修士論文

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

氏　　名	：	古川 達久
学 籍 番 号	：	6124030200-3
指 導 教 員	：	山崎 勝弘 教授
		小柳 滋 教授
提 出 日	：	2005 年 2 月 15 日

立命館大学大学院　理工学研究科　情報システム学専攻

目次

図目次

表目次

内容梗概

本論文では，現在のシステム LSI 設計において，必要となるハード/ソフト・コデザインについて述べ，その中でも，「ハードとソフトの分割探索」に注目し，FPGA とソフト・マクロ CPU を用いてハードウェアとソフトウェアの分割を意識したシステム設計を行った．

そのソフトウェア処理を行う CPU には Xilinx 社のソフト・マクロ CPU である MicroBlaze を用い，この MicroBlaze システムに VerilogHDL で記述した専用ハードウェアを接続することで，ひとつのシステムとした．評価用ボードには Xilinx 社の 60 万システムゲートの Spartan-Ⅱ E(XC25600E-6FG456C) を搭載している MEMEC 社 DS-KIT-MB-S2E6LC-EURO を用いた．

対象アプリケーションとして，AES 暗号化，JPEG エンコーダという性質の異なる二つをとりあげた．それぞれのアプリケーションを C 言語でプログラミングを行い，アルゴリズムの特性と，CPU 過負荷部を調査し，その後，その調査結果及び，回路規模，使用メモリ量，再利用性を考慮し，様々な切り分けパターンを選出した．すべての切り分けパターンを FPGA 上にシステムとして実装し，動作検証とデータ収集を行い，考案した評価式を用いて切り分け結果を評価した．

この研究から，様々な要求仕様に対して，最適な切り分けパターンを見つけ出す一連の実験フローを確立した．また，このソフト・マクロ CPU を用いた実験フローを分割探索手法として提案した場合の有用性を示した．

1. はじめに

近年，半導体はありとあらゆる電化製品に搭載され，高機能化，高性能化，低消費電力化という要望が常に存在している．これらに応じるべく，テクノロジは著しく発展しており，1 チップ上に何百万ゲートといった規模の LSI の製造も可能となっている．さらに，新製品の発売サイクルも速まっており，最も速い発売サイクルであると言われている携帯電話に至っては，1 つのメーカから年に 4，5 度も新製品を発売するケースも少なくない．

しかし，テクノロジの進歩と納品サイクルが速いにもかかわらず，LSI の設計生産性の向上が追いついていないため，設計生産性危機としてこれが問題となっている[1][2][3]．設計生産性危機に対処するため，ハード/ソフト・コデザインの技術が注目されている[4]．

ハード/ソフト・コデザインとは，対象となる組み込みシステムに対して，システム設計の段階からハードウェア設計者とソフトウェア設計者が協調して，コスト，性能，消費電力などを考慮した上で，システム全体が最適になるようなシステムの仕様を決定し，効率的に設計する手法である．ハード/ソフト・コデザインは大きく次の 3 つに分けることができる．

「ハード/ソフト協調検証」システム LSI 設計は SoC による実現が主流となり，1 チップ上に CPU やメモリを搭載することが可能となっており，専用ハードウェアとそれらを協調させて動作させることになる．そういった場合，専用ハードウェア，またはソフトウェアを各々単体で検証するだけでなく，より実機に近い協調動作させた状態で検証する必要が出てくる．

「ハードとソフトの同時開発」前述したように，納品サイクルが速くなる一方，設計規模は拡大の一途をたどっており，従来のようにハードウェアを開発した後に，それを動作させるソフトウェアを開発するといった開発フローでは，間に合わなくなってきている．そこで一手法として，システムレベル記述言語(SpecC，SystemC など)を用いてシステム全体を記述し，同時に開発を進め設計期間の短縮を図る方法が注目されている．

「ハードとソフトの分割探索」組み込みシステムにおいて，要求からシステム仕様を決定段階でのハードウェアとソフトウェアの切り分けが，最終的なシステム LSI の良し悪しを左右する．そこで，コスト，性能，消費電力などの制約を与えた上で，あらゆる方面を考慮した上で最適な切り分けを見つける必要がある．

本論文では，これらの内で，特に「ハードとソフトの分割探索」に注目し，FPGA とソフト・マクロ CPU を用いてハードウェアとソフトウェアの分割を意識したシステム設計を行う．その実験過程と結果から，対象アプリケーションの特性と，要求に合った最適な切り分けとの関連性を見出し，今後の研究において，有用な分割手法の提案を目指す．

今回，用いるソフト・マクロ CPU とは，HDL データの形で提供されるマイクロプロセッサの IP コアである．ユーザによって，カスタマイズが可能であり，ユーザロジックと一緒に論理合成が可能であるため，短期間で無駄のないシステムを設計することができる．

なかでも，Xilinx 社が提供する 32bit のソフト・マクロ CPU である MicroBlaze を用いる．

また，本論文では，この手法の有用性を検証するための対象アプリケーションとして，AES と JPEG を選択した．AES は米商務省標準技術局によって 2001 年に標準認定された共通鍵方式の暗号化技術である[5]．同暗号化技術はワイヤレス LAN などで使用されており，1976 年以降 20 年以上標準となっていた DES の安全性の問題から，その代わりとなる暗号化技術として定められた．JPEG は現在，最も一般的な静止画像データ圧縮方式の 1 つである[14][15]．圧縮の際に，画質劣化を許容する方式と，まったく劣化しない方式を選択することができるが，本研究では前者を取り扱い，中でもベースラインと呼ばれる方式を取り扱う．

これらのアプリケーションに対し，それぞれのシステムを設計する．その流れとして，まず，ハードウェアとソフトウェアの最適な切り分けを探索するために，対象とするアプリケーション内の機能ブロックまたは連続ブロック毎に CPU の過負荷部を調査する．次に，様々な要求仕様に対する制約を想定し，CPU の過負荷部の調査を考慮した上で，それぞれに最適と思われる切り分けパターンを選出する．その後，選出したすべてのパターンに対して，ソフト・マクロ CPU と，VerilogHDL で設計した専用ハードウェアとで構成したシステムを設計し，FPGA 上に実装し評価，考察を行う．

2 章にて本研究で行ったハードウェアとソフトウェアの分割探索フローと実装環境を述べ，3 章では AES を対象としたシステム設計，4 章では JPEG エンコードを対象としたシステム設計について述べる．5 章ではまとめとして，実験結果を検証し，本研究の経験から分割手法を提案し，その有用性と改善点を挙げる．

2. ハード／ソフト分割探索

2.1　分割探索フロー

図 1 に本研究で行う分割探索のフローを示す．まず，アプリケーション全体を C 言語でプログラミングし，アプリケーションの動作検証を行う．次にシステムで用いるプロセッサ MicroBlaze にて，アプリケーションを実行する．ハードウェアクロックカウンタを用いて機能ブロックごとに実行に必要なクロックサイクル数を観測し，プロセッサの負荷が大きい箇所を見つける．その結果を参考に性能，規模，再利用性などの側面からの要求に応じた場合でのハードウェアとソフトウェアの切り分けを検討し，いくつかのパターンを選出する．そして，全てのパターンに対してハード/ソフト協調を考慮した設計を行う．ここで，ソフトウェアを実行するのは MicroBlaze なので設計者は専用ハードウェアを設計するだけでよい．そして，設計したシステムを FPGA 上にて実機検証を行い，計測結果から評価と考察を行う．

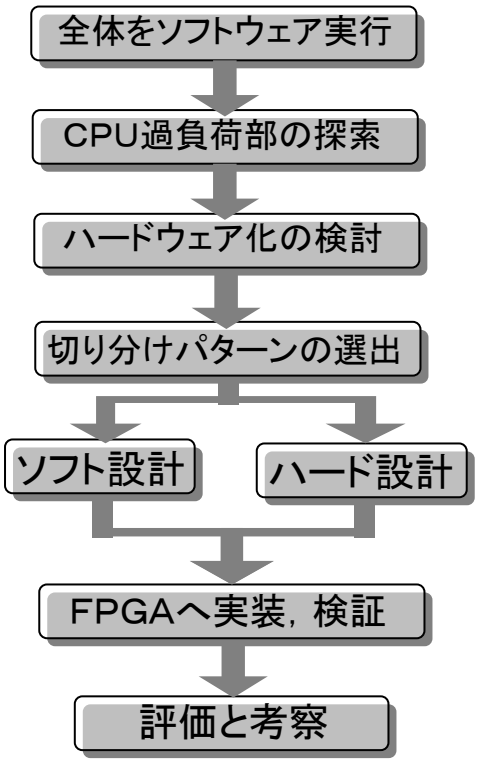


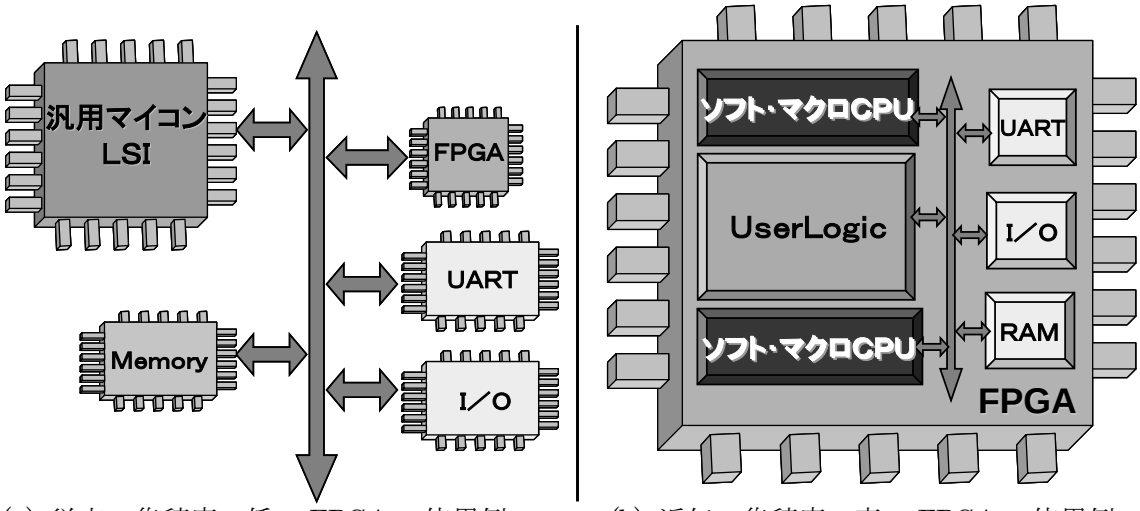
図 1　分割探索フロー

2.2　実装環境

近年，FPGA の高集積化と低価格化が進み，比較的手軽に大規模なシステムを FPGA に組み込むことも可能となっている．また，FPGA ベンダ各社は CPU の IP であるソフト・マクロ CPU を用意し，それを用いた開発環境も充実しつつある．ソフト・マクロ CPU と周辺 IP や，ユーザが用意した論理回路をバスで接続することで，ユーザが自由に設計したシステム LSI を 1 つの FPGA 内に実装することも今や可能となっている．図 2 にソフト・マクロ CPU のイメージを示す．集積度の低い従来の FPGA では，図 2(a)のように周辺モジュールの 1 つでしかなく，汎用マイコン LSI や I/O モジュールなどと基板上で配線されているケースが主であった．しかし，FPGA の集積度の向上により，図 2(b)のようにあらゆる機能を 1 つの FPGA へ搭載することが可能となっている．また，従来の ASIC 設計では，設計者は CPU コアに手を加えることはできず，利用しないハードウェアモジュールのために，他の用途で使用可能なシリコン面積を使用している場合もあったが，設計者が自由にカスタマイズ可能であるため，そのような無駄も少ない．

以下に，ソフト・マクロ CPU を用いる利点をまとめる．

- CPU 搭載 ASIC に比べて安価
- ユーザロジックを含めて論理合成可能であり，設計者が対象アプリケーション用に自由にカスタマイズ可能
- 汎用マイコンは設計者にとって必要のない機能も含まれているが，ソフト・マクロ CPU を用いると組み込んだ機能を全て動作可能
- 仕様変更に迅速に対応可能
- FPGA の生産中止があっても，容易にプロセッサシステム設計データの流用が可能



(a) 従来の集積度の低い FPGA の使用例　　(b) 近年の集積度の高い FPGA の使用例

図 2　ソフト・マクロ CPU

本研究では，以上のような利点を踏まえて，システムを設計し，FPGA に実装，評価を行う．今回，用いるソフト・マクロ CPU は Xilinx 社が提供する MicroBlaze である．MicroBlaze の主な仕様を表 1 に示す．

表 1　MicroBlaze コア仕様

命令セット	32bit RISC 型 3 オペランド
バスアーキテクチャ	ハーバード・アーキテクチャ(IBM CoreConnect) Local Memory Bus: 内蔵 BRAM 用高速アクセス On-chip Peripheral Bus: 周辺 IP 用
アドレス/データ幅	32bit
汎用レジスタ	32bit*32
アドレス空間	4GB
パイプライン	3 段(ハザードはハードウェアで回避)
コアサイズ	約 900LUT（450Slices）
パフォーマンス(現最大)	150MHz,　120DMIPS

図 3 に示すように，評価を行うための FPGA ボードとして MEMEC 社 DS-KIT-MB-S2E6LC-EURO を用いる．同ボードは Xilinx 社の 60 万システムゲートの Spartan-ⅡE(XC25600E-6FG456C)を搭載しており，メインボードには 8M×32 SDRAM や，各種スイッチが搭載されている．また，USB ポートや FLASH RAM，10/100Ethernet などを搭載した P160 Communications Module 拡張ボードも用意されており，これを接続するためのソケットも用意されている．

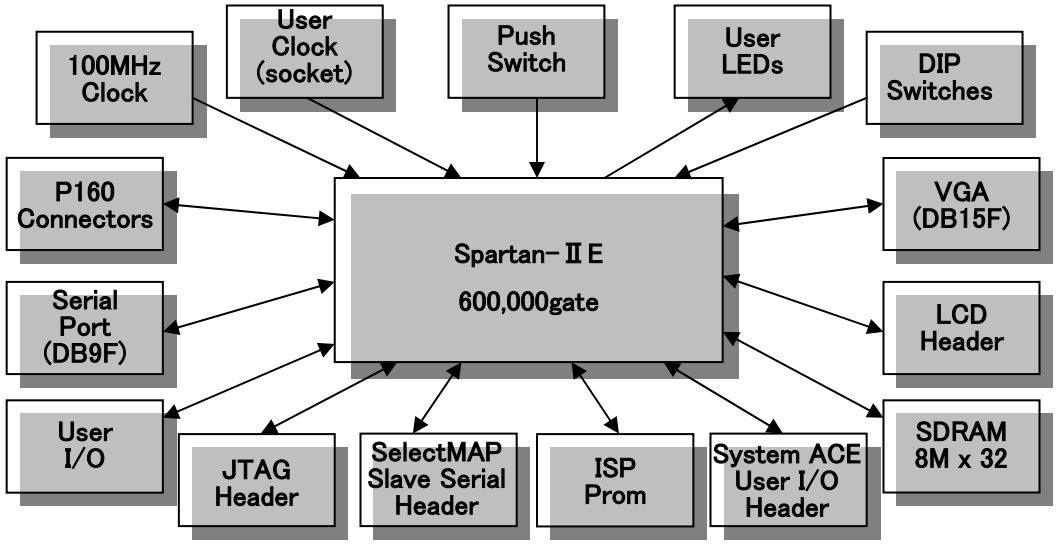


図 3　FPGA ボードの構成(メインボード)

2.3　MicroBlaze システムの開発フロー

Xilinx 社 MicroBlaze を用いたシステムを開発するためには，同社が提供する MicroBlaze システムの開発キット EDK(Embedded Development Kit)を用いる必要がある[21]．本研究ではこの EDK6.3 と，論理合成ツールとして Xilinx 社の ISE6.3，シミュレーションツールとして MentorGraphics 社の ModelsimSE5.8c を用いて設計を行った．図 4 に本研究で行った MicroBlaze システムの開発フローを示す．なお，図中の Xilinx Platform Studio(XPS)は EDK に含まれる統合開発ツールの名称である．

開発はまず，XPS にて，システムに必要な IP コアを選択し，それぞれのコアに対して様々なパラメータを設定する．次に，Plat Form Generator でシステム構成データを生成し，ISE で編集が可能なように，プロジェクトファイルをエクスポートする．そして ISE にて，あらかじめ用意した VerilogHDL で記述した回路と MicroBlaze システムの構成データ(HDL ラッパ)を，接続，統合し，制約ファイルと共に論理合成を行う．論理合成で生成したネットリストをシミュレーションし動作検証を行った後，マッピング，配置配線を行い，FPGA ビットストリームを生成する．この段階では，まだハードウェアの情報しか含まれていない．

一方，ソフトウェア側の設計はシステム上で実行する C プログラムを用意し，コンパイルを行い，オブジェクトファイルを生成する．IP コアを制御するドライバなどのライブラリプログラムは EDK が用意しており，その中から必要なライブラリファイルを Library Generator によって生成する. 次にライブラリとオブジェクトファイルから実行形式ファイルを生成する．これに，先程の FPGA ビットストリーム(HW)を結合し，最終的に FPGA にダウンロードする FPGA ビットストリーム(SW+HW)を生成する．

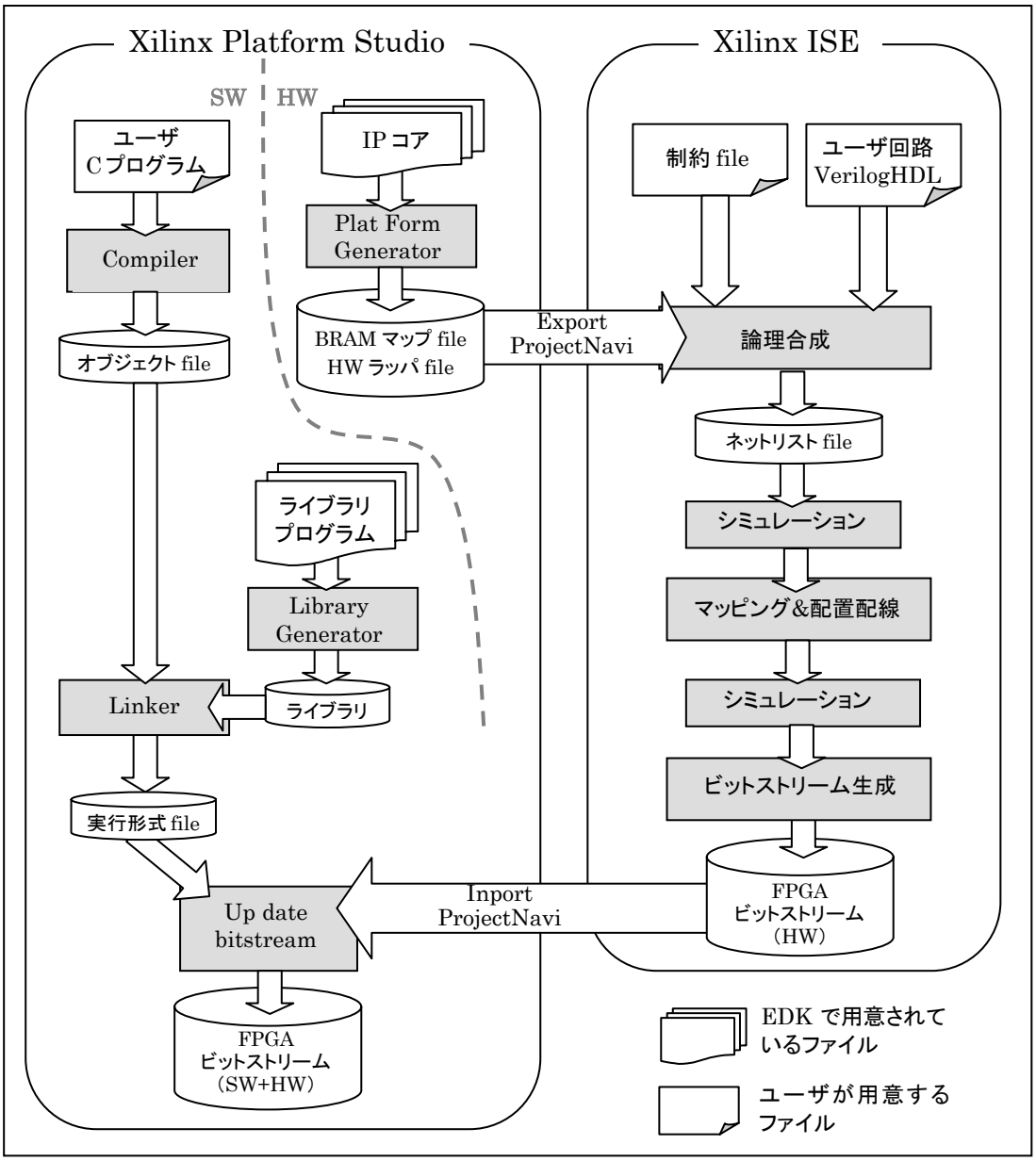


図 4　システムの開発フロー

3. AES 暗号処理システムのハード/ソフト分割

3.1　AES Rijndael アルゴリズム

AES(Advanced Encryption Standard)は，米国商務省標準技術局(NIST)によって 1997 年 9 月に公募，2001 年に標準認定された共通鍵方式の暗号化技術である．AES は，21 世紀の特に前半 20～30 年間に主役として利用される暗号処理であり，DES/TripleDES 暗号以上の十分な耐性があると言われている．AES の仕様は，FIPS197(Federal Information Processing Standards 197)[5]として出版されている．

AES のアルゴリズムである Rijndael アルゴリズムの暗号化フローを図 5 に示す．入力は暗号化対象である平文を 128bit ごとにブロック化した Data と，暗号鍵となる Key である．入力された Key は KeyExpansion 処理部において，必要なラウンド回数分のラウンド鍵に拡張される. ここで, ラウンドとは SubBytes, ShiftRows, MixColumns, AddRoundKey という一連の処理である．このラウンド処理を指定回数(Nr)分繰り返すことによって，入力 Data の暗号化が行われる．

なお，AES の暗号鍵は固定の鍵長ではなく，128，192，256 ビットの鍵長に対応しており, それぞれの鍵長(Nk)に対して, 表 2 のようにブロックサイズ(Nb)とラウンド回数(Nr)などのパラメータが決まっている．以降，AES-128 を対象とし，128bit の Data を 128bit の Key で暗号化を行うことを前提として説明する．

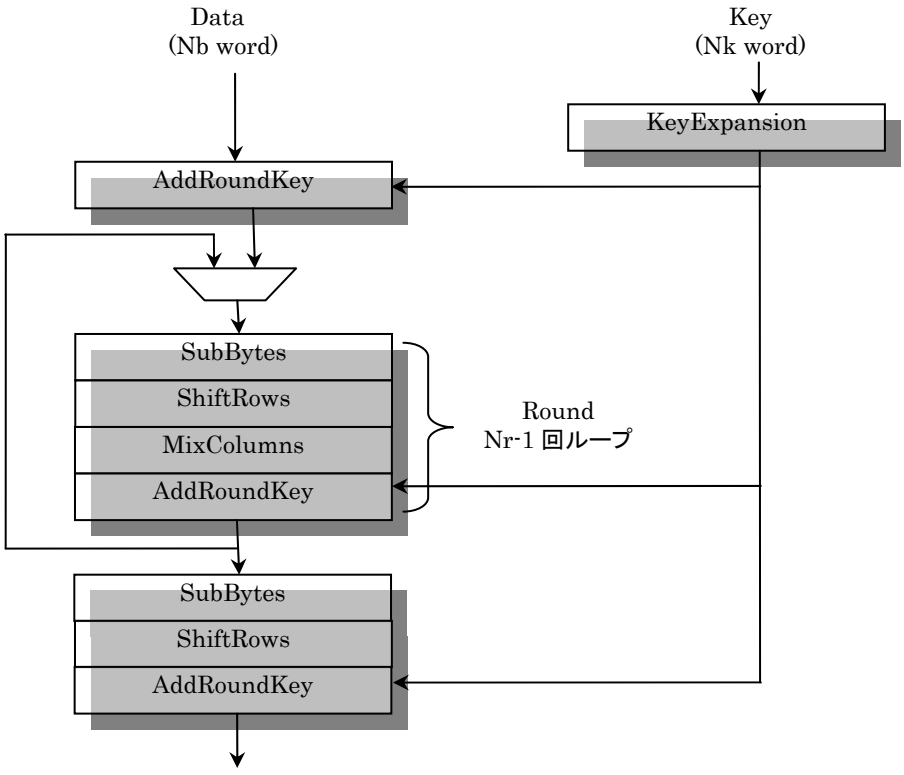


図 5　AES 暗号化フロー

表 2　パラメータの組み合わせ

	Key Length Nk words	Block Size Nb words	Number of Rounds Nr
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

3.1.1 KeyExpansion

与えられた暗合鍵 Key を図 6 のように 1Byte ごとに区切り，矢印の順に配置する．これを 0 番目のラウンド処理で使用するラウンド鍵 RoundKey 0 と呼ぶ．KeyExpansion ではこの RoundKey 0 を用いて，ラウンド処理で必要な計 Nr 個のラウンド鍵を生成する．これ以降，列方向のまとまり Exkey0,c　Exkey1,c　Exkey2,c　Exkey3,c を Wc として解説する．

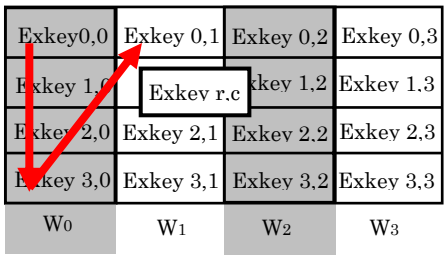


図 6　鍵の配置

次に，RoundKeyを生成するために，以下の処理を行う．cがNk(128bit鍵の場合 4)の倍数である時，つまり，$W_{(n*Nk)}$（nは自然数）を求める場合と，そうでない場合のWcを求める際では演算方法が異なるので注意が必要である．以下，図 7 に前者の場合を，図 8 に後者の場合の処理内容を示す．

(1)　c = n*Nk の場合

例：128bit 鍵(Nk=4)を拡張し，W4 を求める時

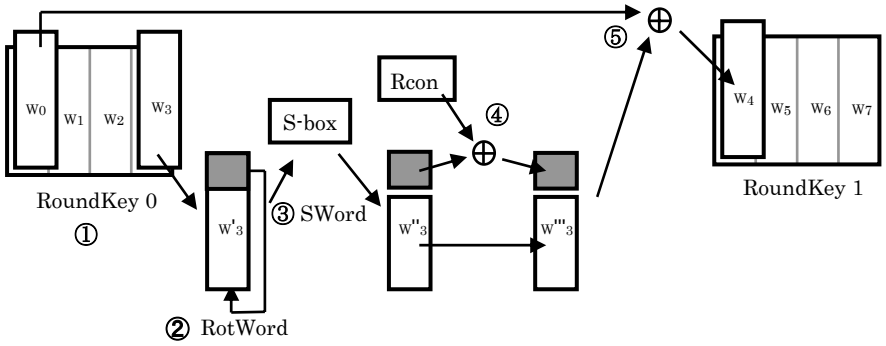


図 7　KeyExpansion 処理（c = n*Nk）

① W_4を求める際にはW_0とW_3が必要となる

② W_3をRotWordと呼ばれる列方向に 1Blockのシフト回転させる処理を行う

③ SWord と呼ばれる処理によって，Sbox テーブルの参照を行う

④ ラウンド定数テーブル Rcon を参照した値と，Exkey0,c の排他的論理和を行う

⑤ W_0と排他的論理和をとりW_4を求める

(2) c ≠n*Nk の場合

例：128bit鍵(Nk=4)を拡張し，W_5，W_6を求める時

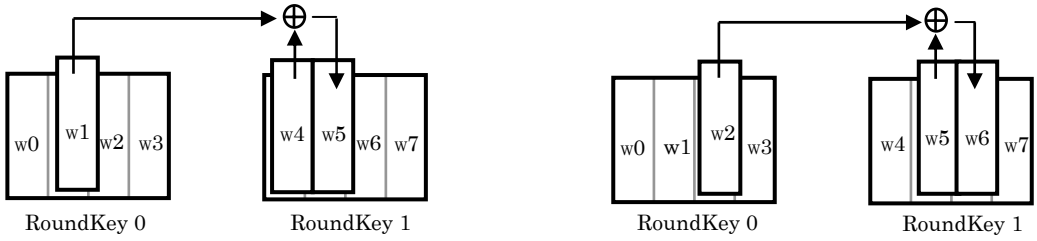


図 8　KeyExpansion 処理（c≠n*Nk）

同様に W7 は，W3 と W6 の XOR で求める．
つまり，一般式で書くと次のようになる．

$Wc = W_{C-Nk} + W_{C-1}$　　(c ≠n*Nk)

以上のようにして，RoundKey0 から各ラウンド処理に必要な計 Nr 個のラウンド鍵を拡張する．

3.1.2 AddRoundKey

与えられた平分を 1Byte ごとに区切り，S0,0 S1,0 S2,0 S3,0 S1,0・・・Sr,c の順に格子状に配置する．暗号処理はこの 16Byte 単位で行う．AddRoundKey では各ラウンド用に前述した KeyExpansion によって拡張された RoundKey と，入力データの排他的論理和をとる処理を行う．

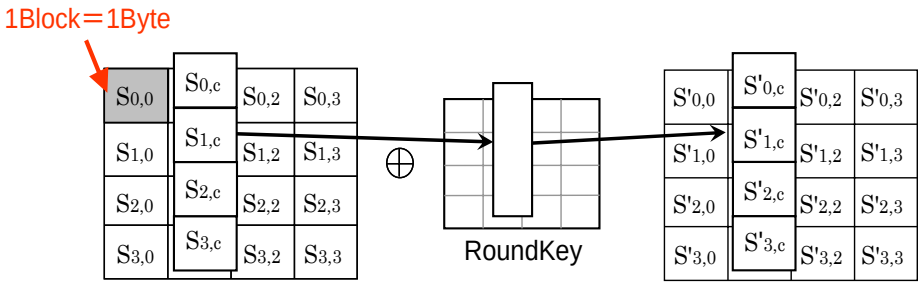


図 9　AddRoundKey

3.1.3 SubBytes

SubBytes変換は，8bitの値を 2^8のガロア体GF(2^8)の逆数に変換し，アフィン変換と呼ばれる行列変換を行って求める方法と，あらかじめ，その演算結果を非線形のByte参照テーブルS-Boxとして用意しておくという 2 つの方法が考えられる[9][13]．SubBytes変換の全体の様子を図 10 に示す．

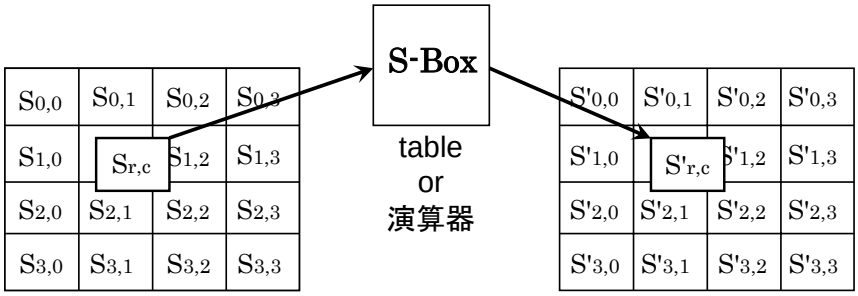


図 10　SubBytes

(1) 演算で求める場合

まず, 8bitの値を 2^8のガロア体GF(2^8)の逆数に変換する演算を行う. ある値yの逆数y^{-1}を求めるために，GF(2^8)のガロア体の要素yが持つ次の性質を利用する．

$$y^{2^8-1} = y^{255} = 1$$

という性質がある．よって，

$$y^{-1} = y^{-1} \bullet y^{255} = 1$$

より，y^{255}を求めれば逆数を求めることができる．

逆数を求めるために，ハードウェアにする際の再利用性を考慮し，乗算と 4 乗だけで算出する逆元演算アルゴリズムを図 11 に示す．

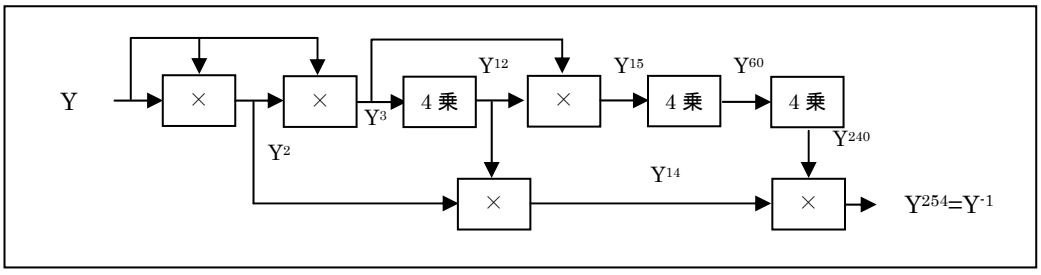


図 11　逆元計算アルゴリズム

次に，逆元計算によって得た逆数値に，アフィン変換と呼ばれる処理を行う．アフィン変換は次の行列式に通すことで行うことができる．アフィン変換の行列式を次に示す．

$$\begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

この行列変換式を経て，SubBytes 変換が完了する．

(2) S-Box テーブル参照で求める場合

S-Box テーブルの内容を表 3 に示す．S-Box テーブルへの参照は 1Block(8bit)の上位 4bit と下位 4bit に分け，それぞれを X，Y として参照を行う．そして，その参照結果が求める値となる．

表 3　S-Box テーブル

		Y															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
X	**0**	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ab	d5	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	A	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	B	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	C	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	D	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	E	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	F	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

3.1.4 ShiftRows

ShiftRows 変換では格子配列において，行方向のシフト処理を行う．図 12 のように 1 行目はシフトなし，2 行目は 1block 左シフト，3 行目は 2block 左シフト，4 行目は 3block 左シフトする．

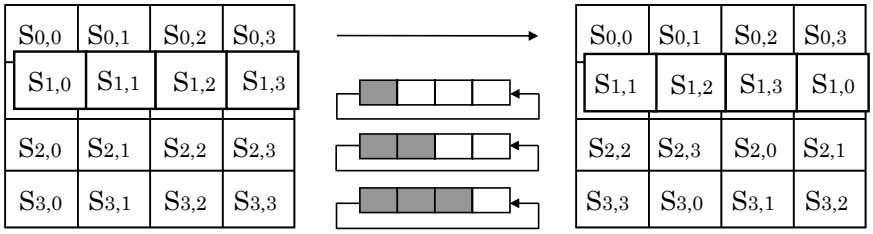


図 12　ShiftRows

3.1.5 MixColumns

MixColumns変換では図 13 のように格子配列において，列方向の変換を行う．変換結果はGF(2⁸)フィールドにおける以下の行列式の積として求めることができる[9].

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

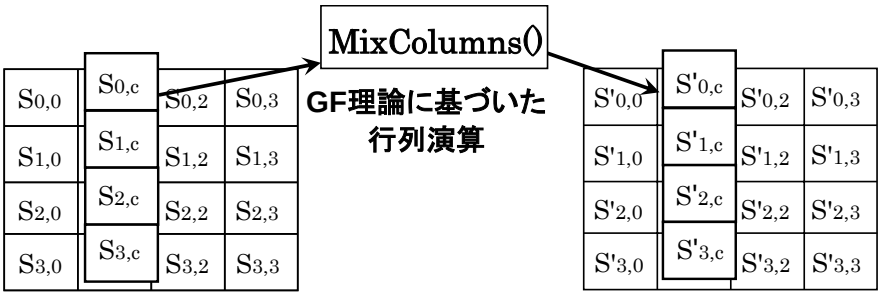


図 13　MixColumns

AES上での加算と乗算は, 数学的なフィールド理論に基づいている. 特に, MixColumnsはこのフィールド理論を用いた演算が必須である．GF(ガロアフィールド)理論の全体の理解はかなり困難であるが，AESで用いるGF(2⁸)フィールドの最終演算結果の算出は容易で，以下のルールに基づいて演算するとよい.

- **フィールド上での加算は XOR 演算**
- **フィールド上での乗算**

  MixColumns の行列式を見ると, AES の暗号化では被乗数と 0x01,0x02,0x03 との乗算を理解しておくだけでいいことがわかる. それぞれの乗数との場合分けをすると以下のようなルールになる.

◊ **0x01 と被乗数との乗算**

そのまま被乗数の値となる．(普通の算術と同じ)

◊ **0x02 と被乗数との乗算**

被乗数が 0x80 未満であれば，被乗数を 1 ビット左シフトした値が乗算結果
被乗数が 0x80 以上であれば, 被乗数を 1 ビット左シフトした値と 0x1b とを XOR 演算した値が乗算結果となる

◊ **0x03 と被乗数との乗算**

GF(2⁸)フィールド上では以下のように分配できるため, 0x01 と 0x02 との演算方法を利用して演算する

被乗数 * 0x03 = 被乗数 * (0x01 + 0x02)
= (被乗数 * 0x01) + (被乗数 * 0x02)

また, 参考までに, 複合化では被乗数と 0x09,0x0b,0x0d,0x0e との乗算が必要になるが, 0x03 の場合と同様に分配して演算できる．

3.2 CPU 過負荷部の調査

ハードウェアとソフトウェアの切り分けのパターン選出するための参考データとして, CPU の過負荷部の調査を行った．今回，プロセッサとして MicroBlaze を用いるため, MicroBlaze システムにクロックカウンタを設け，そのシステム上で AES アプリケーションを実行し，機能ブロック別に必要なクロック数を計測した．また，C プログラムをコンパイルする際の最適化レベルは「Level 2」とし, 計測に用いたデータは AES 仕様書に例として記載されている 16Byte の平文データと 16Byte の鍵である．全体の処理に対する各機能ブロックの CPU 負荷の割合をグラフにしたものを図 14 に示す.

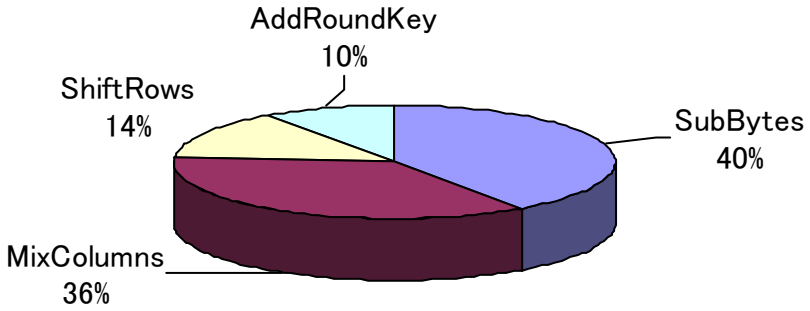


図 14 AES の CPU 負荷

図 14 より，SubBytes と MixCloumns の負荷が大きいことが分かる．SubBytes は演算でソフトウェア処理を行うと明らかに負荷が大きくなるため, S-Box のテーブル参照で実

現している．よって，各 16Byte のデータに対して 10 回実行され，1Byte ごとに逐次テーブル参照が行われるため，その際のアドレス生成に負荷がかかっていると考えられる．MixColumns は各 16Byte のデータに対して 9 回の実行であるが，分岐など，複雑な処理を行うため，そこが負荷となっていると考えられる．

なお，KeyExpansion に関しては，鍵拡張を行った結果を一度メモリに格納しておけば，次の 16Byte のデータでも再利用することができる．そのため，データ量が多い場合，全体の処理量に対する KeyExpansion の処理量の負荷は無視できると考え，ここでは除外している．

3.3 各処理部のハードウェア実現方法

AES 評価システムでは，MicroBlaze システムと各専用ハードウェアとの接続は GPIO で行う．今回，CPU として用いる MicroBlaze の場合，GPIO を用いると，一度に 32bit までのデータ転送が可能である．システムとの通信は PIO(Programmable IO)方式によるデータ転送となるため，大量のデータは扱いにくく，その場合は何度も通信する必要がでてくる．そこで，制御信号の数を少なくすることでシステムとの通信を最低限度に抑え，レスポンスを上げるために，各専用ハードウェアは組み合わせ回路で実現している．

3.3.1 KeyExpansion

与えられた Key を拡張し，AddRoundKey の各 Round で必要な RoundKey を生成する．ここで，生成する RoundKey の数は，鍵長に対して固定的に決まっている．そのため AddRoundKey における処理量は，暗号化対象のデータ量に左右されることなく一定であるといえる．よって，データ量が非常に多い場合，全体の処理量に対する KeyExpansion の処理量の割合は限りなく 0 に近づくことになる．そのためハードウェア化によるクロックサイクル数削減における恩恵は低いと考えられる．

設計したハードウェアの構成を図 15 に示し，KeyExpansion 処理部の入出力を表 4 に示す．

与えられた鍵(今回は 128bit)を基に必要なラウンド回数分×128bit の鍵に拡張する処理を行うため，最低 128bit のデータ転送が必要となる．しかし，GPIO で一度に転送可能なサイズは最大 32bit であるため，図 15 のように鍵データの入出力用に 32bit×4 の入出力ピンを用意し，鍵(128bit)を 4 つに分割することでハードウェアとシステムはデータの授受を行っている．図中の入力 round は回路内部の Rcon テーブルからラウンド定数を決定するためのラウンド引数である．S-box は後に説明する SubBytes 処理と同じである．

N 回目のラウンドで必要なラウンド鍵を生成するためには，N-1 回目で使用したラウンド鍵が必要となる．そこで，システム側では，N-1 回目にハードウェアから出力されたラウンド鍵を N 回目のハードウェアへの入力となるようにプログラムしている．このような処理を必要なラウンド回数分繰り返すことで，鍵拡張を行っている．

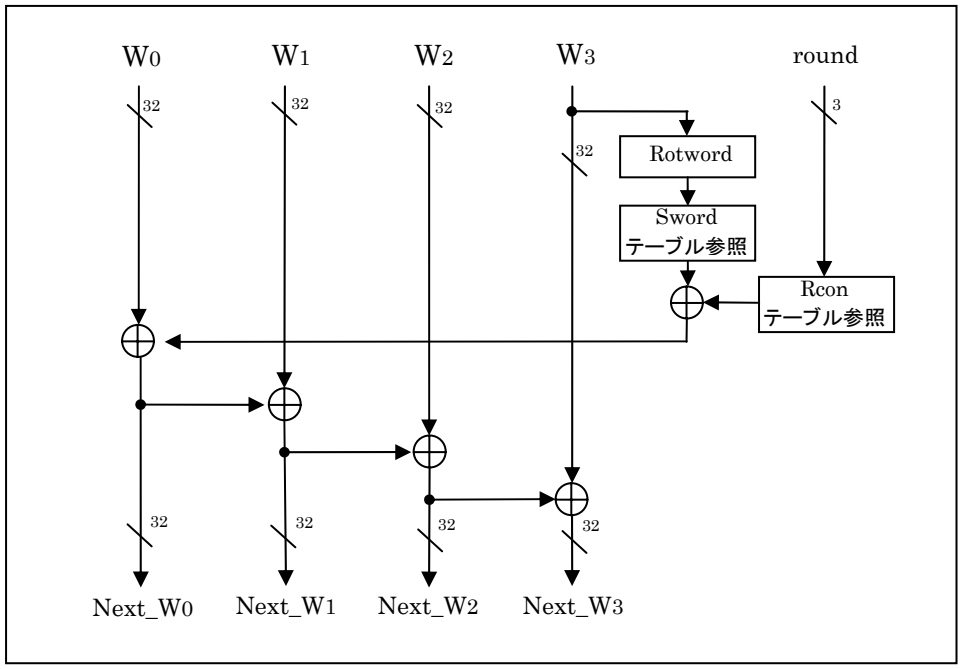


図 15　KeyExpansion 回路

表 4　KeyExpansion の入出力

名前	方向	ビット幅	用途
W0	in	[0:31]	鍵 0～31bit の入力
W1	in	[0:31]	鍵 32～63bit の入力
W2	in	[0:31]	鍵 64～95bit の入力
W3	in	[0:31]	鍵 96～127bit の入力
round	in	[0:2]	Rcon テーブル引数
Next_W0	out	[0:31]	拡張鍵 0～31bit の出力
Next_W1	out	[0:31]	拡張鍵 32～63bit の出力
Next_W2	out	[0:31]	拡張鍵 64～95bit の出力
Next_W3	out	[0:31]	拡張鍵 96～127bit の出力

3.3.2 AddRoundKey

RoundKey とデータの XOR をとる処理である．単純な処理であるため，ソフトウェアでも軽い負荷で実行可能である．ハードウェア実装による速度向上を目指すために，組み合わせ回路で一度にまとめて XOR をとる．

図 16 に AddRoundKey の回路を示し，表 5 にその入出力を示す．

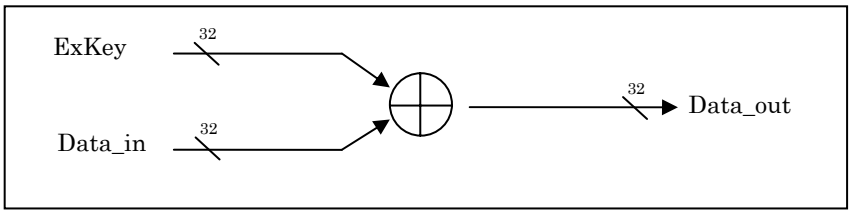


図 16　AddRoundKey 回路

表 5　AddRoundKey の入出力

名前	方向	ビット幅	用途
ExKey	in	[0:31]	拡張鍵の入力
Data_in	in	[0:31]	データ入力
Data_out	out	[0:31]	データ出力

3.3.3 SubBytes

演算による実装方法と，S-box と呼ばれるテーブルの参照による実装方法が考えられる．前者は，ソフトウェアよりもむしろ，演算回路のハードウェア実装により実現することが考えられる．後者はメモリ上に配置されたテーブルを参照することで実現できるため，比較的高速に値を得ることが可能であるが，テーブル分のメモリまたはゲートを消費することになる．

図 17 に ROM によるテーブルの実装を，図 18 に演算回路による実装の様子を示し，表 6 に入出力を示す．

共に 8bit 入出力であるが，実際はそれぞれを 4 つずつ使用して 32bit にしている．演算器で実装した場合，INV 信号を切り替えることで，1 つのハードウェアで暗号処理と複合処理を行えるというメリットがある．だが，テーブルで実装した場合は暗号用と複合用にそれぞれテーブルを用意する必要がでてくる．また，回路規模を比較すると，演算回路は 9441gate，テーブルは 4800gate となることがわかった．

以上のことより，本研究では暗号処理のみを扱うということもあるため，テーブルによる実装を行った．

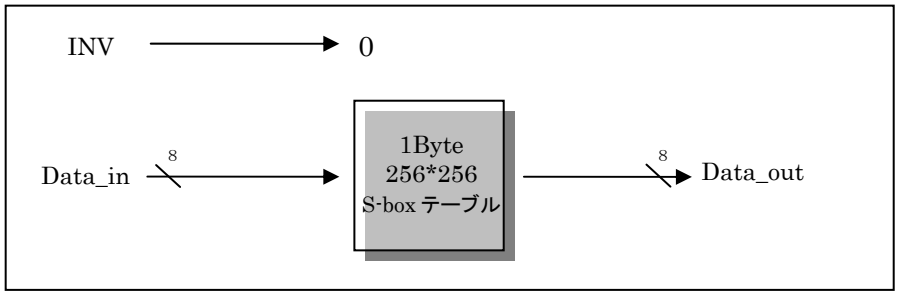


図 17　テーブル参照による S-box 実装

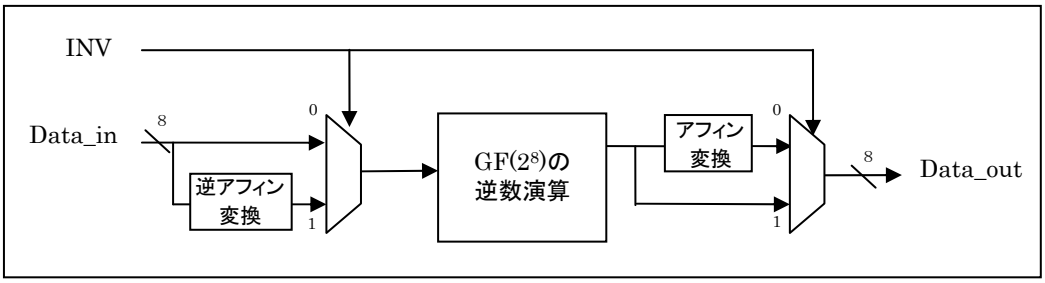


図 18　S-box 演算回路

表 6　SubBytes の入出力

名前	方向	ビット幅	用途
Data_in	in	[0:31]	データ入力
INV	in	[0:0]	暗号化：0　複合化：1
Data_out	ouy	[0:31]	データ出力

3.3.4 ShiftRows

単純な 1 ブロック(1Byte)の回転処理である．ソフトウェアで実装する場合は，対象データが格納されている配列の添え字をずらす形で実装できるため，比較的軽い負荷で実行可能である．ハードウェアによる速度向上を目指すためには，線の繋ぎ変えによって一度に回転処理を行う方法が考えられる．

表 7　ShiftRows の入出力

名前	方向	ビット幅	用途
Data0_in	in	[0:31]	データ 0～31bit の入力
Data1_in	in	[0:31]	データ 32～63bit の入力
Data2_in	in	[0:31]	データ 64～95bit の入力
Data3_in	in	[0:31]	データ 96～127bit の入力
Data0_out	out	[0:31]	データ 0～31bit の出力
Data1_out	out	[0:31]	データ 32～63bit の出力
Data2_out	out	[0:31]	データ 64～95bit の出力
Data3_out	out	[0:31]	データ 96～127bit の出力

3.3.5 MixColumns

行列式と暗号対象データの XOR とシフト演算による演算処理である．事前のソフトウェア過負荷探索によって，負荷が大きい機能ブロックであることが分かっている．ソフトウェアで実装する場合は，演算前に被乗数によって場合分けがあるため，複雑な処理になるが，組み合わせ回路，または順序回路によるハードウェア化によって，大きな速度向上が計れると考えられる．

図 19 に実装した回路図を示し，表 8 に入出力を示す.

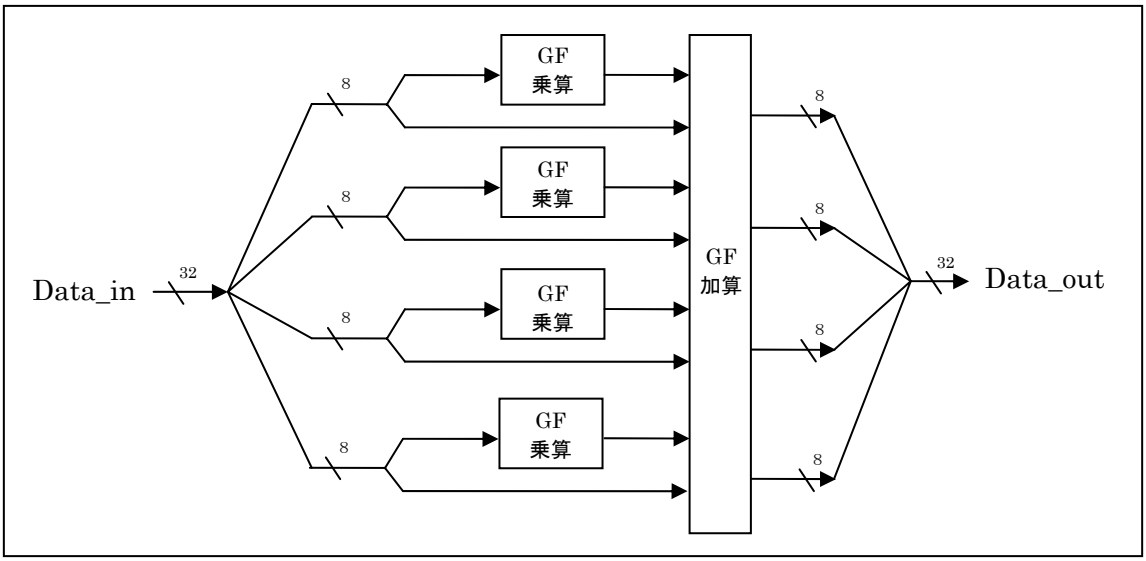


図 19　MixColumns 回路

表 8　MixColumns の入出力

名前	方向	ビット幅	用途
Data_in	in	[0:31]	データ入力
Data_out	out	[0:31]	データ出力

3.1.5 にて述べたように，GF(2⁸)フィールドにおいて，加算はXORで行われ，乗算は分解してシフトで行うことができる．GF乗算器では，被乗数の値の判別によるシフト演算が行われ，その結果を用いてGF加算器にてXORを行い，32bitの結果をまとめて出力する．

演算は 8*8 の格子配列の列方向に対して行う必要があるため，処理対象となる 4 Block(計 4Byte)を列方向に並べてから，MixColumns に接続された GPIO に一気に書き込むようにプログラムしている．そして，結果を GPIO から読み出した後は，再び元のメモリ配置になるように並び替えることを行っている．

3.4　ハードとソフトの切り分け

AES 評価システムではハードとソフトに切り分ける粒度に関して大きく次の 2 つの切り分けを考慮する．機能ブロック単位で切り分ける方法は，CPU とハードウェアとのデータ送受信回数が増加するが，機能ごとに回路として別れているため，IP コアとして再利用が可能であるというメリットがある．連続ブロック単位で切り分ける方法は，図 5 の暗号化フローにおける連続して処理を行う部分を 1 つの回路として切り分け，実装する方法である．IP コアとしての再利用性には欠けるが，回路内で演算の最適化や，また，その連続

ブロック間では CPU によるデータ転送が不要になるなどのメリットがある．

3.4.1 機能ブロック単位での切り分けパターン

3.2 で調査した CPU 過負荷部を考慮した上で, ハードウェアで処理する部分とソフトウェアで処理する部分の切り分けパターンを表 9 のように選出した．なお便宜上，表のパターンの順は評価結果から得たデータを基に，表の上方から回路規模の順で並べてある．

表 9　機能ブロック単位での切り分けパターン

	ハードウェア処理部	ソフトウェア処理部	備考
S		KeyExpansion AddRoundKey SubBytes ShiftRows MixColumns 制御	ソフトウェアで全処理 評価基準となる構成
A	MixColumns	KeyExpansion AddRoundKey SubBytes ShiftRows 制御	過負荷部 MixColumns のみをハードウェア化する回路規模を考慮した構成
B	SubBytes	KeyExpansion AddRoundKey MixColumns ShiftRows 制御	過負荷部 SubBytes のみをハードウェア化する回路規模を考慮した構成
C	SubBytes MixColumns	KeyExpansion AddRoundKey ShiftRows 制御	パターン A と B を組み合わせた構成
D	SubBytes ShiftRows MixColumns	KeyExpansion AddRoundKey 制御	CPU の負荷を抑える構成
E	AddRoundKey SubBytes MixColumns	KeyExpansion ShiftRows 制御	回路規模をあまり考慮せず, CPU の負荷を抑える構成
F	AddRoundKey SubBytes ShiftRows MixColumns	KeyExpansion 制御	回路規模を考慮せず，データ量に比例して CPU の負荷が増減する処理部をハードウェア化する構成
G	KeyExpansion AddRoundKey SubBytes ShiftRows MixColumns	制御	回路規模を考慮せず，複雑な処理である制御のみをソフトウェアで実現する構成

3.4.2 連続ブロック単位での切り分けパターン

連続ブロック単位での切り分けパターンを表 10 に挙げる．こちらも同様に，評価データの結果から回路規模の順で並べている．表中のハードウェア処理部の「－」はそれらが連続ブロックであることを示し，ソフトウェア処理部において網掛けとなっている処理は，そのパターンのハードウェア処理部の内，アルゴリズム上，連続ブロックになっていない処理である．また，その括弧内の数字は処理する回数を示す．

表 10　連続ブロック単位での切り分けパターン

	ハードウェア処理部	ソフトウェア処理部	備考
(イ)	SubBytes－ShiftRows	KeyExpansion，AddRoundKey，MixColumns，制御	全処理中の SubBytes と ShiftRows を完全にハードウェア化できる構成
(ロ)	SubBytes－ShiftRows－MixColumns	KeyExpansion，SubBytes(1)，ShiftRows(1)，AddRoundKey，制御	(イ)に MixColumns を含めてハードウェアする構成
(ハ)	SubBytes － ShiftRows － MixColumns － AddRoundKey	SubBytes(1)，ShiftRows(1)，AddRoundKey(2)，KeyExpansion，制御	１ラウンド処理をハードウェア化する構成
(ニ)	SubBytes－ShiftRows，AddRoundKey－KeyExpansion	MixColumns，制御	SubBytes と KeyExpansion のハードウェア化で S-Box による使用メモリを削減
(ホ)	SubBytes－ShiftRows－MixColumns，AddRoundKey－KeyExpansion	SubBytes(1)，ShiftRows(1)，制御	ラウンド処理をハードウェア化することを意識した構成
(ヘ)	SubBytes － ShiftRows － MixColumns － AddRoundKey－KeyExpansion－制御		全てをハードウェア化する評価の参考とする構成

3.5　AES 評価システムの実装

3.5.1 AES 評価システムの基本構成

前節で述べた全てのパターンの AES 評価システムを設計し，MEMEC 社の FPGA ボード DS-KIT-MB-S2E6LC-EURO に実装した. 図 20 に AES 評価システムの基本構成を示す.

メインメモリとして，MicroBlaze プロセッサコアには高速の LMB（Local Memory Bus）を介して BlockRAM を接続し，これを命令とデータを格納している．切り分けパターンにおけるソフトウェア処理に当たる部分は，MicroBlaze プロセッサコアが行う．一方，ハードウェア処理に当たる部分は，それぞれのモジュールを VerilogHDL で設計し，GPIO(General Purpose Input/Output)を通じで OPB(On-Chip Peripheral Bus)に接続している．この構成をベースとして，AES 専用ハードウェアの着脱や，ソフトウェアの改変などの変更を行うことで，前節で挙げた切り分けパターンを実装し評価した.

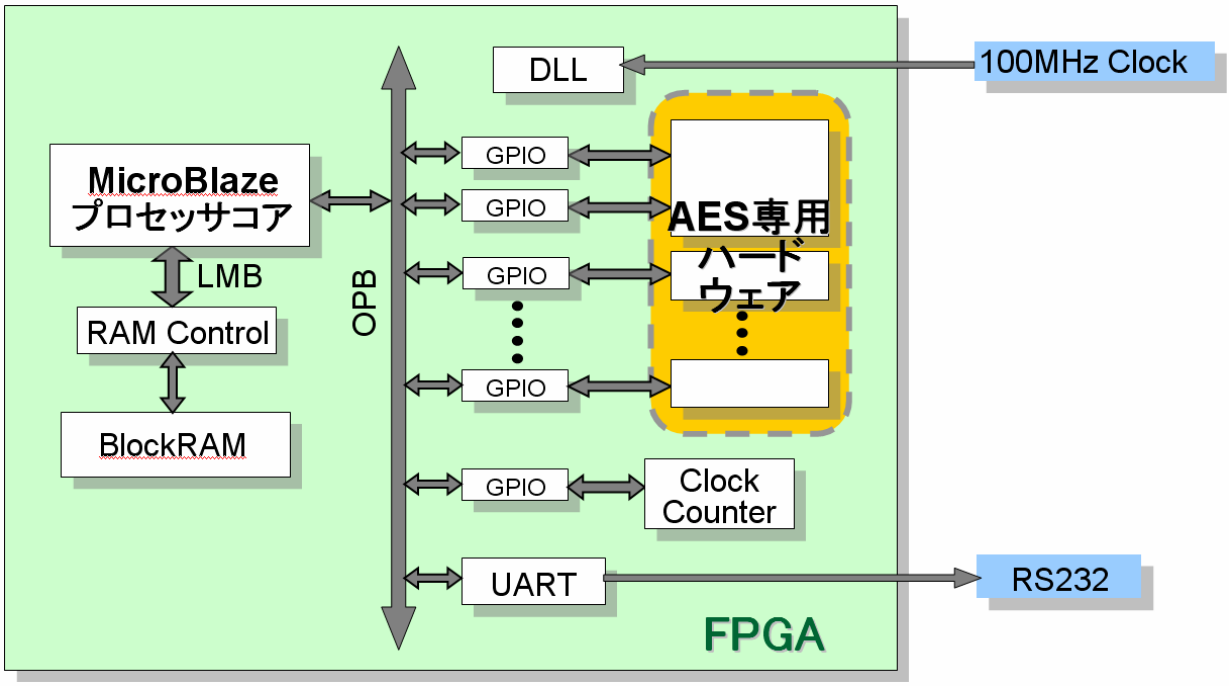


図 20　AES 評価システム基本構成

3.5.2 AES 評価システムの実行フロー

データと命令は，すべて LMB で接続した BlockRAM に格納されている．よって，ソフトウェアに割り当てられた処理を実行する際は，MicroBlaze が BlockRAM にアクセスして実行する．また，ハードウェアに割り当てられた処理を実行する際は，対象の専用ハードウェアに接続された GPIO にデータを書き込むことで実行する．今回，AES 専用ハードウェアは全て組み合わせ回路で実現してあるため，すぐ次の命令で GPIO のデータを読み込むことで，処理結果を得ることができる.

3.5.3 周辺機器

(1) GIPO（General Purpose Input/Output）

GPIO は入出力用のバスインタフェースであり，メモリ空間にマッピングされている．MicroBlaze からは，マッピングされたアドレスを指定することで，一度の命令に最大 32bit のデータ転送(読み書き)を行うことができる．つまり，PIO(Programmable IO)方式によるデータ転送であるため，大量のデータを扱う場合には不向きであるといえるが，AES はデータの内容に性能が左右されないアルゴリズムであるため，今回の評価を行う場合は十分であると考え，インタフェースとして GPIO を採用した．図 21 に GPIO による接続の様子を示す．図に示すように GPIO は，必要に応じて単方向，双方向，複数接続することが可能である．データの流れは非常に単純で，以下の通りである．

① データの送信命令が実行されるとデータは WrDBUS 経由で GPIO 内の書き込みレジスタに書き込まれる
② AES 専用ハードウェアは GPIO 内の書き込みレジスタに繋がっている GPIO_d_out のデータを読み込み，そのデータを用いて処理を行う
③ 処理結果を GPIO_in 経由で GPIO 内の読み込みレジスタに送信する
④ データの受信命令が実行されると，DBUS 経由で GPIO 内の読み込みレジスタの値を読み込む

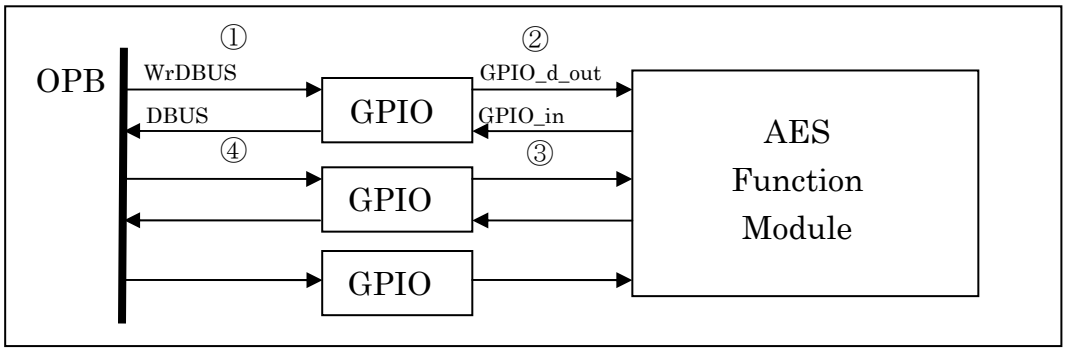


図 21　GPIO を用いた接続

(2) Clock Counter

必要クロックサイクル数の計測用にクロックカウンタを設けている．C プログラム上で，計測対象の処理開始時点で，MicroBlaze から開始信号を送信し，カウントを開始させる．以降，計測対象の処理終了時点でクロックカウンタを接続している GPIO の値を読み取ることで，それまでのクロックサイクル数を得ることができる．

(3) UART（Universal Asynchronous Receiver Transmitter）

UART は RS232 を介してシリアルケーブルと接続されており，そのシリアルケーブルは PC 端末と接続している．ソフトウェアにて標準出力(print 文)を記述した場合，シリア

ルケーブルを通じて，PC 画面上に出力されるように設定した．これを用いることで，デバッグと動作確認を行った．

(4) DLL（Delay Lock Loop）

DLL は分周器である．MicroBlaze は，FPGA ボード DS-KIT-MB-S2E6LC-EURO に搭載されている FPGA Spartan-ⅡE(XC25600E-6FG456C)のテクノロジにおいて，オンボードの発振子 100MHz での動作は保証されていないため，50MHz に分周してからクロックを供給している．

3.5.4 ハードウェアとのデータ送受信プログラム

GPIO とのデータ送受信は，MicroBlaze システム開発環境ツール EDK が用意している GPIO ドライバを用いることで行うことができる．今回は gpio_v1_00_a の low レベルドライバ xgpio_l.h に定義されている XGPIO_mWriteReg();と XGPIO_mReadReg();を主に用いてプログラミングを行った．

図 22 に，SubBytes 回路にデータを転送し，SubBytes をハードウェアで処理するプログラムを示す．ハードウェアとデータ送受信を行うためにはまず，用意した関数に移り，データを送信する場合は，関数 XGPIO_mWriteReg();の第一引数に書き込み先の GPIO のアドレスを指定し，第三引数に書き込むデータを指定する．受信する際には第一引数に読み込み先の GPIO のアドレスを指定することで，戻り値としてデータを読み出すことができる．

```
・・・略・・・・
 for(r = 1; r < Nr; r++){

   HW_SubBytes(state);

   ShiftRows(state);

   MixColumns(state);

   AddRoundKey(state, exkey, r);

}
  /*----------- Hard Ware SubBytes transformation -----------*/
  void HW_SubBytes(u8 state[4][4]){
    int r;
      for(r = 0; r < 4; ++r) {
        XGpio_mWriteReg(XPAR_SUBBYTES_32BIT_BASEADDR, GPIO1_DATA, ((int *)state)[r]);
        ((int *)state)[r]=XGpio_mReadReg(XPAR_SUBBYTES_32BIT_BASEADDR, GPIO1_DATA);
      }

  }  // Hard Ware SubBytes()
```

①SubBytes 回路とデータ送受信する関数に移る

②関数 XGPIO_mWriteReg():を用いて，GPIO にデータを書き込むことで，SubBytes 回路にデータを渡す

③関数 XGPIO_mReadReg():を用いて，GPIO のデータを読み込むことで，SubBytes 回路から結果データを受け取る

図 22 データ送受信プログラム

3.6　実行結果

3.6.1 切り分け評価結果

表 11図 23に表 9で選出した機能ブロックによる切り分けの結果を示し, 表 12, 図 24に表 10 で選出した連続ブロックによる切り分け結果を示す.

なお，クロックサイクル数は 128bit のデータを暗号化するのに必要なクロック数をクロックカウンタで計測した値である．使用メモリは，MicroBlaze がプログラム実行に必要なメモリ量であり，XPS メニューの Tool > Get Program Size のレポートから算出した．回路規模は，ISE の Map > Map Report の Total equivalent gate count for design の値である．ただし，表に示した値はハードウェア化した AES 専用ハードウェアの規模の総和であり，MicroBlaze とその周辺機器の回路規模は含んでいない.

表 11　機能ブロックによる切り分け結果

	S	A	B	C	D	E	F	G
クロックサイクル数	38075	26959	25859	14743	11313	13979	10549	6650
使用メモリ(Byte)	4387	3700	4024	3572	3496	3812	3736	3108
回路規模(gate)	0	324	4800	5124	5124	5316	5316	10932

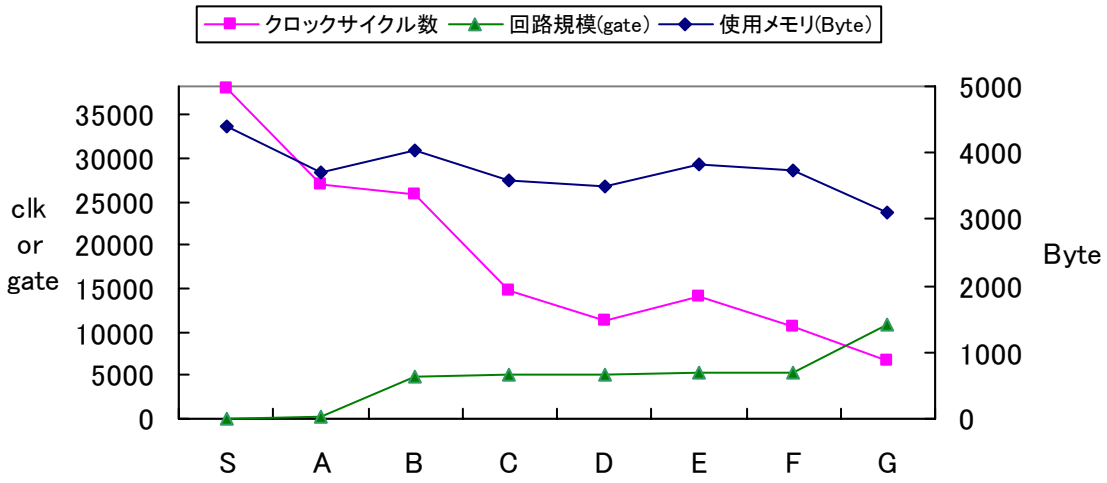


図 23　機能ブロックによる切り分け結果

表 12　連続ブロックによる切り分け結果

	イ	ロ	ハ	ニ	ホ	ヘ
クロック サイクル数	21996	13273	14875	16178	7455	10
使用メモリ (Byte)	3832	3724	4080	3236	3472	0
回路規模 (gate)	5280	7360	7552	15006	17086	38342

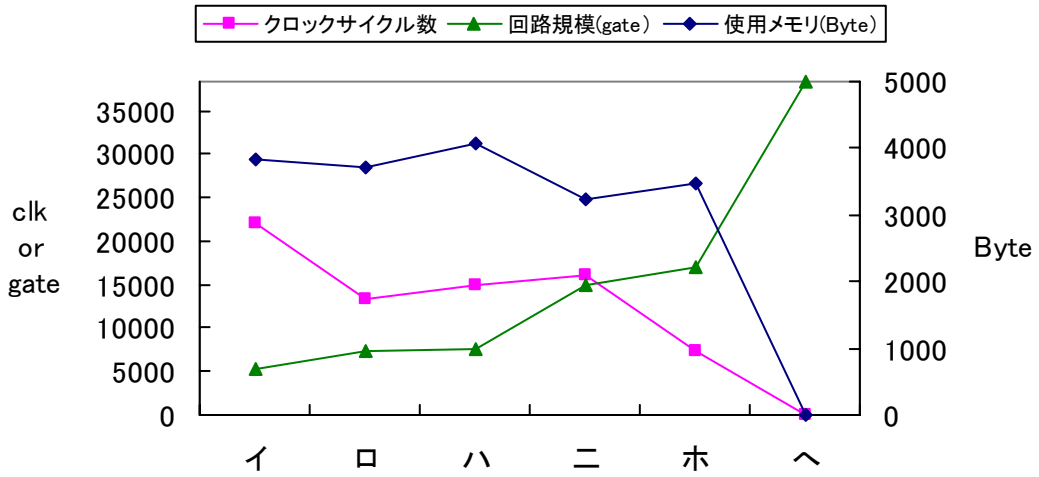


図 24　連続ブロックによる切り分け結果

3.6.2 考察

機能ブロック単位での切り分けは，図 23 の S→A 及び B→C の変化に注目すると，MixColumns のハードウェア化によるクロックサイクル数の短縮への貢献が大きく，なおかつ回路規模の増加量もほとんどみられないことがわかる．また，S→B の変化に注目すると，SubBytes のハードウェア化はクロックサイクル数の短縮は図れるが，回路規模が大きく増大してしまうことが分かる．また，C→D に注目すると，ハードウェア量を消費することなく，クロックサイクル数と使用メモリ量を削減できていることがわかる．また，E,F に関してはあまりハードウェア化による恩恵はほとんどないと言える．よって，ハードウェア化による，貢献度を考慮すると D が最適であるといえる．

一方，メモリ使用量に注目すると，S→A 及び F→G の時，わずかながら削減率が大きい．前者は MixColumns の比較的演算が複雑であり，これのハードウェア化によってヒープ領域が不要になる為であり，後者は S-Box テーブルが必要な KeyExpansion と SubBytes の 2 つをハードウェア化することで，メモリ上の S-Box テーブルが不要になる為であると考えられる．

次に連続ブロック単位での切り分けは，図 24 の(ロ)に注目すると，回路規模を増加さ

せることなくクロックサイクル数を短縮できていることが分かる．ここでもやはり，MixColumns のハードウェア化の有効性が現れている．また，(ハ),(ホ)にように連続処理ブロックでまとめたとしても，表 10 の網掛け部のようにハードウェアにまとめきれない処理ブロックが多数ある場合，逆効果となり，メモリ使用量削減，回路規模の縮小が見込めない場合もあることがわかる．

一方，メモリ使用量に注目すると，(ニ)が回路規模は大きいものの，メモリ使用量を削減できていることが分かる．これは，連続ブロック単位で切り分けを行いながらも，そのブロックを完全にハードウェアにまとめきれているためである．

次に，機能ブロック単位での切り分けと連続ブロック単位での切り分けで比較する．前者は機能ごとにハードウェアが分割されているため，再利用性が高く，柔軟性も高いといえる．しかし，ハードウェア間のやり取りが増すため，制御を行うソフトウェアの負荷が高くなると予想していた．また，後者は連続する複数の処理のハードウェアを 1 つのハードウェアに統合するため，演算の最適化が可能であり，回路規模の削減及び性能向上が図れる一方，自由度，再利用性は低くなり，保守は困難になると考えられた．しかし，今回グラフを比較すると，そのグラフに大きな差異はなく，先に述べた連続ブロック単位での切り分けの大きな利点があまり見られないことがわかる．

総じて，以上より，ハードウェア化によるクロックサイクル数削減の貢献度を考えた場合，図 23 図 24 より，D と(ロ)を候補に挙げることができる．そして，これに再利用性や柔軟性を考慮した場合，D が最も適しているといえる．また，加えて回路規模に余裕がある場合は G と(ホ)が候補に挙がるが，同様に再利用性や柔軟性を考慮に入れると，G が適しているといえる．

4. JPEG エンコーダシステムのハード/ソフト分割

4.1　JPEG アルゴリズム

JPEG は現在, 最も一般的な静止画像データ圧縮方式の 1 つである[14][15]. 名称は ISO により設置された規格策定団体の名前がそのまま使われている．圧縮の際には，画質劣化を許容する方式と，まったく劣化しない方式を選択することができるが，本研究では前者の中でも，一般的なベースライン JPEG と呼ばれる方式を取り扱う．

JPEG エンコードの処理フローを図 25 に示す．まず，原画像データを 16*16 画素の MCU（Minimum Coded Unit）に分割し，以降は MCU 単位でエンコード処理を行う．

色空間変換では MCU に対し 3 原色を値で表す RGB モデルから，輝度と色相を値で表す YUV モデルへの色空間の変換を行う．次に，1 つの MCU に対し，サブサンプリングによって，色差成分の間引きを行い，各 8*8 画素の Y 成分 4 つ，U 成分，V 成分の計 6 つのブロックを生成する．以下，このブロックに対して，離散コサイン変換(DCT)，量子化，ハフマン符号化の流れで処理を行う[19][20].

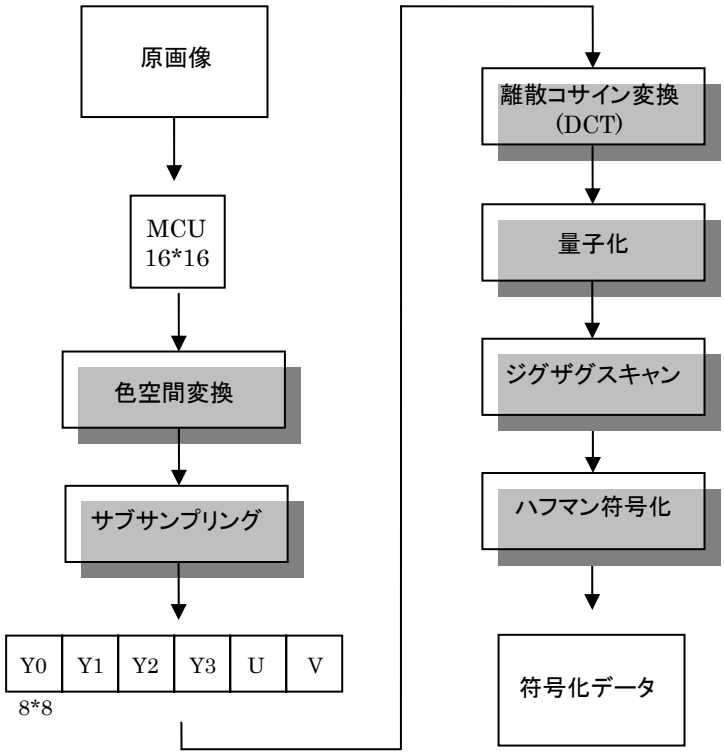


図 25　JPEG エンコード処理フロー

4.1.1 色空間変換

3原色の RGB モデルで表現された MCU に対し，以下の計算を行うことで，輝度と色相を表す YUV モデルへ変換する．

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

このような RGB から YUV への色空間変換によって，色差成分 U,V よりも輝度成分 Y に多くの情報量を偏らせることができる．また，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を利用し，これ以降の処理において，色差成分の精度を落とすことで見た目の画像劣化を抑えつつデータ量を減らすことができる．

4.1.2サブサンプリング

次にサブサンプリングを行う．今回は一般的である Y：U：V＝4：1：1 でのサブサンプリングを行う．図 26 に 4:1:1 のサブサンプリングを行う様子を示す．Y1～Y4 は MCU を図のように 4 等分した時の輝度成分 Y をそれぞれ 1 つにまとめたものであり，U と V は共に MCU 全体から，1 つおきに間引きしたものとなっている．よって，1 つの 16*16 画素の MCU から 6 つの 8*8 画素のブロックが生成される．

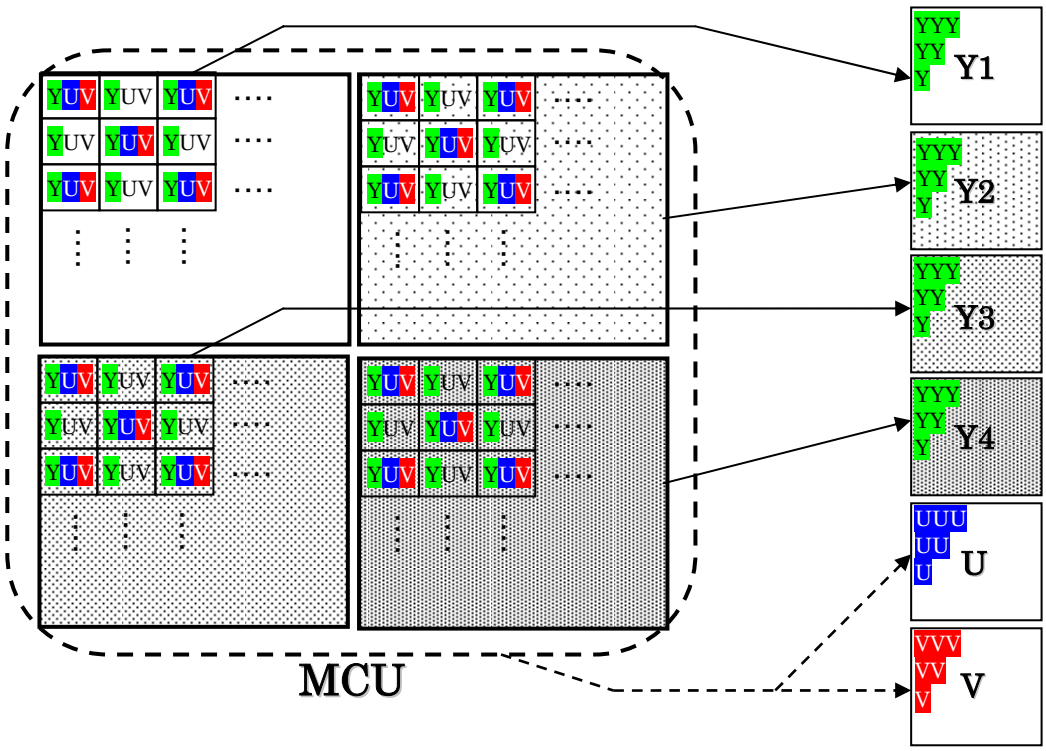


図 26　4:1:1 サブサンプリング

4.1.3 離散コサイン変換（DCT）

サブサンプリングで生成した 6 つの 8*8 のブロックに対し，2 次元の離散コサイン変換(DCT)と呼ばれる変換を行う．この処理によって，精度を落としても比較的目立たない画像の高周波成分を劣化させ，1 ブロックの画像を 64 段階の周波数成分で表した DCT 係数に変換する．具体的に 2 次元の離散コサイン変換は，ブロック内の横方向位置 x，縦方向位置 y の画素値 $f(x,y)$ に対し，下記の演算式を適用して，周波数軸上での横方向位置 u，縦方向位置 v の DCT 係数 $F(u,v)$ を求める[18].

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

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

4.1.4 量子化

量子化は，DCT 係数を量子化テーブルの値で除算し，四捨五入して整数化することで，一定 bit 数のデジタルデータに近似する処理を行う．量子化テーブルは，情報量を減らすと劣化の目立つ低周波成分と除算を行う値は小さい値とし，情報量を減らしても劣化の目立たない高周波成分と除算を行う値は大きい値とすることで効果的に圧縮を行う．また，輝度成分用と色差成分用で異なる量子化テーブル用意する．ここで，色空間変換の時と同様に，視覚は色の変化に鈍感であるという性質を利用して色差成分用の量子化テーブルの方が大きい値にする．JPEG 規定の Annex K ガイドラインにある輝度成分用の量子化テーブルの例を表 13(a)に，色差成分用の量子化テーブルを表 13(b)に示す.

表 13　量子化テーブル

(a)　輝度成分用

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

(b)　色差成分用

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

4.1.5 ジグザグスキャン

離散コサイン変換，量子化の処理によって，低周波成分(8*8 ブロックの左上)から高周波成分(8*8 ブロックの右下)に向かうにつれて値の変化がだんだん少なくなるように変換されている．そこで，次のハフマン符号化によって符号量を効果的に削減できるようにするため，図 27 のように，ジグザグ順にデータを並び替える．この処理によって同じ値(後ろの方は 0)が連続するようなデータの並びになる．

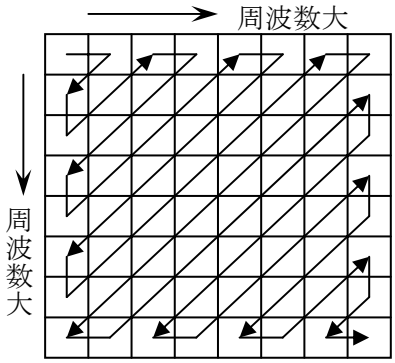


図 27　ジグザグスキャン

4.1.6 ハフマン符号化

ハフマン符号化では統計的な頻出頻度に応じてハフマン符号に置き換える．頻出度が高い値に短いハフマン符号を割り当てる事で，符号量の削減を行う．ジグザグスキャンにより並べられた 8*8 ブロックの先頭要素は DC 成分と呼ばれ，ブロック内の画素値の平均を表す．先頭以外の要素ブロックは AC 成分と呼ばれ，ブロックにおける周波数成分の明るさの変化を表す．このように DC 成分と AC 成分は性質が異なるため，別々の処理を行う．

- **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 成分と値が近いため，直前のブロックの DC 成分との差分を，ハフマン符号表に沿って符号化する．表 14 に AnnexK ガイドラインに示されている DC 成分用のハフマン符号表の例を示す．

例えば，前後のブロックの DC 成分の差分値が-4 の場合，表 14 よりグループ番号は 3 となり，ハフマン符号は 100 となる．次に，-7,-6,-5-4,4,5,6,7 の内どれに当たるのかを示す付加ビット数は 3bit で，3 桁の 2 進数で表すことができる．この場合，付加ビットは 000, 001, 010, 011…と順に割り当てられるので，小さい方から 4 番目にある-4 の付加ビットは 011 となる．よって，-4 はハフマン符号と付加ビットから 100011 と表すことができる．

表 14　DC 成分用ハフマン符号表（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

- **AC 成分のハフマン符号化**

AC 成分は量子化とジグザグスキャンにより，0 の値が多く並ぶ．この特徴を利用して，連続する 0 の数(ゼロランレングス)と，その後に続く 0 でない値をまとめてハフマン符号化する． 表 15 に AnnexK ガイドラインに示されている AC 成分のハフマン符号表の例を示す．また，0 でない値のグループ番号と付加ビット数は，DC 成分と同様の表 14 を用いる．

例えば，0 が 2 つ連続した後に+2 が続く場合，ゼロランレングスは 2 である．また，グループ番号は表 14 を参照し，2 であることが分かる．よって，表 15 より，ハフマン符号は 11111011 となる．また，表 14 より，グループ内の -3,-2,+2,+3 のどれに当たるかを示す付加ビットは 2bit で，2 桁の 2 進数で表すことがわかる．この場合，付加ビットは 00,01,10,11 と順に割り当てられるので，+2 は 10 となる．よって，0 が 2 つ連続した後の +2 は，1111101110 と表すことができる．

表 15　AC 成分用ハフマン符号表（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

（G：グループ番号，Z：ゼロランレングス）

4.2　CPU 過負荷部の調査

ハードウェアとソフトウェアの切り分けパターンを選出するための参考データとして，CPU の過負荷部の調査を行った．AES の調査時と同様に，MicroBlaze システムにクロックカウンタを設け，システム上で JPEG エンコードのアプリケーションを実行し，機能ブロック別に必要なクロック数を計測した．また，C プログラムをコンパイルする際の最適化レベルは「Level 2」とし，計測に用いたデータは，ある写真画像から 1MCU 切り取ったものである．ここで，サブサンプリングは色空間変換に含め，ジグザグスキャンは量子化に含めて処理させている．全体の処理に対する各機能ブロックの CPU 負荷の割合をグラフにしたものを図 28 に示す．

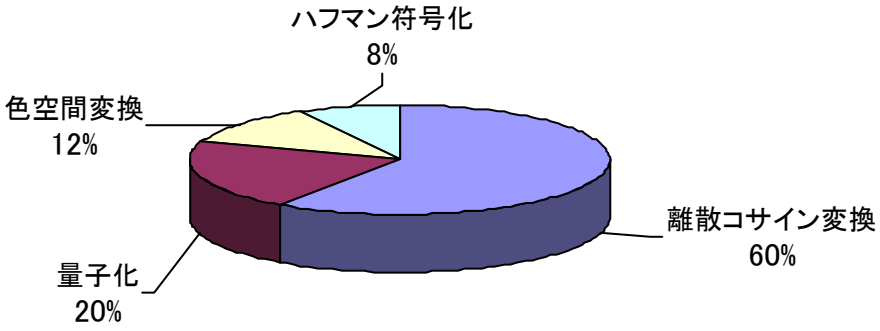


図 28　JPEG エンコードの CPU 負荷

図 28 より，離散コサイン変換（DCT）の CPU 負荷が 6 割を占め，圧倒的に高いことが分かる．離散コサイン変換は，積和演算を中心とした演算を行っており，特に乗算の回数は 1MCU 当たり 6144(8*2*64*6）回実行されている．これが，過負荷となっている原因であると考えられる．

一方，色空間変換においても乗算を行っており，こちらは 1152(3*64*6)回実行される．これは，離散コサイン変換における乗算回数の約 5 分の 1 の回数である．図 28 を見ると確かに，負荷率も 5 分の 1 となっており，乗算の回数が CPU 負荷に深く関係していることが分かる．

また，量子化は 384(64*6)回の除算が実行されている．色空間変換の乗算回数と比較すると，約 3 分の 1 の演算回数であるが，CPU 負荷を見ると量子化の方が倍近く負荷がかかっていることが分かる．よって，乗算器より除算器のハードウェア化のほうが CPU 負荷に対する貢献度が高くなると考えられる．

ハフマン符号化の処理には，除算，乗算は含まれず，分岐，テーブル参照，シフトが主な処理となっており，アルゴリズムは他と比較して極端に複雑である．よって，ハードウェア化するよりソフトウェア処理が有利である．

4.3　各処理部のハードウェア実現方法

JPEG 評価システムではデータ転送に DMA(Direct Memory Access)転送を用いる．これは，JPEG エンコードアルゴリズムはデータの内容によって，エンコードに必要な時間が変わってくるため，大量のデータで評価を行う必要があるからである．各処理部のハードウェアにはバッファ用の RAM があり，それぞれ 1MCU を処理するのに必要なサイズを持っている(4.5 で詳しく述べる)．各処理部が 1MCU の処理を終える毎に，DMA 転送によってバッファ RAM からバッファ RAM へ DMA 転送でデータを転送するようになっている．DMA 転送により CPU に負荷をかけることなく，大量のデータを転送することができる．

また，これから述べるそれぞれの JPEG 専用ハードウェアの制御部には，バッファ RAM への読み書きを行うためのアドレスと，書き込みイネーブル信号の生成を行う機構を備えており，バッファ RAM と連携して動作するようになっている．

4.3.1 色空間変換

それぞれ 8bit の R，G，B の値と，定数テーブルに用意した係数を乗算して，それらを加算することで，Y，U，V の値を計算する．また，サブサンプリングも同時に行っている．図 29 に色空間変換のブロック図を示し，表 16 にその入出力を示す．

回路の内部構成は RGB のそれぞれと乗算を行うための 8bit＊8bit の乗算器を 3 つ，乗算結果を足し合わせる 16bit の加算器を 1 つと，乗数となる定数を格納しておく定数テーブルを用意している．また，それぞれの演算器の間にレジスタを入れて，3 段のパイプライン構成をとっており，動作周波数の向上を狙っている．よって，RGB データを入力してから 3 クロック後にその RGB データに対する結果が出力される．

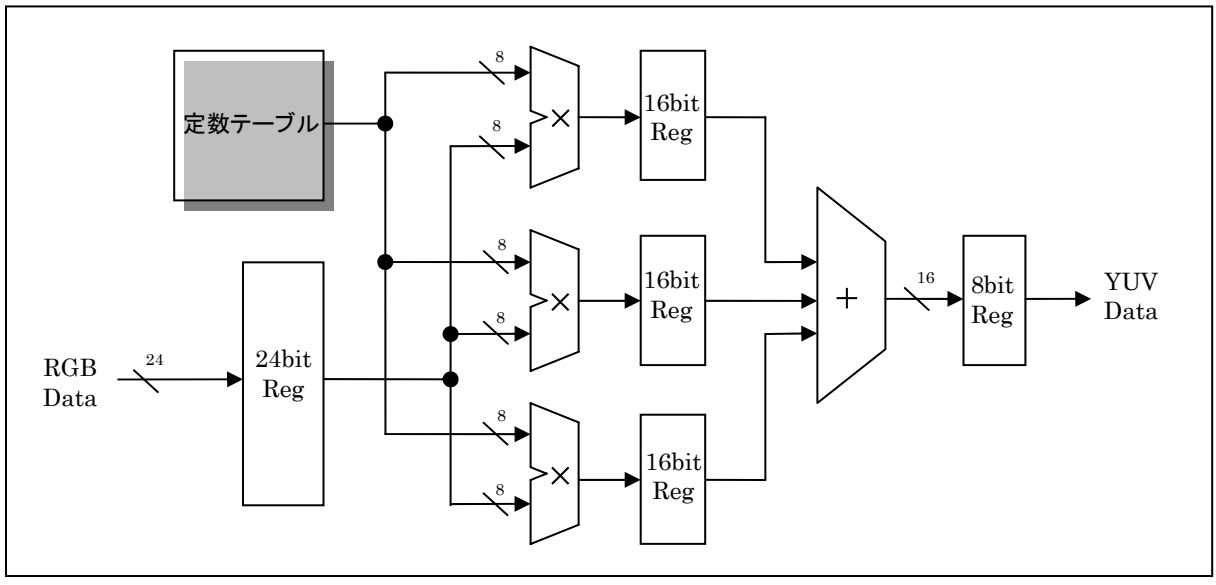


図 29　色空間変換処理ブロック図

また，同時に 4:1:1 のサブサンプリングはアドレス生成の変更で行っている．バッファ RAM への読み書き時のアドレス生成の順を，図 26 のように輝度を表す Y は Y1，Y2，Y3，Y4 の順に，U，V は 1 つおきにアクセスするようなアドレス生成を行っている．このようにすることで，新たにハードウェアを用意することなく，サブサンプリングが行える．

表 16　色空間変換回路の入出力

名前	方向	ビット幅	用途
out_data	out	[0:7]	YUV データの出力
in_data	in	[0:23]	RGB データの入力
read_addr	out	[0:7]	RGB 用バッファ RAM への読み込みアドレス
write_addr	out	[0:8]	YUV 用バッファ RAM への書き込みアドレス
unit_out_en	out	[0:0]	YUV 用バッファ RAM への書き込みイネーブル信号
run	in	[0:0]	システムからの処理開始信号
fin	out	[0:0]	システムへの処理完了信号
rst	in	[0:0]	リセット
clk	in	[0:0]	クロック

4.3.2 離散コサイン変換（DCT）

輝度色差を表す YUV データに，2 次元の離散コサイン変換を行い，周波数成分を表す DCT 係数を計算する．ハードウェア化する際，回路規模を抑えるために，図 30 のように 1 次元離散コサイン変換を行ったデータを転置して，もう一度 1 次元離散コサイン変換を行うことで，2 次元の離散コサイン変換を実現する．

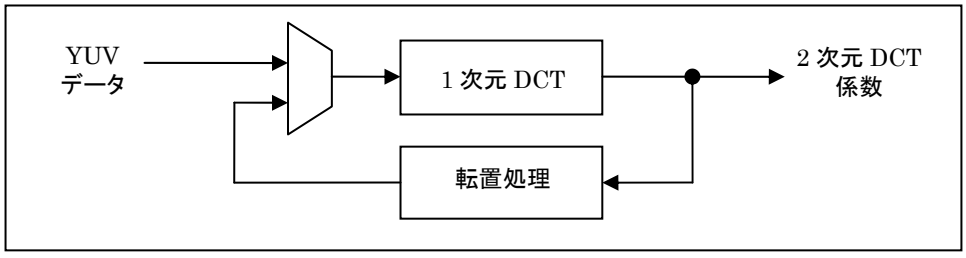


図 30　1 次元 DCT による 2 次元 DCT の実現

図 31 に 1 次元離散コサイン変換によって 2 次元離散コサイン変換を実現したブロック図を示し，表 17 にその入出力を示す．

中心には 11bit*11bit の積和演算器があり，これが，1 次元離散コサイン変換の演算を行っている．その演算時に必要なコサイン値は，あらかじめ計算しておいてコサインテーブルとして用意しており，適時必要なコサイン値を読み出して用いる．また，1 次元離散コサイン変換の結果を一時保存しておく 8*8 のバッファが用意してあり，バッファへ書き込

み時は列方向に書き込みを行い，行方向に読み出すことで，転置処理を行っている．

動作周波数の向上を狙うために，積和演算器内の乗算器と加算器の間にはレジスタを挿入している．よって，処理開始信号(run)で開始してから，処理完了信号(fin)を出すまで 140 クロック必要とする．

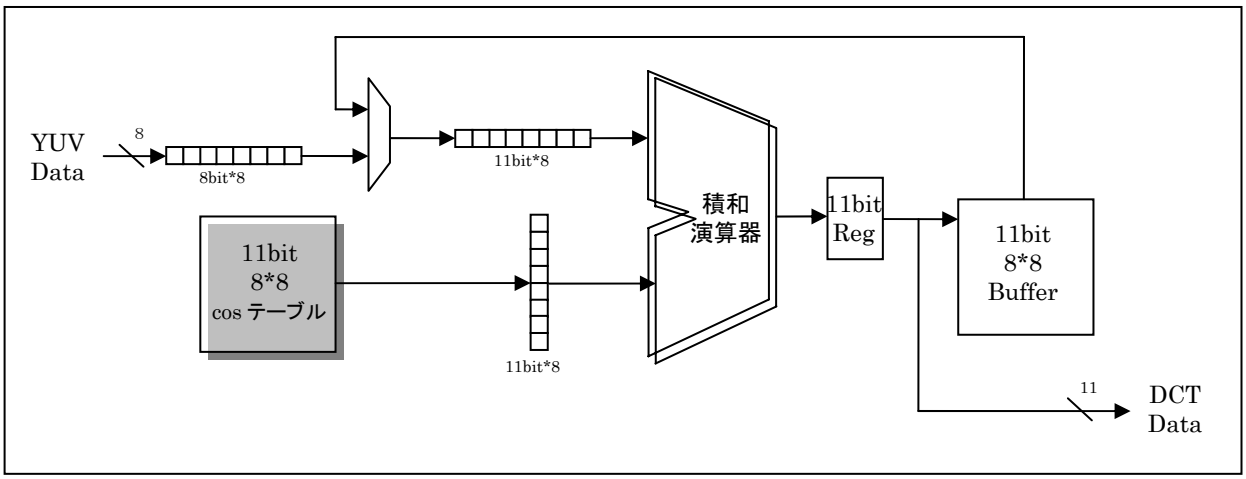


図 31　離散コサイン変換処理ブロック図

表 17　離散コサイン変換回路の入出力

名前	方向	ビット幅	用途
out_data	out	[0:10]	DCT 係数の出力
in_data	in	[0:7]	YUV データの入力
read_addr	out	[0:8]	YUV 用バッファ RAM への読み込みアドレス
write_addr	out	[0:8]	DCT 係数用バッファ RAM への書き込みアドレス
unit_out_en	out	[0:0]	DCT 係数用バッファ RAM への書き込みイネーブル信号
run	in	[0:0]	システムからの処理開始信号
fin	out	[0:0]	システムへの処理完了信号
rst	in	[0:0]	リセット
clk	in	[0:0]	クロック

4.3.3 量子化

DCT 係数を，量子化テーブルに定義された値で除算を行い，近似を行っている．また，ジグザグスキャンによる並び替えも同時に行っている．図 32 に量子化のブロック図を示し，表 18 にその入出力を示す．

演算の中心は除算器と量子化テーブルからなっており，入力の DCT 係数の輝度成分，色差成分に応じて，量子化テーブルを切り替えて使用している．また，除算器の内部は 12 段のパイプライン構造となっており，データの入力後，12 クロックで入力データに対する値が出力される．

出力時のバッファ RAM への書き込みアドレス生成時に，ジグザグスキャンテーブルを参照し，低周波数成分から高周波成分へ順に並び替えている．

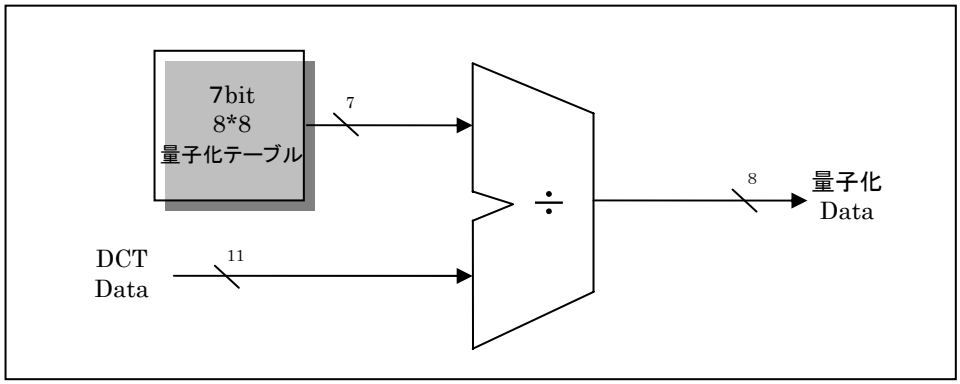


図 32　量子化処理ブロック図

表 18　量子化回路の入出力

名前	方向	ビット幅	用途
out_data	out	[0:7]	量子化データの出力
in_data	in	[0:11]	DCT 係数の入力
read_addr	out	[0:8]	DCT 係数用バッファ RAM への読み込みアドレス
write_addr	out	[0:8]	量子化データ用バッファ RAM への書き込みアドレス
unit_out_en	out	[0:0]	量子化データ用バッファ RAM への書き込みイネーブル信号
run	in	[0:0]	システムからの処理開始信号
fin	out	[0:0]	システムへの処理完了信号
rst	in	[0:0]	リセット
clk	in	[0:0]	クロック

4.3.4 ハフマン符号化

量子化とジグザグスキャンによって，符号化を効果的に行えるデータを，ハフマン符号表に基づいて，ハフマン符号に変更し，符号量の削減を行う．図 33 にハフマン符号の処理ブロック図を示し，表 19 に入出力を示す．

回路構成は大きく，入力・前処理部，ゼロランレングス判定部，符号生成部，符号結合・後処理部の 4 つからなっている．主に，テーブル参照，加減算，シフト，分岐などから構

成されており，処理内容は複雑である．

入力・前処理部では有効成分の判定や，DC 成分の場合，直前のブロックの差分をとるなどの処理を行っている．ゼロランレングス判定部では，AC 成分の処理の際，連続した 0 をカウントする処理を行い，0 の数をハフマンテーブルに送信し，参照を行っている．符号生成部ではハフマンテーブルへの参照結果から，グループ符号，グループ符号長，付加ビット，付加ビット長を生成する．符号結合・後処理部ではそれらの結果を符号列として結合し，8bit ずつ出力する処理を行っている．

また，入力・前処理部，符号生成部，符号結合・後処理部にはそれぞれレジスタを持っており，3 段のパイプライン構造をとっている．よって，データ入力から出力まで，最低 3 クロックかかる，データの内容によっては，それ以上かかる場合もある．

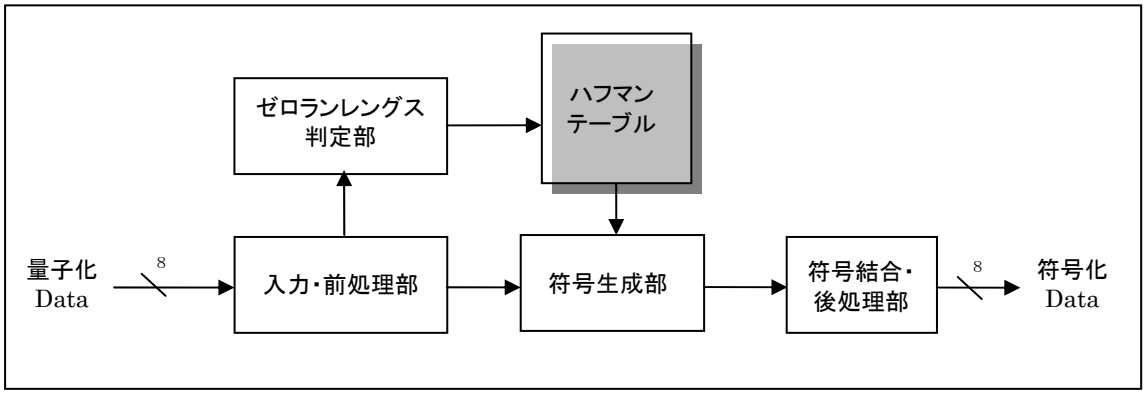


図 33　ハフマン符号化の処理ブロック図

表 19　ハフマン符号化回路の入出力

名前	方向	ビット幅	用途
out_data	out	[0:7]	ハフマン符号化データの出力
in_data	in	[0:7]	量子化データの入力
read_addr	out	[0:8]	量子化データバッファ RAM への読み込みアドレス
write_addr	out	[0:31]	ハフマン符号化データ用バッファ RAM への書き込みアドレス
unit_out_en	out	[0:0]	ハフマン符号化データ用バッファ RAM への書き込みイネーブル信号
run	in	[0:0]	システムからの処理開始信号
fin	out	[0:0]	システムへの処理完了信号
all_image_in_fin	in	[0:0]	システムからの最後のデータを入力したことを示す信号
all_image_out_fin	out	[0:0]	システムへの最後のデータを出力したことを示す信号
rst	in	[0:0]	リセット
clk	in	[0:0]	クロック

4.4　ハードとソフトの切り分け

AES の場合は, 連続ブロックで切り分ける際に演算の最適化によるメリットとデータ転送量の削減のメリットが考えられた．しかし，今回の JPEG 評価システムでは DMA 転送を採用しており，データ転送にかかる CPU 負荷をできる限り削減している．また，演算の最適化のメリットに関しては，各機能ブロックが，完全に独立したアルゴリズムを持つため，あまり見込めないと考えられる．そこで，今回は JPEG エンコードの機能ブロック単位での切り分けを評価する．

4.2 で調査した CPU 過負荷部を考慮した上で, ハードウェアで処理する部分とソフトウェアで処理する部分を選出した切り分けパターンを表 20 に示す.

表 20　JPEG エンコードの切り分けパターン

	ハードウェア処理部	ソフトウェア処理部	備考
S		色空間変換，DCT，量子化， ハフマン符号化，制御	ソフトウェアで全処理 評価基準となる構成
A	量子化	色空間変換，DCT， ハフマン符号化，制御	回路規模の最も小さい量子化をハードウェア化した回路規模を考慮する構成
B	色空間変換	DCT，量子化， ハフマン符号化，制御	回路規模の小さい色空間変換をハードウェア化した回路規模を考慮する構成
C	DCT	色空間変換，量子化， ハフマン符号化，制御	最も CPU 負荷の高い DCT のみをハードウェア化する構成
D	DCT，量子化	色空間変換， ハフマン符号化，制御	パターン A と C を組み合わせた構成
E	色空間変換，DCT	量子化，ハフマン符号化， 制御	パターン B と C を組み合わせた構成
F	色空間変換，DCT， 量子化	ハフマン符号化，制御	回路規模を考慮せず, アルゴリズムが複雑ではない部分をハードウェア化する構成
G	DCT，量子化， ハフマン符号化	色空間変換，制御	演算の多い DCT や，大きいテーブルを持つ量子化, ハフマン符号化をハードウェア化し, 使用メモリを考慮した構成
H	色空間変換，DCT， 量子化，ハフマン符号化	制御	制御以外をハードウェア化し，CPU で他の処理をすることができる構成

4.5　JPEG 評価システムの実装

4.5.1 JPEG 評価システムの基本構成

前節で述べた全てのパターンの JPEG 評価システムを設計し，MEMEC 社の FPGA ボード DS-KIT-MB-S2E6LC-EURO に実装した．図 34 に JPEG 評価システムの基本構成を示す．

メモリとして，MicroBlaze プロセッサコアに高速の LMB（Local Memory Bus）を介して接続した BlockRAM と，OPB に SDRAM コントローラを接続し，これを介して接続した外部 SDRAM を用意した．LMB にて接続された BlockRAM は高速ではあるが，容量が少ない(現在の仕様で MicroBlaze システムに接続できる BlockRAM は最大 16KB までである)ため，命令と実行途中の一時データを格納するようにし，大容量の SDRAM にはエンコード対象のデータを格納している．これは，メモリへのデータ配置を指定するリンカスクリプトを編集することで指定した．

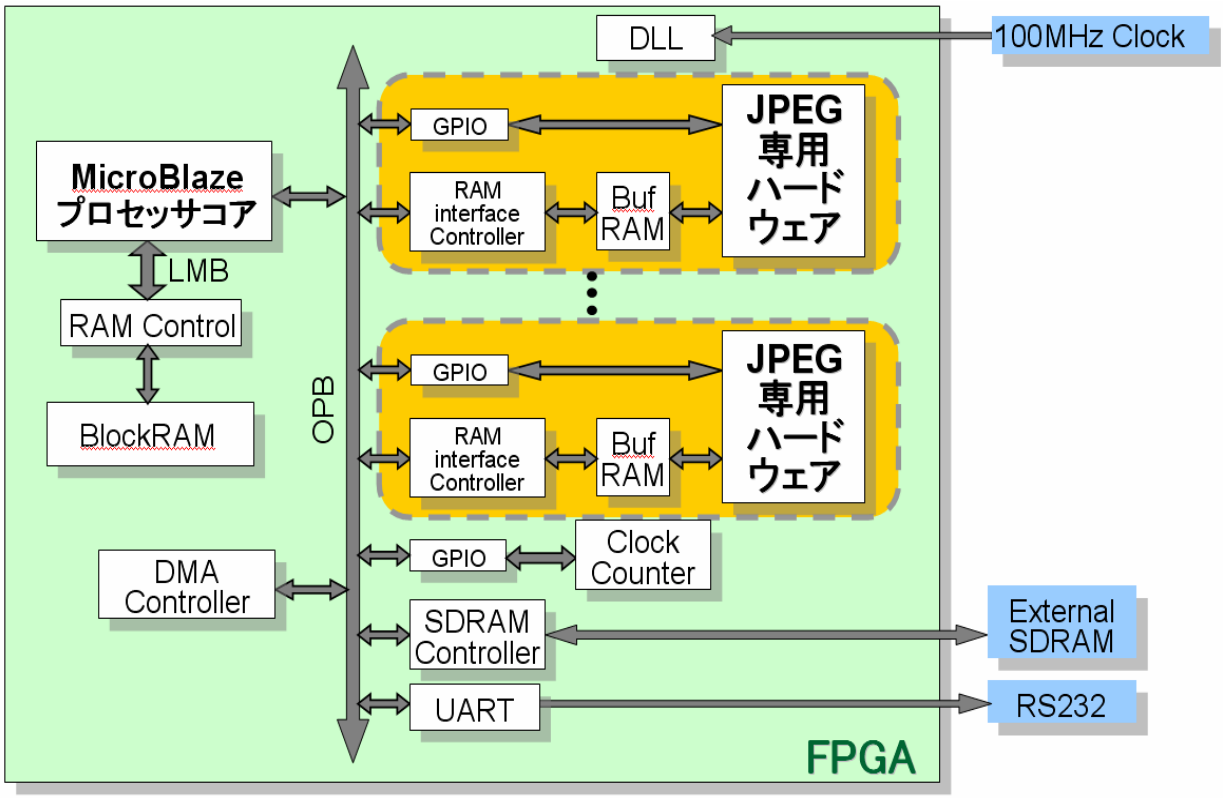


図 34　JPEG 評価システム構成

切り分けパターンにおけるソフトウェア処理に当たる部分は, AES 評価システムと同様に MicroBlaze プロセッサコアが行う．ハードウェア処理に当たる部分は，VerilogHDL で設計した JPEG 専用ハードウェアを，RAM interface Controller を介して接続したバッファ用の RAM に接続している．ハードウェアとのデータ送受信はこのバッファ RAM を介し

て DMA 転送で行う．また，ハードウェアに接続した GPIO は制御用の信号をやり取りするためのものである．

この構成をベースとして，JPEG 専用ハードウェアの着脱や，ソフトウェアの改変を行うことで，前節で挙げた切り分けパターンを実装し評価した．

4.5.2 JPEG 評価システムの実行フロー

始めに外部 SDRAM に格納されているデータを取得する. 始めの処理の割り当てがソフトウェアの場合は，SDRAM からデータを直接参照しながら実行する．ハードウェアの場合は，DMA を起動し，SDRAM から専用ハードウェアのバッファ RAM に，必要な分のデータを転送する．転送終了後，MicroBlaze は専用ハードウェアに処理開始信号を送る．専用ハードウェアは，処理開始信号を確認し，バッファ RAM からデータを読み取りながら処理を行い，処理結果をバッファ RAM に書き込む．すべての実行が終了すると，ハードウェアは，処理完了信号を MicroBlaze に送信する．MicroBlaze はポーリングによって処理完了信号を確認し，次の処理に移る．

次の処理もハードウェアの場合は，再び DMA を起動し，同様の手順でバッファ RAM から，次の専用ハードウェアのバッファ RAM へデータを転送する．次の処理の割り当てがソフトウェアの場合は，バッファ RAM からデータを参照しながら処理を行う．

4.5.3 周辺機器

(1) DMA Controller

JPEG 評価システムではデータ転送に DMA(Direct Memory Access)を用いている. JPEG アルゴリズムではエンコード対象データの内容によって, 大きく処理時間が変わるため, 大量のデータを用いて評価する必要がある. DMA Controller を OPB に接続し, SDRAM とバッファ用 RAM との間で，DMA 転送が可能なように実装した．MicroBlaze から DMA Controller 内のレジスタにソースアドレス, ディスティネーションアドレス, 転送バイト数などの値を設定することで，DMA 転送を行うことができる．

(2) GPIO（General Purpose Input/Output）

データを送受信する目的ではなく，システムとハードウェアとの制御信号の通信の目的で GPIO を用いている．

(3) RAM interface Controller

RAM interface Controller はバッファ用 RAM を OPB に接続するためと，バースト転送に対応させるためのインタフェースである．今回は BlockRAM 用のインタフェースコントローラである OPB Block RAM Interface Controller v1.00a を分散 RAM 用に改変し，分散 RAM であるバッファ用 RAM に接続している．

図 35 にこれらを用いて 4.3 で述べた JPEG 専用ハードウェア(JPEG Function Module)を接続した様子を示す.

データの流れは以下のようになっている.

① OPB から DMA によってバースト転送されたデータは BUS2IP_Data を通って BUS2IP_Buffer に格納される

② DMA 転送の終了後，システム側(MicroBlaze)は GPIO 経由で Run 信号を送信する

③ JPEG 専用ハードウェアは，システムからの Run 信号を確認し，アクティブならば BUS2IP_Buffer にアクセスし，Read_Data を通してデータを読み込む

④ データを処理し，結果を IP2BUS_Buffer に格納する

⑤ 処理が完了したことをシステム側に知らせるために，Finish 信号をアクティブにする

⑥ システム側は GPIO の Finish 信号がアクティブになったのを確認し, IP2BUS_Buffer からデータを DMA 転送する

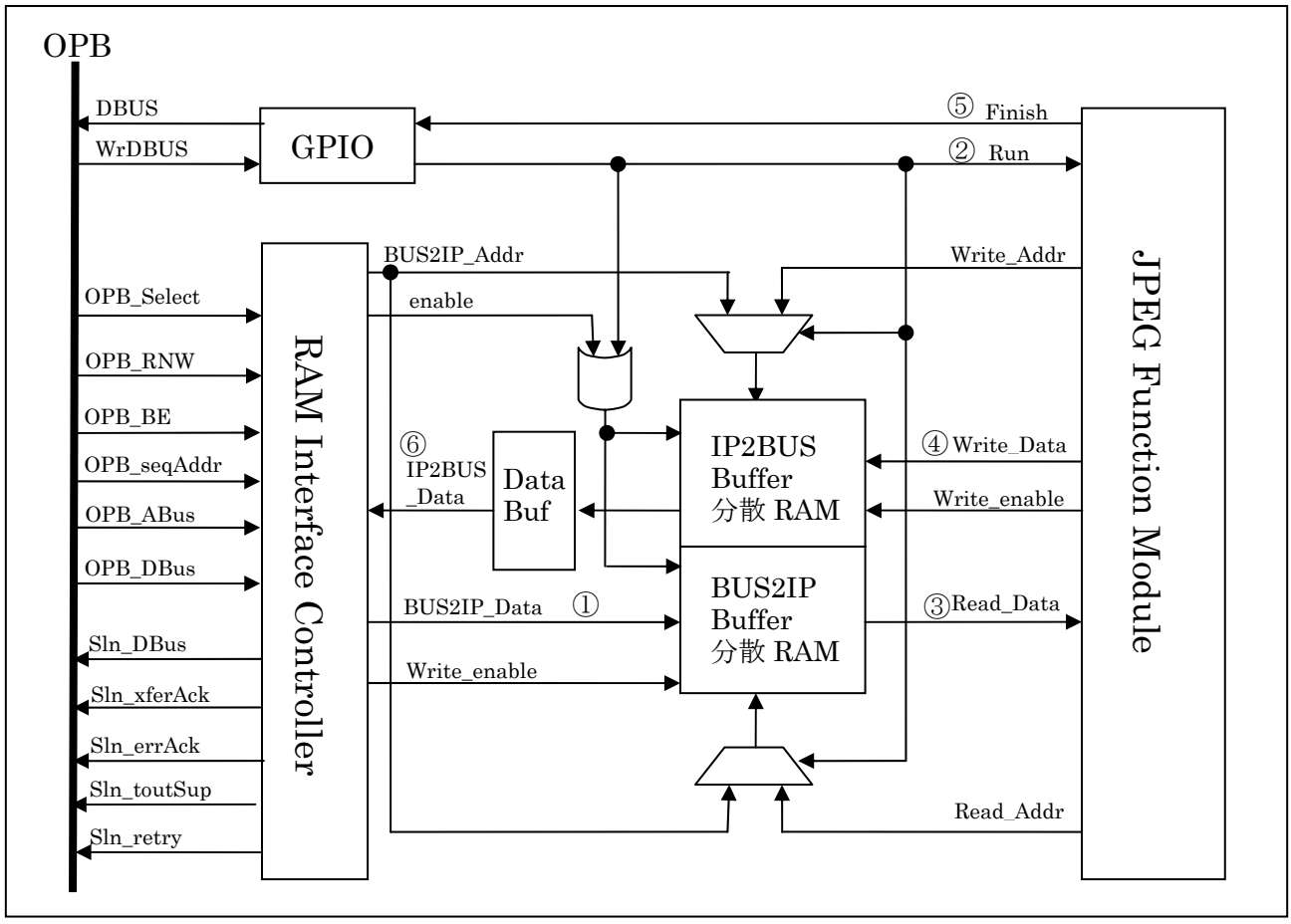


図 35　RAM Interface Controller を用いた分散 RAM の接続

4.5.4 ハードウェアとのデータ送受信プログラム

DMA によるデータ送受信は，MicroBlaze システム開発環境ツール EDK が用意している DMA コントローラドライバを用いることで行った．今回は，ドライバ dmacentral_v1_00_a の low レベルドライバ xdmacentral_l.h を用いた．

図 36 に，DMA によるデータ転送を行うプログラムを示す．DMA の転送設定は，DMA コントローラ内のレジスタ内に設定値を書き込むことで行う．始めにレジスタをリセットし，次のような初期設定を DMA コントローラ内のレジスタに書き込む必要がある．送信側，または受信側が，メモリのようなアドレス幅を持つ機器である場合は，バースト転送時に，アドレスをインクリメントしながら転送させるように設定する．また，今回の MicroBlaze システムでは 32bit バス上でデータ転送を行うので，4Byte 単位で転送するように設定している．転送開始命令は，起点アドレスと終点アドレスを設定後，DMA 内のレジスタにデータ転送量をバイト単位で書き込むと，それと同時に DMA コントローラが起動し，転送が開始される．

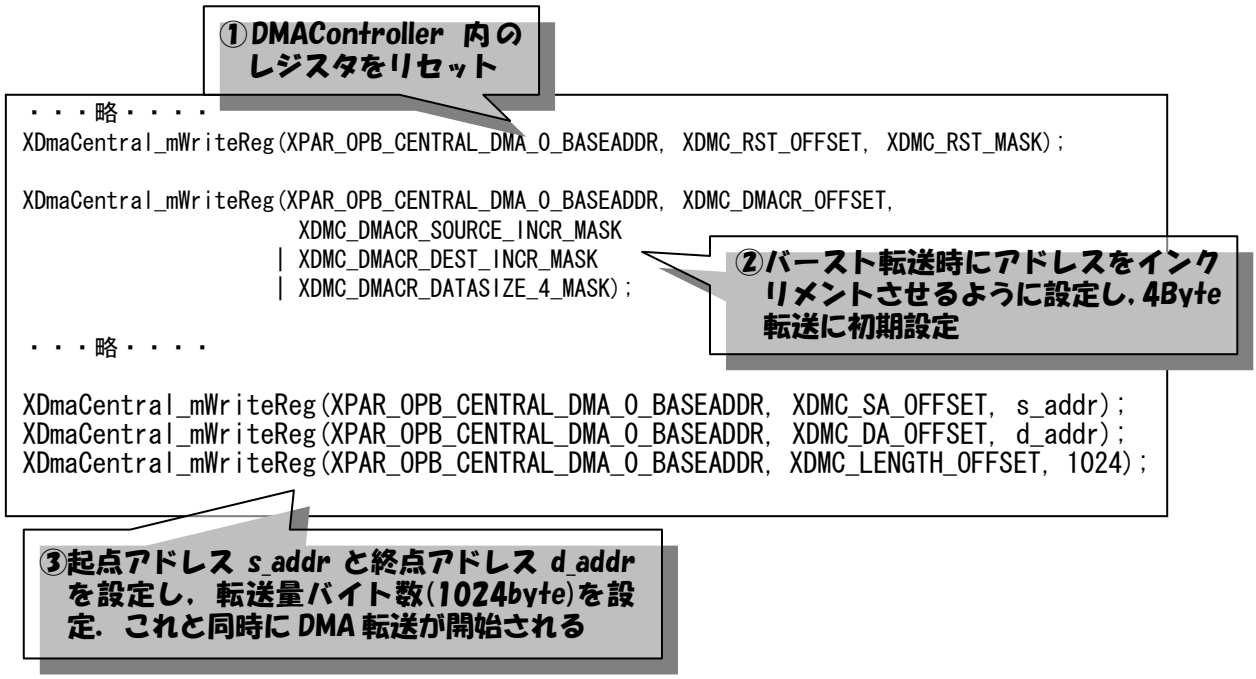


図 36　DMA 転送プログラム

4.6　実行結果

4.6.1 切り分け評価結果

表 21，図 37 に 4.4 で選出した JPEG 評価システムの切り分け結果を示す．

結果は，1MCU を JPEG エンコードするのに必要なクロックサイクル数と使用メモリ量である．クロックサイクル数は MicroBlaze システムに接続したクロックカウンタを用いて計測した値である．使用メモリ量は MicroBlaze がプログラム実行に必要なメモリ量であり，XPS メニューの Tool > Get Program Size のレポートから算出した．回路規模は，ISE の Map > Map Report の Total equivalent gate count for design の値である．ただし，表に示した値はハードウェア化した JPEG 専用ハードウェアの規模の総和であり，MicroBlaze とその周辺機器，バッファ用 RAM の回路規模は含まれていない．

表 21　JPEG エンコーダの切り分け結果

	S	A	B	C	D	E	F	G	H
クロックサイクル数	781522	611439	697408	321998	160787	227262	66057	98889	4125
使用メモリ(Byte)	8372	6988	6772	6508	5332	4820	3644	3019	2244
回路規模(gate)	0	4403	4986	35120	39523	40106	44509	49857	54843

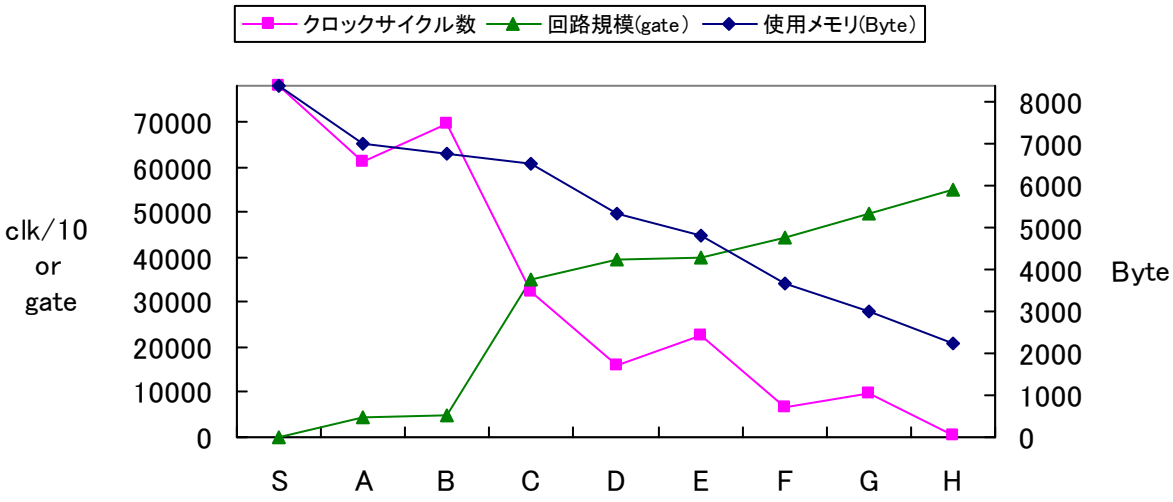


図 37　JPEG エンコーダの切り分け結果

4.6.2 考察

図 37 から，まず，目に付くのは，離散コサイン変換のみをハードウェア化した C の切り分け結果である．S→C の変化に注目すると，ハードウェア規模は大きく増大しているが，クロックサイクル数短縮の効果も大きく出ている．また，演算に用いる領域と，コサインテーブルが，メモリ上に必要なくなったため，使用メモリ量も削減できている．

次に，量子化のみをハードウェア化した S→A と，色空間変換のみをハードウェア化した S→B の変化に注目すると，回路規模は同程度で，あまりゲートを消費せずに，クロックサイクル数を削減できている．特に量子化のみをハードウェア化した時のクロックサイクル数の短縮量は，先述した離散コサイン変換による短縮量の約 3 分の 1 もある．一方，色空間変換のみをハードウェア化したときのクロックサイクル数は，あまり削減できていない．よって，回路規模に余裕があり，離散コサイン変換をハードウェア化しても，多少余裕が残っている場合は，離散コサイン変換と量子化をハードウェア化した D が最適であると言える．

次に，D に色空間変換のハードウェア化を含めた F，D にハフマン符号化のハードウェア化を含めた G の 2 つが候補に挙がる．D→F，D→G に注目すると，これ以上，回路規模に余裕がなく，クロックサイクル数の短縮をさらに目指す場合は，F が候補に挙がり，回路規模にはまだ余裕があるがメモリ使用量を削減したい場合は，G が候補に挙がるといえる．なお，G はハフマン符号表をハードウェア化することで，メモリ上に置く必要なくなったため，使用メモリを大きく削減できており，また，分岐などの複雑な制御が必要であるため，回路規模の増加量は大きくなっている．

5. 分割の評価

5.1　評価式による結果検証

3 章，4 章の評価結果から得た実測値とグラフには，トレードオフが複雑に関係しており，様々な条件から最適な切り分けパターンを一概に見つけ出すのは困難である．そこで，図 38 に示すような最適な切り分けパターンを出力する評価フローを用意して評価を行った．グラフから見た直感ではなく，後述する評価式を用いることで，それぞれのアプリケーションと制約条件に対する最適な切り分けパターンの優先順位を，数値として出力する事ができる．

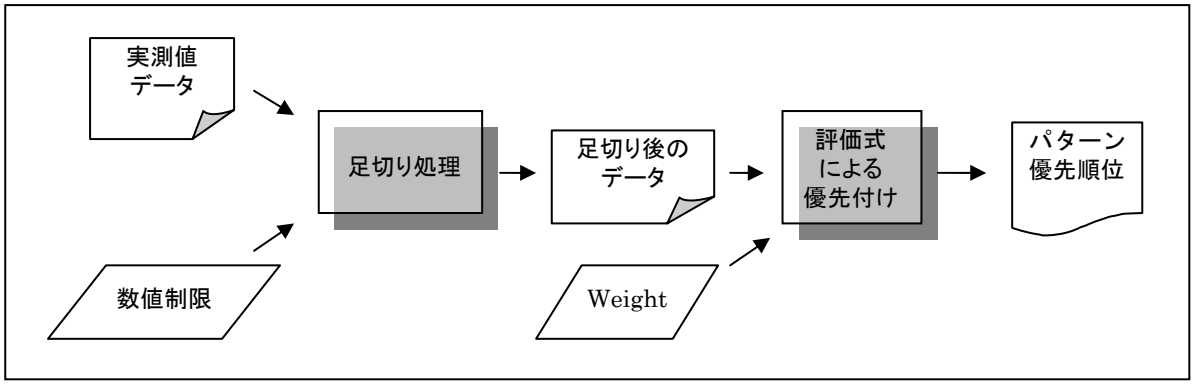


図 38　優先順位評価フロー

本評価フローは，まず，実測値データに数値制限を与え，足切り処理を行う．次に，各評価項目への重み付けと下記に示す評価式を用いて，パターンに優先順位をつける．

$$Priority_{pattern} = \sum_{\substack{Item=gate,\\ memory,\\ clock\cdots}} \left(Weight * \frac{Value_{Worst} - Value}{Value_{Worst} - Value_{Best}} \right)_{Item}$$

$Priority_{pattern}$ ：パターンごとの優先度(Priority)を表す値(ただ $0 \le Priority \le 1$)
$Item$ ：回路規模，クロックサイクル数，使用メモリなどの評価項目
$Value$ ：3 章，4 章の評価結果から得た実測値，
$Worst$ ， $Best$ ：実測値の内の最悪値と最良値
$Weight$ ：制約からの重み付け（ただし $Weight_{gate} + Weight_{memory} + Weight_{clock} + \cdots = 1$）

この評価式は，ある評価項目において，全てのパターン中で最良である実測値を 1 とし，最悪である実測値を 0 としたときの，評価対象となるパターンの実測値の割合を出し，そ

れに重み(制約)を掛け合わせている．このようにして，全ての評価項目に対する重み付き割合を算出し，それらを足し合わせることで，そのパターンの優先度を数値として算出している．つまり，与えた制約条件を考慮した時に，その評価対象のパターンがどのくらいの優先度を持つかを数値として表すことができる評価式である．図 39 に，この評価式を C プログラムし，実際に評価した例を示す．

図 39(a)は，AES 評価システムの評価結果(表 11 表 12)対して，制限が回路規模 6000gate 以下で，クロックサイクル数 : 回路規模 : 使用メモリの重み付けを 0.1 : 0.2 : 0.7 とした場合の評価結果である．この重み付けの値は大きいほど制約が厳しいということを示す．まず，G，ロ，(ハ)，(ニ)，(ホ)，(ヘ) は足切りで対象外となる．次に評価式による結果の Piority 値をみると，D が最も大きい値をつけている．よって，この条件の場合，D が最適な切り分けであるといえる．

図 39(b)は，JPEG 評価システムの評価結果(表 21)に対して，制限がクロックサイクル数 400000 以下で，クロックサイクル数 : 回路規模 : 使用メモリの重み付けを 0.3 : 0.6 : 0.1 とした場合の評価結果である．まず，S，A，B はクロックサイクル数の制限で足切りされ，対象外となる．次に評価式の結果の Piority 値をみると，D が最も大きい値となり，この条件の場合，D が最適な切り分けであるといえる．

```
%a.out 0.1 0.2 0.7 aes.txt
S:38075    0 4387
A:26959  324 3700
B:25859 4800 4024
C:14743 5124 3572
D:11313 5124 3496
E:13979 5316 3812
F:10549 5316 3736
ｲ:21996 5280 3832

MAXc:38075, MINc:10549
MAXg: 5316, MINg:    0
MAXm: 4387, MINm: 3496

priorityS:0.2000
priorityA:0.7679
priorityB:0.3489
priorityC:0.7322
priorityD:0.8044
priorityE:0.5392
priorityF:0.6114
priorityｲ:0.4957
```

(a) AES の評価結果検証

```
%a.out 0.3 0.6 0.1 jpeg.txt
C:321998 35120 6508
D:160787 39523 5332
E:227262 40106 4820
F: 66057 44509 3644
G: 98889 49857 3019
H:  4125 54843 2244

MAXc:321998, MINc: 4125
MAXg: 54843, MINg:35120
MAXm:  6508, MINm: 2244

priorityC:0.6000
priorityD:0.6457
priorityE:0.5773
priorityF:0.6230
priorityG:0.4440
priorityH:0.4000
```

(b) JPEG の評価結果検証

図 39 評価式による検証の様子

同様にして，低クロックサイクル数重視，小回路規模重視，小メモリ使用量重視，で極端に重みを与え，評価した結果を表 22 に示す．ただし，ここでは制限による足切りは行っていない．また，表中の不等号は左から優先度が高いことを示す．

表 22　極端な重みをつけた場合の評価結果

weight(clk:gate:memory)	低クロック サイクル数重視 0.8 : 0.1 : 0.1	小回路規模 重視 0.1 : 0.8: 0.1	小メモリ使用量 重視 0.1 : 0.1 : 0.8
AES システム	ヘ＞**G**＞**ホ**＞**F**	**A**＞S＞**D**＞**F**	ヘ＞**G**＞**ニ**＞**D**
JPEG システム	H＞**F**＞**G**＞**D**	S＞**A**＞**B**＞**C**	H＞**G**＞**F**＞**E**

表 22 に示すように，このような重み付けの場合，赤字で示したハードウェアとソフトウェアの両方で処理するパターンが，最適な切り分けとして優先度の高いパターンとなることが分かった．ただし，事前に足切りの制限を与えることで，その制限の加減に応じて選択すべき切り分けパターンは異なってくる．

5.2　分割探索手法の提案

本研究で行った実験フローから，今後の研究課題としてソフト・マクロ CPU を用いた分割探索手法を提案する．今回，ソフト・マクロ CPU を用いるメリットを生かした，下記のような有用性を確認することができた．

(1) FPGA 上でハードウェアとして評価できるため，評価結果の信頼度が高い
(2) 切り分けを始めの評価で一本化しないので，切り分けながら設計ができる
(3) 設計途中での仕様変更が容易である
(4) CPU とその周辺機器は IP として用意されているため，専用ハードウェアの設計に注力できる
(5) 短期間で評価できる
(6) 自由度が高いため，様々な切り分けを試すことができる
(7) FPGA またはシミュレータ上で，ハードとソフトを協調検証可能である

ただし，この手法を実用として用いるためには，次のような問題点を改善する必要がある．

前述したように，本手法の有用性の１つとして，初期段階で切り分けパターンを一本化しなくても良いということがある．しかし，今回のように，切り分けを選出するための材料が，アプリケーションアルゴリズムの特性と，アプリケーションの実行による調査によって得た各機能ブロックの CPU 過負荷情報からのみである場合，情報が足らず，多くの切り分けパターンを選出する必要があった．

よって，今後の改善点として，アプリケーションの実行の段階で，どれだけ切り分けのための情報収集ができるかが，無駄な設計を抑える決め手となる．そこで，我々は関連研

究として MicroBlaze の動的実行命令数の統計をとるシステムを現在作成中である．これを用いれば，機能ブロックごとの，演算命令数，分岐命令数，メモリアクセス命令数の割合の統計をとることで，切り分ける際の指標の１つとして利用できる．

また，アプリケーションごとのデータ量の変化を考慮した切り分けを行う必要がある．たとえば，今回，対象アプリケーションとした AES 暗号化は入力データ量と出力データ量は１対１であるが，JPEG エンコードは圧縮技術であるため，１対１にはならず，後になればなるほど，処理するデータ量は減少し，そのハードウェア間の転送量も減少することになる．つまり，全体のアルゴリズムに対する，対象の機能ブロックが必要とするデータの転送量を考慮したうえで切り分けを考える必要がある．

さらに，今回は評価項目として，主にクロックサイクル数，回路規模，使用メモリ量を上げたが，今後，設計期間，消費電力，再利用性，コストなどの評価項目も追加する必要がある．

6. おわりに

本研究では，現在のシステム LSI 設計において，必要となるハード/ソフト・コデザインについて述べ，その中でも，「ハードとソフトの分割探索」に注目した研究を行った．ハードウェアとソフトウェアの分割を意識して，AES 暗号化と JPEG エンコードのそれぞれ対して，さまざまな切り分けパターンを選出した．そして，ソフト・マクロ CPU を用いて全てのパターンのシステムを設計し，FPGA に実装した．

また，この実験フローを，FPGA とソフト・マクロ CPU を用いる分割探索手法として提案した．今回，確認できたこの手法の有用性を次に示す．

「信頼性が高い切り分け」設計したシステムは FPGA 上で動作させて検証を行なうため，実機に近い形で切り分け結果を検証できる

「柔軟な切り分け探索」切り分けを 1 つに絞らずに設計を進めるため，途中の仕様変更に対応できる

「短期間での検証」CPU とその周辺は IP として用意されているものも使用するため，設計者は専用ハードウェアに注力でき，短期間で検証を行なえる

「ハードとソフトの協調検証が可能」FPGA 上にハードとソフトを実装し，動作検証ができる

今後の研究として，第 5 章で述べた問題点を解決するとともに，今回，提案した分割探索のみに留まらず，全体を 1 つのハード/ソフト・コデザインのシステムとしてまとめ上げる必要がある．

現在，100 万ゲートもの規模の FPGA を搭載した FPGA ボードが数万円で手に入る時代であり，個人でそれを手にすることも今や難しくない．だが一方，その規模のシステムを設計するためのハード/ソフト協調設計ツールは，まだまだ雲の上の価格である．これは，需要と供給の問題というより，価格競争が行われていないためであると考えている．

そこで，情報学科から電子情報デザイン学科へ移転したという遍歴を持ち，ハードとソフト両方の知識を持った学生を育てるというテーマを掲げ，異色であり貴重な分野を研究している本研究室から，将来，この技術業界に一石を投じられるくらいの研究成果と発展を遂げてくれることを切に願う．

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梅原氏，的場氏，本研究に関して貴重なご意見をいただきました高性能計算研究室の皆様に心より深く感謝致します.

参考文献

[1] 三木良雄:システム LSI 設計特論講義資料,第 1 章システム LSI とは,立命館大学大学院スターク寄附講座,2003.

[2] 内山邦男:システム LSI 設計特論講義資料,第 3 章 LSI 構成要素Ⅰ,立命館大学大学院スターク寄附講座,2003.

[3] 三木良雄: システムLSI設計特論講義資料,第10章設計事例と今後の課題,立命館大学大学院スターク寄附講座,2003.

[4] 若林一敏:高位言語ベースデザイン設計特論講義資料,第 4 章アーキテクチャレベルコンポーネント生成技術 1,立命館大学大学院スターク寄附講座,2003.

[5] Announcing the ADVANCED ENCRYPTION STANDARD (AES), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[6] 下村高範,安部公輝:暗号アルゴリズム Rijndael のハードウェア実装と評価,情報処理学会研究報告.SLDM, Vol.2003, No.7, pp13-17, 2003.01.

[7] 森岡澄夫,佐藤証:共通鍵暗号 AES の低消費電力論理回路構成法,情報処理学会論文誌,Vol.44, No.5, pp1321-1328, 2003.05.

[8] 岡田壮一,鳥居直哉,長谷部高行:スマートカード向け AES ハードウェアの試作,情報処理学会研究報告.CSEC, Vol.2001, No.75, pp111-118, 2001.07.

[9] 清家秀律,黒川恭一:AES 暗号用 SubBytes,MixColumns 変換の方式設計,情報処理学会研究報告.CSEC, Vol.2001, No.75, pp119-126, 2001.07.

[10] 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, Vol.2003, No.120, pp127-132, 2003.11.

[11] 佐藤証,盛岡澄夫:暗号処理のソフト vs ハード,Design Wave Magazine, pp72-79, 2003.9.

[12] 東原朋成:AES 暗号回路の実現方式を検討する, Design Wave Magazine, pp154-160, 2003.12.

[13] Dr.S.Morioka: 数学いらずの AES 暗号 SubBytes 設計ガイド, Design Wave Magazine, pp152-157, 2004.1.

[14] Indepenfdent JPEG Group, http://www.ijg.org/

[15] CCITT Recommendation T.81, http://www.w3.org/Graphics/JPEG/itu-t81.pdf

[16] 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, Vol2003, No.7, pp93-98, 2003.1.

[17] 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, Vol2003, No.7, pp87-92, 2003.1.

[18] 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, Vol2004, No.5, pp13-16, 2004.1.

[19] 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,1999.1.

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

[21] Xilinx:EDK ユーザーガイド,http://www.xilinx.co.jp/ise/embedded/edk_docs.htm