卒業論文

ハード／ソフト最適分割を考慮した JPEG エンコーダの協調設計

氏名　　：的場 督永
学籍番号：2210010209-2
指導教員：山崎 勝弘 教授

提出日　：2005 年 2 月 21 日

立命館大学 理工学部 情報学科

内容梗概

本論文では、現在の LSI 設計で注目を集めているハード／ソフト協調設計において、重要な課題となっているハード／ソフト分割の最適化について述べる。分割の最適化を検証する手段として、FPGA とソフト・マクロ CPU を用いたハード／ソフト協調システムを構築した。また、このシステムを用いた協調設計の手法を評価するために、実装の対象とするアプリケーションとして、JPEG エンコーダをとりあげた。

JPEG エンコードの処理を、4 つの機能ブロックへと分割し、それぞれソフトウェアおよびハードウェアモジュールとして実現した。また、FPGA に実装したシステムの CPU における、ソフトウェアの処理の負荷を機能ブロック毎に計測し、その結果に基づいて様々なハード／ソフトの分割パターンを選出した。その後、それら全ての分割パターンについて協調システムとして FPGA への実装を行った。様々な分割パターンに対し、実装結果として得られた回路規模、性能およびメモリ量から比較評価を行うことで、JPEG エンコーダにおける最適な分割パターンを考察した。

FPGA とソフト・マクロ CPU を用いたハード／ソフト協調システムとして、JPEG エンコーダの実装を行ったことにより、設計期間の短縮や、実装結果の信頼性といった点で、設計手法の効果を確認した。

目次

図目次

表目次

1. はじめに

1.1 研究背景

近年の半導体集積技術の向上は、LSI の小型化、高速化、省電力化などを可能にしている。特に、複数の機能を 1 つのチップ上に集積したシステム LSI は、携帯電話や車載機器、デジタルカメラのような、我々が普段手にする様々な電子機器に広く用いられ、高い付加価値を与えている。

一方で、これらの対象アプリケーションやアーキテクチャの多様化、大規模化は、半導体設計の複雑化という問題をもたらしている。また、信号処理技術の発展のような、対象アプリケーションにおける技術の向上は、製品における開発期間の短縮を促進しており、設計量の増大と同時に、仕様変更に耐え得る柔軟な設計が求められている。現状では、LSI の複雑度が 50%上昇する間に、設計の生産性は 20%しか向上しないという統計があり、設計生産性、設計品質の危機として問題となっている。

この問題に対処するため、ハード／ソフト協調設計という手法が注目されている[1]。これは、ハードウェアとソフトウェアの協調によって様々な機能を実現しているシステム LSI において、ハード／ソフトを別々に設計、検証するのではなく、双方の設計者が協調して設計、検証を進めていくことで、システム全体の最適化と、設計の効率化を図る手法である[12]。

しかし、ハード／ソフト協調設計は、その手法がまだ確立されておらず、特にシステム全体の最適化において重要となるハード／ソフトの分割は、多くの部分が人手に委ねられている。設計生産性の危機を解決するための手段として、これらの設計手法の確立が求められている[13][14]。同時に、設計者に対しては、ハードウェア、ソフトウェア、そしてシステム全体の設計に関する体系的な知識が求められている[8]。

1.2 研究目的

本研究では、ハード／ソフト協調設計における分割の最適化を検証する手段として、FPGA とソフト・マクロ CPU を用いたハード／ソフト協調システムを構築する。JPEG エンコーダという実用アプリケーションを対象とし、様々な分割パターンに対して実機上での評価を行うことで、最適な分割パターンを発見することを目的とする。

ハード／ソフト協調設計における問題点として、システムを実際に LSI として試作する場合、経済的および時間的なコストが非常に大きくなる。さらに、シミュレーションを軸にした検証手法では莫大な時間を必要とするため、ハード／ソフト分割について十分な検証を行うことは難しい。これに対し、FPGA を用いた協調システムでは、ゲート数などの制約はあるものの、実装における経済的コストはほとんど無く、また、書き換えが自由なため、実機上での様々な分割パターンの検証を高速に行うことができる。

設計対象のアプリケーションとして、本研究では JPEG エンコーダをとりあげる。JPEG

エンコード処理は、静止画像を圧縮符号化する標準的な方式であり、携帯電話やデジタルカメラのような電子機器で広く用いられている[6][7]。しかし、多量のデータに対して乗除算を繰り返し行うため、比較的演算量が大きく、組み込み用途の低速な CPU だけで必要となる性能を満たすことは難しい[2][9]。そのため、一部もしくは全ての処理をハードウェアで実現することで性能の向上が図られることが多く、ハード／ソフト協調設計を適用する対象として適していると考えられる[15][16]。

本研究における協調設計の流れは、まず JPEG エンコードの処理全体をソフトウェアによって実現する。また、ハード／ソフト協調システムでの実装を考慮し、処理全体を 4 つの大まかな機能単位で分割する。さらに、システム上の CPU における処理の負荷を、分割した機能ブロック毎に計測し、その結果からハードとソフトの複数の分割パターンを検討する。次に、各機能ブロックをハードウェアモジュールとして設計した後、分割パターンに応じて処理をハード、ソフトから選択し、FPGA 上に複数パターンでのシステムの実装を行う。また、設計の各段階に応じて、シミュレータや FPGA 上のデバッガを用いることで、ハード／ソフトの協調検証を行う。最後に、性能や規模などから各システムを比較評価し、JPEG エンコーダにおける最適な分割パターンを決定する。

これらの流れを通して、FPGA を用いたハード／ソフト協調システムにおける設計の有用性と、改善点について考察する。

1.3 論文の構成

本論文では、本章で研究全体の背景と目的を明らかにし、第 2 章において、ハード／ソフト協調設計の概念の説明と、本研究で構築した FPGA を用いたハード／ソフト協調システムの構成を示す。第 3 章では、設計対象のアプリケーションである JPEG エンコードの概要と、ソフトウェアでの実装について述べ、第 4 章では、機能ブロック毎のハードウェア設計について述べる。第 5 章では、ハード／ソフト協調システムとしての JPEG エンコーダの FPGA 実装と、最適分割パターンの決定、設計手法の考察を行う。最後に、第 6 章において、現在までの成果と今後の課題について述べる。

2. ハード／ソフト協調設計

2.1 ハード／ソフト協調設計とは

ハード／ソフト協調設計とは、ハードウェアとソフトウェアの協調によって様々な機能を実現しているシステム LSI において、ハード／ソフトを別々に設計、検証するのではなく、双方の設計者が協調して同時に開発を進める手法である[1]。

設計対象となるアプリケーションに応じて求められる、性能やコスト、消費電力のような様々な制約に対して、ハード／ソフトの分割によりシステム全体での最適化を行うことを目的としている。同時に、ハードウェアの実装後にソフトウェアの開発や検証を行うといった従来の方法に対し、設計の各段階においてハード／ソフトの協調検証を行うことで同時開発を可能とし、設計を効率化することを目的としている。

具体的な手法としては、ISS（Instruction Set Simulator）や ICE（In-Circuit Emulator）を用いた各段階での協調検証の高速化や、ハードウェアの IP（Intellectual Property）化による再利用の促進、システムレベルでの記述が可能な言語（SystemC、SpecC など）を用いた設計抽象度の引き上げなどが行われている。

しかし、まだ定義が明確になっていない部分や、CAD ツールなどによって自動化されていない部分も多く、現在も活発な研究対象となっている[12][13][14]。

2.2 ハード／ソフト協調設計のフロー

ハード／ソフト協調設計の全体的な流れを図 1 に示す。

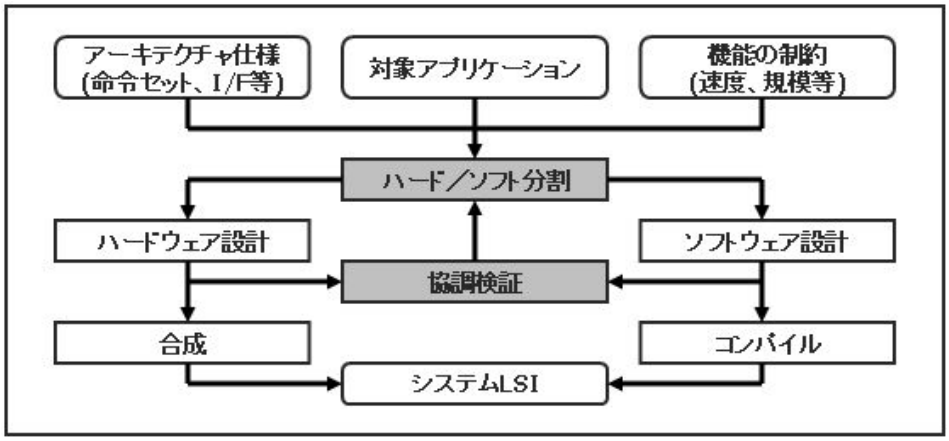

図 1：ハード／ソフト協調設計のフロー

まず、対象となるアプリケーションの機能を、C や UML、システムレベル言語などによって記述したシステムのモデルが、協調設計の開始点となる。これに加えて、実装対象のアーキテクチャにおける CPU や DSP の命令セット、メモリやバスのインタフェースなどの仕様と、製品に求められる性能や規模、消費電力などの制約から、アプリケーション機

能のハード／ソフトへの分割を行う。その後、ハードウェアとソフトウェアをそれぞれ設計し、設計の各段階において、ISS や ICE を用いたハード／ソフトの協調検証を行う。これにより、求める機能の制約を満たせない場合は、再び分割を検討することになる。最終的に、ハード／ソフトの双方において機能の制約を満たした場合、製品としてシステム LSI を実現できる。

現在、CAD ツールの発展によって、協調検証が実用的になりつつあり、検証結果が制約を満たさない場合の発見は容易になってきている。しかし、協調検証の環境は非常に高価であり、検証の高速化はまだ十分とは言えないため、協調検証にかかる時間的、経済的なコストは大きい。さらに、ハード／ソフト分割の自動化はまだ実用的ではなく、人手によって行われていることが多い。このため、分割が良くない場合、検証と分割の繰り返しが多くなり、設計期間に影響を与える。また、制約を満たすことと、最適な分割であることは同義では無く、システムの最適化はまだ解決されていない最も大きな問題である。

ハード／ソフト協調設計におけるこれらの問題点を踏まえ、本研究ではアプリケーションを実装する対象として、FPGA とソフト・マクロ CPU を用いたハード／ソフト協調システムを構築する。このシステムを用いてハード／ソフト協調設計を行うことで、様々な分割パターンを実機上で高速に検証することが可能となり、開発期間の短縮を図ることができる。また、様々な分割パターンを実装し、その比較評価を行うことで、最適な分割を検討することができると考えている。

本研究における実際の設計の流れとしては、設計対象とするアプリケーションである JPEG エンコーダについて、まず C 言語を用いたソフトウェアモデルとして記述し、その処理を大まかな機能単位に分割する。これを FPGA 上のハード／ソフト協調システムへと実装することで、それぞれの処理にかかる負荷を正確かつ高速に計測する。ソフトウェア実行により求められた各処理の負荷の割合は、ハード／ソフト分割パターンの検討に用いる。次に、JPEG エンコーダの全ての機能ブロックを、ハードウェアモジュールとして設計する。本研究では全ての処理についてハードウェア化を行うが、最終的な目標としては、ソフトウェア実行の時点で、処理をハードウェア化するかどうかの取捨選択をある程度行えるようにしたいと考えている。最後に、複数の分割パターンをハード／ソフト協調システムとして FPGA 上に実装する。実機上での動作結果からそれぞれのパターンの比較評価を行い、要求に応じた最適な分割パターンを決定する。

2.3 FPGA を用いたハード／ソフト協調システム

本研究では、アプリケーションの実装、評価を行う対象アーキテクチャとして、FPGA とソフト・マクロ CPU を用いたハード／ソフト協調システムを構築した。ハード／ソフト協調システムの構成を図 2 に示す。

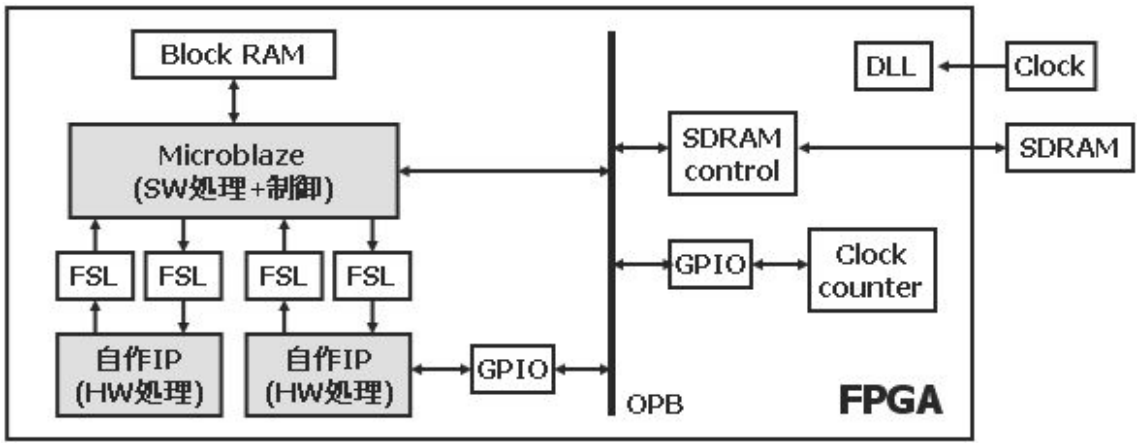

図 2：ハード／ソフト協調システムの構成

図 2 に示すように、FPGA ボード上に、CPU、ハードウェアモジュール、メモリ、およびバス等を実装して接続し、ハードウェアとソフトウェアが協調して動作するシステムを構築する。本研究で使用する FPGA ボードは、Xilinx 社の FPGA である Spartan-ⅡE の 60 万システムゲートを搭載した、MEMEC 社の ES-KIT-MB-S2E6LC-EURO 評価ボードである。また、ボード上には 8M×32 の SDRAM や、各種スイッチが搭載されている。

ソフトウェアによる処理と、システム全体の制御を行う CPU としては、Xilinx 社の提供するソフト・マクロ CPU である Microblaze を用いる。ソフト・マクロ CPU とは、HDL データの形で提供され、FPGA 上の資源を用いて構成することができるマイクロプロセッサの IP であり、バスや周辺機器との接続を自由にカスタマイズすることができる。また、バスやメモリコントローラ、デバッガなどの IP や、コンパイラなどの開発環境も提供されているため、設計期間を短縮することができる。Microblaze は 32bit の RISC プロセッサで、MIPS に準拠した命令セットをもち、3 段パイプラインで動作する。また、プロセッサ上には乗除算器を持っていない。

ハードウェア処理を行う自作 IP モジュールは、FSL バスを介して Microblaze と接続する。FSL バスは FIFO によって実現されており、Microblaze と周辺機器を直接接続することで、一方向の高速通信を同期的に行うことができる。ただし、FIFO を用いているため、バスに格納しているデータのランダムアクセスは不可能である。よって、本研究で扱う JPEG エンコーダのような、大量のストリームデータを処理するアプリケーションにおける、ハード／ソフトの通信インタフェースとして適していると考えられる。

FSL バスはデータの転送のみに用いるため、制御信号の通信が必要な場合は、Microblaze における標準的なデータバスである OPB（On-Chip Peripheral Bus）に接続した GPIO を用いて行う。GPIO は最大 32bit のレジスタを用いて実現されており、PIO 方式によるデータ転送を行う。Microblaze による GPIO の読み出し、書き込み動作は低速であり、連続データの転送には適していない。

また、システム実行時の動作クロック数を計測するため、クロックカウンタを自作し、GPIO を用いて接続している。クロックカウンタは、Microblaze による GPIO への制御信号の書き込みによって動作し、カウント値を別の GPIO に書き込む。これを必要なときに読み出すことで、実行時の動作クロック数を正確かつ高速に計測することができる。

3. JPEG アルゴリズムのソフトウェア実装

3.1 JPEG とは

JPEG とは Joint Photographic Experts Group の略であり、静止画像符号化の標準方式を定義するために設立された規格策定団体を表している[6][7]。一般に、JPEG 画像と呼ばれる形式は、JPEG が策定した規格である JFIF（JPEG File Interchange Format）を指している。

JPEG は広い適用範囲を想定し、汎用性を高めるために、アルゴリズムの機能を 4 つの動作モードに分類している[3]。ユーザは用途と必要な画質に応じて、必要な機能を選択することができる。4 つの動作モードを以下に示す。

（1）シーケンシャル DCT（離散コサイン変換）ベース

8*8 画素のブロックに対し、左から右へ進むラインの符号化処理を、上から下へ 1 回のスキャンで行う。符号化処理は 2 次元 DCT 係数の近似とエントロピー符号化からなる。

（2）プログレッシブ DCT ベース

処理の順番および符号化処理は（1）と同様だが、複数回の処理スキャンを行う。

（3）ロスレス

DCT を用いず、近接画素との差分をエントロピー符号化する DPCM 符号化を用いることで、可逆符号化を行う。

（4）ハイアラーキカル

上記 3 つの動作モードを組み合わせ、複数の空間解像度を持つ画像のピラミッド構造を作成する。

また、これらを具体的に実現する符号化アルゴリズムの基本構成によって、表 1 に示すように分類することができる。

表 1：JPEG エンコードアルゴリズムの分類

	DCT 方式		Spatial 方式
	ベースラインシステム	拡張システム	（DPCM 方式）
符号化によるひずみ	有（非可逆符号化）	有（非可逆符号化）	無（可逆符号化）
入力画像精度	8bit	8bit または 12bit	2～16bit
動作モード	シーケンシャル	シーケンシャルまたはプログレッシブ	シーケンシャル
エントロピー符号化	ハフマン符号化	ハフマン符号化または算術符号化	ハフマン符号化または算術符号化

本研究では、表 1 に示される分類のうち、JPEG における必須機能と位置付けられているベースラインシステムのみを対象とする。次節より、ベースラインシステムのエンコードアルゴリズムを説明する。

3.2 JPEG エンコードアルゴリズム

3.2.1 JPEG エンコードの処理フロー

JPEG ベースラインシステムにおける、エンコード処理の全体的な流れを図 3 に示す。

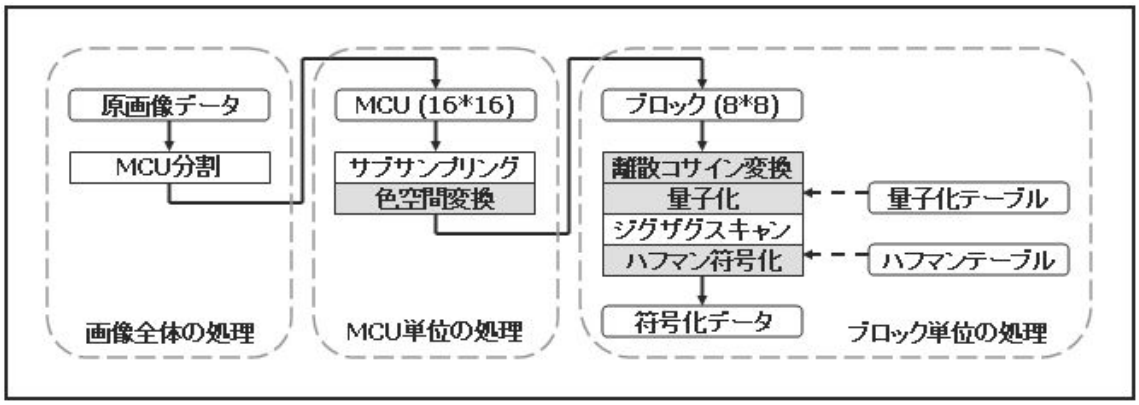

図 3：JPEG エンコード処理のフロー

図 3 に示すように、まず原画像データを 16＊16 画素領域である MCU（Minimum Coded Unit）へと分割する。以下の処理は、MCU を単位として原画像データの全 MCU に対して順に行う。

各 MCU に対し、3 原色の RGB カラーモデルから、輝度と色相を表す YUV カラーモデルへと色空間の変換を行う。ここで、符号化効率を高めるためにサブサンプリングによって色差成分の間引きを行い、成分毎に 8＊8 画素のブロックを構成する。以下の処理は、ブロックを単位として MCU 内の全ブロックに対して順に行う。

各ブロックに対して 2 次元 DCT（離散コサイン変換）を行い、ブロックの要素を周波数データである DCT 係数へと変換する。次に、DCT 係数 1 つあたりの符号量を削減するために量子化を行い、ジグザグスキャンによって低周波成分から順に 64 要素の 1 次元配列へと整列する。最後に、配列の先頭要素である DC（直流）成分と、残り 63 要素の AC（交流）成分に対して順にハフマン符号化を行うことで、連続する符号化データを得ることができる。

主な演算処理は、色空間変換、DCT、量子化、ハフマン符号化の 4 つに分割することができる。また、サブサンプリングは色空間変換と、ジグザグスキャンは量子化と同時に行うことができる。次項より、4 つの処理のそれぞれについてアルゴリズムの説明を行う。

3.2.2　色空間変換

3 原色の RGB カラーモデルで表現された MCU に対し、画素毎にその RGB 値を用いて以下の演算を行うことで、輝度と色相を表す YUV カラーモデルへと変換する。

$$Y = 0.299R + 0.587G + 0.114B - 128$$
$$U = -0.1687R - 0.3313G + 0.5B$$
$$V = 0.5R - 0.4187G - 0.0813B$$

Y は輝度成分と呼び、その画素の明るさを表している。U、V は色差成分と呼び、それぞれ青方向、赤方向への色の変化量を表している。色差成分 U、V に対して、輝度成分 Y は画素間の変化が大きいため、色空間変換を行うことによって、RGB ではほぼ均等だった情報量を、Y に多く偏ったものにすることができる。さらに、人間の視覚は、輝度の変化に比べて色の変化に鈍感であるという性質があるため、後述するサブサンプリングや量子化の処理において色差成分の精度を落とすことで、画像の劣化を見た目上は抑えながら、符号量を少なくすることができる。Y を求める演算において 128 を引いているのは、0～255 で表される 8bit の輝度値を、－128～127 の範囲にシフトすることで、次項で説明する DCT 係数の絶対値を小さくするためである。

サブサンプリングとは、図 4 に示すように、輝度 Y では 16＊16 画素で表される MCU の全ての成分を用いるのに対し、色差 U、V は縦、横方向それぞれ 1 画素置きの成分（図で色付けした画素）のみを用いることで、Y : U : V＝4 : 1 : 1 の割合で U、V の間引きを行い、1 つの MCU から 8＊8 画素のブロックを 6 個構成することである。以降の符号化処理は、各 MCU において Y0、Y1、Y2、Y3、U、V の順でブロック毎に行う。

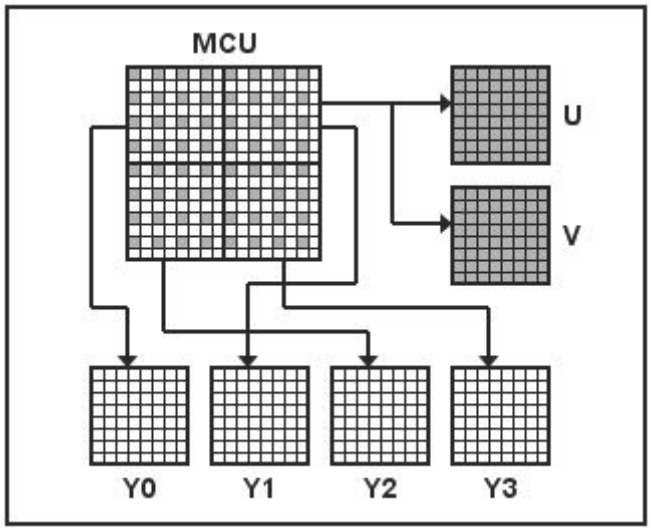

図 4：サブサンプリング（4：1：1）によるブロックの構成

なお、4：2：2 や 4：4：4 のサブサンプリングが用いられることもあるが、4：1：1 が最も一般的である。また、MCU のサイズはこれらのサブサンプリングの比によって決まる。4：2：2 の場合、MCU は縦 8＊横 16 画素となり、色差成分は横方向のみ 1 画素置きに用いることで、2 つの輝度 Y ブロックと、色差 U、V ブロックを 1 つずつ構成する。4：4：4 の場合は、MCU は 8＊8 画素となり、色差成分の間引きは行わない。

3.2.3　離散コサイン変換

8＊8 画素のブロックに対し、2 次元の離散コサイン変換（DCT）を行うことで、画素値から周波数成分への変換を行う。これは、画像の高周波成分の精度を落とすことで起こる画質の劣化は、低周波成分の場合に比べて比較的目立たないという性質を利用するためである。1 ブロックの画像を、2 次元 DCT によって 64 段階の周波数成分に変換することで、次項で説明する量子化において効果的に符号量を減らすことができる。

具体的な DCT の演算は、ブロック内の画素値を $P(x,y)$ とし、得られる周波数成分の大きさ（以下 DCT 係数と呼ぶ）を $D(u,v)$ とする場合、次のような式で表される。

$$D(u,v)=\frac{C(u)\times C(v)}{4}\times\sum_{x=0}^{7}\sum_{y=0}^{7}P(x,y)\times\cos\left(\frac{(2x+1)u\pi}{16}\right)\times\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

$$C(u),C(v)=\begin{cases}\dfrac{1}{\sqrt{2}} & (u,v=0)\\ 1 & (else)\end{cases}$$

DCT 係数 $D(u,v)$ は 8＊8 の 2 次元配列で表される。このうち最も左上の要素である $D(0,0)$ を DC 成分、残りの 63 要素を AC 成分と呼び、右または下の要素ほど高い周波数成分に相当する。

本研究では、演算量を削減するため図 5 のように 1 次元 DCT を 2 度繰り返して行うことで 2 次元 DCT を実現する。ブロック内の各行に対して、x 方向の 8 要素に対する 1 次元 DCT を行った後に、各列に対して y 方向の 1 次元 DCT を行う。これを式で表すと次のようになる。

$$T(u,y)=\frac{C(u)}{2}\times\sum_{x=0}^{7}P(x,y)\times\cos\left(\frac{(2x+1)u\pi}{16}\right)$$

$$D(u,v)=\frac{C(v)}{2}\times\sum_{y=0}^{7}T(u,y)\times\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

$T(u, y)$という 1 次元 DCT の結果を一時的に格納するための領域が必要になるが、1 画素につき 8 項の積和演算が 2 回となるため、1 ブロックの 64 画素に対する 2 次元 DCT は、64＊8＊2 回の積和演算によって実現することができる。

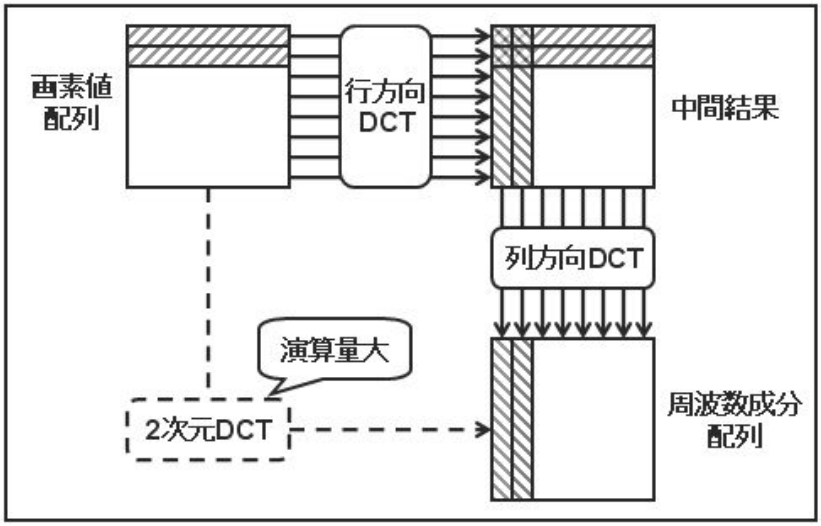

図 5：1 次元 DCT の繰り返しによる 2 次元 DCT の実現

3.2.4　量子化

量子化とは、信号を一定精度のデジタルデータで近似することである。JPEG エンコードにおける量子化は、DCT 係数を量子化ステップと呼ばれる整数で除算して、四捨五入することで整数化するという方法で行う。

量子化によって DCT 係数の精度を削減することで、符号量を少なくすることができるが、元の情報を厳密に復元できない非可逆な符号化となり、画像が劣化する一因となる。前項で述べたように、画像の高周波成分の精度を落とすことで起こる劣化は、低周波成分の場合に対して、比較的目立たないという性質があるため、高周波成分を表す DCT 係数に対する量子化ステップを大きくすることで、見た目上の画像の劣化を抑えつつ符号量を削減することができる。また、3.2.2 項で述べたように、人間の視覚は色差成分の変化に鈍感なため、輝度成分に対して色差成分の量子化ステップを大きくすることで、効果的に符号量を削減することができる。

量子化ステップサイズの最適値は入力画像や表示装置の特性に依存するため、JPEG では具体的な値を規定していない。しかし、JPEG 規定の Annex K ガイドラインでは、表 2 と表 3 に示すような、輝度成分および色差成分ブロックの各 DCT 係数に対する量子化ステップ（以下量子化テーブルと呼ぶ）の例が示されている。本研究では、表 2 および表 3 の量子化テーブルを用いて、量子化を行うこととする。JPEG エンコードにおける画像品質の制御は、量子化テーブルに一定乗数を掛けることで行うことができる。図 6 に、表 2 の量子化テーブルを用いた輝度成分ブロックの DCT 係数に対する量子化の例を示す。

表 2：輝度成分の量子化テーブルの例

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

表 3：色差成分の量子化テーブルの例

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

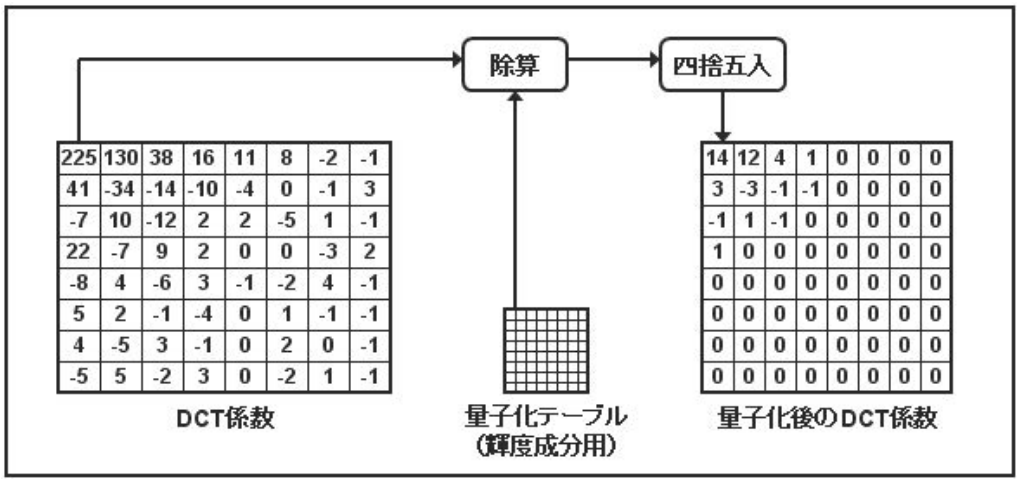

図 6：DCT 係数の量子化の例

また、次項で説明するハフマン符号化によって符号量を効果的に削減するために、2 次元で表された量子化後の DCT 係数を、低い周波数成分から順に 1 次元の配列へ整列する。DCT 係数は 2 次元配列上で右または下へ行くほど高い周波数成分を表すので、図 7 に示すような順番で並べ替えを行う。この操作をジグザグスキャンと呼ぶ。

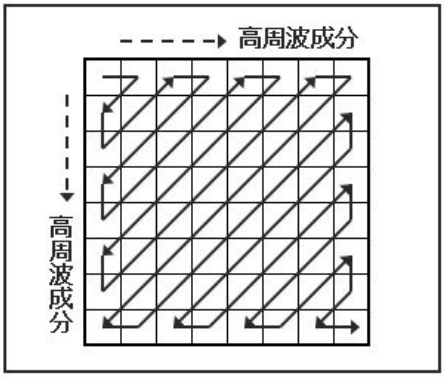

図 7：DCT 係数のジグザグスキャン

3.2.5　ハフマン符号化

量子化された DCT 係数に対して、統計的な出現頻度に応じて可変長のハフマン符号に置き換えることで、エントロピー符号化を行う。出現頻度が高い値ほど短いハフマン符号を割り当てることで、符号量の削減を行う。ジグザグスキャンにより並べ替えられた DCT 係数において、先頭要素である DC 成分はブロック内の画素値の平均を表しており、他の要素である AC 成分とは違った特徴を持つため、DC 成分と AC 成分では別々の処理を行う。

（1）DC 成分のハフマン符号化

DC 成分の値は、隣接するブロックの DC 成分値と近い場合が多いため、直前ブロックの DC 成分値との差分をハフマン符号化する。そのため、差分値は 0 に近い値の出現頻度が高くなることから、まず絶対値によるグループ分けを行い、絶対値が小さいものほど短いハフマン符号を割り当てる。次に、グループ内の値の区別を行うための付加ビットを追加する。量子化の場合と同様に、JPEG によって具体的なハフマン符号は規定されていないが、Annex K ガイドラインにおいて表 4 に示すようなテーブルが例として示されている。

表 4：DC 成分のハフマン符号表（輝度成分用）

グループ番号	DC 差分値	ハフマン符号	付加ビット数
0	0	00	0
1	−1,+1	010	1
2	−3,−2,+2,+3	011	2
3	−7,−6,−5,−4,+4,+5,+6,+7	100	3
4	−15,…,−8,+8,…,+15	101	4
5	−31,…,−16,+16,…,+31	110	5
6	−63,…,−32,+32,…,+63	1110	6
7	−127,…,−64,+64,…,+127	11110	7
8	−255,…−128,+128,…,+255	111110	8
9	−511,…,−256,+256,…,+511	1111110	9
10	−1023,…,−512,+512,…,+1023	11111110	10
11	−2047,…,−1024,+1024,…,+2047	111111110	11

例えば、輝度成分のブロックにおいて、前ブロックとの DC 差分値が−4 の場合、表 4 よりグループ番号は 3 となり、そのハフマン符号は 100 となる。また、グループ内の値を区別するための付加ビット数は 3 ビットとなり、差分値が小さい順に符号を割り当てるため、−4 の付加ビットは 011 となる。よって、DC 差分値−4 は、100011 という 6 ビットの値へ符号化される。

（2）AC 成分のハフマン符号化

AC 成分は、ブロックにおける周波数成分の明るさの変化を表すが、通常 1 つのブロック内での明るさの変化はそれほど大きく無いため、多くの AC 成分が 0 となる。また、高周波成分は大きなステップで量子化されているため、ジグザグスキャンにより高周波成分が集まっている後半部分は 0 であることが多い。

この特徴を利用して、連続する 0 の数（以下ゼロランレングスと呼ぶ）と、その後に出現する 0 でない有効な値とをまとめてハフマン符号に置き換えることで、符号量を削減する。ゼロランレングスが 16 になった場合は、ZRL と呼ばれる特別な符合を出力し、そのカウントをリセットする。また、ブロック内の残りの AC 成分値が全て 0 の場合、EOB と呼ばれる特別な符号を出力し、そのブロックの符号化を終了する。

AC 成分の符号化は、表 5 に示すように有効な AC 成分のグループ番号とゼロランレングスからハフマン符号を決定し、その後付加ビットを追加する。表 5 は輝度成分のブロックに対するハフマン符号の例を示しており、縦軸 G がグループ番号、横軸 Z がゼロランレングスを表している。グループ分けと付加ビット数の決定は、表 4 に示した DC 成分の場合と同様に行う。

表 5：AC 成分のハフマン符号表（輝度成分用）

G＼Z	0	1	2	…	15
0	1010（EOB）	無し		…	11111111001（ZRL）
1	00	1100	11100	…	1111111111110101
2	01	11011	11111011	…	1111111111110110
…	…	…	…	…	…
10	1111111110000011	1111111110001000	1111111110001110	…	1111111111111110

例えば、輝度ブロックの AC 成分において、0 が 2 つ連続した後に＋2 が現れた場合、ゼロランレングスは 2 で、表 4 よりグループ番号が 2 となるため、表 5 より対応するハフマン符号は 11111011 となる。また、付加ビット数は表 4 より 2 ビットとなり、値が小さい順に符号を割り当てるため、＋2 の付加ビットは 10 となる。よって、0 が 2 つ連続した後の有効 AC 成分＋2 は、1111101110 という 10 ビットの値へ符号化される。

輝度成分と色差成分では特徴が違うため、DC 成分と AC 成分それぞれで異なるハフマン符号表（合計 4 種類）を用いる。また、符号化されたデータは可変長となるため、8 ビット毎に出力する。ただし、出力する値が 0xFF となった場合は、JPEG ファイルのヘッダ情報と区別するため、0x00 を続けて出力しなければならない。

3.3 JPEG エンコードのソフトウェア実装

3.3.1 C によるソフトウェア実装

3.2 節で述べた JPEG エンコードアルゴリズムに基づき、C 言語を用いて処理全体をソフトウェアで実現した。2.3 節で述べたように、実装対象となる FPGA を用いたハード／ソフト協調システムの CPU である Microblaze には乗算器、除算器が無いため、ソフトウェアによる乗算、除算命令はシステムへの負荷が大きいと考えられる。そのため、シフトによる置き換えや、定数をあらかじめ計算してテーブルとして用意することなどにより、可能な限り乗除算を削減した。また、小数については、演算量の大きい浮動小数点演算を用いず、シフトによる固定小数点演算を行った。

開発の流れとしては、まず Windows 上の Cygwin 環境において開発と検証を行い、その後、Microblaze 環境への移植を行った。コンパイラは、Cygwin 環境では gcc-3.3.3 を利用し、Microblaze 環境では Xilinx から提供されている Microblaze 向けにカスタマイズされた mb-gcc コンパイラを利用した。

Cygwin 環境におけるソフトウェアは、入力画像ファイルのオープンと MCU への分割、および JPEG ファイルのヘッダ情報の出力を行う前処理部、3.2 節で述べた 4 つの符号化処理を行う演算部、そして、得られたハフマン符号列と残りのヘッダ情報を出力する後処理部の 3 つの処理を行う。

本研究において、ハード／ソフト協調システム上で動作するソフトウェアモデルを構築する目的は、JPEG 画像ファイルの出力ではなく、各処理ブロックの演算の負荷を計測し、ハード／ソフトの分割パターンの検討を行うことである。よって、Microblaze 環境への移植においては、前処理部と後処理部を省略し、色空間変換、DCT、量子化、ハフマン符号化の 4 つの処理を行う演算部のみを実装した。演算の負荷の計測は、FPGA 上に実装したクロックカウンタを用いることで、実際のシステムにおける正確な処理時間を求めることができる。

3.3.2 実行結果と考察

まず、Cygwin 環境において、ソフトウェアの正常な動作を確認した。図 8 に示すような 256＊256 画素のビットマップ画像を入力として実行したところ、図 9 に示すような JPEG 形式のファイルを出力画像として得ることができた。入出力画像のサイズを比較すると、入力画像は 196,662 バイト、出力画像は 6,883 バイトになっており、約 1/30 程度に圧縮されていることが分かる。また、ブロックノイズの発生など、画質の大きな劣化は見られなかった。Cygwin 環境における実行時間は、本研究の実装対象である FPGA 上のシステムでの実行とは関連が薄いため、計測は行っていない。

図 8：JPEG エンコードの入力画像

図 9：JPEG エンコードの出力画像

次に、FPGA 上に実装したハード／ソフト協調システムにおいて、図 8 の入力画像から 1MCU 分（16＊16 画素）の RGB 値を取り出し、テスト入力データとしてソフトウェアを実行した。4 つの処理の演算の負荷をそれぞれ調べるため、クロックカウンタを用いて実行クロック数を計測した。1MCU から作られる Y0～Y3、U、V の 6 個のブロックに対して、色空間変換、DCT、量子化、ハフマン符号化の各処理にかかったクロック数を表 6 にまとめる。また、処理単位毎の負荷の割合を表したグラフを、図 10 に示す。

表 6：ブロック毎の処理クロック数

	Y0	Y1	Y2	Y3	U	V
色空間変換	16690	16422	14663	14844	15883	15727
DCT	82203	78683	79819	79361	70835	74614
量子化	15897	15753	15791	15859	15858	15970
ハフマン符号化	9056	5893	8480	8969	2772	4795

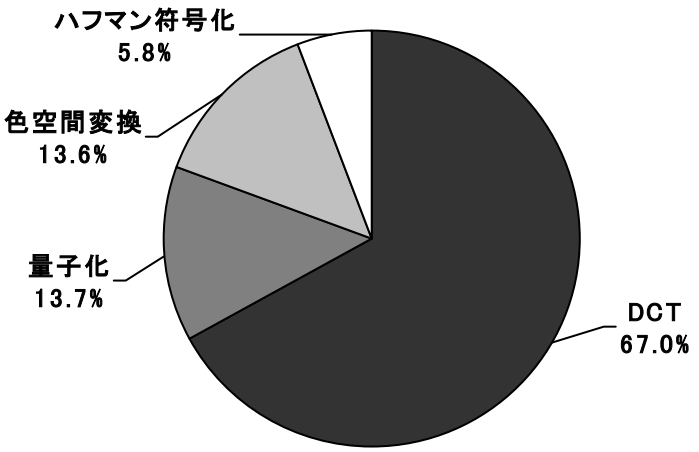

図 10：処理単位毎の負荷割合

図 10 から、処理全体におけて DCT の負荷が占める割合が非常に大きく、逆にハフマン符号化の負荷は小さいことが分かる。3.2 節に示したアルゴリズムから、1MCU のデータに対して各処理が行う乗除算回数は、DCT が 6144 回の乗算、色空間変換が 1152 回の乗算、量子化が 384 回の除算、ハフマン符号化は 0 回となっている。DCT は色空間変換の 5 倍の乗算を行っており、負荷の割合もほぼ 5 倍となっていることから、乗除算の回数が処理の負荷に密接に関係していると考えられる。また、色空間変換と量子化がほぼ同じ負荷となっていることから、対象の CPU における除算の負荷は、乗算の約 3 倍になっていると考えられる。ハフマン符号化は乗除算を行わないが、細かいテーブル参照や非常に複雑な分岐を行っているため、ある程度の負荷となっていると考えられる。表 6 に示したように、色空間変換、DCT、量子化の 3 処理はブロックの種類によって処理クロックはあまり変化していないのに対し、ハフマン符号化では処理クロック数が大幅に変動していることからも、ハフマン符号化における、データの種類に応じた複雑な分岐の影響が確認できる。

これらの結果から、ハード／ソフト協調システムにおいて、DCT のような演算量が大きく処理が単純な部分を優先してハードウェアで処理することで、全体の性能を効果的に向上することができると予測する。

4. JPEG エンコードモジュールの設計

4.1 ハードウェア設計の概要

JPEG エンコードにおける色空間変換、DCT（離散コサイン変換）、量子化、およびハフマン符号化の 4 つの処理を、それぞれハードウェアモジュールとして実現した。各モジュールは Verilog-HDL で記述しており、FoundationISE 6.3i を用いたトップダウン設計を行った[5]。また、ModelSim SE を用いたシミュレーションによって、モジュールの動作検証と評価を行った。

ソフトウェアによる処理の実現との対比を明確にするため、全てのモジュールにおいて回路規模よりも性能を重視した設計を行っている。具体的には、パイプライン化や演算器の並列化を積極的に行うことによって、スループットの向上を図っている。

また、FSL インタフェースとの接続を考慮し、データ入力、出力のそれぞれにおいて、準備完了を表す出力信号と、許可を表す入力信号のやりとりによって制御を行うための handshake ピンを備えている。図 11 に handshake によるデータの入出力制御のタイミングチャートを示す。

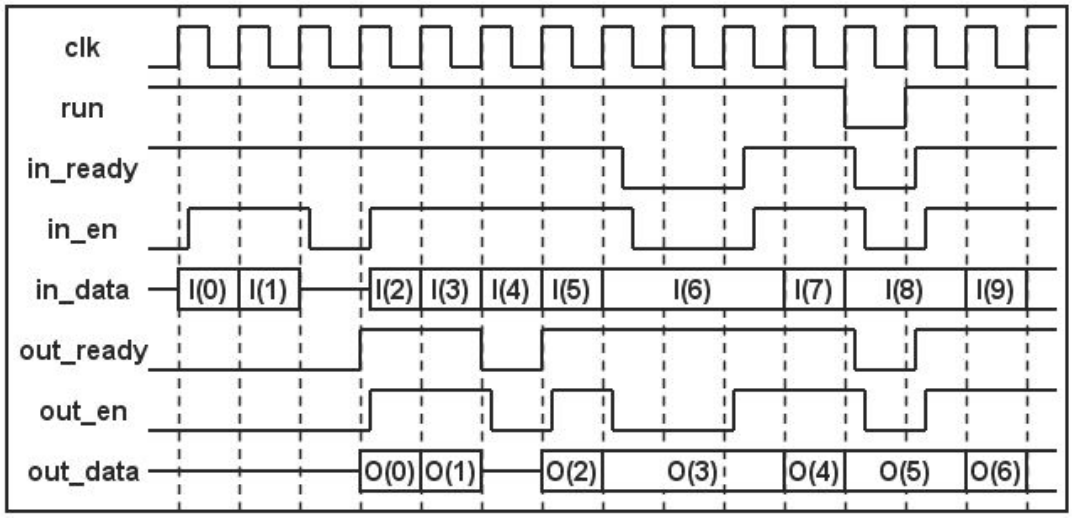

図 11：handshake による入出力の動作

まず、データ入力の動作では、モジュールがデータ入力可能な状態であれば、in_ready 信号を High として出力する。クロックの立ち上がり時に in_ready の状態が High で、入力信号 in_en が High ならば、in_data の値を入力として取り込む。

データ出力の動作では、モジュールがデータ出力可能な状態であれば、out_ready 信号を High として出力する。クロックの立ち上がり時に out_ready の状態が High で、入力信号 out_en が High ならば、out_data を次の値に更新する。out_ready が High で out_en が Low の場合は、out_data を更新できないため、ont_en が High になるまで in_ready を Low

にすることでデータ入力を停止する。

また、クロックの立ち上がり時に入力信号である run の値が Low の場合、モジュールは内部レジスタの値を更新しない。同時に、誤動作を防止するため in_ready と out_ready を Low として出力する。

次節より、色空間変換、離散コサイン変換、量子化、およびハフマン符号化の各モジュールについて、アーキテクチャの概要と評価を示す。

4.2 色空間変換

4.2.1 インタフェース

色空間変換モジュールの入出力インタフェースを図 12 に、信号線の説明を表 7 に示す。

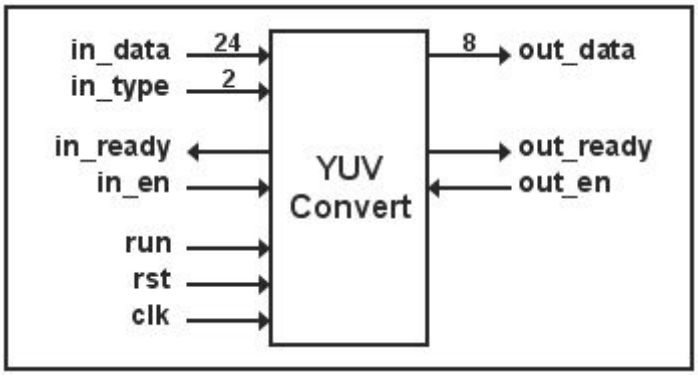

図 12：色空間変換モジュールのインタフェース

表 7：色空間変換モジュールの入出力信号

信号名	方向	幅	用途
out_data	output	8bit	YUV データの出力
out_ready	output	1bit	データ出力が可能かを示すフラグ（handshake）
out_en	input	1bit	データ出力のイネーブル信号（handshake）
in_data	input	24bit	RGB データの入力（RGB 各 8bit）
in_ready	output	1bit	データ入力が可能かを示すフラグ（handshake）
in_en	input	1bit	データ入力のイネーブル信号（handshake）
in_type	input	2bit	ブロックの種類（YUV）を指定
run	input	1bit	モジュール全体の動作イネーブル信号
rst	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

図 11 に示したように、clk の立ち上がり時に handshake と run の状態に応じて、入力データの取り込みや出力データの更新を行う。in_type は、入力の RGB データに対して YUV

のどの演算を行うかを制御する信号で、00 なら Y、01 なら U、10 なら V を計算し、out_data として出力する。

4.2.2　内部構成

色空間変換モジュールは、3＊3 の積和演算部と定数テーブル、パイプライン動作のためのレジスタ、および handshake や定数テーブルアドレスなどの制御信号を生成する制御部から構成される。色空間変換モジュールの主なデータパスを図 13 に示す。

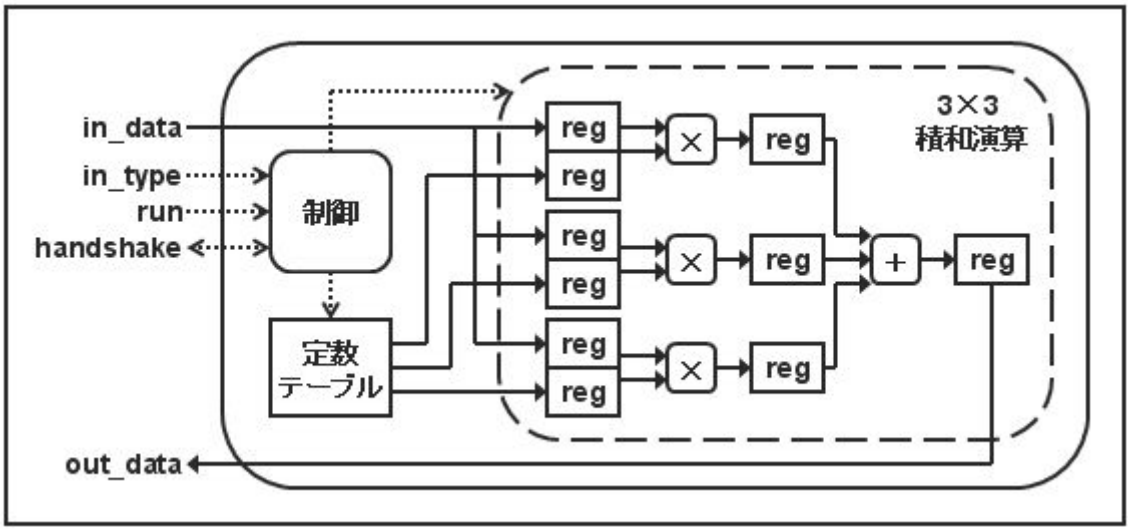

図 13：色空間変換モジュールのデータパス

定数テーブルには、3.2.2 節で示した YUV の計算式における係数を、それぞれ 8bit 左シフト（256 倍）して四捨五入した値を格納している。これは、小数部の精度を 8bit で表現した固定小数点による演算を行うためで、積和演算の終了後に小数部である下位 8bit を切り捨てることで、整数部のみの出力を行っている。入力データにおける RGB 値のフィールドはそれぞれ符号無し 8bit で、出力となる YUV の値は符号付き 8bit のため、定数テーブルの値は符号 1bit＋小数精度 8bit の 9bit としている。入力信号である in_type の値に応じて定数テーブルを参照することで、RGB 値と乗算する係数の読み出しを行う。

積和演算部は、符号無し 8bit の RGB 値と符号付き 9bit の係数との乗算を並列に行い、それらの結果を加算する。小数部 8bit の切り捨て時に四捨五入を行うため、0.5 を 8bit 左シフトした値である 128 を同時に加算している。また、Y を計算する場合は、加算結果から 128 を減算するレベルシフトを行うため、整数部 8bit の最上位 bit を反転する。これらの演算により、YUV の値は整数部 8bit、小数部 8bit の 16bit で表され、小数部を切り捨てることで最終的に 8bit の値を出力する。

また、乗算の前後と加算の後にレジスタを用意することで、3 段パイプラインでの動作を

実現し、動作周波数の向上を図っている。パイプラインにおける入出力の制御は、run と handshake に応じてデータを伝播すると同時に、そのデータが有効な入力データかを表す in_en の値をフラグとして伝播し、加算後のレジスタに対応するフラグを out_ready の値とすることで実現している。

4.2.3　シミュレーションによる評価

対象 FPGA における、色空間変換モジュールの回路規模と性能を表 8 に示す。スライスおよびゲート数と、最高動作周波数は、FoundationISE による配置配線のレポートから算出している。

表 8：色空間変換モジュールの評価

スライス数	ゲート数	最高動作周波数	最大スループット
199/6912	4418	47.6MHz	744000block/s

色空間変換モジュールは 3 段パイプラインで動作するため、入力から出力までのレイテンシは 3 クロックとなる。ただし、出力ができない場合（run が Low、もしくは out_ready が High で out_en が Low）は、内部レジスタの値を更新しないためレイテンシが大きくなる。1 ブロックの処理に必要な最小クロック数は 67 クロックであるが、複数のブロックを連続して滞りなく処理する場合はレイテンシを無視できるため、1 ブロックあたり 64 クロックとなる。よって、最高周波数で動作した場合、表 8 に示すように 1 秒間では最大 74 万ブロックを処理できることになる。

4.3　離散コサイン変換

4.3.1　インタフェース

離散コサイン変換（DCT）モジュールの入出力インタフェースを図 14 に、信号線の説明を表 9 に示す。

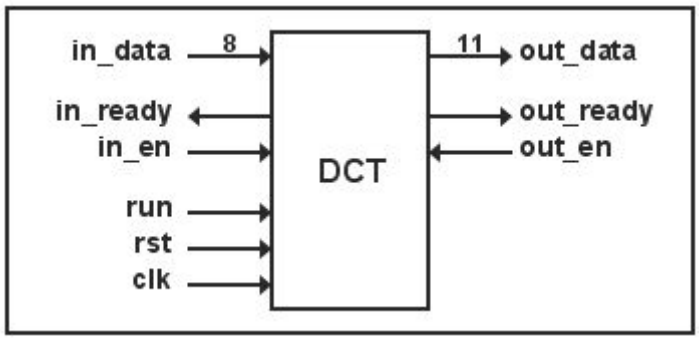

図 14：DCT モジュールのインタフェース

表 9：DCT モジュールの入出力信号

信号名	方向	幅	用途
out_data	output	11bit	DCT 係数の出力
out_ready	output	1bit	データ出力が可能かを示すフラグ（handshake）
out_en	input	1bit	データ出力のイネーブル信号（handshake）
in_data	input	8bit	YUV データの入力
in_ready	output	1bit	データ入力が可能かを示すフラグ（handshake）
in_en	input	1bit	データ入力のイネーブル信号（handshake）
run	input	1bit	モジュール全体の動作イネーブル信号
rst	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

DCT モジュールは、色空間変換モジュールと同様に、clk の立ち上がりで handshake と run の状態に応じてデータの入力、出力を行う。1 ブロック 64 個の入力データに対して、行方向と列方向の 2 度の 1 次元 DCT を行うことで 2 次元 DCT を実現しているため、内部状態は 2 つに分かれる。そのため、入出力動作は 1 ブロック分の YUV データの入力と、1 ブロック分の DCT 係数の出力とを交互に行うことになる。

4.3.2　内部構成

DCT モジュールは、図 15 に示すように、8＊8 の積和演算部、コサインテーブル、YUV データの入力バッファ、中間結果を格納する 8＊8 のバッファ、および制御部などから構成される。

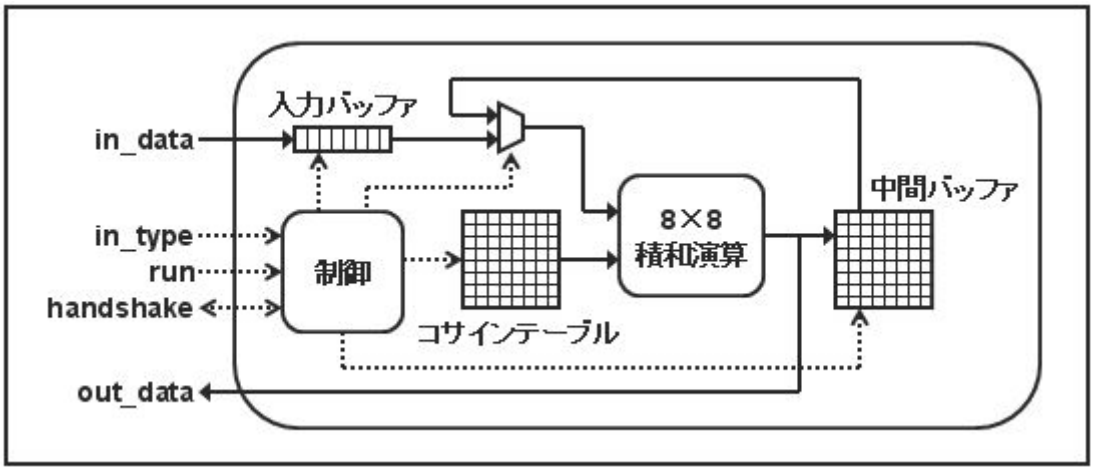

図 15：DCT モジュールのデータパス

コサインテーブルには、3.2.3 節で示した 1 次元 DCT の計算式のうち、コサインを含む係数部を 8＊8 通りであらかじめ計算して格納している。色空間変換の場合と同様に、テーブルの値は固定小数点で表現している。DCT 係数は符号付き 11bit の値で出力するため、コサインテーブルの値は符号 1bit＋小数精度 10bit の合計 11bit（1024 倍）としている。制御部で生成される列アドレスに応じて、1 列の 8 データを 1 度に読み出すことができる。

積和演算部の構造は、基本的には色空間変換モジュールの積和演算部と同様であり、3 段のパイプラインで動作する。8＊8 の積和演算を行うため乗算器を 8 個用意しており、符号付き 11bit の乗算を 8 並列で行った後、それらを加算し整数部 11bit を出力する。

入力バッファは、積和演算で必要となる 1 行分の入力データを用意するため、1 クロック毎に入力される YUV データを順に 8 点格納する。YUV データの行方向の DCT 結果は中間バッファに格納される。行方向の DCT は、入力バッファの 8 段と積和演算の 3 段、中間バッファへの書き込みの合計 12 段のパイプラインで動作する。1 ブロック分の行方向 DCT に続き、中間バッファを列方向に読み出して積和演算の入力とすることで、列方向の DCT を行う。列方向 DCT 時は入力バッファと中間バッファへの書き込みは必要無いため、色空間変換の動作と同様に 3 段パイプラインでの動作となる。

4.3.3　シミュレーションによる評価

対象 FPGA における、DCT モジュールの回路規模と性能を表 10 に示す。スライスおよびゲート数と、最高動作周波数は、FoundationISE による配置配線のレポートから算出している。

表 10：DCT モジュールの評価

スライス数	ゲート数	最高動作周波数	最大スループット
1101/6912	33810	48.3MHz	377000block/s

DCT モジュールは、入力データに対して 12 段のパイプラインで動作している。1 ブロック分のデータを入力した時点で、2 度目の積和演算となる列方向の DCT を行うために入力を受け付けなくなる。1 ブロックの入力が完了した 12 クロック後から 1 ブロック分のデータ出力が開始される。出力が終了する 8 クロック前からバッファリングのために再び入力を受け付けるようになる。よって、1 ブロックのデータの入力開始から出力完了まで 140 クロック必要となるが、複数ブロックを連続して処理する場合は 12 段パイプラインのレイテンシを無視できるため、1 ブロックあたり 128 クロックとなる。よって、最高周波数で動作した場合、表 10 に示すように 1 秒間では最大 37 万ブロックを処理できることになる。

4.4 量子化

4.4.1 インタフェース

量子化モジュールの入出力インタフェースを図 16 に、信号線の説明を表 11 に示す

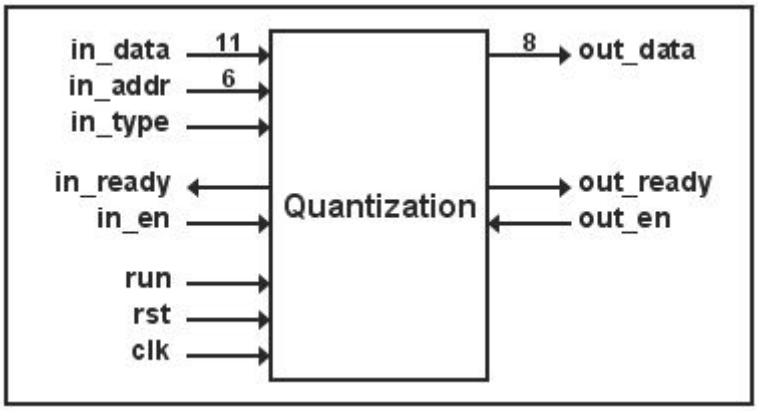

図 16：量子化モジュールのインタフェース

表 11：量子化モジュールの入出力信号

信号名	方向	幅	用途
out_data	output	8bit	量子化データの出力
out_ready	output	1bit	データ出力が可能かを示すフラグ（handshake）
out_en	input	1bit	データ出力のイネーブル信号（handshake）
in_data	input	11bit	DCT 係数の入力
in_ready	output	1bit	データ入力が可能かを示すフラグ（handshake）
in_en	input	1bit	データ入力のイネーブル信号（handshake）
in_type	input	1bit	ブロックの種類（輝度、色差）を指定
in_addr	input	6bit	DCT 係数ブロックのアドレス入力
run	input	1bit	モジュール全体の動作イネーブル信号
rst	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

量子化モジュールは、clk の立ち上がり時に handshake と run の状態に応じて、入力データの取り込みや出力データの更新を行う。in_addr は入力 DCT 係数のブロックにおけるアドレスを示しており、量子化テーブルの値を選択するために用いる。in_type は入力 DCT 係数が輝度 Y と色差 U、V のどちらのブロックであるかを指定する信号であり、0 なら輝度用量子化テーブル、1 なら色差用量子化テーブルを用いる。

4.4.2　内部構成

量子化モジュールは、図 17 に示すように、除算器、量子化テーブル、および制御部などから構成される。

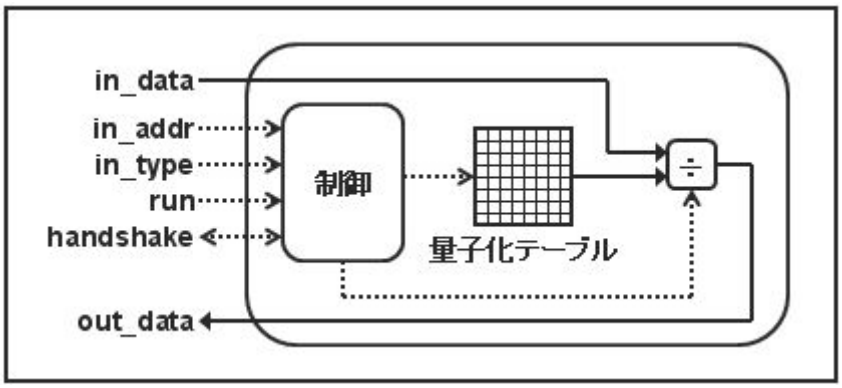

図 17：量子化モジュールのデータパス

量子化テーブルは、3.2.4 節の表 2、表 2 で示した輝度成分用と色差成分用の 2 種類からなる。量子化テーブルの値はそれぞれ符号無し 7bit で表される。in_type と in_addr に応じてテーブルの値を選択し、除数として除算器に入力する。また、除算による四捨五入を実現するために、入力 DCT 係数に量子化テーブル値の半分（下位 1bit を切り捨てた 6bit の値）を加算した値を、被除数として除算器に入力する。

除算器は、符号付き 11bit の被除数と符号無し 7bit の除数を入力とし、符号付き 8bit の商を出力する。除算はシフトと減算、剰余の正負判定などで実現しており、各桁の演算の前後にレジスタを挿入しているため、12 段パイプラインで動作する。

除算器の前後にはレジスタを持たないため、量子化モジュールは 12 段のパイプライン動作となる。パイプラインにおける入出力の制御は、run と handshake に応じてデータを伝播すると同時に、in_en の値を 12 段のフラグ用レジスタで伝播し、除算の終了に対応する 12 段目のフラグを out_ready の値とすることで実現している。

量子化モジュールを対象となるシステムに実装する場合、例えば DMA などで RAM を利用する時は、モジュールのコントローラにおいて出力データのアドレスをジグザグに生成することで、量子化と同時にジグザグスキャンを行うことができる。本研究では FIFO を用いて連続データ転送を行う FSL バスを用いるため、ジグザグに並び替えたデータを一時的に格納するバッファを作成した。実装の際には、量子化モジュールとジグザグスキャンモジュールを接続し、1 つのブロックとして FSL バスに接続する。ジグザグスキャンモジュールは、図 18 に示すように 1 ブロックのデータを格納する RAM と、ROM を用いたジグザグアドレスのテーブルから構成される。

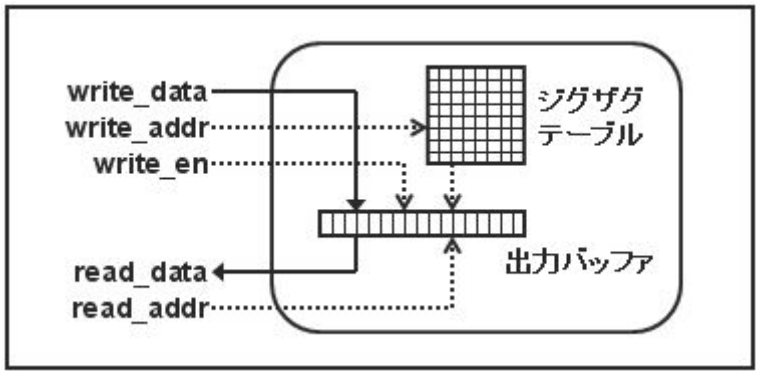

図 18：ジグザグスキャンモジュールのデータパス

4.4.3　シミュレーションによる評価

対象 FPGA における、量子化モジュールとジグザグスキャンモジュールからなる量子化ブロックの回路規模と性能を表 12 に示す。スライスおよびゲート数と最高動作周波数は、FoundationISE による配置配線のレポートから算出している。

表 12：量子化モジュールの評価

スライス数	ゲート数	最高動作周波数	最大スループット
276/6912	8725	85.9MHz	614000block/s

量子化モジュールは、12 段パイプラインで動作するため、単体では入力から出力のレイテンシは 12 クロックで、毎クロック入出力が可能である。ただし、実装時はジグザグスキャンモジュールに 1 ブロック分のデータを書き込んだ後に、順に読み出して出力を行うため、1 ブロックの処理に必要なクロック数は最低 140 クロックとなる。よって、最高周波数で動作した場合、1 秒あたり 61 万ブロックを処理できることになる。

4.5　ハフマン符号化

4.5.1　インタフェース

ハフマン符号化モジュールの入出力インタフェースを図 19 に、信号線の説明を表 13 に示す。

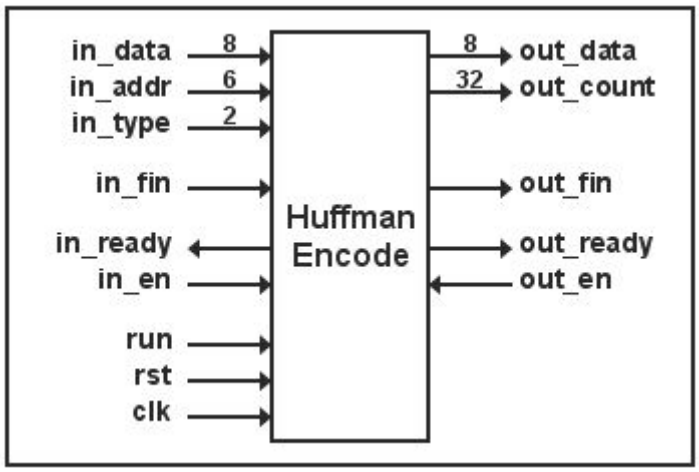

図 19：ハフマン符号化モジュールのインタフェース

表 13：ハフマン符号化モジュールの入出力信号

信号名	方向	幅	用途
out_data	output	8bit	ハフマン符号の出力
out_ready	output	1bit	データ出力が可能かを示すフラグ（handshake）
out_en	input	1bit	データ出力のイネーブル信号（handshake）
out_count	output	32bit	出力ハフマン符号のカウント（Byte）
out_fin	output	1bit	全画像データの出力終了を示すフラグ
in_data	input	8bit	量子化データの入力
in_ready	output	1bit	データ入力が可能かを示すフラグ（handshake）
in_en	input	1bit	データ入力のイネーブル信号（handshake）
in_type	input	2bit	ブロックの種類（YUV）を指定
in_addr	input	6bit	量子化データブロックのアドレス入力
in_fin	input	1bit	全画像データの入力終了を示すフラグ
run	input	1bit	モジュール全体の動作イネーブル信号
rst	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

ハフマン符号化モジュールは、clk の立ち上がり時に handshake と run の状態に応じてデータの入力、出力を行う。in_addr は入力量子化データのブロックにおけるアドレスを入力する。in_type は量子化データが YUV のどのブロックであるかを指定する信号であり、00 なら Y、01 なら U、10 なら V を示している。モジュールが生成したハフマン符号列は、8bit 毎に out_data として出力される。out_count は出力したハフマン符号の総 Byte 数をカウントして出力する。in_fin と out_fin は全画像データの処理終了時に用いるもので、

in_fin が High として入力された場合、8bit 未満の未出力のハフマン符号列があれば、0 で埋めて 8bit にして出力し、処理終了を表す out_fin の値を High にする。

4.5.2　内部構成

ハフマン符号化モジュールは、図 20 に示すように、入力・前処理部、符号生成部、符号結合・出力部という 3 つの主な処理ブロックと、符号生成部において用いるゼロランレングスのカウンタ、そして全体の動作を制御する制御部から構成される。

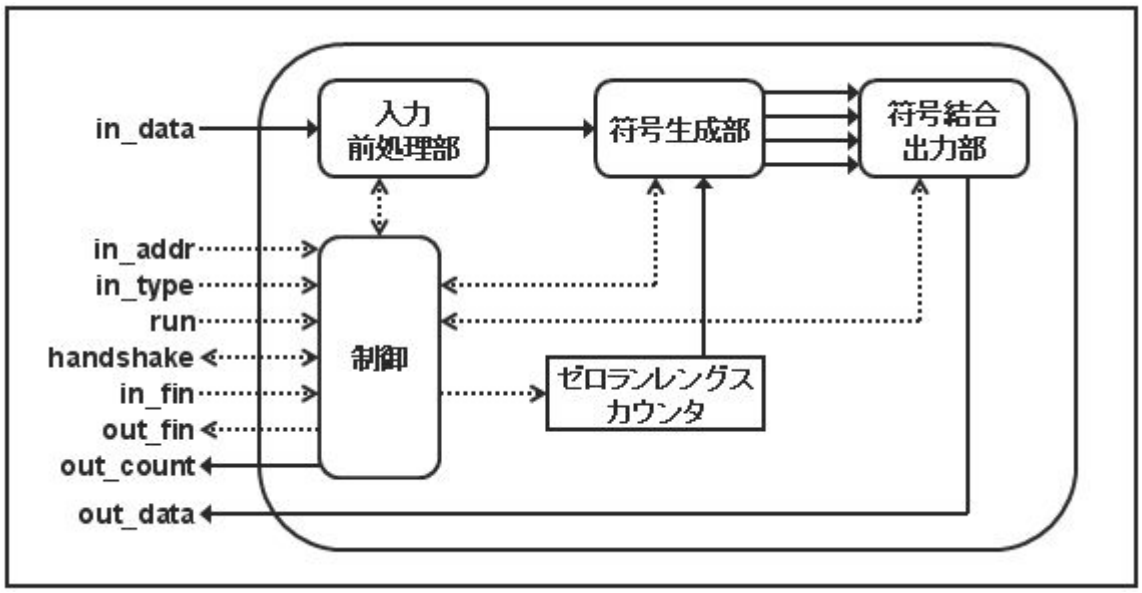

図 20：ハフマン符号化モジュールのデータパス

入力・前処理部は、図 21 に示すように構成される。入力信号である type は入力データの種類を表しており、00 は Y ブロックの DC 成分、01 は U ブロックの DC 成分、10 は V ブロックの DC 成分、11 は AC 成分であることを示している。入力信号の read_en が High の場合、入力データが DC 成分であれば、YUV の種類に応じて直前ブロックの DC 成分との差分を取り、出力レジスタに格納する。同時に、差分用レジスタを入力データの値に更新する。AC 成分の場合は入力データをそのまま出力レジスタに格納する。また、入力データが DC 成分、0 で無い AC 成分、およびブロックの最後の要素の場合、有効な成分であると判定する。

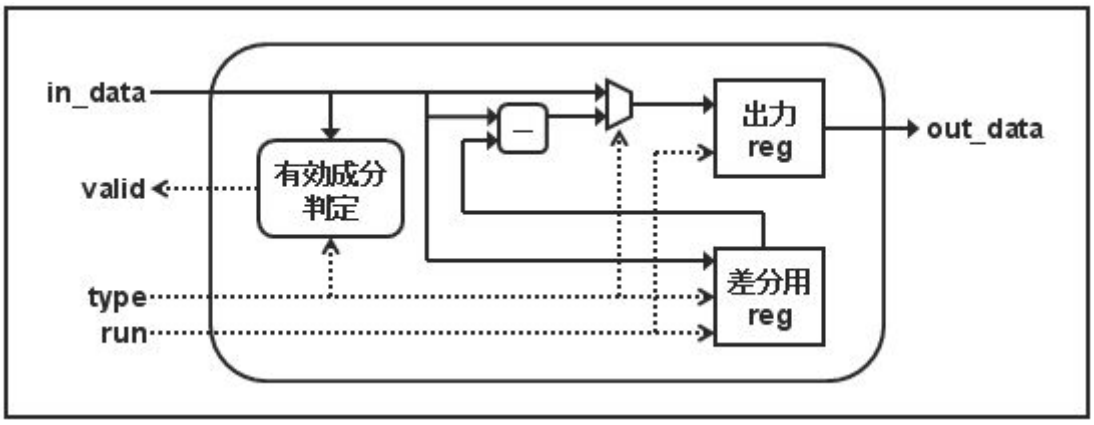

図 21：入力・前処理部のデータパス

符号生成部は、図 22 に示すように、グループ符号を格納したハフマンテーブルと、付加 bit の生成部から構成される。入力信号の run が High の場合、まず、付加 bit の生成部において、3.2.5 節の表 4 で示したように、入力データの絶対値からグループ番号を決定し、付加 bit とその長さをレジスタに格納する。次に、グループ番号とゼロランレングスを用いて、入力データの種類を表す type の値に応じたハフマンテーブルを参照し、グループ符号とその長さをレジスタに格納する。type の値は 00 なら輝度ブロックの DC 成分、01 なら輝度ブロックの AC 成分、10 なら色差ブロックの DC 成分、11 なら色差ブロックの AC 成分を示している。入力データが 0 の時は EOB 符号、ゼロランレングスが 16 以上の場合は ZRL 符号を出力する。

ゼロランレングスカウンタは、入力データが有効成分では無い場合にカウンタを増やし、有効成分の符号生成時にリセットする。ZRL 符号を生成する場合はカウンタを 16 減らす。

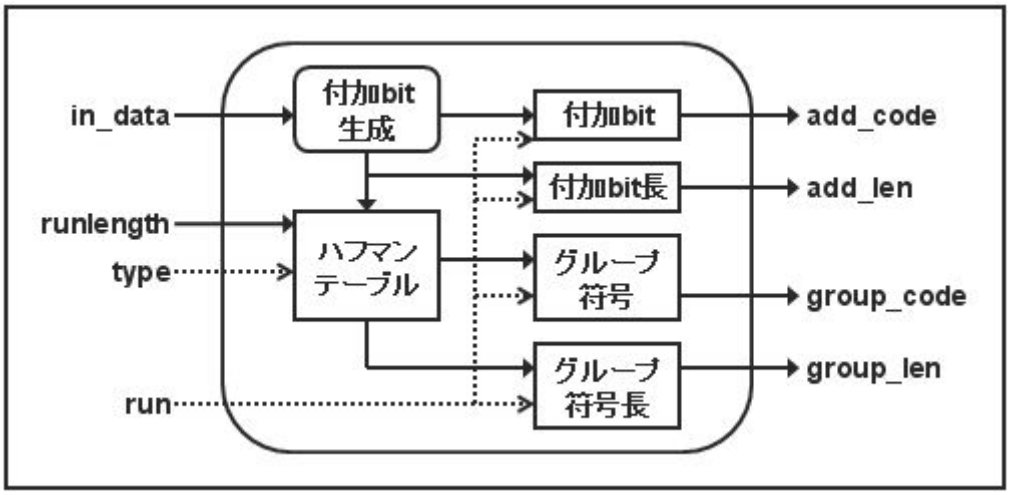

図 22：符号生成部のデータパス

符号結合・出力部は、図 23 に示すように構成される。in_ready と run がともに High の場合、バッファに格納されている未出力の符号列に、バレルシフタを用いてグループ符号、付加 bit の順で入力データを結合してバッファを更新する。グループ符号は最大 16bit、付加 bit は最大 8bit のため、符号列バッファは 32bit としている。符号列の長さが 8bit 以上の場合、out_ready を High として先頭 8bit を出力する。符号列の長さが 16bit 以上の場合や、出力符号列が 0xFF の場合は、連続して出力を行う必要があるため、in_ready の値を Low とする。入力信号の dump_remain が High の場合は、符号列の長さが 8bit 未満でも 0 を埋めて 8bit として出力する。

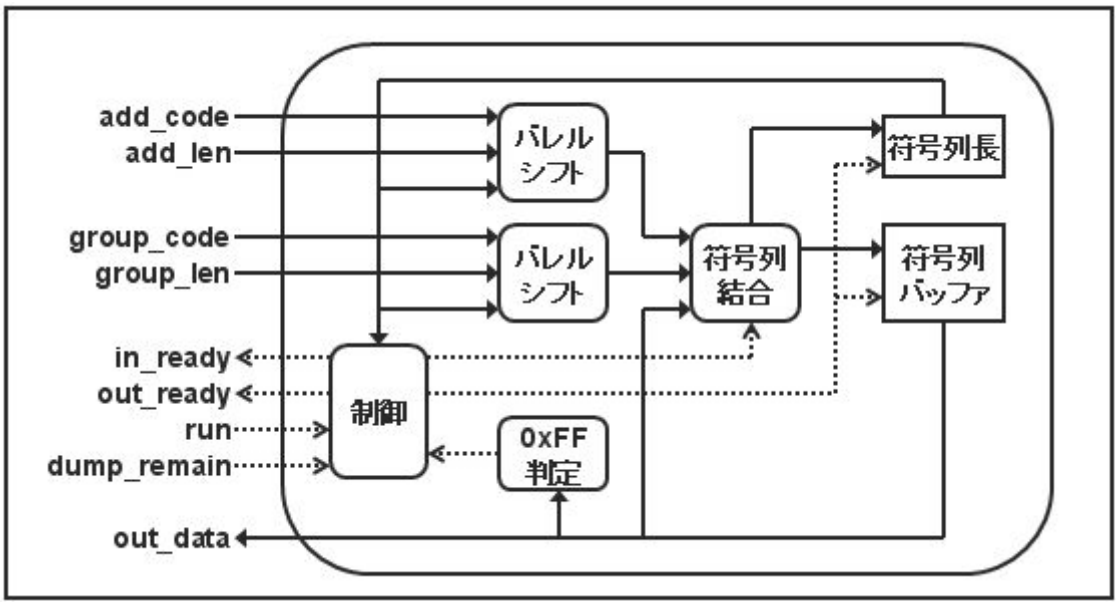

図 23：符号結合・出力部のデータパス

ハフマン符号化モジュールは、これらの 3 つの処理ブロックに応じた 3 段パイプラインで動作する。入力・前処理部で格納した値が有効成分の場合、符号生成部の run フラグを High にし、有効でない場合はゼロランレングスのカウントフラグを High にする。符号結合・出力部の run フラグは、符号生成部の run フラグの値を伝播する。符号結合・出力部において、連続して符号列を出力する場合や、handshake により出力が不可能な場合は、符号生成部にストールを発生させる。同時に、入力・前処理部については、符号生成部の run フラグが High であればストールを発生させる。また、符号生成部が ZRL 符号を生成する場合、その後に行われる有効成分の符号生成が終了するまで、入力・前処理部にストールを発生させる。

4.5.3　シミュレーションによる評価

対象 FPGA における、ハフマン符号化モジュールの回路規模と性能を表 14 に示す。スライスおよびゲート数と最高動作周波数は、FoundationISE による配置配線のレポートから算出している。

表 14：ハフマン符号化モジュールの評価

スライス数	ゲート数	最高動作周波数	最大スループット
698/6912	10191	29.8MHz	466000block/s

ハフマン符号化モジュールは、3 段パイプラインで動作するため、入力から出力までのレイテンシは最短で 3 クロックとなる。ただし、1 データに対する符号長が長くなる場合などにおいて、符号生成部や入力・前処理部にストールを発生する可能性があるため、入力データの偏りによって 1 ブロックの処理に必要なクロック数は変動する。最短では 64 クロックで 1 ブロックを処理できるため、最高周波数で動作させた場合、理想的な最大スループットは 1 秒間に 47 万ブロック程度となる。本研究で扱う JPEG アルゴリズムに基づいた入力の量子化データに対しては、ModelSim によるシミュレーションから、1 ブロックの処理に平均 66 クロック程度必要であることを確認した。

5. JPEG エンコードシステムの実装と評価

5.1 JPEG エンコードシステムの実装

2.3 節の図 2 で示した FPGA によるハード／ソフト協調システムとして、JPEG エンコードシステムを構成する。色空間変換、DCT、量子化、およびハフマン符号化の 4 つの処理ブロックについて、3 章で示したソフトウェアと、4 章で示したハードウェアモジュールの組み合わせによって、JPEG エンコードの処理を実現する。また、ハード／ソフト協調システム全体の実行制御は、ソフトウェアによって行う。ハードウェアモジュールのシステムへの接続と、ソフトウェアによる全体の制御について、それぞれ説明する。

（1）ハードウェアモジュールの接続

FSL インタフェースを用いて、4 章で設計したハードウェアモジュールと、ソフトウェアによる処理やシステムの制御を行う CPU である Microblaze を接続する。

FSL バスとハードウェアモジュールとの接続は、図 24 に示すように、制御信号の調停を行うコントローラを介して行う。また、FSL バスは FIFO によって実現されているため、ブロックの種類やアドレスなどの情報はやりとりされない。そのため、モジュールがそれらの情報を必要とする場合は、コントローラにおいてデータ入出力の個数をカウントすることで信号を生成する。

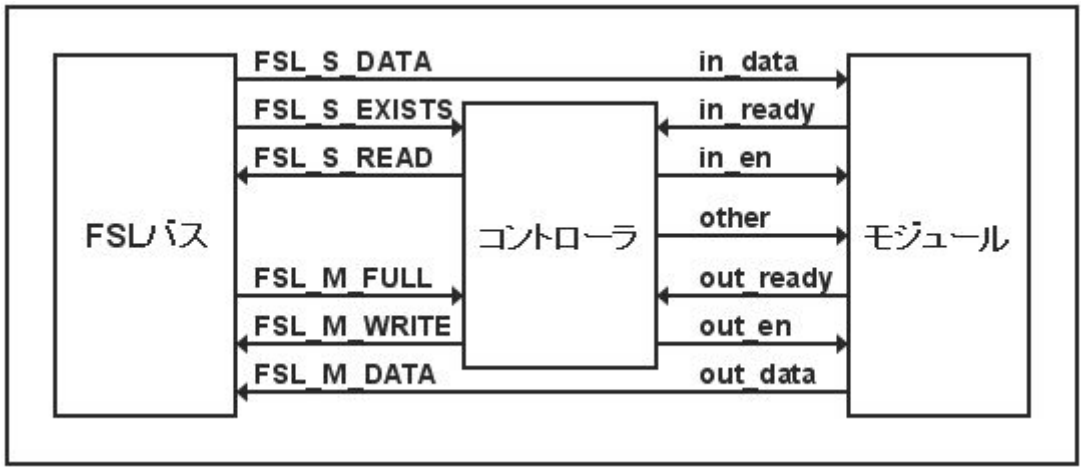

図 24：ハードウェアモジュールと FSL バスの接続

FSL バス側の信号は、自作モジュールと同様の handshake による制御を行うためのものである。FSL_S_EXISTS は FSL バスの FIFO にデータがあるかを示し、FSL_S_READ を入力することで FIFO を次の値に更新する。FSL_M_WRITE は FIFO への書き込みイネーブル信号であり、FSL_M_FULL は FIFO が一杯であるかを示している。コントローラによる FSL バスとモジュールのインタフェース調停の動作例を、図 25 のタイミングチャート

に示す。入力側では、モジュールからの in_ready と FSL バスからの FSL_S_EXISTS がともに High の場合、両方のモジュールにイネーブル信号を送る。出力側では、モジュールからの out_ready が High で、FSL バスからの FSL_M_FULL が Low の場合、両方のモジュールにイネーブル信号を送る。

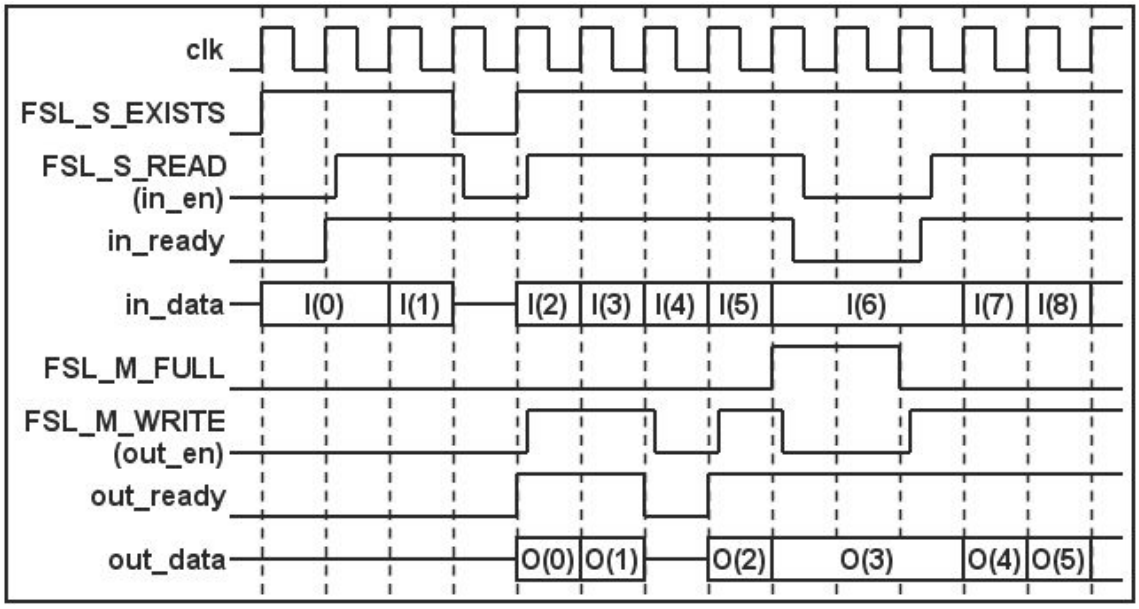

図 25：FSL コントローラの動作

また、Microblaze と FSL バスとの通信は、以下のような C 言語の記述によって行うことができる。bread は FSL バスからの 1 データの読み込み、bwrite は FSL バスへの 1 データの書き込みを行う。id は FSL バスの番号を、val は FSL バスとやりとりするデータを表している。read、write のそれぞれで、最大 8 本の FSL バスを接続できるため、id の値は 0～7 となる。FSL バスは 8、16 および 32bit での転送に対応しているが、Microblaze 側では現時点の制限として、val の値は符号無し 32bit だけの対応となっている。

```
microblaze_bread_datafsl(val, id);
microblaze_bwrite_datafsl(val, id);
```

（2）システム全体の実行制御

1 ブロック 64 画素分の画像データ毎に、色空間変換、DCT、量子化、ハフマン符号化の 4 つの処理を、ソフトウェアもしくはハードウェアで実行する。4 つの処理の間に、各段階での結果を格納する 1 ブロック分の配列を用意し、各処理はそれらの領域の読み出しと書き込みを行う。ハードウェアでの処理中にソフトウェアで他の処理を行うために、4 つの処理を 1 ブロックのデータ毎に逐次的に実行するのではなく、図 26 に示すようにパイプライ

ンで実行する。ただし、ソフトウェアによる制御では、処理を並列で実行することはできないため、各列の処理ステージは下から上の順で行う。図 26 で示した流れで実行することで、ハードウェアを利用した場合の処理全体の最適化を図ることができる。

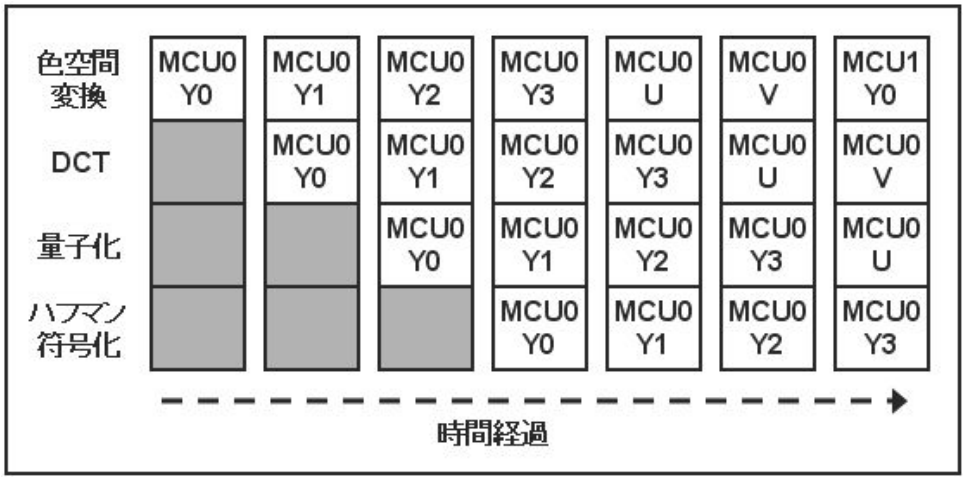

図 26：JPEG エンコードシステムの実行の流れ

5.2 ハード／ソフト分割パターン

ソフトウェアとハードウェアのそれぞれで実現した、色空間変換、DCT、量子化、およびハフマン符号化の 4 つの処理に対して、実際に FPGA を用いた協調システムとして実装する分割パターンを選択する。3.3.2 項の表 6 および図 10 で示したソフトウェアの実行結果における各処理の負荷の割合と、4 章で示した各ハードウェアモジュールの評価から、表 15 に示すような 6 種類の分割パターンを決定した。

表 15：JPEG エンコードシステムのハード／ソフト分割パターン

分割パターン	ハードウェア処理	ソフトウェア処理
A		色空間変換、DCT、量子化、ハフマン符号化、制御
B	DCT	色空間変換、量子化、ハフマン符号化、制御
C	DCT、量子化	色空間変換、ハフマン符号化、制御
D	色空間変換、DCT	量子化、ハフマン符号化、制御
E	色空間変換、DCT、量子化	ハフマン符号化、制御
F	色空間変換、DCT、量子化、ハフマン符号化	制御

DCT のソフトウェア処理における負荷は全体の 2/3 を占めており、ハードウェア化の効果が高いと見込まれるため、優先的にハードウェア処理としている。逆にハフマン符号化については、負荷が 6%程度で最も少なくなっていることと、回路規模が DCT に次ぐ大きさとなっていることから、ハードウェア処理を行うメリットが少ないと予測し、分割パターンは 1 種類としている。

5.3 分割パターン毎の評価

表 15 に示した A から F の 6 種類の分割パターンについて、それぞれ FPGA へ実装した結果を表 16 に示す。評価は回路規模を表す FPGA の使用スライス数、ソフトウェアのメモリ使用量、1MCU あたりの実行クロック数の 3 点から行っている。回路規模とメモリ量はそれぞれ開発ツールのレポートから算出し、実行クロック数は FPGA 上のシステムに接続したクロックカウンタにより計測した。実行時の動作周波数は全て 20MHz としている。

表 16：分割パターン毎の実装結果

分割 パターン	スライス数 （使用率）	プログラム メモリ量	1MCU あたりの 実行クロック数
A	1799（26%）	6532 Byte	694818 Clock
B	3187（46%）	5532 Byte	243104 Clock
C	3758（54%）	4920 Byte	161079 Clock
D	3748（54%）	5068 Byte	165166 Clock
E	4342（62%）	4456 Byte	83072 Clock
F	5425（78%）	2284 Byte	48677 Clock

まず、各パターンの 1MCU あたりの実行クロック数は、処理のハードウェア化によって DCT では 450000 クロック、量子化では 82000 クロック、色空間変換では 78000 クロック、ハフマン符号化では 35000 クロック程度削減されていることが分かる。これは、処理のハードウェア化によって、ソフトウェア実行時の負荷のほとんどを削減できることを示している。ただし、4 つの処理を全てハードウェア化した分割パターン F においても、50000 クロック近い処理が必要になることから、ハードウェア処理部を多くするほど、処理全体において CPU とハードウェアとの通信が占める割合が多くなることが分かる。現在、FSL バスの仕様より CPU 側のデータ幅が 32bit に限定されているため、ハードウェア側でバス利用の効率化を図ることで、さらに実行クロック数を削減できると考えられる。

次に、ハードウェア化による回路規模の増加量は、Microblaze と周辺機器を含めたシステムの基本構成要素のスライス使用率が 26%程度であるのに対し、DCT では 20%、量子化と色空間変換では 8%、ハフマン符号化では 16%程度の増加となっていることが分かる。これは、演算量の大きさに加え、条件分岐による制御の複雑さが、回路規模の増加に与える

影響が大きいことを示していると考えられる。

ソフトウェアのメモリ使用量の減少は、DCT では 1000Byte、量子化では 600Byte、色空間変換では 450Byte、ハフマン符号化では 2200Byte 程度となっていることが分かる。これは、条件分岐の複雑さと、ソフトウェアが使用するテーブルのサイズが大きく影響していると考えられる。

5.4 最適分割パターンの検討

5.3 節の結果を総合して、A から F の分割パターンを比較評価し、最適なパターンを決定する。比較評価のために、回路規模、メモリ量およびクロック数の 3 点について、A の値を 100 とした場合の各パターンの割合を表したグラフを図 27 に示す。

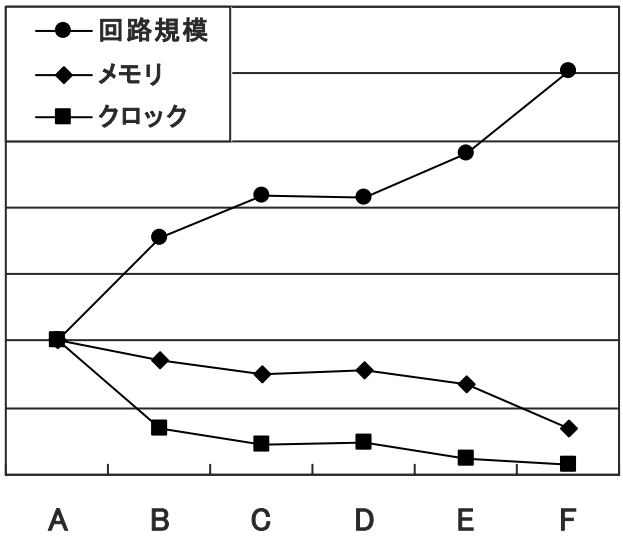

図 27：分割パターン毎の実装結果の比較

図 27 より、まず回路規模の増加が大きい A から B、E から F を比較すると、DCT をハードウェア化した B では、回路規模の増加は A の 80%弱と大きくなっているが、クロック数は A の 65%程度を削減できている。また、メモリ量も 20%弱削減されている。これに対し、ハフマン符号化をハードウェア化した F では、E からのメモリ量の削減は A の 40%近くになっているが、回路規模の増加量が A の 60%程度と大きく、さらにクロック数はほとんど削減されていない。よって、回路規模、メモリ量、クロック数を同じ割合で評価した場合、DCT のハードウェア化の方が優れていると考えられる。

次に、B に対して色空間変換と量子化をハードウェア化した E では、回路規模の増加が A の 60%、クロック数の削減が 20%程度、メモリ量の削減は 20%程度となっている。よって、DCT のハードウェア化に比べて効果が少ないことが分かる。

量子化をハードウェア化した C と、色空間変換をハードウェア化した D では、回路規模の増加はほぼ等しく、クロック数とメモリ量の削減は C の方が若干多いことが分かる。よ

って、量子化のハードウェア化の方が若干優れていると考えられる。

B から D、E から F の比較では、F が回路規模の増加が大きく、クロック数の削減が小さいため、色空間変換のハードウェア化の方が優れていると考えられる。ただし、メモリ量の削減を重視する場合は、ハフマン符号化をハードウェア化する選択肢も考えられる。

これらの点を総合すると、回路規模、メモリ量、クロック数を同じ割合で評価した場合、ハードウェア化の効果は DCT が最も高く、次いで量子化、色空間変換、ハフマン符号化という順番になることが分かる。ソフトウェアで全処理を行うパターン A から、この順番でハードウェア処理部を増やしていった場合の、回路規模に対する速度向上を図 28 のグラフに示す。グラフの横軸は各パターンのスライス数を表し、縦軸は実行クロック数の逆数である処理速度の向上を、A の値を 1 とした場合の比で表している。

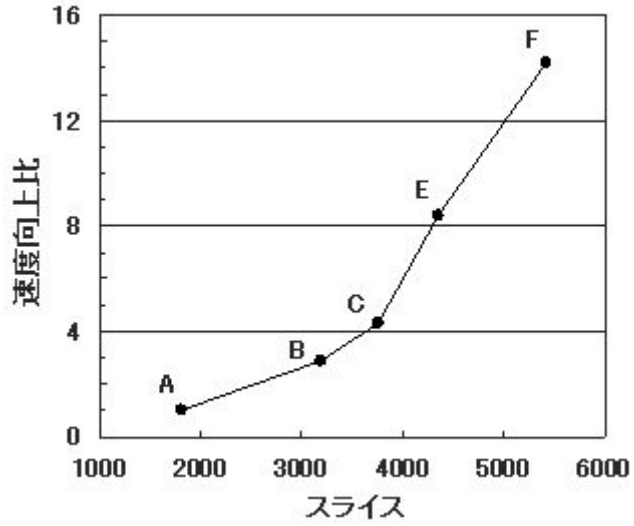

図 28：回路規模に対する速度向上

図 28 より、まずパターン A から B では、回路規模が 1.8 倍に対して速度向上が 3 倍近くになっている。さらに、パターン B から C、および C から E までは、ハードウェア処理が増加する毎に速度向上の傾きが大きくなっている。よって、色空間変換のハードウェア化までは、回路規模の増加に対して効率的な速度向上を得られることが分かる。これに対し、パターン F では A に対して 14 倍以上の速度向上は得られているが、C から E の速度向上と比較した場合、E から F の傾きは小さくなっている。よって、パターン F は E に比べて効率的な速度向上が得られていないことが分かる。

これらのことから、JPEG エンコーダにおける最適な分割パターンは、DCT、量子化、および色空間変換をハードウェア化したパターン E であるといえる。ただし、回路規模に余裕があり更なる速度向上を求める場合は、パターン F も選択肢となりえる。逆に、回路規模が制約を満たさない場合は、C、B、A の順で検討を行うのが適当である。

5.5 考察

本研究の目的としてかかげた、FPGA を用いたハード／ソフト協調システムに基づいた JPEG エンコーダのハード／ソフト分割の最適化について、その効果と今後の改善点を考察する。

設計手法の効果については、まずは FPGA 上にシステムを構築したことで、様々な分割パターンを実装することができ、高速かつ正確な検証結果をもとにして分割の最適化を検討できたことが挙げられる。実際にハードウェアの回路として実行を確認し、その評価を行ったことで、設計および実行結果の信頼性を高いものにすることができたと考えられる。さらに、CPU として Xilinx の IP である Microblaze を用いたことで、バス、メモリコントローラなどの周辺機器の IP や、C コンパイラなどの開発環境を利用することができた。そのため、機能部分のソフトウェアおよびハードウェアの設計に注力することができ、短期間で実装、評価まで行うことができた。研究背景で述べたように、LSI の設計において設計期間の短縮が大きな課題となっているため、この点は非常に有用であると考えられる。また、ソフトウェアとハードウェアの両方で処理を実現したことで、ソフトウェアでの処理量や複雑さと、ハードウェア化した場合の速度向上や回路規模の増大との相関性を考察することができた。今後の課題として、ソフトウェアの処理量や複雑さを、CPU の動的命令実行において演算命令や分岐命令、メモリアクセスなどの種類毎に統計として計測することで、本研究において考察した相関性を、より定量的な指標で表すことができると考えている。これにより、ソフトウェアの実装段階で、ハードウェア化する処理について取捨選択を行う基準を得ることができると考えている。

次に、改善点としては、まず評価手法の更なる検討が必要であると考えている。5.4 節における最適な分割パターンの検討は、回路規模、メモリ量、クロック数の 3 つの項目を同じ割合で評価したものであり、要求に応じてどの項目を重視するかといった重み付けを行うと、最適とされる分割パターンは異なったものとなる。また、最適とされた分割パターンが要求を満たしていない場合、異なる分割パターンを選択しなければならない。よって、様々な要求に応じた定量的な評価のためには、重み付けやパターンの足きりを行う評価式が必要であると考えられる。共同研究者である古川氏や梅原氏の論文において、これらの処理を行う評価式が提案されている[10][11]。しかし、重み付けの値に対する評価値の偏りが大きく、特に全処理をハードウェアもしくはソフトウェアで実装したパターンに対しては、他のパターンとの比較を行うのが難しくなることが多いため、今後の更なる改善が必要であると考えられる。また、現在の評価項目は回路規模、メモリ量、クロック数の 3 点となっているが、消費電力や開発工数などを評価項目として加えることで、より複雑な要求に応じて最適な分割パターンを検討できるようにすることが必要であると考えられる。

6. おわりに

本研究では、ハード／ソフト協調設計において重要な課題となっている、ハード／ソフト分割の最適化について考察するため、FPGA とソフト・マクロ CPU を用いた協調システムを構築した。さらに、実際に JPEG エンコーダというアプリケーションに対し、協調システムを用いて FPGA に実装することで、設計手法の評価を行った。JPEG エンコード処理を 4 つの機能ブロックに分割し、それぞれソフトウェアとハードウェアモジュールの両方を設計した後に、様々なハード／ソフトの分割パターンでの実装と比較評価を行うことによって、最適な分割パターンを決定した。また、ソフト／ハードの両方で処理を実現したことで、ソフトウェアでの処理量や複雑さと、ハードウェア化した場合の速度向上や回路規模の増大との相関性を考察した。

FPGA による協調システムを用いた手法の効果としては、複数の分割パターンについて短期間での実装、評価を行うことができた点と、実際に FPGA 上の回路として実行を検証することで、設計および実行結果の信頼性を高めることができた点などから、その有用性を確認することができた。

今後の課題として、様々な要求に対して最適な分割パターンの決定を定量的に行うためには、評価手法の更なる改善が必要である。また、ソフトウェア実行時の動的実行命令に対し、命令の種類毎に統計を取ることで、ハードウェア化した場合の速度向上や回路規模の増大との関係を調査し、設計手法の効果を高めることが発展として挙げられる。

半導体の集積技術や、実装対象となる信号処理技術は日々進歩しており、複雑化する要求に対応するため、設計技術も発達していく必要がある。その中で、ハードウェアとソフトウェアの両方の知識を持った学生を育てることを目的として、情報学から電子工学にいたる様々な研究活動が行われている本研究室は、ハード／ソフト協調設計を研究する環境として非常に恵まれている。これらの研究活動が更なる発展を遂げ、今後の LSI 設計技術に大きく貢献するような成果に到達することを願ってやまない。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導をいただきました山崎勝弘教授、小柳滋教授に深く感謝いたします。

また、本研究の共同研究者である古川氏、梅原氏、および本研究にあたり様々な助言を下さった高性能計算研究室の皆様に心より感謝いたします。

参考文献

[1] Jorgen Staunstrup, Wayne Wolf: Hardware/Software Co-Design: Principles and Practice, Kluwer Academic Publishers, 1997.

[2] 小野定康, 鈴木順司: わかりやすい JPEG/MPEG2 の技術, オーム社, 2001.

[3] 越智宏, 黒田英夫: JPEG&MPEG 図解でわかる画像圧縮技術, 日本実業出版社, 1999.

[4] 貴家仁志: よくわかるディジタル画像処理, CQ 出版, 1996.

[5] 深山正幸, 北山章夫, 秋山純一, 鈴木正國: HDL による VLSI 設計, 共立出版, 1999.

[6] ISO/IEC 10918-1, ITU-T Recommendation T.81: Information technology - Digital compression and coding of continuous-tone still images - Requirements and guidelines, 1992.

[7] Independent JPEG Group: http://www.ijg.org/

[8] 池田修久: ハード／ソフト・コラーニングシステム上での FPGA ボードコンピュータの設計と検証, 立命館大学理工学部情報学科修士論文, 2004.

[9] 池上広済: PC クラスタ上での OpenMP による JPEG エンコーダの並列化, 立命館大学理工学部情報学科卒業論文,2003.

[10] 古川達久: FPGA 上でのソフト・マクロ CPU によるハードウェア／ソフトウェア分割手法の研究, 立命館大学理工学部情報学科修士論文, 2005.

[11] 梅原直人: ハード／ソフト最適分割を考慮した AES 暗号システムと JPEG エンコーダの設計と検証, 立命館大学理工学部情報学科卒業論文, 2005.

[12] 夏目貴将, 飯山真一, 本田晋也, 富山宏之, 高田広章: エンジン制御システムの HW/SW コデザイン, 情報処理学会研究報告, SLDM, システム LSI 設計技術, Vol.2003, No.120, pp127-132, 2003.11.

[13] 田川博規, 小原俊逸, 戸川望, 柳沢政生, 大附辰夫: ハードウェア IP の応答時間を考慮したプロセッサコアのハードウェア/ソフトウェア分割手法, 情報処理学会研究報告, SLDM, システム LSI 設計技術, Vol.2003, No.7, pp93-98, 2003.1.

[14] 小原俊逸, 田川博規, 戸川望, 柳沢政生, 大附辰夫: ハードウェア IP の応答時間を考慮したプロセッサコア合成システム, 情報処理学会研究報告, SLDM, システム LSI 設計技術, Vol.2003, No.7, pp87-92, 2003.1.

[15] 関根敦司, 後藤源助, 多田十兵衛: DCT 向けプロセッサの構造最適化に関する研究, 情報処理学会研究報告, SLDM, システム LSI 設計技術, Vol.2004, No.5, pp13-16, 2004.1.

[16] Shinsuke Kobayashi, Kentaro Mita, Yoshinori Takeuchi, Masaharu Imai: JPEG Encoder Design Space Exploration Using the ASIP Development System, IPSJ Journal, Vol.44, No.5, pp1190-1201, 2003.5.