卒業論文

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

氏　　名：梅原　直人
学籍番号：2210010037-5
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、ハード/ソフト協調設計においてハードウェアとソフトウェアの設計空間を探索する際に問題となる最適分割について述べている。ハード/ソフト最適分割を理解するために Verilog-HDL を用いて AES 暗号システムと JPEG エンコーダを設計し、ソフト･マクロ CPU を利用して FPGA 実装と検証を行った。

ソフト･マクロ CPU の利用はシステム LSI(SoC)の設計を容易にし、検証の効率も従来の FPGA での方法と比べると格段に向上させることを可能とした。FPGA 上に 7 セグメント LED しかなければ実験結果の正当性を確認することは困難であるが、ソフト･マクロ CPU を用いることで解消された。

AES 暗号システムでは機能ブロックごとにモジュールを分割し、複数のパターンを以ってハード/ソフト分割を行い、FPGA へ実装してそれぞれの分割パターンでの結果を数値化して結果を検討した。ここでは出来るだけ網羅的にハード/ソフト分割を行い、クロックサイクル数やメモリの使用量などに関して、影響を及ぼすファクタについて考察した。JPEG エンコーダも AES 暗号システムと同様に機能ブロックごとにモジュール設計をして、ソフトウェアで実行した後に、ハードウェアの設計を行い FPGA ボード上にソフト･マクロ CPU とともに実装した。その後、考案した評価式をもって分割パターンに対しての評価を行った。

その結果、分割パターンの妥当性、FPGA やソフト･マクロ CPU の用いた設計の簡易化、導き出した評価式の正当性を確認した。

目次

図目次

表目次

1. はじめに

1970 年代に LSI という概念が生まれ、その需要・技術は着実かつ急速に伸び続けている。家庭用電化製品からコンピュータ、自動車、携帯電話と LSI が必要とされる分野は増え続けている。一方で集積素子数が 10 万を超えて VLSI と呼称されるようになってなお規模は肥大化している。現在では 1 億トランジスタを集積するようになり、間もなく 10 億トランジスタを突破するだろう。また、単機能ではなくシステム全体を 1 チップ上に搭載する SoC(System on a Chip)が登場してから LSI の複雑さも増大している。

これに伴い、設計技術も進歩を重ねている。設計ではゲートを記述する回路図の作成から、C 言語のような高級言語を記述する感覚で回路を記述できる HDL、さらには C 言語などから直接回路を作成できるようにする高位合成技術へと進展しており、シミュレーション技術もソフトウェア、FPGA、シミュレーションアクセラレータなどの開発でより簡易・高速になっている。しかし、ドッグイヤーとまで呼ばれる LSI の発展にそれでもなお追いつけていないのが現状である。

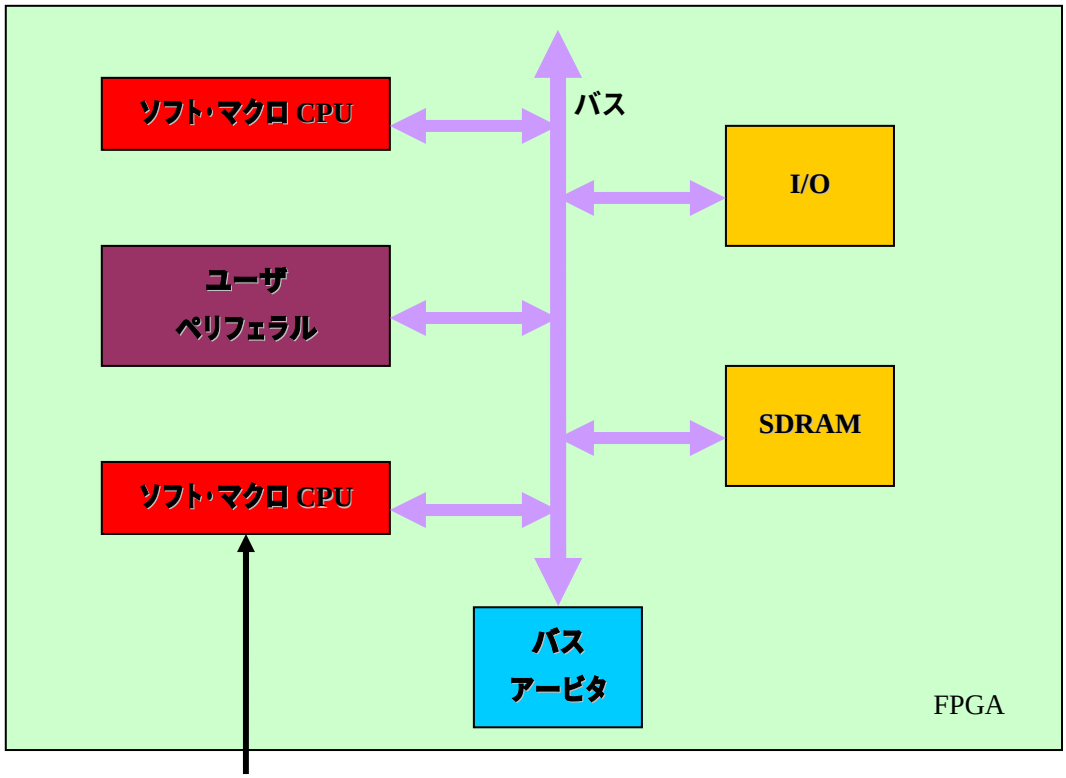
そしてその設計困難の解決策の一つに挙げられるのが「ハード/ソフト協調設計」である。ハード/ソフト協調設計は設計にあたって予め目的や優先事項を決めておき、それに見合うようにシステムのハードウェア部分とソフトウェア部分を分けて設計する手法である。例えば、速度最優先で他の項目を無視出来るならば全て、もしくはほとんどをハードウェアにするべきである。また技術の移り変わりに影響されやすい部分は柔軟性の高いソフトウェアにするべきである。このようにして設計の負担を軽くしようというのがハード/ソフト協調設計である。

このハード/ソフト協調設計という手法の欠点として、考慮する項目が増えるとハードウェアにする部分とソフトウェアにする部分の組み合わせが飛躍的に増えてしまい分割の決定に時間がかかることと、目的の達成にはどの項目を重視すれば良いのか判別できないといけないということが挙げられる。このハード/ソフト協調設計のハードウェアとソフトウェアの分割の手段として、本研究ではハード/ソフト分割手法を目的とする。これはハードウェアとソフトウェアの分割のために懸案項目を定量化して最適な設計空間を探索するものである。

本研究では実装や検証に FPGA やソフト・マクロ CPU を使用している。これらは ASIC などの方法と比較すると、非常に簡単に設計や検証をできるようにする。次に、それら利用した機器などについて説明する。

FPGA(Field Programmable Gate Array)とは内容（内部ロジック）を書き換えられる LSI である。基本素子は SRAM で、論理ブロックをアレイ状に敷き詰めてあり、ユーザの自作したシステムや回路のデータをダウンロードすることによって回路構成を構築・変更することができる。この書き換え可能な LSI を使うことで実機に近い検証を手軽に行うことができる。CPLD という EEPROM をベースとしたプログラマブル・ロジック・デバイスもあるがゲート規模が小さく、現在では FPGA にほとんどのシェアを奪われている。最近は FPGA の低価格化や大容量化も進んでおり 1000 万ゲートを越えるものも出てきていて、実際に製品に搭載されることもあるなど、その重要性は高まっている。

ソフト･マクロ CPU とは FPGA に簡単に搭載できる知的財産（IP）であり、プロセッサコアである。即ち FPGA に容易に CPU を導入できる。ソフト･マクロ CPU を利用することでシステム･オン･チップを簡単に実現できる。また従来の CPU と違い LSI 技術に左右されにくく、FPGA ボードが技術進歩により生産中止になってもスライス容量さえ満たせるなら搭載できるという非常に柔軟な面も持ち合わせている。注意点として、動作周波数などは FPGA によって異なるので留意する必要がある。



FPGA の構成素子に余裕があるならば複数の
ソフト･マクロ CPU を搭載することも可能

図 1：ソフト･マクロ CPU の簡易構成

本研究においての FPGA とソフト･マクロ CPU の利用の意義について述べると、ハード/ソフト協調設計におけるソフトウェア部分の設計を簡単に実現できることがある。同じ FPGA 上に CPU と自作したシステムを載せることで、協調動作させるための通信などが不要になり、容易に協調設計の検証が可能となる。さらに、FPGA での検証を行うことで、ハードとソフトの分割パターンを多く細かく見ていくことができる。また、ソフト･マクロ CPU は通常の CPU と遜色が無いので、作成したモジュールの制御などを容易に受け持てる。それによって、システムの起動やバスの制御など、通常では非常に煩雑な事柄を省略できた。

2. ハード/ソフト分割探索

2.1 分割探索フロー

ハード/ソフト協調設計において、ハードウェアとソフトウェアの最適な設計空間を探索するのは重要であると同時に非常に難しい問題である。そこで本研究での分割探索の流れを図 2 にまとめる。

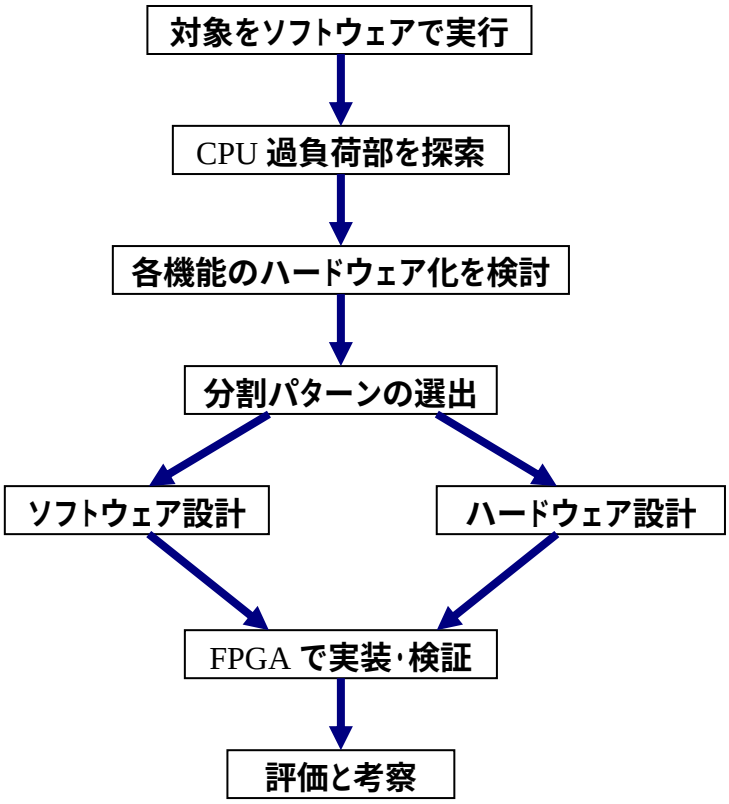


図 2：ハード/ソフト分割探索フロー

図 2 の補説として、まず対象アプリケーションを C 言語で記述してアルゴリズムの理解や正しい結果の確認などを行う。次に FPGA 上に構築した MicroBlaze でのソフトウェア実行において CPU 負荷を計測し、どのモジュールやブロックが過負荷をかけているかを把握する。CPU 負荷の計測には自作したクロックカウンタを使用する。その後、各モジュールのハードウェア化について検討してから、性能や規模について考慮した上で分割パターンを選び出す。次の段階は設計で、ハードウェアとソフトウェアの双方を平行して行う。FPGA ボードに実装して検証を行った後、計測結果から評価と考察をする。

2.2 実装環境

実装環境は Memec 社の提供する MicroBlaze 評価ボードを使用する。搭載している FPGA は Xilinx 社の Spartan II E の 60 万システムゲートを集積したものである。

このボードにユーザ・ペリフェラルを MicroBlaze とともにコンフィグレーションする。本研究で用いるソフト・マクロ CPU は Xilinx 社が提供する MicroBlaze である。MicroBlaze は 32 ビット RISC・CPU で、命令などは MIPS に準拠している。ハーバードスタイルアーキテクチャでパイプラインも使用している。最高動作周波数は 150MHz（本研究で使用する評価ボードでは最高動作周波数は 50MHz）。バスには IBM の CoreConnect バスアーキテクチャを用いていて、オンチップ・ペリフェラル・バス（OPB）を介してユーザ自作のペリフェラルと接続することが可能である。

AES 暗号処理システムと JPEG エンコーダでは扱うデータ量などの点からペリフェラルの接続の仕方などが異なっている。実装方法などに関して、本論文では著者である私が設計に深く関わっている AES 暗号処理システムについては 3 章で詳細に記述するが、JPEG エンコーダについては共同研究者である古川氏と的場氏の論文を参照してほしい[13][15]。4 章では JPEG エンコーダの概要と実験結果、考察について述べる。

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 とは” Advanced Encryption Standard”の略で、米国商務省国立標準技術局(NIST)によって選定作業が行われた米国政府の次世代標準暗号化方式である[1]。これは標準暗号として用いられている DES が、近年のコンピュータの高性能化、暗号理論の発展に伴いその信頼性を低下させたので、より強度の高い新しい暗号として利用する共通鍵暗号方式である。2000 年 10 月に、ベルギーの暗号開発者 Joan Daemen と Vincent Rijmen が開発した「Rijndael」という方式が選ばれた。

Rijndael アルゴリズムは入力データを 4 つの変換方式を用いて、一定回数のデータ変換を経ることで暗号化を行う。鍵長は 128,192,256 ビットの 3 種類。暗号化に使う鍵も拡張を行うことで暗号の強度を上げている。鍵長によってデータのブロックサイズや暗号化のループ回数が決定される。

データ変換には AddRoundKey 変換、SubBytes 変換、ShiftRows 変換、MixColumns 変換の 4 種類がある。さらに鍵拡張部（KeyExpansion）を加えることで大きく 5 つの機能ブロックに分けられる。各機能ブロックの詳細については 3.2 項で述べる。

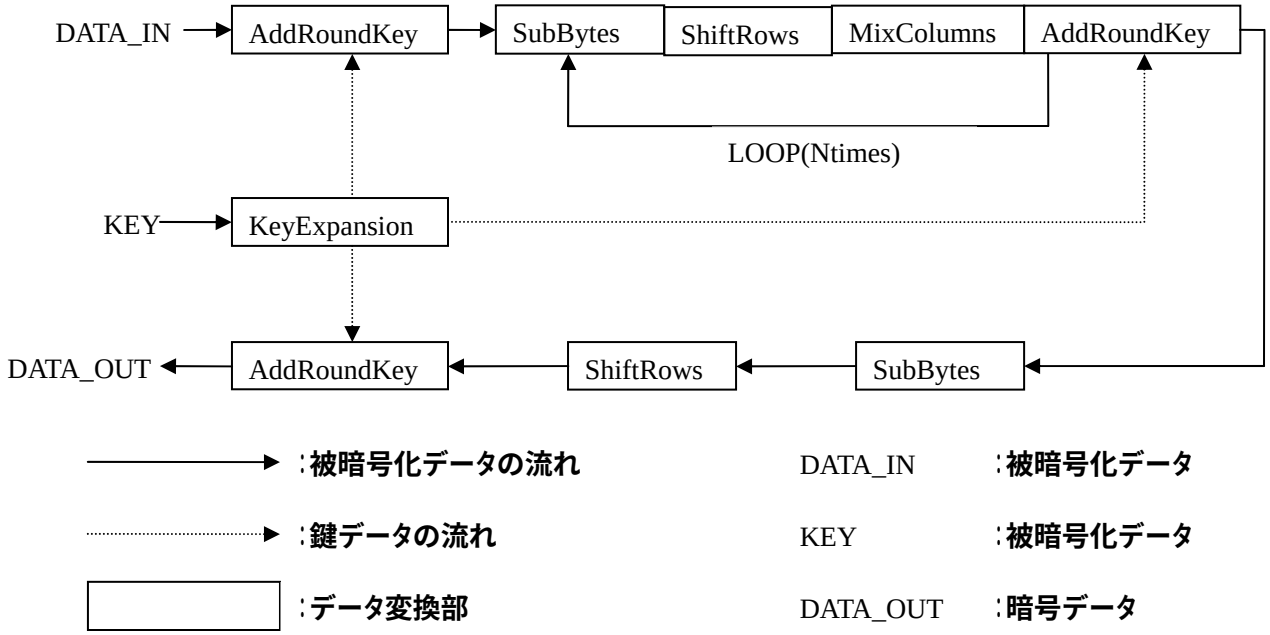


図 3：AES Rijndael アルゴリズムの暗号化フロー

本論では図 3 におけるデータ変換部がそのまま1つのモジュールとして扱われる。

Rijndael アルゴリズムのデータの扱い方は 1 バイトを 1 ブロックとして 4 つのブロックを繋げ

て 1 列としている。データの並びとしては(0,0) →(1,0)→(2,0)→…→(2,3)→(3,3)が通常の順序となる。変換処理をする場合には、大概の機能ブロックでは 1 列単位で処理を行う。

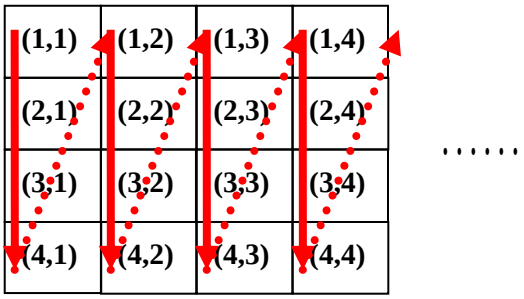


図 4：データの並び

以降、本研究では AES 暗号については 128 ビット鍵長に限定して話を進めていく。

3.2 各モジュールの説明

3.2.1 KeyExpansion

鍵を拡張する KeyExpansion は処理の手順が一番煩雑である。本研究では鍵長を 128 ビットに限定しているので、それを中心に話を進める。

まず鍵長によって拡張する際に使用するパラメータが異なる。表 1 に使用するパラメータを記載する。

表 1：KeyExpansion のパラメータ

Key Length	Key Length Nk Word	Number of Rounds Nr
AES 128bit	4	10
AES 192bit	6	12
AES 256bit	8	14

鍵データの扱い方は 1 バイトを 1 ブロックとして列単位で見て、その列を Wc(“c”は列の順位)とする。ラウンドごとに鍵を 128 ビット拡張するので、128 ビットごとに鍵を RoundKey として分ける。

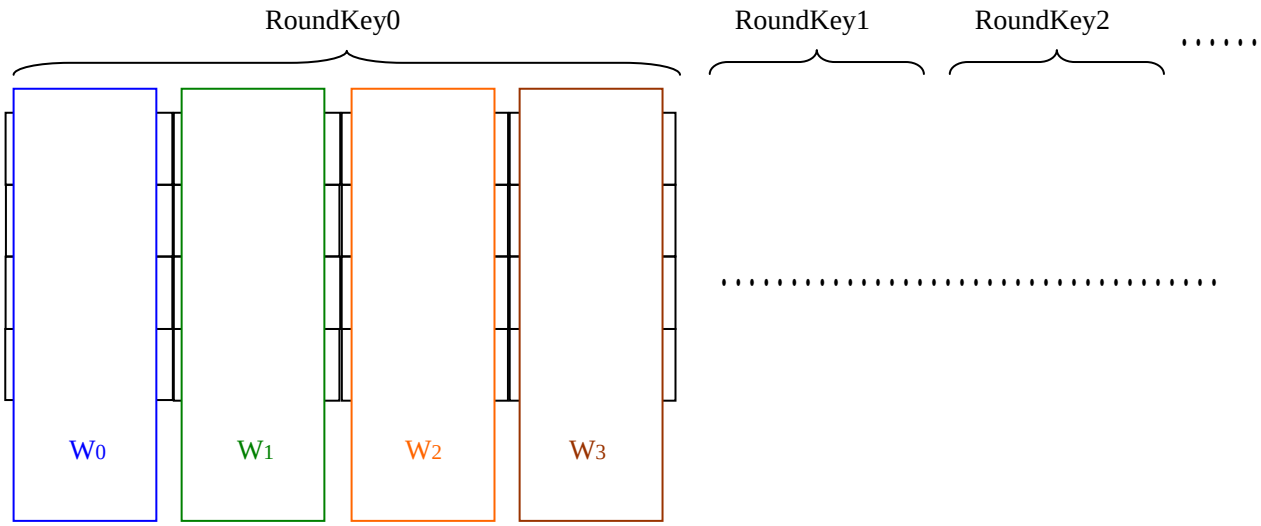


図 5:鍵データの扱い方

次に鍵拡張の方法について述べる。

拡張された次の鍵、RoundKey（この場合、RoundKey は 128 ビットを 1 つと見ている）を生成するには、鍵の順位"c"によって方法が異なるので場合を分けて考える。

(1) c=N*Nk (N は正整数) の場合・・・

Wc を求めるには Wc-Nk と Wc-1 が必要になる。工程は次の通り。

① W c-1 の上位 1 バイト（1 ブロック）を最下位へとシフトさせる。

② W c-1 の各ブロックごとに Sbox というテーブルを参照して各ブロックの値を変換。各 Block において、値を上位 4bit と下位 4bit に分解し、Sbox[上位 4bit ， 下位 4bit]としてテーブルを参照する。

③ Round から Rcon というテーブルを参照して、その値と最上位ブロックとの排他的論理和を計算する。

④ Wc-Nk との排他的論理和をとる。

c=4 の場合……

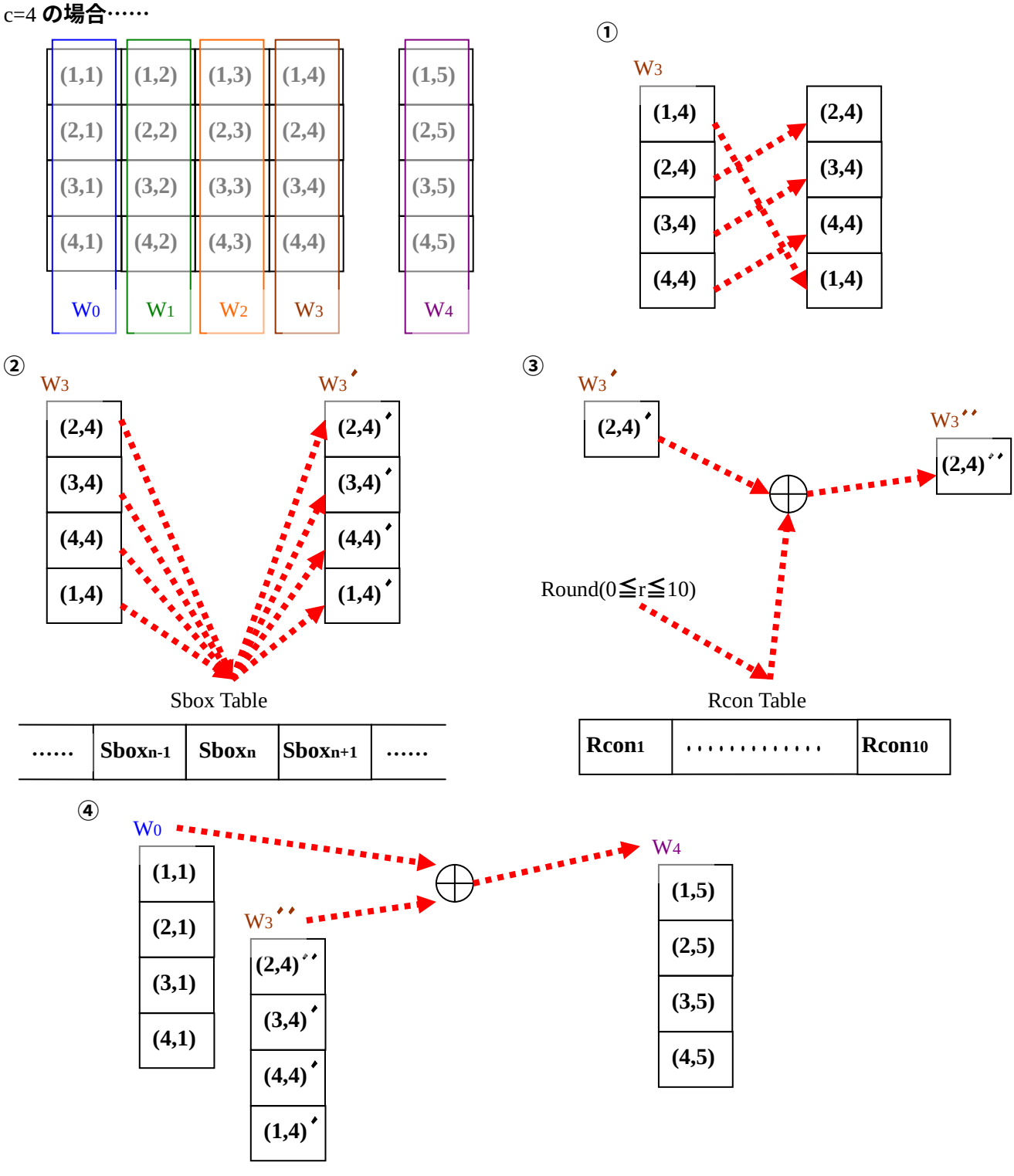


図 6：c=N*Nk の時の KeyExpansion 処理フロー

(2) c≠N*Nk (N は正整数) の場合・・・

Wc を求めるには W c-1 と Wc-Nk の排他的論理和を計算するだけでよい。

c=5,6,7 の場合……

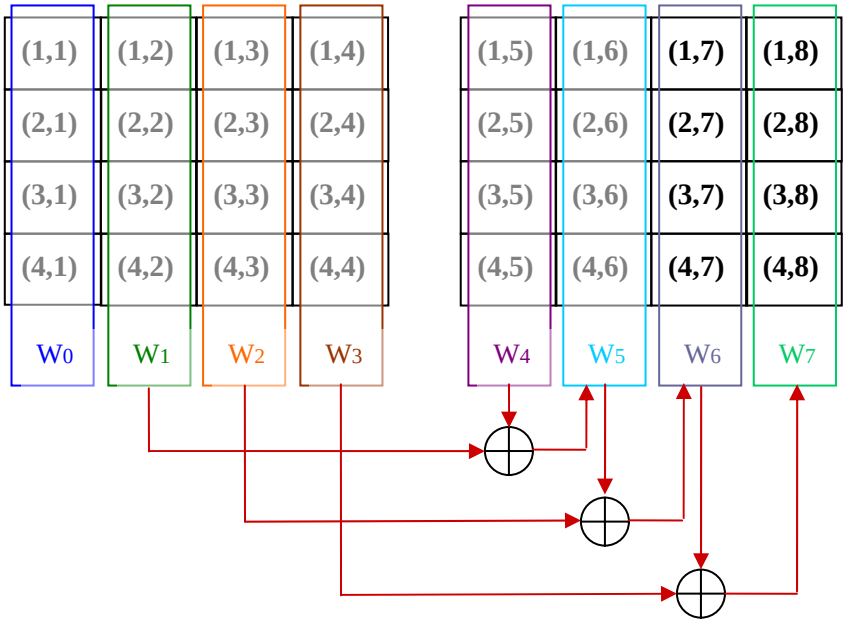


図 7：c≠N*Nk の時の KeyExpansion 処理フロー

3.2.2 AddRoundKey

AddRoundKey 変換は入力されてきた被変換データと鍵データを排他的論理和するだけである。処理はブロック単位で行う。

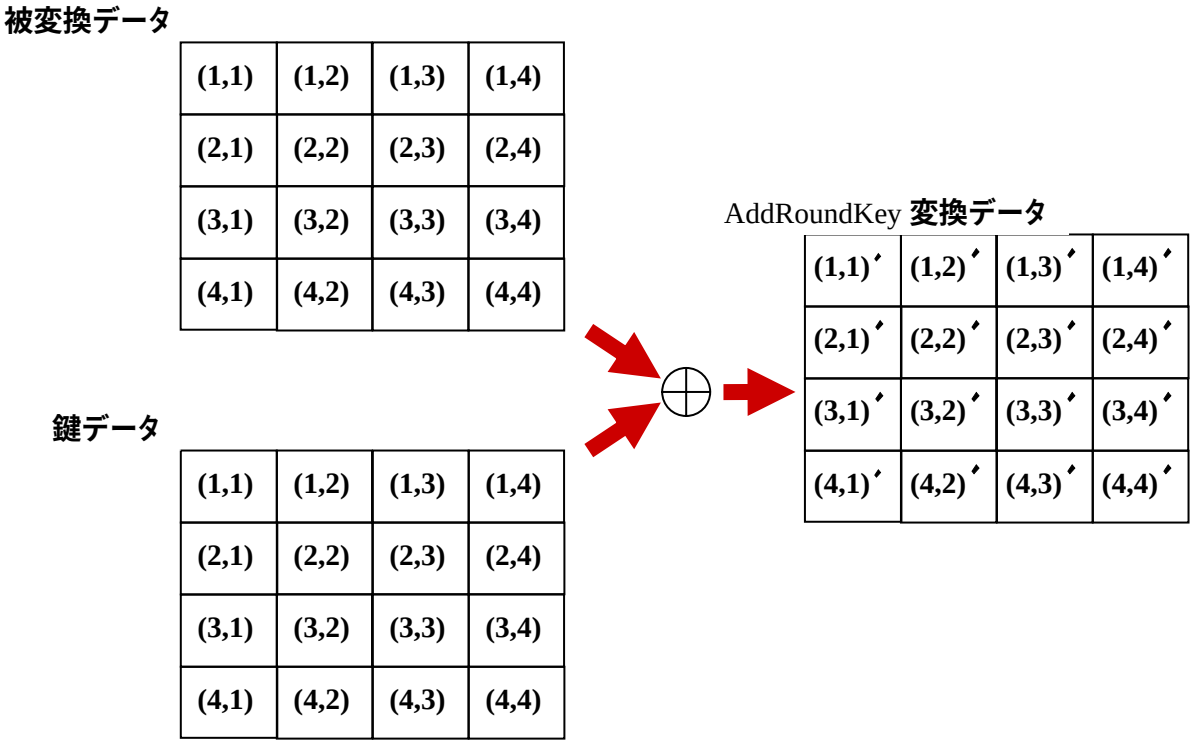


図 8：AddRoundKey 変換

3.2.3 SubBytes

SubBytes 変換は入力されてきた被変換データを Sbox というテーブルを参照して変換するものである。この処理は列単位で行う。

Sbox テーブル参照は KeyExpansion でも述べたように、各ブロックにおいて、値を上位 4 ビットと下位 4 ビットに分解し、Sbox[上位 4 ビット，下位 4 ビット]としてテーブルを参照する。

(3,2)を参照する場合・・・

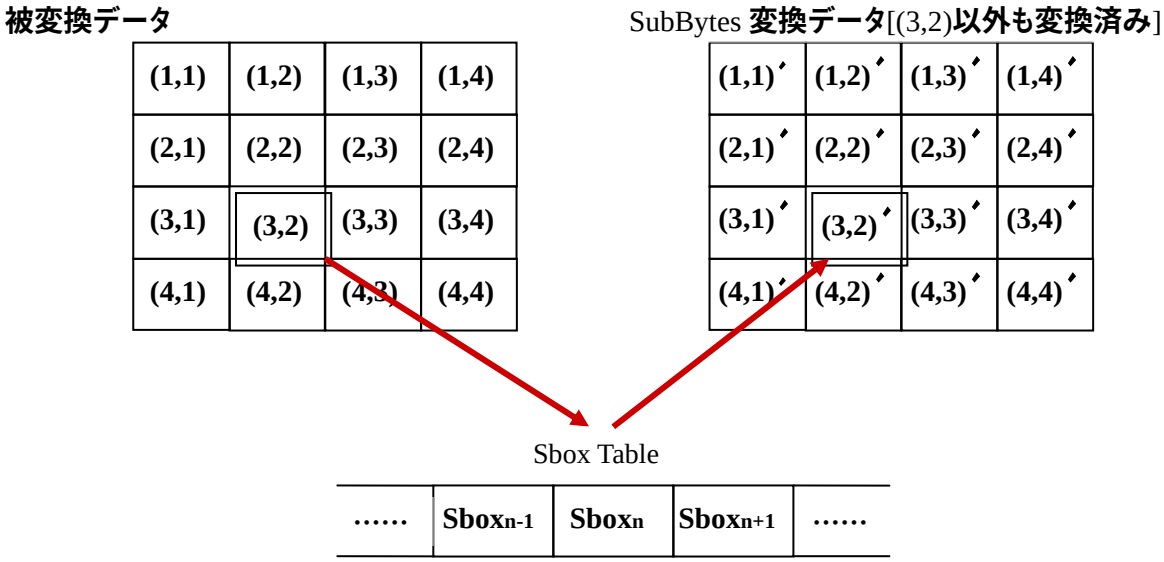


図 9：SubBytes 変換

表 2：Sbox テーブル

	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	Xa	Xb	Xc	Xd	Xe	Xf
0X	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1X	ca	82	c9	7d	fa	59	47	f0	ab	d5	a2	af	9c	a4	72	c0
2X	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3X	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4X	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5X	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6X	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7X	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8X	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9X	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
aX	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bX	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cX	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dX	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
eX	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fX	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

※表中の数値は全て 16 進数、'X'は交差する縦もしくは横の数値

3.2.4 ShiftRows

ShiftRows 変換は入力されてきた被変換データを行ごとにシフトする。注意する点は今まで列ごとにデータを扱ってきたが ShiftRows 変換は行単位でデータを扱うことと、行によってシフトする量が異なることである。シフト量は n 行目では(n-1)ブロック（バイト）分だけ。シフト

方向は左である。

図 10：ShiftRows 変換

3.2.5 MixColumns

MixColumns 変換は入力されてきた被変換データをガロアフィールド理論に基づく演算によって変換する。処理単位は列である。

ここではガロアフィールド理論という難解な数学理論を用いることになるが、MixColumns 変換を使う場合には注意する点は絞られる。まず、MixColumns 変換は次の行列式(1)を計算することで結果を導ける。式内の右辺にある列ベクトルは被変換データで左辺にある列ベクトルは変換後データの列（4 ブロック）、行列の数値は 16 進数である。

$$\begin{bmatrix}(1,i)'\\(2,i)'\\(3,i)'\\(4,i)'\end{bmatrix}=\begin{bmatrix}02&03&01&01\\01&02&03&01\\01&01&02&03\\03&01&01&02\end{bmatrix}\begin{bmatrix}(1,i)\\(2,i)\\(3,i)\\(4,i)\end{bmatrix}\quad\cdots(1)$$

上記の行列計算を行えば一列分の変換データを導出できるが、そのための積和演算がガロアフィールド理論に準じているので注意する必要がある。ガロアフィールド理論での積和演算は以下の項目についてのみ考慮すればよい。

- フィールドでの加算は排他的論理和である。
- 乗算は (1) 0x01 との乗算は被乗数の値のままである。
  (2) 0x02 との乗算は、被乗数が 0x08 未満なら被乗数を 1bit 左シフトした値で、被乗数が 0x08 以上なら 1bit 左シフトした値に 0x1b を EXOR 演算した値。
  (3) 0x03 との乗算は、(被乗数×0x01+被乗数×0x02)に分配できる。

このガロアフィールド理論で注目するのは乗算部分での、0x01 と 0x02 の 2 通りの処理パターンがあることに注意する。この 2 通りの処理はモジュールでは場合分けに相当する。こ

の乗算を行った後に総和を求める。

(1,*i*)´について・・・

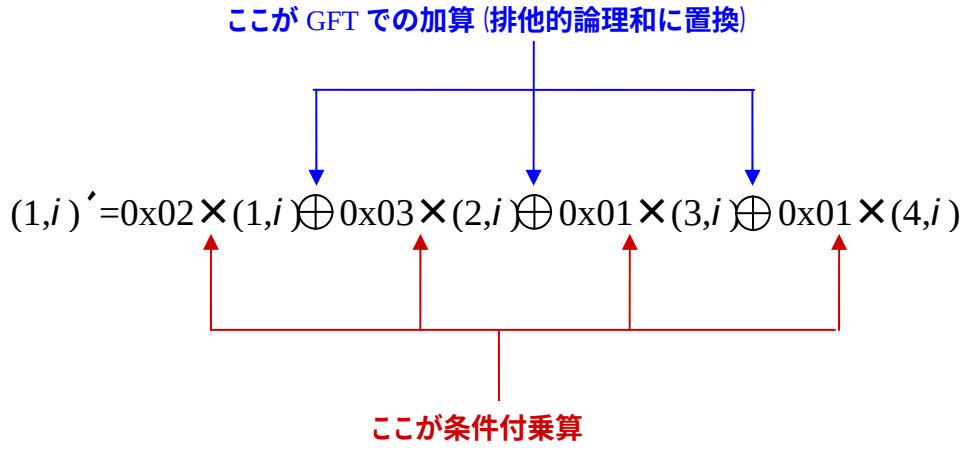


図 11：MixColumns 変換

3.2.6 その他ーAES_Loop, AES_fullHW, AES_ss

AES_Loop、AES_fullHW は AES 暗号システムを全てハードウェア化する際に使用する。厳密にはアルゴリズムではないがここで言及しておく。

AES_fullHW は所謂トップモジュールである。ここではデータ入力直後にある AddRoundKey 変換と次にある AES_Loop 部分を接続・管理する役割を担っている。

AES_Loop は AES 暗号化の SubBytes→ShiftRows→MixColumns→AddRoundKey のループ部分を制御するモジュールに相当する。そのための制御部も含んでいる。また、ループ脱出後にある SubBytes→ShiftRows→AddRoundKey という変換も兼ねている。

AES_ss は上記 2 つのモジュールとは別のものであり、128 ビットの暗号化を最小でシンプルな回路として実現できるかを目的としたシステムである。AES_ss の”ss”は”simplest system”を表す。

これらのモジュールについての詳細仕様は次節において述べる。

3.3 各モジュールのハードウェア実現方法

3.3.1 KeyExpansion

KeyExpansion モジュールは組み合わせ回路として設計している。入力（被変換データ）、出力（変換済みデータ）共にデータ幅は 128 ビットで、一度に 1 ラウンドを処理する。10 ラウンドあるのでこれを 9 回ループさせてやる必要がある。

実装方法に MicroBlaze に繋げるものと、AES 暗号システムの全体をハードウェア化するものとでは少し違いがある。MicroBlaze に接続する方法では鍵拡張を最初に 10 ラウンド分全て実行してしまうのでメモリに 10 ラウンド分の鍵を格納する。一方、全体をハードウェア化する方法では 1 ラウンド拡張を行ってそれを AddRoundKey へと出力しながらレジスタへ記録する。これを次々に繰り返すことで 10 ラウンドの拡張をする。

図 12 の"keyex"は 128 ビットの KeyExpansion 変換を行うモジュールである。

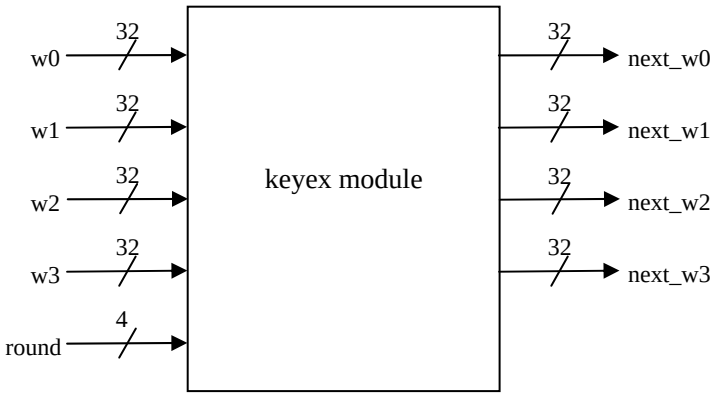


図 12：KeyExpansion 変換モジュール

表 3：keyex の信号線

信号名	方向	幅(bit)	詳細
w0,w1,w2,w3	input	32×4	拡張される鍵データ（第 1，第 2，第 3，第 4）
next_w0, next_w1, next_w2, next_w3	output	32×4	拡張後の鍵データ（第 1，第 2，第 3，第 4）
round	input	4	ラウンド数，鍵拡張の際に使用

3.3.2 AddRoundKey

AddRoundKey モジュールは簡単に内部を表すと、単純な XOR 回路である。入力は被変換データと鍵データ、出力は変換後データである。図 13 の"ARK_128b"は 128 ビットの AddRoundKey 変換を行うモジュールである。

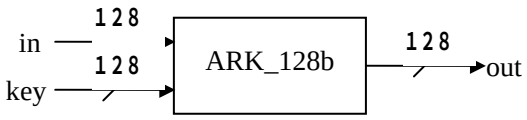


図 13：AddRoundKey 変換モジュール

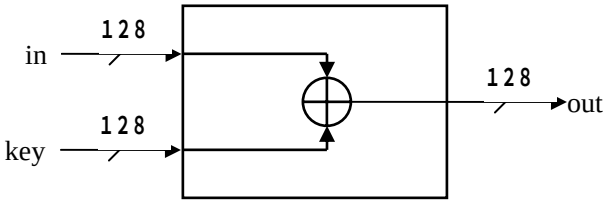


図 14：ARK_128b モジュールの内部構成

表 4：ARK_128b の信号線

信号名	方向	幅(bit)	詳細
out	output	128	AddRoundKey 変換後のデータ
in	input	128	入力データ
key	input	128	入力データと XOR する鍵データ

3.3.3 SubBytes

SubBytes モジュールは入力データをブロックごとにテーブル参照を行って値を変換する。入力値を元に出力値を特定するので ROM で構成できる。図 15 の”SBX_128b”は 128 ビットの SubBytes 変換を行うモジュールである。

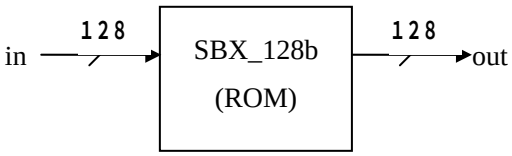


図 15：SubBytes 変換モジュール

表 5：SBX_128b の信号線

信号名	方向	幅(bit)	詳細
out	output	128	SubBytes 変換後のデータ
in	in	128	入力データ

3.3.4 ShiftRows

ShiftRows モジュールは行ごとにシフト量を変えるだけの回路である。結果として内部配線をずらして、入力を出力に直結するだけで事足りる。図 16 の"SR_128b"は 128 ビットの ShiftRows 変換を行うモジュールである。

入力を出力に直結しているので内部は配線のみ、従って回路規模は 0 である。

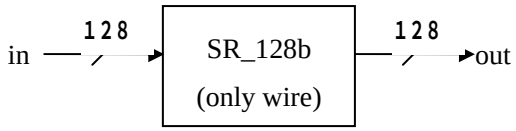


図 16：ShiftRows 変換モジュール

表 6：SR_128b の信号線

信号名	方向	幅(bit)	詳細
out	output	128	ShiftRows 変換後のデータ
in	in	128	入力データ

3.3.5 MixColumns

MixColumns モジュールは前節で述べたようにガロアフィールド理論に基づいた演算を行う。入力は被変換データ 128 ビット、出力も 128 ビットである。図 17 の"MC_128b"は 128 ビットの MixColumns 変換を行うモジュールである。

入力を列に分割した後、各ブロックでガロアフィールド理論での乗算に必要な場合分けをする。

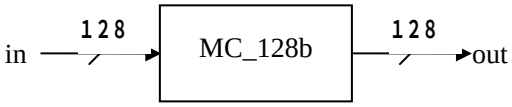


図 17：MixColumns 変換モジュール

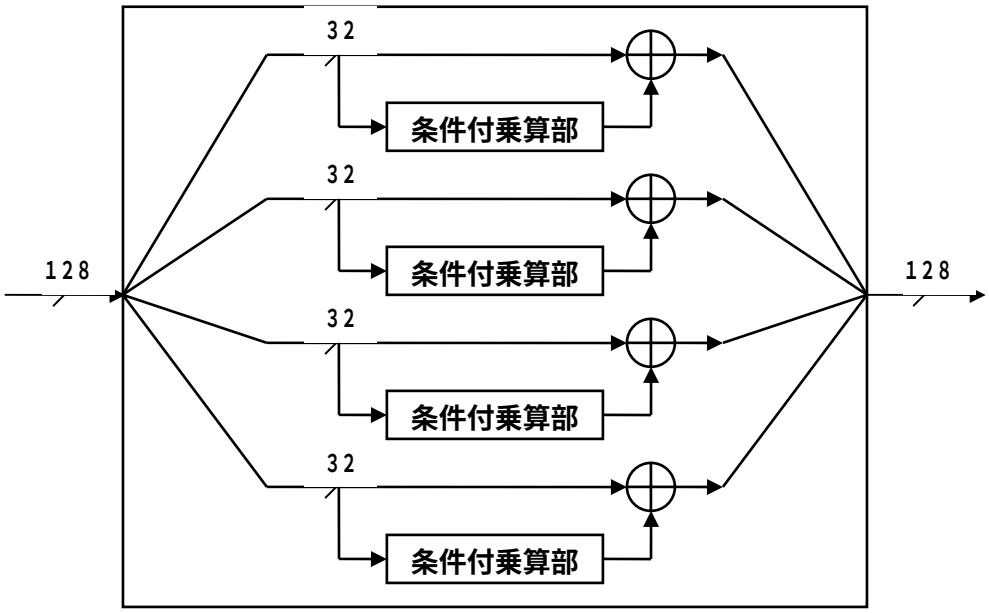


図 18：MC_128b モジュールの内部構成

表 7：MC_128b の信号線

信号名	方向	幅(bit)	詳細
out	output	128	MixColumns 変換後のデータ
in	in	128	入力データ

3.3.6 その他ーAES_Loop, AES_fullHW, AES_ss

まず、関連する AES_Loop と AES_fullHW について説明する。前節で述べたように AES_Loop は AES 暗号システムのループ部分、AES_fullHW はトップモジュールに相当する。

最初に AES_fullHW を実現するのに AES_Loop を用いて暗号化フローの通りにモジュールをつなげた場合について記載する（図 19 と図 20）。これはループ部分に KeyExpansion モジュールを内蔵して、さらに制御器をつけてループをコントロールしている。

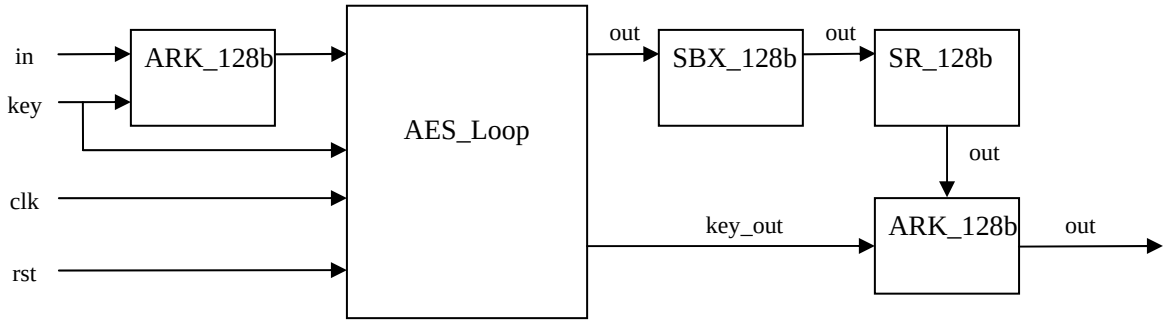


図 19：AES_fullHW モジュールの内部構成 (暫定)

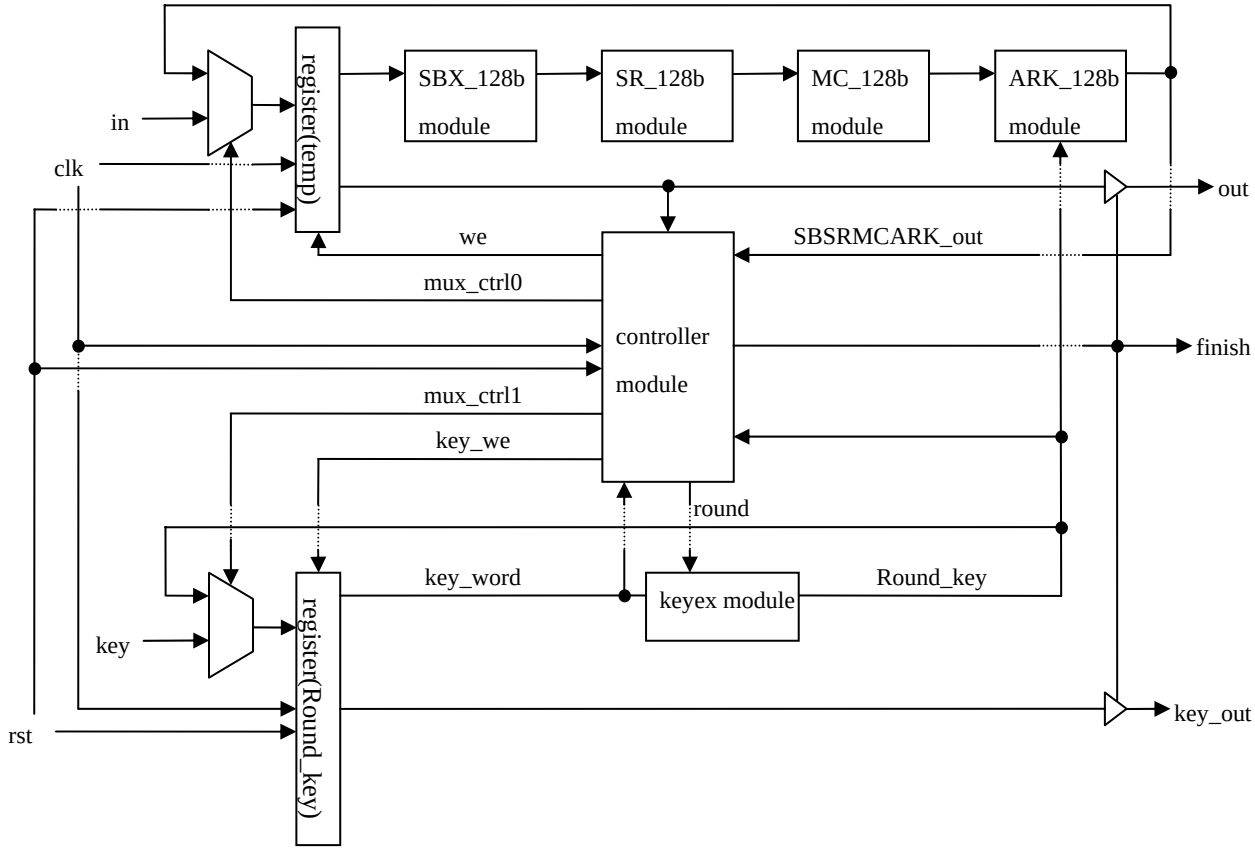


図 20 ：AES_Loop モジュールの内部構成 (暫定)

安易に考えると上記の図のようになるが、AES_Loop の外部と内部でモジュールが重複している。回路規模の削減を考えると冗長なので、AES_Loop 内部のモジュールも利用するほうが望ましい。そこで次に、図 19 の AES_Loop より右のモジュールを AES_Loop に組み込んだ形の構成を挙げる。図 22 の赤い線が主に追加された部分である。

図 22 の"controller module"から追加されたマルチプレクサに出力されている信号は

finish 信号と同じである。

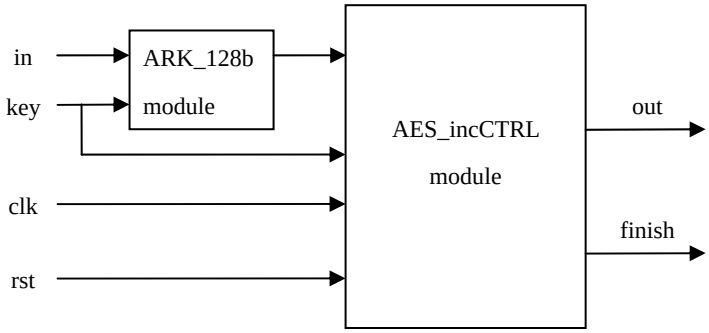


図 21：AES_fullHW モジュールの内部構成

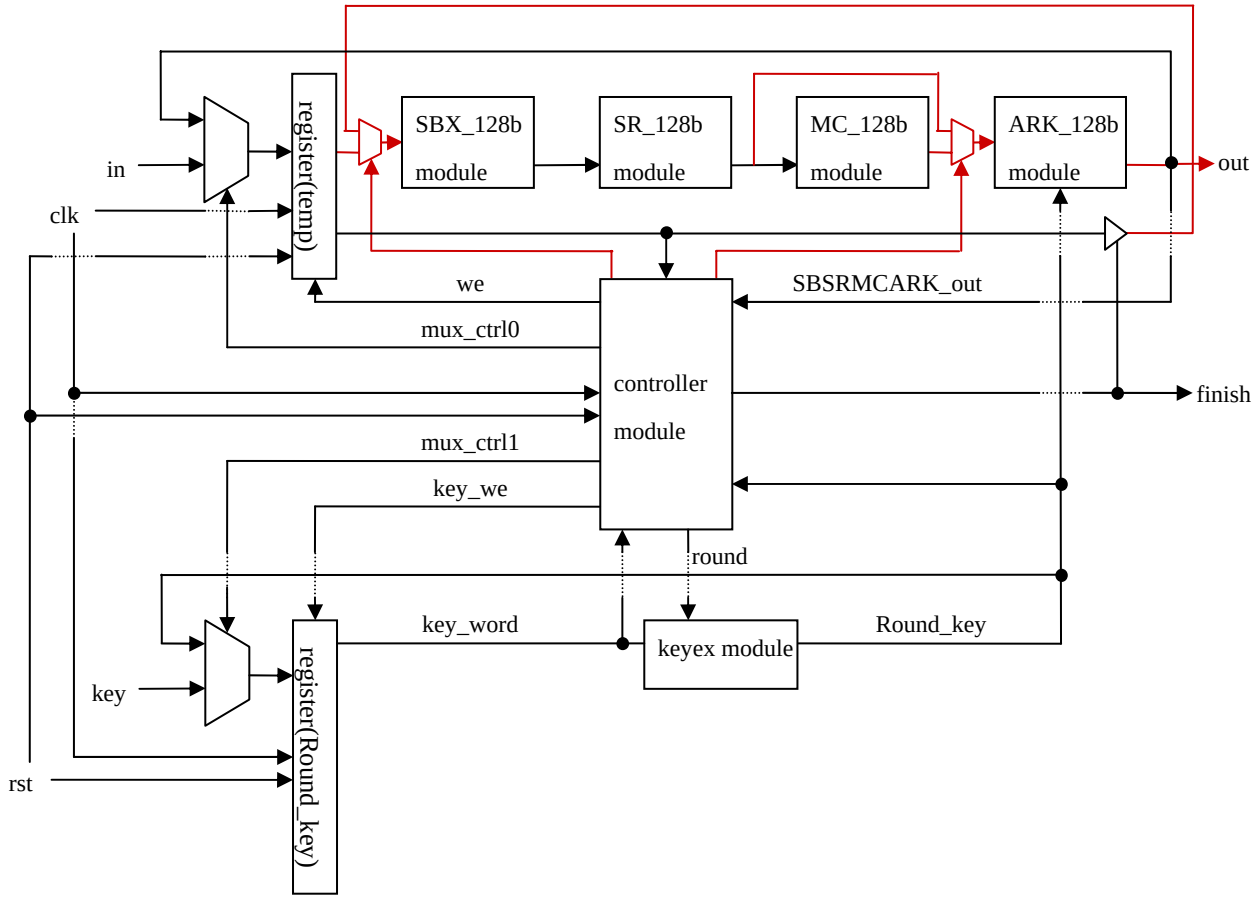


図 22：AES_Loop モジュールの内部構成

表 8: 信号線の詳細

	信号名	方向	幅(bit)	詳細
AES_fullHW	out	output	128	暗号化の最終出力
	in	input	128	暗号化する元データ
	key	input	128	暗号化に使用する鍵, AES_incCTRL 以下では拡張されて使用
	clk	input	1	クロック
	rst	input	1	リセット, AES_incCTRL 以下でレジスタへの書き込みに必要
AES_incCTRL	out	output	128	暗号化の最終出力
	key_out	output	128	暗号化する元データ
	finish	output	1	ループ変換の終了信号
	in	input	128	暗号化するデータ
	key	input	128	暗号化に使用する鍵，拡張される
	clk	input	1	クロック
	rst	input	1	リセット信号，スタート信号も兼ねており 1 のとき処理開始

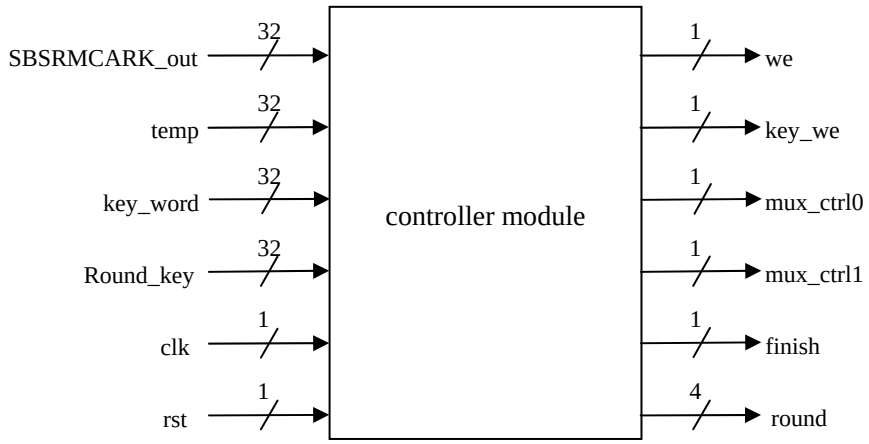


図 23：controller モジュールの内部構成

表 9：controller モジュールの内部構成

信号名	詳細
SBSRMCARK_out	変換後の値
temp	データのレジスタ値，変換した値がレジスタに入力されたかを確認
key_word	拡張後の鍵
Round_key	鍵のレジスタ値，変換した値がレジスタに入力されたかを確認
clk	クロック
rst	リセット
we	レジスタ temp へのライトイネーブル（0:not-writable, 1:writable）
key_we	レジスタ Round_key へのライトイネーブル（0:not-writable, 1:writable）
mux_ctrl0	マルチプレクサの操作信号（0:”in” effective, 1:”SBSRMCARL_out” effective）
mux_ctrl1	マルチプレクサの操作信号（0:”key” effective, 1:”Round_key” effective）
finish	ループ変換の終了信号
round	ラウンド数，初期値は 1

次に AES_ss について説明する。このモジュールはできるだけ簡単にシステムまとめるというコンセプトで設計されている。レジスタを用いてループ部分だけでなく、ループの前と後の変換も行う。機能モジュールはそれぞれ 1 ブロックずつしか使用せずに制御器もほぼカウンタのみとなっている。構成については図 24 に示す。

非常に簡便で分かりやすい構成になっているが、反面モジュールの扱いが厳しくなる上に入出力の安全性には考慮がなされていない。AES_fullHW が 10 クロックで全ての 128 ビット暗号化を行えるのに対して、AES_ss では 12 クロックかかってしまう。さらに 12 クロックが経過しなくても値を取り出せるので、この暗号システムを取り扱うユーザなりモジュールなりが値の正当性を保証する必要がある。

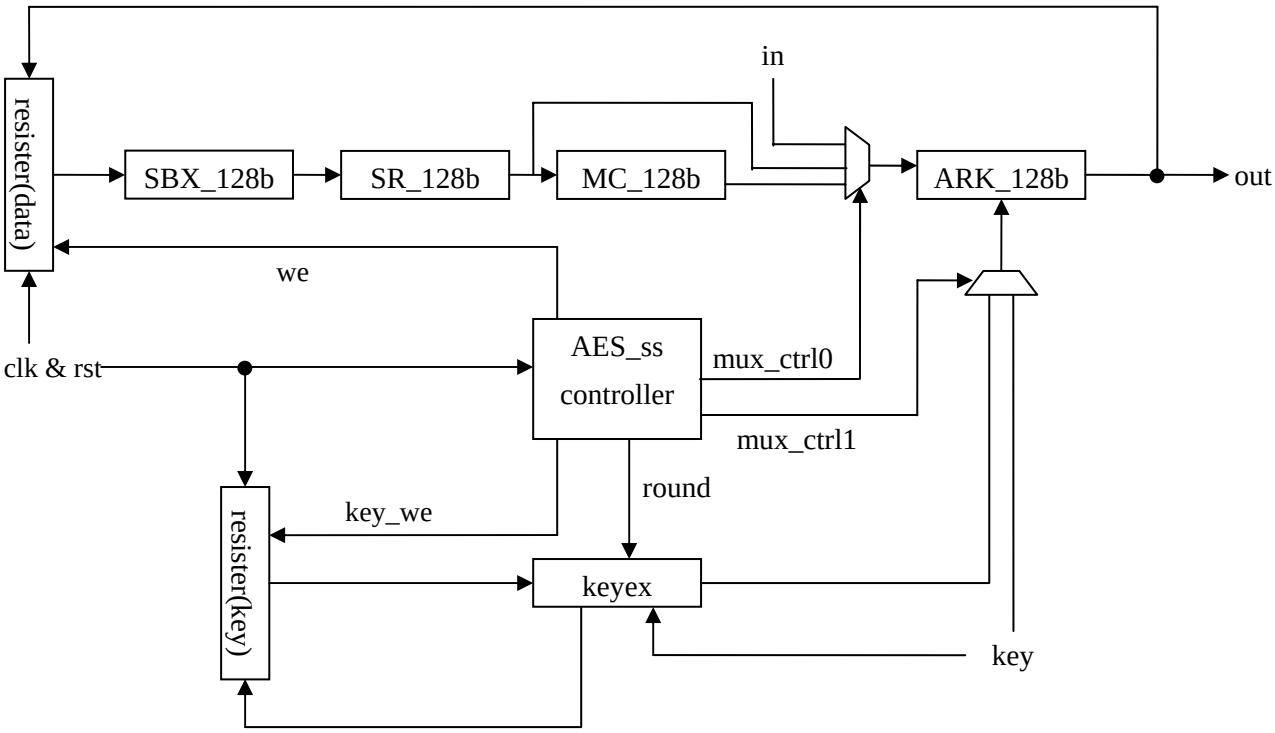


図 24：AES_ss の構成

3.4 ハードとソフトの分割パターン

ハードウェアとソフトウェアの分割探索のために、まずソフトウェアでシステムを実現して、そのモジュールごとの CPU 負荷を計測した。結果は図 25 の通りである。なお、結果に KeyExpansion 変換がないのは、KeyExpansion 変換は一定の処理であり、暗号化データ量が大きくなれば 0 に近似できるからである。

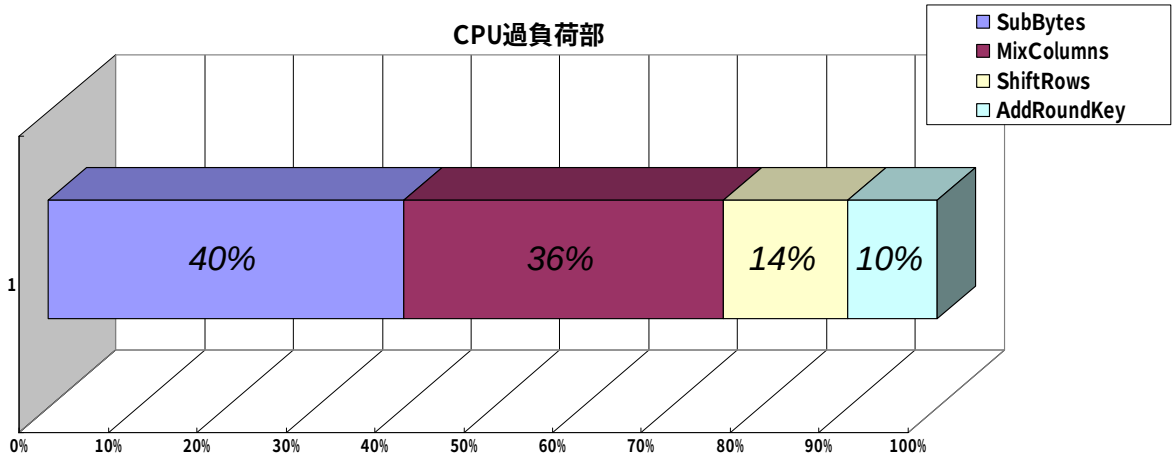


図 25：AES 暗号システムの各モジュールの CPU 負荷

SubBytes 変換と MixColumns 変換がほとんどの割合を占めている。これは SubBytes 変換がテーブル参照、MixColumns 変換が場合分けなどの複雑な演算を行っているためと考えられる。

この結果を参考にしてハードとソフトの切り分けを考える。分割の粒度としては変換モジュールごとに考慮する「機能ブロック単位」と、暗号化の流れを考慮する「連続ブロック単位」がある。機能ブロック単位で分割する利点は、ハードとソフトに分割するに際して柔軟に対処できることである。反面、分割できる範囲が広がるので最適化の探索や、システムの制御が厳しくなる。連続ブロック単位では演算や回路の最適化が行えるが、柔軟性は失われる。

それぞれの分割パターンについては表 10 と表 11 に詳しく記載する。

表 11 の"へ"は AES_fullHW で実現したときの構成で、"ト"は AES_ss で実現したときの構成である。表に違いは認められないが、実験結果は異なる。

表 10：機能ブロックによる分割パターン

分類	ハードウェア処理部	ソフトウェア処理部
S		SubBytes, MixColumns, ShiftRows, AddRoundKey, KeyExpansion, 制御
A	MixColumns	SubBytes, KeyExpansion, ShiftRows, AddRoundKey, 制御
B	SubBytes	MixColumns, AddRoundKey, ShiftRows, KeyExpansion, 制御
C	SubBytes, MixColumns	ShiftRows, AddRoundKey, KeyExpansion, 制御
D	SubBytes, MixColumns, ShiftRows	AddRoundKey, KeyExpansion, 制御
E	SubBytes, MixColumns, AddRoundKey	ShiftRows, KeyExpansion, 制御
F	SubBytes, MixColumns, ShiftRows, AddRoundKey	KeyExpansion, 制御
G	SubBytes, MixColumns, ShiftRows, AddRoundKey, KeyExpansion	制御

表 11：連続ブロックによる分割パターン

分類	ハードウェア処理部	ソフトウェア処理部
イ	SubBytes-ShiftRows	MixColumns, AddRoundKey, KeyExpansion, 制御
ロ	SubBytes-ShiftRows -MixColumns	AddRoundKey, KeyExpansion, 制御
ハ	SubBytes-ShiftRows –MixColumns-AddRoundKey	KeyExpansion, 制御
ニ	SubBytes-ShiftRows, AddRoundKey-KeyExpansion	MixColumns, 制御
ホ	SubBytes-ShiftRows-MixColumns, AddRoundKey-KeyExpansion	制御
ヘ	SubBytes-ShiftRows-MixColumns-AddRoundKey-KeyExpansion-制御	
ト	SubBytes-ShiftRows-MixColumns-AddRoundKey-KeyExpansion-制御	

3.5 実験と考察

実験に使用したソフト･マクロ CPU と AES 暗号モジュールを実装した FPGA ボードの簡単な構成を図 26 に示す。

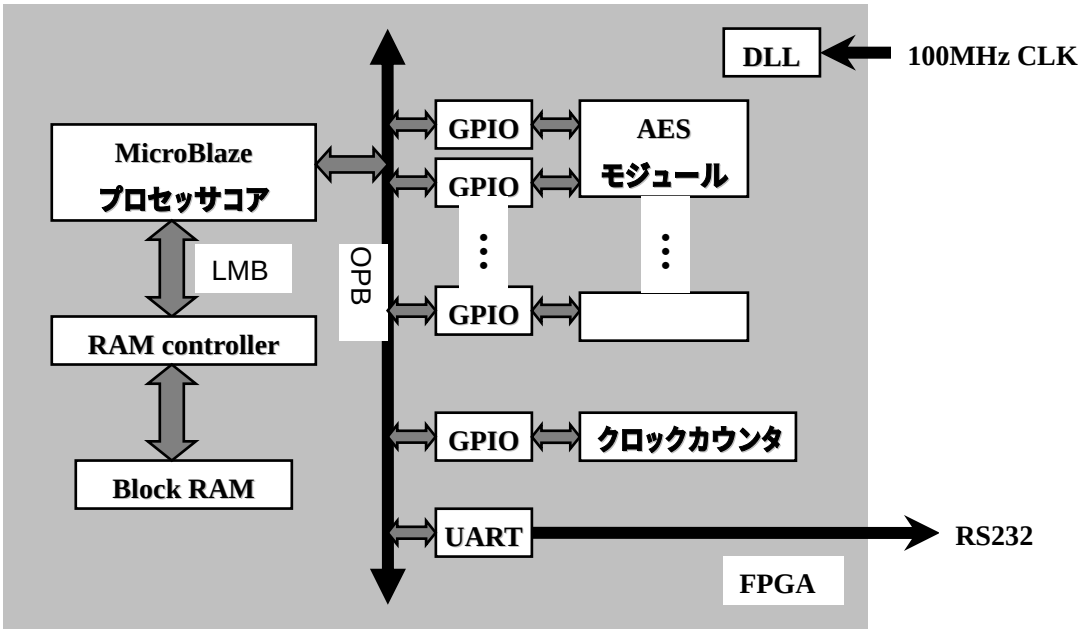
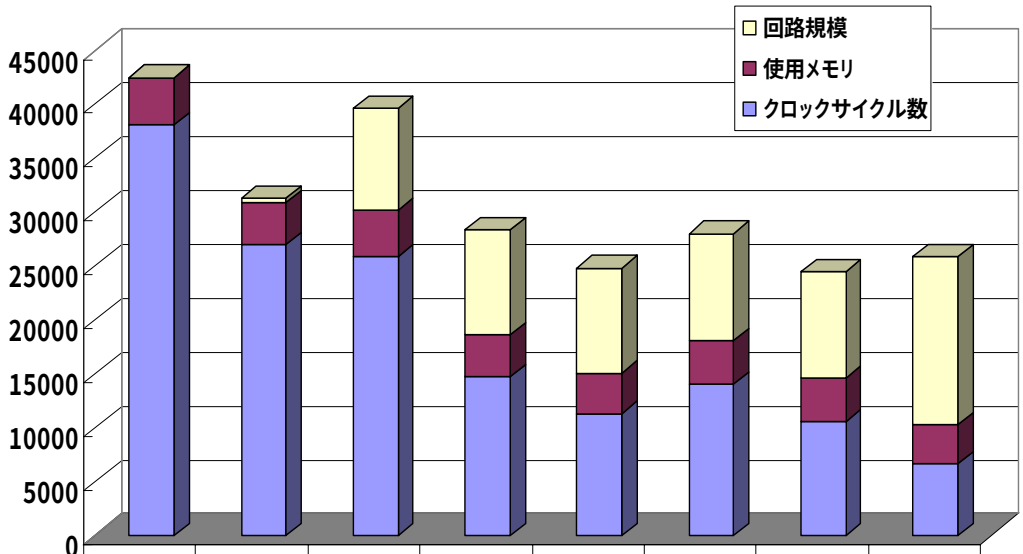


図 26：AES 暗号システムの簡易構成図

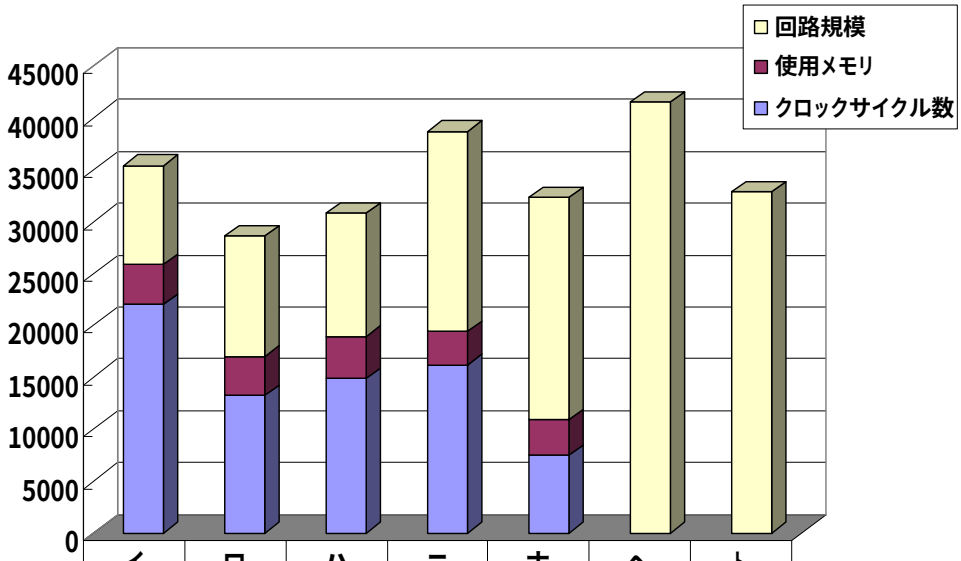
ソフト/マクロ CPU である MicroBlaze がソフトウェア部分を担い、MicroBlaze とユーザ・ペリフェラルの通信にはオンチップ・ペリフェラル・バス（OPB）を介して GPIO という IP を使用する。GPIO は基本的に最大 32 ビット幅の I/O である。AES 暗号モジュールでは、これを複数繋げて協調動作を実現する。また、クロックサイクル数を計るためにクロックカウンタも GPIO で接続する。クロックカウンタは開始信号を受け取るとクロック数をカウントし始める。AES 暗号システムの最終結果は UART (Universal Asynchronous Receiver Transmitter) という IP を介してシリアルポートの RS232 に出力される。ローカル・メモリ・バス（図 26 の LMB）はブロック RAM を使用する際に必要となり、DLL（Delay Lock Loop）は通常 100MHz のクロック周波数を MicroBlaze がこのボードで動作できる 50MHz に分周する。

前節であげた機能ブロック単位と連続ブロック単位、それぞれでの実験結果を図 27 と図 28 のグラフに示す。



	S	A	B	C	D	E	F	G
回路規模	0	324	9441	9765	9765	9957	9957	15573
使用メモリ	4387	3935	4259	3807	3731	4047	3971	3599
クロックサイクル数	38075	26959	25859	14743	11313	13979	10549	6650

図 27：AES 暗号化システムの機能ブロックによる分割パターンの実験結果



	イ	ロ	ハ	ニ	ホ	ヘ	ト
回路規模	9552	11632	11824	19278	21358	41500	32824
使用メモリ	3832	3724	4080	3236	3472	0	0
クロックサイクル数	21996	13273	14875	16178	7455	10	12

図 28：AES 暗号化システムの連続ブロックによる分割パターンの実験結果

図 27 と図 28 に示されている数値は 128 ビットを暗号化する際に必要となるクロックサイクル数とメモリ量である。クロックサイクル数は自作のペリフェラルであるクロックカウンタを MicroBlaze システムに接続して計測した。なお、回路規模は AES 暗号化システムのみの結果であり、MicroBlaze やペリフェラルの I/O などは含まれていない。

2 つの実験結果からハード/ソフト分割の効果を考察する。

機能ブロック単位での分割を見ると、S→A や S→B に注目すると SubBytes 変換と MixColumns 変換をハードウェア化したときのクロックサイクル数の削減が非常に大きいことがわかる。メモリの使用量に注目すると MixColumns 変換が多く、次に SubBytes 変換が続く。これは MixColumns 変換が複雑な計算を行うためと、SubBytes 変換がテーブルを用いているのが要因と見られる。

連続ブロック単位で見ると、MixColumns 変換をハードウェア化したときのクロックサイクル数の激減が見て取れる。また、”ヘ”や”ト”と他を比べると制御に多大なクロックサイクルとメモリを費やしているのも確認できる。

2 つの実験結果を比べると、連続ブロックでのメリットである演算や回路の最適化の効果があまり見られない。

総括すると、連続ブロックの利点が得られないので柔軟性のある機能ブロック単位で分割したほうが最適化に有利であると思われる。AES 暗号システムの機能ブロック単位での分割の中でも強いと見られるのは、回路規模はそこそこで速度を得られる”D”や回路規模は多少大きくなるがメモリとクロックサイクルの両者を取れる”G”だろう。

4. JPEG エンコーダのハード/ソフト分割

4.1 JPEG コーデック

JPEG とは ISO の専門化組織" Joint Photographic Experts Group"の略であり、それがそのまま静止画像の圧縮方式の名称になっている。この圧縮方式は可逆圧縮と非可逆圧縮の両方を選ぶことができ、非可逆圧縮の場合は人間の目には見えづらい高周波成分をカットして 10 分の 1 から 100 分の 1 程度の圧縮を可能としている。

最近では、ウェーブレット変換を採用しているより高圧縮な JPEG2000 や、動画に対応した Motion-JPEG という圧縮方式も出ている。

JPEG コーデックのアルゴリズムは図のフローに示すように、4 つの変換方式、色空間変換（YUV 変換）・離散コサイン変換（DCT）・量子化・ハフマン符号化の順に変換を経てエンコードし、デコードの場合は逆順（ハフマン復号化・逆量子化・逆 DCT・逆色空間変換（RGB 変換））に変換を行う。ただし、量子化とハフマン符号化で使用したテーブルと同一のものでなければ、逆量子化とハフマン復号化ができないので注意が必要である。

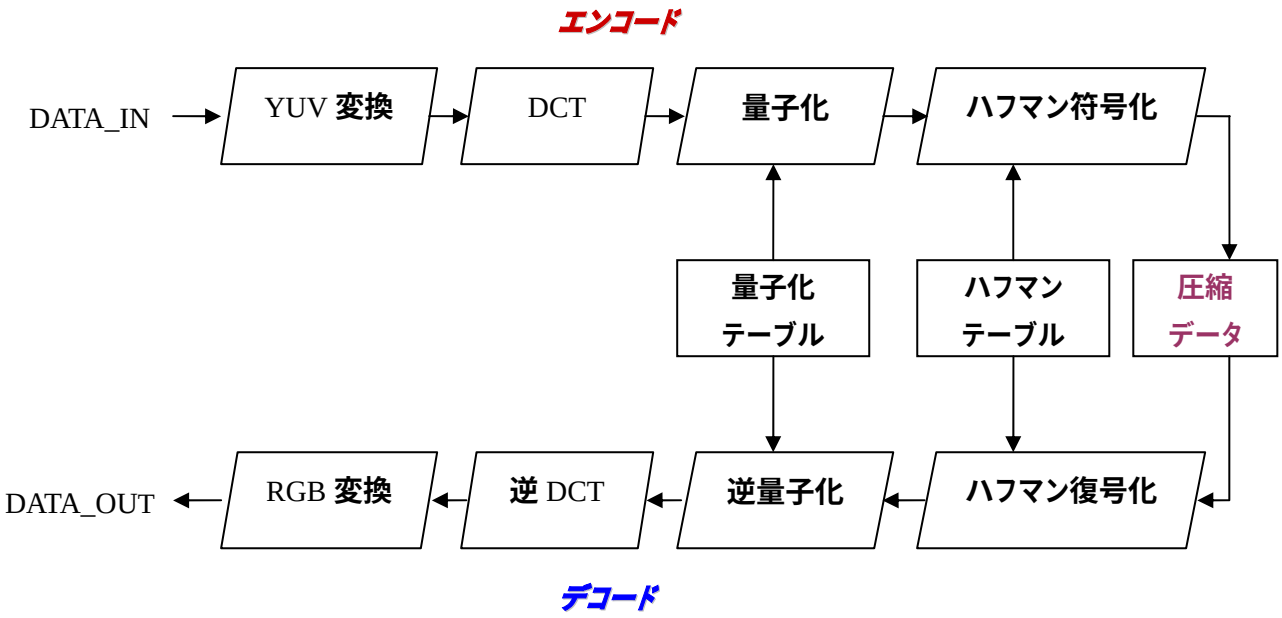


図 29：JPEG コーデックフロー

本研究で作成するのは圧縮を行うエンコーダ部分である。

なお、JPEG エンコーダの各モジュールの設計は本研究の共同研究者である古川氏、的場氏が行ったので設計に関する詳細はそちらを参照すること[13][15]。次節より簡単に JPEG エンコードアルゴリズムについて説明する。

4.2 各モジュールの説明

4.2.1 色空間変換（YUV 変換）

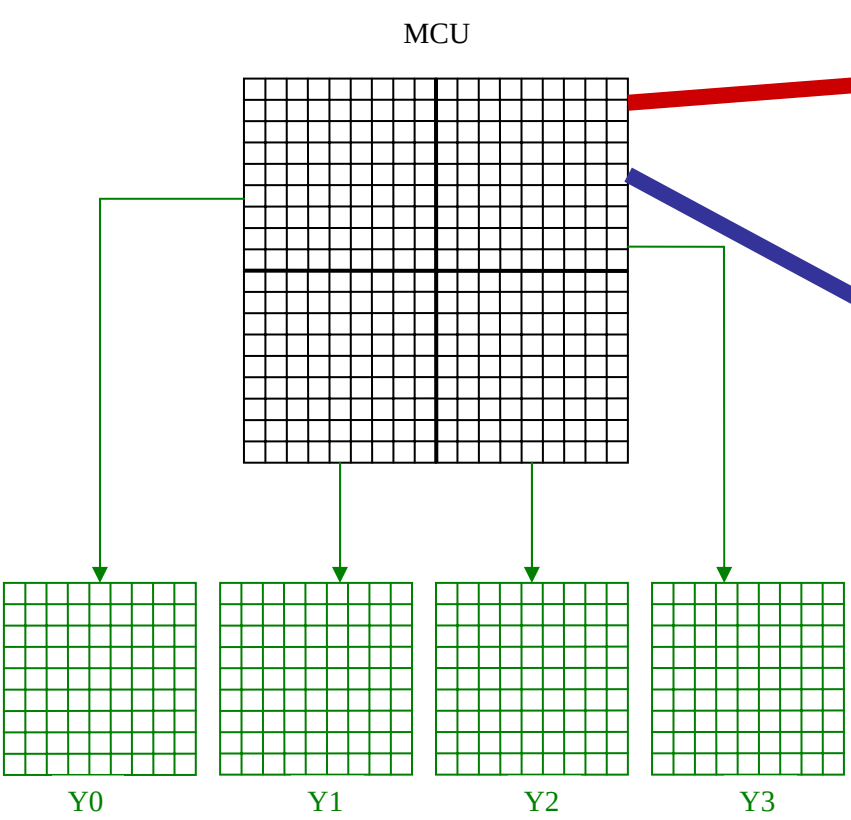
カラー画像を表現する際にもっともよく知られているのは三原色を用いた RGB 値だろう。しかしカラー画像を表すのに、この RGB と線形変換できるならば、どのような値でも等価である。RGB のほかでカラー画像を表現する代表的なものに YUV がある。YUV は輝度信号 Y と色差信号 U、V を用いてカラーを表現する。YUV に似た表現である YIQ というものが、NTSC（National Television System Committee）方式で使用される。

YUV 値に変換する理由は、画像の劣化を防ぐように圧縮する際に、この方が扱いやすいことがあげられる。人間の目は明暗や輝度には非常に敏感だが、色彩に関しては鈍いという特性がある。これを利用して情報量を間引くには輝度と色彩が分かれている YUV が有効である。

YUV 値は次の式(2)によって RGB 値から変換できる。

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B - 128 \\ U &= -0.1687R - 0.3313G + 0.5B \\ V &= 0.5R - 0.4187G - 0.0813B \end{aligned} \quad \cdots(2)$$

色空間変換では色差に関する情報の間引きも行われている。色情報についてサブサンプリングをして YUV を Y：U：V=4：1：1 の割合で情報の取得をする。この色情報を取得する際に使用する 16×16 のデータの塊を MCU と呼ぶ。つまり Y 成分の情報を重視して、U と V の情報量は Y の 4 分の 1 にする。その様子は図 30 に記載する。



U と V は MCU のブロックを
1 つ飛ばしに値を取得することで
Y の 1/4 の情報量になる

図 30：色空間変換

4.2.2 離散コサイン変換 (DCT)

離散コサイン変換 (以下、DCT) は入力された画素値から周波数成分への変換を行う。次の式で実現される。

$$X_i(u,v) = \frac{2}{N} C(u)C(v) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} X_i(m,n) \cos\left[\frac{(2m+1)u\pi}{2N}\right] \cos\left[\frac{(2n+1)v\pi}{2N}\right] \quad \cdots(3)$$

$$C(u)C(v) = \begin{cases} \frac{1}{\sqrt{2}} & (u,v=0) \\ 1 & (others) \end{cases} \quad \cdots(4)$$

式(3)において、N は正方行列の行数もしくは列数である (本研究では常に 8)。m は x 方向、n は y 方向の位置を表す。

画像は 2 次元なので当然、DCT も 2 次元変換をしなければならない。上記の式も 2 次元

DCT を表している。しかし、2 次元の変換をそのまま行おうとすると回路が非常に大きくなる。そのため 1 次元 DCT を入力データに対して施した後に、データを転置してからもう 1 度することによって 2 次元 DCT とする。速度は落ちるが、元の巨大な積和演算ユニットやテーブル参照のためのメモリに比べれば総合的に優れていると判断して採用した。

1 次元 DCT は 8×8 の 11 ビットの固定小数点精度の積和演算で構成される。

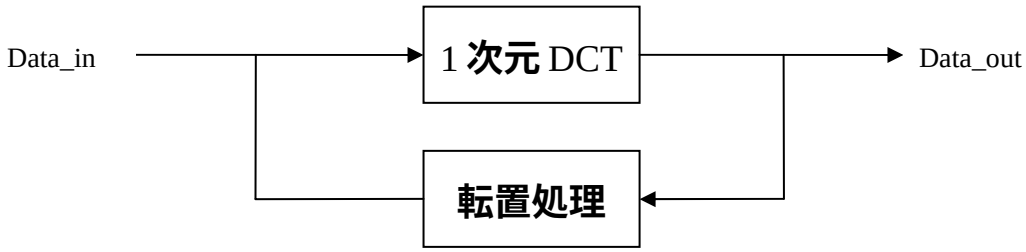


図 31：離散コサイン変換

4.2.3 量子化

量子化は DCT の結果を整数で近似するものである。各 DCT 係数に対する量子化テーブルの値で除算を行い、量子化する。除算は 11 段のパイプラインになっている。

また、量子化モジュールはハフマン符号化をしやすくするためにデータを 1 次元に並べ替える必要があるので、量子化の後、ジグザグスキャンを行う。1 次元配列に並べ替える理由は、符号化効率を高くするため低周波成分から順にする必要があるためである。

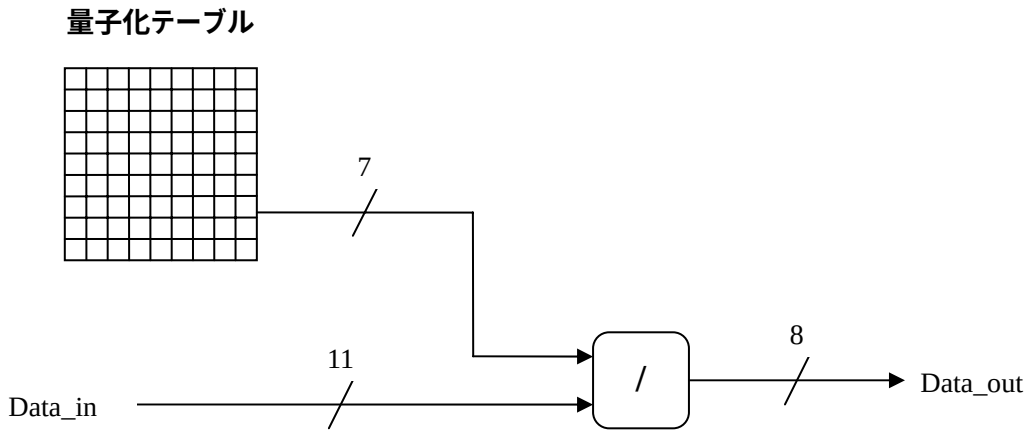


図 32：量子化

4.2.4 ハフマン符号化

統計頻度に応じた可変長ハフマン符号へと変換する。ハフマン符号化は可逆符号化である。内容としてはテーブル参照や加減算、シフト、分岐など多岐に渡り、エンコーダの中でも非常に複雑な制御を必要とする部分となっている。

注意するのは入力されるデータ(ブロック)の先頭を DC 成分と言い、他を AC 成分と言うことである。DC 成分と AC 成分とでは処理の方法が異なる。DC 成分は特に輝度が高いことが特徴である。DC 成分は前ブロックとの差を符号化し、AC 成分はゼロランレングスと有効成分をセットで符号化する。

4.3 ハードとソフトの分割パターン

分割パターンを模索する前に、AES 暗号化システムと同様に JPEG エンコードのモジュールの CPU 負荷について図 33 に記載する。

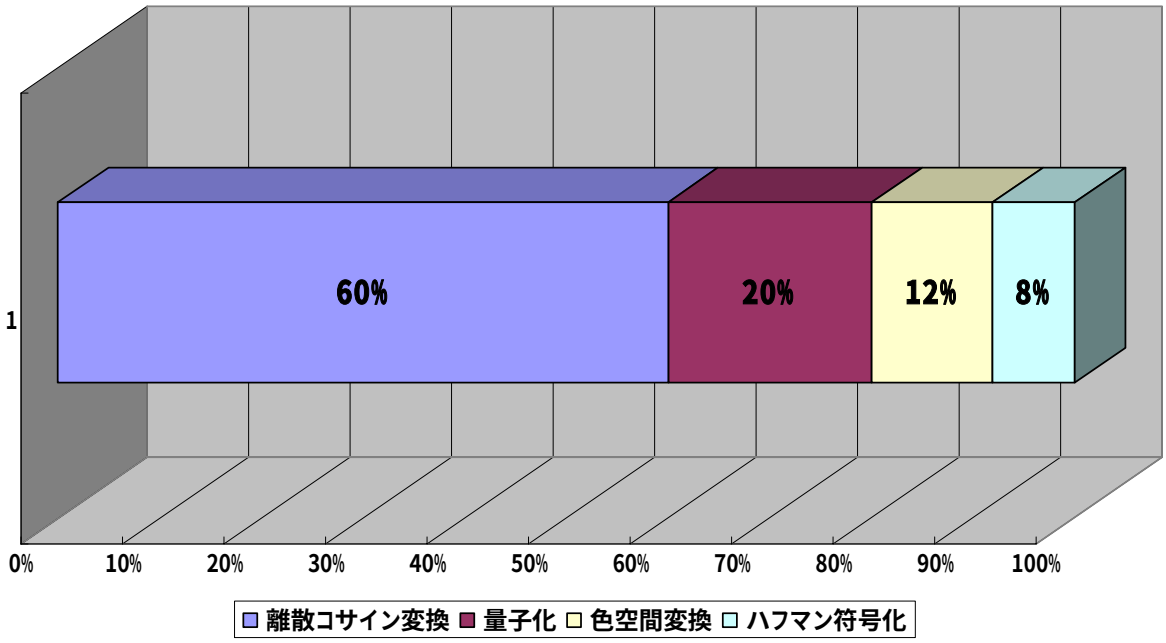


図 33:JPEG エンコードの各モジュールの CPU 負荷

最初に見て取れるのは DCT の割合が圧倒的に高いことである。DCT は積和演算器で構成されており、乗算の回数を数えると、1MCU で 6144 回もなされている。また、色空間変換でも乗算は多数行われているが、こちらは 1152 回となっている。DCT と色空間変換の乗算回数を比較するとちょうど 5:1 の関係になっている。負荷も同じ比率なので、乗算器がそのまま負荷の差となって現れていると考えられる。さらに図 33 で 2 番目に負荷をかけている量子化につ

いてだが、これは主に除算器で構成される。量子化の除算回数は 384 回である。ハフマン符号化は処理の複雑さに比べると、あまり負担が大きくない。

図 33 を参考としてハードウェアとソフトウェアの分割を考えてみる。

JPEG エンコーダでは機能ブロック単位での分割にのみ注目している。理由として、第一にデータ転送に DMA 転送を採用しており、データの授受には負担が少ないこと。第二に演算の最適化を行おうにも、ブロックごとの処理がそれぞれ強く独立しているために効果が見込めない点があげられる。

表 12：JPEG エンコーダの分割パターン

分類	ハードウェア処理部	ソフトウェア処理部
S		色空間変換, DCT, 量子化, ハフマン符号化, 制御
A	量子化	色空間変換, DCT, ハフマン符号化, 制御
B	色空間変換	DCT, 量子化, ハフマン符号化, 制御
C	DCT	色空間変換, 量子化, ハフマン符号化, 制御
D	DCT, 量子化	色空間変換, 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
G	DCT, 量子化, ハフマン符号化	色空間変換, 制御
H	色空間変換, DCT, 量子化, ハフマン符号化	制御

4.4 実験と考察

4.3 節で挙げた JPEG エンコーダの分割パターンでの実験結果を表 13 と図 34 に示す。図 34 のグラフは数値を各項目の最大値を 100 として算出してあり、理論上の最大値は 300 となっている。パーセンテージに換算する理由は、単位が項目ごとで異なる上に数値の幅に差がありすぎるため、使用メモリ量を表記してもグラフでは見えづらくなるからである。

結果は 1MCU のデータを 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である。

表 13:JPEG エンコーダの実験結果

	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
使用メモリ量 (Byte)	8372	6988	6772	6508	5332	4820	3644	3019	2244
回路規模(gate)	0	4403	4986	35120	39523	40106	44509	49857	54843

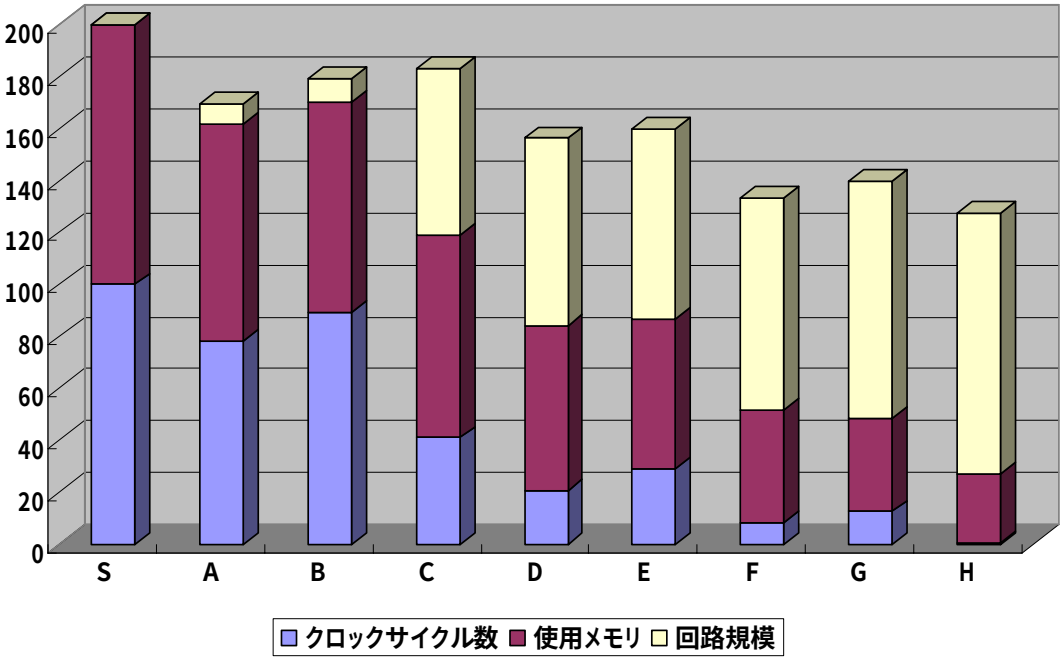


図 34:JPEG エンコーダの分割パターンでの実験結果

次に、これらの実験結果を考察する。

項目 S や H は、それぞれ全てをソフトウェアやハードウェアで実現した状態である。これらを求めるのは要求が極端な場合であり、分割パターンを探索する意味が薄いので考察からは除外する。それら以外に注目すると、S→A や S→C、C→D が 1 つの方針として、項目 F や G も最適な解を求めるのに重要と思われる。

モジュールを 1 つだけハードウェア化した場合を見てみると、C の DCT をハードにした場合がクロックサイクル数の削減と回路規模の増大、そしてメモリ量の削減に関係している。これは図 33 からも予想できる通り、DCT が JPEG エンコーダの処理の大部分を担っているためと、積和演算器という大きな回路を持っているため、さらにコサインを計算する際に使うテーブルが消せるためである。

次に注目するのは量子化をハードウェア化した項目 A である。数値が大きすぎて見取りづ

らいが、量子化をハードにしても DCT ほどのクロックサイクル数の削減ができないが、それでもかなりの量を減らせている。DCT と比較すると約 3 分の 1 である。回路規模も DCT ほどの増加はない。この事は図 33 の C→D からも容易にわかる。一方の色空間変換については、他ほど効果が見られない。量子化のクロックサイクル数削減に対する効率がよい理由は、除算器で構成されていることが挙げられる。除算は乗算よりも、さらに複雑な処理である。これをハードウェアにできるのは大きなメリットとなるだろう。

項目 F や G はメモリの使用量の低さから言えばトップクラスである。項目 G メモリ使用量において有利な理由としては、ハフマン符号化が非常に複雑な処理を行っていたので、それをハードウェアにできることにある。だが、メモリ使用量やクロックサイクル数の削減に貢献できるが、複雑な回路になるためゲート数も急増する。これら 2 つの項目 F、G のうちのひとつを選択する場合、回路規模の制限が多少緩く、かつ速度を求めるなら F が良い。回路規模制限がギリギリでメモリをとにかく減らしたいなら G が優先されるだろう。

以上の点を踏まえて考えると、大幅な速度向上には DCT のハードウェア化は必要不可欠であるが、その分の回路規模の増加にも目を見張るものがある。そして、量子化が非常にハードウェアにした際の効率がよいことから、DCT をハード化するなら回路規模制限と見合わせて量子化もハード化したほうが最適であるといえるだろう。また、メモリを考慮するなら F と G のどちらかであり、実際の制約条件との兼ね合いもあるが、ハフマン符号化をハードにするメリットがあまりないことからすると、F の方がやや有利と言える。

5. 分割手法の提案

実際のアプリケーションに対して分割手法を適応し、3 章と 4 章において実測値を得た。これらを定量的に評価するために評価の流れと評価式を考案した。評価式については式(4)に示す。

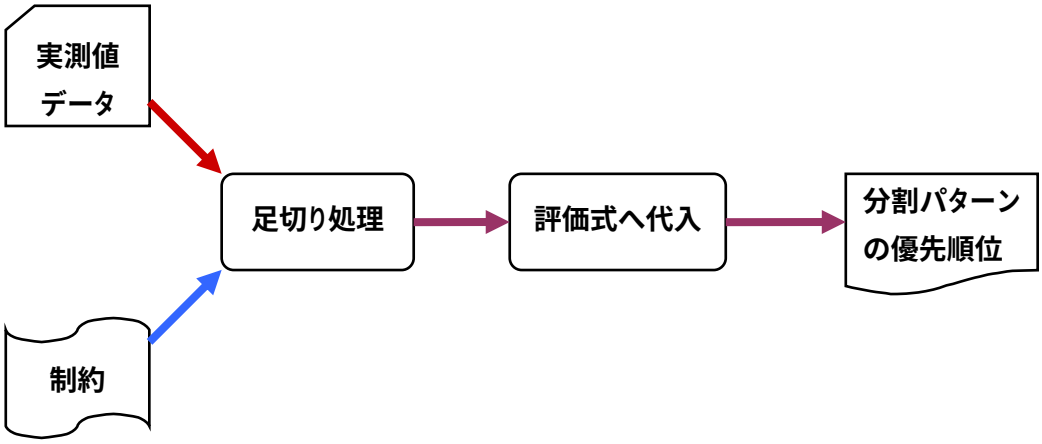


図 35：分割パターンの評価フロー

評価の流れは、まず実測値と充たされなければならない制約条件を用意する。ここでの制約条件は回路規模制限やメモリ量の限界などの数値である。その後、制約条件を充たしていない分割パターンを足切りする。全てをソフトウェアやハードウェアにすると、そのパターンは極端な値が出現しやすい。その安全措置として足切りを行う。次に評価式に実測値を代入してプライオリティ（優先順位度）を決定し、全パターンで同様にして序列をつける。

$$Piority_{pattern} = \sum_{\substack{Item = C, \\ G, M \ldots}} \left(weight_{Item} \frac{Value_{worst} - Value_{this}}{Value_{worst} - Value_{best}} \right)_{Item} \quad \cdots (4)$$

Priority ： あるパターンの優先順位度

Item ： 項目、本研究ではクロックサイクル数（C）、回路規模（G）、使用メモリ量（M）

weight ： 項目（Item）ごとの重み、ただし全項目の weight の総和は常に 1

Value ： 実測値、添え字の”worst”は該当項目するパターンの最大値、”best”は該当するパターン項目の最小値、”this”は該当するパターンの実測値

この式は全パターンにおいて、項目ごとに最悪の値（最大値）と最良の値（最小値）を抽出して、最大値を 1、最小値を 0 として、評価値を導出したいパターンの実測値が最大から最小の間で、どのあたりに位置するかを割り出して重み付けをしてその項目の優先順位を計算し、任意のパ

ターンでの評価対象項目の総和を取って、パターン全体の優先順位を決定するものである。

Priority の最大値は Item 数と等しくなり、その値に近いほど目的に合致する可能性が高いことを示している。

この評価式を実現するプログラムを作成した。その実行の様子を図 36 に載せる。

このプログラムはコマンドライン引数に各評価項目の重みをクロックサイクル数、回路規模、使用メモリ量の順に入力し、最後に実測データの入ったファイルを渡す。結果には、足切りした分割パターンを除くプライオリティが数値として小数点以下 6 桁まで出力される。

(1) AES 暗号システムの実行結果

```
$ ./a 0.4 0.3 0.3 real_values.txt
"S" is cutbacked...
"G" is cutbacked...
MAXc:26959, MINc:10549
MAXg:9957, MINg:324
MAXm:4259, MINm:3731


priority"A":0.484091
priority"B":0.042883
priority"C":0.560567
priority"D":0.687357
priority"E":0.436847
priority"F":0.563636
```

(2) JPEG エンコーダの実行結果

```
$ ./a 0.4 0.3 0.3 jpeg_values.txt
"S" is cutbacked...
"H" is cutbacked...
MAXc:697408, MINc:66057
MAXg:49857, MINg:4403
MAXm:6988, MINm:3019


priority"A":0.354467
priority"B":0.312479
priority"C":0.371392
priority"D":0.533358
priority"E":0.526094
priority"F":0.688056
priority"G":0.679199
```

図 36：評価式プログラムの実行の様子

表 14 に、極端な重み付けをした場合（目的が明確な場合）の組み合わせの結果を載せる。ただし、表 14 の評価には足切り処理はなされていない。表 14 において、分割パターン名が赤く彩色されているのは全機能をハードウェアもしくはソフトウェアにしているものであり、足切り処理で消える可能性の高いパターンである。

表 14：重みが極端な場合の優先順位

	クロックサイクル数重視	回路規模重視	使用メモリ量重視
重み（クロック：回路規模：使用メモリ量）	0.8：0.1：0.1	0.1：0.8：0.1	0.1：0.1：0.8
AES 暗号システム	ト＞ヘ＞G＞ホ＞F	S＞A＞D＞F＞C	ト＞ヘ＞ニ＞ホ＞D
JPEG エンコーダ	H＞F＞G＞D＞E	S＞A＞B＞C＞D	H＞G＞F＞E＞D

本研究から理解できたことをまとめると、まず FPGA やソフト・マクロ CPU を使用した設計や検証は、非常に楽であるということである。FPGA なので実装して誤動作が起こっても、実際に機能的に等価な回路を作った場合と比較すると、金銭的な損失が無いも同然ので、実験に積極的に取り組めた。その上、分割パターンを一本化する必要がなく、複数パターンを同時に実装・評価できる。さらにソフト・マクロ CPU である MicroBlaze を使用することで、デザイナが本当に集中したい回路モジュールや機能以外の煩雑な作業から開放され得る。同一 FPGA 上に CPU を搭載するのでハード/ソフト協調設計が容易になる点もメリットに挙げられるだろう。また、必要な周辺の機能は IP で集められるため、比較的に短期間で設計・検証できる。

次に問題点を挙げると、分割パターンの選出を行う指標が少ないことである。現在は、対象アプリケーションのアルゴリズムの特徴やソフトウェア実行の際の CPU 負荷しか見ていないので、考慮する情報が不足している状態となっている。よって選出したパターンも 5 つ以上と、かなりの数になってしまった。問題点としてもうひとつ、評価項目が少ないこともあるだろう。今回はクロックサイクル数、回路規模、使用メモリ量の 3 つしか上げていない。これでは実際に開発するときの制約には程遠い。

今後の課題として、対象アプリケーションをソフトウェアで動かすことがハード/ソフト最適分割を探る第一段階となるが、そのソフトウェアの段階でどれだけ最適な分割パターンを見つけるための情報を収集できるか、がある。これができれば、さらに設計期間や手間が少なくなる。そのために、現在 MicroBlaze でソフトウェアを動かしたときの命令の実行状況を取得するシステムを考案し、実現に向けて研究を進めている。この動的命令実行の統計を計測し、機能ブロックごとの演算命令や分岐命令、ロード/ストア命令など種類ごとの統計を取ることで、分割パターンでの指標のひとつとしたい。

また、現在の評価項目はクロックサイクル数、回路規模、使用メモリ量の 3 つだが、消費電力や工数、再利用性、コスト、保守性なども評価対象に含める必要がある。

6. おわりに

本研究では、現在のシステム LSI の設計について考察し、ハード/ソフト協調設計における最適分割を検証するために、AES 暗号化システムと JPEG エンコーダを設計した。また、Xilinx 社の提供するソフト・マクロ CPU である MicroBlaze を用いて、Memec 社の FPGA ボードの上に両システムを実装した。

今後の課題として、ソフトウェアプログラムの動的命令実行数を観測して、より精密な予測を行うことがあげられる。さらに、現在のハード/ソフト分割手法における設計空間探索の最適化を行う評価式の精度を上げることも望まれる。

LSI 設計は順調に発展を続けており、今後も拡大し続けるであろうことは想像に難くない。一方で、少し前にあったソフトウェア危機というものと似た現象も LSI の業界では発生しつつあり、LSI の設計が危ぶまれている。当然、その解決策はいくつか挙げられており、主なものにはハード/ソフト協調設計、システムレベル言語設計（高位合成）、設計の再利用などがある。中でもハード/ソフト協調設計や高位合成などではハードウェアとソフトウェアの双方の十分な知識が必要とされる。今後の LSI 設計に要求される事柄は一面的な知識では賄いきれないだろう。そういった意味で、情報学科の出身であり、新設される電子情報デザイン学科というハードとソフトの両面を学ぼうとする学科に組み込まれる本研究室は最適な環境に近いだろう。本研究室の研究活動がさらに活発になり、将来の技術に関われる研究や、必要とされる研究員が多く輩出されることを願って止まない。

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、的場氏、及び色々な面で励ましを下さった高性能計算研究室の皆様に心より深く感謝いたします。

参考文献

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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, pp127-132, 2003.11.

[7] 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, pp72-79,Vol2003.1.29.

[8] 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, pp154-160, Vol2003.1.29.

[9] 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.

[10] 貴家仁志:よくわかるディジタル画像処理,CQ 出版,1996

[11] 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.

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

[13] 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,修士論文,2005.

[14] Pablo Moisset,Pedro Deniz,Joonseok Park:Matching and Searching Analysis for Parallel Hardware Implementation on FPGAs, ACM Press, Feb. 2001.

[15] 的場督永: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,立命館大学,卒業論文,2005.

[16] 出口勝昭,山田裕,天野英晴:DRP でのウェーブレットフィルタの実装,情報処理学会研究報告-システム LSI 設計技術,2003 年 1 月 Vol.2003 no.007.

[17] 中森明:マイクロプロセッサ・アーキテクチャ入門,CQ 出版社,2004.

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

[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.