FFT多倍長乗算器のVLSI設計

矢崎 俊志 *　阿部 公輝 *

∗ 電気通信大学電気通信学研究科情報工学専攻

VLSI Design of FFT Multi-digit Multiplier

Syunji Yazaki*　Kôki Abe*

∗ Department of Computer Science,
Graduate School of Electro-Communications,
The University of Electro-Communications

Abstract. We designed a VLSI (Very Large Scale Integrated circuit) chip of FFT multiplier using a floating-point representation with optimal data length based on an experimental error analysis. Using the hardware implementation, we can perform 2^5 to 2^{13} hexadecimal digit (39 to 9,831 decimal digit) multiplication 25.1 to 45.6 times (33.9 times in average) faster than using FFTW3, at an area cost of 9.05 mm^2. The hardware FFT multiplier has 64 times faster performance than exflib (a multi-digit arithmetic library using Karatsuba method) for longer than 2^{21} hexadecimal digit ($\fallingdotseq$2,520,000 decimal digit) multiplication. Considering the wide applications of its FFT modules, the performance and cost of the FFT multiplier justifies the VLSI implementation.

1　はじめに

近年，数値計算 [6]，素数判定 [1]，カオス計算 [16, 17] など，様々な分野で高速な多倍長演算へのニーズが高まりつつある．中でも乗算は特に重要であり，過去に Karatsuba 法や Toom-Cook 法などの高速な乗算法が考案されている．これらの中に FFT を用いた乗算法がある．Karatsuba 法や Toom-Cook 法などの乗算法が $O(m^{1.585})$ や $O(m^{1.465})$(m は桁数) の計算時間を必要とするのに対し，FFT 乗算は，$O(m \cdot \log m \cdot \log\log m)$ 程度の時間で乗算を行うことができる．FFT は乗算だけでなく，DSP をはじめとする広い分野で利用されており，アルゴリズムの改良が盛んに行われている [7, 11]．このようなアルゴリズムをそのまま応用することができるのも FFT 乗算の利点の一つである．しかし，FFT では浮動小数点演算などの実数演算が必要になるため，他の乗算法と比較して演算のオーバヘッドが大きい．さらに，実数演算による誤差の問題にも注意が必要である．

一方，乗算を高速化するための方法として，VLSI (Very Large Scale Integrated circuit) を用いたハードウェア実装がある．一般に，ハードウェア化には多くのコストを必要とするが，性能向上の度合が大きいため，適切な場面で利用すれば非常に有効な方法である．FFT は広い分野で使われているため，ハードウェア実装に関する研究は広く行われている [2, 14]．これらの FFT ハードウェアを利用して乗算を行うことができれば，Karatsuba 法などを利用した乗算専用ハードウェアを新たに実装することなく，乗算を高速化できる．しかし，従来の FFT ハードウェアは，乗算に利用することが考慮されていないため，そのまま乗算に用いることはできない．

本論文では，DSP などへの転用も視野に入れ，FFT による乗算の VLSI 設計について述べる．まず，FFT ハードウェアを乗算に利用するための条件，回路構成，および必要となる演算器を明らかにする．次に，ソフトウェアとの比較による性能評価と，実装面積コストを調べ，多倍長乗算を FFT ハードウェアで行うことの有用性を性能向上と面積コストの両面から議論する．

本論文では，まず第 2 章で FFT 乗算アルゴリズムについて説明する．次に第 3 章で，FFT 乗算の誤差について述べ，第 4 章で必要となるハードウェアの構成を示す．さらに，第 5 章で，誤差解析に基づいて計算桁数に対する最小のデータ長を求める．その結果に基づき VLSI 実装を行い，第 6 章で評価と考察を行う．最後に，第 7 章でまとめる．

2　FFT 乗算アルゴリズム

r 進数 m 桁の整数 u，v を，虚数部がすべて 0 の $2m$ 次元複素ベクトル $u = (0 \dots 0 a_{m-1} \dots a_0)_r$，$v = (0 \dots 0 b_{m-1} \dots b_0)_r$ で表す．ただし，$2m = 2^k$(k は正整数) とする．u と v の積を $h = u \cdot v = (c_{2m-1} \dots c_0)_r$ とし，関数 $F(\cdot)$ を FFT 順変換，関数 $F^{-1}(\cdot)$ を FFT 逆変換とすると，u，v，h の間に式 (2.1) の関係が成り立つ [13]．ここに，$\otimes$ はベクトルの要素ごとの乗算を表す．

$$h = F^{-1}\left(F(u) \otimes F(v)\right)/2m \tag{2.1}$$

すなわち，整数 m 桁の乗数と被乗数が与えられたとき，それぞれをすべての虚数部が 0 の m 次元複素ベクトルとみなす．さらに，上位を m 個の 0 でパディングし $2m$ 次元複素ベクトルを作り，それぞれに対して FFT を行う．得られた 2 つの結果を要素ごとに乗算し，FFT 逆変換を行うと積が得られる．ただし，この時点の積は，各桁が $2m$ 倍された複素数で表現されていて桁上げも行われていない．そこで，各桁を $2m$ で割り，基数 r として桁上げの処理を行うことで最終的な積を得る．実際の演算では誤差が発生するため，整数への丸めが必要になる．

3　FFT 乗算の誤差

FFT 乗算は，実数の演算を必要とするため，誤差が発生，拡大する．この誤差が蓄積し，最終的に各桁の値を整数に丸める際に，絶対誤差が 10 進数で ± 0.5 を超えると，値が別の整数に丸められてしまう．そこで，FFT 乗算を利用する前にこの誤差の振る舞いを調べた．

FFT 乗算の誤差見積もりは，Henrici によって解析的に行われている [9]．これによると，マシンイプシロンが ϵ_M の計算機上で，桁数 m，基数 r の FFT 乗算を行った場合，正しい積を得るには，式 (3.1) を満足する必要があるとされている．

$$\epsilon_M \leq 1/192m^2(2\log_2 m + 7)r^2 \tag{3.1}$$

一方，平山により実験的な誤差解析も行われている [8]．平山は，Henrici が行った解析的な誤差の見積もりは最悪ケースであり，実際の計算では，誤差の条件はもっと緩いことを指摘し，数値実験でこれを確認している．IEEE754 の 64bit 表現を用いた場合を例にあげると，仮数部長は 52bit であるので，マシンイプシロンは $\epsilon_M \fallingdotseq 2.220446 \times 10^{-16}$ である．これを式 (3.1) で評価した場合，10 進数 77,091 桁まで計算可能であるという結果が得られるのに対し，実際には 100 万桁以上の演算が可能であることが文献 [8] で示されている．

また，平山の実験では，IEEE754 や IBM 形式など，広く一般的に利用されている浮動小数点表現を用いて，基数 r と桁数 m を変化させながら FFT 乗算を行い，正しい積を得ることができる r と m の限界を測定している．この実験において，r 進数 m 桁で表現できるすべての値の組み合わせで正しい積が得られることを保証する必要があるが，演算する値は多倍長であるため，すべての組み合わせの誤差を測定することは現実的ではない．そこで，最大の誤差が発生する組み合わせで正しい結果を得ることで，すべての組合せで正しい積が得られることを保証している．この最大の誤差が発生する値の組み合わせに関して，平山は式 (3.1) を誤差の評価式とみなし，基数 r が大きいほど要求される精度が高くなるという点から，すべての桁が $r-1$ となるような値，すなわち，最大値どうしの乗算が最大の誤差を与えるとしている．

このことを確認するために実験を行った．実験は乗数 $u = (\overbrace{0\ldots0}^{m}\overbrace{a\ldots a}^{m})_{16}$，被乗数 $v = (\overbrace{0\ldots0}^{m}\overbrace{b\ldots b}^{m})_{16}$ の a，b を 0 から F まで変化させた値の組合せで FFT 乗算を行い誤差を測定するものである．FFT には最も基本的な Cooley-Tukey FFT[4]，データ表現には IEEE754 の倍精度 (double) を用いた．結果を Fig. 1 に示す．グラフの x 軸は a，y 軸は b，z 軸は $u \times v$ における絶対誤差を示す．ただし，誤差の値は 10 進数値である．グラフは桁数 m が 1,024 桁と 2,048 桁の場合を示している．図から，$u = (0\ldots0\mathrm{F}\ldots\mathrm{F})_{16}$，$v = (0\ldots0\mathrm{F}\ldots\mathrm{F})_{16}$ の場合に最大の誤差が発生していることが確認できる．4,096 桁，8,192 桁についても測定を行った結果，同様の結果が得られた．これにより，最大値どう

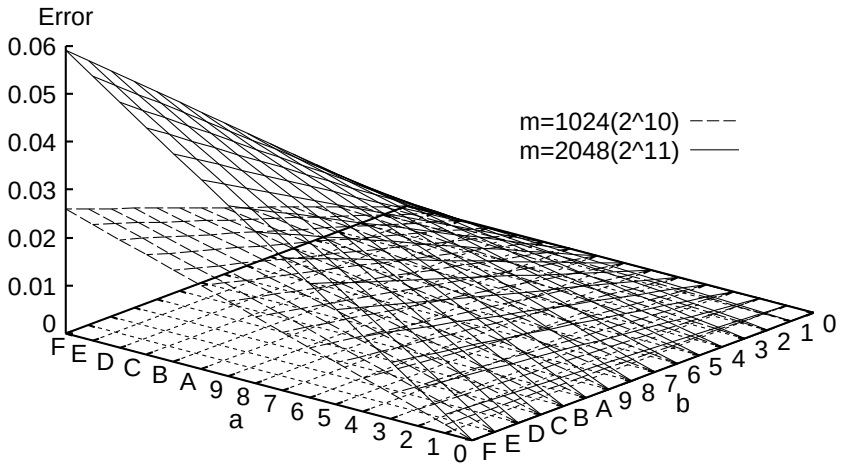

Figure 1: Errors in multiplying $(0^m a^m)_{16}$ and $(0^m b^m)_{16}$ using FFT.

しの乗算において，最大の誤差が発生することが確認できた．したがって，FFT 乗算を行う場合は，最大値どうしの乗算で正しい結果が得られるような精度で演算を行えばよいことが確認できた．

4 FFT 乗算器の構成

FFT 乗算を行うために必要となるモジュール構成を Fig. 2 に示す．各モジュールの詳細

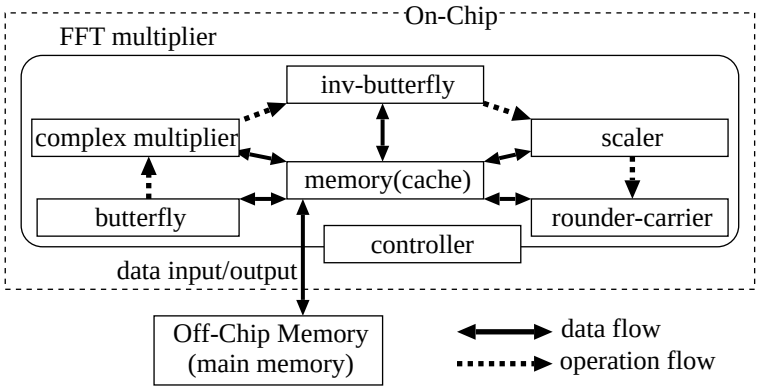

Figure 2: Structure of FFT multiplier

な説明は後で述べる．図中の実線矢印はデータの流れを示している．演算は，第 2 章で述べた FFT 乗算アルゴリズムにしたがい，図中の破線矢印が示す順番で行われる．また，実数表現には広い範囲の実数値を表現することができる浮動小数点表現を用いた．この時，仮数部と指数部の基数は 2 である．全体として，各モジュールが 1 個のメモリを共

有し，メモリからデータを読み出し，演算をして再びメモリに書き戻すという動作の繰り返しで演算が行われる．

4.1 complex multiplier

complex multiplier モジュールは，複素数 $s_r + s_i i$，$t_r + t_i i$ が与えられた時，式 (4.1) にしたがって複素数乗算を行う．ただし，i は虚数単位とする．一般に乗算器より加減算器の方がハードウェアの実装コストが低いため，式 (4.1) に示す式変形を行い加減算の回数を増やすかわりに乗算の数を削減している．

$$\begin{aligned}(4.1)\quad p_r + p_i i = (s_r + s_i i)(t_r + t_i i) &= s_r t_r - s_i t_i + (s_r t_i + s_i t_r) i \\ &= s_r t_r - s_i t_i + ((s_r + s_i)(t_r + t_i) - (s_r t_r + s_i t_i)) i\end{aligned}$$

このモジュールには浮動小数点加減算器が 5 個と浮動小数点乗算器が 3 個必要である．

4.2 butterfly， inv-butterfly

butterfly モジュールは FFT の順変換に，inv-butterfly モジュールは逆変換に用いられるバタフライ演算器である．バタフライ演算の内容は FFT の種類によって変わる．今回は，演算器の単純化を考え，第 3 章の誤差測定実験と同じ Cooley-Tukey FFT を用いた． Cooley-Tukey FFT の順変換を行うためのバタフライ演算は，変換前の値を x，y，変換後の値を X，Y とすると，$X = x + yW$，$Y = x - yW$ で表される．ただし，x，y，X，Y，W は複素数である．W は FFT における回転子を表し，1 の原始 $2m$ 乗根の 1 つである．逆変換のバタフライ演算は，変換前の値を X'，Y'，変換後の値を x'，y' とすると，$x' = X' + Y'$，$y' = (X' - Y')W$ で表される．x'，y'，X'，Y' も複素数である．butterfly と inv-butterfly には複素数乗算器 (complex multiplier モジュール)1 個と浮動小数点加減算器 4 個がそれぞれ必要である．

4.3 scaler

scaler は逆変換後の各桁を $2m$ で除するスケーリングを行うモジュールである．ただし $2m = 2^k$(k は正整数) なので，実際に除算を行う必要はなく，浮動小数点表現の指数部から k を減ずればよい．また，FFT の対称性より，ここで扱う値は，実数ベクトルをフーリエ変換しそれを再び逆変換したものであるので，必ず実数ベクトルである．計算機上での実数演算では誤差によって虚数部に 0 でない値が現れる場合がある，この虚数部の値はすべて誤差であるので，これ以降は虚数部を無視して実数ベクトルとして扱ってよい．したがって，指数部を減ずるための数 bit の単純な減算器 1 個で実現することができる．

4.4 rounder-carrier

rounder-carrier モジュールは，スケーリング後，実数ベクトルで表現された積の各桁を丸めにより整数に変換した後，桁上げを行う演算器である．浮動小数点表現の指数部と仮数部の基数は 2 なので，小数点以下 1 桁目を 0 捨 1 入することで整数に丸める．これにより，絶対誤差が 10 進数で ± 0.5 を超えない限りは正しい結果を得ることができる．その後，r 進数の桁上げを行う．

この演算には，各桁の値を丸めて各桁の整数値と桁上がりを決定するためのマルチプレクサと加算器が 1 個づつ，下位桁からの桁上げを加算するための浮動小数点加算器が 1 個必要である．

4.5 memory

多倍長演算を行う場合，演算の対象となるデータを格納する記憶領域は巨大なものとなる．したがって，データ全体はチップ外部の大規模な記憶装置に格納し，演算に必要なデータのみをチップ内の記憶装置にキャッシュする必要がある．memory モジュールはこのキャッシュとして機能する．また，演算のパイプライン化を考えた場合，同一のメモリに対する同時読み書きによる，構造ハザードが発生する場合がある．これを回避するために，memory モジュールには複数の Read，Write ポートを用意する必要がある．

また，FFT を行うためには回転子 W を計算する必要がある．これをハードウェアで行うと多くのコストが必要となるため，W はあらかじめ計算し，テーブルとしてメモリに書き込んでおく必要がある．このテーブルも，全体はチップの外に持ち，必要な部分だけを memory モジュールに読み込んで使用する．

4.6 controller

FFT 乗算器を構成する各モジュールを制御するモジュールが controller である．各モジュールや memory モジュールへ適切なタイミングで制御信号を送信することで FFT 乗算器全体を制御する．また，チップ外部とのデータのやりとりも制御する．

5 最小のデータ長

FFT 乗算では，ある程度大きな範囲の実数値を扱う．そのため，この実数を表現するために，浮動小数点表現を用いる必要がある．しかし，浮動小数点演算器をハードウェア実装した場合，一般に回路規模は大きくなりコストの増大と性能の低下につながる．そのため，浮動小数点演算器の規模を削減するような工夫が必要となる．

回路規模の削減法として効果的なのは，扱うデータのビット長を削減することである．そこで，浮動小数点表現の指数部，仮数部の長さを削減することを考える．指数部に関しては，演算中にオーバフローを起こさない最小の長さまで削減できる．一方，前

述のように，FFT 乗算は演算の精度によっては正しい演算が行われないため，仮数部の最小ビット長は誤差解析によって必要最低限の精度を求め，それに基づいて決定する必要がある．演算に要求される精度は乗算桁数に依存する．言い換えると，乗算桁数が決まれば正しい演算を保証する最低限の精度が定まり，必要な仮数部長を見積もることができる．そこで，第 3 章の結果を利用し，乗算桁数に対して要求される最小仮数部長を見積もる．

まず，Henrici による解析的な誤差見積もりの式 (3.1) に基づいて最小の仮数部長を考える．データ表現の精度を表すマシンイプシロン ϵ_M は，仮数部長を f とすると，$\epsilon_M = 1/2^f$ と表すことができる．これを，式 (3.1) に適用すると，仮数部長 f，桁数 m，基数 r の関係が次式のように得られ，これから，m と r が与えられた場合の仮数部長を求めることができる．

$$1/2^f = 1/192m^2(2\log_2 m + 7)r^2 \tag{5.1}$$

しかしながら，式 (5.1) で求められる仮数部長は最悪ケースの誤差計算に基づくものである．そこで，実験的な誤差解析に基づいて，より最適な仮数部長を求めた．

実験は，仮数部と指数部のビット長を自由に変更できる浮動小数点表現を定義し，乗算桁数 m を変化させながら，m 桁で表現できる最大値どうしの乗算 (FF..FF×FF..FF) を行い，正しい結果が得られる最小の仮数部長を測定するものである．測定において，基数 r は 16 とした．また，仮数部には MSB の 1 を省略する正規化数を用い，非正規化数は扱わないものとした．これらはハードウェア実装と同じ仕様である．結果を Fig. 3 に示す．グラフの横軸は桁数の対数，縦軸はビット長を表している．× 点は乗算桁数に対

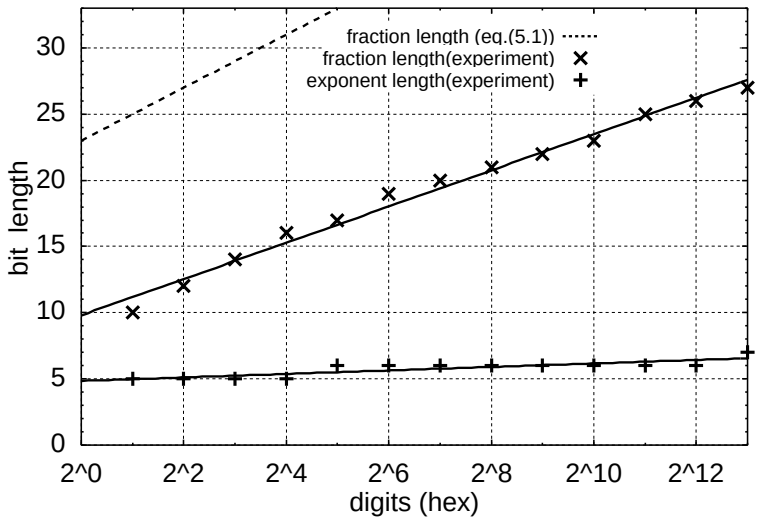

Figure 3: Fraction and exponent bit lengths required for multiplying two multi-digit numbers vs. the number of digits ($r = 16$).

する最小の仮数部長を実験により測定した結果を示し，＋点は演算中にオーバフローしない最小指数部長を表している．また，点線は式 (5.1) を示している．

実験によって得られた仮数部長は，式 (5.1) から求めたものと比較してかなり短いことが確認できる．また，2^{13} 桁 (10 進数 9,831 桁) どうしの乗算を行う場合を例にすると，演算に必要なビット長は，指数部が 7bit，仮数部が 27bit である．したがって，例えば，この桁数の乗算を IEEE754 の 32bit 表現 (指数部 8bit，仮数部 23bit) を用いて行うと，精度が足りず間違った結果を出力してしまう．また，64bit 表現 (指数部 11bit，仮数部 52bit) を用いて行うと精度が余り，演算器の性能低下や面積増大をまねく．ここで示したように，演算に必要なビット長を測定した上で回路を設計することで，精度不足や回路規模増大などの問題を回避し，性能や面積を最適化した演算器を設計することができる．

6　実装結果と比較考察

第 5 章の結果に基づき，桁数に対する最小のデータ長で FFT 乗算器を実装し，その面積と性能を調べた．実装は，HDL(Hardware Description Language) の 1 つである Verilog-HDL を用いて動作レベルで記述した回路を，Synopsys 社の Design Compiler 2003.6-SP1 でゲートレベルまで自動論理合成するという方法で行った．また，評価は論理合成結果から得られた回路面積と，回路の性能を決定する遅延時間を用いて行った．論理合成の際には，論理ゲートをライブラリ化したセル・ライブラリを必要とするが，これには日立製作所の仕様を元に VDEC(VLSI Design and Education Center)[19] が作成した CMOS 0.18μm テクノロジのものを用いた．また，合成時の条件には性能を優先するような制約を与えた．

面積に関する評価には controller 以外のモジュールを個別に合成した結果の合計値を用いた．また，速度に関する評価にはこれらのモジュールの最大遅延時間を用いた．controller を評価から除いた理由は，面積と性能が，乗算桁数に対してほぼ一定であり，その他のモジュールと比較して面積や遅延時間が小さいためである．memory モジュールの容量は 8 エントリを保持できる大きさにした．1 エントリは，1 回の演算で最も多くのデータを必要とする butterfly モジュールでの演算 1 回分のデータをキャッシュするのに必要な大きさとした．

6.1　乗算桁数に対する面積と最大遅延時間

まず，FFT 乗算器の面積と乗算桁数の関係について調べた．Fig. 3 の結果をもとに，与えられた乗算桁数に対して最小のデータ長で FFT 乗算器を構成し，その面積を測定した．結果を Fig. 4 に示す．グラフの横軸は桁数の対数，縦軸は面積を表している．測定を行った最大の桁数である 2^{13} 桁どうしの乗算を例にすると，Fig. 3 より，最小の指数部長は 7bit，仮数部長は 27bit である．これに符号ビットを加えた 35bit が最小のデータ長である．この表現を用いた場合，Fig. 4 から，面積は約 6.54mm^2 になることがわか

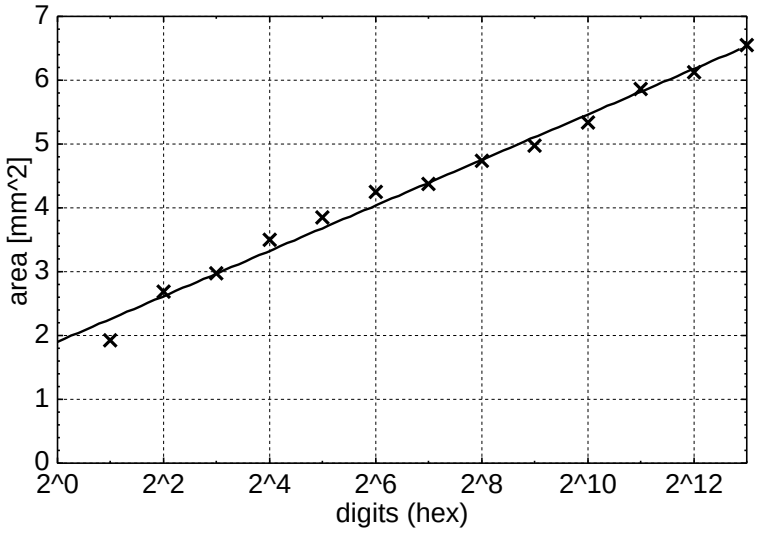

Figure 4: Area cost of FFT multiplier vs. number of digits ($r = 16$).

る．IEEE754 の 64bit 表現と同じく，符号ビット，指数部 11bit，仮数部 52bit からなるデータ表現を用いて本実験と同じ条件で面積を求めた結果，16.0mm^2 であった．(なお，文献 [20] で同様の実験結果が述べられているが，本実験とは論理合成の条件が異なる．) この結果から，最適なデータ長を用いて 2^{13} 桁どうしの乗算を行う FFT 乗算器を構成した場合，64bit の表現と比較して面積を約 60%削減できることがわかる．

同様に，乗算桁数と最大値遅延時間についても調べた．結果を Fig. 5 に示す．グラフの横軸は桁数の対数であり，縦軸は最大遅延時間を表している．グラフから，2^{13} 桁どうしの乗算を行う FFT 乗算器の最大遅延時間は 7.6ns であることがわかる．また，64bit 表現を用いた場合の最大遅延時間を測定したところ，10.3ns であった．このことから，最大遅延時間は約 26%削減できることがわかる．

6.2 パイプライン化による演算速度の向上

前節で評価した FFT 乗算器の浮動小数点加減算器や乗算器などのモジュールは，モジュール間にレジスタが挿入されているものの，モジュール内部はパイプライン化されていない．ここでは，モジュール内部のパイプライン化による速度向上を考察する．

パイプライン化にあたり，回路のクリティカルパスを調べたところ，仮数部を乗算するために浮動小数点乗算器内で使用されている Wallace 木乗算器にあった．適切なパイプライン化を行うためには，面積と性能のトレードオフを調べる必要がある．そこで，2^{13} 桁の場合において Wallace 木乗算器のパイプライン化を行い，浮動小数点乗算器全体の面積と最大遅延時間の変化を調べた．結果を Table 1 に示す．

表から，パイプライン段数が 4 段以上になると，遅延時間があまり変化していないこ

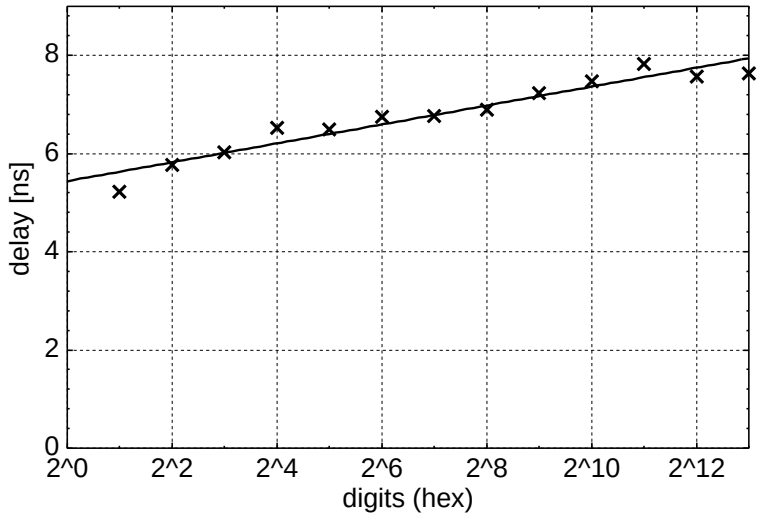

Figure 5: Critical path delay of FFT multiplier vs. number of digits ($r = 16$).

Table 1: Dependence of area cost and critical path on pipeline depth of Wallace-tree floating-point multiplier.

pipeline depth	area[mm^2]	delay[ns]
1	0.25	4.56
2	0.29	2.63
3	0.30	2.06
4	0.31	1.60
5	0.35	1.55
6	0.39	1.54

とがわかる．これは，Wallace 木乗算器に存在したクリティカルパスが，パイプライン化によって短くなり，浮動小数点乗算器内にある他のパスの長さと同じかそれ以下になったことを表している．このことから，浮動小数点乗算器は，4 段パイプラインの Wallace 木乗算器を用いることで，最も面積と遅延時間のバランスが良くなることがわかった．

その他のモジュールも，このときの浮動小数点乗算器の遅延時間にあわせてパイプライン化を行った．これを本 FFT 乗算器における最適なパイプライン化と呼ぶ．最適なパイプライン化を行った結果，各モジュールのパイプライン段数は Table 2 のようになった．ここに T_{fft}，T_{cmul}，T_{scl}，T_{rc} は，それぞれ (inv-)butterfly，complex multiplier，scaler，rounder-carrier のパイプライン段数を表す．

Table 2: Number of pipeline stages of FFT multiplier modules.

Pipeline	T_{fft}	T_{cmul}	T_{scl}	T_{rc}
Not optimized	9	7	2	4
Optimized	22	17	2	10

FFT 乗算器の演算速度は，演算にかかる遅延時間と総クロック数の積で決まる．総クロック数は次式で与えられる．

$$T_{\mathrm{fftmul}} = 3(T_{\mathrm{fft}} + m - 1)\log_2 m + (2T_{\mathrm{rc}} + 7)m + (3T_{\mathrm{fft}} + T_{\mathrm{cmul}} + T_{\mathrm{scl}} - T_{\mathrm{rc}} - 3) \quad (6.1)$$

各モジュールが memory モジュールへアクセスするための時間 (キャッシュレイテンシ) は，memory モジュールが同一チップ内に配置されているため短く，本設計では各モジュールでの演算とオーバラップさせているので，式 (6.1) には現れない．

Table 3 に最適なパイプライン化を行った後の各モジュールの面積と遅延時間をまとめる．表から，FFT 乗算器の最大遅延時間は 1.89ns であることがわかる．式 (6.1) より，2^{13} 桁における総クロック数は 541,563 であるから，演算時間は 1.02ms となる．パイプライン化前の総クロック数は 442,709，遅延時間は 7.63ns であるので，演算時間は 3.38ms である．最適なパイプライン化を行うことにより，約 3.3 倍の速度向上が得られた．この時，面積は 38%増加し，9.05mm^2 となった．この面積は，同世代の汎用プロセッサの面積 200mm^2 程度 [10] の約 5%に相当する．

Table 3: Area and critical path delay of the optimally pipelined FFT multiplier ($m = 2^{13}$).

module	area[mm^2]	delay[ns]
butterfly	2.98	1.81
inv-butterfly	3.06	1.81
complex mul	2.01	1.89
scaler	0.02	1.26
rounder-carrier	0.38	1.74
memory	0.60	1.60
all	9.05	1.89

2^{13} 桁以上の乗算についても以下のように演算時間を見積もることができる．Wallace 木乗算器は，桁上げ保存加算器 (Carry Save Adder:CSA) を数段重ねた構造をしており，その段数は乗算するビット数 (仮数部長) の対数に比例する．Fig. 3 からわかるように桁数が増加すると，仮数部も増加するが，Wallace 木の CSA 1 段がパイプライン 1 ステージになるようにパイプライン化することにより，遅延時間を一定 (CSA の遅延時間) に保つことができる．ただし，パイプライン段数を増やすことにより，演算に必要なクロック数は増加する．

例えば，2^{13} 桁の乗算では，仮数部は 27bit を必要とした．このビット幅の乗算を行うための Wallace 木の段数は 5 段である [18]．(実際には，パイプライン段数を 5 段にすると，回路全体のクリティカルパスが Wallace 木以外に移り，面積増加に対する速度向上が悪くなるので，最適なパイプライン化では 4 段にした．) 一方，2^{21} 桁 (10 進数約 252 万桁に相当) の乗算を行うために必要な仮数部長を Fig. 3 から外挿すると 39bit になる．この場合の Wallace 木の段数は 6 段である．よって，パイプライン段数を 1 段増やすことで，2^{13} 桁と同じ遅延時間を維持することができる．

以上より，ハードウェア FFT 乗算器で 2^{21} 桁どうしの乗算を行う場合，パイプライン段数を 1 段増やして Table 3 で示す 1.89ns の遅延時間を維持したと仮定すると，式 (6.1) から，1 回の乗算に必要となるクロック数は 192,939,645 と見積もることができる．これに 1.89ns を掛けると，演算時間は 0.365s となる．

6.3 ソフトウェア FFT 乗算との速度比較

前章の考察から，2^5 桁 (10 進数 39 桁) から 2^{13} 桁どうしの乗算に必要な演算時間を求め，ソフトウェアとの速度比較をおこなった．2^{13} 桁以下の FFT 乗算器は，2^{13} 桁の場合より小さい回路規模で実現できるため，遅延時間は 2^{13} 桁以下のすべての桁数で 1.89ns とした．

ソフトウェアを実行する計算機の条件として，CPU に本ハードウェア実装と同世代の 0.18μm プロセスを使用している Intel 社 [12] の Celeron 800MHz，OS に FreeBSD 4.8，コンパイラに gcc 2.95.4 を用いた．また，FFT には FFTW 3.0.1 を用いた．FFTW は様々な種類の FFT を複数の計算機アーキテクチャ用に最適化して実装した高速な FFT ライブラリである．今回は FFT の対象となるデータが整数ベクトルであるため，複素数 FFT と比較して速度や使用メモリ量の面で有利な実数 FFT を用いた．比較結果を Table 4 に示す．

ソフトウェアとハードウェアの速度倍率が一定でないのは，ソフトウェアで使われている FFTW が 1,000 から 10,000 点の FFT において高い性能を発揮するように実装されているからである [3]．表から速度倍率は，2^5 から 2^{13} の範囲で，25.1 倍から 45.6 倍，平均は約 33.9 倍であることがわかった．

さらに高速化を行う方法として，FFT に用いるバタフライ演算器を複数にして，並列処理を行うことが考えられる．FFT は in-place 演算であるため高い並列度を実現できる．FFT 乗算にバタフライ演算器を 2 個用いることで，並列度を 2 倍にすることができる．このとき，2^{13} 桁どうしのソフトウェア FFT 乗算と比較して 73 倍，2^{21} の演算においては，123 倍の性能が期待できる．ただしその場合，面積はバタフライ演算器 1 個分増加する．この増加量は，2^{13} 桁の乗算においては，全体の 34%である．

Table 4: Comparison of FFT multiplication performance of software and hardware implementations.

Digits	Soft[ms]	Hard[ms]	Soft/Hard
2^5	0.1504	0.0033	45.6
2^6	0.2236	0.0063	35.5
2^7	0.5049	0.0126	40.1
2^8	0.9193	0.0258	35.6
2^9	1.7650	0.0535	33.0
2^{10}	3.0130	0.1116	27.0
2^{11}	5.8560	0.2337	25.1
2^{12}	12.990	0.4893	26.6
2^{13}	37.560	1.0236	36.7
...	...	...	...
2^{21}	22360	364.66	61.3
...	...	...	...

6.4 ソフトウェア実装された他の演算法との比較

前節と同様の計算機環境において，Karatsuba 法を用いた高速な多倍長乗算を提供する exflib ライブラリ [5] とソフトウェア FFT 乗算の演算速度を比較した．exflib はバージョン 20050301 を用いた．結果を両対数グラフにして Fig. 6 に示す．exflib のグラフの傾き

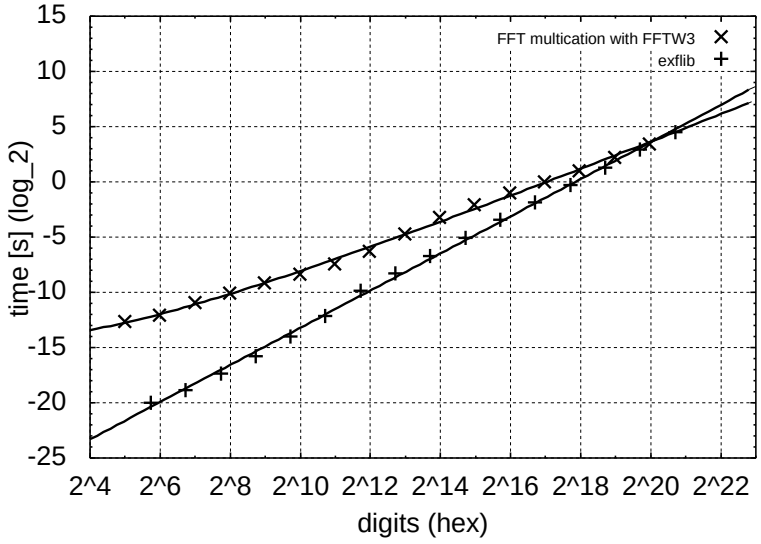

Figure 6: Multiplication performance of FFT and exflib.

は 1.68 であった．これは，Karatsuba 法の計算時間 $O(n^{1.585})$ に近い．また，グラフから約 2^{21} 桁以上で FFT 乗算が有利になることが読み取れる．この時の exflib の乗算時間は 23.2s であった．前節の結果からこの桁数でのハードウェア FFT 乗算器の演算時間は，

0.36s であることがわかっている．したがって，ハードウェア FFT 乗算器を用いた場合は，exflib より 64 倍高速である．並列度を 2 倍にした場合には，exflib を用いたソフトウェア乗算の 128 倍高速になる．

6.5　既存の FFT ハードウェアとの関連と他のハードウェア乗算器との比較

過去に発表されている FFT ハードウェアには様々な種類がある．これらは，バタフライ演算器とメモリを実装したものである．これらの実装は，データ長やデータ表現については，主な使用目的が DSP であるので，乗算に必要な精度の考慮がされていない．必要な精度を満たしていると仮定すれば，complex multiplier，scaler，rounder-carrier を追加することで乗算が可能となる．また，アーキテクチャによっては，バタフライ演算器で使われている複素数乗算器を流用することも考えられるので，その場合には complex multiplier は不要となる．

Fig. 6 より，2^{21} 桁以下では，FFT 乗算より，exflib の方が性能が良い．よって，exflib で使われている Karatsuba 法をハードウェア実装するという選択もある．ここでは，ハードウェア FFT 乗算器と 32bit ハードウェア Karatsuba 乗算器 [15] の面積コストを比較する．比較にあたり，広い応用範囲を持つ FFT ハードウェア (butterfly,inv-butterfly,complex multiplier) はすでに用意されているものとし，コストとしてカウントしない．32bit(16 進数 2^3 桁)FFT 乗算器の面積 (scaler と rounder-carrier モジュールの合計面積) が 0.15mm^2 であるのに対し，32bit Karatsuba 乗算器の面積は，0.18μm テクノロジに換算すると，最も小さいもので 0.10mm^2 である．すなわち，32bit においては，FFT 乗算器と Karatsuba 乗算器の面積コストは同程度と言える．より詳しい比較には 2^{21} 桁までのハードウェア Karatsuba 乗算器のコスト評価が必要である．

7　おわりに

本論文では，FFT 乗算で使用する浮動小数点表現の最適なデータ長を実験的な誤差解析に基づいて求め，最適な面積と遅延時間で FFT 乗算器を実装した．最適なデータ表現を求めるために，最大値どうしの乗算が最大の誤差を与えるという点に着目し，r 進数 m 桁における最大値どうしの乗算で正しい結果が得られるようなデータ長を測定し，そのデータ長でハードウェアを構成した．その結果，ハードウェア FFT 乗算は，同世代のチップテクノロジを用いた汎用プロセッサ上で実行したソフトウェア FFT 乗算と比較して，16 進数 2^5 桁から 2^{13} 桁 (10 進数 39 桁から 9,831 桁) で，25.1 倍から 45.6 倍，平均 33.9 倍の性能を実現できることがわかった．このときのチップ面積 9.05mm^2 は，同世代の汎用プロセッサの 5%程度である．ソフトウェア FFT 乗算が，exflib とほぼ同じ速度になる 16 進数 2^{21} 桁 (10 進数約 252 万桁) 以上の乗算において，ハードウェア FFT 乗算は，exflib と比較して 64 倍以上の性能を実現できる．さらに，演算の並列度を 2 倍にすることにより，34%の面積コスト増 (2^{13} 桁の場合) により，2 倍の性能が期待できる．

FFT 乗算器の主要部分である FFT ハードウェアがそのまま DSP などへ転用できることを考慮すると，ハードウェア化に要するコストは，得られる性能向上に見合うと考えられる．

今後の課題としては，FFT ハードウェアの並列度を高めるなどアーキテクチャ面での工夫，より洗練された FFT アルゴリズムの適用，アプリケーションへの適用とその有効性の検証，Karatsuba 法を利用したハードウェア多倍長乗算器との詳細な比較などが考えられる．

謝辞

本研究を進めるにあたり，有益な議論をしていただいた電気通信大学の山本野人助教授，楯岡孝道博士，清水雅一氏，東京大学の葛毅博士，東京工科大学の月江伸弘博士に感謝する．また，本研究の応用面での意義に関し，有益な示唆をいただいた電気通信大学の加古孝教授に感謝する．また，本論文の執筆にあたり，要所で的確なご指摘をして頂いた査読者の皆様に感謝する．なお，本研究は，東京大学大規模集積システム設計教育センターを通して，シノプシス株式会社とケイデンス株式会社の協力で行われたものである．また，日本学術振興会科学研究費補助金 (基盤研究 (C)(2)16500026) の助成をいただいた．ここに記して謝意を表す.

References

[1] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES Is in P*, http://www.cse.iitk.ac.in/, 2002.

[2] B. M. Baas, “A Low-Power, High-Performance 1024-Point FFT Processor,” *IEEE Journal of Solid-State Circuits*, Vol.34, No.3, Mar.1999, 380-387.

[3] benchFFT, http://www.fftw.org/benchfft/

[4] J. W. Cooley and J. W. Tukey, “An Algorithm for Machine Computation of the Complex Fourier Series,” *Mathematics of Computation*, Vol.19, Apr. 1965, 297-301.

[5] exflib - Extended Precision Float-Point Arithmetic Library, http://www-an.acs.i.kyoto-u.ac.jp/ fujiwara/exflib/exflib-index.html

[6] H. Fujiwara, “High-Accurate Numerical Method for Integral Equations of the First Kind under Multiple-Precision Arithmetic” *Theoretical and Applied Mechanics Japan*, Vol. 52, 193-203, 2003.

[7] M. Frigo, “A Fast Fourier Transform Compiler,” *ACM SIGPLAN Notices*, Vol. 39, Issue 4, Apr. 2004, 642-655.

[8] 平山 弘, “FFT による高精度数の乗算,” 情報処理学会 研究報告 *High Performance Computing*, Vol.65, No.65-6, 1997, 27-32.

[9] P. Henrici, *Applied and Computational Complex Analysis*, Vol.3, Chap.13, John Wiley & Sons, NY, 1986.

[10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, 3rd Edition*, Morgan Kaufmann, CA, 2003.

[11] J. van der Hoeven, “The Truncated Fourier Transform and Applications,” *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, 2004, 290-296.

[12] Intel Co., http://www.intel.com

[13] D. E. Knuth, *The Art of Computer Programming, Volume 2, 2nd Edition : Seminumerical Algorithms*, Addison-Wesley, MA, 1981.

[14] N. Miyamoto, L. Karnan, K Maruo, K. Kotani, and T. Ohmi, “A Small-Area High-Performance 512-Point 2-Dimensional FFT Single-Chip Processor,” *Proceedings of the 2004 Conference on Asia South Pacific Design Automation : Electronic Design and Solution Fair 2004*, Jan. 2004, 537-538.

[15] 柴岡雅之, 高木直史, 高木一義, “Karatsuba アルゴリズムに基づく小面積並列乗算”, 電子情報通信学会 2005 年総合大会講演論文集, Vol.A-3, Mar. 2005, 66.

[16] 夛田光輝, 上島直樹, 茶谷芳裕, 鎌田弘之, “カオスシステムにおける有限桁演算の影響度の評価について,” 第 *17*回 回路とシステム軽井沢ワークショップ論文集, 2004, 13-18.

[17] 月江伸弘, 小澤智, “カオス方程式の数値解における有効桁数の減少,” 情報処理学会論文誌, Vol.44, No.11, 2003, 2778-2786.

[18] C. S. Wallace, “A Suggestion for a Fast Multiplier,” *IEEE Trans. Electronic Computers*, Vol.EC-13, No.2, Feb, 1964, 14-17.

[19] VLSI Design and Education Center Homepage, http://www.vdec.u-tokyo.ac.jp

[20] 矢崎俊志, 阿部公輝, “高速 Fourier 変換を用いた多倍長乗算器の設計と評価および VLSI への実装,” 電子情報通信学会技術報告, Vol.103, No.476, Nov. 2003, 253-258.

矢崎俊志 (正会員) 〒 359-0025 埼玉県所沢市上安松 220-5．2004 年，電気通信大学博士前期課程修了．現在，同大学博士後期課程在学．多倍長演算とハードウェアの研究に従事．

阿部 公輝 (非会員) 〒 182-8585 調布市調布ヶ丘 1-5-1．1974 年，東京大学大学院理学研究科博士課程単位取得退学．同年，電気通信大学電子計算機学科助手．現在，同大学助教授．理学博士．コンピュータアーキテクチャ，算術演算回路，コンピュータネットワークなどの研究に従事．電子情報通信学会，情報処理学会，電気学会，IEEE 各会員．