卒 業 研 究 報 告

題　目

VHDL による JPEG コーデックの設計
- 量子化回路、逆量子化回路、DCT 変換回路の設計 -

指 導 教 員

橘 昌良 助教授

報 告 者

学籍番号: 1030208

氏名:　　中田 勝

平成 15 年 1 月 27 日

高知工科大学 電子・光システム工学科

目次

第1章　序論

1-1 背景

近年、インターネットやディジタルカメラ、パーソナルコンピュータなど静止画像も扱えるマルチメディアシステムが、身近な存在となりつつある。それら静止画像の圧縮アルゴリズムには、JPEG(Joint Photographic Experts Group) を使用しているものが数多くある。現在、このような圧縮アルゴリズムは、さらに高機能化、高品質化に向かって進化を続けている。[1]

1-2 研究目的

本研究では、JPEG コーデック (COmpression / DECompression) の設計とシミュレーションをすることで、JPEG の仕組と、データ量削減のための基本的な圧縮技術について学習する。また、回路の設計に VHDL(VHSIC HDL) を使用することで、HDL(ハードウェア記述言語：Hardware Description Language) による設計手法についての知識も得る。[4][5][8]

1-3 論文構成

本論文では、本章で研究全体の目的を明らかにし、第 2 章で JPEG の概要について説明する。第 3 章において本研究で設計した JPEG コーデック (量子化回路、逆量子化回路、DCT(Discrete Cosine Transform) 変換回路) について述べ、第 4 章でそれら回路の VHDL の階層とシミュレーションについて述べる。第 5 章ではまとめ・今後の見通しを述べ、最後に、付録として本研究で作成した VHDL 記述と、C 言語の記述を添付する。[1]

第2章　JPEGの概要

2-1 ディジタル画像とその情報量

現在、ディジタル (スチール) カメラでは画像情報をディジタルデータでフラッシュモリに記憶している。ディジタル化された画像の大きさを表す単位に、画素 (pixel) というものがある。横方向に 2048 個の画素、縦方向に 1536 個の画素を持つカラー画像があるとする。この画像は 2048 × 1536pixel(2048 × 1536　3.14Mpixel) つまり 314 万画素ある。1 画素ごとに 1bit の情報を割り当てると、3.14Mbit(3.14 ÷ 8　0.39MB) の情報量となる。さらに、カラー画像を表現するには、一つ一つの画素に色の情報を追加しなくてはならない。色を表現する方法には RGB モード、24 ビットカラーといったものがある。RGB とは Red(赤)、Green(緑)、Blue(青) で構成された「光の三原色」の輝度を表し、これによりいろいろな色を表現することができる。この RGB の各 3 原色 128 ビット (256 階調) を割り当てることで、256(赤) × 256(緑) × 256(青) = 224 = 1677216 色 (24 ビットカラー) が表現できるようになる。これらのことから、カラー画像の情報を含めて計算すると、2048 × 1536 × (24 ÷ 8)　9.43MB の情報量となる。この場合では、16MB のフラッシュメモリに 1 枚の画像しか記憶できない。以上のことから通常、画像データは圧縮して記憶される。[1][2][3]

2-2 JPEG(画像圧縮)

静止画像を圧縮する技術の一つに JPEG がある。JPEG とは ”Joint Photographic Experts Group ”の頭文字で、ISO(International Organization for Standardization) と CCITT(Consultative Committee for International Telegraph and Telephone) の二つの組織がカラー静止画像符号化標準方式を目指し、ジョイントして設立した検討グループを指している。標準方式の正式な名称は ”Digital compression and coding of continuous-tone still images ”であり、この名称が示すとおりモノクロまたはカラーの連続的な階調を有する静止画の符号である。つまり、正確にいえば JPEG はファイルの圧縮伸張方式の名称で、ファイルフォーマット名は JFIF(JPEG File Interchange Format) と呼ばれることもある。[1][2]

2-3 JPEG 符号化アルゴリズムの分類

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複数の動作モードに分類している。ユーザは必要な機能を選択することによって、JPEG を、利用する用途にあった表示・圧縮・保存・転送を効率よく行うことができる。ここでは図 2-1 に示した JPEG 符号化アルゴリズムの分類について説明する。[1][2]

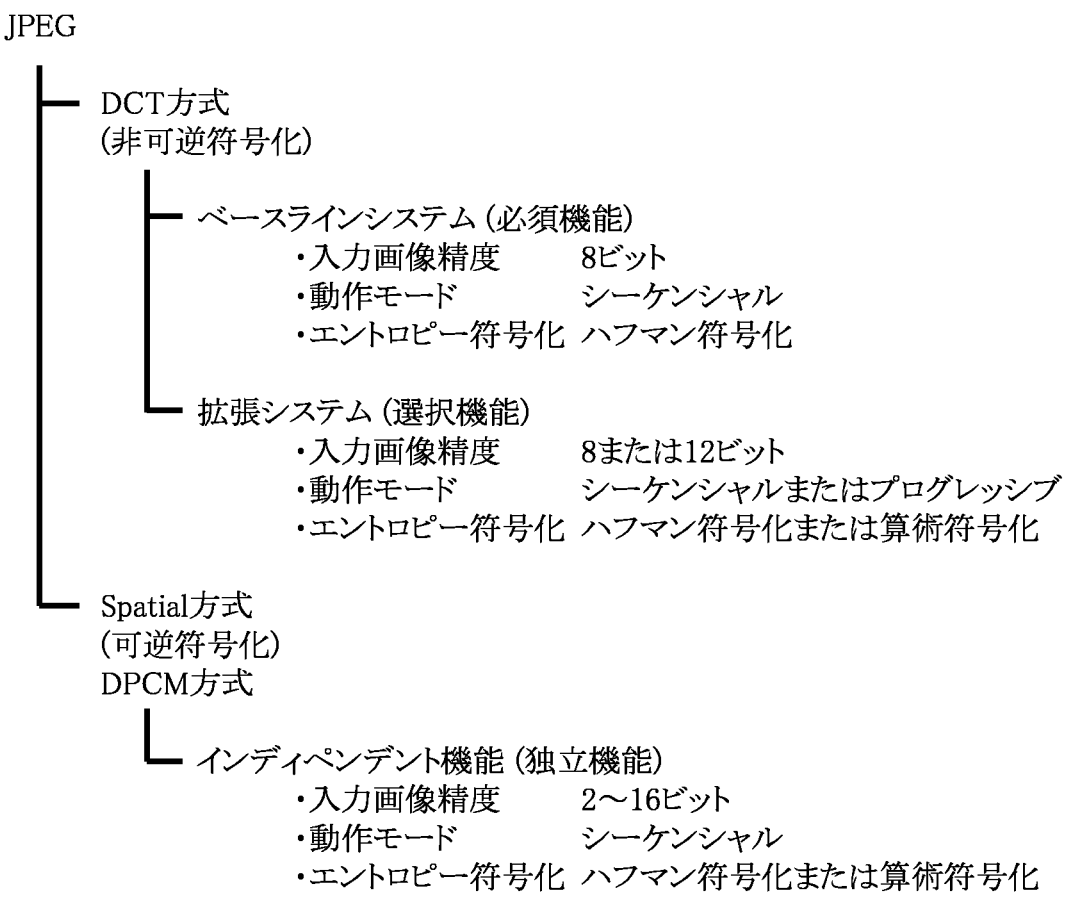

図 2-1 JPEG 符号化アルゴリズムの分類

2-3-1 DCT 方式と Spatial 方式

JPEG では符号化を行う前の画像とまったく同じ画像が得られる可逆符号化と、同じ画像が得られない非可逆符号化とがある。符号化アルゴリズムの基本構成によって分類すると、可逆符号化には Spatial 方式が、非可逆符号化には DCT 方式が用いられている。

非可逆符号化の DCT 方式にはベースラインシステムと拡張システムがある。ベースラインシステムは、すべての JPEG 符号器・復号器が有しなければならない最も基本的な機能 (必須機能) である。これに対し、拡張システムはより広い範囲への対応のために選択できる機能 (選択機能) である。また、Satial 方式は符号化に、DPCM(Differential PCM) とエントロピー符号化を使用しており、DCT 方式とはアルゴリズムの連続性はないためインディペンデント機能 (独立機能) と呼ばれる。[1][2]

2-3-2 シーケンシャルとプログレッシブ

JPEG には画像を表示していく方式にシーケンシャル・ビルトアップ方式とプログレッシブ・ビルトアップ方式がある。シーケンシャル・ビルトアップ方式とは、画像を表示するときに上から順に鮮明な画像を表示していく方式で、ファクシミリなどの通信ではこのような方法が用いられる。だだし、Web のような場所では、この方式だと画像の全体像がわかるのに時間がかかることが有る。そこで、プログレッシブ・ビルトアップ方式では最初に画質は悪いが、全体像がある程度わかる画像を表示後、画質を徐々によくしていくという方法をとっている。この方式では画像検索サービスや、Web などの画像の全体像を早く知りたいときに有効である。[2]

2-3-3 ハフマン符号化と算術符号化

エントロピー符号化にはハフマン符号化と算術符号化がある。ベースラインシステムではハフマン符号化を使用する。算術符号化は拡張システムと Spatial 方式でハフマン符号化と選択比べて圧縮効率が若干高いが、それに伴って処理が複雑になる。[1]

2-4 JPEG アルゴリズム (DCT 方式)

DCT 方式のベースラインシステムの構成を図 2-2 に示す。本研究では、JPEG コーデックの設計を行が、JPEG のアルゴリズムには DCT 方式のベースラインシステムを使用する。そのため、以下の節では本研究で設計した、DCT 変換と DCT 逆変換、量子化と逆量子化のアルゴリズムについて説明する。[1][2]

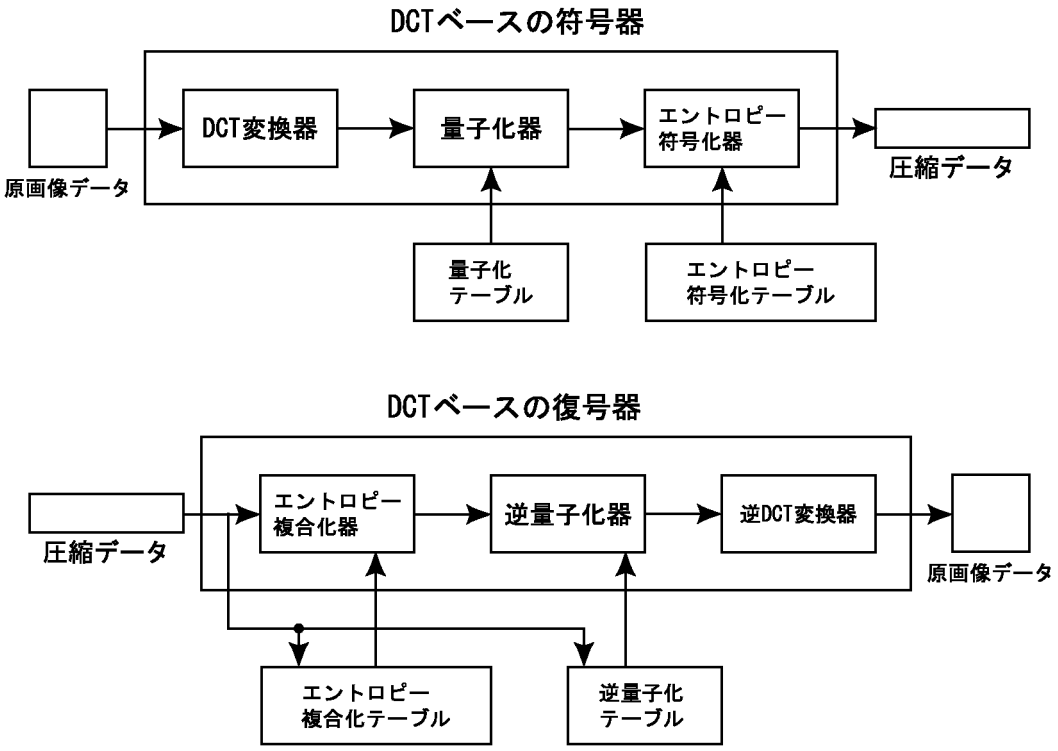

図 2-2 DCT ベースの JPEG 符号器・復号器の構成

2-4-1 DCT 変換と DCT 逆変換

DCT 変換 (離散コサイン変換：Discrete Cosine Transform) は画像信号を空間周波数に変換するための一手法である。JPEG の DCT 方式では、最初に 8 × 8 画素から構成されるブロックの集合である MCU(Minimum Coded Unit) と呼ばれる単位に分割する。このことから、ブロック内の 2 次元データを $s_{yx}(y, x = 0, 1, \cdots, 7)$、DCT 変換係数を $S_{vu}(v, u = 0, 1, \cdots, 7)$ とすると、DCT 変換および DCT 逆変換は以下の式で定義される。[1]

DCT 変換:

$$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \quad (2-1)$$

DCT 逆変換:

$$s_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{vu} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \quad (2-2)$$

ここで、DCT 変換と DCT 逆変換での C_u, C_v と u, v の関係は

$$C_u, C_v = \begin{cases} 1/\sqrt{2} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases} \quad (2-3)$$

となる。

DCT 変換係数 S_{vu} は、一つのブロックに対して 64 個 ($S_{00} \sim S_{77}$) 存在する。図 2-3 に JPEG における DCT 変換の例を示す。この図を見て分かるように、S_{00} 周辺に大きな値が集中している。この後量子化を行った後、ハフマン符号により、S_{00} 付近の情報は大きなビットを与え、それ以外のところは、小さなビットを割り当てることにより、情報を圧縮することができる。そのため、S_{00} を DC 係数と呼び、それ以外 ($S_{01} \sim S_{77}$) を AC 係数と呼ぶ。[1][2]

127	127	127	127	127	127	127	127
-123	-123	-123	-123	-123	-123	-123	-123
127	127	127	127	127	127	127	127
-123	-123	-123	-123	-123	-123	-123	-123
127	127	127	127	127	127	127	127
-123	-123	-123	-123	-123	-123	-123	-123
127	127	127	127	127	127	127	127
-123	-123	-123	-123	-123	-123	-123	-123

2 次元データ s_{yx}

→

15	0	0	0	0	0	0	0
180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
212	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
318	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
906	0	0	0	0	0	0	0

DCT 変換係数 S_{uv}

図 2-3 JPEG における DCT 変換の例

2-4-2 量子化と逆量子化

DCT 係数の量子化とは、定められた量子化ステップサイズの何倍であるかを四捨五入した整数値で求めることである。量子化することにより、64 個の DCT 係数 S_{vu} の取り得るレベル数を制限することが出来る。すなわち、2 次元 DCT 係数の量子化ステップサイズを $Q_{vu}(v,u=0,1,\cdots,7)$、量子化された 2 次元 DCT 係数を $Sq_{vu}(v,u=0,1,\cdots,7)$ とすれば、量子化および逆量子化は、それぞれ以下の式で定義される。[1]

量子化:

$$Sq_{vu} = round\left(\frac{S_{vu}}{Q_{vu}}\right) \quad (2-4)$$

逆量子化:

$$R_{vu} = Sq_{vu} \cdot Q_{vu} \quad (2-5)$$

ここで、$round$ は最も近い整数値への四捨五入を表す。このように DCT 係数の大きさに依存せず、量子化ステップサイズが一定である量子化を一様量子化といい、この場合の量子化誤差は、以下の範囲である。[1]

$$0 \leq |S_{vu} - R_{vu}| \leq \frac{Q_{vu}}{2} \quad (2-6)$$

量子化ステップサイズの最適値は、入力画像および画像の表示装置の特性に依存して定まるため、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 のガイドラインに図 2-4 で示している、輝度成分用と、色差成分用の量子化テーブルの例がある。人間の目は、高い空間周波数のひずみほど知覚されにくいという視覚特性がある。そのことを利用して、図 2-4 の量子化ステップサイズは、空間周波数が高いほど大きくなり、粗い量子化となる。なお、JPEG における画像品質の制御は、この量子化テーブルに一定乗数を掛けることによって行うことができる。そのため、本研究では表 2-4 量子化テーブルを使用して量子化回路と逆量子化回路を設計する。[1]

横方向量子化ステップサイズ

縦方向量子化ステップサイズ

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

(a) 輝度成分用量子化テーブル

横方向量子化ステップサイズ

縦方向量子化ステップサイズ

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

(b) 色差成分用量子化テーブル

図 2-4 量子化テーブルの例 (Annex K のガイドラインより)

第3章　JPEGコーデックの設計

本研究ではJPEGコーデックの中でも、量子化、逆量子化、DCT変換を設計する。量子化回路、量子化回路、DCT回路についてはそれぞれ順に3-1節、3-2節、3-3節で述べる。

3-1 量子化回路の設計

DCT変換後の信号を、輝度Y成分と色差UV成分に、それぞれ別々の量子化テーブルを用いて、量子化を行う回路を設計する。

3-1-1 全体の概要

量子化回路のデータパスを図3-1に示す。量子化回路(Quantization)には、DCT信号の読み出し/書き込みを行う6個のRAM(Random Access Memory)、(DCT_k_RAM : $k = 1, 2, \cdots, 6$)と、量子化された信号を読み出し/書き込みを行う6個のRAM(QDCT_k_RAM : $k = 1, 2, \cdots, 6$)がある。また、量子化信号を出力するための演算は、ALU(Arithmetic and Logic Unit：算術論理演算装置)で行う。ALUの説明を3-1-4節で、DCT信号用のRAMの説明を3-1-5節で、量子化信号用のRAMの説明を3-1-6節で詳しく行うことにする。[1][7][9]

3-1-2 入出力線

本研究で設計する全ての回路では、データは全て小数点位置を固定し、数によって動かさない固定小数点(fixed point number)表現を使用する。また、負数を表現するには最上位ビット(MSB：Most Significant Bit))で符号を表現('0'のとき正数、'1'のとき負数とする)する符号-絶対値表現(sign-magnitude representation)を使用する。これらのことを考慮したうえで、入出力線の各機能およびその詳細を次に示す。[7]

入力線　CLK　：1ビット
量子化回路のクロック信号のための線
(回路は立ち上がりエッジで動作)

RESET　：1ビット
量子化回路のリセット信号のための線
(回路では'1' を入力すると強制リセット)

START_QUANTIZATION　：1ビット
この線に'1' が入力されると量子化回路の
状態が初期状態から次の状態に動き出す。

SCALE　：8ビット (整数4ビット、小数4ビット)
0から7までの量子化スケールファクタの値が入る。

DCTZ_1,DCTZ_2　：13ビット
DCT信号を入力するための線 (輝度Y成分)

DCTZ_3,DCTZ_4　：13ビット
DCT信号を入力するための線 (輝度Y成分)

DCTZ_5,DCTZ_6　：13ビット
DCT信号を入力するための線 (色差UV成分)

出力線　QDCTZ_1,QDCTZ_2　：13ビット
量子化信号を出力するための線 (輝度Y成分)

QDCTZ_3,QDCTZ_4　：13ビット
量子化信号を出力するための線 (輝度Y成分)

QDCTZ_5,QDCTZ_6　：13ビット
量子化信号を出力するための線 (色差UV成分)

START_DCTZ_IN　：1ビット
この線が'1' の信号を出したときDCT信号の
入力許可を許可する。

DONE_OUT_QDCTZ_K　：1ビット
量子化信号が出力される間'1' を出力する。

DONE_QUANTIZATION　：1ビット
量子化回路が全ての量子化信号を出し終えると
'1' を出力する。

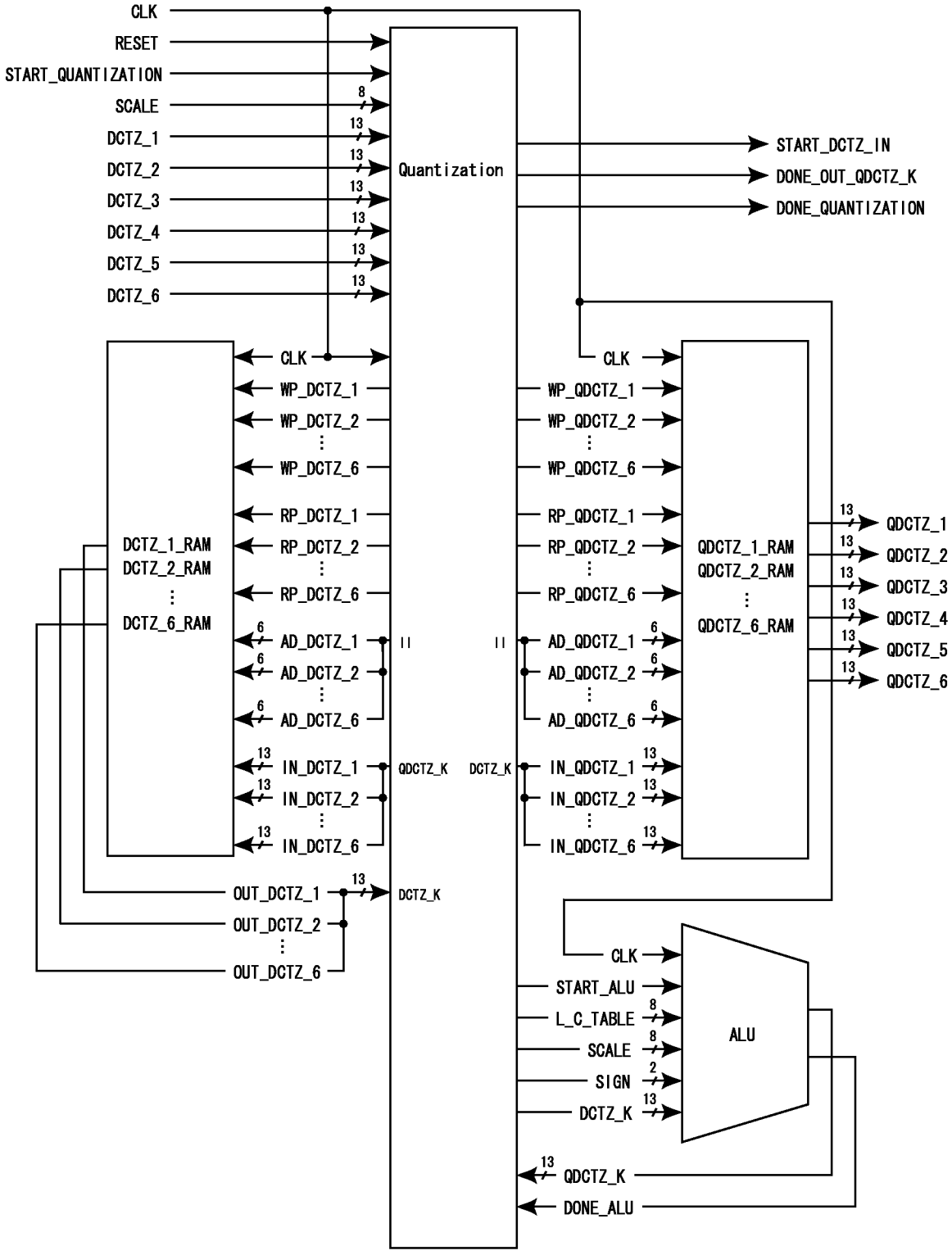

図 3-1 量子化回路のデータパス

3-1-3 動作設計

量子化回路の状態遷移図を図 3-2 に示す。この回路は全ての状態を、大きく分けて 3 つの役割に分けることが出来る。まず始めに INIT から II_DCTZ_RAM_UP までの状態で、入力されてくる DCT 信号を DCT 信号用の RAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$) に一時記憶していく。次に、K_VALUE から K_UP までの間で、DCT 信号を量子化信号に変換する演算を行っている。最後に、QDCTZ_OUT_II_VALUE から QDCTZ_OUT_II_UP までの状態で、量子化信号を量子化回路の外に出力している。このことをふまえて、以下で各状態について説明していく。

- INIT

  量子化回路の初期状態、START_QUANTIZATION に'1' が入力されるまで初期状態を持続する。このとき内部カウンタ (K と II) に 0 を入力し初期化する。

- MAKE_DCTZ_RAM

  カウンタ II(0 から 63 までの値を出力するカウンタ) が示すアドレスに、6 個の DCT 信号を各 RAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$) に記憶する。

- IF_DCTZ_RAM

  6 個の DCT 信号用の RAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$) に、それぞれ 8×8 画素分の DCT 信号が送られたか調べる。

- II_DCTZ_RAM_UP

  カウンタ II の値を 1 増やし、6 個の DCT 信号用の RAM に新しいアドレスを示す。

- K_VALUE

  カウンタ K(1 から 6 までの値を出力するカウンタ) に 0 を入力し初期化する。

- II_VALUE

  カウンタ II に 0 を入力し初期化する。

- MAKE_ALU

  START_ALU に'1' を入力し、ALU の状態を初期状態から変化させる。ALU から演算終了を知らせる DONE_ALU に'1' が送られるまで量子化回路の状態を待機

させ、ALU の状態を変化させる。

- MAKE_QDCTZ_K
  量子化した信号を、6 個の量子化信号用の RAM(QDCT_k_RAM : $k = 1, 2, \cdots, 6$) のうち、カウンタ K が示す RAM に、カウンタ II が示すアドレスに記憶する。

- IF_II
  8×8 画素分の DCT 信号を、全て量子化することが出来たか調べる。

- II_UP
  カウンタ II の値を 1 増やし、カウンタ K が示す量子化信号用の RAM の新しいアドレスをカウンタ II が示す。

- IF_K
  6 個の 8×8 画素ブロックを、全て量子化することが出来たか調べる。

- K_UP
  カウンタ K の値を 1 増やし、6 個ある量子化信号用の RAM の中で、今示してある RAM と違う RAM を示す。

- QDCTZ_OUT_II_VALUE
  RAM に記憶した量子化信号を量子化回路の外に出力するため、カウンタ II に 0 を入れて初期化する。

- QDCTZ_OUT
  6 個の量子化信号用の RAM のうち、カウンタ II が示すアドレスの量子化信号を、量子化回路の外に出力する。

- IF_QDCTZ_OUT_II
  8×8 画素分の量子化信号を、全て量子化回路の外に出力したか調べる。

- QDCTZ_OUT_II_UP
  カウンタ II の値を 1 増やし、6 個の量子化信号用の RAM の新しいアドレスをカウンタ II が示す。

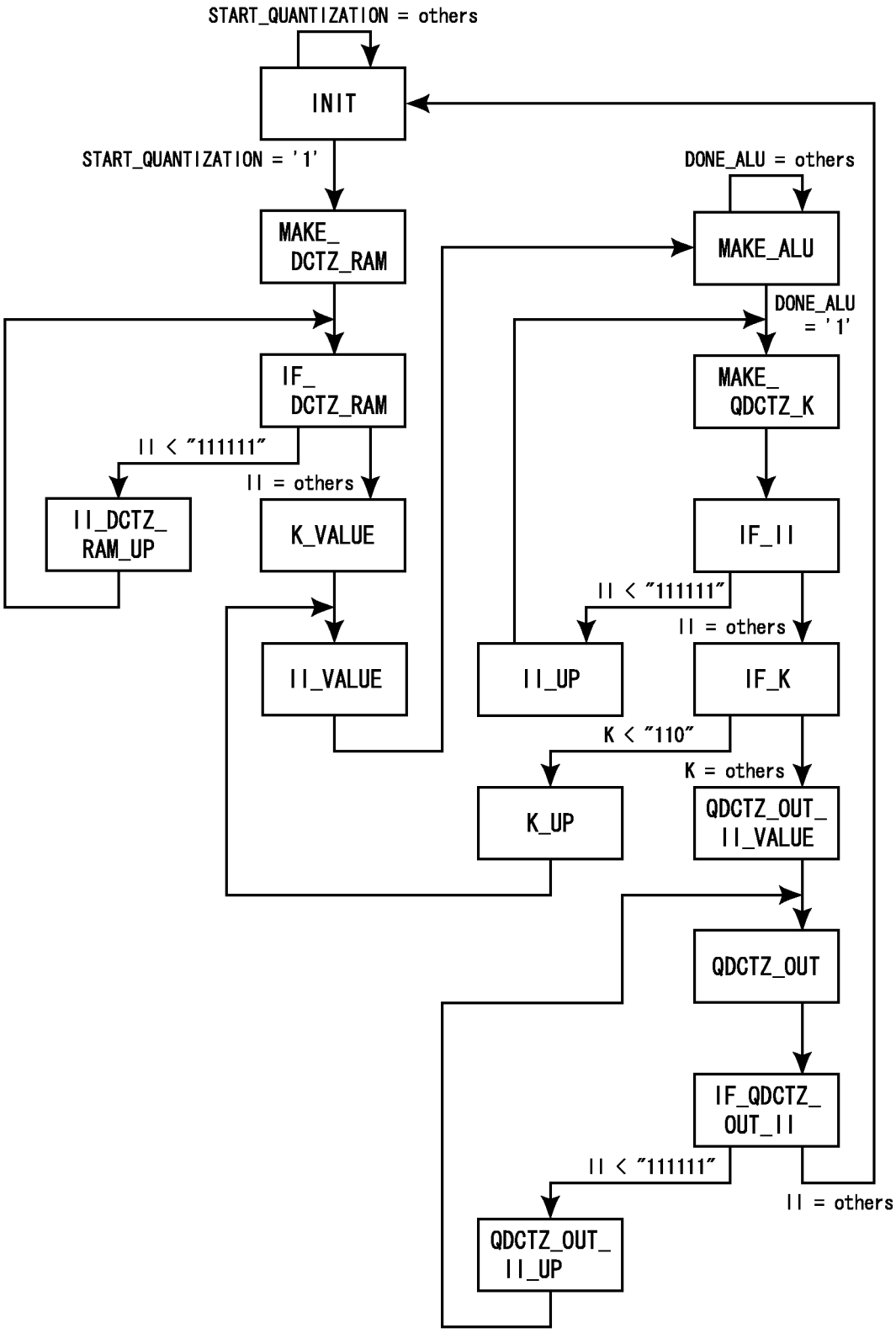

図 3-2 量子化回路の状態遷移図

3-1-4 ALU の設計

DCT 信号を量子化信号に変換するための算術演算装置を設計する。[7]

3-1-4-1 ALU の概要

ALU のデータパスを図 3-3 に示す。ALU は符号-絶対値表現で固定小数点表現の小数点数の乗算を行う MUL と、加減算を行う ADD、除算を行う DIV で構成されている。回路規模をなるべく小さくするため MUL、ADD、DIV は量子化回路では 1 個づつしか設置しない。量子化するたに必要な演算は ALU でこれら装置を制御することで実現する。また MUL の説明は 3-1-4-4 節で、ADD の説明は 3-1-4-5 節で,DIV の説明は 3-1-4-6 節でそれぞれ行う。[7]

3-1-4-2 ALU の入出力線

ALU の入出力線の各機能およびその詳細を次に示す。

入力線　CLK　：1 ビット
量子化回路のクロック信号のための線
(回路は立ち上がりエッジで動作)

START_ALU　：1 ビット
この線に'1' が入力されると ALU の
状態が初期状態から次の状態に動き出す。

L_C_TABLE　：8 ビット
量子化テーブルの値が入る。
'1' を出力する。

SCALE　：8 ビット (整数 4 ビット、小数 4 ビット)
0 から 7 までの量子化スケールファクタの値が入る。

SIGN　：2 ビット
ALU に入力される DCT 信号の値が 0 より大きければ +1 を、
0 より小さければ −1 を、0 なら 0 の値が入る。

DCTZ_K　：13 ビット
DCT 信号を入力するための線

出力線　QDCTZ_K　：　13 ビット
　　量子化信号を出力するための線
　　DONE_ALU　：　1 ビット
　　ALU が演算を終え、内部の状態が初期状態になるときに
　　'1' を出力する。

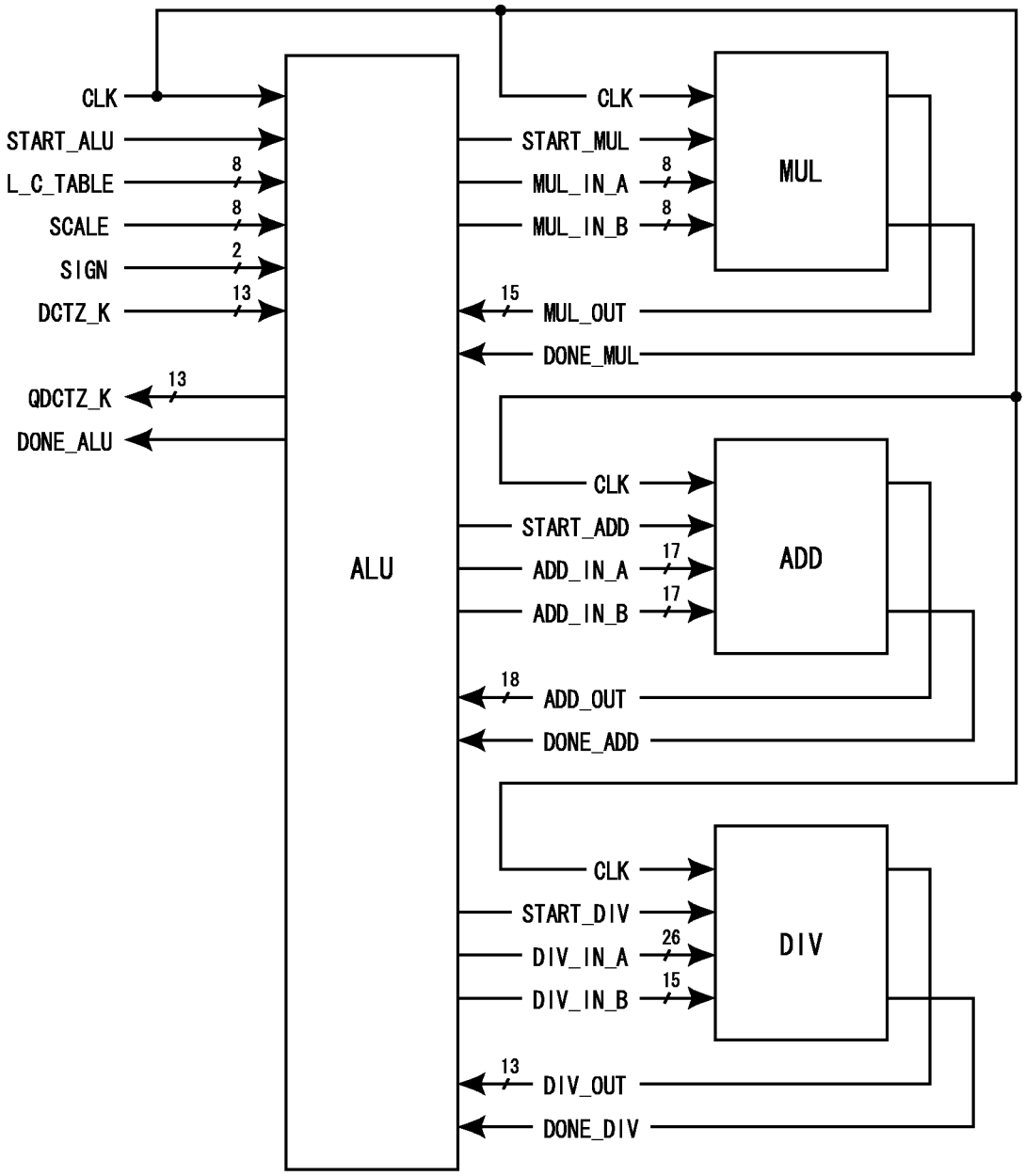

図 3-3 ALU のデータパス

3-1-4-3 ALU の動作設計

ALU の状態遷移図を図 3-4 に示し、以下で各状態について説明していく。

- INIT

  ALU の初期状態、START_ALU に'1' が入力されるまで初期状態を持続する。

- MAKE_STEPSIZE

  量子化テーブルと量子化スケールファクタを掛け、量子化ステップサイズを作る。このとき、量子化スケールファクタの値によって圧縮符号化画像の品質と圧縮率が変化する。[1]

- MAKE_MUL_SIGN

  量子化ステップサイズと SIGN の信号を掛ける。

- MAKE_ADD_DCTZ_K

  MAKE_MUL_SIGN で作った信号に DCT 信号を足す。

- MAKE_DIV_STEPSIZE

  MAKE_ADD_DCTZ_K で作った信号を 2 ビット右にシフトした後、量子化ステップサイズで割ることによって量子化信号を作る。

- MAKE_DIV_OUT

  ALU で作成した量子化信号を量子化回路に渡す。

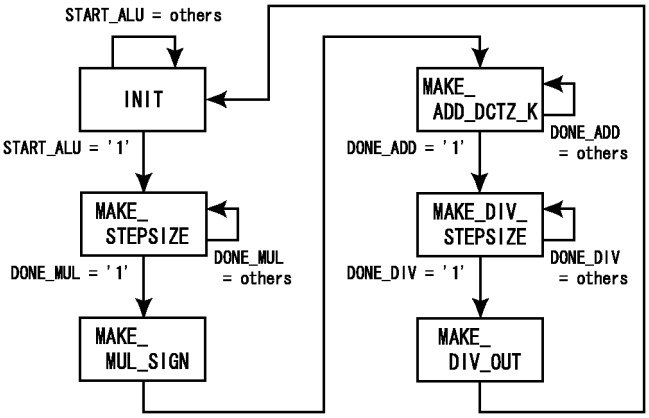

図 3-4 ALU の状態遷移図

3-1-4-4 MUL の設計

ALU から送られてきた 8 ビットの被乗算数と乗算数に対して、符号-絶対値表現で固定小数点表現の乗算を行い、15 ビットの乗算結果を出力する回路を設計する。[7]

3-1-4-4-1 MUL の入出力線

MUL の入出力線図を図 3-5 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　MUL のクロック信号のための線
　　(回路は立ち上がりエッジで動作)
　　START_MUL　：1 ビット
　　この線に'1' が入力されると MUL の
　　状態が初期状態から次の状態に動き出す。
　　MUL_IN_A　　：8 ビット
　　被乗数を入力するための線
　　MUL_IN_B　　：8 ビット
　　乗数を入力するための線

出力線　MUL_OUT　　：15 ビット
　　乗算結果を出力するための線
　　DONE_MUL　：1 ビット
　　MUL が乗算を終え、内部の状態が初期状態になるときに
　　'1' を出力する。

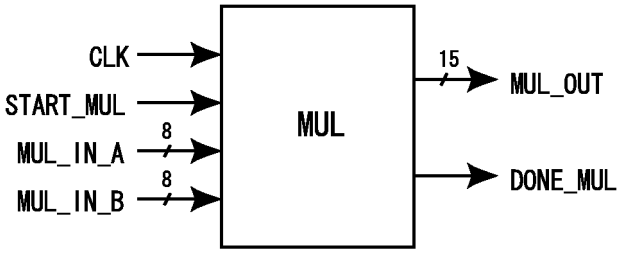

図 3-5 MUL の入出力線図

3-1-4-4-2 MULの動作設計

MULの状態遷移図を図3-6に示し、以下で各状態について説明していく。また、本研究で使用する全てのMULは、被乗数と乗数、乗算後の値のビット数が違うだけで、MULの動作は全て同じ設計をしている。よって後の節で出てくるMULの動作設計については説明を省略する。

- INIT
  MULの初期状態、START_MULに'1'が入力されるまで初期状態を持続する。

- MAKE_OUT_SIGN
  被乗数と乗数の符号を調べ、乗算後の値の符号を作る。

- MAKE_OUT_SUM
  被乗数と乗数の符号ビットを除く値の乗算を行い、結果をOUT_SUMに送る。

- MAKE_MUL_OUT
  OUT_SUMが0の場合0をMULの外に送る。また、それ以外の場合はOUT_SUMにMAKE_OUT_SIGNで作った符号を付けMULの外に送る。

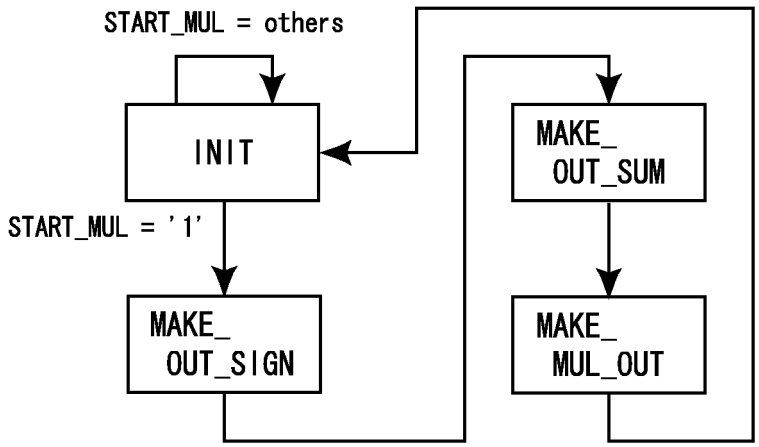

図3-6 MULの状態遷移図

3-1-4-5 ADD の設計

ALU から送られてきた 17 ビットの被加減数と加減数に対して、符号-絶対値表現で固定小数点表現の加減算を行い、18 ビットの加減算結果を出力する回路を設計する。[7]

3-1-4-5-1 ADD の入出力線

ADD の入出力線図を図 3-7 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　ADD のクロック信号のための線
　　(回路は立ち上がりエッジで動作)
　　START_ADD　：1 ビット
　　この線に'1' が入力されると MUL の
　　状態が初期状態から次の状態に動き出す。
　　ADD_IN_A　　：17 ビット
　　被加減数を入力するための線
　　ADD_IN_B　　：17 ビット
　　加減数を入力するための線

出力線　ADD_OUT　　：18 ビット
　　加減算結果を出力するための線
　　DONE_ADD　：1 ビット
　　ADD が加減算を終え、内部の状態が初期状態になるときに
　　'1' を出力する。

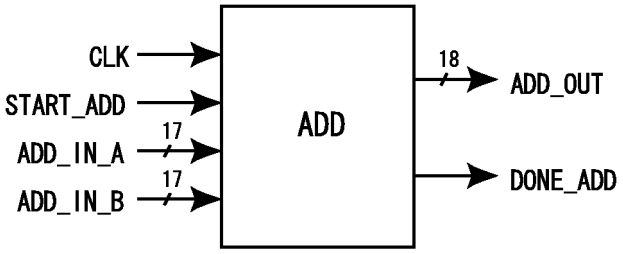

図 3-7 ADD の入出力線図

3-1-4-5-2 ADD の動作設計

ADD の状態遷移図を図 3-8 に示し、以下で各状態について説明していく。また、本研究で使用する全ての ADD は、被加減数と加減数、加減算後の値のビット数が違うだけで、ADD の動作は全て同じ設計をしている。よって後の節で出てくる ADD の動作設計については説明を省略する。

- INIT

  ADD の初期状態、START_ADD に'1' が入力されるまで初期状態を持続する。

- MAKE_NOSIGN_IN_AB

  被加減数と加減数から符号ビットを除き、さらに MSB に'0' を追加した値をそれぞれ NOSIGN_IN_A(被加減数)、NOSIGN_IN_B(加減数) を作る。

- MAKE_RE_NOSIGN_IN_AB

  被加減数と加減数の両方の信号を、正数か負数であるか調べる。もし信号が正数ならそのまま NOSIGN_IN_A を RE_NOSIGN_IN_A に、NOSIGN_IN_B なら RE_NOSIGN_IN_B に送る。また信号が負数なら、1 の補数表現をするために信号を反転して送る。

- MAKE_OUT_SUM

  RE_NOSIGN_IN_A と RE_NOSIGN_IN_B の信号を足す。このとき最上位桁の加算によって生じる桁上げ (オーバーフロー： overflow) のことを考えて、信号には MSB に'0' を付けて足している。[4]

- MAKE_RE_OUT_SUM

  MAKE_OUT_SUM で作った信号が、加算結果の信号か、または減算結果の信号かを調べる。この時もし減算結果の信号で 1 の補数表現であれば符号-絶対値表現に変換する。

- MAKE_ADD_OUT

  RE_OUT_SUM が 0 の場合 0 を ADD の外に送る。また、それ以外の場合は RE_OUT_SUM を ADD の外に送る。

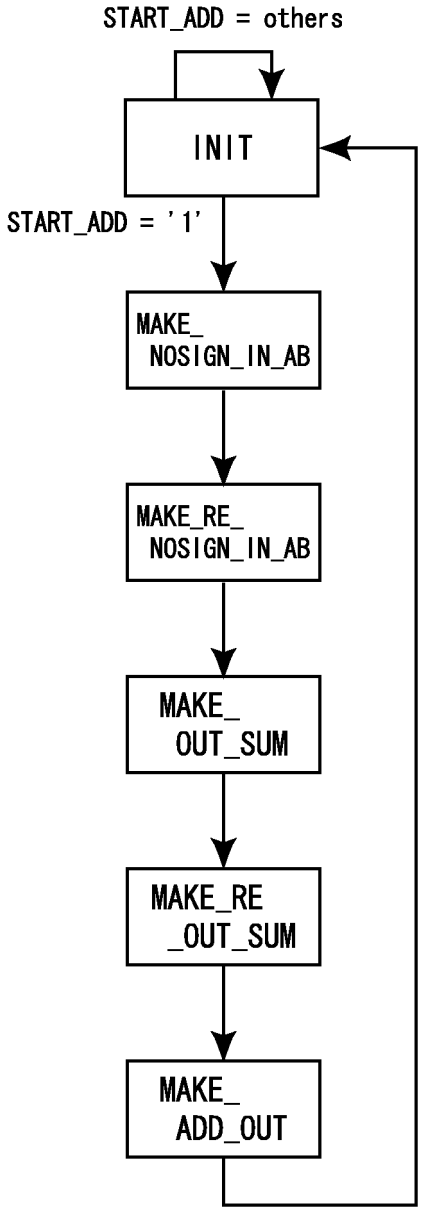

図 3-8 ADD の状態遷移図

3-1-4-6 DIV の設計

ALU から送られてきた 26 ビットの被除数と 15 ビットの除数に対して、符号-絶対値表現で固定小数点表現の除算を行い、13 ビットの除算結果を出力する回路を設計する。[7]

3-1-4-6-1 DIV の入出力線

DIV の入出力線図を図 3-9 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　：1 ビット
DIV のクロック信号のための線
(回路は立ち上がりエッジで動作)

START_DIV　：1 ビット
この線に'1' が入力されると DIV の
状態が初期状態から次の状態に動き出す。

DIV_IN_A　：26 ビット
被除数を入力するための線

DIV_IN_B　：15 ビット
除数を入力するための線

出力線　DIV_OUT　：13 ビット
除算結果を出力するための線

DONE_DIV　：1 ビット
DIV が除算を終え、内部の状態が初期状態になるときに
'1' を出力する。

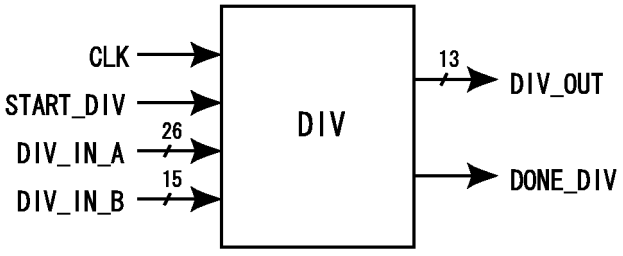

図 3-9 DIV の入出力線図

3-1-4-6-2 DIV の動作設計

DIV の状態遷移図を図 3-10 に示し、以下で各状態について説明していく。

- INIT

DIV の初期状態、START_DIV に'1' が入力されるまで初期状態を持続する。

- MAKE_DD_DASH

符号ビットを除いた乗数を DS に送る。また、被乗数の符号ビットを除き、MSB から DC のビット数ずらした所から LSB までの値を DD_DASH に送る。

- MAKE_PR

DS(符号無し乗数) と大小比較を行うための信号 PR を作る。

- MAKE_PQ

DS と PR の大小比較を行う。このとき DS の方が大きい場合は、DS の MSB に'0' を追加して RE_PR に送り、PQ の N ビット目には'0' を送る。また、DS が PR 以下の場合は、PR を被減数、DS を減数にして減算し RE_PR に送り、PQ の N ビット目には'1' を送る。

- MAKE_RE02_PR

RE_PR 信号にまだ計算していない被除数を 1 ビット左シフトした後、LSB に結合し RE02_PR に送る。

- IF_N

被除数の LSB まで計算したか調べる。まだ全て計算していなければ、状態を N_MINUS に移し、そうでなければ状態を OUT_PQ に移す。

- N_MINUS

PQ のビット数を表すカウンタ N を 1 減らす。

- OUT_PQ

PQ に符号ビットを付けて DIV の外に出力した後、状態を INIT に移す。

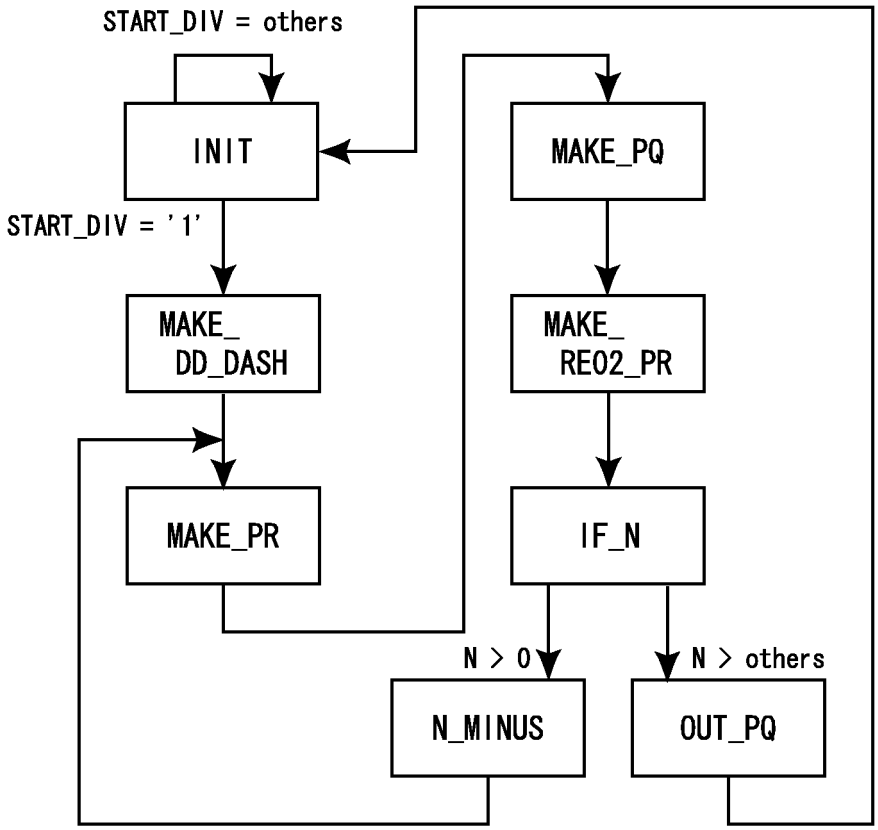

図 3-10 DIV の状態遷移図

3-1-5 DCT 信号用の RAM の設計

13 ビットの DCT 信号を 1 つの束とし、それを 64 個読み出し/書き込みを行える RAM を設計する。また、RAM は 6 個 (DCT_k_RAM : $k = 1, 2, \cdots, 6$) 設計する。[8]

3-1-5-1 DCT 信号用の RAM の入出力線

DCT 信号用の RAM の入出力線図を図 3-11 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　DCT 信号用の RAM のクロック信号のための線
　　(回路は立ち上がりエッジで動作)

　　WP_DCTZ_k　：1 ビット
　　この線に'1' が入力されると、AD_DCTZ_k が示している
　　RAM のアドレスに DCT 信号を書き込む。

　　RP_DCTZ_k　：1 ビット
　　この線に'1' が入力されると、AD_DCTZ_k が示している
　　RAM のアドレスの信号を外部に読み出す。

　　AD_DCTZ_k　：6 ビット
　　13 ビットの信号を読み出し/書き込みを行うための
　　RAM のアドレスを示すための線

　　IN_DCTZ_k　：13 ビット
　　DCT 信号を RAM に書き込むための線

出力線　OUT_DCTZ_k　：13 ビット
　　RAM に記憶している DCT 信号を外部に読み出すための線

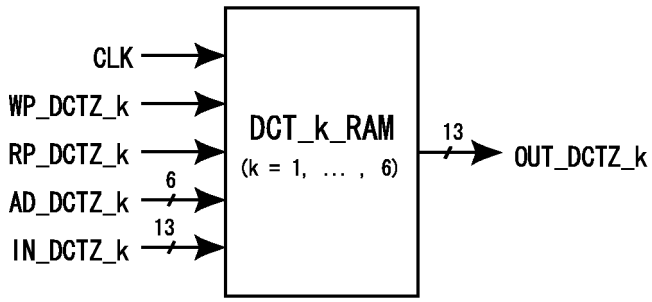

図 3-11 DCT 信号用の RAM の入出力線図

3-1-6 量子化信号用の RAM の設計

13 ビットの量子化信号を 1 つの束とし、それを 64 個読み出し/書き込みを行える RAM を設計する。また RAM は 6 個 (QDCT_k_RAM : $k = 1, 2, \cdots, 6$) 設計する。[8]

3-1-6-1 量子化信号用の RAM の入出力線

量子化信号用の RAM の入出力線図を図 3-12 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　：1 ビット
量子化信号用の RAM のクロック信号のための線
(回路は立ち上がりエッジで動作)

WP_QDCTZ_k　：1 ビット
この線に'1' が入力されると、AD_QDCTZ_k が示している RAM のアドレスに DCT 信号を書き込む。

RP_QDCTZ_k　：1 ビット
この線に'1' が入力されると、AD_QDCTZ_k が示している RAM のアドレスの信号を外部に読み出す。

AD_QDCTZ_k　：6 ビット
13 ビットの信号を読み出し/書き込みを行うための RAM のアドレスを示すための線

IN_QDCTZ_k　：13 ビット
量子化信号を RAM に書き込むための線

出力線　OUT_QDCTZ_k　：13 ビット
RAM に記憶している量子化信号を外部に読み出すための線

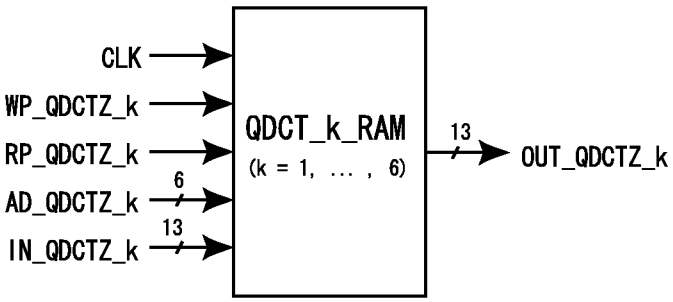

図 3-12 量子化信号用の RAM の入出力線図

3-2 逆量子化回路の設計

量子化信号を、輝度Y成分と色差UV成分に、それぞれ別々の量子化テーブルを用いて、逆量子化を行いDCT信号を出力する回路を設計する。[1]

3-2-1 全体の概要

逆量子化回路のデータパスを図3-13に示す。逆量子化回路(DeQuantization)には、量子化信号を読み出し/書き込みを行う6個のRAM(QDCT_k_RAM : $k = 1, 2, \cdots, 6$)と、DCT信号を読み出し/書き込みを行う6個のRAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$)がある。また、量子化信号を出力するための演算はALUで行う。ALUの説明を3-2-4節で詳しく行う。また、量子化信号用のRAMとDCT信号用のRAMは、量子化回路で使用した物と全く同じ作りなので、説明を省略する。

3-2-2 入出力線

入出力線の各機能およびその詳細を次に示す。

入力線　CLK　：1ビット
逆量子化回路のクロック信号のための線
(回路は立ち上がりエッジで動作)

RESET　：1ビット
逆量子化回路のリセット信号のための線
(回路では'1'を入力すると強制リセット)

START_DEQUANTIZATION　：1ビット
この線に'1'が入力されると逆量子化回路の
状態が初期状態から次の状態に動き出す。

SCALE　：8ビット(整数4ビット、小数4ビット)
0から7までの量子化スケールファクタの値が入る。

QDCTZ_1,QDCTZ_2　：13ビット
量子化信号を入力するための線(輝度Y成分)

QDCTZ_3,QDCTZ_4　：13ビット
量子化信号を入力するための線(輝度Y成分)

QDCTZ_5,QDCTZ_6　：13ビット
量子化信号を入力するための線(色差UV成分)

出力線　DCTZ_1,DCTZ_2　：13 ビット
DCT 信号を出力するための線 (輝度 Y 成分)

DCTZ_3,DCTZ_4　：13 ビット
DCT 信号を出力するための線 (輝度 Y 成分)

DCTZ_5,DCTZ_6　：13 ビット
DCT 信号を出力するための線 (色差 UV 成分)

START_QDCTZ_IN　：1 ビット
この線が'1' の信号を出したとき量子化信号の
入力許可を許可する。

DONE_OUT_DCTZ_K　：1 ビット
DCT 信号が出力される間'1' を出力する。

DONE_DEQUANTIZATION　：1 ビット
逆量子化回路が全ての DCT 信号を出し終えると
'1' を出力する。

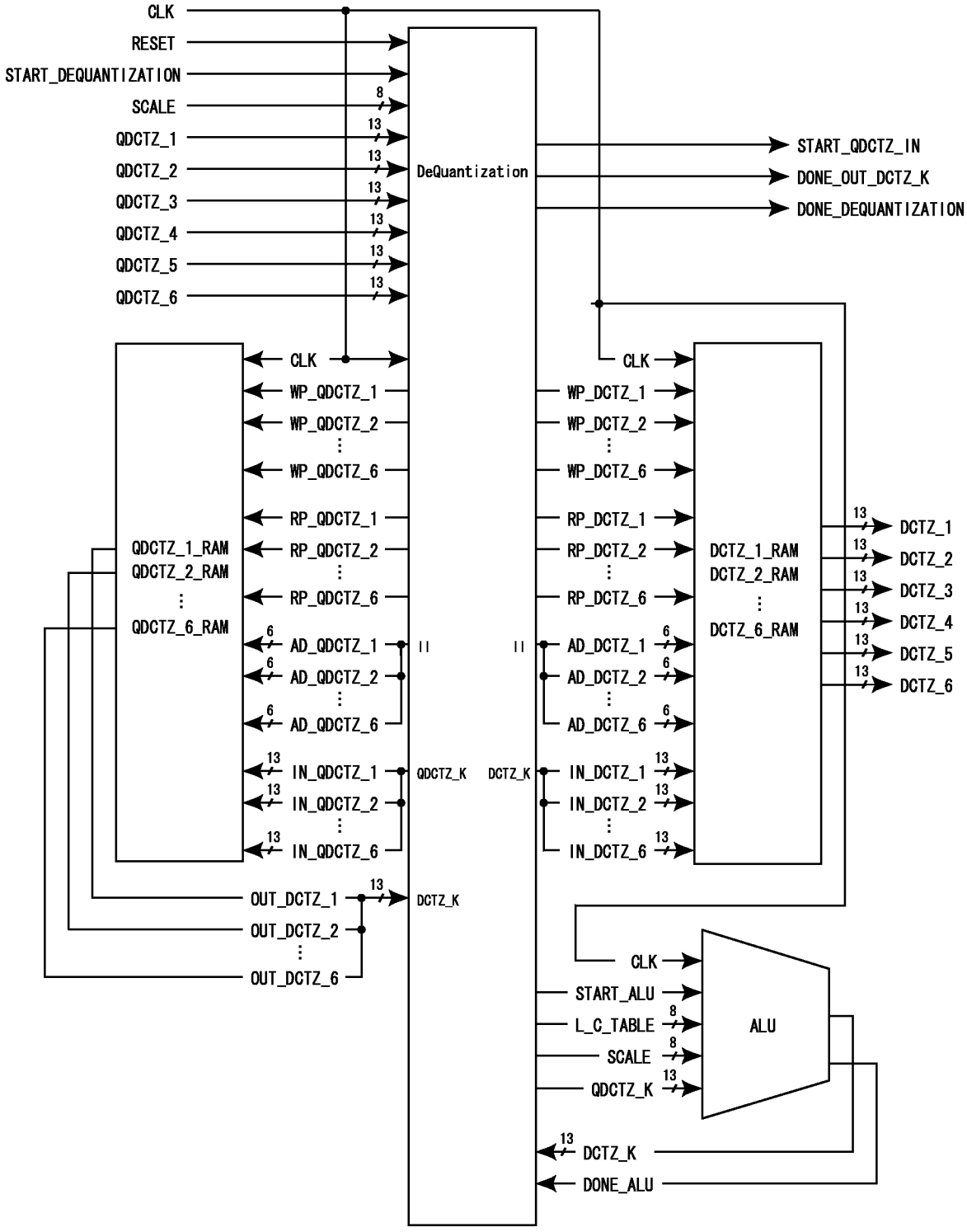

図 3-13 逆量子化回路のデータパス

3-2-3 動作設計

逆量子化回路の状態遷移図を図 3-14 に示す。この回路は全ての状態を、大きく分けて 3 つの役割に分けることが出来る。まず始めに INIT から II_QDCTZ_RAM_UP までの状態で、入力されてくる量子化信号を量子化信号用の RAM(QDCT_k_RAM : $k = 1, 2, \cdots, 6$) に一時記憶していく。次に、K_VALUE から K_UP までの間で、量子化信号を DCT 信号に変換する演算を行っている。最後に、DCTZ_OUT_II_VALUE から DCTZ_OUT_II_UP までの状態で、DCT 信号を逆量子化回路の外に出力している。このことをふまえて、以下で各状態について説明していく。

- INIT

  逆量子化回路の初期状態、START_DEQUANTIZATION に'1' が入力されるまで初期状態を持続する。このとき内部カウンタ (K と II) に 0 を入力し初期化する。

- MAKE_QDCTZ_RAM

  カウンタ II(0 から 63 までの値を出力するカウンタ) が示すアドレスに、6 個の量子化信号を各 RAM(QDCT_k_RAM : $k = 1, 2, \cdots, 6$) に記憶する。

- IF_QDCTZ_RAM

  6 個の量子化信号用の RAM(QDCT_k_RAM : $k = 1, 2, \cdots, 6$) に、それぞれ 8×8 画素分の量子化信号が送られたか調べる。

- II_QDCTZ_RAM_UP

  カウンタ II の値を 1 増やし、6 個の量子化信号用の RAM に新しいアドレスを示す。

- K_VALUE

  カウンタ K(1 から 6 までの値を出力するカウンタ) に 0 を入力し初期化する。

- II_VALUE

  カウンタ II に 0 を入力し初期化する。

- MAKE_ALU

  START_ALU に'1' を入力し、ALU の状態を初期状態から変化させる。ALU から演算終了を知らせる DONE_ALU に'1' が送られるまで逆量子化回路の状態を待

機させ、ALU の状態を変化させる。

• MAKE_DCTZ_K
　逆量子化した信号を、6 個の DCT 信号用の RAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$) のうち、カウンタ K が示す RAM に、カウンタ II が示すアドレスに記憶する。

• IF_II
　8×8 画素分の量子化信号を、全て逆量子化することが出来たか調べる。

• II_UP
　カウンタ II の値を 1 増やし、カウンタ K が示す量子化信号用の RAM の新しいアドレスをカウンタ II が示す。

• IF_K
　6 個の 8×8 画素ブロックを、全て逆量子化することが出来たか調べる。

• K_UP
　カウンタ K の値を 1 増やし、6 個ある DCT 信号用の RAM の中で、今示してある RAM と違う RAM を示す。

• DCTZ_OUT_II_VALUE
　RAM に記憶した DCT 信号を逆量子化回路の外に出力するため、カウンタ II に 0 を入れて初期化する。

• DCTZ_OUT
　6 個の DCT 信号用の RAM のうち、カウンタ II が示すアドレスの DCT 信号を、逆量子化回路の外に出力する。

• IF_DCTZ_OUT_II
　8×8 画素分の DCT 信号を、全て逆量子化回路の外に出力したか調べる。

• DCTZ_OUT_II_UP
　カウンタ II の値を 1 増やし、6 個の DCT 信号用の RAM の新しいアドレスをカウンタ II が示す。

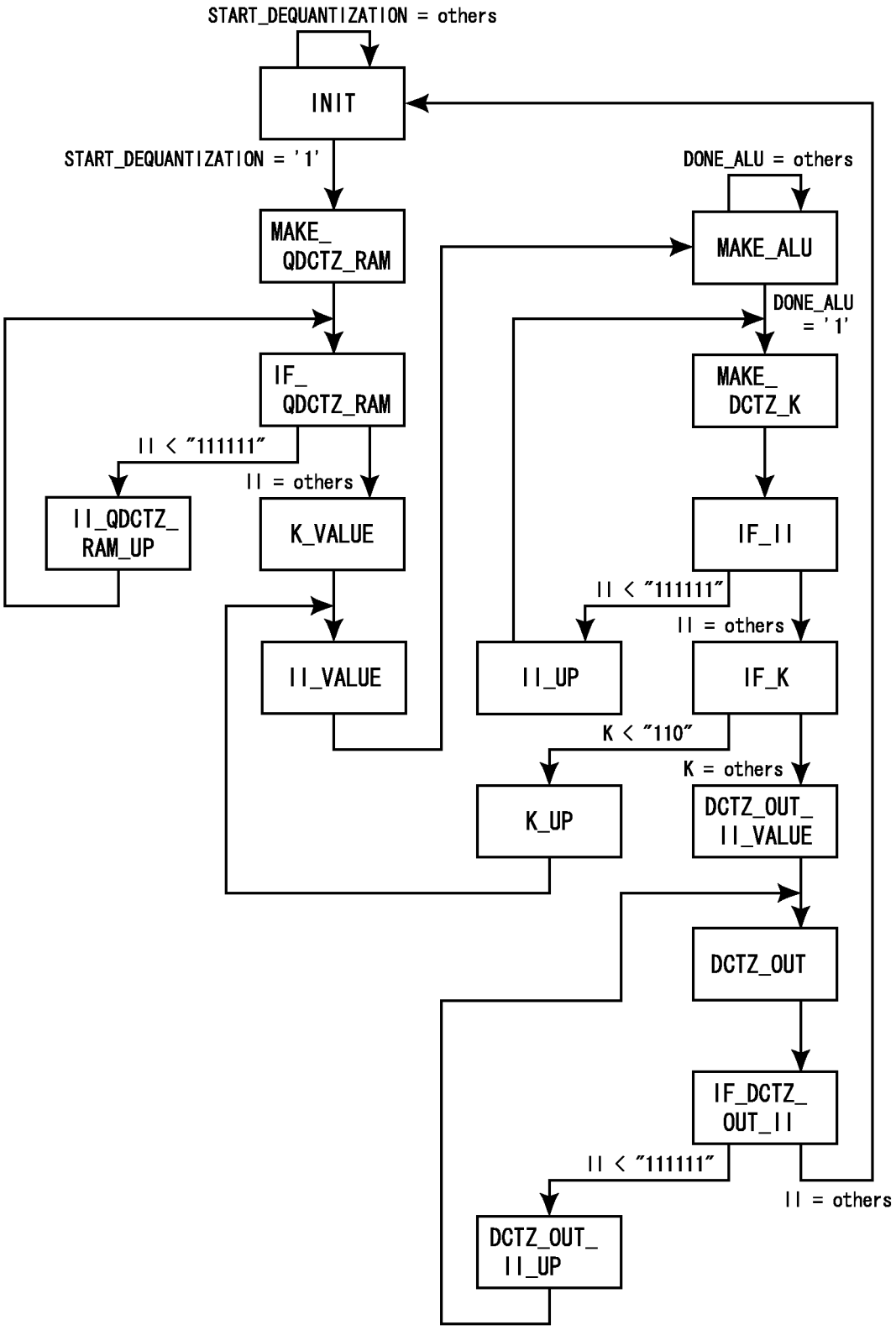

図 3-14 逆量子化回路の状態遷移図

3-2-4 ALU の設計

量子化信号を DCT 信号に変換するための算術演算装置を設計する。[7]

3-2-4-1 ALU の概要

ALU のデータパスを図 3-15 に示す。ALU は符号-絶対値表現で固定小数点表現の小数点数の乗算を行う MUL で構成されている。回路規模をなるべく小さくするため MUL は 1 個しか設置しない。逆量子化するたに必要な演算は ALU でこれら装置を制御することで実現する。MUL の説明は 3-2-4-4 節で行う。

3-2-4-2 ALU の入出力線

ALU の入出力線の各機能およびその詳細を次に示す。

入力線 CLK : 1 ビット
量子化回路のクロック信号のための線
(回路は立ち上がりエッジで動作)

START_ALU : 1 ビット
この線に'1' が入力されると ALU の
状態が初期状態から次の状態に動き出す。

L_C_TABLE : 8 ビット
量子化テーブルの値が入る。
'1' を出力する。

SCALE : 8 ビット (整数 4 ビット、小数 4 ビット)
0 から 7 までの量子化スケールファクタの値が入る。

QDCTZ_K : 13 ビット
量子化信号を入力するための線

出力線 DCTZ_K : 13 ビット
DCT 信号を出力するための線

DONE_ALU : 1 ビット
ALU が演算を終え、内部の状態が初期状態になるときに
'1' を出力する。

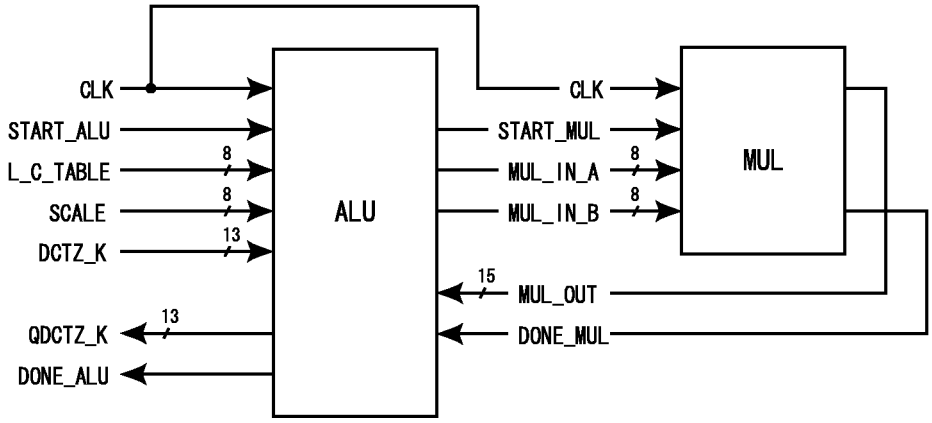

図 3-15 ALU のデータパス

3-2-4-3 ALU の動作設計

ALU の状態遷移図を図 3-16 に示し、以下で各状態について説明していく。

- INIT
  ALU の初期状態、START_ALU に'1' が入力されるまで初期状態を持続する。

- MAKE_STEPSIZE
  量子化テーブルと量子化スケールファクタを掛け、量子化ステップサイズを作る。

- MAKE_DCZ_K
  量子化ステップサイズと量子化信号を掛ける。

- OUT_DCTZ_K
  ALU で作成した DCT 信号を逆量子化回路に渡す。

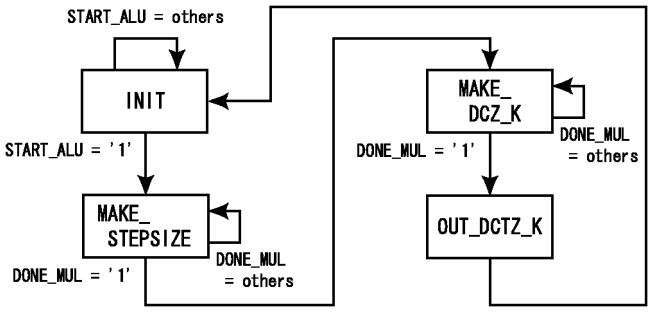

図 3-16 ALU の状態遷移図

3-2-4-4 MUL の設計

ALU から送られてきた 13 ビットの被乗算数と乗算数に対して、符号-絶対値表現で固定小数点表現の乗算を行い、25 ビットの乗算結果を出力する回路を設計する。[7]

3-2-4-4-1 MUL の入出力線

MUL の入出力線図を図 3-17 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　MUL のクロック信号のための線
　　(回路は立ち上がりエッジで動作)
　　START_MUL　：1 ビット
　　この線に'1' が入力されると MUL の
　　状態が初期状態から次の状態に動き出す。
　　MUL_IN_A　　：13 ビット
　　被乗数を入力するための線
　　MUL_IN_B　　：13 ビット
　　乗数を入力するための線

出力線　MUL_OUT　　：25 ビット
　　乗算結果を出力するための線
　　DONE_MUL　：1 ビット
　　MUL が乗算を終え、内部の状態が初期状態になるときに
　　'1' を出力する。

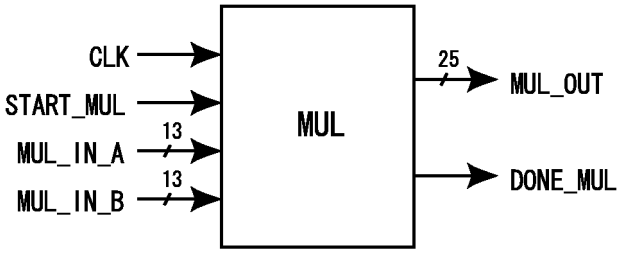

図 3-17 MUL の入出力線図

3-3 DCT 変換回路の設計

9 ビットをの YUV 信号を、13 ビットの DCT 信号に変換するして、外部に出力する回路を設計する。[1]

3-3-1 全体の概要

DCT 変換回路のデータパスを図 3-18 に示す。DCT 変換回路 (ForwardDct) には、9 ビットの YUV 信号を読み出し/書き込みを行う 6 個の RAM(YUV_k_RAM : $k = 1, 2, \cdots, 6$) と、13 ビットの DCT 信号を読み出し/書き込みを行う 6 個の RAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$) がある。また、YUV 信号を DCT に変換するための演算は ALU で行う。YUV 信号用の RAM を 3-3-5 節で、ALU の説明を 3-3-4 節で行う。また、DCT 信号用の RAM は、量子化回路で使用した物と全く同じ作りなので、説明を省略する。

3-3-2 入出力線

入出力線の各機能およびその詳細を次に示す。

入力線	CLK	:	1 ビット DCT 変換回路のクロック信号のための線 (回路は立ち上がりエッジで動作)
	RESET	:	1 ビット DCT 変換回路のリセット信号のための線 (回路では'1' を入力すると強制リセット)
	START_FORWARDDCT	:	1 ビット この線に'1' が入力されると DCT 変換回路の状態が初期状態から次の状態に動き出す。
	YUV_1,YUV_2	:	9 ビット YUV 信号を入力するための線 (輝度 Y 成分)
	YUV_3,YUV_4	:	9 ビット YUV 信号を入力するための線 (輝度 Y 成分)
	YUV_5,YUV_6	:	9 ビット YUV 信号を入力するための線 (色差 UV 成分)

出力線　DCTZ_1,DCTZ_2　：13 ビット
DCT 信号を出力するための線 (輝度 Y 成分)

DCTZ_3,DCTZ_4　：13 ビット
DCT 信号を出力するための線 (輝度 Y 成分)

DCTZ_5,DCTZ_6　：13 ビット
DCT 信号を出力するための線 (色差 UV 成分)

START_YUV_IN　：1 ビット
この線が’1’ の信号を出したとき YUV 信号の
入力許可を許可する。

DONE_OUT_DCTZ_K　：1 ビット
DCT 信号が出力される間’1’ を出力する。

DONE_FORWARDDCT　：1 ビット
DCT 変換回路が全ての DCT 信号を出し終えると
’1’ を出力する。

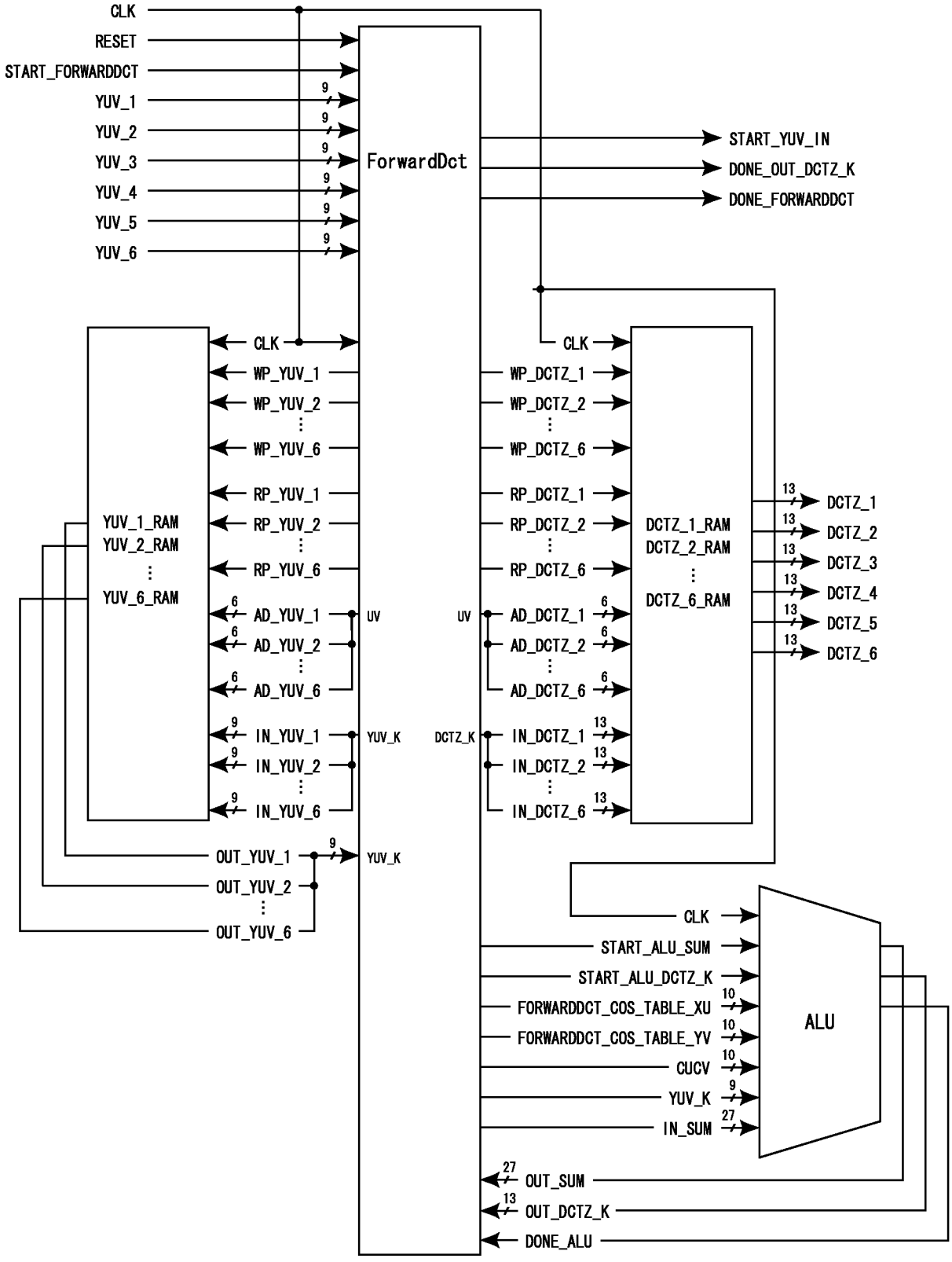

図 3-18 DCT 変換回路のデータパス

3-3-3 動作設計

DCT 変換回路の状態遷移図を図 3-19 に示す。この回路は全ての状態を、大きく分けて 3 つの役割に分けることが出来る。まず始めに INIT から II_YUV_RAM_UP までの状態で、入力されてくる YUV 信号を YUV 信号用の RAM(YUV_k_RAM : $k = 1, 2, \cdots, 6$) に一時記憶していく。次に、K_VALUE から K_UP までの間で、YUV 信号を DCT 信号に変換する演算を行っている。最後に、DCTZ_OUT_II_VALUE から DCTZ_OUT_II_UP までの状態で、DCT 信号を DCT 変換回路の外に出力している。このことをふまえて、以下で各状態について説明していく。

- INIT

  DCT 変換回路の初期状態、START_FORWARDDCT に'1' が入力されるまで初期状態を持続する。このとき内部カウンタ (K,UV,II) に 0 を入力し初期化する。

- MAKE_YUV_RAM

  カウンタ II(0 から 63 までの値を出力するカウンタ) が示すアドレスに、6 個の YUV 信号を各 RAM(YUV_k_RAM : $k = 1, 2, \cdots, 6$) に記憶する。

- IF_YUV_RAM

  6 個の YUV 信号用の RAM(YUV_k_RAM : $k = 1, 2, \cdots, 6$) に、それぞれ 8×8 画素分の YUV 信号が送られたか調べる。

- II_YUV_RAM_UP

  カウンタ II の値を 1 増やし 6 個の YUV 信号用の RAM に新しいアドレスを示す。

- K_VALUE

  カウンタ K(1 から 6 までの値を出力するカウンタ) に 0 を入力し初期化する。

- UV_VALUE

  カウンタ UV に 0 を入力し初期化する。

- II_VALUE

  カウンタ II に 0 を入力し初期化する。

- MAKE_ALU_SUM

  START_ALU_SUM に'1' を入力し、ALU の状態を初期状態から変化させる。ALU から演算終了を知らせる DONE_ALU に'1' が送られるまで DCT 変換回路の状態を

待機させ、ALU の状態を変化させる。

- IF_II

  8×8 画素分の YUV 信号を、全て MAKE_ALU_SUM で処理することが出来たか調べる。

- II_UP

  カウンタ II の値を 1 増やし、カウンタ K が示す YUV 信号用の RAM の新しいアドレスをカウンタ II が示す。

- MAKE_ALU_DCTZ_K

  DCT 変換した信号を、6 個の DCT 信号用の RAM(DCT_k_RAM : $k = 1, 2, \cdots, 6$) のうち、カウンタ K が示す RAM に、カウンタ UV が示すアドレスに記憶する。

- IF_UV

  8×8 画素ブロックを、全て DCT 変換することが出来たか調べる。

- IF_K

  6 個の 8×8 画素ブロックを、全て DCT 変換することが出来たか調べる。

- K_UP

  カウンタ K の値を 1 増やし、6 個ある DCT 信号用の RAM の中で、今示してある RAM と違う RAM を示す。

- DCTZ_OUT_UV_VALUE

  RAM に記憶した DCT 信号を DCT 変換回路の外に出力するため、カウンタ UV に 0 を入れて初期化する。

- DCTZ_OUT

  6 個の DCT 信号用の RAM のうち、カウンタ UV が示すアドレスの DCT 信号を、DCT 変換回路の外に出力する。

- IF_DCTZ_OUT_UV

  8×8 画素分の DCT 信号を、全て DCT 変換回路の外に出力したか調べる。

- DCTZ_OUT_UV_UP

  カウンタ UV の値を 1 増やし、6 個の DCT 信号用の RAM の新しいアドレスをカウンタ UV が示す。

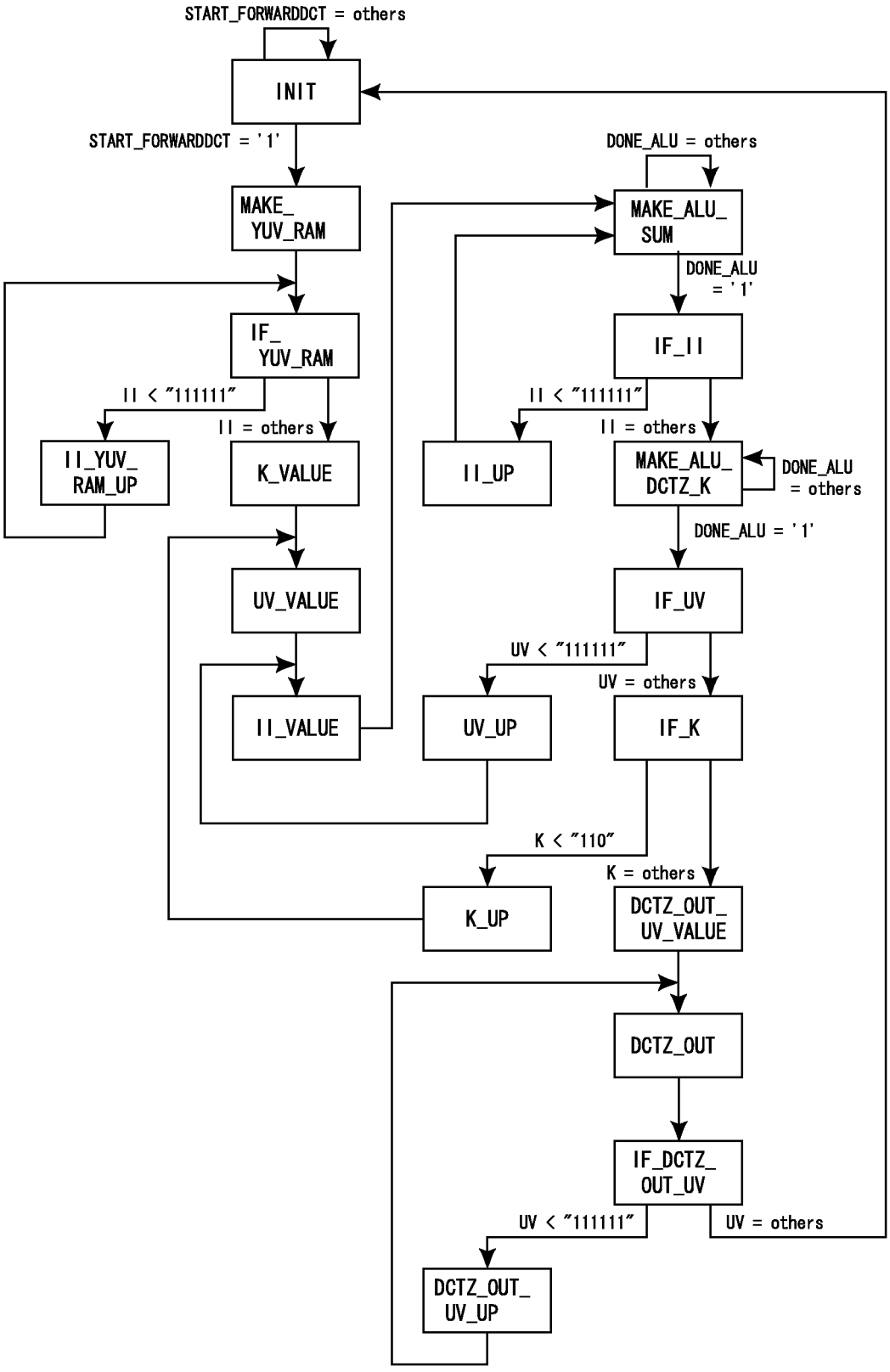

図 3-19 DCT 変換回路の状態遷移図

3-3-4 ALU の設計

YUV 信号を DCT 信号に変換するための算術演算装置を設計する。[7]

3-3-4-1 ALU の概要

ALU のデータパスを図 3-20 に示す。ALU は符号-絶対値表現で固定小数点表現の小数点数の乗算を行う MUL と、加減算を行う ADD で構成されている。回路規模をなるべく小さくするため MUL と ADD は 1 個しか設置しない。DCT 変換するたに必要な演算は ALU でこれら装置を制御することで実現する。MUL の説明は 3-3-4-4 節で行う。

3-3-4-2 ALU の入出力線

ALU の入出力線の各機能およびその詳細を次に示す。

入力線　CLK　：1 ビット
DCT 変換回路のクロック信号のための線
(回路は立ち上がりエッジで動作)

START_ALU_SUM　：1 ビット
この線に'1' が入力されると DCT 信号の元となる OUT_SUM を作るため ALU の状態が初期状態から次の状態に動き出す。

START_ALU_DCTZ_K　：1 ビット
この線に'1' が入力されると DCT 信号を作るため ALU の状態が初期状態から
次の状態に動き出す。

FORWARDDCT_COS_TABLE_XU　：10 ビット
$\cos\frac{(2x+1)u\pi}{16}$ を計算した値が入る。

FORWARDDCT_COS_TABLE_YV　：10 ビット
$\cos\frac{(2y+1)v\pi}{16}$ を計算した値が入る。

CUCV　　：10 ビット
　　空間周波数を入力するための線
YUV_K　　：9 ビット
　　YUV 信号を入力するための線
IN_SUM　　：27 ビット
　　DCT 信号の元となる信号を
　　入力するための線

出力線　OUT_SUM　　：27 ビット
　　DCT 信号の元となる信号を ALU の
　　外に出力するための線
OUT_DCTZ_K　：13 ビット
　　DCT 信号を ALU の外に出力するための線
DONE_ALU　　：1 ビット
　　ALU が演算を終え、内部の状態が初期状態
　　になるときに'1' を出力する。

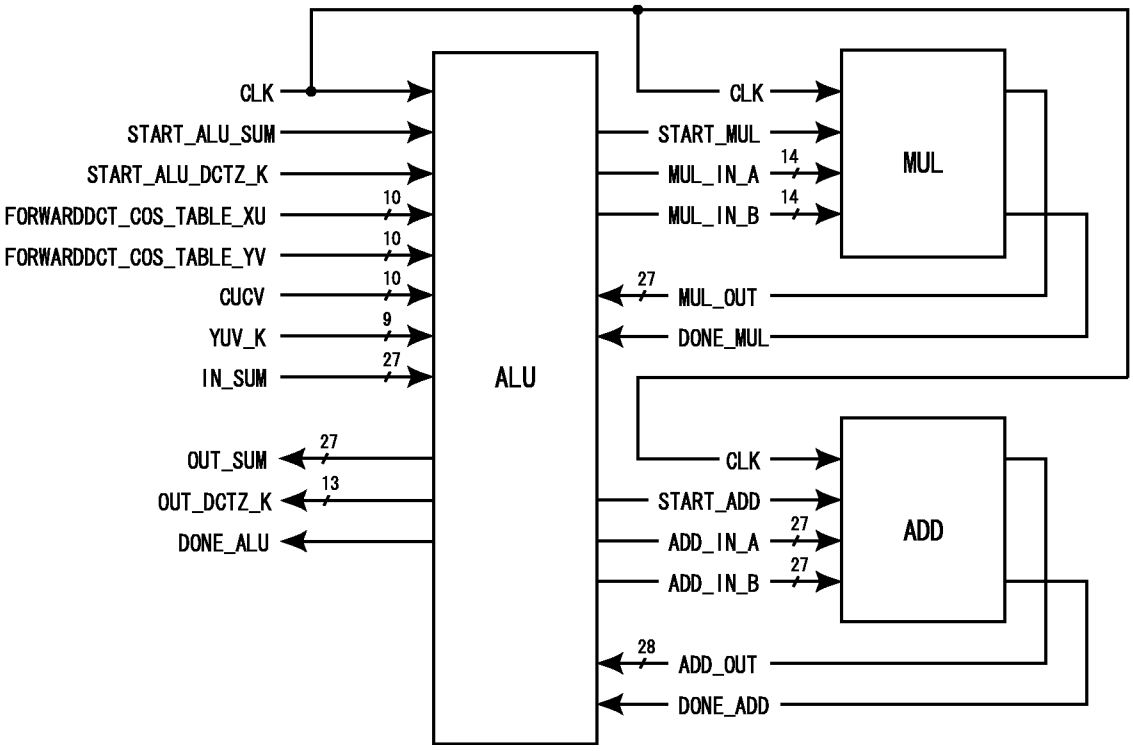

図 3-20 ALU のデータパス

3-3-4-3 ALU の動作設計

ALU の状態遷移図を図 3-21 に示し、以下で各状態について説明していく。

- INIT

  ALU の初期状態、START_ALU_SUM または START_ALU_DCTZ_K に'1' が入力されるまで初期状態を持続する。

- IF_START

  START_ALU_SUM が'1' のときは、ALU は DCT 信号を求めるための信号を作り、ALU は START_ALU_DCTZ_K が'1' のときは、DCT 信号を作る。

- MAKE_FORWARDDCT_COS

  $\cos\dfrac{(2x+1)u\pi}{16}\cos\dfrac{(2y+1)v\pi}{16}$ を計算する。

- MAKE_MUL_YUV

  YUV 信号に $\cos\dfrac{(2x+1)u\pi}{16}\cos\dfrac{(2y+1)v\pi}{16}$ を掛ける。

- MAKE_ADD_SUM

  ALU は DCT 信号を求めるための信号 (SUM) を作る。

- MAKE_OUT_SUM

  MAKE_ADD_SUM までで作った信号を ALU の外部に出力する。

- MAKE_MUL_CUCV

  $\dfrac{C_uC_v}{4}$ による大きさ補正を行う。

- MAKE_DCZ_K

  DCT 信号を作り、ALU の外部に出力する。

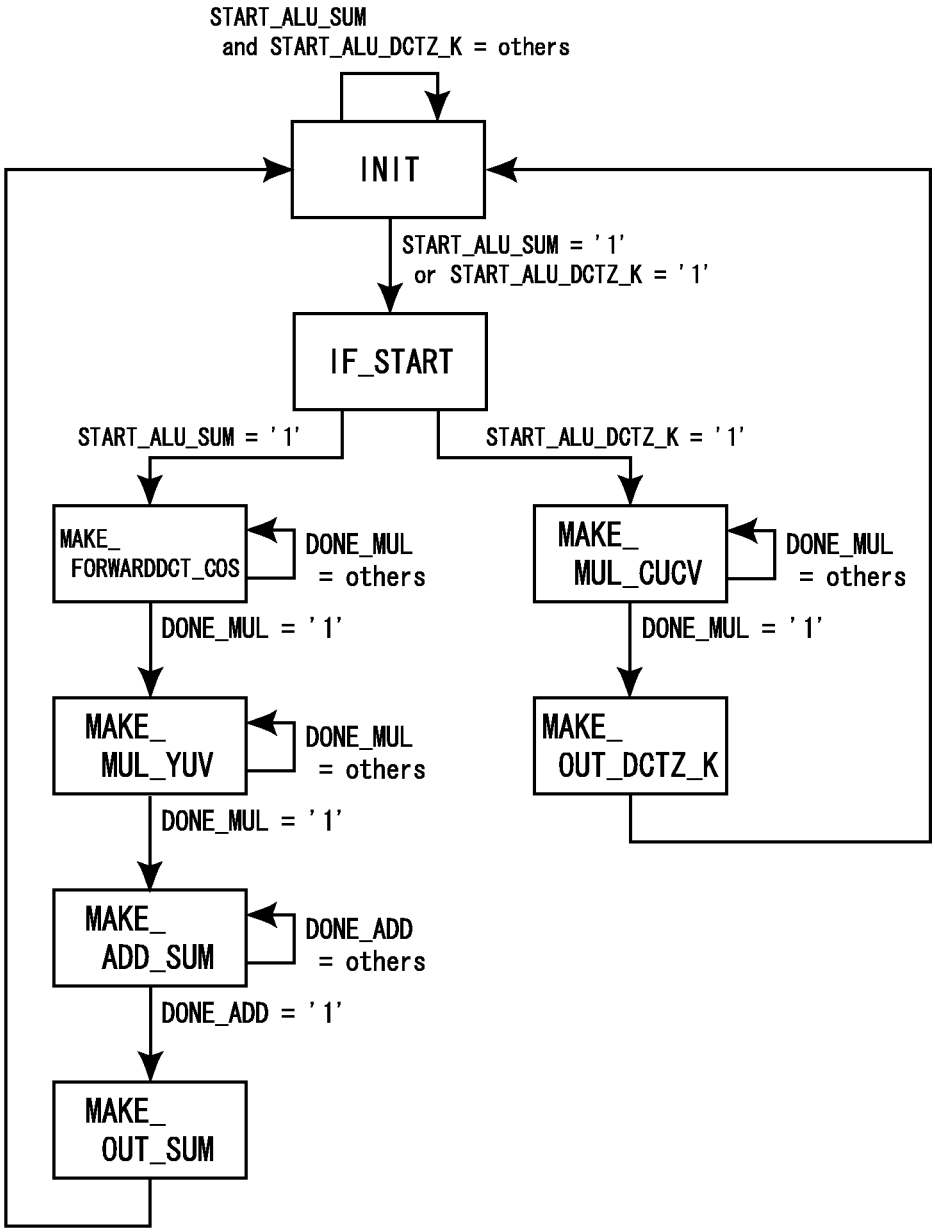

図 3-21 ALU の状態遷移図

3-3-4-4 MUL の設計

ALU から送られてきた 14 ビットの被乗算数と乗算数に対して、符号-絶対値表現で固定小数点表現の乗算を行い、27 ビットの乗算結果を出力する回路を設計する。[7]

3-3-4-4-1 MUL の入出力線

MUL の入出力線図を図 3-22 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　MUL のクロック信号のための線
　　(回路は立ち上がりエッジで動作)
　　START_MUL　：1 ビット
　　この線に'1' が入力されると MUL の
　　状態が初期状態から次の状態に動き出す。
　　MUL_IN_A　　：14 ビット
　　被乗数を入力するための線
　　MUL_IN_B　　：14 ビット
　　乗数を入力するための線

出力線　MUL_OUT　　：27 ビット
　　乗算結果を出力するための線
　　DONE_MUL　：1 ビット
　　MUL が乗算を終え、内部の状態が初期状態になるときに
　　'1' を出力する。

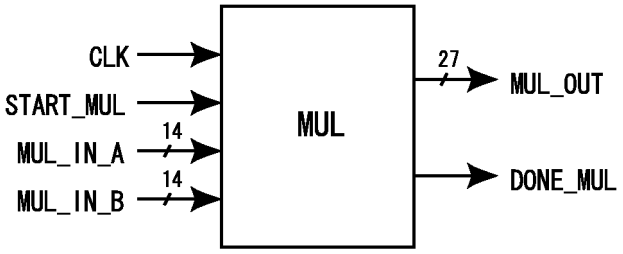

図 3-22 MUL の入出力線図

3-3-4-5 ADD の設計

ALU から送られてきた 27 ビットの被加減数と加減数に対して、符号-絶対値表現で固定小数点表現の加減算を行い、28 ビットの加減算結果を出力する回路を設計する。[7]

3-3-4-5-1 ADD の入出力線

ADD の入出力線図を図 3-23 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　ADD のクロック信号のための線
　　(回路は立ち上がりエッジで動作)
　　START_ADD　：1 ビット
　　この線に'1' が入力されると MUL の
　　状態が初期状態から次の状態に動き出す。
　　ADD_IN_A　　：27 ビット
　　被加減数を入力するための線
　　ADD_IN_B　　：27 ビット
　　加減数を入力するための線

出力線　ADD_OUT　　：28 ビット
　　加減算結果を出力するための線
　　DONE_ADD　：1 ビット
　　ADD が加減算を終え、内部の状態が初期状態になるときに
　　'1' を出力する。

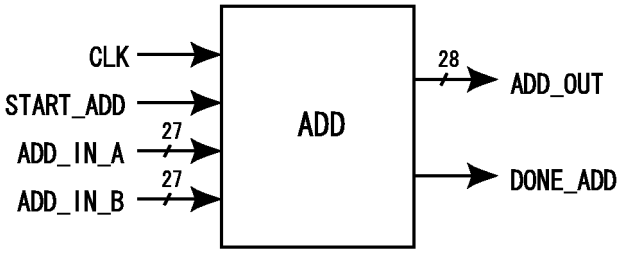

図 3-23 ADD の入出力線図

3-3-5 YUV 信号用の RAM の設計

9 ビットの YUV 信号を 1 つの束とし、それを 64 個読み出し/書き込みを行える RAM を設計する。また、RAM は 6 個 (YUV_k_RAM : $k = 1, 2, \cdots, 6$) 設計する。[8]

3-3-5-1 YUV 信号用の RAM の入出力線

YUV 信号用の RAM の入出力線図を図 3-24 に示し、以下で入出力線の各機能について説明していく。

入力線　CLK　　：1 ビット
　　YUV 信号用の RAM のクロック信号のための線
　　(回路は立ち上がりエッジで動作)

　　WP_YUV_k　：1 ビット
　　この線に'1' が入力されると、AD_YUV_k が示している
　　RAM のアドレスに DCT 信号を書き込む。

　　RP_YUV_k　：1 ビット
　　この線に'1' が入力されると、AD_YUV_k が示している
　　RAM のアドレスの信号を外部に読み出す。

　　AD_YUV_k　：6 ビット
　　13 ビットの信号を読み出し/書き込みを行うための
　　RAM のアドレスを示すための線

　　IN_YUV_k　：9 ビット
　　YUV 信号を RAM に書き込むための線

出力線　OUT_YUV_k　：9 ビット
　　RAM に記憶している YUV 信号を外部に読み出すための線

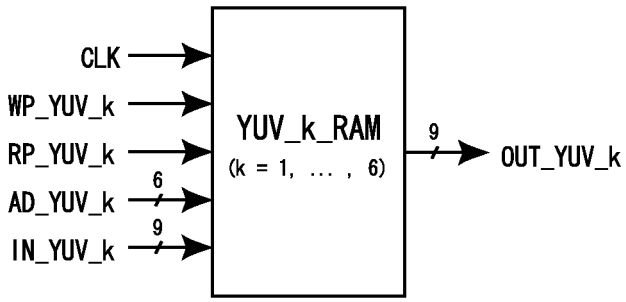

図 3-24 YUV 信号用の RAM の入出力線図

第4章　VHDLの階層とシミュレーション

4-1 VHDLの階層

第3章で説明した回路のVHDLの階層を以下に示す。また、本研究で作成したVHDLの記述は付録で掲載している。

4-1-1 量子化回路のVHDL階層

量子化回路のVHDL階層を図4-1に示す。量子化回路のVHDL階層は大きく分けると、パッケージのL_C_TABLES、メインのQuantization、演算のALU、RAMで構成されている。

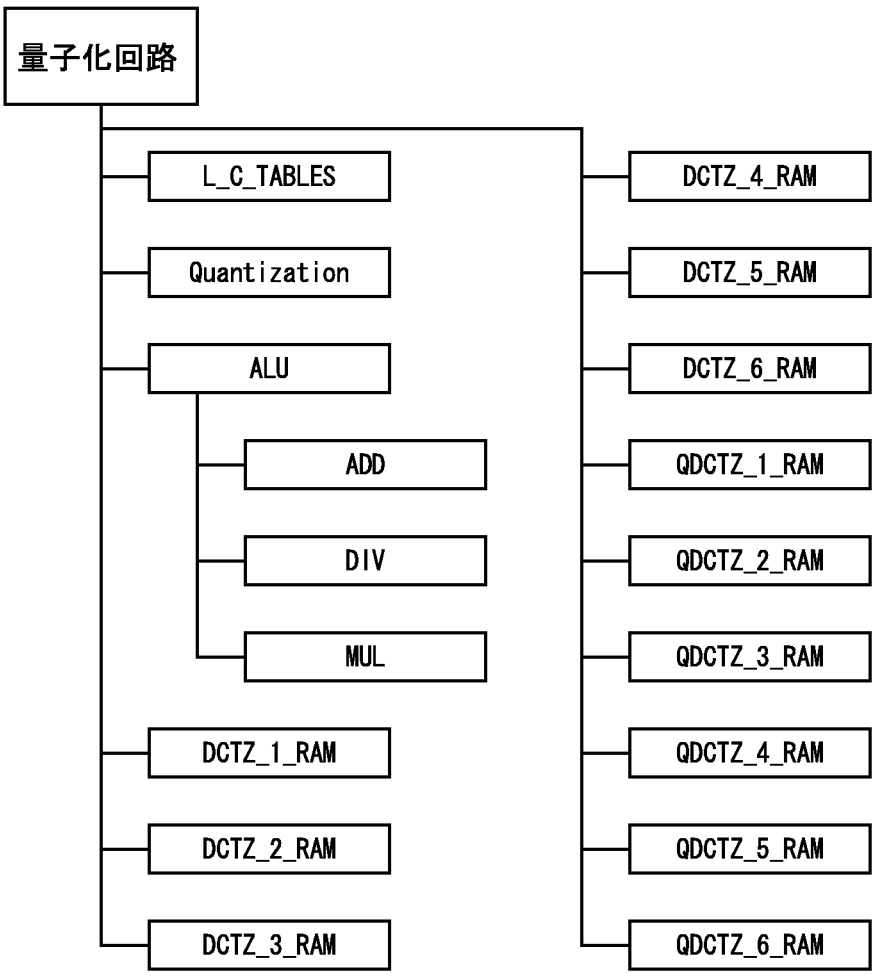

図4-1 量子化回路のVHDL階層

4-1-2 逆量子化回路の VHDL 階層

逆量子化回路の VHDL 階層を図 4-2 に示す。逆量子化回路の VHDL 階層は大きく分けると、パッケージの L_C_TABLES、メインの DeQuantization、演算の ALU、RAM で構成されている。パッケージの L_C_TABLES と RAM は、量子化回路で使用した物と同じ物を使用している。

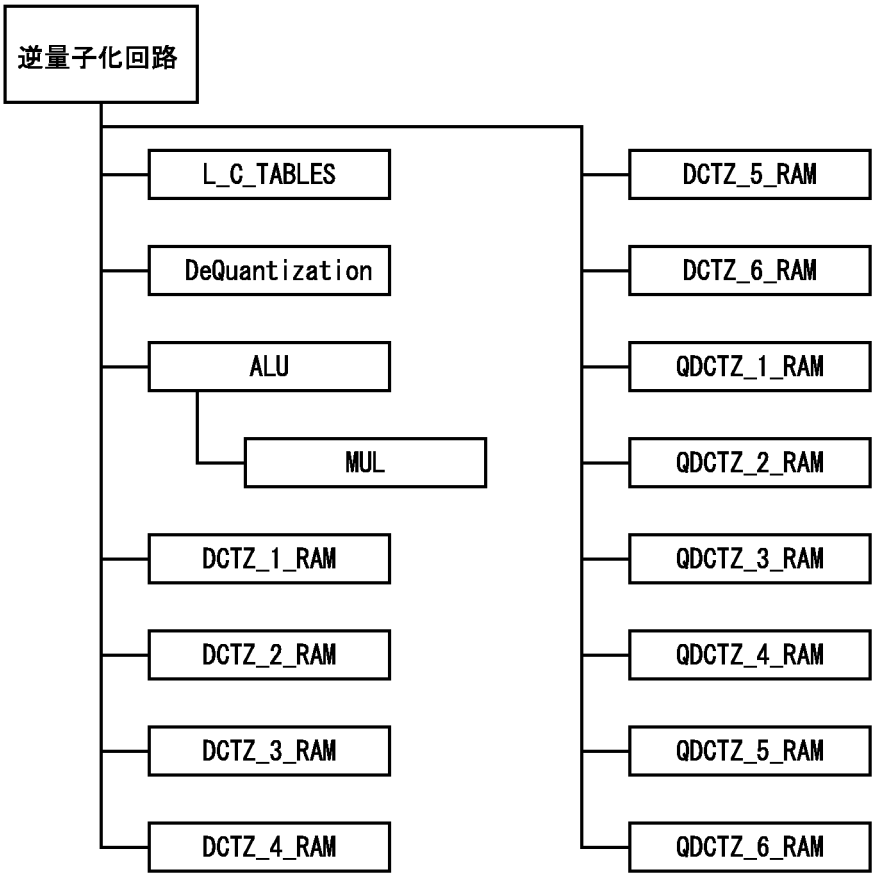

図 4-2 逆量子化回路の VHDL 階層

4-1-3 DCT 変換回路の VHDL 階層

DCT 変換回路の VHDL 階層を図 4-3 に示す。DCT 変換回路の VHDL 階層は大きく分けると、ForwardDct のコンポーネントを集めた COM パッケージ、cos((2 ∗ (x or y)+1)∗(u or v)∗π/16) を計算した値を収録している FORWARDDCT_COS_TABLES パッケージ、メインの ForwardDct、演算の ALU、YUV 信号用 RAM と DCT 信号用 RAM で構成されている。DCT 信号用 RAM は、量子化回路で使用した物と同じ物を使用している。[1]

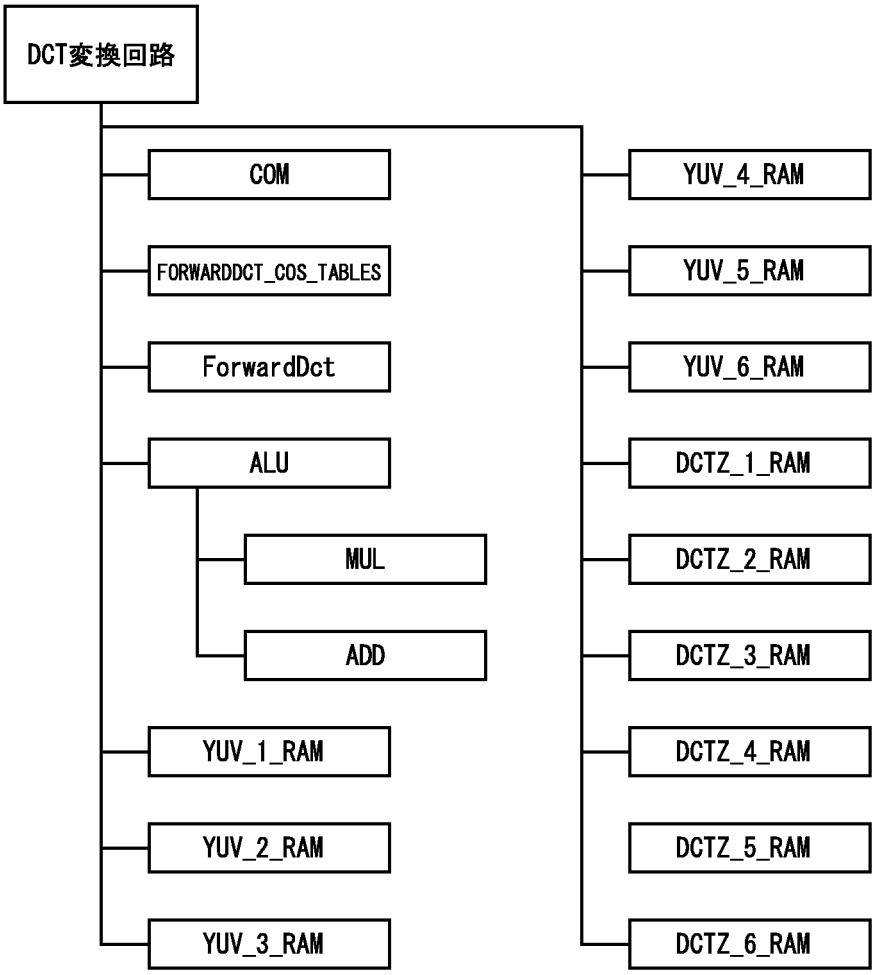

図 4-3 DCT 変換回路の VHDL 階層

4-2 回路のシミュレーション

本研究で作成した各回路のシミュレーションには、VHDL のテスト・パターンのインターフェースとして TEXTIO を使用する。TEXTIO では、各回路に入力するテスト・パターンを外部のファイルから読み出してシミュレーションを行う。そのため、C 言語プログラムでこれらファイルを一度に出力することで、シミュレーション作業の効率化を図る。[4][5]

4-2-1 C 言語プログラムの説明

本研究で作成した C 言語プログラムは、各回路に入力するテスト・パターンと、シミュレーション結果と照合するファイルを作成する。以下では、作成した C 言語プログラムの関数について説明する。また、C 言語の記述は付録として収録する。[1]

- main
  この C 言語プログラムのメインである。全てのプログラムを実行する。

- encode
  エンコードを行うための関数を実行する。

- encode_yuv_k_bit_outfile
  DCT 変換回路のシミュレーションに必要なテスト･パターン (Encode_yuv_k_bit_outfile.txt) を作る。

- encode_dct_k_bit_outfile
  量子化回路のシミュレーションに必要なテスト･パターン (Encode_dct_k_bit_outfile.txt) を作る。

- encode_qdctz_k_bit_outfile
  逆量子化回路のシミュレーションに必要なテスト･パターン (Encode_qdct_k_bit_outfile.txt) を作る。

- decode
  デコードを行うための関数を実行する。

- encode_outfile
  シュミレーション結果と照合するエンコード結果 (Encode_outfile.txt) を出力す

る。

- decode_outfile
  シュミレーション結果と照合するデコード結果 (Decode_outfile.txt) を出力する。

- rgbfileout
  10 進数の RGB の色信号をファイルに書き込む。

- yuvfileout
  10 進数の輝度と色差成分 YUV 信号をファイルに書き込む。

- yuv_kfileout
  輝度 Y 成分を $-128, \cdots, +127$ の範囲に変換した、yuv_k $(k : 1, \cdots, 6)$ の値をファイルに書き込む。

- dct_kfileout
  量子化信号 dct_k $(k : 1, \cdots, 6)$ の値をファイルに書き込む。

- qdctz_kfileout
  逆量子化信号 qdct_k $(k : 1, \cdots, 6)$ の値をファイルに書き込む。

- convertrgbyuv
  RGB 信号を、輝度 Y 成分と、色差 UV 成分に変換する。

- composemcu
  YUV 成分から色差成分の間引きを行い、8×8 画素から構成されるブロックの集合 (MCU) を作る。また、輝度成分を $-128, \cdots, +127$ の範囲に変換する。

- decomposemcu
  MCU と色差成分の間引き率 $4 : 1 : 1$ から輝度と色差成分 YUV を作成する。また、輝度 Y 成分に 128 を加え、元の範囲に戻す。

- convertyuvrgb
  輝度と色差成分 YUV を RGB 信号に変換する。

- forwarddct
  MCU 内に存在する 6 個の 8×8 の YUV 信号に対して 2 次元 DCT 変換を行う。

- quantization

MCU 内に存在する 6 個の 8 × 8 の DCT 信号に対して量子化を行う。

- dequantization

  MCU 内に存在する 6 個の 8 × 8 の量子化信号に対して逆量子化を行う。

- inversedct

  MCU 内に存在する 6 個の 8 × 8 の DCT 信号に対して逆 DCT 変換を行う。

- carrybit

  10 進数の値を符号-絶対値表現の 2 進数に変換する。

4-2-2 シミュレーションのテスト・パターン

以下で述べる各回路のシミュレーションには、図4-4のRGB信号を(rgbdata_01.h)作り、C 言語プログラムを実行して出力された値を、TEXTIO のテスト・パターンとして回路に入力してある。C 言語プログラムの出力ファイル(テキストファイル)は量が多いため、必要な部分を表として抜粋していく。またC 言語プログラムにはRGB 信号の他に、1MCU 内の輝度成分の水平画素数 (horizontalsize = 16)、1MCU 内の輝度成分の垂直画素数 (verticalsize = 16)、量子化スケールファクタ (scale = 1.0) を入力しておく。[1][4][5]

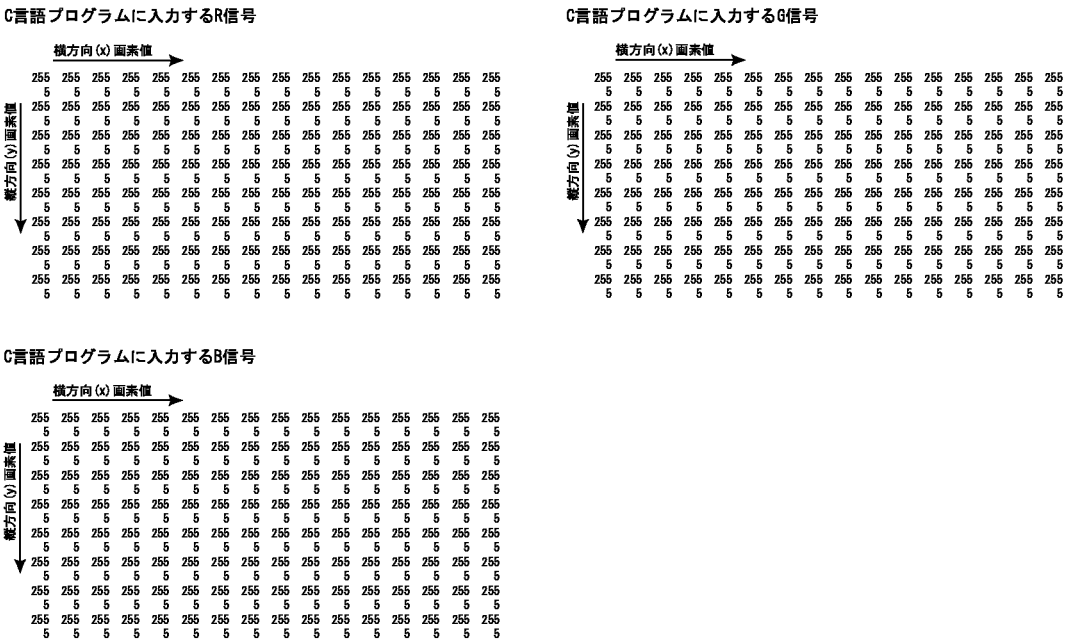

図 4-4 C 言語プログラムに入力する RGB 信号

4-2-3 量子化回路のシミュレーション

4-2-3-1 量子化回路のシミュレーションの概要

C 言語プログラムから出力された値 (Encode_dct_k_bit_outfile.txt) を TEXTIO のテスト・パターンとして入力する。次に、シミュレーション結果と、C 言語プログラムから出力されたファイル (Encode_outfile.txt) と照合することにより、量子化回路の検証を行う。以下では、図 4-5 に TEXTIO のテスト・パターン (dctz_k ($k = 1, \cdots, 6$)) を 10 進数で示し、照合する値 (qdctz_k ($k = 1, \cdots, 6$)) を 10 進数で図 4-6 に示す。また、量子化回路のシミュレーション記述は、付録として添付してある。[4][5]

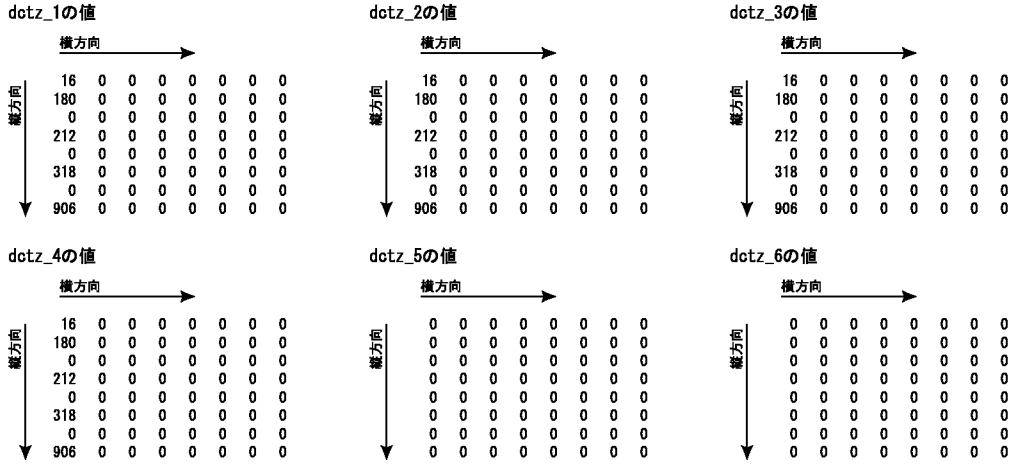

図 4-5 TEXTIO のテスト・パターン (dctz_k ($k = 1, \cdots, 6$))

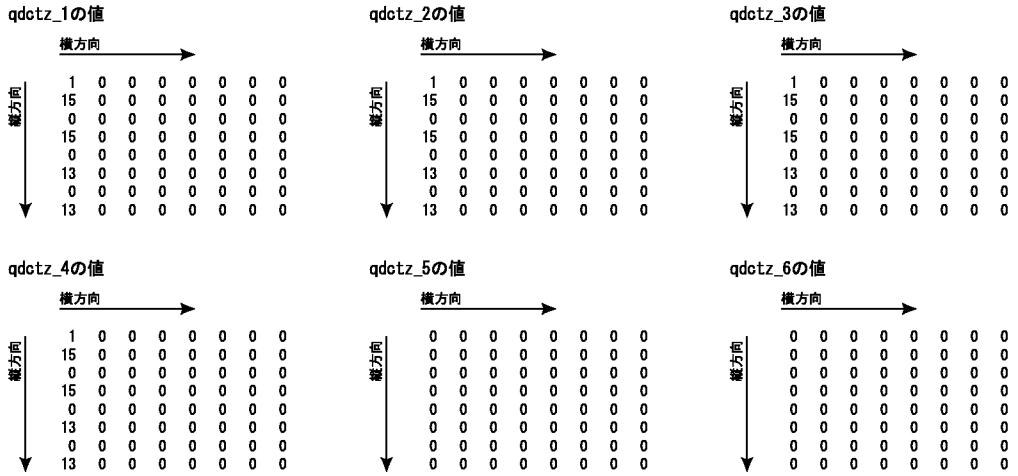

図 4-6 照合する値 (qdctz_k ($k = 1, \cdots, 6$))

4-2-3-2 量子化回路のシミュレーション結果と考察

図 4-7 に、量子化回路のシミュレーション結果 (0ps から 350us まで) を示す。この図はシミュレーションの開始から、量子化回路が動作終了 (done_quantization) を出力するまでを示している。0ps から 350us までの結果では、dctz_k (k = 1;¢¢¢ ; 6) に信号を入力する部分が小さく表示されているため、図 4-8 と図 4-9 に拡大して示す。また、qdctz_k (k = 1;¢¢¢ ; 6) の信号を出力する部分も、図 4-10 から、図 4-12 に拡大して示す。

以下に示した出力結果の量子化信号 (qdctz_k (k = 1;¢¢¢ ; 6)) と、図 4-6 の値を照合した結果、全く同じ値が得られており、量子化回路が正常に動作していることを確認できた。

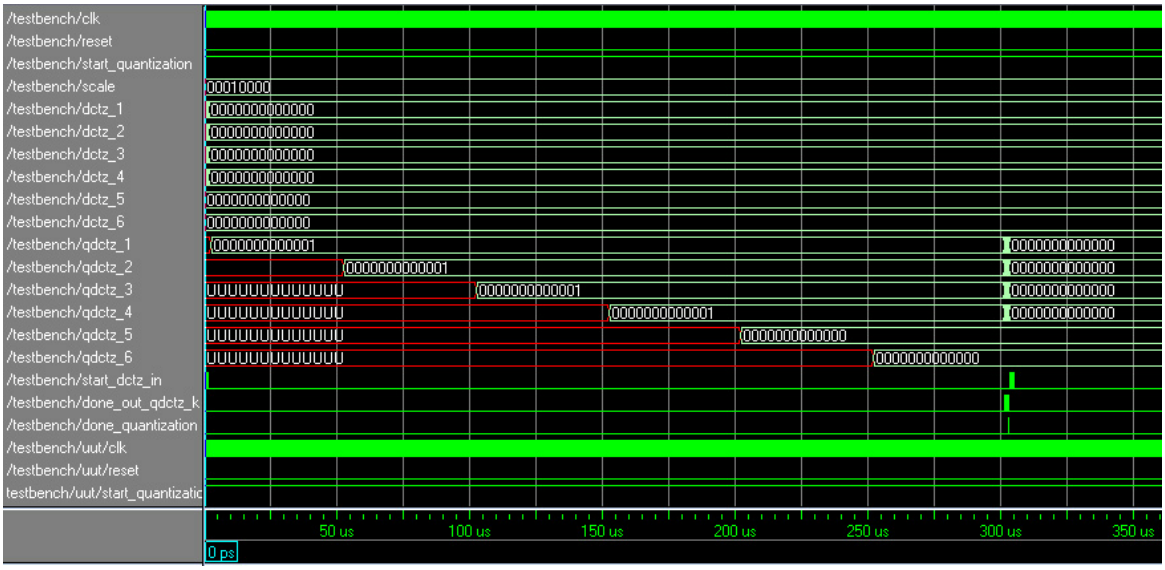

図 4-7 量子化回路のシミュレーション結果 (0ps から 350us まで)

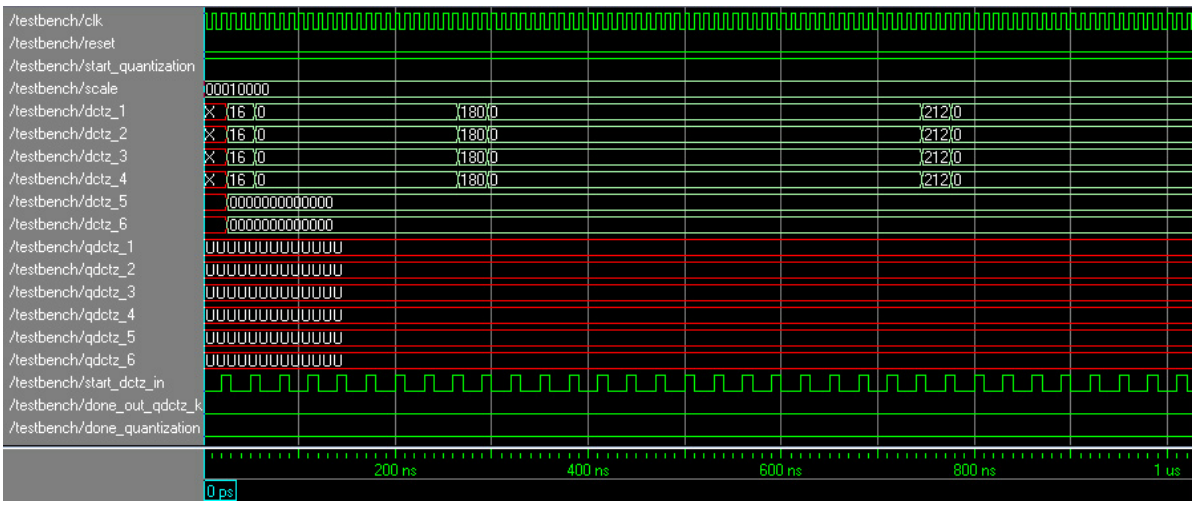

図 4-8 量子化回路のシミュレーション結果 (0ps から 1us まで)

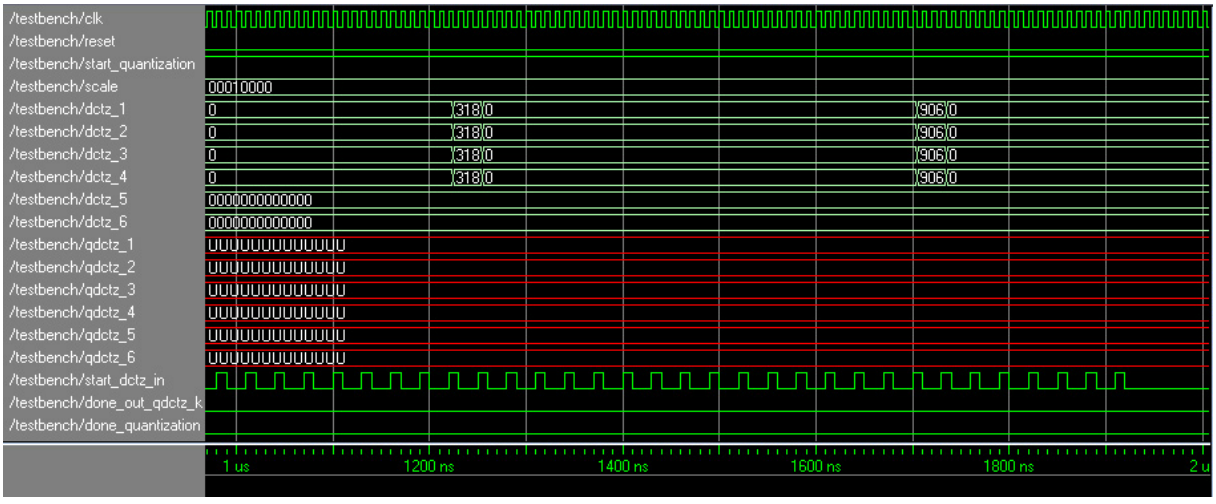

図 4-9 量子化回路のシミュレーション結果 (1us から 2us まで)

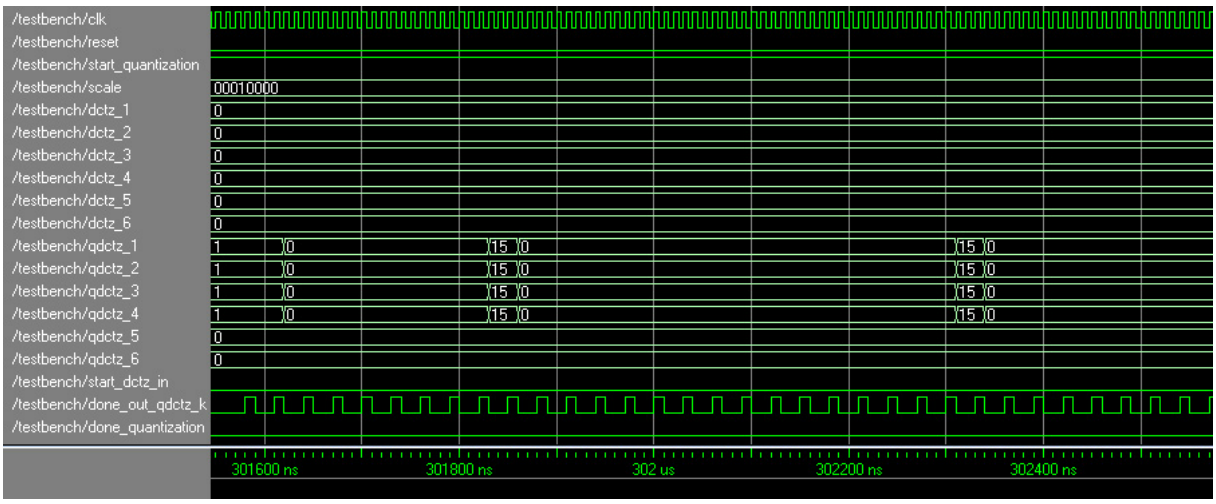

図 4-10 量子化回路のシミュレーション結果 (31600ns から 302400ns まで)

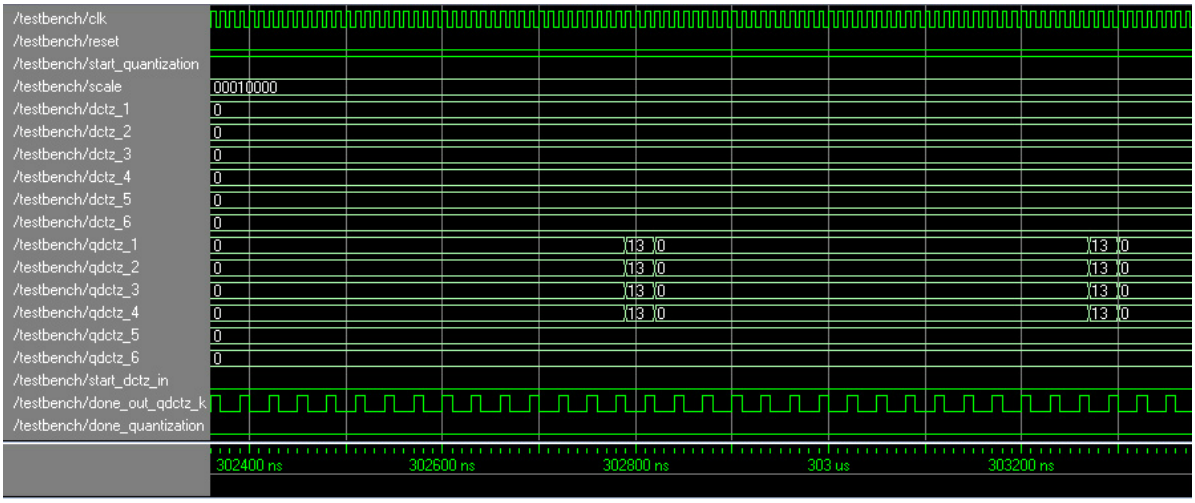

図 4-11 量子化回路のシミュレーション結果 (31600ns から 302400ns まで)

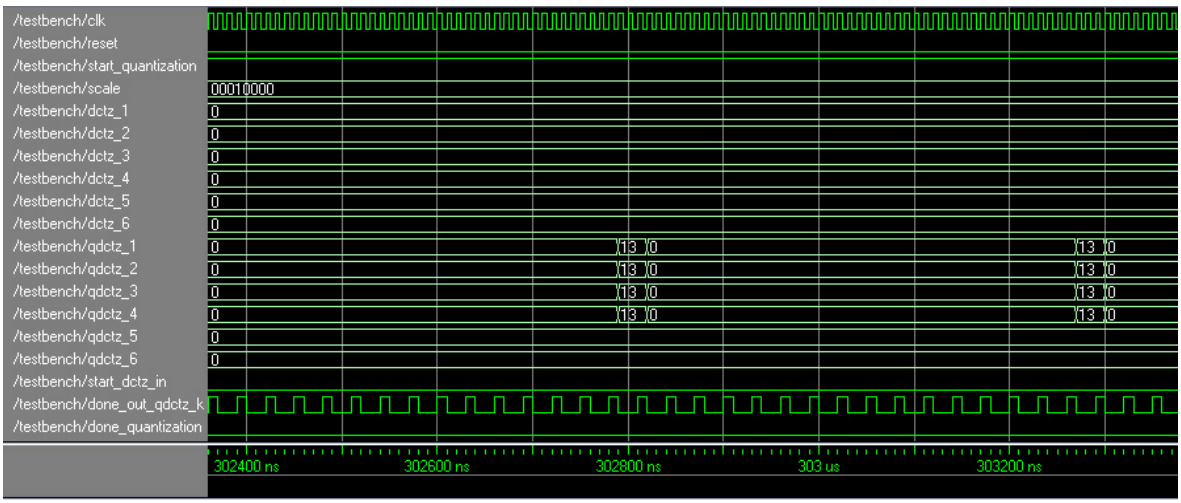

図 4-12 量子化回路のシミュレーション結果 (303200ns から 304us まで)

4-2-4 逆量子化回路のシミュレーション

4-2-4-1 逆量子化回路のシミュレーションの概要

C 言語プログラムから出力された値 (Encode_qdct_k_bit_outfile.txt) を TEXTIO のテスト・パターンとして入力する。次に、シミュレーション結果と、C 言語プログラムから出力されたファイル (Decode_outfile.txt) と照合することにより、量子化回路の検証を行う。以下では、図 4-13 に TEXTIO のテスト・パターン (qdctz_k ($k = 1, \cdots, 6$)) を 10 進数で示し、照合する値 (dctz_k ($k = 1, \cdots, 6$)) を 10 進数で図 4-14 に示す。また、逆量子化回路のシミュレーション記述は、付録として添付してある。[4][5]

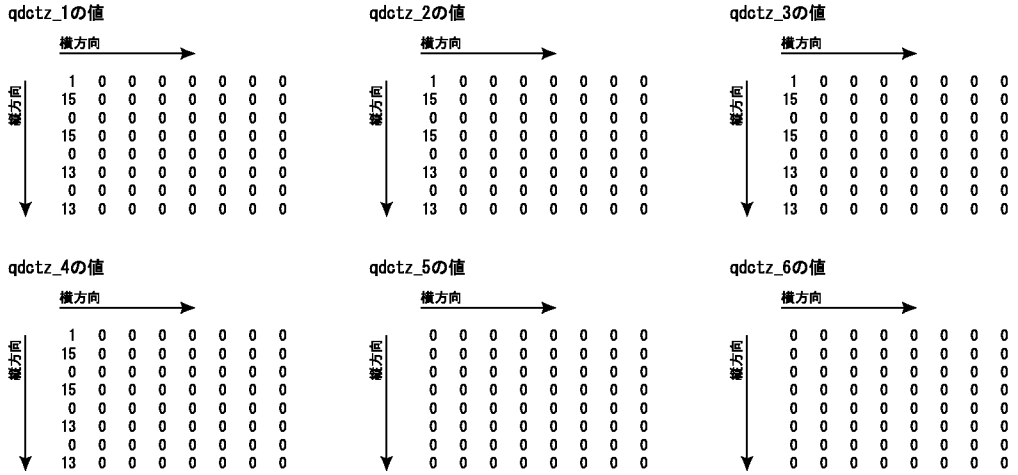

図 4-13 TEXTIO のテスト・パターン (qdctz_k ($k = 1, \cdots, 6$))

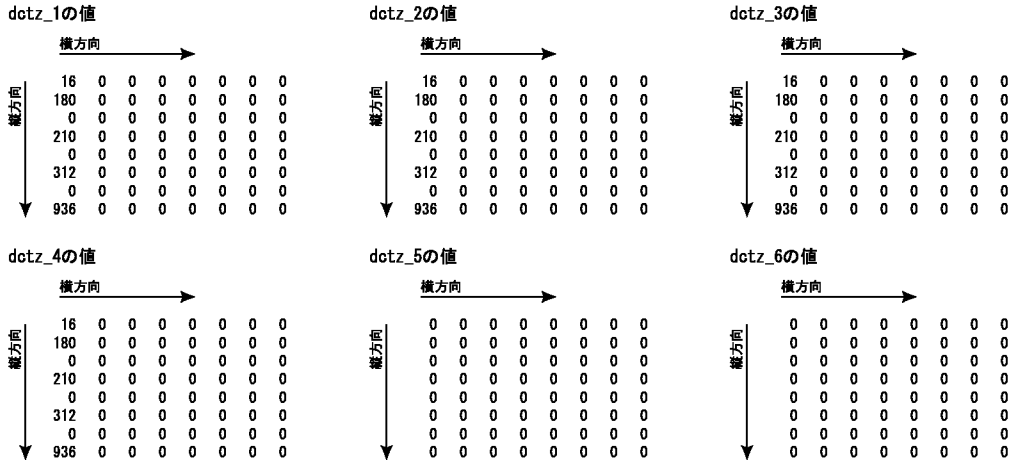

図 4-14 照合する値 (dctz_k ($k = 1, \cdots, 6$))

4-2-4-2 逆量子化回路のシミュレーション結果と考察

図 4-15 に、逆量子化回路のシミュレーション結果 (0ps から 60us まで) を示す。この図はシミュレーションの開始から、逆量子化回路が動作終了 (done_dequantization) を出力するまでを示している。0ps から 60us までの結果では、qdctz_k (k = 1;¢¢¢ ; 6) に信号を入力する部分が小さく表示されているため、図 4-16 から、図 4-19 に拡大して示す。また、dctz_k (k = 1;¢¢¢ ; 6) の信号を出力する部分も、図 4-20 から、図 4-24 に拡大して示す。

以下に示した出力結果の DCT 信号 (dctz_k (k = 1;¢¢¢ ; 6)) と、図 4-14 の値を照合した結果、全く同じ値が得られており、逆量子化回路が正常に動作していることを確認できた。

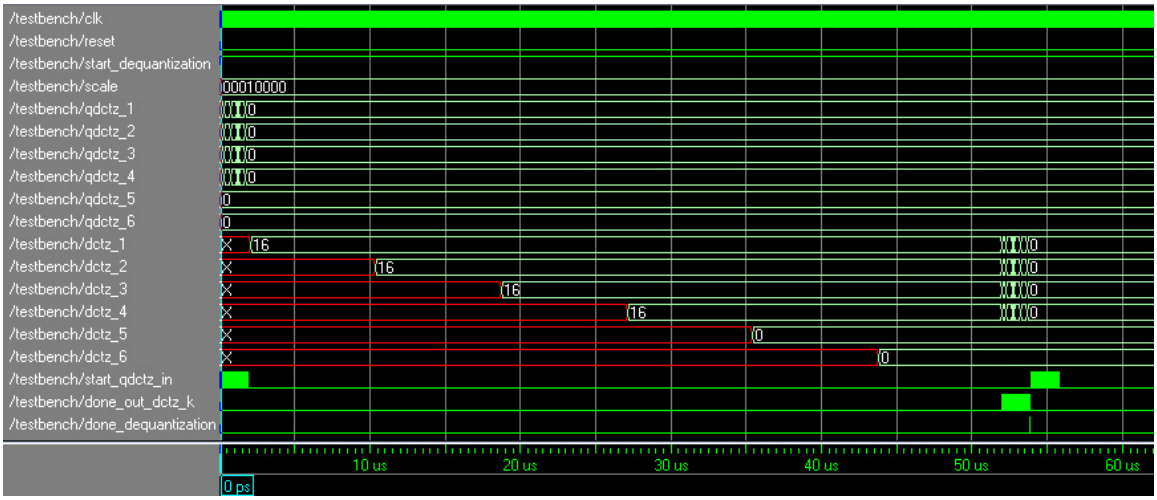

図 4-15 逆量子化回路のシミュレーション結果 (0ps から 60us まで)

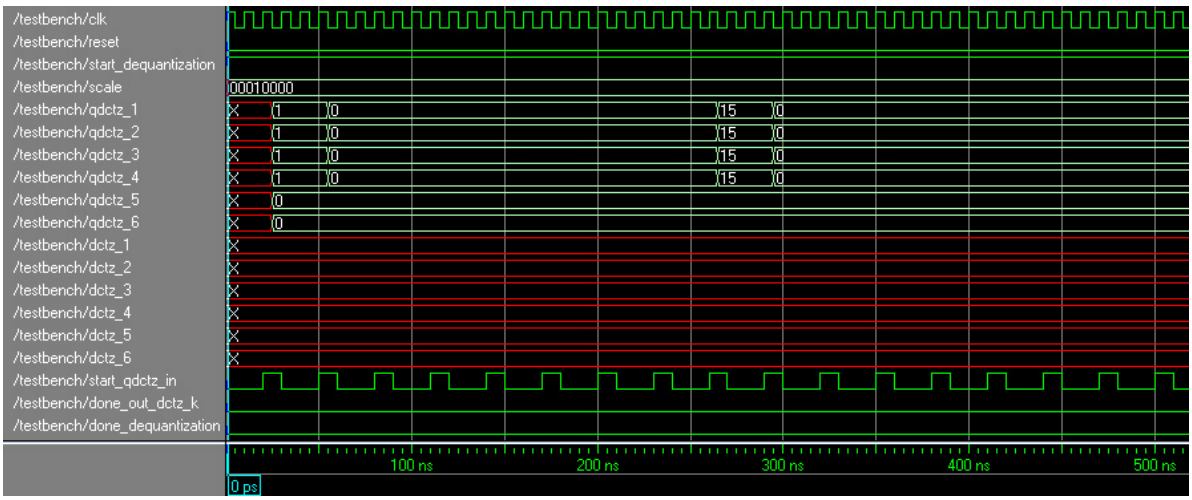

図 4-16 逆量子化回路のシミュレーション結果 (0ps から 500ns まで)

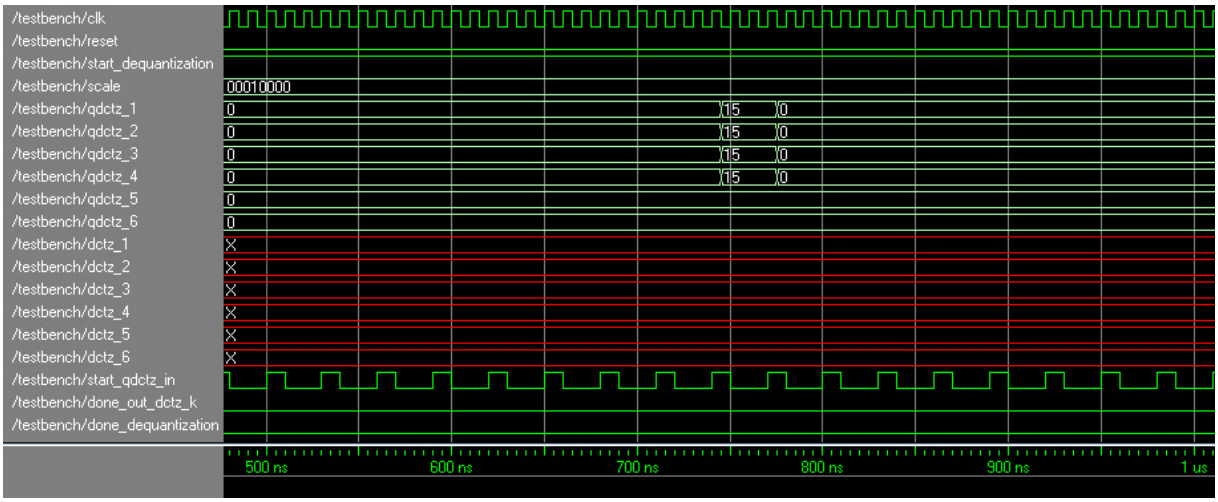

図 4-17 逆量子化回路のシミュレーション結果 (500ns から 1us まで)

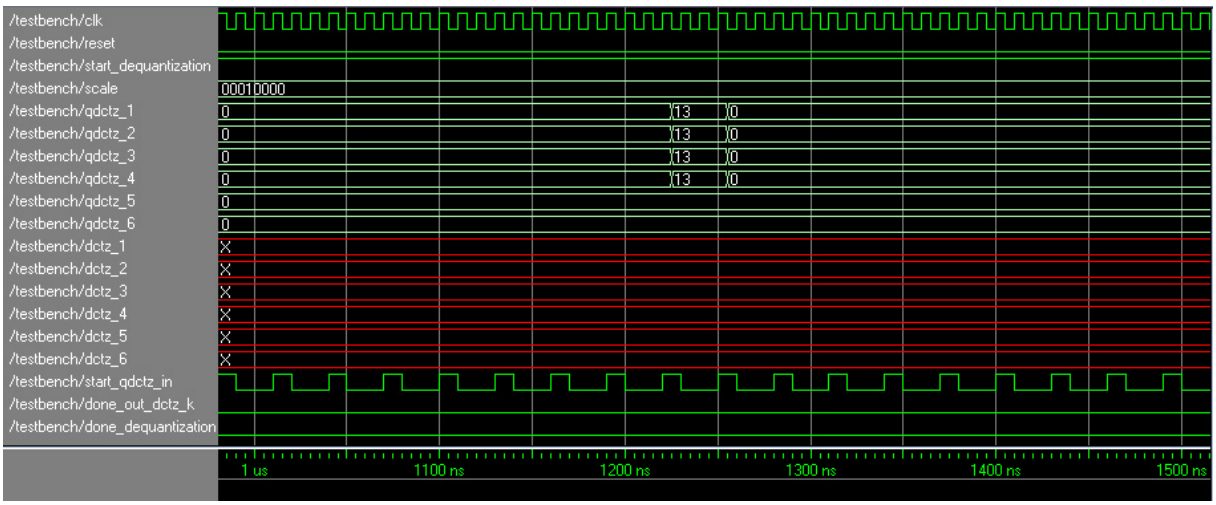

図 4-18 逆量子化回路のシミュレーション結果 (1us から 1500ns まで)

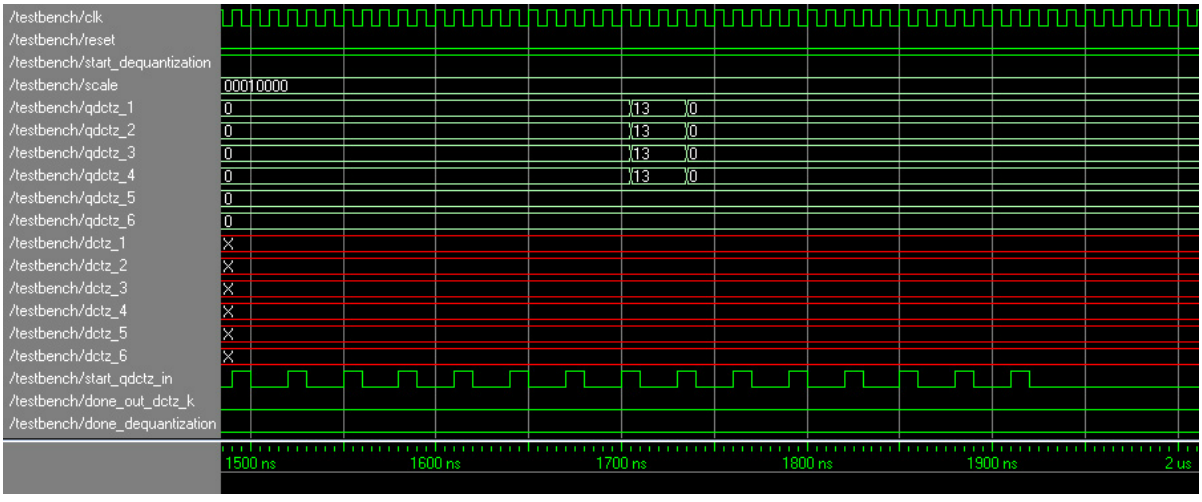

図 4-19 逆量子化回路のシミュレーション結果 (1500ns から 2us まで)

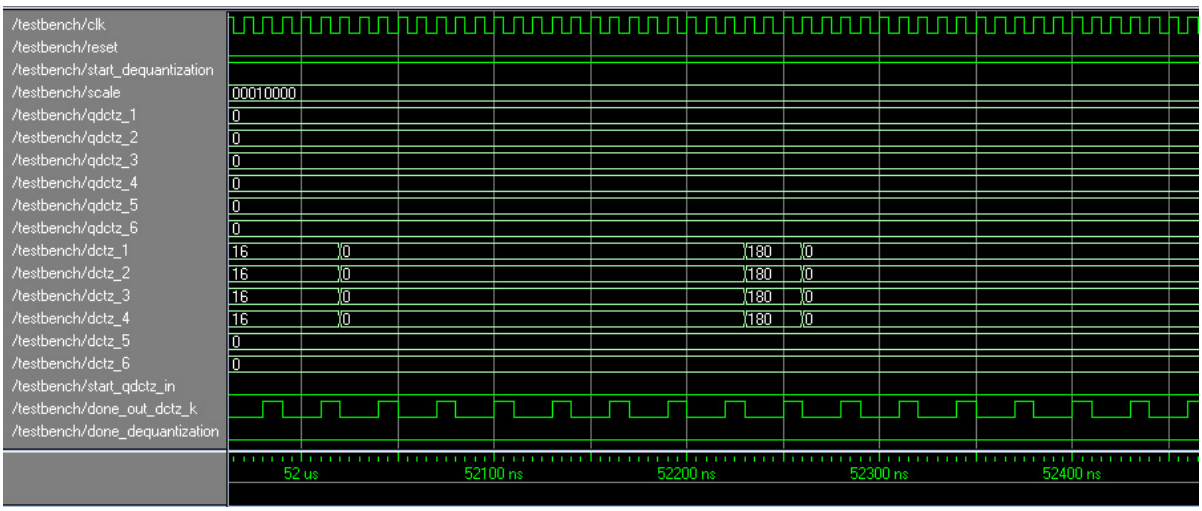

図 4-20 逆量子化回路のシミュレーション結果 (52us から 52400ns まで)

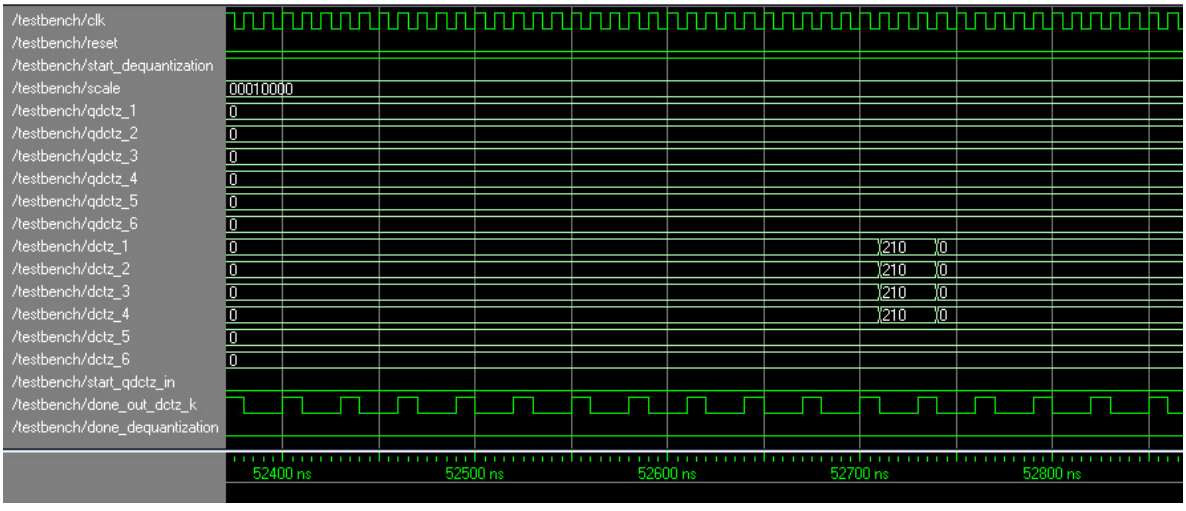

図 4-21 逆量子化回路のシミュレーション結果 (52400ns から 52800ns まで)

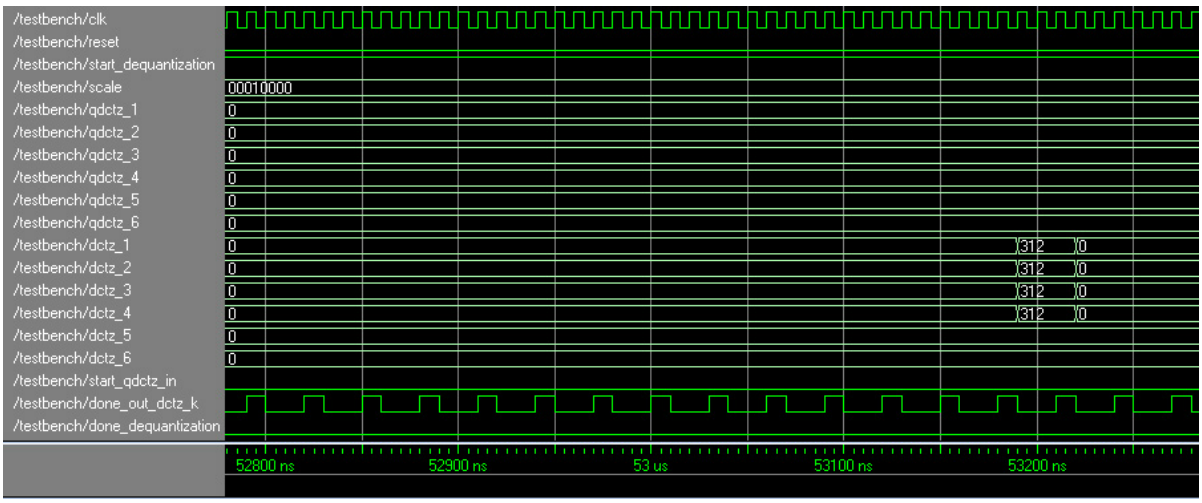

図 4-22 逆量子化回路のシミュレーション結果 (52800ns から 53200ns まで)

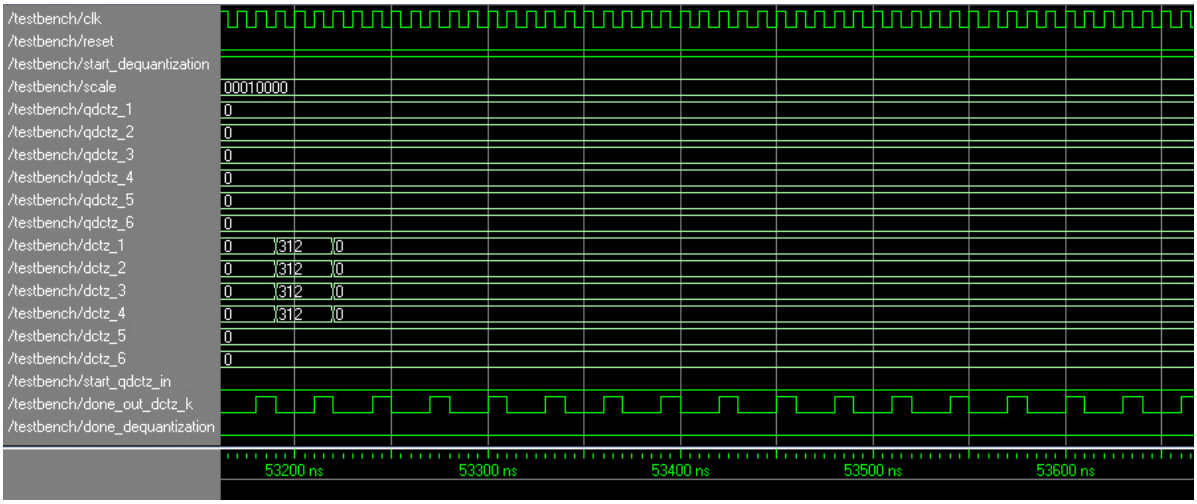

図 4-23 逆量子化回路のシミュレーション結果 (53200ns から 53600ns まで)

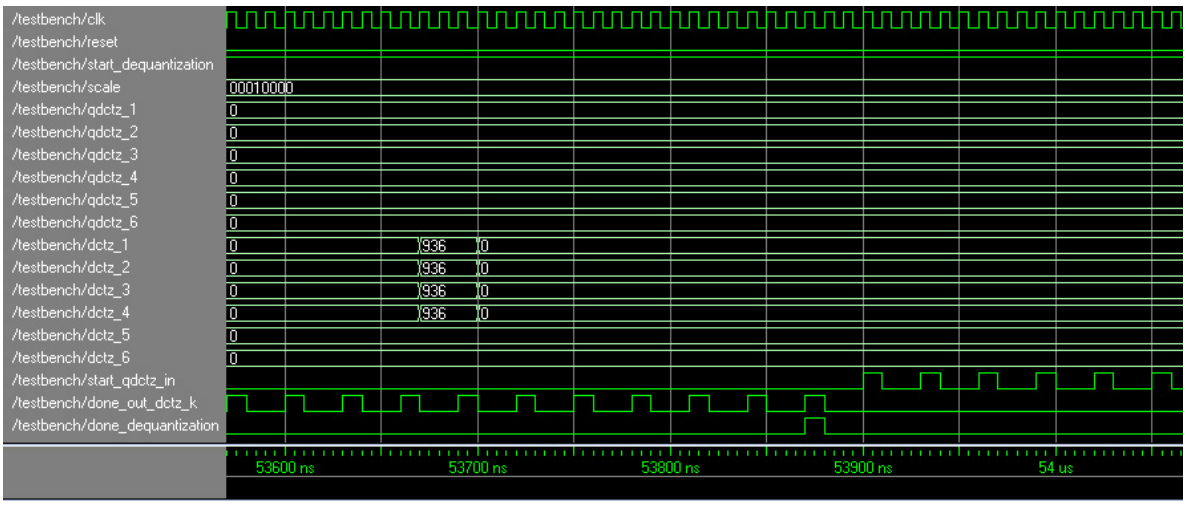

図 4-24 逆量子化回路のシミュレーション結果 (53600ns から 54us まで)

4-2-5 DCT 変換回路のシミュレーション

4-2-5-1 DCT 変換回路のシミュレーションの概要

C 言語プログラムから出力された値 (Encode_yuv_k_bit_outfile.txt) を TEXTIO のテスト・パターンとして入力する。次に、シミュレーション結果と、C 言語プログラムから出力されたファイル (Encode_outfile.txt) と照合することにより、 DCT 変換回路の検証を行う。以下では、図 4-24 に TEXTIO のテスト・パターン (yuv_k ($k = 1, \cdots, 6$)) を 10 進数で示し、照合する値 (dctz_k ($k = 1, \cdots, 6$)) を 10 進数で図 4-25 に示す。また、逆量子化回路のシミュレーション記述は、付録として添付してある。[4][5]

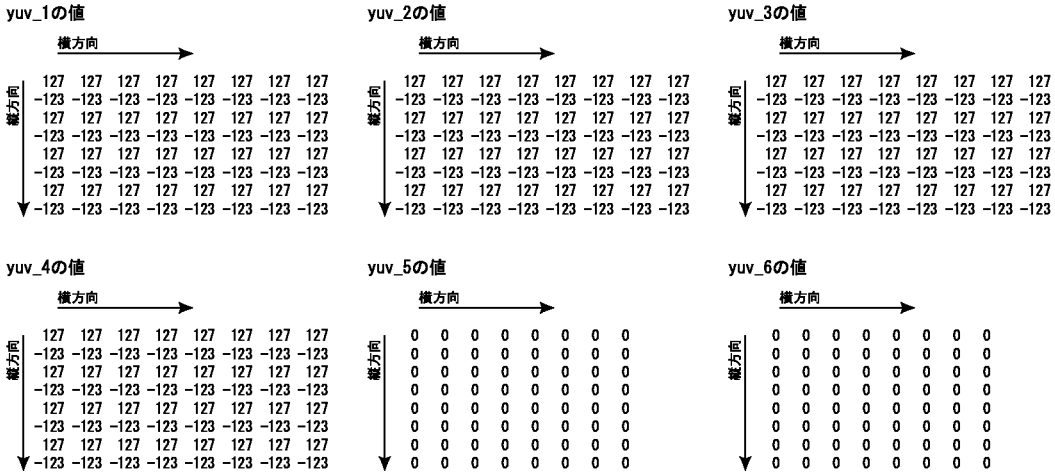

図 4-24 TEXTIO のテスト・パターン (yuv_k ($k = 1, \cdots, 6$))

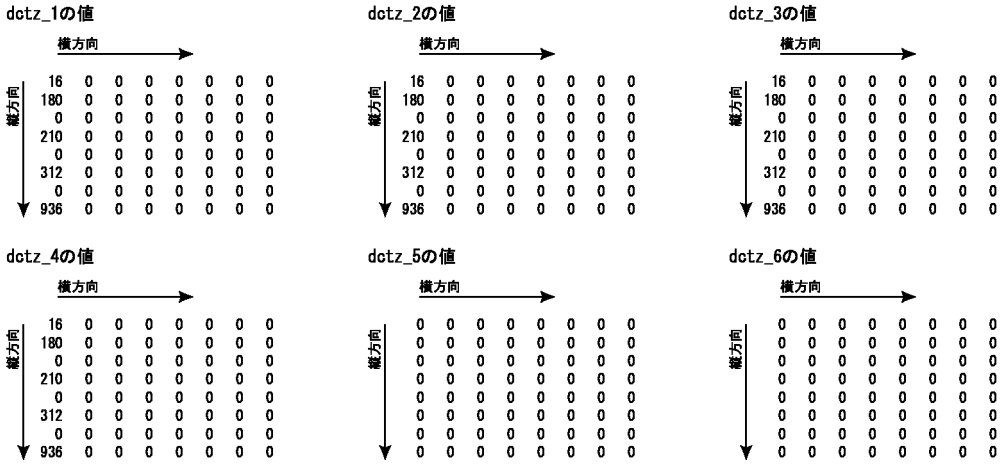

図 4-25 照合する値 (dctz_k ($k = 1, \cdots, 6$))

4-2-5-2 DCT 変換回路のシミュレーション結果と考察

図 4-26 に、DCT 変換回路のシミュレーション結果 (0ps から 5ms まで) を示す。この図はシミュレーションの開始から、DCT 変換回路が動作終了 (done_forwarddct) を出力するまでを示している。0ps から 5ms までの結果では、yuv_k (k = 1;¢¢¢ ; 6) に信号を入力する部分が小さく表示されているため、図 4-27 に拡大して示す。また、dctz_k (k = 1;¢¢¢ ; 6) の信号を出力する部分も、図 4-28 から、図 4-30 に拡大して示す。

以下に示した出力結果の DCT 信号 (dctz_k (k = 1;¢¢¢ ; 6)) と、図 4-25 の値を照合した結果、全く同じ値が得られており、DCT 変換回路が正常に動作していることを確認できた。

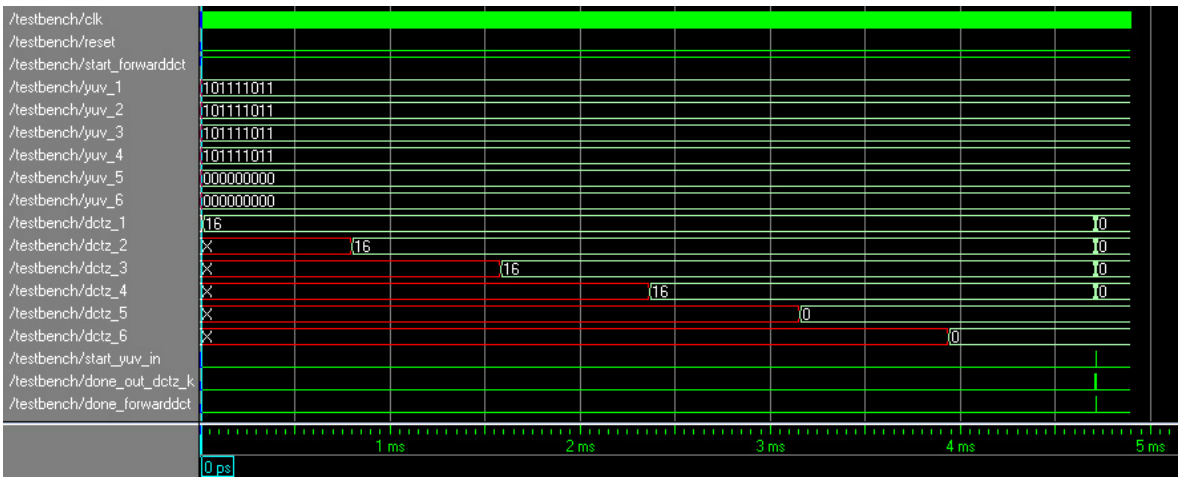

図 4-26 DCT 変換回路のシミュレーション結果 (0ps から 5ms まで)

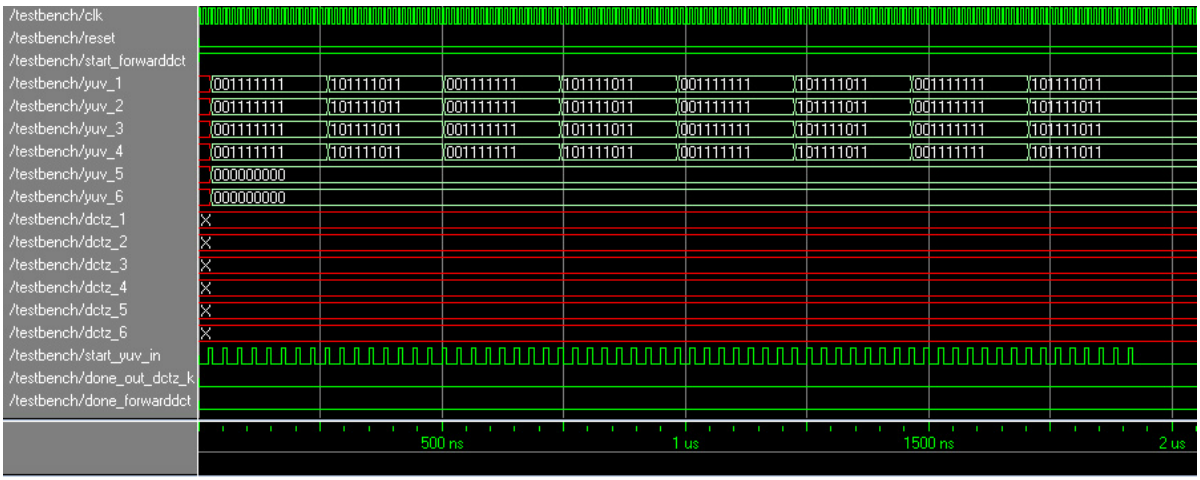

図 4-27 DCT 変換回路のシミュレーション結果 (0ps から 2us まで)

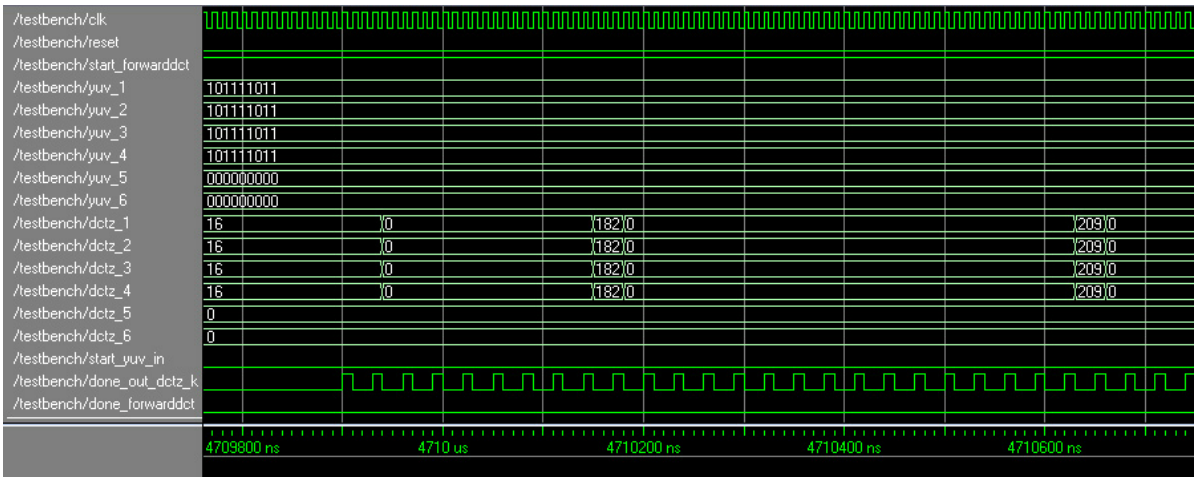

図 4-28 DCT 変換回路のシミュレーション結果
(4709800ns から 4710600ns まで)

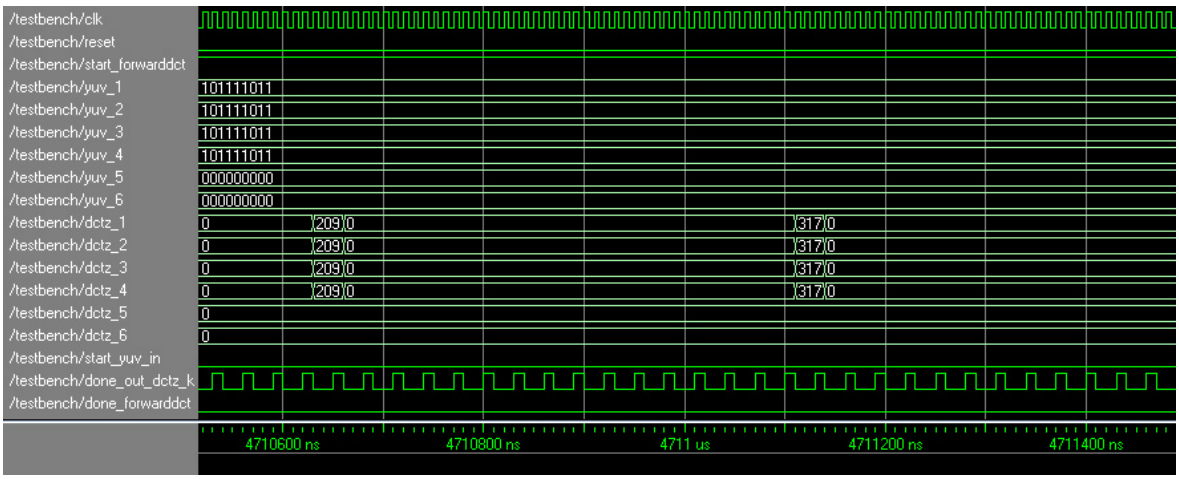

図 4-29 DCT 変換回路のシミュレーション結果

(4710600ns から 4711400ns まで)

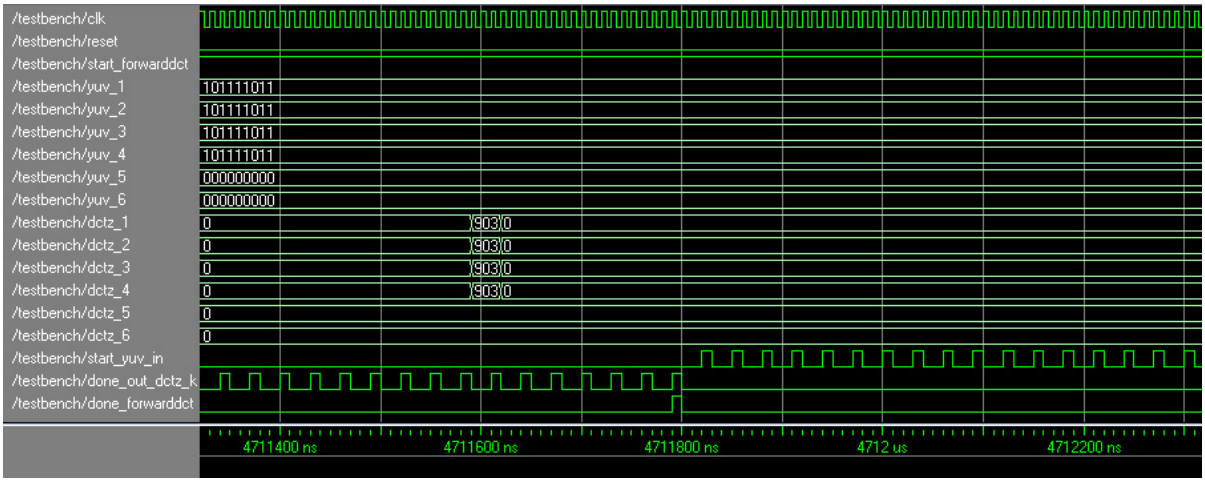

図 4-30 DCT 変換回路のシミュレーション結果

(4711400ns から 4712200ns まで)

第5章　まとめ

5-1 まとめ

本研究では、事前に実験計画を立てていたが、JPEG アルゴリズムの理解と、VHDL の理解に、時間がかってしまったため、FPGA への実装はできなかった。しかし、回路を設計するための過程を順番にこなしていき、最終的には VHDL のシミュレーションまで出来たことが、良い経験になったと思われる。今後、FPGA への実装をして回路を完成したい。

謝辞

本研究を進めるにあたり、御指導いただきました橘 昌良助教授、適宜ご指導いただいた原 央教授、矢野 政顕教授に厚く御礼申し上げます。また、本研究を進めるにあたり、御協力いただきました小原 健治氏、左古 美波子氏、松久 治可氏をはじめ、電子・光システム工学科教員、卒研室学生の同僚に感謝いたします。

参考文献

[1] 小野 定康 : わかりやすい JPEG/MPEG2 の技術, オーム社 (2001)

[2] 松本 紳, 小高 和己 : マルチメディア ビギナーズテキスト 第 2 版,
東京電機大学出版局 (1997)

[3] 村田 正幸 : マルチメディア情報ネットワーク, 共立出版 (1999)

[4] 長谷川 裕泰 : VHDL によるハードウェア設計入門,CQ 出版 (1995)

[5] 深山 正幸, 北川 章夫, 秋田 純一, 鈴木 正國 :
HDL による VLSI 設計 -VerilogHDL と VHDL による CPU 設計-, 共立出版 (1999)

[6] 坂巻 佳壽美 : 見てわかる VHDL, 工業調査会 (2002)

[7] 柴山 潔 : コンピュータアーキテクチャの基礎, 近代科学社 (1993)

[8] ウェブサイト : アスキーデジタル用語大辞典, アスキー
http://yougo.ascii24.com/

[9] ウェブサイト : なひたふ新聞
http://member.nifty.ne.jp/nahitafu/

付録

1)　はじめに

この付録では本誌で記載しきれなかった、VHDL ならびに C 言語の記述を掲載する。VHDL ではエディタに Xilinx 社の WebPackISE 4.2、シミュレータは ModelSim XE 5.5e を使用した。また、C 言語ではエディタに BCC Developer 1.1.16 を、コンパイラに Borland C++ Compiler 5.5 を使用した。

2)　量子化回路の記述

2-1)　L_C_TABLES の記述

```
-----------------------------
--
--   L_C_TABLES Package
--
-----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package L_C_TABLES is

subtype LUMINANCEQTABLE_WORD is std_logic_vector(7 downto 0);
type LUMINENCEQTABLE_ARRAY is array(0 to 63) of LUMINANCEQTABLE_WORD;
subtype CHROMINANCEQTABLE_WORD is std_logic_vector(7 downto 0);
type CHROMINANCEQTABLE_ARRAY is array(0 to 63) of CHROMINANCEQTABLE_WORD;

-----------------------------
--
--   LUMINANCEQTABLE
--
-----------------------------
constant LUMINANCEQTABLE : LUMINENCEQTABLE_ARRAY :=
(
   "00010000",      "00001011",      "00001010",      "00010000", --  0  1  2  3
   "00011000",      "00101000",      "00110011",      "00111101", --  4  5  6  7
   "00001100",      "00001100",      "00001110",      "00010011", --  8  9 10 11
   "00011010",      "00111010",      "00111100",      "00110111", -- 12 13 14 15
   "00001110",      "00001101",      "00010000",      "00011000", -- 16 17 18 19
   "00101000",      "00111001",      "01000101",      "00111000", -- 20 21 22 23
   "00001110",      "00010001",      "00010110",      "00011101", -- 24 25 26 27
   "00110011",      "01010111",      "01010000",      "00111110", -- 28 29 30 31
   "00010010",      "00010110",      "00100101",      "00111000", -- 32 33 34 35
   "01000100",      "01101101",      "01100111",      "01001101", -- 36 37 38 39
   "00011000",      "00100011",      "00110111",      "01000000", -- 40 41 42 43
   "01010001",      "01101000",      "01110001",      "01011100", -- 44 45 46 47
   "00110001",      "01000000",      "01001000",      "01010111", -- 48 49 50 51
   "01100111",      "01111001",      "01111000",      "01100101", -- 52 53 54 55
   "01001000",      "01011100",      "01011111",      "01100010", -- 56 57 58 59
   "01110000",      "01100100",      "01100111",      "01100011"  -- 60 61 62 63
```

```
);

-----------------------------
--
--   CHROMINANCEQTABLE
--
-----------------------------
constant CHROMINANCEQTABLE : CHROMINANCEQTABLE_ARRAY :=
(
   "00010001",      "00010010",      "00011000",      "00101111", --  0  1  2  3
   "01100011",      "01100011",      "01100011",      "01100011", --  4  5  6  7
   "00010010",      "00010101",      "00011010",      "01000010", --  8  9 10 11
   "01100011",      "01100011",      "01100011",      "01100011", -- 12 13 14 15
   "00011000",      "00011010",      "00111000",      "01100011", -- 16 17 18 19
   "01100011",      "01100011",      "01100011",      "01100011", -- 20 21 22 23
   "00101111",      "01000010",      "01100011",      "01100011", -- 24 25 26 27
   "01100011",      "01100011",      "01100011",      "01100011", -- 28 29 30 31
   "01100011",      "01100011",      "01100011",      "01100011", -- 32 33 34 35
   "01100011",      "01100011",      "01100011",      "01100011", -- 36 37 38 39
   "01100011",      "01100011",      "01100011",      "01100011", -- 40 41 42 43
   "01100011",      "01100011",      "01100011",      "01100011", -- 44 45 46 47
   "01100011",      "01100011",      "01100011",      "01100011", -- 48 49 50 51
   "01100011",      "01100011",      "01100011",      "01100011", -- 52 53 54 55
   "01100011",      "01100011",      "01100011",      "01100011", -- 56 57 58 59
   "01100011",      "01100011",      "01100011",      "01100011"  -- 60 61 62 63
);

end L_C_TABLES;
```

2-2) Quantization の記述

```
-----------------------------
--
--   Quantization
--
-----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.L_C_TABLES.ALL;

entity Quantization is
    Port ( CLK,RESET : in std_logic;
           START_QUANTIZATION : in std_logic;
           SCALE : in std_logic_vector(7 downto 0);
           DCTZ_1 : in std_logic_vector(12 downto 0);
           DCTZ_2 : in std_logic_vector(12 downto 0);
           DCTZ_3 : in std_logic_vector(12 downto 0);
           DCTZ_4 : in std_logic_vector(12 downto 0);
           DCTZ_5 : in std_logic_vector(12 downto 0);
           DCTZ_6 : in std_logic_vector(12 downto 0);
```

```
           QDCTZ_1 : out std_logic_vector(12 downto 0);
           QDCTZ_2 : out std_logic_vector(12 downto 0);
           QDCTZ_3 : out std_logic_vector(12 downto 0);
           QDCTZ_4 : out std_logic_vector(12 downto 0);
           QDCTZ_5 : out std_logic_vector(12 downto 0);
           QDCTZ_6 : out std_logic_vector(12 downto 0);
           START_DCTZ_IN : out std_logic;
           DONE_OUT_QDCTZ_K : out std_logic;
           DONE_QUANTIZATION : out std_logic);
end Quantization;

architecture Behavioral of Quantization is

component DCTZ_1_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_1 : in std_logic;
           RP_DCTZ_1 : in std_logic;
           AD_DCTZ_1 : in std_logic_vector(5 downto 0);
           IN_DCTZ_1 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_1 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_2_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_2 : in std_logic;
           RP_DCTZ_2 : in std_logic;
           AD_DCTZ_2 : in std_logic_vector(5 downto 0);
           IN_DCTZ_2 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_2 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_3_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_3 : in std_logic;
           RP_DCTZ_3 : in std_logic;
           AD_DCTZ_3 : in std_logic_vector(5 downto 0);
           IN_DCTZ_3 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_3 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_4_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_4 : in std_logic;
           RP_DCTZ_4 : in std_logic;
           AD_DCTZ_4 : in std_logic_vector(5 downto 0);
           IN_DCTZ_4 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_4 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_5_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_5 : in std_logic;
```

```
           RP_DCTZ_5 : in std_logic;
           AD_DCTZ_5 : in std_logic_vector(5 downto 0);
           IN_DCTZ_5 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_5 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_6_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_6 : in std_logic;
           RP_DCTZ_6 : in std_logic;
           AD_DCTZ_6 : in std_logic_vector(5 downto 0);
           IN_DCTZ_6 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_6 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_1_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_1 : in std_logic;
           RP_QDCTZ_1 : in std_logic;
           AD_QDCTZ_1 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_1 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_1 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_2_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_2 : in std_logic;
           RP_QDCTZ_2 : in std_logic;
           AD_QDCTZ_2 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_2 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_2 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_3_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_3 : in std_logic;
           RP_QDCTZ_3 : in std_logic;
           AD_QDCTZ_3 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_3 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_3 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_4_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_4 : in std_logic;
           RP_QDCTZ_4 : in std_logic;
           AD_QDCTZ_4 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_4 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_4 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_5_RAM
```

```
    Port ( CLK : in std_logic;
           WP_QDCTZ_5 : in std_logic;
           RP_QDCTZ_5 : in std_logic;
           AD_QDCTZ_5 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_5 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_5 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_6_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_6 : in std_logic;
           RP_QDCTZ_6 : in std_logic;
           AD_QDCTZ_6 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_6 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_6 : out std_logic_vector(12 downto 0));
end component;

component ALU
    Port ( CLK : in std_logic;
           START_ALU : in std_logic;
           L_C_TABLE : in std_logic_vector(7 downto 0);
           SCALE : in std_logic_vector(7 downto 0);
           SIGN : in std_logic_vector(1 downto 0);
           DCTZ_K : in std_logic_vector(12 downto 0);
           QDCTZ_K : out std_logic_vector(12 downto 0);
           DONE_ALU : out std_logic);
end component;

type STATE is (INIT,MAKE_DCTZ_RAM,IF_DCTZ_RAM,II_DCTZ_RAM_UP,
               K_VALUE,II_VALUE,MAKE_ALU,MAKE_QDCTZ_K,IF_II,II_UP,IF_K,K_UP,
               QDCTZ_OUT_II_VALUE,QDCTZ_OUT,IF_QDCTZ_OUT_II,QDCTZ_OUT_II_UP);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal K  : std_logic_vector(2 downto 0);
signal II : std_logic_vector(5 downto 0);
signal WP_DCTZ_1,WP_DCTZ_2,WP_DCTZ_3 : std_logic;
signal WP_DCTZ_4,WP_DCTZ_5,WP_DCTZ_6 : std_logic;
signal RP_DCTZ_1,RP_DCTZ_2,RP_DCTZ_3 : std_logic;
signal RP_DCTZ_4,RP_DCTZ_5,RP_DCTZ_6 : std_logic;
signal WP_QDCTZ_1,WP_QDCTZ_2,WP_QDCTZ_3 : std_logic;
signal WP_QDCTZ_4,WP_QDCTZ_5,WP_QDCTZ_6 : std_logic;
signal RP_QDCTZ_1,RP_QDCTZ_2,RP_QDCTZ_3 : std_logic;
signal RP_QDCTZ_4,RP_QDCTZ_5,RP_QDCTZ_6 : std_logic;
signal OUT_DCTZ_1,OUT_DCTZ_2,OUT_DCTZ_3 : std_logic_vector(12 downto 0);
signal OUT_DCTZ_4,OUT_DCTZ_5,OUT_DCTZ_6 : std_logic_vector(12 downto 0);
signal IN_QDCTZ_1,IN_QDCTZ_2,IN_QDCTZ_3 : std_logic_vector(12 downto 0);
signal IN_QDCTZ_4,IN_QDCTZ_5,IN_QDCTZ_6 : std_logic_vector(12 downto 0);
signal START_ALU : std_logic;
signal L_C_TABLE : std_logic_vector(7 downto 0);
signal SIGN : std_logic_vector(1 downto 0);
signal DONE_ALU : std_logic;
signal QDCTZ_K : std_logic_vector(12 downto 0);
```

```
signal DCTZ_K : std_logic_vector(12 downto 0);

begin
------------------------------
--
-- U_DCTZ_k_RAM
--
------------------------------
   U_DCTZ_1_RAM : DCTZ_1_RAM
         port map(CLK,WP_DCTZ_1,RP_DCTZ_1,II,DCTZ_1,OUT_DCTZ_1);
   U_DCTZ_2_RAM : DCTZ_2_RAM
         port map(CLK,WP_DCTZ_2,RP_DCTZ_2,II,DCTZ_2,OUT_DCTZ_2);
   U_DCTZ_3_RAM : DCTZ_3_RAM
         port map(CLK,WP_DCTZ_3,RP_DCTZ_3,II,DCTZ_3,OUT_DCTZ_3);
   U_DCTZ_4_RAM : DCTZ_4_RAM
         port map(CLK,WP_DCTZ_4,RP_DCTZ_4,II,DCTZ_4,OUT_DCTZ_4);
   U_DCTZ_5_RAM : DCTZ_5_RAM
         port map(CLK,WP_DCTZ_5,RP_DCTZ_5,II,DCTZ_5,OUT_DCTZ_5);
   U_DCTZ_6_RAM : DCTZ_6_RAM
         port map(CLK,WP_DCTZ_6,RP_DCTZ_6,II,DCTZ_6,OUT_DCTZ_6);

------------------------------
--
-- U_QDCTZ_k_RAM
--
------------------------------
   U_QDCTZ_1_RAM : QDCTZ_1_RAM
         port map(CLK,WP_QDCTZ_1,RP_QDCTZ_1,II,IN_QDCTZ_1,QDCTZ_1);
   U_QDCTZ_2_RAM : QDCTZ_2_RAM
         port map(CLK,WP_QDCTZ_2,RP_QDCTZ_2,II,IN_QDCTZ_2,QDCTZ_2);
   U_QDCTZ_3_RAM : QDCTZ_3_RAM
         port map(CLK,WP_QDCTZ_3,RP_QDCTZ_3,II,IN_QDCTZ_3,QDCTZ_3);
   U_QDCTZ_4_RAM : QDCTZ_4_RAM
         port map(CLK,WP_QDCTZ_4,RP_QDCTZ_4,II,IN_QDCTZ_4,QDCTZ_4);
   U_QDCTZ_5_RAM : QDCTZ_5_RAM
         port map(CLK,WP_QDCTZ_5,RP_QDCTZ_5,II,IN_QDCTZ_5,QDCTZ_5);
   U_QDCTZ_6_RAM : QDCTZ_6_RAM
         port map(CLK,WP_QDCTZ_6,RP_QDCTZ_6,II,IN_QDCTZ_6,QDCTZ_6);

------------------------------
--
-- MAKE_ALU
--
------------------------------
   U_ALU : ALU port map(CLK,START_ALU,L_C_TABLE,SCALE,SIGN,DCTZ_K,QDCTZ_K,DONE_ALU);

------------------------------
--
-- STATE CONTROL
--
------------------------------
   process(CLK,RESET)begin
```

```
      if(RESET='1')then
         CURRENT_STATE <= INIT;
      elsif(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
--   Quantization MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_QUANTIZATION,DONE_ALU,K,II)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_QUANTIZATION = '1')then
               START_ALU <= '0';
               DONE_QUANTIZATION <= '0';
               NEXT_STATE <= MAKE_DCTZ_RAM;
            else
               START_ALU <= '0';
               DONE_QUANTIZATION <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_DCTZ_RAM =>
            START_ALU <= '0';
            DONE_QUANTIZATION <= '0';
            NEXT_STATE <= IF_DCTZ_RAM;
         when IF_DCTZ_RAM =>
            if(II < "111111")then
               START_ALU <= '0';
               DONE_QUANTIZATION <= '0';
               NEXT_STATE <= II_DCTZ_RAM_UP;
            else
               START_ALU <= '0';
               DONE_QUANTIZATION <= '0';
               NEXT_STATE <= K_VALUE;
             end if;
         when II_DCTZ_RAM_UP =>
            START_ALU <= '0';
            DONE_QUANTIZATION <= '0';
            NEXT_STATE <= MAKE_DCTZ_RAM;

         when K_VALUE =>
            START_ALU <= '0';
            DONE_QUANTIZATION <= '0';
            NEXT_STATE <= II_VALUE;
         when II_VALUE =>
            START_ALU <= '0';
            DONE_QUANTIZATION <= '0';
            NEXT_STATE <= MAKE_ALU;
         when MAKE_ALU =>
```

```
   if(DONE_ALU = '1')then
      START_ALU <= '0';
      DONE_QUANTIZATION <= '0';
      NEXT_STATE <= MAKE_QDCTZ_K;
   else
      START_ALU <= '1';
      DONE_QUANTIZATION <= '0';
      NEXT_STATE <= MAKE_ALU;
   end if;
when MAKE_QDCTZ_K =>
   START_ALU <= '0';
   DONE_QUANTIZATION <= '0';
   NEXT_STATE <= IF_II;
when IF_II =>
   if(II < "111111")then
      START_ALU <= '0';
      DONE_QUANTIZATION <= '0';
      NEXT_STATE <= II_UP;
   else
      START_ALU <= '0';
      DONE_QUANTIZATION <= '0';
      NEXT_STATE <= IF_K;
   end if;
when II_UP =>
   START_ALU <= '0';
   DONE_QUANTIZATION <= '0';
   NEXT_STATE <= MAKE_ALU;
when IF_K =>
   if(K < "110")then
      START_ALU <= '0';
      DONE_QUANTIZATION <= '0';
      NEXT_STATE <= K_UP;
    else
      START_ALU <= '0';
      DONE_QUANTIZATION <= '0';
      NEXT_STATE <= QDCTZ_OUT_II_VALUE;
   end if;
when K_UP =>
   START_ALU <= '0';
   DONE_QUANTIZATION <= '0';
   NEXT_STATE <= II_VALUE;

when QDCTZ_OUT_II_VALUE =>
   START_ALU <= '0';
   DONE_QUANTIZATION <= '0';
   NEXT_STATE <= QDCTZ_OUT;
when QDCTZ_OUT =>
   START_ALU <= '0';
   DONE_QUANTIZATION <= '0';
   NEXT_STATE <= IF_QDCTZ_OUT_II;
when IF_QDCTZ_OUT_II =>
   if(II < "111111")then
```

```
               START_ALU <= '0';
               DONE_QUANTIZATION <= '0';
               NEXT_STATE <= QDCTZ_OUT_II_UP;
            else
               START_ALU <= '0';
               DONE_QUANTIZATION <= '1';
               NEXT_STATE <= INIT;
            end if;
         when QDCTZ_OUT_II_UP =>
            START_ALU <= '0';
            DONE_QUANTIZATION <= '0';
            NEXT_STATE <= QDCTZ_OUT;
         end case;
   end process;

------------------------------
--
-- MAKE_DCTZ_RAM
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DCTZ_RAM)then
            START_DCTZ_IN <= '1';
            WP_DCTZ_1 <= '1';
            WP_DCTZ_2 <= '1';
            WP_DCTZ_3 <= '1';
            WP_DCTZ_4 <= '1';
            WP_DCTZ_5 <= '1';
            WP_DCTZ_6 <= '1';
         else
            START_DCTZ_IN <= '0';
            WP_DCTZ_1 <= '0';
            WP_DCTZ_2 <= '0';
            WP_DCTZ_3 <= '0';
            WP_DCTZ_4 <= '0';
            WP_DCTZ_5 <= '0';
            WP_DCTZ_6 <= '0';
         end if;
      end if;
   end process;

------------------------------
--
-- II
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = II_VALUE
            or CURRENT_STATE = IF_K or CURRENT_STATE = QDCTZ_OUT_II_VALUE)then
            II <= "000000";
```

```
         elsif(CURRENT_STATE = II_DCTZ_RAM_UP or CURRENT_STATE = II_UP
               or CURRENT_STATE = QDCTZ_OUT_II_UP)then
            II <= II + 1;
         end if;
      end if;
   end process;

-----------------------------
--
-- K
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = K_VALUE
            or CURRENT_STATE = QDCTZ_OUT_II_VALUE)then
            K <= "001";
         elsif(CURRENT_STATE = K_UP)then
            K <= K + 1;
         end if;
      end if;
   end process;

-----------------------------
--
-- MAKE DCT_K
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU <= '1')then
            if(K = "001")then
               RP_DCTZ_1 <= '1';
               DCTZ_K <= OUT_DCTZ_1;
            elsif(K = "010")then
               RP_DCTZ_2 <= '1';
               DCTZ_K <= OUT_DCTZ_2;
            elsif(K = "011")then
               RP_DCTZ_3 <= '1';
               DCTZ_K <= OUT_DCTZ_3;
            elsif(K = "100")then
               RP_DCTZ_4 <= '1';
               DCTZ_K <= OUT_DCTZ_4;
            elsif(K = "101")then
               RP_DCTZ_5 <= '1';
               DCTZ_K <= OUT_DCTZ_5;
            elsif(K = "110")then
               RP_DCTZ_6 <= '1';
               DCTZ_K <= OUT_DCTZ_6;
            end if;
         else
            RP_DCTZ_1 <= '0';
```

```
            RP_DCTZ_2 <= '0';
            RP_DCTZ_3 <= '0';
            RP_DCTZ_4 <= '0';
            RP_DCTZ_5 <= '0';
            RP_DCTZ_6 <= '0';
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_L_C_TABLE
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU <= '1')then
            if(K = "101" or K = "110")then
               L_C_TABLE <= CHROMINANCEQTABLE(CONV_INTEGER(II));
            else
               L_C_TABLE <= LUMINANCEQTABLE(CONV_INTEGER(II));
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE SIGN
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU <= '1')then
            if(DCTZ_K = "0000000000000")then
               SIGN <= "00";
            elsif(DCTZ_K(12) = '0')then
               SIGN <= "01";
            else
               SIGN <= "11";
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_QDCTZ_K
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
```

```
         if(CURRENT_STATE = MAKE_QDCTZ_K)then
            if(K = "001")then
               WP_QDCTZ_1 <= '1';
               IN_QDCTZ_1 <= QDCTZ_K;
            elsif(K = "010")then
               WP_QDCTZ_2 <= '1';
               IN_QDCTZ_2 <= QDCTZ_K;
            elsif(K = "011")then
               WP_QDCTZ_3 <= '1';
               IN_QDCTZ_3 <= QDCTZ_K;
            elsif(K = "100")then
               WP_QDCTZ_4 <= '1';
               IN_QDCTZ_4 <= QDCTZ_K;
            elsif(K = "101")then
               WP_QDCTZ_5 <= '1';
               IN_QDCTZ_5 <= QDCTZ_K;
            elsif(K = "110")then
               WP_QDCTZ_6 <= '1';
               IN_QDCTZ_6 <= QDCTZ_K;
            end if;
         else
            WP_QDCTZ_1 <= '0';
            WP_QDCTZ_2 <= '0';
            WP_QDCTZ_3 <= '0';
            WP_QDCTZ_4 <= '0';
            WP_QDCTZ_5 <= '0';
            WP_QDCTZ_6 <= '0';
         end if;
      end if;
   end process;

-----------------------------
--
-- QDCTZ_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = QDCTZ_OUT)then
            RP_QDCTZ_1 <= '1';
            RP_QDCTZ_2 <= '1';
            RP_QDCTZ_3 <= '1';
            RP_QDCTZ_4 <= '1';
            RP_QDCTZ_5 <= '1';
            RP_QDCTZ_6 <= '1';
            DONE_OUT_QDCTZ_K <= '1';
         else
            RP_QDCTZ_1 <= '0';
            RP_QDCTZ_2 <= '0';
            RP_QDCTZ_3 <= '0';
            RP_QDCTZ_4 <= '0';
            RP_QDCTZ_5 <= '0';
```

```
            RP_QDCTZ_6 <= '0';
            DONE_OUT_QDCTZ_K <= '0';
         end if;
      end if;
   end process;

end Behavioral;
```

2-3) ALU の記述

```
------------------------------
--
-- ALU
--
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    Port ( CLK : in std_logic;
           START_ALU : in std_logic;
           L_C_TABLE : in std_logic_vector(7 downto 0);
           SCALE : in std_logic_vector(7 downto 0);
           SIGN : in std_logic_vector(1 downto 0);
           DCTZ_K : in std_logic_vector(12 downto 0);
           QDCTZ_K : out std_logic_vector(12 downto 0);
           DONE_ALU : out std_logic);
end ALU;

architecture Behavioral of ALU is

component MUL
    Port ( CLK : in std_logic;
           START_MUL : in std_logic;
           MUL_IN_A : in std_logic_vector(7 downto 0);
           MUL_IN_B : in std_logic_vector(7 downto 0);
           MUL_OUT : out std_logic_vector(14 downto 0);
           DONE_MUL : out std_logic);
end component;

component ADD
    Port ( CLK : in std_logic;
           START_ADD : in std_logic;
           ADD_IN_A : in std_logic_vector(16 downto 0);
           ADD_IN_B : in std_logic_vector(16 downto 0);
           ADD_OUT : out std_logic_vector(17 downto 0);
           DONE_ADD : out std_logic);
end component;

component DIV
    Port ( CLK : in std_logic;
```

```
           START_DIV : in std_logic;
           DIV_IN_A : in std_logic_vector(25 downto 0);
           DIV_IN_B : in std_logic_vector(14 downto 0);
           DIV_OUT : out std_logic_vector(12 downto 0);
           DONE_DIV : out std_logic);
end component;

type STATE is (INIT,MAKE_STEPSIZE,MAKE_MUL_SIGN,MAKE_ADD_DCTZ_K,
               MAKE_DIV_STEPSIZE,MAKE_DIV_OUT);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal MUL_IN_A,MUL_IN_B : std_logic_vector(7 downto 0);
signal MUL_OUT : std_logic_vector(14 downto 0);
signal MUL_SIGN : std_logic_vector(14 downto 0);
signal ADD_IN_A,ADD_IN_B : std_logic_vector(16 downto 0);
signal ADD_OUT : std_logic_vector(17 downto 0);
signal DIV_IN_A : std_logic_vector(25 downto 0);
signal DIV_IN_B : std_logic_vector(14 downto 0);
signal DIV_OUT : std_logic_vector(12 downto 0);
signal START_MUL,START_ADD,START_DIV : std_logic;
signal DONE_MUL,DONE_ADD,DONE_DIV : std_logic;

begin
------------------------------
--
-- port map
--
------------------------------
   U_MUL : MUL port map(CLK,START_MUL,MUL_IN_A,MUL_IN_B,MUL_OUT,DONE_MUL);
   -- S8 X S8 OUT S15
   U_ADD : ADD port map(CLK,START_ADD,ADD_IN_A,ADD_IN_B,ADD_OUT,DONE_ADD);
   -- S17 X S17 OUT S18
   U_DIV : DIV port map(CLK,START_DIV,DIV_IN_A,DIV_IN_B,DIV_OUT,DONE_DIV);
   -- S26 X S15 OUT S13

------------------------------
--
-- STATE CONTROL
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

------------------------------
--
--   ALU MAIN CIRCUIT
--
------------------------------
   process(CURRENT_STATE,START_ALU,DONE_MUL,DONE_ADD,DONE_DIV)begin
```

```
case CURRENT_STATE is
   when INIT =>
      if(START_ALU = '1')then
         START_MUL <= '0';
         START_ADD <= '0';
         START_DIV <= '0';
         DONE_ALU <= '0';
         NEXT_STATE <= MAKE_STEPSIZE;
      else
         START_MUL <= '0';
         START_ADD <= '0';
         START_DIV <= '0';
         DONE_ALU <= '0';
         NEXT_STATE <= INIT;
      end if;
   when MAKE_STEPSIZE =>
      if(DONE_MUL = '1')then
         START_MUL <= '0';
         START_ADD <= '0';
         START_DIV <= '0';
         DONE_ALU <= '0';
         NEXT_STATE <= MAKE_MUL_SIGN;
      else
         START_MUL <= '1';
         START_ADD <= '0';
         START_DIV <= '0';
         DONE_ALU <= '0';
         NEXT_STATE <= MAKE_STEPSIZE;
      end if;
   when MAKE_MUL_SIGN =>
      START_MUL <= '0';
      START_ADD <= '0';
      START_DIV <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_ADD_DCTZ_K;
   when MAKE_ADD_DCTZ_K =>
      if(DONE_ADD = '1')then
         START_MUL <= '0';
         START_ADD <= '0';
         START_DIV <= '0';
         DONE_ALU <= '0';
         NEXT_STATE <= MAKE_DIV_STEPSIZE;
      else
         START_MUL <= '0';
         START_ADD <= '1';
         START_DIV <= '0';
         DONE_ALU <= '0';
         NEXT_STATE <= MAKE_ADD_DCTZ_K;
      end if;
   when MAKE_DIV_STEPSIZE =>
      if(DONE_DIV = '1')then
         START_MUL <= '0';
```

```
               START_ADD <= '0';
               START_DIV <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_DIV_OUT;
            else
               START_MUL <= '0';
               START_ADD <= '0';
               START_DIV <= '1';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_DIV_STEPSIZE;
            end if;
         when MAKE_DIV_OUT =>
            START_MUL <= '0';
            START_ADD <= '0';
            START_DIV <= '0';
            DONE_ALU <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

-----------------------------
--
--   MAKE_STEPSIZE
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_STEPSIZE)then
            MUL_IN_A <= L_C_TABLE;
            MUL_IN_B <= SCALE;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_MUL_SIGN
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_MUL_SIGN)then
            if(SIGN(0) = '0')then
               MUL_SIGN <= "000000000000000";
            elsif(SIGN(1) = '0')then
               MUL_SIGN <= MUL_OUT;
            elsif(SIGN(1) = '1')then
               MUL_SIGN <= (not MUL_OUT(14)) & MUL_OUT(13 downto 0);
            end if;
         end if;
      end if;
   end process;
```

```
-----------------------------
--
--   MAKE_ADD_DCTZ_K
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_ADD_DCTZ_K)then
            if(MUL_SIGN(0) = '0')then
               ADD_IN_A <= MUL_SIGN(14) & "00" & MUL_SIGN(0) & MUL_SIGN(13 downto 1);
               ADD_IN_B <= DCTZ_K & "0000";
            else
               ADD_IN_A <= MUL_SIGN(14) & "000" & MUL_SIGN(13 downto 1);
               ADD_IN_B <= DCTZ_K & "0000";
            end if;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_DIV_STEPSIZE
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DIV_STEPSIZE)then
            DIV_IN_A <= ADD_OUT(17) & "00000000" & ADD_OUT(16 downto 0);
            DIV_IN_B <= MUL_OUT;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_DIV_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DIV_OUT)then
            QDCTZ_K <= DIV_OUT;
         end if;
      end if;
   end process;
end Behavioral;
```

2-3-1) MULの記述

```
-----------------------------
--
-- MUL
--
-----------------------------
-- S8 X S8 OUT S15
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUL is
   generic( IN_A_BIT : integer := 7;
            IN_B_BIT : integer := 7;
            OUT_BIT : integer := 14);

    Port ( CLK : in std_logic;
           START_MUL : in std_logic;
           MUL_IN_A : in std_logic_vector(IN_A_BIT downto 0);
           MUL_IN_B : in std_logic_vector(IN_B_BIT downto 0);
           MUL_OUT : out std_logic_vector(OUT_BIT downto 0);
           DONE_MUL : out std_logic);
end MUL;

architecture Behavioral of MUL is

type STATE is (INIT,MAKE_OUT_SIGN,
               MAKE_OUT_SUM,MAKE_MUL_OUT);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal OUT_SUM : std_logic_vector(OUT_BIT - 1 downto 0);
signal OUT_SIGN : std_logic;

begin
-----------------------------
--
-- STATE CONTROL
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
-- MUL MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_MUL)begin
```

```
      case CURRENT_STATE is
         when INIT =>
            if(START_MUL = '1')then
               DONE_MUL <= '0';
               NEXT_STATE <= MAKE_OUT_SIGN;
            else
               DONE_MUL <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_OUT_SIGN =>
            DONE_MUL <= '0';
            NEXT_STATE <= MAKE_OUT_SUM;
         when MAKE_OUT_SUM =>
            DONE_MUL <= '0';
            NEXT_STATE <= MAKE_MUL_OUT;
         when MAKE_MUL_OUT =>
            DONE_MUL <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

------------------------------
--
-- MAKE_OUT_SIGN
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SIGN)then
            if((MUL_IN_A(IN_A_BIT) = '0' and MUL_IN_B(IN_B_BIT) = '0') or
               (MUL_IN_A(IN_A_BIT) = '1' and MUL_IN_B(IN_B_BIT) = '1'))then
               OUT_SIGN <= '0';
            else
               OUT_SIGN <= '1';
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_OUT_SUM
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SUM)then
            OUT_SUM <= MUL_IN_A(IN_A_BIT - 1 downto 0) * MUL_IN_B(IN_B_BIT - 1 downto 0);
         end if;
      end if;
   end process;
```

```
-----------------------------
--
-- MAKE_MUL_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_MUL_OUT)then
            if(OUT_SUM = 0)then
               MUL_OUT <= "000000000000000"; -- OUT_BIT + 1
            else
               MUL_OUT <= OUT_SIGN & OUT_SUM;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

2-3-2) ADD の記述

```
-----------------------------
--
-- ADD
--
-----------------------------
-- S17 X S17 OUT S18
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ADD is
   generic( IN_A_BIT : integer := 16;
            IN_B_BIT : integer := 16;
            OUT_BIT : integer := 17);

    Port ( CLK : in std_logic;
           START_ADD : in std_logic;
           ADD_IN_A : in std_logic_vector(IN_A_BIT downto 0);
           ADD_IN_B : in std_logic_vector(IN_B_BIT downto 0);
           ADD_OUT : out std_logic_vector(OUT_BIT downto 0);
           DONE_ADD : out std_logic);
end ADD;

architecture Behavioral of ADD is

type STATE is (INIT,MAKE_NOSIGN_IN_AB,MAKE_RE_NOSIGN_IN_AB,
               MAKE_OUT_SUM,MAKE_RE_OUT_SUM,MAKE_ADD_OUT);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal NOSIGN_IN_A : std_logic_vector(IN_A_BIT downto 0);
signal RE_NOSIGN_IN_A : std_logic_vector(IN_A_BIT downto 0);
signal NOSIGN_IN_B : std_logic_vector(IN_B_BIT downto 0);
```

```
signal RE_NOSIGN_IN_B : std_logic_vector(IN_B_BIT downto 0);
signal OUT_SUM : std_logic_vector(OUT_BIT downto 0);
signal RE_OUT_SUM : std_logic_vector(OUT_BIT downto 0);

begin
-----------------------------
--
-- STATE CONTROL
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
-- ADD MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_ADD)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_ADD = '1')then
               DONE_ADD <= '0';
               NEXT_STATE <= MAKE_NOSIGN_IN_AB;
            else
               DONE_ADD <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_NOSIGN_IN_AB =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_RE_NOSIGN_IN_AB;
         when MAKE_RE_NOSIGN_IN_AB =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_OUT_SUM;
         when MAKE_OUT_SUM =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_RE_OUT_SUM;
         when MAKE_RE_OUT_SUM =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_ADD_OUT;
         when MAKE_ADD_OUT =>
            DONE_ADD <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

-----------------------------
--
-- MAKE_NOSIGN_IN_AB
```

```
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_NOSIGN_IN_AB)then
            NOSIGN_IN_A <= '0' & ADD_IN_A(IN_A_BIT - 1 downto 0);
            NOSIGN_IN_B <= '0' & ADD_IN_B(IN_B_BIT - 1 downto 0);
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_RE_NOSIGN_IN_AB
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_RE_NOSIGN_IN_AB)then
            if(ADD_IN_A(IN_A_BIT) = '1' and ADD_IN_B(IN_B_BIT) = '1')then
               RE_NOSIGN_IN_A <= not NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= not NOSIGN_IN_B;
            elsif(ADD_IN_A(IN_A_BIT) = '1' and ADD_IN_B(IN_B_BIT) = '0')then
               RE_NOSIGN_IN_A <= not NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= NOSIGN_IN_B;
            elsif(ADD_IN_A(IN_A_BIT) = '0' and ADD_IN_B(IN_B_BIT) = '1')then
               RE_NOSIGN_IN_A <= NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= not NOSIGN_IN_B;
            else
               RE_NOSIGN_IN_A <= NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= NOSIGN_IN_B;
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_OUT_SUM
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SUM)then
            OUT_SUM <= ('0' & RE_NOSIGN_IN_A) + ('0' & RE_NOSIGN_IN_B);
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_RE_OUT_SUM
```

```
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_RE_OUT_SUM)then
            if(OUT_SUM(OUT_BIT) = '1')then
               if(ADD_IN_A(IN_A_BIT) = '1' and ADD_IN_B(IN_B_BIT) = '1')then
                  RE_OUT_SUM <= '1' & (not (OUT_SUM(OUT_BIT - 1 downto 0) + 1));
               else
                  RE_OUT_SUM <= '0' & (OUT_SUM(OUT_BIT - 1 downto 0) + 1);
               end if;
            else
               if(ADD_IN_A(IN_A_BIT) = '0' and ADD_IN_B(IN_B_BIT) = '0')then
                  RE_OUT_SUM <= OUT_SUM;
               else
                  RE_OUT_SUM <= '1' & (not (OUT_SUM(OUT_BIT - 1 downto 0)));
               end if;
            end if;
         end if;
      end if;
   end process;

-----------------------------
--
-- MAKE_ADD_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_ADD_OUT)then
            if(RE_OUT_SUM(OUT_BIT - 1 downto 0) = 0)then
               ADD_OUT <= "00000000000000000"; --OUT_BIT + 1
            else
               ADD_OUT <= RE_OUT_SUM;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

2-3-3)　DIV の記述

```
-----------------------------
--
-- DIV
--
-----------------------------
-- S26 X S15 OUT S13
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity DIV is
   generic( IN_A_BIT : integer := 25;
            IN_B_BIT : integer := 14;
            OUT_BIT : integer := 12);

   Port ( CLK : in std_logic;
          START_DIV : in std_logic;
          DIV_IN_A : in std_logic_vector(IN_A_BIT downto 0); -- dividend
          DIV_IN_B : in std_logic_vector(IN_B_BIT downto 0); -- divisor
          DIV_OUT : out std_logic_vector(OUT_BIT downto 0);
          DONE_DIV : out std_logic);
end DIV;

architecture Behavioral of DIV is

type STATE is (INIT,MAKE_DD_DASH,MAKE_PR,MAKE_PQ,
               MAKE_RE02_PR,IF_N,N_MINUS,OUT_PQ);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal PR : std_logic_vector(IN_B_BIT - 1 downto 0);
signal DS : std_logic_vector(IN_B_BIT - 1 downto 0);
signal RE_PR : std_logic_vector(IN_B_BIT downto 0);
signal RE02_PR : std_logic_vector(IN_B_BIT - 1 downto 0);
signal DD_DASH : std_logic_vector(IN_A_BIT - IN_B_BIT - 1 downto 0);
signal PQ : std_logic_vector(IN_A_BIT - IN_B_BIT downto 0);
signal N : integer range 0 to 11;                        -- IN_A_BIT - IN_B_BIT

begin
------------------------------
--
-- STATE CONTROL
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

------------------------------
--
-- DIV MAIN CIRCUIT
--
------------------------------
   process(CURRENT_STATE,START_DIV,N)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_DIV = '1')then
               DONE_DIV <= '0';
               NEXT_STATE <= MAKE_DD_DASH;
            else
               DONE_DIV <= '0';
```

```
               NEXT_STATE <= INIT;
            end if;
         when MAKE_DD_DASH =>
            DONE_DIV <= '0';
            NEXT_STATE <= MAKE_PR;
         when MAKE_PR =>
            DONE_DIV <= '0';
            NEXT_STATE <= MAKE_PQ;
         when MAKE_PQ =>
            DONE_DIV <= '0';
            NEXT_STATE <= MAKE_RE02_PR;
         when MAKE_RE02_PR =>
            DONE_DIV <= '0';
            NEXT_STATE <= IF_N;
         when IF_N =>
            if(N > 0)then
               DONE_DIV <= '0';
               NEXT_STATE <= N_MINUS;
            else
               DONE_DIV <= '0';
               NEXT_STATE <= OUT_PQ;
            end if;
         when N_MINUS =>
            DONE_DIV <= '0';
            NEXT_STATE <= MAKE_PR;
         when OUT_PQ =>
            DONE_DIV <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

------------------------------
--
-- MAKE_DD_DASH
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DD_DASH)then
            DD_DASH <= DIV_IN_A(IN_A_BIT - IN_B_BIT - 1 downto 0);
            DS <= DIV_IN_B(IN_B_BIT - 1 downto 0);
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_PR
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
```

```
         if(CURRENT_STATE = MAKE_PR)then
            if(N = 11)then                          -- IN_A_BIT - IN_B_BIT
               PR <= DIV_IN_A(IN_A_BIT - 1 downto IN_A_BIT - IN_B_BIT);
            else
               PR <= RE02_PR;
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_PQ
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_PQ)then
            if(DS > PR)then
               RE_PR <= '0' & PR;
               PQ(N) <= '0';
            else
               RE_PR <= ('0' & PR) + ('0' & (not DS)) + 1;
               PQ(N) <= '1';
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_RE02_PR
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_RE02_PR)then
            if(N > 0)then
               RE02_PR <= RE_PR(IN_B_BIT - 2 downto 0) & DD_DASH(N-1);
            else
               RE02_PR <= RE_PR(IN_B_BIT downto 2) & DD_DASH(0);
            end if;
         end if;
      end if;
   end process;

------------------------------
--
-- MAKE_DD_DASH
--
------------------------------
   process(CLK)begin
```

```
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DD_DASH)then
            N <= 11;                                -- IN_A_BIT - IN_B_BIT
         elsif(CURRENT_STATE = N_MINUS)then
            N <= N - 1;
         end if;
      end if;
   end process;

-----------------------------
--
-- OUT_PQ
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = OUT_PQ)then
            if(DIV_IN_A(IN_A_BIT) = '1' and DIV_IN_B(IN_B_BIT) = '1')then
               DIV_OUT <= '0' & PQ;
            elsif(DIV_IN_A(IN_A_BIT) = '0' and DIV_IN_B(IN_B_BIT) = '0')then
               DIV_OUT <= '0' & PQ;
            else
               DIV_OUT <= '1' & PQ;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

2-4)　DCTZ_k_RAM (k= $1, \cdots, 6$) の記述

```
-----------------------------
--
-- DCTZ_k_RAM
--
-----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DCTZ_k_RAM is
    Port ( CLK : in std_logic;
           WP_DCTZ_k : in std_logic;
           RP_DCTZ_k : in std_logic;
           AD_DCTZ_k : in std_logic_vector(5 downto 0);
           IN_DCTZ_k : in std_logic_vector(12 downto 0);
           OUT_DCTZ_k : out std_logic_vector(12 downto 0));
end DCTZ_k_RAM;

architecture Behavioral of DCTZ_k_RAM is
   subtype DCTZ_k_WORD is std_logic_vector(12 downto 0);
   type DCTZ_k_ARRAY is array(0 to 63) of DCTZ_k_WORD;
```

```
   signal DCTZ_k : DCTZ_k_ARRAY;
   signal AD : std_logic_vector(5 downto 0);

begin
------------------------------
--
-- WRITE
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK = '1')then
         if(WP_DCTZ_k = '1')then
            DCTZ_k(CONV_INTEGER(AD_DCTZ_k)) <= IN_DCTZ_k;
         end if;
      end if;
   end process;

------------------------------
--
-- READ
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK = '1')then
         if(RP_DCTZ_k = '1')then
            AD <= AD_DCTZ_k;
         end if;
      end if;
   end process;
   OUT_DCTZ_k <= DCTZ_k(CONV_INTEGER(AD));
end Behavioral;
```

2-5)　QDCTZ_k_RAM (k= $1, \cdots, 6$) の記述

```
------------------------------
--
-- QDCTZ_k_RAM
--
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity QDCTZ_k_RAM is
    Port ( CLK : in std_logic;
           WP_QDCTZ_k : in std_logic;
           RP_QDCTZ_k : in std_logic;
           AD_QDCTZ_k : in std_logic_vector(5 downto 0);
           IN_QDCTZ_k : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_k : out std_logic_vector(12 downto 0));
end QDCTZ_k_RAM;
```

```
architecture Behavioral of QDCTZ_k_RAM is
   subtype QDCTZ_k_WORD is std_logic_vector(12 downto 0);
   type QDCTZ_k_ARRAY is array(0 to 63) of QDCTZ_k_WORD;
   signal QDCTZ_k : QDCTZ_k_ARRAY;
   signal AD : std_logic_vector(5 downto 0);

begin
-----------------------------
--
-- WRITE
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK = '1')then
         if(WP_QDCTZ_k = '1')then
            QDCTZ_k(CONV_INTEGER(AD_QDCTZ_k)) <= IN_QDCTZ_k;
         end if;
      end if;
   end process;

-----------------------------
--
-- READ
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK = '1')then
         if(RP_QDCTZ_k = '1')then
            AD <= AD_QDCTZ_k;
         end if;
      end if;
   end process;
   OUT_QDCTZ_k <= QDCTZ_k(CONV_INTEGER(AD));
end Behavioral;
```

3) 逆量子化回路の記述

3-1) DeQuantization の記述

```
-----------------------------
--
--   DeQuantization
--
-----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.L_C_TABLES.ALL;

entity DeQuantization is
    Port ( CLK,RESET : in std_logic;
```

```
           START_DEQUANTIZATION : in std_logic;
           SCALE : in std_logic_vector(7 downto 0);
           QDCTZ_1 : in std_logic_vector(12 downto 0);
           QDCTZ_2 : in std_logic_vector(12 downto 0);
           QDCTZ_3 : in std_logic_vector(12 downto 0);
           QDCTZ_4 : in std_logic_vector(12 downto 0);
           QDCTZ_5 : in std_logic_vector(12 downto 0);
           QDCTZ_6 : in std_logic_vector(12 downto 0);
           DCTZ_1 : out std_logic_vector(12 downto 0);
           DCTZ_2 : out std_logic_vector(12 downto 0);
           DCTZ_3 : out std_logic_vector(12 downto 0);
           DCTZ_4 : out std_logic_vector(12 downto 0);
           DCTZ_5 : out std_logic_vector(12 downto 0);
           DCTZ_6 : out std_logic_vector(12 downto 0);
           START_QDCTZ_IN : out std_logic;
           DONE_OUT_DCTZ_K : out std_logic;
           DONE_DEQUANTIZATION : out std_logic);
end DeQuantization;

architecture Behavioral of DeQuantization is

component DCTZ_1_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_1 : in std_logic;
           RP_DCTZ_1 : in std_logic;
           AD_DCTZ_1 : in std_logic_vector(5 downto 0);
           IN_DCTZ_1 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_1 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_2_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_2 : in std_logic;
           RP_DCTZ_2 : in std_logic;
           AD_DCTZ_2 : in std_logic_vector(5 downto 0);
           IN_DCTZ_2 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_2 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_3_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_3 : in std_logic;
           RP_DCTZ_3 : in std_logic;
           AD_DCTZ_3 : in std_logic_vector(5 downto 0);
           IN_DCTZ_3 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_3 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_4_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_4 : in std_logic;
           RP_DCTZ_4 : in std_logic;
```

```
           AD_DCTZ_4 : in std_logic_vector(5 downto 0);
           IN_DCTZ_4 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_4 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_5_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_5 : in std_logic;
           RP_DCTZ_5 : in std_logic;
           AD_DCTZ_5 : in std_logic_vector(5 downto 0);
           IN_DCTZ_5 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_5 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_6_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_6 : in std_logic;
           RP_DCTZ_6 : in std_logic;
           AD_DCTZ_6 : in std_logic_vector(5 downto 0);
           IN_DCTZ_6 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_6 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_1_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_1 : in std_logic;
           RP_QDCTZ_1 : in std_logic;
           AD_QDCTZ_1 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_1 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_1 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_2_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_2 : in std_logic;
           RP_QDCTZ_2 : in std_logic;
           AD_QDCTZ_2 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_2 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_2 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_3_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_3 : in std_logic;
           RP_QDCTZ_3 : in std_logic;
           AD_QDCTZ_3 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_3 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_3 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_4_RAM
    Port ( CLK : in std_logic;
```

```
           WP_QDCTZ_4 : in std_logic;
           RP_QDCTZ_4 : in std_logic;
           AD_QDCTZ_4 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_4 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_4 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_5_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_5 : in std_logic;
           RP_QDCTZ_5 : in std_logic;
           AD_QDCTZ_5 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_5 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_5 : out std_logic_vector(12 downto 0));
end component;

component QDCTZ_6_RAM
    Port ( CLK : in std_logic;
           WP_QDCTZ_6 : in std_logic;
           RP_QDCTZ_6 : in std_logic;
           AD_QDCTZ_6 : in std_logic_vector(5 downto 0);
           IN_QDCTZ_6 : in std_logic_vector(12 downto 0);
           OUT_QDCTZ_6 : out std_logic_vector(12 downto 0));
end component;

component ALU
    Port ( CLK : in std_logic;
           START_ALU : in std_logic;
           L_C_TABLE : in std_logic_vector(7 downto 0);
           SCALE : in std_logic_vector(7 downto 0);
           QDCTZ_K : in std_logic_vector(12 downto 0);
           DCTZ_K : out std_logic_vector(12 downto 0);
           DONE_ALU : out std_logic);
end component;

type STATE is (INIT,MAKE_QDCTZ_RAM,IF_QDCTZ_RAM,II_QDCTZ_RAM_UP,
               K_VALUE,II_VALUE,MAKE_ALU,MAKE_DCTZ_K,IF_II,II_UP,IF_K,K_UP,
               DCTZ_OUT_II_VALUE,DCTZ_OUT,IF_DCTZ_OUT_II,DCTZ_OUT_II_UP);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal K  : std_logic_vector(2 downto 0);
signal II : std_logic_vector(5 downto 0);
signal WP_QDCTZ_1,WP_QDCTZ_2,WP_QDCTZ_3,WP_QDCTZ_4,WP_QDCTZ_5,WP_QDCTZ_6 : std_logic;
signal RP_QDCTZ_1,RP_QDCTZ_2,RP_QDCTZ_3,RP_QDCTZ_4,RP_QDCTZ_5,RP_QDCTZ_6 : std_logic;
signal WP_DCTZ_1,WP_DCTZ_2,WP_DCTZ_3,WP_DCTZ_4,WP_DCTZ_5,WP_DCTZ_6 : std_logic;
signal RP_DCTZ_1,RP_DCTZ_2,RP_DCTZ_3,RP_DCTZ_4,RP_DCTZ_5,RP_DCTZ_6 : std_logic;
signal OUT_QDCTZ_1,OUT_QDCTZ_2,OUT_QDCTZ_3 : std_logic_vector(12 downto 0);
signal OUT_QDCTZ_4,OUT_QDCTZ_5,OUT_QDCTZ_6 : std_logic_vector(12 downto 0);
signal IN_DCTZ_1,IN_DCTZ_2,IN_DCTZ_3 : std_logic_vector(12 downto 0);
signal IN_DCTZ_4,IN_DCTZ_5,IN_DCTZ_6 : std_logic_vector(12 downto 0);
signal START_ALU : std_logic;
signal L_C_TABLE : std_logic_vector(7 downto 0);
```

```
signal DONE_ALU : std_logic;
signal DCTZ_K : std_logic_vector(12 downto 0);
signal QDCTZ_K : std_logic_vector(12 downto 0);

begin
------------------------------
--
--   U_QDCTZ_1_RAM
--
------------------------------
   U_QDCTZ_1_RAM : QDCTZ_1_RAM
                   port map(CLK,WP_QDCTZ_1,RP_QDCTZ_1,II,QDCTZ_1,OUT_QDCTZ_1);
   U_QDCTZ_2_RAM : QDCTZ_2_RAM
                   port map(CLK,WP_QDCTZ_2,RP_QDCTZ_2,II,QDCTZ_2,OUT_QDCTZ_2);
   U_QDCTZ_3_RAM : QDCTZ_3_RAM
                   port map(CLK,WP_QDCTZ_3,RP_QDCTZ_3,II,QDCTZ_3,OUT_QDCTZ_3);
   U_QDCTZ_4_RAM : QDCTZ_4_RAM
                   port map(CLK,WP_QDCTZ_4,RP_QDCTZ_4,II,QDCTZ_4,OUT_QDCTZ_4);
   U_QDCTZ_5_RAM : QDCTZ_5_RAM
                   port map(CLK,WP_QDCTZ_5,RP_QDCTZ_5,II,QDCTZ_5,OUT_QDCTZ_5);
   U_QDCTZ_6_RAM : QDCTZ_6_RAM
                   port map(CLK,WP_QDCTZ_6,RP_QDCTZ_6,II,QDCTZ_6,OUT_QDCTZ_6);

------------------------------
--
--   U_DCTZ_1_RAM
--
------------------------------
   U_DCTZ_1_RAM : DCTZ_1_RAM
                   port map(CLK,WP_DCTZ_1,RP_DCTZ_1,II,IN_DCTZ_1,DCTZ_1);
   U_DCTZ_2_RAM : DCTZ_2_RAM
                   port map(CLK,WP_DCTZ_2,RP_DCTZ_2,II,IN_DCTZ_2,DCTZ_2);
   U_DCTZ_3_RAM : DCTZ_3_RAM
                   port map(CLK,WP_DCTZ_3,RP_DCTZ_3,II,IN_DCTZ_3,DCTZ_3);
   U_DCTZ_4_RAM : DCTZ_4_RAM
                   port map(CLK,WP_DCTZ_4,RP_DCTZ_4,II,IN_DCTZ_4,DCTZ_4);
   U_DCTZ_5_RAM : DCTZ_5_RAM
                   port map(CLK,WP_DCTZ_5,RP_DCTZ_5,II,IN_DCTZ_5,DCTZ_5);
   U_DCTZ_6_RAM : DCTZ_6_RAM
                   port map(CLK,WP_DCTZ_6,RP_DCTZ_6,II,IN_DCTZ_6,DCTZ_6);

------------------------------
--
--   U_ALU
--
------------------------------
   U_ALU : ALU port map(CLK,START_ALU,L_C_TABLE,SCALE,QDCTZ_K,DCTZ_K,DONE_ALU);

------------------------------
--
--   STATE CONTROL
--
```

```
-----------------------------
   process(CLK,RESET)begin
      if(RESET='1')then
         CURRENT_STATE <= INIT;
      elsif(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
--   DeQuantization MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_DEQUANTIZATION,DONE_ALU,K,II)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_DEQUANTIZATION = '1')then
               START_ALU <= '0';
               DONE_DEQUANTIZATION <= '0';
               NEXT_STATE <= MAKE_QDCTZ_RAM;
            else
               START_ALU <= '0';
               DONE_DEQUANTIZATION <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_QDCTZ_RAM =>
            START_ALU <= '0';
            DONE_DEQUANTIZATION <= '0';
            NEXT_STATE <= IF_QDCTZ_RAM;
         when IF_QDCTZ_RAM =>
            if(II < "111111")then
               START_ALU <= '0';
               DONE_DEQUANTIZATION <= '0';
               NEXT_STATE <= II_QDCTZ_RAM_UP;
            else
               START_ALU <= '0';
               DONE_DEQUANTIZATION <= '0';
               NEXT_STATE <= K_VALUE;
             end if;
         when II_QDCTZ_RAM_UP =>
            START_ALU <= '0';
            DONE_DEQUANTIZATION <= '0';
            NEXT_STATE <= MAKE_QDCTZ_RAM;

         when K_VALUE =>
            START_ALU <= '0';
            DONE_DEQUANTIZATION <= '0';
            NEXT_STATE <= II_VALUE;
         when II_VALUE =>
            START_ALU <= '0';
            DONE_DEQUANTIZATION <= '0';
```

```
   NEXT_STATE <= MAKE_ALU;
when MAKE_ALU =>
   if(DONE_ALU = '1')then
      START_ALU <= '0';
      DONE_DEQUANTIZATION <= '0';
      NEXT_STATE <= MAKE_DCTZ_K;
   else
      START_ALU <= '1';
      DONE_DEQUANTIZATION <= '0';
      NEXT_STATE <= MAKE_ALU;
   end if;
when MAKE_DCTZ_K =>
   START_ALU <= '0';
   DONE_DEQUANTIZATION <= '0';
   NEXT_STATE <= IF_II;
when IF_II =>
   if(II < "111111")then
      START_ALU <= '0';
      DONE_DEQUANTIZATION <= '0';
      NEXT_STATE <= II_UP;
   else
      START_ALU <= '0';
      DONE_DEQUANTIZATION <= '0';
      NEXT_STATE <= IF_K;
   end if;
when II_UP =>
   START_ALU <= '0';
   DONE_DEQUANTIZATION <= '0';
   NEXT_STATE <= MAKE_ALU;
when IF_K =>
   if(K < "110")then
      START_ALU <= '0';
      DONE_DEQUANTIZATION <= '0';
      NEXT_STATE <= K_UP;
    else
      START_ALU <= '0';
      DONE_DEQUANTIZATION <= '0';
      NEXT_STATE <= DCTZ_OUT_II_VALUE;
   end if;
when K_UP =>
   START_ALU <= '0';
   DONE_DEQUANTIZATION <= '0';
   NEXT_STATE <= II_VALUE;

when DCTZ_OUT_II_VALUE =>
   START_ALU <= '0';
   DONE_DEQUANTIZATION <= '0';
   NEXT_STATE <= DCTZ_OUT;
when DCTZ_OUT =>
   START_ALU <= '0';
   DONE_DEQUANTIZATION <= '0';
   NEXT_STATE <= IF_DCTZ_OUT_II;
```

```
         when IF_DCTZ_OUT_II =>
            if(II < "111111")then
               START_ALU <= '0';
               DONE_DEQUANTIZATION <= '0';
               NEXT_STATE <= DCTZ_OUT_II_UP;
            else
               START_ALU <= '0';
               DONE_DEQUANTIZATION <= '1';
               NEXT_STATE <= INIT;
            end if;
         when DCTZ_OUT_II_UP =>
            START_ALU <= '0';
            DONE_DEQUANTIZATION <= '0';
            NEXT_STATE <= DCTZ_OUT;
         end case;
   end process;

------------------------------
--
--   MAKE_QDCTZ_RAM
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_QDCTZ_RAM)then
            START_QDCTZ_IN <= '1';
            WP_QDCTZ_1 <= '1';
            WP_QDCTZ_2 <= '1';
            WP_QDCTZ_3 <= '1';
            WP_QDCTZ_4 <= '1';
            WP_QDCTZ_5 <= '1';
            WP_QDCTZ_6 <= '1';
         else
            START_QDCTZ_IN <= '0';
            WP_QDCTZ_1 <= '0';
            WP_QDCTZ_2 <= '0';
            WP_QDCTZ_3 <= '0';
            WP_QDCTZ_4 <= '0';
            WP_QDCTZ_5 <= '0';
            WP_QDCTZ_6 <= '0';
         end if;
      end if;
   end process;

------------------------------
--
--   II
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = II_VALUE
```

```
            or CURRENT_STATE = IF_K or CURRENT_STATE = DCTZ_OUT_II_VALUE)then
            II <= "000000";
         elsif(CURRENT_STATE = II_QDCTZ_RAM_UP or CURRENT_STATE = II_UP
               or CURRENT_STATE = DCTZ_OUT_II_UP)then
            II <= II + 1;
         end if;
      end if;
   end process;

------------------------------
--
--   K
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = K_VALUE
            or CURRENT_STATE = DCTZ_OUT_II_VALUE)then
            K <= "001";
         elsif(CURRENT_STATE = K_UP)then
            K <= K + 1;
         end if;
      end if;
   end process;

------------------------------
--
--   MAKE QDCTZ_K
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU <= '1')then
            if(K = "001")then
               RP_QDCTZ_1 <= '1';
               QDCTZ_K <= OUT_QDCTZ_1;
            elsif(K = "010")then
               RP_QDCTZ_2 <= '1';
               QDCTZ_K <= OUT_QDCTZ_2;
            elsif(K = "011")then
               RP_QDCTZ_3 <= '1';
               QDCTZ_K <= OUT_QDCTZ_3;
            elsif(K = "100")then
               RP_QDCTZ_4 <= '1';
               QDCTZ_K <= OUT_QDCTZ_4;
            elsif(K = "101")then
               RP_QDCTZ_5 <= '1';
               QDCTZ_K <= OUT_QDCTZ_5;
            elsif(K = "110")then
               RP_QDCTZ_6 <= '1';
               QDCTZ_K <= OUT_QDCTZ_6;
            end if;
```

```
         else
            RP_QDCTZ_1 <= '0';
            RP_QDCTZ_2 <= '0';
            RP_QDCTZ_3 <= '0';
            RP_QDCTZ_4 <= '0';
            RP_QDCTZ_5 <= '0';
            RP_QDCTZ_6 <= '0';
         end if;
      end if;
   end process;

------------------------------
--
--   MAKE L_C_TABLE
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU <= '1')then
            if(K = "101" or K = "110")then
               L_C_TABLE <= CHROMINANCEQTABLE(CONV_INTEGER(II));
            else
               L_C_TABLE <= LUMINANCEQTABLE(CONV_INTEGER(II));
            end if;
         end if;
      end if;
   end process;

------------------------------
--
--   MAKE_DCTZ_K
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DCTZ_K)then
            if(K = "001")then
               WP_DCTZ_1 <= '1';
               IN_DCTZ_1 <= DCTZ_K;
            elsif(K = "010")then
               WP_DCTZ_2 <= '1';
               IN_DCTZ_2 <= DCTZ_K;
            elsif(K = "011")then
               WP_DCTZ_3 <= '1';
               IN_DCTZ_3 <= DCTZ_K;
            elsif(K = "100")then
               WP_DCTZ_4 <= '1';
               IN_DCTZ_4 <= DCTZ_K;
            elsif(K = "101")then
               WP_DCTZ_5 <= '1';
               IN_DCTZ_5 <= DCTZ_K;
            elsif(K = "110")then
```

```
               WP_DCTZ_6 <= '1';
               IN_DCTZ_6 <= DCTZ_K;
            end if;
         else
            WP_DCTZ_1 <= '0';
            WP_DCTZ_2 <= '0';
            WP_DCTZ_3 <= '0';
            WP_DCTZ_4 <= '0';
            WP_DCTZ_5 <= '0';
            WP_DCTZ_6 <= '0';
         end if;
      end if;
   end process;

------------------------------
--
--   DCTZ_OUT
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = DCTZ_OUT)then
            RP_DCTZ_1 <= '1';
            RP_DCTZ_2 <= '1';
            RP_DCTZ_3 <= '1';
            RP_DCTZ_4 <= '1';
            RP_DCTZ_5 <= '1';
            RP_DCTZ_6 <= '1';
            DONE_OUT_DCTZ_K <= '1';
         else
            RP_DCTZ_1 <= '0';
            RP_DCTZ_2 <= '0';
            RP_DCTZ_3 <= '0';
            RP_DCTZ_4 <= '0';
            RP_DCTZ_5 <= '0';
            RP_DCTZ_6 <= '0';
            DONE_OUT_DCTZ_K <= '0';
         end if;
      end if;
   end process;

end Behavioral;
```

3-2)　ALU の記述

```
------------------------------
--
--   ALU
--
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    Port ( CLK : in std_logic;
           START_ALU : in std_logic;
           L_C_TABLE : in std_logic_vector(7 downto 0);
           SCALE : in std_logic_vector(7 downto 0);
           QDCTZ_K : in std_logic_vector(12 downto 0);
           DCTZ_K : out std_logic_vector(12 downto 0);
           DONE_ALU : out std_logic);
end ALU;

architecture Behavioral of ALU is

component MUL
    Port ( CLK : in std_logic;
           START_MUL : in std_logic;
           MUL_IN_A : in std_logic_vector(12 downto 0);
           MUL_IN_B : in std_logic_vector(12 downto 0);
           MUL_OUT : out std_logic_vector(24 downto 0);
           DONE_MUL : out std_logic);
end component;

type STATE is (INIT,MAKE_STEPSIZE,MAKE_DCZ_K,OUT_DCTZ_K);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal MUL_IN_A,MUL_IN_B : std_logic_vector(12 downto 0);
signal MUL_OUT : std_logic_vector(24 downto 0);
signal START_MUL : std_logic;
signal DONE_MUL : std_logic;

begin
------------------------------
--
--   U_MUL
--
------------------------------
   U_MUL : MUL port map(CLK,START_MUL,MUL_IN_A,MUL_IN_B,MUL_OUT,DONE_MUL); -- S13 X S13 OUT S25

------------------------------
--
--   STATE CONTROL
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

------------------------------
--
```

```
--   ALU MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_ALU,DONE_MUL)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_ALU = '1')then
               START_MUL <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_STEPSIZE;
            else
               START_MUL <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_STEPSIZE =>
            if(DONE_MUL = '1')then
               START_MUL <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_DCZ_K;
            else
               START_MUL <= '1';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_STEPSIZE;
            end if;
         when MAKE_DCZ_K =>
            if(DONE_MUL = '1')then
               START_MUL <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= OUT_DCTZ_K;
            else
               START_MUL <= '1';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_DCZ_K;
            end if;
         when OUT_DCTZ_K =>
            START_MUL <= '0';
            DONE_ALU <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

-----------------------------
--
--   MAKE_STEPSIZE
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_STEPSIZE)then
            MUL_IN_A <= L_C_TABLE(7) & "00000" & L_C_TABLE(6 downto 0);
            MUL_IN_B <= SCALE(7) & "00000" & SCALE(6 downto 0);
```

```
         elsif(CURRENT_STATE = MAKE_DCZ_K)then
            MUL_IN_A <= MUL_OUT(24) & MUL_OUT(11 downto 0);
            MUL_IN_B <= QDCTZ_K;
         end if;
      end if;
   end process;

-----------------------------
--
--   OUT_DCTZ_K
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = OUT_DCTZ_K)then
            if(MUL_OUT(23 downto 16) = 1)then
               DCTZ_K <= MUL_OUT(24) & MUL_OUT(15 downto 12) &
                         (MUL_OUT(11 downto 4) and MUL_OUT(23 downto 16));
            else
               DCTZ_K <= MUL_OUT(24) & MUL_OUT(15 downto 4);
            end if;
         end if;
      end if;
   end process;

end Behavioral;
```

3-2-1)　MULの記述

```
-----------------------------
--
--   MUL
--
-----------------------------
-- S13 X S13 OUT S25
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUL is
   generic( IN_A_BIT : integer := 12;
            IN_B_BIT : integer := 12;
            OUT_BIT : integer := 24);

    Port ( CLK : in std_logic;
           START_MUL : in std_logic;
           MUL_IN_A : in std_logic_vector(IN_A_BIT downto 0);
           MUL_IN_B : in std_logic_vector(IN_B_BIT downto 0);
           MUL_OUT : out std_logic_vector(OUT_BIT downto 0);
           DONE_MUL : out std_logic);
end MUL;
```

```
architecture Behavioral of MUL is

type STATE is (INIT,MAKE_OUT_SIGN,
               MAKE_OUT_SUM,MAKE_MUL_OUT);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal OUT_SUM : std_logic_vector(OUT_BIT - 1 downto 0);
signal OUT_SIGN : std_logic;

begin
-----------------------------
--
--   STATE CONTROL
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
--   MUL MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_MUL)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_MUL = '1')then
               DONE_MUL <= '0';
               NEXT_STATE <= MAKE_OUT_SIGN;
            else
               DONE_MUL <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_OUT_SIGN =>
            DONE_MUL <= '0';
            NEXT_STATE <= MAKE_OUT_SUM;
         when MAKE_OUT_SUM =>
            DONE_MUL <= '0';
            NEXT_STATE <= MAKE_MUL_OUT;
         when MAKE_MUL_OUT =>
            DONE_MUL <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

-----------------------------
--
--   MAKE_OUT_SIGN
--
```

```
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SIGN)then
            if((MUL_IN_A(IN_A_BIT) = '0' and MUL_IN_B(IN_B_BIT) = '0') or
               (MUL_IN_A(IN_A_BIT) = '1' and MUL_IN_B(IN_B_BIT) = '1'))then
               OUT_SIGN <= '0';
            else
               OUT_SIGN <= '1';
            end if;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_OUT_SUM
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SUM)then
            OUT_SUM <= MUL_IN_A(IN_A_BIT - 1 downto 0) * MUL_IN_B(IN_B_BIT - 1 downto 0);
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_MUL_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_MUL_OUT)then
            if(OUT_SUM = 0)then
               MUL_OUT <= "000000000000000000000000"; -- OUT_BIT + 1
            else
               MUL_OUT <= OUT_SIGN & OUT_SUM;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

4) DCT 変換回路の記述

4-1) COM の記述

```
------------------------------
--
--   COM Package
--
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package COM is

------------------------------
--
--   component DCTZ
--
------------------------------
component DCTZ_1_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_1 : in std_logic;
           RP_DCTZ_1 : in std_logic;
           AD_DCTZ_1 : in std_logic_vector(5 downto 0);
           IN_DCTZ_1 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_1 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_2_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_2 : in std_logic;
           RP_DCTZ_2 : in std_logic;
           AD_DCTZ_2 : in std_logic_vector(5 downto 0);
           IN_DCTZ_2 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_2 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_3_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_3 : in std_logic;
           RP_DCTZ_3 : in std_logic;
           AD_DCTZ_3 : in std_logic_vector(5 downto 0);
           IN_DCTZ_3 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_3 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_4_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_4 : in std_logic;
           RP_DCTZ_4 : in std_logic;
           AD_DCTZ_4 : in std_logic_vector(5 downto 0);
           IN_DCTZ_4 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_4 : out std_logic_vector(12 downto 0));
```

```
end component;

component DCTZ_5_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_5 : in std_logic;
           RP_DCTZ_5 : in std_logic;
           AD_DCTZ_5 : in std_logic_vector(5 downto 0);
           IN_DCTZ_5 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_5 : out std_logic_vector(12 downto 0));
end component;

component DCTZ_6_RAM
    Port ( CLK : in std_logic;
           WP_DCTZ_6 : in std_logic;
           RP_DCTZ_6 : in std_logic;
           AD_DCTZ_6 : in std_logic_vector(5 downto 0);
           IN_DCTZ_6 : in std_logic_vector(12 downto 0);
           OUT_DCTZ_6 : out std_logic_vector(12 downto 0));
end component;

-----------------------------
--
--   component YUV
--
-----------------------------
component YUV_1_RAM
    Port ( CLK : in std_logic;
           WP_YUV_1 : in std_logic;
           RP_YUV_1 : in std_logic;
           AD_YUV_1 : in std_logic_vector(5 downto 0);
           IN_YUV_1 : in std_logic_vector(8 downto 0);
           OUT_YUV_1 : out std_logic_vector(8 downto 0));
end component;

component YUV_2_RAM
    Port ( CLK : in std_logic;
           WP_YUV_2 : in std_logic;
           RP_YUV_2 : in std_logic;
           AD_YUV_2 : in std_logic_vector(5 downto 0);
           IN_YUV_2 : in std_logic_vector(8 downto 0);
           OUT_YUV_2 : out std_logic_vector(8 downto 0));
end component;

component YUV_3_RAM
    Port ( CLK : in std_logic;
           WP_YUV_3 : in std_logic;
           RP_YUV_3 : in std_logic;
           AD_YUV_3 : in std_logic_vector(5 downto 0);
           IN_YUV_3 : in std_logic_vector(8 downto 0);
           OUT_YUV_3 : out std_logic_vector(8 downto 0));
end component;
```

```
component YUV_4_RAM
    Port ( CLK : in std_logic;
           WP_YUV_4 : in std_logic;
           RP_YUV_4 : in std_logic;
           AD_YUV_4 : in std_logic_vector(5 downto 0);
           IN_YUV_4 : in std_logic_vector(8 downto 0);
           OUT_YUV_4 : out std_logic_vector(8 downto 0));
end component;

component YUV_5_RAM
    Port ( CLK : in std_logic;
           WP_YUV_5 : in std_logic;
           RP_YUV_5 : in std_logic;
           AD_YUV_5 : in std_logic_vector(5 downto 0);
           IN_YUV_5 : in std_logic_vector(8 downto 0);
           OUT_YUV_5 : out std_logic_vector(8 downto 0));
end component;

component YUV_6_RAM
    Port ( CLK : in std_logic;
           WP_YUV_6 : in std_logic;
           RP_YUV_6 : in std_logic;
           AD_YUV_6 : in std_logic_vector(5 downto 0);
           IN_YUV_6 : in std_logic_vector(8 downto 0);
           OUT_YUV_6 : out std_logic_vector(8 downto 0));
end component;

-----------------------------
--
--   component ALU
--
-----------------------------
component ALU
    Port ( CLK : in std_logic;
           START_ALU_SUM : in std_logic;
           START_ALU_DCTZ_K : in std_logic;
           FORWARDDCT_COS_TABLE_XU : in std_logic_vector(9 downto 0);
           FORWARDDCT_COS_TABLE_YV : in std_logic_vector(9 downto 0);
           CUCV : in std_logic_vector(9 downto 0);
           YUV_K : in std_logic_vector(8 downto 0);
           IN_SUM : in std_logic_vector(26 downto 0);
           OUT_SUM : out std_logic_vector(26 downto 0);
           OUT_DCTZ_K : out std_logic_vector(12 downto 0);
           DONE_ALU : out std_logic);
end component;

end COM;
```

4-2) FORWARDDCT_COS_TABLES の記述

```
------------------------------
--
--   FORWARDDCT_COS_TABLES Package
--
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package FORWARDDCT_COS_TABLES is

subtype FORWARDDCT_COS_WORD is std_logic_vector(9 downto 0);
type FORWARDDCT_COS_ARRAY is array(0 to 63) of FORWARDDCT_COS_WORD;

------------------------------
--
--   FORWARDDCT_COS
--
------------------------------
constant FORWARDDCT_COS : FORWARDDCT_COS_ARRAY :=
(
   "0100000000",   "0100000000",   --   0  1
   "0100000000",   "0100000000",   --   2  3
   "0100000000",   "0100000000",   --   4  5
   "0100000000",   "0100000000",   --   6  7
   "0011111011",   "0011010100",   --   8  9
   "0010001110",   "0000110001",   --  10 11
   "1000110001",   "1010001110",   --  12 13
   "1011010100",   "1011111011",   --  14 15
   "0011101100",   "0001100001",   --  16 17
   "1001100001",   "1011101100",   --  18 19
   "1011101100",   "1001100001",   --  20 21
   "0001100001",   "0011101100",   --  22 23
   "0011010100",   "1000110001",   --  24 25
   "1011111011",   "1010001110",   --  26 27
   "0010001110",   "0011111011",   --  28 29
   "0000110001",   "1011010100",   --  30 31
   "0010110101",   "1010110101",   --  32 33
   "1010110101",   "0010110101",   --  34 35
   "0010110101",   "1010110101",   --  36 37
   "1010110101",   "0010110101",   --  38 39
   "0010001110",   "1011111011",   --  40 41
   "0000110001",   "0011010100",   --  42 43
   "1011010100",   "1000110001",   --  44 45
   "0011111011",   "1010001110",   --  46 47
   "0001100001",   "1011101100",   --  48 49
   "0011101100",   "1001100001",   --  50 51
   "1001100001",   "0011101100",   --  52 53
   "1011101100",   "0001100001",   --  54 55
   "0000110001",   "1010001110",   --  56 57
   "0011010100",   "1011111011",   --  58 59
   "0011111011",   "1011010100",   --  60 61
```

```
   "0010001110",   "1000110001"   --  62 63
);

end FORWARDDCT_COS_TABLES;
```

4-3) ForwardDct の記述

```
-----------------------------
--
--   ForwardDct
--
-----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.FORWARDDCT_COS_TABLES.ALL;
use work.COM.ALL;

entity ForwardDct is
    Port ( CLK,RESET : in std_logic;
           START_FORWARDDCT : in std_logic;
           YUV_1 : in std_logic_vector(8 downto 0);
           YUV_2 : in std_logic_vector(8 downto 0);
           YUV_3 : in std_logic_vector(8 downto 0);
           YUV_4 : in std_logic_vector(8 downto 0);
           YUV_5 : in std_logic_vector(8 downto 0);
           YUV_6 : in std_logic_vector(8 downto 0);
           DCTZ_1 : out std_logic_vector(12 downto 0);
           DCTZ_2 : out std_logic_vector(12 downto 0);
           DCTZ_3 : out std_logic_vector(12 downto 0);
           DCTZ_4 : out std_logic_vector(12 downto 0);
           DCTZ_5 : out std_logic_vector(12 downto 0);
           DCTZ_6 : out std_logic_vector(12 downto 0);
           START_YUV_IN : out std_logic;
           DONE_OUT_DCTZ_K : out std_logic;
           DONE_FORWARDDCT : out std_logic);
end ForwardDct;

architecture Behavioral of ForwardDct is

type STATE is (INIT,MAKE_YUV_RAM,IF_YUV_RAM,II_YUV_RAM_UP,
               K_VALUE,UV_VALUE,II_VALUE,
               MAKE_ALU_SUM,IF_II,II_UP,
               MAKE_ALU_DCTZ_K,MAKE_DCTZ_K,
               IF_UV,UV_UP,IF_K,K_UP,
               DCTZ_OUT_UV_VALUE,DCTZ_OUT,IF_DCTZ_OUT_UV,DCTZ_OUT_UV_UP);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal K  : std_logic_vector(2 downto 0);
signal II : std_logic_vector(5 downto 0);
signal UV : std_logic_vector(5 downto 0);
```

```
signal WP_YUV_1,WP_YUV_2,WP_YUV_3,WP_YUV_4,WP_YUV_5,WP_YUV_6 : std_logic;
signal RP_YUV_1,RP_YUV_2,RP_YUV_3,RP_YUV_4,RP_YUV_5,RP_YUV_6 : std_logic;
signal WP_DCTZ_1,WP_DCTZ_2,WP_DCTZ_3,WP_DCTZ_4,WP_DCTZ_5,WP_DCTZ_6 : std_logic;
signal RP_DCTZ_1,RP_DCTZ_2,RP_DCTZ_3,RP_DCTZ_4,RP_DCTZ_5,RP_DCTZ_6 : std_logic;
signal OUT_YUV_1,OUT_YUV_2,OUT_YUV_3 : std_logic_vector(8 downto 0);
signal OUT_YUV_4,OUT_YUV_5,OUT_YUV_6 : std_logic_vector(8 downto 0);
signal IN_DCTZ_1,IN_DCTZ_2,IN_DCTZ_3 : std_logic_vector(12 downto 0);
signal IN_DCTZ_4,IN_DCTZ_5,IN_DCTZ_6 : std_logic_vector(12 downto 0);
signal START_ALU_SUM : std_logic;
signal START_ALU_DCTZ_K : std_logic;
signal FORWARDDCT_COS_TABLE_XU : std_logic_vector(9 downto 0);
signal FORWARDDCT_COS_TABLE_YV : std_logic_vector(9 downto 0);
signal DONE_ALU : std_logic;
signal DCTZ_K : std_logic_vector(12 downto 0);
signal YUV_K : std_logic_vector(8 downto 0);
signal IN_SUM : std_logic_vector(26 downto 0);
signal OUT_SUM : std_logic_vector(26 downto 0);
signal CUCV : std_logic_vector(9 downto 0);

begin
------------------------------
--
--   U_YUV_k_RAM
--
------------------------------
   U_YUV_1_RAM : YUV_1_RAM port map(CLK,WP_YUV_1,RP_YUV_1,II,YUV_1,OUT_YUV_1);
   U_YUV_2_RAM : YUV_2_RAM port map(CLK,WP_YUV_2,RP_YUV_2,II,YUV_2,OUT_YUV_2);
   U_YUV_3_RAM : YUV_3_RAM port map(CLK,WP_YUV_3,RP_YUV_3,II,YUV_3,OUT_YUV_3);
   U_YUV_4_RAM : YUV_4_RAM port map(CLK,WP_YUV_4,RP_YUV_4,II,YUV_4,OUT_YUV_4);
   U_YUV_5_RAM : YUV_5_RAM port map(CLK,WP_YUV_5,RP_YUV_5,II,YUV_5,OUT_YUV_5);
   U_YUV_6_RAM : YUV_6_RAM port map(CLK,WP_YUV_6,RP_YUV_6,II,YUV_6,OUT_YUV_6);

------------------------------
--
--   U_DCTZ_k_RAM
--
------------------------------
   U_DCTZ_1_RAM : DCTZ_1_RAM port map(CLK,WP_DCTZ_1,RP_DCTZ_1,UV,IN_DCTZ_1,DCTZ_1);
   U_DCTZ_2_RAM : DCTZ_2_RAM port map(CLK,WP_DCTZ_2,RP_DCTZ_2,UV,IN_DCTZ_2,DCTZ_2);
   U_DCTZ_3_RAM : DCTZ_3_RAM port map(CLK,WP_DCTZ_3,RP_DCTZ_3,UV,IN_DCTZ_3,DCTZ_3);
   U_DCTZ_4_RAM : DCTZ_4_RAM port map(CLK,WP_DCTZ_4,RP_DCTZ_4,UV,IN_DCTZ_4,DCTZ_4);
   U_DCTZ_5_RAM : DCTZ_5_RAM port map(CLK,WP_DCTZ_5,RP_DCTZ_5,UV,IN_DCTZ_5,DCTZ_5);
   U_DCTZ_6_RAM : DCTZ_6_RAM port map(CLK,WP_DCTZ_6,RP_DCTZ_6,UV,IN_DCTZ_6,DCTZ_6);

------------------------------
--
--   U_ALU
--
------------------------------
   U_ALU : ALU port map(CLK,START_ALU_SUM,START_ALU_DCTZ_K,
                        FORWARDDCT_COS_TABLE_XU,FORWARDDCT_COS_TABLE_YV,
                        CUCV,YUV_K,IN_SUM,OUT_SUM,DCTZ_K,DONE_ALU);
```

```
-----------------------------
--
--   STATE CONTROL
--
-----------------------------
   process(CLK,RESET)begin
      if(RESET='1')then
         CURRENT_STATE <= INIT;
      elsif(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
--   ForwardDct MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_FORWARDDCT,DONE_ALU,K,UV,II)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_FORWARDDCT = '1')then
               START_ALU_SUM <= '0';
               START_ALU_DCTZ_K <= '0';
               DONE_FORWARDDCT <= '0';
               NEXT_STATE <= MAKE_YUV_RAM;
            else
               START_ALU_SUM <= '0';
               START_ALU_DCTZ_K <= '0';
               DONE_FORWARDDCT <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_YUV_RAM =>
            START_ALU_SUM <= '0';
            START_ALU_DCTZ_K <= '0';
            DONE_FORWARDDCT <= '0';
            NEXT_STATE <= IF_YUV_RAM;
         when IF_YUV_RAM =>
            if(II < "111111")then
               START_ALU_SUM <= '0';
               START_ALU_DCTZ_K <= '0';
               DONE_FORWARDDCT <= '0';
               NEXT_STATE <= II_YUV_RAM_UP;
            else
               START_ALU_SUM <= '0';
               START_ALU_DCTZ_K <= '0';
               DONE_FORWARDDCT <= '0';
               NEXT_STATE <= K_VALUE;
             end if;
         when II_YUV_RAM_UP =>
            START_ALU_SUM <= '0';
```

```
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= MAKE_YUV_RAM;

when K_VALUE =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= UV_VALUE;
when UV_VALUE =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= II_VALUE;
when II_VALUE =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= MAKE_ALU_SUM;

when MAKE_ALU_SUM =>
   if(DONE_ALU = '1')then
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= IF_II;
   else
      START_ALU_SUM <= '1';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= MAKE_ALU_SUM;
   end if;
when IF_II =>
   if(II < "111111")then
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= II_UP;
   else
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= MAKE_ALU_DCTZ_K;
   end if;
when II_UP =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= MAKE_ALU_SUM;

when MAKE_ALU_DCTZ_K =>
   if(DONE_ALU = '1')then
```

```
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= MAKE_DCTZ_K;
   else
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '1';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= MAKE_ALU_DCTZ_K;
   end if;
when MAKE_DCTZ_K =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= IF_UV;

when IF_UV =>
   if(UV < "111111")then
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= UV_UP;
    else
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= IF_K;
   end if;
when UV_UP =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= II_VALUE;
when IF_K =>
   if(K < "110")then
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= K_UP;
    else
      START_ALU_SUM <= '0';
      START_ALU_DCTZ_K <= '0';
      DONE_FORWARDDCT <= '0';
      NEXT_STATE <= DCTZ_OUT_UV_VALUE;
   end if;
when K_UP =>
   START_ALU_SUM <= '0';
   START_ALU_DCTZ_K <= '0';
   DONE_FORWARDDCT <= '0';
   NEXT_STATE <= UV_VALUE;

when DCTZ_OUT_UV_VALUE =>
```

```
            START_ALU_SUM <= '0';
            START_ALU_DCTZ_K <= '0';
            DONE_FORWARDDCT <= '0';
            NEXT_STATE <= DCTZ_OUT;
         when DCTZ_OUT =>
            START_ALU_SUM <= '0';
            START_ALU_DCTZ_K <= '0';
            DONE_FORWARDDCT <= '0';
            NEXT_STATE <= IF_DCTZ_OUT_UV;
         when IF_DCTZ_OUT_UV =>
            if(UV < "111111")then
               START_ALU_SUM <= '0';
               START_ALU_DCTZ_K <= '0';
               DONE_FORWARDDCT <= '0';
               NEXT_STATE <= DCTZ_OUT_UV_UP;
            else
               START_ALU_SUM <= '0';
               START_ALU_DCTZ_K <= '0';
               DONE_FORWARDDCT <= '1';
               NEXT_STATE <= INIT;
            end if;
         when DCTZ_OUT_UV_UP =>
            START_ALU_SUM <= '0';
            START_ALU_DCTZ_K <= '0';
            DONE_FORWARDDCT <= '0';
            NEXT_STATE <= DCTZ_OUT;
         end case;
   end process;

------------------------------
--
--   MAKE_YUV_RAM
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_YUV_RAM)then
            START_YUV_IN <= '1';
            WP_YUV_1 <= '1';
            WP_YUV_2 <= '1';
            WP_YUV_3 <= '1';
            WP_YUV_4 <= '1';
            WP_YUV_5 <= '1';
            WP_YUV_6 <= '1';
         else
            START_YUV_IN <= '0';
            WP_YUV_1 <= '0';
            WP_YUV_2 <= '0';
            WP_YUV_3 <= '0';
            WP_YUV_4 <= '0';
            WP_YUV_5 <= '0';
            WP_YUV_6 <= '0';
```

```
         end if;
      end if;
   end process;

------------------------------
--
--   II
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = II_VALUE
            or CURRENT_STATE = MAKE_ALU_DCTZ_K)then
            II <= "000000";
         elsif(CURRENT_STATE = II_YUV_RAM_UP or CURRENT_STATE = II_UP)then
            II <= II + 1;
         end if;
      end if;
   end process;

------------------------------
--
--   UV
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = UV_VALUE
            or CURRENT_STATE = IF_K or CURRENT_STATE = DCTZ_OUT_UV_VALUE)then
            UV <= "000000";
         elsif(CURRENT_STATE = UV_UP or CURRENT_STATE = DCTZ_OUT_UV_UP)then
            UV <= UV + 1;
         end if;
      end if;
   end process;

------------------------------
--
--   K
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = INIT or CURRENT_STATE = K_VALUE
            or CURRENT_STATE = DCTZ_OUT_UV_VALUE)then
            K <= "001";
         elsif(CURRENT_STATE = K_UP)then
            K <= K + 1;
         end if;
      end if;
   end process;
```

```
-----------------------------
--
--   MAKE YUV_K
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU_SUM <= '1')then
            if(K = "001")then
               RP_YUV_1 <= '1';
               YUV_K <= OUT_YUV_1;
            elsif(K = "010")then
               RP_YUV_2 <= '1';
               YUV_K <= OUT_YUV_2;
            elsif(K = "011")then
               RP_YUV_3 <= '1';
               YUV_K <= OUT_YUV_3;
            elsif(K = "100")then
               RP_YUV_4 <= '1';
               YUV_K <= OUT_YUV_4;
            elsif(K = "101")then
               RP_YUV_5 <= '1';
               YUV_K <= OUT_YUV_5;
            elsif(K = "110")then
               RP_YUV_6 <= '1';
               YUV_K <= OUT_YUV_6;
            end if;
         else
            RP_YUV_1 <= '0';
            RP_YUV_2 <= '0';
            RP_YUV_3 <= '0';
            RP_YUV_4 <= '0';
            RP_YUV_5 <= '0';
            RP_YUV_6 <= '0';
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE FORWARDDCT_COS_TABLE
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU_SUM <= '1')then
            FORWARDDCT_COS_TABLE_XU
                  <= FORWARDDCT_COS
                     (8*CONV_INTEGER(UV(2 downto 0))+CONV_INTEGER(II(2 downto 0)));
            FORWARDDCT_COS_TABLE_YV
                  <= FORWARDDCT_COS
                     (8*CONV_INTEGER(UV(5 downto 3))+CONV_INTEGER(II(5 downto 3)));
```

```
         end if;
      end if;
   end process;

------------------------------
--
--   MAKE IN_SUM
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(II = "000000" and START_ALU_DCTZ_K <= '0')then
            IN_SUM <= "00000000000000000000000000000000";
         else
            IN_SUM <= OUT_SUM;
         end if;
      end if;
   end process;

------------------------------
--
--   MAKE CUCV
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(START_ALU_DCTZ_K <= '1')then
            if(UV = "000000")then
               CUCV <= "0010000000";
            elsif(UV(5 downto 3)="000" or UV(2 downto 0)="000")then
               CUCV <= "0010110101";
            else
               CUCV <= "0100000000";
            end if;
         end if;
      end if;
   end process;

------------------------------
--
--   MAKE_DCTZ_K
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_DCTZ_K)then
            if(K = "001")then
               WP_DCTZ_1 <= '1';
               IN_DCTZ_1 <= DCTZ_K;
            elsif(K = "010")then
               WP_DCTZ_2 <= '1';
               IN_DCTZ_2 <= DCTZ_K;
```

```
            elsif(K = "011")then
               WP_DCTZ_3 <= '1';
               IN_DCTZ_3 <= DCTZ_K;
            elsif(K = "100")then
               WP_DCTZ_4 <= '1';
               IN_DCTZ_4 <= DCTZ_K;
            elsif(K = "101")then
               WP_DCTZ_5 <= '1';
               IN_DCTZ_5 <= DCTZ_K;
            elsif(K = "110")then
               WP_DCTZ_6 <= '1';
               IN_DCTZ_6 <= DCTZ_K;
            end if;
         else
            WP_DCTZ_1 <= '0';
            WP_DCTZ_2 <= '0';
            WP_DCTZ_3 <= '0';
            WP_DCTZ_4 <= '0';
            WP_DCTZ_5 <= '0';
            WP_DCTZ_6 <= '0';
         end if;
      end if;
   end process;

-----------------------------
--
--   DCTZ_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = DCTZ_OUT)then
            RP_DCTZ_1 <= '1';
            RP_DCTZ_2 <= '1';
            RP_DCTZ_3 <= '1';
            RP_DCTZ_4 <= '1';
            RP_DCTZ_5 <= '1';
            RP_DCTZ_6 <= '1';
            DONE_OUT_DCTZ_K <= '1';
         else
            RP_DCTZ_1 <= '0';
            RP_DCTZ_2 <= '0';
            RP_DCTZ_3 <= '0';
            RP_DCTZ_4 <= '0';
            RP_DCTZ_5 <= '0';
            RP_DCTZ_6 <= '0';
            DONE_OUT_DCTZ_K <= '0';
         end if;
      end if;
   end process;

end Behavioral;
```

4-4) ALUの記述

```
------------------------------
--
--   ALU
--
------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    Port ( CLK : in std_logic;
           START_ALU_SUM : in std_logic;
           START_ALU_DCTZ_K : in std_logic;
           FORWARDDCT_COS_TABLE_XU : in std_logic_vector(9 downto 0);
           FORWARDDCT_COS_TABLE_YV : in std_logic_vector(9 downto 0);
           CUCV : in std_logic_vector(9 downto 0);
           YUV_K : in std_logic_vector(8 downto 0);
           IN_SUM : in std_logic_vector(26 downto 0);
           OUT_SUM : out std_logic_vector(26 downto 0);
           OUT_DCTZ_K : out std_logic_vector(12 downto 0);
           DONE_ALU : out std_logic);
end ALU;

architecture Behavioral of ALU is

component MUL
    Port ( CLK : in std_logic;
           START_MUL : in std_logic;
           MUL_IN_A : in std_logic_vector(13 downto 0);
           MUL_IN_B : in std_logic_vector(13 downto 0);
           MUL_OUT : out std_logic_vector(26 downto 0);
           DONE_MUL : out std_logic);
end component;

component ADD
    Port ( CLK : in std_logic;
           START_ADD : in std_logic;
           ADD_IN_A : in std_logic_vector(26 downto 0);
           ADD_IN_B : in std_logic_vector(26 downto 0);
           ADD_OUT : out std_logic_vector(27 downto 0);
           DONE_ADD : out std_logic);
end component;

type STATE is (INIT,IF_START,MAKE_FORWARDDCT_COS,MAKE_MUL_YUV,MAKE_ADD_SUM,
               MAKE_OUT_SUM,MAKE_MUL_CUCV,MAKE_OUT_DCTZ_K);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal MUL_IN_A,MUL_IN_B : std_logic_vector(13 downto 0);
signal MUL_OUT : std_logic_vector(26 downto 0);
```

```
signal ADD_IN_A,ADD_IN_B : std_logic_vector(26 downto 0);
signal ADD_OUT : std_logic_vector(27 downto 0);
signal START_MUL,START_ADD : std_logic;
signal DONE_MUL,DONE_ADD : std_logic;

begin
------------------------------
--
--   port map
--
------------------------------
   U_MUL : MUL port map(CLK,START_MUL,MUL_IN_A,MUL_IN_B,MUL_OUT,DONE_MUL); -- S14 X S14 OUT S27
   U_ADD : ADD port map(CLK,START_ADD,ADD_IN_A,ADD_IN_B,ADD_OUT,DONE_ADD); -- S27 X S27 OUT S28

------------------------------
--
--   STATE CONTROL
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

------------------------------
--
--   ALU MAIN CIRCUIT
--
------------------------------
   process(CURRENT_STATE,START_ALU_SUM,START_ALU_DCTZ_K,DONE_MUL,DONE_ADD)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_ALU_SUM = '1' or START_ALU_DCTZ_K = '1')then
               START_MUL <= '0';
               START_ADD <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= IF_START;
            else
               START_MUL <= '0';
               START_ADD <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= INIT;
            end if;
         when IF_START =>
            if(START_ALU_SUM = '1')then
               START_MUL <= '0';
               START_ADD <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_FORWARDDCT_COS;
            else
               START_MUL <= '0';
```

```
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_MUL_CUCV;
   end if;
when MAKE_FORWARDDCT_COS =>
   if(DONE_MUL = '1')then
      START_MUL <= '0';
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_MUL_YUV;
   else
      START_MUL <= '1';
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_FORWARDDCT_COS;
   end if;
when MAKE_MUL_YUV =>
   if(DONE_MUL = '1')then
      START_MUL <= '0';
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_ADD_SUM;
   else
      START_MUL <= '1';
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_MUL_YUV;
   end if;
when MAKE_ADD_SUM =>
   if(DONE_ADD = '1')then
      START_MUL <= '0';
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_OUT_SUM;
   else
      START_MUL <= '0';
      START_ADD <= '1';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_ADD_SUM;
   end if;
when MAKE_OUT_SUM =>
   START_MUL <= '0';
   START_ADD <= '0';
   DONE_ALU <= '1';
   NEXT_STATE <= INIT;

when MAKE_MUL_CUCV =>
   if(DONE_MUL = '1')then
      START_MUL <= '0';
      START_ADD <= '0';
      DONE_ALU <= '0';
      NEXT_STATE <= MAKE_OUT_DCTZ_K;
```

```
            else
               START_MUL <= '1';
               START_ADD <= '0';
               DONE_ALU <= '0';
               NEXT_STATE <= MAKE_MUL_CUCV;
            end if;
         when MAKE_OUT_DCTZ_K =>
            START_MUL <= '0';
            START_ADD <= '0';
            DONE_ALU <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

-----------------------------
--
--   MAKE_FORWARDDCT_COS
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_FORWARDDCT_COS)then
            MUL_IN_A <= FORWARDDCT_COS_TABLE_XU & "0000";
            MUL_IN_B <= FORWARDDCT_COS_TABLE_YV & "0000";
         elsif(CURRENT_STATE = MAKE_MUL_YUV)then
            MUL_IN_A <= MUL_OUT(26 downto 16) & "000" ;
            MUL_IN_B <= YUV_K(8) & "000" & YUV_K(7 downto 0) & "00";
         elsif(CURRENT_STATE = MAKE_MUL_CUCV)then
            MUL_IN_A <= IN_SUM(26 downto 13);
            MUL_IN_B <= CUCV & "0000";
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_ADD_SUM
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_ADD_SUM)then
            ADD_IN_A <= IN_SUM;
            ADD_IN_B <= MUL_OUT;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_OUT_SUM
--
```

```
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SUM)then
            if(ADD_OUT(26)='1')then
               OUT_SUM <= (ADD_OUT(27) and ADD_OUT(26)) & ADD_OUT(25 downto 0);
            else
               OUT_SUM <= ADD_OUT(27) & ADD_OUT(25 downto 0);
             end if;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_OUT_DCTZ_K
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_DCTZ_K)then
            OUT_DCTZ_K <= MUL_OUT(26 downto 14);
         end if;
      end if;
   end process;

end Behavioral;
```

4-4-1)　MULの記述

```
-----------------------------
--
--   MUL
--
-----------------------------
-- S14 X S14 OUT S27
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUL is
   generic( IN_A_BIT : integer := 13;
            IN_B_BIT : integer := 13;
            OUT_BIT : integer := 26);

    Port ( CLK : in std_logic;
           START_MUL : in std_logic;
           MUL_IN_A : in std_logic_vector(IN_A_BIT downto 0);
           MUL_IN_B : in std_logic_vector(IN_B_BIT downto 0);
           MUL_OUT : out std_logic_vector(OUT_BIT downto 0);
           DONE_MUL : out std_logic);
end MUL;
```

```
architecture Behavioral of MUL is

type STATE is (INIT,MAKE_OUT_SIGN,
               MAKE_OUT_SUM,MAKE_MUL_OUT);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal OUT_SUM : std_logic_vector(OUT_BIT - 1 downto 0);
signal OUT_SIGN : std_logic;

begin
------------------------------
--
--   ALU MAIN CIRCUIT
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

   process(CURRENT_STATE,START_MUL)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_MUL = '1')then
               DONE_MUL <= '0';
               NEXT_STATE <= MAKE_OUT_SIGN;
            else
               DONE_MUL <= '0';
               NEXT_STATE <= INIT;
            end if;
         when MAKE_OUT_SIGN =>
            DONE_MUL <= '0';
            NEXT_STATE <= MAKE_OUT_SUM;
         when MAKE_OUT_SUM =>
            DONE_MUL <= '0';
            NEXT_STATE <= MAKE_MUL_OUT;
         when MAKE_MUL_OUT =>
            DONE_MUL <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

------------------------------
--
--   MAKE_OUT_SIGN
--
------------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SIGN)then
```

```
            if((MUL_IN_A(IN_A_BIT) = '0' and MUL_IN_B(IN_B_BIT) = '0') or
               (MUL_IN_A(IN_A_BIT) = '1' and MUL_IN_B(IN_B_BIT) = '1'))then
               OUT_SIGN <= '0';
            else
               OUT_SIGN <= '1';
            end if;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_OUT_SUM
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SUM)then
            OUT_SUM <= MUL_IN_A(IN_A_BIT - 1 downto 0) * MUL_IN_B(IN_B_BIT - 1 downto 0);
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_MUL_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_MUL_OUT)then
            if(OUT_SUM = 0)then
               MUL_OUT <= "00000000000000000000000000000"; -- OUT_BIT + 1
            else
               MUL_OUT <= OUT_SIGN & OUT_SUM;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

4-4-2)　ADD の記述

```
-----------------------------
--
--   ADD
--
-----------------------------
-- S27 X S27 OUT S28
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity ADD is
   generic( IN_A_BIT : integer := 26;
            IN_B_BIT : integer := 26;
            OUT_BIT : integer := 27);

    Port ( CLK : in std_logic;
           START_ADD : in std_logic;
           ADD_IN_A : in std_logic_vector(IN_A_BIT downto 0);
           ADD_IN_B : in std_logic_vector(IN_B_BIT downto 0);
           ADD_OUT : out std_logic_vector(OUT_BIT downto 0);
           DONE_ADD : out std_logic);
end ADD;

architecture Behavioral of ADD is

type STATE is (INIT,MAKE_NOSIGN_IN_AB,MAKE_RE_NOSIGN_IN_AB,
               MAKE_OUT_SUM,MAKE_RE_OUT_SUM,MAKE_ADD_OUT);

signal CURRENT_STATE,NEXT_STATE : STATE;
signal NOSIGN_IN_A : std_logic_vector(IN_A_BIT downto 0);
signal RE_NOSIGN_IN_A : std_logic_vector(IN_A_BIT downto 0);
signal NOSIGN_IN_B : std_logic_vector(IN_B_BIT downto 0);
signal RE_NOSIGN_IN_B : std_logic_vector(IN_B_BIT downto 0);
signal OUT_SUM : std_logic_vector(OUT_BIT downto 0);
signal RE_OUT_SUM : std_logic_vector(OUT_BIT downto 0);

begin
-----------------------------
--
--   STATE CONTROL
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         CURRENT_STATE <= NEXT_STATE;
      end if;
   end process;

-----------------------------
--
--   ADD MAIN CIRCUIT
--
-----------------------------
   process(CURRENT_STATE,START_ADD)begin
      case CURRENT_STATE is
         when INIT =>
            if(START_ADD = '1')then
               DONE_ADD <= '0';
               NEXT_STATE <= MAKE_NOSIGN_IN_AB;
            else
               DONE_ADD <= '0';
```

```
               NEXT_STATE <= INIT;
            end if;
         when MAKE_NOSIGN_IN_AB =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_RE_NOSIGN_IN_AB;
         when MAKE_RE_NOSIGN_IN_AB =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_OUT_SUM;
         when MAKE_OUT_SUM =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_RE_OUT_SUM;
         when MAKE_RE_OUT_SUM =>
            DONE_ADD <= '0';
            NEXT_STATE <= MAKE_ADD_OUT;
         when MAKE_ADD_OUT =>
            DONE_ADD <= '1';
            NEXT_STATE <= INIT;
         end case;
   end process;

-----------------------------
--
--   MAKE_NOSIGN_IN_AB
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_NOSIGN_IN_AB)then
            NOSIGN_IN_A <= '0' & ADD_IN_A(IN_A_BIT - 1 downto 0);
            NOSIGN_IN_B <= '0' & ADD_IN_B(IN_B_BIT - 1 downto 0);
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_RE_NOSIGN_IN_AB
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_RE_NOSIGN_IN_AB)then
            if(ADD_IN_A(IN_A_BIT) = '1' and ADD_IN_B(IN_B_BIT) = '1')then
               RE_NOSIGN_IN_A <= not NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= not NOSIGN_IN_B;
            elsif(ADD_IN_A(IN_A_BIT) = '1' and ADD_IN_B(IN_B_BIT) = '0')then
               RE_NOSIGN_IN_A <= not NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= NOSIGN_IN_B;
            elsif(ADD_IN_A(IN_A_BIT) = '0' and ADD_IN_B(IN_B_BIT) = '1')then
               RE_NOSIGN_IN_A <= NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= not NOSIGN_IN_B;
            else
```

```
               RE_NOSIGN_IN_A <= NOSIGN_IN_A;
               RE_NOSIGN_IN_B <= NOSIGN_IN_B;
            end if;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_OUT_SUM
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_OUT_SUM)then
            OUT_SUM <= ('0' & RE_NOSIGN_IN_A) + ('0' & RE_NOSIGN_IN_B);
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_RE_OUT_SUM
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
         if(CURRENT_STATE = MAKE_RE_OUT_SUM)then
            if(OUT_SUM(OUT_BIT) = '1')then
               if(ADD_IN_A(IN_A_BIT) = '1' and ADD_IN_B(IN_B_BIT) = '1')then
                  RE_OUT_SUM <= '1' & (not (OUT_SUM(OUT_BIT - 1 downto 0) + 1));
               else
                  RE_OUT_SUM <= '0' & (OUT_SUM(OUT_BIT - 1 downto 0) + 1);
               end if;
            else
               if(ADD_IN_A(IN_A_BIT) = '0' and ADD_IN_B(IN_B_BIT) = '0')then
                  RE_OUT_SUM <= OUT_SUM;
               else
                  RE_OUT_SUM <= '1' & (not (OUT_SUM(OUT_BIT - 1 downto 0)));
               end if;
            end if;
         end if;
      end if;
   end process;

-----------------------------
--
--   MAKE_ADD_OUT
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK='1')then
```

```
         if(CURRENT_STATE = MAKE_ADD_OUT)then
            if(RE_OUT_SUM(OUT_BIT - 1 downto 0) = 0)then
               ADD_OUT <= "00000000000000000000000000000000"; --OUT_BIT + 1
            else
               ADD_OUT <= RE_OUT_SUM;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

4-5) YUV_k_RAM (k= 1, · · · , 6) の記述

```
----------------------------
--
--   YUV_k_RAM
--
----------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity YUV_k_RAM is
    Port ( CLK : in std_logic;
           WP_YUV_k : in std_logic;
           RP_YUV_k : in std_logic;
           AD_YUV_k : in std_logic_vector(5 downto 0);
           IN_YUV_k : in std_logic_vector(8 downto 0);
           OUT_YUV_k : out std_logic_vector(8 downto 0));
end YUV_k_RAM;

architecture Behavioral of YUV_k_RAM is
   subtype YUV_k_WORD is std_logic_vector(8 downto 0);
   type YUV_k_ARRAY is array(0 to 63) of YUV_k_WORD;
   signal YUV_k : YUV_k_ARRAY;
   signal AD : std_logic_vector(5 downto 0);

begin
----------------------------
--
--   WRITE
--
----------------------------
   process(CLK)begin
      if(CLK'event and CLK = '1')then
         if(WP_YUV_k = '1')then
            YUV_k(CONV_INTEGER(AD_YUV_k)) <= IN_YUV_k;
         end if;
      end if;
   end process;

----------------------------
```

```
--
--   READ
--
-----------------------------
   process(CLK)begin
      if(CLK'event and CLK = '1')then
         if(RP_YUV_k = '1')then
            AD <= AD_YUV_k;
         end if;
      end if;
   end process;
   OUT_YUV_k <= YUV_k(CONV_INTEGER(AD));
end Behavioral;
```

5)　回路のシミュレーション記述

5-1)　量子化回路のシミュレーション記述

```
-----------------------------
--
-- Quantization TestBench
--
-----------------------------
LIBRARY ieee,STD;
USE STD.TEXTIO.ALL;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_textio.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

COMPONENT Quantization
    PORT ( CLK,RESET : in std_logic;
           START_QUANTIZATION : in std_logic;
           SCALE : in std_logic_vector(7 downto 0);
           DCTZ_1 : in std_logic_vector(12 downto 0);
           DCTZ_2 : in std_logic_vector(12 downto 0);
           DCTZ_3 : in std_logic_vector(12 downto 0);
           DCTZ_4 : in std_logic_vector(12 downto 0);
           DCTZ_5 : in std_logic_vector(12 downto 0);
           DCTZ_6 : in std_logic_vector(12 downto 0);
           QDCTZ_1 : out std_logic_vector(12 downto 0);
           QDCTZ_2 : out std_logic_vector(12 downto 0);
           QDCTZ_3 : out std_logic_vector(12 downto 0);
           QDCTZ_4 : out std_logic_vector(12 downto 0);
           QDCTZ_5 : out std_logic_vector(12 downto 0);
           QDCTZ_6 : out std_logic_vector(12 downto 0);
           START_DCTZ_IN : out std_logic;
           DONE_OUT_QDCTZ_K : out std_logic;
           DONE_QUANTIZATION : out std_logic);
```

```
END COMPONENT;

   file LNV : TEXT is in "Encode_dct_k_bit_outfile.txt";
   SIGNAL CLK,RESET : std_logic;
   SIGNAL START_QUANTIZATION : std_logic;
   SIGNAL SCALE : std_logic_vector(7 downto 0);
   SIGNAL DCTZ_1 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_2 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_3 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_4 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_5 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_6 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_1 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_2 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_3 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_4 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_5 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_6 : std_logic_vector(12 downto 0);
   SIGNAL START_DCTZ_IN : std_logic;
   SIGNAL DONE_OUT_QDCTZ_K : std_logic;
   SIGNAL DONE_QUANTIZATION : std_logic;

BEGIN
   uut: Quantization PORT MAP(
      CLK => CLK,
      RESET => RESET,
      START_QUANTIZATION => START_QUANTIZATION,
      SCALE => SCALE,
      DCTZ_1 => DCTZ_1,
      DCTZ_2 => DCTZ_2,
      DCTZ_3 => DCTZ_3,
      DCTZ_4 => DCTZ_4,
      DCTZ_5 => DCTZ_5,
      DCTZ_6 => DCTZ_6,
      QDCTZ_1 => QDCTZ_1,
      QDCTZ_2 => QDCTZ_2,
      QDCTZ_3 => QDCTZ_3,
      QDCTZ_4 => QDCTZ_4,
      QDCTZ_5 => QDCTZ_5,
      QDCTZ_6 => QDCTZ_6,
      START_DCTZ_IN => START_DCTZ_IN,
      DONE_OUT_QDCTZ_K => DONE_OUT_QDCTZ_K,
      DONE_QUANTIZATION => DONE_QUANTIZATION
   );

   PROCESS BEGIN
      CLK <= '1';
      wait for 5 ns;
      CLK <= '0';
      wait for 5 ns;
   END PROCESS;
```

```
      RESET <= '0';
      SCALE <= "00010000";
      START_QUANTIZATION <= '1';

   PROCESS
      variable LI : line;
      variable DCTZ_1_DATA,DCTZ_2_DATA,DCTZ_3_DATA,
               DCTZ_4_DATA,DCTZ_5_DATA,DCTZ_6_DATA : std_logic_vector(12 downto 0);
   BEGIN
      if(START_DCTZ_IN = '1')then
         readline(LNV, LI);
         read(LI, DCTZ_1_DATA);
         read(LI, DCTZ_2_DATA);
         read(LI, DCTZ_3_DATA);
         read(LI, DCTZ_4_DATA);
         read(LI, DCTZ_5_DATA);
         read(LI, DCTZ_6_DATA);
         DCTZ_1 <= DCTZ_1_DATA;
         DCTZ_2 <= DCTZ_2_DATA;
         DCTZ_3 <= DCTZ_3_DATA;
         DCTZ_4 <= DCTZ_4_DATA;
         DCTZ_5 <= DCTZ_5_DATA;
         DCTZ_6 <= DCTZ_6_DATA;
         wait for 10 ns;
         if(endfile(LNV))then
            wait;
         end if;
      else
         wait for 5 ns;
      end if;
   END PROCESS;
END;
```

5-2) 逆量子化回路のシミュレーション記述

```
------------------------------
--
-- DeQuantization TestBench
--
------------------------------
LIBRARY ieee,STD;
USE STD.TEXTIO.ALL;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_textio.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

COMPONENT DeQuantization
    PORT ( CLK,RESET : in std_logic;
```

```
            START_DEQUANTIZATION : in std_logic;
            SCALE : in std_logic_vector(7 downto 0);
            QDCTZ_1 : in std_logic_vector(12 downto 0);
            QDCTZ_2 : in std_logic_vector(12 downto 0);
            QDCTZ_3 : in std_logic_vector(12 downto 0);
            QDCTZ_4 : in std_logic_vector(12 downto 0);
            QDCTZ_5 : in std_logic_vector(12 downto 0);
            QDCTZ_6 : in std_logic_vector(12 downto 0);
            DCTZ_1 : out std_logic_vector(12 downto 0);
            DCTZ_2 : out std_logic_vector(12 downto 0);
            DCTZ_3 : out std_logic_vector(12 downto 0);
            DCTZ_4 : out std_logic_vector(12 downto 0);
            DCTZ_5 : out std_logic_vector(12 downto 0);
            DCTZ_6 : out std_logic_vector(12 downto 0);
            START_QDCTZ_IN : out std_logic;
            DONE_OUT_DCTZ_K : out std_logic;
            DONE_DEQUANTIZATION : out std_logic);
END COMPONENT;

   file LNV : TEXT is in "Encode_qdctz_k_bit_outfile.txt";
   SIGNAL CLK,RESET : std_logic;
   SIGNAL START_DEQUANTIZATION : std_logic;
   SIGNAL SCALE : std_logic_vector(7 downto 0);
   SIGNAL QDCTZ_1 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_2 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_3 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_4 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_5 : std_logic_vector(12 downto 0);
   SIGNAL QDCTZ_6 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_1 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_2 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_3 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_4 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_5 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_6 : std_logic_vector(12 downto 0);
   SIGNAL START_QDCTZ_IN : std_logic;
   SIGNAL DONE_OUT_DCTZ_K : std_logic;
   SIGNAL DONE_DEQUANTIZATION : std_logic;

BEGIN
   uut: DeQuantization PORT MAP(
      CLK => CLK,
      RESET => RESET,
      START_DEQUANTIZATION => START_DEQUANTIZATION,
      SCALE => SCALE,
      QDCTZ_1 => QDCTZ_1,
      QDCTZ_2 => QDCTZ_2,
      QDCTZ_3 => QDCTZ_3,
      QDCTZ_4 => QDCTZ_4,
      QDCTZ_5 => QDCTZ_5,
      QDCTZ_6 => QDCTZ_6,
      DCTZ_1 => DCTZ_1,
```

```
      DCTZ_2 => DCTZ_2,
      DCTZ_3 => DCTZ_3,
      DCTZ_4 => DCTZ_4,
      DCTZ_5 => DCTZ_5,
      DCTZ_6 => DCTZ_6,
      START_QDCTZ_IN => START_QDCTZ_IN,
      DONE_OUT_DCTZ_K => DONE_OUT_DCTZ_K,
      DONE_DEQUANTIZATION => DONE_DEQUANTIZATION
   );

   PROCESS BEGIN
      CLK <= '1';
      wait for 5 ns;
      CLK <= '0';
      wait for 5 ns;
   END PROCESS;

      RESET <= '0';
      SCALE <= "00010000";
      START_DEQUANTIZATION <= '1';


   PROCESS
      variable LI : line;
      variable QDCTZ_1_DATA,QDCTZ_2_DATA,QDCTZ_3_DATA,
               QDCTZ_4_DATA,QDCTZ_5_DATA,QDCTZ_6_DATA : std_logic_vector(12 downto 0);
   BEGIN
      if(START_QDCTZ_IN = '1')then
         readline(LNV, LI);
         read(LI, QDCTZ_1_DATA);
         read(LI, QDCTZ_2_DATA);
         read(LI, QDCTZ_3_DATA);
         read(LI, QDCTZ_4_DATA);
         read(LI, QDCTZ_5_DATA);
         read(LI, QDCTZ_6_DATA);
         QDCTZ_1 <= QDCTZ_1_DATA;
         QDCTZ_2 <= QDCTZ_2_DATA;
         QDCTZ_3 <= QDCTZ_3_DATA;
         QDCTZ_4 <= QDCTZ_4_DATA;
         QDCTZ_5 <= QDCTZ_5_DATA;
         QDCTZ_6 <= QDCTZ_6_DATA;
         wait for 10 ns;
         if(endfile(LNV))then
            wait;
         end if;
      else
         wait for 5 ns;
      end if;
   END PROCESS;
END;
```

5-3) DCT 変換回路のシミュレーション記述

```
-----------------------------
--
-- ForwardDct TestBench
--
-----------------------------
LIBRARY ieee,STD;
USE STD.TEXTIO.ALL;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_textio.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

COMPONENT ForwardDct
    Port ( CLK,RESET : in std_logic;
           START_FORWARDDCT : in std_logic;
           YUV_1 : in std_logic_vector(8 downto 0);
           YUV_2 : in std_logic_vector(8 downto 0);
           YUV_3 : in std_logic_vector(8 downto 0);
           YUV_4 : in std_logic_vector(8 downto 0);
           YUV_5 : in std_logic_vector(8 downto 0);
           YUV_6 : in std_logic_vector(8 downto 0);
           DCTZ_1 : out std_logic_vector(12 downto 0);
           DCTZ_2 : out std_logic_vector(12 downto 0);
           DCTZ_3 : out std_logic_vector(12 downto 0);
           DCTZ_4 : out std_logic_vector(12 downto 0);
           DCTZ_5 : out std_logic_vector(12 downto 0);
           DCTZ_6 : out std_logic_vector(12 downto 0);
           START_YUV_IN : out std_logic;
           DONE_OUT_DCTZ_K : out std_logic;
           DONE_FORWARDDCT : out std_logic);
END COMPONENT;

   file LNV : TEXT is in "Encode_yuv_k_bit_outfile.txt";
   SIGNAL CLK,RESET : std_logic;
   SIGNAL START_FORWARDDCT : std_logic;
   SIGNAL YUV_1 : std_logic_vector(8 downto 0);
   SIGNAL YUV_2 : std_logic_vector(8 downto 0);
   SIGNAL YUV_3 : std_logic_vector(8 downto 0);
   SIGNAL YUV_4 : std_logic_vector(8 downto 0);
   SIGNAL YUV_5 : std_logic_vector(8 downto 0);
   SIGNAL YUV_6 : std_logic_vector(8 downto 0);
   SIGNAL DCTZ_1 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_2 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_3 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_4 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_5 : std_logic_vector(12 downto 0);
   SIGNAL DCTZ_6 : std_logic_vector(12 downto 0);
```

```
   SIGNAL START_YUV_IN : std_logic;
   SIGNAL DONE_OUT_DCTZ_K : std_logic;
   SIGNAL DONE_FORWARDDCT : std_logic;

BEGIN
   uut: ForwardDct PORT MAP(
      CLK => CLK,
      RESET => RESET,
      START_FORWARDDCT => START_FORWARDDCT,
      YUV_1 => YUV_1,
      YUV_2 => YUV_2,
      YUV_3 => YUV_3,
      YUV_4 => YUV_4,
      YUV_5 => YUV_5,
      YUV_6 => YUV_6,
      DCTZ_1 => DCTZ_1,
      DCTZ_2 => DCTZ_2,
      DCTZ_3 => DCTZ_3,
      DCTZ_4 => DCTZ_4,
      DCTZ_5 => DCTZ_5,
      DCTZ_6 => DCTZ_6,
      START_YUV_IN => START_YUV_IN,
      DONE_OUT_DCTZ_K => DONE_OUT_DCTZ_K,
      DONE_FORWARDDCT => DONE_FORWARDDCT
   );

   PROCESS BEGIN
      CLK <= '1';
      wait for 5 ns;
      CLK <= '0';
      wait for 5 ns;
   END PROCESS;

      START_FORWARDDCT <= '1';
      RESET <= '0';

   PROCESS
      variable LI : line;
      variable YUV_1_DATA,YUV_2_DATA,YUV_3_DATA,
               YUV_4_DATA,YUV_5_DATA,YUV_6_DATA : std_logic_vector(8 downto 0);
   BEGIN
      if(START_YUV_IN = '1')then
         readline(LNV, LI);
         read(LI, YUV_1_DATA);
         read(LI, YUV_2_DATA);
         read(LI, YUV_3_DATA);
         read(LI, YUV_4_DATA);
         read(LI, YUV_5_DATA);
         read(LI, YUV_6_DATA);
         YUV_1 <= YUV_1_DATA;
         YUV_2 <= YUV_2_DATA;
         YUV_3 <= YUV_3_DATA;
```

```
         YUV_4 <= YUV_4_DATA;
         YUV_5 <= YUV_5_DATA;
         YUV_6 <= YUV_6_DATA;
         wait for 10 ns;
         if(endfile(LNV))then
            wait;
         end if;
      else
         wait for 5 ns;
      end if;
   END PROCESS;
END;
```

6) シミュレーションで使用するC言語記述

```
#include "rgbdata_01.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PAI 3.1415926536

#define HORIZONTALSIZE_CONTROL   16      /*horizontalsize control*/
#define VERTICALSIZE_CONTROL     16      /*verticalsize control*/
#define SCALE_CONTROL            1.0     /*scale control (0 - 7)*/

void encode(void);
void decode(void);
void encode_outfile(void);
void encode_yuv_k_bit_outfile(void);
void encode_dct_k_bit_outfile(void);
void encode_qdctz_k_bit_outfile(void);
void decode_outfile(void);

void rgbfileout(void);
void yuvfileout(void);
void yuv_kfileout(void);
void dct_kfileout(void);
void qdctz_kfileout(void);

void convertrgbyuv(void);
void composemcu(void);
void decomposemcu(void);
void convertyuvrgb(void);
void forwarddct(void);
void quantization(void);
void dequantization(void);
void inversedct(void);

void carrybit(int number,int bits);

FILE *str_ptr;
```

```
int horizontalsize = HORIZONTALSIZE_CONTROL,
   verticalsize = VERTICALSIZE_CONTROL;

double scale = SCALE_CONTROL;

int y[16][16],u[16][16],v[16][16];

int yuv_1[8][8],yuv_2[8][8],yuv_3[8][8],
   yuv_4[8][8],yuv_5[8][8],yuv_6[8][8];

int dct_1[8][8],dct_2[8][8],dct_3[8][8],
   dct_4[8][8],dct_5[8][8],dct_6[8][8];

int qdctz_1[64],qdctz_2[64],qdctz_3[64],
   qdctz_4[64],qdctz_5[64],qdctz_6[64];

int luminanceqtable[64] = { 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};

int   chrominanceqtable[64] = {   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};

//////////////////////////////
//
//   main()
//
//////////////////////////////
main(){
   encode();
   encode_outfile();
   encode_yuv_k_bit_outfile();
   encode_dct_k_bit_outfile();
   encode_qdctz_k_bit_outfile();

   decode();
   decode_outfile();
   return(0);
}

//////////////////////////////
```

```
//
//   encode()
//
////////////////////////////
void encode(void)
{
   convertrgbyuv();
   composemcu();
   forwarddct();
   quantization();

   printf("Encode end\n");                          /* Display Encode end */
}

////////////////////////////
//
//   encode_yuv_k_bit_outfile()
//
////////////////////////////
void encode_yuv_k_bit_outfile(void)
{
   int yy,xx;
   printf("Encode_yuv_bit_outfile start\n");       /* Display Encode_rgb_bit_outfile start */
   str_ptr = fopen("Encode_yuv_k_bit_outfile.txt","w");

   for(yy=0 ; yy<8 ; yy++){
      for(xx=0 ; xx<8 ; xx++){
         carrybit(yuv_1[yy][xx],9);
         fputs(" ",str_ptr);
         carrybit(yuv_2[yy][xx],9);
         fputs(" ",str_ptr);
         carrybit(yuv_3[yy][xx],9);
         fputs(" ",str_ptr);
         carrybit(yuv_4[yy][xx],9);
         fputs(" ",str_ptr);
         carrybit(yuv_5[yy][xx],9);
         fputs(" ",str_ptr);
         carrybit(yuv_6[yy][xx],9);
         fputs("\n",str_ptr);
      }
   }
   fclose(str_ptr);
   printf("Encode_yuv_k_bit_outfile end\n");       /* Encode_rgb_bit_outfile end */
}

////////////////////////////
//
//   encode_dct_k_bit_outfile()
//
////////////////////////////
void encode_dct_k_bit_outfile(void)
{
```

```
   int yy,xx;
   printf("Encode_dctz_bit_outfile start\n");      /* Display Encode_dctz_k_bit_outfile start */
   str_ptr = fopen("Encode_dct_k_bit_outfile.txt","w");

   for(yy=0 ; yy<8 ; yy++){
      for(xx=0 ; xx<8 ; xx++){
         carrybit(dct_1[yy][xx],13);
         fputs(" ",str_ptr);
         carrybit(dct_2[yy][xx],13);
         fputs(" ",str_ptr);
         carrybit(dct_3[yy][xx],13);
         fputs(" ",str_ptr);
         carrybit(dct_4[yy][xx],13);
         fputs(" ",str_ptr);
         carrybit(dct_5[yy][xx],13);
         fputs(" ",str_ptr);
         carrybit(dct_6[yy][xx],13);
         fputs("\n",str_ptr);
      }
   }
   fclose(str_ptr);
   printf("Encode_dctz_k_bit_outfile end\n");      /* Encode_dctz_k_bit_outfile end */
}

//////////////////////////////
//
//   encode_qdctz_k_bit_outfile()
//
//////////////////////////////
void encode_qdctz_k_bit_outfile(void)
{
   int yy;
                                                  /* Display Encode_qdctz_k_bit_outfile start */
   printf("Encode_qdctz_bit_outfile start\n");

   str_ptr = fopen("Encode_qdctz_k_bit_outfile.txt","w");

   for(yy=0 ; yy<64 ; yy++){
      carrybit(qdctz_1[yy],13);
      fputs(" ",str_ptr);
      carrybit(qdctz_2[yy],13);
      fputs(" ",str_ptr);
      carrybit(qdctz_3[yy],13);
      fputs(" ",str_ptr);
      carrybit(qdctz_4[yy],13);
      fputs(" ",str_ptr);
      carrybit(qdctz_5[yy],13);
      fputs(" ",str_ptr);
      carrybit(qdctz_6[yy],13);
      fputs("\n",str_ptr);
   }
```

```
   fclose(str_ptr);
                                                /* Encode_qdctz_k_bit_outfile end */
   printf("Encode_qdctz_k_bit_outfile end\n");
}

////////////////////////////
//
//   decode()
//
////////////////////////////
void decode(void)
{
   printf("Decode start\n");                    /* Display Decode start */

   dequantization();
   inversedct();
   decomposemcu();
   convertyuvrgb();

                                                /* Display Decode out */
   printf("Decode end\n");
}

////////////////////////////
//
//   encode_outfile()
//
////////////////////////////
void encode_outfile(void)
{
   printf("Encode_outfile start\n");            /* Display Encode_outfile start */

   str_ptr = fopen("Encode_outfile.txt","w");

   fprintf(str_ptr,"horizontalsize = %4d\n",horizontalsize);
   fprintf(str_ptr,"verticalsize = %4d\n",verticalsize);
   fprintf(str_ptr,"scale = %1.4f\n\n",scale);

   fputs("r g b\n", str_ptr);                   /*rgb file out*/
   rgbfileout();

   fputs("convertrgbyuv\n", str_ptr);           /*convertrgbyuv file out*/

   fputs("y u v\n", str_ptr);                   /*yuv file out*/
   yuvfileout();

   fputs("composemcu\n", str_ptr);              /*composemcu file out*/

   fputs("yuv_k\n", str_ptr);                   /*yuv_k file out*/
   yuv_kfileout();

   fputs("forwarddct\n", str_ptr);              /*forwarddct file out*/
```

```
   fputs("dct_k\n", str_ptr);                    /*dct_k file out*/
   dct_kfileout();

   fputs("quantization\n", str_ptr);             /*quantization file out*/

   fputs("qdctz_k\n", str_ptr);                  /*qdctz_k file out*/
   qdctz_kfileout();

   fclose(str_ptr);

   printf("Encode_outfile end\n");               /* Display Encode_outfile end */
}

//////////////////////////////
//
//   decode_outfile()
//
//////////////////////////////
void decode_outfile(void)
{
   printf("Decode_outfile start\n");             /* Display Decode_outfile start */

   str_ptr = fopen("Decode_outfile.txt","w");

   fputs("decode start\n\n", str_ptr);

   fputs("dequantization\n", str_ptr);           /*dequantization file out*/

   fputs("dct_k\n", str_ptr);                    /*dct_k file out*/
   dct_kfileout();

   fputs("inversedct\n", str_ptr);               /*inversedct file out*/

   fputs("yuv_k\n", str_ptr);                    /*yuv_k file out*/
   yuv_kfileout();

   fputs("decomposemcu\n", str_ptr);             /*decomposemcu file out*/

   fputs("y u v\n", str_ptr);                    /*yuv file out*/
   yuvfileout();

   fputs("convertyuvrgb\n", str_ptr);            /*convertyuvrgb file out*/

   fputs("r g b\n", str_ptr);                    /*rgb file out*/
   rgbfileout();

   fclose(str_ptr);

   printf("Decode_outfile end\n");               /* Display Decode_outfile end */
}
```

```
//////////////////////////////
//
//   rgbfileout()
//
//////////////////////////////
void rgbfileout(void)
{
   int yy,xx;

   /*r*/
   fputs("r \n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 16 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 16 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 16 ; xx++)
         fprintf(str_ptr,"%4d",r[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*g*/
   fputs("g\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 16 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 16 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 16 ; xx++)
         fprintf(str_ptr,"%4d",g[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*b*/
   fputs("b \n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 16 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 16 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 16 ; xx++)
         fprintf(str_ptr,"%4d",b[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);
}
```

```
////////////////////////////
//
//   yuvfileout()
//
////////////////////////////
void yuvfileout(void)
{
   int yy,xx;
   /*y*/
   fputs("y\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 16 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 16 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 16 ; xx++)
         fprintf(str_ptr,"%4d",y[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*u*/
   fputs("u\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 16 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 16 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 16 ; xx++)
         fprintf(str_ptr,"%4d",u[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*v*/
   fputs("v\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 16 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 16 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 16 ; xx++)
         fprintf(str_ptr,"%4d",v[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);
}

////////////////////////////
```

```
//
//   yuv_kfileout()
//
//////////////////////////////
void yuv_kfileout(void)
{
   int yy,xx;
   /*yuv_1*/
   fputs("yuv_1\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",yuv_1[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*yuv_2*/
   fputs("yuv_2\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",yuv_2[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*yuv_3*/
   fputs("yuv_3\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",yuv_3[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*yuv_4*/
   fputs("yuv_4\n", str_ptr);
   fputs("    ", str_ptr);
```

```
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",yuv_4[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*yuv_5*/
   fputs("yuv_5\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",yuv_5[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*yuv_6*/
   fputs("yuv_6\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",yuv_6[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);
}

//////////////////////////////
//
//   dct_kfileout()
//
//////////////////////////////
void dct_kfileout(void)
{
   int yy,xx;
   /*dct_1*/
   fputs("dct_1\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
```

```
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",dct_1[yy][xx]);
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);

/*dct_2*/
fputs("dct_2\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",dct_2[yy][xx]);
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);

/*dct_3*/
fputs("dct_3\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",dct_3[yy][xx]);
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);

/*dct_4*/
fputs("dct_4\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",dct_4[yy][xx]);
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);
```

```
   /*dct_5*/
   fputs("dct_5\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",dct_5[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*dct_6*/
   fputs("dct_6\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++)
         fprintf(str_ptr,"%4d",dct_6[yy][xx]);
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);
}

//////////////////////////////
//
//   qdctz_kfileout()
//
//////////////////////////////
void qdctz_kfileout(void)
{
   int yy,xx;
   /*qdctz_1*/
   fputs("qdctz_1\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++){
         fprintf(str_ptr,"%4d",qdctz_1[yy*8+xx]);
      }
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);
```

```
/*qdctz_2*/
fputs("qdctz_2\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++){
      fprintf(str_ptr,"%4d",qdctz_2[yy*8+xx]);
   }
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);

/*qdctz_3*/
fputs("qdctz_3\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++){
      fprintf(str_ptr,"%4d",qdctz_3[yy*8+xx]);
   }
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);

/*qdctz_4*/
fputs("qdctz_4\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
for(yy=0 ; yy < 8 ; yy++){
   fprintf(str_ptr,"%4d",yy);
   for(xx=0 ; xx < 8 ; xx++){
      fprintf(str_ptr,"%4d",qdctz_4[yy*8+xx]);
   }
fputs("\n", str_ptr);
}
fputs("\n", str_ptr);

/*qdctz_5*/
fputs("qdctz_5\n", str_ptr);
fputs("    ", str_ptr);
for(xx=0 ; xx < 8 ; xx++)
   fprintf(str_ptr,"%4d",xx);
fputs("\n", str_ptr);
```

```
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++){
         fprintf(str_ptr,"%4d",qdctz_5[yy*8+xx]);
      }
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);

   /*qdctz_6*/
   fputs("qdctz_6\n", str_ptr);
   fputs("    ", str_ptr);
   for(xx=0 ; xx < 8 ; xx++)
      fprintf(str_ptr,"%4d",xx);
   fputs("\n", str_ptr);
   for(yy=0 ; yy < 8 ; yy++){
      fprintf(str_ptr,"%4d",yy);
      for(xx=0 ; xx < 8 ; xx++){
         fprintf(str_ptr,"%4d",qdctz_6[yy*8+xx]);
      }
   fputs("\n", str_ptr);
   }
   fputs("\n", str_ptr);
}

//////////////////////////////
//
//   convertrgbyuv()
//
//////////////////////////////
void convertrgbyuv(void)
{
   int yy,xx;
   double yf[16][16],uf[16][16],vf[16][16];

   for(yy=0 ; yy < 16 ; yy++){
      for(xx=0 ; xx < 16 ; xx++){
         y[yy][xx]=0;            /*y[16][16] の初期化*/
         u[yy][xx]=0;            /*u[16][16] の初期化*/
         v[yy][xx]=0;            /*v[16][16] の初期化*/
      }
   }

   for(yy=0 ; yy < verticalsize ; yy++){
      for(xx=0 ; xx < horizontalsize ; xx++){
         yf[yy][xx] = 0.0000;
         uf[yy][xx] = 0.0000;
         vf[yy][xx] = 0.0000;

         yf[yy][xx] = (double)r[yy][xx] * 0.2990
            + (double)g[yy][xx] * 0.5870
            + (double)b[yy][xx] * 0.1140;
```

```
         y[yy][xx] = (int)yf[yy][xx];
         uf[yy][xx] =-(double)r[yy][xx] * 0.1684
            - (double)g[yy][xx] * 0.3316
            + (double)b[yy][xx] * 0.5000;
         u[yy][xx] = (int)uf[yy][xx];
         vf[yy][xx] = (double)r[yy][xx] * 0.5000
            - (double)g[yy][xx] * 0.4187
            - (double)b[yy][xx] * 0.0813;
         v[yy][xx] = (int)vf[yy][xx];
      }
   }
}

////////////////////////////
//
//   composemcu()
//
////////////////////////////
void composemcu(void)
{
   int yy,xx;
   for(yy=0 ; yy<8 ; yy++){
      for(xx=0 ; xx<8 ; xx++){
         yuv_1[yy][xx] = y[yy][xx] - 128;
         yuv_2[yy][xx] = y[yy][xx+8] - 128;
         yuv_3[yy][xx] = y[yy+8][xx] - 128;
         yuv_4[yy][xx] = y[yy+8][xx+8] - 128;
         yuv_5[yy][xx] = u[yy*2][xx*2];
         yuv_6[yy][xx] = v[yy*2][xx*2];
      }
   }
}

////////////////////////////
//
//   decomposemcu()
//
////////////////////////////
void decomposemcu(void)
{
   int yy,xx;
   for(yy=0 ; yy<8 ; yy++){
      for(xx=0 ; xx<8 ; xx++){
         y[yy][xx] = yuv_1[yy][xx] + 128;
         y[yy][xx+8] = yuv_2[yy][xx] + 128;
         y[yy+8][xx] = yuv_3[yy][xx] + 128;
         y[yy+8][xx+8] = yuv_4[yy][xx] + 128;
         u[yy*2][xx*2] = u[yy*2][xx*2+1]
            =u[yy*2+1][xx*2] = u[yy*2+1][xx*2+1] = yuv_5[yy][xx];
         v[yy*2][xx*2] = v[yy*2][xx*2+1]
            =v[yy*2+1][xx*2] = v[yy*2+1][xx*2+1] = yuv_6[yy][xx];
      }
```

```
   }
}

//////////////////////////////
//
//   convertyuvrgb()
//
//////////////////////////////
void convertyuvrgb(void)
{
   int yy,xx;
   double rf[16][16],gf[16][16],bf[16][16];
   for(yy=0 ; yy < verticalsize ; yy++){
      for(xx=0 ; xx < horizontalsize ; xx++){
         rf[yy][xx] = (double)y[yy][xx] + (double)v[yy][xx] * 1.4020;
         r[yy][xx] = (int)rf[yy][xx];
         if(r[yy][xx] < 0)
            r[yy][xx] = 0;
         else if(r[yy][xx] > 255)
            r[yy][xx] = 255;

         gf[yy][xx] = (double)y[yy][xx] - (double)u[yy][xx] * 0.3441 - (double)v[yy][xx] * 0.7139;
         g[yy][xx] = (int)gf[yy][xx];
         if(g[yy][xx] < 0)
            g[yy][xx] = 0;
         else if(g[yy][xx] > 255)
            g[yy][xx] = 255;

         bf[yy][xx] = (double)y[yy][xx] + (double)u[yy][xx] * 1.7718 - (double)v[yy][xx] * 0.0012;
         b[yy][xx] = (int)bf[yy][xx];
         if(b[yy][xx] < 0)
            b[yy][xx] = 0;
         else if(b[yy][xx] > 255)
            b[yy][xx] = 255;
      }
   }
}

//////////////////////////////
//
//   forwarddct()
//
//////////////////////////////
void forwarddct(void)
{
   int vv,uu,yy,xx;
   double cv,cu;
   double sum_1,sum_2,sum_3,
         sum_4,sum_5,sum_6;

   for(vv=0 ; vv<8 ; vv++){
      if(vv == 0)
```

```
         cv = 1.0 / sqrt(2.0);
      else
         cv = 1.0;
      for(uu=0 ; uu<8 ; uu++){
         if(uu == 0)
            cu = 1.0 /  sqrt(2.0);
         else
            cu = 1.0;

         sum_1 =  sum_2 =  sum_3 =
         sum_4 =  sum_5 =  sum_6 = 0.0;

         for(yy=0 ; yy<8 ; yy++){
            for(xx=0 ; xx<8 ; xx++){
               sum_1 = sum_1 + (double)yuv_1[yy][xx]
                  *cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  *cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);

               sum_2 = sum_2 + (double)yuv_2[yy][xx]
                  *cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  *cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);

               sum_3 = sum_3 + (double)yuv_3[yy][xx]
                  *cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  *cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);

               sum_4 = sum_4 + (double)yuv_4[yy][xx]
                  *cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  *cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);

               sum_5 = sum_5 + (double)yuv_5[yy][xx]
                  *cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  *cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);

               sum_6 = sum_6 + (double)yuv_6[yy][xx]
                  *cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  *cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
            }
         }
         dct_1[vv][uu] = (int)(sum_1 * cu * cv / 4.0);
         dct_2[vv][uu] = (int)(sum_2 * cu * cv / 4.0);
         dct_3[vv][uu] = (int)(sum_3 * cu * cv / 4.0);
         dct_4[vv][uu] = (int)(sum_4 * cu * cv / 4.0);
         dct_5[vv][uu] = (int)(sum_5 * cu * cv / 4.0);
         dct_6[vv][uu] = (int)(sum_6 * cu * cv / 4.0);
      }
   }
}

////////////////////////////
//
//   quantization()
```

```
//
//////////////////////////////
void quantization(void)
{
   int vv,uu,ii;
   double stepsize[64];
   int dctz_1[64],dctz_2[64],dctz_3[64],
      dctz_4[64],dctz_5[64],dctz_6[64];
   double sign1,sign2,sign3,sign4,sign5,sign6;

   for(vv=0 ; vv<8 ; vv++){
      for(uu=0 ; uu<8 ; uu++){
         dctz_1[vv*8+uu] = dct_1[vv][uu];
         dctz_2[vv*8+uu] = dct_2[vv][uu];
         dctz_3[vv*8+uu] = dct_3[vv][uu];
         dctz_4[vv*8+uu] = dct_4[vv][uu];
         dctz_5[vv*8+uu] = dct_5[vv][uu];
         dctz_6[vv*8+uu] = dct_6[vv][uu];
      }
   }
   for(ii=0 ; ii<63 ; ii++){
      if(dctz_1[ii] == 0)
         sign1 = 0.0;
      else if(dctz_1[ii] > 0)
         sign1 = 1.0;
      else
         sign1 = -1.0;

      if(dctz_2[ii] == 0)
         sign2 = 0.0;
      else if(dctz_2[ii] > 0)
         sign2 = 1.0;
      else
         sign2 = -1.0;

      if(dctz_3[ii] == 0)
         sign3 = 0.0;
      else if(dctz_3[ii] > 0)
         sign3 = 1.0;
      else
         sign3 = -1.0;

      if(dctz_4[ii] == 0)
         sign4 = 0.0;
      else if(dctz_4[ii] > 0)
         sign4 = 1.0;
      else
         sign4 = -1.0;

      if(dctz_5[ii] == 0)
         sign5 = 0.0;
      else if(dctz_5[ii] > 0)
```

```
         sign5 = 1.0;
      else
         sign5 = -1.0;

      if(dctz_6[ii] == 0)
         sign6 = 0.0;
      else if(dctz_6[ii] > 0)
         sign6 = 1.0;
      else
         sign6 = -1.0;

      stepsize[ii] = (double)luminanceqtable[ii] * (double)scale;
      qdctz_1[ii] = (int)(((double)dctz_1[ii] + sign1 * stepsize[ii] / 2.0) / stepsize[ii]);
      qdctz_2[ii] = (int)(((double)dctz_2[ii] + sign2 * stepsize[ii] / 2.0) / stepsize[ii]);
      qdctz_3[ii] = (int)(((double)dctz_3[ii] + sign3 * stepsize[ii] / 2.0) / stepsize[ii]);
      qdctz_4[ii] = (int)(((double)dctz_4[ii] + sign4 * stepsize[ii] / 2.0) / stepsize[ii]);

      stepsize[ii] = chrominanceqtable[ii] * (double)scale;
      qdctz_5[ii] = (int)(((double)dctz_5[ii] + sign5 * stepsize[ii] / 2.0) / stepsize[ii]);
      qdctz_6[ii] = (int)(((double)dctz_6[ii] + sign6 * stepsize[ii] / 2.0) / stepsize[ii]);
   }
}

//////////////////////////////
//
//   dequantization()
//
//////////////////////////////
void dequantization(void)
{
   int vv,uu,ii;
   double stepsize[64];
   double dctz_1[64],dctz_2[64],dctz_3[64],
         dctz_4[64],dctz_5[64],dctz_6[64];

   for(ii=0 ; ii<63 ; ii++){
      stepsize[ii] = (double)luminanceqtable[ii] * (double)scale;
      dctz_1[ii] = (double)qdctz_1[ii] * stepsize[ii];
      dctz_2[ii] = (double)qdctz_2[ii] * stepsize[ii];
      dctz_3[ii] = (double)qdctz_3[ii] * stepsize[ii];
      dctz_4[ii] = (double)qdctz_4[ii] * stepsize[ii];

      stepsize[ii] = (double)chrominanceqtable[ii] * (double)scale;
      dctz_5[ii] = (double)qdctz_5[ii] * stepsize[ii];
      dctz_6[ii] = (double)qdctz_6[ii] * stepsize[ii];
   }

   for(vv=0 ; vv<8 ; vv++){
      for(uu=0 ; uu<8 ; uu++){
         dct_1[vv][uu] = (int)(dctz_1[vv*8+uu]);
         dct_2[vv][uu] = (int)(dctz_2[vv*8+uu]);
         dct_3[vv][uu] = (int)(dctz_3[vv*8+uu]);
```

```
         dct_4[vv][uu] = (int)(dctz_4[vv*8+uu]);
         dct_5[vv][uu] = (int)(dctz_5[vv*8+uu]);
         dct_6[vv][uu] = (int)(dctz_6[vv*8+uu]);
      }
   }
}

////////////////////////////
//
//   inversedct()
//
////////////////////////////
void inversedct(void)
{
   int yy,xx,vv,uu;
   double cv,cu;
   double sum_1,sum_2,sum_3,
         sum_4,sum_5,sum_6;

   for(yy=0 ; yy<8 ; yy++){
      for(xx=0 ; xx<8 ; xx++){
         sum_1 =  sum_2 =  sum_3 =
         sum_4 =  sum_5 =  sum_6 = 0.0;
         for(vv=0 ; vv<8 ; vv++){
            if(vv == 0)
               cv = 1.0 / sqrt(2.0);
            else
               cv = 1.0;
            for(uu=0 ; uu<8 ; uu++){
               if(uu == 0)
                  cu = 1.0 / sqrt(2.0);
               else
                  cu = 1.0;
               sum_1 = sum_1 + cu * cv * (double)dct_1[vv][uu]
                  * cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  * cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
               sum_2 = sum_2 + cu * cv * (double)dct_2[vv][uu]
                  * cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  * cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
               sum_3 = sum_3 + cu * cv * (double)dct_3[vv][uu]
                  * cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  * cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
               sum_4 = sum_4 + cu * cv * (double)dct_4[vv][uu]
                  * cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  * cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
               sum_5 = sum_5 + cu * cv * (double)dct_5[vv][uu]
                  * cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  * cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
               sum_6 = sum_6 + cu * cv * (double)dct_6[vv][uu]
                  * cos((2.0 * (double)xx + 1.0) * (double)uu * PAI / 16.0)
                  * cos((2.0 * (double)yy + 1.0) * (double)vv * PAI / 16.0);
            }
```

```
         }
         yuv_1[yy][xx] = (int)(sum_1 / 4.0);
         yuv_2[yy][xx] = (int)(sum_2 / 4.0);
         yuv_3[yy][xx] = (int)(sum_3 / 4.0);
         yuv_4[yy][xx] = (int)(sum_4 / 4.0);
         yuv_5[yy][xx] = (int)(sum_5 / 4.0);
         yuv_6[yy][xx] = (int)(sum_6 / 4.0);
      }
   }
}

////////////////////////////
//
//   carrybit()
//
////////////////////////////
void carrybit(int number,int bits)
{
   int ii;
   float fnumber;
   char bit[17];

   fnumber = (float)(abs(number));

   for(ii=bits-1 ; ii>=0 ; ii--){
      if(ii == 0){
         if(number >= 0)
            bit[ii] = '0';
         else
            bit[ii] = '1';
      }
      else{
         if((int)fnumber == 0)
            bit[ii] = '0';
         else if(fnumber/2 == (int)fnumber/2)
            bit[ii] = '0';
         else
            bit[ii] = '1';
         fnumber = (float)((int)(fnumber/2.0));
      }
   }
   for(ii=0 ; ii<bits ; ii++)
      fprintf(str_ptr,"%c",bit[ii]);
}
```

6-1) rgbdata_01.h の記述

```
int r[16][16] = {{255, 255, 255, 255, 255, 255, 255, 255, /*00*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*01*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*02*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*03*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*04*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*05*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*06*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*07*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*08*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*09*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*10*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*11*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*12*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*13*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*14*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*16*/
                    5,   5,   5,   5,   5,   5,   5,   5}};

int g[16][16] = {{255, 255, 255, 255, 255, 255, 255, 255, /*00*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*01*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*02*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*03*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*04*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*05*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*06*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*07*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*08*/
                  255, 255, 255, 255, 255, 255, 255, 255},
```

```
                    {  5,   5,   5,   5,   5,   5,   5,   5, /*09*/
                       5,   5,   5,   5,   5,   5,   5,   5},
                    {255, 255, 255, 255, 255, 255, 255, 255, /*10*/
                     255, 255, 255, 255, 255, 255, 255, 255},
                    {  5,   5,   5,   5,   5,   5,   5,   5, /*11*/
                       5,   5,   5,   5,   5,   5,   5,   5},
                    {255, 255, 255, 255, 255, 255, 255, 255, /*12*/
                     255, 255, 255, 255, 255, 255, 255, 255},
                    {  5,   5,   5,   5,   5,   5,   5,   5, /*13*/
                       5,   5,   5,   5,   5,   5,   5,   5},
                    {255, 255, 255, 255, 255, 255, 255, 255, /*14*/
                     255, 255, 255, 255, 255, 255, 255, 255},
                    {  5,   5,   5,   5,   5,   5,   5,   5, /*16*/
                       5,   5,   5,   5,   5,   5,   5,   5}};

int b[16][16] = {{255, 255, 255, 255, 255, 255, 255, 255, /*00*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*01*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*02*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*03*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*04*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*05*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*06*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*07*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*08*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*09*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*10*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*11*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*12*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*13*/
                    5,   5,   5,   5,   5,   5,   5,   5},
                 {255, 255, 255, 255, 255, 255, 255, 255, /*14*/
                  255, 255, 255, 255, 255, 255, 255, 255},
                 {  5,   5,   5,   5,   5,   5,   5,   5, /*16*/
                    5,   5,   5,   5,   5,   5,   5,   5}};
```