卒 業 研 究 報 告

題　　目

ＶＨＤＬによる画像フィルタ回路の設計

指 導 教 員

橘　昌良　助教授

報 告 者

河村　恵美

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 概要

ＶＨＤＬを使用したＬＳＩ設計の方法を学ぶと共に、ＶＨＤＬを使用して実際にデジタル回路を設計する。ここでは画像フィルタ回路の設計を行う。また、これに付随して必要な、ＶＨＤＬの文法や設計支援ツールの使用方法、システム設計についての知識を得る為に、文献調査や演習等も行う。

本論分の構成を以下に述べる。

- 第１章では、本論分の概要を述べる。
- 第２章では、本論文の目的を述べる。
- 第３章では、LSI 設計の歴史・開発の背景、VHDL について述べる。
- 第４章では、システム設計の概要、設計工程とその作業について述べる。
- 第５章では、画像処理とそのアルゴリズムについて述べる。
- 第６章では、画像フィルタ回路の設計と製作について述べる。
- 第７章では、本研究の考察を述べる。
- 第８章には、本研究のまとめを記述した。
- 第９章には、参考文献を紹介する。
- 第１０章は、謝辞を述べる。

最後に付録として、C 言語のソースを添付するものである。

2 目的

ＬＳＩ設計に必要な知識と技術を、実際のデジタル回路設計を通して学び、ＬＳＩ設計技術者の基礎を身に付ける。

本研究では、現在ＬＳＩ開発・設計現場で主流となっている、ハードウェア記述言語（以下ＨＤＬ）を用い、ＨＤＬからゲート回路を自動的に合成（Synthesis）する設計手法について学ぶ。使用するＨＤＬ言語は Very High Speed Integrated Circuit Hardware Description Language（以下 VHDL）とし、VHDL の文法や基本構文を学ぶ。

また、実際にデジタル回路設計を行いながら、システム設計に関しても、文献等で調査学習し、これらの知識をもとに再度回路設計に還元しつつ進める。設計するデジタル回路は「画像フィルタ回路」で、この回路のシステム実現の為の画像処理やフィルタアルゴリズムについても学習する必要がある。

3 LSI 設計

LSI 開発・設計の種類から、VHDL を用いた設計自動化まで、LSI と VHDL の歴史的流れを把握し、LSI 設計工程と作業内容について学ぶ。

3.1 LSI 開発の背景

LSI を、汎用 LSI と、カスタム LSI の二つに大別する。汎用 LSI を、市場において誰でもが購入する事のできる、標準的な論理を構成する IC や CPU、ROM/RAM、通信用、テレビ・チューナー用といったアプリケーション固有の LSI であるが、複数ユーザ向け LSI の ASSP（Application Specific Standard Product）と分ける。カスタム LSI を、特定ユーザごとに開発される、特定用途向け LSI の ASIC（Application Specific Integrated Circuit）と分けると、それぞれの開発の背景は以下の様にまとめられる。

1960 年代後半、電卓に論理用集積回路（ロジック IC）が利用される様になり、1970 年代の初めには、インテル社によって、回路規模数千ゲートの初めての 4 ビット CPU、i4004 が製品化された。日本においても、電卓、時計、電子楽器用等にカスタムＬＳＩが市場に出荷され、応用機器のデジタル化を前進させた。【1】

1970 年代後半には数万ゲート規模の 8 ビット CPU、1970 年代の終わりから 1980 年代にかけて、数十万ゲート規模の 16 ビット CPU が開発された。さらに 1990 年代に入ると、回路規模数百万ゲートの 64 ビット CPU が開発された。【3】

また、メモリも 1986 年には１Mbit　DRAM が、1997 年には 64Mbit　DRAM が開発された。【1】

LSI の集積度は 6 年間でやく 10 倍になり、現在でも集積度は上がり続けている。この様な CPU とメモリの集積度の向上が、現在の小型で高性能な電子機器の出現に結びついていると言える。【3】

カスタム LSI は、汎用ロジック IC が 1980 年代半ばにはゲートアレイ、スタンダードセルを利用した ASIC の開発へと発展し、LSI ベンダーが設計を行うフルカスタム ASIC と、一般ユーザが設計するセミカスタム ASIC とに分類される。【1】【3】

ゲートアレイは設計した回路に従って、アレイ状に配置された未接続の汎用ロジック（セル）の配線の変更だけを行えば、回路機能が実現される。トランジスタの製造工程が終了したシリコン・ウェハに、数枚のマスクで配線の変更ができるので、製造コストや開発費が安く済み、開発期間も短い為に修正しやすい。

スタンダードセルは用意されたセルを組み合わせて回路ブロックを構築し、回路固有のマスクを起こして製造する。製造に必要なマスクをすべて作成する必要がある為、開発費が高くなり、製造期間も長くなる。【3】

ゲートアレイは、決められた大きさのセルを使用する為、回路ごとに LSI のサイズを最

小化する事ができないので、スタンダードセルに比べてチップあたりの単価は高くなる。スタンダードセルでは、性能を最適化した回路を実現しやすく、回路ごとにチップサイズを最小化できる為に、チップあたりの単価は安くなる。

最近では、FPGA（Field Programmable Gate Array）や CPLD（Complex Programmable Logic Device）の様に、プログラム可能なゲートアレイや、一つのチップでゲートアレイとスタンダードセルの両方を実現できるものも開発されている。

こうした LSI の製造工程には、1 ヶ月以上の長い期間と高い開発コストが必要であった。また、一端製造を終えた LSI に動作不良が見つかっても、パッケージングされた内部ゲート回路の動作をデバッグする事は難しく、ミスを発見した後の修正にも、再び長い期間と高いコストが必要となる。【3】

その為、製造工程に入る前に回路機能を検証し、設計ミスを削減する必要が出てきた。それにはコンピュータを導入し、単純作業で人手を煩わす事を少なくした、設計作業の自動化が進められた。その後も、コンピュータを使用した設計生産性の向上が進められ、LSI の大規模化と市場への早期投入に対応している。【3】

3.2 LSI 設計の自動化

LSI の集積度は 6 年で 10 倍というスピードで高くなり、現在でも高集積化が進んでいる。LSI の高集積化に反して、設計期間は短くすることが求められ続けてきた。こうした要求を解決する為には、設計生産性を向上させる必要があり、そのためにコンピュータを導入した設計の自動化が進められてきた。

3.3 VHDL

代表的な HDL として、VHDL と Verilog-HDL の 2 種類がある。今回は設計仕様のドキュメント言語として開発され、読解性に優れた VHDL を使用する。

3.3.1. VHDL の歴史

VHDL は、1970 年代に米国国防省で発足した VHSIC（Very High Speed Integrated Circuit）委員会で 1981 年に提唱された。

当時、国防省向けの ASIC の開発期間は、3 年から 4 年もかかるという状況で、その間に半導体のプロセス技術が進歩し、開発当初に最速の ASIC を使用しても、開発の完了時点では時代遅れの産物となってしまう、といった問題点が生じていた。【2】

また、この頃大規模 IC の開発には、より上位のレベルで検証する事が求められていた事もあり、直接ロジック・ゲートを回路図で入力するのではなく、ハードウェア記述言語（HDL）を使用して設計する事により、開発完了時に最速の AISC が使用できる様にする必要があった。

こうして、1983 年には米国空軍の出資によって開発チームが発足、VHDL の仕様作成が

始まり、1985 年に作業終了、1986 年にはマニュアルにまとめられ、バージョン 7.2 として公開された。

その後、1986 年には米国電気電子技術者協会（IEEE：Institute of Electrical and Electronic Engineers）で、標準化作業が始まり、1987 年 5 月には言語仕様書（LRM：Language Reference Manual）が作成され、12 月には IEEE-1076 として承認されている。

言語仕様は 5 年ごとに見直し、改訂され、1992 年には IEEE-1164 として採択されている。【3】【2】

3.3.2. VHDL 設計フロー

各参考文献を元にまとめた、VHDL によるハードウェア設計の設計工程を下図 3-1 に示す。

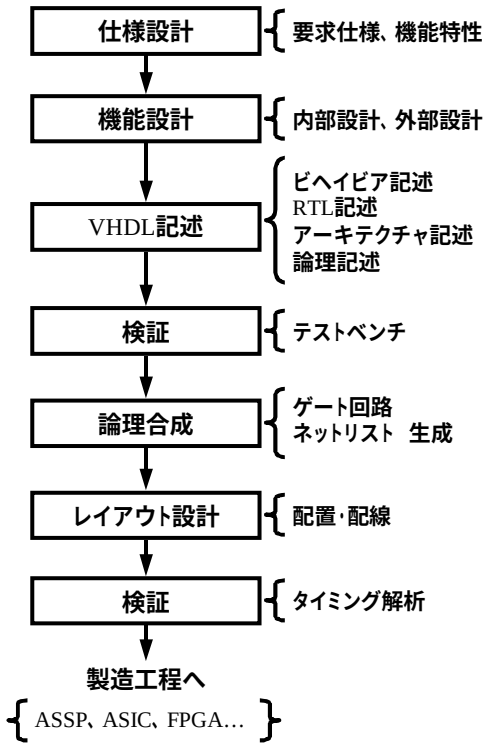


図 3-1　VHDL 設計工程

まず、仕様設計で、設計しようとする回路のシステムに要求される特性や機能について検討した後、アルゴリズムを構成する各部分をどういった構造に基づいて実現するかを検討する。ソフトウェアとして実現する場合や汎用 ASSP を使用したり、ASIC として開発したりする方法がある。

次に、機能設計で決定した内部構成をもとに、VHDL 記述を行う。VHDL の記述レベル

には、システム全体の記述が可能なアーキテクチャ・レベルや、レジスタの動作を明確にしたロジック回路生成が可能な RTL（Register Transfer Level）、回路の振る舞いだけを考えて抽象度の高い記述が可能なビヘイビア・レベル、論理レベル等、様々なレベルの記述が可能である。

記述した内容は、動作確認用の信号を与える HDL 記述のテストベンチを作成して、シミュレータ上でその信号を入力して動作を検証する。

検証の終わった VHDL 記述に対して、論理合成ツールを用いて論理合成を行うと、自動的にゲート回路が生成される。合成された論理回路は最終的には、ゲートの接続情報を示すネットリストとなり、レイアウト設計では、実際にチップを製造する為に必要なマスクを作成する為に、セルの自動配置・配線を行う。論理合成の段階では、仮の値を使用していたセル間の配線長も、ここで実際の配線長が得られ、実配線に基づいたタイミング解析と最終動作確認の為のゲートレベル・シミュレーションが可能である。

このあと、ゲートアレイ、スタンダードセルといったチップにあわせた枚数のレイアウト・マスクを作成し、製造工程に入る。【1】【2】【3】

3.4 ハードウェア設計と FPGA/CPLD

先にも述べたとおり、FPGA（Field Programmable Gate Array）や CPLD（Complex Programmable Logic Device）は書き替え可能なゲートアレイの一種である。チップ上に多数の基本セル、CLB(Configurable Logic Block)を配置し、回路データをダウンロードする事によって、CLB と CLB の間の配線部分を結線し任意の回路を生成できる。CLB は SRAM のファンクションジェネレータ（FG）と、フリップフロップ（FF）で構成されている。

FPGA や CPLD は、チップ内に CLB の様なプログラム可能なデバイスが組み込まれているおり、チップ面積が大きい為、製造コストが高くデバイス単価もゲートアレイに比べると高くなる。しかし、作業はコンピュータ上でプログラムする事で済み、手軽に大規模な回路を実現することができる為、結果的に開発コストは安く、ASIC 開発においてプロトタイプの生成等に使われることが多い。

近年、製品のライフサイクルが短くなり、製品をより短い開発期間で市場へ投入する必要に迫られている。最近では FPGA/CPLD の性能も向上して動作速度も速くなり、市販品でも数千から数万のゲートの回路を実現できる様になった為、量産時でも FPGA/CPLD を活用する例が増えている。【3】

4 システム設計

ソフトウェア・プログラミングと比較して、LSI 設計現場で回路を記述する様になってから、まだ日が浅い。システム開発の関連図書にしても、ハードウェア設計のために書かれたものより、ソフトウェアのシステム設計の為に書かれたものが多く、よりまとまってい

る印象を受ける。設計工程を見ると、要求仕様、外部設計、内部設計、プログラム作成、検証等共通点が多い。HDL も言語の一つであり、ソフトウェア開発のシステム化手法に寄る所は大きい。ここでもソフトウェアの手法を参考に、ハードウェア・システム設計への応用を検討したい。

4.1 システム開発の概要

ソフトウェア・システム開発のプロセスを図 4-1 の様な構成で表す。ここでのシステム開発とは、人間やコンピュータによる情報処理の仕組みを、設計、作成し、稼動させるまでの作業工程をいう。【5】

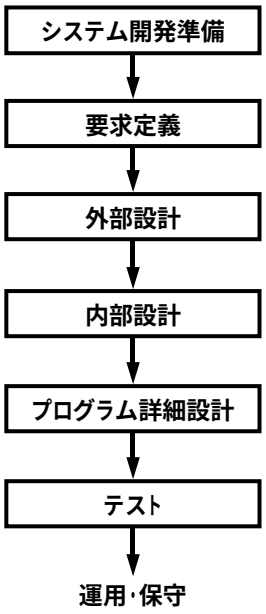


図 4-1　ソフトウェア・システム開発設計工程

4.2 システム開発準備

システム開発プロジェクトの、範囲や規模に応じて開発手法を決定し、使用ツールなどの開発環境を整える。現在、たくさんのシステム開発技法が提唱されているが、基本的な開発モデルとその特徴を学ぶ。

4.2.1. ウォータフォールモデル（waterfall model）

ウォータフォールモデルは、初期に確立されたモデルで、システム開発工程を各段階に区切って、一つの工程内で確認と検証を行う。一つの工程が終了してから次の工程へと、滝の水が流れるように、上流工程から下流工程へ順を追って作業を進めるモデルである。よって、基本的に上流工程への逆戻りはしない。このモデルの基本計画には、図 4-1 のシス

テム開発準備と要求定義が含まれている。

このモデルの長所は、作業手順がわかりやすく工程管理がしやすいことである。ゆえに大規模な基幹システムを開発する際に取り入れられることが多い。

短所は、要求仕様の変更がそれ以下の全ての工程へ反映される為、仕様変更が生じた場合の対応がしづらく、修正作業が困難な点や、後々になって前の工程の誤りが発覚した場合に、後戻り作業が発生する恐れがある、開発工程の後半にならないとユーザ要求が見える形にならないので、仕様ミスや設計の間違いを発見するのが遅れる、といった危険性がある。【5】【7】

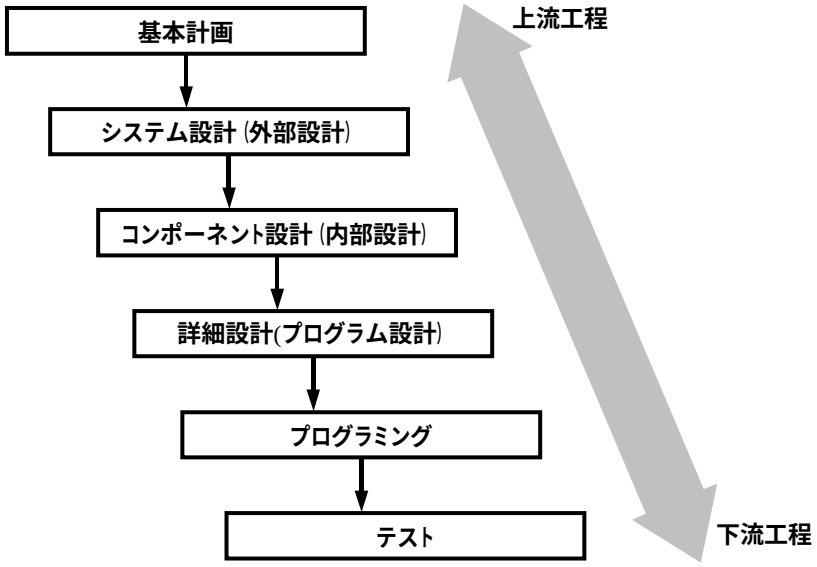


図 4-2　ウォータフォールモデル

4.2.2. プロトタイプモデル(prototype model)

ウォータフォールモデルの欠点を補うものとして、この様なモデルが提案されている。

プロトタイプモデルでは、詳細設計(プログラム設計)以前の、比較的開発の初期の段階で、システムの機能の一部やユーザインタフェース部分などを試作する。試作品をユーザと開発者で評価することによって、システムの要求仕様や条件を再確認して開発にあたる事ができる。

こうした手法により、ウォータフォールモデルで生じていた、ユーザと開発者の認識のずれやあいまいな点を少なくできる。この他にも、ユーザに開発作業への参加意識を持たせて協力を得られやすくなる、ユーザの潜在的ニーズを発掘できる、仕様の漏れを減らせる等の利点がある。

しかしながら、開発作業がユーザとの同期作業になる為、作業スケジュールの管理や調整が難しくなる事や、試作の評価段階で意見が発散してしまい収拾がつかなくなる等の短

所がある。【5】【7】

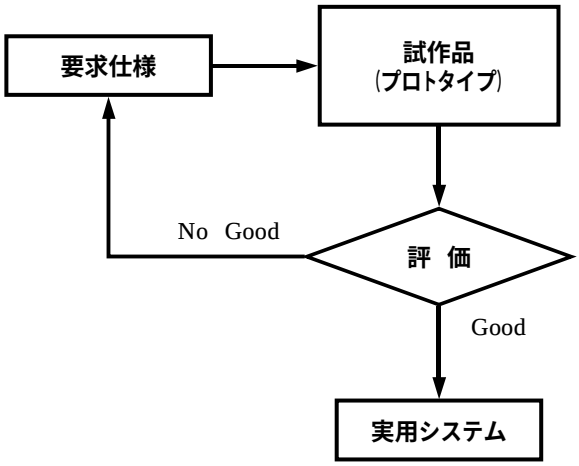


図 4-3　プロトタイプモデル

4.2.3. ラウンドトリップモデル（roundtrip model）

主にオブジェクト指向の設計で採用されている手法である。オブジェクト指向では、データと処理を別々に扱っていた従来のプログラムと異なり、データと処理を一体化（カプセル化）させて扱う。一つのオブジェクトの変更がそれ以外のオブジェクトには影響しないので、仕様の変更に柔軟に対処できる。

ラウンドトリップモデルでは、各工程の結果が互いに行き交い、フィードバックをかけながら循環的に進めていくモデルである。オブジェクトがカプセル化によって自律性を保っている為、分析や設計の区別なく同時に行われる。

このモデルでは、問題となる部分が見つけやすく仕様変更に強い事や、実装を意識せずに分析作業ができる、開発初期にユーザの協力を得やすいという長所がある。【5】

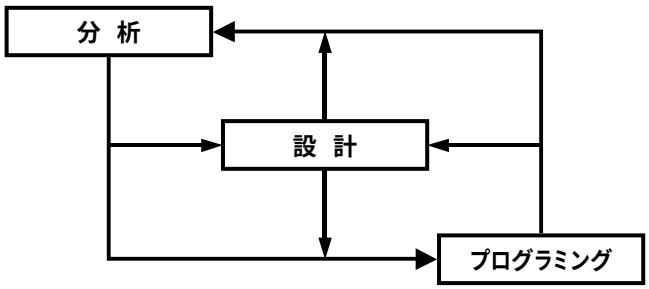


図 4-4　ラウンドトリップモデル

4.3 要求定義

システムに対する要求を調査・分析し、開発するシステムでどのような機能をどうやって実現させるか、どの程度の性能を求めるか等を明確にする。決定事項は要求仕様書としてまとめられる。要求定義の手順を図 4-5 に示す。

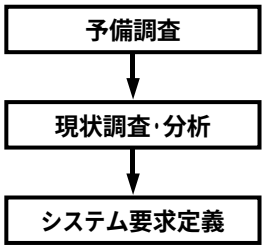


図 4-5　システム要求定義の手順

予備調査では、システム開発の可能性を検討する。いわば企画の段階で、検討の為の材料を技術面、費用的な面等多角的に収集し、開発するかどうかを決定する。システム開発の目的と目標をはっきりさせ、開発メンバーの意思を統一する事が重要である。【5】

新しいシステムを開発する際には、ユーザが現状にどういった不満を持っているのか、何を望んでいるのか、どのようなデータを扱い、どう加工しているのか知る必要がある。現状を調査・分析し、分析結果を基に新しくシステムの構想を練る。

要求事項を調査する方法には以下のようなものが挙げられる。企画者や応用システムの専門家からヒアリングする方法、調査会社依頼したり市場の調査文献からシステムへの需要を抜き出したりする方法、類似分野の既存システムと業界について調査する方法等である。【1】

資料は収集するだけでなく、的確に分析する事が求められる。代表的な分析方法を挙げると、集団で個々の自由な発想を自由に発言させる中から、独創的、創造的アイディアを引き出すブレーンストーミング（BS）法や、提起されたアイディアや発言を紙(カード)に書き出し、似通った内容ごとにグループ化し、タイトル付けをしてさらに上位グループへとまとめていく JK 法などである。【5】

4.4 外部設計

システム要求仕様で決められた内容を基に、利用者の立場から見たシステムとのインタフェース部分の設計、つまり、コンピュータの内部機能を切り離した、コンピュータの外側から見たシステム機能を設計する段階である。システムの持つべき機能を確定し、どのようなシステムにするかの概要を設計する。

外部設計の手順を図 4-6 に示し、各作業の説明を以下に述べる。

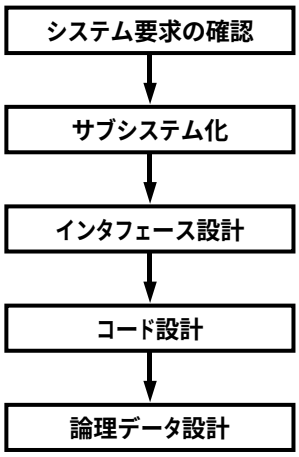


図 4-6　外部設計手順

通常、開発者がシステム利用者の希望を正確に把握するのは、簡単なことではない。要求仕様の確認を怠ると、利用者の希望していたものと異なるものができてしまい、システム開発は失敗となってしまう。こうならない為にも、ここで十分な要求確認をする必要がある。具体的には、システムの目的、利用者の要求確認を行う。

システムは、下位に複数のサブシステムを持つ階層構造を形作っている。要求確認後、要求を処理ごとにまとめ、システム開発の運用上の効率や効果を考えながら、システムをいくつかのサブシステムに分割していく。

利用者から見たシステム定義を行うのが、利用者とのインタフェースを中心としたシステム設計である。インタフェース設計は、直接人間に関係する部分で、画面設計等を行う。利用者から色々な要求が出される所である。開発者にとっては、作りやすいシステムにしがちな所であるが、利用者にとって使いやすいシステムを作ることを心掛けなければならい。

コード設計は、システムでの効率良い処理を行う為に、情報をコンピュータで扱うことのできる、２進符号へと変換する作業である。情報をコード化して、一つのコードに一つの情報を対応させておけば、似通った情報も区別しやすく、あいまいな表現がなくなり情報の簡素化や標準化が行える、管理がしやすくなる等の利点がある。コード化に際して、コード化の目的や使用範囲、特性などを総合的に判断して、採用する桁数、意味付け、コード表の作成等を行い、最適なコード体系（コードの種類）を選定する。複数のコードを組み合わせて作るコード体系もある。

一般的にコンピュータシステムは、入力データを加工して出力データを生成する。その為、必ずシステムで用いるデータの設計を行わなければならない。また、関連したデータが集まると、ファイルを形成する。データを加工する時には、内部で蓄積されている情報（ファイル）の参照・更新が行われている。データ処理にはファイルの存在も不可欠である。

ファイルの構成要素を図 4-7 に示す。

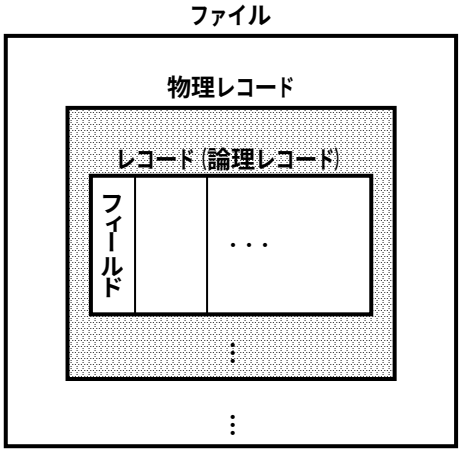


図 4-7　ファイル構成要素

フィールドは、論理的に意味のある情報の最小単位で、項目に該当する。一つのフィールドを細分化しサブフィールドを設ける場合もある。このフィールドが集まって 1 回の情報処理の対象（READ や WRITE など）となる単位、レコードを形成する。論理レコードは、狭義のレコードと同じく、1 回の情報処理の対象となる単位である。物理レコードは、記憶媒体上で入出力の対象となる単位となる。外部設計では、システム機器を意識せず、論理レコードを扱う。

論理データ設計とは、ディスク容量、データ件数、通信回線のデータ転送速度等の物理的な制限を考慮しない本来のデータの形を表すものである。論理データ設計では、システム内に格納するデータの構造と、主要ファイルのデータ項目を決定する作業を行う。システムで取り扱う全ての情報を抽出、分析して洗い出した後、情報項目の関連性を調べてグループ化し、さらに同一グループをファイルへとまとめる。ファイル化する項目を知ることで、システムで保存する必要のあるデータを見分けることができる。【5】【7】

4.5 内部設計

情報システムの内部、ハードウェアや基本ソフトウェアに依存するプログラムの、構成要素（コンポーネント）を設計する。外部設計を基に、システムを構築する上で必要となる機能をプログラムに分割し、プログラム間の処理の流れを明確にする。入出力装置、端末装置などの特性、システムの最適化を考慮しプログラムの作成ができるレベルまで、詳細な設計を行う工程である。

外部設計が利用者から見たシステム定義であるのに対し、内部設計では、コンピュータ側またはシステム側から見たシステムの定義となる。

内部設計は以下の手順で進められる。

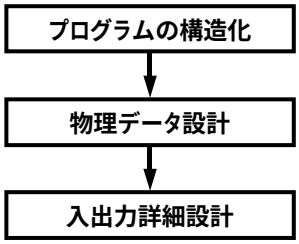


図 4-8　内部設計手順

外部設計で、システムから機能分割を行った複数のサブシステムを、正確かつ効率よく実現する為の処理方式を決定する。プログラム全体を一気に作るのではなく、小さな機能単位に分割して考えを絞り込みつつ、段階を追って設計を進める。処理のタイミングや共通ファイルを使用するプログラムを一括する、わかりやすさ等の観点から、分割・構造化すると他人から見ても理解しやすい。【7】

物理データ設計では、論理データ設計で決定したファイルのデータ項目から、データ特性を分析・検討し、ファイルの編成方法とレコードレイアウトを決める。

具体的には、現在のデータ量とこれからのデータ増加率、データの追加・変更・削除の比率等のデータ特性を分析し、明確にする。レコード内のデータ項目の表現形式（文字、数字など)、大きさ(桁数)、配置を決定する。処理形態、処理速度、オペレーティングシステムの機能を考慮し、ファイル編成を決定する作業を行う。

入出力詳細設計では、先に行ったインタフェース設計を基に、ハードウェアの制約などを考慮して詳細な仕様を作成していく。入力データの項目、配置など見やすさ、使いやすさを考慮して設計を行う。出力データも同様に、項目の選定や配置、出力桁数などを検討する。最近ではユーザインタフェースに GUI がよく採用され、いろいろな分野でシステム利用の容易化が図られている。【5】【7】

4.6 プログラム詳細設計

プログラム詳細設計の作業手順を図 4-9 に示し、説明する。

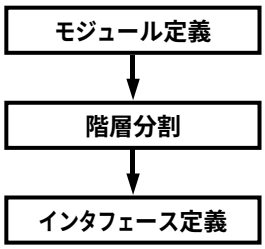


図 4-9　プログラム詳細設計手順

内部設計で作成したコンポーネント設計を基に、各プログラム内の構造化設計を行い、

プログラムをいくつかのモジュール（最小翻訳単位）にまで分割し、モジュール間のインタフェースを決定する。

プログラム構造化設計とは、以下の作業を行うことである。　大きなプログラムでは、全体を同時に考えることが難しくなる。その為、プログラムを小さく分割した方が思考範囲を狭くでき、プログラムの作成が容易になる。しかし、無意味に分割すると、かえって作業を複雑化させる。よって、一つのモジュール内では一つの機能を持たせる、を基本として適当な大きさに分割し、複数のモジュールで共通する機能を分散・重複させない様にする事がよい分割の仕方の指針となっている。

適切なモジュール分割を行うには、データの流れに沿って入力処理、変換処理、出力処理の機能に分割する源泉／変換／吸収分割（STS 分割）や、ジャクソン法の様に、プログラムを入力データ構造から出力データ構造へと変換するものと考えて、データ構造に着目した分割を行う技法などがある。

また、各モジュールが機能的にまとまりを持ち、それぞれのモジュールの関連が単純になる様に、適度な独立性を持たせた分割を行う。

階層構造化では、一つのモジュールから呼び出す下位のモジュール数を制限して、階層が深くなりすぎない構造を考える。

分割されたモジュールの間で、どのようなデータが受け渡されるのかを明確に定義する。【7】

4.7 テスト

設計段階で要求された機能、性能をシステムが満たしているかどうかをテストする。プログラムに誤りが含まれているものとして動作させることで、誤りを発見する。エラーを取り除き、信頼性を向上させる為に重要な工程である。テスト手順を下図に示す。

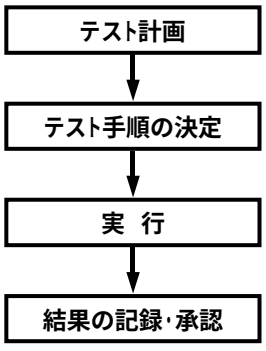


図 4-10　テスト手順

テストにはプログラム詳細設計で行った、個々のモジュールの動作を検査する単体（ユニット）テストや、単体テストを終えた、モジュール同士を結合して動作確認するコンポ

ーネントテスト、更にテスト済みのプログラムを組み合わせて、プログラム間のインタフェースが適切かどうかテストする、システムテスト等がある。

それぞれのテストは図の様に、テスト計画の立案、テスト手順の決定、テストの実行、結果の記録・承認、と共通する手順で行われる。【5】【7】

5 画像処理

今回製作した回路の最も重要な機能である画像処理、特に今回の回路設計で必要なデジタル画像処理について述べる。

デジタル画像処理は 1960 年代の後半に始まり、主に人工衛星画像に対して画質の改善、画像の復元・補正といった処理を施すことを目的としていた。その他の対象画像としては、医用画像で顕微鏡画像、胸部 X 線写真に対して対象物の検出、特徴箇所の抽出・分類といった処理が試みられた。

1970 年代にはコンピュータ断層撮影法（CT：computed tomography）が開発され、医学と共に画像処理技術の分野にも影響を与えた。同時代には、人工衛星ランドサットによる地球の観測画像が撮影されるようになり、リモートセンシング画像処理といった技術分野を形成させた。

1980 年代以降も、コンピュータの能力の向上に伴って、より複雑な画像処理がより早くできるようになった。技術的にも様々な 3 次元グラフィックスの処理で新分野を生み出し、医用分野では医用画像全体のデジタル化、データベース化が促進され、産業分野では目視検査に変わる機器やロボットと画像処理の組み合わせが実用化される事となる。

1980 年代末から 1990 年代にかけては、これまでの技術と人間の動きを読み取るヒューマンインタフェースにおける画像処理技術を結合し、仮想現実（バーチャルリアリティ：VR）への応用が盛んになる。マルチメディア、IT といった言葉をキーワードとして、画像の持つ役割、利用方法と共に、今後の画像処理のあり方が問われている。【10】

5.1 画像のデータ表現

画像処理を行うにあたり、論理的に画像データを扱う為、画像のデータ表現を理解する必要がある。

デジタル画像では、大きく分けて二つのデータ表現がある。一つは画素の並びで面を構成するラスタ型データ表現、もう一つは画像を構成する点、線、面、文字等の構成要素の幾何学的位置や長さといった属性情報を記録したベクトル型データ表現である。

ラスタ型データ表現では、階調の多い写真などの濃淡画像に適した表現である。色の濃淡を表す最小単位である画素の値を 2 次元の配列で表現している。白黒または単色の濃淡だけの画素を 2 次元配列した画像をモノクローム画像という。カラー画像は、R（赤）、G（緑）、B（青）の光の三原色で全ての色が表現できる為、RGB それぞれの濃淡を表す 3 枚

のモノクローム画像を合成することで表現される。また、RGB 以外にも、可視光以外の赤外領域なども含め、複数の波長域で撮影された画像が考えられる。こうした複数のモノクローム画像で構成される画像をマルチチャンネル（多重）画像という。

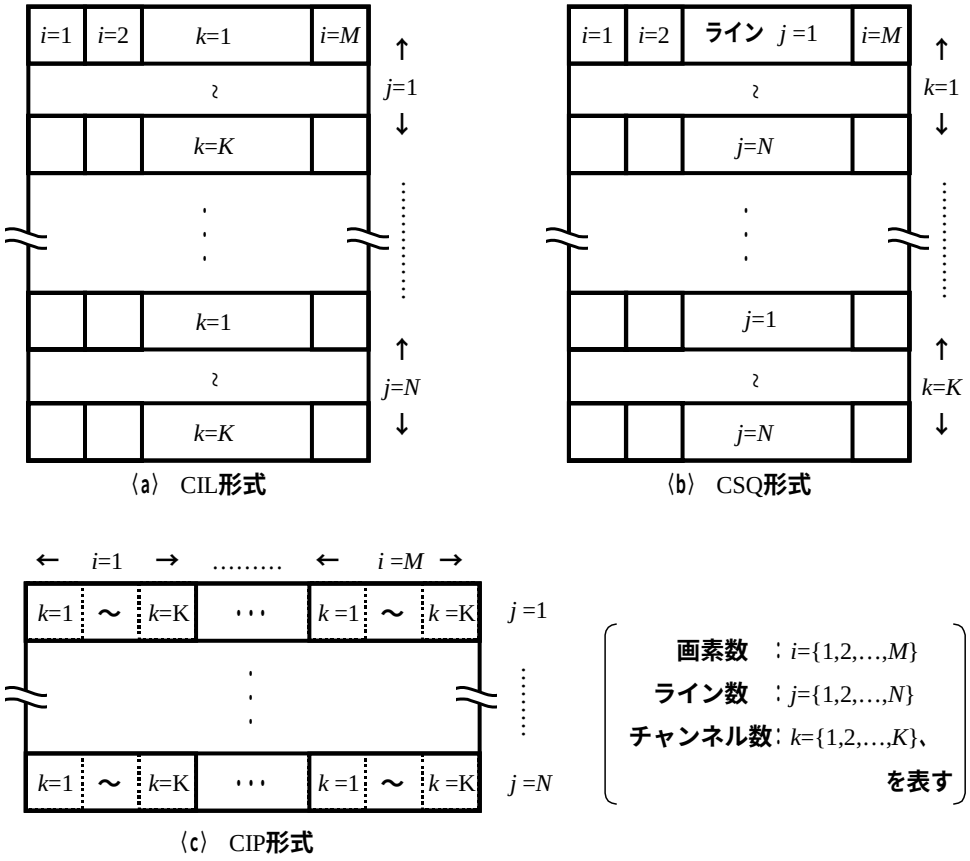


図 5-1　ラスタ型データ表現

マルチチャンネル画像のデータ表現を図 5-1 に示した。チャンネルの並ぶ順序の違いで、〈a〉CIL(channel interleaved by line)、〈b〉CSQ(channel sequential)、〈c〉CIP（channel interleaved by pixel）といった三種類のデータ表現ができる。〈a〉～〈c〉の各図共、M×N の画素数で表されるモノクローム画像を、K チャンネル分集めた「M 画素×N ライン×K チャンネル」の表現である。

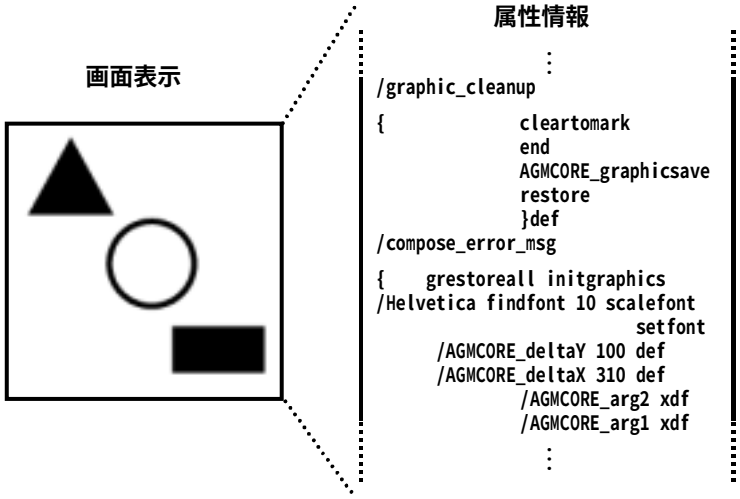


図 5-2　ベクトル型データ表現

ベクトル型のデータ表現例を図 5-2 に示す。ベクトル型データ表現は、二値または比較的階調の少ない画像、線図形等で幾何学的精度が要求される画像、例えば地図や各種図面に適した表現形式である。画像の構成要素をベクトルの集合として扱う事からベクトル型と呼ばれる。

例えば、線分は基準点からの始点と終点の位置情報で表される。曲線は多くの線分で近似され、面は境界を多角形で近似する事で、同様に線分の集合としてベクトル扱いができる。

ラスタ型では座標が、画素番号とライン番号の整数で表されるが、ベクトル型では座標が実数で表される為、位置情報を高精度で表現する事が可能である。拡大・縮小、回転等の幾何学的変換を高い精度で容易に行える等の利点がある。【10】

5.2 画像ファイルフォーマット

画像データはファイルとして、コンピュータやその周辺機器の間でやり取りされる。画像データをファイルに格納する方式を画像ファイルフォーマットという。画像フォーマットでは、ファイルへの画像データの書き込み方法、ファイルからの読み込み方法が定義されている。

画像ファイルフォーマットは、画像データを扱うコンピュータやその OS（オペレーティングシステム）等のハードウェア環境、画像処理の為のアプリケーションソフトウェア、画像ファイルの使用目的等に応じて、様々な方式が提案されている。

5.2.1. BMP（Microsoft device independent bitmap）フォーマット

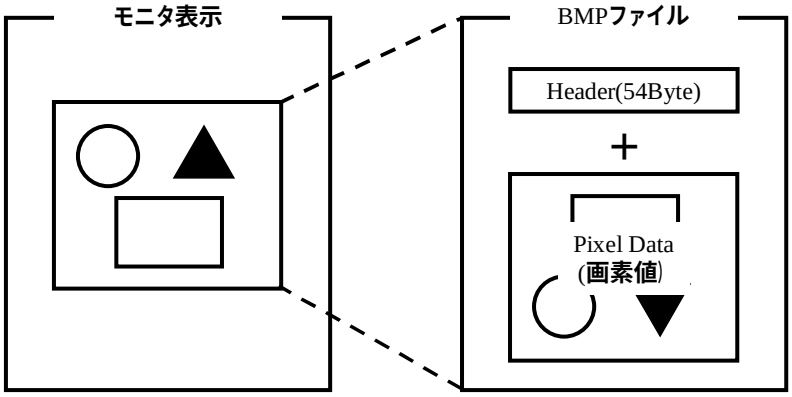
C 言語記述の為に、いくつかの汎用画像フォーマットの中から、BMP フォーマット形式の画像データを使用する事にした。

BMP フォーマットは、米国の Microsoft 社が開発した OS の Windows 上で、標準画像

フォーマットとされている。このフォーマットで取り扱える画像は、チャンネル数が１または３で、一つのチャンネルで 8 ビットまでのデータを持つ画像である。1 画素 24 ビット（1 チャンネル 8 ビットで 3 チャンネル）の画像以外になると、256 色同時表示以下のカラーマップを使用する。

画素の情報以外に、画像の大きさや解像度、圧縮形式等の情報を持つ 54Byte のヘッダ部分を持ち、図 6-2 に示す様に画像データを左下から右上に向かって表示した時、モニタ上で正常に表示される様、上下が反転した状態で画素値を格納しているのが特徴である。画像の横方向のデータは、4byte の境界に揃える様になっており、横１ラインのデータ長は 4byte で割り切れなくてはならない。

BMP フォーマットは対応する OS によって数種類あり、ヘッダ内の情報サイズや内容が変わってくる。【10】



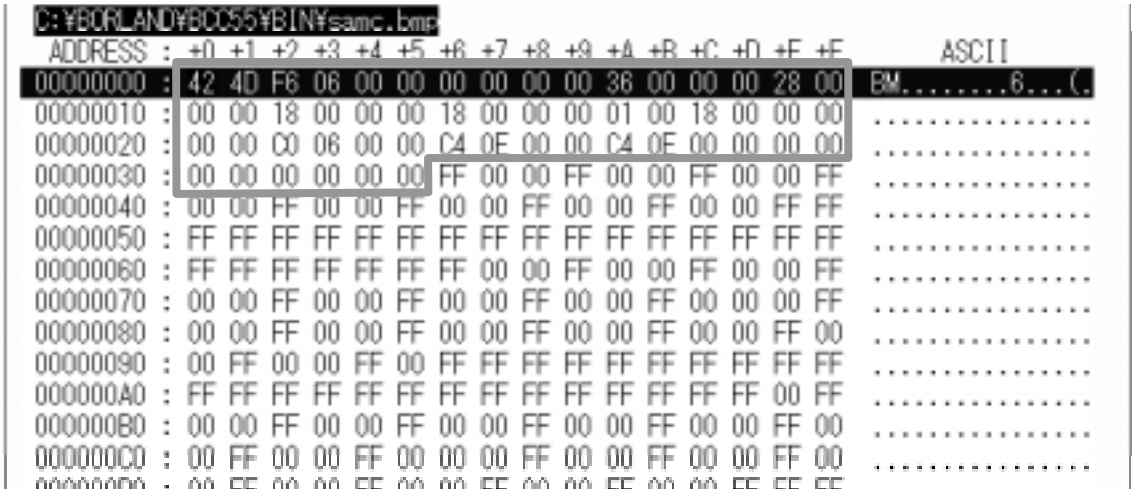


図 5-3　ビットマップファイル構造とバイナリデータ

（太枠内、アドレス 0_{16} 番から 35_{16} 番が 54Byte 分のヘッダを指す。36_{16} 番以降が画素データ。）

5.3 デジタル画像処理モデル

画像処理の多くは、与えられた画像(入力画像)から、別の画像(出力画像)を作り出す役割を持つ。画像処理の目的は大別すると以下の様にまとめられる。

- 画像圧縮：画像情報は、音響情報などの一次元情報に比べて、膨大な情報量を持ち、画像を使った通信や、ファイル操作、データ転送等の際にハードウェアへ負担をかける。この情報量を減らし、データを軽くするのが圧縮技術である。
- 画質改善・変換：画像は撮影の条件、撮影機材の性能によって品質が変化し、常に満足できる保証がされているわけではない。明るすぎる、暗すぎる等、雑音や歪みが混入して画質が劣化した画像を、見やすい画像、満足できる画像に改善する技術。
- 画像解析または画像認識：与えられた画像から任意の形状を抽出したり、特定の性質部分のみを加工したりする処理を行う。

デジタル画像は、画面を構成する最小単位の画素とその中の濃度値の組み合わせで表される、と考えられる。数式的には、M 行×N 列の画素で表されるデジタル画像を、

$$F = \{ f_{ij} \} \quad \{ i = 1,2,\Lambda\ ,M \}, \{ j = 1,2,\Lambda\ ,N \} \tag{5.1}$$

と表せる。この時 f_{ij} は第 i 行 j 列の画素の濃度値を表す。【10】

5.3.1. 1 入力 1 出力の画像処理

今回の回路設計で、基本とする 1 入力 1 出力の画像処理のモデルを図 5-4 に示し、特徴を述べる。

一つの画像 $F = \{ f_{ij} \}$ が与えられて、それに対する出力画像 $G = \{ g_{ij} \}$ を求めるとすると、出力画像 G を求める方法には以下の様なものがある。【10】

- g_{ij} の値を同じ位置にある入力画像の濃度値 f_{ij} だけを使って計算する、階調処理、しきい値処理等。
- g_{ij} を同じ位置(*i,j*)とその周辺(2K+1)行(2L+1)列分の入力画像の画素値、

$$f_{pq}, \{ i - K \leqq p \leqq i + K, j - L \leqq q \leqq j + L \} \tag{5.2}$$

  を使って計算する、局所処理、マスク処理、フィルタリング処理等。
- g_{ij} ひとつを計算するにも入力画像 F 全体を使う、2 次元離散フーリエ変換等。【10】

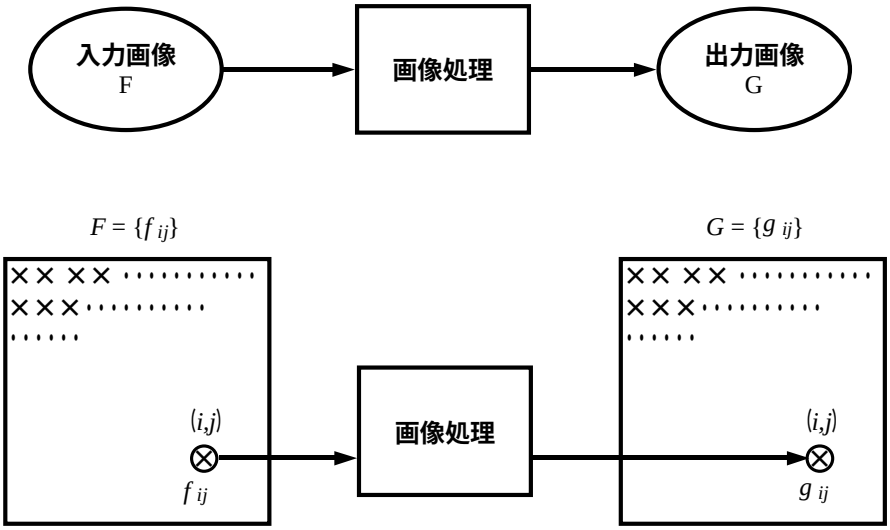


図 5-4　1 入力 1 出力の画像処理モデル

5.4 アルゴリズム

先述した画像処理モデルの様に、デジタル画像処理で行われる作業は、入力画像を基に出力画像の画素値を算出する事である。この時行われる計算の仕方を、画像処理のアルゴリズムという。

ここでは、1 入力 1 出力の画像処理を基本として、調査した代表的なアルゴリズムとその特徴を述べ、使用するアルゴリズムの検討を行う。

5.4.1. 濃淡情報の変換

コントラスト変換について述べる。コントラスト変換では、画像の濃淡を一定の規則に従って変換する処理を行うものだが、コントラストだけでなく階調性も変化させる事から階調変換ともいう。明るすぎる、暗すぎるといった画像や、微妙な濃淡変化をさせたい時に使用される。変換の方法には、関数を用いるものとヒストグラムを用いるものがある。

関数を用いた変換では、画像の目的に応じて入力画像の濃淡レベル x を、適当な関数 f で出力画像の濃淡レベル $y = f(x)$ へと変換する。

使用する関数によって得られる変化の例を示すと、図 5-5 の線形変換関数〈a〉では、入力画像における $x_{\min}$ から x_{max} までの一部の濃淡レベル領域が、関数の使用により出力画像では、 $y_{\min}$ から $y_{\max}$ の濃淡レベル全域まで広がった領域を持つ様になる事を示している。この時の出力画像の濃淡レベル y は次式、

$$y = \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}}(x - x_{\min}) + y_{\min} \tag{5.3}$$

によって表される。

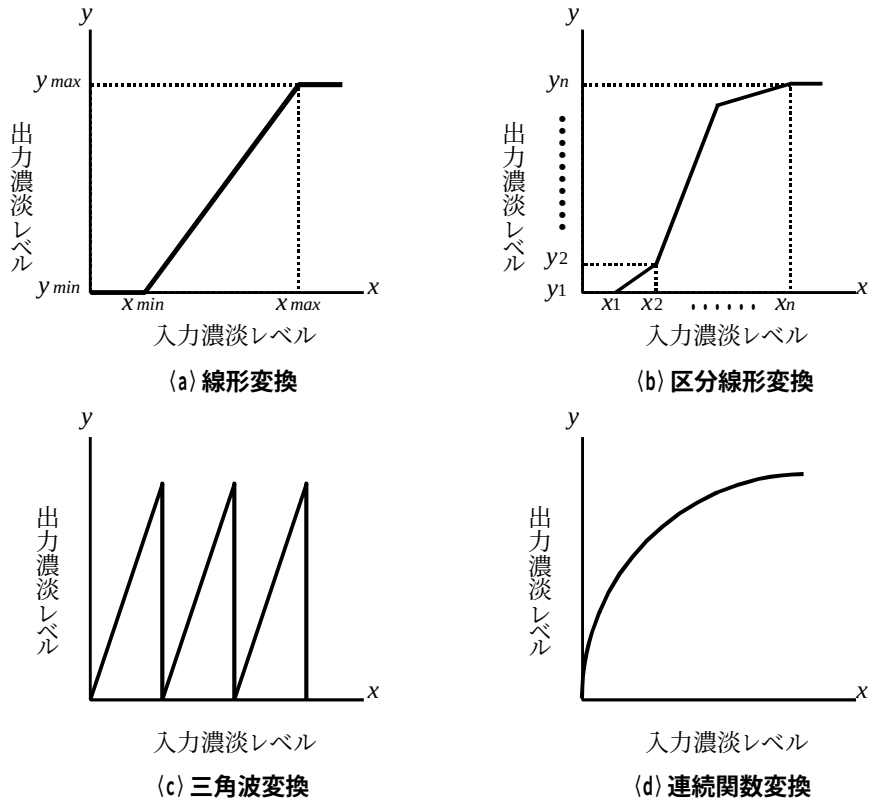


図 5-5　コントラスト変換関数の例

同様に〈b〉の区分線形変換では、入力側の画像の濃淡レベル $x_1, x_2, \Lambda\ x_n$ に適当な出力側濃淡レベル $y_1, y_2, \Lambda\ y_n$ を対応させ、それぞれの区間で線形変換を行っている。区間 $[x_i, x_{i+1}]$ における線形変換は、

$$y = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + y_i \tag{5.4}$$

で表される。

コントラスト変換では、〈c〉の様な三角波を繰り返す周期関数を用いる変換も行われる。これは一定の区間ごとに出力濃淡レベルが変換される為、空間的に連続的な濃淡分布を持つ画像に、出力後は縞状に繰り返される濃淡のパターンが現れる画像へと変換される。一種のコントラスト強調効果が見られる変換である。

この他にも、〈d〉や２次関数、対数関数、指数関数の様な連続関数を用いた変換も行われる。連続関数を使うと、区間線形変換や周期的な変換の様に段階的に濃淡レベルの変換率が変化するのではなく、滑らかに濃淡レベルが変化でき、より微妙な濃淡変化の効果が得られる。【10】

5.4.2. 空間的情報の変換

画像上において、濃淡の空間的な分布が表す情報を画像の空間的情報という。これに注目した変換の例としては、濃淡の過剰な変化を低減する平滑化、濃淡の変化を強調する鮮鋭化、濃淡が急激に変化する濃淡エッジや線の抽出等がある。

5.3.1 で先述した 1 入力 1 出力の画像処理モデルで、入力側の画素値 f_{ij} と周辺(2K+1)行(2L+1)列分の画素値を使ったフィルタリング処理に注目した例を示す。

移動平均フィルタリング処理では、注目する入力画像の画素値 f_{ij} を中心として、水平方向±M分、垂直方向±N分の局所領域の平均値を、出力画像の画素値 g_{ij} として出力するフィルタを、入力画像の対象画素全てに適用させるもので、

$$g(i,j)=\frac{1}{(2M+1)(2N+1)}\sum_{k=-M}^{M}\sum_{l=-N}^{N}f(i+k,j+l) \tag{5.5}$$

以下に示す移動平均フィルタリングの例では、M＝1、N＝1 とした、局所矩形領域のサイズ 3×3 画素、の平均値を求める。

平均するのは 9 画素分であるから、例の場合では出力画素値 g_{ij} は、

$$\begin{aligned}g(i,j)=\{&f(i-1,j\text{-}1)+f(i,j-1)+f(i+1,j-1)+f(i-1,j)+f(i,j)+\\&f(i+1,j)+f(i-1,j+1)+f(i,j+1)+f(i+1,j+1)\}/9\end{aligned} \tag{5.6}$$

で求められる。上式において入力画像にかかる重み係数は、図 5-6 の 3×3 フィルタが示すとおり、一様に各画素とも 1/9 ずつの配置となる。

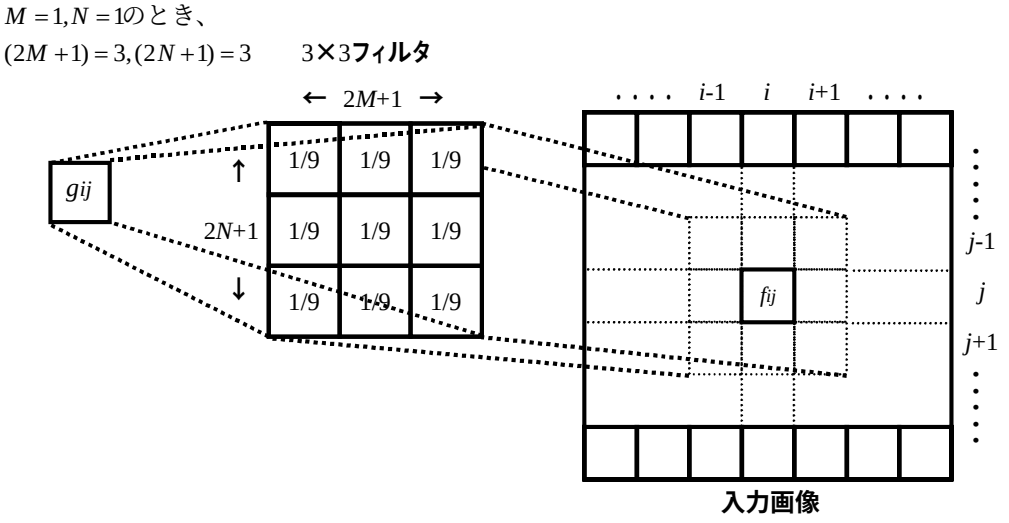


図 5-6　移動平均フィルタリング

これをフィルタ h として重み係数の行列式で表すと、

$$h = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \tag{5.7}$$

と表され、式(5.6)を h と f との積和の形、

$$g(i,j) = \sum_{k=-M}^{M} \sum_{l=-N}^{N} f(i+k,j+l)\, h(k,l) \tag{5.8}$$

と置き替えられる。

また、以上は M=1、N=1 の場合であり、h の要素 $h(k,l)$ は次を参照されるものとなる。

$$h = \begin{bmatrix} h(-1,-1) & h(-1,0) & h(-1,1) \\ h(0,-1) & h(0,0) & h(0,1) \\ h(1,-1) & h(1,0) & h(1,1) \end{bmatrix} \tag{5.9}$$

式(5.8)を見る様に、h は全ての入力画素に対して局所平均値を出力するフィルタと見なす事ができる為、h を移動平均フィルタと呼ぶ。

移動平均フィルタ処理では、平滑化の効果が得られる。平滑化は画像の雑音を低減する、画像をぼやかすといった効果を得たい場合などに行われる。

(2K+1)×(2L+1)のフィルタを用いたその他の平滑化の例では、対象画素への重みを変化させ、中心画素への周辺画素の影響を少なくした加重平均フィルタリングや、微小な濃淡変動のみを選んで平滑化しようとする、可変加重平均フィルタリングがある。【10】

5.4.3. 幾何学的情報の変換

画像の持つ幾何学的な情報のうち、画像の回転、変形といった 2 次元の幾何変換について述べる。変換前の画像の座標を (x,y)、変換後の画像の座標を (u,v) とし、x 軸、u 軸の正方向を水平右方向、y 軸、v 軸の正方向を鉛直下方向とした座標系を用いる。

以下に平行移動、拡大縮小、回転といった例を示す。

・ 平行移動：x 方向への移動量を a、y 方向への移動量を b とすると、変換後の座標は次式で表される。

$$u = x + a, v = y + b$$
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \tag{5.10}$$

・ 拡大・縮小：x 方向の拡大・縮小率を a、y 方向の拡大・縮小率を b とすると、

$$u = ax, y = by$$
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.11}$$

と表せる。このとき $a>1$ では x 方向の拡大、$0<a<1$ では x 方向の縮小、$b>1$ では y 方向の拡大、$0<b<1$ では y 方向の縮小となる。

・ 回転：座標の原点を中心とする回転は、回転角 θ（反時計回り）として、次式で表される。

$$u = x\cos\theta + y\sin\theta$$
$$v = -x\sin\theta + y\cos\theta \tag{5.12}$$
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

２次元の幾何変換にはこれらを基本に、いくつかの変換を組み合わせて一つの変換方法としたものも存在する。【10】

6 画像フィルタ回路の設計

これまでの学習を基に、HDL を使用したデジタル回路設計の基本的な技術を身に付ける為、実際に VHDL を使用して、画像フィルタ回路の設計に取り掛かる。

6.1 画像フィルタ回路とは

製作する画像フィルタ回路の大まかなブロック図を以下に示す。画像フィルタ回路とは、図に示すとおり、回路へ画像データを入力すると、そのデータに何らかの処理を施して、再び画像として出力するという回路をイメージしたものである。下図では出力例として、入力したサンプル画像に対して、フィルタ回路で左右反転（鏡像を生成）したもの、上下反転したもの、色調を変更したものを出力している。

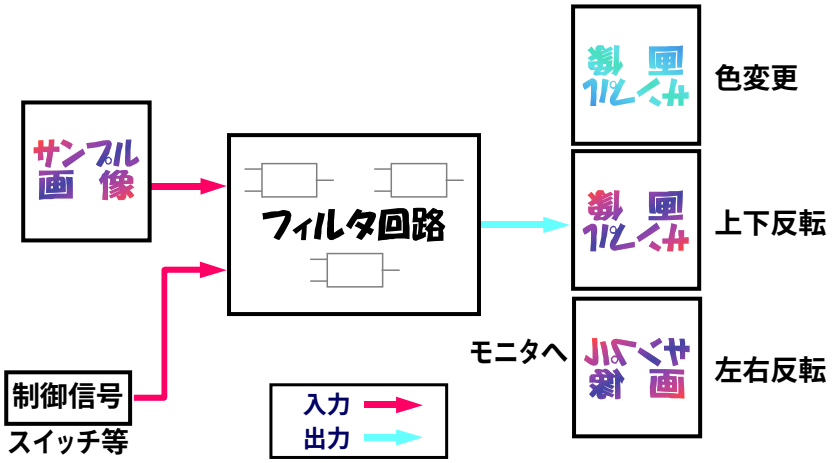


図 6-1　画像フィルタ回路のイメージ

回路製作にあたって、画像フィルタ回路を製作しようとしたきっかけは、回路の動作を、はっきりと目で見て確認できるようなものにしたい、と考えた事で、回路の出力部分にモニタを接続して結果を表示させるという仕様へと行き着いた。

次に、出力部のモニタに何を表示するかであるが、できればグラフや信号波形の様に、ただ回路動作の確認を行うだけでなく、見た目にも楽しめる様なものにしたいと思い、画像を出力しようと考えた。結果、自然と回路で画像データを生成する仕様となった。

その後、画像処理に興味があったこともあり、回路に画像データを入力し、入力されたデータに対して何らかの手を加えた後、再び画像として出力するという画像フィルタ回路の大まかな仕様へと行き着いた。

6.2 回路設計

画像フィルタ回路の具体的な設計に入る。先に第 3 章で述べた VHDL 設計と第 4 章のシステム設計の手順から、今回の回路設計のフローを作成し、それぞれの工程でさらに詳しく設計を進める事にした。

今回設計した回路は、最終的に以下の図のような構成となった。この回路構成へと到った設計手順を述べる。

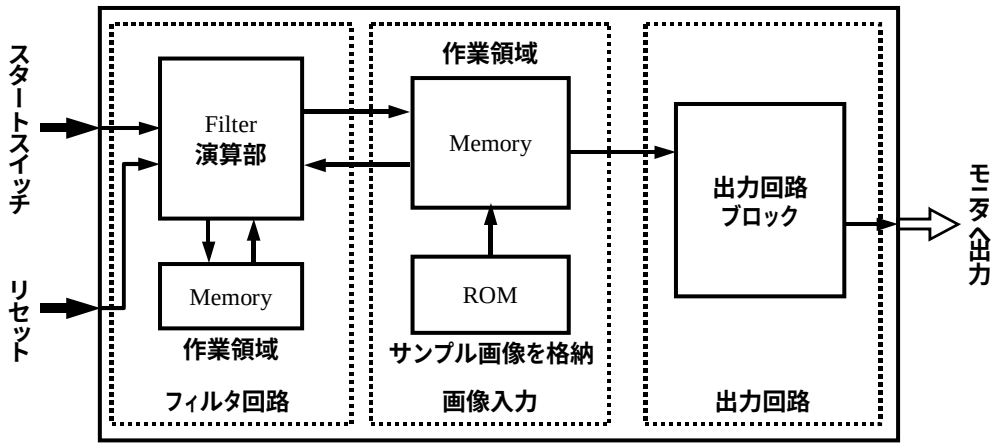


図 6-2　ハードウェア回路構成

この回路設計の当初の最終目標は、正しい回路動作を確認するまでだが、シミュレーションの検証結果、コンピュータ上の回路設計が終了すれば、FPGA へ書き出して回路を動かす事もできる。

図 6-3 に回路設計フローを示す。設計の前半では、オブジェクト指向設計で採用される、ラウンドトリップモデルの考え方を取り入れた。第 4 章 2.3 で述べた様にラウンドトリップモデルの特徴は、問題となる部分が見つけやすく、仕様変更に強い事である。こういった設計フローを考えたのは、無謀な要求定義や、開発環境の限界を超える様な仕様をした場合に、臨機応変に対応できる様にしようとした結果である。これにより、工程を順次追い

かける事なく、できる所から手をつけられる、各工程で問題が発生して変更が生じてもお互いにフィードバックをかけながら進められる、といった点を利用する事ができると考えた。

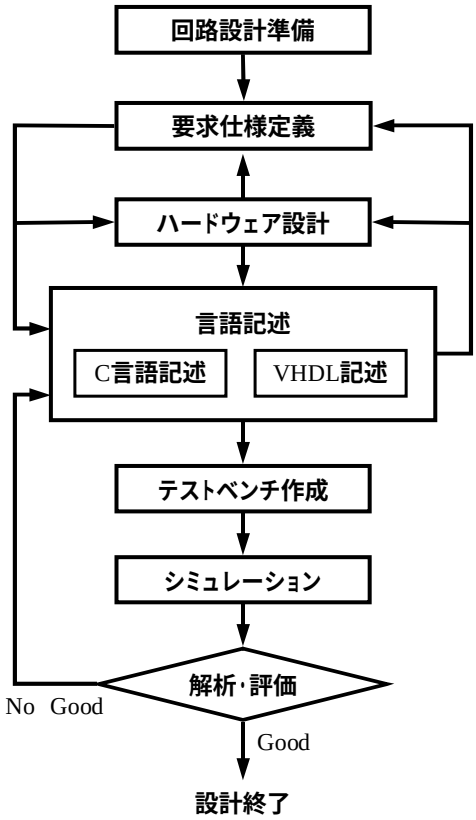


図 6-3　回路設計工程フロー

6.3 回路設計準備

画像フィルタ回路を設計する為の環境を整える。

VHDL を使用した回路設計では、実際の回路素子を基盤に取り付ける、といった工程を必要としない為、コンピュータ上の作業で設計から言語記述、シミュレーション、解析・評価まで行える。その代わり、それぞれの作業にあったアプリケーションソフトウェアを用意する必要がある。

今回は、以下のようなソフトウェアを用意した。

- C 言語コンパイラ：Borland C++ Compiler 5.5
- デザインツール：Xilinx WebPACK Ver4.1 Project Navigator
  Altera MAX+PLUS II
- シミュレータ：Xilinx WebPACK 4.1 ModelSim XE Starter

これらのソフトウェアを利用して、A205 室のワークステーション端末、及び自分で用意し

たノートパソコンを使用して、設計を進める事にした。

6.4 要求仕様定義

仕様設計では、製作する回路に何が求められているのか、要求を分析して、システム全体に対して仕様を検討する。要求仕様をしっかりと分析して十分に仕様を検討する事は、LSI 設計において、もっとも大切な局面であるとされている。

通常は、仕様設計でシステム全体を把握した後、LSI にどこまでさせるか、何をさせるか、また LSI で要求される仕様の、どこまでを受け持つのかを決定する外部仕様へと移る。これを画像フィルタ回路に何をさせるか、何を受け持たせるかに適用させる。

また、この段階で画像処理について下調べを行い、フィルタ回路に適応させるアルゴリズムを検討する。

ここで採用した要求の分析方法は、JK 法（第 4 章 3 参照）を参考に、自分に対してヒアリングを行い、要求を抜き出すといったものである。簡単に図式化したものを以下に示す。

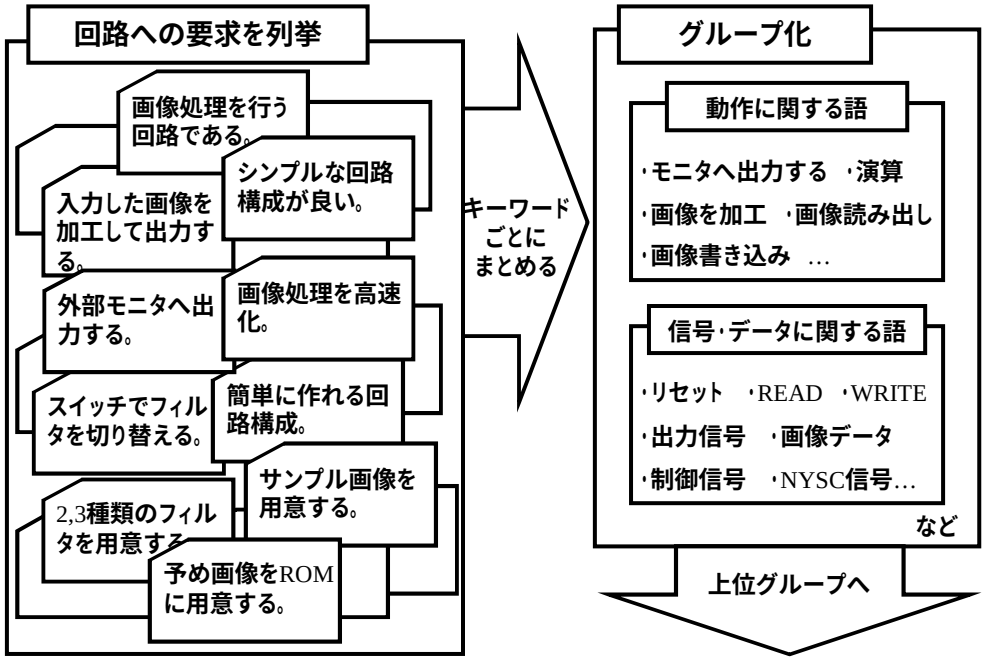


図 6-4　要求分析方法の例

まず、自分がこの回路に望む事、回路のイメージといったものを、次々と紙（ノート）に書き出し、機能や動作、性能、データに関する単語や、それらの単語を含む文章の中から、大切だと思われるものをキーワード、キーセンテンスとして抜き出す。その中から、関連性のあるものをまとめてグループ化する。そのグループの中から再びキーワード、キーセンテンスを抜き出しグループ化する、といった作業を上位レベルへと繰り返し、回路機能や動作の流れ、データの流れを洗い出す作業を行った。要求をまとめ出したものを表 1

に示す。

～キーワード・キーセンテンス～ 入力、出力、画像の読み込み（READ)、画像の書き出し（WRITE)、メモリ、フィルタ演算、制御、リセット、スタート、スイッチ、…			
	動　作	信号・データ	相互動作・データ関係
下位レベル	入力、出力、READ、WRITE、リセット、スタート、演算、…	画像データ、入力、出力、READ、WRITE、リセット、スタート、…	画像データの読み・書き、メモリの読み・書き、モニタ出力、回路の初期化・始動、…
上位レベル	画像出力　→	出力信号、同期信号	
	フィルタ演算　→	画像データ、シフト、桁上げ	
	画像読み・書き　→	アドレス指定、READ、WRITE	
	回路の始動・初期化　→	START・RESET	

表 1　要求仕様の分析

6.5 ハードウェア設計

要求仕様定義で抜き出した動作、データの流れを基にハードウェアの構成を考える。前述の表１から最上位の機能、動作、データを抜き出し、回路を最も外側から見た 2 入力 1 出力型の最上位ブロックを図 6-4 の様に考えた。最上位の入出力は、要求仕様の検討結果から、回路動作の結果を確認できる様にする出力信号、回路を動作させるスタート信号と、内部状態の初期化を目的としたリセット信号を入力として、この様な回路構成となったものである。

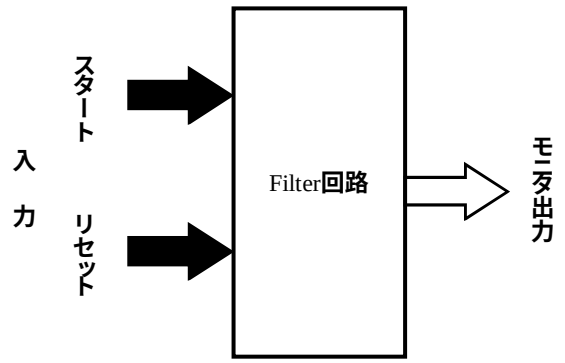


図 6-5　最上位ブロック

次に、内部構成を考える。要求仕様定義の段階で出された要求というのは、自分の希望

が強く反映されたもので、システム的、またはハードウェア的に正しい回路として製作できるかどうか、という事は後回しにされていたが、ここへきて、一人で製作する事を考え、まず動作させる事を第一目標に、ハードウェア構成はなるべく単純化しようと考えた。そこで、画像データ部分をあらかじめメモリ内に格納しておく案が浮び上がった。また C 言語、VHDL で記述する事を考えて、簡単化の為、画像サイズも固定した方がやりやすいと考えた。

表 1 の要求仕様分析で上位レベルの動作を一つのモジュールとしてまとめていく。

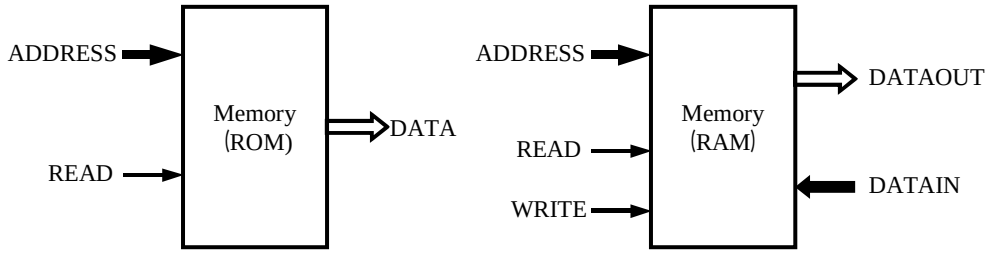


図 6-6　メモリブロック

画像を格納するメモリブロックは、読み出し専用の ROM でも、読み書き自由な RAM でも実現できる。サンプル画像は用意した物しか使わない（外部から画像の入力を行わない）ならば、読み出しだけを行う ROM で十分構成できる。メモリ外部から入力がある場合や作業領域の為に、RAM の構成も考えておく。

次にフィルタ演算部をモジュール化する。一般的な ALU（Arithmetic and Logical Unit；算術論理演算装置）のハードウェア構成は以下の図の様になる。演算ブロックで扱う信号は、被演算数の DATA1、2 と桁上げ、OP コード（演算コード）を入力、演算結果を DATAOUT として出力する。

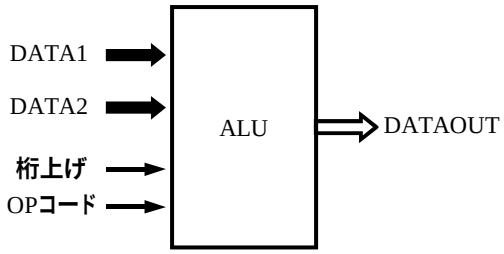


図 6-7　演算部ブロック

次に、画像の出力部をモジュール化する。出力先の媒体によって、出力信号の形が変化するが、一般的な TV モニタなどに映し出す場合を考えると、図 6-7 のような構成を考えることができる。

TV モニタへの出力は、表示方式によっていくつかのパターンで構成できる。下図では、

アナログ信号の RGB をそのまま表示できる方式と RGB から生成したコンポジット信号を出力する方式の二通りが表されている。

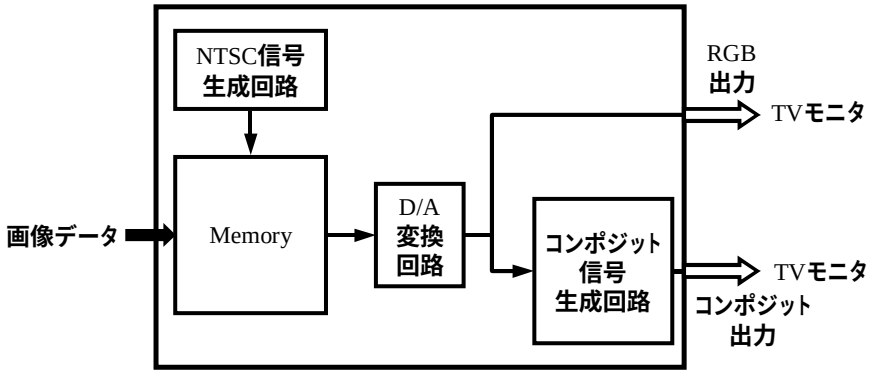


図 6-8　出力回路ブロック

最終的にはこれらのモジュールを一つにして、図 6-4 の最上位ブロックの仕様を満たす様に構成した。結果、最終的に下図に示したハードウェア回路構成となった。

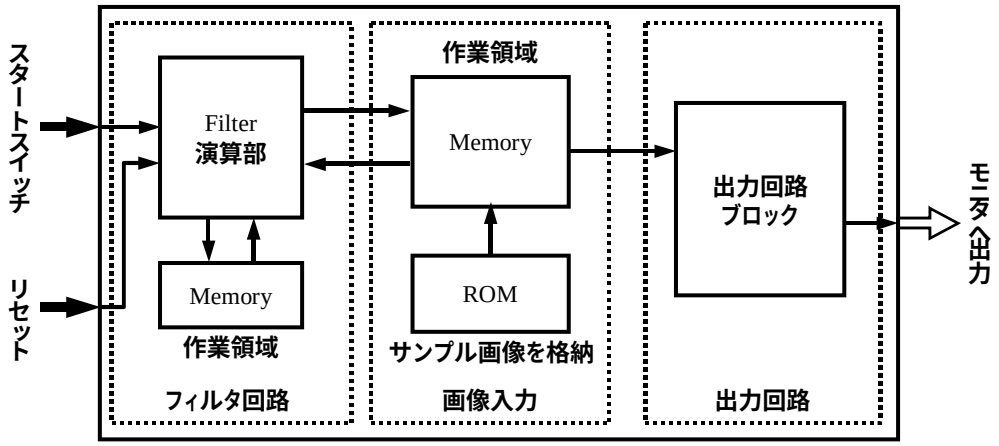


図 6-9　ハードウェア回路構成

6.6 C 言語記述

小規模の回路設計では、VHDL でいきなり記述することも可能だが、LSI 設計の様に大規模な回路を設計する場合の実際の手法に沿って、VHDL で記述する前にアルゴリズムの確認や回路動作の検証の為に、C 言語で回路動作を記述する。

画像フィルタ回路なので、取り扱うデータはもちろん画像データである。コンピュータ上の画像の表現を把握する為に、画像ファイルのフォーマット、データ構造について調査し、第 5 章ではその概要を述べた。画像処理の簡単化の為、1 入力 1 出力の画像処理モデルを採用し、扱う画像データは BMP（ビットマップ）形式のファイルとする。画像の入力か

ら加工して出力するまでを一連の操作として、以下のような流れのソース記述を行う。

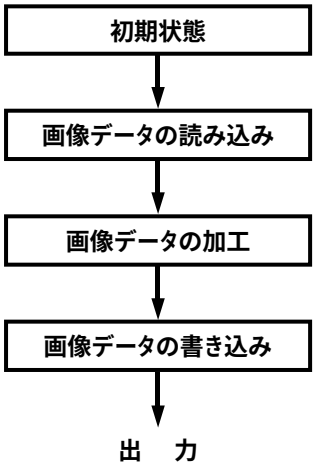


図 6-10　画像データ操作の流れ

使用するサンプル画像を、BMP（ビットマップ）ファイル形式の 24bit カラー、160×160 のサイズに固定した。光の三原色 Red、Green、Blue で全ての色を再現するカラー画像で、RGB 各 8bit の情報を持ち、3 チャンネルが横に並ぶ CIP 形式のラスタ型データ表現（第 5 章 5.1、図 5.1 参照）の画像であるので、画素値を格納する配列の *x* 方向（横方向）の大きさ XSIZE が、

$$\mathrm{XSIZE}=[3\mathrm{Channel}]\times[x\text{方向の画像サイズ}]=3\times160=480 \tag{6.1}$$

となり、横方向には 3 倍の領域が必要である。これを図 6-11 に示す。

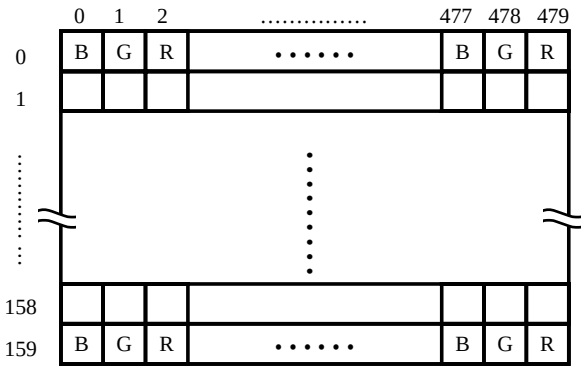


図 6-11　入力画像のデータ表現

これを前処理で定義しておく。以下に main 関数までの記述を記載する。

```
#define XSIZE 480          /*画像データを格納するｘ方向領域を固定*/
#define YSIZE 160          /*画像データを格納するｙ方向領域を固定*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*関数宣言*/
void NAME(char *FILENAME);
void IMAGE_READ (unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME);
void IMAGE_WRITE(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME);
```

ここでは、図 6-10 の画像データ操作の流れに基づいて、それぞれの処理を関数化して用いる形をとった。関数の役割は以下のとおりである。それぞれのソースは付録に記し、注意点だけをここに説明する。

- void NAME：読み込むファイルの名前を確認する。
- void IMAGE_READ：画像データを読み込む。
- void IMAGE_WRITE：画像データを書き込む。

IMAGE_READ を詳細に記述する為のフローチャートを起こし、記述する。

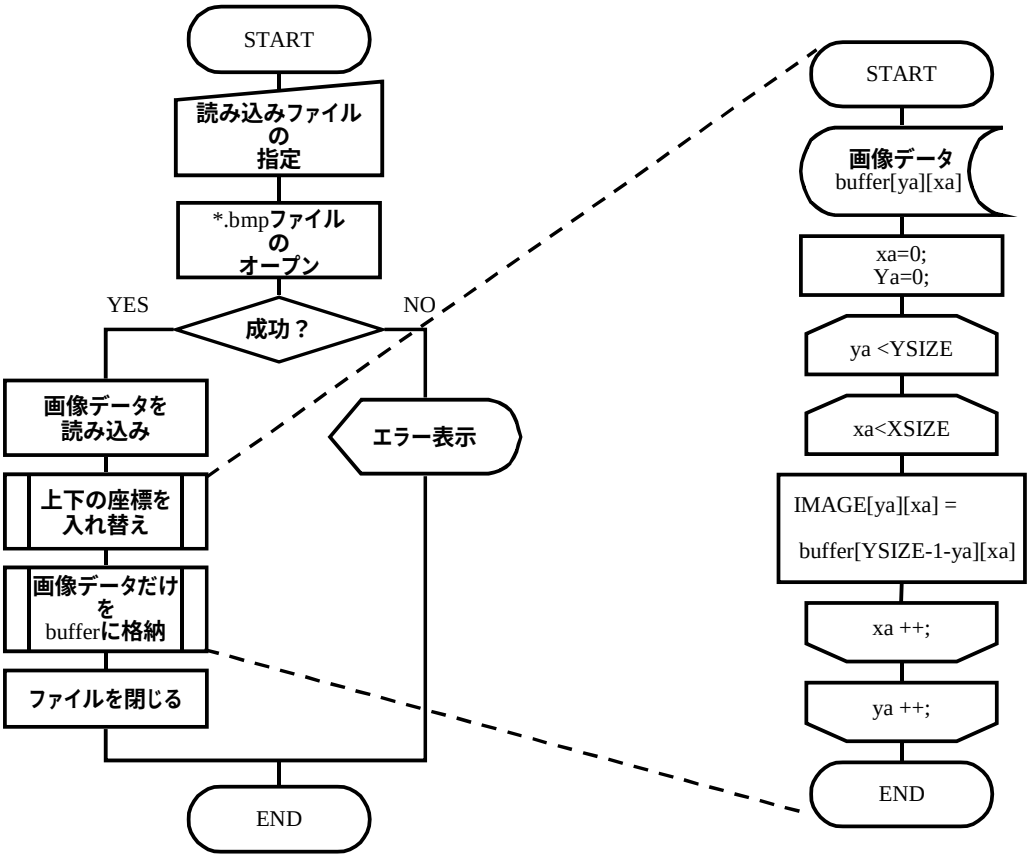


図 6-12　画像ファイル読み出しフロー

ここで必要になるのは、BMP ファイル内の画素値データのみであるので、BMP ファイルのヘッダ 54Byte 分を取り除き、画素値だけを 2 次元配列内に格納する。また、BMP フォーマットの特色として、ファイルの状態にある画像データは、上下を反転させて格納されているので、これを元に戻す作業が必要となった。

この作業内の画像データの流れを述べる。まず、元になる入力画像ファイルのヘッダ部分だけを取り除いただけの画像データを、2 次元配列 buffer[ya][xa]へ格納する。このときはまだ上下が反転された状態なので、buffer[ya][xa]の y 座標を反転させて、別の配列 IMAGE[ya][xa]へと格納する。buffer の y 座標を buffer[ya]、IMAGE の y 座標を IMAGE[ya]とすると次式、

$$IMAGE[\,ya\,] = YSIZE - 1 - buffer[\,ya\,] \tag{6.2}$$

で上下を入れ換える事ができる。これも一つの座標変換といえる。

画像処理は、この IMAGE[ya][xa]へ読み込まれたデータを加工する事となる。

画像ファイルへの書き出し作業は、読み出しとは逆に、画像処理後の加工されたデータが格納された、IMAGE[ya][xa]の内容を上下反転させて、buffer[ya][xa]へ格納した後、書き込み用ファイルを生成・書き込み作業を行うものである。画像ファイルの書き出しフローを図 6-13 の様に表し、記述を行った。

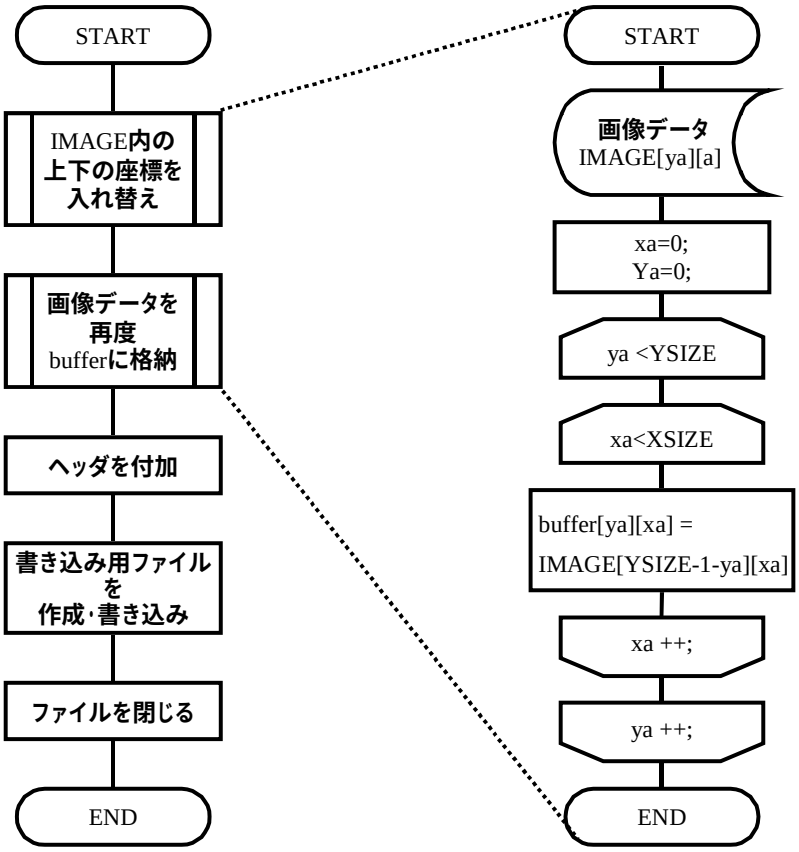


図 6-13　画像ファイル書き出しフロー

書き出しの場合は、読み出しと逆に画像データ部分を、

$$buffer[\,ya\,] = YSIZE - 1 - IMAGE[\,ya\,] \tag{6.3}$$

としてファイルへ書き出す。

こうして処理を分けて置く事で、画像処理のアルゴリズム部分だけを入れ換え・追加記述することで、機能を増やす事ができると考えた。

画像処理アルゴリズムの記述について述べる。BMP ファイル内の画像データ部分を上下反転させるものも含め、比較的記述が簡単だと思われる座標変換アルゴリズムの記述を行った。

ここでは、幾何学的情報の変換（第 5 章 5.4.3 参照）で述べたように、画像の座標を一定の規則に沿って変換するが、BMP フォーマットの画像は、CIP 形式のラスタ型データ表現であるので、1 画素を表現するのに、チャンネルの数だけの画素値を必要とする。すなわち、前述したとおり、画像データを格納する配列、buffer、IMAGE は画像サイズのチャンネル数だけ倍された横幅を持つ事になる。この場合は横方向が 3 倍の大きさ（式 6.1、図 6-11 参照）となる。

まず始めに試みた変換では、ラスタ型のデータをそのまま、一度に座標変換しようと試行錯誤してみたが、チャンネルの並びが壊れて画素値のデータを壊してしまい、色を再現できなくなった。チャンネルの並びが横並びから縦並びへと変化する事になった為である。これを図示すると以下の様になり、

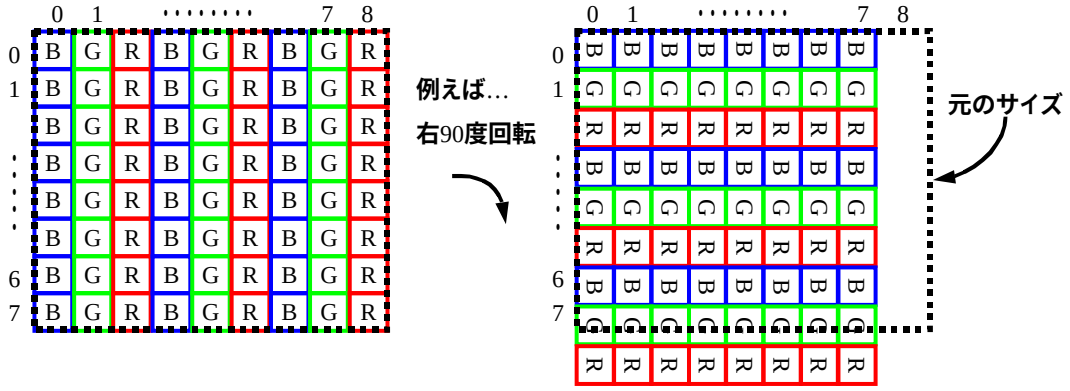


図 6-14　ラスタ型画像の回転

元の配列と回転後の配列ではその大きさと画素値の並びが崩れる事がわかる。その後、一度に変換する事をやめ、別の方法を考えるに到った。

ラスタ型の場合、x、y 座標をそのまま変換するだけでは、元画像の形を保ったままの変換画像を出力することはできない。よって、まず読み出された画像データを図 6-12 の様に、チャンネルごとに配列し直し、RGB 各色のモノクローム画像（第 5 章 1 参照）を取り出す。その後、図 6-15 の様にデータの座標変換を行い、ラスタ型のデータ表現通りに配列を戻し

て、RGB の各信号の並びを崩さず出力画像を生成しようと考え、実践した。

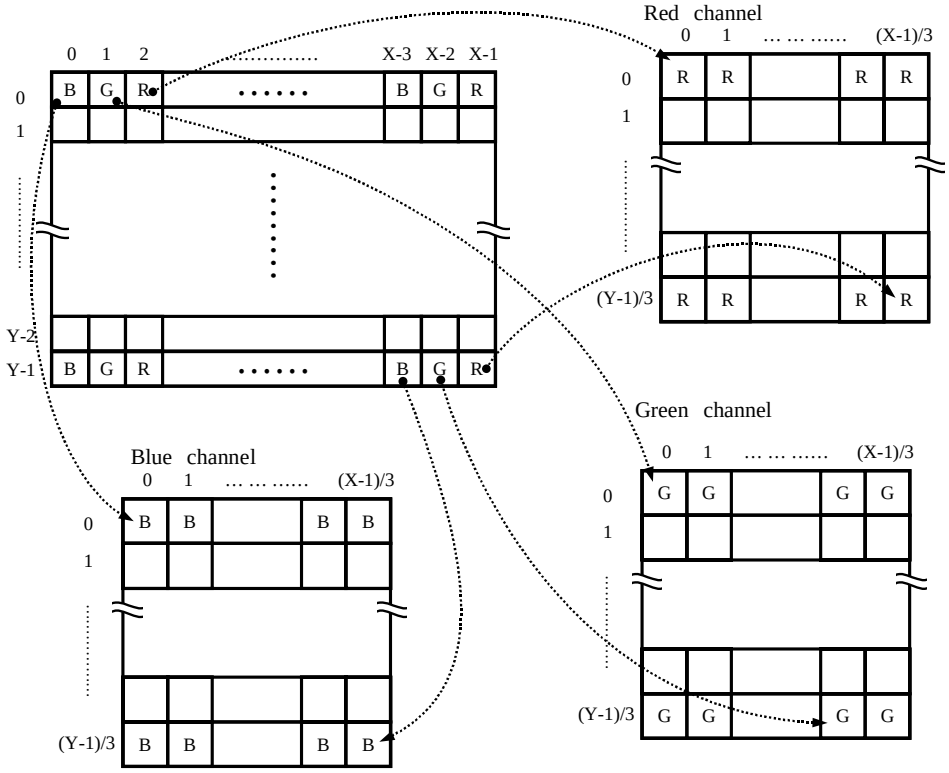


図 6-15　チャンネルデータの生成

チャンネル分割の為に 2 次元配列 B、G、R を設け、それぞれ青、緑、赤の画素値を格納する。分割の記述は以下の記述、

```
for(y=0;y<YSIZE;y++)
        {
                for(x=0;x<XSIZE/3;x++)
                {
                        B[y][x] = IMAGE[y][x*3];
                        G[y][x] = IMAGE[y][x*3+1];
                        R[y][x] = IMAGE[y][x*3+2];
                }
        }
```

で可能になる。座標変換はこの 2 次元配列 B、G、R に対して行う。

ラスタ型への再構成の際には、以下の記述で行うことができた。

```
for(yb=0;yb<YSIZE;yb++)
{
        for(xb=0;xb<XSIZE/3;xb++)
        {
                IMAGE[yb][3*xb] = b[yb][xb];
                IMAGE[yb][3*xb+1] = g[yb][xb];
                IMAGE[yb][3*xb+2] = r[yb][xb];
        }
}
```

このとき使用した 2 次元配列 b[yb][xb]、g[yb][xb]、r[yb][xb]は各チャンネルの座標変換後に、一時的にデータを格納させる為に使用したものである。データの流れを図示した。

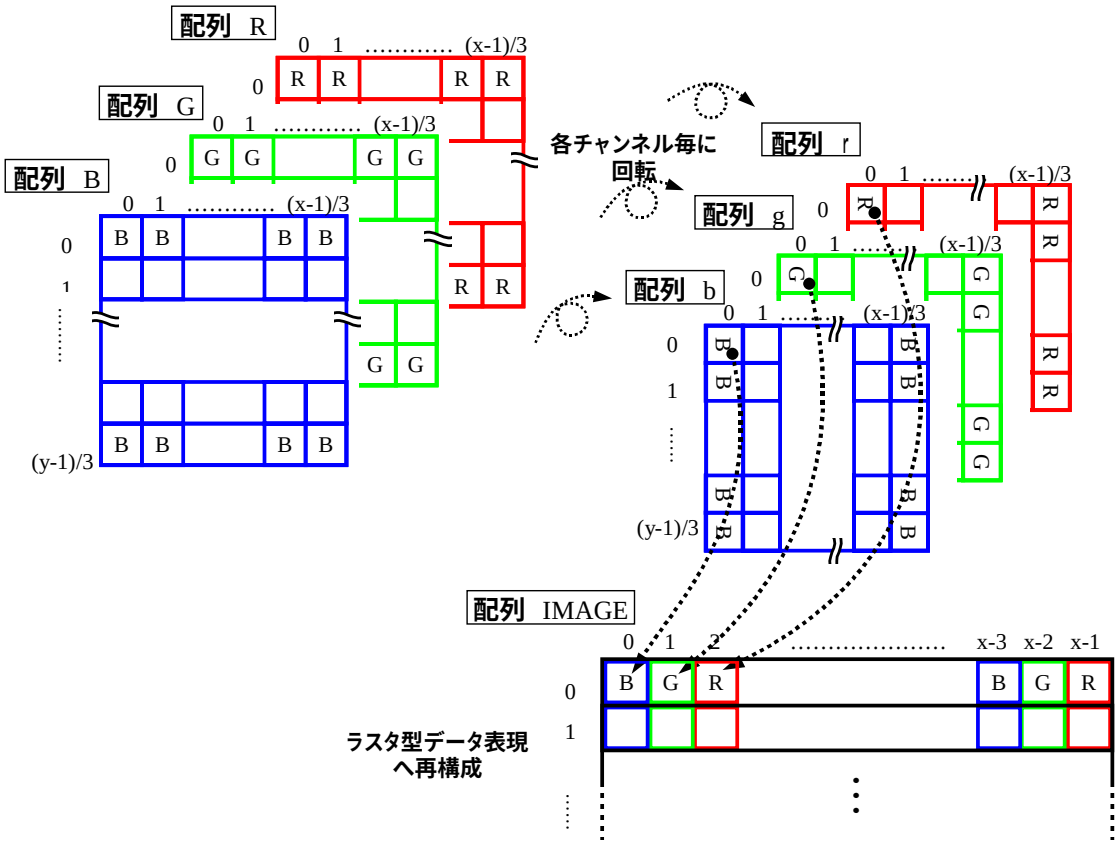


図 6-16　座標変換時のデータと配列の関係例

以下には右 90 度回転、左 90 度回転の例を示す。

・ 右 90 度回転

```
for(y=0;y<YSIZE;y++)
        {
                for(x=0;x<XSIZE/3;x++)
                {
                        b[y][x] = B[YSIZE-1-x][y];
                        g[y][x] = G[YSIZE-1-x][y];
                        r[y][x] = R[YSIZE-1-x][y];
                }
        }
```

・ 左 90 度回転

```
for(y=0;y<YSIZE;y++)
{
        for(x=0;x<XSIZE/3;x++)
        {
                b[y][x] = B[x][YSIZE-1-y];
                g[y][x] = G[x][YSIZE-1-y];
                r[y][x] = R[x][YSIZE-1-y];
        }
}
```

これらの座標変換アルゴリズムの動作確認は、生成された出力画像をビューアで見る事で確認できるが、データ変換を確認する為に、プログラムの要所に 2 次元配列の中身をテキストファイルに書き出す操作を記述したり、16 進ダンプを行ったりして、各チャンネルの画素値を数値で読み取り、確認することもできた（16 進ダンプの例は付録を参照)。

6.7 VHDL 記述

ハードウェア設計において決定した回路の構成と、C 言語で記述した内容を基に、ＶＨＤＬを用いて、各回路ブロックごとにビヘイビア・レベルから記述し、RTL 定義を考える。

ハードウェア構成に必要な機能ブロックを記述していく。

6.7.1. メモリの記述

第 6 章５で述べたハードウェア設計の図 6-6 を参照して、メモリの記述を行う。ここではスタティック RAM の記述を行った。図 6-14 に RAM の動作を表すタイミングチャートを示す。

VHDL では範囲を 2 つ以上定義することで、多次元の配列を宣言することができるが、この多次元配列はロジック回路を生成できないので、シミュレーションパターンの作成、ハードウェアのモデル化のみに限定使用されるものである。【1】

今回の RAM 記述では、2 次元配列を使用せず、通常のデータタイプとサブタイプ記述を利用して、配列の二重定義を行う。

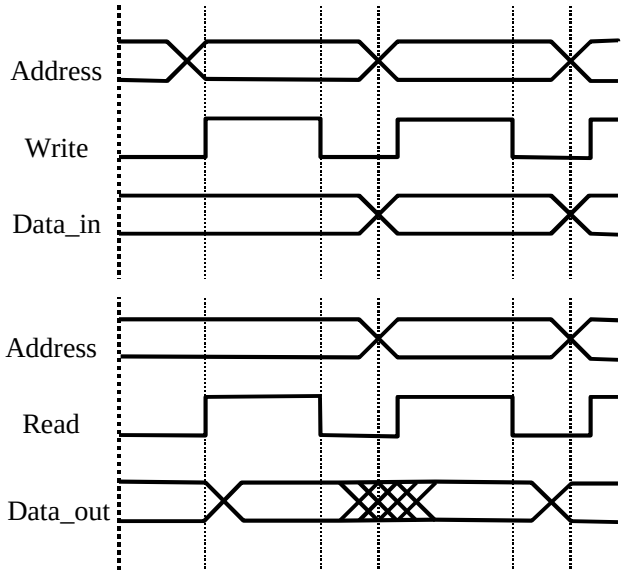


図 6-17　RAM のタイミングチャート

データタイプの 2 重定義の記述部分を以下に示す。

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

entity SRAM is

Port ( RD,WR,CS : in std_logic;
       ADR : in std_logic_vector(7 downto 0);
       DIN : in std_logic_vector(7 downto 0);
       DOUT : out std_logic_vector(7 downto 0));
end SRAM;

architecture Behavior of SRAM is

       subtype SRAMWR is std_logic_vector(7 downto 0);      ・・・・二重定義の為のサブタイプ文
       type SRAMARRAY is array (0 to 255)ofSRAMWR;      ・・・・サブタイプを利用して
                                                                データタイプを作る
       signal SRAMDATA : SRAMARRAY;    ・・・・・ROMARRAY をタイプとした信号宣言
       signal INT : integer range 0 to 255;
```

ここではまず、SRAMWR がサブタイプ宣言され、新しく SRAMARRAY というデータタイプが作られているのがわかる。この後、SRAMARRAY をデータタイプとした信号、SRAMDATA が宣言されている。

この後も、ハードウェ設計を参照して、各回路ブロックの VHDL 記述を行う。

7 結果と考察

自分が設計した回路が動作する所を見る、というのを目標として本研究に従事したが、結果として実際の回路へと到達するまでには到らず、最終目標はクリアできなかった。その原因として自覚するところは、長期にわたる研究に対して時間配分、計画性が甘かった為と思われる。

しかし、VHDL を使用してどの様に回路設計、LSI 設計を進めていくか、広範囲の学習ができた。但し、今回初めて学んだシステム設計という分野では、ほとんどソフトウェア開発に関する資料を参考にするしかなかったので、ハードウェア設計への適用が正しいかどうか、はっきりしない結果となっている。

また、C 言語でのプログラミングにおいて、画像処理のアルゴリズムとしては簡単であるものの、きちんとした動作を確認する事ができた。

今後の課題として、この後も回路をしっかりと製作し、動作確認ができるレベルまで仕上げたい。また、動作する事が検証されたプログラムも、色々なアルゴリズムを追記して、機能を増やして行きたいと考えている。

8 まとめ

本研究では、最初に定義した目標を完全に全うする事はできなかったが、画像フィルタ回路を製作する作業は、ＶＨＤＬ、システム開発、画像処理、Ｃ言語、それぞれの知識を必要とする、複合的作業である。広範囲に調査・学習が進められた事は、今後も役立つ事であり、良い経験になると思われる。長期にわたる実験・研究では、計画性を持つことが重要であると痛感した。

ものづくりを最終課題として良かったと思う。何もない所から始めて、形にするまでが、いかに多くの工程を要する難しい作業であるかが良くわかった。今後の為にも、残された課題をきちんと解決したい。

9 参考文献

1） 白石　肇　著：“わかりやすいシステム LSI 入門”オーム社（1999）

2） 長谷川　裕恭　著：“VHDL によるハードウェア設計入門”CQ 出版社（1995）

3） 桜井　至　著：“HDL によるデジタル設計の基礎”テクノプレス（1997）

4） 桜井　至　著：“LSI 設計の基礎技術”テクノプレス（1999）

5） 松尾　三郎　監修：“新版システム開発の基礎”電子開発学園出版局（1998）

6） 伊東　静男　著：“はじめてのシステム設計入門”同文舘（1995）

7） 日本ユニシス情報処理システム教育研究会　著：“基本情報短期集中ゼミ”（2000）

8） 高田　美樹　著：“C 言語の総合研究”技術評論社（2000）

9） 柴山　潔　著：“コンピュータアーキテクチャの基礎”近代科学社（1998）

10） 画像処理標準テキストブック編集委員会　監修：“画像処理標準テキストブック”
財団法人画像情報教育振興協会（1999）

11）財団法人　日本情報処理開発協会　中央情報教育研究所：
“情報処理技術者　スキル標準（2001）”

謝辞

本卒業研究にあたり、所属研究室指導教員であります橘昌良助教授、原央学科長、矢野政顕教授には、多大なご心配とご指導を賜り、深謝の念に堪えません。本学入学以来ご教授賜り、本研究への礎を築き、お導き下さった、電子・光システム工学科の教員・職員の皆様と共に、ここに厚く御礼申し上げます。

また、共に最後まで卒業研究室で互いに切磋琢磨し、研究に励んだ同期生諸氏に感謝します。

平成 14 年 2 月 7 日　河村　恵美

付録

・本研究で記述したＣ言語ソースを記載する。

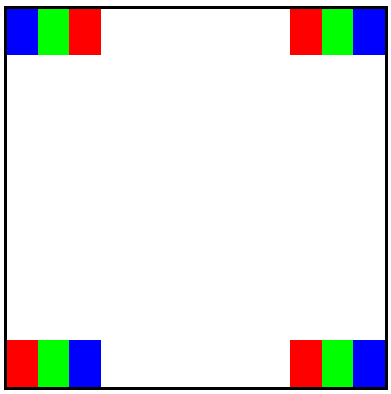
```
/*
*********************************************
 2002.1.19
 BMP ファイルの入出力
 上下反転出力
 左 90 度回転
 右 90 度回転
*********************************************
*/

        #define XSIZE 480
        #define YSIZE 160

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void NAME(char *FILENAME);
void Filter1(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME);
void Filter2(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME);
void Filter3(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME);
void IMAGE_WRITE(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME);

/*ヘッダサイズ５４Ｂ、イメージバッファ、ファイルネームバッファ確保　B,G,R3 チャンネ
ルバッファ確保
*/

unsigned char HEADER[54],image[YSIZE][XSIZE];
char buff[80];
unsigned char b[YSIZE][XSIZE/3],g[YSIZE][XSIZE/3],r[YSIZE][XSIZE/3];


void main()
        {
        char *filename=buff;

/*Filter1 実行*/
        printf("Please Reading Image File Name.... ¥n");
        NAME(filename);Filter1(image,filename);
        printf("Image File Read end.[%s] ¥n¥n",filename);

        printf("Please Writing Filter1 Imgae Fle Name.... ¥n");
        NAME(filename); IMAGE_WRITE(image,filename);
        printf("Image File Write end.[%s] ¥n¥n",filename);

/*Filter2 実行*/
        Filter2(image,filename);
        printf("Please Writing Filter2 Imgae File Name.... ¥n");
        NAME(filename); IMAGE_WRITE(image,filename);
        printf("Image File Write end.[%s] ¥n¥n",filename);
```

```
/*Filter3 実行*/
        Filter3(image,filename);
        printf("Please Writing Filter3 Imgae File Name.... ¥n");
        NAME(filename); IMAGE_WRITE(image,filename);
        printf("Image File Write end.[%s] ¥n¥n",filename);

}

/*ファイル読み込み、書き込み時、拡張子の追加*/
void NAME(char *FILENAME)
{
        scanf("%s",FILENAME);
        if(strstr(FILENAME,".bmp") == NULL){
                strcat(FILENAME,".bmp");
        }
}

/*BMP イメージの上下反転関数*/
void Filter1(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME)
{
        int xa,ya;

        FILE *fp;
        unsigned char buffer[YSIZE][XSIZE];
        unsigned char B[YSIZE][XSIZE/3],G[YSIZE][XSIZE/3],R[YSIZE][XSIZE/3];

        if((fp=fopen(FILENAME,"rb")) == NULL)
        {
                printf("Image Read Error. [%s] ¥n",FILENAME);
         exit(1);
        }
        else
        {
                fread(HEADER,1,54,fp); fread(buffer,YSIZE,XSIZE,fp);
                fclose(fp);
        }

                for(ya=0; ya<YSIZE; ya++)
                {
                        for(xa=0; xa<XSIZE; xa++)
                        {
                                IMAGE[ya][xa] = buffer[YSIZE-1-ya][xa];

                        }

                }

}

/*BMP イメージの左 90 度回転関数*/
void Filter2 (unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME)
{
        int xa,ya;
        int x,y,xb,yb;


        FILE *
```

```
unsigned char buffer[YSIZE][XSIZE];
unsigned char B[YSIZE][XSIZE/3],G[YSIZE][XSIZE/3],R[YSIZE][XSIZE/3];
unsigned char b[YSIZE][XSIZE/3],g[YSIZE][XSIZE/3],r[YSIZE][XSIZE/3];

if((fp=fopen(FILENAME,"rb")) == NULL)
{
        printf("Image Read Error. [%s] ¥n",FILENAME);
 exit(1);
}
else
{
        fread(HEADER,1,54,fp); fread(buffer,YSIZE,XSIZE,fp);
        fclose(fp);
}

        for(ya=0; ya<YSIZE; ya++)
        {
                for(xa=0; xa<XSIZE; xa++)
                {
                        IMAGE[ya][xa] = buffer[YSIZE-1-ya][xa];


                }

        }

for(y=0;y<YSIZE;y++)
{
        for(x=0;x<XSIZE/3;x++)
        {

                        B[y][x] = IMAGE[y][x*3];

                        G[y][x] = IMAGE[y][x*3+1];

                        R[y][x] = IMAGE[y][x*3+2];
        }
}

for(y=0;y<YSIZE;y++)
{
        for(x=0;x<XSIZE/3;x++)
        {
                        b[y][x] = B[x][YSIZE-1-y];

                        g[y][x] = G[x][YSIZE-1-y];

                        r[y][x] = R[x][YSIZE-1-y];


        }
}


/*復元*/
        for(yb=0;yb<YSIZE;yb++)
        {
```

```
                for(x=0;x<XSIZE/3;x++)
                {
                        b[y][x] = B[YSIZE-1-x][y];

                        g[y][x] = G[YSIZE-1-x][y];

                        r[y][x] = R[YSIZE-1-x][y];

                }
        }

/*復元*/
        for(yb=0;yb<YSIZE;yb++)
        {
                for(xb=0;xb<XSIZE/3;xb++)
                {
                        IMAGE[yb][3*xb] = b[yb][xb];
                        IMAGE[yb][3*xb+1] = g[yb][xb];
                        IMAGE[yb][3*xb+2] = r[yb][xb];

                }
        }
}
/*BMP イメージの書き出し関数*/
void IMAGE_WRITE(unsigned char IMAGE[YSIZE][XSIZE],char *FILENAME)
        {
        int xa,ya,xb;

        unsigned char buffer[YSIZE][XSIZE];
        unsigned char B[YSIZE][XSIZE/3],G[YSIZE][XSIZE/3],R[YSIZE][XSIZE/3];
        FILE *fp;

        for(ya=0; ya<YSIZE; ya++)
        {
                for(xa=0; xa<XSIZE; xa++)
                {
                        buffer[ya][xa] = IMAGE[ya][xa];
                }
        }

        if((fp=fopen(FILENAME,"wb")) == NULL)
        {
                printf("Write Image Error. [%s] ¥n",FILENAME); exit(1);
        }
        else
        {
                fwrite(HEADER,1,54,fp); fwrite(buffer,YSIZE,XSIZE,fp);
                fclose(fp);
        }

}
```

・座標変換前と座標変換後の 16 進ダンプの例



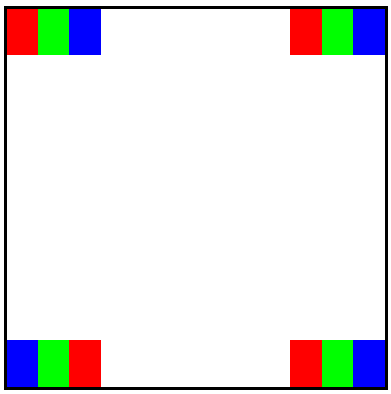
画像サイズを 24×24 として、16 進ダンプを行った例を示す。左のサンプル画像の画素値を 16 進数で見ると、下の図の様になる。下の図では、サンプルの BMP ファイル内をアドレス 00000000 番地から 16 進数で見たものと、アドレスを最後から見たものである。始めの 54 番地分をヘッダとして、ヘッダの次から画像左下から右上へ向かって画素値が並んでいる。

ヘッダの次から、ＢＧＲの順にチャンネルが並んでいる。始めの色は左下の赤、Ｒ100%で、画素値に直すとＢ、Ｇ共に 0、Ｒは 10 進で 255、図では 16 進で FF。

```
C:¥BORLAND¥BCC55¥BIN¥sam.bmp
 ADDRESS : +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F      ASCII
00000000 : 42 4D F6 06 00 00 00 00 00 00 36 00 00 00 28 00  BM........6...(.
00000010 : 00 00 18 00 00 00 18 00 00 00 01 00 18 00 00 00  ................
00000020 : 00 00 C0 06 00 00 C4 0E 00 00 C4 0E 00 00 00 00  ................
00000030 : 00 00 00 00 00 00 00 00 FF 00 00 FF 00 FF 00 00  ................
00000040 : FF 00 FF 00 00 FF 00 00 FF FF FF FF FF FF FF FF  ................
00000050 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000060 : FF FF FF FF FF FF FF FF FF FF FF FF 00 00 FF 00  ................
00000070 : 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00 00 00  ................
00000080 : FF 00 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00  ................
00000090 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000000A0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000000B0 : FF FF FF FF 00 00 FF 00 00 FF 00 FF 00 00 FF 00  ................
000000C0 : FF 00 00 FF 00 00 00 00 FF 00 00 FF 00 FF 00 00  ................
000000D0 : FF 00 FF 00 00 FF 00 00 FF FF FF FF FF FF FF FF  ................
000000E0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000610 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00  ................
00000620 : 00 FF 00 00 00 FF 00 00 FF 00 00 00 FF 00 00 FF  ................
00000630 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000640 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000650 : FF FF FF FF 00 00 FF 00 00 FF 00 FF 00 00 FF 00  ................
00000660 : FF 00 00 FF 00 00 FF 00 00 FF 00 00 00 FF 00 00  ................
00000670 : FF 00 00 00 FF 00 00 FF FF FF FF FF FF FF FF FF  ................
00000680 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000690 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00  ................
000006A0 : 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00 FF 00  ................
000006B0 : 00 FF 00 00 00 FF 00 00 FF 00 00 00 FF 00 00 FF  ................
000006C0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000006D0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000006E0 : FF FF FF FF 00 00 FF 00 00 FF 00 FF 00 00 FF 00  ................
000006F0 : FF 00 00 FF 00 00                               ......
```

アドレスの終わり部分は右上に向かって、2 ピクセルずつ赤緑青の並びになっている。画素値はそれぞれ 16 進数で、Ｒ(FF)、Ｇ(FF)、Ｂ(FF)。

・上下反転させた画像の 16 進ダンプの例



左の画像は、先のサンプル画像を上下反転させたもので、この画像で 16 進ダンプを行った例を下図に示す。前の例と比較すると、画素値の並びが逆になっているのがわかる。

```
C:¥BORLAND¥BCC55¥BIN¥samx.bmp
 ADDRESS : +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F      ASCII
00000000 : 42 4D F6 06 00 00 00 00 00 00 36 00 00 00 28 00  BM........6...(.
00000010 : 00 00 18 00 00 00 18 00 00 00 01 00 18 00 00 00  ................
00000020 : 00 00 C0 06 00 00 C4 0E 00 00 C4 0E 00 00 00 00  ................
00000030 : 00 00 00 00 00 00 FF 00 00 FF 00 00 00 FF 00 00  ................
00000040 : FF 00 00 00 FF 00 00 FF FF FF FF FF FF FF FF FF  ................
00000050 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000060 : FF FF FF FF FF FF FF FF FF FF FF FF 00 00 FF 00  ................
00000070 : 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00 FF 00  ................
00000080 : 00 FF 00 00 00 FF 00 00 FF 00 00 00 FF 00 00 FF  ................
00000090 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000000A0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000000B0 : FF FF FF FF 00 00 FF 00 00 FF 00 FF 00 00 FF 00  ................
000000C0 : FF 00 00 FF 00 00 FF 00 00 FF 00 00 00 FF 00 00  ................
000000D0 : FF 00 00 00 FF 00 00 FF FF FF FF FF FF FF FF FF  ................
 ADDRESS : +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F      ASCII
00000600 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000610 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00  ................
00000620 : FF 00 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00  ................
00000630 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000640 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000650 : FF FF FF FF 00 00 FF 00 00 FF 00 FF 00 00 FF 00  ................
00000660 : FF 00 00 FF 00 00 00 00 FF 00 00 FF 00 FF 00 00  ................
00000670 : FF 00 FF 00 00 FF 00 00 FF FF FF FF FF FF FF FF  ................
00000680 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000690 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 00  ................
000006A0 : 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00 00 00  ................
000006B0 : FF 00 00 FF 00 FF 00 00 FF 00 FF 00 00 FF 00 00  ................
000006C0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000006D0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
000006E0 : FF FF FF FF 00 00 FF 00 00 FF 00 FF 00 00 FF 00  ................
000006F0 : FF 00 00 FF 00 00                               ......
```