

【特許請求の範囲】

【請求項 1】

被乗数を A 、 u とし、乗数を B 、 N とし、乗算剰余演算結果を S としたとき、 $S = S + A \times B + u \times N$ を算出するための乗算剰余演算器であって、

Booth法に基づいて変換された複数のビット数 q 単位で供給される前記乗数 B の値に対応する前記被乗数 A の整数倍の値を選択して出力し、前記Booth法に基づいて変換された複数のビット数 q 単位で供給される前記乗数 N の値に対応する前記被乗数 u の整数倍の値を選択して出力する論理回路と、

前記論理回路から順次出力される値を用いて $A \times B + u \times N$ の演算を実行する桁上げ保存加算器と、

前記桁上げ保存加算器から前記ビット数 q 単位で出力される前記 $A \times B + u \times N$ の演算結果と、前記ビット数 q 単位で供給される過去の該演算結果とを加算し、該加算結果を前記乗算剰余演算結果 S として出力する加算器と、

を有する乗算剰余演算器。

10

【請求項 2】

前記被乗数 A を保持し、前記セレクトアに供給する第 1 の記憶素子と、

前記被乗数 u を保持し、前記セレクトアに供給する第 2 の記憶素子と、

前記加算器から出力される前記乗算剰余演算結果 S を保持し、前記ビット数 q 単位で該乗算剰余演算結果 S を前記加算器に供給する第 3 の記憶素子と、

をさらに有する請求項 1 記載の乗算剰余演算器。

20

【請求項 3】

前記Booth法に基づいて変換した変換後の乗数 B 及び乗数 N を前記論理回路に供給すると共に、前記桁上げ保存加算器の動作を制御する制御部をさらに有する請求項 1 または 2 記載の乗算剰余演算器。

【請求項 4】

前記制御部は、

前記第 1 の記憶素子に前記被乗数 A をセットし、

前記第 2 の記憶素子に前記被乗数 u をセットする請求項 3 記載の乗算剰余演算器。

【請求項 5】

予め算出された、前記被乗数 A 、前記乗数 B 、前記乗数 N 、及び前記乗算剰余演算結果 S の値に対する前記被乗数 u の値の関係が格納される u 生成部をさらに有し、

30

前記制御部は、

前記 $S = S + A \times B + u \times N$ の演算時に前記 u 生成部を参照することで前記被乗数 u の値を決定する請求項 3 または 4 記載の乗算剰余演算器。

【請求項 6】

被乗数を A 、 u とし、乗数を B 、 N とし、乗算剰余演算結果を S としたとき、 $S = S + A \times B + u \times N$ を算出するための乗算剰余演算器であって、

複数のビット数 $q + 1$ 単位で供給される乗数 B の値をBooth法に基づいて変換し、該変換後の値に対応する前記被乗数 A の整数倍の値を選択して出力し、前記ビット数 $q + 1$ 単位で供給される前記乗数 N の値をBooth法に基づいて変換し、該変換後の値に対応する前記被乗数 u の整数倍の値を選択して出力する論理回路と、

40

前記論理回路から順次出力される値を用いて $A \times B + u \times N$ の演算を実行する桁上げ保存加算器と、

前記桁上げ保存加算器から前記ビット数 q 単位で出力される前記 $A \times B + u \times N$ の演算結果と、前記ビット数 q 単位で供給される過去の該演算結果とを加算し、該加算結果を前記乗算剰余演算結果 S として出力する加算器と、

を有する乗算剰余演算器。

【請求項 7】

前記被乗数 A を保持し、前記セレクトアに供給する第 1 の記憶素子と、

前記被乗数 u を保持し、前記セレクトアに供給する第 2 の記憶素子と、

50

前記加算器から出力される前記乗算剰余演算結果 S を保持し、前記ビット数 q 単位で該乗算剰余演算結果 S を前記加算器に供給する第 3 の記憶素子と、
をさらに有する請求項 6 記載の乗算剰余演算器。

【請求項 8】

前記桁上げ保存加算器の動作を制御する制御部をさらに有する請求項 5 記載の乗算剰余演算器。

【請求項 9】

前記制御部は、

前記第 1 の記憶素子に前記被乗数 A をセットし、

前記第 2 の記憶素子に前記被乗数 u をセットし、

前記論理回路に前記乗数 B 及び前記乗数 N を供給する請求項 8 記載の乗算剰余演算器。

10

【請求項 10】

予め算出された、前記被乗数 A 、前記乗数 B 、前記乗数 N 、及び前記乗算剰余演算結果 S の値に対する前記被乗数 u の値の関係が格納される u 生成部をさらに有し、

前記制御部は、

前記 $S = S + A \times B + u \times N$ の演算時に前記 u 生成部を参照することで前記被乗数 u の値を決定する請求項 8 または 9 記載の乗算剰余演算器。

【請求項 11】

前記ビット数 q は 2 である請求項 1 乃至 10 のいずれか 1 項記載の乗算剰余演算器。

【請求項 12】

前記ビット数 q は 4 である請求項 1 乃至 10 のいずれか 1 項記載の乗算剰余演算器。

20

【請求項 13】

請求項 1 に記載の乗算剰余演算器と、

前記被乗数 A を保持し、前記セレクトに供給する第 1 の記憶素子と、

前記被乗数 u を保持し、前記セレクトに供給する第 2 の記憶素子と、

前記加算器から出力される前記乗算剰余演算結果 S を保持し、前記ビット数 q 単位で該乗算剰余演算結果 S を前記加算器に供給する第 3 の記憶素子と、
を有する情報処理装置。

【請求項 14】

前記 Booth 法に基づいて変換した変換後の乗数 B 及び乗数 N を前記論理回路に供給すると共に、前記桁上げ保存加算器の動作を制御する制御部をさらに有する請求項 13 記載の情報処理装置。

30

【請求項 15】

前記制御部は、

前記第 1 の記憶素子に前記被乗数 A をセットし、

前記第 2 の記憶素子に前記被乗数 u をセットする請求項 14 記載の情報処理装置。

【請求項 16】

予め算出された、前記被乗数 A 、前記乗数 B 、前記乗数 N 、及び前記乗算剰余演算結果 S の値に対する前記被乗数 u の値の関係が格納される u 生成部をさらに有し、

前記制御部は、

前記 $S = S + A \times B + u \times N$ の演算時に前記 u 生成部を参照することで前記被乗数 u の値を決定する請求項 14 または 15 記載の情報処理装置。

40

【請求項 17】

請求項 6 記載の乗算剰余演算器と、

前記被乗数 A を保持し、前記セレクトに供給する第 1 の記憶素子と、

前記被乗数 u を保持し、前記セレクトに供給する第 2 の記憶素子と、

前記加算器から出力される前記乗算剰余演算結果 S を保持し、前記ビット数 q 単位で該乗算剰余演算結果 S を前記加算器に供給する第 3 の記憶素子と、
を有する情報処理装置。

【請求項 18】

50

前記桁上げ保存加算器の動作を制御する制御部をさらに有する請求項 17 記載の情報処理装置。

【請求項 19】

前記制御部は、

前記第 1 の記憶素子に前記被乗数 A をセットし、

前記第 2 の記憶素子に前記被乗数 u をセットし、

前記論理回路に前記乗数 B 及び前記乗数 N を供給する請求項 18 記載の情報処理装置。

【請求項 20】

予め算出された、前記被乗数 A、前記乗数 B、前記乗数 N、及び前記乗算剰余演算結果 S の値に対する前記被乗数 u の値の関係が格納される u 生成部をさらに有し、

10

前記制御部は、

前記 $S = S + A \times B + u \times N$ の演算時に前記 u 生成部を参照することで前記被乗数 u の値を決定する請求項 18 または 19 記載の情報処理装置。

【請求項 21】

前記ビット数 q は 2 である請求項 13 乃至 20 のいずれか 1 項記載の情報処理装置。

【請求項 22】

前記ビット数 q は 4 である請求項 13 乃至 20 のいずれか 1 項記載の情報処理装置。

【発明の詳細な説明】

【技術分野】

【0001】

20

本発明はべき乗剰余演算を効率よく処理するための乗算剰余演算器及びそれを備えた情報処理装置に関する。

【背景技術】

【0002】

近年、パーソナルコンピュータや PDA (Personal Digital(Data) Assistants) あるいは携帯電話機等の各種情報処理装置の処理能力が飛躍的に向上し、さらに各種記録メディアの大容量化や通信インフラストラクチャーの整備が進んだことで、個人情報や企業情報等がネットワークや無線手段を介して送受信される機会が増大している。そのため、それらの情報を秘匿化し第三者への漏洩を防ぐ技術が益々重要になってきている。

【0003】

30

送受信データを秘匿化するための一般的な手法としては、データを送受信する端末装置どうしが共通の鍵を用いて該データの暗号化と復号を行う共通鍵暗号方式がよく知られている。さらに、近年では B to B、B to C 等の電子商取引の拡大に伴って PKI (Public Key Infrastructure) 技術が注目されている。

【0004】

PKI の基本技術である公開鍵暗号方式は、公開鍵を用いて送信データを暗号化し、該公開鍵とペアとなる公開することのない秘密鍵を用いて受信データを復号する方式である。この公開鍵暗号方式は、送信側と受信側で異なる鍵を用い、かつ秘密鍵を通信相手に通知する必要が無い場合、上述した共通鍵暗号方式に比べて秘匿化性能が向上する。

【0005】

40

公開鍵暗号方式では、現在、RSA (Rivest, Shamir Adleman) 暗号が主として用いられている (例えば、非特許文献 1 参照)。RSA 暗号は、任意の 2 つの素数を乗算した値 N の素因数分解の困難性と N を法とする数の世界の性質とを利用する暗号化方式であり、暗号化及び復号化のためにべき乗剰余演算 ($M^d \bmod N$) を実行する。

【0006】

べき乗剰余演算は、通常、以下に示す乗算剰余演算の繰り返し処理に置き換えて実行される。

【0007】

例えば、 $d = 19$ とするとき、 $C = M^d \bmod N$ は、

$d = 19 = 1 + 2 \times (1 + 2 \times (0 + 2 \times (0 + 2 \times 1)))$ により、

50

$$\begin{aligned}
 C &= M^{1^9} \bmod N \\
 &= M^{1+2^x(1+2^x(0+2^x(0+2^x1)))} \bmod N \\
 &= ((((M^1)^2 M^0)^2 M^0)^2 M^1)^2 M^1 \bmod N \\
 &= ((M^2)^2)^2 M^2 M \bmod N
 \end{aligned}$$

となる。このようにdを分解すれば、Mを単純にd回掛けるよりも演算回数を低減できるため、演算時間を短縮できる。なお、dの分解方法については様々な方法が知られており、上記はその一例を示している。

【0008】

しかしながら、このような乗算剰余演算も、乗算によって演算桁数が倍になり、さらにその乗算結果をNで除算するため、ハードウェアまたはソフトウェアのいずれを利用して

10

【0009】

モンゴメリ法を応用すると、除算を実質的に行わずに乗算と加減算で上記乗算剰余演算が実現可能であり、乗算剰余演算 $P(A \cdot B)_N = A \cdot B \cdot r^{-n} \bmod N = S$ は、例えば、以下の(1)～(8)で示す手順で求めることができる。但し、 $0 < N < r^n$ 、Nは奇数(Nとrは互いに素である)、 $0 < A < N$ 、 $0 < B < N$ 、 $A = A_{n-1} A_{n-2} \dots A_0$ (例えば $A = A_3 A_2 A_1 A_0 = 1234$) である。

$$(1) \quad v = -N^{-1} \bmod r$$

20

$$(2) \quad S = 0$$

$$(3) \quad \text{for } i = 0 \text{ to } n - 1 \quad \{$$

$$(4) \quad S = S + A_i \cdot B$$

$$(5) \quad u = S \cdot v \bmod r$$

$$(6) \quad S = S + u \cdot N$$

$$(7) \quad S = S / r$$

$$(8) \quad \}$$

乗算剰余演算は、上記アルゴリズムから $S = S + A_i \times B + u \times N$ ($i = 0 \sim n - 1$) の繰り返し演算処理に置き換え可能であり、この処理を実現するための回路である乗算剰余演算器は、例えば図7に示すような構成になる。

30

【0010】

図7は従来の乗算剰余演算器の構成を示すブロック図である。

【0011】

図7に示すように、従来の乗算剰余演算器は、被乗数である上記Aの値を保持する第1のラッチ回路51と、被乗数である上記uの値を保持する第2のラッチ回路52と、 $A + u$ の値を保持する第3のラッチ回路53と、1ビット毎に供給される乗数B、Nの値に応じて被乗数A、u、 $A + u$ 、または0H(全ビット0)を選択し出力するセクタ57と、セクタ57から出力される値を用いて $A \times B + u \times N$ の演算を行う周知の桁上げ保存加算器(Carry Save Adder:以下、CSAと称す)56と、CSA56から出力される乗算剰余演算結果Sと外部で保持された算出済みの乗算剰余演算結果Sとを加算し、該加算結果を乗算剰余演算結果Sとして出力する加算器59とを有する構成である。なお、A、u、及び $A + u$ の各値は、例えば不図示の制御部により第1のラッチ回路51～第3のラッチ回路53に供給され、乗数B、N、及び0Hの各値は、例えば不図示の制御部によりセクタ57に供給される。

40

【0012】

図7に示す乗算剰余演算器では、乗算剰余演算器の処理ビット長(例えば、512bit)の乗数B、Nがそれぞれ1ビット単位でセクタ57に供給される。また、被乗数A、u、 $A + u$ は、CSA56の処理ビット長(図7ではmビット)に対応して、該ビット長単位でラッチ回路に格納され、CSA56に供給される。したがって、例えば乗算剰余演算器の処理ビット長が512bitであり、CSA56の処理ビット長が128bitの場合、図7に示す

50

構成では、被乗数 A 、 u 、 $A + u$ の選択処理を 512 回繰り返すことで $A(128\text{bit}) \times B(512\text{bit}) + u(128\text{bit}) \times N(512\text{bit})$ の演算が完了し、さらに $A(128\text{bit}) \times B(512\text{bit}) + u(128\text{bit}) \times N(512\text{bit})$ の演算を 4 回繰り返すことで、 $A(512\text{bit}) \times B(512\text{bit}) + u(512\text{bit}) \times N(512\text{bit})$ の演算処理が完了することになる。

【0013】

セクタ 57 は、1 ビットずつ供給される乗数 B 、 N の値に応じて、第 1 のラッチ回路 51 ~ 第 3 のラッチ回路 53 から供給される被乗数 A 、 u 、 $A + u$ 、または $0H$ を選択し $CSA56$ に供給する。 $CSA56$ は、セクタ 57 から順次供給される被乗数 A 、 u 、 $A + u$ または $0H$ をシフト加算することで $A \times B + u \times N$ を算出し、その中間演算結果を保持しつつ乗算剰余演算結果 S を 1 ビット単位で出力する。

10

【非特許文献 1】三谷政昭著、「やり直しのための工業数学」、第 5 版、CQ 出版社、2003 年 2 月 1 日、p. 115 - 122

【特許文献 1】特表 2001 - 527673 号公報

【発明の開示】

【発明が解決しようとする課題】

【0014】

現在、公開鍵暗号方式では、上記べき乗剰余演算の C 、 M 、 N 、 d に 1024 ビットの数値を用いた RSA 暗号が広く利用され、さらにビット数が増えることも予想される。そのため、暗号化及び復号化に膨大な量の乗算剰余演算を実行しなければならない。公開鍵暗号方式は、暗号化及び復号化に要する処理時間が共通鍵暗号方式に比べて長いことが問題であり、乗算剰余演算に要する演算時間の短縮が重要な課題となっている。

20

【0015】

図 7 に示した従来の乗算剰余演算器では、例えば被乗数を保持するラッチ回路や CSA の処理ビット長を拡張して一度に処理できるビット数を増やせば、繰り返し処理回数が低減するため演算時間が短縮する。しかしながら、 CSA の処理ビット長を拡張すると、 CSA 内部の中間演算結果を保持するレジスタ、被乗数を保存するためのラッチ回路、及びセクタ回路のビット長が増えるため、乗算剰余演算器の回路規模が増大してしまう問題がある。

【0016】

市場では、携帯電話機、PDA、パーソナルコンピュータやサーバ装置等の情報処理装置の普及に伴い、処理性能が高く、かつ低コストな製品が求められている。したがって、このような要求を満たすためには、乗算剰余演算に要する演算時間を短縮すると共に、回路規模の削減を実現できる乗算剰余演算器が必須となる。

30

【0017】

本発明は上記したような従来の技術が有する問題点を解決するためになされたものであり、演算時間をより短縮できる乗算剰余演算器及び情報処理装置を提供することを目的とする。

【0018】

また、本発明のさらなる目的は、回路規模を増大させることなく演算時間を短縮できる乗算剰余演算器及び情報処理装置を提供することにある。

40

【課題を解決するための手段】

【0019】

上記目的を達成するため本発明の乗算剰余演算器は、被乗数を A 、 u とし、乗数を B 、 N とし、乗算剰余演算結果を S としたとき、 $S = S + A \times B + u \times N$ を算出するための乗算剰余演算器であって、

Booth法に基づいて変換された複数のビット数 q 単位で供給される前記乗数 B の値に対応する前記被乗数 A の整数倍の値を選択して出力し、前記 Booth法に基づいて変換された複数のビット数 q 単位で供給される前記乗数 N の値に対応する前記被乗数 u の整数倍の値を選択して出力する論理回路と、

前記論理回路から順次出力される値を用いて $A \times B + u \times N$ の演算を実行する桁上げ保

50

存加算器と、

前記桁上げ保存加算器から前記ビット数 q 単位で出力される前記 $A \times B + u \times N$ の演算結果と、前記ビット数 q 単位で供給される過去の該演算結果とを加算し、該加算結果を前記乗算剰余演算結果 S として出力する加算器と、
を有する構成である。

【0020】

または、被乗数を A 、 u とし、乗数を B 、 N とし、乗算剰余演算結果を S としたとき、 $S = S + A \times B + u \times N$ を算出するための乗算剰余演算器であって、

複数のビット数 $q + 1$ 単位で供給される乗数 B の値をBooth法に基づいて変換し、該変換後の値に対応する前記被乗数 A の整数倍の値を選択して出力し、前記ビット数 $q + 1$ 単位で供給される前記乗数 N の値をBooth法に基づいて変換し、該変換後の値に対応する前記被乗数 u の整数倍の値を選択して出力する論理回路と、

前記論理回路から順次出力される値を用いて $A \times B + u \times N$ の演算を実行する桁上げ保存加算器と、

前記桁上げ保存加算器から前記ビット数 q 単位で出力される前記 $A \times B + u \times N$ の演算結果と、前記ビット数 q 単位で供給される過去の該演算結果とを加算し、該加算結果を前記乗算剰余演算結果 S として出力する加算器と、
を有する構成である。

【0021】

一方、本発明の情報処理装置は、上記乗算剰余演算器と、

前記被乗数 A を保持し、前記セクタに供給する第1の記憶素子と、

前記被乗数 u を保持し、前記セクタに供給する第2の記憶素子と、

前記加算器から出力される前記乗算剰余演算結果 S を保持し、前記ビット数 q 単位で該乗算剰余演算結果 S を前記加算器に供給する第3の記憶素子と、
をさらに有する構成である。

【0022】

上記のように構成された乗算剰余演算器及び情報処理装置では、Booth法に基づいて乗数を変換し、該変換後の値に対応する被乗数の整数倍の値を選択してCSAに供給するため、CSAの処理ビット長を短縮できる。

【0023】

また、本発明の乗算剰余演算器及び情報処理装置は、予め算出された、前記被乗数 A 、前記乗数 B 、前記乗数 N 、及び前記乗算剰余演算結果 S の値に対する前記被乗数 u の値の関係が格納される u 生成部をさらに有し、

制御部により、前記 $S = S + A \times B + u \times N$ の演算時に前記 u 生成部を参照することで前記被乗数 u の値を決定する構成である。ここで、前記ビット数 q は2または4であることが望ましい。

【0024】

上記のような乗算剰余演算器は、ビット数 q を2または4とすることで、 u 生成部の回路規模の増大を抑制できる。

【発明の効果】

【0025】

本発明の乗算剰余演算器及び情報処理装置は、CSAの処理ビット長を短縮できるため、従来の乗算剰余演算器よりも演算時間を短縮できる。

【0026】

また、CSAの処理ビット長を短縮することで、CSAが備えるフリップフロップ数が低減するため、乗算剰余演算器の回路規模が低減する。特に、ビット数 q を2または4とすれば、 u 生成部の回路規模が増大することがないため、回路規模を増大させることなく演算時間を短縮できる。

【発明を実施するための最良の形態】

【0027】

10

20

30

40

50

次に本発明について図面を参照して説明する。

【0028】

まず、本発明の乗算剰余演算器で利用するBooth法について簡単に説明する。

【0029】

Booth法とは、2の補数表現を利用することで乗算の演算回数を低減する手法である。

例えば、 $A \times 011111$ の演算を行う場合、通常、 $A \times 011111 = A \times 010000 + A \times 001000 + A \times 000100 + A \times 000010 + A \times 000001$ を実行するため、5回の演算処理が必要である。しかしながら、上記2の補数表現を利用すると、乗数である011111を $10000 - 1$ で表すことができるため、 $A \times 011111 = A \times 10000 - 1 = A \times 100000 - A \times 000001$ となり、2回の演算処理で済む。

【0030】

Booth法では、 $A \times B$ を計算する際に、例えば乗数Bを2bit + 重複1bit = 3bit毎に分割し、該分割した乗数Bによる部分積を繰り返し実行する。分割した3bitに対応する部分積の値は表1のようになる。なお、図1はBooth法により乗数011111を2ビット毎に（上記重複1bitを加えると3ビット）変換する際の具体例を示している。

【0031】

【表1】

基数4 : 0, 1, 2, 3 \Rightarrow 0, ± 1 , ± 2

	B[i+1]	B[i]	B[i-1]	Z[i+1]	Z[i]	備考
0	0	0	0	0	0	0
	0	0	1	0	1	A
1	0	1	0	0	1	A
	0	1	1	1	0	2A
2	1	0	0	-1	0	-2A
	1	0	1	0	-1	-A
3	1	1	0	0	-1	-A
	1	1	1	0	0	0

B : 入力値

Z : 出力値

【0032】

乗数を2ビット毎に変換する場合、変換対象である乗数は0、1、2、3のいずれかの値となる（基数4）。一方、Booth法による変換後の乗数は、表1に示したように0、+1、-1、+2、-2のいずれかの値となる。

【0033】

したがって、変換前の乗数（2bit）を用いて乗算を行う場合、乗算結果に対応する値として被乗数の0～3倍の値をそれぞれ用意する必要がある。例えば、被乗数をA、乗数をBとすると、乗数Bが0(0,0)の場合は0、乗数Bが1(0,1)の場合は1A、乗数Bが2(1,0)の場合は2A、乗数Bが3(1,1)の場合は3AをCSAへ供給するため、これらの値を予め用意する必要がある。ここで、0及び1Aは演算処理を必要としない値であり、2Aは、2進数である1Aの値を1ビットずつシフトし、最下位ビットに0をセットすればよい。したがって、実質的に演算処理を必要としない値である。しかしながら、3Aは、1A + 2Aの値を事前に計算するか、または1A及び2Aの2つの値をCSAへそれぞれ供給する必要がある。

【0034】

このような処理でも、被乗数に対して乗数を2bit毎に乗算するため、従来の乗算剰余演算器のように被乗数に対して乗数を1bit毎に乗算する構成（図7参照）に比べて処理時間を短縮できる。しかしながら、 $1A + 2A$ を事前に計算しておく場合は、そのための加算器が必要になるため回路規模が増大する。一方、 $1A$ 及び $2A$ の2つの値をCSAへ供給する場合は、CSAへの入力データ数が増大するため、CSAの回路規模が増大してしまう。

【0035】

これに対して、Booth法を用いて乗数を変換すると、 0 、 ± 1 、 ± 2 倍の被乗数、すなわち、 0 、 $\pm 1A$ 、 $\pm 2A$ のいずれかをCSAへ供給すればよい。このとき、 0 、 $1A$ 、 $2A$ の値は、上述したように実質的な演算処理を必要としないため容易に得ることができる。但し、 $-1A$ （ $-2A$ ）の値は、 $1A$ （ $2A$ ）の値を反転し、 1 を足すことで表現するため、負の数であることを示すサインビット（1bit）が必要となる。

【0036】

本発明の乗算剰余演算器は、乗数 B 、 N のビット列を、所定のビット数毎にBooth法を用いて変換し、変換後の乗数 B 、 N の値に対応する被乗数 A 、 u の整数倍の値（ 0 、 ± 1 、 ± 2 ）を用いてCSAにより $A \times B + u \times N$ の演算処理を行う構成である。

【0037】

図2は本発明の乗算剰余演算器の一構成例を示すブロック図である。

【0038】

図2に示すように、本発明の乗算剰余演算器は、被乗数 A の値を保持する第1のラッチ回路1と、被乗数 u の値を保持する第2のラッチ回路2と、複数ビット（図2では3bit）毎に供給される乗数 B の値に対応する被乗数 A の整数倍の値（ 0 、 $\pm 1A$ 、 $\pm 2A$ ）を選択して出力する第1の論理回路（logic1）4と、複数ビット（図2では3bit）毎に供給される乗数 N の値に対応する被乗数 u の整数倍の値（ 0 、 $\pm 1u$ 、 $\pm 2u$ ）を選択して出力する第2の論理回路（logic2）5と、第1の論理回路4及び第2の論理回路5から供給される値を用いて $A \times B + u \times N$ の演算を実行する周知のCSA6と、CSA6から複数ビット（図2では2bit）単位で出力される乗算剰余演算結果 S を保持し、複数ビット（図2では2bit）単位で出力する第1のシフトレジスタ8と、CSA6から出力される $A \times B + u \times N$ の演算結果と第1のシフトレジスタ8の出力とを加算し、加算結果を乗算剰余演算結果 S として第1のシフトレジスタ8に再び格納する加算器9と、被乗数 u の値を生成するためのテーブルが格納される u 生成部10と、被乗数 A 、 u の値を第1のラッチ回路1及び第2のラッチ回路2に供給し、乗数 B 、 N の値を第1及び第2の論理回路4、5に供給すると共に、CSA6、第1のシフトレジスタ8及び u 生成部10の動作を制御する制御部11とを有する構成である。

【0039】

本発明の乗算剰余演算器は、制御部11による被乗数 A 、 u のラッチ回路へのセット、及び乗数 B 、 N の第1の論理回路4及び第2の論理回路5へのセットを契機に、外部から供給される所定周波数のクロック（CK）にしたがって動作する回路であり、制御部11は、例えばプログラムにしたがって動作するCPU、DSPあるいは論理回路等によって実現される。

【0040】

このような構成において、本発明の乗算剰余演算器では、被乗数 A 、 u が、例えばCSA6の処理ビット長に対応して複数に分割され、制御部11により該分割単位で第1及び第2のラッチ回路1、2に格納される。また、第1のラッチ回路1から第1の論理回路4へはCSA6の処理ビット長に対応して n ビット単位で被乗数 A が供給され、第2のラッチ回路2から第2の論理回路5へはCSA6の処理ビット長に対応して n ビット単位で被乗数 u が供給される。一方、乗数 B 、 N は、例えば制御部11から3bit単位で第1及び第2の論理回路4、5に供給される。

【0041】

なお、乗数 B 、 N は、例えばシフトレジスタやRAM等のように、格納されたデータを

10

20

30

40

50

複数ビット単位で出力できる記憶素子に一旦格納し、該記憶素子から所定の複数ビット単位で第1及び第2の論理回路4、5へ供給してもよい。その場合、記憶素子には、制御部11により乗算剰余演算器の処理ビット長単位、あるいはそれを複数ビット長毎に分割した分割単位で乗数B、Nが格納される。

【0042】

また、図2では、乗数B、Nを3bit(2bit+重複1bit)単位で第1及び第2の論理回路4、5に供給する例を示しているが、乗数B、Nの供給単位は4bit以上であってもよい。例えば、基数が16の場合、乗数B、Nは5bit(4bit+重複1bit)単位で第1及び第2の論理回路4、5に供給される。

【0043】

第1の論理回路4は、第1のラッチ回路1から供給される被乗数Aの値を用いて $\pm 1A$ 、 $\pm 2A$ を生成し、3bit毎に供給される乗数BをBooth法に基づいて変換し、該変換結果に対応する0、 $\pm 1A$ 、 $\pm 2A$ のいずれかを選択し、選択結果を $n+4$ ビット単位でCSA6へ供給する。また、第2の論理回路5は、第2のラッチ回路2から供給される被乗数uの値を用いて $\pm 1u$ 、 $\pm 2u$ を生成し、3bit毎に供給される乗数NをBooth法に基づいて変換し、該変換結果に対応する0、 $\pm 1u$ 、 $\pm 2u$ のいずれかを選択し、選択結果を $n+4$ ビット単位でCSA6へ供給する。図2では2つの論理回路を用いて0、 $\pm 1A$ 、 $\pm 2A$ 、または0、 $\pm 1u$ 、 $\pm 2u$ を選択する例を示しているが、乗数B、Nの値に対応する0、 $\pm 1A$ 、 $\pm 2A$ 、または0、 $\pm 1u$ 、 $\pm 2u$ を選択できれば、論理回路の数はいくつであってもよい。また、図2では第1の論理回路4及び第2の論理回路5により3bit毎に供給される乗数BをBooth法に基づいて変換する例を示しているが、制御部11により変換後の値を第1の論理回路4及び第2の論理回路5に供給する構成であってもよい。その場合、第1の論理回路4には2bit毎に乗数Bが供給され、第2の論理回路5には2bit毎に乗数Nが供給される。

【0044】

第1の論理回路4及び第2の論理回路5から出力される被乗数の選択値が $n+4$ ビット単位となる理由は以下による。

【0045】

例えば、最初の演算において乗数B、Nの値により $2A$ 、 $2u$ が選択された場合、CSA6による演算結果Sは、

$$S = 2A[n:0] + 2u[n:0]$$

となる。

【0046】

このとき、 $(n+1bit) + (n+1bit)$ より、演算結果Sの桁数は $(n+2bit)$ となる。

【0047】

この演算結果Sのうち、下位2ビットがCSA6から出力され、残りのnビットはCSA6に保存されて次の演算で加算される。

【0048】

続いて、次の演算において乗数B、Nの値により再び $2A$ 、 $2u$ が選択されると、CSA6による演算結果Sは、

$$S = 2A[n:0] + 2u[n:0] + S[n-1:0]$$

となる。

【0049】

このとき、演算結果Sの桁数は $(n+1bit) + (n+1bit) + (nbit)$ より $(n+3bit)$ となる。

【0050】

この演算結果Sのうち、下位2ビットがCSA6から出力され、残りの $n+1$ ビットはCSA6に保存されて次の演算で加算される。

【0051】

さらに、次の演算において乗数B、Nの値により再び $2A$ 、 $2u$ が選択されると、CSA6による演算結果Sは、

10

20

30

40

50

$$S = 2A[n:0] + 2u[n:0] + S[n:0]$$

となる。

【0052】

このとき、演算結果 S の桁数は $(n+1\text{bit}) + (n+1\text{bit}) + (n+1\text{bit})$ より $(n+3\text{bit})$ となる。

【0053】

この演算結果 S のうち、下位 2 ビットが $CSA6$ から出力され、残りの $n+1$ ビットは $CSA6$ に保存されて次の演算で加算される。以下、同様の演算処理が繰り返され、演算の終了毎に下位 2 ビットが出力され、 $n+1$ ビットが $CSA6$ で保存されて次の演算で利用される。このとき、演算結果 S の桁数は $(n+1\text{bit}) + (n+1\text{bit}) + (n+1\text{bit})$ であり、必ず $(n+3\text{bit})$ 内に収まる。

【0054】

したがって、最大値である $2A$ 、 $2u$ が加算される場合を考慮しても演算結果 S の桁数は最大でも $n+3$ ビットとなる。但し、負の最大値 ($-2A$ 、 $-2u$) が繰り返し選択される場合を考慮すると、負の数であることを示すサインビット (1bit) が必要となるため、演算結果 S の桁数は合計で $n+4$ ビットになる。よって、第 1 の論理回路 4 及び第 2 の論理回路 5 から $CSA6$ に供給する被乗数の選択値も演算結果 S の桁数に合わせて最大で $n+4$ ビットとなる。

【0055】

$CSA6$ は、各論理回路から順次供給される値をシフト加算することで $A \times B$ 、及び $u \times N$ をそれぞれ算出し、それらの加算結果 S を出力する。本発明の乗算剰余演算器が備える $CSA6$ は、第 1 及び第 2 の論理回路 4、5 から最大で $n+4$ ビットのデータが供給されるため、このビット拡張に対応する分だけ従来の乗算剰余演算器が備える CSA よりも処理ビット長が拡張される。 $CSA6$ は、桁上げ (carry) 出力及び加算結果 (sum) 出力が格納されるシフトレジスタをそれぞれ備え、該シフトレジスタを用いて中間演算結果を保持しつつ演算結果 S を複数ビット単位 (図 2 では 2bit) で出力する。 $CSA6$ から出力された演算結果 S は、第 1 のシフトレジスタ 8 の出力 (過去の乗算剰余演算結果 S) と複数ビット単位で加算され、加算結果は第 1 のシフトレジスタ 8 に再び格納される。

【0056】

なお、図 2 に示した第 1 のラッチ回路 1、第 2 のラッチ回路 2、第 1 のシフトレジスタ 8 及び u 生成部 10 は、乗算剰余演算器の内部に備えている必要はなく、乗算剰余演算器を利用する情報処理装置に備えていてもよい。同様に、乗数 B 、 N の値を一時的に保持する記憶素子を備えている場合、該記憶素子は乗算剰余演算器の内部に備えている必要はなく、乗算剰余演算器を利用する情報処理装置に備えていてもよい。さらに、制御部 11 も乗算剰余演算器の内部に備えている必要はなく、乗算剰余演算器を利用する情報処理装置が備える処理装置 (CPU) によって実現してもよい。すなわち、乗算剰余演算器は、図 2 の点線内の構成要素のみを備えていればよい。

【0057】

また、被乗数 A 、 u は、ラッチ回路に格納する必要はなく、例えばシフトレジスタや RAM 等のようにデータを一時的に保持できる記憶素子であればどのようなものを用いてもよい。

【0058】

図 3 に示すように、本発明の情報処理装置は、例えばパーソナルコンピュータやサーバ装置等のコンピュータシステムであり、プログラムにしたがって所定の処理を実行する処理装置 20 と、処理装置 20 に対してコマンドや情報等を入力するための入力装置 30 と、処理装置 20 の処理結果をモニタするための出力装置 40 とを有する構成である。

【0059】

処理装置 20 は、CPU 21 と、CPU 21 の処理に必要な情報を一時的に記憶する主記憶装置 22 と、CPU 21 に上記制御部 11 の処理を実行させるプログラムが記録された記録媒体 23 と、処理に必要なデータ等を蓄積するデータ蓄積装置 24 と、主記憶装置 22、記録媒体 23、及びデータ蓄積装置 24 とのデータ転送を制御するメモリ制御イン

10

20

30

40

50

タフェース部 25 と、入力装置 30 及び出力装置 40 とのインタフェース装置である I/O インタフェース部 26 と、図 1 に示した乗算剰余演算器 27 と、ネットワーク等との通信を制御するインタフェースである通信制御装置 28 とを備え、それらがバス 29 等を通じて接続された構成である。なお、処理装置 20 には、乗算剰余演算器 27 の構成に応じて、被乗数 A、u を保持するラッチ回路、及び乗数 B、N、及び演算結果 S を保持するシフトレジスタ等を備えていてもよい。

【0060】

処理装置 20 は、記録媒体 23 に記録されたプログラムにしたがって CPU 21 により上記制御部 11 の処理を実行し、乗算剰余演算器 27 を用いて $S = S + A_i \times B + u \times N$ の演算を実行する。なお、記録媒体 23 は、磁気ディスク、半導体メモリ、光ディスクあるいはその他の記録媒体であってもよい。

【0061】

次に、本発明の乗算剰余演算器の動作について図面を用いて具体的に説明する。

【0062】

以下では、A、u、B、N がそれぞれ 512bit であり、処理ビット長が 64bit の CSA6 を用い、乗数 B、N が 3bit 単位で第 1 の論理回路 4 及び第 2 の論理回路 5 へ供給され、第 1 のシフトレジスタ 8 が 2bit 単位で乗算剰余演算結果 S を入出力する場合を例にして説明する。また、第 1 及び第 2 のラッチ回路 1、2 には被乗数 A、u が CSA6 の処理ビット長に合わせて 64bit 単位で格納されるものとする。

【0063】

処理ビット長が 64bit の CSA6 を用い、乗数 B、N を 3bit 単位で出力する場合、A、u、B、N がそれぞれ 512bit の乗算剰余演算 ($512\text{bit} \times 512\text{bit} \times 2^{-512} \bmod 512\text{bit}$) は、 $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ ($A \times B \times 2^{-64} \bmod N$) の演算を繰り返し実行すればよい。

【0064】

本発明の乗算剰余演算器では、モンゴメリ法による乗算剰余演算の特徴である、下位ビットが 0 になることを利用して（ここでは、下位 64bit が 0 H）、上記 S、A、B、N の値に対応する u を予め算出し、u 生成部 10 にテーブル形式で格納しておく。

【0065】

例えば、乗数を 2bit（重複 1bit を除く）単位で出力する場合、u の値を以下のようにして求める（但し、N は奇数）。

【0066】

N[1:0]=01, (S+AiB)[1:0]=00 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=00
 N[1:0]=01, (S+AiB)[1:0]=01 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=11
 N[1:0]=01, (S+AiB)[1:0]=10 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=10
 N[1:0]=01, (S+AiB)[1:0]=11 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=01
 N[1:0]=11, (S+AiB)[1:0]=00 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=00
 N[1:0]=11, (S+AiB)[1:0]=01 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=01
 N[1:0]=11, (S+AiB)[1:0]=10 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=10
 N[1:0]=11, (S+AiB)[1:0]=11 のとき、
 $S = S + AiB + uN = 00$ となる u は、u[1:0]=11

以上をまとめると、表 2 のようになる。

【0067】

10

20

40

50

【表 2】

N[1]	S+AiB[1:0]	u
0	00	00
0	01	11
0	10	10
0	11	01
1	00	00
1	01	01
1	10	10
1	11	11

10

【0068】

ここで、A、B、Nはいずれも既知の値であり、Sは0H（演算開始時）または直前の $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ の演算結果を用いるため既知である。なお、Nは奇数であるため、N[1:0]=01または11で固定である。したがって、A、B、及びSの各値を基に算出した被乗数uの値をテーブル形式でu生成部10に格納しておき、制御部11は該テーブルを参照して被乗数uの値を決定する。

20

【0069】

本発明の乗算剰余演算器では、まず、制御部11により、第1のラッチ回路1に被乗数A（512bit）の最下位64bitのデータをセットし、乗数B（512bit）のデータを第1の論理回路4へ供給し、乗数N（512bit）のデータを第2の論理回路5へ供給する。

【0070】

続いて、制御部11は、64bitの被乗数A、64bitの乗数B、64bitの乗数Nからu生成部10に格納されたテーブルを参照してu（64bit分）の値を求め、第2のラッチ回路2に格納する。

【0071】

制御部11による第1のラッチ回路1、第2のラッチ回路2、第1の論理回路4及び第2の論理回路5に対する被乗数または乗数のセットが完了すると、乗算剰余演算器は $S = S + A \times B + u \times N$ の演算を開始する。

30

【0072】

乗算剰余演算器は、まず、第1の論理回路4にて、3bitの乗数Bの値からBooth法による変換を行い、該変換後の値に対応する0、+1A（64+4bit）、-1A（64+4bit）、+2A（64+4bit）または-2A（64+4bit）を選択しCSA6へ供給する。同様に、乗算剰余演算器は、第2の論理回路5にて、3bitの乗数Nの値からBooth法による変換を行い、該変換後の値に対応する0、+1u（64+4bit）、-1u（64+4bit）、+2u（64+4bit）または-2u（64+4bit）を選択しCSA6へ供給する。

【0073】

CSA6は、第1の論理回路4及び第2の論理回路5から順次供給される値を、桁合わせを実行しつつ加算することで $A \times B$ 、及び $u \times N$ を算出し、それらの加算結果（乗算剰余演算結果）Sを2bit単位で出力する。CSA6から出力された演算結果は、第1のシフトレジスタ8の出力と2bit単位で加算器9にて加算され、加算後の値が第1のシフトレジスタ8に再び格納される。以上の処理を乗数B、Nの全てのビットデータに対して繰り返し実行することで、 $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ の演算が終了する。但し、この段階ではCSA6の内部に部分積の演算結果の上位64bitが残っているため、このデータを制御部11の指示により第1のシフトレジスタ8に格納する。その結果、該記憶素子に $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ の演算結果Sが格納される。

40

【0074】

50

乗算剰余演算器は、 $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ の演算が完了すると、制御部 1 により第 1 のラッチ回路 1 に被乗数 A (512bit) の次の下位 64bit のデータ (最下位から 65bit 目 ~ 128bit 目のデータ) をセットし、上記と同様に u 生成部 10 のテーブルを参照して被乗数 u の値を求め、求めた値を第 2 のラッチ回路 2 に格納した後、再び $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ の演算を開始する。

【0075】

以降、第 1 のラッチ回路 1 に格納される被乗数 A (512bit) の全てのビットデータに対して同様の処理を繰り返し実行する。すなわち、上記 $64\text{bit} \times 512\text{bit} \times 2^{-64} \bmod 512\text{bit}$ の演算を 8 回繰り返す。その結果、本発明の乗算剰余演算器による $512\text{bit} \times 512\text{bit} \times 2^{-512} \bmod 512\text{bit}$ の演算が終了する。

10

【0076】

次に、本発明の乗算剰余演算器の効果について図面を用いて説明する。

【0077】

図 4 は乗数を 1bit 単位で出力する従来の乗算剰余演算器のレイアウト面積及び Booth 法を採用する本発明の乗算剰余演算器のレイアウト面積を示すグラフである。また、図 5 は乗数を 1bit 単位で出力する従来の乗算剰余演算器の処理クロック数及び Booth 法を採用する本発明の乗算剰余演算器の処理クロック数を示すグラフである。

【0078】

また、図 6 は乗数を 1bit 単位で出力する従来の乗算剰余演算器及び Booth 法を採用する本発明の乗算剰余演算器の処理クロック数に対するレイアウト面積をそれぞれ示すグラフ

20

【0079】

図 4 及び図 5 に示す「1bit」とは乗数を 1bit 単位で出力する従来の乗算剰余演算器の構成を示し、「Booth 2bit」とは Booth 法による変換後の乗数を用いる (基数 4) 本発明の乗算剰余演算器の構成を示している。また、図 4 及び図 5 に示すグラフの横軸 (処理性能) は、表 3 に示すように乗算剰余演算器の処理ビット長 (32bit、64bit、128bit、256bit) に対応する、従来の乗算剰余演算器が備える CSA の処理ビット長と本発明の乗算剰余演算器が備える CSA の処理ビット長とを示している。本発明の乗算剰余演算器は、乗数を 2bit 単位で被乗数に掛けるため、処理性能を比較する際には、表 3 に示すように乗数を 1bit 単位で被乗数に掛ける従来の乗算剰余演算器に対して CSA の処理ビット長を $1/2$ にしている。なお、表 3 の各エントリは (CSA の処理ビット長) * (出力ビット数) を示している。

30

【0080】

【表 3】

		処理性能			
		32bit	64bit	128bit	256bit
構成	1bit	32bit*1bit	64bit*1bit	128bit*1bit	256bit*1bit
	Booth	16bit*2bit	32bit*2bit	64bit*1bit	128bit*1bit

40

【0081】

図 4 から分かるように、乗算剰余演算器としての処理ビット長が同じである場合、本発明の乗算剰余演算器は、乗数を複数ビット単位で処理できるため、乗数を 1bit 単位で処理する従来の乗算剰余演算器に比べて回路のレイアウト面積が低減する。これは Booth 2bit とすることで CSA の処理ビット長を従来の半分にできるためである。

【0082】

例えば、乗算剰余演算器の処理ビット長を 128bit とした場合、従来の乗算剰余演算器では、CSA で加算結果 (sum) の値と桁上げ (carry) の値をそれぞれ 128 個ずつ保持する必

50

要があるため、256個のフリップフロップ(Data-F/F)が必要になる。

【0083】

それに対して、Booth 2bitを採用する本発明の乗算剰余演算器が備えるCSA6では、処理ビット長が従来の半分の64bitで済むため、加算結果(sum)の値と桁上げ(carry)の値を保持するフリップフロップも128個で済む。すなわち、Booth法を採用することで複数ビット単位で乗数进行处理するため、CSA6が備えるフリップフロップの数が大きく削減され、回路規模を低減できる。また、CSA6の処理ビット長が短縮することで第1及び第2のラッチ回路や論理回路(従来の構成ではセレクトに相当)のビット長も短縮されるため、乗算剰余演算器としての回路規模が低減する。但し、上述したようにBooth法を採用することでCSAの処理ビット長を拡張する必要があり(基数4の場合、4bit)、さらに第1の論理回路4及び第2の論理回路5による回路規模の増大もあるため、本発明の乗算剰余演算器のレイアウト面積は従来の1/2よりも大きくなる。

10

【0084】

一方、図5から分かるように、乗算剰余演算器の処理ビット長が同じである場合、本発明の乗算剰余演算器は、乗数を複数ビット単位で処理するため、乗数を1bit単位で処理する従来の乗算剰余演算器に比べて処理クロック数が少なくなる。これは上述したCSA6内に残る部分積の演算結果を出力する処理時間の差から生じる結果である。

【0085】

本発明の乗算剰余演算器では、上述したようにCSA6の処理ビット長を従来の半分にできるが(基数4の場合)、被乗数を分割して処理するため、乗算剰余演算を複数回繰り返すことになる。そのため、本発明の乗算剰余演算器では、従来の乗算剰余演算器よりも繰り返し演算の回数が増え、CSA6内に残る部分積の演算結果を出力する回数も増えてしまう。

20

【0086】

しかしながら、本発明の乗算剰余演算器では、CSA6の処理ビット長を短縮できることから、CSA6内に残る演算結果を出力する処理時間も従来の1/2となる(基数4の場合)。そのため、僅かではあるが、1つのA、u、B、Nに対する乗算剰余演算の処理時間は従来よりも低減する。

【0087】

本発明の乗算剰余演算器は、処理時間の大幅な低減は実現できないが、多数の数字の配列に対して大きな値のべき乗剰余演算を行うRSAによる暗号化及び復号に本発明の乗算剰余演算器を用いる場合は、この僅かな処理時間の向上が非常に有益となる。

30

【0088】

図6に示すように、Booth法を採用する本発明の乗算剰余演算器は、乗数を1bit単位で出力する従来の乗算剰余演算器に比べて、回路規模が少なく、かつ高速な処理を実現できることが分かる。

【0089】

なお、参考までに、Booth法を採用する本発明の乗算剰余演算器の基数を増やした場合の回路規模の増大量を表4及び表5に示す。本発明の乗算剰余演算器では、基数が16の場合、乗数B、Nは4bit毎に処理されるため、CSA6のビット幅が同じ場合、処理性能は従来の乗算剰余演算器の4倍になる。なお、表4及び表5の各エントリ内の数字の単位は[mm²]である。

40

【0090】

【表 4】

処理性能	従来 (1bit 毎)	Booth 手法	
		基数 4 (2bit 毎)	基数 16 (4bit 毎)
64bit	0.292	0.214	0.224
128bit	0.580	0.403	0.393
256bit	1.153	0.778	0.741

【0091】

10

表 4 に示すように、Booth法を採用する本発明の乗算剰余演算器は、基数 4、16 共にほぼ同じ回路規模で構成され、従来の乗算剰余演算器と比較してレイアウト面積が約 30 % 削減されることが分かる。

【0092】

【表 5】

CSAビット幅	従来 (1bit 毎)	Booth 手法	
		基数 4 (2bit 毎)	基数 16 (4bit 毎)
16bit	0.076	0.117	0.224
32bit	0.148	0.214	0.393
64bit	0.292	0.402	0.741
128bit	0.580	0.778	1.463
256bit	1.153	1.529	2.894

20

【0093】

表 5 に示すように、Booth法を採用する本発明の乗算剰余演算器は、従来の乗算剰余演算器に比べて、基数 4 の場合、処理速度は約 2 倍になるがレイアウト面積は 1.3 倍程度で済む。また、基数 16 の場合、処理速度は約 4 倍になるがレイアウト面積は 2.6 倍程度で済む。

【0094】

30

ところで、被乗数 u は、乗数 B 、 N の出力ビット数を q とすると、上記モンゴメリ法を応用したアルゴリズムの (1)、(5) から以下の式で算出できる。

【0095】

$$v = -N^{-1} \bmod 2^q$$

$$u = Sv \bmod 2^q$$

ここで、 v は演算開始時に一度だけ計算する値である。なお、 r に代えて 2^q としているのは r を 2 進数で表したためである。

【0096】

$q = 1$ となる従来の乗算剰余演算器では、 N が奇数であることから $v = 1$ となるため、 $u = S \bmod 2 = S[0]$ となり、被乗数 u は S の下位ビットに等しくなる。したがって、被乗数 u を実施的に計算する必要はない。

40

【0097】

しかしながら、 $q > 1$ となる本発明の乗算剰余演算器では、 $u = S[0]$ が成立しないため、上記 2 つの演算が必要になる。但し、 q の値が小さい場合 (例えば、 $q = 2, 4$) は、 v, u も 2bit または 4bit であり、その演算に必要な N, S も 2bit または 4bit である。そのため、本発明では A, B, S, N の値から予め u の値を算出してテーブルを作成しておき、該テーブルを参照することで第 2 のラッチ回路 2 に格納する u を決定している。

【0098】

Booth法による乗数の変換に用いる基数の値を大きくし q の値を増やせば、CSA の処理ビット長をさらに短縮できるため、乗算剰余演算の処理時間をさらに短縮することが

50

できる。

【 0 0 9 9 】

しかしながら、 $q > 4$ の場合、すなわち乗数 B 、 N を 8 ビット以上で出力する（基数 64 以上）構成では、被乗数 u をテーブル内から選択するために必要な、例えばデコーダ等の回路規模が増大するため、記憶素子を含む u 生成部 10 の回路規模が増大し、上述した $CSA6$ の処理ビット長を短縮することによる乗算剰余演算器の回路規模の低減効果を相殺してしまう。

【 0 1 0 0 】

表 6 に q の値に対する u 生成部 10 のレイアウト面積（単位： mm^2 ）を示し、表 7 に q の値に対する CSA と u 生成部とを含む総レイアウト面積（単位： mm^2 ）を示す。

【 0 1 0 1 】

【表 6】

$q=1$	$q=2$	$q=4$	$q=8$
0	0.003	0.014	0.937

【 0 1 0 2 】

【表 7】

$CSA+u$ 生成部	$q=1$	$q=2$	$q=4$	$q=8$
32bit	0.103	0.169	0.308	1.371
64bit	0.292	0.423	0.592	1.903
128bit	0.580	0.842	1.171	2.988
256bit	1.153	1.691	2.310	5.135

【 0 1 0 3 】

表 6 及び表 7 から分かるように、例えば CSA の処理ビット長を 256bit としたとき、 $q = 1$ のときの総レイアウト面積に対して、 CSA の処理ビット長を 128bit にできる $q = 2$ の場合（基数 4）及び CSA の処理ビット長を 64bit にできる $q = 4$ の場合（基数 16）の総レイアウト面積は低減する。しかしながら、 $q = 8$ （基数 64）にすると総レイアウト面積が増大してしまう。

【 0 1 0 4 】

したがって、本発明の乗算剰余演算器では、 q の値が 2 または 4 であることが回路規模の増大を抑制しつつ演算時間を短縮できるために望ましい。但し、回路規模よりも演算時間の向上を優先する場合は、 q の値を 8 以上に設定してもよい。その場合、 q の値は u 生成部 10 のレイアウト面積の増大を考慮しつつ最適な値を選択すればよい。

【図面の簡単な説明】

【 0 1 0 5 】

【図 1】Booth 法による乗数の具体的な変換例を示す模式図である。

【図 2】本発明の乗算剰余演算器の一構成例を示すブロック図である。

【図 3】本発明の情報処理装置の一構成例を示すブロック図である。

【図 4】本発明の乗算剰余演算器のレイアウト面積を示すグラフである。

【図 5】本発明の乗算剰余演算器の処理クロック数を示すグラフである。

【図 6】本発明の乗算剰余演算器の処理クロック数に対するレイアウト面積の関係を示すグラフである。

【図 7】従来の乗算剰余演算器の構成を示すブロック図である。

【符号の説明】

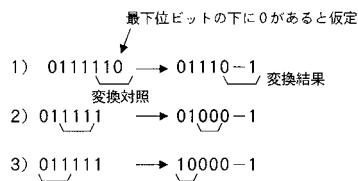
【0106】

- 1 第1のラッチ回路
- 2 第2のラッチ回路
- 4 第1の論理回路
- 5 第2の論理回路
- 6 CSA
- 8 第1のシフトレジスタ
- 9 加算器
- 10 u生成部
- 11 制御部
- 20 処理装置
- 21 CPU
- 22 主記憶装置
- 23 記録媒体
- 24 データ蓄積装置
- 25 メモリ制御インタフェース部
- 26 I/Oインタフェース部
- 27 乗算剰余演算器
- 28 通信制御装置
- 29 バス
- 30 入力装置
- 40 出力装置

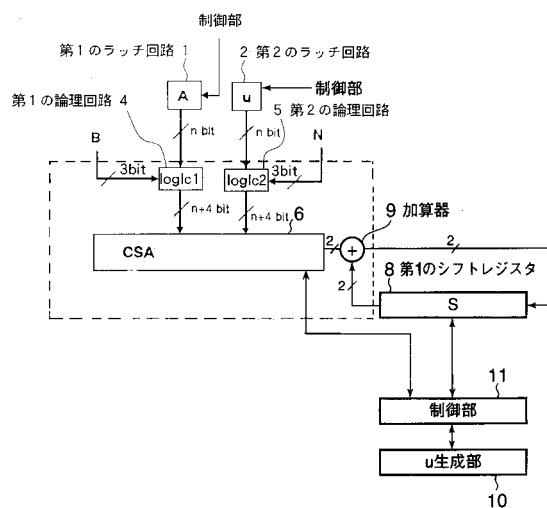
10

20

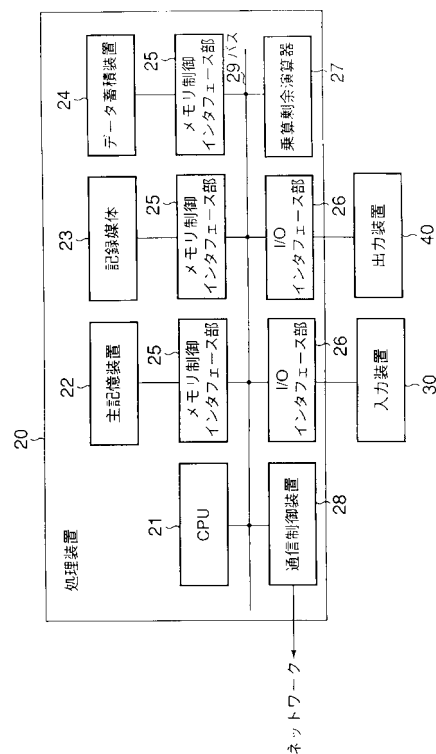
【図1】



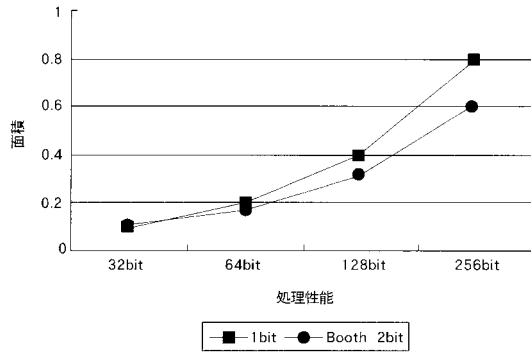
【図2】



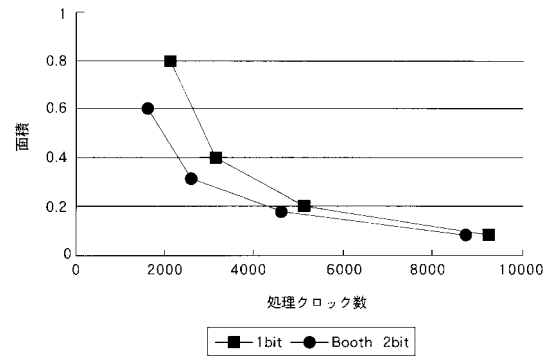
【図3】



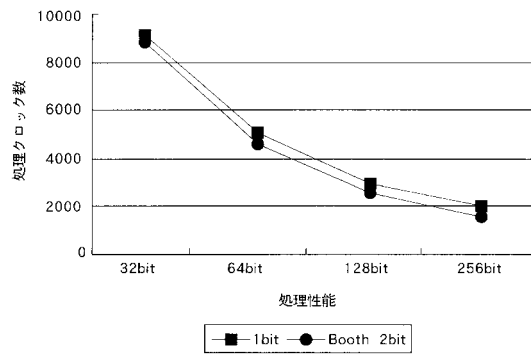
【図 4】



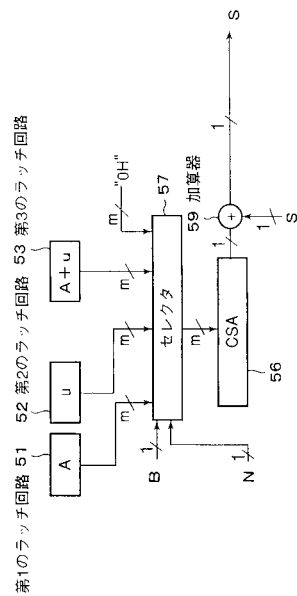
【図 6】



【図 5】



【図 7】



フロントページの続き

(72)発明者 東 邦彦

神奈川県川崎市中原区小杉町1丁目403番53 NECマイクロシステム株式会社内

(72)発明者 久門 亨

神奈川県川崎市中原区小杉町1丁目403番53 NECマイクロシステム株式会社内

(72)発明者 後藤 敏

東京都新宿区戸塚町一丁目104番地 学校法人 早稲田大学内

(72)発明者 池永 剛

東京都新宿区戸塚町一丁目104番地 学校法人 早稲田大学内

Fターム(参考) 5B016 AA01 AA02 BA06 BB01 EA07

5J104 AA18 AA22 JA21 NA18