卒業研究報告

題　目

組織的ディザ法による画像フィルタおよび
RunLength エンコーダの設計

指導教員

橘　昌良　助教授

報告者

野口　輝幸

平成 14 年 2 月 7 日

高知工科大学　電子･光システム工学科

目次

1 章　はじめに

近年通信網の目覚しい発達により、従来の音声のみならず動画のような大量の情報量を有するデータなどでも遠隔地へ伝送することが可能になってきている。
しかし、通信網においてその伝送経路の利用できる帯域というものは限られたものであり、その限られた通信網の帯域を利用して、いかにして大量の情報を伝送するかという事が非常に重要となってきている。
そこで考えられるものが伝送すべきデータを何らかの手法を用いて加工することである。その手法として扱うべきデータを“圧縮する”また“不要である情報を削る”という方法が上げられる。ことにより伝送する情報量を少なくすることが可能となる。

本研究ではそれら情報処理についての理解を深め、ハードウェア記述言語である VHDL[1] を用いて実際にハードウェアで設計することで、その言語仕様および順序回路の設計における一貫した作業プロセスを習得する。また、Windows プログラミングによる評価プログラムの作成などを行い、実際のデータ構造の理解やプログラミングに関する知識を習得することを目的とする。また、画像フィルタの試作などを試み画像処理に対する理解を深める。

まず、本報告書の 2 章では各種圧縮アルゴリズムの比較を行い、ハードウェアでの実装に関して最も適したアルゴリズムの考察を行う。今回は画像の処理を行うため、3 章および 4 章にて画像がどのような構造をもって記録されているか、またこれをハードウェアで扱うための手法などの考察を行う。5 章および 6 章では実際にの画像処理に関する手法および考察を行い、これらを VHDL で記述し、その回路のシミュレーションを行う。

[1] Very Highspeed Integrated Circuit（VHSIC）Hardware　Discription Language

2 章　エントロピーと情報の圧縮

２－１ 情報エントロピー

ある情報を損失なく圧縮する際、情報にはこれ以上圧縮することができない限界が存在する。この限界値を示すものがエントロピーである。理想的なデータ圧縮とは如何にしてこのエントロピーに達するかということである。

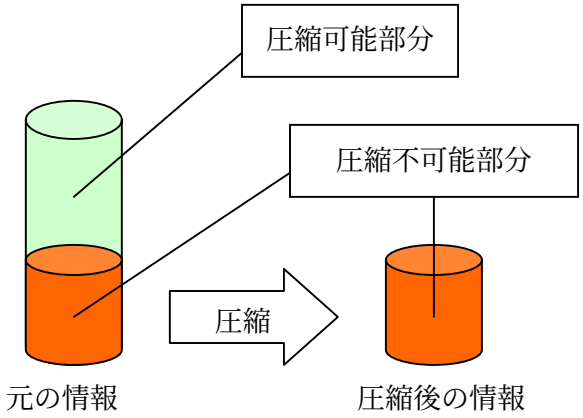


図 2-1

２－２　圧縮アルゴリズムについて

データ圧縮とは、情報の冗長性を取り除き必要最小限のデータ量に減らすという考え方である。データ圧縮には数多くのアルゴリズムが存在し、それらは可逆圧縮と非可逆圧縮に分類することができる。可逆圧縮とは損失のない圧縮法であり、非可逆圧縮とは損失のある圧縮法である。

代表的な可逆圧縮アルゴリズムについては以下のようなものが存在する。

◇ 静的ハフマン符号法　(Static Huffman Coding)

圧縮されるデータを走査し、最も多く出現する文字に対しては短いコードを出現率の少ない文字に対しては短いコードを割り当てることでデータを圧縮する方法である。

ハフマン符号法では、文字列を重さを持った“ 葉 ”と想定し、データ全体を走査することで、その葉の重さを決定する。その葉の重さから、“ ハフマンの木 “ というツリー構造のリストを構築する。ここで、” 葉の重さ “ とは、各文字列の出現率を表し、最も出現率の多いものが重たい葉ということになる。

◇適応型ハフマン符号法　(Adaptive Huffman Coding)

上記静的ハフマン符号法の改良型アルゴリズムである。静的ハフマン符号法では、一度圧縮するデータを走査してハフマンの木を構築する必要があったため 2 パスの処理を必要としデータをオンラインで圧縮できないという欠点があったが、この適応型ハフマンでは、入力されるデータに応じでそのハフマンの木を再構築してゆくためオンラインでの圧縮が可能となっている。

◇静的辞書圧縮法　(Static dictionary based compression)

この方法では、まず読み込まれたデータを辞書として登録し、その中から最も長い一致文字列を探し出す。そしてその“位置”と“一致した長さ”で元の情報を置き換えることで、データを圧縮する方法である。適用範囲が一部に限られる局所的な辞書を使用している。

◇動的辞書圧縮法　（Dynamic dictionary based compression）

上記静的辞書圧縮法の改良版であり、圧縮に使われた文字列だけを辞書として記憶することにより、大域的な辞書の作成を行っている。これは国語辞典に例えると、索引の中から一致する記号列を探して置き換えることと考えることができる。

◇連長圧縮法 (Run Length Encoding)

連続したデータを“コード”と“連続数”に置き換えることでデータを圧縮する。このため、２値の画像データのような特定のコードがある程度連続して出現するものに対して非常に高い圧縮率が期待できる。逆にコードに連続性がないと圧縮率は期待できない。

これらの圧縮アルゴリズムを実際にハードウェアで実装する際、ハードウェアの特性上多くの問題が発生する。よって設計に際しては、このハードウェア面での制限をクリアーしたアルゴリズムを選択する必要がある。中でももっとも考慮すべきが回路規模である。メモリを大量に使用するものや、複雑な演算を必要とするもは VHDL で記述した場合、大量のロジックを生成してしまうことがある。

これらの圧縮アルゴリズムの中でも連長圧縮法は非常にハードウェアで設計しやすいアルゴリズムであるといえる。２パスの処理を必要とせず、かつ大量の内部メモリを必要としない。さらにアルゴリズムも容易であるため極めて小規模な回路で実現できる。よって本研究では連長圧縮法のアルゴリズムを用いてハードウェアエンコーダを設計することとした。

３章　画像データについて

３－１　画像フォーマットの解析

今回､エンコーダの性能評価に用いるための画像データとして BMP[2]ファイルを用いることとした。論理回路では情報を“１”と“0”の符号で扱うため、処理する画像情報を論理回路で扱える形に変換する必要がある。また、ここで BMP のファイルフォーマットを解析するため、簡単なファイルダンパーを製作した。このプログラムは処理する対象のファイルに対してバイナリ[3]モードで１バイトずつデータをリードしていき、その読み込まれたデータの基数を変換し 2 進数 8bit 固定長の符号として TXT 形式[4]で出力するものである。

[2]ビットマップファイル（BitMaP）Windows 環境で標準的に使用される画像ファイルフォーマット
[3]「binary」は「２進法の」の意味。ただし「バイナリ」という語は、ASCII コードからなる文字データと相対するものとして用いられることもある。
[4]基本的にすべてがキャラクタコードからなるファイル。改行やファイルの終端などにはキャラクタ以外の制御コードが入っている。

３－２　ビットマップファイルフォーマットについて

以下は BMP ファイルフォーマットの仕様である。

ファイルヘッダ	bfType	2 byte	ファイルタイプ	BM
	bfSize	4 byte	ファイルサイズ	byte
	bfReserved1	2 byte	予約領域	
	bfReserved2	2 byte	予約領域	
	bfOffBits	4 byte	ファイル先頭から画像データまでのオフセット	byte
情報ヘッダー	biSize	4 byte	情報ヘッダーのサイズ	40 (byte)
	biWidth	4 byte	画像の幅	ピクセル
	biHeight	4 byte	画像の高さ	ピクセル
	biPlanes	2 byte	プレーン数	1
	biBitCount	2 byte	1 画素あたりのデータサイズ	bit
	biCopmression	4 byte	圧縮形式	
	biSizeImage	4 byte	画像データ部のサイズ	byte
	biXPixPerMeter	4 byte	横方向解像度	1m あたりの画素数
	biYPixPerMeter	4 byte	縦方向解像度	1m あたりの画素数
	biClrUsed	4 byte	格納されているパレット数	使用色数
	biCirImportant	4 byte	重要なパレットのインデックス	
画像データ				

表 3-1

３－３　ビットマップファイルヘッダーの構造

以下に作成したプログラムを用いて実際の BMP データの出力結果を示す。以下は 50×50 ピクセルの 24bitBMP ファイルのヘッダー部分を 8bit 固定長にてダンプした結果である。

bfType	**01000010**	biPlanes	**00000000**
bfType	**01001101**	biBitCount	**00011000**
bfSize	**11101000**	biBitCount	**00000000**
bfSize	**00011101**	biCopmression	**00000000**
bfSize	**00000000**	biCopmression	**00000000**
bfSize	**00000000**	biCopmression	**00000000**
bfReserved1	**00000000**	biCopmression	**00000000**
bfReserved1	**00000000**	biSizeImage	**00000000**
bfReserved2	**00000000**	biSizeImage	**00000000**
bfReserved2	**00000000**	biSizeImage	**00000000**
bfOffBits	**00110110**	biSizeImage	**00000000**
bfOffBits	**00000000**	biXPixPerMeter	**00010000**
bfOffBits	**00000000**	biXPixPerMeter	**00010111**
bfOffBits	**00000000**	biXPixPerMeter	**00000000**
biSize	**00101000**	biXPixPerMeter	**00000000**
biSize	**00000000**	biYPixPerMeter	**00010000**
biSize	**00000000**	biYPixPerMeter	**00010111**
biSize	**00000000**	biYPixPerMeter	**00000000**
biWidth	**00110010**	biYPixPerMeter	**00000000**
biWidth	**00000000**	biClrUsed	**00000000**
biWidth	**00000000**	biClrUsed	**00000000**
biWidth	**00000000**	biClrUsed	**00000000**
biHeight	**00110010**	biClrUsed	**00000000**
biHeight	**00000000**	biCirImportant	**00000000**
biHeight	**00000000**	biCirImportant	**00000000**
biHeight	**00000000**	biCirImportant	**00000000**
biPlanes	**00000001**	biCirImportant	**00000000**

表 3-2

画像の情報はファイルのヘッダーと呼ばれるファイルの先頭数バイトの部分に記録されており、BMP の場合は表 4-１のような構造で、計 54Byte（432 bit）で表される。
たとえば、BfSize の 4byte 部分が画像のデータサイズを記録しており、表 3-2 の場合
始めの 1 バイトに“ **11101000** ”次が“ **00011101** ”以降“ **00000000** ”“ **00000000** ”という情報が記録されている。“ **11101000** ”は 10 進数で“ 232 ” “ **00011101** ”は“ 29 ”である。また“ 00000000 ”は 10 進数においても“ 0 ”であるため、このデータのファイルサイズは次のように求められる。

$$232 + (29 \times 256) = 7659$$

このように、ヘッダー情報のみで画像全体のサイズを取得することが可能である。
また、同様にして biXPixPerMeter，biYPixPerMeter のセッションより、イメージのサイズを求めることも可能である。この情報は画像フィルタなどを設計する上で非常に重要となる。

また、BMP ファイルでは 1 画素あたりのデータサイズの違いに応じで、そのフォーマットが大きく異なってくる。この１画素あたりのデータサイズを定義しているセッションが biBitCount である。表 3-2 の場合では、この数値は“ **00011000** ”であり、1 画素あたりのデータサイズは 24bit であることがわかる。BiBitCount の値により BMP ファイルフォーマットは以下のように分類される。

00000001	1bit	モノクロビットマップ
00000100	4bit	16 色ビットマップ
00001000	8bit	256 色ビットマップ
00011000	24bit	1677 万色ビットマップ

表 3-3

ここで、画素のビット数が２４以下のものは、カラーパレット方式となる。カラーパレットとは、その画像の中で用いられる色をパレットとして定義しておき、画像データはそのパレットに応じで構成さる。例えば、4bit のビットマップではカラーパレットには１６色の色を登録しておき、その色のみが画像の中で使える色ということになる。
一方、24bit のビットマップでは 1 つの画素に対して２４ビットのデータが割り当てられているため、Red,Green,Blue の色について各 8 ビットで表される。よって、

$$256 \times 256 \times 256 = 16777216$$

より、16,777,216 色の表現が可能となっている。

３－４　画像データ部分

画像データは左下から右上に向かって記録されており、24bit 以下のビットマップファイルでは、画像データの前にカラーパレットの情報が入ることになる。

カラーパレット方式の画像では、カラーパレット内で定義されている色の場所をカラーパレット番号とし、画像データではその番号で色が表される。

例えば、カラーパレット番号５番の色が“赤”を表すバイナリ列であったとすれば、画像データ内でのそのパレット番号である５は“赤”であることを意味する。

24bit のフォーマットではカラーパレットは存在せず、RGB[5]各 8bit で表現されたデータとなっている。以下に実際の 24bitBMP データ部のダンプ結果の一部を示す。

Red	Green	Blue	Black	white
00000000	**00000000**	**11111111**	**00000000**	11111111
00000000	**11111111**	**00000000**	**00000000**	11111111
11111111	**00000000**	**00000000**	**00000000**	11111111
00000000	**00000000**	**11111111**	**00000000**	11111111
00000000	**11111111**	**00000000**	**00000000**	11111111
11111111	**00000000**	**00000000**	**00000000**	11111111

表 3-4

表 3-4 は 24bit BMP のヘッダー以降 6 バイトのダンプ結果である。ここで用いた画像は Red,Gerrn,Blue,White,Black の 1 色のみの画像データである。これより、24bit BMP の画像データは表 3-5 のような構造であることがわかる。

青	00000000～11111111
緑	00000000～11111111
赤	00000000～11111111

表 3-5

[5] Red（赤）、Green（緑）、Blue（青）という光の３原色。

4 章　画像の情報量

４－１　画像の持つ情報量について

画像の持つ情報量であるが、3 章で述べた通り、フォーマットの違いにより画像の情報サイズが大きく異なるということが理解できる。実際、24bit の画像と 1bit の画像ではおよそ 24 倍の違いがあるということになる。また、画像サイズが同じである、同じフォーマットの BMP ファイルのサイズは、その画像に視覚的違いがあるものであっても、共に同じファイルサイズとなる。

視覚的に異なる 2 つの BMP ファイルがあったとする。それらは同じフォーマットであり同じファイルサイズである。しかし、画像の本質的な情報量も同じという訳ではない。この本質的な情報量というのが、2 章で述べた情報エントロピーである。

たとえば、圧縮可能なデータを圧縮する場合を考えてみると、そのデータの圧縮可能な部分とはデータの本質的な情報でないと言える。つまり、圧縮可能なデータには本質的情報以外にムダな情報（冗長性）が含まれているということになる。この本質的情報量とは、そのデータを認識するために最小限必要な部分と考えることができる。

実際データを認識できる最小限の情報とはどのようなものなのだろうか。例えば、ある画像データを認識する場合を考えてみる。

認識すべき画像データ	画像データを表す情報	最小限の情報
 24bit 50x50 ピクセル	**01000010** **01001101** **11101000** **00011101**.....	・画像の大きさ ・画像を構成する色

表 4-1

表 4-1 で、この画像は 7,656byte の情報量を持っている。しかし、この情報のほとんどはムダな部分つまり冗長性であるといえる。なぜなら、この画像は非常に単調な画像であるため、データ構造が規則的で、本質的情報量が極めて少ないためである。

この画像を認識するために必要な情報とは、“ 画像の大きさ ” と “ 画像を構成する色 ” であり、この情報があれば、その規則に従い元の画像を構成し認識することが可能となる。この情報こそが、そのデータの本質的情報量である。

しかし、この画像が写真のような画像では本質的情報量は爆発的に膨れ上がる。写真のような不規則なデータ構造をもつ画像は、“画像の大きさ”と“画像を構成する色”のみの情報では構成することは不可能である。情報エントロピーとは、ある情報が伝える情報量の本質量である。このエントロピー量とは一般に予測がしにくい状態（乱雑さが高い）の情報ほどエントロピー量は大きいとされる。

４－２　画像データの圧縮について

同じフォーマットの画像データである場合でも、その画像の持つ本質的情報量に応じて画像データの圧縮率も変わってくる。実際、写真のような不規則なデータ構造を持つ画像データを圧縮するのは極めて難しいといえる。また、いくら優れた圧縮アルゴリズム用いても、それが可逆圧縮である場合はデータのもつエントロピー量を下回る圧縮は不可能であり、エントロピー量に近づくことはできても完全に達成することは不可能である。

データ構造の規則性というものも、1 つの冗長性であり、これを見つけ出すことで、その冗長性を取り除き、データを圧縮できる。たとえば、表 5-1 の場合はある色の画素が縦５０横５０並んでいるという規則が存在する。この規則がわかる情報があれば、この画像を規則に従い、構成できる。しかし、実際一般的な画像のデータ構造には規則性など全くと言ってよいほど存在していない。

では、写真のような非常に不規則なデータを圧縮する場合はどうすればいいのだろうか。その手段の 1 つは非可逆圧縮という方法である。この非可逆圧縮とは 2 章でも述べた通り、損失のある圧縮法である。しかし、この圧縮法では圧縮することによって何らかのデータが間引かれ、そのデータを伸張した際に、本来のデータとは異なるデータになってしまうということである。この非可逆圧縮は画像などのデータに対しては非常に有効な手段の 1 つであると言える。実際テキストデータのようなデータを圧縮する場合は非可逆圧縮を用いることができないが、画像データのような多少変化してもかまわないデータに関しては用いることができる。画像データの非可逆圧縮には離散コサイン変換(DCT[6])が非常によく用いられている。これは直行変換のひとつで、時系列信号を周波数成分に分解する変換を行うことであり、画像などの場合には多くが低域成分となるためその特性を活かし圧縮に利用する。しかし、DCT そのものが圧縮をするわけではなく、その変換特性を利用してエントロピー符号化などにより圧縮を行っている。

---

[6] Discrete Cosine Transform

もう 1 つの手段として、画像を構成する色数を減らすという方法がある。これは一般的に減色と呼ばれる処理である。無論、減色処理により色数が減少するため、画像はそれだけ劣化することになるが、減色することによりデータの構造が簡単になるため、冗長性が増えデータを圧縮しやすくなる。

たとえば、1 画素で 24bit の情報量を持つ画像データでは、画素は 16,777,216 通りの値をとることができる。しかし、これが 8bit の画像データであれば、“00000000”～“11111111”までの 256 通りしかとることができない。よって、減色処理により画素のとりうる値は、この 256 通りの中に絞りこまれることになる。また、1 bit の BMP 画像では色数は 2 色しか扱えないため、各画素の色は 1bit で表される。よって、均一な色調の部分では同じ符号が連続して出現することになる。この減色処理は、減色することによってデータサイズも減少するため、あまり色数を必要としない画像には極めて有効な手段であるといえる。

5 章　画像の減色処理

５－１　カラー画像の減色処理

では、実際にこのような画像データを扱うにはどうすればよいだろうか。まずは、4 章で述べた減色に関するアルゴリズムを考えてみる。例えば、24bit 画像データを 8bit に減色する場合の処理の流れは以下の通りである。

1. 処理する画像の中から最も出現率の高い色を 256 個抽出し、パレットに登録する。
2. 各画素において、パレットの中で最も近い色を探す。

まず、出現率の最も多い色を探す場合、色数に応じた要素数の配列を用意し、その配列の番号を色の番号として、その番号の色が出現した際に色番号の配列要素中の数値をカウントしてやることで、各色の出現回数を求めることができる。この場合、RGB の各色の階調情報は 0～255 までの数値で独立した値をとっているため、そのままでは 1 つの配列要素として格納することができない。よってこれを一元化してやる必要がある。これは、

$$R \times 2^{16} + G \times 2^{8} + B$$

とすることで求めることができる。例えば R=255,G=255,B=255 であった場合の値は 16,777,215 と一元化できる。これは 2 進数では 111111111111111111111111 となる。ここで各値を 2 進数で表すと以下のようになり

$$(255)_{10} = (11111111)_{2}$$
$$(255 \times 2^{8})_{10} = (1111111100000000)_{2}$$
$$(255 \times 2^{16})_{10} = (111111110000000000000000)_{2}$$

これを元の RGB の独立した値に分離する際にはそれぞれ 8bit を取り出し基数を変換すればよいことが分かる。

最も出現率の高い色 256 色をピックアップするには、この配列要素の数値を大きい順にソートしてやり、その上位 256 色を選びパレットに登録してやればよい。

次に各画素においてパレットの中で最も近い色を探す方法であるが、これは RGB 各パラメータを３次元座標の点と考え、これをパレットに登録されている色と、原画像の２点の距離として考える。

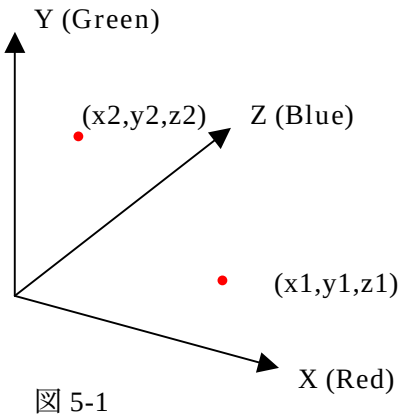


図 5-1

この場合、点(x1,y1,z1)(x2,y2,z2)の座標の距離は以下の式で求められる。

$$\sqrt{(x2-x1)^2+(y2-y1)^2+(z2-z1)^2}$$

このようにしてパレットに登録されている色と比較し、距離が最小となったパレットの色と置き換えることで画像を減色することができる。

５－２　カラー画像の２値化

次にカラー画像を２値化する場合について考えてみる。

画像を 2 値化する場合の処理の流れを以下に示す。

1. RGB の荷重平均[7]より輝度を算出する
2. 輝度を閾値に応じて処理し、画素を決定する

まず、減色する元となる画像の画素ごとの輝度を求める必要がある。この輝度の求め方であるが、これは RGB の荷重平均から求められる。ただ、輝度は視覚的特性を考慮し、RGB の成分ごとに異なった数値の係数が掛けられる。輝度は一般に以下の式で求められる。

$$0.298912\times R+0.586611\times G+0.114478\times B$$

５－２－1 固定閾値による画像の 2 値化

では、この輝度に基づき画像を閾値に従って処理する。例えば、閾値が 128 と与えられていた場合、輝度 128 以下は画素を１とし、128 より大きい値であればその画素は 0 とする。以下に実際の画像を処理した結果を示す。

[7] 集合を成すデータのそれぞれがもつ重みを考慮して算出する平均値

原画像

画像 5-1

閾値 50 で減色後の画像

画像 5-2

閾値 128 で減色後の画像

画像 5-3

閾値 200 で減色後の画像

画像 5-4

５－２－2 考察

　このように、固定閾値での 2 値化では、画像の階調情報が失われてしまう。よって、写真のような画像データの 2 値化には不向きであるといえる。また、この結果から各色に掛けられた係数の影響により、RGB ごとに輝度の基準が違うということが分かる。実際、人間の目には波長の短い光である紫や青色は赤や緑より暗く見える。この結果より、最も輝度の低い色が青で赤→緑の順で輝度が高くなることがわかる。

５－３　2 値画像における階調の再現

2 値画像では画像の階調を表現することが不可能である。よって 2 値画像で階調を擬似的に再現する方法としてディザ法が用いられる。ディザ法とは画像情報にある特定の割合でディザというノイズを加える処理のことでる。実際、ディザ法は 1 色しか使うことのできないプリンターや新聞の印刷などで、階調を再現するときなどに用いられている。

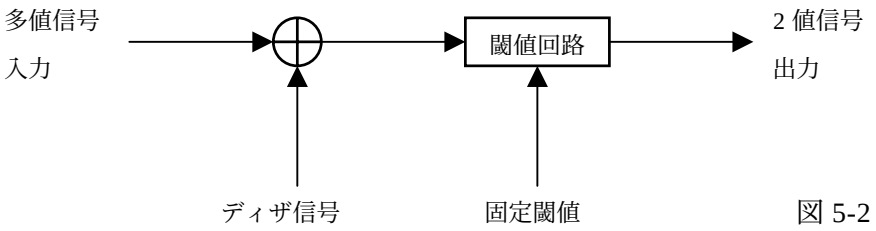


図 5-2

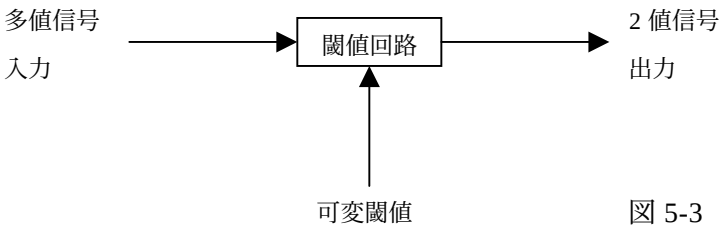


図 5-3

ディザ処理には図 5-2 のようにあらかじめ画像データにディザ信号を加えた後に閾値回路にて処理するものと、図 5-3 のように閾値回路の閾値を可変させるものがあり、これらは共に等価である。

ディザ処理には幾つかの方法が考案されており、代表的なものとしては、ランダムディザ法や組織的ディザ法がある。このランダムディザ法とは、文字通りランダムにディザ信号を画像に加える方法である。具体的な処理法としては、まず減色する元の画像の輝度情報が 0～255 で与えられている場合、同様に 0～255 までの乱数を発生させる。そしてその乱数の値が画像における輝度の範囲内の値であればその画素を白とし、範囲外の場合は黒とする。

次に、組織的ディザ法であるが、これはディザ行列というマトリクスを用意し、その要素に応じで画像を処理する方法である。4×4 のマトリクスを用いた場合の処理の流は以下の通りである。

1. 5－２で述べた手順に従い、色情報を輝度情報に置き換える。
2. 処理する画像を 4×4 画素のメッシュ状に分割する。
3. 分割されたブロックごとにディザ行列の閾値に応じて 2 値化する。

まず、RGB 値より輝度を算出する方法は 5－２で述べた通りである。

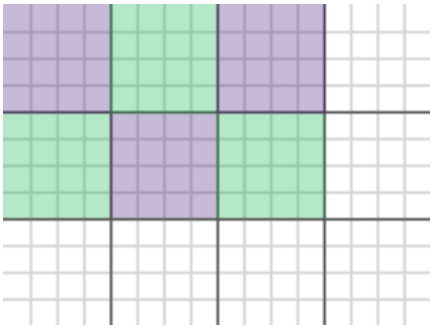


図 5-4

ここで求められた輝度を画素の情報として画像全体を図 5-4 のように 4×4 画素のメッシュ状に分割する。

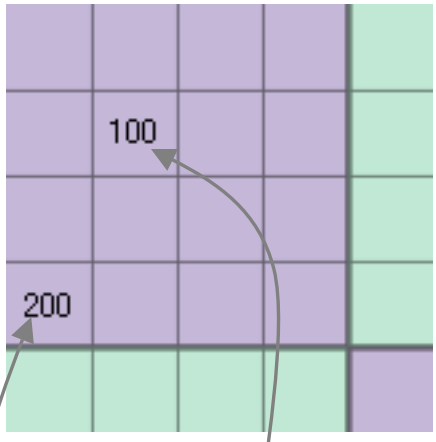


図 5-5

この画像の輝度情報の一部が図 5-5 のような状態であった場合、この輝度情報を図 5-6 のようなディザ行列の閾値と比較することで、図 5-7 のように画素を決定する

0	128	32	160
192	64	224	96
48	176	16	144
240	112	208	80

図 5-6

	255		
0			

図 5-7

５－３－１　ディザ行列の評価

このディザ行列には Bayer 形、ハーフトーン型、Screw 型、中間調強調型、DotConcentrate 型等が存在する。これらのディザ行列は人間の視覚的特性を考慮し定められている。

ここで、実際にディザ処理を行うプログラムを製作し、これら行列ごとの視覚的特性を実際に検証してみることにする。

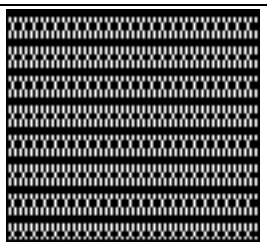
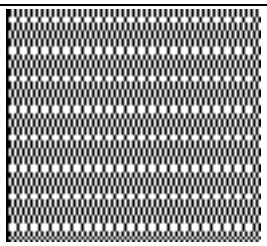
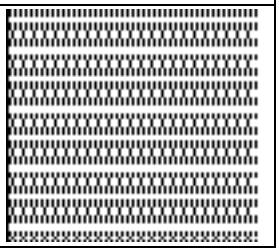
Bayer 型				輝度:64	輝度:128	輝度:160
0	8	2	10			
12	4	14	6			
3	11	1	9			
15	7	13	5			
処理後の画像						

表 5-1

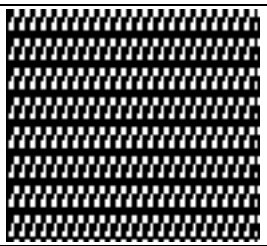
ハーフトーン型				輝度:64	輝度:128	輝度:160
10	4	6	8			
12	0	2	14			
7	9	11	5			
3	15	13	1			
処理後の画像						

表 5-2

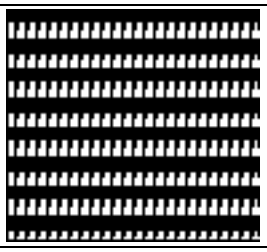
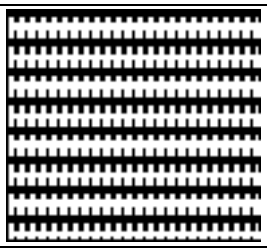
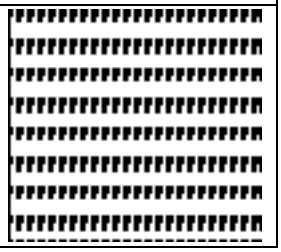
Screw 型(1)				輝度:64	輝度:128	輝度:160
13	7	6	12			
8	1	0	5			
9	2	3	4			
14	10	11	5			
処理後の画像						

表 5-3

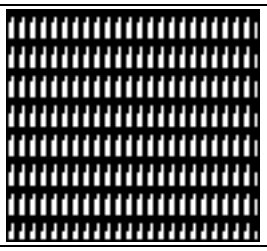
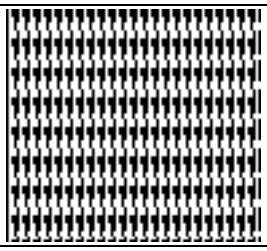
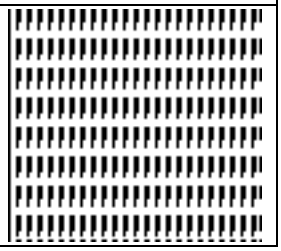
Screw 型(2)				輝度:64	輝度:128	輝度:160
15	4	8	12			
11	0	1	5			
7	3	2	9			
14	10	6	13			
処理後の画像						

表 5-4

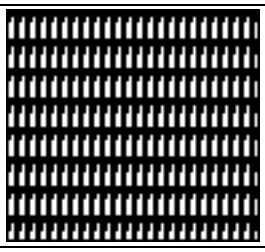
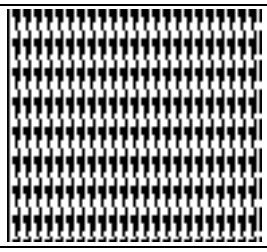
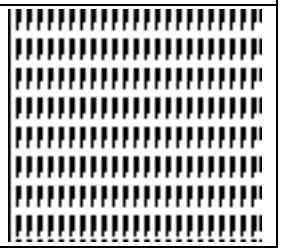
中間調強調型				輝度:64	輝度:128	輝度:160
12	4	8	14			
11	0	2	6			
7	3	1	10			
15	9	5	13			
処理後の画像						

表 5-5

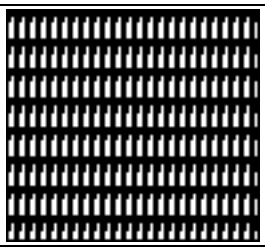
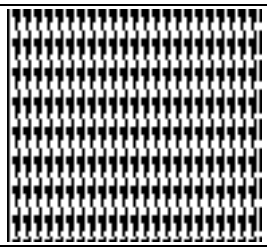
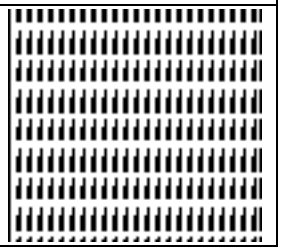
DotConcentrate 型				輝度:64	輝度:128	輝度:160
13	4	8	14			
10	0	1	7			
6	3	2	11			
15	9	5	13			
処理後の画像						

表 5-6

５－３－２　考察

以上の結果より、Bayer 型、ハーフトーン型、Screw 型(1)、Screw 型(2)で非常に特徴的な結果をえることができた。Bayer 型の模様は中でも最も緻密なものであることが分かった。また、Screw 型(2)および中間調強調型ではこの 3 段階の輝度では同じ結果を示している。中間調強調型および DotConcentrate 型では、輝度 64 と輝度 128 では同じ結果が得られているが、輝度 160 では微妙に異なる結果を得ることができた。この実験の結果では、Bayer 型行列およびハーフトーン以外のディザ行列は非常に模様が大きく、これらの行列を用いた場合、細かい写真のような画像の再現は難しいと考えられる。

５－３－３　　各ディザ行列における擬似階調の評価

次に、階調情報をもった画像に適応するプログラムを作成する。この場合、画像の画素ごとに輝度が異なるため、ディザ行列と比較する輝度も画素ごとに異なってくる。

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

図 5-8

図 5-8 をディザ行列とすると、この 1～16 までの要素には異なる閾値が設定されていることになる。

1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12
13	14	15	16	13	14	15	16
1	2	3	4				

図 5-9

図 5-9 を実際の画像とした場合、この升目が画像の画素と考える。この画像をディザ法で処理する場合、図 5-9 番号と、図 5-8 の番号が対応するように輝度を比較する必要がある。

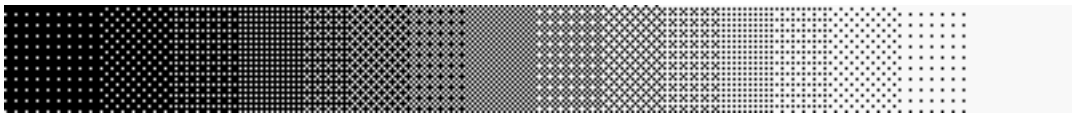
画像 5-5

例えば、画像 5-5 は階調を持った 60×20 ピクセルの画像である。以下はこの画像の輝度の配列一部である。この画像をディザ処理する場合、この配列とディザ行列を比較し、ドットを決定する。

0 ,2, 5, 8, 11, 14 ,17, 21, 25, 29, 33, 37, 42, 46, 51,56,61,65,,,,253
0 ,2, 5, 8, 11, 14 ,17, 21, 25, 29, 33, 37, 42, 46, 51,56,61,65,,,,253
0 ,2, 5, 8, 11, 14 ,17, 21, 25, 29, 33, 37, 42, 46, 51,56,61,65,,,,253
0 ,2, 5, 8, 11, 14 ,17, 21, 25, 29, 33, 37, 42, 46, 51,56,61,65,,,,253
,,,,,,,,,,,,,,,,

以下に実際に作成したプログラムを用いで実際の画像データを処理した結果を示す。

原画像　256 階調

Bayer 型

ハーフトーン型

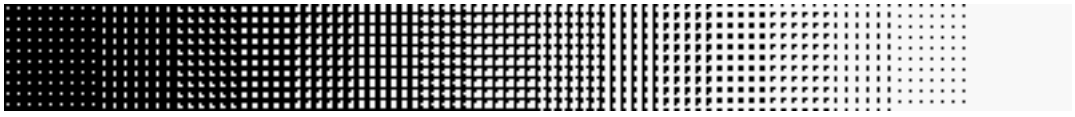
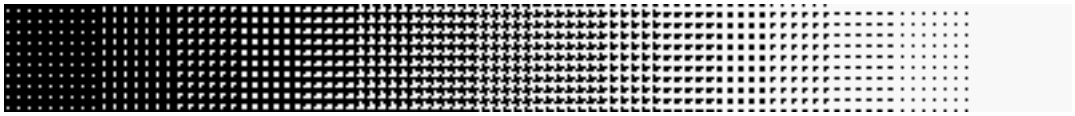
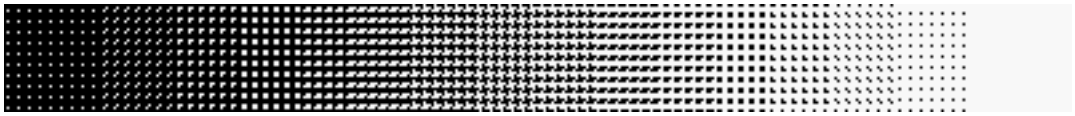
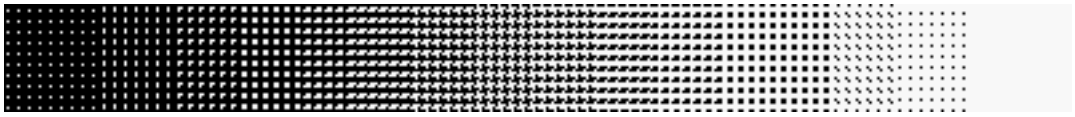
Screw 型(1)

Screw 型(2)

中間調強調型

DotConcentrate 型


表 5-7

原画像

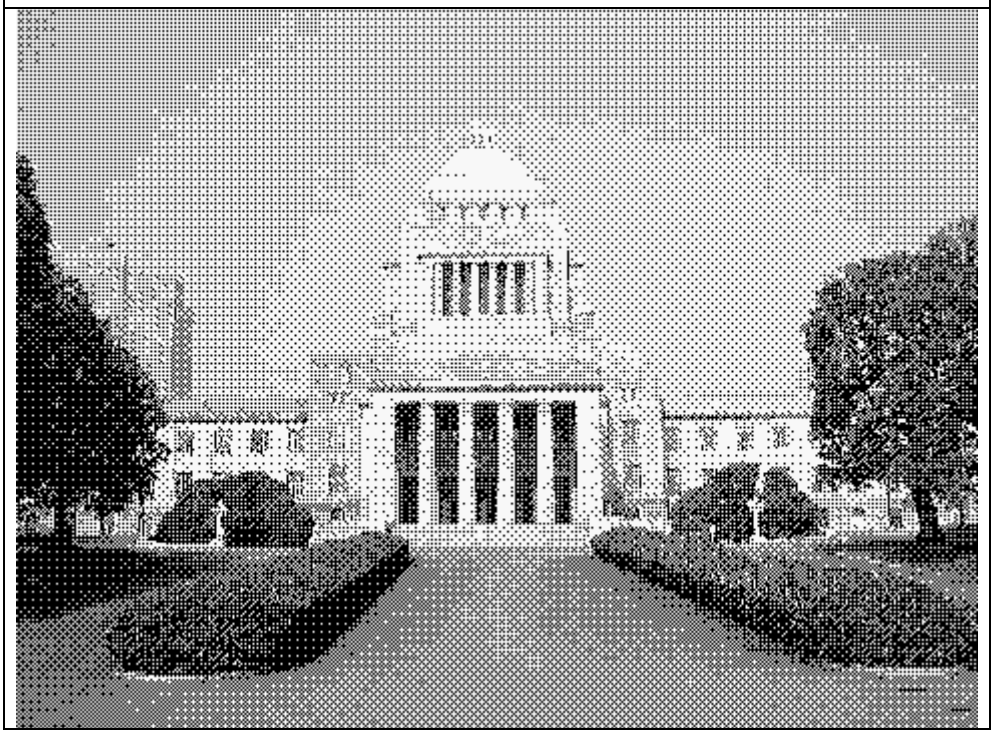
Bayer 型


表 5-8

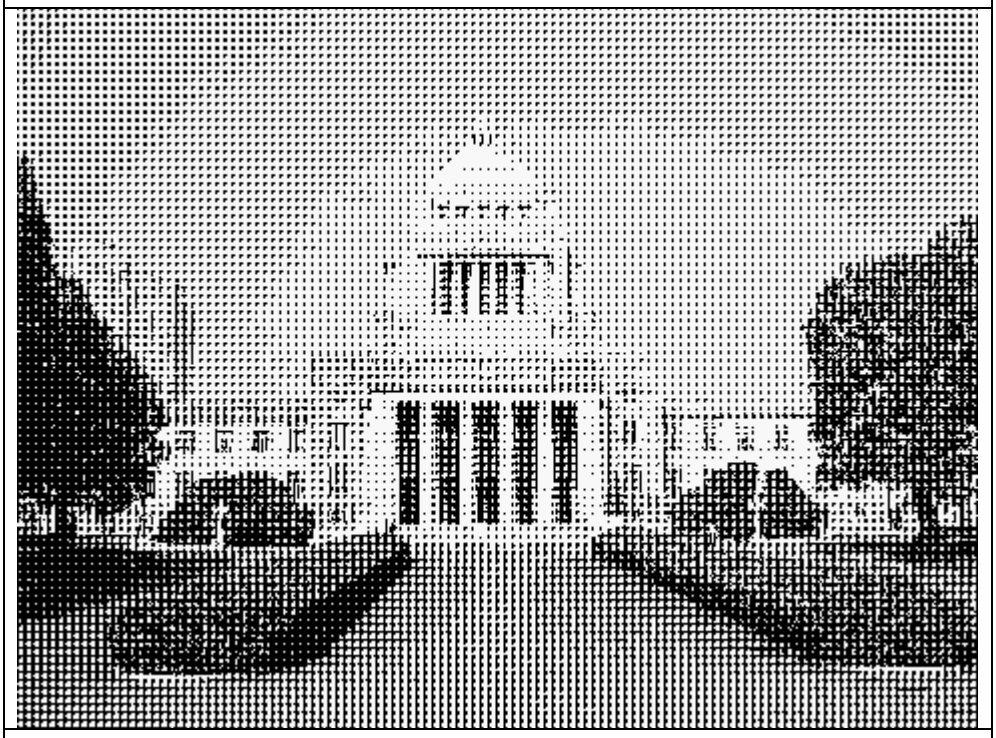
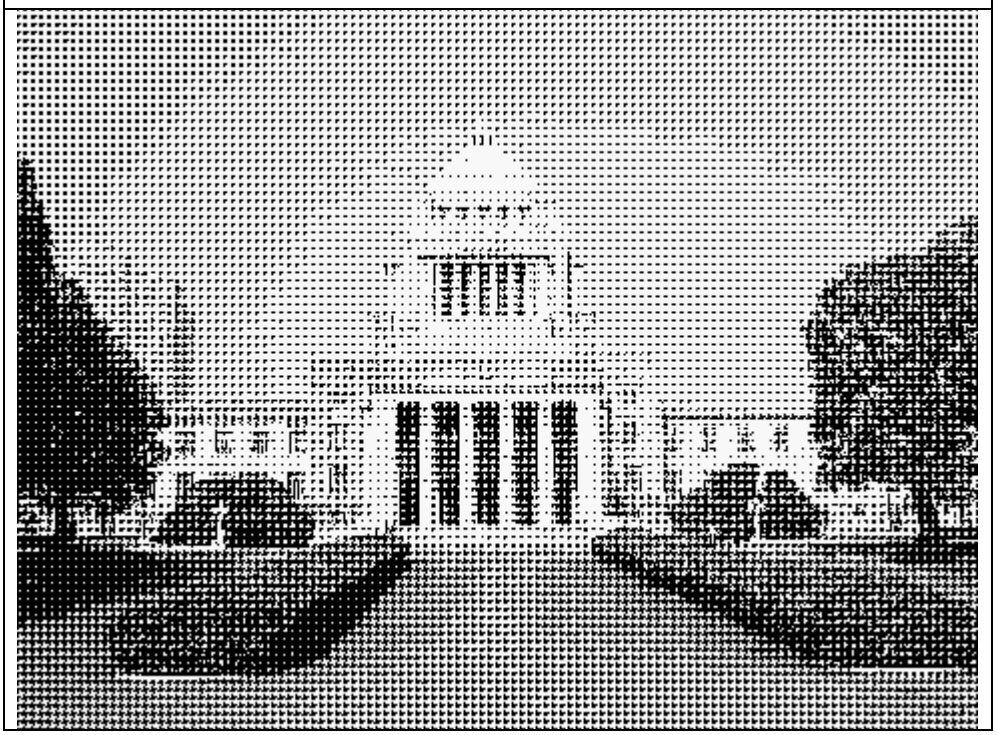
ハーフトーン型

Screw 型(1)


表 5-9

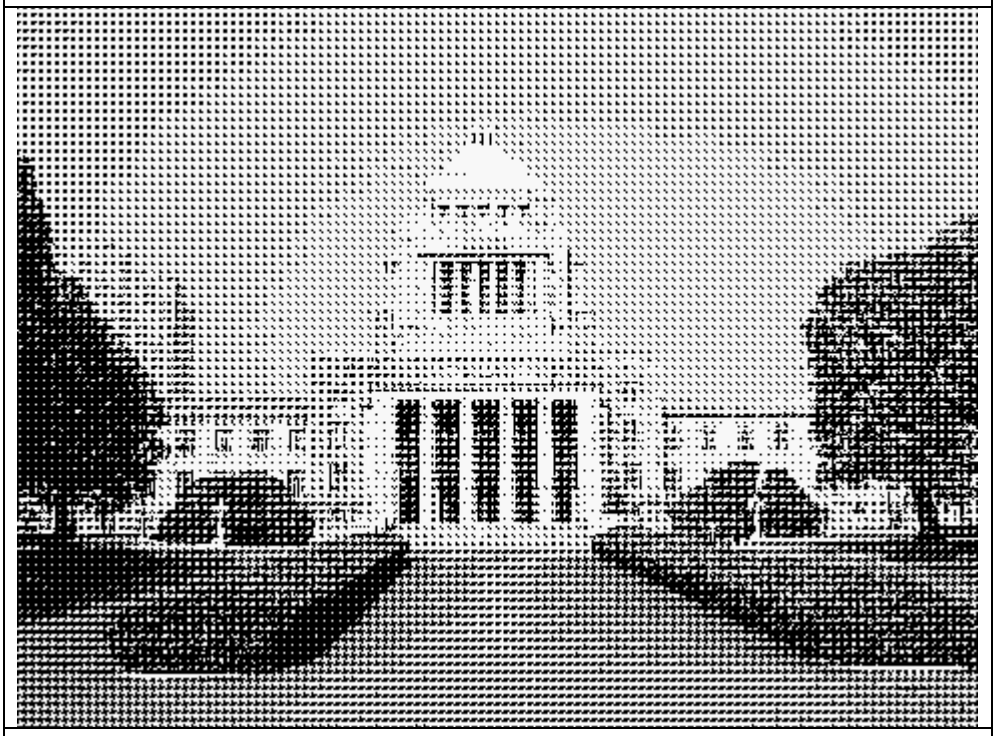
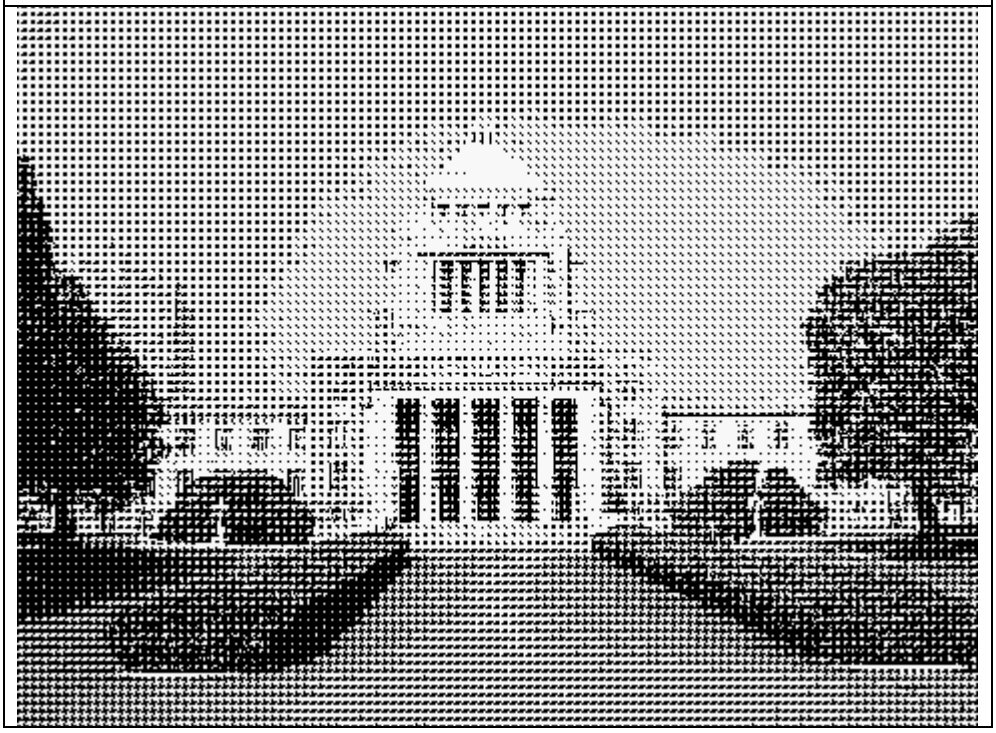
Screw 型(2)

中間調強調型


表 5-10

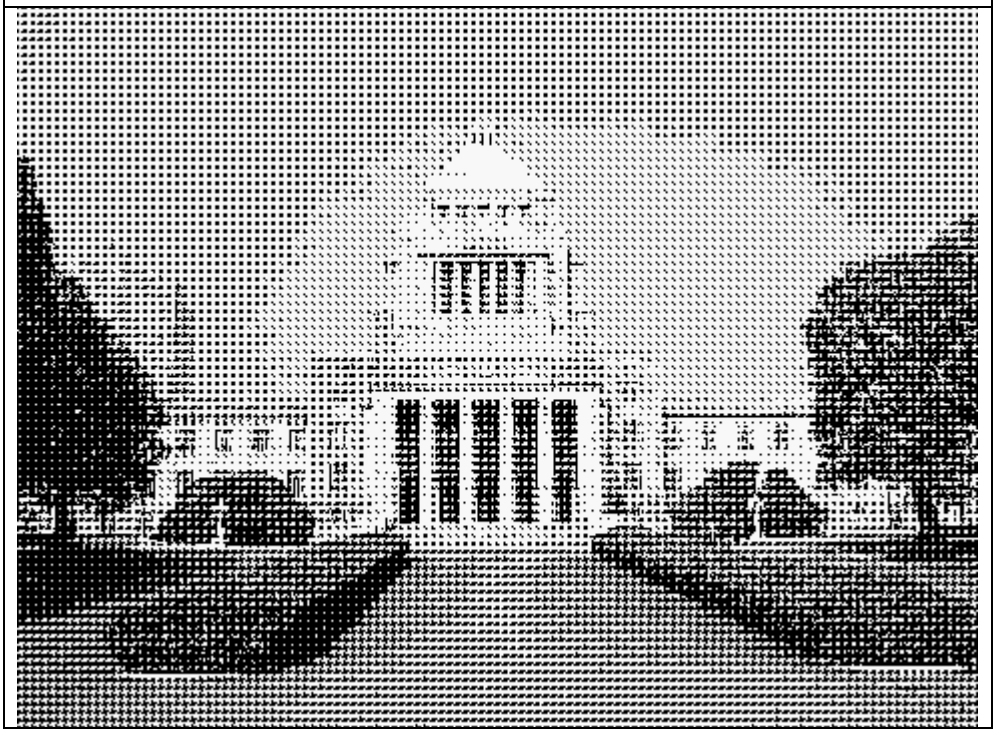
DotConcentrate 型


表 5-11

５－３－４　考察

これらの結果より、行列の要素数に応じた階調が再現されているのが見て取れる。表 5-7 から Bayer 型行列を用いて処理した画像が最も緻密な模様であることが分かった。また、Bayer 型は各階調ごとの模様の変化が他の行列に比べ非常に特徴てきである。一方、DotConcentrate 型では、中間諧調の差が Bayer 型に比べ非常に分かりにくいものとなっている。階調が最も緩やかに再現されているのも Bayer 型であり、Screw 型は階調の再現が急であり模様も大きい。このことより、緻密な模様をなすディザ行列ほど、階調をより緩やかに再現できるといえるだろう。

Screw 型(1) Screw 型(2)中間調強調型 DotConcentrate 型は非常に結果が似ており、こらの行列と Bayer 型の行列は全く異なる結果を示しているといえる。ハーフトーン型は模様も比較的細かく、これらの中間と位置付けることができるだろう。

また、表 5-8～表 5-11 の結果より、Bayer 型が最も中間の階調が再現されており、写真向けといえる。逆に、Screw 型等は、階調が非常に明確に再現されているため、グラフィックやグラフなどの画像データに適しているといえる。求める画像のコントラストに応じてこれらの行列を使い分けるとよいだろう。

５－３－５ハードウェアでの設計

では、このフィルタをハードウェアにて設計する。VHDL での記述においては、ハードウェアの規模の制限などを考慮し、入力する画像の解像度や行列型の種類を制限することとした。今回はディザ行列に Bayer 型を採用し、入力するソースの解像度は 320×240 画素の画像を用いることを考慮して設計を行った。設計するフィルターは 24bit の信号を入力し、１bit のモノクロ画像を出力する回路である。以下に信号の流れを示す。

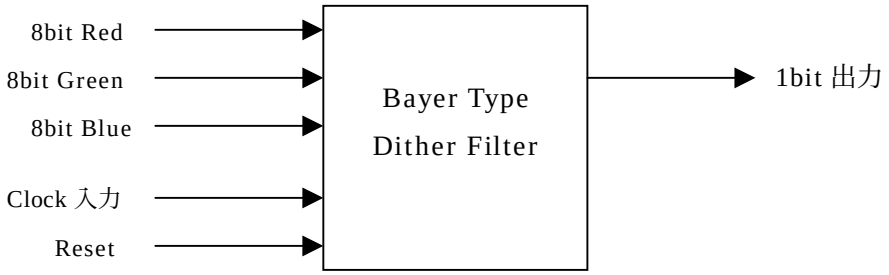


図 5-10

VHDL は一般的な言語と異なりハードウェアの設計を前提としているため、ファイル操作に関する処理が存在していない。よって、ここで記述したプログラムに信号を入力する際には、テストベンチと呼ばれる信号を作るための記述が必要となってくる。図 5-10 のような回路であれば、Clock,Reset,RGB の各データの信号を生成できるテストベンチ記述が必要である。また、出力結果はシミュレータによりタイムチャートとして出力される。

VHDL での記述ではディザ行列の値を格納する配列に 2 次元配列を用いることができない。実際、多次元配列を VHDL にて記述することは可能であるが、これをロジック回路として生成することは不可能である。よって VHDL における多次元配列はシミュレーションなどのみで用いることに限定される。本回路ではディザ行列を 1 次元の配列を用いることで実現した。また、算術演算子などを多様すると回路規模が莫大なものとなる可能性があるので、不要な演算子を省くこととした。よって今回の設計では、輝度の計算に関する係数を 10 倍し、小数点以下の値は切り捨てることとした。

以下に回路のシミュレーション結果を示す。

図 5-11 は合成された回路のシミュレーション結果である。ここで、rgb_red,rgb_green,rgb_blue が入力されるデータで、テストベンチではこれらのポートに対して”0”と”255”の信号が特定の周期で入力される記述となっている。

また、今回輝度の計算を整数型で行うため、各係数はそれぞれ 10 倍し、小数点以下を切り捨てた形で処理するようになっている。よってここでの輝度を求める式は

$$R \times 3 + G \times 6 + B$$

である。

ここで、test_out という出力があるが、これはシミュレーションに際して輝度の計算が確実に行われているかを検証する目的で作成したポートであり実際の回路では必要としない。このポートは 12bit のバスで図 5-11 ではこの値を 10 進数で表示している。例えば、R=255,G=255,B=255 の場合、求める輝度は 2550 でありこの回路で輝度の計算が正常に動作していることが分かる。

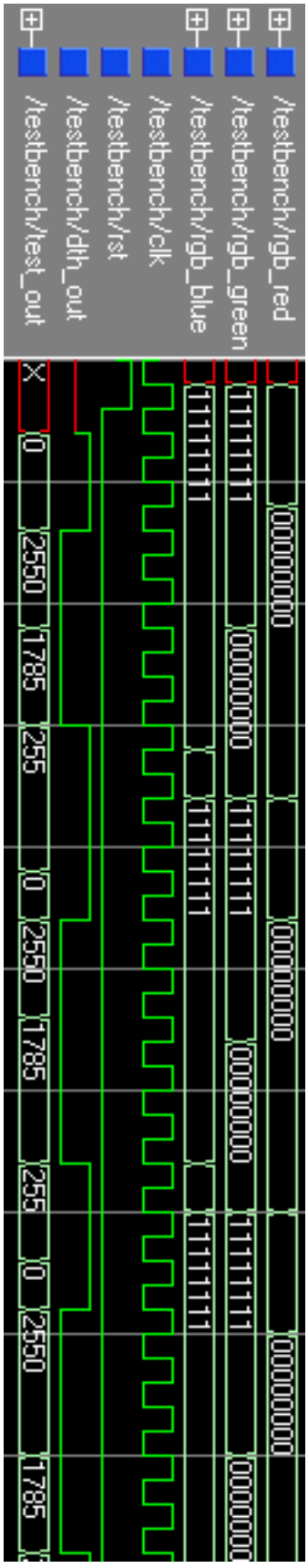


図 5-11

５－4　カラー画像におけるディザの応用

今まではディザ法は画像の 2 値化において階調を擬似的に再現する方法として用いてきたが、ディザ法はカラー画像においても暫し用いられることがある。その代表的な例がインクジェットプリンタで、現在主流のインクジェットプリンタは 3 色から５色程度のインクを用いて、中間色や階調を再現している。

また、フルカラー画像を 256 色や、16 色などに減色した際に生じるマッハバンドを除去するために用いられる。マッハバンドとは、減色された画像においてパレットとパレットの色の差によって画像に生じる色の境界線のことで、この境界線の部分にディザ信号を加えてやることで境界線を目立たなくすることができる。

6 章　画像情報の圧縮

６－１Run Length による圧縮

次に画像情報の圧縮について考える。ここで設計するエンコーダおよびデコーダは 2 章で述べた Run Length のアルゴリズムを用いたものである。Run Length のアルゴリズムには以下のような特徴がある。

- アルゴリズムが容易で設計しやすい
- 2 パス処理を必要としない
- 2 値画像に対して効果的である

Run Length はコードが連続するものに対してのみ高い効果を示す。また、Run Length はデータをコードとその連続数に置き換えることで圧縮するため、コードに連続性がないデータを処理した場合データが逆に大きくなる可能性がある。

このエンコーダをプログラムで作成する場合の処理の流れについては以下の通りである。

1. 入力としてデータを１byte づつのコードにして入力する。

次に入力されたコードが同じコードであれば、カウンタを加算する。違うコードが入力されれば、コードとカウント値を出力する。

今回はこれをハードウェアで設計し、また動作を検証するため同様の処理を行うソフトウェアを作成、設計したエンコーダの動作を検証することにした。

６－２ハードウェアでの設計

では、このエンコーダをハードウェアにて設計する。プログラムの構造は基本的には同じであるが、ハードウェアの特性上注意しなくてはならない部分が幾つか存在している。その 1 つがコード長の制限である。これは設計するハードウェアの仕様にもよるが、今回設計するエンコーダでは入力コード長は 8bit で出力コード長および、連続数の出力も各 8bit とした。この場合、カウントするコードの連続数は最大 255 までしか行うことができず、255 以上コードが連続するデータを入力することができない。よって、ハードウェアで設計する場合はこの問題を考慮した設計でなければならない。具体的には、カウント値が 255 に達した場合カウント値とコードを出力し、カウンタをリセットするという処理を追加しなければならない。設計するエンコーダの各信号の流れは図 6-1 の通りである。

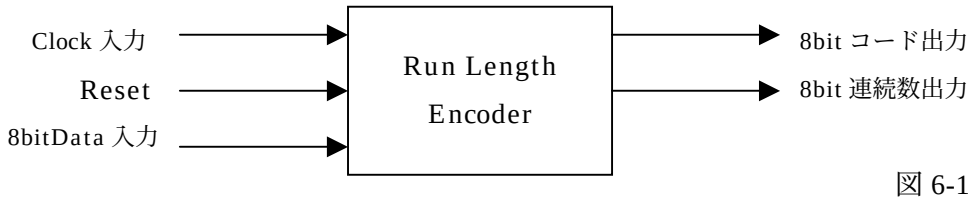


図 6-1

具体的な回路の構成であるが、まず内部にメモリを 2 つ用意する。これは、8bit のコードを格納できるもので合計 2 byte 必要となる。これらを仮に、BUF_A ,BUF_B とすると、BUF_A は入力されたデータを格納するメモリで、BUF_B は過去に入力されたデータを保持しておくためのメモリである。次に、コードの連続数をカウントするカウンタが必要となる。これを CNT_A とする。これは、8bit で最大 255 までカウントできるカウンタである。

まず、入力されたコードは BUF_A に格納される。その後、BUF_A の内容は過去に入力されたコードである BUF_B と比較される。ここで、BUF_A と BUF_B の内容が一致すれば、入力されたコードは過去に 1 回出現したことになり、カウント値を 1 加算する。そして、BUF_A と BUF_B の内容が異なるものであれば、BUF_B のコード並びに、カウント値を出力し、BUF_B のコードを現在の入力である BUF_A のコードに更新し、次にカウント値をリセットする。

ここで、問題として考えられるのがカウンタの最大カウント値である。ここで用いるカウンタは 8bit であるため最大カウント数は 255 に制限される。よって、入力されるコードが 255 個以上連続するようなデータの場合はここで不具合が生じることになる。この問題を解決するために、CNT_A の値が 255 に達したときにコードとカウント値 255 を出力し、カウンタをリセットする処理を追加する。以下に実際のシミュレーション結果の一部を示す。

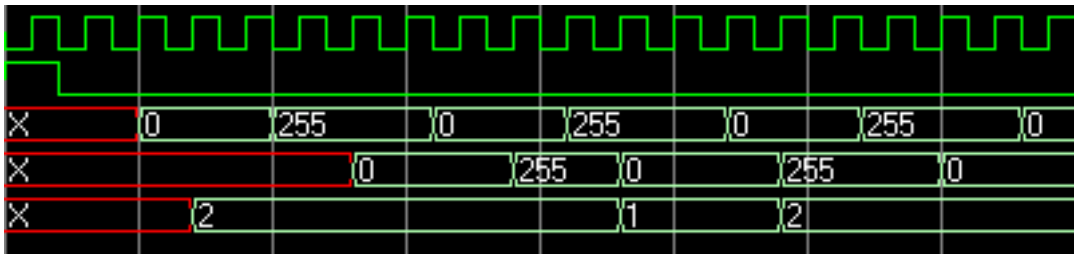


図 6-2

図 6-2 のタイムチャートでは、信号は上の 2 本がクロック、リセット信号で以降 3 本のバスが、データ入力、コード出力、カウント出力である。これらのバスは各 8bit であり、タイムチャートに表示されている数字は 10 進数での値である。ハードウェアでの設計では、このリセット信号が必ず必要となってくる。ハードウェアで記憶素子を構成する際に必要となるのがフリップフロップと呼ばれる順序回路で、これは 1 ビットの情報を記憶できる順序回路である。この回路の特徴として、電源を入れた直後の出力の状態が不定であるという特徴を持つ。よってフリップフロップなどを用いたカウンタなどでは、初期値が 0 に揃わないため、これをリセット信号を用いて初期化してやる必要がある。

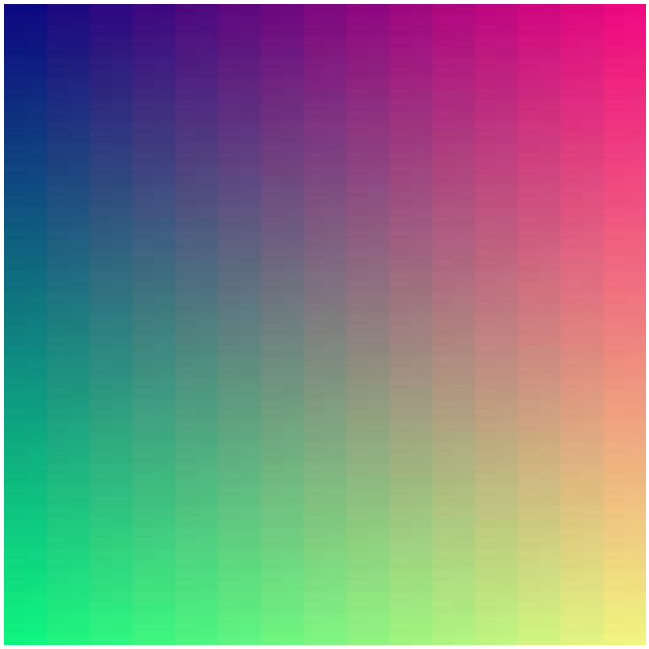
この回路は同期回路で、同期回路とは内部の順序回路が入力されたクロックパルスに同期して動作する回路である。この回路の場合は 1 クロックで 1byte のデータが入力される。例えば、図 6-2 の場合は初めに“ 0 ”という信号が 3 クロックで 3byte 回路に入力されている。次の 3 クロックでは“ 255 ”という信号が 3byte 入力されていることになる。この入力された信号を RLE[8]に従い処理した出力が図 6-2 における下 2 本のバスの出力である。この場合

“ 0 ”が 3byte 入力され、“ 255 ”が 3byte 入力されているため出力は 0×3、255×3、ということになる。ここで、カウント値の出力が 2 となっているのはカウンタが 0 から開始されているためであり、0,1,2,とカウントされることより出力が 2 となっている。

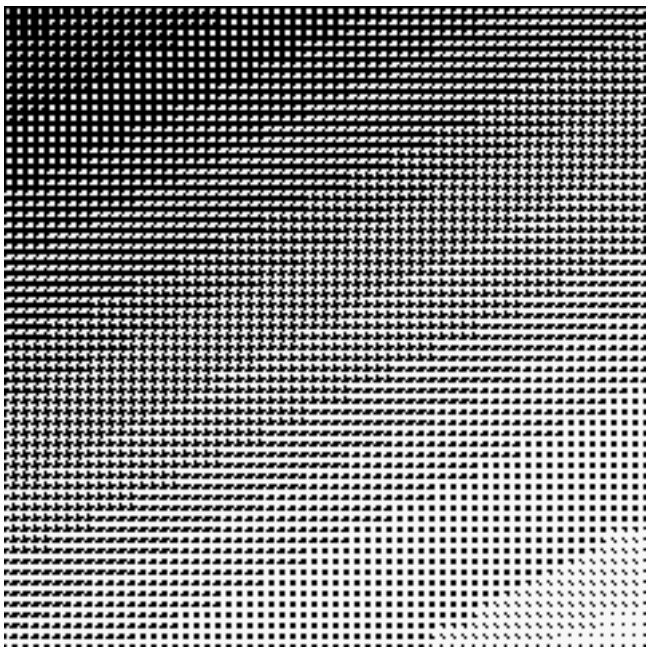
[8] Run Length Encoding

６－３　性能評価

次にエンコーダの性能評価を行う。この際、VHDL によるシミュレーションでは実際の画像データの圧縮効率を評価することが困難であるため、設計した回路と同様の動作を行うソフトウェアを作成し、性能評価することとした。



画像 6-1



画像 6-2

評価の対象として用いた画像 6-1 は 240×240 ピクセルの BMP フォーマットの画像である。また、画像 6-2 は画像 6-1 を DotConcentrate 型の行列を用いて、ディザ処理したものである。これらの画像はどちらも 24bit で出力したもので、バイナリ状態では 172,845byte のファイルサイズである。

今回作成したプログラムは画像データを 8bit 固定長でダンプした ASCII データとして入力し、RLE のアルゴリズムに従い処理したものを 8bit のコードおよび 8bit の連続数で出力するものである。

	バイナリサイズ	ダンプファイルサイズ	圧縮ファイルサイズ	圧縮率
画像 6-1	172,845 byte	1,728,450 byte	3,210,715 byte	185.8%
画像 6-2	172,845 byte	1,728,450 byte	336,490 byte	80.53%

表 6-1

このように、画像 6-1 のようなコードに連続性のない画像データを RLE で処理した際は元のファイルサイズよりも大きくなってしまう。これは、コードに対して連続数が付加されるためでる。一方ディザ処理により 2 値化された画像では圧縮率は 80.5%と非常に効果があることが分かる。

この結果より、画像の 2 値化が RLE に対して非常に有効であるということが証明できたといえる。

まとめ

本研究を終えて、実際の画像情報がどのような構造をもって記録され、その情報をどのようにハードウェアまたはソフトウェアで扱うかなど、画像処理に関する理解を深める上で非常に有意義であった。また、これらを検証するにあたり、VisualBasic と VHDL の 2 つの言語を使うことで、これらの各言語の特徴や相違点、プログラミングにおける基本的な条件処理、アルゴリズムなどを学ぶことができた。

参考文献

[1]並木　秀明・永井　垣道　[著]　“ VHDL によるデジタル回路入門 ”　技術評論社
[2]長谷川　裕恭　[著]　“ VHDL によるハードウェア設計入門 ”　三晃印刷㈱
[3]中村　次男　[著]　“ デジタル回路設計法 ”　日本理工出版会
[4]安居院　猛・中嶋　正之　[著]　“ 画像情報処理 ”　森北出版株式会社
[5]原島　博　[監修]　“ 画像情報圧縮 ”　オーム社
[6]堀内　司郎・有村　一郎　[監修]　“ 画像圧縮技術のはなし ”　工業調査会
[7]松本　紳・小高　和巳　[著]　“ マルチメディアビギナーズテキスト ”　東京電機大学出版局
[8]Jon Clark Craig [著]　“ Visual Basic4.0 テクニックライブラリ ”　アスキー出版局
[9]河西　朝雄　[著]　“ Visual Basic 初級プログラミング入門 ”　技術評論社
[10]川口　輝久・河野　勉　[共著]“ かんたんプログラミング Visual Basic 基礎編 ”技術評論社

謝辞

本研究を行うに際し、終始ご指導、御鞭撻を賜った高知工科大学　電子・光システム工学科の橘　昌良　助教授ならびに矢野　政顕　教授、原　央　教授に心より感謝申し上げます。

付録

ここでは本研究で各検証を行うために作成したプログラムの主要なソースコード並びに GUI の外観、またその解説を収録する。尚、今回プログラムの作成に用いた言語は VisualBasic5.0 並びにハードウェア記述言語である VHDL である。

付録-1 バイナリ→ASCII 変換


```
Private Sub Dir1_Change()
        File1.Path = Dir1.Path
End Sub
```

```
Private Sub Drive1_Change()
        Dir1.Path = Drive1.Drive
End Sub
```

```
Private Sub File1_Click()
    If Right(Dir1.Path, 1) = "¥" Then
        Fname = Dir1.Path & File1.filename
    Else
        Fname = Dir1.Path + "¥" + File1.filename
    End If
End Sub
```

```
Command1
```

General

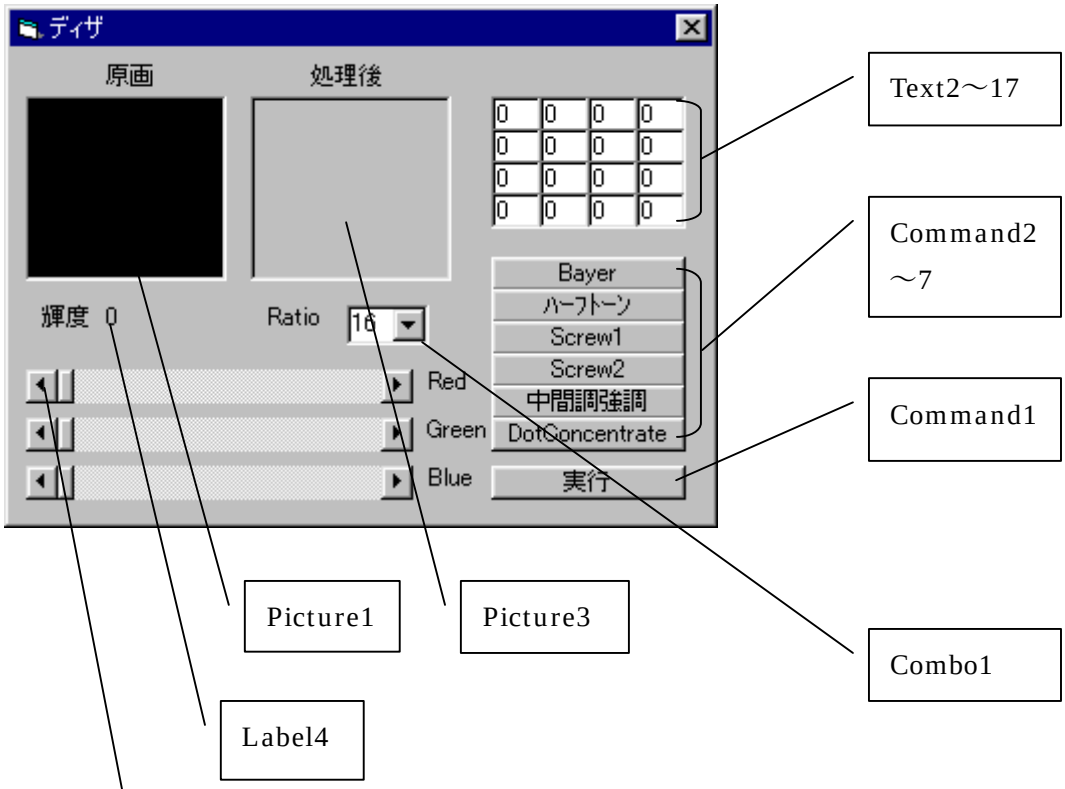
```
Dim LD As Byte
Dim a As Byte
Dim c As Long
Dim i As Long
Dim ansA As Currency
Dim ansB As Currency
Dim ansC As Currency
Dim ansD As Currency
Dim ansE As Currency
Dim ansF As Currency
Dim ansG As Currency
Dim ansH As Currency
Dim ansI As Currency
Dim ansJ As Currency

Dim Fname As String

Dim subA As Byte
Dim subB As Byte
Dim subC As Byte
Dim subD As Byte
Dim subE As Byte
Dim subF As Byte
Dim subG As Byte
Dim subH As Byte
Dim subI As Byte
```

Command1

```
Private Sub Command1_Click()
    Command1.Caption = "処理中"
    If Fname = "" Then
        ret = MsgBox("ファイルを選択してください", 48, "エラー")
    Else
    Open Fname For Binary As #1
        Do Until EOF(1)
            Get #1, , a
            i = i + 1
        Loop
    Close #1
    Open Fname For Binary As #1
    Open Fname & ".Bit" For Append As #2
        For c = 1 To i - 1 Step 1
            Get #1, , LD
            subA = LD Mod 2      ‘基数を変換
            If subA = 1 Then
                ansA = LD / 2 - 0.5
            Else
                ansA = LD / 2
            End If
            subB = ansA Mod 2
            If subB = 1 Then
                ansB = ansA / 2 - 0.5
            Else
                ansB = ansA / 2
            End If
            subC = ansB Mod 2
            If subC = 1 Then
                ansC = ansB / 2 - 0.5
            Else
                ansC = ansB / 2
            End If
           subD = ansC Mod 2
```

```
            If subD = 1 Then
                ansD = ansC / 2 - 0.5
            Else
                ansD = ansC / 2
            End If
            subE = ansD Mod 2
            If subE = 1 Then
                ansE = ansD / 2 - 0.5
            Else
                ansE = ansD / 2
            End If
            subF = ansE Mod 2
            If subF = 1 Then
                ansF = ansE / 2 - 0.5
            Else
                ansF = ansE / 2
            End If
            subG = ansF Mod 2
            If subG = 1 Then
                ansG = ansF / 2 - 0.5
            Else
                ansG = ansF / 2
            End If
            subH = ansG Mod 2
            If subH = 1 Then
                ansH = ansG / 2 - 0.5
            Else
                ansH = ansG / 2
            End If
            Print #2, subH & subG & subF & subE & subD & subC & subB & subA
        Next c
    Close #1, #2
    End If
    Command1.Caption = "終了"
End Sub
```

付録-2 ディザ行列評価プログラム




```
Private Sub HScroll1_Change()
    Cred = HScroll1.Value
    Cgre = HScroll2.Value
    Cblu = HScroll3.Value
    lum = (0.298912 * Cred + 0.586611 * Cgre + 0.114478 * Cblu)
    Picture1.BackColor = RGB(Cred, Cgre, Cblu)
End Sub
'Hscroll2,3 も同様
```

General

```
Dim Cred As Byte
Dim Cgre As Byte
Dim Cblu As Byte

Dim X As Long
Dim Y As Long

Dim i As Long
Dim j As Long

Dim lum

Dim SH As Byte

Dim SHred As Byte
Dim SHgre As Byte
Dim SHblu As Byte

Dim BSH(1 To 4, 1 To 4) As Byte
Dim MTX(1 To 4, 1 To 2) As Byte

Dim BX As Long
Dim BY As Long
Dim Bi As Long
Dim Bj As Long
```

Command1

```
Private Sub Command1_Click()
   Picture3.BackColor = RGB(255, 255, 255)
   BX = 1
   BY = 1
   For Bj = 1 To 350 Step 1
   For Bi = 1 To 350 Step 1
        If lum < BSH(1, 1) Then
             Picture3.PSet (BX, BY), RGB(0, 0, 0)
        ElseIf lum > BSH(1, 1) Or lum = BSH(1, 1) Then
             Picture3.PSet (BX, BY), RGB(255, 255, 255)
        End If
        BX = BX + 1
         If lum < BSH(1, 2) Then
              Picture3.PSet (BX, BY), RGB(0, 0, 0)
         ElseIf lum > BSH(1, 2) Or lum = BSH(1, 2) Then
              Picture3.PSet (BX, BY), RGB(255, 255, 255)
         End If
         BX = BX + 1
         If lum < BSH(1, 3) Then
              Picture3.PSet (BX, BY), RGB(0, 0, 0)
         ElseIf lum > BSH(1, 3) Or lum = BSH(1, 3) Then
              Picture3.PSet (BX, BY), RGB(255, 255, 255)
         End If
         BX = BX + 1
         If lum < BSH(1, 4) Then
              Picture3.PSet (BX, BY), RGB(0, 0, 0)
         ElseIf lum > BSH(1, 4) Or lum = BSH(1, 4) Then
              Picture3.PSet (BX, BY), RGB(255, 255, 255)
         End If
         BX = BX + 1
    Next Bi
   BY = BY + 1
   BX = 1
```

```
For Bi = 1 To 350 Step 1
    If lum < BSH(2, 1) Then
        Picture3.PSet (BX, BY), RGB(0, 0, 0)
    ElseIf lum > BSH(2, 1) Or lum = BSH(2, 1) Then
        Picture3.PSet (BX, BY), RGB(255, 255, 255)
    End If
    BX = BX + 1
    If lum < BSH(2, 2) Then
        Picture3.PSet (BX, BY), RGB(0, 0, 0)
    ElseIf lum > BSH(2, 2) Or lum = BSH(2, 2) Then
        Picture3.PSet (BX, BY), RGB(255, 255, 255)
    End If
    BX = BX + 1
    If lum < BSH(2, 3) Then
        Picture3.PSet (BX, BY), RGB(0, 0, 0)
    ElseIf lum > BSH(2, 3) Or lum = BSH(2, 3) Then
        Picture3.PSet (BX, BY), RGB(255, 255, 255)
    End If
    BX = BX + 1
    If lum < BSH(2, 4) Then
        Picture3.PSet (BX, BY), RGB(0, 0, 0)
    ElseIf lum > BSH(2, 4) Or lum = BSH(2, 4) Then
        Picture3.PSet (BX, BY), RGB(255, 255, 255)
    End If
    BX = BX + 1
Next Bi
BY = BY + 1
BX = 1
For Bi = 1 To 350 Step 1
    If lum < BSH(3, 1) Then
        Picture3.PSet (BX, BY), RGB(0, 0, 0)
    ElseIf lum > BSH(3, 1) Or lum = BSH(3, 1) Then
        Picture3.PSet (BX, BY), RGB(255, 255, 255)
    End If
    BX = BX + 1
```

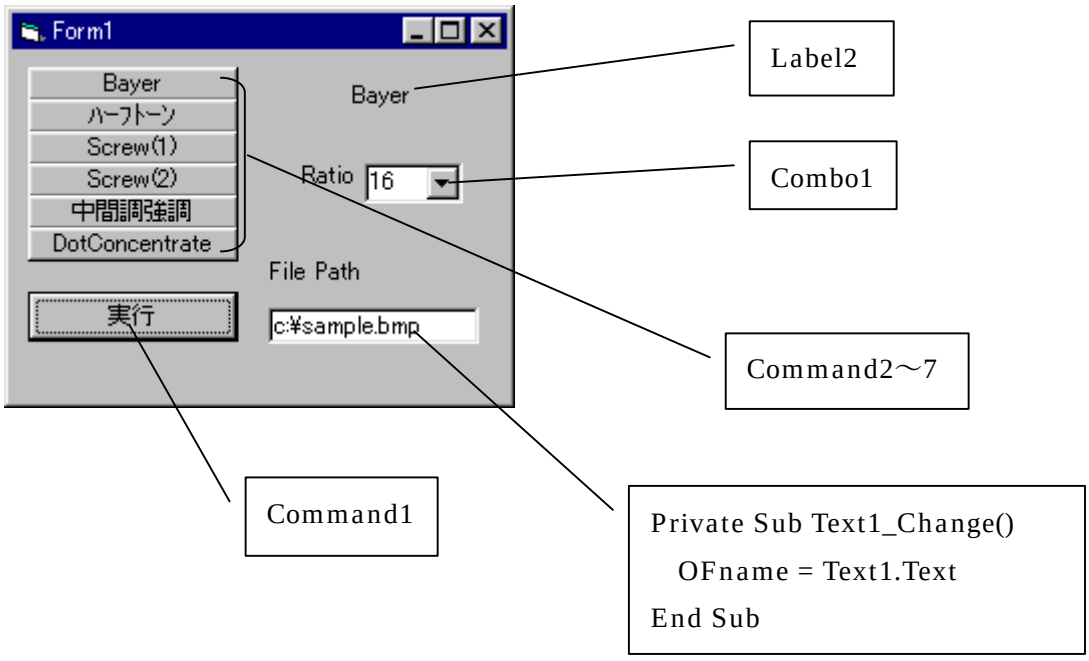
```
      If lum < BSH(3, 2) Then
            Picture3.PSet (BX, BY), RGB(0, 0, 0)
       ElseIf lum > BSH(3, 2) Or lum = BSH(3, 2) Then
            Picture3.PSet (BX, BY), RGB(255, 255, 255)
       End If
       BX = BX + 1
       If lum < BSH(3, 3) Then
            Picture3.PSet (BX, BY), RGB(0, 0, 0)
       ElseIf lum > BSH(3, 3) Or lum = BSH(3, 3) Then
            Picture3.PSet (BX, BY), RGB(255, 255, 255)
       End If
       BX = BX + 1
       If lum < BSH(3, 4) Then
            Picture3.PSet (BX, BY), RGB(0, 0, 0)
       ElseIf lum > BSH(3, 4) Or lum = BSH(3, 4) Then
            Picture3.PSet (BX, BY), RGB(255, 255, 255)
       End If
       BX = BX + 1
  Next Bi
BY = BY + 1
BX = 1
For Bi = 1 To 350 Step 1
      If lum < BSH(4, 1) Then
           Picture3.PSet (BX, BY), RGB(0, 0, 0)
      ElseIf lum > BSH(4, 1) Or lum = BSH(4, 1) Then
           Picture3.PSet (BX, BY), RGB(255, 255, 255)
      End If
      BX = BX + 1
      If lum < BSH(4, 2) Then
           Picture3.PSet (BX, BY), RGB(0, 0, 0)
      ElseIf lum > BSH(4, 2) Or lum = BSH(4, 2) Then
           Picture3.PSet (BX, BY), RGB(255, 255, 255)
      End If
      BX = BX + 1
```

```
        If lum < BSH(4, 3) Then
            Picture3.PSet (BX, BY), RGB(0, 0, 0)
        ElseIf lum > BSH(4, 3) Or lum = BSH(4, 3) Then
            Picture3.PSet (BX, BY), RGB(255, 255, 255)
        End If
        BX = BX + 1
        If lum < BSH(4, 4) Then
            Picture3.PSet (BX, BY), RGB(0, 0, 0)
        ElseIf lum > BSH(4, 4) Or lum = BSH(4, 4) Then
            Picture3.PSet (BX, BY), RGB(255, 255, 255)
        End If
        BX = BX + 1
    Next Bi
    BY = BY + 1
    BX = 1
    Next Bj
End Sub
```

Command2

```
Private Sub Command2_Click() 'Command3～７も同様（パラメータの変更のみ）
   BSH(1, 1) = 0 * Val(Combo1.Text)
   BSH(1, 2) = 8 * Val(Combo1.Text)
   BSH(1, 3) = 2 * Val(Combo1.Text)
   BSH(1, 4) = 10 * Val(Combo1.Text)
   BSH(2, 1) = 12 * Val(Combo1.Text)
   BSH(2, 2) = 4 * Val(Combo1.Text)
   BSH(2, 3) = 14 * Val(Combo1.Text)
   BSH(2, 4) = 6 * Val(Combo1.Text)
   BSH(3, 1) = 3 * Val(Combo1.Text)
   BSH(3, 2) = 11 * Val(Combo1.Text)
   BSH(3, 3) = 1 * Val(Combo1.Text)
   BSH(3, 4) = 9 * Val(Combo1.Text)
   BSH(4, 1) = 15 * Val(Combo1.Text)
   BSH(4, 2) = 7 * Val(Combo1.Text)
   BSH(4, 3) = 13 * Val(Combo1.Text)
   BSH(4, 4) = 5 * Val(Combo1.Text)
   Text2.Text = BSH(1, 1)
   Text3.Text = BSH(1, 2)
   Text4.Text = BSH(1, 3)
   Text5.Text = BSH(1, 4)
   Text6.Text = BSH(2, 1)
   Text7.Text = BSH(2, 2)
   Text8.Text = BSH(2, 3)
   Text9.Text = BSH(2, 4)
   Text10.Text = BSH(3, 1)
   Text11.Text = BSH(3, 2)
   Text12.Text = BSH(3, 3)
   Text13.Text = BSH(3, 4)
   Text14.Text = BSH(4, 1)
   Text15.Text = BSH(4, 2)
   Text16.Text = BSH(4, 3)
   Text17.Text = BSH(4, 4)
End Sub
```

付録-3 ディザ法を用いた画像減色フィルタ



General

```
Dim n As Long     '出力処理座標
Dim o As Long     '出力処理座標
Dim p As Byte    'ヘッダ出力用
Dim x As Byte    'マトリクス座標
Dim y As Byte
Dim WCD As Byte '白
Dim BCD As Byte '黒
Dim tmp As Byte    'テンポラリ
```

Command1

```
Private Sub Command1_Click()
  Command1.Caption = "処理中"
  i = 0
  k = 0
  j = 0
  m = 0
  l = 0
  n = 0
  o = 0
  '--------  ヘッダー情報の取得--------
  Open OFname For Binary As #1

     For i = 1 To 54 Step 1    'ヘッダーを配列に格納
          Get #1, , HdR
          Hed(i) = HdR
      Next i
     '画像サイズを取得
     WH = 2 ^ 24 * Hed(22) + 2 ^ 16 * Hed(21) + 2 ^ 8 * Hed(20) + Hed(19)
     HI = 2 ^ 24 * Hed(26) + 2 ^ 16 * Hed(25) + 2 ^ 8 * Hed(24) + Hed(23)
     ReDim lum(0 To WH, 0 To HI) As Byte '輝度を格納する配列を定義
     ReDim Dot(0 To WH, 0 To HI) As Byte '画素のドットを格納
```

```
'--------  輝度計算--------
      For j = 1 To HI
            For k = 1 To WH Step 1
                Get #1, , Cblu
                Get #1, , Cgre
                Get #1, , Cred
                lum(k, j) = (0.298912 * Cred + 0.586611 * Cgre + 0.114478 * Cblu)
            Next k
      Next j
Close #1

'--------閾値処理--------
x = 1
y = 1
For m = 1 To HI Step 1
      For l = 1 To WH Step 1
            If BSH(x, y) > lum(l, m) Then
                Dot(l, m) = 0
            Else
                Dot(l, m) = 1
            End If
            If x = 4 Then
                x = 1
            ElseIf x <> 4 Then
                x = x + 1
            End If
      Next l
      If y = 4 Then
            y = 1
      ElseIf y <> 4 Then
            y = y + 1
      End If
Next m
```

```
'--------出力処理--------
WCD = 0
BCD = 255
Open SFname For Binary As #2
   For p = 1 To 54
        Put #2, p, Hed(p)
   Next p
   For o = 1 To HI Step 1
        For n = 1 To WH Step 1
             If Dot(n, o) = 1 Then
                  Put #2, , BCD
                  Put #2, , BCD
                  Put #2, , BCD
             Else
                  Put #2, , WCD
                  Put #2, , WCD
                  Put #2, , WCD
             End If
        Next n
   Next o
Close #2
```

Command2

```
Private Sub Command2_Click()'Command3～７も同様（パラメータの変更のみ）
   Label2.Caption = "Bayer"
   SFname = "bayer.bmp"
   ‘ ディザ行列を配列に格納
   BSH(1, 1) = 0 * Val(Combo1.Text)
   BSH(1, 2) = 8 * Val(Combo1.Text)
   BSH(1, 3) = 2 * Val(Combo1.Text)
   BSH(1, 4) = 10 * Val(Combo1.Text)
   BSH(2, 1) = 12 * Val(Combo1.Text)
   BSH(2, 2) = 4 * Val(Combo1.Text)
   BSH(2, 3) = 14 * Val(Combo1.Text)
   BSH(2, 4) = 6 * Val(Combo1.Text)
   BSH(3, 1) = 3 * Val(Combo1.Text)
   BSH(3, 2) = 11 * Val(Combo1.Text)
   BSH(3, 3) = 1 * Val(Combo1.Text)
   BSH(3, 4) = 9 * Val(Combo1.Text)
   BSH(4, 1) = 15 * Val(Combo1.Text)
   BSH(4, 2) = 7 * Val(Combo1.Text)
   BSH(4, 3) = 13 * Val(Combo1.Text)
   BSH(4, 4) = 5 * Val(Combo1.Text)
End Sub
```

付録-4　VHDL による Bayer 型ディザフィルターの記述

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

entity Dither is
   Port ( rgb_red : in std_logic_vector(7 downto 0);
      rgb_green : in std_logic_vector(7 downto 0);
      rgb_blue : in std_logic_vector(7 downto 0);
      CLK : in std_logic;
      RST : in std_logic;
      dth_out : out std_logic;
      test_out : out std_logic_vector(11 downto 0));
end Dither;

architecture RTL of Dither is

signal int_lum :integer range 0 to 2550;
signal int_red :integer range 0 to 255;
signal int_green :integer range 0 to 255;
signal int_blue :integer range 0 to 255;

subtype Bayer is integer range 0 to 2240; --subtype を定義
type mtx1 is array (1 to 4)of Bayer;   --Dither 行列を格納する配列を作成
type mtx2 is array (1 to 4)of Bayer;
type mtx3 is array (1 to 4)of Bayer;
type mtx4 is array (1 to 4)of Bayer;
constant bayer1 : mtx1 :=(0,1280,320,1600); --配列に Bayer 型行列の閾値を格納
constant bayer2 : mtx2 :=(1920,640,2240,960);
constant bayer3 : mtx3 :=(480,1760,160,1440);
constant bayer4 : mtx4 :=(240,1120,2080,800);

begin
   process(CLK,RST)begin
      if(RST='1')then
         null;
      elsif(CLK'event and CLK='1')then
         int_red<=CONV_INTEGER(rgb_red);
         int_green<=CONV_INTEGER(rgb_green);
         int_blue<=CONV_INTEGER(rgb_blue);
         int_lum<=(int_red * 3) + (int_green * 6) + int_blue; --輝度の算出
         test_out<=CONV_STD_LOGIC_VECTOR(int_lum,12);
         for y in 0 to 59 loop
            for i in 0 to 79 loop
               if (int_lum>bayer1(1)) then
                  dth_out<='0';
               else
                  dth_out<='1';
               end if;
```

```
        if (int_lum>bayer1(2)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
        if (int_lum>bayer1(3)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
        if (int_lum>bayer1(4)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
      end loop;
      for i in 0 to 79 loop
        if (int_lum>bayer2(1)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
        if (int_lum>bayer2(2)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
        if (int_lum>bayer2(3)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
        if (int_lum>bayer2(4)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
      end loop;
      for i in 0 to 79 loop
        if (int_lum>bayer3(1)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
        if (int_lum>bayer3(2)) then
          dth_out<='0';
        else
          dth_out<='1';
        end if;
```

```
            if (int_lum>bayer3(3)) then
              dth_out<='0';
            else
              dth_out<='1';
            end if;
            if (int_lum>bayer3(4)) then
              dth_out<='0';
            else
              dth_out<='1';
            end if;
          end loop;
          for i in 0 to 79 loop
            if (int_lum>bayer4(1)) then
              dth_out<='0';
            else
              dth_out<='1';
            end if;
            if (int_lum>bayer4(2)) then
              dth_out<='0';
            else
              dth_out<='1';
            end if;
            if (int_lum>bayer4(3)) then
              dth_out<='0';
            else
              dth_out<='1';
            end if;
            if (int_lum>bayer4(4)) then
              dth_out<='0';
            else
              dth_out<='1';
            end if;
          end loop;
        end loop;
      end if;
    end process;
end RTL;
```

テストベンチ

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

COMPONENT dither
  PORT(
    rgb_red : IN std_logic_vector(7 downto 0);
    rgb_green : IN std_logic_vector(7 downto 0);
    rgb_blue : IN std_logic_vector(7 downto 0);
    CLK : IN std_logic;
    RST : IN std_logic;
    dth_out : OUT std_logic;
    test_out : OUT std_logic_vector(11 downto 0)
  );
END COMPONENT;

SIGNAL rgb_red :   std_logic_vector(7 downto 0);
SIGNAL rgb_green :   std_logic_vector(7 downto 0);
SIGNAL rgb_blue :   std_logic_vector(7 downto 0);
SIGNAL CLK :   std_logic;
SIGNAL RST :   std_logic;
SIGNAL dth_out :   std_logic;
SIGNAL test_out : std_logic_vector(11 downto 0);
constant STEP :time :=20ns;

BEGIN

  uut: dither PORT MAP(
    rgb_red => rgb_red,
    rgb_green => rgb_green,
    rgb_blue => rgb_blue,
    CLK => CLK,
    RST => RST,
    dth_out => dth_out,
    test_out => test_out
  );

-- *** Test Bench - User Defined Section ***
   tb : PROCESS --CLK Pals
   BEGIN
      CLK<='0';
      wait for STEP;CLK<='1';
      wait for STEP;
   END PROCESS;
```

```
   process --RST pals
   begin
      RST<='1';
      wait for STEP*2;RST<='0';
      wait;
   end process;

   process
      begin
      wait for STEP *1;rgb_red<="11111111";rgb_green<="11111111"
      rgb_blue<="11111111";
      wait for STEP *5;rgb_red<="00000000";
      wait for STEP *5;rgb_green<="00000000";
      wait for STEP *5;rgb_blue<="00000000";
      wait for STEP;
   end process;
-- *** End Test Bench - User Defined Section ***

END;
```

Synthesis Report

```
Synthesizing Unit <dither>.
    Related source file is C:/xilinx_webpack/bin/Dither/Dither.vhd.
    Found 1-bit register for signal <dth_out>.
    Found 12-bit register for signal <test_out>.
    Found 12-bit comparator greater for signal <$n0002> created at line 125.
    Found 8x2-bit multiplier for signal <$n0003> created at line 40.
    Found 8x3-bit multiplier for signal <$n0005> created at line 40.
    Found 11-bit adder carry out for signal <$n0006> created at line 40.
    Found 10-bit adder carry out for signal <$n0007>.
    Found 8-bit register for signal <int_blue>.
    Found 8-bit register for signal <int_green>.
    Found 12-bit register for signal <int_lum>.
    Found 8-bit register for signal <int_red>.
    Summary:
        inferred   49 D-type flip-flop(s).
        inferred    2 Adder/Subtracter(s).
        inferred    2 Multiplier(s).
        inferred    1 Comparator(s).
Unit <dither> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                      : 6
  8-bit register                 : 3
  12-bit register                : 2
  1-bit register                 : 1
# Adders/Subtractors             : 2
  11-bit adder carry out         : 1
  10-bit adder carry out         : 1
# Multipliers                    : 2
  8x2-bit multiplier             : 1
  8x3-bit multiplier             : 1
# Comparators                    : 1
  12-bit comparator greater      : 1
```

Performance Summary Report

Design: dither
Device: XC95144-7-PQ100
Speed File: Version 3.0
Program: Timing Report Generator: version E.32
Date: Thu Jan 31 20:14:05 2002

Performance Summary:

Clock net 'clk' path delays:

Clock Pad to Output Pad (tCO) : 4.5ns (1 macrocell levels)
Clock Pad 'clk' to Output Pad 'dth_out' (GCK)

Clock to Setup (tCYC) : 72.5ns (6 macrocell levels)
Clock to Q, net 'int_red_2.Q' to DFF Setup(D) at 'int_lum_10.D' (GCK)

Setup to Clock at the Pad (tSU) : 6.5ns (0 macrocell levels)
Data signal 'rst' to DFF D input Pin at 'int_lum_11.D'
Clock pad 'clk'
(GCK)

Minimum Clock Period: 72.5ns
Maximum Internal Clock Speed: 13.7Mhz
(Limited by Cycle Time)

付録-5 VHDL による Run Length Encoder の記述

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

entity RLE_Enc is
     Port ( CLK : in std_logic;
               RST : in std_logic;
               IN_D : in std_logic_vector(7 downto 0);
               OUT_D : out std_logic_vector(7 downto 0);
               OUT_R : out std_logic_vector(7 downto 0));
end RLE_Enc;

architecture RTL of RLE_Enc is

signal BUF_A,BUF_B : std_logic_vector(7 downto 0);
signal CNT_A : std_logic_vector(7 downto 0);

begin
  process (CLK,RST) begin
    if (RST='1') then
      CNT_A <= "00000000";
    elsif (CLK'event and CLK='1') then
      BUF_A <= IN_D;
      if (BUF_A = BUF_B) then
        if (CNT_A = "11111111" ) then
          OUT_R <= "11111111";
          OUT_D <=BUF_A;
          CNT_A <= "00000000";
        else
          CNT_A <= CNT_A + '1';
        end if;
      else
        OUT_R <= CNT_A;
        OUT_D <= BUF_B;
        CNT_A <= "00000000";
        BUF_B <= BUF_A;
      end if;
  end process;
end RTL;
```

テストベンチ

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

  COMPONENT rle_enc
    PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    IN_D : IN std_logic_vector(7 downto 0);
    OUT_D : OUT std_logic_vector(7 downto 0);
    OUT_R : OUT std_logic_vector(7 downto 0)
    );
  END COMPONENT;

  SIGNAL CLK :   std_logic;
  SIGNAL RST :   std_logic;
  SIGNAL IN_D :   std_logic_vector(7 downto 0);
  SIGNAL OUT_D :   std_logic_vector(7 downto 0);
  SIGNAL OUT_R :   std_logic_vector(7 downto 0);
  constant STEP : time := 20ns;

BEGIN

  uut: rle_enc PORT MAP(
    CLK => CLK,
    RST => RST,
    IN_D => IN_D,
    OUT_D => OUT_D,
    OUT_R => OUT_R
  );
-- *** Test Bench - User Defined Section ***
   tb : PROCESS   -- CLOCK PALS
   BEGIN
    CLK <='0';
    wait for STEP; CLK <='1';
    wait for STEP;
    -- wait; -- will wait forever
   END PROCESS;

  process   -- RESTE PALS
  begin
    RST <='1';
    wait for STEP*2; RST <='0';
    wait;
  end process;
```

```
   process
   begin
      wait for STEP    * 5; IN_D <="00000000";
      wait for STEP * 5;              IN_D <="11111111";
      wait for step;
   end process;
-- *** End Test Bench - User Defined Section ***

END;
```

Synthesis Report

```
Synthesizing Unit <rle_enc>.
    Related source file is C:/xilinx_webpack/bin/RLE_Enc/RLE_Enc.vhd.
    Found 8-bit register for signal <out_d>.
    Found 8-bit register for signal <out_r>.
    Found 8-bit comparator equal for signal <$n0007> created at line 25.
    Found 8-bit adder for signal <$n0011> created at line 31.
    Found 8-bit register for signal <buf_a>.
    Found 8-bit register for signal <buf_b>.
    Found 8-bit register for signal <cnt_a>.
    Found 8 1-bit 2-to-1 multiplexers.
    Summary:
        inferred   40 D-type flip-flop(s).
        inferred    1 Adder/Subtracter(s).
        inferred    1 Comparator(s).
        inferred    8 Multiplexer(s).
Unit <rle_enc> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                      : 5
   8-bit register                : 5
# Multiplexers                   : 1
   2-to-1 multiplexer            : 1
# Adders/Subtractors             : 1
   8-bit adder                   : 1
# Comparators                    : 1
   8-bit comparator equal        : 1
```

Performance Summary Report

Design: rle_enc
Device: XC95144-7-PQ100
Speed File: Version 3.0
Program: Timing Report Generator: version E.32
Date: Fri Nov 16 08:22:07 2001

Performance Summary:

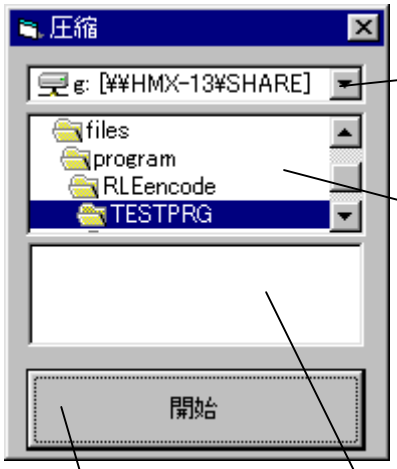
Clock net 'clk' path delays:

Clock to Setup (tCYC) : 8.0ns (1 macrocell levels)
Clock to Q, net 'buf_a_0.Q' to DFF Setup(D) at 'buf_a_0.D' (GCK)
Target FF drives output net 'buf_a_0'

Setup to Clock at the Pad (tSU) : 5.5ns (0 macrocell levels)
Data signal 'rst' to DFF D input Pin at 'buf_a_0.D'
Clock pad 'clk'
(GCK)

Minimum Clock Period: 8.0ns
Maximum Internal Clock Speed: 125.0Mhz
(Limited by Clock Pulse Width)

付録-6　Run Length Encoder 評価プログラム




```
Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
End Sub
```

```
Private Sub Dir1_Change()
    File1.Path = Dir1.Path
End Sub
```

```
Private Sub File1_Click()
    If Right(Dir1.Path, 1) = "¥" Then
        Fname = Dir1.Path & File1.filename
    Else
        Fname = Dir1.Path & "¥" & File1.filename
    End If
End Sub
```

Command1

General

```
Dim Cnt As Long
Dim i As Long
Dim BufA
Dim BufB
Dim PrvA
Dim PrvCnt As Long
Dim AllCntB As Long
Dim EndCnt As Long
Dim Ans(1 To 8) As Currency
Dim ModAns(1 To 8) As Byte
Dim AAA
Dim BBB
Dim Fname As String
```

Command1

```
Private Sub Command1_Click()
  Command1.Caption = "処理中"
  If Fname = "" Then
      ret = MsgBox("ファイルが選択されていません", 48, "エラー")
  Else
  Cnt = 0 'カウンタの初期値を設定
  PrvCnt = 0
  AllCntA = 0
  AllCntB = 0
  Open Fname For Input As #1
  Open Fname & ".RLE" For Append As #2
      Do Until EOF(1)
          i = i + 1
          Line Input #1, BufA
          If BufA Xor BufB Then
              PrvA = BufB '前回のバッファの内容を格納
              BufB = BufA 'バッファの内容を更新
              PrvCnt = Cnt + 1 '前回のカウント値を記録
              AllCntB = AllCntB + PrvCnt
              Cnt = 0 'カウンタをリセット
              ModAns(1) = PrvCnt Mod 2
              If ModAns(1) = 1 Then
                  Ans(1) = PrvCnt / 2 - 0.5
              Else
                  Ans(1) = PrvCnt / 2
              End If
              ModAns(2) = Ans(1) Mod 2
              If ModAns(2) = 1 Then
                  Ans(2) = Ans(1) / 2 - 0.5
              Else
                  Ans(2) = Ans(1) / 2
              End If
              ModAns(3) = Ans(2) Mod 2
              If ModAns(3) = 1 Then
                  Ans(3) = Ans(2) / 2 - 0.5
              Else
                  Ans(3) = Ans(2) / 2
              End If
              ModAns(4) = Ans(3) Mod 2
              If ModAns(4) = 1 Then
                  Ans(4) = Ans(3) / 2 - 0.5
              Else
                  Ans(4) = Ans(3) / 2
              End If
```

```
            ModAns(5) = Ans(4) Mod 2
            If ModAns(5) = 1 Then
                Ans(5) = Ans(4) / 2 - 0.5
            Else
                Ans(5) = Ans(4) / 2
            End If
            ModAns(6) = Ans(5) Mod 2
            If ModAns(6) = 1 Then
                Ans(6) = Ans(5) / 2 - 0.5
            Else
                Ans(6) = Ans(5) / 2
            End If
            ModAns(7) = Ans(6) Mod 2
            If ModAns(7) = 1 Then
                Ans(7) = Ans(6) / 2 - 0.5
            Else
                Ans(7) = Ans(6) / 2
            End If
            ModAns(8) = Ans(7) Mod 2
            If ModAns(8) = 1 Then
                Ans(8) = Ans(7) / 2 - 0.5
            Else
                Ans(8) = Ans(7) / 2
            End If
             AAA = ModAns(8) & ModAns(7) & ModAns(6) & ModAns(5);
                & ModAns(4) & ModAns(3) & ModAns(2) & ModAns(1)
            If PrvA = "" Then
            Else
                Print #2, PrvA & " " & AAA
            End If
        ElseIf EOF(1) Then 'ファイルエンド時の処理
            EndCnt = (i - AllCntB) + 1
            ModAns(1) = EndCnt Mod 2
            If ModAns(1) = 1 Then
                Ans(1) = EndCnt / 2 - 0.5
            Else
                Ans(1) = EndCnt / 2
            End If
            ModAns(2) = Ans(1) Mod 2
            If ModAns(2) = 1 Then
                Ans(2) = Ans(1) / 2 - 0.5
            Else
                Ans(2) = Ans(1) / 2
            End If
            ModAns(3) = Ans(2) Mod 2
            If ModAns(3) = 1 Then
                Ans(3) = Ans(2) / 2 - 0.5
            Else
                Ans(3) = Ans(2) / 2
            End If
```

```
                ModAns(4) = Ans(3) Mod 2
                If ModAns(4) = 1 Then
                    Ans(4) = Ans(3) / 2 - 0.5
                Else
                    Ans(4) = Ans(3) / 2
                End If
                ModAns(5) = Ans(4) Mod 2
                If ModAns(5) = 1 Then
                    Ans(5) = Ans(4) / 2 - 0.5
                Else
                    Ans(5) = Ans(4) / 2
                End If
                ModAns(6) = Ans(5) Mod 2
                If ModAns(6) = 1 Then
                    Ans(6) = Ans(5) / 2 - 0.5
                Else
                    Ans(6) = Ans(5) / 2
                End If
                ModAns(7) = Ans(6) Mod 2
                If ModAns(7) = 1 Then
                    Ans(7) = Ans(6) / 2 - 0.5
                Else
                    Ans(7) = Ans(6) / 2
                End If
                ModAns(8) = Ans(7) Mod 2
                If ModAns(8) = 1 Then
                    Ans(8) = Ans(7) / 2 - 0.5
                Else
                    Ans(8) = Ans(7) / 2
                End If

                BBB = ModAns(8) & ModAns(7) & ModAns(6) & ModAns(5);
                  & ModAns(4) & ModAns(3) & ModAns(2) & ModAns(1)
                Print #2, BufA & " " & BBB
            Else
                Cnt = Cnt + 1
            End If
        Loop
    Close #1, #2
    End If
    Command1.Caption = "完了"
End Sub
```