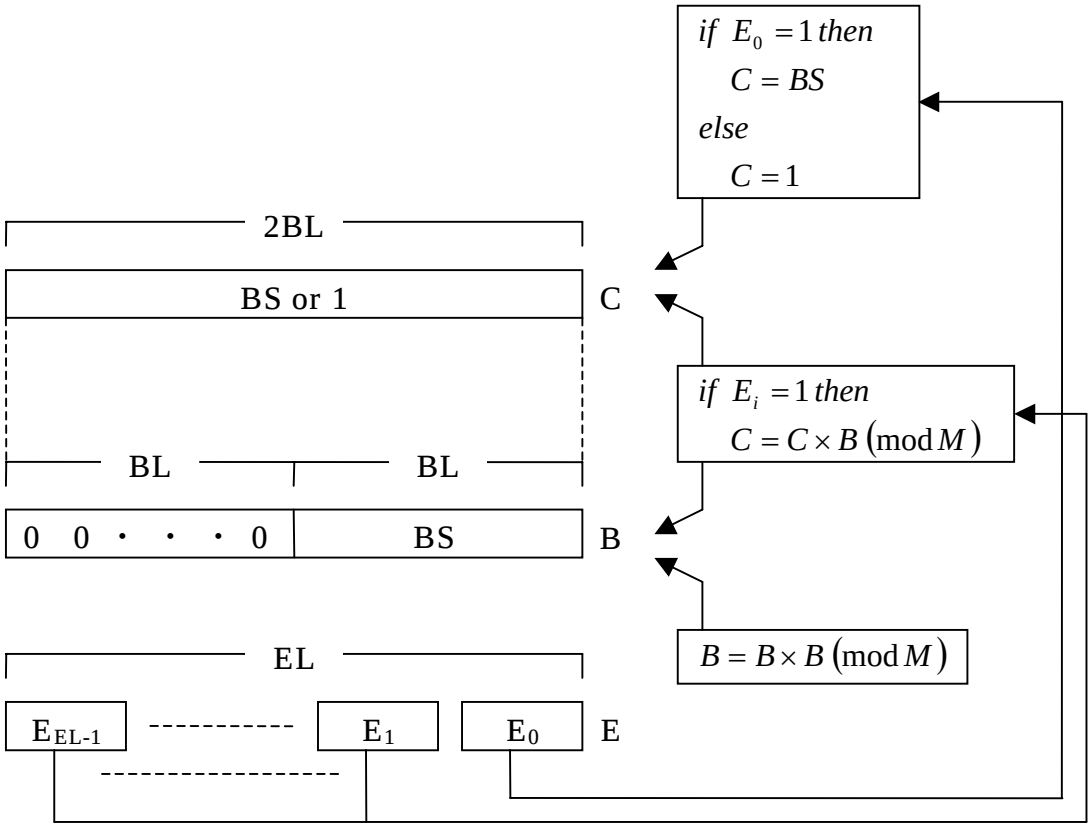

図 3.2　RSA 暗号の暗号化アルゴリズム

第 4 章　VHDL による RSA 暗号器の設計

4.1　mod 演算器の組み合わせ回路としての設計

まず、 mod 演算器を組み合わせ回路で実現するにはどの程度の規模がふさわしいかというと、安全性を考慮した場合は 1024 ビットから 2048 ビット程度のものが必要になる。そのため、RSA 暗号器を組み合わせ回路として設計すると mod 演算器や乗算器のビット数が大きいため数百万ゲートを超える規模となり、コストの観点から一般的に非現実的なためあまり考えられない。そのためここでは、 mod 演算器のみを組み合わせ回路で設計例として示す。

4.1.1　mod演算器のブロック図

3.3 節で示した RSA 暗号の暗号化アルゴリズムから、 mod 演算器には、平文 1 文字分の文字コードを 2 乗した値と法が入力されることがわかる。今回の RSA 暗号のエンティティに関する仕様を表 4.1 に示す。

表 4.1　RSA 暗号のエンティティ仕様

平文文字コードのビット長	*CL* = 7 ビット
ブロックサイズ	*BS* = 1 文字
1 ブロックあたりのビット長	*BL* = 7 ビット
公開鍵のビット長	*KL* = 7 ビット
法のビット長	*ML* = 7 ビット

ここで、なぜ 7 ビットにしたかというと、RSA 暗号器は(3－15)式で示した演算で暗号化も復号化も行うので、平文に現れる文字コードのサイズと暗号文に現れるコードのサイズ同じにする必要がある。ところが、暗号文に現れるコー

ドのサイズは、法のサイズと同じになるため、平分の文字コードを 7 ビットにした。

mod 演算器の組み合わせ回路としてのブロック図を図 4.1 に示す。

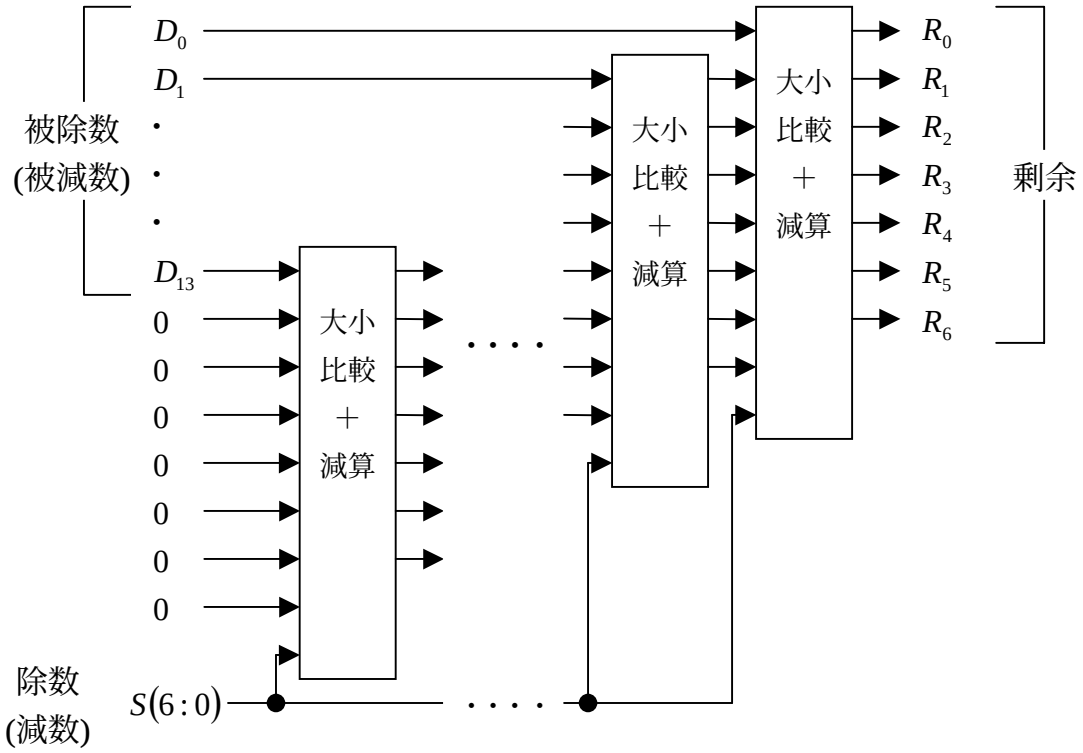

図 4.1　mod 演算器の組み合わせ回路としてのブロック図

図 4.1 の大小比較＋減数のところは、以下のような動作を行う。

● if 被減数≧減数 then

出力＝被減数－減数
(下位 7 ビット)

else

出力＝被減数
(下位 7 ビット)

これは、被減数が減数以上ならば、出力は被減数から減数を引いた値の下位 7 ビットとなり、減数のほうが大きければ、出力は被減数の下位 7 ビットになることを意味している。

4.1.2 VHDL による mod演算器の設計

図 4.1 のブロック図を VHDL を用いて設計する(図 4.2 参照)。また、そのテストベンチを図 4.3 に示す。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_ARITH.all;
use ieee.std_logic_unsigned.all;

entity MODULO is
    port( D : in    std_logic_vector(13 downto 0);
          S : in    std_logic_vector(6    downto 0);
          R : out std_logic_vector(6    downto 0)
          );
end MODULO;

architecture stimulus of MODULO is

begin
    process (D, S)
    variable TMP_D : std_logic_vector(D'length+S'length-1 downto 0)
    variable TMP_S : std_logic_vector(S'length downto 0);
    constant ZV_MD : std_logic_vector(S'range) := (others => '0');
    begin
        TMP_D := ZV_MD & D;
        TMP_S := '0' & S;
        for I in D'range loop;
```

図 4.2　mod 演算器の組み合わせ回路としての VHDL

```
            if (TMP_D(S'length+I downto I) >= TMP_S) then
               TMP_D(S'length+I downto I) := TMP_D(S'length+I downto I) - TMP_S;
            else
               null;
            end if;
         end loop;
         R <= TMP_D(S'range);
   end process;
end stimulus;
```

図 4.2　mod 演算器の組み合わせ回路としての VHDL2/2

この VHDL では、

$$D \quad R \ (\mathrm{mod} \quad S) \qquad (4-1)$$

の演算を行っており、入力信号 D,S を代入すると(4－1)式の演算が行われ、出力信号 R が求められる。

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_ARITH.all;
use ieee.std_logic_unsigned.all;

entity testbench is
end testbench;

architecture stimulus of testbench is
```

図 4.3　mod 演算器のテストベンチ 1/3

```
component MODULO is
    port(
          D : in    std_logic_vector(13 downto 0);
          S : in    std_logic_vector(6    downto 0);
          R : out std_logic_vector(6    downto 0)
          );
end component;

          signal in_D    : std_logic_vector(13 downto 0);
          signal in_S    : std_logic_vector(6    downto 0);
          signal out_R : std_logic_vector(6    downto 0);

begin
    uut : MODULO port map(
                                  D => in_D,
                                  S => in_S,
                                  R => out_R
                                  );

    stimulus1:process
        begin
            in_D <= "00000000000000" ;    wait for 30 ns;
            in_D <= "10010000000111" ;    wait for 30 ns;
            in_D <= "01010101010101" ;    wait for 30 ns;
            in_D <= "10101010101010" ;    wait for 30 ns;
            in_D <= "11111100000000" ;     wait for 30 ns;
            in_D <= "00000011111111" ;     wait for 30 ns;
            in_D <= "00110011001100" ;     wait for 30 ns;
            in_D <= "11001100110011" ;     wait for 30 ns;
            in_D <= "11111111111111" ;      wait for 30 ns;
    end process stimulus1;
```

図 4.3　mod 演算器のテストベンチ 2/3

```
            in



   end p     ess stimulus2;

nd stimulus;
```

4.3 ンチ 3/3

4.1.3 シミュレーション結果

図 4.2 の VHDL と、図 4.3 のテストベンチによるシミュレーション結果を図 4.4 として最後のページに示す。

入力信号を見てみると、S の値が 16 進数で $0F$ のとき、D は 16 進数で $2407, 1555, 2AAA, 3F00, 00FF$ の順に変化している。

すると出力信号 R は、$0D, 01, 02, 03, 00$ の順に変化し、(4－1)式の演算が正常に行われていることがわかる。

その後、S の値が 16 進数で 70 のとき、D は 16 進数で $0CCC, 3333, 3FFF$ の順に変化し、出力信号 R は、$1C, 03, 1F$ の順に出力されるので、この場合も(4－1)式の演算が行われていることがわかる。

4.2 RSA 暗号器の順序回路としての設計

4.2.1 乗算＋mod演算器のブロック図と状態遷移図

次に、RSA 暗号器を順序回路として設計する。その手始めに暗号器の核とな

る乗算＋mod 演算器の回路を設計する。図 4.5 にこのブロック図を示す。

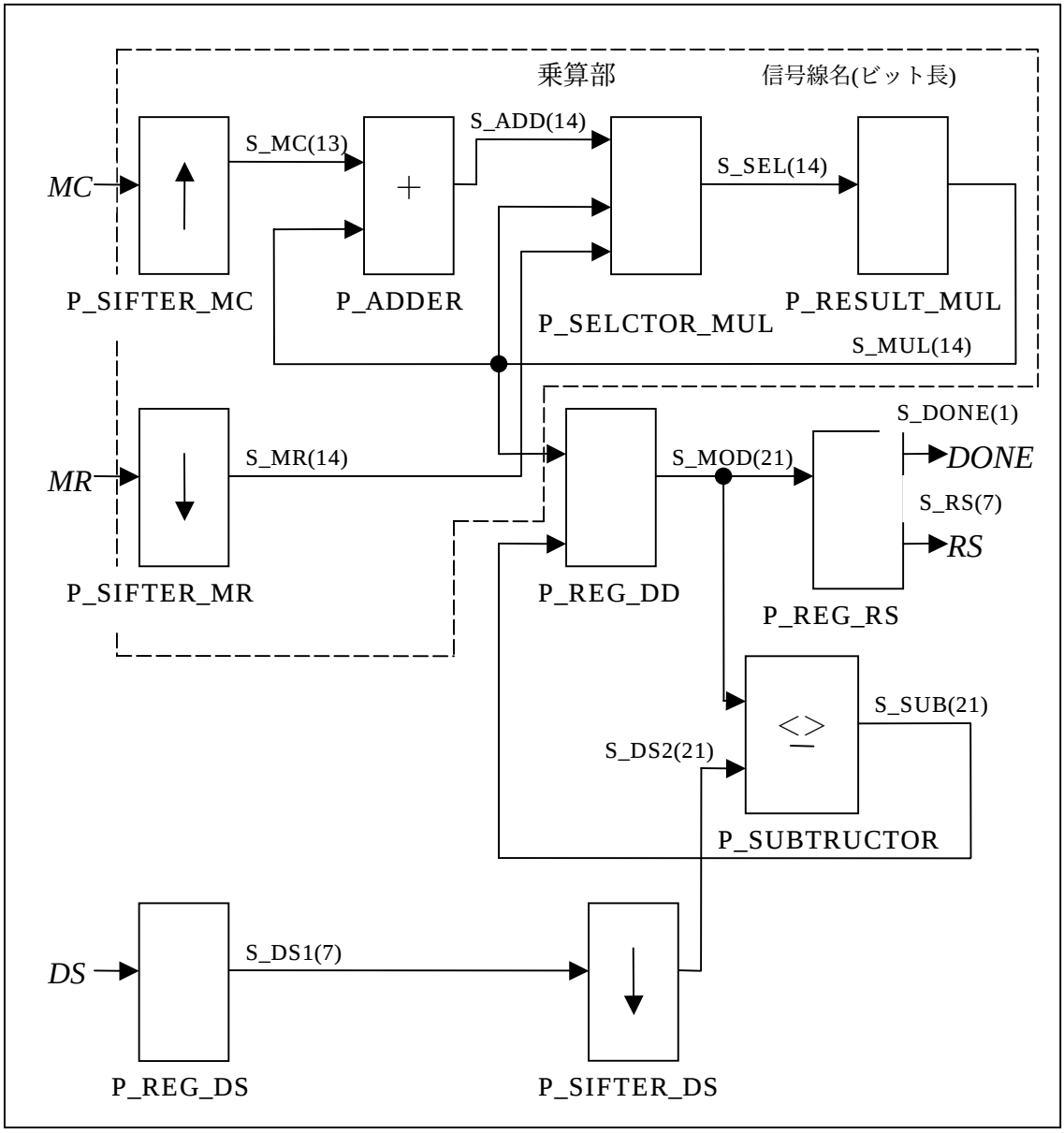

図 4.5　乗算＋mod 演算器の順序回路としてのブロック図

図 4.5 の各ブロックの機能を説明する。

・P_SIFTER_MC…入力信号 *MC* を格納、左シフトを行うシフトレジスタ。
・P_SIFTER_MR…入力信号 *MR* を格納、右シフトを行うシフトレジスタ。
・P_SIFTER_DS…信号 *S _ DS* を格納、右シフトを行うシフトレジスタ。
・P_REG_DS…入力信号 *DS* を格納するレジスタ。
・P_REG_DD…信号 *S _ MUL, S _ SUB* を格納するレジスタ。
・P_REG_RS…信号 *S _ MOD* を格納するレジスタ。
・P_RESULT_MUL…信号 *S _ SEL* を格納するレジスタ。
・P_SELCTOR_MUL…信号 *S _ MR* の値によって信号 *S _ ADD, S _ MUL* を選択するセレクタ。
・ P_SUBTRUCTOR…信号 *S _ MOD, S _ DS*2 の大小比較と減算を行う演算器。
・ P_ADDER…信号 *S _ MC, S _ MUL* を加算する加算器。

次に、図 4.5 の制御回路の状態遷移図を図 4.6 に示す。

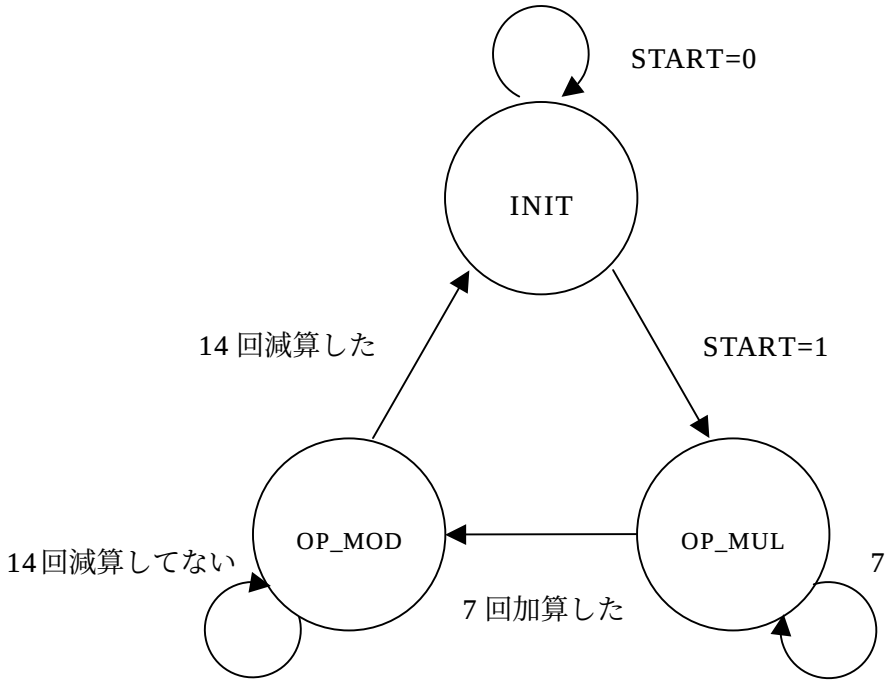

図 4.6　乗算＋mod 演算器の制御回路の状態遷移図

この状態遷移図は、演算の回数によって 3 つの状態に移り変わることを示している。

4.2.2　VHDL による乗算＋mod演算器の設計

図 4.5、4.6 のブロック図ならびに状態遷移図に従い、VHDL で回路を設計する。その VHDL 記述を図 4.7 に、そのテストベンチを図 4.8 に示す。

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity MUL_MOD is
    port (
        CLK, RESET, START : in    std_logic;
        MC                : in    std_logic_vector(6 downto 0);
        MR                : in    std_logic_vector(6 downto 0);
        DS                : in    std_logic_vector(6 downto 0);
        DONE              : out std_logic;
        RS                : out std_logic_vector(6 downto 0)
         );
end MUL_MOD;

architecture stimulus of MUL_MOD is

type STATE is (INIT, OP_MUL, OP_MOD);
signal CRST, NTST : STATE;
signal SET_MUL, SET_MOD      : std_logic;
signal MM_DONE, S_DONE       : std_logic;
signal S_MC                  : std_logic_vector(12 downto 0);
signal S_MR                  : std_logic_vector(6 downto 0);
signal S_DS1, S_RS           : std_logic_vector(6 downto 0);
signal S_DS2, S_MOD, S_SUB   : std_logic_vector(20 downto 0);
signal S_ADD, S_SEL, S_MUL   : std_logic_vector(13 downto 0);
```

図 4.7　乗算＋mod 演算器の順序回路としての VHDL

```
signal C_MUL                    : integer range 0 to 8;
signal C_MOD                    : integer range 0 to 15;
constant ZV_MC                  : std_logic_vector(5 downto 0) :=(others => '0');
constant ZV_DD                  : std_logic_vector(6 downto 0) :=(others => '0');
constant ZV_DS                  : std_logic_vector(12 downto 0) :=(others => '0');

begin

    P_CONTROL_REG: process (CLK, RESET)
    begin
        if (RESET = '1') then
            CRST <= INIT;
        elsif (CLK'event and CLK = '1') then
            CRST <= NTST;
        end if;
    end process;

    P_CONTROL_STF: process (CRST, C_MUL, C_MOD, START)
    begin
        case CRST is
            when INIT    => MM_DONE <= '0';
                                        SET_MOD <= '0';
                            if (START = '1') then
                                SET_MUL <= '1';
                                NTST <= OP_MUL;
                            else
                                SET_MUL <= '0';
                                NTST <= INIT;
                            end if;
            when OP_MUL => SET_MUL <= '0';
                            if (C_MUL = 7) then
                                SET_MOD <= '1';
                                NTST <= OP_MOD;
```

図 4.7　乗算＋mod 演算器の順序回路としての VHDL2/7

```
                        else
                            NTST <= OP_MUL;
                        end if;
            when OP_MOD => SET_MOD <= '0';
                        if (C_MOD = 14) then
                            MM_DONE <= '1';
                            NTST <= INIT;
                        else
                            NTST <= OP_MOD;
                        end if;
        end case;
    end process;

    P_CONTROL_CNT: process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (CRST = INIT) then
                C_MUL <= 0;
                C_MOD <= 0;
            elsif (CRST = OP_MUL) then
                C_MUL <= C_MUL + 1;
                C_MOD <= 0;
            elsif (CRST = OP_MOD) then
                C_MUL <= 0;
                C_MOD <= C_MOD + 1;
            end if;
        end if;
    end process;

    P_SIFTER_MC: process (CLK, RESET)
    begin
        if (RESET = '1') then
            S_MC <= (others => '0');
```

図 4.7　乗算＋mod 演算器の順序回路としての VHDL3/7

```
        elsif (CLK'event and CLK = '1') then
            if (SET_MUL = '1') then
                S_MC <= ZV_MC & MC;
            else
                S_MC <= S_MC(11 downto 0) & '0';
            end if;
        end if;
    end process;

    P_SIFTER_MR: process (CLK, RESET)
    begin
        if (RESET = '1') then
            S_MR <= (others => '0');
        elsif (CLK'event and CLK = '1') then
            if (SET_MUL = '1') then
                S_MR <= MR;
            else
                S_MR <= '0' & S_MR(6 downto 1);
            end if;
        end if;
    end process;

    P_REG_DS: process (CLK, RESET)
    begin
        if (RESET = '1') then
            S_DS1 <= (others => '0');
        elsif (CLK'event and CLK = '1') then
            if (SET_MUL = '1') then
                S_DS1 <= DS;
            else
                S_DS1 <= S_DS1;
            end if;
        end if;
```

図 4.7　乗算＋mod 演算器の順序回路としての VHDL4/7

```
end process;

P_ADDER: S_ADD <= S_MUL + ('0' & S_MC);

P_SELECTOR_MUL: process (S_MR, S_ADD, S_MUL)
begin
    if (S_MR(0) = '1') then
        S_SEL <= S_ADD;
    else
        S_SEL <= S_MUL;
    end if;
end process;

P_RESULT_MUL: process (CLK, RESET)
begin
    if (RESET = '1') then
        S_MUL <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        if (SET_MUL = '1') then
            S_MUL <= (others => '0');
        else
            S_MUL <= S_SEL;
        end if;
    end if;
end process;

P_REG_DD: process (CLK, RESET)
begin
    if (RESET = '1') then
        S_MOD <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        if (SET_MOD = '1') then;
            S_MOD <= ZV_DD & S_MUL;
```

図 4.7　乗算＋mod 演算器の順序回路としての VHDL5/7

```
            else
                S_MOD <= S_SUB;
            end if;
        end if;
    end process;

    P_SIFTER_DS: process (CLK, RESET)
    begin
        if (RESET = '1') then
            S_DS2 <= (others => '0');
        elsif (CLK'event and CLK = '1') then
            if (SET_MOD = '1') then
                S_DS2 <= '0' & S_DS1 & ZV_DS;
            else
                S_DS2 <= '0' & S_DS2(20 downto 1);
            end if;
        end if;
    end process;

    P_SUBTRUCTOR: process (S_DS2, S_MOD)
    begin
        if (S_MOD >= S_DS2) then
            S_SUB <= S_MOD - S_DS2;
        else
            S_SUB <= S_MOD;
        end if;
    end process;

    P_REG_RS: process (CLK, RESET)
    begin
        if (RESET = '1') then
            S_DONE <= '0';
            S_RS <= (others => '0');
```

図 4.7　乗算＋mod 演算器の順序回路としての VHDL6/7

```
                'event and CLK = '1') then

                   ONE <= '1';
              S_RS <= S_MOD(6 downto 0);
               SET_MUL = '1') then
              S_DONE <= '0';

        else
                              _DONE;




                NE;
```

路としての VHD

この VHDL では、

$$\equiv RS \pmod{DS} \qquad (4-2)$$

の演算を行っている。入力信号 *MC*, *MR*, *DS* の値にしたがい(4－2)式を満たす *RS* が出力信号として表われる。

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity testbench is
end testbench;

architecture stimulus of testbench is

component MUL_MOD
    port(
        CLK, RESET, START : in    std_logic;
        MC                : in    std_logic_vector(6 downto 0);
        MR                : in    std_logic_vector(6 downto 0);
        DS                : in    std_logic_vector(6 downto 0);
        DONE              : out std_logic;
        RS                : out std_logic_vector(6 downto 0)
        );
end component;

constant PERIOD: time := 10 ns;

signal CLK          : std_logic;
signal RESET        : std_logic;
signal START        : std_logic;
signal in_MC        : std_logic_vector(6 downto 0);
signal in_MR        : std_logic_vector(6 downto 0);
signal in_DS        : std_logic_vector(6 downto 0);
signal out_DONE : std_logic;
signal out_RS       : std_logic_vector(6 downto 0);

begin
    uut : MUL_MOD port map(
```

図 4.8　乗算＋mod 演算器のテストベンチ 1/3

```
                                CLK => CLK,
                                RESET => RESET,
                                START => START,
                                MC => in_MC,
                                MR => in_MR,
                                DS => in_DS,
                                DONE => out_DONE,
                                RS => out_RS
                                );

stimulus1:process
variable clktmp: std_ulogic := '0';
    begin
        loop wait for PERIOD/2;
        clktmp := not clktmp;
        CLK <= clktmp;
        end loop;
    wait;
end process stimulus1;

stimulus2:process
    begin
        RESET <= '1' ; wait for 10 ns;
        RESET <= '0' ; wait;
end process stimulus2;

stimulus3:process
    begin
        START <= '0' ; wait for 40 ns;
        START <= '1' ; wait for 20 ns;
        START <= '0' ; wait;
end process stimulus3;
```

図 4.8　乗算＋mod 演算器のテストベンチ 2/3

```
    stimulus4:process
        begin
            in_MC <= "1101011" ; wait;
    end process stimulus4;

    stimulus5:process
        begin
            in_MR <= "1100000" ; wait;
    end process stimulus5;

    stimulus6:process
        begin
            in_DS <= "1000111" ; wait;
    end process stimulus6;

end stimulus;
```

図 4.8　乗算＋mod 演算器のテストベンチ 3/3

4.2.3　シミュレーション結果

図 4.7 の VHDL と、図 4.8 のテストベンチによるシミュレーション結果を図 4.9 とし最後のページに示す。

入力信号 *MC, MR, DS* にそれぞれ 2 進数で1101011 , 1100000 , 1000111 が入力され、START の信号が '1' かつ、CLK の立ち上がりで演算を開始し、図 4.5 の状態遷移図に示した合計 21 回の演算とレジスタの格納による遅れで、24CLK 目で出力 *RS* に 2 進数で 0110000　の値が表われている。

この値は(4－2)式を満たしている。

4.2.4　RSA 暗号器のブロック図と状態遷移図

4.2.2 節で設計した乗算＋mod 演算器を用いて RSA 暗号器の設計を行う。RSA 暗号器のブロック図を図 4.10 に示す。

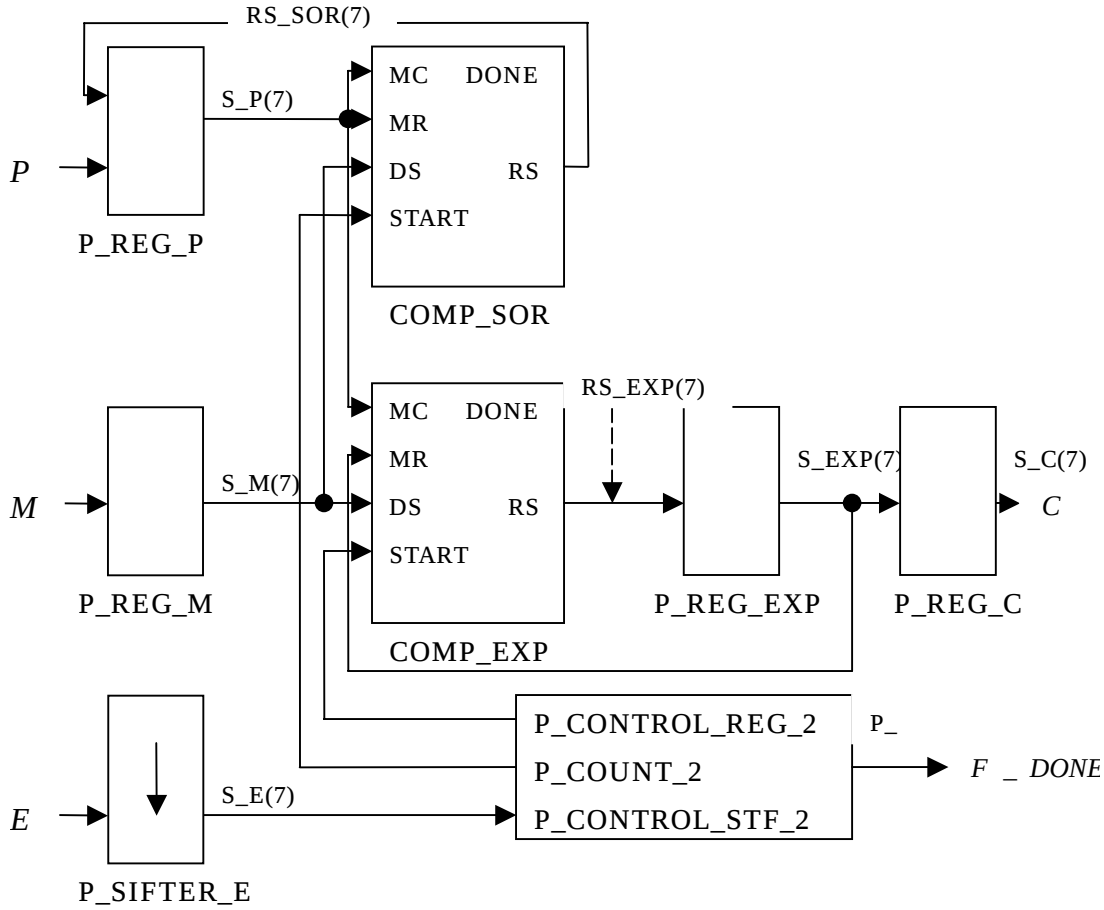

図 4.10　RSA 暗号器のブロック図

図 4.10 の各ブロックの機能について説明する。

・P_REG_P…入力信号 *P*、COMP_SOR の出力 *RS _ SOR* を格納するレジスタ。
・P_REG_M…入力信号 *M* を格納するレジスタ。
・P_REG_EXP…COMP_EXP の出力 *RS _ EXP* を格納するレジスタ。

・P_REG_C…信号S_EXPを格納するレジスタ。
・P_SIFTER_E…入力信号Eを格納、右シフトするシフトレジスタ。
・COMP_SOR、COMP_EXP…図 4.5 のブロック図と同じもの。
・P_CONTROL_REG_2 …状態を格納するレジスタ。
・P_COUNT_2…状態回数のカウントを行うカウンタ
・P_CONTROL_STF_2…状態遷移の制御回路。

次に、RSA 暗号器の状態遷移図を図 4.11 に示す。

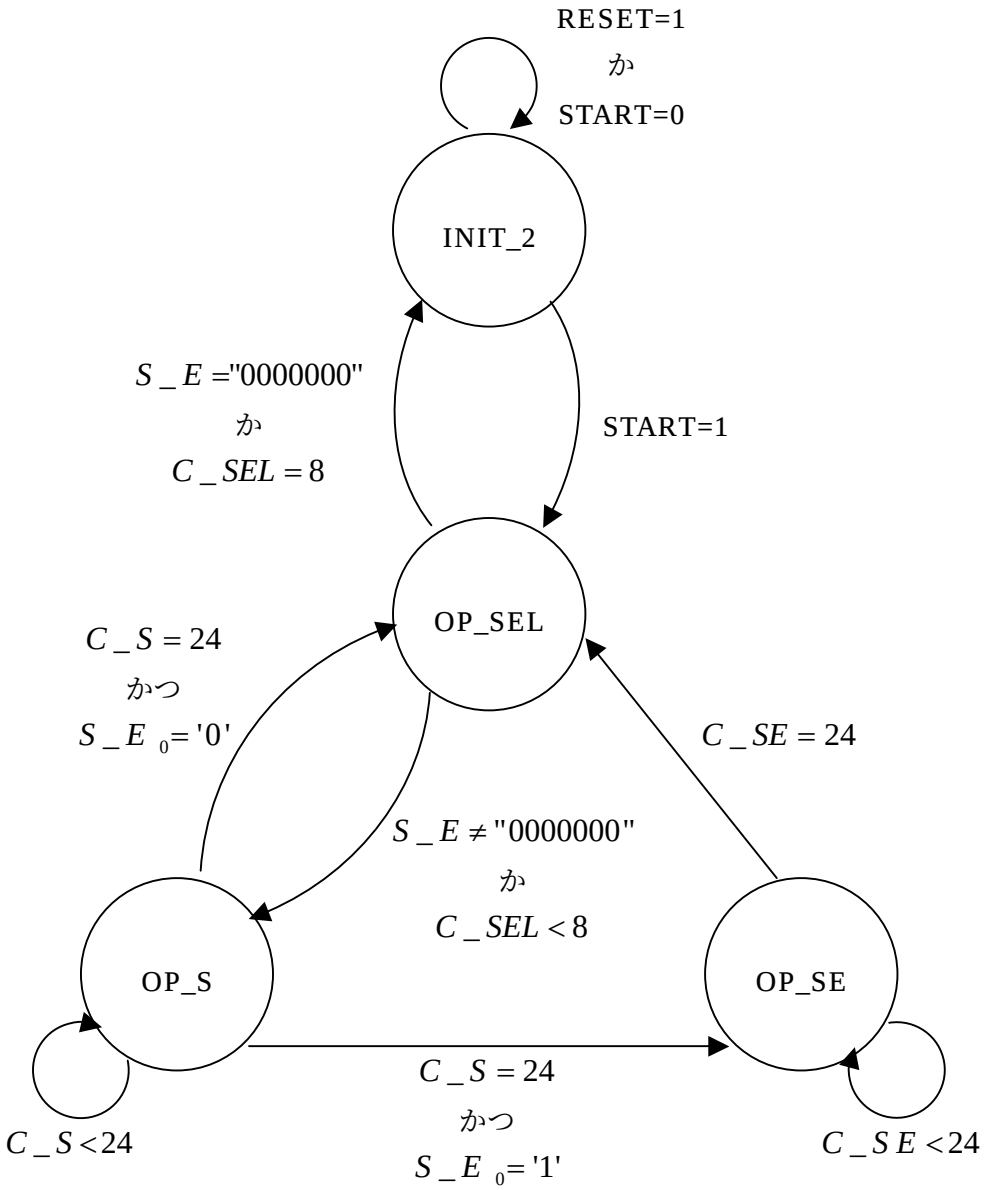

図 4.11　RSA 暗号器の状態遷移図

この状態遷移図には 4 つの状態があり、図中の要因で移り変わるようになっている。

4.2.5　VHDL による RSA 暗号器の設計

図 4.10、4.11 のブロック図ならびに状態遷移図に従い、VHDL で回路を設計する。その VHDL 記述を図 4.12 に、そのテストベンチを図 4.13 に示す。

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity RSA is
    port(
        CLK,RESET,START   : in    std_logic;
        P                 : in    std_logic_vector(6 downto 0);
        M                 : in    std_logic_vector(6 downto 0);
        E                 : in    std_logic_vector(6 downto 0);
        C                 : out std_logic_vector(6 downto 0);
        F_DONE            : out std_logic
        );
end RSA;

--------------------------------------------------------------------------------------------------------------
--   P^E = C (mod M)
--------------------------------------------------------------------------------------------------------------
architecture stimulus of RSA is

component MUL_MOD
    port(
```

図 4.12　RSA 暗号器の VHDL1/7

```
        CLK, RESET, START : in    std_logic;
        MC                : in    std_logic_vector(6 downto 0);
        MR                : in    std_logic_vector(6 downto 0);
        DS                : in    std_logic_vector(6 downto 0);
        DONE              : out std_logic;
        RS                : out std_logic_vector(6 downto 0)
        );
end component;

type    STATE_2 is (INIT_2, OP_SEL, OP_SE, OP_S);
signal CRST_2, NTST_2 : STATE_2;
signal SET_SEL, SET_SE, SET_S : std_logic;
signal P_DONE, MMM_DONE      : std_logic;
signal S_P, S_M, S_E, S_EXP, S_C : std_logic_vector(6 downto 0);
signal C_SEL                 : integer range 0 to 9;
signal C_SE                  : integer range 0 to 25;
signal C_S                   : integer range 0 to 25;
signal START_SOR, START_EXP  : std_logic;
signal DONE_SOR, DONE_EXP    : std_logic;
signal RS_SOR                : std_logic_vector(6 downto 0);
signal RS_EXP                : std_logic_vector(6 downto 0);
signal RESET_SOR, RESET_EXP  : std_logic;

begin

   COMP_SOR : MUL_MOD port map (CLK, RESET_SOR, START_SOR, S_P, S_P,
                                                S_M, DONE_SOR, RS_SOR);
   COMP_EXP : MUL_MOD port map (CLK, RESET_EXP, START_EXP, S_P,
                                              S_EXP, S_M, DONE_EXP, RS_EXP);

RESET_SOR <= RESET;
RESET_EXP <= RESET;
```

図 4.12　RSA 暗号器の VHDL2/7

```
-------------------------------------------------------------------------------------------------------------
--  controller
-------------------------------------------------------------------------------------------------------------
    P_CONTROL_REG_2:process(CLK, RESET)
        begin
            if (RESET = '1') then
                CRST_2 <= INIT_2;
            elsif (CLK'event and CLK = '1') then
                CRST_2 <= NTST_2;
            end if;
    end process;

    P_COUNT_2:process(CLK)
        begin
            if (CLK'event and CLK = '1') then
                if (CRST_2 = INIT_2) then
                    C_SEL <= 0;
                    C_SE   <= 0;
                    C_S    <= 0;
                elsif (CRST_2 = OP_SEL) then
                    C_SEL <= C_SEL + 1;
                    C_SE   <= 0;
                    C_S    <= 0;
                elsif (CRST_2 = OP_SE) then
                    C_SEL <= C_SEL;
                    C_SE   <= C_SE + 1;
                    C_S    <= 0;
                elsif (CRST_2 = OP_S) then
                    C_SEL <= C_SEL;
                    C_SE   <= 0;
                    C_S    <= C_S + 1;
                end if;
            end if;
    end process;
```

図 4.12　RSA 暗号器の VHDL3/7

```
P_CONTROL_STF_2:process(CRST_2, C_SEL, C_SE, C_S, START, S_E(0), S_E)
    begin
        case CRST_2 is
            when INIT_2 => MMM_DONE <= '0';
                           SET_SEL <= '0';
                           SET_SE <= '0';
                           SET_S <= '0';
                           if (START = '1') then
                               SET_SEL <= '1';
                               NTST_2 <= OP_SEL;
                           else
                               NTST_2 <= INIT_2;
                           end if;
            when OP_SEL => SET_SEL <= '0';
                           if (C_SEL = 8 or S_E = "0000000") then
                               MMM_DONE <= '1';
                               NTST_2 <= INIT_2;
                           else
                               MMM_DONE <= '0';
                               SET_S <= '1';
                               NTST_2 <= OP_S;
                           end if;
            when OP_S   => SET_S <= '0';
                           START_SOR <= '1';
                           if (C_S = 24) then
                               START_SOR <= '0';
                               if (S_E(0) = '1') then
                                   SET_SE <= '1';
                                   NTST_2 <= OP_SE;
                               else
                                   SET_SEL <= '1';
                                   NTST_2 <= OP_SEL;
                               end if;
```

図 4.12　RSA 暗号器の VHDL4/7

```
                                    else
                                        NTST_2 <= OP_S;
                                    end if;
                when OP_SE    => SET_SE <= '0';
                                    START_EXP <= '1';
                                    if (C_SE = 24) then
                                        START_EXP <= '0';
                                        SET_SEL <= '1';
                                        NTST_2 <= OP_SEL;
                                    else
                                        NTST_2 <= OP_SE;
                                    end if;
            end case;
    end process;
-------------------------------------------------------------------------------------------------------------
    P_REG_P:process(RESET, START, DONE_SOR)
        begin
            if (RESET = '1') then
                S_P <= (others => '0');
            elsif (START = '1') then
                S_P <= P;
            elsif (DONE_SOR = '1') then
                S_P <= RS_SOR;
            else
                S_P <= S_P;
            end if;
    end process;

    P_REG_M:process(CLK, RESET, START)
        begin
            if (RESET = '1') then
                S_M <= (others => '0');
            elsif (CLK'event and CLK = '1') then
```

図 4.12　RSA 暗号器の VHDL5/7

```
            if (START = '1') then
                S_M <= M;
            else
                S_M <= S_M;
            end if;
        end if;
end process;

P_SIFTER_E:process(RESET, START, SET_SEL)
    begin
        if (RESET = '1') then
            S_E <= (others => '0');
        elsif (START = '1') then
            S_E <= E;
        elsif (SET_SEL'event and SET_SEL = '0')then
            S_E <= '0' & S_E(6 downto 1);
        else
            S_E <= S_E;
        end if;
end process;

P_REG_EXP:process(RESET, START, E(0), DONE_EXP)
    begin
        if (RESET = '1') then
            S_EXP <= (others => '0');
        elsif (START = '1' and E(0) = '1') then
            S_EXP <= P;
        elsif (START = '1' and E(0) = '0') then
            S_EXP <= "0000001";
        elsif (DONE_EXP = '1') then
            S_EXP <= RS_EXP;
        else
            S_EXP <= S_EXP;
```

図 4.12　RSA 暗号器の VHDL6/7

```
        en
end p

P_RE                          SET)
    be
                  T = '1') then
                 <= (others => '0');
        elsif(CLK'event and CLK = '1') then

               S_C <= S_EXP;




                          <= '0';


end pr      ;


                _DONE;
```

4.12　RSA　号器の VHDL7/7

この VHDL では、

$$\bmod M \qquad (4-3)$$

の演算を行っている。入力信号P,M,Eの値によって、出力信号Cが決まる。

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.ALL;

entity testbench is
end testbench;

architecture stimulus of testbench is

component RSA
    port(
        CLK       : IN   std_logic;
        RESET     : IN   std_logic;
        START     : IN   std_logic;
        P         : IN   std_logic_vector(6 downto 0);
        M         : IN   std_logic_vector(6 downto 0);
        E         : IN   std_logic_vector(6 downto 0);
        C         : OUT std_logic_vector(6 downto 0);
        F_DONE : OUT std_logic
        );
end component;

signal CLK        : std_logic := '0';
signal RESET      : std_logic;
signal START      : std_logic;
signal P          : std_logic_vector(6 downto 0);
signal M          : std_logic_vector(6 downto 0);
signal E          : std_logic_vector(6 downto 0);
signal C          : std_logic_vector(6 downto 0);
signal F_DONE     : std_logic;
signal CYCLES     : integer := 0;
```

図 4.13　RSA 暗号器のテストベンチ 1/3

```
begin
    uut : RSA port map(
                          CLK => CLK,
                          RESET => RESET,
                          START => START,
                          P    => P,
                          M    => M,
                          E    => E,
                          C    => C,
                          F_DONE => F_DONE
                          );

    stimulus1:process
        begin
            if (CYCLES < 2000) then
                CYCLES <= CYCLES + 1;
                wait for 10 ns;
                CLK <= not CLK;
            else wait;
            end if;
    end process stimulus1;

    stimulus2:process
        begin
            RESET <= '1' ; wait for 10 ns;
            RESET <= '0' ; wait;
    end process stimulus2;

    stimulus3:process
        begin
            START <= '0' ; wait for 50 ns;
            START <= '1' ; wait for 20 ns;
            START <= '0' ; wait;
    end process stimulus3;
```

図 4.13　RSA 暗号器のテストベンチ 2/3

```
      ulus4:process


end p

 timu
   be

end p


   begin
                        1" ; wait;
           s stimulus6;
```

4.13　RSA 暗号器のテストベンチ 3/3

4.2.6　シミュレーション結果

図 4.12 の VHDL と、図 4.13 のテストベンチによるシミュレーション結果を図 4.14 とし示す。

入力信号 P,M,E にそれぞれ 2 進数で 0110000 ,1001101 , 0100101 が入力され、START の信号が '1' かつ、CLK の立ち上がりで演算を開始し、図 4.11 の状態遷移図に示した動作を行い、演算終了の合図である F_DONE に '1'が立つと同時に、出力信号 C に 2 進数で 0011011 の値が表われている。

この値は(4－3)式を満たす。

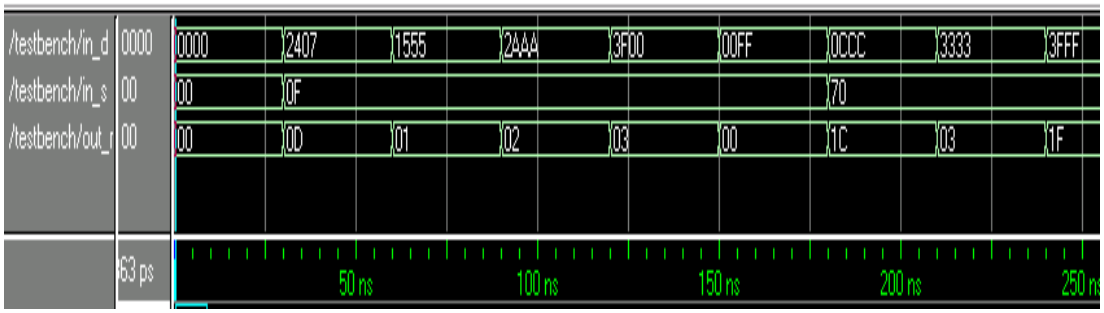

図 4.4　mod 演算器のシミュレーション波形

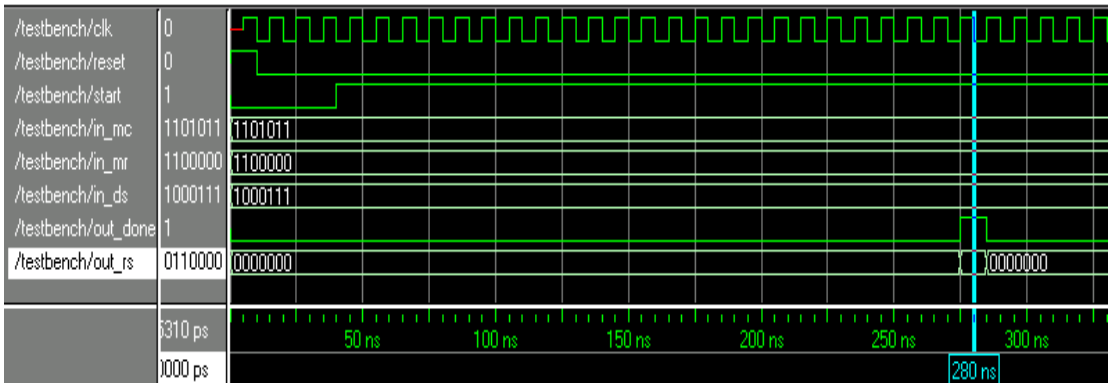

図 4.9　乗算＋mod 演算器のシミュレーション波形

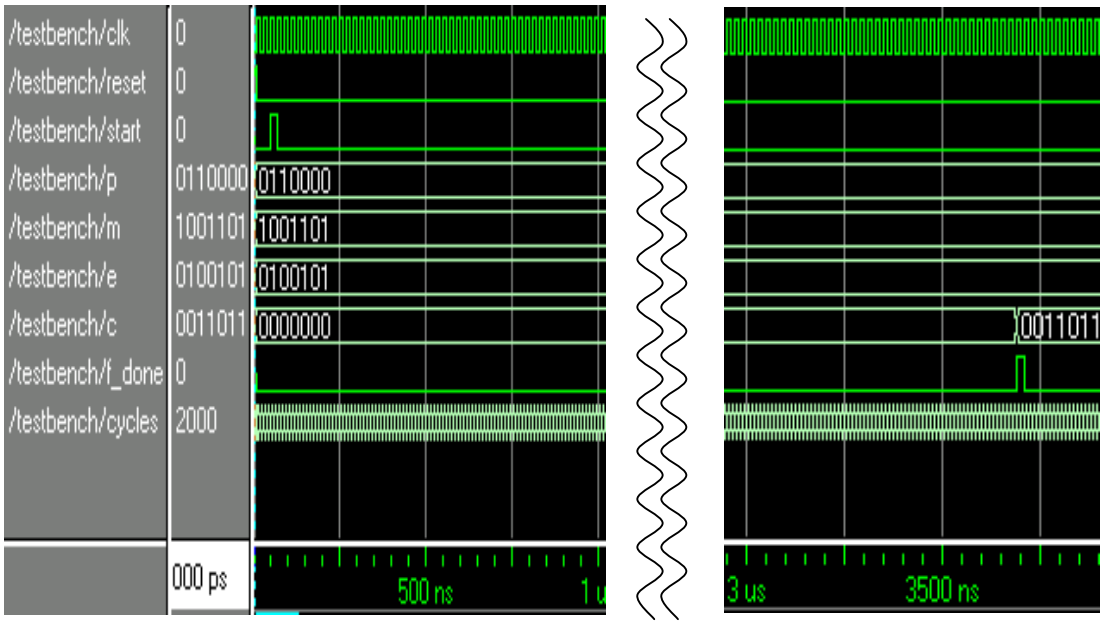

図 4.14　RSA 暗号器のシミュレーション波形

第 5 章　おわりに

RSA 暗号器の設計を行うに当たって、前述したように初等整数論を一から学び、VHDL の基礎を簡単な回路を作ることで学んだ。それにより、序盤は比較的スムーズに進めることができた。しかし、第 4 章での RSA 暗号器の設計では、なかなか思い通りにいかなかった。この原因は、VHDL 記述にまだ不慣れな点があったことにほかならない。

今回の研究では、RSA 暗号器の設計と暗号についての知識を得るという当初の目的を達成できたが、それと同時に自分に不足している部分を再確認することができた。今後は、VHDL 記述に関する勉強に意欲的に取り組みたい。

謝辞

本研究を行うに際し、懇切丁寧な御指導を御鞭撻賜わりました高知工科大学工学部電子・光システム工学科 矢野政顕 教授に心から感謝いたします。

研究中、懇切丁寧な御指導を御鞭撻賜わりました高知工科大学工学部電子・光システム工学科 原央 教授、橘昌良 助教授に厚くお礼申し上げます。

また、同研究室の木村知史氏、ほか矢野研究室の皆様に心から感謝し、お礼申し上げます。

参考文献

[1] “暗号攻防史”…ルドルフ・キッペンハーン著、文春文庫 PP.96~103(2001)

[2] “暗号の数理”…一松信著、講談社 PP.92~93(1980)

[3] “暗号と情報社会”…辻井重男著、文春新書 PP.125~133、PP139~141(1999)

[4] “初等整数論”…H.スターク著、現代数学社 PP.16~117(2000)

[5] “入門！VHDL による回路設計の基礎”
…Design Wave Magazine 2000 年 12 月号 CQ 出版社 PP.67~81 (2000)