A-07

高位合成技術を用いた回路高速化の一手法

High Speed Circuit Design Using High-level Synthesis Technology

北村 友一†　　長田 安弘††　　神戸 尚志‡
Yuichi Kitamura　Yasuhiro Nagata　Takashi Kambe

1．はじめに

近年、システム LSI は各種電子機器に広く利用され、高度化、複雑化が進んでいる。デバイス製造技術の進歩に対して LSI の設計生産性が追いつかないという問題が指摘されている。特に、複雑で大規模なシステムの設計では設計量が膨大になり、設計効率の向上と Time To Market を意識した短期間での設計が求められ、設計の危機が叫ばれている。これを解決するため、より抽象度の高いレベルでの設計が注目されている。これにより、記述量の削減や再利用化などにより効率的に行ない、高位合成ツールを用いて、より低い抽象度へと自動的に変換を行う。

現在の LSI 設計法は、RT（レジスタ転送）レベルが主流であるが、動作とタイミングを同時に設計するため、設計が詳細化され、アーキテクチャを短期間に変更することは困難である。これに比べ、C 言語などによるシステム LSI を設計する手法は、自動合成ツールが動作タイミングを決定するため、設計者はアーキテクチャ設計に集中し、開発期間を大幅に削減できる。C 言語による動作記述に対して、ビット精度指定、メモリ宣言やアクセス記述、並列動作記述、関数構造に対する変更、及びプラグマによる各種パラメータ指定等を行うことにより、様々なアーキテクチャを短期間に設計することが出来る。

各種アプリケーションをハードウエア化するためには、アーキテクチャ設計の最適化が重要となっており、高速化の手法の１つにパイプライン処理がある。パイプライン処理とは、複数の工程から成る処理を分担し、流れ作業で各工程を処理することによって、順次処理に比べ、ほぼパイプラインステージ数倍の高速化を実現する。本研究は、C 言語設計(Bach システム)において、DCT(離散コサイン変換)を対象にパイプライン処理など各種高速化手法を適用し、その性能を比較評価することを目的とする。

2．C 言語設計によるアーキテクチャ最適化

2.1　ビット幅の指定

C 言語設計では、変数宣言時に変数のビット幅を指定できる。これにより、冗長な演算回路、フリップフロップや配線などを削減することができる。しかし、ビット数は演算中のオーバーフローやアンダーフローを考慮する必要がある。

†近畿大学大学院総合理工学研究科エレクトロニクス系工学専攻
‡ 近畿大学　理工学部電気電子工学科
††NEC システムテクノロジー

2.2　パイプライン化・並列化

代表的な並列処理には、データの分割による並列化とパイプライン化がある。

データ分割による並列化は、大規模なデータに対して有効な場合があるが、並列度に応じて回路規模が増大する。また、分割境界において計算が正しく行われるようデータにオーバーラップを持たせるなど付加的な処理が必要になる場合がある。

パイプライン化は一般に回路規模への影響が少なく、性能向上に有効であるが、各ステージの処理時間やパイプラインステージ数の決定が問題となる。あるステージの処理時間が大きく、そのステージの処理を一定時間以内に終わらない場合は、パイプライン処理に乱れが生じ処理機能が低下するので、パイプライン化時に各ステージが一定時間以内に終了するように、全体の処理を均等に分割する。

C 言語設計における並列化、パイプライン化は関数単位もしくは for 文単位に行う。まず、設計者は各ステージの処理時間がほぼ同じになるようボトルネックとなるステージを高速化する。具体的にはボトルネックの要因を解析し、メモリアクセスがネックの場合はデータアクセス法の改良、特定の演算がネックの場合は専用演算回路の設計、処理の手順や構成が問題であればモジュール構成の変更などを行なう。

C 言語設計では、for 文のパイプライン化はループパイプライン化を図る。Bach システムでは、”throughout”プラグマによって指定されたサイクル数のパイプライン回路が自動的に生成される。ただし、同期通信などサイクル数が不定の処理は除外する必要がある。

関数単位のパイプラインは、モジュール間でハンドシェイク通信を行うことにより実現する。各パイプラインステージの処理時間に差があっても正しい動作が得る事ができる。

一方、動作合成技術は一般に演算器を共有し回路規模の最適化も行なう。演算器を共有するようにスケジューリングされているループ処理に対し、パイプライン化を指定すると、パイプライン化を優先するため演算器の共有が減り、回路規模が増加する場合がある。これらのメリット・デメリットを用途ごとに分析・比較し、設計対象に最も適した性能向上を図る。

3. 離散コサイン変換(DCT)回路

本文では、JPEG 圧縮の機能の中で最も処理時間の多い DCT を例に、ハードウェア化とその高速化を考える。

JPEG 圧縮で用いる 2 次元 DCT の 8×8 画素のブロックに対しての演算は、ブロック内の画素値を $p_b(i, j)$、得られる DCT 係数を $P_b(\mu, \nu)$ とすると、式 3-1 で表される。

$$P_b(\mu,\nu)=\frac{1}{4}C(\mu)C(\nu)\sum_{i=0}^{7}\sum_{j=0}^{7}p_b(i,j)\cos\frac{(2i+1)\mu\pi}{16}\cos\frac{(2j+1)\nu\pi}{16} \quad (3\text{-}1)$$

但し、
$$C(\mu)=\begin{cases}\frac{1}{\sqrt{2}} & (\mu=0)\\ 1 & (\mu\neq 0)\end{cases} \qquad C(\nu)=\begin{cases}\frac{1}{\sqrt{2}} & (\nu=0)\\ 1 & (\nu\neq 0)\end{cases}$$

この 2 次元 DCT は行列演算を行い、多くの乗算と加算が必要である。

本文では、Chen の高速演算アルゴリズムを適用した DCT を使用する。Chen のアルゴリズムは、2 次元 DCT の式(3-1)を 1 次元 DCT の式(3-2)、(3-3)に変換し、零行列の多い疎な行列に分解し、高速演算が可能である。(3-3)式に(3-2)式を代入すると(3-1)式となる。これにより 2 次元 DCT は 1 次元 DCT を行方向と列方向に行うことで計算できる。

$$g(\mu,j)=\frac{1}{2}C(\mu)\sum_{i=0}^{7}p_b(i,j)\cos\frac{(2i+1)\mu\pi}{16} \quad (3\text{-}2)$$

$$P_b(\mu,\nu)=\frac{1}{2}C(\nu)\sum_{j=0}^{7}g(\mu,j)\cos\frac{(2j+1)\nu\pi}{16} \quad (3\text{-}3)$$

但し、
$$C(\mu)=\begin{cases}\frac{1}{\sqrt{2}} & (\mu=0)\\ 1 & (\mu\neq 0)\end{cases} \qquad C(\nu)=\begin{cases}\frac{1}{\sqrt{2}} & (\nu=0)\\ 1 & (\nu\neq 0)\end{cases}$$

Chen アルゴリズムを適用した DCT の処理の流れ(バタフライ演算図)は図 3-1 に示すように step1～step5 の段階に分けることができる。

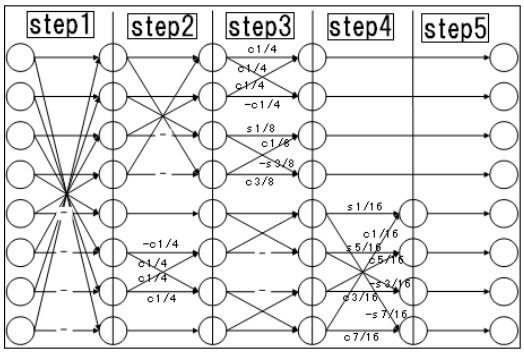

図 3-1　1 次元 DCT の流れ図(バタフライ演算図)

本研究で対象とするのは 300MCU(320×240 画素)の静止画であり、1MCU は Y 成分 4 ブロック、Cb 成分 1 ブロック、Cr 成分 1 ブロックの合計 6 ブロックで構成される。

4. DCT 回路設計

DCT 回路に適用する 10 種類の高速化手法について述べる。

(1)　ビット幅の指定

本文では、float 型変数を用いてハードウェア設計を行う。ここで、float 型は固定小数点表現であり、小数点以下の数値を表わすために必要な bit 数を実験より 12bit とする。

(2)　乗除算のシフト演算化

図 3-1 に示す step1 における乗算、step5 における除算を、シフト演算で実現することで高速化を図る。シフト演算適用後の記述例を図 3-2 に示す。

```
step1[0] = ls1[(i << 3) + 0] + ls1[(i << 3) + 7];
   :
   :
work[i] = step3[0]>>1;
   :
   :
```

図 3-2　シフト演算による記述例

(3)　専用回路化

図 3-1 の各 step を各々関数として記述し、各関数を”preserve”プラグマを用いて専用回路化する。Preserve とは BACH システムの動作合成における関数実現方法を指定するプラグマである。これを指定された関数は動作合成時に独立した処理回路として合成され、他の回路との資源の共有が無く、処理の高速化が図れる。

(4)　パイプライン化

BACH システムの動作合成において、throughput プラグマはループパイプラインを自動生成する。指定されたサイクル数が 1 ステージの処理時間となる。図 3-1 の回路を 1 ステージ 1 サイクルとして 5 ステージパイプライン回路として実現する。1 次元 DCT の処理は、行方向または列方向に 8 画素単位に 8 回ループとなるので、1 次元 DCT のパイプライン処理は 12 サイクルとなる（図 4-1）。

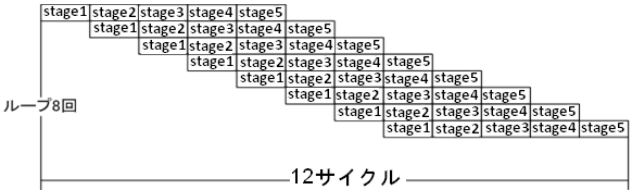

図 4-1　1 ステージ 1 サイクルのパイプライン

(5)　ステージ数の削減

(4)の回路をクロック周波数 100MHz で動作させる場合、各ステージの演算に時間的余裕があるので、さらに多くの演算を 1 ステージ中で行う。1 次元 DCT のパイプラインでは、ループ回数が 8 回と少なくかつブロック数分だけ実行されるのでステージ数を減らす事により処理時間を削減する。パイプライン・ステージ数を 4，3，2 ステージと削減すると 1 次元 DCT の処理は 11，10，9 サイクルとなる。4 ステージパイプラインを図 4-2 に、3 ステージパイプラインを図 4-3 に、2 ステージパイプラインを図 4-4 に示す。

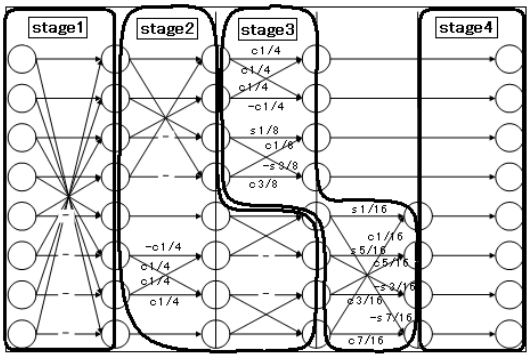

図 4-2　4 ステージの分割方法

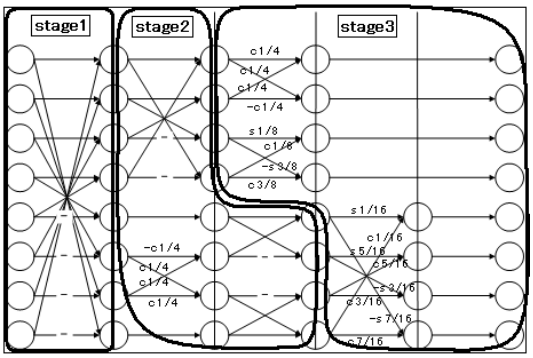

図 4-3　3 ステージの分割方法

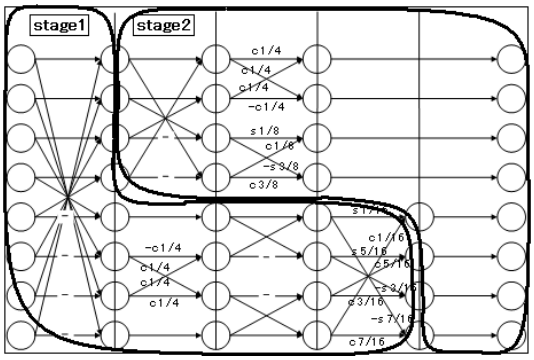

図 4-4　2 ステージの分割方法

(6)　ブロックの複数並列処理

DCT を 2 もしくは 3 ブロックの処理を並列化し、高速化を図る。この手法では、その分記述量が 2 倍、3 倍に増加する。

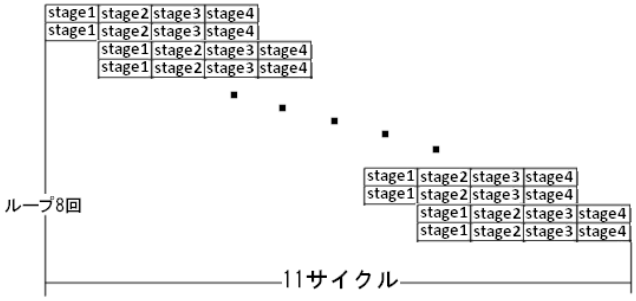

図 4-5　2 ブロック並列処理のイメージ図

(7)　1MCU 単位処理

次に、回路へのデータ入力方法を変更し、配列に 1MCU 分のデータを格納することにより、ループ数を 48 回とする 1MCU 単位のパイプラインを考える。次のブロックの処理に移る際に計算結果を格納する回路を追加する。

(8)　x・y 成分のパイプライン化

本文では Chen のアルゴリズムを用いているので、x・y 成分に関する 2 ステージパイプライン回路を考える(図 4-6)。

x成分	Y1	Y2	Y3	Y4	Cb	Cr
y成分	Y1	Y2	Y3	Y4	Cb	Cr

図 4-6　x・y 成分の 2 ステージパイプライン

(9)　三角関数の近似計算

DCT の演算で使用する三角関数を加減算とシフト演算の組み合わせで近似計算し、演算回数を削減する。近似計算における誤差評価を行った結果、小数点 1 桁の精度でも問題のない変換ができたので、これを用いる（表 4-2）。図 4-7、4-8 に演算を簡単化する前後の出力画像を示す。

表 4-2　小数点 1 桁精度の三角関数近似計算

	小数点8桁まで表示	シフトと加減算で表現	変換後の数値
cos(π/4)	0.707106781	>>1 + >>2	0.75
cos(π/8) 、sin(3π/8)	0.923879533	1 - >>4	0.9375
cos(3π/8)、sin(π/8)	0.382683432	>>2 + >>3	0.375
cos(π/16)、sin(7π/16)	0.980785280	1 - >>6	0.984375
cos(3π/16)、sin(5π/16)	0.831469612	1 - >>3	0.875
cos(5π/16)、sin(3π/16)	0.555570233	>>1 + >>4	0.5625
cos(7π/16)、sin(π/16)	0.195090322	>>3 + >>4	0.1875


図 4-7　変更前の画像


図 4-8　変更後の画像

(10)　ステージ分割アルゴリズム

(9)まではバタフライ演算をもとに 5 ステージ以下のパイプラインを考えたが、ここでは、DCT を一つの演算回路として最適なステージ数を求めるアルゴリズムを考える。まず、DCT の回路を 1 つの関数で実現し、演算の種類をもとにステージを分割する。次に、処理時間が小さい演算を他のステージに統合する。ここでは、三角関数の近似に用いたシフト演算の処理時間が加減算の処理時間に対して十分小さいので、シフト演算と加減算を 1 つのステージに統合する。さらに、処理時間の一番大きいステージに対し演算器を考慮した再分割を行い、各関数の処理時間を均一化する。以上の手順を表 4-3 に示す。

表 4-3　ステージ分割アルゴリズムの手順

手順	適用内容
1	Chen アルゴリズムの適用
2	三角関数を加減算+シフト演算に変換(小数点 1 桁精度)
3	処理ごとにステージ分割
4	シフト演算と加減算を 1 つのステージへ
5	ボトルネックのステージを再分割

5. 設計結果と考察

4 章で述べた高速化手法を DCT 回路に適用し、14 種類の回路について処理時間、回路規模の測定を行った。なお、本設計はクロック周波数を 100MHz とし、論理合成には日立 0.18 ミクロンライブラリ、Synopsys 社 Design Compiler を用いた。

表 5-1　回路への適用内容

	適用内容	関係
①	Chen アルゴリズム 3 章	
②	ビット幅の指定 4(1)	①→②
③	乗除算のシフト演算化 4(2)	②→③
④	専用回路化 4(3)	③→④
⑤	パイプライン化(5 ステージ) 4(4)	④→⑤
⑥	パイプライン化(4 ステージ) 4(5)	⑤→⑥
⑦	パイプライン化(3 ステージ) 4(5)	⑤→⑦
⑧	パイプライン化(2 ステージ) 4(5)	⑤→⑧
⑨	2 ブロック並列処理 4(6)	⑧→⑨
⑩	3 ブロック並列処理 4(6)	⑧→⑩
⑪	1MCU 単位処理 4(7)	⑧→⑪
⑫	x･y 成分のパイプライン化 4(8)	⑧→⑫
⑬	三角関数の簡単化 4(9)	⑧→⑬
⑭	ステージ分割アルゴリズム 4(10)	⑬→⑭

5.1　計算方法の改善

①～④の計算方法を改善した回路について、300MCU(320×240 画素)の画像の処理時間、回路規模の測定結果を表 5-2 に示す。また、処理時間比率、回路規模比率は①の回路を 1 とする。

表 5-2　専用回路化まで手法を適用

	処理時間 [ns]	処理時間比率	回路規模 [ゲート]	回路規模比率
①	7,776,000	1	850,473	1
②	5,472,000	0.704	567,082	0.667
③	2,880,000	0.370	254,930	0.300
④	1,440,000	0.185	180,804	0.213

②の回路は float 型変数の bit 幅を整数部 18bit、小数部 18bit から、整数部 13bit、小数部 12bit に変更することにより、より bit 幅の少ない演算器を使用し、2,304,000[ns]の高速化、283,391[ゲート]の回路削減を実現した。さらに③の回路では、step1 中の乗算や step5 中の除算を、シフト演算に変更することにより 2,592,000[ns]の処理時間削減を実現し、また演算器の規模も小さくなるので、回路規模も 312,152[ゲート]削減した。④の回路では、各関数を専用回路として実現することにより、処理時間は 1,440,000[ns]削減、回路規模は 74,126[ゲート]削減した。

5.2　パイプライン化と並列化による最適化

⑤から⑩のパイプライン化と並列化による回路について、300MCU(320×240 画素)の画像の処理時間、回路規模の測定結果を表 5-3 に示す。また、処理時間比率、回路規模比率は①の回路を 1 とする。

表 5-3　パイプライン化・並列化

	処理時間 [ns]	処理時間比率	回路規模 [ゲート]	回路規模比率
⑤	432,000	0.056	399,801	0.470
⑥	396,000	0.051	369,270	0.434
⑦	360,000	0.046	297,732	0.350
⑧	324,000	0.042	343,266	0.404
⑨	162,000	0.021	768,778	0.904
⑩	108,000	0.014	1,188,321	1.397

⑤の回路では、preserve プラグマと throughput プラグマにより各関数を 1 サイクル実行とし、図 4-1 のように 1 次元 DCT をスループット 12 サイクルで処理するループパイプラインを実装した。1 次元 DCT は x 成分と y 成分で 2 回行うので、1 ブロックあたり 24 サイクル、1MCU あたり 6 個のブロックがあるので、1MCU 全体のサイクル数は 144 となる。320*240 画素すなわち 300MCU の画像を処理するのにかかる時間は 12*2*6*10*300=432000[ns]となり、測定結果と一致する。

また、⑥の 4 ステージパイプラインも同様の計算から 300MCU の画像を処理するのにかかる時間は 10*2*6*10*300=360,000[ns]となり、理論値と一致する。⑧の 2 ステージパイプラインでは 1 次元 DCT のスループットが 9 サイクルであり、1 次元 DCT は x・y 成分で 2 回行うので、1 ブロックあたり 18 サイクルかかる。1MCU あたり 6 ブロックあるので、1MCU 全体のサイクル数は 108 となる。300MCU の画像の処理時間は 9*2*6*10*300=324,000[ns]となり、5 ステージ構成と比較し 25%の処理時間を削減した。

複数ブロックの並列処理の手法では、2 ブロック単位処理では 2 倍、3 ブロック単位処理では 3 倍の処理性能となるが、回路規模は約 2 倍、約 3 倍となる。

4、3、2 ステージパイプライン(⑥、⑦、⑧の回路)の各々に 1MCU 単位処理の設計結果を表 5-4 に、x・y 成分の並列化の設計結果を、表 5-5 に示す。ここでは処理時間比率、回路規模比率は⑥、⑦、⑧の回路を 1 とする。

表 5-4　1MCU 単位処理

対象回路	処理時間 [ns]	処理時間比率	回路規模 [ゲート]	回路規模比率
⑥	306,000	0.773	266,998	0.723
⑦	300,000	0.833	311,034	1.045
⑧	318,000	0.981	311,035	0.906

表 5-5　x・y成分のパイプライン処理

対象回路	処理時間[ns]	処理時間比率	回路規模[ゲート]	回路規模比率
⑥	246,000	0.621	429,640	1.163
⑦	225,000	0.625	452,038	1.518
⑧	204,000	0.630	502,729	1.465

表 5-4 より、1MCU 単位の処理に変更した結果、⑥の回路では約 23%、⑦の回路では約 17%、⑧に対する 1MCU 単位処理では約 2%の処理時間を削減した。しかし、⑧の 1MCU 単位処理では⑦のそれよりも処理時間が増加している。これは、⑧に対する 1MCU 単位処理回路を動作合成した結果、6 ステージとなったためである。

これらの回路は 1MCU(6 ブロック)の処理であり、48 回ループのパイプラインとして実現される。4 ステージパイプラインでは、1 次元 DCT を 51 サイクル、1MCU 全体のサイクル数は 102 となる。300MCU の画像処理時間は 51*2*10*300=306,000[ns]となり、測定結果と一致する。また、3 ステージパイプラインも同様な計算により測定結果と一致する。

表 5-5 より、x・y成分をパイプライン処理した結果、⑥、⑦、⑧の各々の回路で約 38%、約 37%、約 37%の処理時間を削減したが、 x ・y 成分パイプライン処理により、各々の回路規模が約 1.16 倍、約 1.52 倍、約 1.47 倍の増加となった。

また、⑥の回路では 1 次元 DCT の処理に 11 サイクルかかり、図 4-2 の様に x・y成分合わせて 1 次元 DCT7 つ分の処理時間となる。さらに、並列処理 1 つにつきオーバーヘッドが 1 サイクルあり、合計 5 サイクルとなる。300MCU の画像を処理するのにかかる時間は (11*7+5)*10*300=246,000[ns]となり、測定結果と一致する。また、⑦、⑧の回路でも、同様に理論値は測定結果と一致した。

5.3　動作周波数の向上とステージ分割アルゴリズム適用

三角関数の近似計算手法を⑥、⑦、⑧の回路に適用し、動作周波数、300MCU の処理時間、回路規模を表 5-6 に示す。また、ここでは処理時間比率、回路規模比率は⑥、⑦、⑧の回路を 1 とする。

表 5-6　三角関数の近似計算

対象回路	動作周波数	処理時間[ns]	処理時間比率	回路規模[ゲート]	回路規模比率
⑥	144.93	237,600	0.60	361,382	0.979
⑦	144.93	216,000	0.60	304,863	1.024
⑧	156.25	194,400	0.60	391,650	1.141

三角関数の近似計算を適用した結果、各ステージの処理時間が削減された分だけ動作周波数を上げることができた。これは、⑥の場合、クリティカルパスが、8.42ns から 6.36ns、同様に⑦では 9.03ns から 6.36ns に短縮でき、⑧では 8.80ns から 5.91ns に短縮できたためである。

さらにステージ分割アルゴリズムを適用した結果、DCT は 9 つの関数で実現でき、動作合成の結果、パイプラインステージ数は 7 となった。この回路では、クリティカルパスは 3.4ns となり、動作周波数は 217.39[MHz]となった。300MCU の処理時間、回路規模を表 5-7 に示す。

表 5-7　ステージ分割アルゴリズム

処理時間[ns]	回路規模[ゲート]
201,600	555,239

ステージ分割アルゴリズムを適用した結果、動作周波数は向上したが、ステージ数が増加し、その分処理時間の増加につながった。それぞれの関数ごとに専用演算器化しているので、他の回路との共有が少なくなり、回路規模も増大した。

図 5-1 に本研究で行った回路についての処理時間と回路規模の関係をグラフで示す。ここで、1MCU 単位処理の手法から三角関数の近似計算手法は 2 ステージパイプラインの回路を採用した。

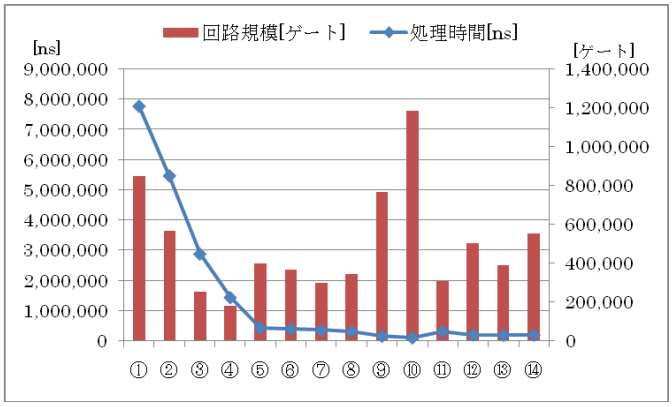

図 5-1　本研究の手法を適用した回路の測定結果

6. まとめ

本研究では、C 言語設計における回路高速化手法を提案し、DCT(離散コサイン変換)を対象にパイプライン処理など各種高速化手法を適用し、その性能を比較評価した。

具体的には、Chen アルゴリズム、bit 幅の指定、乗算・除算のシフト演算化、パイプライン化、並列化、パイプライン化のステージ数や処理単位を変更、1MCU 単位処理、x・y 成分の並列化、三角関数の近似計算、ステージ分割アルゴリズムなどの手法を適用することで処理時間の高速化を行うことが出来た。その結果、最速な回路は最初の回路と比較し、処理時間を 72 倍に高速化することに成功した。

今後の課題としては、更なる DCT の高速化や回路面積の削減や低消費電力化について考える。さらに、設計期間の短縮を図るために、パイプライン化処理の最適化を自動で行う手法の検討を進める。

謝辞

Bach システムを用いたハードウェア設計を実現するに当たり、多大なる御指導を頂いたシャープ株式会社電子デバイス開発本部先端技術開発研究所の BACH 開発グループの皆様に心から御礼申し上げます。

また、本研究は、東京大学大規模集積システム設計教

育研究センターを通し、シノプシス株式会社の協力で行われたものである

文献

[1] 中森章; “マイクロプロセッサ・アーキテクチャ入門”、CQ 出版、2004 年

[2] John L. Hennessy and D. A. Patterson: コンピュータの構成と設計, 日経BP社、1999.

[3] アズウィ、橋本晋之介、“JPEG 概念から C++の実装まで第 2 版”、SoftBank Creative

[4] K. Okada, *et al.*: Hardware Algorithm Optimization Using Bach C, IEICE Trans. Fundamentals vol.E85-A, No.4, pp835-841, 2002.

[5] T. Kambe, T. Kohara, "An Object-Oriented System LSI Design Methodology and Its Evaluation," The proceeding of The IASTED International Conference on Circuits and Systems, 2010/8.

[6] A. Eguchi, K. Kishida, T. Kambe, M. Saituji, "An Application Specific Circuits Design for a LVCSR System," The proceeding of ECCTD09.