小規模回路で実現する高速多倍長乗算方式の検証
Examination of Fast Multiple-Precision Multiplication Method Implemented on Small Circuits

情報工学専攻　久田 大貴
Information and System Engineering, HISADA Hiroki

1　はじめに

通信の傍受や情報の改竄を防いで安全に通信するための仕組みとして SSL/TLS (Secure Socket Layer/Transport Layer Security) [1] のような公開鍵暗号化方式, 秘密鍵暗号化方式および電子署名などを組み合わせた技術が利用されている. SSL/TLS で用いられる公開鍵暗号化方式のアルゴリズムとしては RSA (Rivest-Shamir-Adleman) 暗号が最も普及している暗号化アルゴリズムの 1 つとして知られている. RSA 暗号は暗号化および復号化処理に大きな桁数のべき剰余の演算をおこなう. このとき安全性の観点から鍵のサイズには 1024 ビット以上のものが用いられており, 演算の処理負荷が大きく, 処理時間も長期化する. そのため RSA 暗号の演算は主にソフトウェア上で実装されてきた. これをハードウェア上で実装する際にブースの乗算アルゴリズムまたはモンゴメリの乗算アルゴリズムを適用して, べき剰余の演算を高速化する技術が知られている [2][3]. 例えば, モンゴメリのアルゴリズムを用いることでべき剰余の演算をシフト演算, 加算および乗算に置き換えて, 四則演算の中で最も時間のかかる除算を回避し一定の高速化を測ることができる. ここから更に高速化を測るためには次に演算処理のボトルネックとなる乗算を高速化する必要がある. ハードウェアにおける乗算の高速化には並列乗算回路などが知られているが, この回路は演算する桁数が増加するにつれて急激に回路規模が増大してしまうので, 1024 ビットを超えるような大規模な乗算に使用する際には生産コストなどの観点から何らかの工夫が必要である.

そこで本研究では少ない回路規模で高速化が見込める工夫を用いた乗算器の開発を目指し, 加算器の桁上がり伝搬遅延を一定確率で削減するキャリー・トラッピングという現象 [4] を利用した多倍長加算器を用いて, 比較的小規模で高速な多倍長乗算器を構築する. 最終的には FPGA[5] 上での実装を目指し, Intel 社の FPGA 開発環境である Quartus にて回路の構築をおこなった.

2　論理回路における速度の指標

一般にデジタル装置を構成する論理回路は, 入力から出力までの間に挟まれる最長経路の論理ゲートの数 (段数) が多いほど伝搬遅延が発生し, 回路のレイテンシが大きくなる. また, 一度の演算をおこなうのに同じ回路を繰り返して用いるような装置の場合, 使用する回路のレイテンシが大きいと 1 サイクルの処理時間が増加するので, 装置のスループットが低下する. そのため論理回路の段数が少ないほど, 高速に動作する回路であると言える.

論理回路の例として半加算器 (Half Adder: HA) の回路図を図 1 に示す. 半加算器は 1 ビットの被加数 X と 1 ビットの加数 Y を入力として, 1 ビットの和 S と 1 ビットの上位桁への桁上がり伝搬 $Cout$ を出力する. この図では入力 X, Y から出力 $S, Cout$ までそれぞれ 1 つの論理ゲートを通過するので, この論理回路の段数は 1 である.

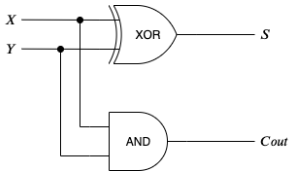

図 1　半加算器

論理回路のレイテンシやスループットに影響要素としては各論理ゲート間の配線による信号の伝搬遅延や, 順序回路であれば装置のクロック周波数など他にも様々な要因が存在する. 本研究では演算器を構成する際に, 論理ゲートの段数とクロック周波数に焦点を当てている.

3　加算器における桁上がり伝搬遅延

加算器は基本的に全加算器 (Full Adder: FA) という 1 ビットの加算をおこなう加算器の組み合わせで構築される．全加算器は半加算器 2 つと OR ゲートを図 2 のように繋いだものである．入力と出力は全て 1 ビットの値であり，被加数 X と加数 Y および下位桁からの桁上がり Cin を入力して，和 S と上位桁への桁上がり $Cout$ を出力する．半加算器の出力はどちらも 1 段なので，この全加算器の論理回路は 2 段である．以降は回路の段数の具体的な値について触れることはしないが，この考えを基に回路図の伝搬遅延について論じる．

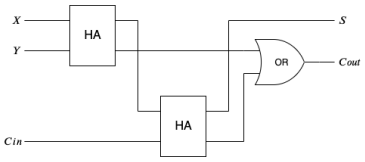

図 2　全加算器

ここで 1 以上の自然数 n について，n 個の全加算器を接続して n ビットの加算をおこなう複数ビット加算器を構築する．図 3 は n=4 とした際の複数ビット加算器であり，このような構成を桁上がり伝搬加算器 (Ripple Carry Adder: RCA) と呼ぶ．各桁の全加算器は，被加数 X と加数 Y の対応する桁ビットの値と下位桁の全加算器からの桁上がり信号を受け取り，和 S の対応する桁ビットの値と上位桁の全加算器への桁上がり信号を出力する．

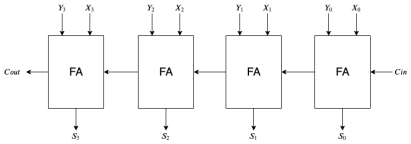

図 3　4 ビット桁上げ伝搬加算器

このような構成の桁上げ伝搬加算器で最も長い経路は，最下位ビットへの桁上がり入力から最上位ビットまでの桁上がり伝搬が最長経路であるので，高速な加算器を構築するには桁上がり伝搬の扱いが課題である．

4　乗算器の構築

乗算器は筆算の仕組みを基に，加算器を複数回利用して部分積を繰り返し足し合わせることで積を算出する．2 進数の筆算は 10 進数の筆算と同様におこなうことができる．n ビットの被乗数 $X = (x_{n-1} \cdots x_0)_2$ と乗数 $Y = (y_{n-1} \cdots y_0)_2$ について，n=4 の場合に被乗数 $X = (1011)_2$ と乗算 $Y = (1001)_2$ の 2 進数の筆算でおこなう様子を図 4 に示す．

1 0 1 1 : X = $(11)_{10}$
× 1 0 0 1 : Y = $(9)_{10}$

1 0 1 1	部分積 (X * y_0)
0 0 0 0 0	部分積 (X * y_1)
0 0 0 0 0 0	部分積 (X * y_2)
+ 1 1 0 1 0 0 0	部分積 (X * y_3)
0 1 1 0 0 0 1 1 : Z = $(99)_{10}$	

※ゼロパディング

図 4　2 進数 4 ビットの乗算

1 段目の部分積 $(1011)_2$ は被乗数 $X = (1011)_2$ と乗数 Y の 1 桁目のビット $y_0 = 1$ の AND 演算を取ったものであると分かる．2 段目の部分積 $(00000)_2$ は同様に被乗数 $X = (1011)_2$ と乗数 Y の 2 桁目のビット $y_1 = 2$ の AND 演算を取ったものを，1 桁分だけ上位桁にシフトしたものであると分かる．同様に 3 段目の部分積 $(000000)_2$ は同様に被乗数 $X = (1011)_2$ と乗数 Y の 3 桁目のビット $y_2 = 0$ の AND 演算を取ったものを，2 桁分だけ上位桁にシフトしたものである．最後に，4 段目の部分積 $(1101000)_2$ は同様に被乗数 $X = (1011)_2$ と乗数 Y の 4 桁目のビット $y_3 = 1$ の AND 演算を取ったものを，3 桁分だけ上位桁にシフトしたものである．この 4 つの部分和を全て足し合わせることで，最終的な積 $P = (01100011)_2$ を得られる．

この手順を一般化すると，n ビットの乗算は被乗数 X と乗数 Y の k 桁目のビットの AND 演算を取ったものを (k-1) 桁分だけ上位桁にシフトしたものを，k を 1 から n まで動かしながら足し合わせていくことで最終的な積を得られるということである．これを論理回路として考えると，複数ビット乗算器は AND ゲートと対応する複数ビット加算器，そしてシフタを組み合わせることで構築できると分かる．

この手順を一度におこなう乗算器の論理回路を，n=4 として図 5 および図 6 に示す．図 5 は被乗数 X と乗数 Y の各桁ビットの AND 演算をおこない，部分積 PP_0 から PP_3 までを作成する部分積生成器である．図 6 は

加算器を繋ぎ合わせ, 部分積 PP_0 から PP_3 までを足し合わせていく論理回路である. 部分積生成から加算までは並列に動作していくので, この構成は並列型乗算器と呼ばれている.

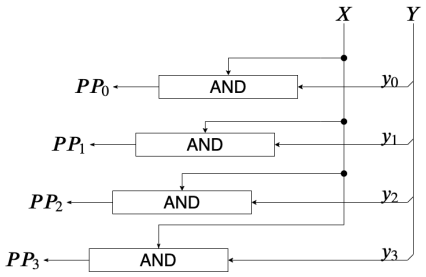

図 5　並列部分積生成器

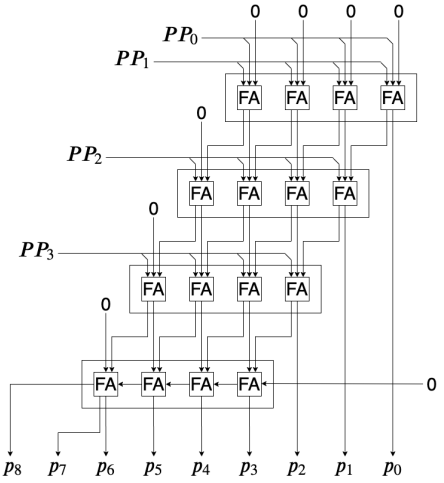

図 6　並列型乗算器

並列型乗算器は被乗数 X と乗数 Y を入力した瞬間から値が回路全体に伝搬していくので, 論理ゲートや配線による信号の伝搬遅延のみが回路におけるレイテンシとなる. そのため並列型乗算器は高速に動作し有用だが, 桁数が増加するにつれて回路規模が急激に増大してしまう.

回路規模を抑えつつ乗算をおこなう乗算器として逐次型乗算器が知られている. 逐次型乗算器は計算途中の部分積の和を保存するシフトレジスタを追加することなどにより, 1 つの加算器を繰り返して利用しながら生成した部分積を加算していく. 逐次型乗算器の回路図を, 図 7 に示す. この回路は一定周期で 0 と 1 を交互に繰り返すクロック信号を必要とし, クロック信号が 0 から 1 に切り替わるタイミングをクロックの立ち上がりと呼ぶ. 逐次型乗算器はクロックの立ち上がりのタイミングでシフトレジスタを更新し, 1 つの部分積を足し合わせる. 積が確定するのは演算が始まってから桁数クロック目である.

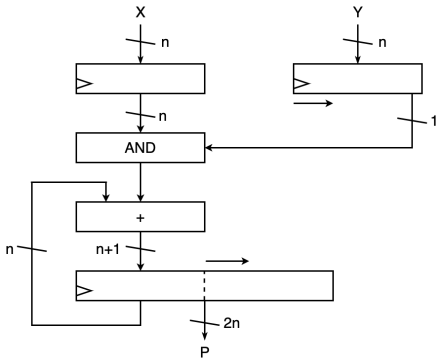

図 7　n ビット逐次型乗算器

5　キャリー・トラッピングを利用した多倍長演算器

5.1　キャリー・トラッピング

複数ビット列の加算において, ある桁 T で桁上がりが発生すると, それより下の桁からの桁上がりがその桁 T で停止し, その桁 T を超えて伝搬しなくなる現象をキャリー・トラッピングと呼ぶ [4]. また, 加算の結果 0 になるビットのことをキャリー・トラップと呼ぶ. 複数ビット列の加算においてキャリー・トラップが存在すれば, それよりも下位からの桁上がりは, その桁に捕捉されてそれよりも上位の桁上がりに影響を及ぼさない事になる.

5.2　多倍長加算器のシミュレーション

複数の加算器を接続し, 任意の桁数の加算器を生成する多倍長加算器を, FPGA 向け論理回路の開発に使われるハードウェア記述言語である verilog HDL を用いてモジュールとして設計した. この多倍長加算器はキャリー・トラッピングを利用して, 一定確率で途中の桁上がり伝搬を下位桁の加算が確定するよりも事前に確定する機能を備えている. この仕組みにより, 一定確率で大規模桁数の加算器の長い桁上がり伝搬を省略して, 高速に演算を確定することができる. 生成したモジュールをシミュレータ上で動かし, その挙動を確認した.

5.3　多倍長乗算器のシミュレーション

キャリー・トラッピングを利用して高速化した多倍長加算器を逐次型乗算器に用いることで, 同様に一定確率

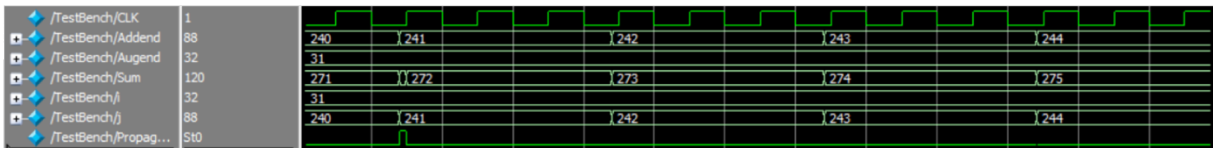

図 8　多倍長加算器のシミュレーション

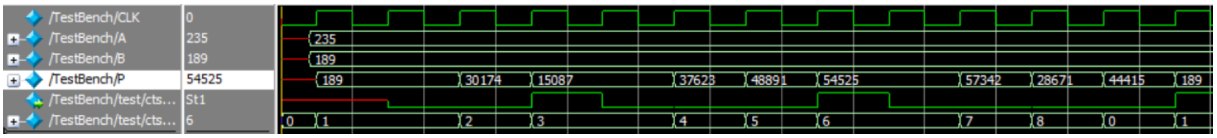

図 9　多倍長乗算器のシミュレーション

で高速に演算を確定できる多倍長演算器を設計する．この多倍長乗算器は n ビットを計算するのに n ビットに近いクロック数で演算を確定できる．通常の逐次型乗算器と比較すると必要クロック数は僅かに増加するものの, 回路の平均レイテンシ下げることで 1 クロックの間隔を短くすることができ, 結果的に乗算のスループットが増加する．これを verilog HDL によりモジュール化し, シミュレータでその挙動を確認した．実際に正しく高速に計算できることを確認した．

6　おわりに

キャリー・トラッピングを利用して演算をおこなう加算器および乗算器は, 演算に必要なクロック数が増加するので一見演算時間が増加するように思える．しかし実際には, 加算器部分の演算が完了するまでに必要な桁上がり伝搬時間が連結した加算器の個数分の 1 に近似されるので, 回路全体のクロックの動作間隔を短くすることができる．このため例に挙げた RSA 暗号に用いられる 1024 ビット以上の加算や乗算をおこなうような, 演算器を複数連結した演算器においてはクロックの動作間隔が非常に短くなると考えられるので効果的な高速化が期待できる．

キャリー・トラッピングを利用した多倍長加算器は, 加算に必要な時間を固定せず確率的に変動する仕組みを作り, 加算器を含む回路全体のクロック周波数を上げることで加算器のスループットの向上を図ったものである．この多倍長加算器を逐次型加算器に用いると同様に乗算に必要な時間が確率的に変動し, 乗算器を含む回路全体のクロック周波数を上げることで乗算器のスループットの向上を図ることができる．今後は実際の配線での伝搬遅延時間やクロック周波数を考慮して開発を進める必要があるだろう．また今回は n ビットコア加算器の内部の加算器に桁上がり伝搬加算器を用いて設計したが, 一定規模以下であれば代わりに桁上がり先見加算器を利用することなどでさらなる高速化も考えられる．一方でキャリー・トラッピングを利用する工夫は, 高速化を測りつつ少なからず回路規模の増加を招く．実際に活用する際にはどの程度規模が増加するのかを調べ, 利用できる物理的資源と相談する必要があるだろう．

参考文献

[1] Thomas Stephen, “SSL and TLS essentials,” New York, Mar. 2000.

[2] Jasbir Kaur, “Design and Implementation of an efficient Modified Booth Multiplier using VHDL,” International Journal of Advances in Engineering Sciences, Vol. 3, No. 3, pp. 78–81, 2013.

[3] Peter Montgomery, “Modular Multiplication Without Trial Division,” Math. Computation, Vol. 44, pp. 519–521, 1985.

[4] 鈴木寿, 島田道雄, “多倍長演算装置, 暗号化装置, 復号装置, 演算方法, プログラムおよび記録媒体,” 公開特許公報 特願 2010-091855, Apr. 13, 2010.

[5] 天野 英晴, *FPGA* の原理と構成, オーム社, Apr. 2016.