第５版

画像認識ユニット

NVP-Ax230SDK

Software Development Kit

Fine Vision Processor

ユーザーズ マニュアル

maxell
マクセルシステムテック株式会社

はじめに

このたびは、ＮＶＰ－Ａｘ２３０シリーズの画像処理ユニット（ＮＶＰ－Ａｘ２３０ＣＬ/２３５ＣＬ）および「ＮＶＰ－Ａｘ２３０ＳＤＫ」をお買い上げいただきまして、誠にありがとうございます。

本マニュアルは、ＮＶＰ－Ａｘ２３０シリーズを使用したアプリケーション作成のための基本ソフトウェアである、「ＮＶＰ－Ａｘ２３０ＳＤＫ」について記載しております。ハードウェアについては、各ボードのハードウェアマニュアルをご参照ください。

なお、本マニュアルではＮＶＰ－Ａｘ２３０ＣＬ/２３５ＣＬの画像処理ユニットを特に区別する必要がない場合には、ＮＶＰ－Ａｘ２３０と記載しております。

⚠ ご注意

●システムの構築やプログラム作成などの操作を行う前に、本マニュアルの記載内容をよく読み、書かれている指示や注意を十分理解して下さい。誤った操作によりシステムの故障が発生することがあります。

●本マニュアルの記載内容について理解できない内容、疑問点または不明点がございましたら、弊社営業窓口までお知らせ下さい。
また、弊社ホームページのお問い合わせのページからも受け付けておりますのでご利用ください。
http://www.systemtech.maxell.co.jp/solution/vp/

●お客様の誤った操作に起因する、事故発生や損害につきましては、弊社は責任を負いかねますのでご了承ください。

●弊社提供のハードウェアおよびソフトウェアを無断で改造しないでください。この場合の品質および安全につきましては、弊社は責任を負いかねますのでご了承ください。

●本マニュアルの内容について予告なく変更する場合がございます。

－ 目次 －

第1章 製品概要

1.1 NVP－Ax230SDKの概要

本ＳＤＫは画像処理ユニットＮＶＰ－Ａｘ２３０シリーズのソフトウェア開発キットです。ＮＶＰ－Ａｘ２３０シリーズのアプリケーションソフトウェア開発に必要なライブラリ、ツール、ドキュメントが含まれています。

1.1.1 特徴

・Ｗｉｎｄｏｗｓ、Ｖｉｓｕａｌ Ｃ／Ｃ＋＋環境でのアプリケーションの構築が可能です。
・スタンドアロンシステムのアプリケーション開発が可能です。
・オンボードＣＰＵでのインテリジェント処理モジュールを容易に構築可能です。
・ＮＶＰ－Ａｘ２３０からの入力映像だけでなく、映像ファイルからの映像入力もサポートしています。

1.1.2 動作環境

本ＳＤＫを使用しＷｉｎｄｏｗｓ上のＶｉｓｕａｌ Ｃ／Ｃ＋＋環境でのアプリケーション開発とオンボードＣＰＵで動作するアプリケーションやインテリジェント処理モジュールなどのダウンロードモジュール開発を行うことができます。
Ｗｉｎｄｏｗｓ上のＶｉｓｕａｌ Ｃ／Ｃ＋＋環境でのアプリケーション開発とオンボードアプリケーションのダウンロードモジュール開発にはそれぞれ本ＳＤＫ以外に以下の環境が必要です。

表1-1-1 Ｗｉｎｄｏｗｓ上のＶｉｓｕａｌ Ｃ／Ｃ＋＋環境でのアプリケーション開発環境

項目		内容
ハードウェア	パソコン	Ｗｉｎｄｏｗｓが動作するパソコン 10/100Base T/TX ×1 ポート以上
	NVP-Ax230	ＮＶＰ－Ａｘ２３０ＣＬ／２３５ＣＬ　　：　最大１６台 （論理的に認識可能なユニット数）
ＯＳ		Ｍｉｃｒｏｓｏｆｔ Ｗｉｎｄｏｗｓ ＸＰ （ＳｅｒｖｉｃｅＰａｃｋ２以上） Ｍｉｃｒｏｓｏｆｔ Ｗｉｎｄｏｗｓ ７ ３２Ｂｉｔ版 （６４Ｂｉｔ版は不可） （ＳｅｒｖｉｃｅＰａｃｋ１以上）
コンパイラ		Ｍｉｃｒｏｓｏｆｔ Ｖｉｓｕａｌ Ｓｔｕｄｉｏ ２００５ 日本語版 （Ｖｉｓｕａｌ Ｃ/C++　ＳｅｒｖｉｃｅＰａｃｋ１以上） Ｍｉｃｒｏｓｏｆｔ Ｖｉｓｕａｌ Ｓｔｕｄｉｏ ２０１０ 日本語版 （Ｖｉｓｕａｌ Ｃ/C++　ＳｅｒｖｉｃｅＰａｃｋ１以上）

インストールディスク容量　約３００ＭＢ

表1-1-2 オンボードアプリケーションのダウンロードモジュール開発環境

項目	内容
コンパイラ	SuperH RISC engine C/C++ コンパイラパッケージ Ver9.0 (ルネサスエレクトロニクス)

1.2 画像処理ユニットNVP－Ax230の概要

1.2.1 NVP－Ax230CLのハードウェア構成

画像処理ユニットＮＶＰ－Ａｘ２３０ＣＬは、次の図に示すように、

・ＣＰＵ（SH-4Aコア）
・画像処理プロセッサ
・画像メモリ及びシステムメモリ（ＵＭＡ方式)(2GB)
・フラッシュメモリ
・映像入力部（カメラリンク入力用２ＣＨ）
・映像出力部
・アイソレーションＩ／Ｏポート
・イーサネット（100Base-T)装備
・ＳＣＩ（２ＣＨ＋カメラリンクデータ通信用１ＣＨ）

から構成されております。

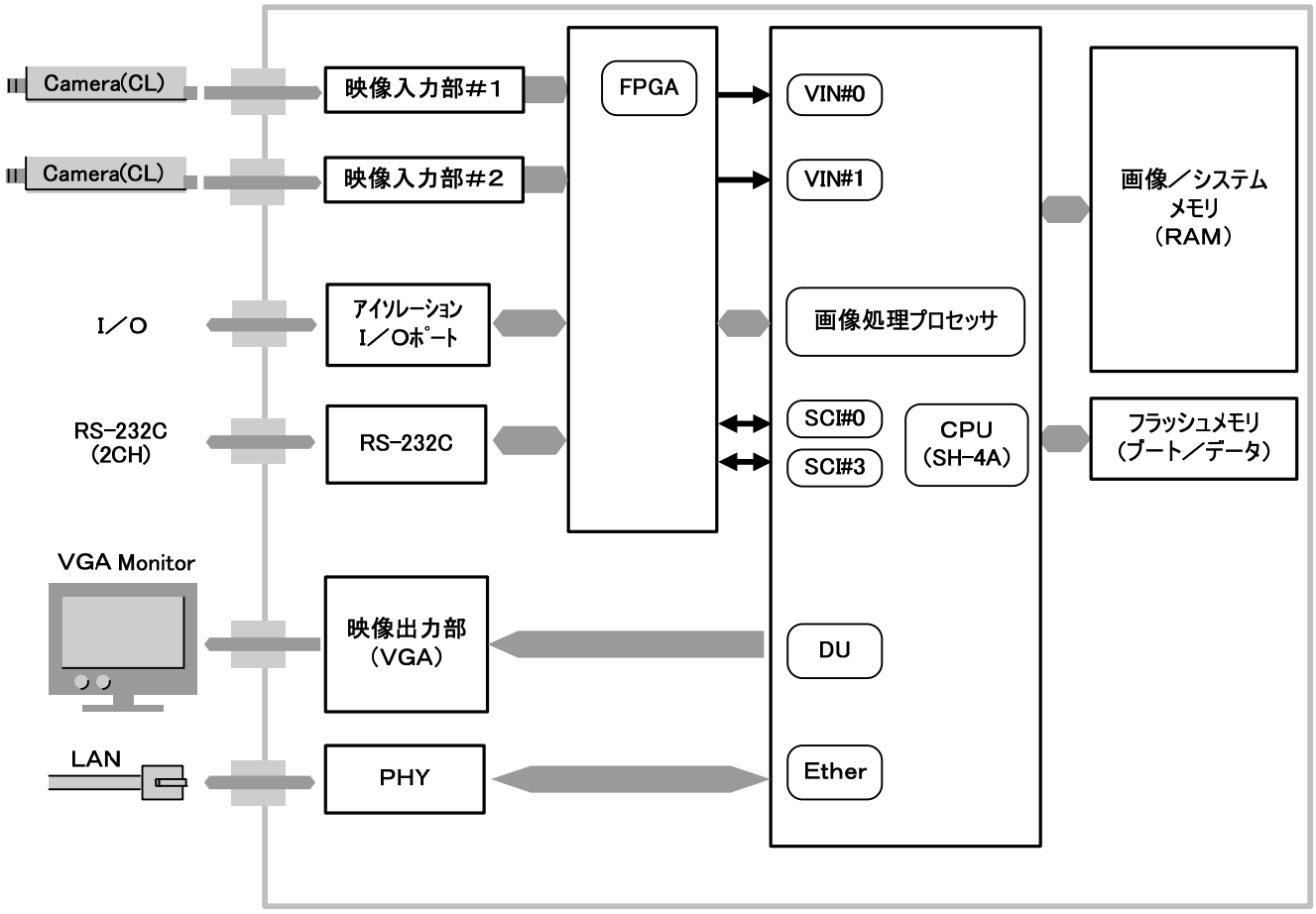


図1-2-1　画像処理ユニットＮＶＰ－Ａｘ２３０ＣＬ

1.2.2 NVP－Ax235CLのハードウェア構成

画像処理ユニットＮＶＰ－Ａｘ２３０ＣＬは、次の図に示すように、

・ＣＰＵ（SH-4Aコア）
・画像処理プロセッサ
・画像メモリ及びシステムメモリ（ＵＭＡ方式）(2GB)
・フラッシュメモリ
・映像入力部（カメラリンク入力用４ＣＨ）
・映像出力部
・アイソレーションＩ／Ｏポート
・イーサネット（100Base-T)装備
・ＲＴＣ（バッテリバックアップ）
・ＵＳＢホスト（２ＣＨ）
・ＳＤメモリカード
・ＳＣＩ（２ＣＨ＋カメラリンクデータ通信用１ＣＨ）

から構成されております。

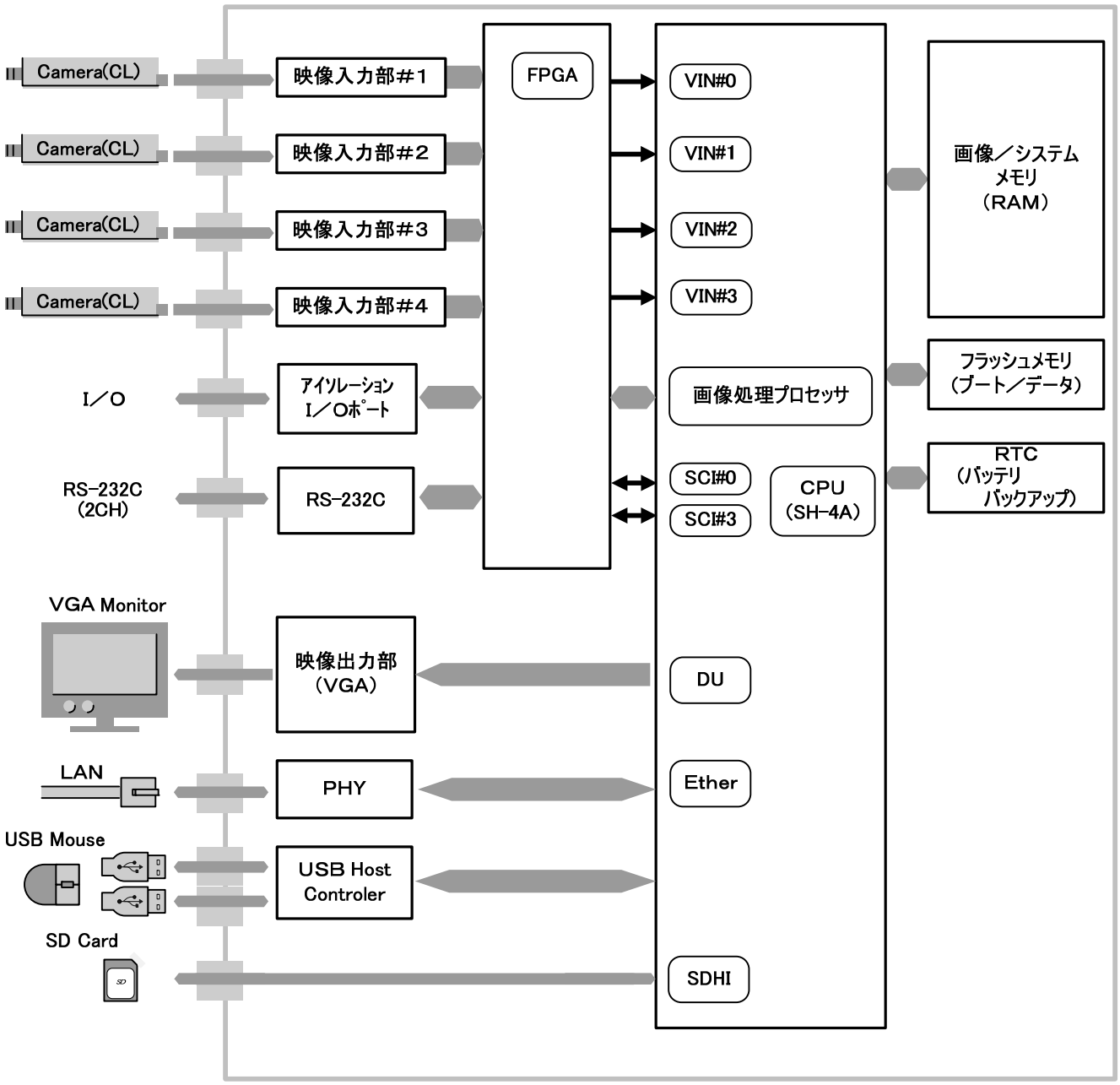


図1-2-2　画像処理ユニットＮＶＰ－Ａｘ２３５ＣＬ

1.3 画像認識コマンドとリモートコマンド

画像認識コマンドとは、画像認識を行うための画像入力、画像２値化といった個々のサブルーチンで、Ｃ言語でいえば関数であり画像認識関数と言う場合もあります。また画像認識コマンドは、ユーザーがＮＶＰ－Ａｘ２３０で画像認識アプリケーションを作成する場合の最小の実行単位であり、画像認識アプリケーションは画像認識コマンドを複数組み合わせて作成されます。

ＮＶＰ－Ａｘ２３０は、画像認識やシステム制御等をスタンドアロンで実行します。実行モジュールはＮＶＰ－Ａｘ２３０のＲＯＭやＲＡＭ上にあり、オンボードＣＰＵで実行されます。一方、Ｗｉｎｄｏｗｓパソコンから画像認識コマンドを実行できればＷｉｎｄｏｗｓで動作するコンパイラ、デバッガなどを使用し画像認識の開発を行うことができます。そこで本ＳＤＫではＷｉｎｄｏｗｓパソコンからコマンド通信により、画像処理プロセッサとオンボードＣＰＵに画像認識をリモートで動作実行させます。本ＳＤＫでは、Ｗｉｎｄｏｗｓパソコンからのコマンドをリモートコマンドと呼びます。

1.3.1 Windowsでのリモートコマンド構成

Ｗｉｎｄｏｗｓ上のリモートコマンドは、下の図に示すように、ＤＬＬ（Dynamic Link Library）、カーネルモードのデバイスドライバ、オンボードＣＰＵのオンボードシステム実行ファイルで構成されます。

画像処理コマンドシステムは、起動する際にオンボードＣＰＵで実行される画像処理コマンドを含むオンボードシステムの実行ファイルをＮＶＰ－Ａｘ２３０へダウンロードします。

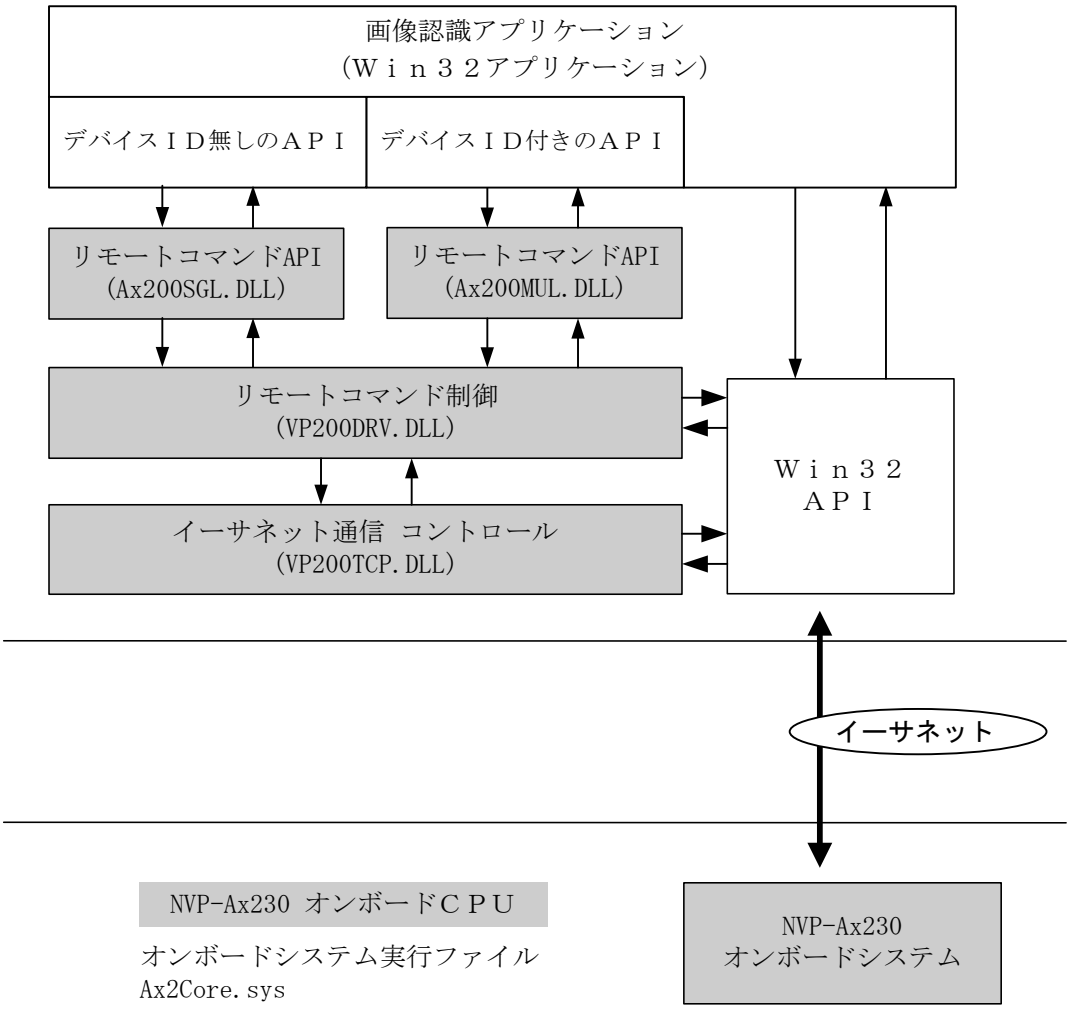


図1-3-1　リモートコマンドの構成

1.3.2 リモートコマンドAPI

リモートコマンドＡＰＩは、シングルユニット構成やマルチユニット構成でシステムに対応するため、VisualC/C++で次の２種類のインタフェースを用意しています。

(1)ユニットを識別するためのユニット識別子（デバイスＩＤ）無しのＡＰＩ

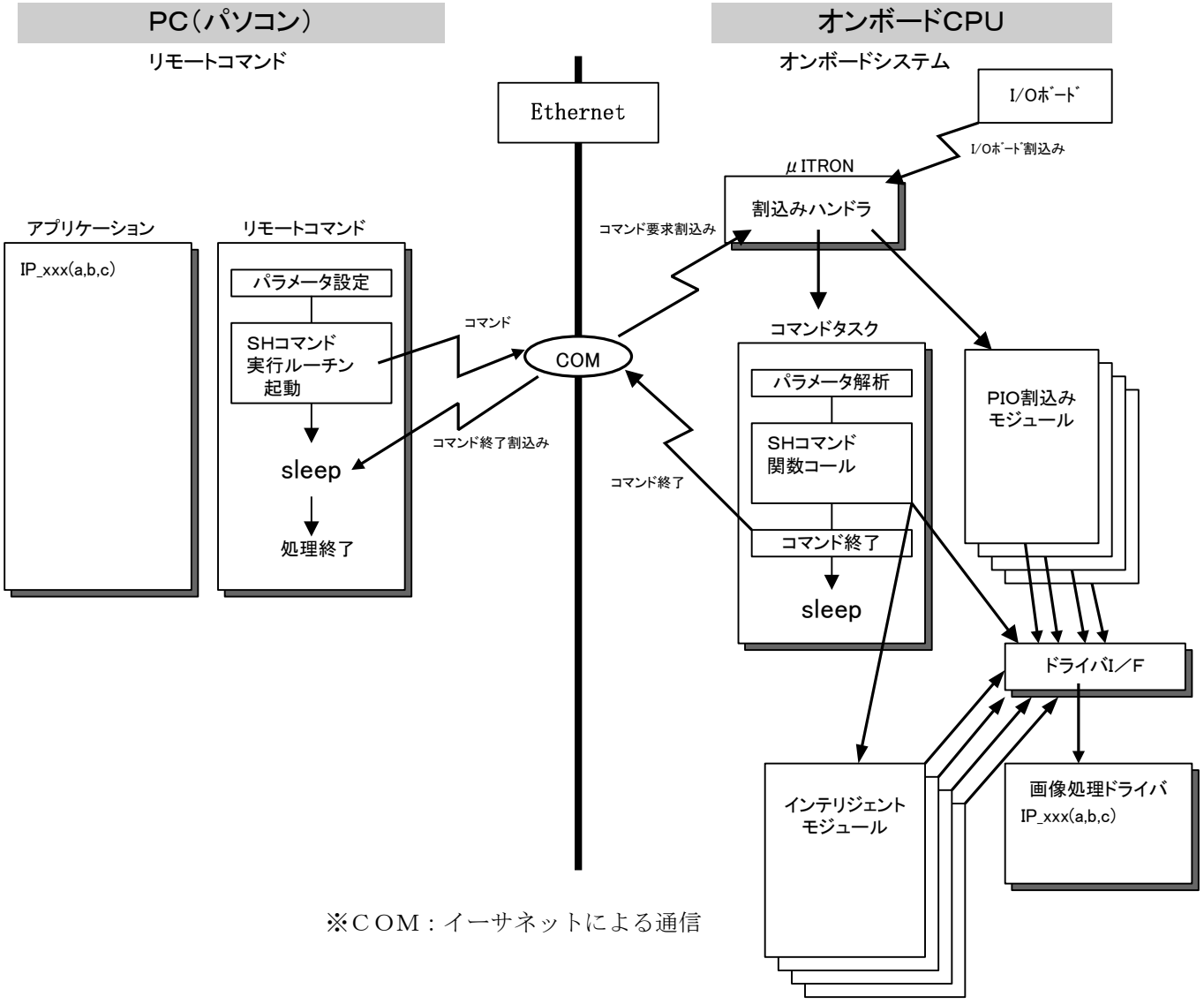
(2)ユニットを識別するためのユニット識別子（デバイスＩＤ）付きのＡＰＩ

シングルユニット構成のシステムには、ユニットを識別するためのユニット識別子（デバイスＩＤ）無しのＡＰＩを使用します。画像処理ＡＰＩのベーシックな形式で、オンボードシステムのモジュールは、常にこの形式のＡＰＩで動作します。ユニットを識別するためのユニット識別子（デバイスＩＤ）付きのＡＰＩは、マルチユニット構成でＮＶＰ－Ａｘ２３０を使用する場合にデバイスＩＤで常に複数のユニットを切換えながら画像処理を行う場合に使用します。ユーザアプリケーションに応じてデバイスＩＤ無しのＡＰＩ（Ax200SGL）と有りのＡＰＩ（Ax200MUL）を実行ファイル単位にそれぞれのライブラリを選択することで使い分けてください。

1.3.3 リモートコマンドの動作概要

ＮＶＰ－Ａｘ２３０の画像処理コマンドシステムでは、実行モジュールはＮＶＰ－Ａｘ２３０のＲＡＭ上にあり、また、オンボードシステムの画像処理コマンドは、画像処理プロセッサとオンボードＣＰＵにより処理が行われます。

リモートコマンドは下の図に示すようにＰＣ（パソコン）とＮＶＰ-Ａｘ２３０のオンボードＣＰＵとの間でコマンド通信することによりオンボードＣＰＵで画像処理を実行します。



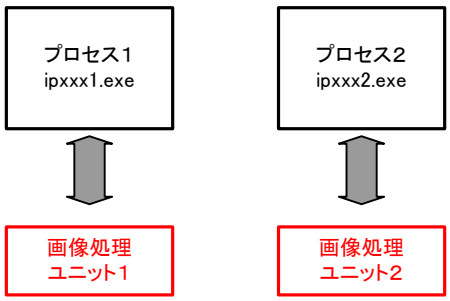
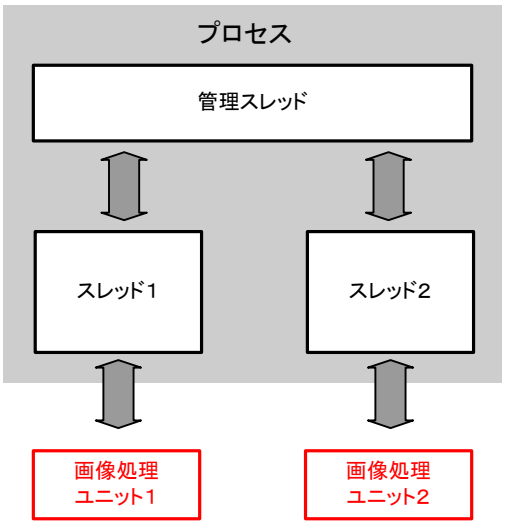
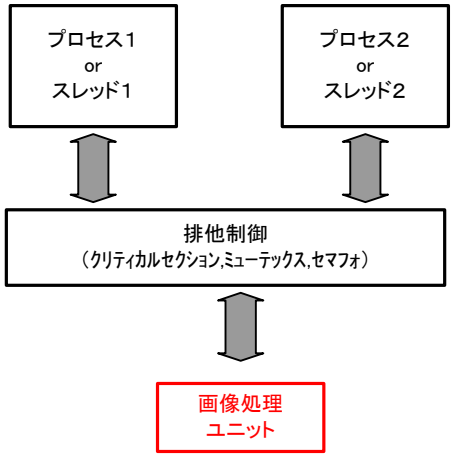
※ＣＯＭ：イーサネットによる通信

図1-3-2　リモートコマンドの動作概要

1.3.4 マルチユニット構成時の動作

画像処理ユニットは１台のパソコンでＮＶＰ－Ａｘ２３０ＣＬ/２３５ＣＬは最大１６台までコントロール可能です。Ｗｉｎｄｏｗｓ上でのマルチプロセス、マルチスレッドでのアプリケーションにも対応しています。１ユニットに２つ以上のプロセスやスレッドからアクセスする場合にはクリティカルセクション、ミューテックス、セマフォ等を使用して２つ以上のプロセスやスレッドから１ユニットに対するコマンド要求がお互いに中断されないように排他制御を行う必要があります。

表1-3-1　マルチユニットでの対応可能な構成

処理内容	ハード構成	備考
複数のプロセスから各々異なる画像処理ユニットをアクセスする		
複数のスレッドから各々異なる画像処理ユニットをアクセスする		
複数のプロセスやスレッドから１台の画像処理ユニットをアクセスする		排他制御を行う

1.4 オンボードモジュール

本ＳＤＫでは、オンボードＣＰＵで動作させるモジュールをオンボードモジュールと呼びます。オンボードモジュールには、アプリケーションモジュールとダウンロードモジュールがあります。

1.4.1 アプリケーションモジュール

アプリケーションモジュールとは、ユニットのファイルシステム（フラッシュメモリ等）にアプリケーションを書き込んでおき起動時に動作させるようなユニット単独で動作するアプリケーションです。

1.4.2 ダウンロードモジュール

ダウンロードモジュールとは、オンボードＣＰＵで動作する実行モジュールをＷｉｎｄｏｗｓパソコンからＮＶＰ－Ａｘ２３０にダウンロードして実行するようなモジュールです。ダウンロードモジュールには、インテリジェントモジュール、割込み起動モジュールがあります。Ｗｉｎｄｏｗｓパソコンからダウンロードモジュールを制御するための関数を用意しています。

(1) インテリジェントモジュール

インテリジェントモジュールとは、画像認識コマンドを複数組み合わせて作成するユーザーオリジナルの画像認識コマンドのことです。ＮＶＰ－Ａｘ２３０のリモートシステムでは、画像処理は画像処理プロセッサとオンボードＣＰＵにより行っているため、ユーザーの画像認識アプリケーションの一部をインテリジェントモジュールとして実装することにより、処理の分散化を実現できます。インテリジェントモジュールは画像認識コマンドの一部として登録し実行されます。

(2) 画像認識タスクモジュール

ボード上のＣＰＵでは、システムをリアルタイムに動作させるためリアルタイムＯＳ：μＩＴＲＯＮを使用しております。タスクとはアプリケーションを独立して並列に処理可能な単位で分割したプログラムのことです。ボード上のＣＰＵではシステムとしてさまざまなタスクがすでに動作しており、スタンドアロンの実行モジュールはタスクとして管理されます。インテリジェントモジュールは、リモートコマンドの一部として動作しているため独立したタスクとして動作しているわけではありません。画像認識タスクモジュールは、μＩＴＲＯＮの１つのタスクとして登録、実行されるため、パソコンとは独立して動作することが可能です。

(3) 割込み起動モジュール

割込み起動モジュールとは、ボード上のアイソレーションパラレルＩ／Ｏ（ＰＩＯ）からの割込みにより起動することができるモジュールです。基本的に画像処理タスクモジュールの１つであり、特に、ユーザーアプリケーションでＰＩＯからの割込み処理を簡単に扱うことができるようにフレームを用意してあります。画像処理コマンドを複数組み合わせて作成したユーザーオリジナルの画像認識モジュールをＰＩＯの割込み起動モジュールとして登録し、ＰＩＯの割込みにより起動することができます。

第2章 パッケージ内容

本ＳＤＫは以下の内容を提供します。

・リモートコマンドライブラリ（Ｌｉｂ、ＤＬＬ、ヘッダファイル含む）
・オンボードＣＰＵ用コマンドライブラリ（Ｌｉｂ）
・ＮＶＰ－Ａｘ２３０用オンボードシステムファイル

2.1 リモートコマンドライブラリ

本ライブラリは、Ｍｉｃｒｏｓｏｆｔ　Ｖｉｓｕａｌ　C/C++用のライブラリです。
このライブラリは、ＬｉｂファイルとＤＬＬ（Dynamic Link Library）で提供され、Ｌｉｂファイルをアプリケーションとリンクすることでアプリケーションを構築できます。また、リモートコマンドＡＰＩは、シングルユニット構成やマルチユニット構成でシステムに対応するため、Windows Visual C/C++でユニットを識別するためのユニット識別子（デバイスＩＤ）無しのＡＰＩとデバイスＩＤ付きのＡＰＩを用意しています。

2.2 オンボードCPUコマンドライブラリ

本ライブラリは、SuperH RISC engine C/C++ コンパイラパッケージ Ver9.0用のライブラリです。
このライブラリは、Ｌｉｂファイルで提供され、Ｌｉｂファイルをアプリケーションとリンクすることでアプリケーションを構築できます。

2.3 NVP－Ax230用オンボードシステムファイル

オンボードシステムファイルは、ＮＶＰ－Ａｘ２３０上でハードウェアを動作させ、画像処理を実行するためのサービスプログラムです。

2.4 インストール内容

本ＳＤＫのインストールで以下の内容のファイルがインストールされます。

(1) フォルダ構成

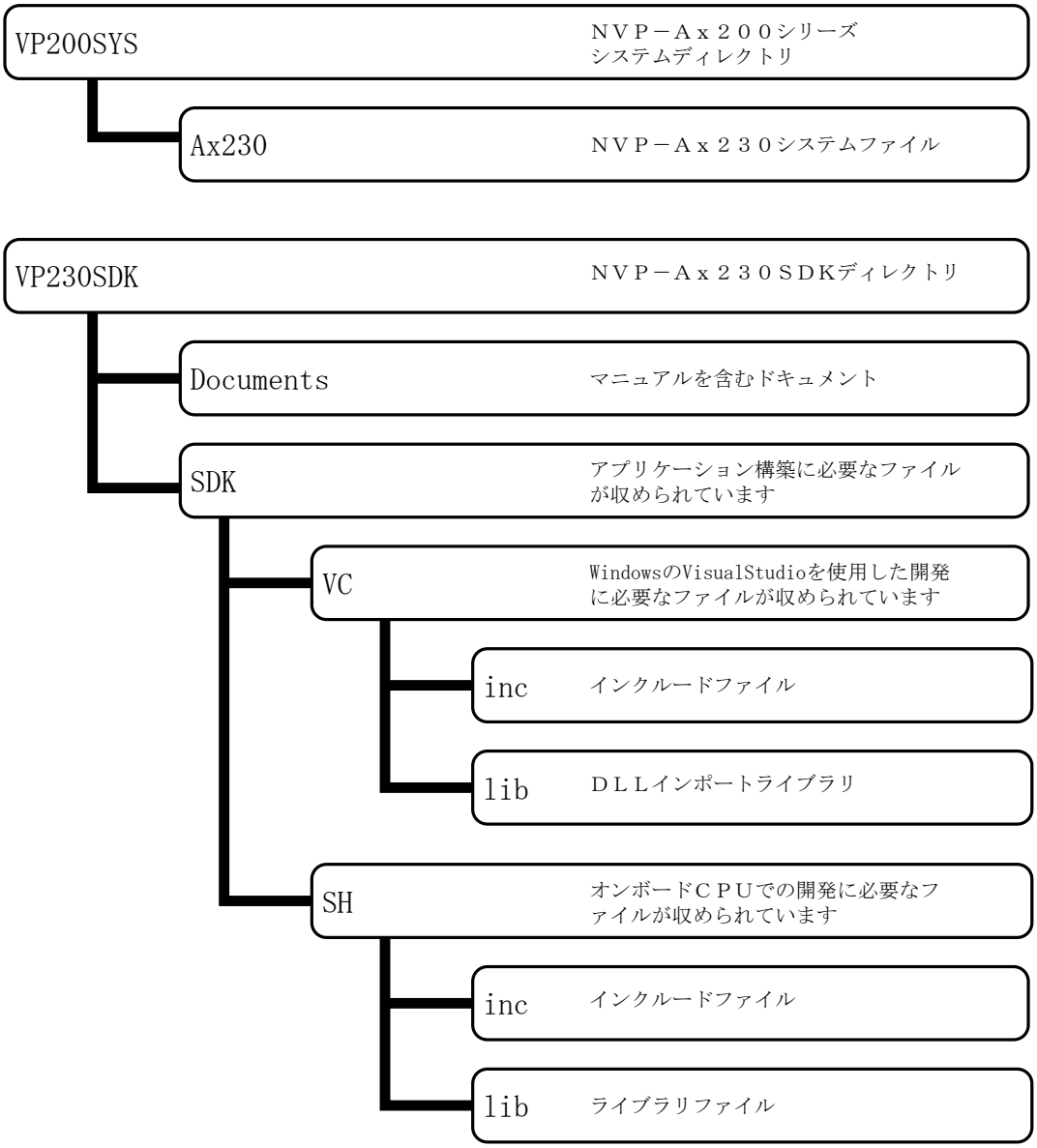


図2-4-1 ＳＤＫのフォルダ構成

(2) ファイル構成

本ＳＤＫのライブラリで提供する主なモジュールを以下に示します。

表2-4-1　ＳＤＫの主なモジュール

項目	ファイル	内容
リモートコマンド ライブラリ	Ax200MUL.LIB	画像認識ライブラリＬＩＢファイル （デバイスＩＤ付き版）
	Ax200SGL.LIB	画像認識ライブラリＬＩＢファイル （デバイスＩＤ無し版）
	IPXDEF.H	画像認識ライブラリ用ヘッダファイル
	IPXSYS.H	画像認識ライブラリ用ヘッダファイル
	IPXPROT.H	画像認識ライブラリ用ヘッダファイル
オンボードシステム ライブラリ	ipxcmd2.lib	画像認識コマンドＡＰＩ
	ipxctl2.lib	ボード制御コマンドＡＰＩ
	stdfnc2.rel	Ｃ言語標準入出力関数
	knlsvc2.lib	μＩＴＲＯＮサービスコールＡＰＩ
	fmdisk2.lib	ファイルアクセスＡＰＩ
	tcpip2.lib	TCP/IP ＢＳＤソケット通信ＡＰＩ
	usbapi2.lib	ＵＳＢ制御ＡＰＩ
	IPXDEF.H	画像認識ライブラリ用ヘッダファイル
	IPXSYS.H	画像認識ライブラリ用ヘッダファイル
	IPXPROT.H	画像認識ライブラリ用ヘッダファイル
ＤＬＬ	Ax200MUL.DLL	リモートコマンド制御ＤＬＬファイル （デバイスＩＤ付き版）
	Ax200SGL.DLL	リモートコマンド制御ＤＬＬファイル （デバイスＩＤ無し版）
	VP200DRV.DLL	リモートコマンド制御ＤＬＬファイル
	VP200TCP.DLL	イーサネット通信コントロールＤＬＬファイル
	VP200REG.DLL	レジストリ設定制御ＤＬＬファイル
オンボードシステム	Ax2Core.SYS	NVP-Ax230用オンボードシステムファイル

第3章 NVP－Ax230SDKの使用方法

3.1 画像認識アプリケーションの開発手順

3.1.1 アプリケーションモジュールの開発

本ＳＤＫでは、プロジェクトジェネレータによりVisualStudio C/C++とHewのプロジェクトを簡単に作成することができます。また、リモートコマンドによりボード上のコマンドをＷｉｎｄｏｗｓパソコン上から実行することでＷｉｎｄｏｗｓ上のVisualStudio C/C++のデバッガを使用しアルゴリズムの評価などを行うことができます。そしてリモートコマンドで開発したソースコードをそのままＳＨのＣ言語コンパイラでコンパイルすることでオンボードモジュールを作成し、NVP-Ax230上でのパフォーマンスを評価します。以下にアプリケーションモジュールの開発手順の概略を示します。

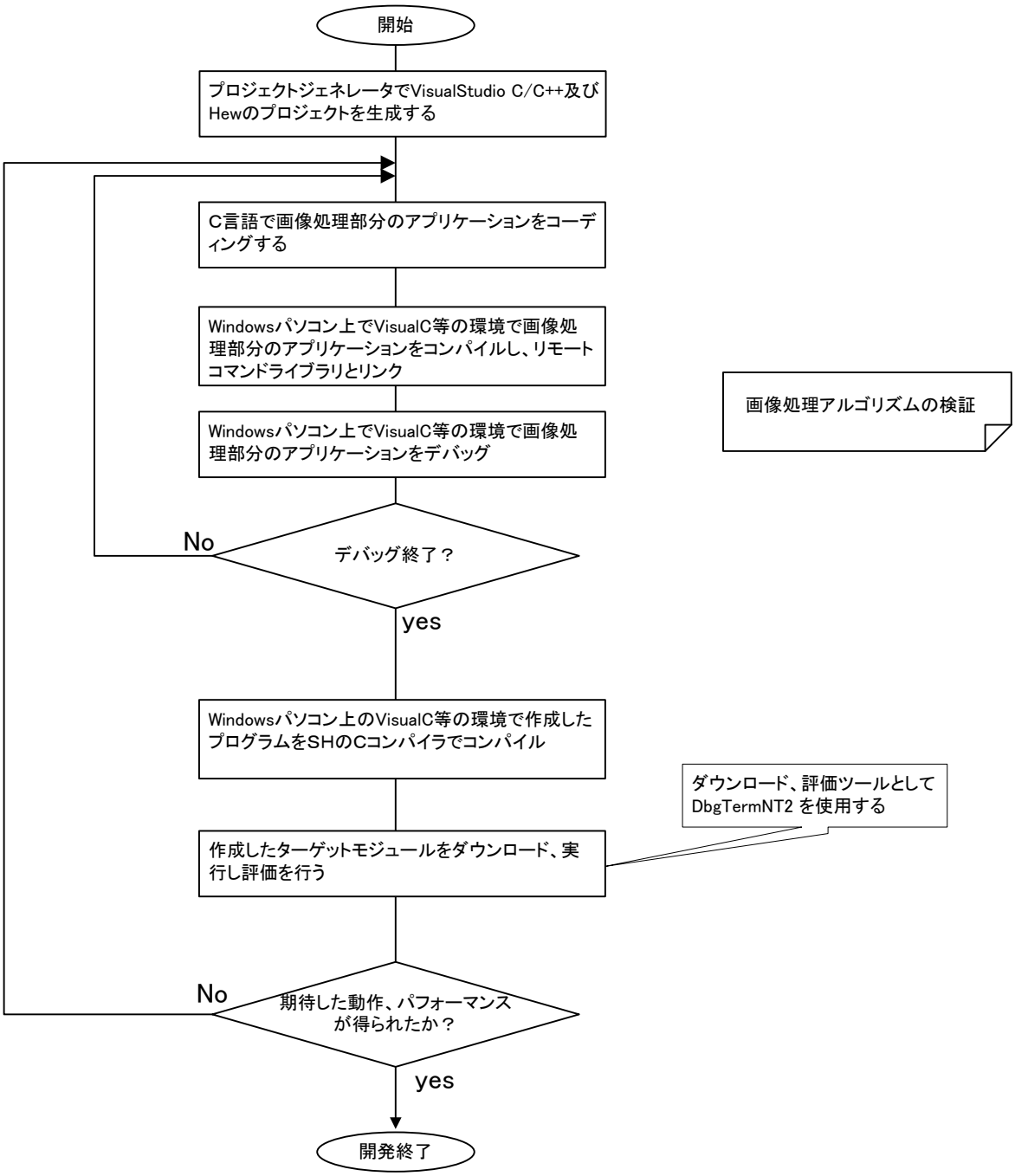


図3-1-1　アプリケーションモジュールの開発手順

3.1.2 ダウンロードモジュールの開発

オンボードＣＰＵで動作するインテリジェントモジュールや画像認識タスクといったダウンロードモジュールの開発では、アプリケーションモジュールと同様にリモートコマンドによるアルゴリズムの評価などを行い、そのソースコードをＳＨのＣ言語コンパイラでコンパイルすることでオンボードモジュールを作成する以外にＷｉｎｄｏｗｓパソコン上でのモジュールのダウンロード、Ｗｉｎｄｏｗｓパソコンとユニット間のデータの受け渡しなどの部分を作成する必要があります。

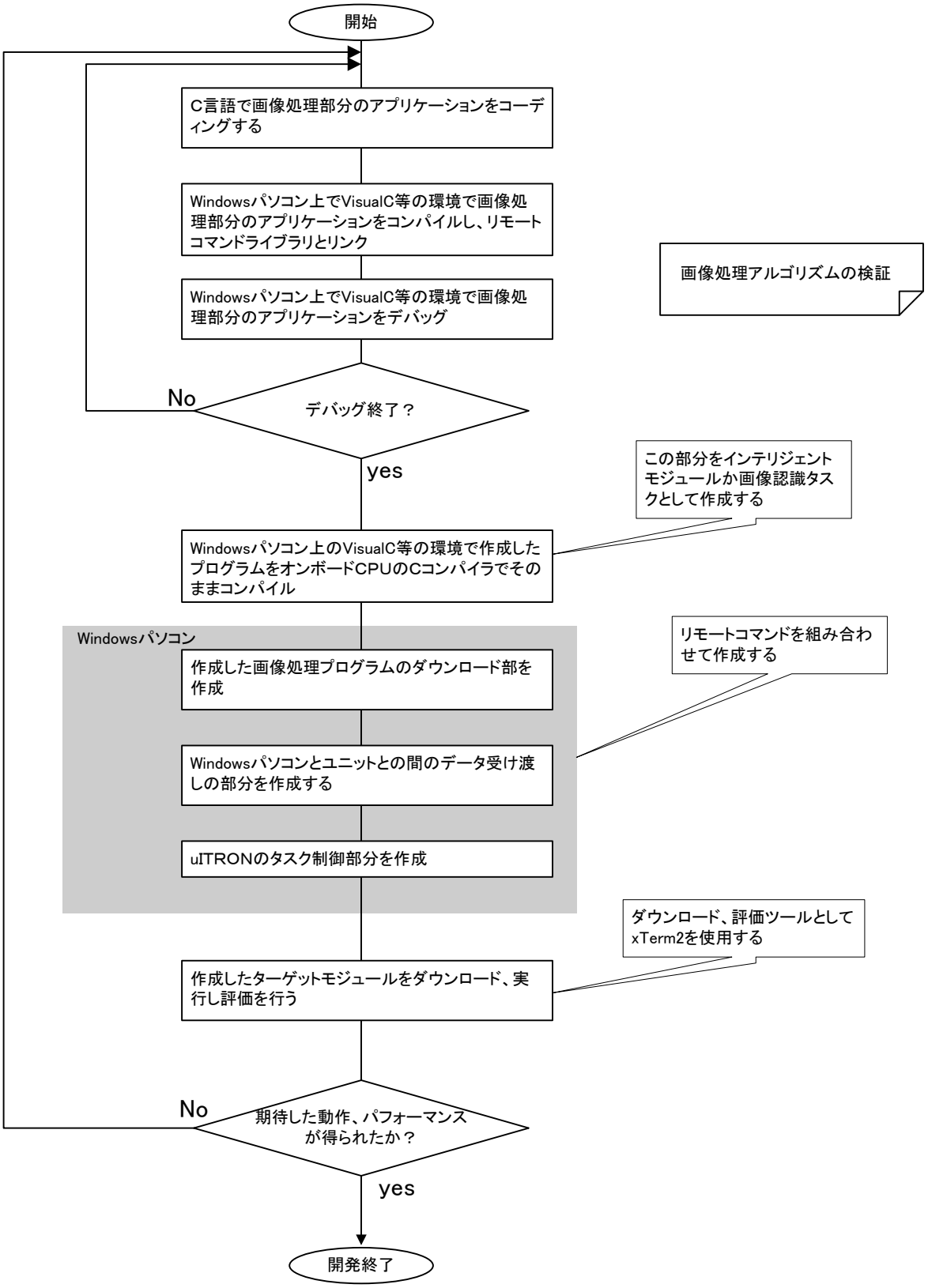


図3-1-2　ダウンロードモジュールの開発フロー

3.1.3 画像認識アプリケーション開発の留意点

オンボードモジュール開発の際の留意点を次に記載します。

(1) 割込みハンドラのスタックサイズ

オンボードシステム上の割込みハンドラ（非タスク）のスタックサイズは、多重割込の場合を入れて最大１ＫＢ（1024B）に設定されています。

(2) 画像処理コマンドの排他制御

オンボードシステムの画像処理コマンドは、画像処理プロセッサとオンボードＣＰＵにより処理が行われます。複数タスクから画像処理プロセッサに処理を要求する場合、互いの処理が中断されないように排他制御を行う必要があります。

3.2 プロジェクトジェネレータ

本ＳＤＫで用意しているプロジェクトジェネレータを使用することで、Ｖｉｓｕａｌ Ｓｔｕｄｉｏ用のプロジェクトとＨｅｗ用のプロジェクトを簡単に作成することができます。また、プロジェクトジェネレータにより、画像認識ライブラリを使用して画像認識アプリケーションを作成する際のアプリケーションモジュール用のメインプログラムのテンプレートとして、以下の「main.c」のソースコードも同時に作成します。プロジェクトジェネレータの詳細は「ProGen2操作マニュアル」を参照して下さい。なお、プロジェクトジェネレータを使用しないでプロジェクトを作成する場合は、「付録Ａ　コンパイラの設定」を参照しプロジェクトを作成してください。

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{
    int  ret;

    /* 画像処理ボードの起動 */
    ret = StartIP( IPBOARD_0, 0 );
    if(ret < 0){
        printf("ボードの起動に失敗しました¥n");
        return;
    }

    /* 画像処理コマンドの初期化 */
    InitIP( );

    /* ここからユーザプログラムを記述してください  */



    /* 画像認識ボードの停止 */
    StopIP( IPBOARD_0 );
}
```

図3-2-1　アプリケーションモジュール　main関数のテンプレート

3.3 コーディング

3.3.1 インクルードファイルの記述

ＷｉｎｄｏｗｓアプリケーションやオンボードＣＰＵで動作するダウンロードモジュールアプリケーションを作成する場合、以下に示すインクルードファイルを以下の順番でプログラムにインクルードしてください。

表3-3-1 インクルードファイル一覧

ファイル名	内容	インクルードする順番
ipxdef.h	define,enum定義	1
ipxsys.h	構造体定義	2
ipxprot.h	プロトタイプ定義	3

3.3.2 ユーザアプリケーションの作成

アプリケーションモジュールを作成する場合、Visual Studioで作成したソースをそのままHewでコンパイルしターゲットボードで評価する場合、Visual Studioのプロジェクトはコンソールアプリケーションとして構築する必要があります。下図のように「main()」関数から処理が始まるようにプログラムを作成してください。なお、ダウンロードモジュールに関しては「main()」関数は必要ありません。

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{


}
```

図3-3-1　メイン関数

アプリケーションモジュールやダウンロードモジュールの評価時は、Ｃの標準入出力として以下の関数が使用可能ですのでそれらを使用しデバッグを行って下さい。標準入出力関数の入出力が可能なアプリケーションがそれぞれ違いますので注意してください。アプリケーションモジュールは『DbgTermNT2』から実行する場合は『DbgTermNT2』、『Startup.bat』で電源投入時に自動的に実行する場合は『xTerm2』になります。また、ダウンロードモジュールは『xTerm2』になります(詳細は3.4章を参照して下さい）。

表3-3-2　オンボードモジュールでの標準関数

関数名	内容	入出力アプリケーション		
		アプリケーション モジュール		ダウンロード モジュール
		DbgTermNT2から 実行する場合	Startupから 実行する場合	
printf	標準出力に文字列を出力する。 Ｃの標準入出力関数のprintfと同等。	DbgTermNT2	xTerm2	xTerm2
scanf	標準入力から文字列を入力する。 Ｃの標準入出力関数のscanfと同等。	DbgTermNT2	xTerm2	xTerm2
malloc	Ｃの標準関数のmallocと同等。	-	-	-
free	Ｃの標準関数のfreeと同等。	-	-	-

3.3.3 リモートコマンドでのボードの初期化

リモートコマンドを使用する場合は、Ｗｉｎｄｏｗｓのユーザアプリケーションから必ず下記の画像認識ボードのセットアップを行う必要があります。また、画像認識ボードのセットアップは、デバイスＩＤ無しのリモートコマンドとデバイスＩＤ付きのリモートコマンドでそれぞれ異なります。

(1) デバイスID無しリモートコマンドでのボードの初期化

デバイスＩＤ無しのリモートコマンドで画像認識ボードを使用する場合は、ボード起動（StartIP）を実行し初期化します。そしてアプリケーションの終了時に画像認識ボードの停止処理（StopIP）を実行して下さい。

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{
/* ローカル変数の設定  */
    int  ret,ImgID1,ImgID2,ImgID3;

/*  プログラム          */

    /* 画像処理ボードの起動 */
    ret = StartIP( IPBOARD_0, 0 );
    if(ret < 0){
        printf("ボードの起動に失敗しました¥n");
        return;
    }

    /* 画像処理コマンドの初期化 */
    InitIP( );
    ActiveVideoPort( VIDEO_PORT1 );
    SelectCamera( CAMERA_PORT0, BW_CAMERA );
    SetVideoFrame( INTERLACE, VIDEO_FS_512H_480V );
    ImgID1 = AllocImg( IMG_FS_512H_512V );
    ImgID2 = AllocImg( IMG_FS_512H_512V );
    ImgID3 = AllocImg( IMG_FS_512H_512V );
    GetCamera( ImgID1 );
    IP_AddConst( ImgID1, ImgID2, 20 );
    IP_Binarize( ImgID2, ImgID3, 120 );
    DispImg( ImgID3 );

    /* 画像認識ボードの停止 */
    StopIP( IPBOARD_0 );
}
```

このインクルードファイルは必ずインクルードして下さい

画像認識ボードの起動

画像認識コマンド初期化

画像認識ボードの停止

図3-1-1　コーディング例

(2) デバイスID付きリモートコマンドでのボードの初期化

デバイスＩＤ付きのリモートコマンドで画像認識ボードを使用する場合は、ボードのオープン処理（OpenIPDev）を実行しデバイスＩＤを取得して下さい。そしてプログラムの終了時に画像認識ボードのクローズ処理（CloseIPDev）を実行して下さい。

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{
/* ローカル変数の設定  */
    DEVID   devID;
    int     ImgID1,ImgID2,ImgID3;

/*  プログラム          */

    /* 画像認識ボードのオープン */
    devID = OpenIPDev( IPBOARD_0, 0 );
    if( devID == ISPX_NULL ){
        printf("ボードの起動に失敗しました¥n");
        return;
    }

    /* 画像認識コマンドの初期化 */
    InitIP(devID);
    ActiveVideoPort(devID,VIDEO_PORT1 );
    SelectCamera( devID,CAMERA_PORT0, BW_CAMERA );
    SetVideoFrame(devID,INTERLACE,VIDEO_FS_512H_480V);
    ImgID1 = AllocImg(devID,IMG_FS_512H_512V);
    ImgID2 = AllocImg(devID,IMG_FS_512H_512V);
    ImgID3 = AllocImg(devID,IMG_FS_512H_512V);
    GetCamera(devID,ImgID1);
    IP_AddConst(devID,ImgID1,ImgID2,20);
    IP_Binarize(devID,ImgID2,ImgID3,120);
    DispImg(devID,ImgID3);

    /* 画像認識ボードのクローズ */
    CloseIPDev(devID);
}
```

このインクルードファイルは必ずインクルードして下さい

画像認識ボードのオープン

画像認識コマンド初期化

画像認識ボードのクローズ

図3-1-2　コーディング例

3.4 オンボードモジュールのデバッグ・評価方法

オンボードモジュールの構築は、Ｈｅｗでコンパイル、リンクを行いＥＬＦ形式の実行モジュールを作成します。デバッグ時の標準入出力のモニタリングは、アプリケーションモジュールは『DbgTermNT2』から実行する場合は『DbgTermNT2』、『Startup.bat』で電源投入時に自動的に実行する場合は『xTerm2』、ダウンロードモジュールは『xTerm2』で行ってください。

(1) アプリケーションモジュール

ＨｅｗでアプリケーションモジュールのＥＬＦ形式の実行モジュール(.ABSファイル)を作成し、本ＳＤＫで用意しているVPVisorでＮＶＰのディスクにファイルをコピーし、DbgTermNT2を使用して実行・評価を行います。DbgTermNT2には、ダウンロードして実行する機能もあり、その場合はＮＶＰのディスクにファイルをコピーしなくても実行可能です。DbgTermNT2の詳細は「DbgTermNT2操作マニュアル」を参照して下さい。また、電源投入時に自動的にStartup.batから実行する場合は、xTerm2を使用しprintf()やscanf()などの標準入出力のモニタリングを行います。

・DbgTermNT2で実行する場合

表3-4-1　アプリケーションモジュールでの標準入出力関数(DbgTermNT2)

関数名	内容	
printf	DbgTermNT2に文字列を出力する。Ｃの標準関数のprintfと同様。	
scanf	DbgTermNT2から文字列を入力する。Ｃの標準関数のscanfと同様。	

・電源投入時、Startupから実行する場合

表3-4-2　アプリケーションモジュールでの標準入出力関数(Startup.bat)

関数名	内容	
printf	xTerm2に文字列を出力する。Ｃの標準関数のprintfと同様。	
scanf	xTerm2から文字列を入力する。Ｃの標準関数のscanfと同様。	

(2) ダウンロードモジュール

ＨｅｗでダウンロードモジュールのＥＬＦ形式の実行モジュール(.ABSファイル)を作成し、Ｗｉｎｄｏｗｓ側のユーザアプリケーションからダウンロード、パラメータ転送、実行を行います。ダウンロードモジュール内の標準入出力(printf,scanf)はxTerm2を使用してモニタリングして下さい。なお、xTerm2を使用する場合は、あらかじめshellコマンド『tconnect』でxTerm2とネットワーク接続を行う必要があります。xTerm2の詳細は「xTerm2操作マニュアル」を参照して下さい。

表3-4-3　ダウンロードモジュールでの標準入出力関数

関数名	内容	
printf	xTerm2に文字列を出力する。Ｃの標準関数のprintfと同様。	
scanf	xTerm2から文字列を入力する。Ｃの標準関数のscanfと同様。	

第4章 画像処理の概要

4.1 画像処理コマンドの構成

画像間演算、空間フィルタリングなどの画像処理コマンドは、サブルーチン形式のコマンドとしてＣ言語で利用できます。

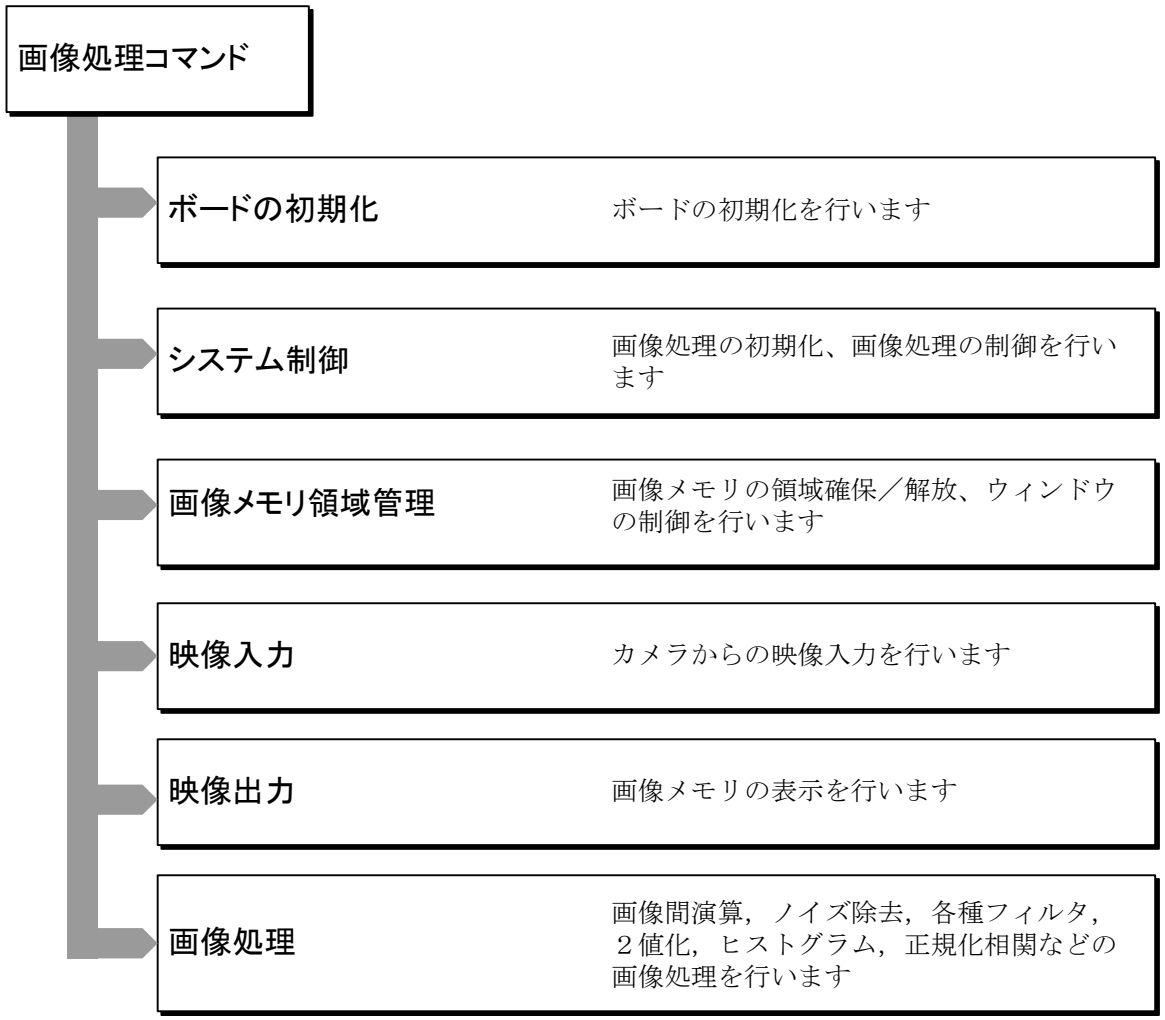


図4-1-1 画像処理コマンドの構成

4.2 ボードの初期化

Ｗｉｎｄｏｗｓパソコンでリモートコマンドから画像認識コマンドを実行する場合、ボードの初期化（ボードのリセット、オンボードシステムファイルのダウンロード、オンボードシステムのブート）を行いオンボードシステムが動作できる状態にする必要があります。以下にその要領を説明します。

4.2.1 デバイスID無しリモートコマンドでのボードの初期化

デバイスＩＤ無しのリモートコマンドで画像認識ボードを使用する場合は、ボード起動（StartIP）を実行し初期化します。そしてアプリケーションの終了時に画像認識ボードの停止処理（StopIP）を実行して下さい。

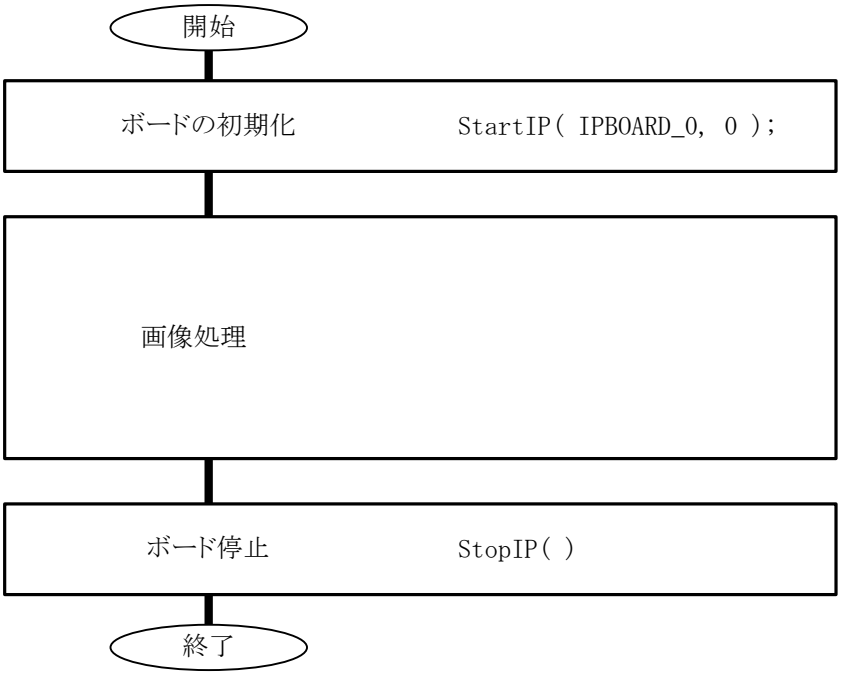


図4-2-1　デバイスＩＤ無しリモートコマンドでのボードの初期化

1台のＰＣに複数のボードを使用する場合は、Ｗｉｎｄｏｗｓのスレッド毎に各ボード番号でStartIP()を実行し、ActiveIP()でボードを切り換えて下さい。

4.2.2 デバイスID付きリモートコマンドでのボードの初期化

デバイスＩＤ付きのリモートコマンドで画像認識ボードを使用する場合は、ボードのオープン処理（OpenIPDev）を実行しデバイスＩＤを取得して下さい。そしてプログラムの終了時に画像認識ボードのクローズ処理（CloseIPDev）を実行して下さい。

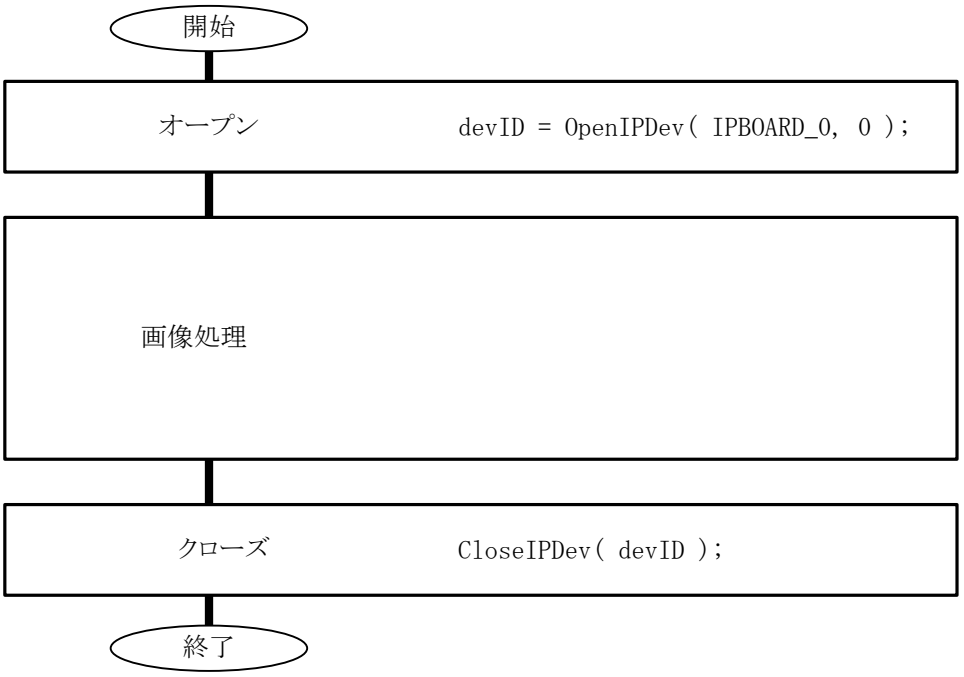


図4-2-2　デバイスＩＤ付きリモートコマンドでのボードの初期化

4.3 システム制御

システム制御コマンドは、システムの初期化やエラー情報読み出し等、システム全体を制御します。

4.3.1 システムの初期化

電源投入時は、画像処理コマンドが初期化されていません。アプリケーションの初期化部でInitIP()コマンドを実行し、画像処理コマンドの初期化を行って下さい。下の図に InitIP()コマンドを実行した時のシステムの初期状態を示します。詳細は、コマンドリファレンスを参照して下さい。

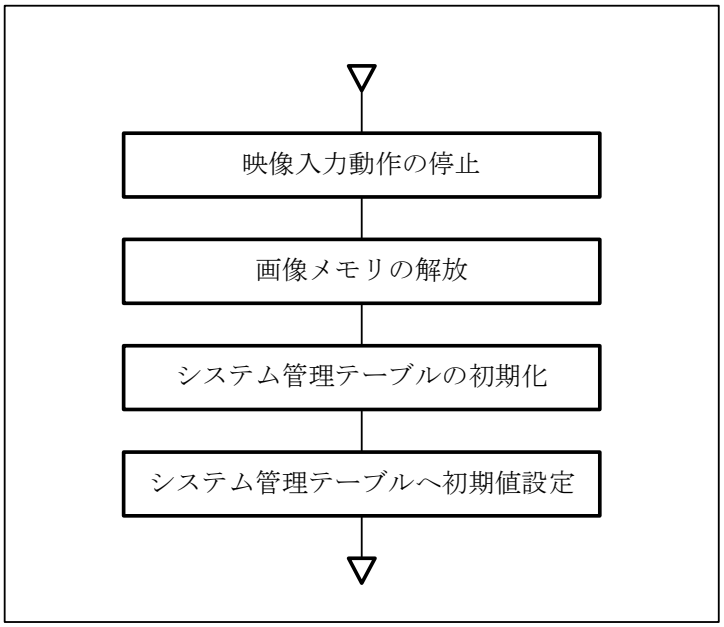


図4-3-1　システムの初期化

『InitIP』コマンドにより初期設定される項目と内容を次に示します。本コマンドでは、有効ビデオポート番号設定、ビデオフレーム設定、カメラタイプの設定、カメラ映像ライブ表示は行いません。

4.3.2 システムの状態遷移図

画像認識ライブラリには、画像処理システム未初期化状態、動作可能状態、エラー状態の３種類の状態が存在します。

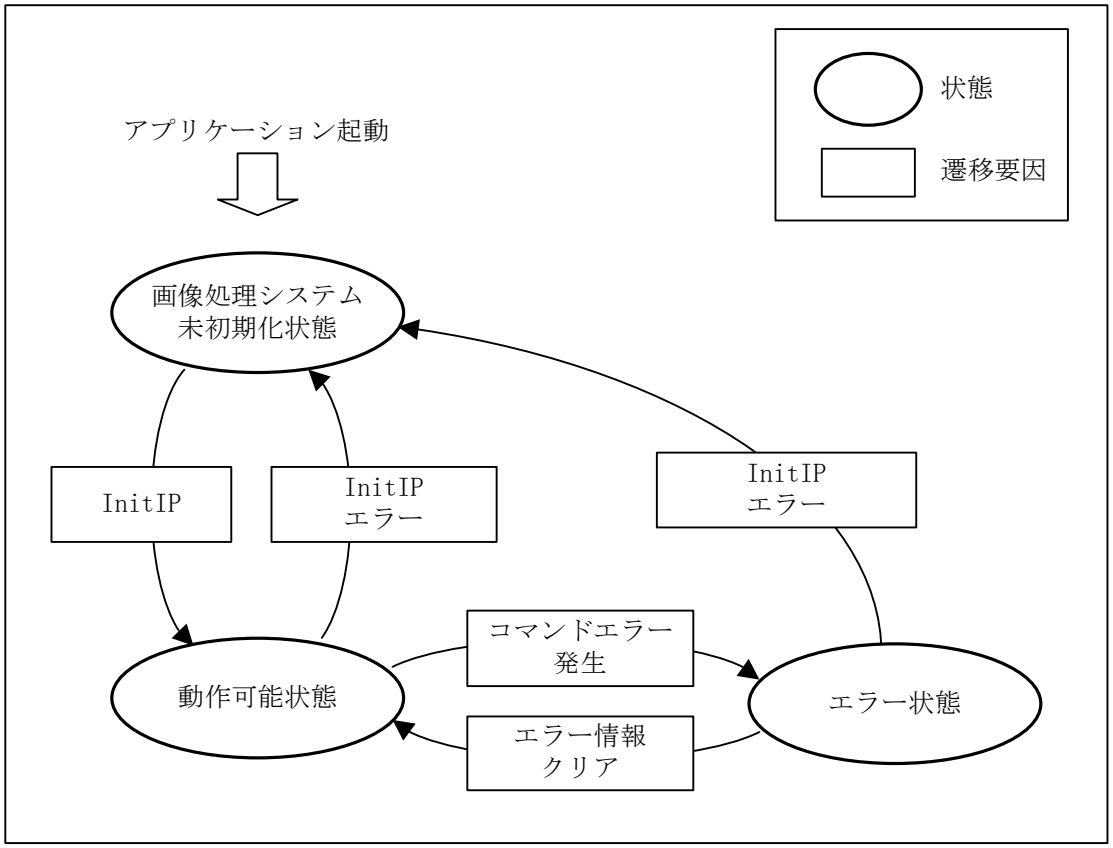


図4-3-2　システムの状態遷移図

4.3.3 エラー情報管理

画像処理コマンドでエラーが発生すると、エラー情報のダイアログボックスが表示され、ＯＫボタンを押すまで処理が中断します。画像処理コマンド以外のコマンドでは、下記のエラー情報管理は行われません。

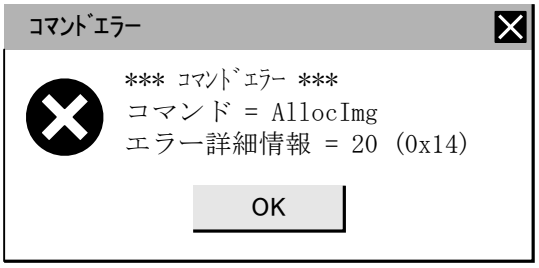


図4-3-3　コマンドエラー処理

このエラー情報のダイアログボックスは『DisableIPErrorMessage』及び『EnableIPErrorMessage』で出力を制御できます。

画像処理コマンドでエラーが発生すると、コマンドのリターンコードにはエラーコードが返されますが、詳細エラー情報はリターンコードに反映されずシステムに登録されます。また、エラー発生後の画像処理は、エラー情報をクリアしない限り全て実行することができません。
そのため、ユーザは『ReadIPErrorTable』コマンドでエラー情報を読み出すか、または『ClearIPError』コマンドでエラー情報をクリアする必要があります※1※2。

以下に画像処理コマンドでエラーが発生した際の処理手順を示します。

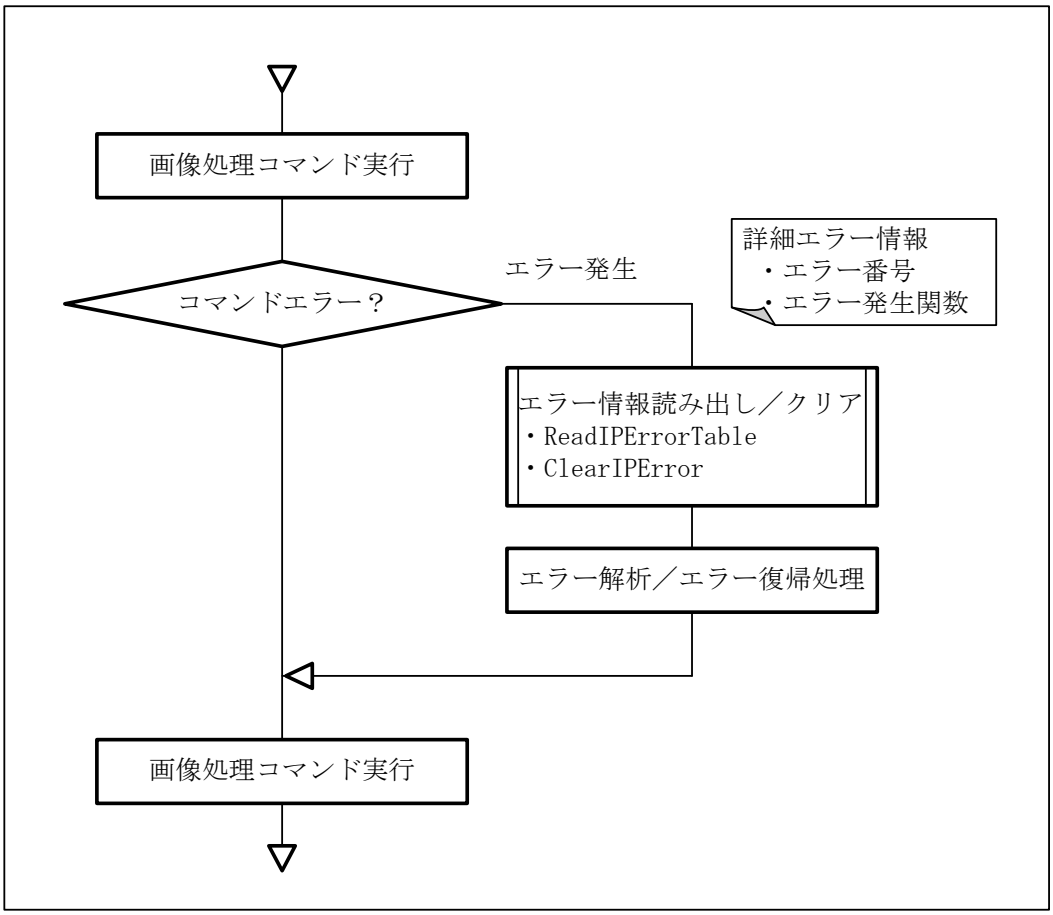


図4-3-4　コマンドエラー処理手順

※1　復帰のためにボードリセットが必要な場合があります。エラーコードについては、コマンドリファレンスの「付録Aエラーコード一覧」を参照して下さい。

※2　リモートコマンドではスレッド毎、オンボードのコマンドではタスク毎に管理されているためスレッド毎、タスク毎にエラー情報読出し及びエラークリアの発行が必要です。

4.4 画像メモリ領域管理

画像メモリを使用するための領域の確保、解放と、ウィンドウの制御を行います。

4.4.1 画像メモリの概要

画像メモリは、カメラ映像の入力データや画像処理演算結果を格納するメモリです。
ＮＶＰ-Ａｘ２３０の画像メモリは、AllocImgの画像メモリ確保コマンドでモノクロ画面を確保します。
システムメモリは２ＧＢで、８Ｂｉｔのモノクロ５００万画素（２５６０×２０４８）で ３００面以上を確保できます。（※）
画像メモリには濃淡画像メモリと２値画像メモリの区別はなく、２値画像として使用するときは、黒は０、白は２５５の値で使用します。

※ 画像メモリはＲＡＭ上に確保され、ＲＡＭ領域からシステムが使用する領域を除いた領域を画像メモリとヒープ領域に使用します。メモリサイズが２ＧＢの場合は約１．７ＧＢ画像メモリとヒープ領域に使用することができます。確保できる画面数はヒープの使用状況により変わりますので注意が必要です。

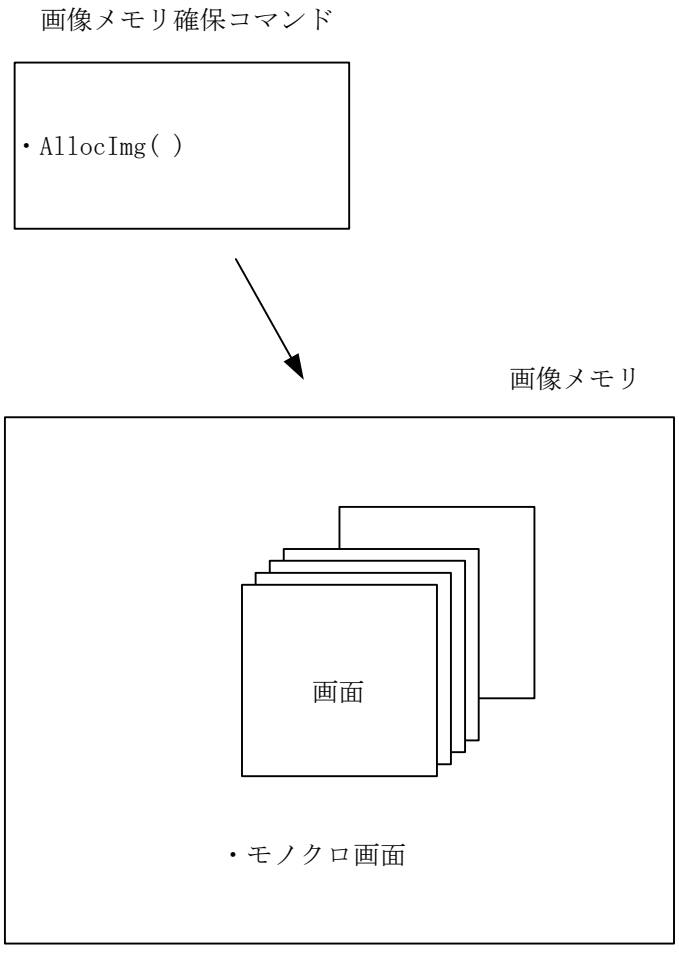
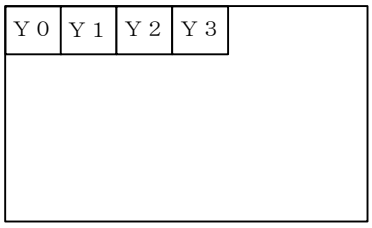
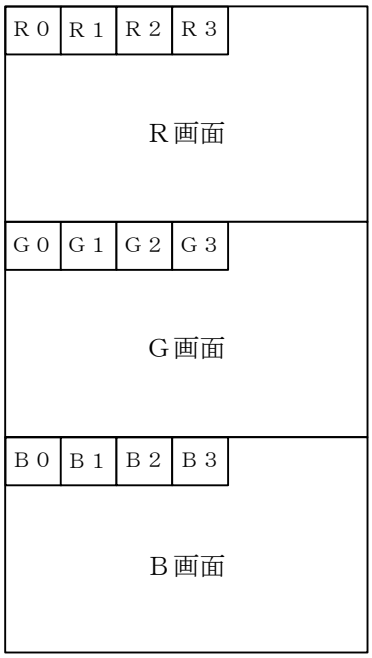
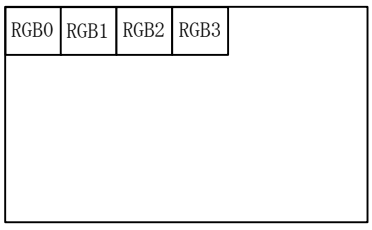
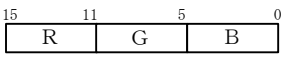


図4-4-1　画像メモリ

4.4.2 画像メモリの画面タイプ

画像処理コマンドでは、モノクロ画面（Ｙ画面）とＲＧＢカラー画面の画像メモリを確保可能です。Ｙ画面は映像入力、画像処理および映像表示に使用できます。コンポーネントＲＧＢ画面は映像入力(※)と画像処理、ＲＧＢ１６画面は映像表示に使用できます。以下に画面タイプの一覧を示します。

表4-4-1　画像メモリの画面タイプ（１）

画面タイプ	フォーマット	画面確保コマンド	詳細
Ｙ	 ８bit/pixel	AllocImg	８bit/pixelの画面。濃淡及び２値の画像処理、モノクロ映像入力表示に使用できます。
コンポーネント ＲＧＢ	 ２４bit/pixel	AllocRGBImg	８bit/pixelのＲ画面、Ｇ画面、Ｂ画面で構成される画面。Ｒ、Ｇ、Ｂの３画面でＲＧＢカラー情報を構成します。ＲＧＢカラーの映像入力(※)に使用できます。映像表示はできません。 Ｒ画面、Ｇ画面、Ｂ画面はそれぞれモノクロ画面として画像処理に使用できます。
ＲＧＢ１６	 １６bit/pixel 	AllocRGB16Img	１６bit/pixelの画面。１６bitデータは上位からＲデータ５bit、Ｇデータ６bit、Ｂデータ５bitで、１画面でＲＧＢカラー情報を構成します。ＲＧＢカラーの映像表示に使用できます。映像入力、画像処理はできません。

※ＲＧＢカラーカメラはカスタム対応です。

4.4.3 画像メモリのサイズ

画像処理に使用できる画面サイズは、以下に示すサイズで画像メモリを確保できます。なお、画像処理ハードウェアの制限により、横幅は１２８バイトで割り切れるサイズを指定する必要があります。
また、１０２４×１０２４を超える大きさの範囲では画像処理ができないコマンドがありますので注意してください（コマンドリファレンス「付録Ｃ画像処理サイズ一覧」参照）。

表4-4-2　確保容量一覧

論理画面サイズ	物理画面サイズ	備考
	Ｙ画面	
256H×256V	←	
256H×512V	←	
512H×256V	←	
512H×512V	←	
640H×256V	←	
640H×512V	←	
768H×512V	←	
768H×576V	←	
1024H×768V	←	
1024H×1024V	←	
1280H×1024V	←	
2560H×1920V	←	
2560H×2048V	←	
2688H×2048V	←	
2688H×2176V	←	

4.4.4 画像メモリの座標系

画面の座標系は、画面の左上隅を原点とした下図のような座標系となります。

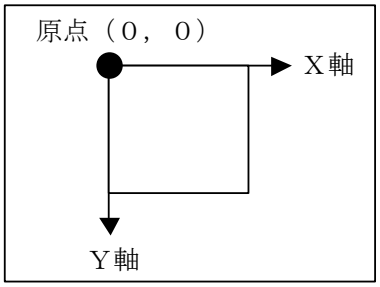


図4-4-2　画像メモリの座標系

4.4.5 画像メモリのデータタイプ

画像メモリに格納されるデータは、濃淡データと２値データの２種類があります。濃淡データは２５６階調を持ち、符号なし８ビット（０～２５５）と符号付８ビット（－１２８～１２７）の２種類があります。基本設定は符号なし８ビットです。
２値データは、黒と白の２階調のデータです。黒は０、白は２５５の値で画像メモリに格納されます。

4.4.6 ウィンドウの概要

ウィンドウとは、処理対象画面、または結果格納画面内の処理領域のことです。画像処理は画像メモリで確保したウィンドウのサイズで行われます。よって、画面のサイズより小さいウィンドウを設定することにより、処理を高速に行うことができます。

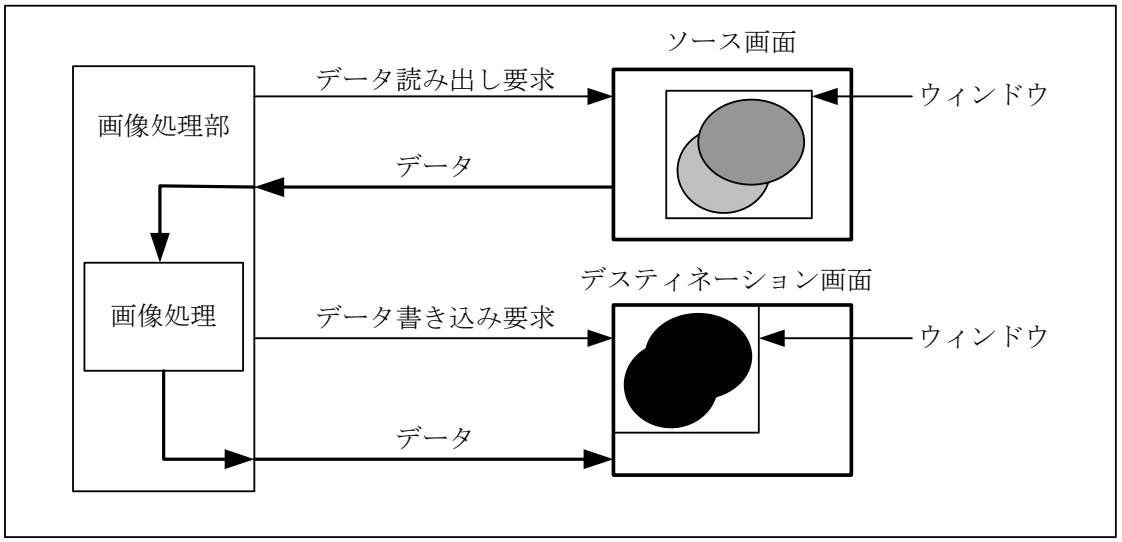


図4-4-3　ウィンドウの役割

4.4.7 ウィンドウの座標系

ウィンドウの座標系は、画面の左上隅を原点とした下図のような座標系となります。ウィンドウ設定値は、画面の原点相対の座標を指定します。また、ヒストグラム等の画像処理はウィンドウで指定した始点を原点として抽出座標を出力します。

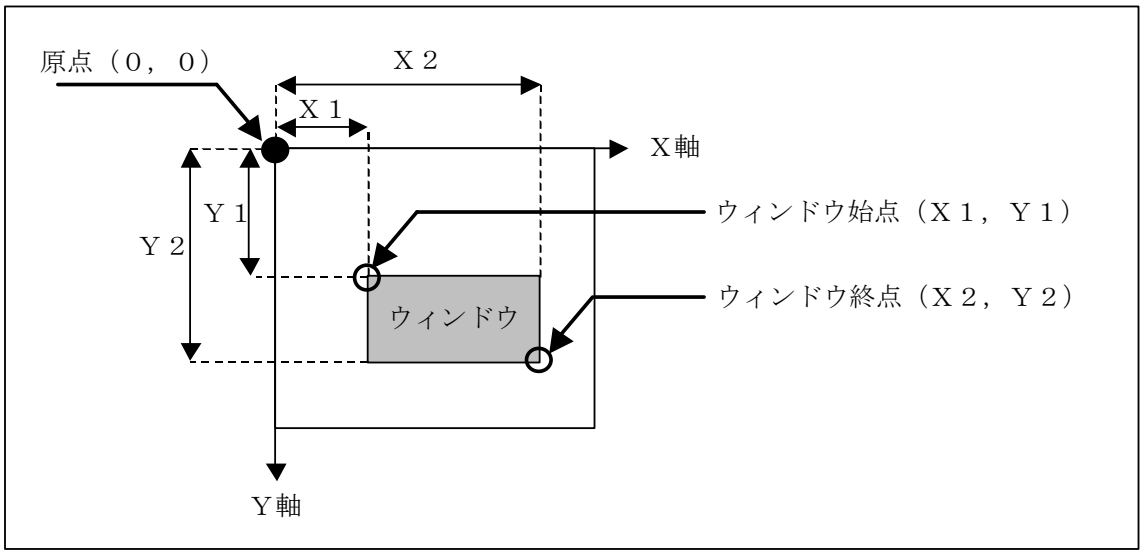


図4-4-4　ウィンドウの座標系

4.4.8 ウィンドウの種類

ウィンドウには、以下６つの種類があります。

表4-4-3　ウィンドウの種類

ウィンドウ種類	用　　途
ソース０画面 (SRC0_WIN)	画像処理でのソース画面に使用します
ソース１画面 (SRC1_WIN)	画像間演算処理でのソース画面に使用します （ソース画面を２画面使用するときの第２ソース画面）
ソース拡張画面 (SRC2_WIN)	画像処理でのソース拡張画面に使用します（未使用）
デスティネーション画面（DST_WIN）	画像処理でのデスティネーション画面に使用します
画像メモリアクセス（SYS_WIN）	画像メモリ制御コマンド（WriteImg, ReadImg, WritePixel,ReadPixel,WritePixelContinue, ReadPixelContinue）で有効なウィンドウ
デスティネーション拡張画面 (DST_EXT_WIN)	画像処理でのデスティネーション拡張画面に使用します （未使用）

ソース画面とデスティネーション画面で異なるサイズのウィンドウを設定した場合、それぞれのウィンドウサイズ内の最小のサイズで処理を行います。

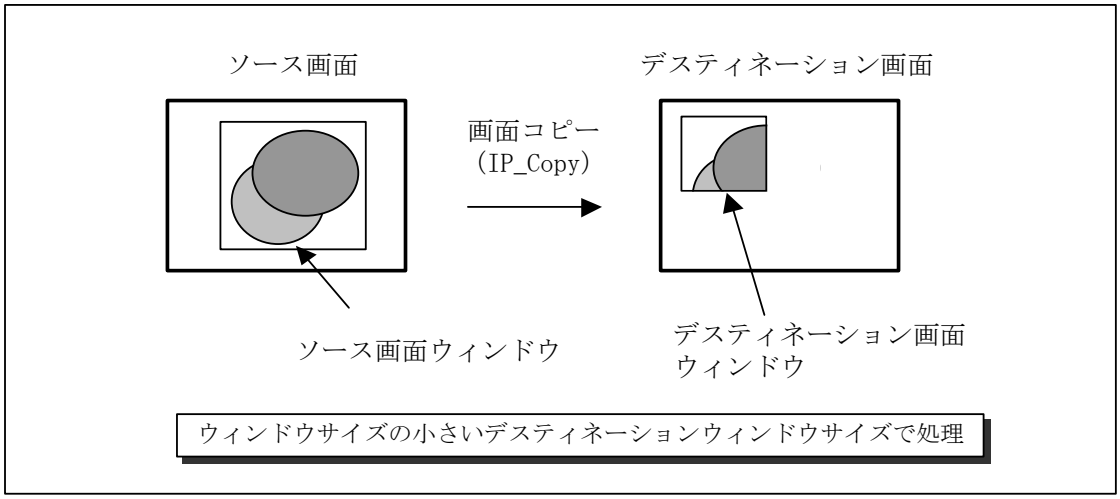


図4-4-5　ウィンドウ処理サイズ

4.4.9 ウィンドウの有効／無効

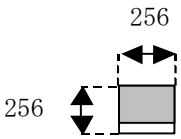
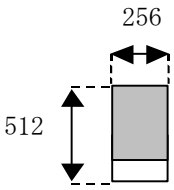
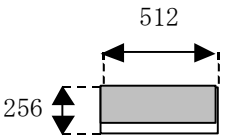
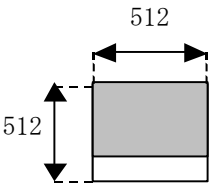
画像処理は、ビデオフレームサイズ、またはウィンドウのサイズで行われます。どちらで動作するかはウィンドウの有効／無効状態で決まり、ウィンドウが無効のときはビデオフレームサイズ、有効のときは設定したウィンドウサイズとなります。この設定は、EnableIPWindowおよびDisableIPWindowコマンドによって行います。

ウィンドウ無効は、DisableIPWindowコマンド、有効はEnableIPWindowコマンドで行います。

ビデオフレームサイズとは、カメラから取り込む映像サイズのことであり、SetVideoFrame コマンドで現在設定されている値です。

下の表に、ウィンドウ無効時のウィンドウサイズの例を示します。

表4-4-4　ウィンドウ無効時のウィンドウサイズ例

画面サイズ	２５６×２５６	２５６×５１２	５１２×２５６	５１２×５１２
ビデオ フレーム サイズ	２５６×２２０	２５６×４４０	５１２×２２０	５１２×４４０
□ 画面サイズ ■ ウィンドウ サイズ				

4.4.10 データタイプの属性

本ライブラリでは、映像入出力、画像処理等の処理データを以下のいずれかのタイプであるとみなして処理します。

表4-4-5　データタイプ

データタイプ		値の範囲	オーバーフロー／アンダーフロー
符号なし ８ビット	UNSIGN8_DATA	０～２５５	処理結果がアンダーフローの場合、０（０ｘ００） 処理結果がオーバーフローの場合、２５５（０ｘＦＦ）
符号付 ８ビット	SIGN8_DATA	－１２８ ～１２７	処理結果がアンダーフローの場合、－１２８（０ｘ８０） 処理結果がオーバーフローの場合、１２７（０ｘ７Ｆ）
２値	BINARY_DATA	０または ２５５	なし

※　デフォルト設定は、符号なし８ビット

また、データタイプの種類として、システムデータタイプと画面データタイプの２種類があります。

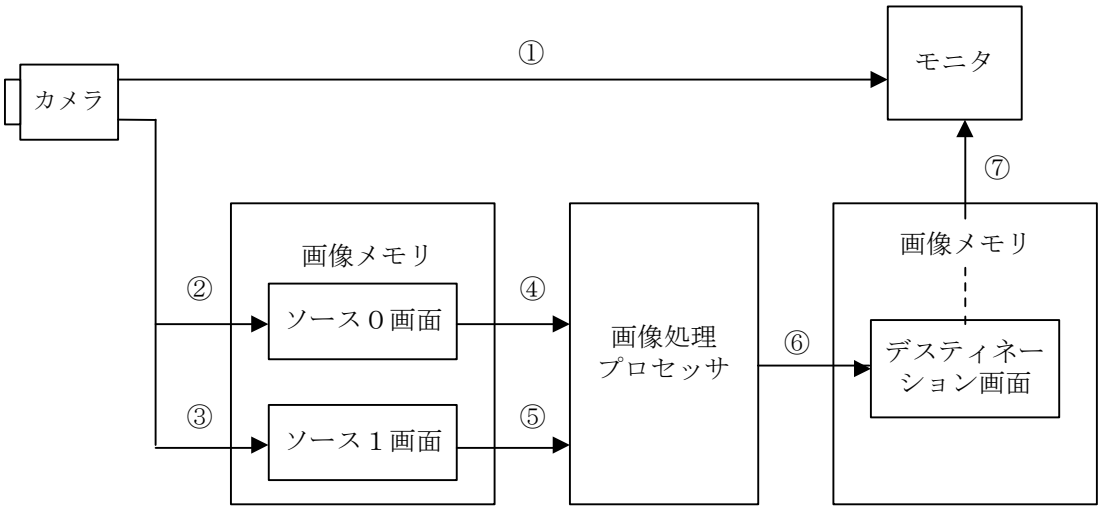
（1）システムデータタイプ

システムとしてのデフォルト設定のデータタイプ。カメラからの映像取り込み、画像処理結果のデスティネーション画面データタイプの設定等に使用されます。
システムデータタイプは、システムイニシャライズ時に符号なし８ビットに設定されますが、SetIPDataType コマンドで変更することも可能です。

(2) 画面データタイプ

画面ごとに設定されているデータタイプで、その画面のデータがどのタイプ（符号付８ビット、符号なし８ビット、２値）であるかを示します。
AllocImgコマンドで確保された画面の画面データタイプは、符号なし８ビットに設定されます。
画面データタイプは、ChangeImgDataType コマンドで変更できます。

以下に、各処理において使用されるデータタイプについて示します。
また、画像処理実行後の処理結果のデータタイプは、コマンドの出力属性により変更されます。
デスティネーション画面のデータタイプを次の表に示します。



①	カメラ映像の表示（DispCamera／DispOverlapコマンド） ・データタイプに依存しません。
②、③	カメラ映像の取り込み（GetCameraコマンド） ・システムデータタイプで処理されます。ただし、システムデータタイプが２値に設定されていた場合は、符号なし８ビットで処理します。
④、⑤	画像処理の際のソース画面のデータタイプ ・ソース画面の画面データタイプで処理します。
⑥	処理結果のデータタイプ ・使用したコマンドによって異なります。また、デスティネーション画面のデータタイプは、コマンドの出力属性により変更されます。
⑦	画面表示のデータタイプ（DispImgコマンド） ・データタイプに依存しません。

図4-4-6　処理の流れとデータタイプ

表4-4-6 (1)　デスティネーション画面のデータタイプの例
（詳細はコマンドリファレンスを参照ください）

コマンド名	デスティネーション画面データタイプ
IP_ClearAllImg	システムデータタイプ
IP_ClearImg	システムデータタイプ　※１
IP_Const	システムデータタイプ
IP_Copy	ソース０画面データタイプ
IP_Zoom	ソース０画面データタイプ
IP_ZoomExt	ソース０画面データタイプ
IP_ZoomOut	ソース０画面データタイプ
IP_ZoomOutExt	ソース０画面データタイプ
IP_Shift	ソース０画面データタイプ
IP_ZoomS	ソース０画面データタイプ
IP_Rotate	ソース０画面データタイプ
IP_ZoomIn	ソース０画面データタイプ
IP_ZoomInExt	ソース０画面データタイプ
IP_Binarize	２値
IP_BinarizeExt	２値
IP_Invert	システムデータタイプ　※２
IP_Minus	システムデータタイプ
IP_Abs	システムデータタイプ
IP_AddConst	システムデータタイプ
IP_SubCons	システムデータタイプ
IP_SubConstAbs	システムデータタイプ
IP_MultConst	システムデータタイプ
IP_MinConst	システムデータタイプ
IP_MaxConst	システムデータタイプ
IP_ConvertLUT	ソース０画面データタイプ
IP_ShiftDown	ソース０画面データタイプ
IP_ShiftUp	ソース０画面データタイプ
IP_Add	システムデータタイプ
IP_Sub	システムデータタイプ
IP_SubAbs	システムデータタイプ
IP_Comb	システムデータタイプ
IP_CombAbs	システムデータタイプ
IP_Mult	システムデータタイプ
IP_Average	システムデータタイプ
IP_Min	システムデータタイプ
IP_Max	システムデータタイプ
IP_CombDrop	システムデータタイプ
IP_And	ソース０画面データタイプ　※３
IP_Or	ソース０画面データタイプ　※３
IP_Xor	ソース０画面データタイプ　※３
IP_InvertAnd	ソース０画面データタイプ　※３
IP_InvertOr	ソース０画面データタイプ　※３
IP_Xnor	ソース０画面データタイプ　※３

※１　ＲＧＢ__Ｇ、ＲＧＢ__Ｂ画面は符号無し８ビット
※２　ソース０画面が２値の場合、２値、それ以外はシステムデータタイプ
※３　ソース０画面が２値の場合、ソース１画面データタイプ、それ以外はソース０画面データタイプ

表4-4-6 (2)　デスティネーション画面のデータタイプの例
(詳細はコマンドリファレンスを参照ください)

コマンド名	デスティネーション画面データタイプ
IP_PickNoise4	２値
IP_PickNoise8	２値
IP_Outline4	２値
IP_Outline8	２値
IP_Dilation4	２値
IP_Dilation8	２値
IP_Erosion4	２値
IP_Erosion8	２値
IP_Thin4	２値
IP_Thin8	２値
IP_Shrink4	２値
IP_Shrink8	２値
IP_SmoothFLT	システムデータタイプ
IP_EdgeFLT	システムデータタイプ
IP_EdgeFLTAbs	システムデータタイプ
IP_Lapl4FLT	システムデータタイプ
IP_Lapl8FLT	システムデータタイプ
IP_Lapl4FLTAbs	システムデータタイプ
IP_Lapl8FLTAbs	システムデータタイプ
IP_LineFLT	システムデータタイプ
IP_LineFLTAbs	システムデータタイプ
IP_MinFLT	システムデータタイプ
IP_MinFLT4	システムデータタイプ
IP_MinFLT8	システムデータタイプ
IP_MaxFLT	システムデータタイプ
IP_MaxFLT4	システムデータタイプ
IP_MaxFLT8	システムデータタイプ
IP_LineMinFLT	システムデータタイプ
IP_LineMaxFLT	システムデータタイプ
IP_RankFLT	システムデータタイプ
IP_Rank4FLT	システムデータタイプ
IP_Rank8FLT	システムデータタイプ
IP_MedFLT	システムデータタイプ
IP_Med4FLT	システムデータタイプ
IP_Med8FLT	システムデータタイプ
IP_Label4	符号なし８ビット
IP_Label8	符号なし８ビット
IP_Label4withAreaFLT	符号なし８ビット
IP_Label8withAreaFLT	符号なし８ビット
IP_Label4withAreaFLTSort	符号なし８ビット
IP_Label8withAreaFLTSort	符号なし８ビット
IP_TrsPipelineFLT	２値
IP_BinMatchFLT	システムデータタイプ

4.5 映像入力

4.5.1 NVP－Ax230での映像入力の概要

ＮＶＰ－Ａｘ２３０ＣＬ/２３５ＣＬは、カメラリンクカメラの映像入力をサポートします。カメラのインタフェースはＮＶＰ－Ａｘ２３０ＣＬが２ＣＨ、ＮＶＰ－Ａｘ２３５ＣＬが４ＣＨです。

画像処理コマンドは、以下の映像入力をサポートします。ただし、カメラによっては実現できない機能がありますので、本マニュアルの内容をご確認ください。

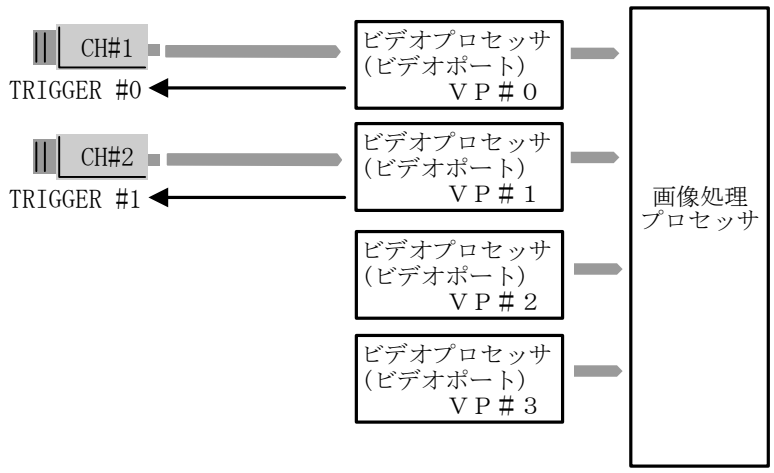
・間引き映像入力
・複数カメラ同時入力
・キャプチャモード
・プリフェッチ映像入力
・ストロボ映像入力
・パーシャル映像入力

ＲＧＢカラー映像入力については４．７ ＲＧＢカラー処理機能を参照してください。

4.5.2 ビデオポートとカメラポートの選択

ＮＶＰ－Ａｘ２３５ＣＬ搭載の画像処理プロセッサにはビデオポートが４ポートあり、それぞれ独立した動作が可能です。また、ビデオポート毎にカメラポートが１ポートあり、最大で４台のカメラが接続できます。そのため、カメラによって２カメラ～４カメラの同時入力が可能です。
ＮＶＰ－Ａｘ２３０ＣＬは、ビデオポートが４ポートあり、最大で２台のカメラが接続できます。カメラによっては２カメラの同時入力が可能です。
ビデオポートの切り替えはActiveVideoPortコマンドで行います。カメラポートはSelectCameraコマンドで行いますが１ポート固定となります。

(a)ＮＶＰ－Ａｘ２３０ＣＬ



(b)ＮＶＰ－Ａｘ２３５ＣＬ

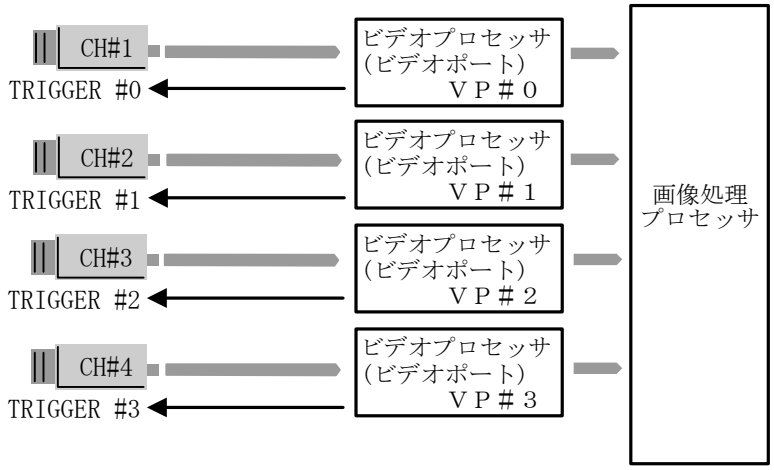


図4-5-1　映像入力ハードウェア構成

4.5.3 サポートカメラとカメラの選択

ＮＶＰ－Ａｘ２３０ＣＬ／２３５ＣＬがサポートするカメラはカメラリンクカメラです。カメラは以下の表のいづれかに分類されます。表に示しましたように、カメラの属性によっては間引き入力や同時入力ができませんので、カメラ選定の際にご確認ください。具体的なカメラとそのカメラ属性については、コマンドリファレンスのSelectCameraを参照ください。

画像処理コマンドでは、SelectCameraコマンドでカメラの選択と同時入力の有無、SetVideoFrameコマンドで間引き入力を設定します。

表4-5-1(a)　カメラリンクＴＡＰ、ＶＩＮモードと映像入力

○：可、×：不可

	カメラリンク TAPモード	最大映像入力 サイズ	クロック	間引き
1	1TAP	4096 x 2048	85MHz以下	×
2	2TAP(Single)	4096 x 2048	50MHz以下	×
3	2TAP(Dual)	4096 x 4096	85MHz以下	×
4	4TAP/8TAP ※1	4096 x 8192	85MHz以下 (4TAP) 50MHz以下 (8TAP)	×
5	RGBカラー	4096 x 2048	50MHz以下	×

表4-5-1(b)　カメラリンクTAPモードと同時入力可否

○：可、×：不可

	カメラリンク TAPモード ※2	1カメラ	2カメラ同時入力		3カメラ 同時入力 CH1～CH3	4カメラ 同時入力 CH1～CH4
			CH1&CH2 CH3&CH4	CH1&CH3 CH2&CH4		
1	1TAP	○	○	○	○	○
2	2TAP(Single)	○	○	○	○	○
3	2TAP(Dual)	○ ※5	○ ※4	×	×	×
4	4TAP/8TAP ※1	×	×	×	×	×
5	RGBカラー ※3	○	×	×	×	×

※１ 未サポート
※２ Ａｘ２３０ＣＬの場合は、ＣＨ３、ＣＨ４は使用できません。したがって２カメラ同時のＣＨ１＆ＣＨ２を除く組み合わせおよび３カメラ、４カメラ同時入力はできません。
※３ ＣＨ１のみ使用が可能です。その際ＣＨ２～ＣＨ４は使用できません。
※４ ＣＨ１＆ＣＨ２の同時入力のみ可能です。その際ＣＨ３とＣＨ４は使用できません。
※５ ＣＨ１で使用した場合、ＣＨ３は映像入力できません。ＣＨ２で使用する場合、ＣＨ４は映像入力できません。

4.5.4 キャプチャモード

キャプチャモードは、単一キャプチャモード及び連続キャプチャモードをサポートします。
走査方式はノンインタレースモードです。
また、画像処理コマンドではプリフェッチ映像入力をサポートしますが、プリフェッチ映像入力は連続キャプチャモードで動作し、プリフェッチ映像入力以外は単一キャプチャモードで動作します。
プリフェッチ映像入力については4.5.6プリフェッチ映像入力を参照してください。

表4-5-2　カメラの走査方式とキャプチャモード

No.	走査方式	キャプチャモード		備考
		単一	連続	
1	ノンインタレースモード	○	○	

(1) 単一キャプチャモード

単一キャプチャモードは、１フィールドまたは１フレームのみＣＰＵバスが使用され、それ以外の場合はＣＰＵバスが使用されません。バスの占有率を考えると効率が良いといえます。
単一キャプチャモードの場合、『GetCamera』コマンドで映像入力を行います。

(2) 連続キャプチャモード

連続キャプチャモードは、キャプチャを開始すると停止するまでＣＰＵバスが使用され、アプリケーションで映像入力の必要がない期間もキャプチャを行っているために、バスの占有率を考えると効率が良くないといえます。
連続キャプチャモードでは、常に映像が入力され、入力データをバッファリングすることができるので、取込開始をかけたフレームまたはフィールドのデータを取込データにすることができ、アプリケーションの処理時間を単一キャプチャモードに比べて短縮できます。
画像処理コマンドでは、『CaptureContinuous』で連続キャプチャを有効にし、『StopCaptureContinuous』で連続キャプチャを停止します。
連続キャプチャモードの場合、『GetCamera』、『GetCameraReqScene』、『GetCameraResScene』コマンドで映像入力を行います。

4.5.5 ビデオフレームサイズ

ビデオフレームサイズはカメラ映像を入力する際の解像度のことで、デフォルトサイズは５１２×４８０に設定されています。

ビデオフレームサイズはSetVideoFrame()コマンドで設定します。使用するカメラの取得可能なサイズを超えない様に設定してください。

また、カメラ映像を入力する画像メモリの大きさは、そのビデオフレームサイズ以上の大きさである必要があります。ビデオフレームサイズに合わせて画像メモリを確保して下さい。

4.5.6 プリフェッチ映像入力

通常の画像処理アプリケーションでは、映像入力を行い、その入力した映像に対し画像処理を実行するため、映像入力と画像処理を順番に行っています。ビデオレートで画像処理アプリケーションを実行させるには、映像入力と画像処理をパイプラインで実行する必要があります。

プリフェッチ映像入力は、画像先行取得を行うことであり、映像入力と画像処理をパイプライン処理し、全体的な処理速度の向上を図ります。

以下にプリフェッチ映像入力による画像処理の例を示します。

(a) プリフェッチ映像入力(画像先行取得)をしない場合

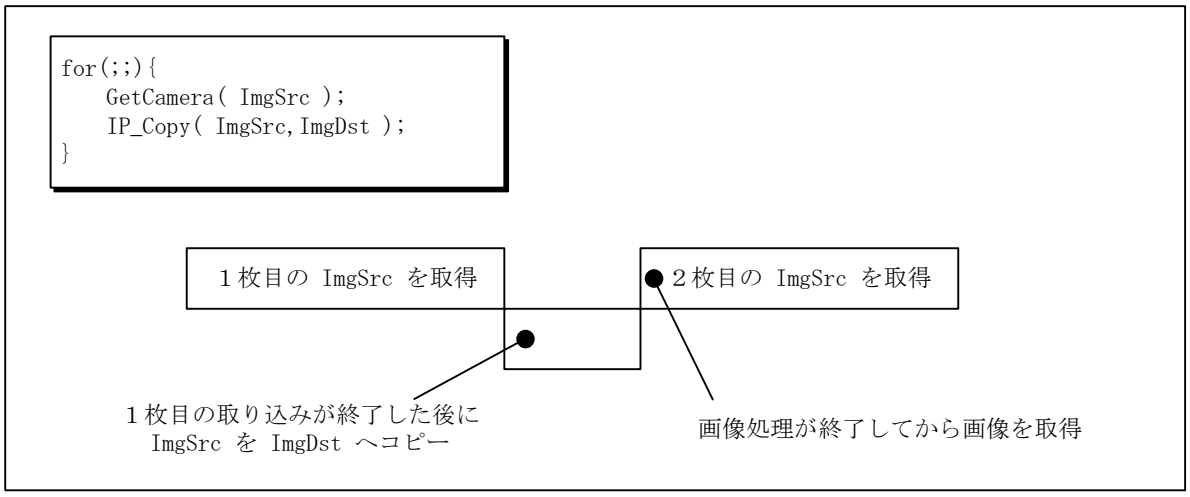


図4-5-2　通常の画像処理アプリケーション例

(b) プリフェッチ映像入力(画像先行取得)をした場合

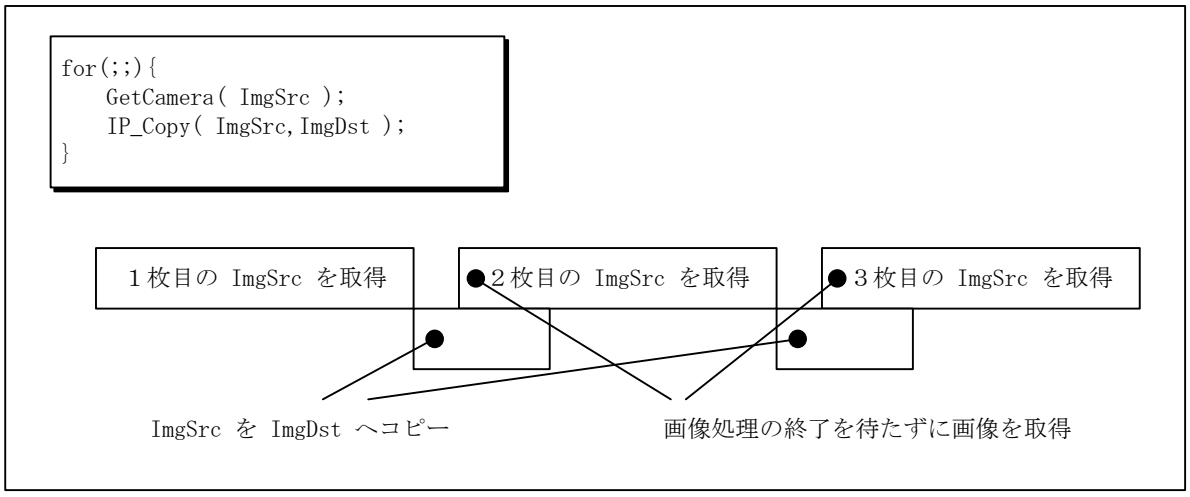


図4-5-3　プリフェッチ映像入力を行う画像処理アプリケーション例

(1) 画像処理コマンドでの連続映像入力

連続映像入力モードでは、常に映像のシーン番号を管理しています。そのため映像のシーン番号を指定して入力したり、入力した映像のシーン番号を取得することができます。画像処理コマンドでは、以下のコマンドで連続映像入力を行います。

表4-5-3　連続映像入力コマンド

No.	コマンド名	機能
1	GetCamera	最新のフレームを入力します。コマンド実行時、映像入力が終了している最新のフレームを入力します。
2	GetCameraReqScene	指定されたシーン番号のフレームを入力します。コマンド実行時、指定されたシーン番号のフレームの映像入力が終了していない場合は、指定されたシーン番号のフレームの映像入力が終了するまでウェイトします。
3	GetCameraResScene	最新のフレームを入力し、そのフレームのシーン番号を返します。コマンド実行時、映像入力が終了している最新のフレームを入力します。

映像のシーン番号は内部カウンタで管理していますが、そのカウンタ値は、シーン番号カウンタ取得コマンド(GetSceneCounter)、シーン番号カウンタ設定コマンド(SetSceneCounter)で任意に設定し、利用することができます。

連続映像入力での映像フレームの画像メモリへの転送とそれに対する画像処理可能フレームのタイムチャートを下図に示します。連続映像入力では、常に３画面の画像メモリバッファを切り替えながら映像入力しているため、常に映像転送中の画面の前２画面の映像を保持しています。映像フレームがシーン番号ｎのタイミングで映像入力コマンド(GetCameraResScene)を実行すると取得シーン番号は（ｎ－１）を返します。このとき、フレーム番号指定の映像入力(GetCameraReqScene)は（ｎ－２）～３２ｂｉｔの範囲で有効になります。

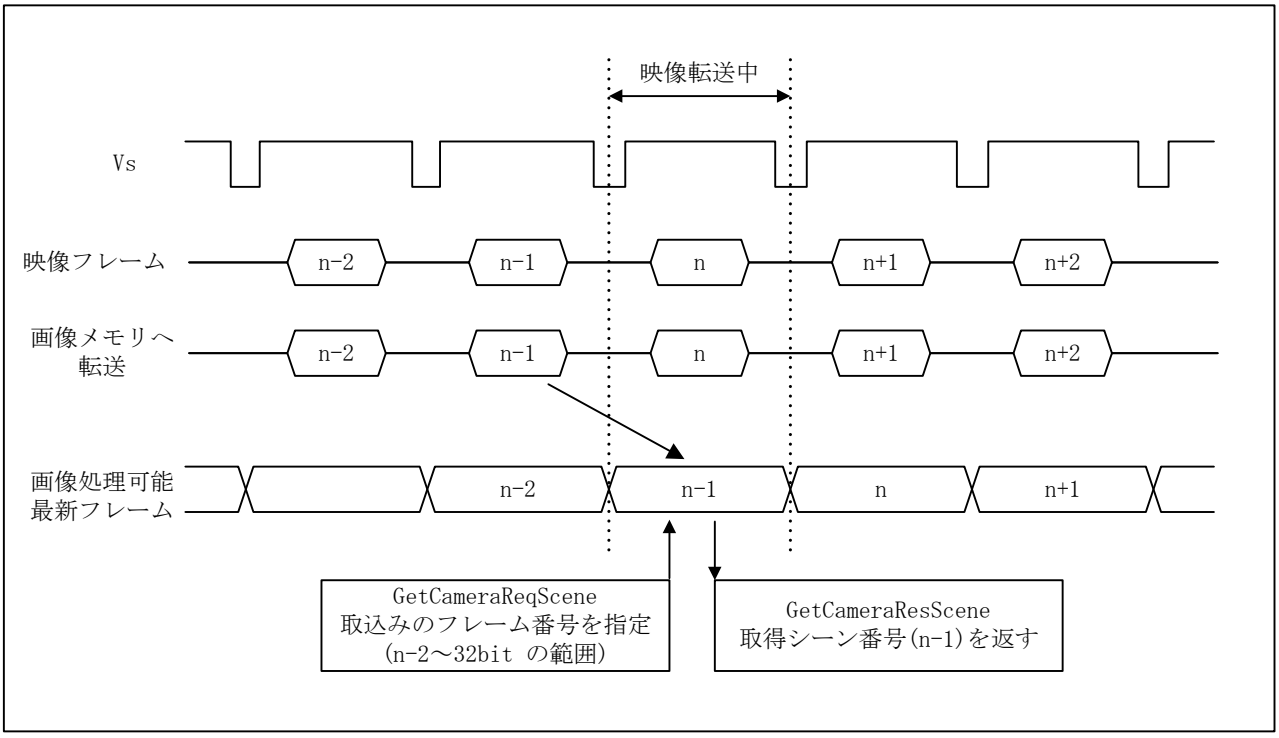


図4-5-4　連続映像入力のタイムチャート

(2) 画像処理コマンドでの連続映像入力方法

画像処理コマンドでは、CaptureContinuousコマンドで連続映像入力モードの設定と起動を行ってから、StopCaptureContinuousコマンドで連続映像入力を終了するまで、映像入力コマンドでの映像入力はすべてプリフェッチ映像入力が行われます。以下にフローを示します。

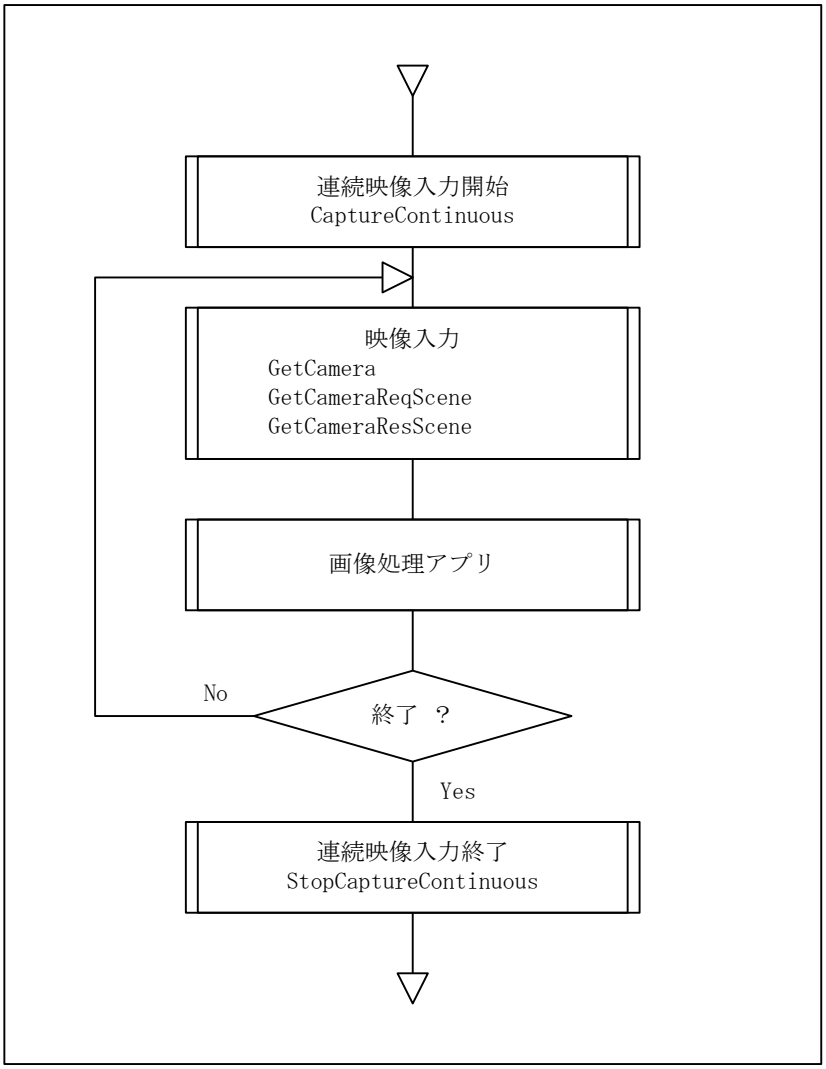


図4-5-5　プリフェッチ映像入力フロー

4.5.7 カメラ映像の入力方法

(1) 基本のカメラ映像入力

基本のカメラ映像入力は以下の順序で実行します。ActiveVideoPortコマンド、SelectCameraコマンド、SetVideoFrameコマンドの実行順序が異なるとエラーになります。

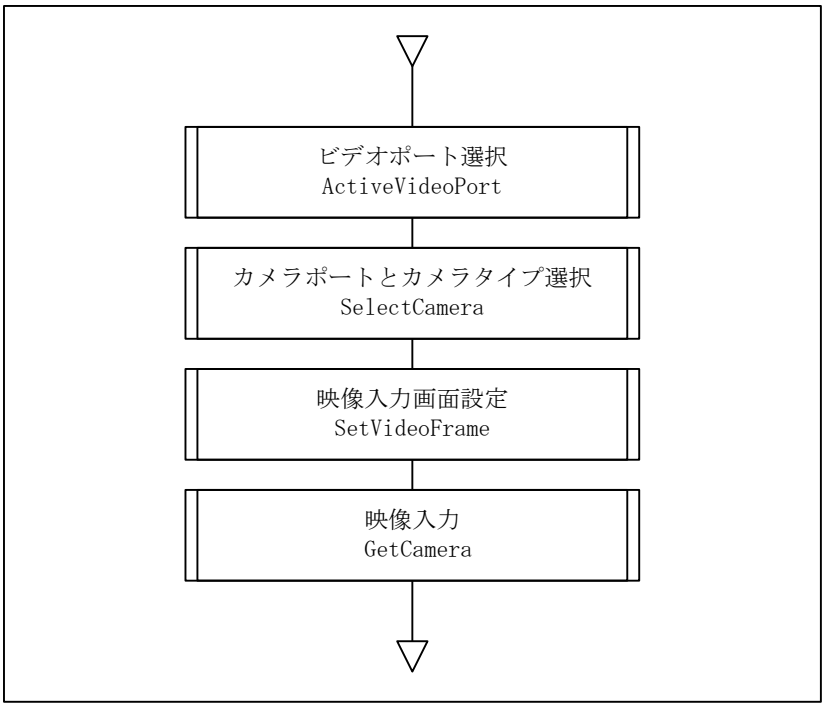


図4-5-6　基本の映像入力フロー

(2) フレームシャッタを使用するカメラ映像入力

フレームシャッタを使用する場合のカメラ映像入力は、以下の順序で実行します。基本の映像入力の順序でコマンドを実行し、SetTriggerModeコマンドでトリガモードの設定とシャッタスピードの設定を行います。

トリガモードを解除する場合には、SetTriggerModeコマンドでノーマルモードを設定するか、再度SelectCameraコマンドとSetVideoFrameコマンドを実行します。

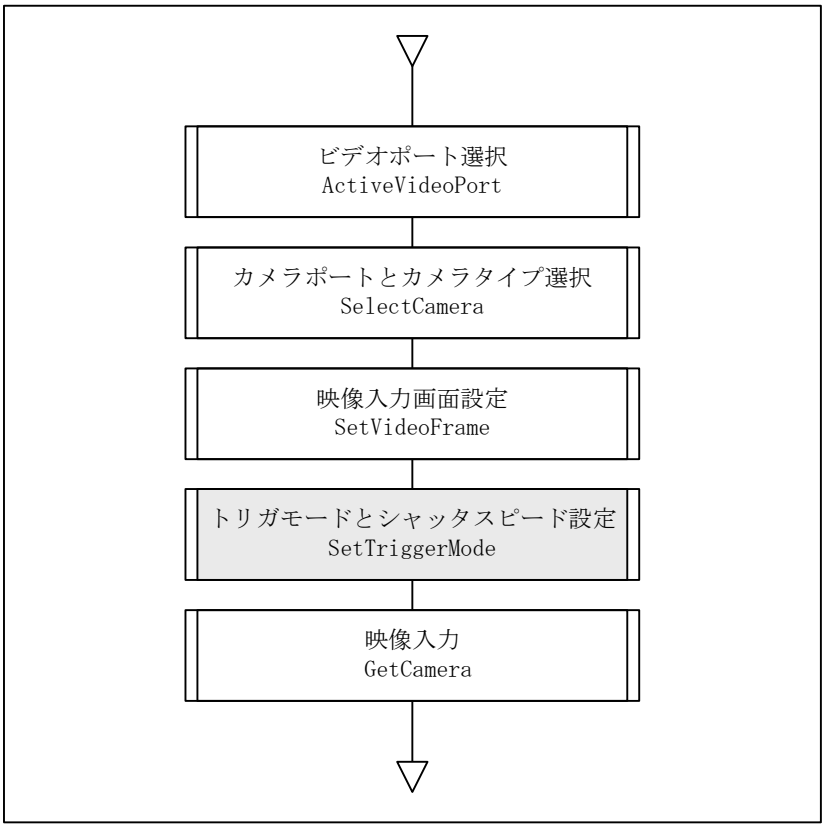


図4-5-7　フレームシャッタを使用する場合の映像入力フロー

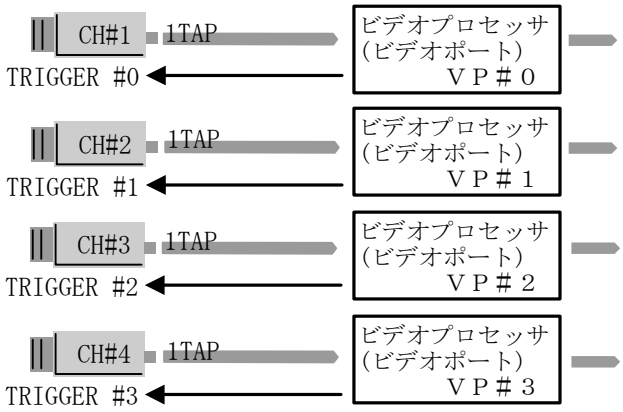
(3) 複数カメラ同時入力

ＮＶＰ－Ａｘ２３０ＣＬは独立した４つのビデオポートと２つのカメラチャネルがあります。ＮＶＰ－Ａｘ２３５ＣＬは独立した４つのビデオポートと４つのカメラチャネルがあります。
ビデオポート＃０はカメラチャネルＣＨ１、ビデオポート＃１はカメラチャネルＣＨ２にそれぞれ対応します。

カメラの同時入力は、ＮＶＰ－Ａｘ２３０ＣＬ／２３５ＣＬのボード同期による映像入力機能で実現します。ＮＶＰ－Ａｘ２３０ＣＬの同時入力は２カメラ同時、ＮＶＰ－Ａｘ２３５ＣＬの同時入力は最大４カメラ同時まで対応可能です。カメラの同時入力は同じタイプのカメラで可能です。

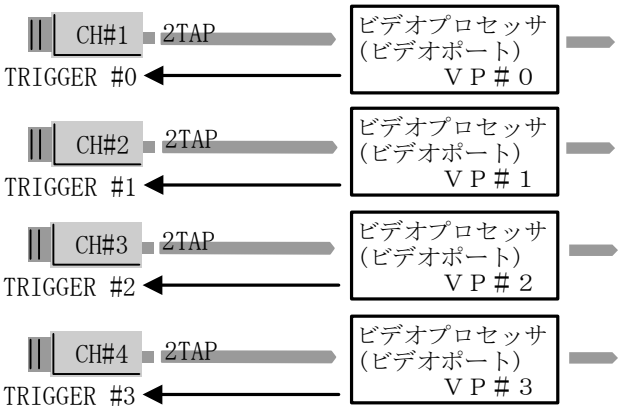
複数カメラ同時入力時のＴＡＰモードを以下に示します。モードにより同時入力機能が制限されます。詳しくは、4.5.3 サポートカメラとカメラの選択 を参照してください。

(a) １ＴＡＰ



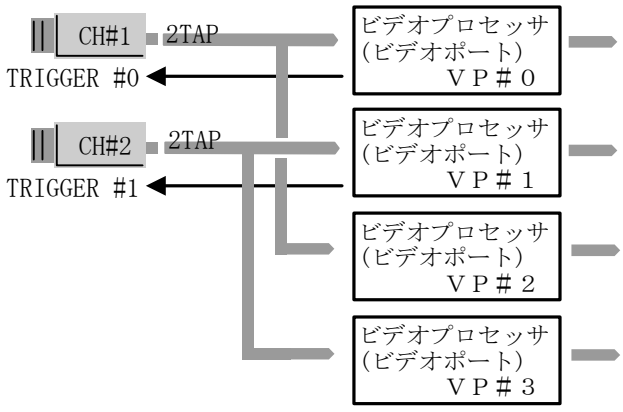
※ NVP-230CLは2台までカメラ接続可
※ NVP-235CLは4台までカメラ接続可

(b) ２ＴＡＰ(Ｓｉｎｇｌｅ)



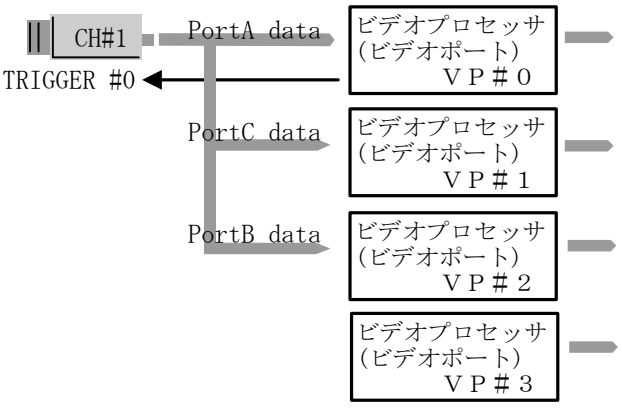
※ NVP-230CLは2台までカメラ接続可
※ NVP-235CLは4台までカメラ接続可

(c) ２ＴＡＰ(Ｄｕａｌ)



※ NVP-230CL,NVP-235CLは2台までカメラ接続可

(d) ＲＧＢ



※ NVP-230CL,NVP-235CLは1台までカメラ接続可

図4-5-8　複数カメラ同時入力のＴＡＰモード

カメラの同時入力は、以下の順序で実行します。フレームシャッタを使用する場合と同じですが、同時入力情報をSelectCameraコマンドで設定します。
カメラの同時入力を解除する場合には、１カメラ入力に設定してSelectCameraコマンドとSetVideoFrameコマンドを実行します。

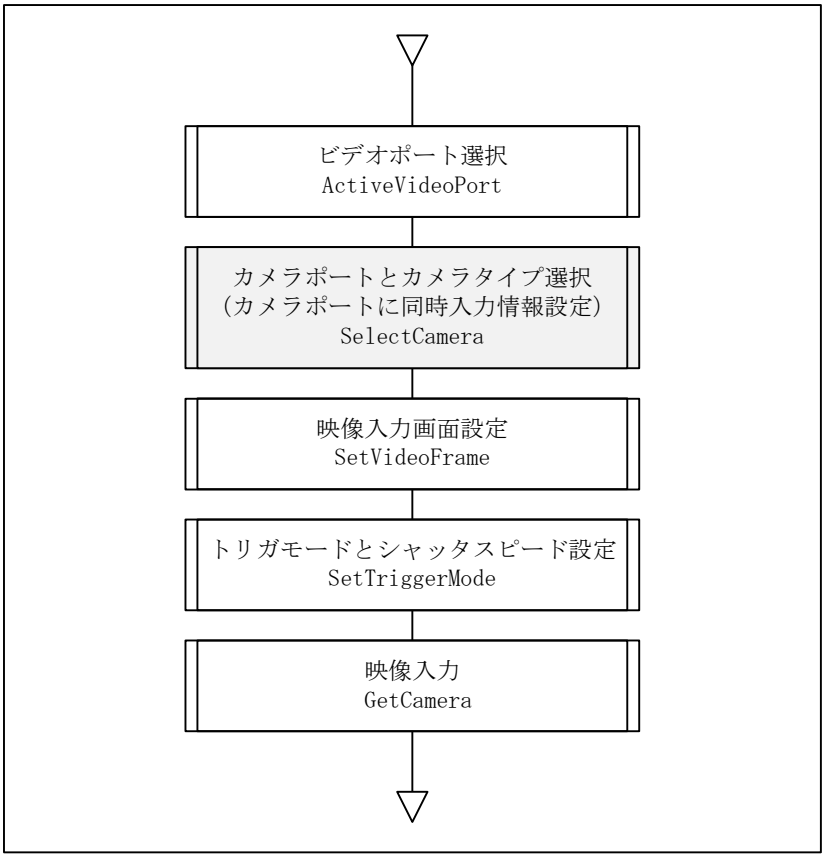


図4-5-9　カメラ同時入力時の映像入力フロー

(4) カメラ映像入力位置の調整

カメラ映像入力を行った際の映像は、画面中央に入力するようにカメラタイプ毎にデフォルト値が設定されています。この映像入力位置は、現位置からのオフセットを設定することで微調整することができます。

映像入力位置の微調整は、SetVFDelayコマンドで水平／垂直方向の遅延サイズを設定します。画面中央の位置に戻す場合は、水平／垂直方向の遅延サイズを０に設定するか、再度、SetVideoFrameコマンドを実行します。

また、フレームシャッタを使用する場合は、SetTriggerModeコマンドを実行してからSetVFDelayコマンドを実行してください。

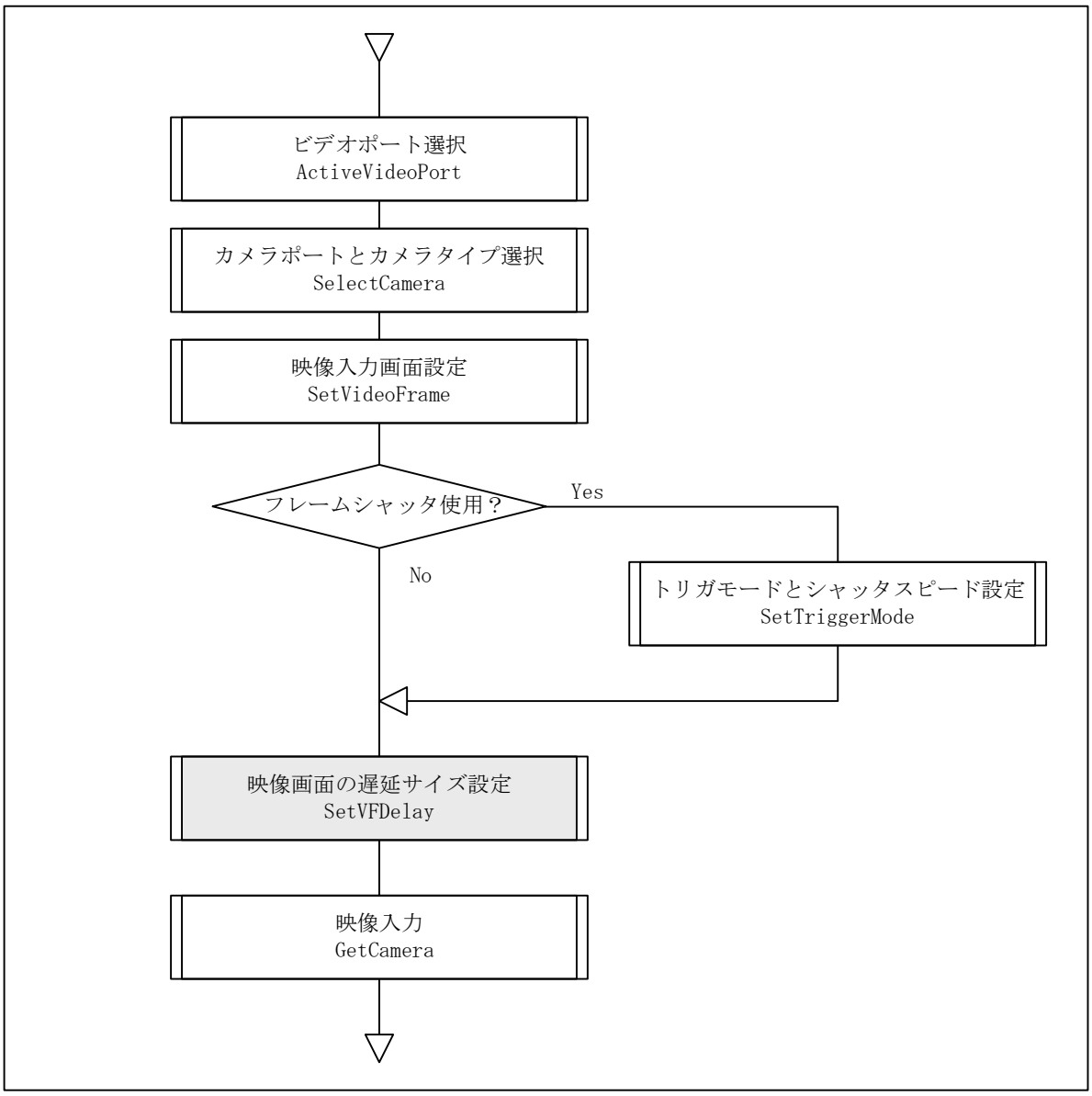


図4-5-10　カメラ映像入力位置調整フロー

(5) ストロボ映像入力

ＮＶＰ－Ａｘ２３０ＣＬにはストロボが２チャネル、ＮＶＰ－Ａｘ２３５ＣＬにはストロボが４チャネル、それぞれ独立にストロボ情報を設定することができます。以下にトリガとストロボ出力のタイミングを示した図とストロボ情報テーブルを示します。ストロボは指定したビデオポートのトリガ出力からDelay時間後に、Lengthの長さで出力されます。ストロボ出力ディレイ、ストロボ出力長、ストロボ極性の設定値は使用するストロボにより異なります。

ストロボ映像入力は、トリガモードの映像入力設定に加えて、SetStrobeModeコマンドでストロボ情報を設定します。SetStrobeModeコマンドの実行タイミングは特に制限されません。

ストロボモードを解除する場合には、SetStrobeModeコマンドでストロボ無効に設定してください。このとき、ストロボ情報テーブルの設定値はストロボ有効に設定した値をそのまま設定してください。設定値が異なるとストロボモードを解除できないことがあります。

```
/* ストロボ情報テーブル */
typedef struct {
    int  VideoPort;    /* ビデオポート番号                          */
    int      Delay;    /* 出力ディレイ                              */
    int     Length;    /* ストロボ出力長                            */
    int       Pole;    /* ストロボ極性          (0:正極性,1:負極性) */
} STRB_INFO;
```

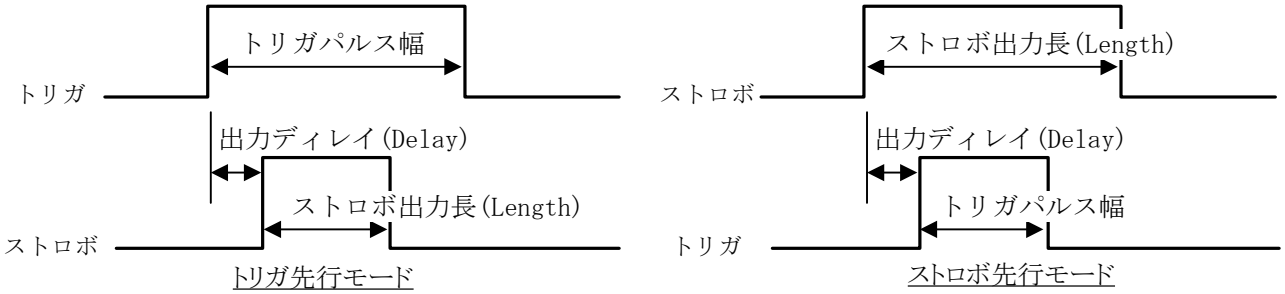


図4-5-11　トリガおよびストロボ出力タイミング

図4-5-12　ストロボを使用する場合の映像入力フロー

(6) パーシャル映像入力

パーシャル映像入力可能なカメラに対して、パーシャル映像入力をサポートします。
パーシャル映像入力を行う場合、カメラの有効サイズが変わるため、カメラデータ(4.5.8項を参照ください。)をパーシャルモードに合わせて変更する必要がありますが、以下のフローに従ってディレイオフセットを変更すれば、カメラデータの変更なしに対応できます。
パーシャルモードや設定するパーシャル情報はカメラにより異なります。カメラインターフェースガイド、コマンドリファレンスを確認の上、パーシャル映像入力を行ってください。
『SetPartialMode』コマンドによるパーシャルモード設定は未サポートです。

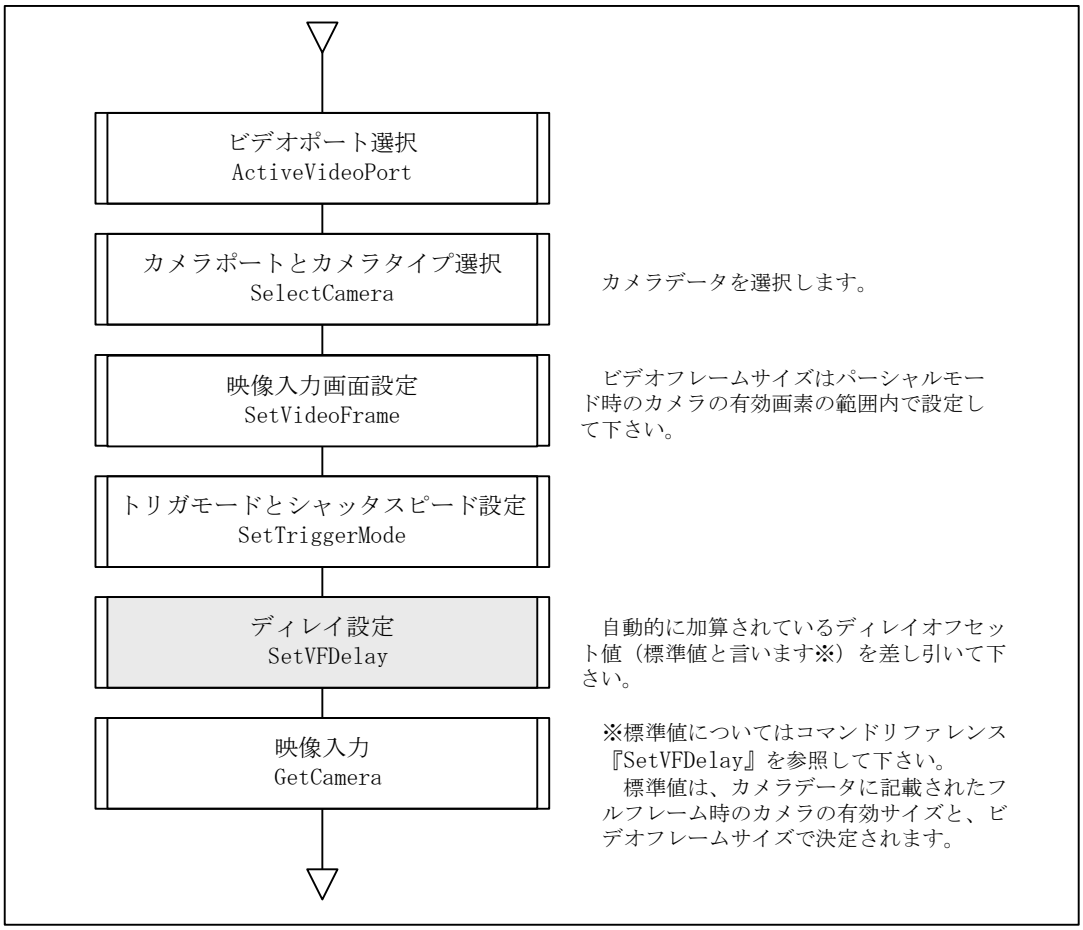


図4-5-13　パーシャル映像入力フロー

4.5.8 カメラデータの設定

ＮＶＰ－Ａｘ２３０ＳＤＫは多種のカメラの映像入力をサポートします。そのため映像入力に必要なデータをデフォルトデータとして管理していますが、映像入力のアプリケーションによって、デフォルト値を変更したい場合があります。また、サポートカメラ以外のカメラで映像入力を行いたい場合があります。
画像処理コマンドは、ユーザカメラとして新規カメラを４つまで登録可能です。システムに設定されているカメラデータを取得するGetCameraDataコマンド、カメラデータを設定するSetCameraDataコマンド、および、テキストファイルに記述されたカメラデータをシステムに設定するLoadCameraFileコマンドで、カメラデータの変更や新規カメラデータを設定することができます。
但し、カメラデータとして不適切な値を設定した場合には、故障・破損の原因になります。カメラデータおよびカメラデータファイルについては、弊社が提供するもの以外を設定しないでください。

4.5.9 NVP－Ax230からの映像入力以外の映像入力

画像処理コマンドは、ＮＶＰ－Ａｘ２３０を使用した映像入力以外に、キャプチャデバイスや動画ファイル（ＡＶＩファイルなど）、ＢＭＰファイル等からの映像入力をサポートします。SetConfigCameraコマンドで映像入力構成を設定することで、「VFWCam.exe」アプリケーションが起動され、上記メディアからの映像入力ができます。
ただし、これらの映像入力はフルフレーム取込でGetCameraによる単一映像入力だけをサポートし、ビデオポートの選択はできません。

4.6 映像出力

4.6.1 NVP－Ax230での映像出力の概要

ＮＶＰ－Ａｘ２３０は、アナログＲＧＢでの映像出力をサポートします。
画像処理コマンドでサポートする映像表示は、カメラ映像の表示(ライブ表示)、画像メモリの表示、画像メモリとカメラ映像の重ね合わせ表示(オーバレイ表示)です。
また、オーバレイのカラー表示やの画像メモリの擬似カラー表示をサポートします。

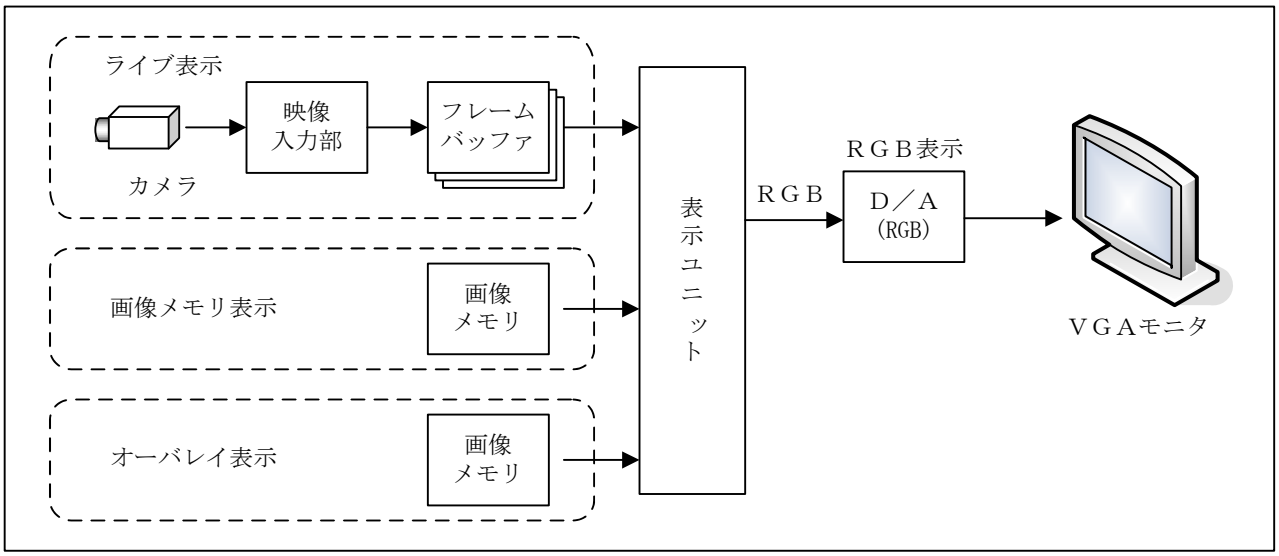


図4-6-1　映像表示機能概要

(1) 映像表示プレーン

ＮＶＰ－Ａｘ２３０には、映像表示プレーンと呼ぶ表示用画面が８面あります。
各プレーンは重ね合わせて表示することが可能で、この機能を用いてオーバレイ表示を行います。映像表示プレーンは１～８(DISP_PLANE_1～DISP_PLANE_8)までの番号で表し、以下のようにプレーン番号が小さいほど表示の優先順位が高くなります。画像メモリ表示やオーバレイ表示のプレーン画面は、表示画面の構成制御コマンドで設定します。

DISP_PLANE_1 ＞ DISP_PLANE_2 ＞　(中略)　＞ DISP_PLANE_7 ＞ DISP_PLANE_8

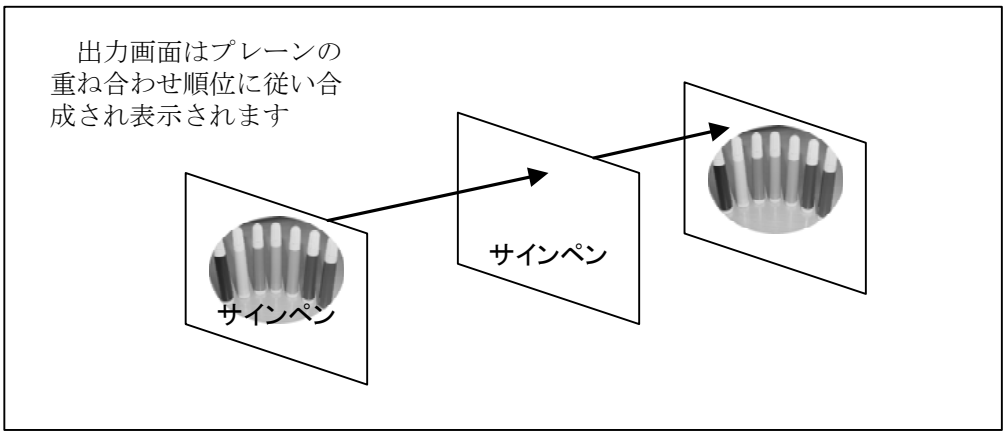


図4-6-2　プレーンの重ね合わせ

(2)カラーパレット

ＮＶＰ－Ａｘ２３０には、２６万色中、同時に２５６色が表示可能なカラーパレットが６面あり、オーバレイのカラー表示、カメラ映像や画像メモリの擬似カラー表示が可能です。カラーパレットは、１～６(DISP_PALLET_1～DISP_PALLET_6)までの番号で表し、表示画面の構成制御コマンドで使用するパレット番号を設定します。

(3)映像表示機能

画像処理コマンドが制御する映像表示機能一覧を以下に示します。

表4-6-1　ＲＧＢ表示機能一覧

機能名	コマンド名	表示画面	機能
ライブ表示	DispCamera()	Ｙ ※２	モノクロカメラ映像表示 ＹＣモード
		Ｙ ※２	モノクロカメラ擬似カラー映像表示 ※１ 8bit/pixelモード
画像メモリ表示	DispImg()	Ｙ	モノクロ画像メモリ表示 ＹＣモード
		Ｙ	モノクロ画像メモリ擬似カラー映像表示 8bit/pixelモード
		ＲＧＢ１６ ※３	カラー画像メモリ表示 16bit/pixelモード
オーバレイ表示	DispOverlap()	Ｙ	モノクロ画像メモリ擬似カラー映像表示 8bit/pixelモード
		ＲＧＢ１６	カラー画像メモリ表示 16bit/pixelモード

※１ 『SelectDisp』コマンドで設定
※２ ライブ表示の場合は、内部確保画面タイプ
※３ コンポーネントＲＧＢ画面はＲＧＢ１６画面に変換して表示

4.6.2 表示画面の構成制御設定

画像処理コマンドでサポートするライブ表示、画像メモリ表示、オーバレイ表示は、表示画面の構成制御設定『SetConfigDisp』コマンドで設定された情報で行います。以下に表示画面の構成制御パラメータを示します。

(1) 表示フレームサイズ

表4-6-2　表示フレームサイズ

設定内容	動作	設定範囲	備考
xsize	X方向の表示サイズ	１～２０４６　※	実際の表示は１～６４０
ysize	Y方向の表示サイズ	１～１０２３　※	実際の表示は１～４８０

(2) 表示位置

表4-6-3　表示位置

設定内容	動作	設定範囲	備考
dpx	X方向の表示開始位置	１～２０４６　※	
dpy	Y方向の表示開始位置	１～１０２３　※	

※　対象システムが実際に表示可能な範囲内で設定してください。また、X方向の表示サイズはハード制約により、偶数に設定しないと設定サイズ分表示されません。

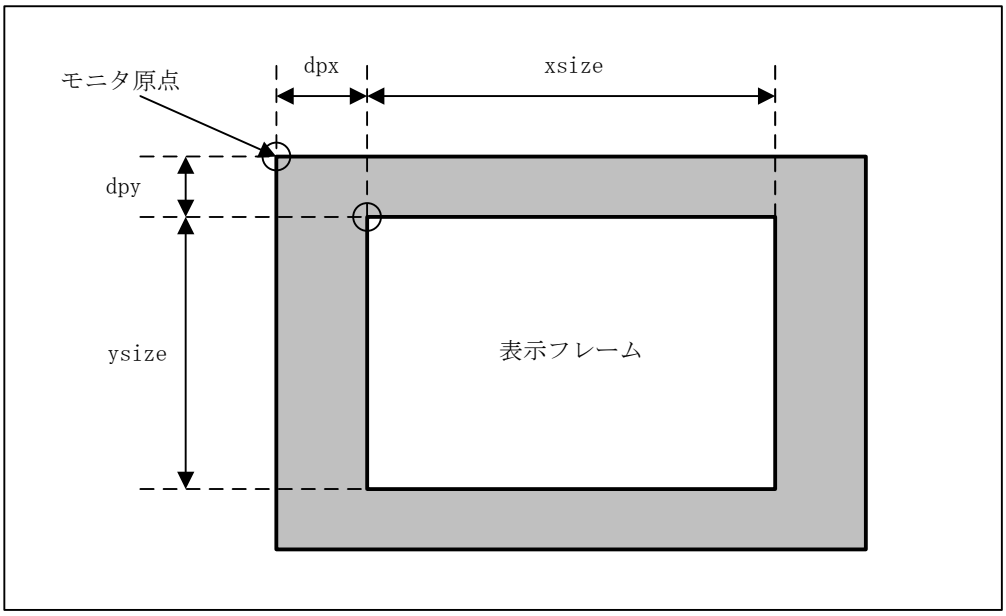


図4-6-3　表示画面の構成制御パラメータ

(3) 画像表示プレーン番号

ライブ表示／画像メモリ表示に使用するプレーン番号を指定する。

(4) オーバレイ表示プレーン番号

オーバレイ表示に使用するプレーン番号を指定する。

(5) オーバレイ表示パレット番号

オーバレイ表示に使用するカラーパレット番号を指定する。

4.6.3 表示フレームサイズ

画像処理コマンドでサポートする映像入力フレームサイズ、表示フレームサイズ、画面サイズの組合せを以下に示します。
映像入力フレームサイズはSetVideoFrameコマンド、表示フレームサイズはSetConfigDispコマンドで指定します。画面サイズはAllocImgコマンドで領域確保した画像メモリのサイズです。
ただし、動画ファイルからのカメラ映像入力を行う場合、映像入力フレームサイズは動画ファイルの映像サイズで決まります。Windows上の画面へ画像メモリを表示する場合、表示フレームサイズはSetConfigViewコマンドで指定します。

表4-6-4　映像入力フレームサイズと表示フレームサイズ

No.	映像入力 フレームサイズ	表示フレームサイズ	画面サイズ	備考
1	256H×220V	256H×220V	256H×256V	
2	512H×220V	512H×220V	512H×256V	
3	256H×440V	256H×440V	256H×512V	
4	512H×440V	512H×440V	512H×512V	
5	256H×240V	256H×240V	256H×256V	
6	320H×240V	320H×240V	320H×256V	
7	512H×240V	512H×240V	512H×256V	
8	640H×240V	640H×240V	640H×256V	
9	256H×480V	256H×480V	256H×512V	
10	320H×480V	320H×480V	320H×512V	
11	512H×480V	512H×480V	512H×512V	
12	640H×480V	640H×480V	640H×512V	
13	1024H×768V	640H×480V	1024H×768V	表示は640H×480Vに制限される
14	768H×512V	640H×480V	768H×512V	表示は640H×480Vに制限される
15	768H×576V	640H×480V	768H×576V	表示は640H×480Vに制限される
16	2432H×2048V	640H×480V	2560H×2048V	表示は640H×480Vに制限される

4.6.4 Windows上の画面への画像メモリ表示

画像処理コマンドは、ＮＶＰ－Ａｘ２３０によるアナログＲＧＢでの映像表示以外にWindows上の画面への画像メモリ表示をサポートします。SetConfigViewで画像メモリ表示構成を設定することで「IPView.exe」アプリケーションが起動され、DispImgで指定した画像メモリが表示できます。
ただし、オーバレイ表示はできません。

4.7 RGBカラー処理機能

ＲＧＢカラー処理機能には、以下の３点の機能があります。

- コンポーネントＲＧＢカラーカメラからの映像取り込み
- カラー画像のＬＵＴ変換
- カラー画像表示

4.7.1 コンポーネントRGB信号について

コンポーネントＲＧＢカラー信号は、Ｒ（赤）、Ｇ（緑）、Ｂ（青）の信号で構成され、ＲＧＢカラーカメラからは別々の信号線で送られます。以下に主なカラーバーに対するＲＧＢ値を示します。

表4-7-1 カラーバーに対するＲＧＢ値

	R	G	B		R	G	B
White	255	255	255	Blue	0	0	255
Black	0	0	0	Yellow	255	255	0
Red	255	0	0	Cyan	0	255	255
Green	0	255	0	Magenta	255	0	255

4.7.2 RGBカラー時の画像メモリ使用方法

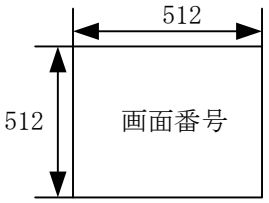
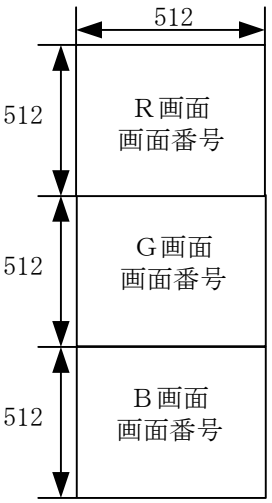
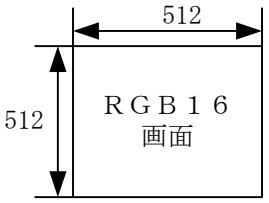
ＲＧＢカラー映像データは、Ｒ(赤)、Ｇ(緑)、Ｂ(青)の３つのデータで構成されるため、画像メモリは３画面分を使用します。

ＲＧＢカラー用の画像メモリを確保するコマンドがAllocRGBImg()です。AllocRGBImg()では、カラー映像用にＲ画面とＧ画面とＢ画面が連続する３画面を確保します。ユーザにはこのうちＲ画面の画面番号を返し、Ｇ、Ｂ用の画面はシステム内部で管理されます。

ＲＧＢ画面は、ＲＧＢカラー映像入力､画像処理に使用できますが、映像表示に使用できません。
映像表示用には、ＲＧＢ１６画面（16bit/pixcel）を使用します。

ＲＧＢ１６画面を確保するコマンドがAllocRGB16Imgです。ＲＧＢ１６画面は、映像入力、画像処理には使用できません。

表4-7-2 画面の種類と確保メモリ

画面の種類	コマンド	確保例
モノクロ画面	AllocImg	(例) AllocImg （IMG_FS_512H_512V ）  512 ×512 の画面を １画面確保
ＲＧＢ カラー画面	AllocRGBImg	(例) AllocRGBImg （IMG_FS_512H_512V ）  512 ×512 の画面を ３画面連続した領域 に確保 Ｒ画面の画面番号を ユーザに返す
ＲＧＢ１６ カラー画面	AllocRGB16Img	 16bit/Pixelの 512 ×512 の画面を １画面確保

4.7.3 コンポーネントRGBカメラからの映像取り込み

コンポーネントＲＧＢカラー映像では、Ｒ信号、Ｇ信号、Ｂ信号それぞれ別々の信号として伝送されます。Ｒ信号、Ｇ信号、Ｂ信号はディジタル化され、画像メモリ上に連続したＲ画面/Ｇ画面/Ｂ画面の３画面として取り込まれます。

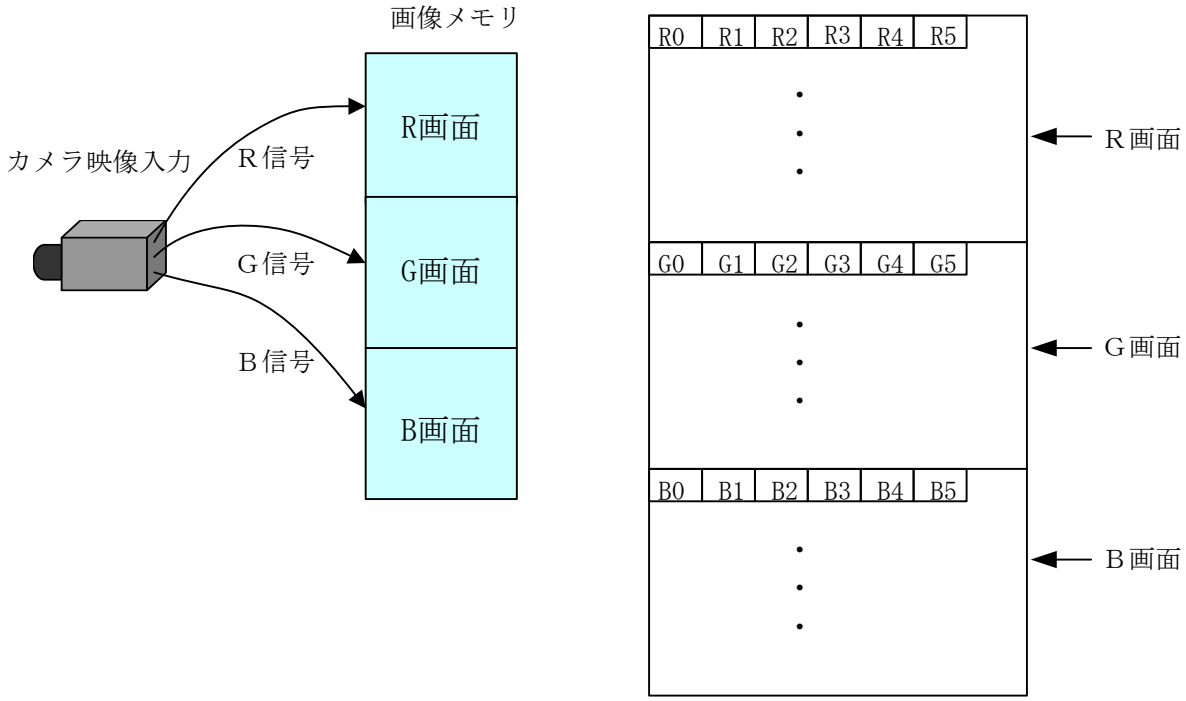


図4-7-1 ＲＧＢ画面構成

ＮＶＰ－Ａｘ２３０ＣＬ/２３５ＣＬでは、カメラリンクのＲＧＢカメラから映像入力します。

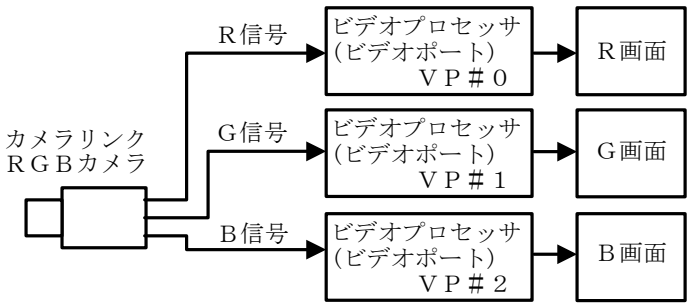


図4-7-2 ＲＧＢ映像入力部のハード構成

以下に、ＲＧＢ映像入力・出力の例を示します。

```
// 画面確保

      // ＲＧＢ画面確保
      ImgRGB = AllocRGBImg(IMG_FS_512H_512V);
      // ＲＧＢ１６表示画面確保
      ImgRGB16 = AllocRGB16Img(IMG_FS_512H_512V);


// ＲＧＢ映像入力

      // カメラポート配置の設定
      ActiveVideoPort(VIDEO_PORT0);
      // ＲＧＢカメラの選択
      SelectCamera(CAMERA_PORT0,CIS_VCC8350A);
      // 映像入力画面設定
      SetVideoFrame(NONINTERLACE,VIDEO_FS_512H_480V);
      // ＲＧＢカメラからの映像入力
      GetCamera(ImgRGB);

// ＲＧＢカラー映像表示

      // RGB -> RGB16 変換
      IP_CombineRGB(ImgRGB,ImgRGB16);
      // RGB カラー映像表示
      DispImg(ImgRGB16);
```

4.7.4 RGBカラー映像の表示出力

ＮＶＰ－Ａｘ２３０ではＲＧＢカラー映像を直接ＶＧＡカラーモニタに表示できません。そのため、CombineRGB()コマンドを使用してＲＧＢ画像をＲＧＢ１６画像に変換する必要があります。

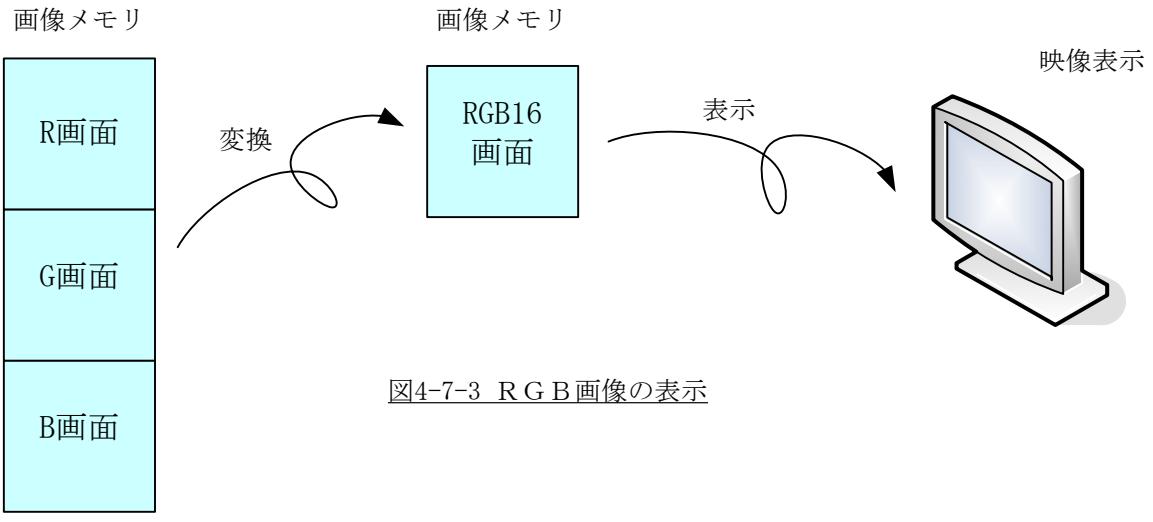


図4-7-3 ＲＧＢ画像の表示

4.7.5 RGBカラー画面の画像処理

AllocRGBImgコマンドで確保したＲＧＢカラー画面は、モノクロ画面として画像処理をすることができます。ただし、ＲＧＢＬＵＴコマンド等の一部のコマンドは、Ｒ，Ｇ，Ｂの全ての画面を処理しますので、コマンドリファレンスでご確認ください。

ＲＧＢカラー画面に対して画像処理する場合は、Ｒ画面のみ処理します。Ｇ、Ｂ画面に対して処理する場合は、Ｒ画面の画面番号で管理するＲＧＢカラー画面のＧ、Ｂ画面番号を取得し、このＧ、Ｂ画面番号を使用して実行できます。GetGImgID コマンドでカラー画面のＧ画面番号を、GetBImgID コマンドでカラー画面のＢ画面番号を取得します。

AllocRGB16Imgコマンドで確保したカラー画面に対しては画像処理することができません。ただし、ReadImg、WriteImg、IP_Copyコマンド等、一部のコマンドを使用することが出来ますので、コマンドリファレンスでご確認ください。

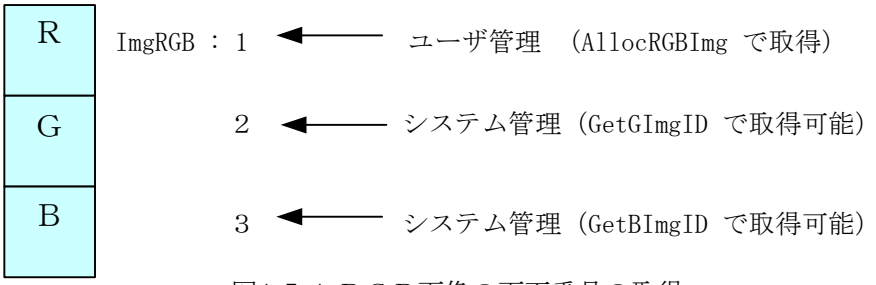
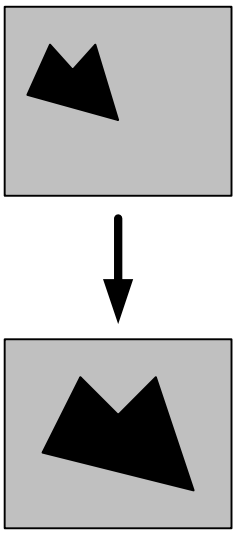
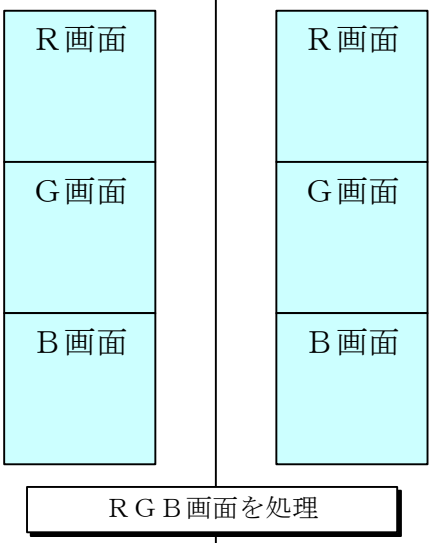
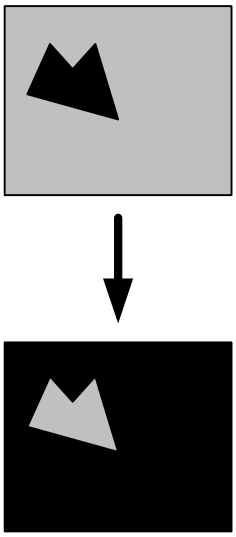
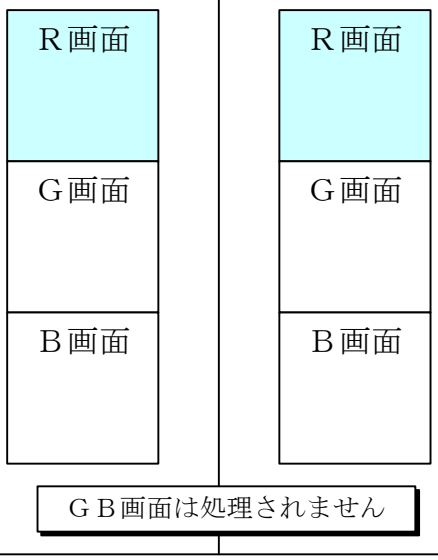
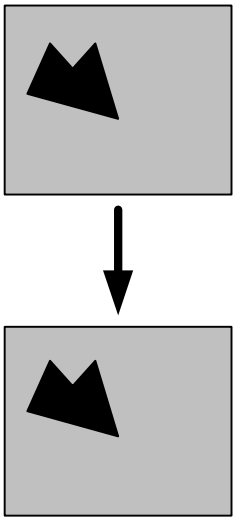
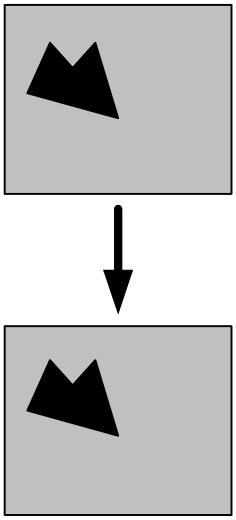
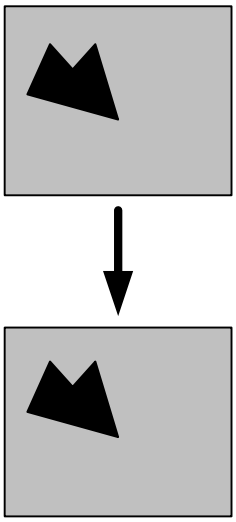


図4-7-4 ＲＧＢ画像の画面番号の取得

表4-7-3 ＲＧＢ画像処理の対象メモリ画面

画像処理例	画像処理	ソース	デスティネーション
拡大 IP_Zoom			
論理反転 IP_Invert			

画像処理例	画像処理	ソース	デスティネーション
RGBLUT変換 IP_ConvertRGBLUT		R画面 G画面 B画面 ＲＧＢ画面をＹ画面へ変換	Ｙ画面
RGBからRGB16へ 変換 CombineRGB		R画面 G画面 B画面 ＲＧＢ画面からＲＧＢ16画面へ変換	RGB16画面
コピー IP_Copy		RGB16画面 ＲＧＢ16画面をＲＧＢ16画面へコピー	RGB16画面

第5章　画像処理コマンドの概要

画像処理は、機能ごとに下記のように分類されます。

（１）アフィン変換
（２）２値化
（３）濃度変換
（４）画像間算術演算
（５）画像間論理演算
（６）２値画像形状変換
（７）コンボリューション
（８）ランクフィルタ
（９）ラベリング
（１０）ヒストグラム
（１１）画像メモリアクセス
（１２）２値パイプラインフィルタ
（１３）パイプライン制御
（１４）２値マッチングフィルタ
（１５）正規化相関
（１６）グラフィックス
（１７）線分化
（１８）穴埋め
（１９）ＶＰ－９１０Ａ互換
（２０）直線抽出
（２１）イメージキャリパ
（２２）エッジファインダ

5.1 画像処理コマンドの概要

画像処理は、処理対象画像データの画像メモリからの読み出し、画像処理プロセッサによる演算、処理結果画像の画像メモリへの書き込みの順番で行われます。
処理対象画像は、ソース画像またはソース画面と呼びます。
処理結果画像は、デスティネーション画像またはデスティネーション画面と呼びます。

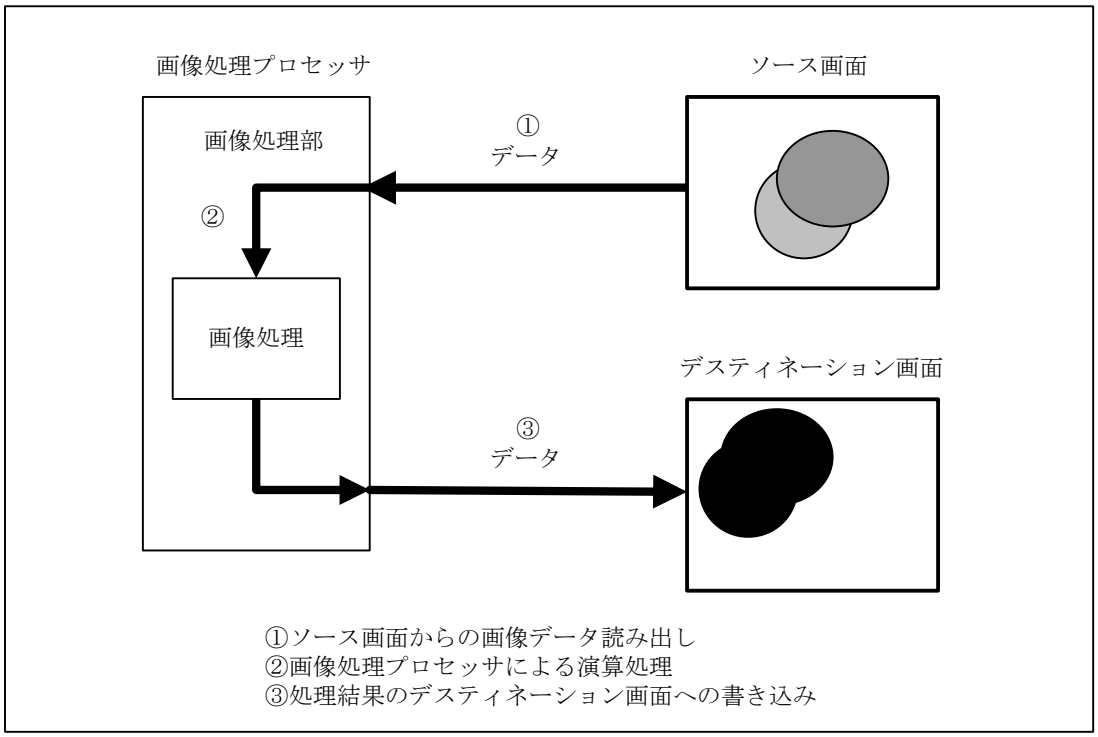
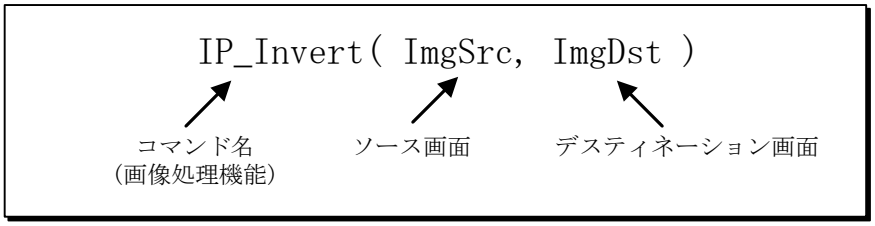


図5-1-1　画像処理の概要

本ライブラリでは、画像処理の機能をＣ言語のサブルーチン形式で提供します。画像処理のプログラムでは、コマンド（画像処理機能）の選択、ソース画面とデスティネーション画面の選択によって、画像処理機能を実行します。



画像処理の手順と各機能との関係は次のとおりです。ただし、これは一般的な場合の例であり、アプリケーションごとに、若干前後することがあります。

表5-1-1　画像処理の手順と各機能の対応

画像処理の手順	システム制御	画像メモリ領域管理	画像メモリ制御	映像入力表示制御	画像処理	パターン作成
開始 ↓ イニシャライズ（初期化）	○					
↓ 処理画像領域の設定		○				
↓ 画像データ入力				○		
↓ 画像間演算・微分等の画像前処理					○	
↓ 面積等の特徴量を抽出					○	
↓ 画像メモリの内容を読み出し／書き込み			○			
↓ 処理結果を表示する際の文字等の書き込み／重ね合せ表示				○		○
↓ 終了						

5.2 アフィン変換

アフィン変換とは、拡大／縮小、移動、回転などの幾何学変換のことです。本ライブラリでは、下の式の２次元のアフィン変換をハード処理とソフト処理によりサポートします。２～８倍及び１／２～１／８倍の整数倍の拡大／縮小と移動はハード処理で実現し、それ以外はソフト処理で実現します。

$$X = ax + by + c$$
$$Y = dx + ey + f$$

x, y	変換前の座標
X, Y	変換後の座標
a, b, c, d, e, f	任意定数

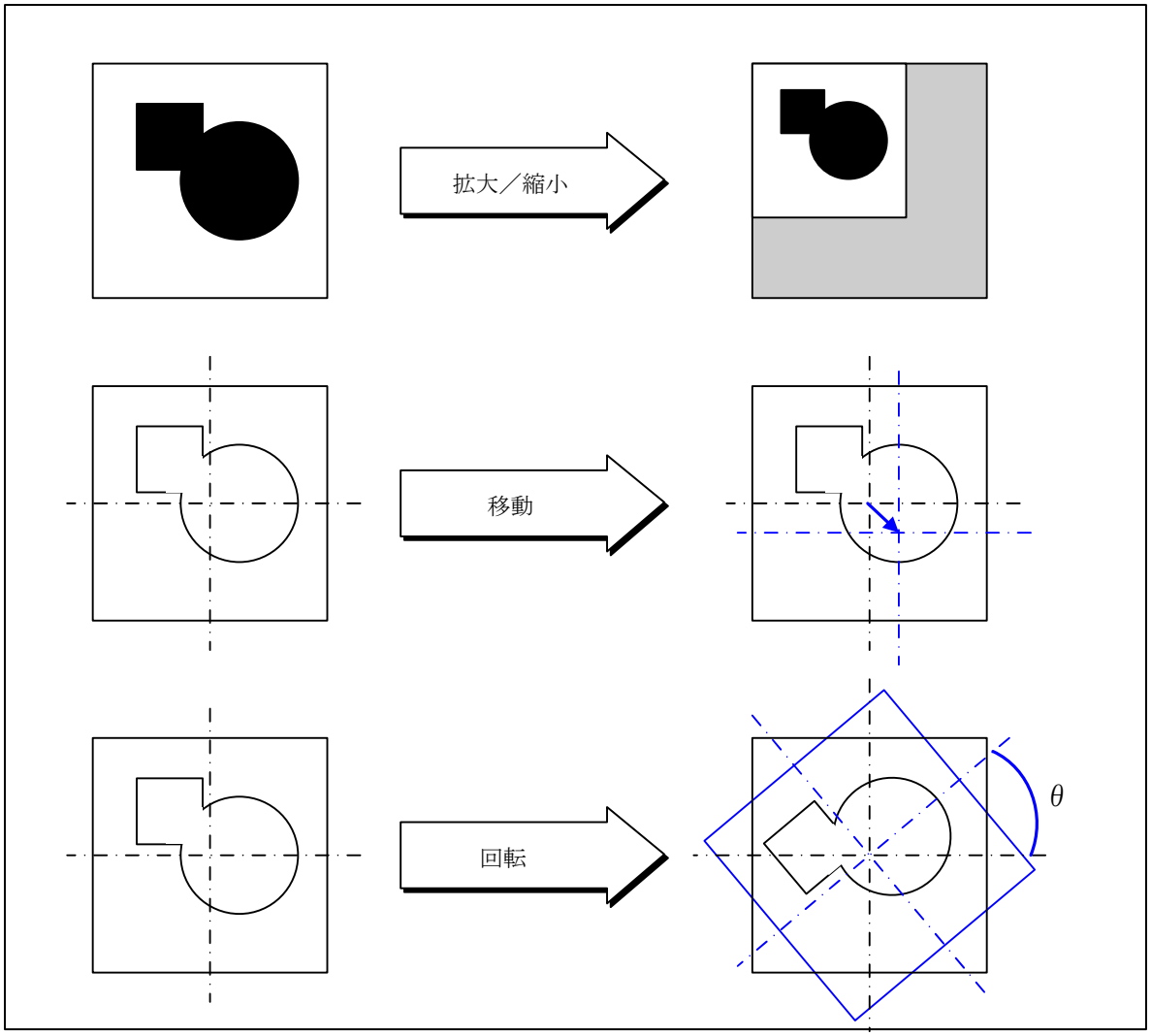


図5-2-1　アフィン変換

5.3 2値化

２値化とは、濃淡画像（２５６階調）を黒（濃度０）と白（濃度２５５）の２階調の画像に変換することです。本ライブラリコマンドでは、任意のしきい値以上の濃度値を白、それ以外の濃度値を黒とする通常の２値化と、任意の濃度値の範囲内と範囲外で白か黒にする範囲・反転付２値化をサポートします。

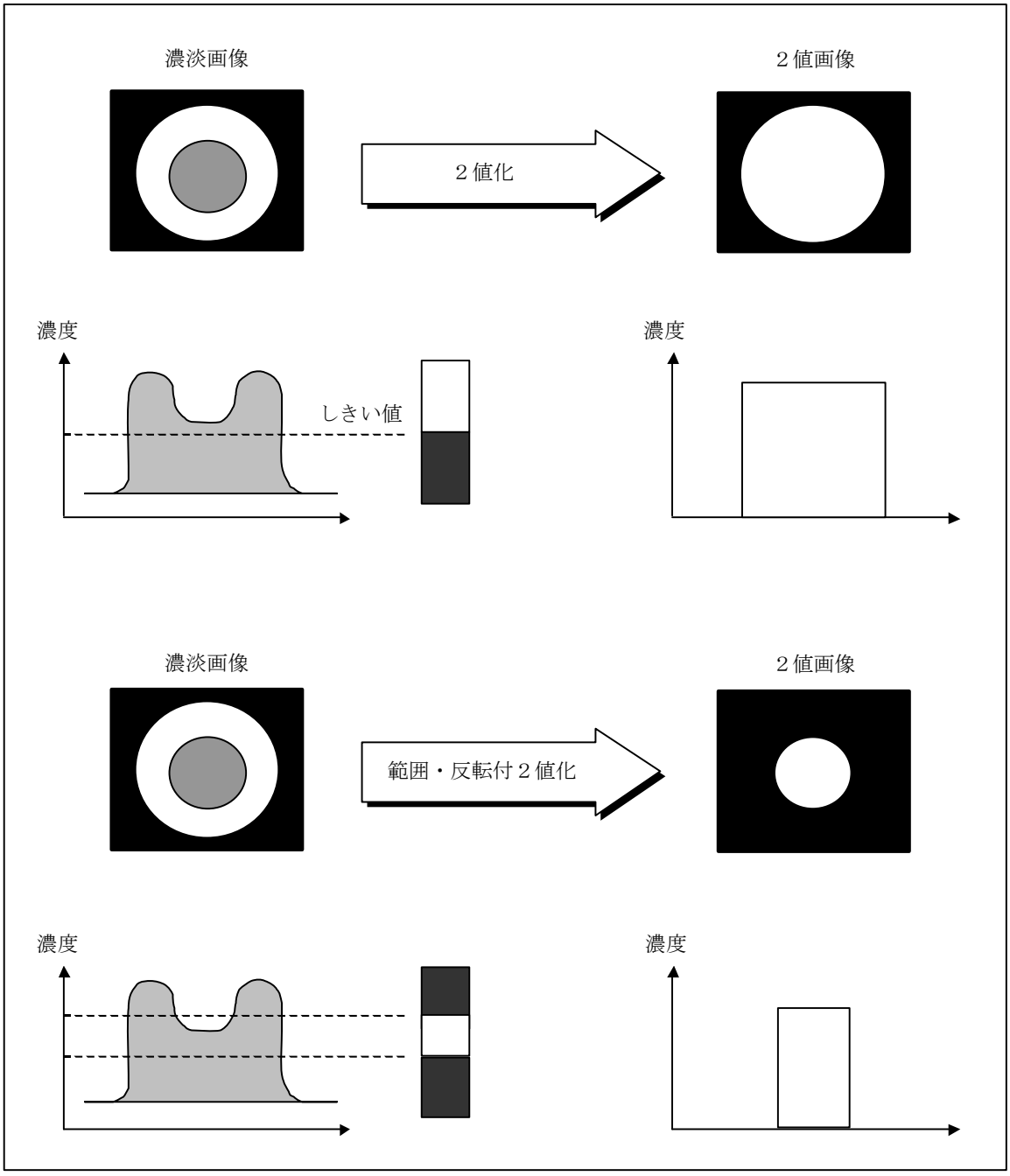


図5-3-1　2値化

5.4 濃度変換

濃度変換とは、濃度（輝度・明るさ）を変換する処理です。
濃度変換のパターンを換えることにより、暗い部分の濃度を高くしたり、逆に明るい部分の濃度を低くしたりする画質改善が可能です。画質改善以外にも、等濃度縞（ストライプ）のデータを使用すれば、曲面の曲がり、くぼみなどを見つけることができます。さらに、この考えを拡大していけば、３値化、４値化という処理が行えます。濃度変換のパターンは、自由に設定可能です。

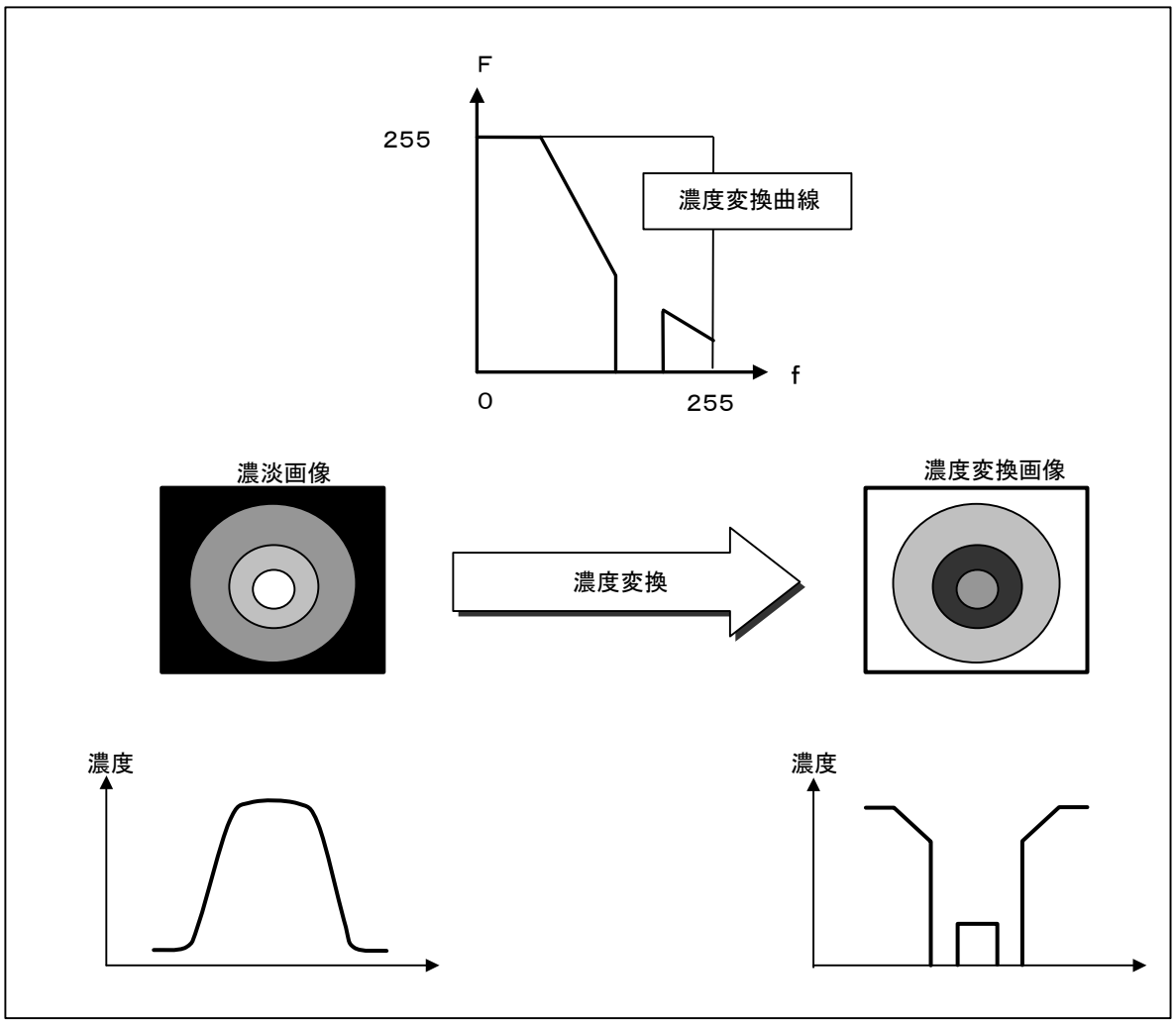
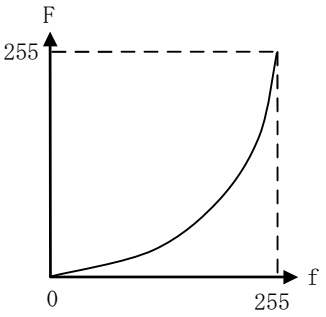
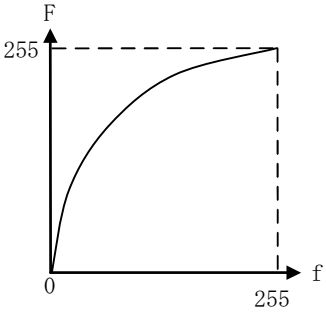
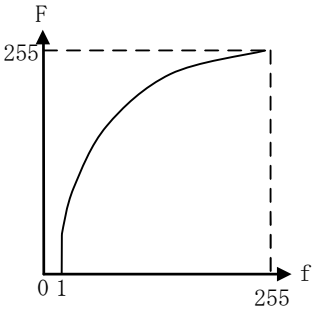
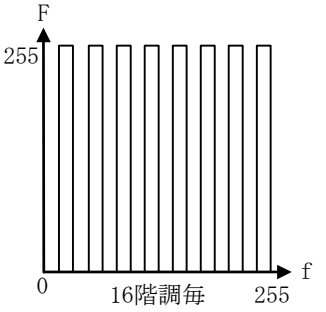
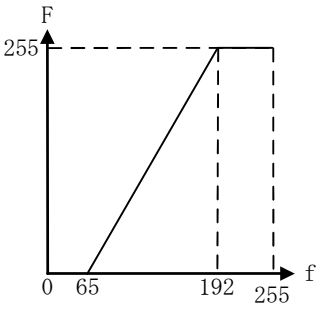


図5-4-1　濃度変換

表5-4-1　濃度変換詳細

種類	濃度変換曲線	詳細	効果
γ補正①		0≦f≦255において $F = af^{1.2}$ （但し $a = 255^{-0.2}$ ） f ： 処理対象画像濃度 F ： 処理結果画像濃度	低濃度部の強調
γ補正②		0≦f≦255において $F = af^{1/1.2}$ （但し $a = 255^{-0.2/1.2}$ ） f ： 処理対象画像濃度 F ： 処理結果画像濃度	高濃度部の強調
log変換		1≦f≦255において F = a*log(f) （但し a = 255/log(255)） f = 0 において F = 0 f ： 処理対象画像濃度 F ： 処理結果画像濃度	１以下を０とし高濃度部を強調
等濃度縞		0≦f≦255において 16階調毎にF=0 or F=255 f ： 処理対象画像濃度 F ： 処理結果画像濃度	１６階調単位に黒と白に変換
強調		0≦f≦255において f = 0 ～ 64 において F = 0 f = 65 ～ 191 において F = 255/127*(f-65) f = 192 ～ 255において F = 255 f ： 処理対象画像濃度 F ： 処理結果画像濃度	低濃度部と高濃度部の強調 （コントラスト改善）

5.5 画像間算術演算

画像間演算とは、２画面の画像で演算を行い演算後別の１画面に合成することです。画像間演算には加算、減算、乗算などの画像間算術演算と論理和、論理積などの画像間論理演算があります。

画像間算術演算の用途としては、基準パターンとの減算による差画像から欠陥を検出したり、何度も同一画像を取込み加算することでランダムノイズ（ホワイトノイズ）を除去するといったことがあります。また、リアルタイムに２画面の画像の最大値をとることで、明るい物体を次から次へと１画面の画像に合成することができます。これは、移動物体の軌跡や移動量を検出するために有効です。

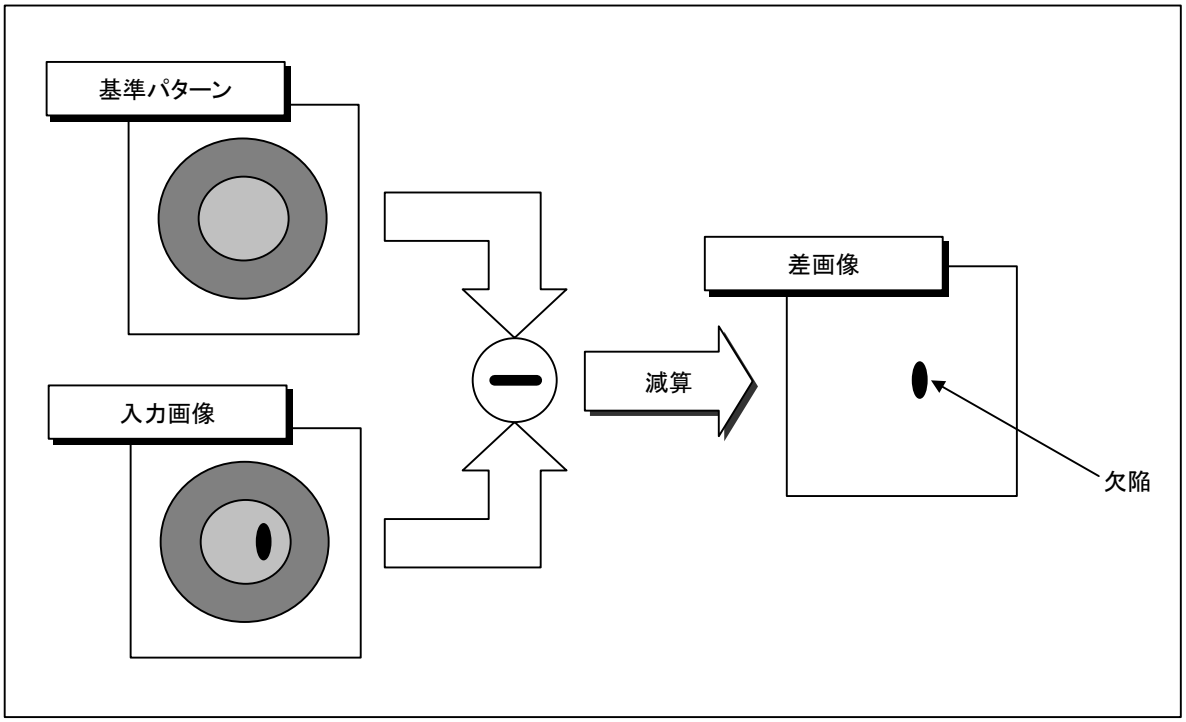


図5-5-1　画像間算術演算

5.6 画像間論理演算

２画面間で１画素毎に論理和、論理積などの論理演算を行い結果を別の画像に格納します。
２値画像での基準パターンと検査パターンとのチェック（ＸＯＲ演算）、画像の合成（ＯＲ演算）、キズの検出に有効です。
また、画像に任意のパターンでマスクをかける場合にも任意パターンとターゲット画像のＡＮＤ演算を行います。

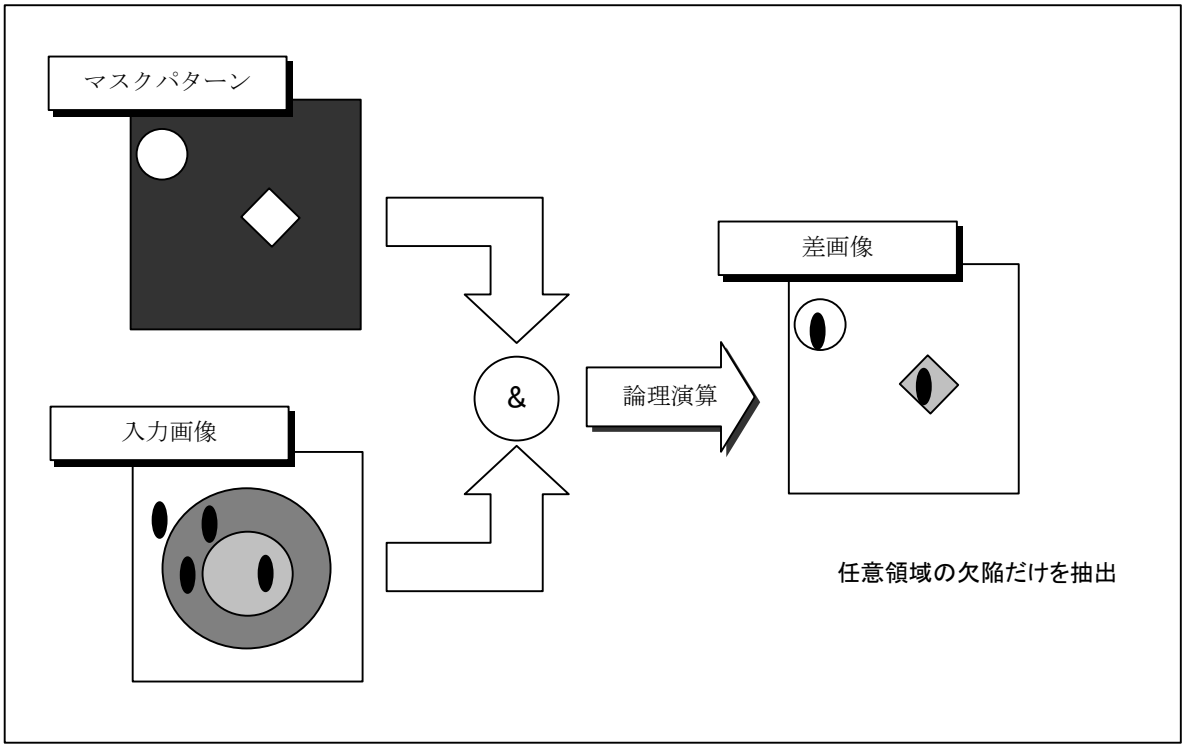


図5-6-1　画像間論理演算

5.7 2値画像形状変換

２値画像に対してノイズ除去、輪郭抽出、膨張、収縮、細線化、縮退化といった処理を行うことができます。これらの処理は、目的に応じて４連結と８連結を選択できます。

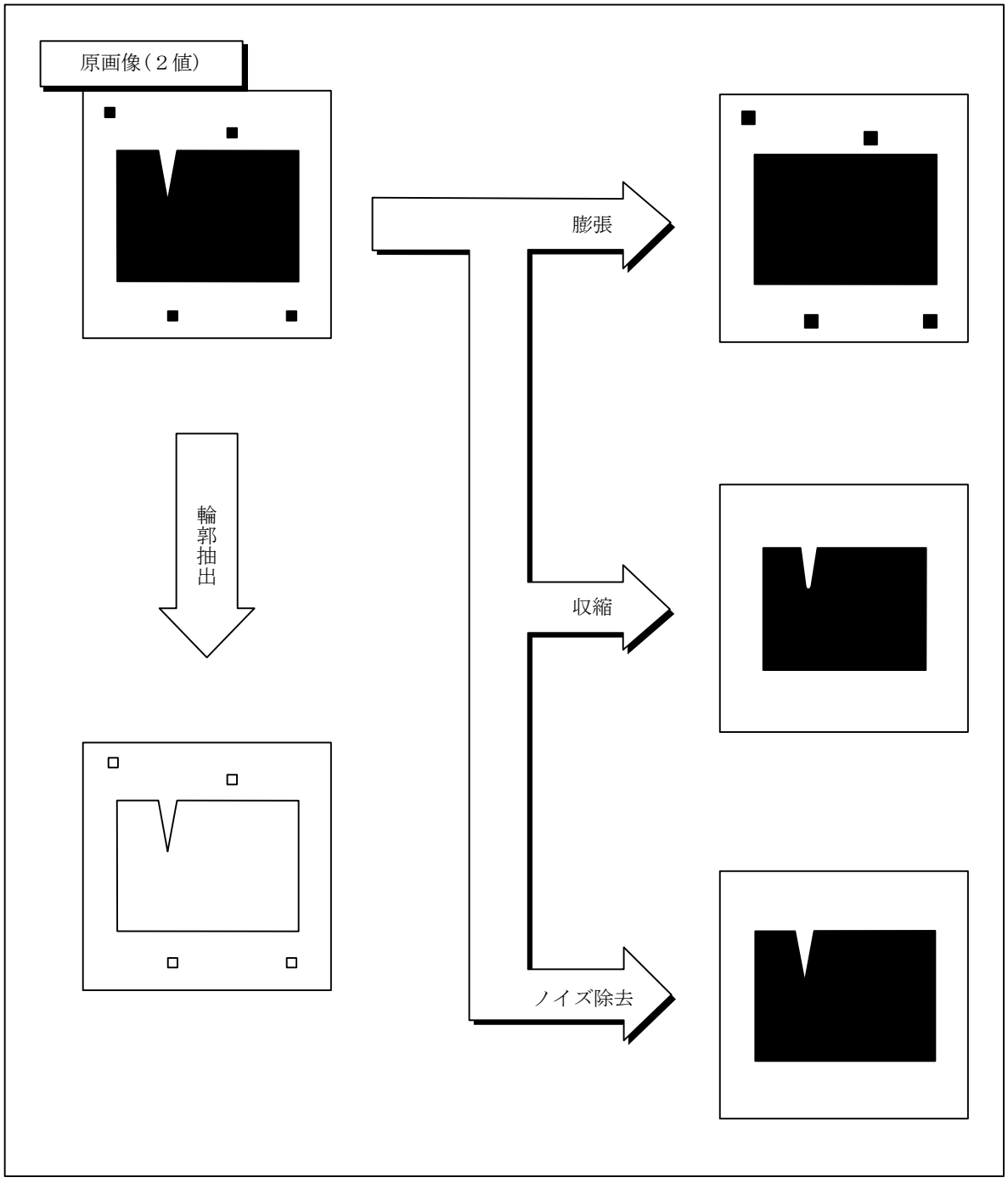


図5-7-1　２値画像形状変換

5.8 コンボリューション

コンボリューションは、近傍領域の積和演算です。下図に示すように、ソース画面の指定領域内の全画素に対して、注目画素を中心とした３×３近傍の局所領域で与えられた荷重係数との積和演算を行います。

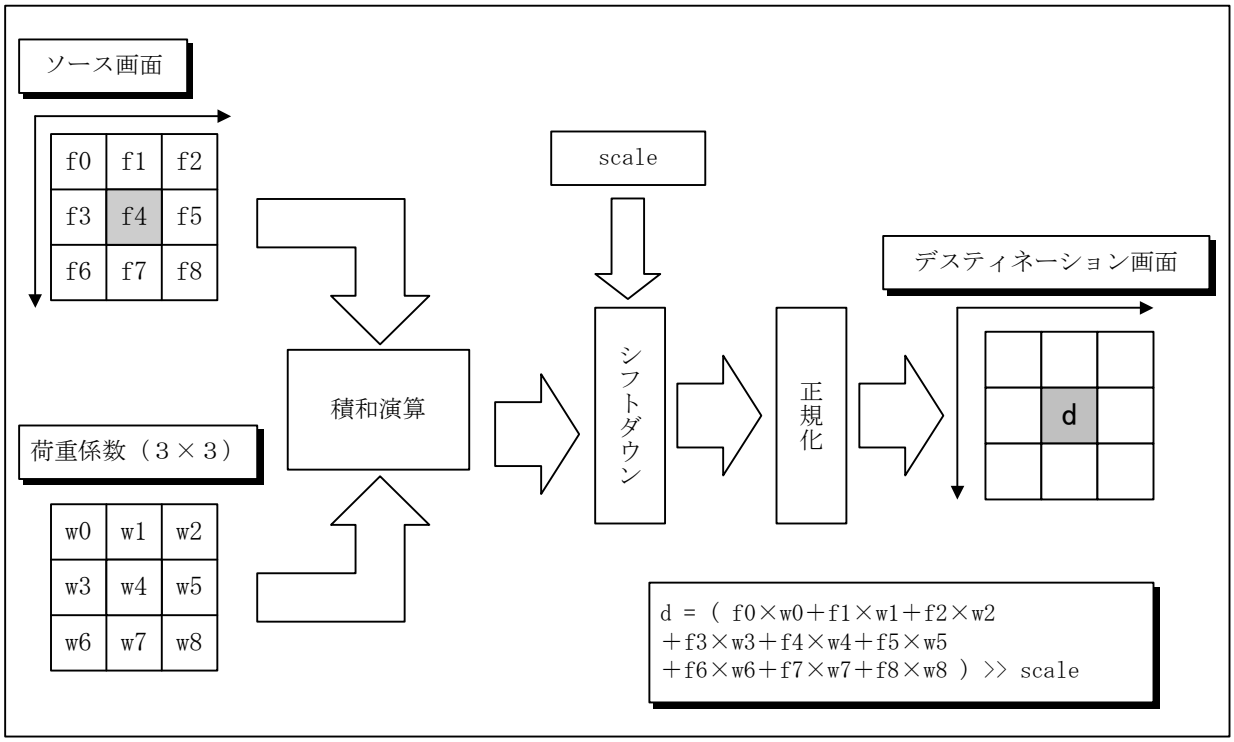


図5-8-1　コンボリューション

この演算は下表に示す荷重係数の設定により、濃淡画像での平滑化や輪郭強調などを行うことができます。

表5-8-1　コンボリューション詳細

種類･名称		荷重係数	scale	効果
平滑化	－	1 1 1 1 1 1 1 1 1	3	平滑化。画像の輪郭を滑らかにする。
１次微分 グラディエント	微分	0 0 0 0 1 -1 0 0 0	0	輪郭強調（X方向）
		0 0 0 0 1 0 0 -1 0	0	輪郭強調（Y方向）

種類･名称		荷重係数	scale	効果
１次微分 グラディエント	Roberts	0 0 0 0 1 0 0 0 -1	0	輪郭強調（X方向）
		0 0 0 0 0 1 0 -1 0	0	輪郭強調（Y方向）
	Sobel	-1 0 1 -2 0 2 -1 0 1	0	輪郭強調（X方向）
		-1 -2 -1 0 0 0 1 2 1	0	輪郭強調（Y方向）
２次微分 ラプラシアン	４連結	0 -1 0 -1 4 -1 0 -1 0	0	輪郭強調
	８連結	-1 -1 -1 -1 8 -1 -1 -1 -1	0	輪郭強調

5.9 ランクフィルタ

ミニマムフィルタ、マックスフィルタ、メディアンフィルタなどをランクフィルタと呼びます。ランクフィルタ処理は、ソース画面の指定領域内の全画素に対して、注目画素を中心とした３×３近傍の局所領域内の画素データをソート（順に並べる）し、任意の順番の画素データを抽出します。また、３×３近傍の局所領域内で任意のカーネルマスクでマスク処理を行い、有効となった画素データ内でソートすることができます。

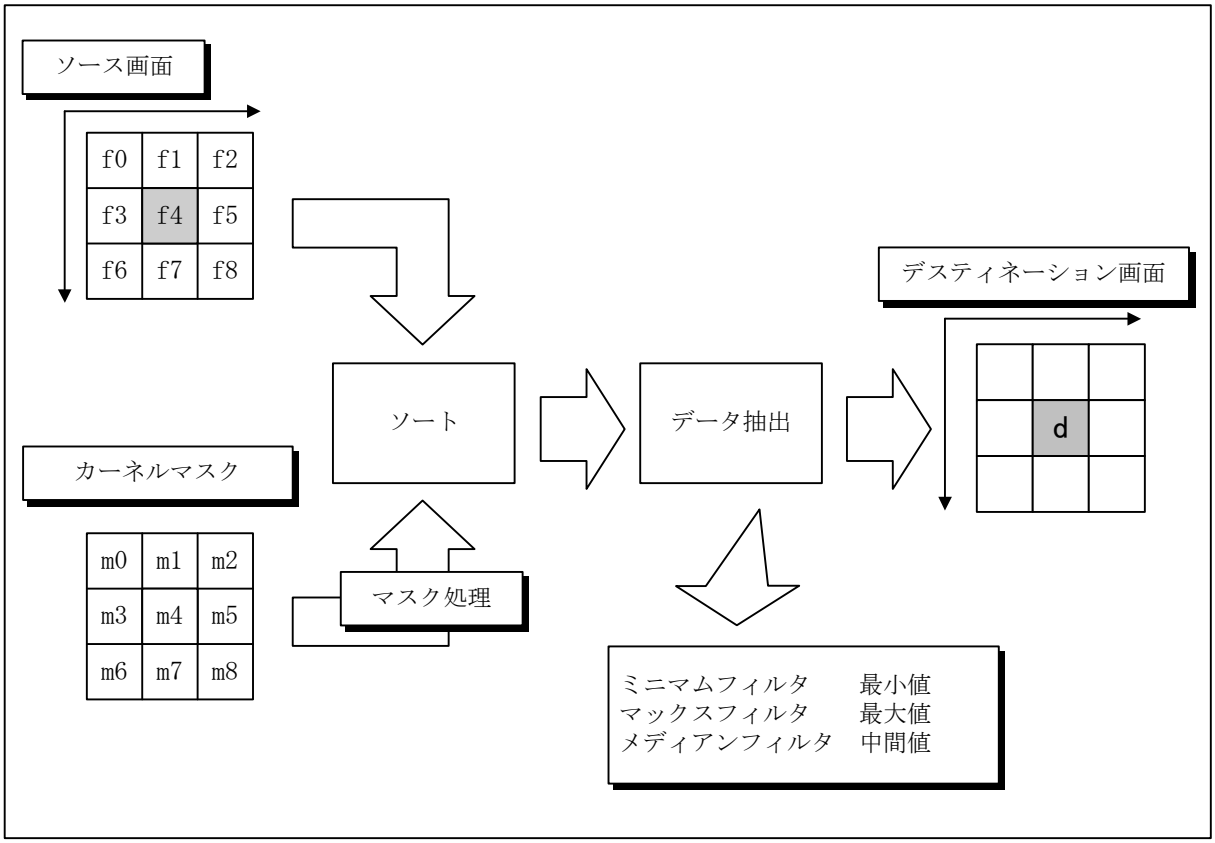


図5-9-1　ランクフィルタ

5.10 ラベリング

ラベリング処理では、２値画像に対して連結している物体ごとに番号を付けます。ラベリング処理の種類には、すべての物体をラベリングするものと、一定範囲内の面積物体にのみラベリングするものがあり、それぞれにオプションとして白をラベリングするか、黒をラベリングするかが選択できます。

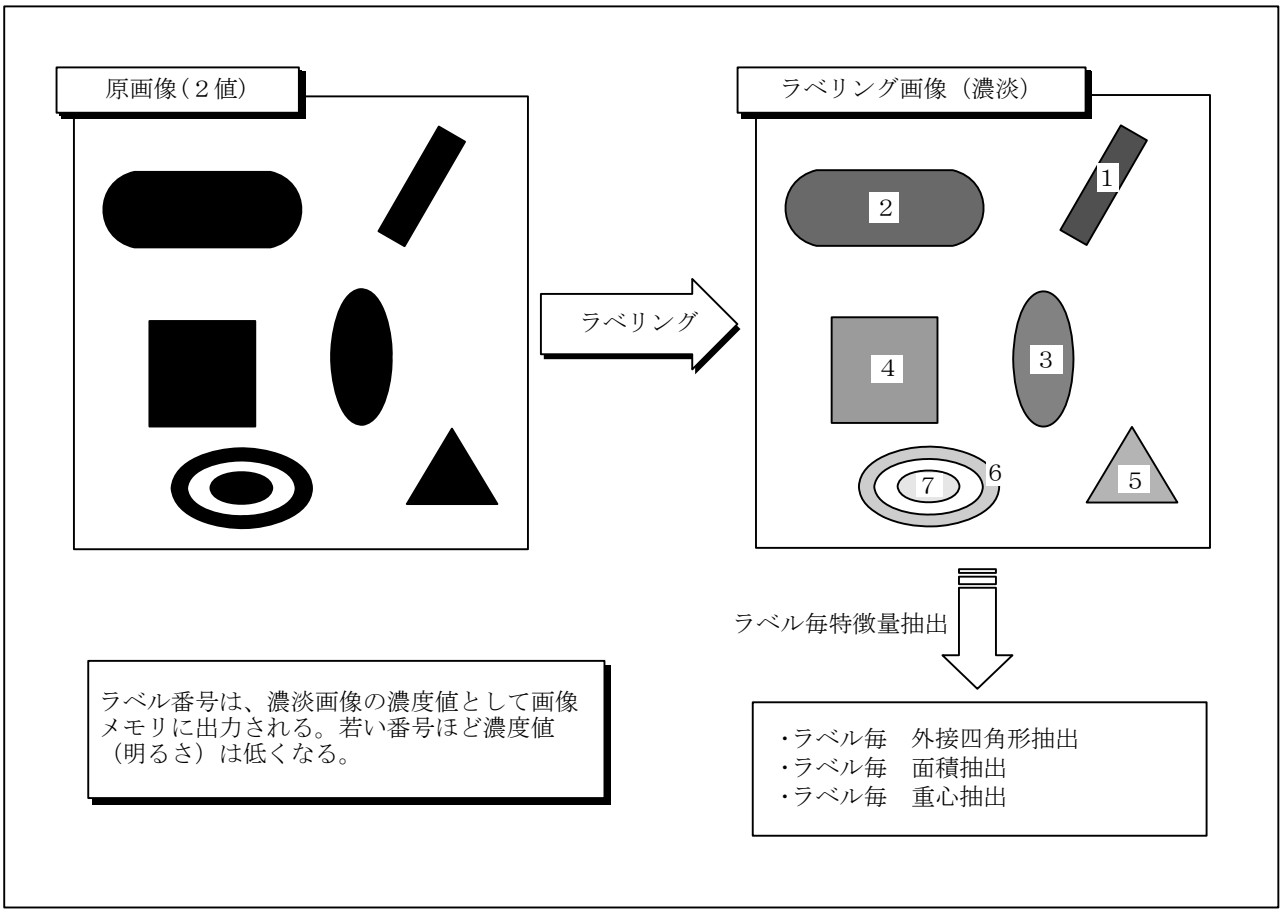


図5-10-1　ラベリング概要

ラベリング処理は、仮ラベル付け、合流対検出、真ラベル付けの３段階で行われます。

次にそれぞれの処理について説明します。
原画像図ａに対してラベリング処理を起動すると、画面左上から仮ラベル付けが始まります。ラスタスキャンにて番号付けを行うため、図ｂに示すように同じ物体を異なるラベルとして認識します。この段階では物体Ａに対して１、２、３の３つの仮ラベル番号が、物体Ｂに対しては４、５の２つの仮ラベル番号が割り当てられます。そこで次の段階として合流対検出を行います。合流対検出とは同じ物体に対して複数の仮ラベル番号が付いていないかを調べる処理であり、図ｂに対しては図ｃに示すように、１、２、３は同じ、４、５は同じという結果が得られます。この結果をもとに真ラベル付けを行うことにより、最終的に図ｄに示す結果が得られます。
なお、各々の処理には、
・仮ラベル付け番号：最大　１０２２番まで
・合流対　　：最大　１０２２件まで
・真ラベル　　：最大　２５５個まで
との制約があり、この値を超えるとオーバーフローとなり、正しい結果が得られません。

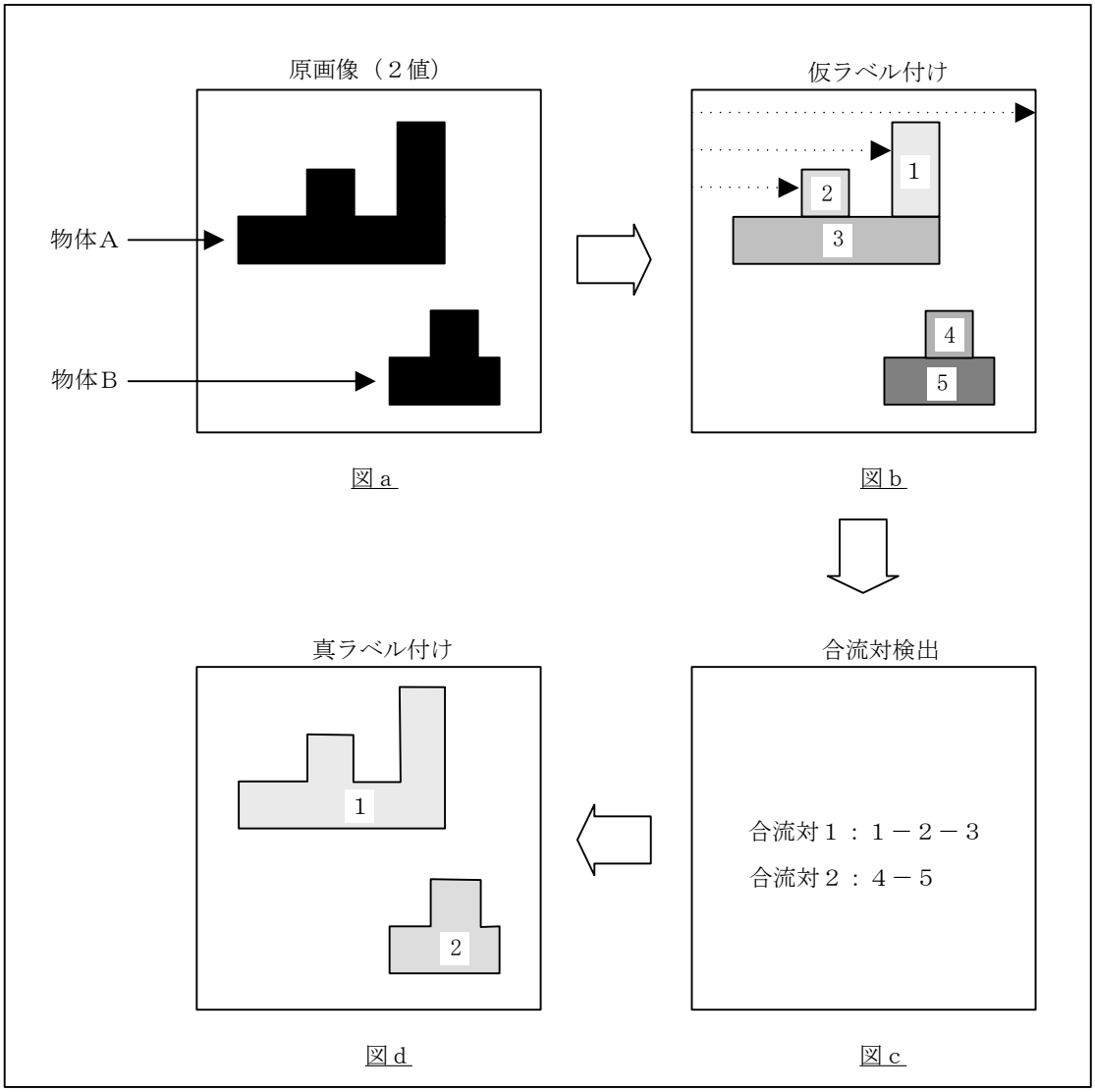


図5-10-2　ラベリング処理

5.11 ヒストグラム

ヒストグラム処理では、２値画像と濃淡画像に対し、最大／最小濃度、濃度頻度分布抽出等の統計処理が行えます。

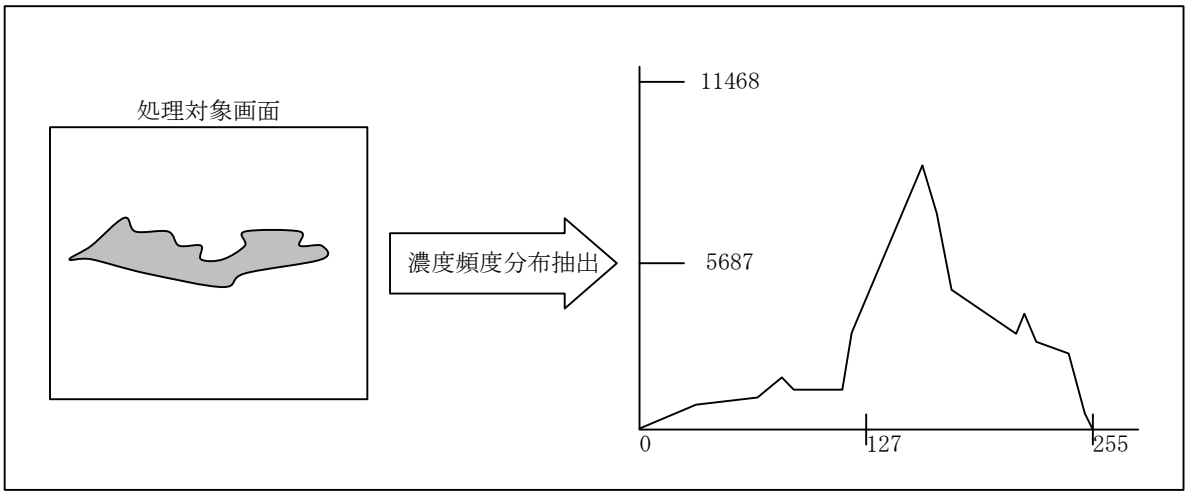


図5-11-1　濃度頻度分布抽出

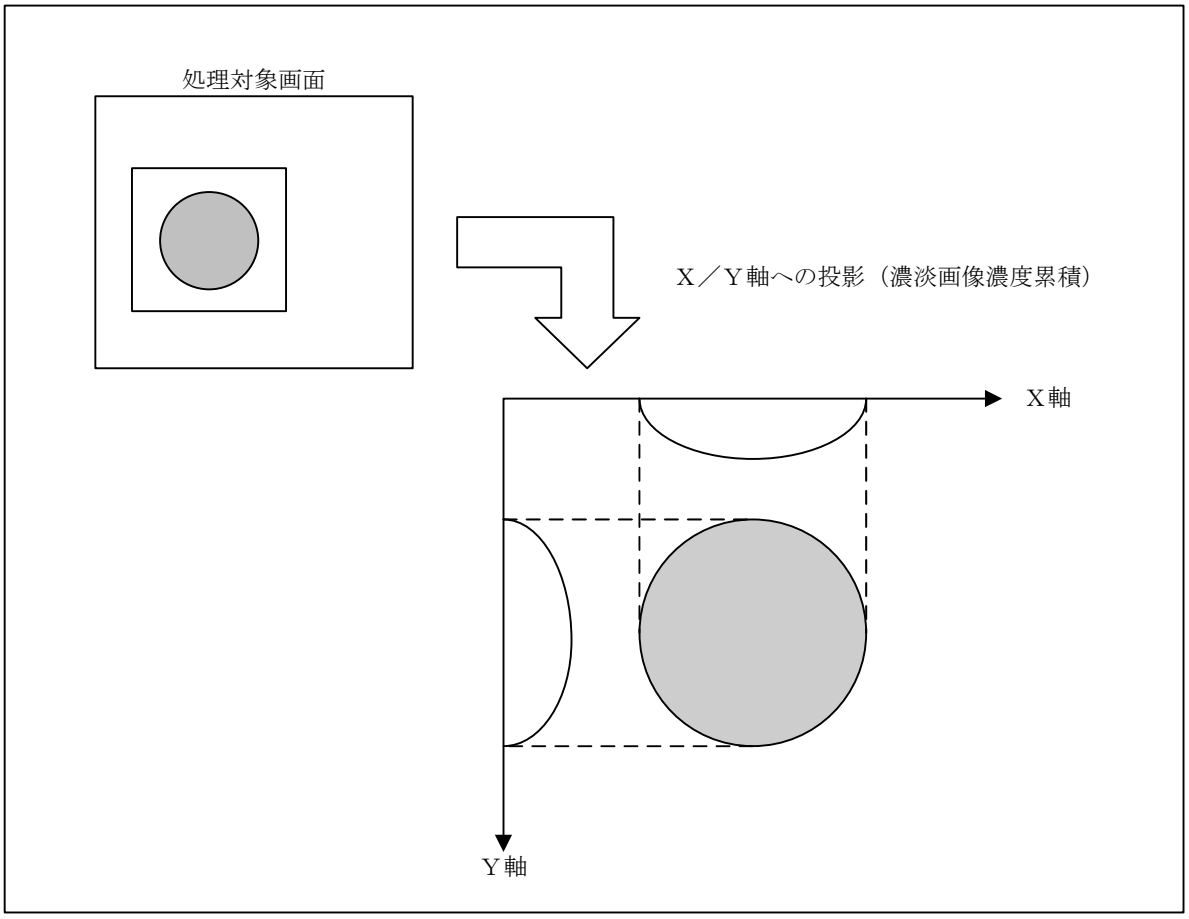


図5-11-2　X／Y軸への投影

ヒストグラム処理での座標系は、ウィンドウ相対であり、ウィンドウの始点が座標０になります。
よって、処理結果の座標はウィンドウ始点を基準とした座標であり、処理結果のテーブルはウィンドウ内の有効データがテーブルの先頭から格納されます。

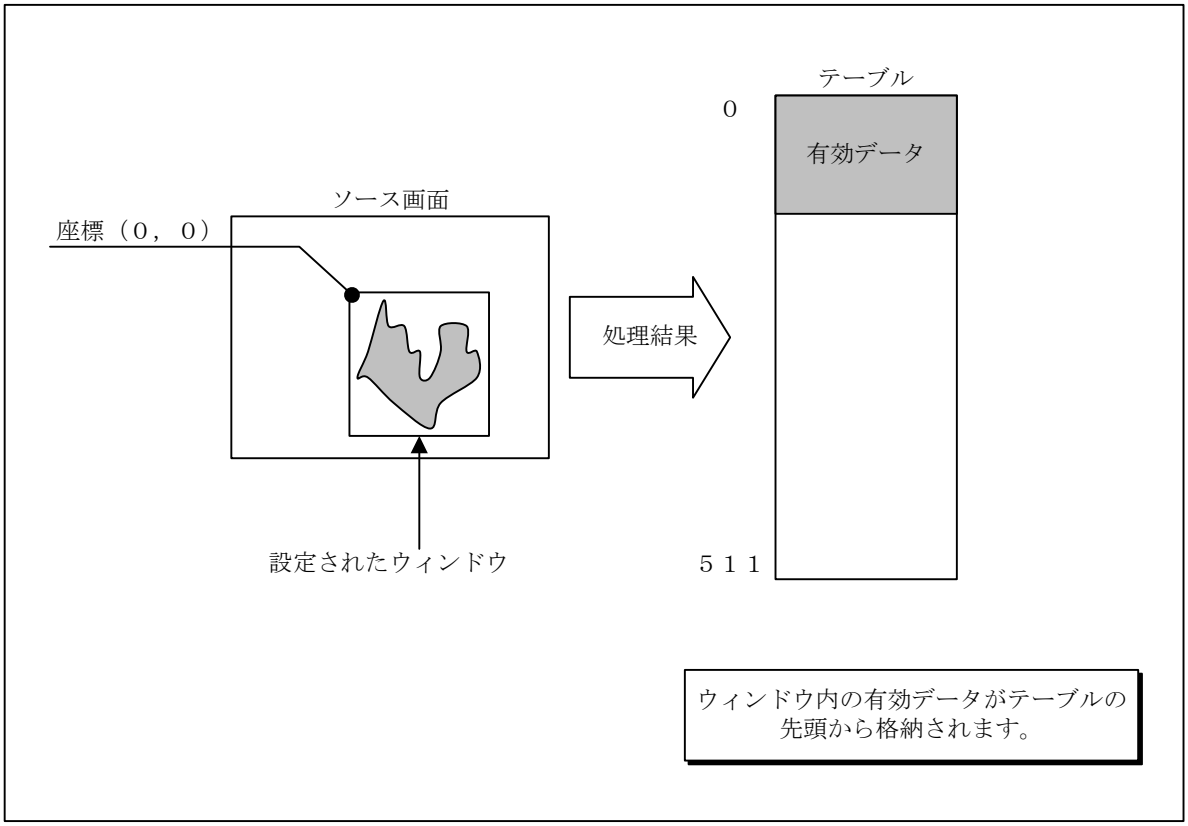


図5-11-3　処理結果テーブル

5.12 画像メモリアクセス

画像メモリアクセスコマンドでは、画像メモリのデータを１画素単位または、ブロック単位でアクセスすることができます。

5.12.1 画像メモリアクセス手順フロー

以下に、画像メモリアクセス手順フローを示します。

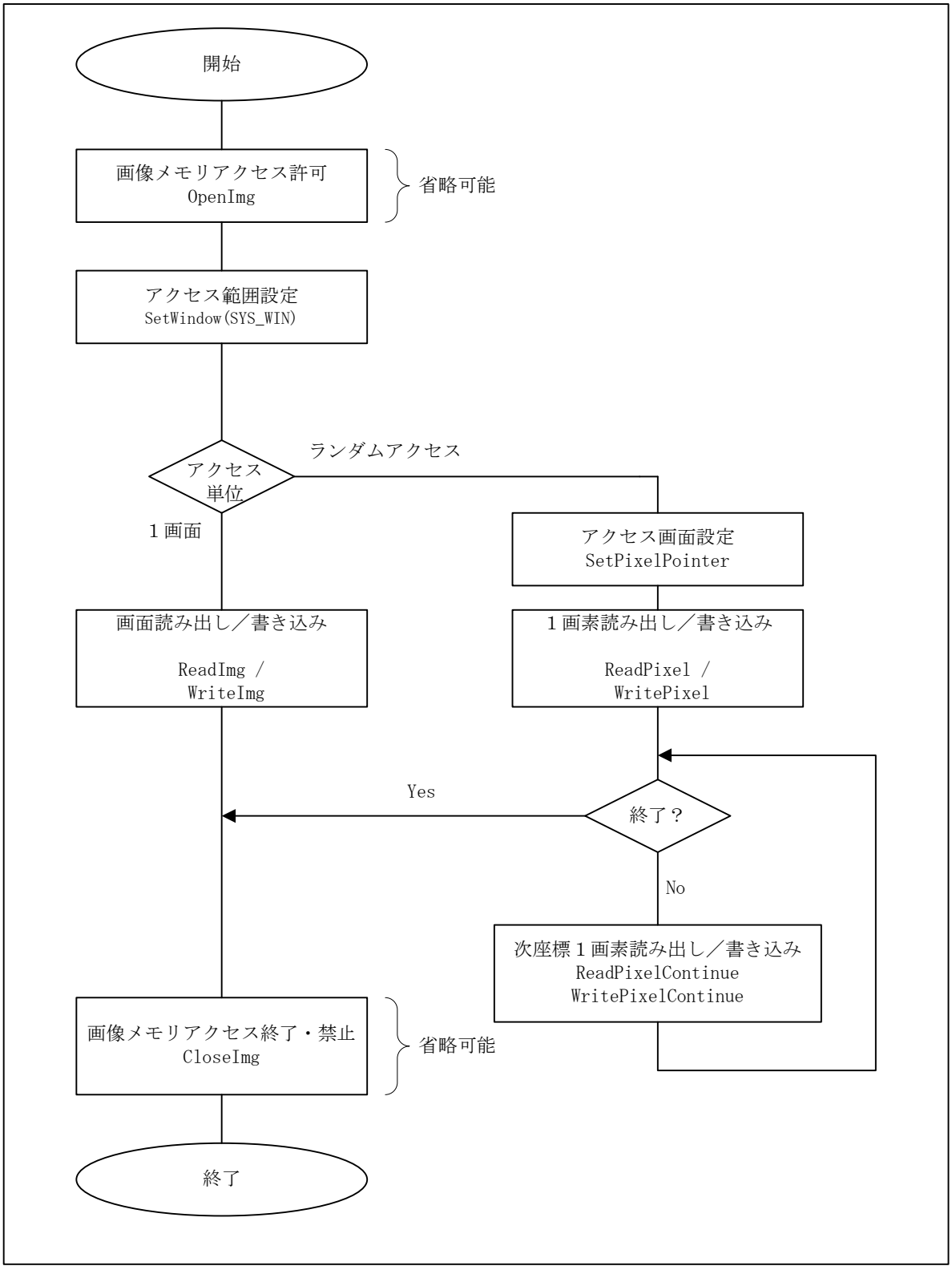


図5-12-1　画像メモリアクセス手順

5.12.2 ウィンドウの設定

画像メモリアクセスコマンドは任意のウィンドウが設定できます。設定するウィンドウの種類はSYS_WINです。ReadImgコマンドはSYS_WINで設定された矩形領域の画素データをローカルメモリにブロック単位に読込み、WriteImgコマンドはローカルメモリ上のデータをSYS_WINで設定された矩形領域の画像メモリにブロック単位に書込みます。
また、画像メモリ制御コマンドでの座標系はウィンドウ相対であり、ウィンドウの始点が座標０となります。

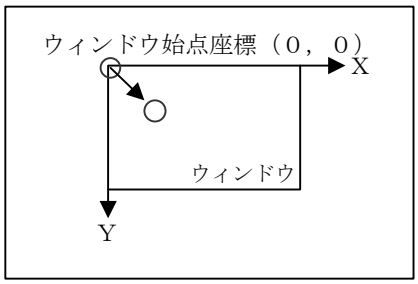


図5-12-2　ウィンドウ座標系

5.12.3 画面データタイプ

画像メモリアクセスを行う画面の画面データタイプは符号なし８ビット、符号付８ビット、２値です。なお、２値データ０は画素データ「０」、２値データ１は画素データ「２５５」です。

5.12.4 画像メモリ直接アクセス

本ライブラリでは、画像メモリ直接アクセスコマンドにより画像メモリの先頭アドレスを取得し、画像メモリを直接アクセスすることができます。
実際の処理では、OpenImgコマンドはパラメータで指定した画面の画面サイズでメモリを確保し、その領域に指定画面の画像データを格納します。OpenImgコマンドが返すアドレスはこの領域の先頭アドレスです。CloseImgDirectコマンドは、OpenImgDirectコマンドで確保したメモリの画像データをOpenImgDirectコマンドで指定した画面の画像メモリに書き戻してから、確保メモリを解放します。そのため、CloseImgDirectコマンドの実行で画像データの変更が反映されます。OpenImgDirectコマンド実行後は必ずCloseImgDirectコマンドを実行して下さい。
以下に、画像メモリ直接アクセス手順フローを示します。

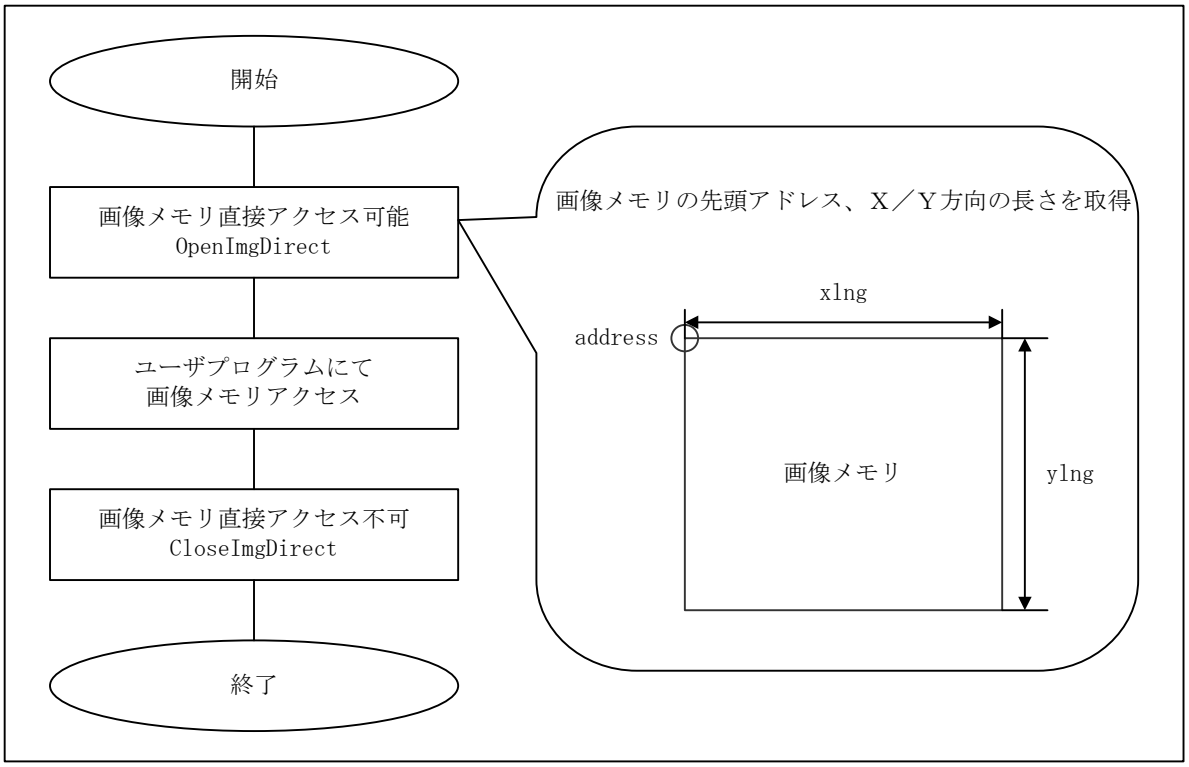


図5-12-3　画像メモリ直接アクセス手順

5.13 2値パイプラインフィルタ

２値パイプラインフィルタコマンドは、２値画像に対し、３×３カーネルを用いてノイズ除去、膨張、収縮、輪郭抽出の４つの画像処理を組み合わせて、最大８段までのフィルタ処理のパイプライン処理を行います。また、８段パイプラインを４段×２の構成にして、４段目の出力結果を論理演算して出力することもできます。

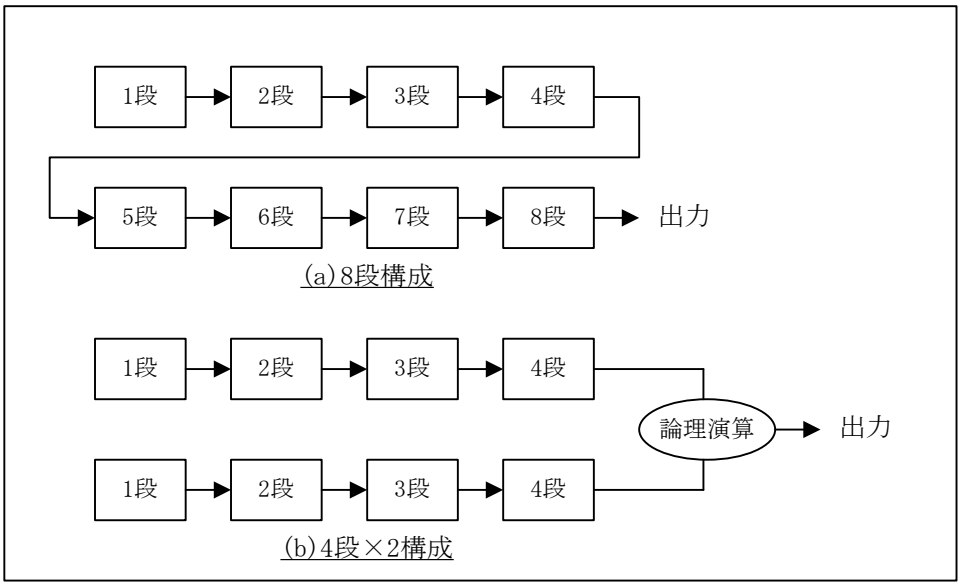
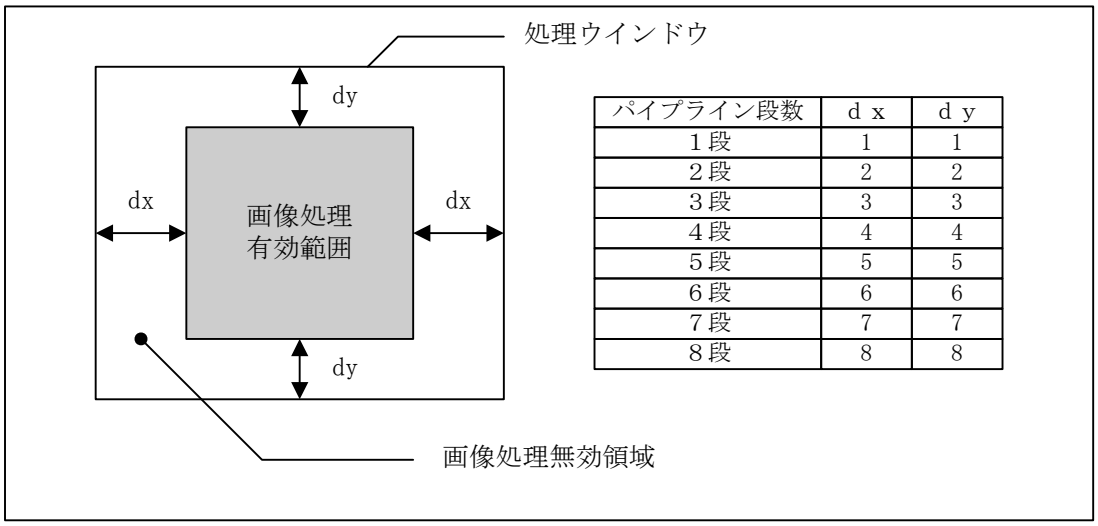


図5-13-1　２値パイプラインフィルタ

5.13.1 2値パイプラインフィルタ処理領域

３×３カーネルを用いて画像処理を行う為、画像処理対象エリアのエッジ部分（周辺）に画像処理結果無効の範囲が生じます。また、この画像処理無効範囲は、パイプラインの段数によって決まります。
段数による無効範囲を以下に示します。



パイプライン段数	ｄｘ	ｄｙ
１段	1	1
２段	2	2
３段	3	3
４段	4	4
５段	5	5
６段	6	6
７段	7	7
８段	8	8

図5-13-2　処理領域

5.14 パイプライン制御

画像処理プロセッサは、画像処理、２値化、ヒストグラムのプロセッサを独立して持っています。そして、その３つの処理を画像処理→２値化→ヒストグラムの順番でパイプライン処理を行うことが可能であり、画像処理、２値化、ヒストグラムプロセッサの処理を１画面の処理時間で実行できます。

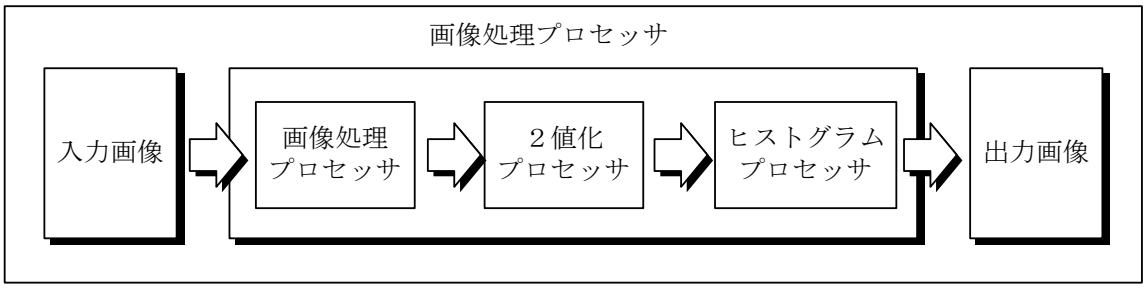


図5-14-1　画像処理プロセッサパイプライン処理部

また、下記の組み合わせでパイプライン処理が実行可能です。
各処理については、「パイプライン処理の実行条件」を参照ください。

（１）画像処理(濃淡)　＋　２値化　＋　ヒストグラム処理(２値)
（２）画像処理(濃淡)　＋　２値化
（３）２値化　　＋　ヒストグラム処理(２値)
（４）画像処理(濃淡)　＋　ヒストグラム処理(濃淡)

5.14.1 パイプライン処理の実行方法

EnablePipeline(),DisablePipeline()コマンドでパイプラインモード制御を行います。

EnablePipeline()コマンドを実行すると、パイプラインモードになります。パイプラインモードでは、最初に発行された画像処理コマンドを実行せず、次のコマンド処理へ移行します。そのコマンドの処理がパイプラインで実行できない場合は、パイプライン制御により実行を待たされていた（ペンディング状態）画像処理コマンドを実行します。つまり、コマンドの処理がパイプラインでつながる場合、その時点での処理を行わずペンディング状態にして、コマンドバッファにバッファリングを行い、パイプラインのチェインが切れた時点で処理を行うわけです。

DisablePipeline()コマンドを実行すると、コマンドバッファにバッファリングされていたペンディング処理を実行し、パイプラインモードを解除し、通常の処理に戻ります。

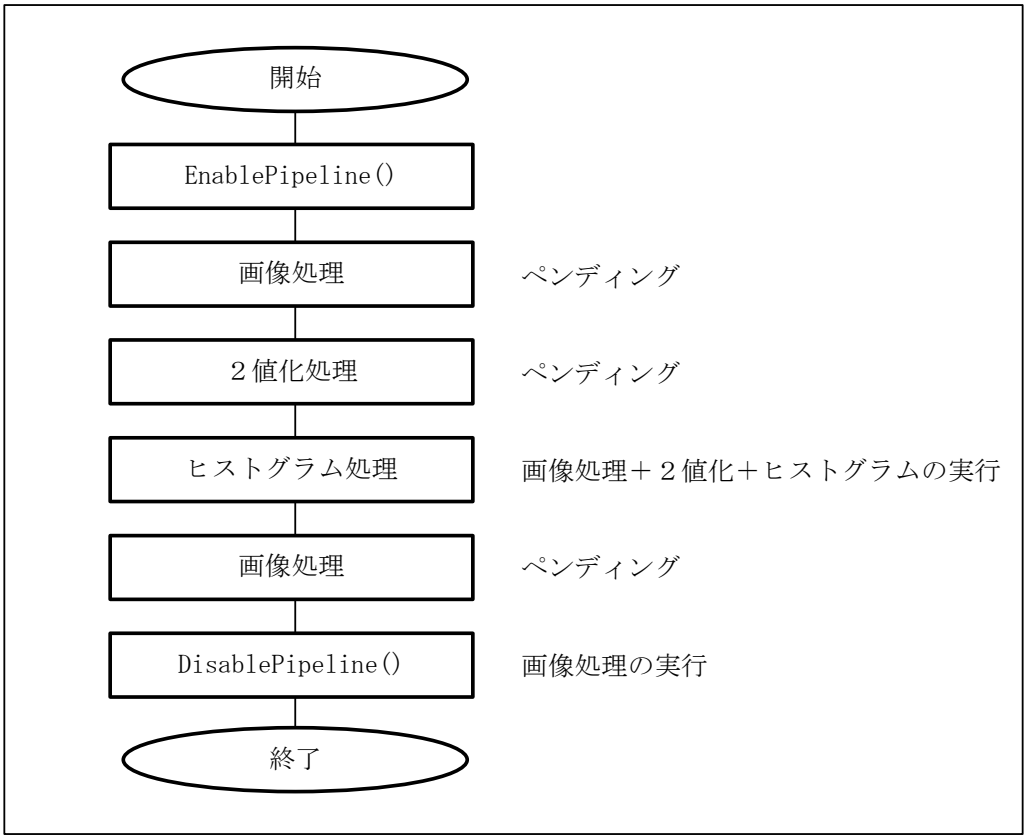


図5-14-2　パイプライン処理の動作

5.14.2 パイプライン処理の実行条件

パイプライン処理が行われるには、以下の条件があります。

(1) 画像処理の対象画面

先に実行されるコマンドのデスティネーション画面と、次に実行されるコマンドのソース画面が同一であることです。

(2) パイプラインが構成可能な処理と順番

下の表にパイプライン処理構成として、パイプラインが構成可能な処理と順番を示します。
他の組み合わせについては結果を保証いたしません。

表5-14-1　パイプライン処理構成

No.	処理と順番	備考
1	画像処理(濃淡)　⇒ ヒストグラム(濃淡)	注意事項２
2	画像処理(２値)　⇒ ヒストグラム(２値)	
3	２値化　⇒ ヒストグラム(２値)	
4	画像処理(濃淡)　⇒ ２値化 ⇒ ヒストグラム(２値)	注意事項５

パイプライン処理を構成するコマンドである画像処理(濃淡)、画像処理(２値)、２値化処理、ヒストグラム処理(濃淡)、ヒストグラム(２値)、 および、パイプライン処理とは無関係に動作するコマンドを以下に示します。

表5-14-2　画像処理(濃淡)コマンド

項目	関数名	備考
画像転送／アフィン変換	IP_Copy	分離YUV画面はﾊﾟｲﾌﾟﾗｲﾝ不可
	IP_ZoomOut	注意事項２
	IP_ZoomOutExt	注意事項２
	IP_ZoomIn	注意事項２
	IP_ZoomInExt	注意事項２
画素変換	IP_Invert	
	IP_Minus	
	IP_Abs	
	IP_AddConst	
	IP_SubConst	
	IP_SubConstAbs	
	IP_MultConst	
	IP_MinConst	
	IP_MaxConst	
	IP_ConvertLUT	
	IP_ShiftDown	
画像間算術演算	IP_Add	
	IP_Sub	
	IP_SubAbs	
	IP_Comb	
	IP_CombAbs	
	IP_Mult	
	IP_Average	
	IP_Min	
	IP_Max	
	IP_SubConstAbsAdd	
	IP_SubConstMultAdd	
	IP_SubConstMult	
	IP_CombDrop	

画像間論理演算	IP_And	
	IP_Or	
	IP_Xor	
	IP_InvertAnd	
	IP_InvertOr	
	IP_Xnor	
コンボリューション	IP_SmoothFLT	
	IP_EdgeFLT	
	IP_EdgeFLTAbs	
	IP_Lapl4FLT	
	IP_Lapl8FLT	
	IP_Lapl4FLTAbs	
	IP_Lapl8FLTAbs	
	IP_LineFLT	
	IP_LineFLTAbs	
ミニ／マックスフィルタ	IP_MinFLT	
	IP_MinFLT4	
	IP_MinFLT8	
	IP_MaxFLT	
	IP_MaxFLT4	
	IP_MaxFLT8	
	IP_LineMinFLT	
	IP_LineMaxFLT	
ランクフィルタ	IP_RankFLT	
	IP_Rank4FLT	
	IP_Rank8FLT	
	IP_MedFLT	
	IP_Med4FLT	
	IP_Med8FLT	
２値マッチングフィルタ	IP_BinMatchFLT	

表5-14-3　画像処理(２値)コマンド

項目	関数名	備考
２値画像形状変換	IP_PickNoise4	
	IP_PickNoise8	
	IP_Outline4	
	IP_Outline8	
	IP_Dilation4	
	IP_Dilation8	
	IP_Erosion4	
	IP_Erosion8	
	IP_Thin4	
	IP_Thin8	
	IP_Shrink4	
	IP_Shrink8	
２値パイプラインフィルタ	IP_TrsPipelineFLT	

表5-14-4　２値化処理コマンド

項目	関数名	備考
２値化	IP_Binarize	
	IP_BinarizeExt	

表5-14-5　ヒストグラム処理(濃淡)コマンド

項目	関数名	備考
濃淡画像特徴量抽出	IP_ExtractGOFeatures	
	IP_Histogram	
	IP_HistogramShort	
	IP_ProjectGO	
	IP_ProjectGOonX	
	IP_ProjectGOonY	
	IP_ProjectGOMaxValue	
	IP_ProjectGOMinValue	
	IP_ProjectBlockGO	注意事項６
	IP_ProjectBlockGOMinMaxValue	注意事項６
ラベリング	IP_ExtractLORegionX	
	IP_ExtractLORegionY	
	IP_ExtractLOArea	
	IP_ExtractLOAreaExt	
	IP_ExtractLOGravity	

表5-14-6　ヒストグラム処理(２値)コマンド

項目	関数名	備考
２値画像特徴量抽出	IP_ExtractBOFeatures	
	IP_ProjectBO	
	IP_ProjectBORegionX	
	IP_ProjectBORegionY	
	IP_ProjectBlockBO	注意事項６

パイプライン処理と無関係に動作するコマンドには、ペンディングされているコマンドを実行（パージ）してから動作するコマンドと、ペンディングされているコマンドを実行せずに単独で動作するコマンドがあります。

表5-14-7　パイプライン処理と無関係に動作するコマンド　　[パージ]○:有 / ×:無

項目	関数名	パージ	備考
システム制御	InitIP	×	
	InitIPExt	×	
	ReadIPErrorTable	×	
	ClearIPError	×	
	SetIPDataType	×	
	IPLibVersion	×	
	DriverIoControl	×	
	DisableIPErrorMessage	×	
	EnableIPErrorMessage	×	
画像メモリ領域管理	AllocImg	×	
	AllocYUVImg	×	
	AllocCYUVImg	×	
	FreeImg	×	
	FreeAllImg	×	
	GetUVImgID	×	
	ReadImgTable	×	
	ReadYUVImgTable	×	
	ChangeImgDataType	×	
	SetWindow	×	注意事項７
	SetAllWindow	×	注意事項７
	ResetAllWindow	×	注意事項７
	EnableIPWindow	×	注意事項７
	DisableIPWindow	×	注意事項７
	ReadWindow	×	

映像入力	ActiveVideoPort	×	
	SetVideoFrame	×	
	SelectCamera	×	
	GetCamera	×	
	SetVFDelay	×	
	CaptureContinuous	×	
	StopCaptureContinuous	×	
	GetCameraReqScene	×	
	GetCameraResScene	×	
	SetConfigCamera	×	
	ExitIPCamera	×	
映像出力	SetConfigDisp	×	
	SetDispPalette	×	
	SelectDisp	×	
	DispCamera	×	
	DisableDisp	×	
	DispImg	×	
	NoDisp	×	
	DispOverlap	×	
	DisableOverlap	×	
	CombineYUV	○	
	SetConfigView	×	
	ExitIPView	×	
画像クリア	IP_ClearAllImg	○	
	IP_ClearImg	○	
	IP_Const	○	
画像転送／アフィン変換	IP_Zoom	○	
	IP_ZoomExt	○	
	IP_ZoomS	○	
	IP_Shift	○	
	IP_Rotate	○	
画素変換	WriteConvertLUT	○	
	IP_ShiftUp	○	
ラベリング	IP_Label4	○	
	IP_Label8	○	
	IP_Label4withAreaFLT	○	
	IP_Label8withAreaFLT	○	
	IP_Label4withAreaFLTSort	○	
	IP_Label8withAreaFLTSort	○	
濃淡画像特徴量抽出	IP_ProjectLabelGO	○	
	IP_ProjectLabelGOMinMaxValue	○	
	EnableRotateProject	×	
	DisableRotateProject	×	
画像メモリアクセス	OpenImg	○	
	OpenImgExt	○	
	CloseImg	○	
	ReadImg	○	
	WriteImg	○	
	SetPixelPointer	○	
	ReadPixel	×	
	WritePixel	×	
	ReadPixelContinue	×	
	WritePixelContinue	×	
	RefreshImg	○	
	OpenImgDirect	○	
	CloseImgDirect	○	
２値パイプラインフィルタ	SetTrsPipelineFLTMode	×	
パイプライン制御	EnablePipeline	×	
	DisablePipeline	○	

２値マッチングフィルタ	SetBinMatchTemplate	×	
正規化相関	SetCorrMode	×	
	DisableCorrMask	×	
	EnableCorrMask	×	
	SetCorrBreakThr	×	
	DisableCorrBreak	×	
	EnableCorrBreak	×	
	SetCorrControl	×	
	SetCorrTemplate	○	
	SetCorrTemplateExt	○	
	IP_Corr	○	
	IP_CorrPrecise	○	
グラフィックス	SetDrawMode	×	
	SetStringAttributes	×	
	RefreshGraphics	○	
	DrawString	○	
	DrawLine	○	
	DrawSegments	○	
	DrawLines	○	
	DrawRectangle	○	
	DrawPolygon	○	
	DrawArc	○	
画像ファイリング	LoadBMPFile	×	
	SaveBMPFile	×	

[注意事項]

１．パイプラインモードでパイプライン処理とは無関係に動作するコマンドを実行した場合、パイプライン有効／無効で処理の順序が変わるため、処理結果が一致しない場合があります。

２．パイプラインモードで以下のアフィン変換関数を実行した場合、パイプライン有効／無効で処理領域が変わるため、処理結果が一致しない場合があります。
パイプライン有効時には、アフィン変換関数実行後の処理領域はアフィン変換の結果領域になります。

IP_ZoomOut、IP_ZoomOutExt、IP_ZoomIn、IP_ZoomInExt

３．パイプラインモードで、１段目の処理が画像間算術演算や画像間論理演算のようにソース０画面とソース１画面の２枚のソース画面を使用する場合、２段目以降の処理範囲にSRC1_WINが影響します。これによってパイプライン有効／無効で、処理結果が一致しない場合があります。

４．パイプラインモード中の以下コマンドによるウィンドウ操作は禁止します。

SetWindow、SetAllWindow、ResetAllWindow、
EnableIPWindow、DisableIPWindow

５．ヒストグラム処理（濃淡）の対象となる２値画像は、符号なし濃淡画像として処理されます。

６．２段目で二値化を行うパイプライン処理では、処理方式上、１段目の画像処理の結果画像が生成されません。このため、前述の場合は１段目の画像処理のデスティネーション画面のデータタイプを不定画面とすることによって、有効データが格納されている画面と区別しています。
不定画面は、デスティネーション画面に使用することは可能ですが、ソース画面として使用するとエラーになります。
ChangeImgDataTypeコマンドで不定画面を他の画面データタイプに変更することは可能ですが、ソースとして参照した場合にパイプラインの有効/無効で異なる結果となるため推奨しません。
以下に不定画面の発生例を示します。ImgSrc0とImgSrc1を加算処理後、２値化処理を実行し、結果をImgDstに格納する処理です。ImgIDは加算処理の処理結果が格納されず、不定画面になります。

```
EnablePipeline();                    // パイプラインモード設定
IP_Add(ImgSrc0, ImgSrc1, ImgID);     // 加算処理
IP_Binarize(ImgID, ImgDst, thr);     // ２値化処理
DisablePipeline();                   // パイプラインモード解除
```

７．パイプラインモードで、２値化に先行する画像処理（濃淡）のデスティネーション画面データタイプが符号付となる論理演算である場合、パイプライン処理しないで実行（パージ）されます。

８．パイプラインモードで、画面領域を分割して処理する以下のコマンドは、Ｘ方向分割数が１７～３２かつＹ方向分割数が１６より大きい場合には、ペンディングされているコマンドを実行（パージ）してから濃度累積処理を行うため、パイプライン処理になりません。

IP_ProjectBlockGO、IP_ProjectBlockGOMinMaxValue、IP_ProjectBlockBO

5.14.3 パイプライン処理の処理例

以下にパイプライン可能／不可能の例を３例示します。

(1) 画像処理 ＋ 2値化

```
＜パイプライン動作可能な場合＞
EnablePipeline();                       パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);        IP_Addコマンドはペンディング
IP_Binarize(ImgID, ImgDst, thr);        IP_AddとIP_Binarizeをパイプライン実行
DisablePipeline();                      パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Binarizeのソース画面が同じなので、
  パイプライン処理を行います。
```

```
＜パイプライン動作不可能な場合＞
EnablePipeline();                       パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);        IP_Addコマンドはペンディング
IP_Binarize(ImgSrc0, ImgDst, thr);      パイプライン実行できず、IP_Addを実行後に
                                        IP_Binarizeを実行
DisablePipeline();                      パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Binarizeのソース画面が異なるので、
  パイプライン処理できません。
```

(2) 画像処理 ＋ ヒストグラム

```
＜パイプライン動作可能な場合＞
EnablePipeline();                               パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);                IP_Addコマンドはペンディング
IP_Histogram(ImgID, &Tbl, &Regs, opt);          IP_AddとIP_Histogramを
                                                パイプライン実行
DisablePipeline();                              パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Histogramのソース画面が同じなので、
  パイプライン処理を行います。
```

```
＜パイプライン動作不可能な場合＞
EnablePipeline();                               パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);                IP_Addコマンドはペンディング
IP_Histogram(ImgSrc0, &Tbl, &Regs, opt);        パイプライン実行できず、IP_Addを
                                                実行後にIP_Histogramを実行
DisablePipeline();                              パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Histogramのソース画面が異なるので、
  パイプライン処理できません。
```

(3) 画像処理 ＋ 2値化 ＋ ヒストグラム

```
＜パイプライン動作可能な場合＞
EnablePipeline();                          パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);           IP_Addコマンドはペンディング
IP_Binarize(ImgID, ImgID2, thr);           IP_Binarizeコマンドはペンディング
IP_ProjectBO(ImgID2, &TblX, &TblY);        IP_AddとIP_Binarizeと
                                           IP_ProjectBOをパイプライン実行
DisablePipeline();                         パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Binarize のソース画面、IP_Binarize
  のデスティネーション画面とIP_ProjectBOのソース画面が同じなので、パイプライン処理を行い
  ます。
```

```
＜パイプライン動作不可能な場合＞
EnablePipeline();                          パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);           IP_Addコマンドはペンディング
IP_Binarize(ImgID, ImgID2, thr );          IP_Binarizeコマンドはペンディング
IP_ProjectBO(ImgID3, &TblX, &TblY);        IP_BinarizeとIP_ProjectBOが
                                           パイプライン実行できないため、IP_Add
                                           とIP_Binarizeをパイプライン実行後に
                                           IP_ProjectBOを実行
DisablePipeline();                         パイプラインモード終了

・IP_Binarizeのデスティネーション画面とIP_ProjectBOのソース画面が異なるので、
  パイプライン処理できません。
```

5.14.4 パイプライン処理による不定画面

パイプライン処理で、結果が格納されずデータが不定になった画面を不定画面と言います。不定画面が発生する手順を除いて、不定画面を画像処理ソース画面として使用するとエラーになります。
また、不定画面をデスティネーション画面に使用することは可能です。

```
＜不定画面の発生例＞
ImgSrc0とImgSrc1をADD処理後、２値化処理を実行し、結果をImgDstに格納します。
このとき、ImgIDには処理結果が格納されず、ImgIDは不定画面となります。

EnablePipeline();
IP_Add(ImgSrc0, ImgSrc1, ImgID);
IP_Binarize(ImgID, ImgDst, thr);
DisablePipeline();
```

5.15 2値マッチングフィルタ

２値マッチングフィルタコマンドは、２値画像に対してテンプレートでマッチングフィルタ処理を行い、その処理結果を出力します。

入力画像に対し、９×９画素のテンプレートでＡＮＤ／ＸＮＯＲのマッチングフィルタ処理を行います。処理結果は、”１”となる画素の総数となります。

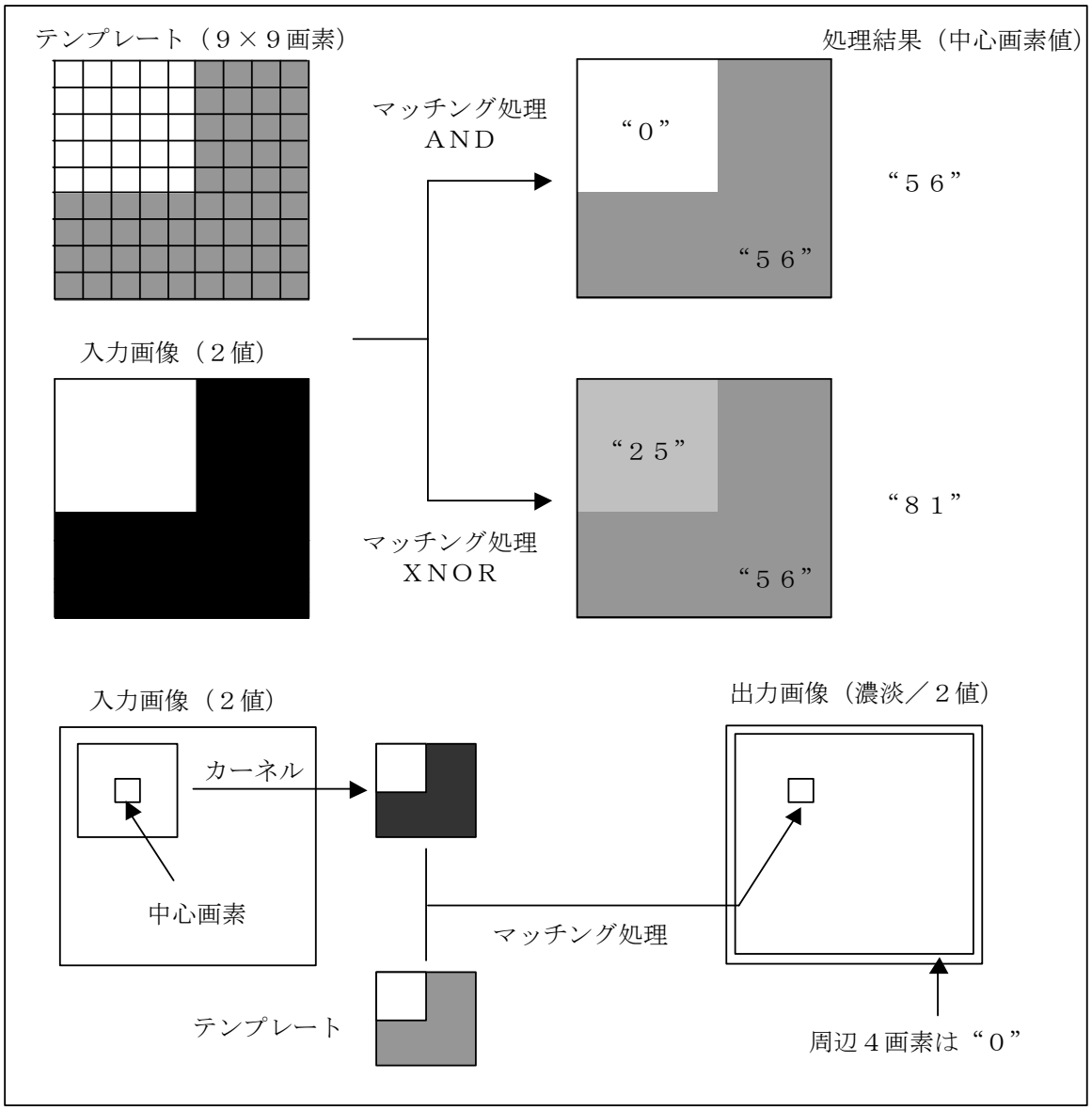


図5-15-1　２値マッチングフィルタ処理

２値マッチングフィルタ処理は、９×９テンプレートを用いて画像処理を行うため、画像処理対象領域の周辺４画素に画像処理結果無効の領域が生じます。この画像処理無効領域は”０”となります。

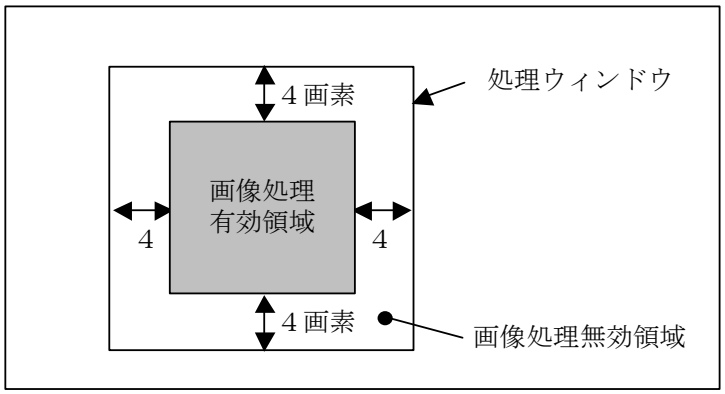


図5-15-2　２値マッチングフィルタ領域

5.16 正規化相関

正規化相関処理とは、濃淡画像によるパターンマッチングのことであり、ユーザの登録した濃淡テンプレートを、処理対象の画面の中から探し当てる（座標、相関値）ことができます。

正規化相関では、テンプレートと対象画面間で下記演算処理を行います。相関値（ｒ）は０～１の値を取り、濃淡テンプレート（ｇ）とサーチ対象画面（ｆ）が完全に一致すると１となります。

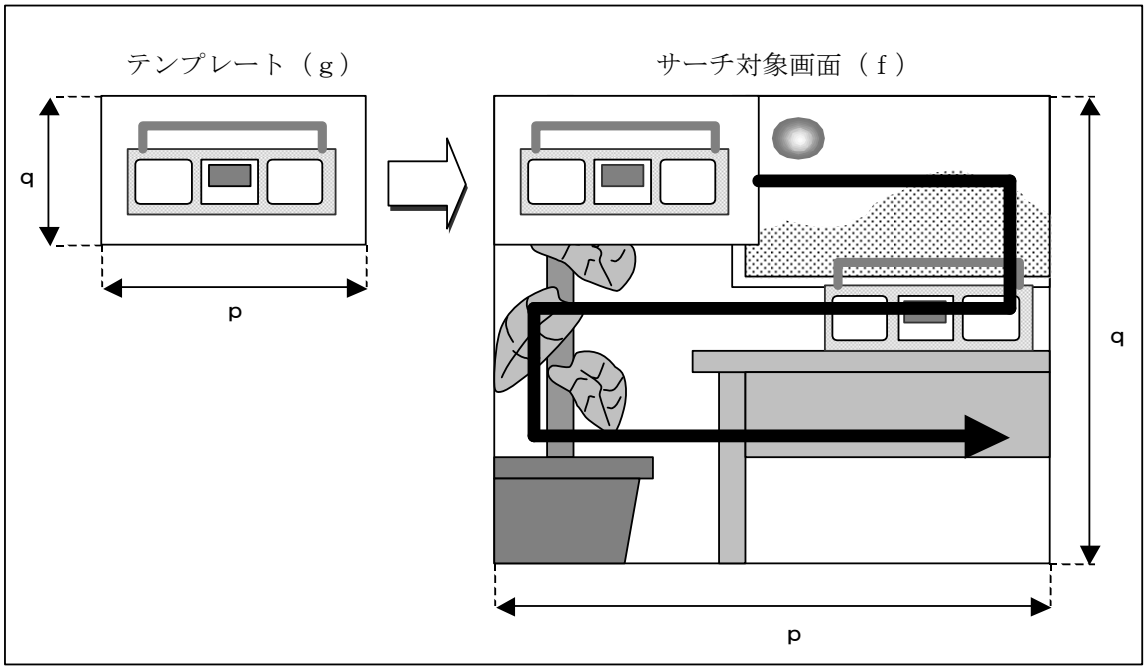


図5-16-1　正規化相関の概要

$$r^2 = \frac{\{ N \sum_{u}^{p} \sum_{v}^{q} f(u,v)g(u,v) - \sum_{u}^{p} \sum_{v}^{q} f(u,v) \times \sum_{u}^{p} \sum_{v}^{q} g(u,v) \}^2}{[N \sum_{u}^{p} \sum_{v}^{q} f(u,v)^2 - \{ \sum_{u}^{p} \sum_{v}^{q} f(u,v) \}^2] [N \sum_{u}^{p} \sum_{v}^{q} g(u,v)^2 - \{ \sum_{u}^{p} \sum_{v}^{q} g(u,v) \}^2]}$$

上記の式のうち、以下の項の計算はハードウェアによって局所並列的に実行されます。
相関値は、これらの結果を用いてソフトウェアで計算することにより求められます。

(1) $\sum_{u}^{p} \sum_{v}^{q} f(u,v)$　　(2) $\sum_{u}^{p} \sum_{v}^{q} g(u,v)$　　(3) $\sum_{u}^{p} \sum_{v}^{q} f(u,v)^2$　　(4) $\sum_{u}^{p} \sum_{v}^{q} g(u,v)^2$

(5) $\sum_{u}^{p} \sum_{v}^{q} f(u,v)\, g(u,v)$

ｒ：相関値
ｕ：Ｘ座標
ｖ：Ｙ座標
Ｎ：テンプレートの有効画素数

5.16.1 正規化相関実行手順

正規化相関処理の処理手順は、

①テンプレート登録
②サーチモード指定（サーチ個数、サーチ方向やサーチ精度等）
③正規化相関実行

の３段階です。

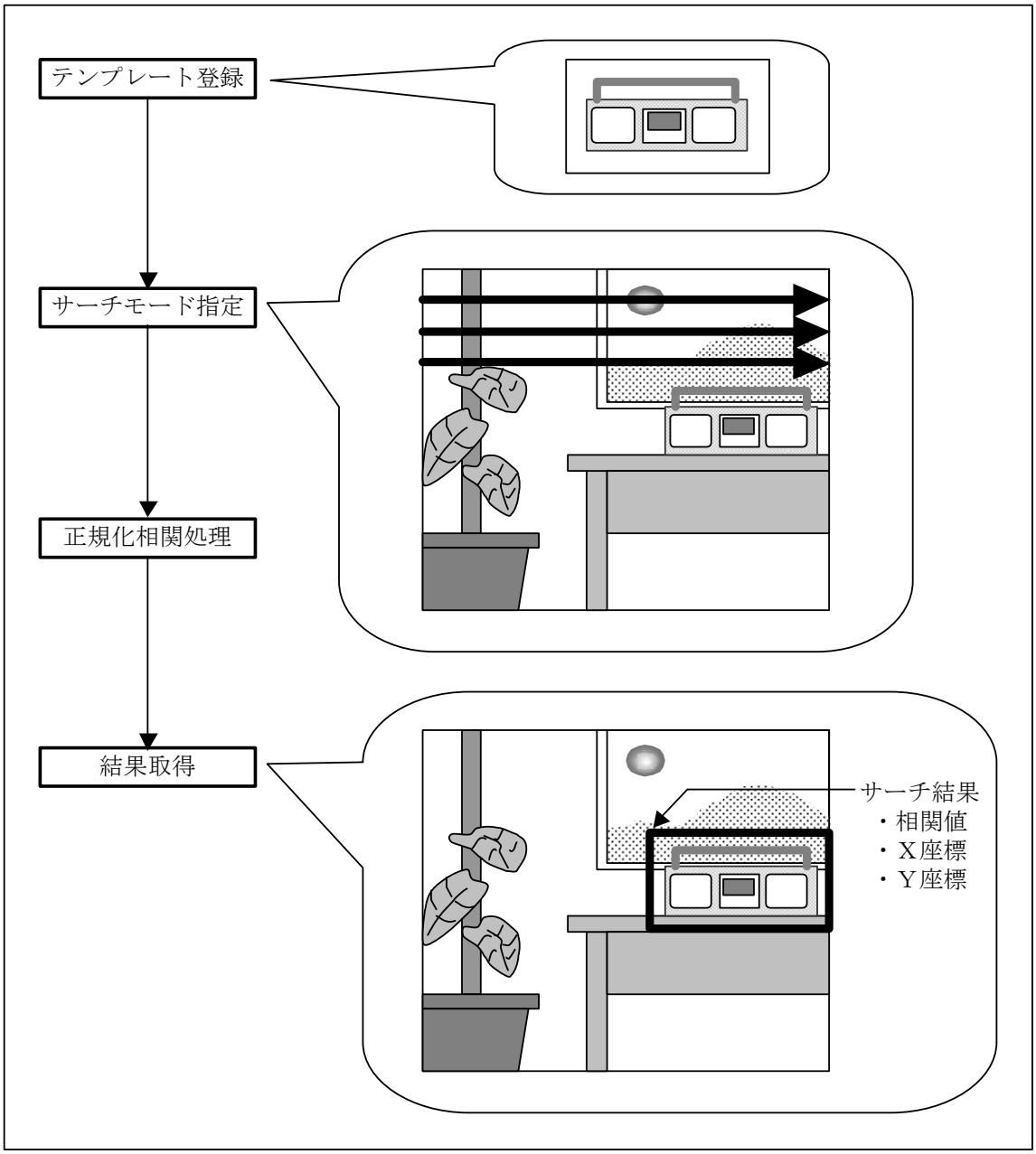


図5-16-2　正規化相関実行手順

以下に正規化相関の実行手順フローを示します。

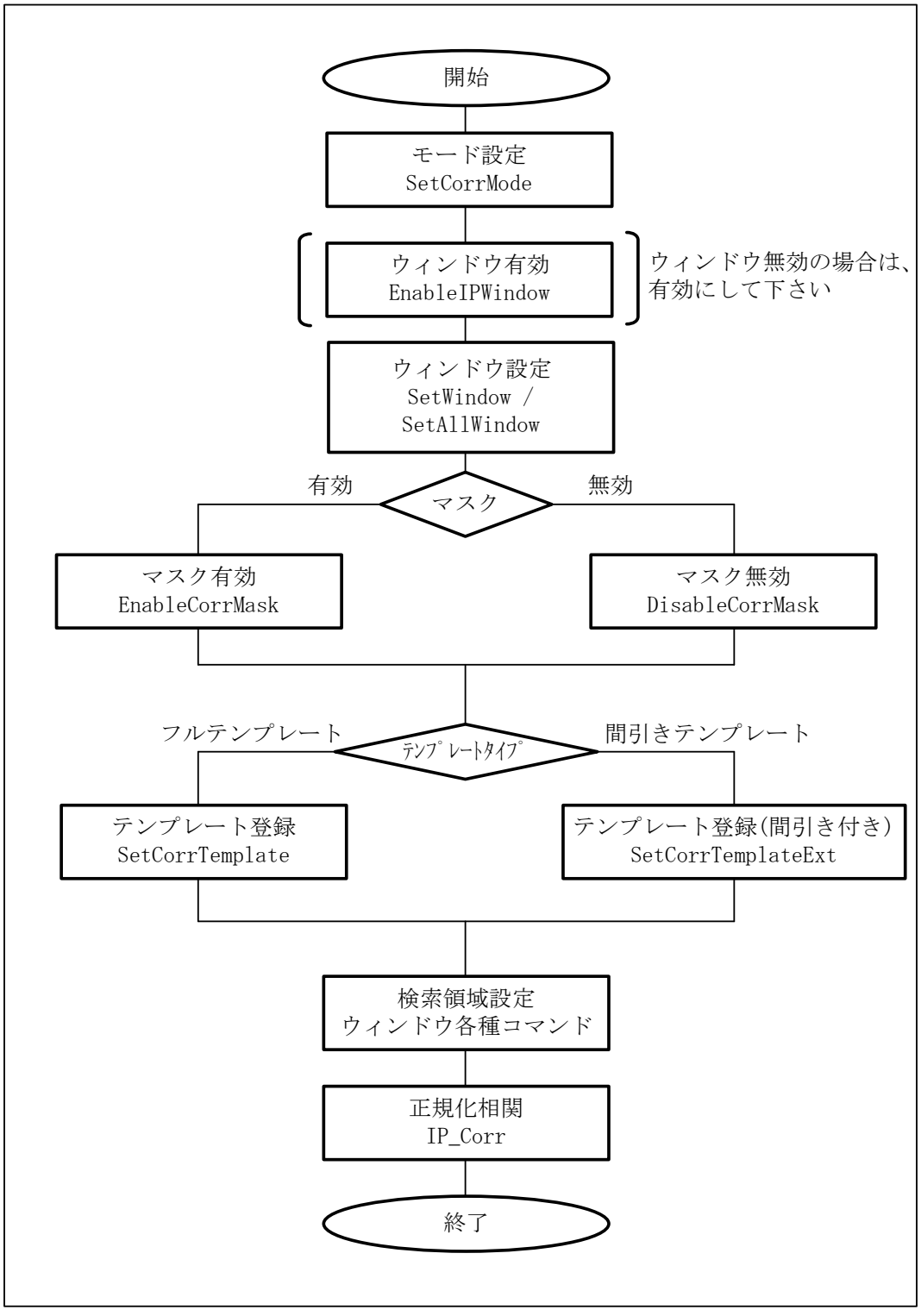


図5-16-3　正規化相関実行手順フロー

5.16.2 テンプレートタイプ

正規化相関のテンプレートには、２種類のタイプがあります。

（1）フルテンプレート

SetCorrTemplateコマンドで登録されたテンプレート。フルテンプレートは、IP_Corr実行時に、処理対象画面内全画素を処理する（フルサーチ）ので、高精度な処理が可能です。

（2）間引きテンプレート

SetCorrTemplateExtコマンドで登録されたテンプレートです。間引きテンプレートは、Ｘ／Ｙ方向の間引き率を登録でき、この間引き率により、テンプレートデータおよび処理対象画面を間引いて演算処理するので、精度はフルテンプレートに対して落ちるが演算回数が減るため、高速処理が可能です。

処理対象画像に応じて、これら２つのテンプレートタイプを評価し使用します。

5.16.3 正規化相関マスク

正規化相関マスクは、テンプレート内で濃度０の画素を処理対象外にします。よって、矩型でない任意の形をテンプレートとしたい場合に有効です。矩型のテンプレート内で必要物体部以外を０で塗りつぶしておき、IP_Corrを実行する前に、EnableCorrMaskを実行すれば、０画素部分は処理されないため、任意形のテンプレートで処理したことと同じになります。
正規化相関マスクはEnableCorrMaskコマンドで設定を行います。

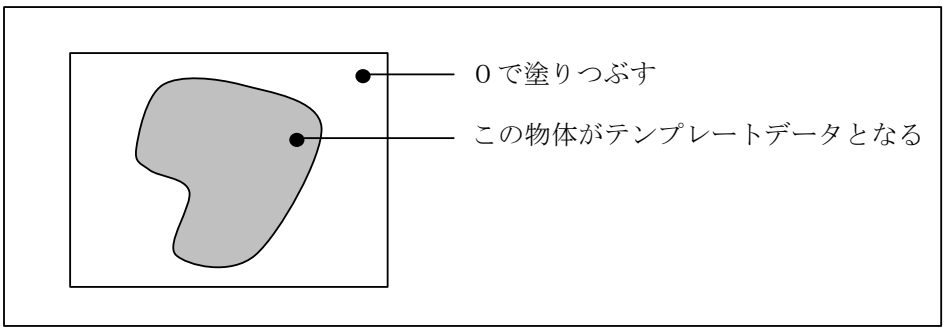


図5-16-4　正規化相関マスク

5.16.4 相関演算途中打ち切りによるサーチの高速化

正規化相関の演算として画像処理プロセッサによりテンプレートカーネル内の積和演算を行っていますが、その他にもう一つ差分累積演算を行う演算器を持っています。これは、テンプレートカーネル内の画像と差分累積演算を行い累積誤差が指定しきい値よりも大きくなったとき、正規化相関の積和演算の処理を途中で打ち切るためのものです。正規化相関の積和演算の処理を途中で打ち切ることにより、正規化相関サーチの高速化を図ることができます。
サーチ対象画像がテンプレート画像とあきらかに異なる場合、相関演算実行中に累積誤差が非常に大きくなると考えられます。そこで、この累積誤差があるしきい値を超えた時点で、ミスマッチと判断し、途中で演算を中止させます。途中で演算を中止することにより処理の高速化を図ることができますが、照明の変動による許容値が低くなります。

（1）しきい値の設定

しきい値とモードの設定はSetCorrBreakThrコマンドで行います。
実際に演算を途中で打ち切るための評価値である累積誤差しきい値は、

累積誤差しきい値＝ ｔｈｒ × テンプレート画素数

で計算されます。
また、ＳＳＤＡ法による自動しきい値変更モードもサポートしています。これは、常に最小をしきい値として、演算の打ち切りを行います。この場合、照明変動には弱くなります。

（2）相関演算打ち切りの設定

相関演算途中打切りを行う場合は、IP_Corrコマンドを実行する前にEnableCorrBreakコマンド、DisableCorrBreakコマンドで相関演算途中打ち切りモードを設定します。画像処理コマンドシステムの初期設定は、相関演算途中打ち切り無効になっています。

5.17 グラフィックス

グラフィックス処理では、文字列、図形描画等を行うことができます。本コマンドでの座標系は画像メモリアクセス（SYS_WIN）ウィンドウ相対であり、ウィンドウ始点が座標０となります。
以下にグラフィックス処理の実行手順フローを示します。

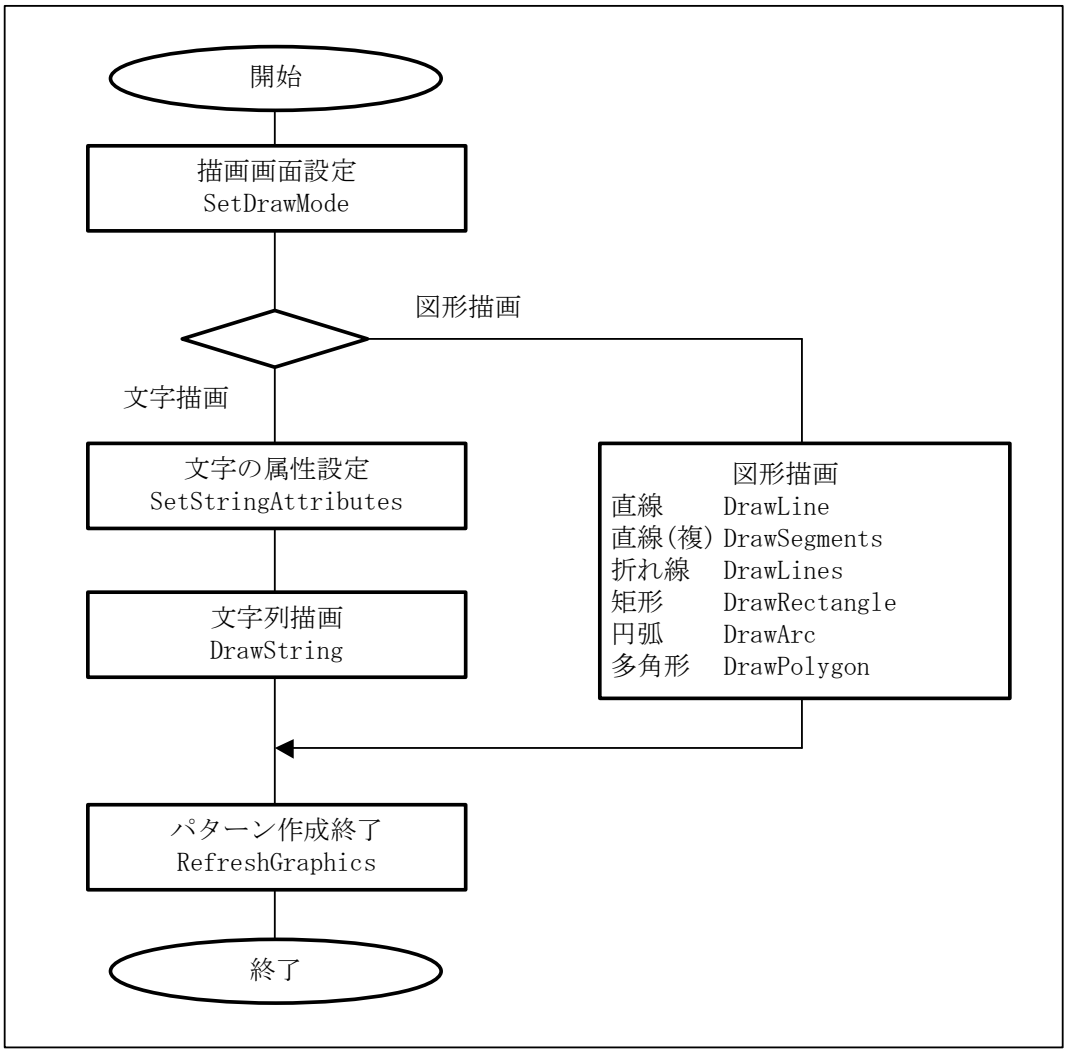


図5-17-1　グラフィックス処理の実行手順

5.18 線分化

画像処理コマンドでは線分化コマンドを用意しています。線分化は、２値画像の物体に対して線分列外周座標を抽出し、線分列外周座標から特徴量の抽出を行います。

線分列外周座標の抽出は、ExtractPolylineコマンドで行います。ExtractPolylineコマンドでは、指定画面の探索開始座標(sx,sy)からラスタースキャンで指定した濃度の探索対象物体の外周探索開始座標を取得し、取得した外周探索開始座標から時計回りに探索対象物体の外周を探索して、線分列外周座標を取得します。

対象物体の特徴量は、PolyArea、PolyPerm、PolyGrav, PolyFeatureコマンドで抽出します。

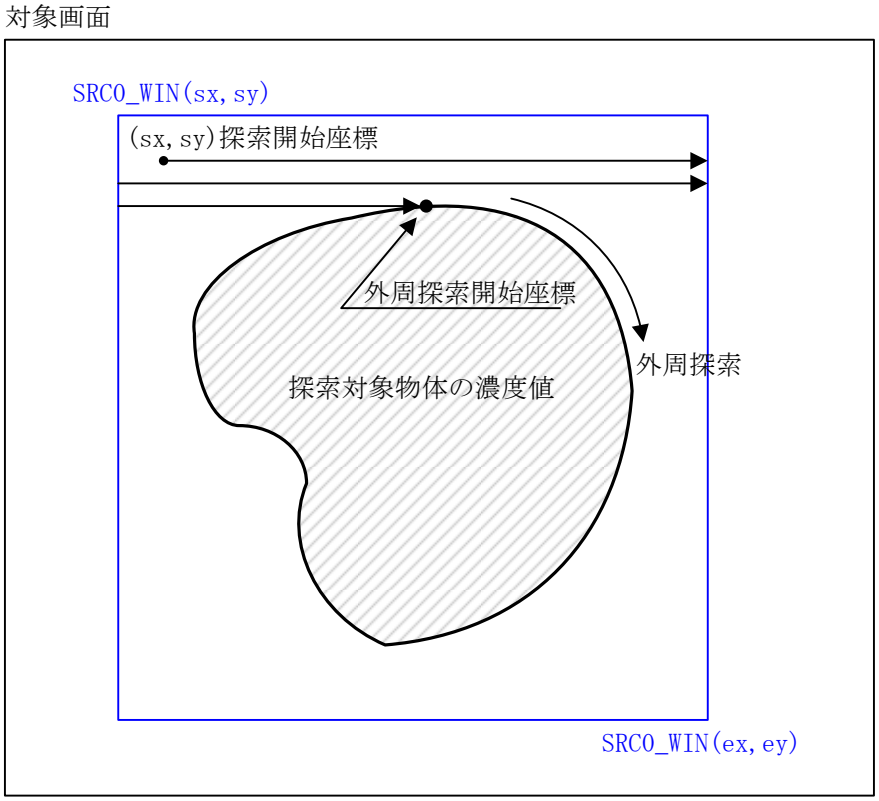


図5-18-1　線分化処理

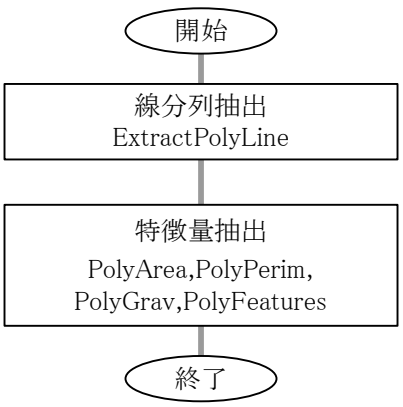


図5-18-2　線分化処理フロー

5.19 2値画像の穴埋め

画像処理コマンドでは穴埋めコマンドを用意しています。穴埋めは、２値画像の物体に対してラベリング処理を行い、指定した面積しきい値の条件を満たすオブジェクト（穴）を塗りつぶす処理を行います。

下の図は、対象画面中の穴埋め対象物が”白”オブジェクト、穴が”黒”オブジェクトの例です。
穴埋め処理は、”黒”オブジェクトに対してラベリング処理を行い、面積が最小面積以上最大面積以下のものを塗りつぶします。
尚、穴埋め処理はラベリング処理を用いているため、２５５個以上の穴埋めはできません。

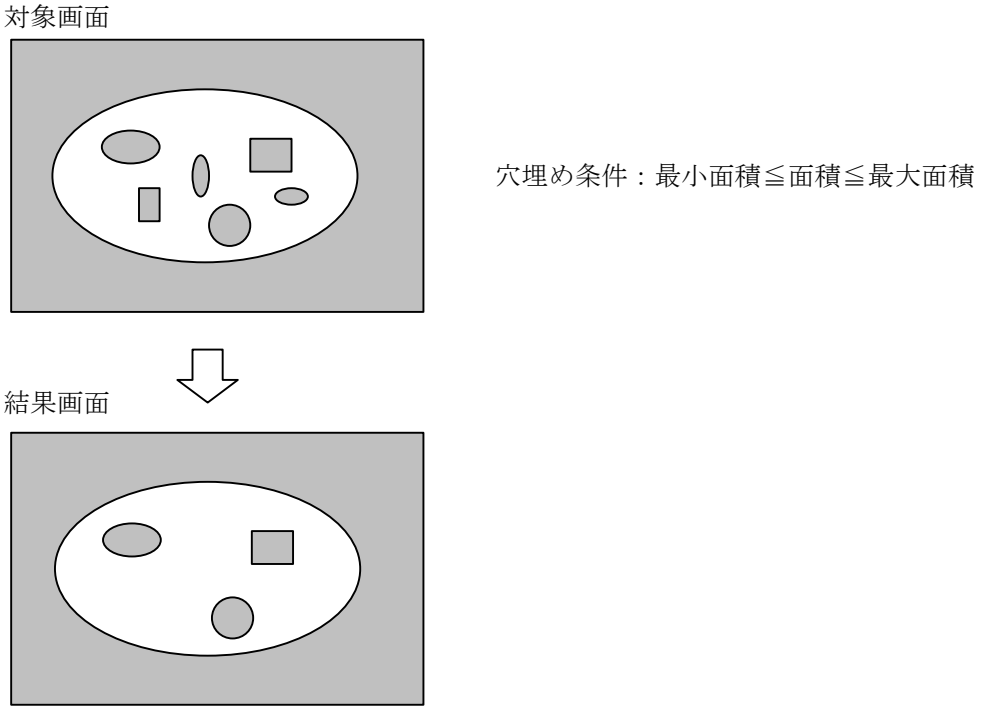


図5-19-1　２値画像穴埋め処理

5.20 正規化相関（VP－910A互換）

5.20.1 テンプレートデータ領域管理コマンド

正規化相関サーチを行う際、テンプレート特徴量データ領域確保コマンドにより、必ずテンプレート特徴量データ領域を確保しなければなりません。

(1) テンプレートデータについて

トレーニングによりテンプレートが登録されると正規化相関サーチに必要なデータはテンプレート番号（テンプレートＩＤ）で管理されます。
正規化相関サーチのテンプレートのデータは、テンプレート特徴量と画像が必要です。トレーニングを行うと指定された画像からテンプレート特徴量を計算し、テンプレート特徴量データ領域にセーブされます。指定された画像は、そのまま画像メモリにあり、その画面番号とウィンドウのデータだけがテンプレート特徴量データ領域にセーブされます。つまり、テンプレート特徴量データと画像データは別々の領域にあるということで、特に画像データは誤って別の画像に書換えられる可能性があります。テンプレートデータ内の特徴量データと画像メモリの内容が異なると正常にサーチできなくなるのでテンプレートに指定する画面の管理には十分注意が必要です。

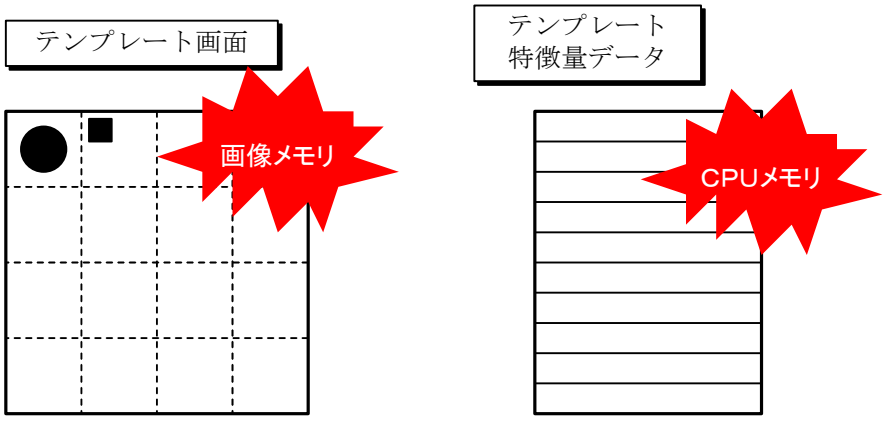


図5-20-1　テンプレートデータ

5.20.2 セットアップとトレーニング

正規化相関サーチは、セットアップ、トレーニング、サーチという３つのステップで実行されます。ここでは、セットアップとトレーニングのコマンドについて説明します。

（1） セットアップと画像の切出し

正規化相関サーチのセットアップとは、サーチを行う際のテンプレートを準備する操作のことです。
テンプレートとして登録したい画像を入力画像から切出したり、目的とする画像を画像描画コマンドで描画したりしてテンプレートを用意します。
正規化相関サーチでは、セットアップコマンドとして特別に用意しているものはありません。画像転送、画像縮小コマンドで画像をテンプレート画面として確保した画面に画像を切出し（転送）して下さい。また、画像描画コマンドを使用して目的の図形の画像をテンプレート画面に直接描画するか、別の画面に描画したものを画像転送、画像縮小コマンドでその画像を切出して下さい。

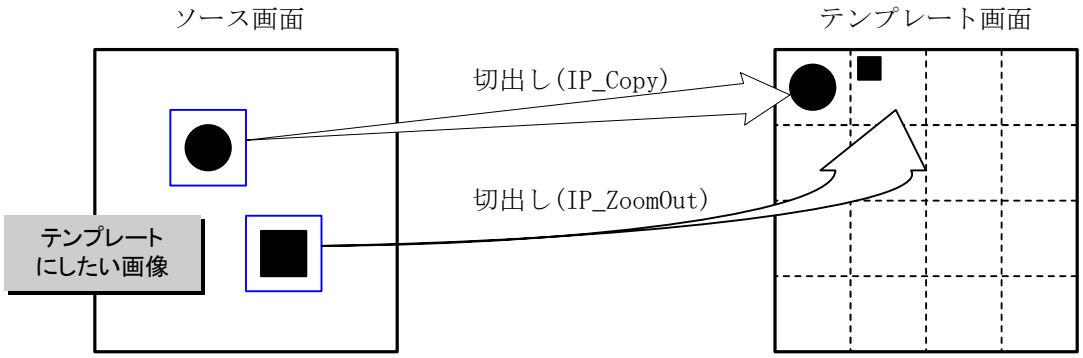


図5-20-2　画像の切出し

(2) テンプレートの情報量と処理時間

正規化相関サーチは、画像処理プロセッサとオンボードＣＰＵにより行っています。画像処理プロセッサではテンプレートカーネル内の積和演算を行います。オンボードＣＰＵでは、積和演算の結果から相関値を計算し、テンプレートカーネルを移動します。

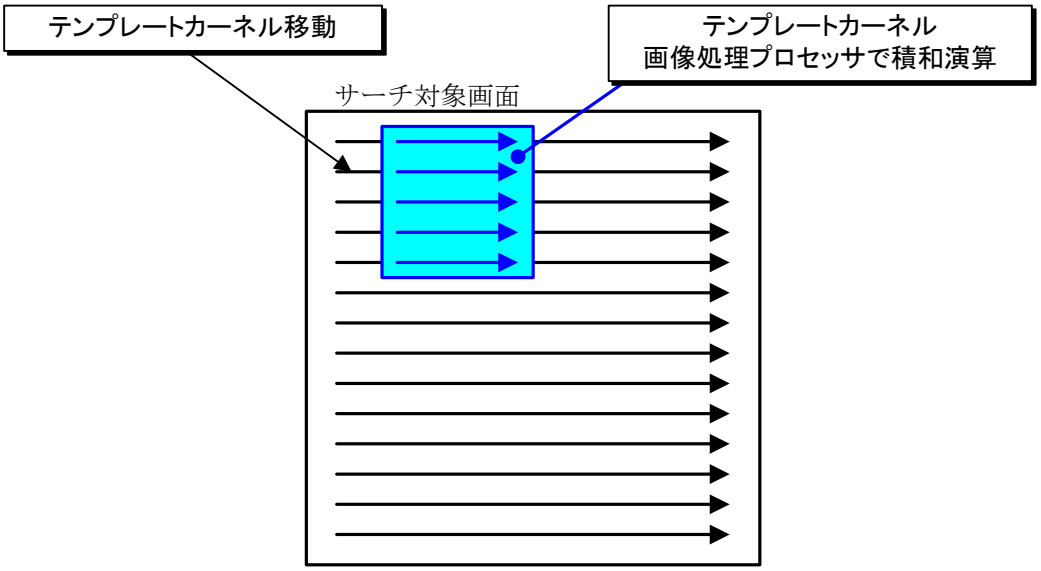


図5-20-3　正規化相関処理

画像処理プロセッサは、テンプレートカーネル内の積和演算を約４ｎＳ／画素で実行します。そしてオンボードＣＰＵでテンプレートカーネルを移動しながら、それぞれのポイントでの相関値を求め、最大の相関値とその座標を求めるわけです。ですから、サーチ時間はハードのオーバヘッド等を無視して単純に計算すると

サーチ時間 ＝ ４ｎＳ × テンプレートの大きさ × カーネルの移動回数

の式で計算できます。例えば、テンプレートの大きさが１２８×１２８画素で５１２×４８０のサーチ対象画面をサーチするとすると

4e-9 ×（128 × 128）×（512 × 480）= 16.1秒

１６秒もかかることになります。
処理時間を縮めるにはテンプレートの大きさを小さくするかカーネルの移動回数を減らせばいいことになります。
画像処理コマンドでは、テンプレートに関しては、テンプレートの大きさを小さくする代わりに情報量を減らすことで等価的にテンプレートの大きさを小さくできるようにしています。情報量を減らすということは、サーチのときのカーネル内の積和演算のデータのサンプリングをハード的に間引いて実行するということです。また、カーネルの移動も間引いて実行できるようになっています。
上記の例で、テンプレート情報量を縦８画素、横８画素間引き、カーネル移動を縦８画素、横８画素間引くと

4e-9 ×（128/8 × 128/8）×（512/8 × 480/8）= 0.004秒

１６秒の処理が４０９６分の１の４ミリ秒（実際には、レジスタアクセスのオーバヘッドで１０ｍＳ程度）でできることになります。
また、このときのテンプレートの情報量は

128/8 × 128/8 = 256

になります。

このように、実際の処理は情報量を減らして高速化を図ります。

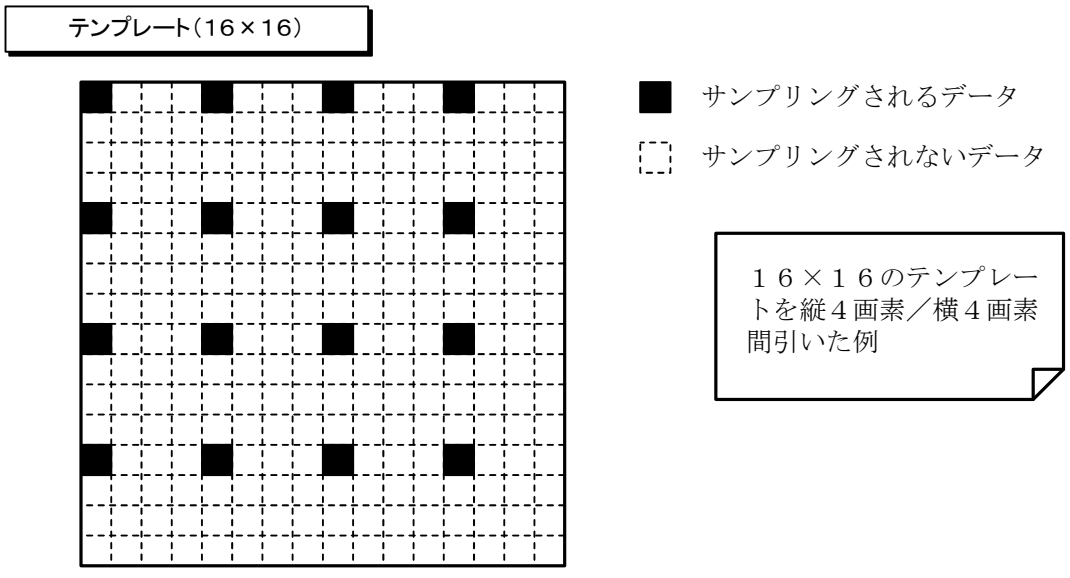


図5-20-4　テンプレートの間引き

(3) テンプレートマスクの設定

テンプレート画像は矩形領域しか設定できません。しかし、テンプレート画像として矩形以外の形の領域を設定したい場合があります。そこで、テンプレート画像にはマスク（不感帯）領域を設定できる機能があります。テンプレート画像のマスク設定された領域は、ハード処理される積和演算で計算の対象から除外されます。

設定したいマスク領域のテンプレート画像の値を「０」にすることでその領域がマスク領域になります。

通常は、円等の簡単な図形でマスクしますが、複雑なマスクも設定可能です。

下図にマスクの設定例を示します。

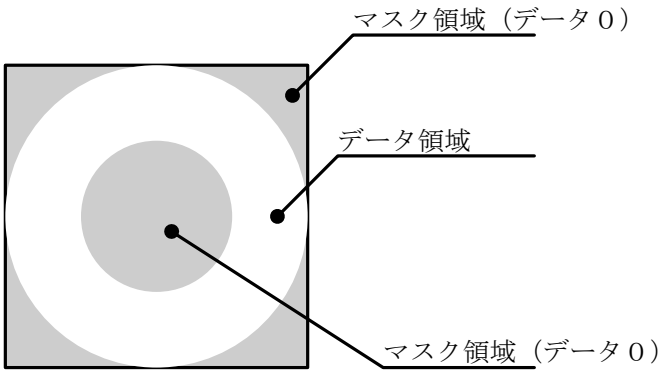


図5-20-5　マスクの設定例

次にマスクの設定手順を説明します。まず、テンプレートを切出し、テンプレートのマスクしたい領域に「０」を書込みます。

次にEnableCorrMaskコマンドでマスクモードを有効にします。

そして、SetCorrTemplateまたは、SetCorrTemplateExtコマンドでトレーニング（テンプレートの登録）を行います。

マスクを解除する場合は、DisableCorrMaskコマンドでマスクモードを解除して下さい。

マスクなしのテンプレートに戻したい場合は、テンプレート画像をもう一度切出し、トレーニングを行って下さい。

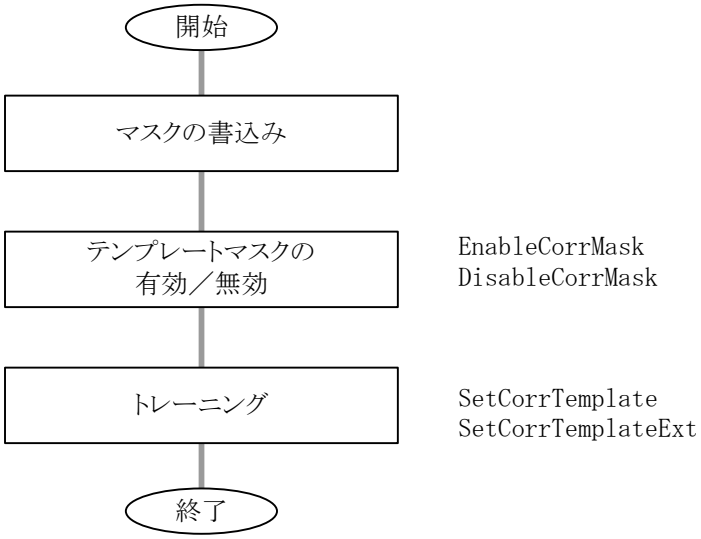


図5-20-6　テンプレートマスクの設定フロー

5.20.3 正規化相関サーチ

正規化相関サーチは、セットアップ、トレーニング、サーチという３つのステップで実行されます。ここでは、サーチのコマンドについて説明します。

（1） 高速サーチの方法

サーチコマンドには、vpxIP_CorrStep , vpxIP_CorrPoint , vpxIP_CorrPrecise コマンドがあります。
サーチコマンドは、テンプレートカーネルを移動しながら相関値を演算し最大の相関値と座標を見つけ出す処理を行っています。
最も単純なサーチでは、前項で説明しているように処理時間がユーザの要求に応えられるものではありません。そのため、テンプレートカーネルの間引きとサーチの間引きを行うわけです。前項ではテンプレートカーネルの間引きについて説明しましたが、ここではテンプレートカーネルを移動する時の間引きについてと高速化の方法を説明します。

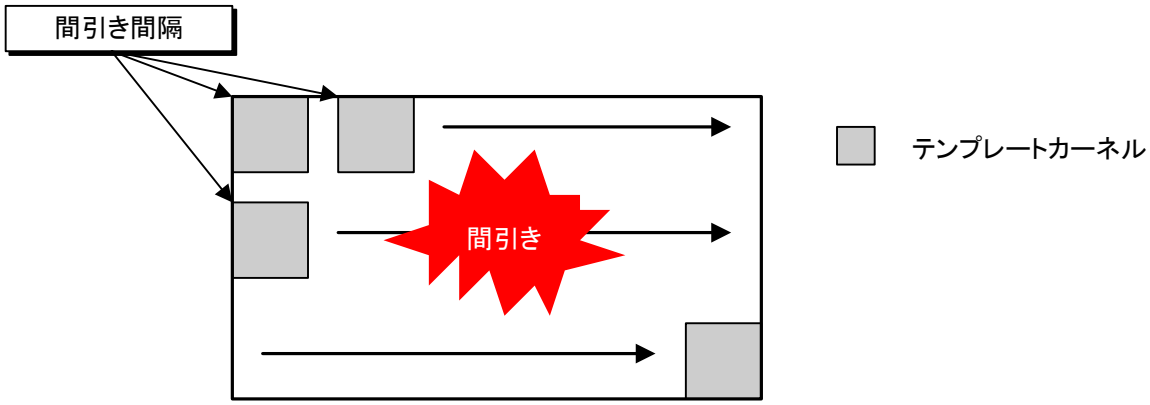


図5-20-7　vpxIP_CorrStepコマンド

vpxIP_CorrPointコマンドは、入力された座標の近傍領域でサーチを行います。通常は、vpxIP_CorrStep コマンドで間引いたぶんの補間をするためのサーチとして使用します。

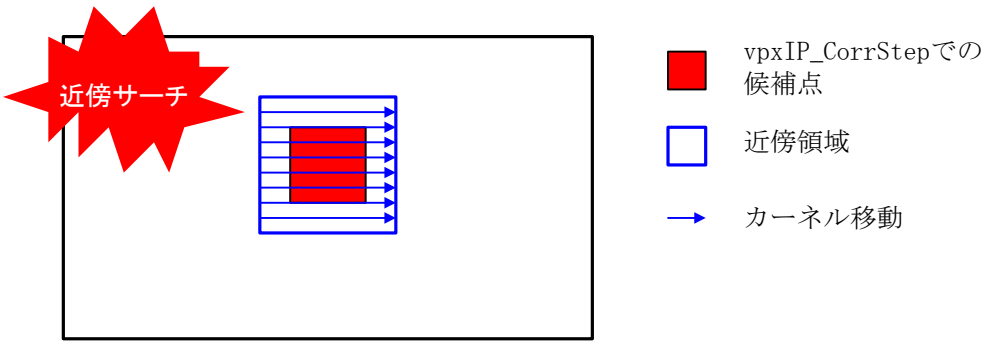


図5-20-8　vpxIP_CorrPointコマンド

vpxIP_CorrPrecise コマンドは、入力された座標の近傍領域の相関値からサブピクセルの位置を計算します。
通常は、vpxIP_CorrPoint コマンドで得られた最終位置座標を vpxIP_CorrPrecise コマンドに入力します。

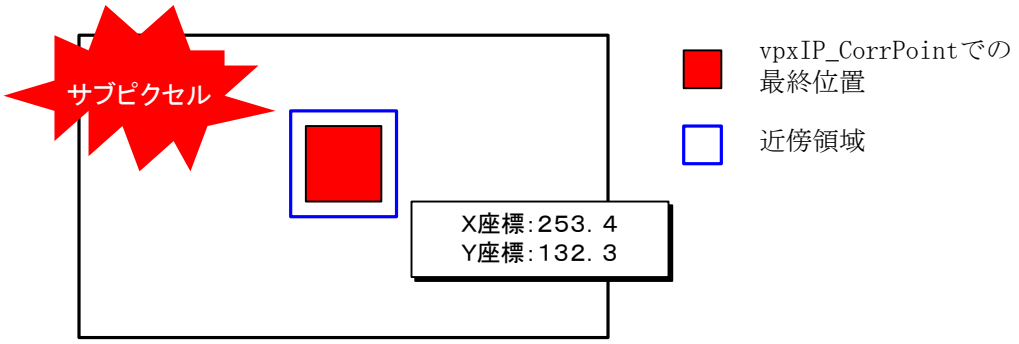


図5-20-9　vpxIP_CorrPreciseコマンド

（2） 並列演算カーネルによるサーチの高速化

画像処理プロセッサは、正規化相関演算用の積和演算器を１６組内蔵しています。その並列演算カーネルを使用して１つのテンプレートカーネルに対して１６組（４×４）の積和演算を並列に実行することができます。
並列演算カーネルを使用すると処理時間は原理的に並列演算カーネルを使わない時とくらべ１／１６になります。
また、原理的に４×４カーネルの間隔は通常は１画素ですが、テンプレートカーネルの間引きを行うと４×４カーネルの間隔はテンプレートカーネルの間引き間隔になります。

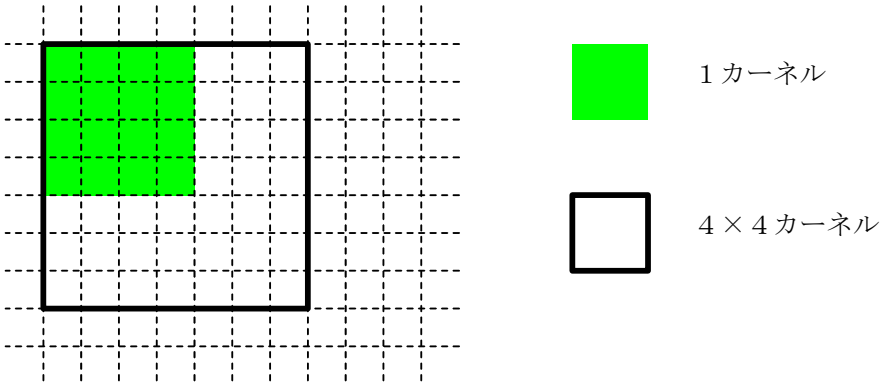


図5-20-10　４×４カーネル

サーチの高速化には上記の並列演算カーネルを使用する方法とテンプレートとサーチ移動の間引きによる方法がありますが、両方の方法を使用することで画像処理ボードの最大の性能を引き出すことができます。　しかし、並列演算カーネルとテンプレートとサーチ移動の間引きを同時に使用するには以下に示す条件を満足しなければなりません。

① サーチ移動の間引き間隔がテンプレートカーネルの間引き間隔と同じ
　または、その整数分の１
　または、その２倍

①の条件は、テンプレートカーネルの間引きを行うと４×４カーネルの間隔はテンプレートカーネルの間引き間隔になることに起因しています。
vpxIP_CorrStep ， vpxIP_CorrPoint コマンドは、上記①の条件のとき自動的に並列演算カーネルモードで処理を実行します。

（3） 相関演算途中打切りによるサーチの高速化

正規化相関の演算として画像処理プロセッサによりテンプレートカーネル内の積和演算を行っていますが、その他にもう一つ差分累積演算を行う演算器を持っています。これは、テンプレートカーネル内の画像と差分累積演算を行い累積誤差が指定閾値よりも大きくなったとき、正規化相関の積和演算の処理を途中で打切るためのものです。正規化相関の積和演算の処理を途中で打切ることにより、正規化相関サーチの高速化を図ることができます。
サーチ対象画像がテンプレート画像とあきらかに異なる場合、相関演算実行中に累積誤差が非常に大きくなると考えられます。そこで、この累積誤差がある閾値を超えた時点で、ミスマッチと判断し、途中で演算を中止します。
途中で演算を中止することにより処理の高速化を図ることができますが、照明の変動による許容値が低くなるので、注意して下さい。

● 閾値の設定

閾値とモードの設定は vpxSetCorrBreak コマンドで行います。
実際に演算を途中で打ち切るための評価値である累積誤差閾値は、

累積誤差閾値　＝　ｔｈｒ　×　テンプレート画素数

で計算されます。
また、ＳＳＤＡ法による自動閾値変更モードもサポートしています。これは、常に最小を閾値として、演算の打ち切りを行います。この場合、照明変動にはかなり弱くなりますので十分に注意して下さい。

● 相関演算打切りの設定

相関演算途中打切りを行う場合は、vpxIP_CorrStep ， vpxIP_CorrPoint コマンドを実行する前に vpxEnableCorrBreak ， vpxDisableCorrBreak コマンドで相関演算途中打切りモードを設定して下さい。
画像処理コマンドシステムの初期設定は、相関演算途中打切り無効になっています。

（4） サーチ除外領域設定とマッチング候補点

vpxIP_CorrStep コマンドでは、テンプレートカーネルを移動しながら相関値の計算を行い、マッチングポイントを探します。通常は、間引いてサーチを行うので、探し出したポイントは正確なマッチングポイントではありません。その正確ではないマッチングポイントをマッチング候補点と呼びます。
vpxIP_CorrStep コマンドでは、探し出すマッチング候補点の個数を任意に設定できるようになっています。しかし、複数のマッチング候補点をサーチすると、近傍領域にその候補点が集中します。
そこで、マッチング候補点がその近傍領域に集中しないようにサーチ除外領域を設定します。サーチ除外領域として設定した距離以内にマッチング候補点が２つ以上存在しないようにポイントを探します。
サーチ除外領域を設定／解除は、vpxSetSearchDistance コマンドで行います。

5.21 直線抽出

画像処理コマンドではハフ変換による直線・矩形抽出コマンドを用意しています。以下にその使用方法を説明します。

5.21.1 ハフ変換による直線抽出

画像処理コマンドでは、ハフ変換を用いて離散的な座標データから直線を抽出します。ハフ変換の演算はオンボードＣＰＵで高速に処理します。
また、直線抽出コマンドでは下図のようにρ－θで示される直線が抽出されます。

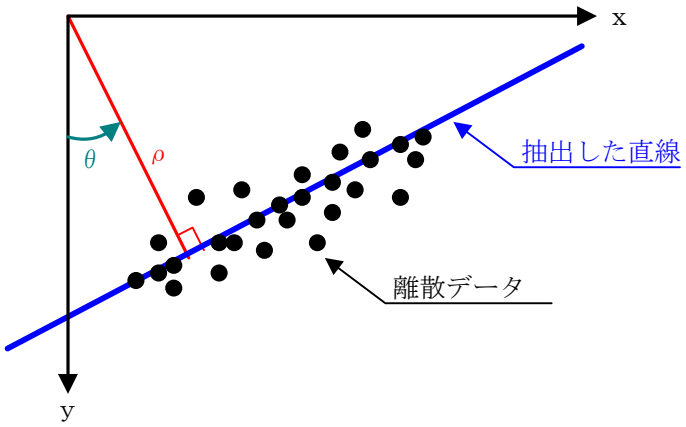


図5-21-1　離散データからの直線抽出

離散データからの直線抽出では、まず、上の図に示すような離散データの座標を２値化した画像からIP_ProjectBORegionX ， IP_ProjectBORegionY コマンドなどの座標抽出を行い抽出します。
次に、その座標データ群からハフ変換によりρ－θで示される直線を抽出します。

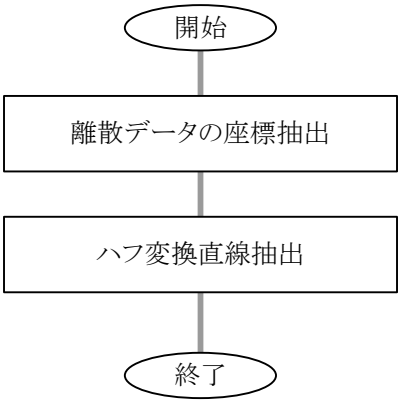


図5-21-2　離散データからの直線抽出フロー

5.21.2 ハフ変換直線からの矩形算出

画像処理コマンドでは、ハフ変換で抽出されたρ－θで示される４つの直線からそれぞれの交点座標を算出し、矩形の頂点座標や角度等を算出することができます。

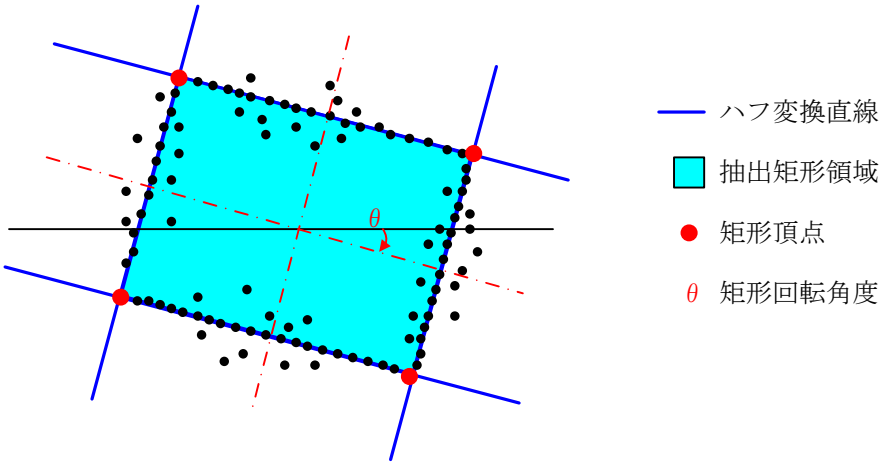


図5-21-3　ハフ変換直線からの矩形算出

GetCrossPoint,GetRectPoint,GetRectCenter,GetAnglePoint4,GetAnglePoint2コマンドにより、ハフ変換で抽出されたρ－θで示される２つまたは４つの直線からそれぞれの交点座標の算出、矩形の頂点座標や角度等の算出を行うことができます。

5.22 イメージキャリパ

画像処理コマンドではキャリパコマンドを用意しています。キャリパとはノギスのことで、画像処理により対象物のエッジを検出し、その対象物のエッジで平行なペアをサーチし、２つのエッジ間の距離やその中心位置を計測することができます。
以下にその使用方法を説明します。

5.22.1 イメージキャリパによる寸法計測

以下にイメージキャリパコマンドによる寸法計測のフローを示します。例は、集積回路（ＩＣ）パッケージのリード幅やリード間隔を求める場合です。

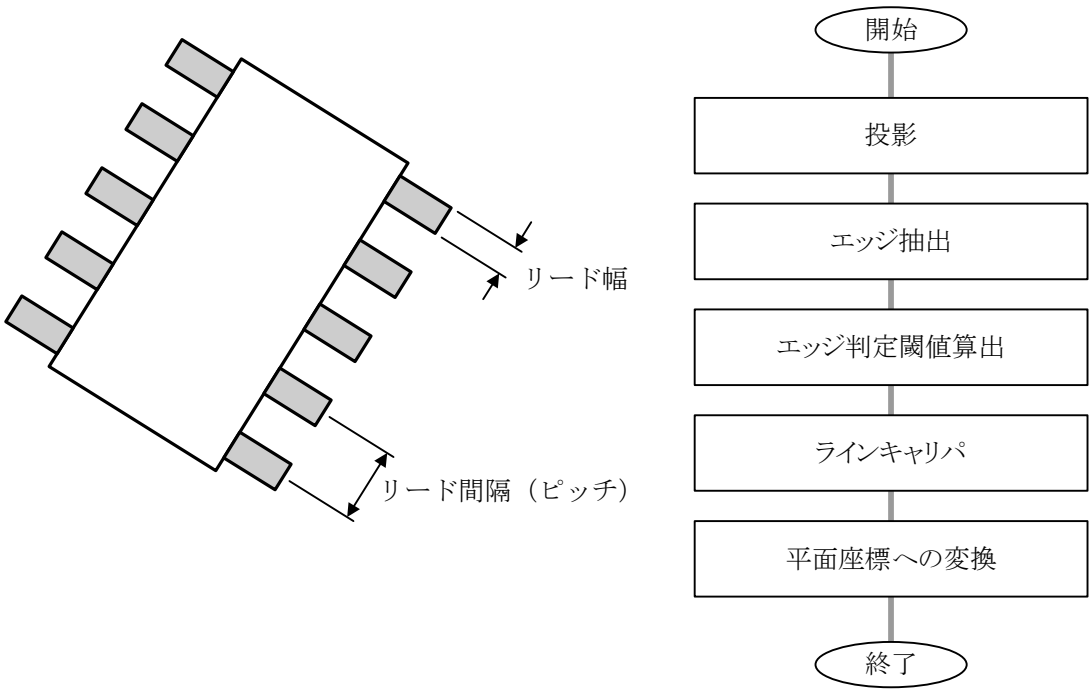


図5-22-1　寸法計測フロー

まず、ProjectLineコマンドにより、ＩＣのリードに沿って投影を行い、平面のデータをラインのデータに変換します。次にLineEdgeFilter,LineEdgeFilterExtコマンドによりエッジ抽出を行います。エッジデータからGetCaliperScoreコマンドでエッジ判定閾値を算出し、LineCaliperコマンドでエッジ判定閾値に従い、アップエッジとダウンエッジのペアをサーチします。そして、CaliperLPtoSPコマンドによりラインデータを平面のデータに変換して処理が完了します。そのデータからリード幅とリード間隔を計算します。

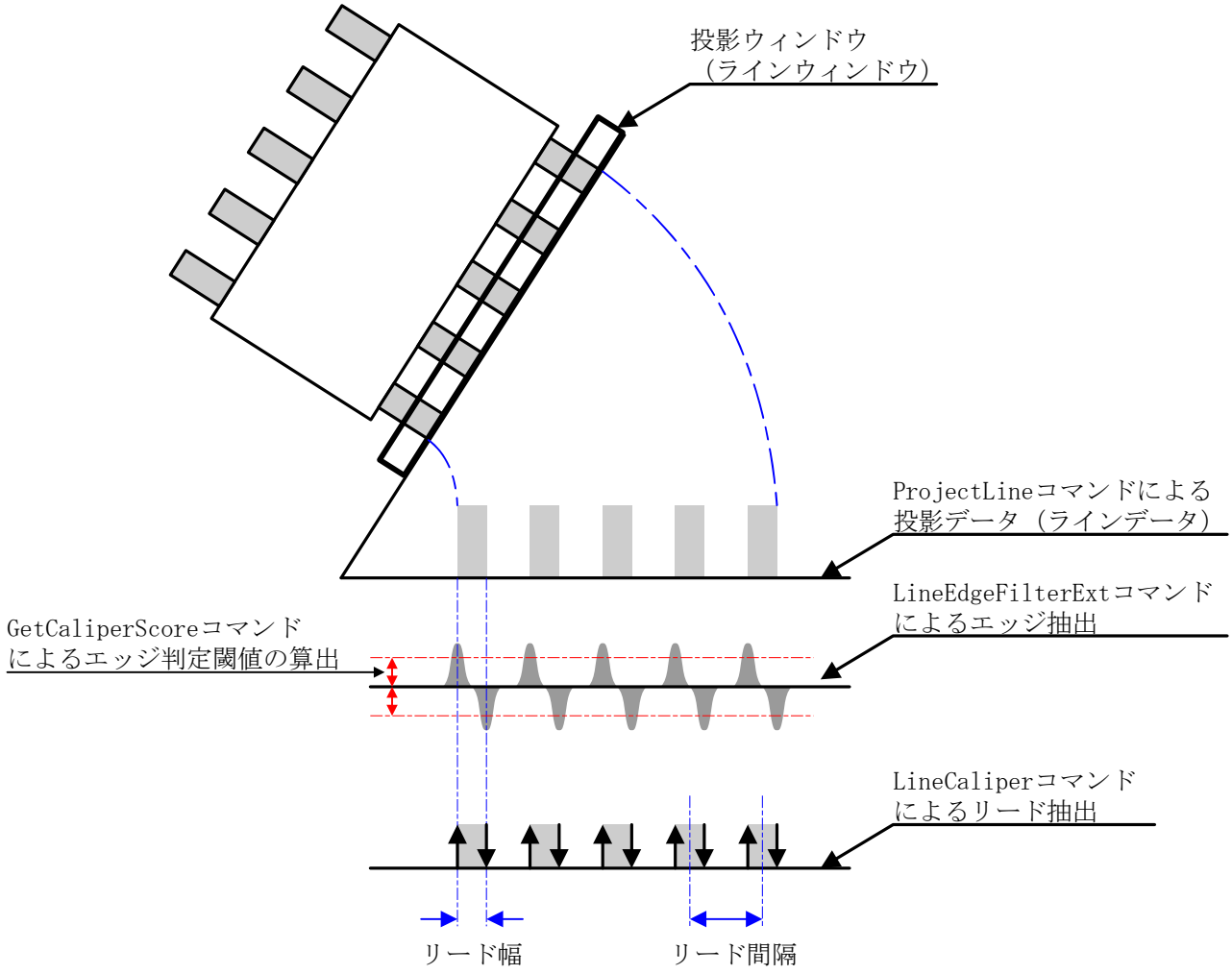


図5-22-2　寸法計測方法

5.22.2 ラインウィンドウについて

ProjectLineコマンドで投影を行う場合やCaliperLPtoSPコマンドによりラインデータを平面のデータに変換する場合、投影を行う領域を指定する必要があります。その場合の領域は、ラインウィンドウにより指定します。通常の画像処理コマンドのウィンドウは、傾斜した矩形領域を設定することはできませんが、ProjectLineコマンドでは、ラインウィンドウにより傾斜した矩形領域を設定し、傾斜した領域を投影することができます。

以下に、ラインウィンドウの構造体を示します。

```
typedef struct {
    short     sx;       // 開始点Ｘ座標
    short     sy;       // 開始点Ｙ座標
    short     ex;       // 終了点Ｘ座標
    short     ey;       // 終了点Ｙ座標
    short   leng;       // 投影幅
    short    opt;       // オプション（未使用）
} LINEWINDOW;
```

次に、ラインウィンドウの例を挙げながら構造体の各メンバーについて説明します。

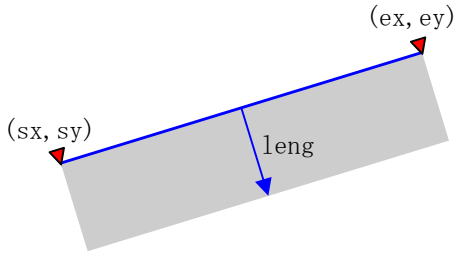


図5-22-3　ラインウィンドウ

ラインウィンドウでは、(sx, sy)と(ex, ey)で結ばれる直線で投影領域を指定し、lengで投影する画素数を指定します。(sx, sy)と(ex, ey)で結ばれる直線上のポイントが投影の開始ポイントになり、直線上のポイントに沿って(sx, sy)と(ex, ey)で結ばれる直線と垂直な方向にlengで指定される画素数分画素データの濃度累積が行われなす。

また、lengは(sx, sy)と(ex, ey)で結ばれる直線の状態と符号により濃度累積する方向が変わります。

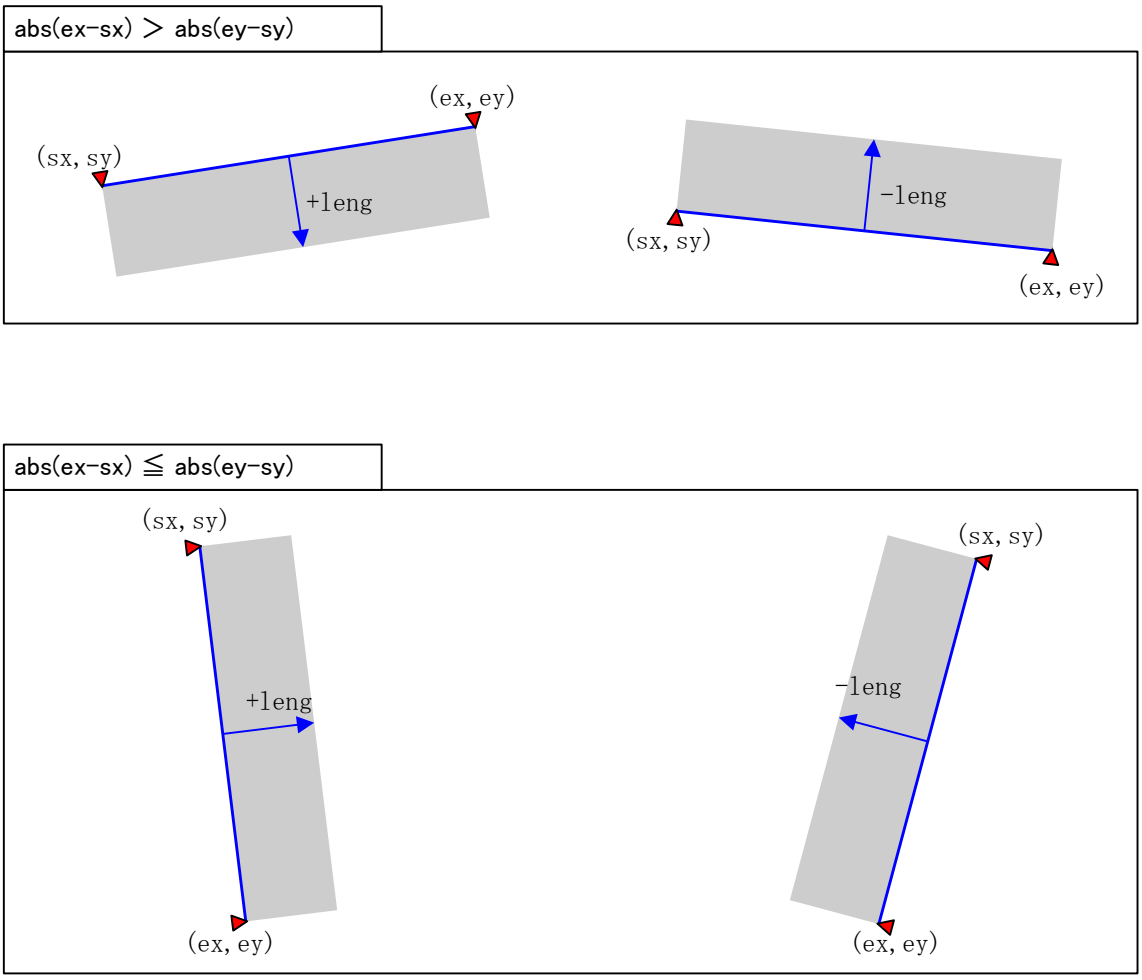


図5-22-4　lengの符号

5.23 エッジファインダ

画像処理コマンドではエッジファインダコマンドを用意しています。イメージキャリパでは画像処理により対象物のエッジを検出し、その対象物のエッジで平行なペアをサーチするため、エッジのペアがないパターンでは、エッジを抽出することはできません。そこで、エッジファインダーではその対象物に含まれるあらゆるエッジを解析し、位置、ポラリティ、レベルといった情報を抽出します。

5.23.1 ラインエッジファインダのエッジ抽出

以下にラインエッジファインダコマンドによるエッジ抽出のフローを示します。処理手順は基本的にイメージキャリパコマンドと同じですので、そちらも参照して下さい。

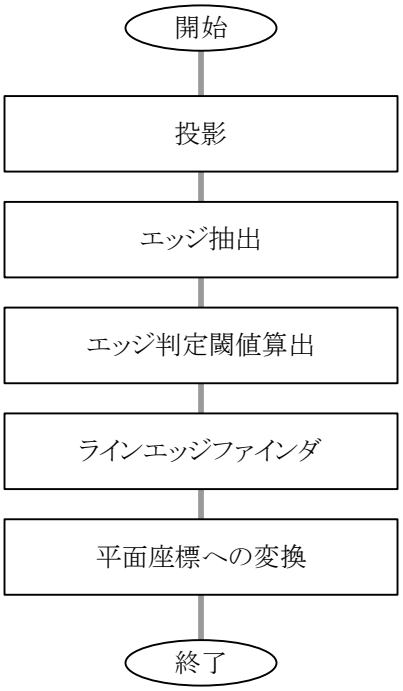


図5-23-1　エッジ抽出フロー

5.24 RGBLUT変換

ＲＧＢＬＵＴ変換は、２４ビットのＲＧＢカラー画像を濃度変換テーブル(ＬＵＴ)により、８ビットの画像に変換する処理です。濃度変換のパターンを換えることにより、ＲＧＢカラー画像から特定カラーを抽出したり、色相、彩度、明度を抽出する等、様々な処理が可能です。

画像処理コマンドでは、ＲＧＢＬＵＴ変換情報をＲＧＢＬＵＴオブジェクトに格納し、ＲＧＢＬＵＴハンドルで管理します。あらかじめ、ＲＧＢＬＵＴオブジェクトにＲＧＢＬＵＴ変換情報を設定し、コマンドを実行することによりＲＧＢＬＵＴ変換を行います。ＲＧＢＬＵＴオブジェクトの構成とＲＧＢＬＵＴ変換の手順を以下に示します。

5.24.1 RGBLUT変換手順

ＲＧＢＬＵＴ変換するには、まず、CreateRGBLUTコマンドでＲＧＢＬＵＴオブジェクトを生成します。次に、OpenRGBLUTコマンドでＬＵＴをアクセス可能にしてから、ＬＵＴに変換データを設定します。ＬＵＴへは、直接、変換データを書き込むか、あらかじめ用意されているマクロ(5.24.3 参照)で設定します。ＬＵＴの設定が終了したら、CloseRGBLUTコマンドでＬＵＴのアクセスを終了します。ＲＧＢＬＵＴ変換は、ConvertRGBLUTまたはIP_ConvertRGBLUTExコマンドを実行します。最後にDeleteRGBLUTコマンドでＲＧＢＬＵＴオブジェクトを削除して処理を終了します。

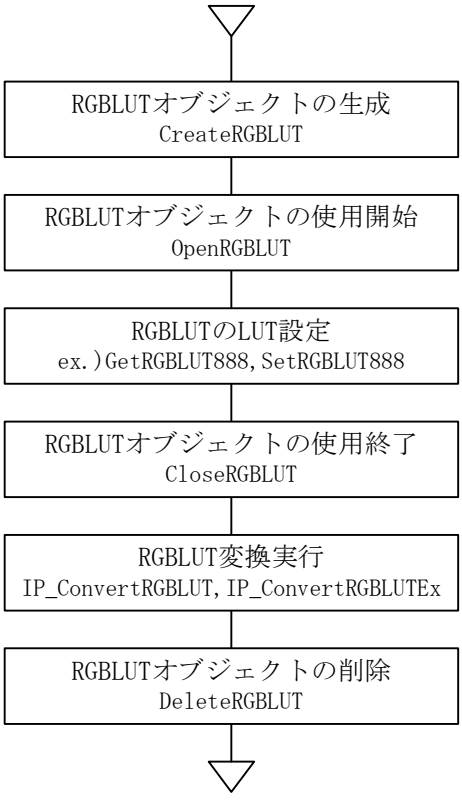


図5-24-1 ＲＧＢＬＵＴ変換手順

5.24.2 RGBLUTオブジェクト

ＲＧＢＬＵＴオブジェクトはCreateRGBLUTコマンドでオンボードＣＰＵ上に生成され、ＲＧＢＬＵＴハンドルで管理します。ＲＧＢＬＵＴオブジェクトの構成を以下に示します。

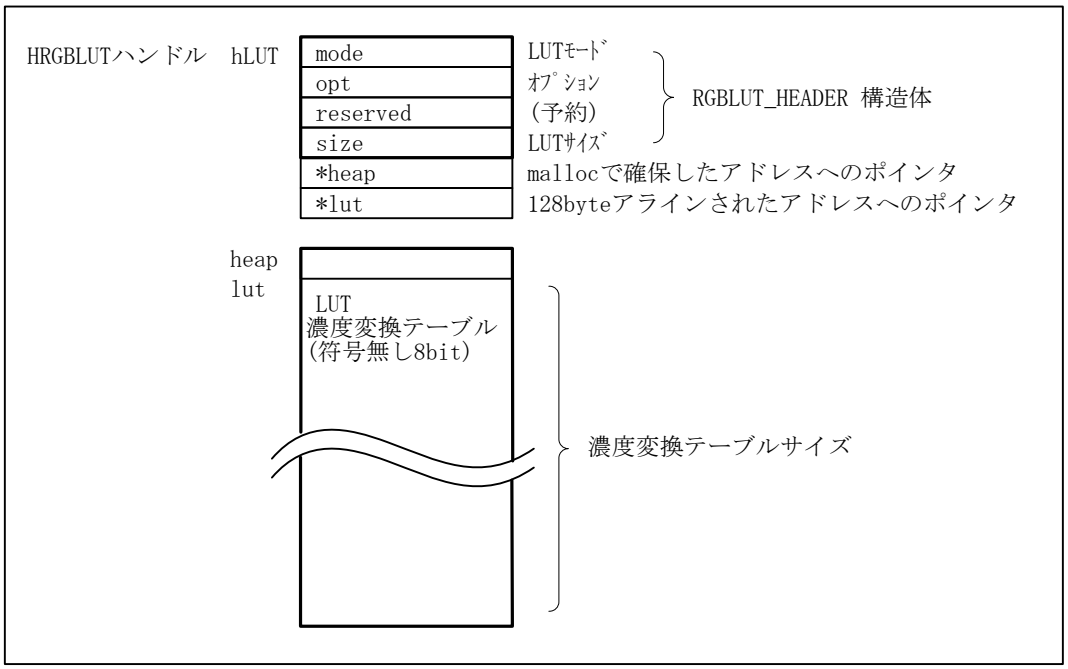


図5-24-2　ＲＧＢＬＵＴオブジェクト構成

ＬＵＴモード(mode)は以下の通りで、濃度変換処理と濃度変換テーブル(ＬＵＴ)の大きさを決定します。例えば、RGBLUT888は、Ｒデータが０～２５５(８bit)、Ｇデータが０～２５５(８bit)、Ｂデータが０～２５５(８bit)のＲＧＢカラー画像を濃淡画像０～２５５(８bit)に変換するモードで、CreateRGBLUTコマンド実行時に16,777,216byteの変換テーブルを確保します。

表5-24-1　ＬＵＴモード

ＬＵＴモード	定数値	内　容
RGBLUT888	0	R:8bit / G:8bit / B:8bit (16,777,216byte)
RGBLUT777	1	R:7bit / G:7bit / B:7bit (2,097,152byte)
RGBLUT666	2	R:6bit / G:6bit / B:6bit (262,144byte)
RGBLUT887	3	R:8bit / G:8bit / B:7bit (8,388,608byte)
RGBLUT878	4	R:8bit / G:7bit / B:8bit (8,388,608byte)
RGBLUT788	5	R:7bit / G:8bit / B:8bit (8,388,608byte)
RGBLUT787	6	R:7bit / G:8bit / B:7bit (4,194,304byte)
RGBLUT2CH88	7	R:8bit / G:8bit (65,536byte)
RGBLUT565	8	R:5bit / G:6bit / B:5bit (65,536byte)

5.24.3 濃度変換テーブル

濃度変換テーブル(ＬＵＴ)は、OpenRGBLUTコマンドを実行して、ＬＵＴサイズと先頭アドレスを取得し、直接アクセスすることが可能です。ＬＵＴへは、直接データを書き込むか、あらかじめ「ipxdef.h」に定義されている以下のマクロを用いて設定します。ここで、lutはＬＵＴの先頭アドレス、ｒ、ｇ、ｂはそれぞれＲＧＢ画像のＲデータ、Ｇデータ、Ｂデータ、ｄは変換データです。マクロ定義から分かるように、ｒ、ｇ、ｂ値がＬＵＴのアドレスを示すパラメータになります。
ＬＵＴの設定が終了したら、必ず、CloseRGBLUTコマンドでＬＵＴのアクセスを終了してください。

ＬＵＴへのデータ書き込みマクロ

```
#define SetRGBLUT888(lut,r,g,b,d) lut[r << 16 | g << 8 | b]= (d)
#define SetRGBLUT777(lut,r,g,b,d) lut[(r & 0xfe) << 13 | (g & 0xfe) << 6 | b >> 1]= (d)
#define SetRGBLUT666(lut,r,g,b,d) lut[(r & 0xfc) << 10 | (g & 0xfc) << 4 | b >> 2]= (d)
#define SetRGBLUT887(lut,r,g,b,d) lut[r << 15 | g << 7 | b >> 1]= (d)
#define SetRGBLUT878(lut,r,g,b,d) lut[r << 15 | (g & 0xfe) << 7 | b]= (d)
#define SetRGBLUT788(lut,r,g,b,d) lut[(r & 0xfe) << 15 | g << 8 | b]= (d)
#define SetRGBLUT787(lut,r,g,b,d) lut[(r & 0xfe) << 14 | g << 7 | b >> 1]= (d)
#define SetRGBLUT2CH88(lut,r,g,d) lut[r << 8 | g]= (d)
#define SetRGBLUT565(lut,r,g,b,d) lut[(r & 0xf8) << 8  | (g & 0xfc) << 5 | b >> 3]= (d)
```

ＬＵＴからのデータ読み出しマクロ

```
#define GetRGBLUT888(lut,r,g,b)   lut[r << 16 | g << 8 | b]
#define GetRGBLUT777(lut,r,g,b)   lut[(r & 0xfe) << 13 | (g & 0xfe) << 6 | b >> 1]
#define GetRGBLUT666(lut,r,g,b)   lut[(r & 0xfc) << 10 | (g & 0xfc) << 4 | b >> 2]
#define GetRGBLUT887(lut,r,g,b)   lut[r << 15 | g << 7 | b >> 1]
#define GetRGBLUT878(lut,r,g,b)   lut[r << 15 | (g & 0xfe) << 7 | b]
#define GetRGBLUT788(lut,r,g,b)   lut[(r & 0xfe) << 15 | g << 8 | b]
#define GetRGBLUT787(lut,r,g,b)   lut[(r & 0xfe) << 14 | g << 7 | b >> 1]
#define GetRGBLUT2CH88(lut,r,g)   lut[r << 8 | g]
#define GetRGBLUT565(lut,r,g,b)   lut[(r & 0xf8) <<  8 | (g & 0xfc) << 5 | b >> 3]
```

5.24.4 RGBLUT濃度変換

ＲＧＢ画像のＬＵＴ変換は、IP_ConvertRGBLUTコマンド、または、IP_ConvertRGBLUTExコマンドで実行します。以下にＬＵＴモードに対する演算内容を示します。
ここで、ImgSrc(R)、ImgSrc(G)、ImgSrc(B)は、それぞれソース画像となるＲＧＢ画像のＲ、Ｇ、Ｂデータ、ImgDstはデスティネーション画像に設定されるデータです。

表5-24-2　ＬＵＴモードと演算内容

LUTモード	演算内容
RGBLUT888	ImgDst = LUT[ImgSrc(R) << 16 | ImgSrc(G) << 8 | ImgSrc(B)]
RGBLUT777	ImgDst = LUT[(ImgSrc(R) & 0xFE) << 13 | (ImgSrc(G) & 0xFE) << 6 | ImgSrc(B) >> 1]
RGBLUT666	ImgDst = LUT[(ImgSrc(R) & 0xFC) << 10 | (ImgSrc(G) & 0xFC) << 4 | ImgSrc(B) >> 2]
RGBLUT887	ImgDst = LUT[ImgSrc(R) << 15 | ImgSrc(G) << 7 | ImgSrc(B) >> 1]
RGBLUT878	ImgDst = LUT[ImgSrc(R) << 15 | (ImgSrc(G) & 0xFE) << 7 | ImgSrc(B)]
RGBLUT788	ImgDst = LUT[(ImgSrc(R) & 0xFE) << 15 | ImgSrc(G) << 8 | ImgSrc(B)]
RGBLUT787	ImgDst = LUT[(ImgSrc(R) & 0xFE) << 14 | ImgSrc(G) << 7 | ImgSrc(B) >> 1]
RGBLUT2CH88	ImgDst = LUT[ImgSrc(R) << 8 | ImgSrc(G)]
RGBLUT565	ImgDst = LUT[(ImgSrc(R) & 0xF8) << 8 | (ImgSrc(G) & 0xFC << 5) | (ImgSrc(B) << 3)]

5.24.5 RGBカラー抽出(色相・彩度・明度)

ＲＧＢＬＵＴ変換は、ＬＵＴのパターン設定により様々な変換をすることができます。
色の性質を色合い／色調(色相)、あざやかさ(彩度)、明るさ(明度)の３つの要素に分ける表現方法がありますが、ＲＧＢＬＵＴ変換を用いて、ＲＧＢカラー画像を色相(Hue)、彩度(Saturation)、明度(Intensity)の画像に変換することが可能です。以下に色相、彩度、明度の関係を示した図と画像変換の演算式を示します。

・色相 (Hue)
　色を円周上に配列していると考え、色あい／色調を角度で表わしたものです。
　Hの範囲は 0～359 です。
・彩度 (Saturation)
　色のあざやかさを示します。Ｓの範囲は 0 ～255 です。
・明度 (Intensity)
　色の明るさを示します。Iの範囲は 0 ～255 です。

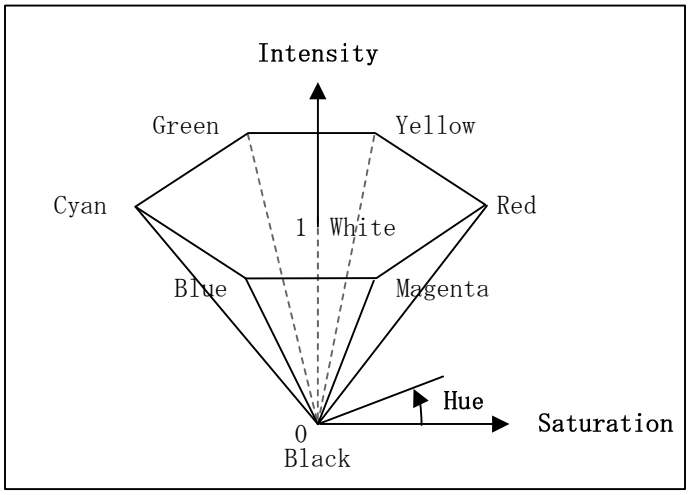


図5-24-3　色相、彩度、明度の関係

演算式	I= max(R,G,B)
	I= 0: S= 0: H= 0(不定)
	I≠ 0: i= min(R,G,B)
	S=(I- i)* 255/ I
	R= I: H=(G- B)* 60 /(I- i)
	G= I: H=(B- R)* 60 /(I- i)+ 120
	B= I: H=(R- G)* 60 /(I- i)+ 240
	(H < 0 の時は、H = H + 360 とする)
	H：Hue, S：Saturation, I:Intensity

画像メモリには、０ ～２５５ までしかデータを格納することができません。そこで、０～３６０度までのＨｕｅの値は０．７１ 倍するなどしてＬＵＴに設定してください。

以下に、ＲＧＢ画像から色相(Hue)画像へのＲＧＢＬＵＴ変換例を示します。

```
/* 画面確保 */
ImgRGB = AllocRGBImg( IMG_FS_512H_512V );
ImgID  = AllocImg( IMG_FS_512H_512V );

.....................................

/* RGBLUTオブジェクト作成 */
mode = RGBLUT888;
hLUT = CreateRGBLUT( mode , 0 );

/*********************************************
  ＲＧＢ画像から色相(Hue)画像へ変換
*********************************************/
/* RGBLUTのLUTアクセス可能 */
mode = 0;
OpenRGBLUT( hLUT, &size, &lut, mode );

/* RGBLUTのLUT設定[色相(Hue)] */
for( r = 0; r < 256; r++ ){                          r,g,b値(0～255)の組合せで
   for( g = 0; g < 256; g++ ){                       H(Hue)値が決定されます
      for( b = 0; b < 256; b++ ){
         I = max( max( b, g ), max( g, r ) );
         i = min( min( b, g ), min( g, r ) );
         if( I == 0 ){
            H = 0;
         }else{
            val = 60.0f / ( I - i );
            if( I == r ){
               H = ( g - b ) * val;
            }else if( I== g ){
               H = ( b - r ) * val + 120;
            }else{
               H = ( r - g ) * val + 240;
            }
            if( H < 0 ) H += 360;
            H *= 0.71f;
         }
         SetRGBLUT888( lut, r, g, b, H );            r,g,b値のときのH(Hue)値
      }                                              をLUTへ設定します
   }
}

/* RGBLUTのLUTアクセス禁止 */
CloseRGBLUT( hLUT, lut );

/* RGBLUT変換 */
IP_ConvertRGBLUT( ImgRGB, ImgID , 0 , hLUT);

.....................................

/* RGBLUTオブジェクト削除 */
DeleteRGBLUT( hLUT );

/* 画面解放 */
FreeImg( ImgID );
FreeImg( ImgRGB );
```

5.25 歪み補正

歪み補正とは、作成した歪み補正モデルにより、画像の変形を行う処理です。元の画像に対して、歪み補正を実行することにより、画像の傾きや歪みを解消することが可能です。
図5-25のような格子をソース画面(補正前画面)とデスティネーション画面(補正後画面)に作成することで、歪み補正を実行することができます。

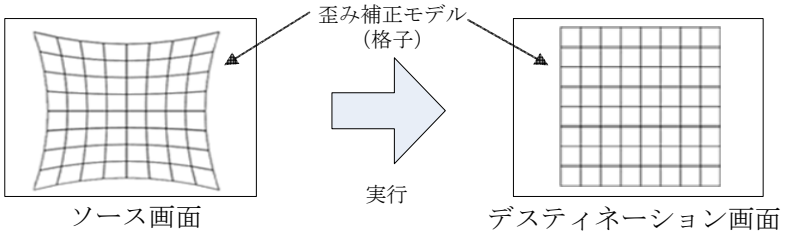


図5-25 歪み補正

5.25.1 歪み補正の実行手順

歪み補正実行の基本フローは以下のようになります。

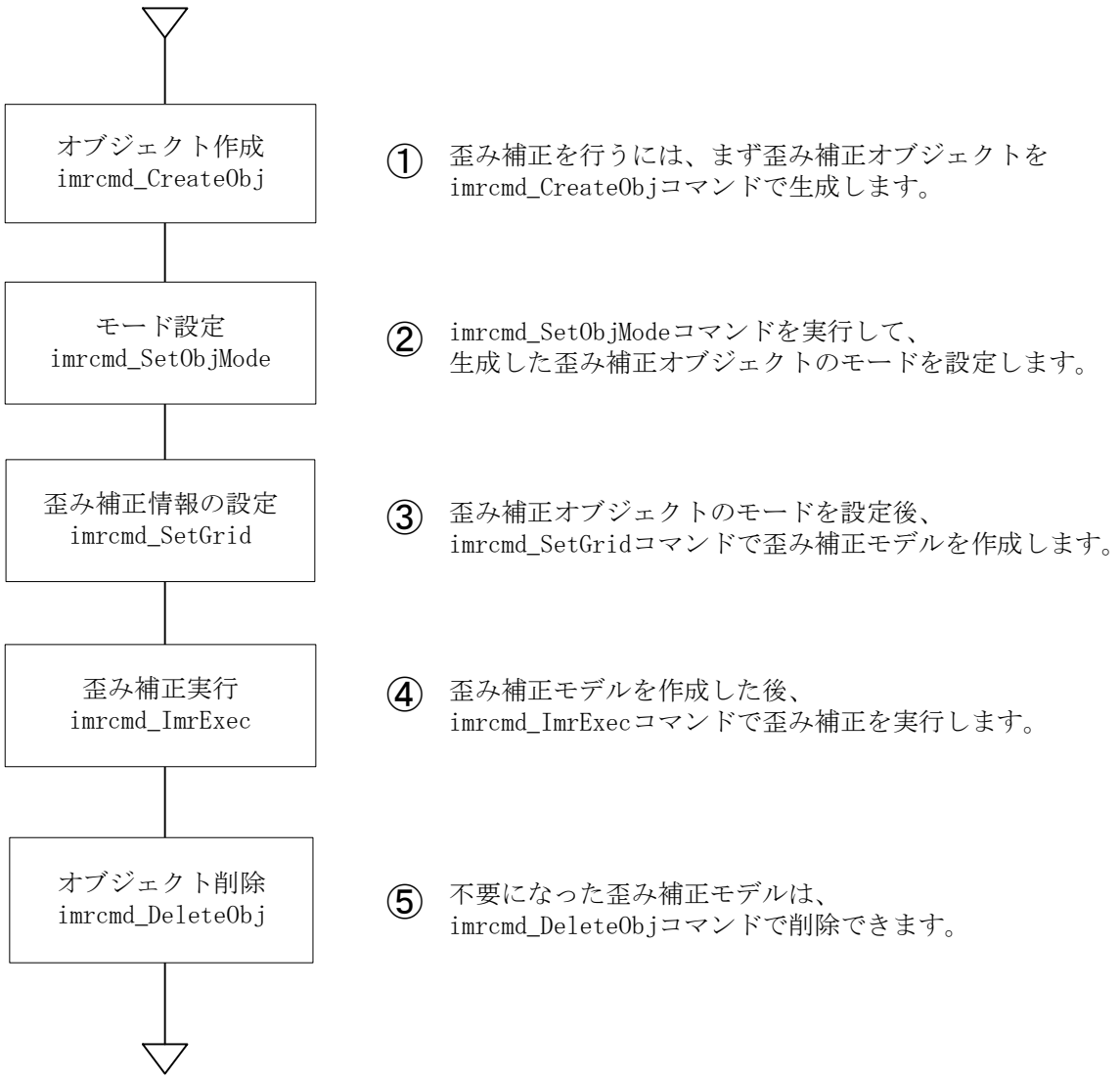


図5-25-1 歪み補正実行手順

5.25.2 歪み補正オブジェクト

歪み補正を行うためには、歪み補正モデルをあらかじめ作成します。
imrcmd_CreateObjコマンドで歪み補正オブジェクトを生成し、歪み情報をセットすることで、歪み補正モデルが完成します。歪み補正モデルは複数作成することが可能です。

5.25.3 歪み補正のモード

歪み補正モードの設定は、imrcmd_SetObjModeコマンドで行います。
歪み補正オブジェクトには、補間モードと描画モードの２つのモードがあります。

・補間モード　：バイリニア補間を実行することが可能です。
　　また、バイリニア補間のオンオフの切り替えが可能です。

・描画モード　：補正データをソース画面から参照せず、指定した画素値で単色描画されます。

5.25.4 歪み補正情報の設定

歪み補正を行う領域を、ソース画面(補正前画面)およびデスティネーション画面(補正後画面)の対応する頂点座標リストで指定します。imrcmd_SetGridコマンドは、指定された頂点情報をもとに歪み補正モデルを生成します。

図5-25-4-aは補正前画面の歪み補正モデル、図5-25-4-bは補正後画面の歪み補正モデルの例です。

補正前画面と補正後画面で、頂点①～⑥の頂点座標を座標リストに登録します。それからimrcmd_SetGridコマンドを実行することで、歪み補正モデルを生成されます。このモデルでimrcmd_ImrExecコマンドを実行すると、補正前の画像を変形することができます。

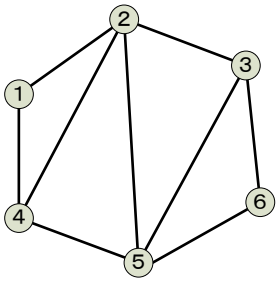


図5-25-4-a 補正前モデル

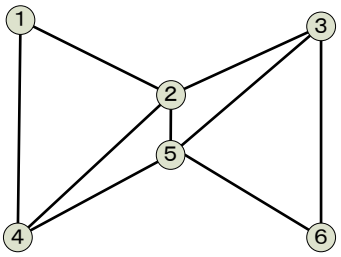


図5-25-4-b 補正後モデル

5.25.5 頂点座標の設定について

頂点座標の描画基準について、頂点座標は、画素の左上が基準となります。
画素の座標値と頂点座標値は、図5-25-5のようになっているので、注意が必要です。
下図の例のように格子の頂点数4つのモデルを作成した場合、頂点座標値は次のようになっています。

頂点座標　　:①(1,1), ②(2,1), ③(1,2), ④(2,2)

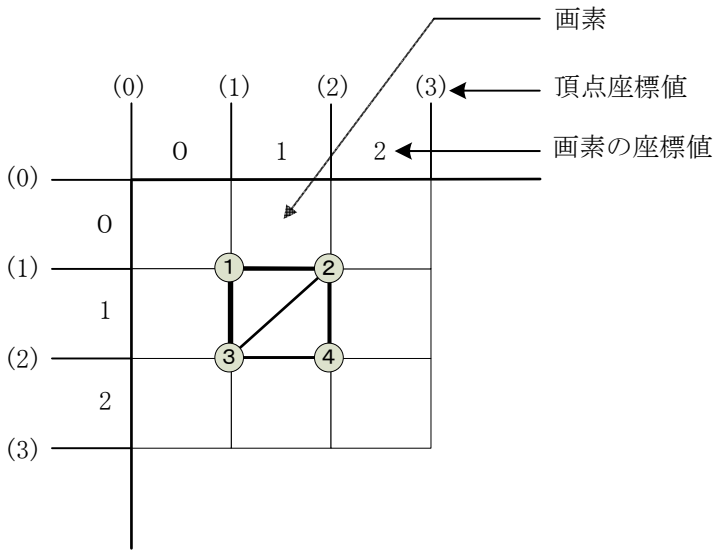


図5-25-5 頂点座標の基準

5.25.6 処理領域

歪み補正の処理領域はIMRウィンドウ(IMR_WINDOW構造体)で指定します。処理領域とデスティネーション画面の関係を図5-25-6に示します。補正前(ソース)側と補正後(デスティネーション)側の処理領域の設定は次のようになっています。

・ソースウィンドウ
　　ウィンドウサイズはx座標が2048、y座標が2048で固定、描画始点の設定可能

・デスティネーションウィンドウ
　　ウィンドウサイズはx座標が最大1025, y座標最大1025で設定でき、描画始点の設定可能

に歪み補正の処理領域を示します。

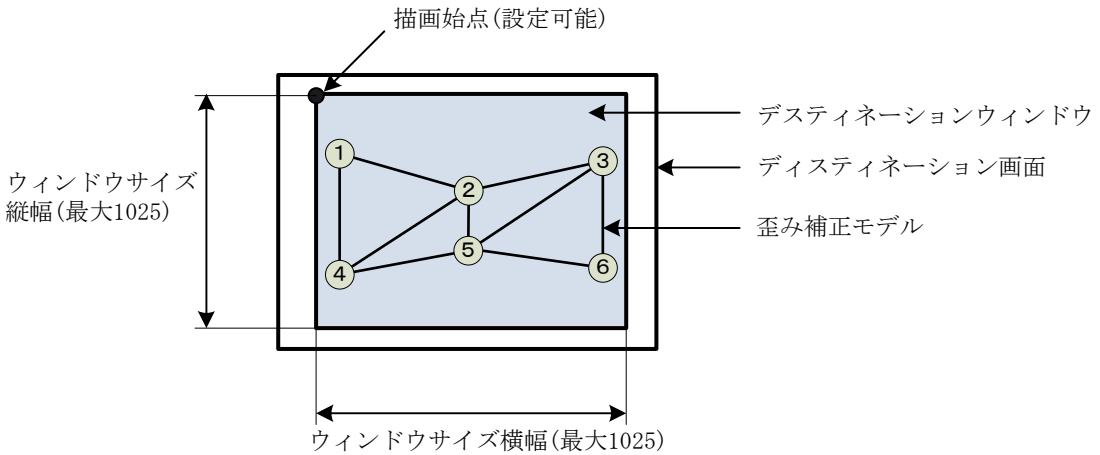


図5-25-6 歪み補正の処理領域(補正後側)

第6章　アプリケーションモジュール

アプリケーションモジュールは、ボード上でのスタンドアロン動作を実現します。ユニット内のファイルシステムにセーブしておいたアプリケーションモジュールを起動時に動作させることが可能です。アプリケーションモジュールの作成方法及び実行方法について説明します。

6.1 概要

ＮＶＰ－Ａｘ２３０はｓｈｅｌｌ（コマンド解釈して実行するプログラム）を実装しており、アプリケーションの主記憶へのロードや実行をｓｈｅｌｌのコマンドで実行します。また、ＮＶＰ－Ａｘ２３０では、フラッシュメモリディスク、ＳＤカード、または、ＵＳＢメモリを実装しており、アプリケーションの実行ファイルをそれらのディスクに書き込み、ｓｈｅｌｌの実行コマンドで実行します。現在、ＮＶＰ－Ａｘ２３０の実行ファイルとして、ＥＬＦ形式のＡｂｓｏｌｕｔｅファイル（．ａｂｓ）のみサポートしています。

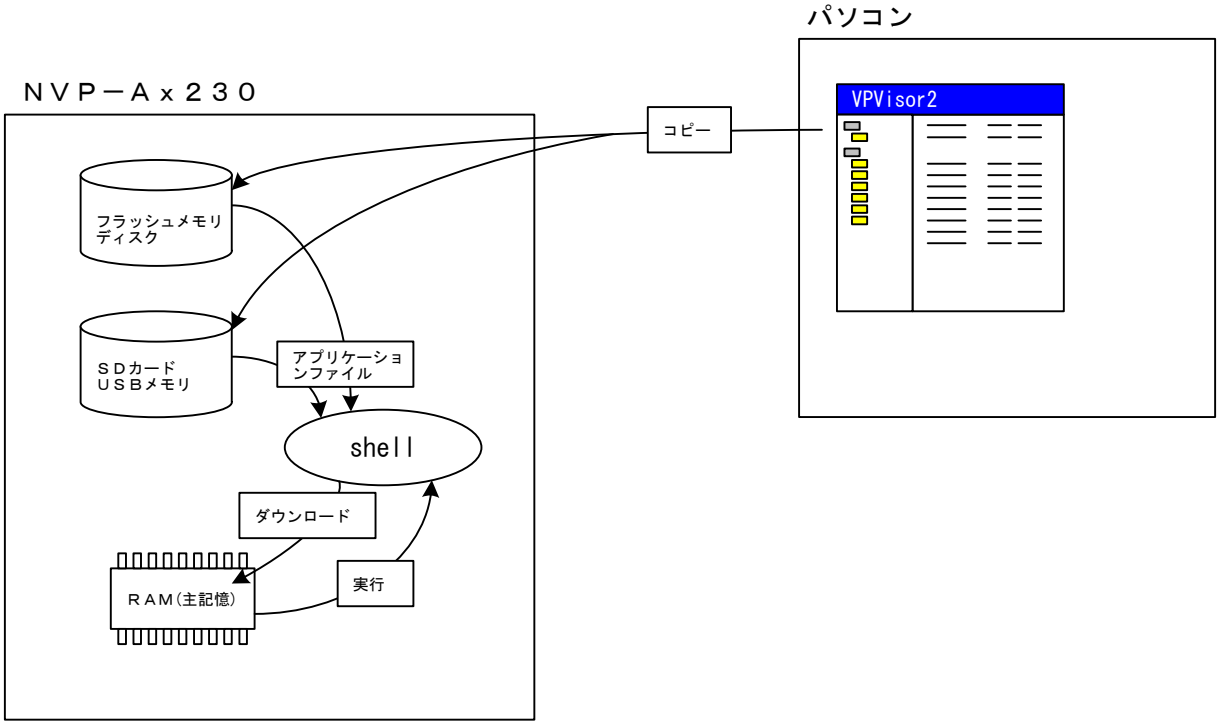


図6-1-1 アプリケーション開発

ＮＶＰ－Ａｘ２３０では、アプリケーションを実行する際にフラッシュメモリディスク、ＳＤカード、または、ＵＳＢメモリにディスクにアプリケーションの実行ファイルを書き込むことが必要です。ＮＶＰ－Ａｘ２３０ではWindows上のＧＵＩツール「VPVisor2」を使用し、パソコンのディスクからフラッシュメモリディスク、ＳＤカード、またはＵＳＢメモリにアプリケーションの実行ファイルを簡単にコピーすることができます。

6.2 アプリケーションの動作

6.2.1 メイン関数と終了関数

アプリケーションは必ず、メイン関数と終了関数を持ちます。それぞれの関数仕様は、以下のように決められています。アプリケーションの実行マネージャーは、アプリケーションの実行時、アプリケーションの実行ファイルの以下の関数エントリを検索し、タスクをクリエイトしてmain(又はMain)エントリアドレスから実行を開始します。そして、main関数からのリターンでTerminate関数を実行し、アプリケーションのタスクを終了、削除します。実行ファイルに少なくともmain関数エントリが無い場合は実行できません。

```
メイン関数      :      void main( int argc, char *argv[] )
                 又は  void Main( int argc, char *argv[] )
終了関数        :      void Terminate( void )
```

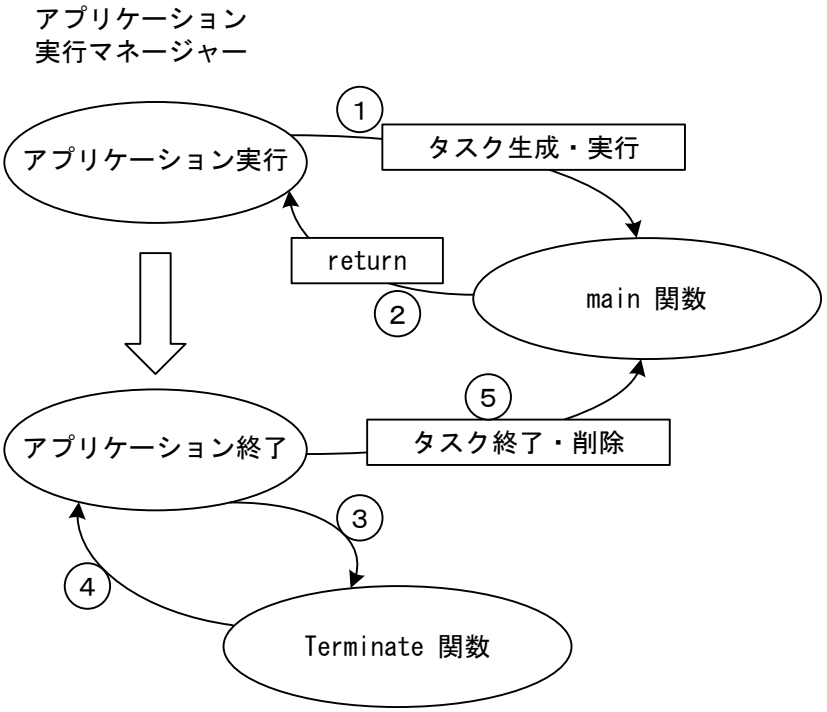


図6-2-1 実行マネージャー

6.2.2 メイン(main)関数

アプリケーション実行マネージャーは、アプリケーション実行ファイルからMain関数のエントリを検索し、タスク生成・実行を行います。そのとき、main関数の引数 argc, argv には、ｓｈｅｌｌのコマンドラインで指定したパラメータが渡されます。このmain関数からのリターンはアプリケーションの自立的な終了となり、アプリケーション実行マネージャーは、アプリケーションタスクの終了と削除を行います。

6.2.3 終了(Terminate)関数

Terminate関数は、main関数のリターン時、アプリケーション実行マネージャーからコールされます。Terminate関数終了後アプリケーション実行マネージャーは、アプリケーションタスクの終了と削除を行います。Terminate関数には、アプリケーションが終了する場合に特に必要な処理を行います。Terminate関数は、Terminate時のデッドロックを防ぐため、タイムアウト（３秒）を設けています。特に必要の無い場合は、Terminate関数は記述の必要がありません(デフォルトのTerminate関数が実行されます)。

6.3 アプリケーションの作成

アプリケーションモジュールの作成手順は、コーディング及びスタック領域の設定以外ダウンロードモジュールと同じです。

6.3.1 コーディング

アプリケーションは、少なくともメイン(main)関数が必要です。終了処理で特別な処理が必要な場合、終了(Terminate)関数を記述してください。また、ダウンロードモジュールと同様に"ipxdef.h","ipxsys.h","ipxprot.h"をインクルードし、C言語の関数（サブルーチン）形式で記述して下さい。

メイン関数

```
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"


void main(int argc, char *argv[])
{


   ...........
   ...........
}
```

終了関数

```
void Terminate(void)
{
   ...........
   ...........
}
```

6.3.2 ユーザー領域

ダウンロードモジュールの実行領域としてユーザーが使用できるエリアは、

0xB000:0000 ～ 0xB3FF:FFFF

までの６４Mバイトです。この領域にプログラム、スタティックデータなどをマッピングするように、リンク時に指定して下さい。本システムのメモリマップは３２ビットアドレス空間で、ユーザ領域は全てキャッシング対象となります。

6.3.3 プログラムローダー

ユーザー領域(0xB000:0000 ～ 0xB3FF:FFFF)は、システムにより管理されています。
プログラムのローダーは、実行ファイル(.abs)からプログラムがロードされる領域を計算し、その領域をユーザ領域から確保し、その領域に実行コードをダウンロードします。そして、Main関数及びTerminate関数エントリを検索し、Mainエントリタスクを生成実行します。実行コードがダウンロードされる領域がすでに確保されている場合、ローダーは実行ファイルの実行コードをダウンロードできません。その場合は、すでに実行しているアプリケーションを終了するか、アドレスのマッピングを変更してください。

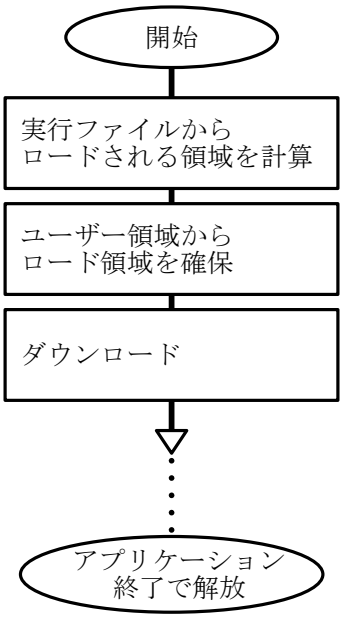


図6-3-1 ロードフロー

6.3.4 スタック

アプリケーションで使用できるスタック領域は、デフォルトで６４Ｋバイトです。変更する場合は、ｓｈｅｌｌのコマンドラインからアプリケーション実行オプション「#s」で必要な容量を指定して下さい。

スタックオプション ： #s<スタックサイズ(16進数)>

例　スタックサイズを２５６Ｋバイトにする場合

```
sample.abs #s40000
```

6.3.5 プロジェクトのビルド

オンボードＣＰＵのコンパイラでコンパイル、リンクを行い実行ファイル(.abs)をビルドします。なお、コンパイラのプロジェクトの設定等の詳細は、ＳＨＣ　Ｃｏｍｐｉｌｅｒ設定ガイドを参照して下さい。

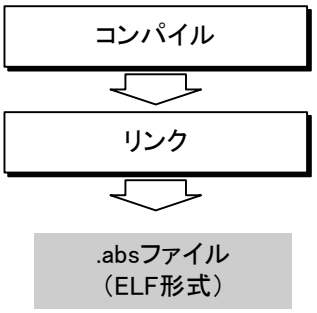


図6-3-2　プロジェクトのビルド

6.3.6 デバッグ

デバッグターミナル(DbgTermNT2)やLANターミナル(xTerm2)を使用し、オンボードＣＰＵ側のprintf()やscanf()などの標準入出力関数でデータの確認などの簡単なデバッグ作業を行うことができます。

アプリケーションモジュールの開発環境や開発手法については第３章で説明していますので、そちらを参照してください。

6.4 アプリケーションの実行

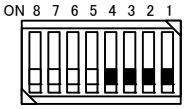
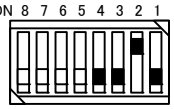
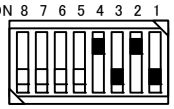
6.4.1 オンボードシステムファイル

ＮＶＰ－Ａｘ２３０で電源投入時にアプリケーションモジュールを自動的に実行するためには、オンボードフラッシュメモリディスクの『sys:』ドライブのルートにオンボードシステムファイル『ax2core.sys』が存在する必要があります。オンボードシステムファイルが存在しない場合、電源投入時にアプリケーションモジュールを立ち上げる機能は動作しません。なお、ＮＶＰ－Ａｘ２３０は、出荷時にオンボードシステムファイル『ax2core.sys』が『sys:』ドライブにコピーされています。

6.4.2 デフォルトドライブ

ＮＶＰ－Ａｘ２３０は、電源投入やシステムリセット時にあらかじめデフォルトのドライブが設定されます。デフォルトドライブは『fmd:』、『ata:』、『usb:』をフロントパネルのディップスイッチ（ＳＷ１：１－４)の設定により指定することができます。出荷設定では『fmd:』に設定されています。下表にSW1の設定値とデフォルトドライブを示します。

ただし、ＮＶＰ－Ａｘ２３０ＣＬでは『ata:』、『usb:』を設定してもデフォルトドライブは『fmd:』になります。

SW1 1-4 の設定			
デフォルトドライブ	fmd:	ata:	usb:

6.4.3 スタートアップファイル

ＮＶＰ－Ａｘ２３０は、システムが立ち上ると、最初にスタートアップファイル(startup.bat)実行のためのｓｈｅｌｌが起動し、スタートアップファイルに記述されているｓｈｅｌｌコマンドやアプリケーションを実行します。電源投入時又は、リセット時にユーザーアプリケーションを実行する場合は、このスタートアップファイルにコマンドを記述します。スタートアップファイルは、「startup.bat」というファイル名のテキストファイルです。スタートアップファイルが実行されるためには、「startup.bat」がデフォルトドライブのルートに書き込まれてある必要があります。他のドライブやディレクトリに存在してもサーチしませんので注意してください。

記述 例

```
sample.abs↵
```

sample.abs を実行します

※改行は必ず入れてください。

※ボード出荷時、startup.bat は書き込まれていません。

※startup.bat は『VPVisor2』で編集可能です。

※電源投入時、startup.batから実行した場合の標準入出力関数のモニタリングは『xTerm2』になります。

6.4.4 shell からの実行

Windows上のアプリケーション『DbgTermNT2.exe』でターミナル上から対話形式でコマンドを実行できるようになります。

『DbTermNT2.exe』起動すると「fmd:¥>」のプロンプトが出力され、コマンド入力待ち状態になります。

（例は、デフォルトドライブが「fmd:」に設定されている場合です）

```
fmd:¥>
```

第7章 ダウンロードモジュール

7.1 モジュール化の概要

オンボードＣＰＵで動作させるアプリケーションをダウンロードモジュールと呼びます。ダウンロードモジュールには動作の特徴から

・インテリジェントモジュール
・画像認識タスクモジュール
・割込起動モジュール

という３通りのモジュール化の方法があります。

ＮＶＰ－Ａｘ２３０のシステムでは、実行モジュールはボード上のＲＯＭやＲＡＭにあり、画像認識コマンドでの処理は画像認識プロセッサとオンボードＣＰＵにより行われています。

Ｗｉｎｄｏｗｓパソコンから実行される画像認識コマンドでは、コマンド１つ１つについて以下のフローに従いオンボードＣＰＵで処理が実行されます。そのため、画像処理コマンドを実行するたびにオンボードＣＰＵの処理時間以外にデータ転送やコマンド起動に数ｍＳ程度のオーバヘッドが加わります。

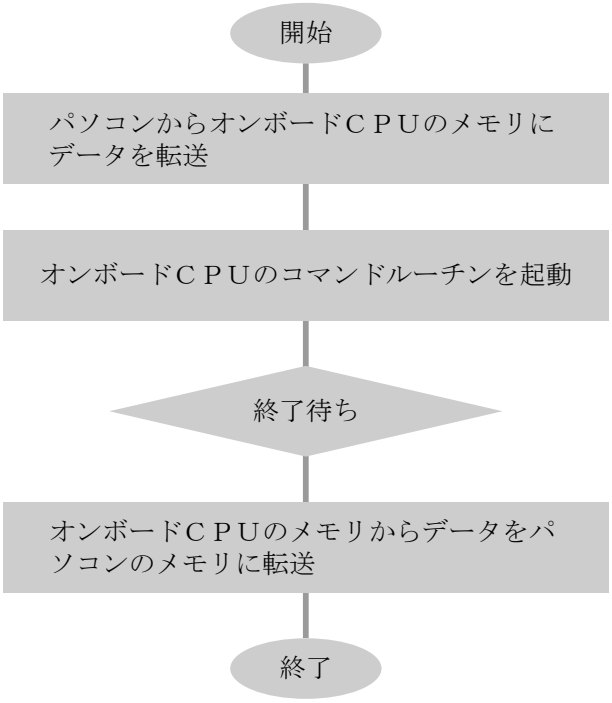


図7-1-1　コマンド実行フロー

そこで、画像認識コマンドでは、パソコンとのＩ／Ｆでのオーバヘッド削減と処理の分散化のために、ユーザーオリジナルの画像認識アプリケーションをオンボードＣＰＵで実行できるように

・インテリジェントモジュール
・画像認識タスクモジュール
・割込起動モジュール

という３つのフレームモデルを実装しています。このフレームモデルの特徴は、Ｃ言語の関数形式でパソコンからオンボードＣＰＵ側に簡単にデータを渡すことができることと、モジュールの実行制御を簡単に行うことができるということです。なお、オンボードＣＰＵで実行されるユーザーオリジナルの画像処理アプリケーションは、オンボードＣＰＵ（ＳＨ４）のコンパイラでコンパイルし実行オブジェクトを作成します。

7.1.1 コーディング

ダウンロードモジュールは、"ipxdef.h","ipxsys.h","ipxprot.h"をインクルードし、Ｃ言語の関数（サブルーチン）形式で記述して下さい。
関数名は任意でパラメータはint型、float型、ポインタ型で０から最大３２個まで指定できます。これらのパラメータは、ＰＣ側のプログラムで「モジュールサポートコマンド」によりデータが設定されます。

```
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void imgprc(int a,float b,short *tbl)
{
   ..........
   ..........
}
```

7.1.2 ユーザー領域

ダウンロードモジュールの実行領域としてユーザーが使用できるエリアは、

0xB000:0000 ～ 0xB3FF:FFFF

までの６４Mバイトです。この領域にプログラム、スタティックデータなどをマッピングするように、リンク時に指定して下さい。

7.1.3 プログラムローダー

ユーザー領域(0xB000:0000 ～ 0xB3FF:FFFF)は、システムにより管理されています。
プログラムのローダーは、実行ファイル(.abs)からプログラムがロードされる領域を計算し、その領域をユーザ領域から確保し、その領域に実行コードをダウンロードします。実行コードがダウンロードされる領域がすでに確保されている場合、ローダーは実行ファイルの実行コードをダウンロードできません。その場合は、すでに実行しているアプリケーションを終了しその領域を解放するか、アドレスのマッピングを変更してください。プログラムローダーのコマンドはLoadIPDLM、ロードした領域を解放するコマンドはUnloadIPDLMです。また、LoadIPDLMは、ダウンロードと同時に指定シンボルのアドレスを抽出します。

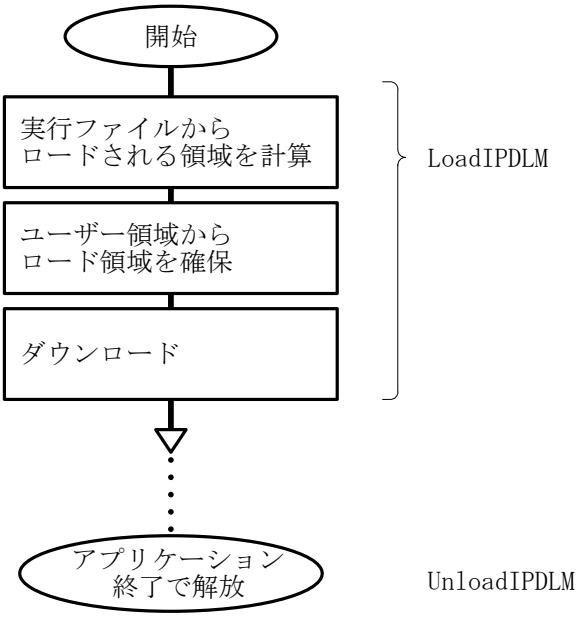


図7-1-2 ロードフロー

7.1.4 スタック領域

ダウンロードモジュールのスタック領域は、システム領域から割り当てられ、それぞれのモジュールにより割り当てられるスタックサイズが異なります。

表7-1-1　スタックサイズ

モジュールの属性	スタックサイズ	備考
インテリジェントモジュール	６４Ｋバイト	
画像認識タスクモジュール	タスク生成時に指定	ユーザータスクのスタックの合計が１．５Ｍバイトを超えないこと
割込み起動モジュール	タスク生成時に指定	

7.1.5 プロジェクトのビルド

オンボードＣＰＵのコンパイラでコンパイル、リンクを行い実行ファイル(.abs)をビルドします。なお、コンパイラのプロジェクトの設定等の詳細は、ＳＨＣ　Ｃｏｍｐｉｌｅｒ設定ガイドを参照して下さい。

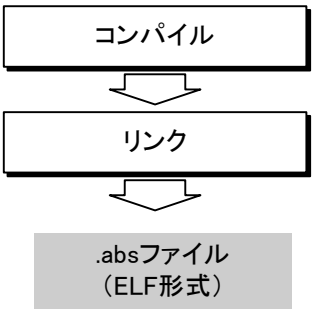


図7-1-3　プロジェクトのビルド

7.1.6 デバッグ

LANターミナル(xTerm2)を使用し、オンボードＣＰＵ側のprintf()やscanf()などの標準入出力関数でデータの確認などの簡単なデバッグ作業を行うことができます。

アプリケーションモジュールの開発環境や開発手法については第３章で説明していますので、そちらを参照してください。

7.2 インテリジェントモジュール

インテリジェントモジュールとは、画像認識コマンドを複数組み合わせて作成するユーザーオリジナルの画像認識コマンドのことです。

ＮＶＰ－Ａｘ２３０のシステムでは、画像認識は画像認識プロセッサとオンボードＣＰＵにより行っているため、ユーザーの画像認識アプリケーションの一部をインテリジェントモジュールとして実装することにより、処理の分散化を実現できます。

インテリジェントモジュールは画像認識コマンドの一部として登録し実行されます。最大２５６モジュールの登録が可能です。

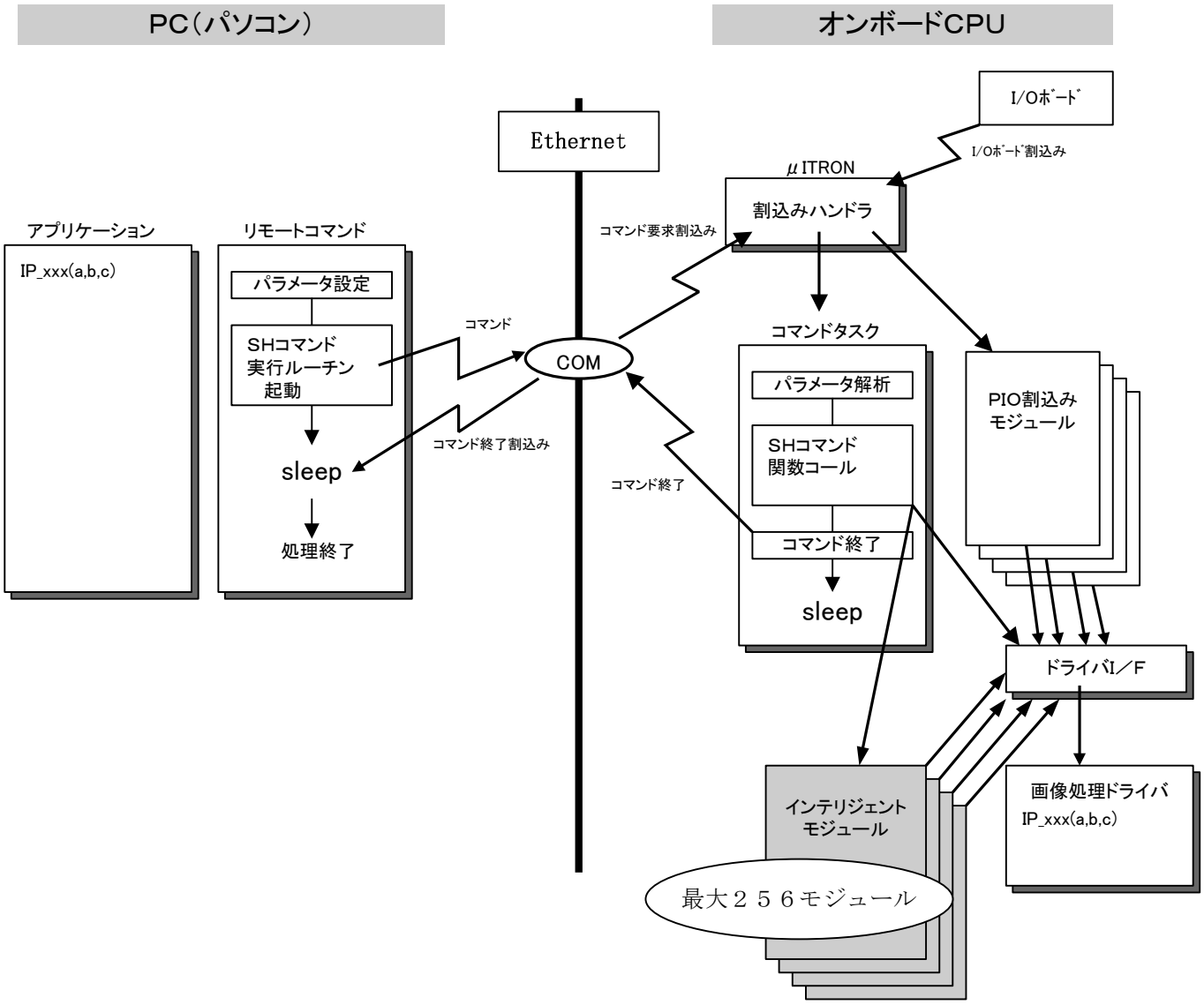


図7-2-1　インテリジェントモジュールの仕組み

7.2.1 インテリジェントモジュールのコーディング

Ｗｉｎｄｏｗｓパソコンから起動されるオンボードＣＰＵのプログラムは、Ｃ言語の関数（サブルーチン）形式で記述して下さい。関数名は任意でパラメータはint型、float型、ポインタ型で０から最大３２個まで指定できます。これらのパラメータは、ＰＣ側のプログラムで「モジュールサポートコマンド」によりデータが設定されます。それ以外は、ＰＣでのプログラムと同一にして下さい。

```
void inteli_module0(int a,float b,short *tbl)
{
   ...........
   ...........
}
```

7.2.2 インテリジェントモジュールの実行

本項では、オンボードＣＰＵ側のインテリジェントモジュールを実行するためのＷｉｎｄｏｗｓパソコン側のプログラムについて説明します。
以下に実行フローを示します。

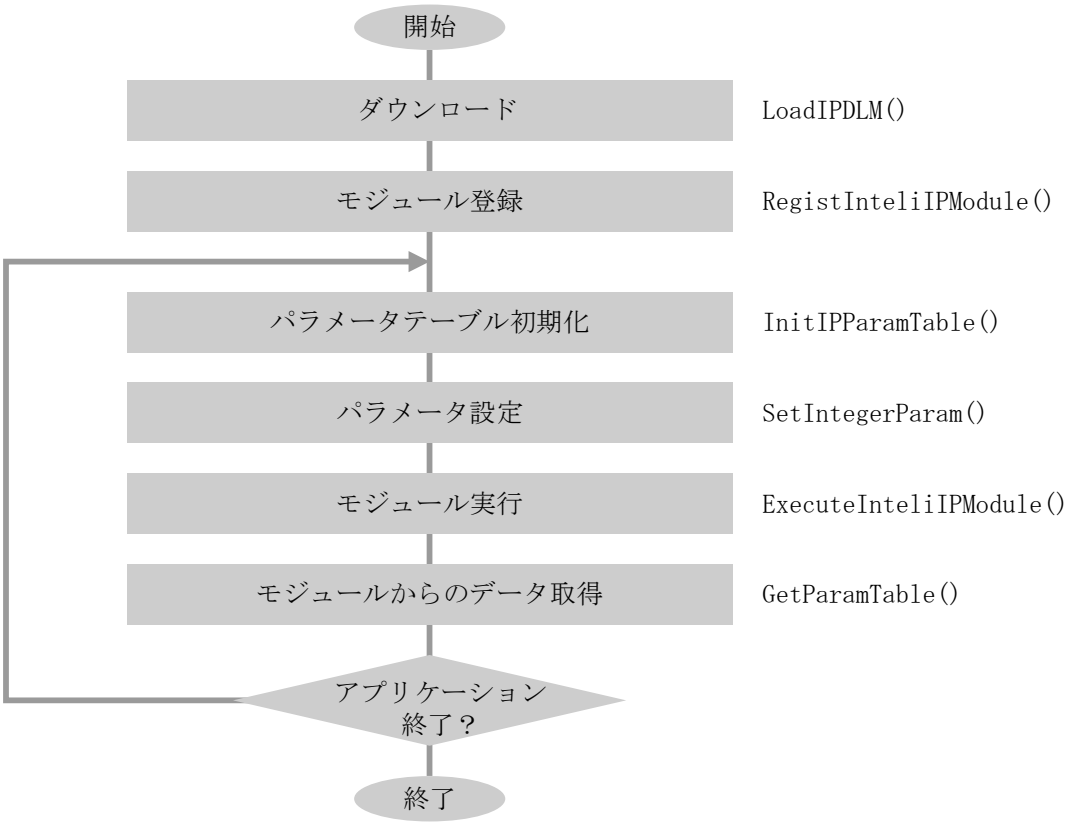


図7-2-2　モジュール実行フロー

(1) インテリジェントモジュールのダウンロード

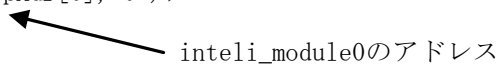
ビルドした「abs」ファイルをLoadIPDLM()コマンドを使用してボードにダウンロードします。

```
char            *filename = "intelimod.abs";/* ファイルパス                          */
char            *modulename = NULL;         /* モジュール名                          */
int             mode = 0;                   /* モード                                */
IPDLMSymbolTbl  InOpt;                      /* ダウンロードモジュール情報構造体      */
IPDLMEntryInf   OutInf;                     /* ダウンロードモジュールエントリ情報構造体 */
unsigned long   symadr[3];                  /* シンボルアドレス                      */
char            *symbol[] = {               /* シンボル名                            */
                    "inteli_module0",
                    "inteli_module1",
                    "inteli_module2"
                };

/* 変数初期化 */
memset( &InOpt, 0, sizeof( InOpt ) );
memset( &OutInf, 0, sizeof( OutInf ) );
memset( symadr, 0, sizeof( symadr ) );

/* ダウンロードモジュール情報設定*/
InOpt.style = IPDLM_SYMBOL_C_STYLE;
InOpt.count = 3;
InOpt.pSymbol = symbol;

/* ダウンロードモジュールエントリ情報設定 */
OutInf.pAdr = symadr;

/* ダウンロードモジュールのロード */
LoadIPDLM( filename, mode, modulename, &InOpt, &OutInf );
```

(2) インテリジェントモジュールの登録

ダウンロードしたインテリジェントモジュールをRegistInteliIPModule()コマンドで登録します。

```
RegistInteliIPModule( MODULE_0, OutInf.pAdr[0], 0 );
```



(3) インテリジェントモジュールへのパラメータ渡し

登録したインテリジェントモジュールにパラメータを渡す方法を説明します。

(a) パラメータテーブルの初期化

InitIPParamTable()コマンドでパラメータテーブルを初期化します。

```
MODULE_PARAM_TBL   prmtbl;

InitIPParamTable( &prmtbl, 3, INTELI_MODULE, 0, 0 );
```

(b) パラメータの設定

インテリジェントモジュールのパラメータがint型やfloat型の場合、SetIntegerParam(),SetFloatParam()コマンドでパラメータテーブルにデータを設定します。

```
SetIntegerParam( &prmtbl, PARAM_1, 123 );
SetFloatParam( &prmtbl, PARAM_2, 4.567 );
```

(c) テーブル（配列）パラメータの設定

インテリジェントモジュールのパラメータが配列の場合、SetParamTable()コマンドでパラメータテーブルにデータを設定します。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(4) インテリジェントモジュールの実行

ExecuteInteliIPModule()コマンドにより、パラメータのデータを転送し、登録したインテリジェントモジュールを実行します。

```
ExecuteInteliIPModule( MODULE_0, &prmtbl, NULL );
```

このコマンドはオンボードＣＰＵのインテリジェントモジュールを起動し、その処理が終了するまでスリープします。そして、処理が終了した時点でウェイクアップしてこのコマンドが終了します。

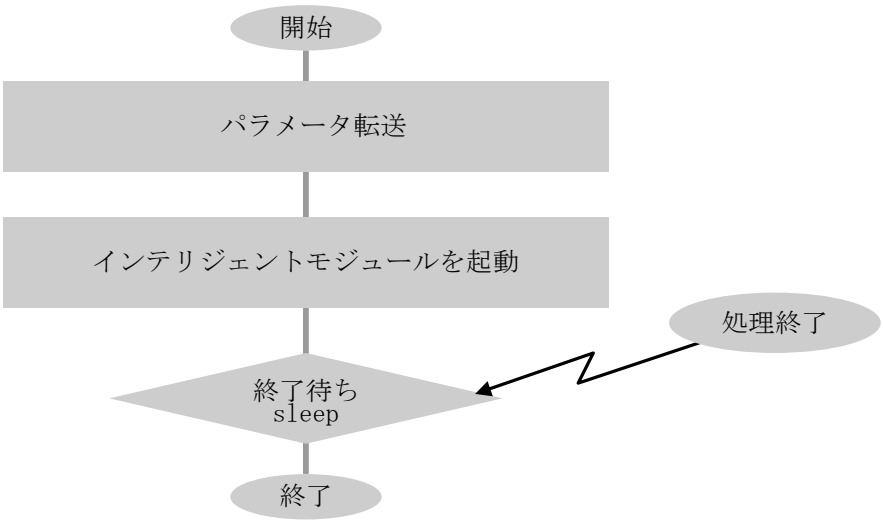


図7-2-3　コマンド実行フロー

(5) インテリジェントモジュールからのデータ取得

インテリジェントモジュール実行後にデータ取得する方法を説明します。

(a) 指定した配列パラメータが入出力データの場合

インテリジェントモジュールのパラメータが配列でインテリジェントモジュールにデータを渡してなおかつインテリジェントモジュール実行終了後にデータを取得する場合、SetParamTable()コマンドでパラメータテーブルにデータを設定し、インテリジェントモジュール実行終了後、GetParamTable()コマンドでデータを取得して下さい。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
ExecuteInteliIPModule( MODULE_0, &prmtbl, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(b) 指定した配列パラメータが出力のみのデータの場合

インテリジェントモジュールのパラメータが配列でインテリジェントモジュール実行終了後にデータを取得する場合、AllocParamTable()コマンドでパラメータテーブルの領域を確保し、インテリジェントモジュール実行終了後、GetParamTable()コマンドでデータを取得して下さい。

```
AllocParamTable( &prmtbl, PARAM_3, sizeof(tbl) );
ExecuteInteliIPModule( MODULE_0, &prmtbl, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

7.3 画像認識タスクモジュール

ボード上のＣＰＵは、システムをリアルタイムに動作させるためリアルタイムＯＳ：μＩＴＲＯＮを使用しています。タスクとはアプリケーションを独立して並列に処理可能な単位で分割したプログラムのことです。ボード上のＣＰＵではシステムとしてさまざまなタスクがすでに動作しており、スタンドアロンの実行モジュールはタスクとして管理されます。画像認識タスクモジュールは、μＩＴＲＯＮの１つのタスクとして生成、実行されるため、パソコンとは独立して動作することが可能です。ユーザータスクは最大３２モジュール（割込み起動モジュール含む）の生成が可能です。

7.3.1 画像認識タスクモジュールの動作

画像認識タスクモジュールはμＩＴＲＯＮの１つのタスクとして生成し実行します。そのため、モジュールをμＩＴＲＯＮのタスクの動作に適合するようにプログラムします。

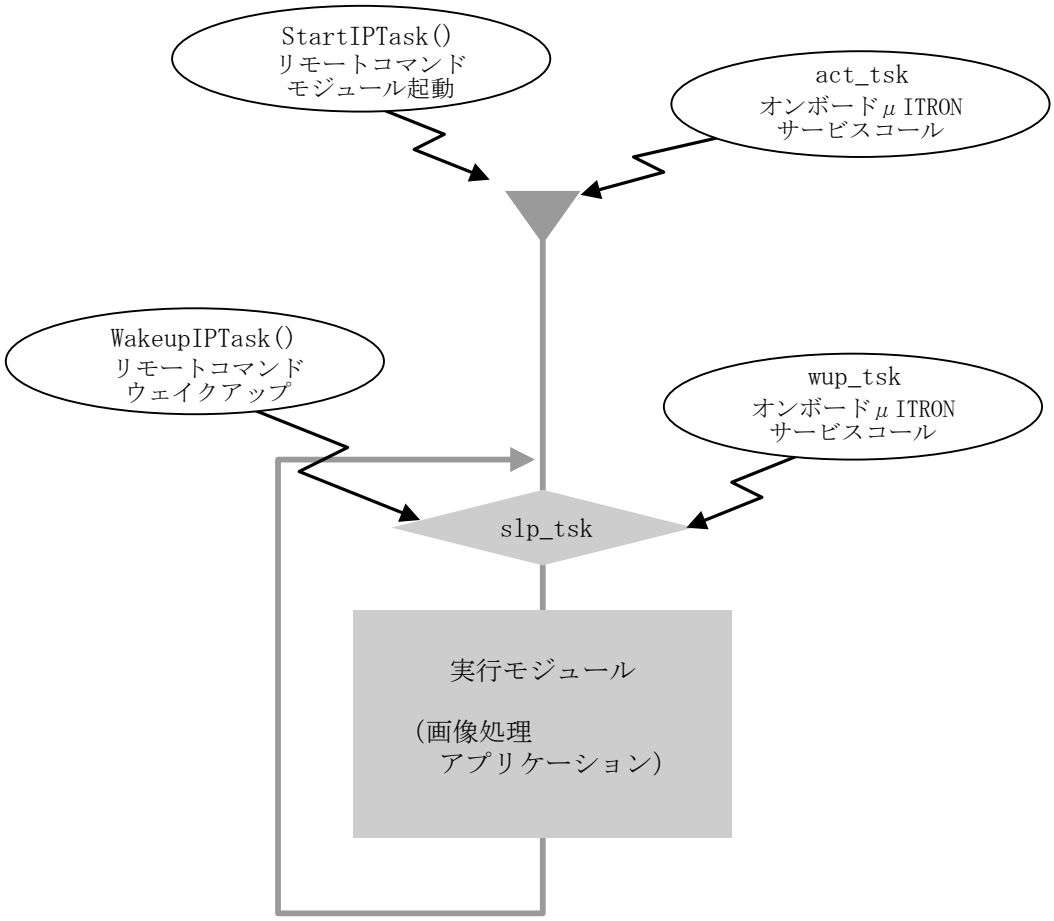


図7-3-1　画像処理タスクモジュールの動作

7.3.2 画像認識タスクモジュールのコーディング

画像認識タスクは、Ｃ言語の関数（サブルーチン）形式で記述して下さい。関数名は任意ですがパラメータは渡すことができません。

```
void task_module0( )
{
   ..........
   ..........

   for( ;; ){
      slp_tsk();

      /*画像処理*/
      ..........
      ..........
   }

   exd_tsk();
}
```

7.3.3 パソコンとの同期

画像認識タスクモジュールに対して、Ｗｉｎｄｏｗｓパソコン側の処理とオンボードＣＰＵとの同期をとる方法としてWaitforIPTaskSignal()とSendSignaltoPC()コマンドを用意しています。

まず、パソコン側はWaitforIPTaskSignal()コマンドでオンボードＣＰＵの処理が終了するのを待ちます。そして、オンボードＣＰＵ側からSendSignaltoPC()コマンドを実行すると、パソコン側に終了シグナルが送信され、パソコン側はWaitforIPTaskSignal()コマンドのウェイト処理から抜けます。

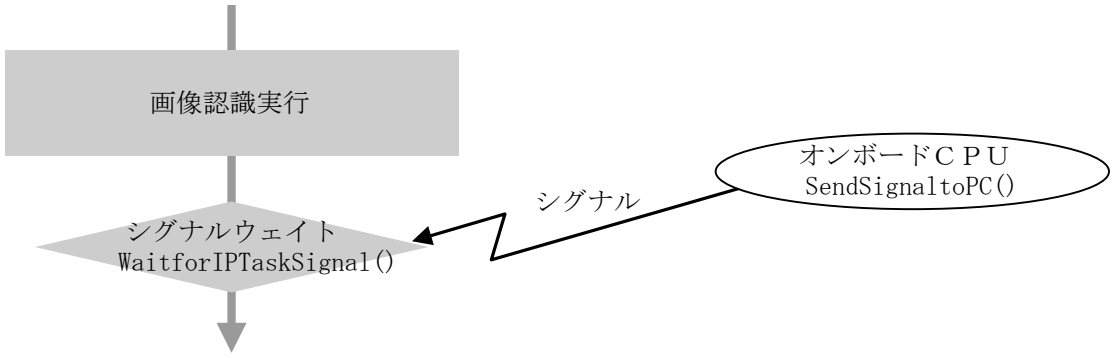


図7-3-2　ＰＣとの同期

7.3.4 画像認識タスクモジュールの制御

画像認識タスクモジュールを制御するためのパソコン側の制御フローを示します。

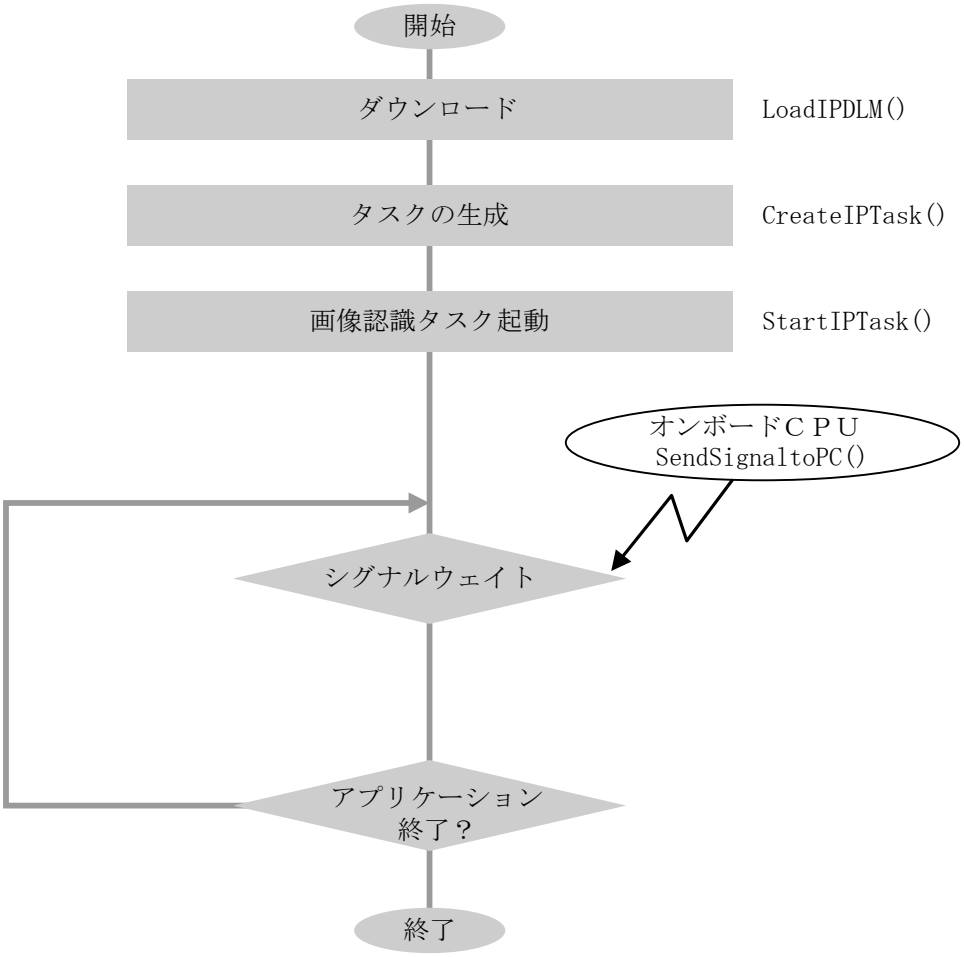


図7-3-3　画像認識タスクモジュール制御フロー

(1) 画像認識タスクモジュールのダウンロード

ビルドした「abs」ファイルをLoadIPDLM()コマンドを使用してボードにダウンロードします。

```
char            *filename = "taskmod.abs";  /* ファイルパス                          */
char            *modulename = NULL;         /* モジュール名                          */
int             mode = 0;                   /* モード                                */
IPDLMSymbolTbl     InOpt;                   /* ダウンロードモジュール情報構造体      */
IPDLMEntryInf      OutInf;                  /* ダウンロードモジュールエントリ情報構造体 */
unsigned long      symadr[3];               /* シンボルアドレス                      */
char            *symbol[] = {               /* シンボル名                            */
                   "task_module0",
                   "task_module1",
                   "task_module2"
                };

/* 変数初期化 */
memset( &InOpt, 0, sizeof( InOpt ) );
memset( &OutInf, 0, sizeof( OutInf ) );
memset( symadr, 0, sizeof( symadr ) );

/* ダウンロードモジュール情報設定*/
InOpt.style = IPDLM_SYMBOL_C_STYLE;
InOpt.count = 3;
InOpt.pSymbol = symbol;

/* ダウンロードモジュールエントリ情報設定 */
OutInf.pAdr = symadr;

/* ダウンロードモジュールのロード */
LoadIPDLM( filename, mode, modulename, &InOpt, &OutInf );
```

(2) タスクの生成

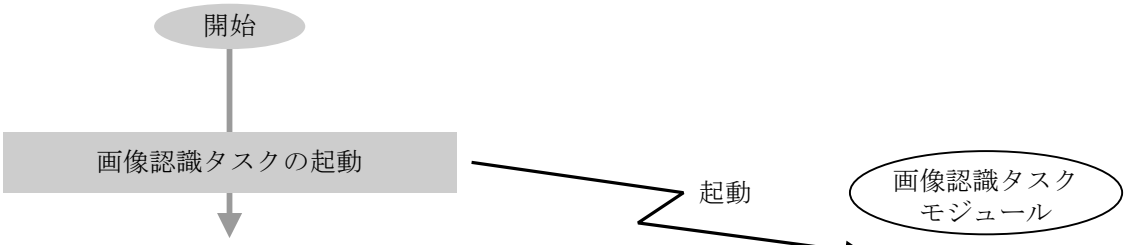
ダウンロードした画像認識タスクモジュールを生成します。

```
int tskid;
CREATE_TASK_TBL   iptsk;

iptsk.task_addr  = OutInf.pAdr[0];/* task_module0のアドレス */
iptsk.priority   = TASK_PRI_NORMAL;
iptsk.pbuff_size = 0;
iptsk.stack_size = 0x4000;
iptsk.task_opt   = 0;
iptsk.param_opt  = 0;

tskid= CreateIPTask( &iptsk );
```

(3) 画像認識タスクモジュールの起動

CreateIPTaskで生成した画像認識タスクを起動します。

```
StartIPTask( tskid );
```

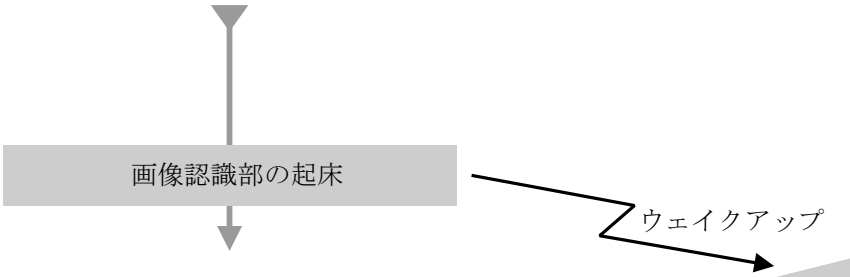


(4) 画像認識タスクモジュールのウェイクアップ

WakeupIPTask()コマンドにより、画像認識タスクをウェイクアップします。

このコマンドはオンボードＣＰＵの割込起動モジュールの実行部をウェイクアップするだけです。処理終了ウェイトは行いません。

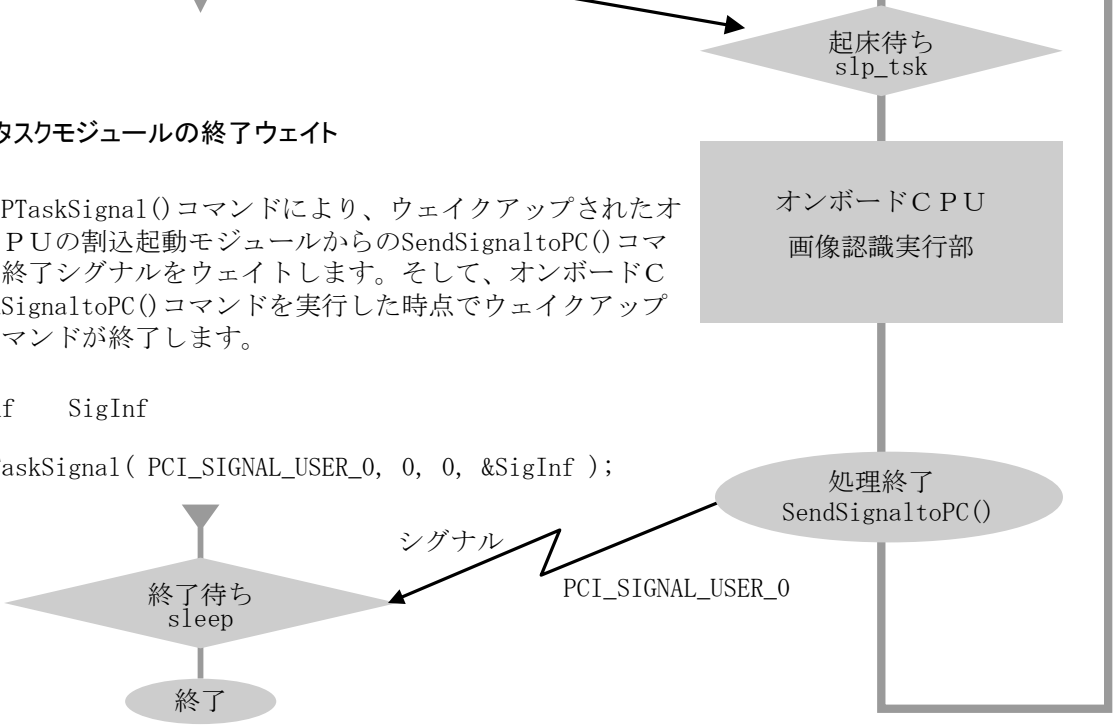
```
WakeupIPTask(  tskid  );
```



(5) 画像認識タスクモジュールの終了ウェイト

WaitforIPTaskSignal()コマンドにより、ウェイクアップされたオンボードＣＰＵの割込起動モジュールからのSendSignaltoPC()コマンドによる終了シグナルをウェイトします。そして、オンボードＣＰＵでSendSignaltoPC()コマンドを実行した時点でウェイクアップしてこのコマンドが終了します。

```
WaiIPSigInf    SigInf

WaitforIPTaskSignal( PCI_SIGNAL_USER_0, 0, 0, &SigInf );
```



7.4 割込み起動モジュール

7.4.1 PIO割込について

ＮＶＰ－Ａｘ２３０は、フォトカプラによりアイソレートされたパラレルの入力／出力ポート（ＰＩＯ）をそれぞれ８ビット持っています。

入力ポートはビット毎に割込に対応しています。入力ポートのレベルが「L」→「H」になる時の立ち上がりエッジによりオンボードＣＰＵに対して割込が発生します。

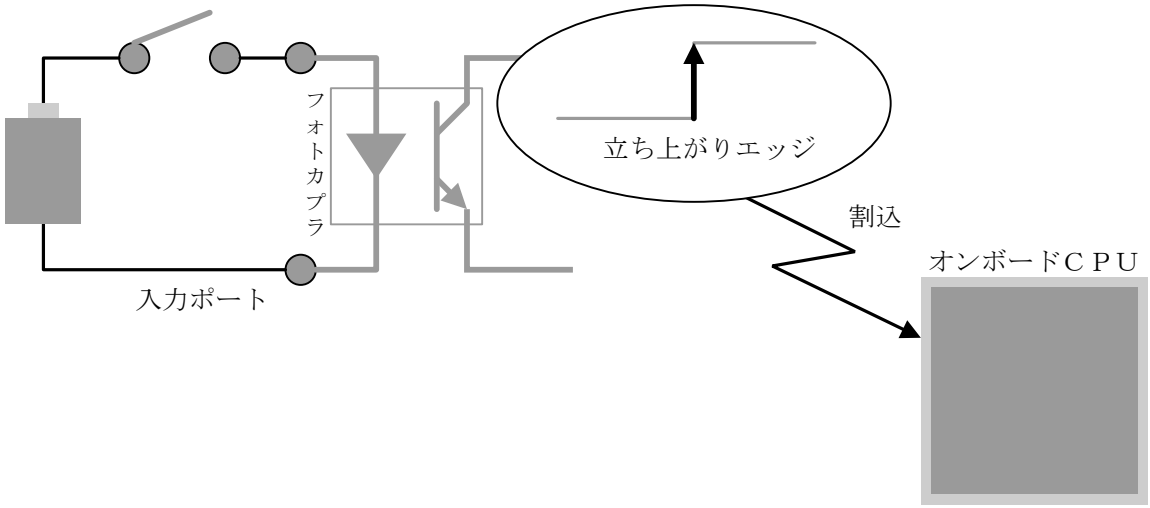


図7-4-1　入力ポート割込

ＰＩＯ割込に対する処理をオンボードＣＰＵのユーザオリジナルの画像認識アプリケーションで簡単に実行できるように割込起動モジュールというフレームモデルを実装しています。そのフレームモデルにより、オンボードＣＰＵのモジュールを登録するだけで、任意のＰＩＯ割込でそのモジュールを起動することができます。

ＮＶＰ－Ａｘ２３０はＰＩＯの入力ポートは８ビットあり、ＤＩ０～ＤＩ７ビットすべて割込起動モジュールに割り当てられます。

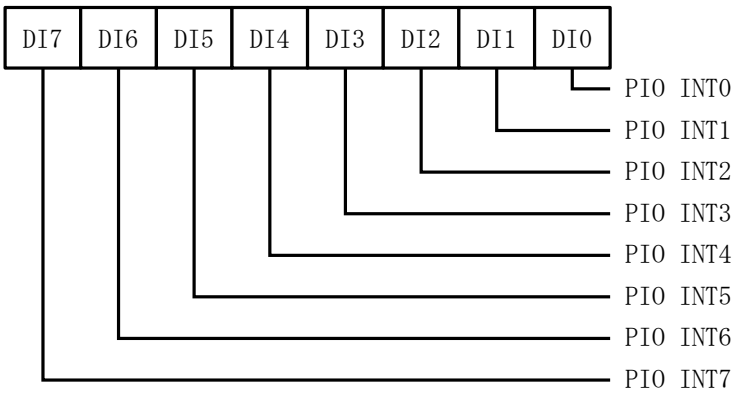


図7-4-2　ＰＩＯ入力ポートの割り当て

7.4.2 割込起動モジュールの概要

割込み起動モジュールとは、ＮＶＰ－Ａｘ２３０のパラレルＩ／Ｏ（ＰＩＯ）等の割込みにより起動することができるモジュールです。画像処理コマンドを複数組み合わせて作成したユーザーオリジナルの画像処理モジュールをＰＩＯの割込み等により起動するモジュールとして登録することができます。

ＰＩＯからの割込で起動できる割込起動モジュールは画像認識タスクモジュールの１つとして登録され実行されます。ＮＶＰ－Ａｘ２３０ではＰＩＯ入力ポートのデータビット０～データビット７（ＤＩ０～ＤＩ７）の割込にそれぞれ任意のタスクを対応させたモジュールとして登録することができます。ＮＶＰ－Ａｘ２３０では、ユーザは全部で３２個のタスクを生成することができますが、そのうちの８個を割込起動モジュールとして管理することが可能です。

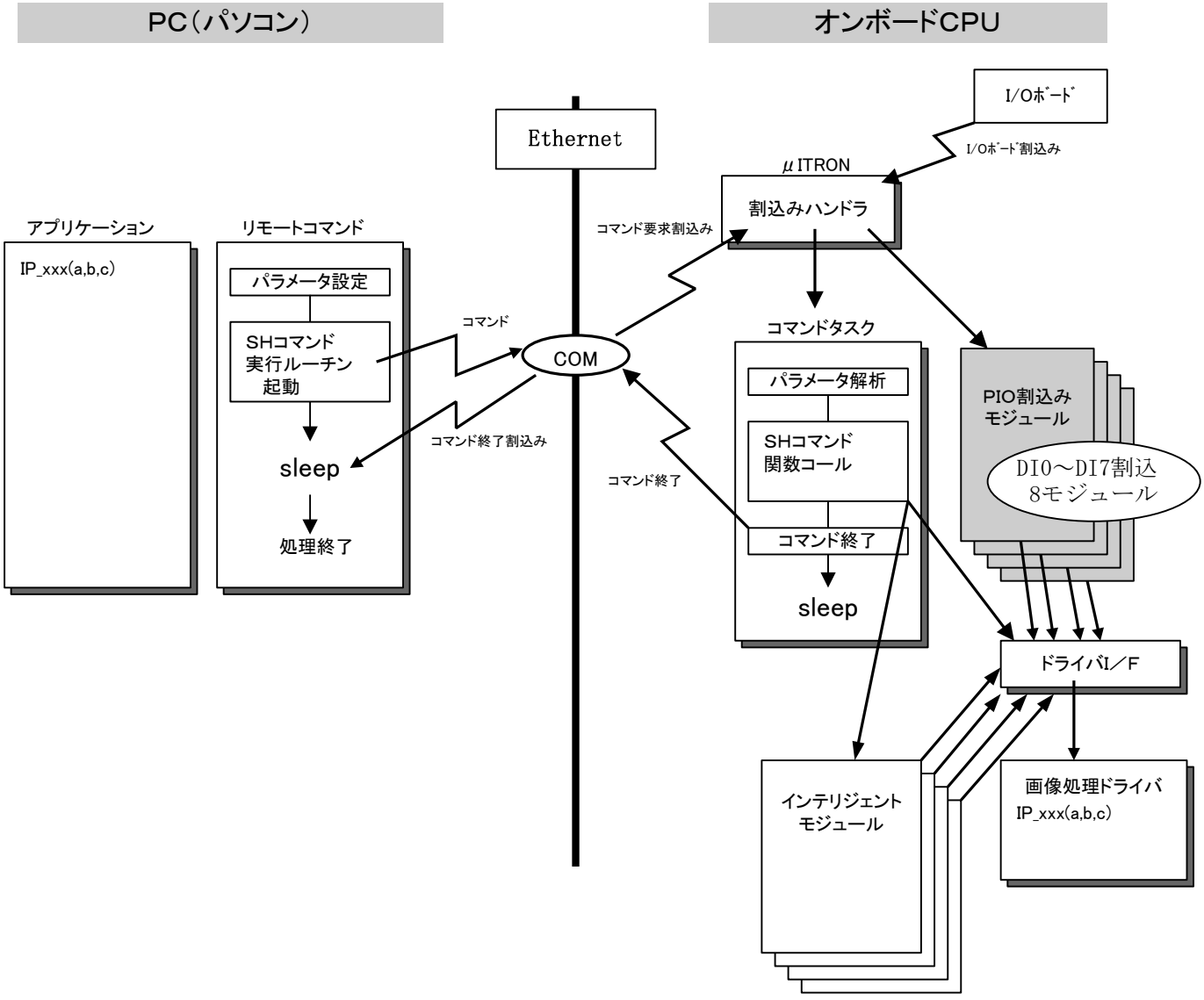


図7-4-3　割込モジュールの仕組み

7.4.3 割込起動モジュールの動作

割込起動モジュールはμＩＴＲＯＮの１つのタスクとして登録され実行されます。そのため、画像処理コマンドでは、そのモジュールをμＩＴＲＯＮのタスクの動作に適合するように実装します。

割込起動モジュールは、初期化部モジュールと実行部モジュールのペアのモジュールで１つのタスクを構成します。

初期化部モジュールは、StartIPTaskwithParam()コマンドで起動されTerminateIPTask()コマンドでタスクが終了するまで１度だけ動作します。初期化部は、ユーザオリジナルアプリケーションのグローバル変数などの初期化に使用します。

実行部モジュールは、WakeupIPTaskwithParam()コマンドや対応するＰＩＯの入力割込により起床（ウェイクアップ）します。実際の画像処理アプリケーションはこの実行部に登録されます。

また、初期化部モジュール、実行部モジュールには、インテリジェントモジュールと同様にパラメータを渡すことが可能です。

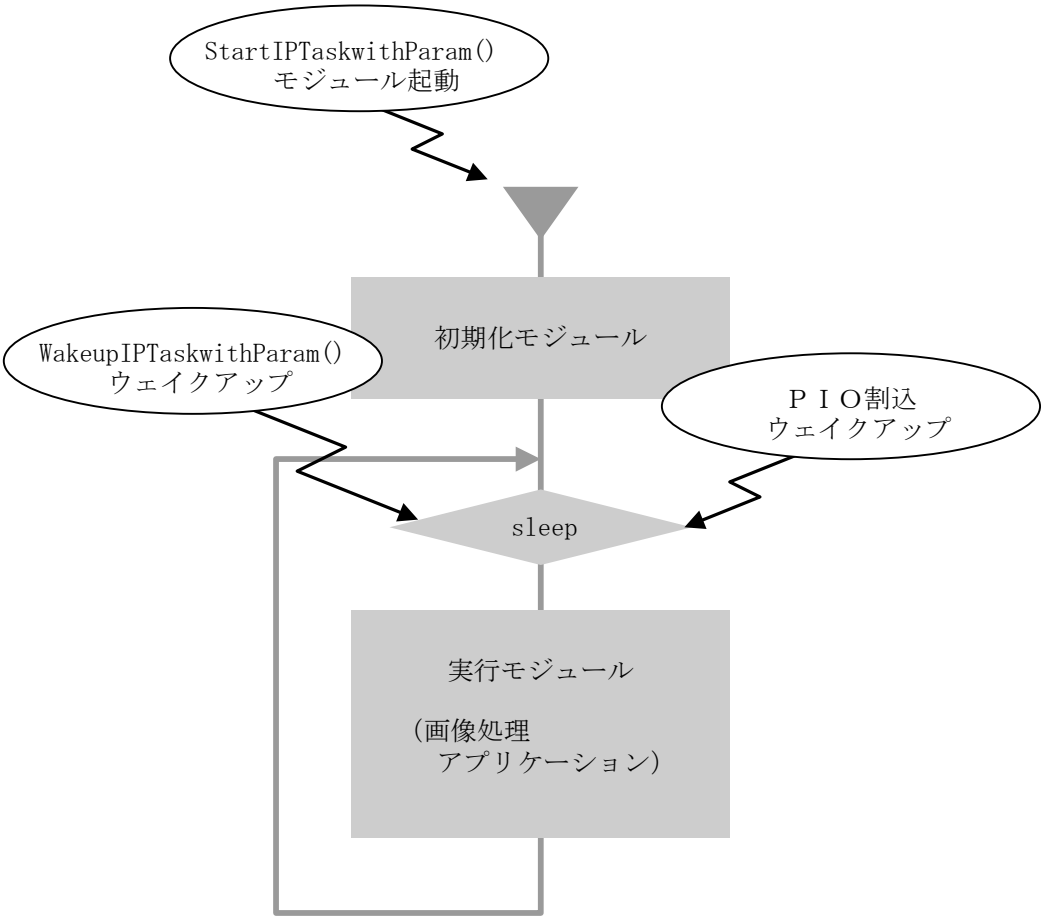


図7-4-4　割込モジュールの動作

7.4.4 割込み起動モジュールのコーディング

割込起動モジュールでは、初期化部モジュールと実行部モジュールの２つのモジュールを作成します。
Ｗｉｎｄｏｗｓパソコン側から起動されるオンボードＣＰＵのプログラムは、Ｃ言語の関数（サブルーチン）形式で記述して下さい。

初期化部モジュール

```
void pioint0_task_init(int a,float b)
{
   ..........
   ..........
}
```

実行部モジュール

```
void pioint0_task(int a,float b,short *tbl)
{
   ..........
   ..........
   SendSignaltoPC(.........);
}
```

関数名は任意でパラメータはint型、float型、ポインタ型で０から最大３２個まで指定できます。これらのパラメータは、パソコン側のプログラムで「モジュールサポートコマンド」によりデータが設定されます。それ以外は、パソコンでのコーディングと同一にして下さい。

7.4.5 割込起動モジュールの制御

本項では、オンボードＣＰＵ側の割込起動モジュールを制御するためのパソコン側のプログラムについて説明します。
以下に制御フローを示します。

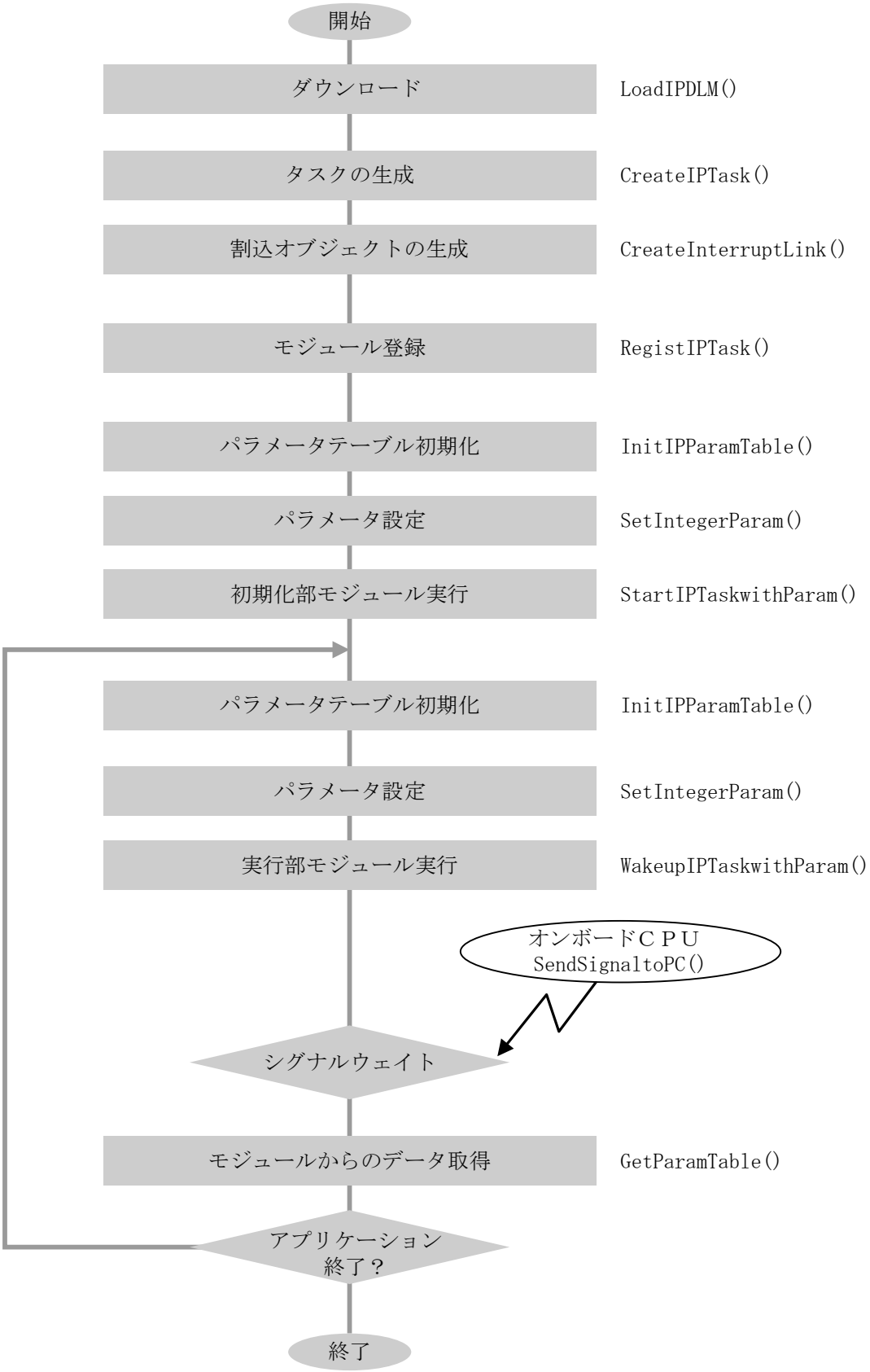


図7-4-5　割込起動モジュール制御フロー

(1) 割込起動モジュールのダウンロード

ビルドした「abs」ファイルをLoadIPDLM()コマンドを使用してボードにダウンロードします。

```
char            *filename = "pintmod.abs";  /* ファイルパス                          */
char            *modulename = NULL;         /* モジュール名                          */
int             mode = 0;                   /* モード                                */
IPDLMSymbolTbl  InOpt;                      /* ダウンロードモジュール情報構造体      */
IPDLMEntryInf   OutInf;                     /* ダウンロードモジュールエントリ情報構造体 */
unsigned long   symadr[2];                  /* シンボルアドレス                      */
char            *symbol[] = {               /* シンボル名                            */
                    "pint0_init",
                    "pint0_main",
                };

/* 変数初期化 */
memset( &InOpt, 0, sizeof( InOpt ) );
memset( &OutInf, 0, sizeof( OutInf ) );
memset( symadr, 0, sizeof( symadr ) );

/* ダウンロードモジュール情報設定*/
InOpt.style = IPDLM_SYMBOL_C_STYLE;
InOpt.count = 2;
InOpt.pSymbol = symbol;

/* ダウンロードモジュールエントリ情報設定 */
OutInf.pAdr = symadr;

/* ダウンロードモジュールのロード */
LoadIPDLM( filename, mode, modulename, &InOpt, &OutInf );
```

(2) タスクの生成

ダウンロードした割込み起動モジュールのタスクを生成します。

```
int tskid;
CREATE_TASK_TBL   iptsk;

iptsk.task_addr  = 0;  /* 割込み起動モジュールの場合は「０」 */
iptsk.priority   = TASK_PRI_NORMAL;
iptsk.pbuff_size = 0;
iptsk.stack_size = 0x4000;
iptsk.task_opt   = 0;
iptsk.param_opt  = 0;

tskid= CreateIPTask( &iptsk );
```

(3) 割込オブジェクトの生成

```
INT_DEVICE_OBJ   intobj;

intobj.intdev = INTDEV_PIO;
intobj.evtpri = 0;
intobj.intbit = 0;        /* PIO[0]の割込みの場合 */
intobj.opt    = 0;

CreateInterruptLink( &intobj, tskid, INTEVENT_WAKEUP, 0 );
```

(4) 割込起動モジュールの登録

ダウンロードした割込起動モジュールをRegistIPTask()コマンドで登録します。

```
RegistIPTask( tskid, OutInf.pAdr[0], OutInf.pAdr[1], TASK_NO_OPTION );
```

(5) 割込起動モジュールへのパラメータ渡し

登録した割込起動モジュールにパラメータを渡す方法を説明します。初期化モジュール、実行モジュールの両方が同じ方法でパラメータの設定ができます。

(a) パラメータテーブルの初期化

InitIPParamTable()コマンドでパラメータテーブルを初期化します。

```
MODULE_PARAM_TBL   prmtbl;

InitIPParamTable( &prmtbl, 3, PIOINT_MODULE, tskid, 0 );
```

(b) パラメータの設定

割込起動モジュールのパラメータがint型やfloat型の場合、SetIntegerParam(),SetFloatParam()コマンドでパラメータテーブルにデータを設定します。

```
SetIntegerParam( &prmtbl, PARAM_1, 123 );
SetFloatParam( &prmtbl, PARAM_2, 4.567 );
```

(c) テーブル（配列）パラメータの設定

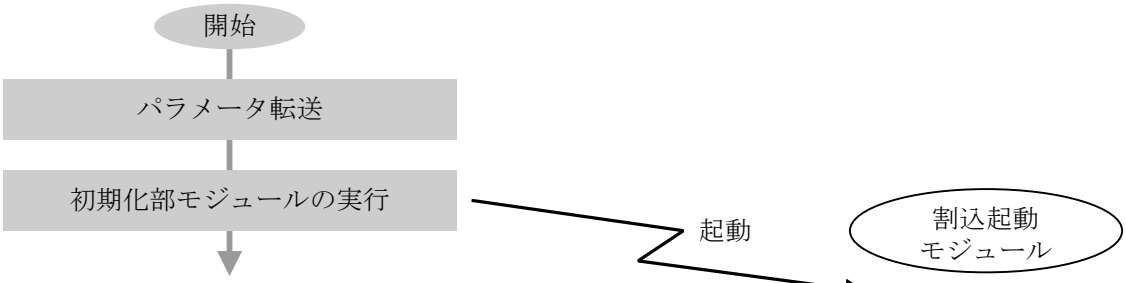
割込起動モジュールのパラメータが配列の場合、SetParamTable()コマンドでパラメータテーブルにデータを設定します。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(6) 初期化部モジュールの実行

RegistIPTask()コマンドにより、パラメータのデータを転送し、登録した割込起動モジュールの初期化部モジュールを実行します。

```
StartIPTaskwithParam( tskid, &prmtbl );
```

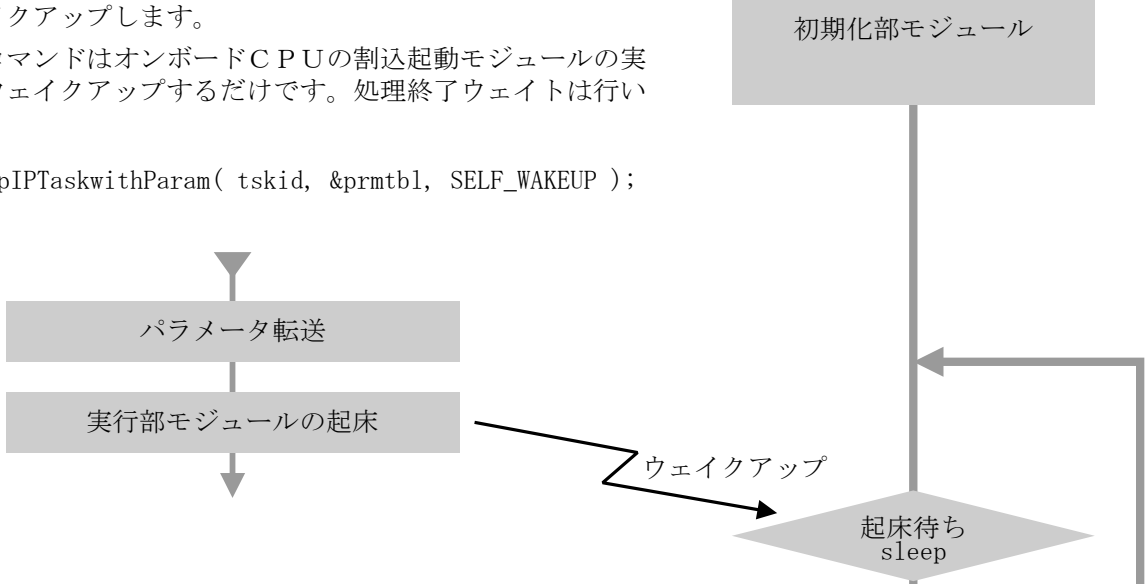


(7) 実行部モジュールのウェイクアップ

WakeupIPTaskwithParam()コマンドにより、パラメータのデータを転送し、登録した割込起動モジュールの実行部モジュールをウェイクアップします。

このコマンドはオンボードＣＰＵの割込起動モジュールの実行部をウェイクアップするだけです。処理終了ウェイトは行いません。

```
WakeupIPTaskwithParam( tskid, &prmtbl, SELF_WAKEUP );
```



(8) モジュールの終了ウェイト

WaitforIPTaskSignal()コマンドにより、ウェイクアップされたオンボードＣＰＵの割込起動モジュールからのSendSignaltoPC()コマンドによる終了シグナルをウェイトします。そして、オンボードＣＰＵでSendSignaltoPC()コマンドを実行した時点でウェイクアップしてこのコマンドが終了します。

```
WaiIPSigInf    SigInf

WaitforIPTaskSignal( PCI_SIGNAL_USER_0, 0, 0, &SigInf );
```

(9) 割込起動モジュールからのデータ取得

割込起動モジュール実行後にデータ取得する方法を説明します。

(a) 指定した配列パラメータが入出力データの場合

割込起動モジュールのパラメータが配列で割込起動モジュールにデータを渡してなおかつ割込起動モジュール実行終了後にデータを取得する場合、SetParamTable()コマンドでパラメータテーブルにデータを設定し、割込起動モジュール実行終了後、GetParamTable()コマンドでデータを取得して下さい。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
WakeupIPTaskwithParam( tskid, &prmtbl, PIO_WAKEUP );
WaitforIPTaskSignal( PCI_SIGNAL_USER_0, &prmtbl, 0, 0, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(b) 指定した配列パラメータが出力のみのデータの場合

割込起動モジュールのパラメータが配列で割込起動モジュールにデータを渡してなおかつ割込起動モジュール実行終了後にデータを取得する場合、AllocParamTable()コマンドでパラメータテーブルの領域を確保し、割込起動モジュール実行終了後、GetParamTable()コマンドでデータを取得して下さい。

```
AllocParamTable( &prmtbl, PARAM_3, sizeof(tbl) );
WakeupIPTaskwithParam( tskid, &prmtbl, PIO_WAKEUP );
WaitforIPTaskSignal( PCI_SIGNAL_USER_0, &prmtbl, 0, 0, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

第8章 ファイルシステム

8.1 概要

ＮＶＰ－２３５ＣＬはオンボードのフラッシュメモリディスク、ＳＤカード、ＵＳＢマスストレージデバイス、ＲＡＭディスクをファイルシステムとして使用することが可能です。
また、ＮＶＰ－Ａｘ２３０ＣＬはオンボードのフラッシュメモリディスク、ＲＡＭディスクのみをファイルシステムとして使用することが可能です。

8.1.1 オンボードフラッシュメモリディスクドライブ

ＮＶＰ－Ａｘ２３０はボード上に６４Ｍバイトのフラッシュメモリを実装しています。オンボードフラッシュメモリは、ＮＶＰ－Ａｘ２３０のシステムブートモジュールが含まれていますが、それ以外の部分（約６０Ｍ）は未使用の領域です。ＮＶＰ－Ａｘ２３０は、そのフラッシュメモリの未使用領域を使用してデータの保存や読み出しを行うために、簡単なフラッシュメモリディスクの機能を実装し、ファイルとしてのアクセスが可能です。なお、フラッシュメモリは、その物理的な特性により、消去、書き込みできる回数に制限があります。通常は１０万回程度ですので、常に消去、書き込みを繰り返すようなアプリケーションは注意して下さい。

8.1.2 SDカードドライブ

ＮＶＰ－Ａｘ２３５ＣＬはＳＤカードスロットを１スロット実装しており、ＳＤカードメモリを１枚使用することが可能です。使用可能なＳＤカードは、ＳＤ／ＳＤＨＣカードに対応しており、ＳＤＸＣカードには対応しておりません。
また、ＳＤカードのファイルシステムは、ＦＡＴ１６及びＦＡＴ３２のフォーマットに対応しており、Ｗｉｎｄｏｗｓとファイルの互換性がありデータ交換が可能です。ｅｘＦＡＴ及びその他フォーマットには対応しておりません。
ＮＶＰ－Ａｘ２３０ＣＬはＳＤカードドライブは搭載していません。

8.1.3 USBドライブ

ＮＶＰ－Ａｘ２３５ＣＬはＵＳＢホストインタフェースを２ポート実装しており、１ポートをＵＳＢマスストレージデバイスとして使用することが可能です。ＵＳＢマスストレージデバイスとしてＵＳＢメモリやＵＳＢハードディスクが使用可能です。ＵＳＢマスストレージデバイスのファイルシステムは、ＦＡＴ１６及びＦＡＴ３２のフォーマットに対応しており、Ｗｉｎｄｏｗｓとファイルの互換性がありデータ交換が可能です。ｅｘＦＡＴ及びその他フォーマットには対応しておりません。
ＮＶＰ－Ａｘ２３０ＣＬはＵＳＢドライブは搭載していません。

8.1.4 RAMディスクドライブ

ＮＶＰ－Ａｘ２３０は搭載しているメモリ(ＲＡＭ)をＲＡＭディスクとして使用することが可能です。ＲＡＭディスクを使用する場合は、あらかじめシェルコマンド【credsk】でＲＡＭディスクを生成して下さい。

```
credsk ram: 100000      [容量1MBのRAMディスクの生成例]
```

シェルコマンドの詳細は「シェルコマンド リファレンス」を参照して下さい。

【制限事項】

・ＳＤカード及びＵＳＢメモリは、すべてのメディアへの対応を保証するわけではありません。

・オンボードフラッシュメモリ、ＳＤカード、ＵＳＢマスストレージデバイス、ＲＡＭディスクへの書込みは、電源を入れたままの抜き差しやディスクアクセス時の電源切断及びリセットなど、すべての状況においてのデータの確実性を保証するものではありません。
また、破損データのリカバリは不可能ですのでご了承ください。

・ＮＶＰ－Ａｘ２３０は、活線挿抜に対応していません。そのため活線挿抜を行うとシステムダウンに陥る可能性がありますので注意してください。

8.2 固定ドライブとリムーバブルドライブ

オンボードフラッシュメモリディスクドライブは、着脱が不可能なため固定ドライブとしてファイルシステムに存在し、電源投入時から常にディスクドライブとして認識されます。一方、ＳＤカードやＵＳＢドライブは、着脱が可能なためリムーバブルディスクとして認識され、装着された時にマウントされ抜去されたときにマウントが解除されます。

8.3 ドライブ、パス、ファイル名

ファイルを指定する場合は、基本的に

″[ドライブ名:]¥[ディレクトリ¥]ファイル名″

で指定します。ドライブ名を省略すると定義されているカレントドライブに対してのディレクトリとファイル名になります(デフォルトでは[fmd:]がカレントドライブ)。また「ドライブ名＋ディレクトリ名＋ファイル名」のパス名の最大の長さは半角文字で５１１文字(漢字の場合は２５５文字)です。

ドライブ名は、あらかじめ決められており、下表に示します。

表8-3-1　ドライブ名

ドライブ名	デバイス名	ドライブ詳細
sys:	オンボードフラッシュメモリドライブ	システムドライブ ・オンボードシステムファイルやシステムブートファイルを格納する
fmd:	オンボードフラッシュメモリドライブ	アプリケーションドライブ ・デフォルトのカレントドライブ ・スタートアップファイルやアプリケーション実行ファイルを格納する
ata:	ＳＤカードドライブ	リムーバブルドライブ ・アプリケーション実行ファイルを格納する ・ＤＩＰスイッチ設定でカレントドライブに指定可能
usb:	ＵＳＢマスストレージドライブ	リムーバブルドライブ ・アプリケーション実行ファイルを格納する ・ＤＩＰスイッチ設定でカレントドライブに指定可能

8.4 ファイルの作成とアクセス

ファイルの作成とオープンはfmOpen関数で行います。アプリケーションは、ファイルから読取るか、ファイルに書き込むかその両方かを指定することができます。fmOpenで作成又はオープンしたファイルはfmRead/fmWrite関数でアクセスします。アクセスが終了したらfmClose関数でファイルをクローズします。

8.4.1 ファイルアクセスフロー

以下にファイルアクセスのフローを示します。

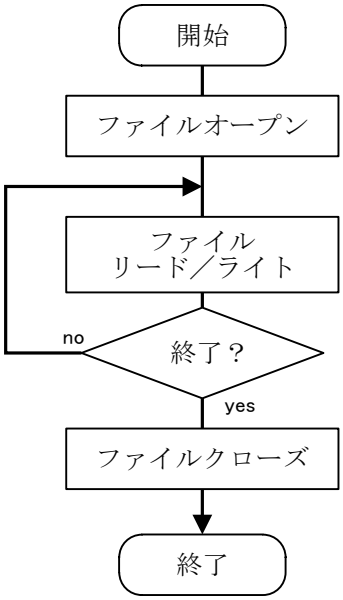


図8-4-1　ファイルアクセスのフロー

8.4.2 ファイルアクセスの例

以下にファイルアクセスのサンプルを示します。

```
char data[128];
FM_HFILE  hFile;

    // 書き込み
    hFile = fmOpen("sample.dat",FM_CREATE | FM_RDWR);
    if(hFile == 0){
       // エラー
    }
    fmWrite(hFile,data,sizeof(data));
    fmClose(hFile);

    // 読み込み
    hFile = fmOpen("sample.dat",FM_RDONLY);
    if(hFile == 0){
       // エラー
    }
    fmRead(hFile,data,sizeof(data));
    fmClose(hFile);
```

8.5 ディレクトリ操作

ディレクトリの作成はfmCreateDirectory関数、ディレクトリの削除はfmRemoveDirectory関数で行います。

8.6 ファイル検索

指定した１つのファイルが存在するかどうか検索する場合、fmFindFile関数を使用します。また、ディスク全体を巡回して存在するすべてのディレクトリ、ファイルをチェックしながら検索する場合はfmFindFirstFile,fmFindNextFile,fmFindCloseを使用し検索します。

fmFindFirstFileは、システムに対してファイルを検索することを通知し、指定されたディレクトリ内で指定された内容に一致するファイルを見つけるとファイル検索結果テーブル(FM_FIND_TBL)を初期化し、検索ハンドルを返します。

fmFindFirstFileが検索に成功したら検索ハンドルと初期化されたFM_FIND_TBL構造体でfmFindNextFileを実行しファイル検索を行います。fmFindNextFileが成功するとFM_FIND_TBL構造体が更新されます。巡回して存在するすべてのディレクトリ、ファイルを検索するとfmFindNextFileが検索終了を通知します。

ファイル検索処理が終了したら、fmFindCloseを呼出してfmFindFirstFileが返した検索ハンドルをクローズしてください。

8.6.1 ファイル巡回検索フロー

以下にファイル巡回検索のフローを示します。

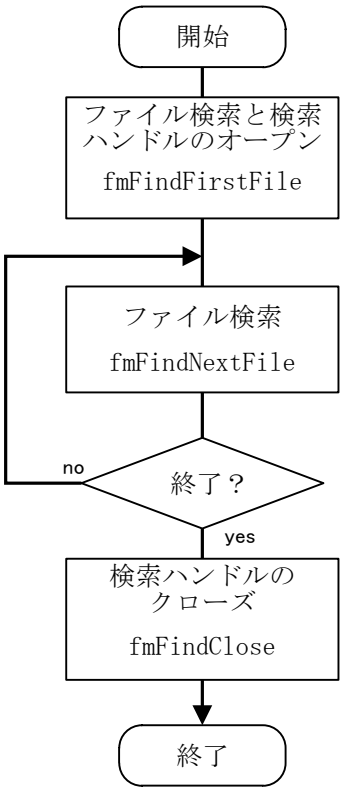


図8-6-1　ファイル巡回検索のフロー

8.6.2 ファイル巡回検索の例

```
int             ret;            /* 関数戻り値                 */
int             ErrorCode;      /* エラー情報                 */
int             i;              /* ループ用変数               */
FM_HFIND        hFind;          /* 検索ハンドル               */
FM_FIND_TBL     findTbl;        /* ファイル検索結果テーブル */
IP_SYSTEM_TIME  systime;        /* 時刻情報テーブル           */
char *pathName = "fmd:¥¥*";     /* 検索パス                   */

  /* ファイルの検索と検索ハンドルのオープン */
  memset( &findTbl, 0, sizeof( findTbl ) );
  hFind = fmFindFirstFile( pathName, &findTbl );
  if( hFind == 0 ){
    ErrorCode = fmGetError();
    printf( "ErrorCode = %d¥n",ErrorCode  );
    return;
  }

  for( i=0; ; i++ ){
    /* ファイルの検索 */
    memset( &findTbl, 0, sizeof( findTbl ) );
    ret = fmFindNextFile( hFind, &findTbl );
    if( ret == 0 ){
      /* 検索終了 */
      break;
    }

    printf( "[%d] ------------------------------¥n", i );
    printf( "name : %s¥n", findTbl.name );
    printf( "attr : %08x¥n", findTbl.attr );
    printf( "size : %d¥n", findTbl.size );

    FileTimeToIPTime( &findTbl.time, &systime );

    printf( "time : %d/%d/%d/ %d:%d:%d¥n", systime.year, systime.month, systime.day,
                        systime.hour, systime.minute, systime.second );
  }

  /* 検索ハンドルのクローズ */
  fmFindClose( hFind );
```

8.7 ファイルコピー

ファイルコピー関数には、ＰＣのディスクとボード上のディスク間でのコピーとボード上のディスク同士のコピーの２種類があります。

表8-6-1　ファイルコピー関数

関数名	機能
fmFileCopy	ＰＣのディスクとボード上のディスク間でのコピー
fmCopyFile	ボード上のディスク同士のコピー

8.8 ファイル管理ツール

ＮＶＰ－Ａｘ２３０のフラッシュメモリディスクのファイルをＰＣから操作するためのWindows上のＧＵＩツール「VPVisor2」を用意しました。VPVisor2を使用することでフラッシュメモリディスクからＰＣに、ＰＣからフラッシュメモリに簡単にファイルのコピーやその他の操作が可能です。
VPVisor2の操作については、「VPVisor2 操作マニュアル」を参照して下さい。

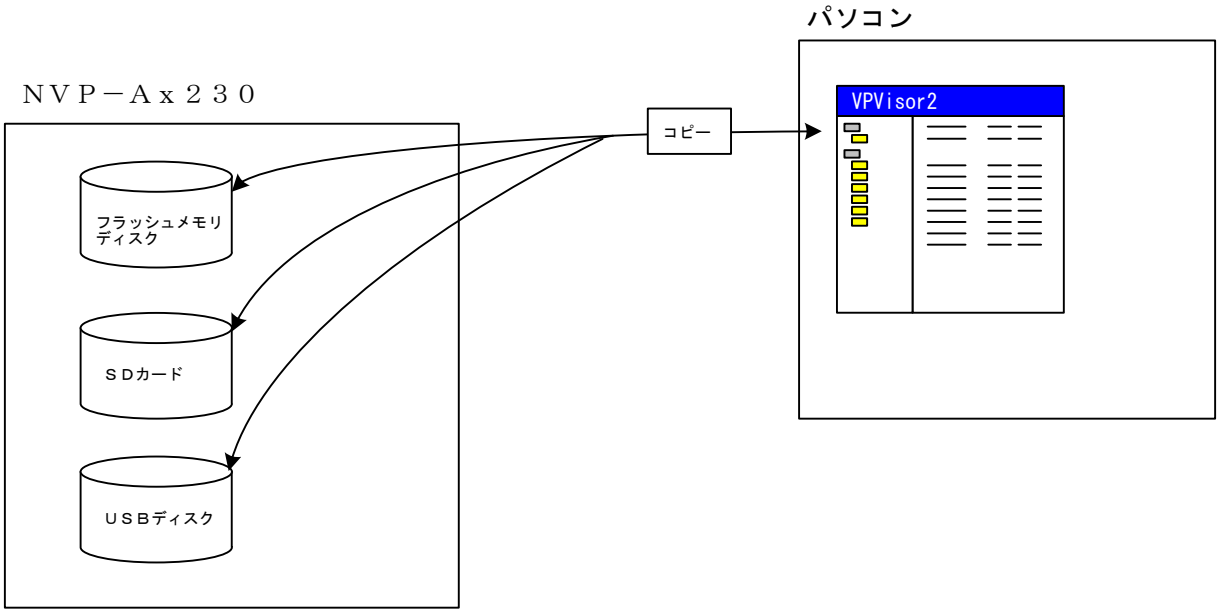


図8-8-1　ファイル管理ツール

第9章 イーサネットでの通信

9.1 概要

イーサネットの通信はＵＮＩＸのＢＳＤソケットのイメージでプログラミングが可能です。
なお、ＢＳＤソケットコマンドを使用する場合は、"BSDSOCK.H"をインクルードして下さい。

9.1.1 ソケットインタフェース

IPアドレスとポート番号を組み合わせた組のことをソケットと呼びます。ＢＳＤソケットモデルは、ネットワーク通信が２つのソケット間で発生するものとして概念化しています。通信を行うには、クライアントソケットとサーバソケットという２つのソケットが存在する必要があります。

9.1.2 通信方式

ＢＳＤソケットモデルは、クライアント／サーバー方式で通信を行います。クライアントとサーバーの２つのネットワークアプリケーションは同時に起動するのではなく、サーバー側のアプリケーションは常に使用可能な状態に置き、クライアント側のアプリケーションは必要に応じてサービスをサーバー側のアプリケーションに要求するようにします。サーバーはソケットを作成し、クライアントが識別できるように名前を付けサービスの要求を待機します。クライアントは、ソケットを作成し、サーバーソケットをアドレスで識別してから接続を開始します。いったん接続した後は、データを双方向に送信することができます。

9.2 クライアント／サーバーアプリケーション

クライアント／サーバーアプリケーションは、基本的に接続型アプリケーションか非接続型アプリケーションのどちらかになります。アプリケーションが通信に使用するプロトコルにより、アプリケーションが接続型か非接続型のどちらかに分類されます。

9.2.1 接続型アプリケーション

データをバイトストリームとして送受信するアプリケーションは、一般に接続型アプリケーションです。データを転送したり受信したりするには、まず２つのプロセス間で接続を確立しなければなりません。各端点(ソケット)の識別情報は、接続の開始時に一度だけ確認されこれ以降のデータの送受信は、その接続の２点の間で行うものとみなされます。そのため、各アプリケーションは、データ転送のたびに識別情報を確認する必要がありません。接続型アプリケーションは通常ＴＣＰ(Transmission Control Protocol)で送受信を行うアプリケーションを指します。

まずサーバーアプリケーションが起動し、socket()を使ってソケットを作成し、bind()でソケットに名前を付けます。ソケットに名前を付けることにより、サーバーのアドレス、接続先のポート、使用するプロトコルを特定することができます。ソケットに名前を付けたら、サーバーはlisten()を使ってクライアントからの要求を待機し、accept()でそれを受け入れます。

これでクライアントアプリケーションを実行する準備が整ったことになります。クライアントアプリケーションはsocket()を使ってソケットを作成し、connect()を使ってサーバーにプラグインし、サービス要求を送ります。この時点で両アプリケーションの識別情報が確認されて仮想回線が確立されます。両方のアプリケーションでは、少なくとも１つのソケットをお互いの通信専用に使用する必要があります。この後は、send()とrecv()を使ってどちらの方向にもデータを送受信することができます。アプリケーション間の通信が終わったら両アプリケーションでshutdown()とclosesocket()を使ってそれぞれのソケットを閉じます。

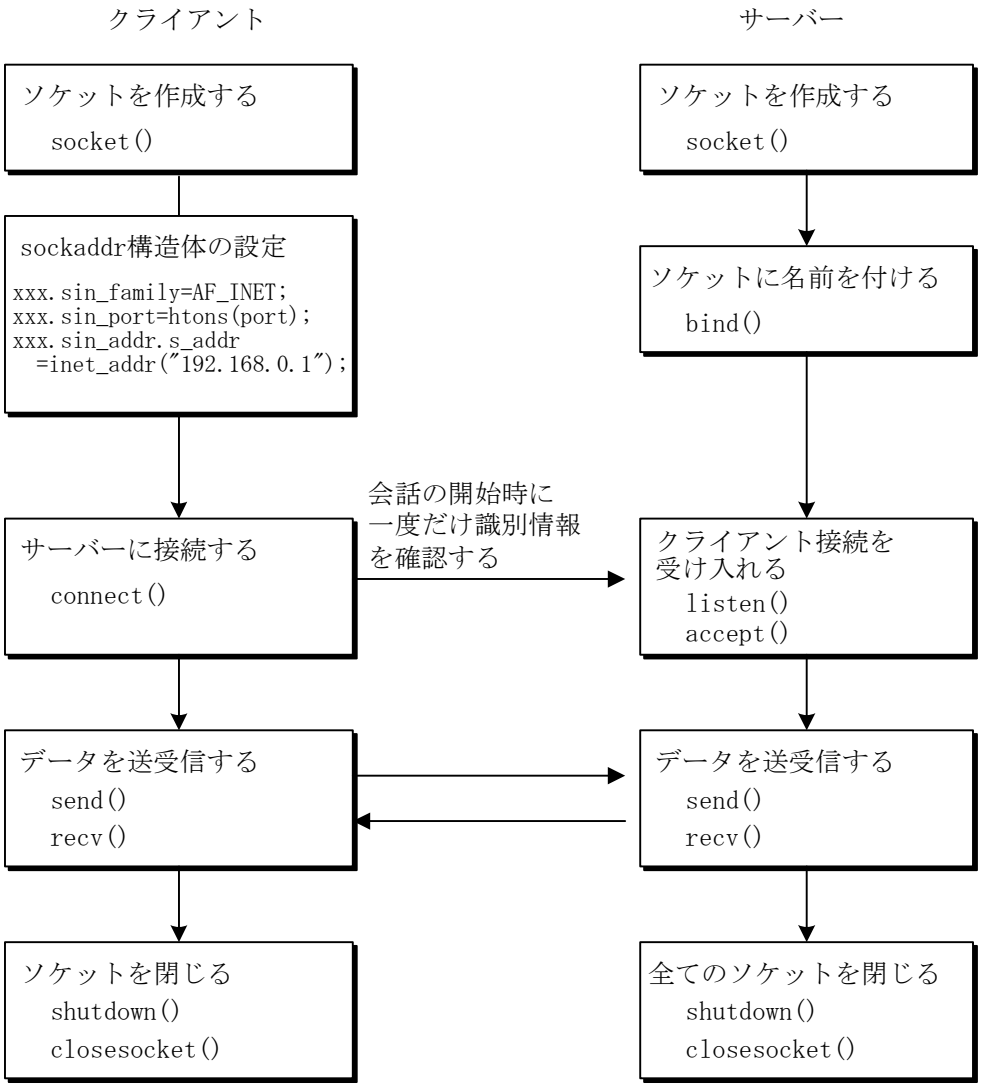


図9-2-1 接続型アプリケーション

サンプル１、２にＴＣＰ接続でのサーバー側とクライアント側のプログラム例を示します。

サンプル１ ： ＴＣＰでのサーバー側のプログラム例

```
#include <string.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

#include "bsdsock.h"

typedef int SOCKET;
#define INADDR_NONE      0
#define SOCKET_ERROR     -1

#define   BUFSIZE         1000
#define   DEFAULT_PORT    5320

void TCPServer()
{
struct sockaddr_in client, server;
int             port, len, i, n, j, k;
unsigned char   buf[BUFSIZE];
SOCKET          wait_sock,sock;

   port = DEFAULT_PORT;

   if((wait_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){
      printf("socket error !!¥n");
      return;
   }

   // 受付クライアントの設定
    memset((char *)&server, 0x00, sizeof(server));
   server.sin_family      = AF_INET;
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   server.sin_port        = htons((unsigned short)port);

   // バインド
   if(bind(wait_sock, (struct sockaddr *)&server, sizeof(server)) < 0){
      printf("bind error !!¥n");
      closesocket(wait_sock);
      return;
   }

   // 受付けキューの設定
   if(listen(wait_sock, 1) < 0){
      printf("listen error !!¥n");
      closesocket(wait_sock);
      return;
   }

   // TCPコネクションの受付け
   len = sizeof(client);
   if((sock = accept(wait_sock, (struct sockaddr *)&client, (socklen_t*)&len)) < 0){
      printf("accept error !!¥n");
      closesocket(wait_sock);
      return;
   }

   // TCPを利用したデータの送受信
   while((n = recv(sock, buf, BUFSIZE-1, 0)) > 0){
      for(i = 0;i < n;) {
         printf("Receive [ ");
         for (j = 0 ; i < n && j < 10 ; i++ ,j++) {
               printf("H'%02X ", buf[i]);
         }
         printf("]¥n");
      }

      k= 0;
      for(i = 0;i < n;) {
         printf("Send    [ ");
         for (j = 0 ; i < n && j < 10 ; i++ ,j++) {
            printf("H'%02X ", buf[i]);
         }
         printf("]¥n");
      }

      send(sock, buf, n, 0);
   }

   shutdown(sock,SHUT_RDWR);
   closesocket(sock);
   closesocket(wait_sock);
}
```

サンプル２ ： ＴＣＰでのクライアント側のプログラム例

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

#include "bsdsock.h"

typedef int SOCKET;
#define INADDR_NONE      0
#define SOCKET_ERROR     -1

#define  DEFAULT_PORT    5320

void TCPClient()
{
struct sockaddr_in svr;
int            ret;
char server[] = "192.168.0.3";
unsigned long   ipaddr;
unsigned short  port= DEFAULT_PORT;
SOCKET sock;
char op_msg[] = "Connect TCPServer OK !!¥r¥nYou can communicate with TCPServer ...¥r¥n";

   // サーバーのIPアドレス
   ipaddr= inet_addr(server);
   if(ipaddr == INADDR_NONE){
      printf("inet_addr error !!¥n");
      return;
   }

   // ソケットの作成
   sock= socket(AF_INET,SOCK_STREAM,0);
   if(sock == SOCKET_ERROR){
      printf("socket error !!¥n");
      return;
   }

   // サーバーの設定
   memset((char*)&svr, 0x00, sizeof(svr));
   svr.sin_family      = AF_INET;
   svr.sin_port        = htons(port);
   svr.sin_addr.s_addr = ipaddr;

   // サーバーに接続
   ret= connect(sock,(struct sockaddr*)&svr,sizeof(svr));
   if(ret<0){
      printf("connect error !!¥n");
      closesocket(sock);
      return;
   }

   ret= send(sock, op_msg, strlen(op_msg), 0);

   closesocket(sock);

}
```

9.2.2 非接続型アプリケーション

データをデータグラムの形で送受信するアプリケーションは一般に非接続型アプリケーションです。アプリケーションは、データ送信のたびに識別情報を確認します。非接続型アプリケーションは通常ＵＤＰ(User Datagram Protocol)で送受信を行うアプリケーションを指します。

まずサーバーアプリケーションが起動し、socket()を使ってソケットを作成し、bind()でソケットに名前を付け、クライアントから要求が送られてくるのを待機します。この種のサーバーは一般にlisten()やaccept()を使わず、recvfrom()を呼び出すことによってクライアント要求を待機します。recvfrom()は接続型サーバーで使うrecv()の持つ機能に加えて、サーバーにデータの送信元を知らせる役割も果たします。サーバーはこの情報を元にsendto()を使ってクライアントに応答を返します。アプリケーション間の通信が終わったら両アプリケーションでclosesocket()を使ってそれぞれのソケットを閉じます。

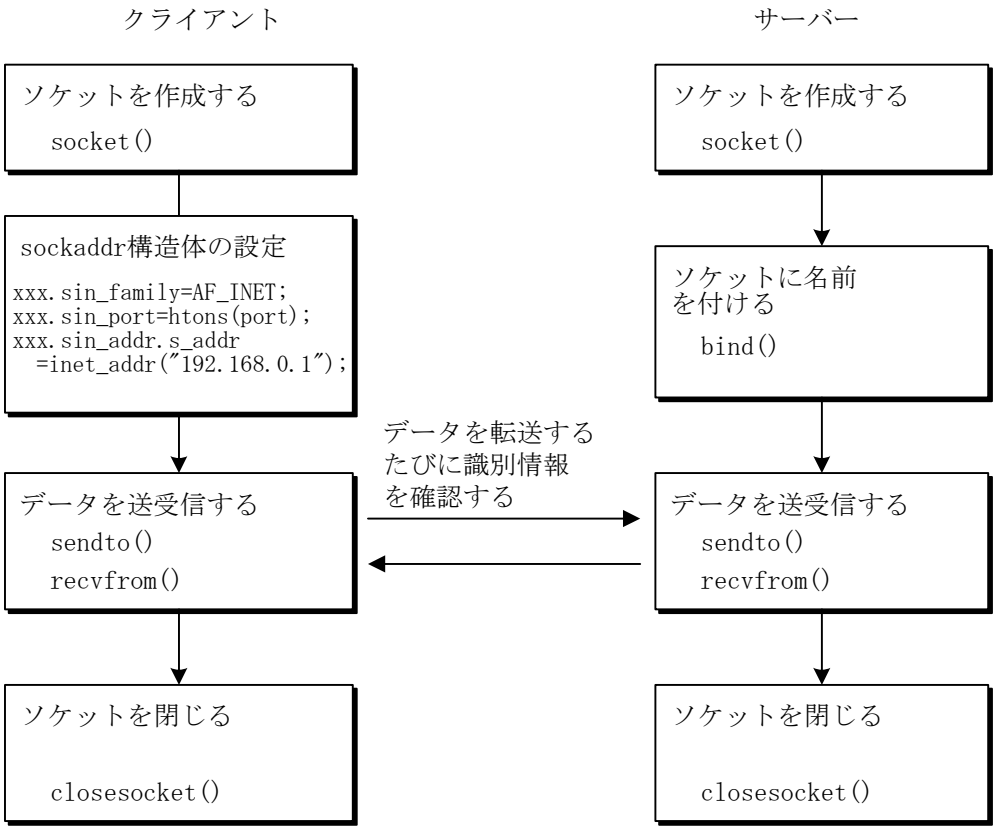


図9-2-2 非接続型アプリケーション

サンプル３、４にＵＤＰ接続でのサーバー側とクライアント側のプログラム例を示します。

サンプル３ ： ＵＤＰでのサーバー側のプログラム例

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

#include "bsdsock.h"

typedef int SOCKET;
#define INADDR_NONE      0
#define SOCKET_ERROR     -1

#define BUFSIZE          1000
#define DEFAULT_PORT     5320
#define END_CODE         0

void UDPServer()
{
struct sockaddr_in client, server;
int             port, len, i, n, j, k;
unsigned char   buf[BUFSIZE];
SOCKET          sock;

   port = DEFAULT_PORT;

   if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
      printf("socket error !!¥n");
      return;
   }

   // 受付クライアントの設定
   memset((char *)&server, 0x00, sizeof(server));
   server.sin_family      = AF_INET;
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   server.sin_port        = htons((unsigned short)port);

   // バインド
   if(bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0){
      printf("bind error !!¥n");
      closesocket(sock);
      return;
   }

   // UDPを利用したデータの送受信
   while(1){
      len= sizeof(client);
      n = recvfrom(sock, buf, BUFSIZE-1, 0, (struct sockaddr *)&client, (socklen_t*)&len);
      if(n <= 0)
         break;
      for(i = 0;i < n;) {
         printf("Receive [ ");
         for (j = 0 ; i < n && j < 10 ; i++ ,j++) {
            if(buf[i] == END_CODE)
               goto CLOSE_SOKET;
            printf("H'%02X ", buf[i]);
         }
         printf("]¥n");
      }

      k= 0;
      for(i = 0;i < n;) {
         printf("Send    [ ");
         for (j = 0 ; i < n && j < 10 ; i++ ,j++) {
            printf("H'%02X ", buf[i]);
         }
         printf("]¥n");
      }

      sendto(sock, buf, n, 0, (struct sockaddr *)&client, sizeof(client));
   }

CLOSE_SOKET:

   closesocket(sock);

}
```

サンプル４ ： ＵＤＰでのクライアント側のプログラム例

```
#include <string.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

#include "bsdsock.h"

typedef int SOCKET;
#define INADDR_NONE      0
#define SOCKET_ERROR     -1

#define DEFAULT_PORT     5320

void UDPClient()
{
struct sockaddr_in svr;
int            ret;
char server[] = "192.168.0.3";
unsigned long   ipaddr;
unsigned short  port= DEFAULT_PORT;
SOCKET sock;
char op_msg[] = "Connect UDPServer OK !!¥r¥nYou can communicate with UDPServer ...¥r¥n";
char buf[4]= {0};

   // サーバーのIPアドレス
   ipaddr= inet_addr(server);
   if(ipaddr == INADDR_NONE){
      printf("inet_addr error !!¥n");
      return;
   }

   // ソケットの作成
   sock= socket(AF_INET,SOCK_DGRAM,0);
   if(sock == SOCKET_ERROR){
      printf("socket error !!¥n");
      return;
   }

   // サーバーの設定
   memset((char*)&svr, 0x00, sizeof(svr));
   svr.sin_family      = AF_INET;
   svr.sin_port        = htons(port);
   svr.sin_addr.s_addr = ipaddr;

   sendto(sock, op_msg, strlen(op_msg), 0, (struct sockaddr*)&svr, sizeof(svr));
   sendto(sock, buf, 1, 0, (struct sockaddr*)&svr, sizeof(svr));

   closesocket(sock);

}
```

9.3 イーサネット通信の環境設定

イーサネット通信を使用する前にあらかじめユーザーが使用する環境にあわせて環境設定値を設定する必要があります。システム起動時に「BOOT.INI」ファイルに記述されているイーサネット通信の環境設定値により、通信に必要なメモリの確保やその他の環境変数を設定しイーサネット通信のプロトコルスタックを起動します。以下の環境設定の詳細は「環境設定マニュアル」を参照ください。

表9-3-1　イーサネット通信環境設定

セクション名	内容
IPADR	ＩＰアドレスを指定します。 デフォルトは、”１９２．１６８．０．２０５”が設定されます。
IPMSK	サブネットマスクを指定します。 デフォルトは、”２５５．２５５．２５５．０”が設定されます。
GWADR	デフォルトゲートウェイを指定します。 デフォルトは、”１９２．１６８．０．１”が設定されます。
PORTSTR	自動割り当てにしようするポート番号の先頭を指定します。使用可能なポート番号は、１～６５５３５ です。デフォルトは、”１５０００”が設定されます。
PORTCNT	自動割り当てにしようするポート番号の数を指定します。使用可能なポート番号は、１～６５５３５ です。デフォルトは、”１６”が設定されます。
TCPACT	ＴＣＰで接続するソケットの総数を指定します。設定範囲は、４ ～ ３２です。但し、(TCPACT+TCPCON) <= 32に設定して下さい。デフォルトは、”８”が設定されます。
TCPLST	ＴＣＰで接続待ちに使用するソケットの総数を指定します。設定範囲は、 ４ ～ ３２ です。デフォルトは、”８”が設定されます。
UDPCNT	ＵＤＰで接続するソケットの総数を指定します。設定範囲は、０ ～ ３２です。デフォルトは、”８”が設定されます。
TCPCON	ＴＣＰで同時に受け入れ可能なコネクション要求の総数を指定します。設定範囲は、４ ～ ３２ です。但し、(TCPACT+TCPCON) <= 32に設定して下さい。デフォルトは、”８”が設定されます。
RCVWB	ＴＣＰの受信バッファのデフォルト長を指定します。１つあたりのＴＣＰソケットで使用する受信バッファの最大長です。設定範囲は、２９２０～ １７５２０です。デフォルトは、”８７６０”が設定されます。
SNDWB	ＴＣＰの送信バッファのデフォルト長を指定します。１つあたりのＴＣＰソケットで使用する送信バッファの最大長です。設定範囲は、２９２０～ １７５２０です。デフォルトは、”８７６０”が設定されます。 なお、各ＴＣＰソケットの受信バッファ、送信バッファの長さはsetsockopt()により指定することが可能です。このとき、[RCVWB]セクション、[SNDWB]セクションで指定した長さを超える指定はできません。
UDPDGM	ＵＤＰの最大受信データグラム長を指定します。ＵＤＰソケットが受信するデータグラムの最大長（通常は５７６から１４７２）を指定します。設定範囲は、０ ～ １４７２ です。デフォルトは、”１４７２”が設定されます。
UDPQUE	ＵＤＰの受信バッファキューのデフォルト数を指定します。１つあたりのＵＤＰソケットで用意するデータグラム受信キューのデフォルトの数（通常はrecvまたはrecvfromによって取得する前に受信できるデータグラムの数＋１）を指定します。設定範囲は、０ ～ ２５５ です。デフォルトは、”９”が設定されます。

注）画像処理コマンドサーバのポート番号がデフォルトで”３００００”に設定されています。
ユーザーでこのポート番号を使用する場合は「PORT」を変更して下さい。

第10章　USB－HIDデバイスの制御

10.1 概要

ＮＶＰ－Ａｘ２３５ＣＬはボード上にＵＳＢ－ＨＩＤ（Ｈｕｍａｎ　Ｉｎｔｅｒｆａｃｅ　Ｄｅｖｉｃｅ）ドライバを実装しています。ＨＩＤドライバでは、キーボードとマウスが制御可能です。キーボードとマウスのＨＩＤを起動／停止、デバイスステータスを取得するためのＵＳＢ－ＨＩＤ制御コマンドによりＨＩＤの制御が可能です。

ＮＶＰ－Ａｘ２３０ＣＬはＵＳＢ－ＨＩＤドライバを実装していません。

【制限事項】

・ＵＳＢ－ＨＩＤを接続出来る最大数は、キーボード、マウスとも１個です。

・ＮＶＰ－Ａｘ２３０は、活線挿抜に対応しておりません。そのため、ＮＶＰ－Ａｘ２３０で活線挿抜を行うと最悪システムダウンに陥る可能性がありますので注意して下さい。

10.2 USB－HIDデバイスの制御方法

10.2.1 制御の開始、終了

マウスやキーボードといったＵＳＢ－ＨＩＤデバイス制御を行う場合、USB_HID_StartでＵＳＢ－ＨＩＤデバイス制御を開始させてください。また、終了する場合USB_HID_Terminateを使用し終了してください。

USB_HID_Startで制御を開始するとキーボードやマウスなどが接続されている場合は、直ちにＵＳＢ通信を開始し、ＵＳＢデバイスとのコンフィグレーションを行います。また、USB_HID_Startでの制御開始時、キーボードやマウスなどが接続されていない場合は、ＵＳＢ－ＨＩＤデバイスが挿入されるまで待機します。

10.2.2 挿抜とコールバックルーチン

ＵＳＢデバイスのコンフィグレーションが終了すると、USB_HID_Startで指定したコールバックルーチンがコールされます。

コールバックルーチンでは、ＵＳＢ－ＨＩＤデバイスが挿入された場合、キーボードやマウスなどのステータスを取得するタスクの生成を行います。また、ＵＳＢ－ＨＩＤデバイスが抜去された場合、キーボードやマウスなどのステータスを取得するタスクの終了処理を行います。

10.2.3 デバイスステータスの取得

コールバックルーチンで生成したタスクで、マウスやキーボードといったＵＳＢ－ＨＩＤデバイスのステータス取得をUSB_HID_GetStatusを用いて行います。

USB_HID_GetStatusでＵＳＢ－ＨＩＤデバイスのステータス取得を行う場合、ポーリング処理にてステータス取得を繰り返して実行します。ポーリング周期時間はユーザ任意で設定してください。

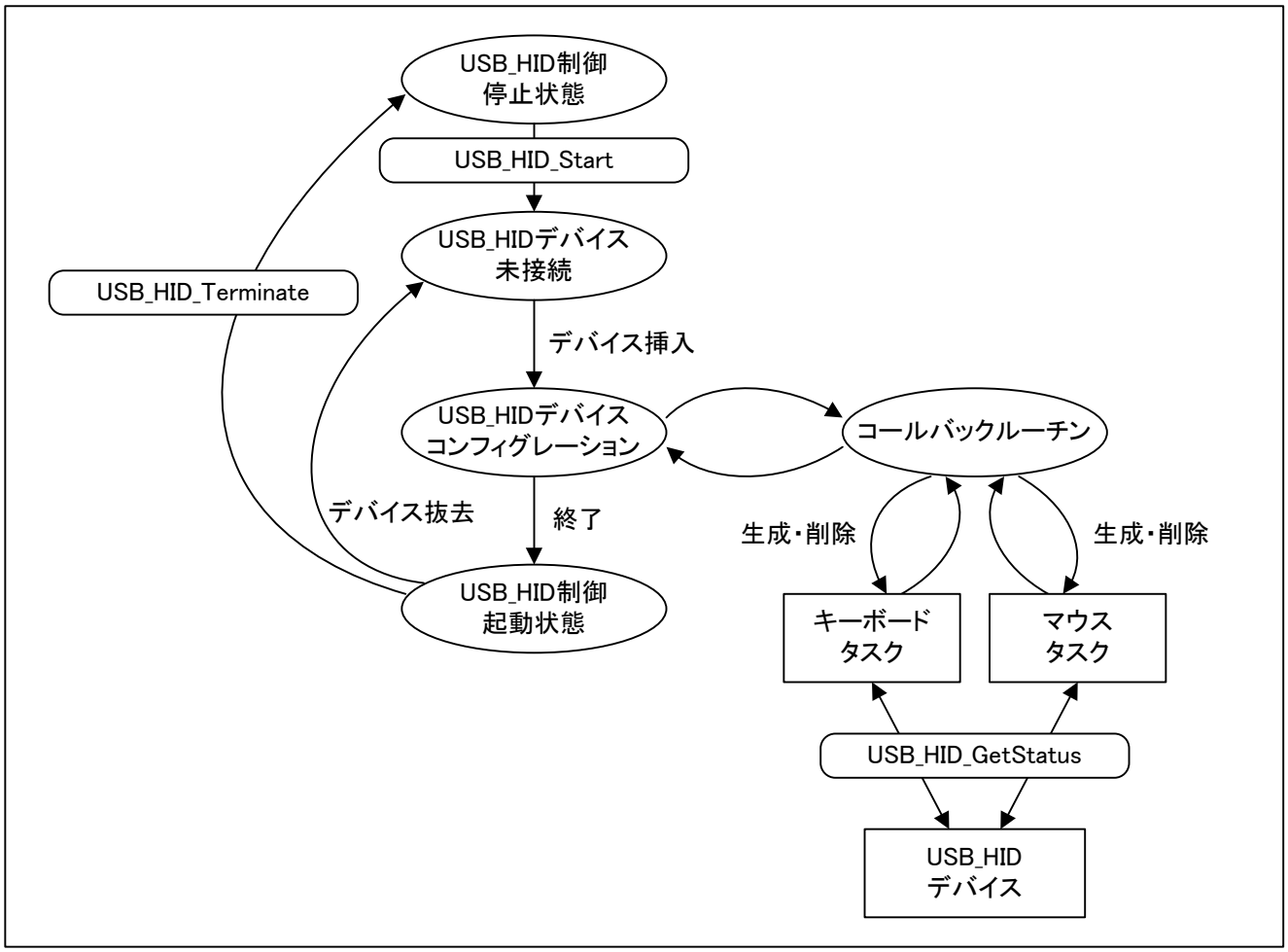


図10-2-1　ＵＳＢ－ＨＩＤデバイス制御の状態遷移

10.2.4 USB－HIDデバイス制御の例

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

#include "itron.h"
#include "kernel.h"

#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

#include "usbapi.h"

int HID_CallBack( int DeviceState, USB_HID_INFO *pUsbHidInfo );
ER_ID TASK_CREATE( char *pName, PRI pri, ATR atr, SIZE stksz, FP task, VP_INT exinf );

static int HID_KBDTaskTerminate;
static int HID_MSETaskTerminate;

#define MALLOC          malloc
#define FREE            free
#define MSEC_WAIT       dly_tsk
#define DBGPRINTF(x)    printf x

void main()
{
int sel;
int rtcd;

    while( 1 ){
        printf("*****************************¥n");
        printf("  0 : Exit¥n");
        printf("  1 : USB_HID_Start¥n");
        printf("  2 : USB_HID_Terminate¥n");
        printf("*****************************¥n");
        printf(">");
        scanf("%d",&sel);

        switch( sel ){
        case 0:
            return;
        case 1:
            printf("USB_HID_TestTask Start...¥n");
            rtcd = USB_HID_Start( (USB_HID_CALLBACK)HID_CallBack );
            if( rtcd != 0 ){
                printf("USB_HID_Start Error(%d)¥n",rtcd);
            }
            break;
        case 2:
            printf("USB_HID_Terminate¥n");
            HID_KBDTaskTerminate = 1;
            HID_MSETaskTerminate = 1;
            rtcd = USB_HID_Terminate();
            if( rtcd != 0 ){
                printf("USB_HID_Terminate Error(%d)¥n",rtcd);
            }
            break;
        }
    }
}

/*********************************************************************/
/*  name     = KeyboardTask                                          */
/*  function = Keyboardデバイスデータ取得処理                        */
/*  input    = usbhidPIPE *pUsbHidInfo                               */
/*  output   = none                                                  */
/*********************************************************************/
void KeyboardTask( USB_HID_INFO *pUsbHidInfo )
{
int              rtcd;
USB_HID_HPIPE    pPipe;
unsigned char    bBuff[8];
unsigned long    len;
signed long      timeout;

    pPipe   = pUsbHidInfo->hPipe;
    len     = pUsbHidInfo->DataSize;
    timeout = 500;

    memset( bBuff, 0, sizeof( len ) );

    while( 1 ){
        MSEC_WAIT( 500 );
        while( 1 ){
            if( HID_KBDTaskTerminate ){
                exd_tsk();
            }
            rtcd = USB_HID_GetStatus( pPipe, bBuff, len, timeout );
            if( rtcd >= 0 ){
                DBGPRINTF(("ukbdTest Get Data %x ¥n", bBuff[2]));
            } else if( rtcd == -18 ){
                DBGPRINTF(("ukbdTest Get Data (Time Out)¥n"));
                break;
            } else {
                DBGPRINTF(("ukbdTest Get Data (Error) ¥n"));
                break;
            }
            MSEC_WAIT( pUsbHidInfo->Interval );
        }
    }

}
```

```
/*************************************************************************/
/*  name     = MoseTask                                                  */
/*  function = Mouseデバイスデータ取得処理                               */
/*  input    = usbhidPIPE *pUsbHidInfo                                   */
/*  output   = none                                                      */
/*************************************************************************/
void MoseTask( USB_HID_INFO *pUsbHidInfo )
{
int             rtcd;
USB_HID_HPIPE   pPipe;
char            bBuff[8];
unsigned long   len;
signed long     timeout;

    pPipe   = pUsbHidInfo->hPipe;
    len     = pUsbHidInfo->DataSize;
    timeout = 500;

    memset( bBuff, 0, sizeof( len ) );

    while( 1 ){
        MSEC_WAIT( 500 );
        while( 1 ){
            if( HID_MSETaskTerminate ){
                exd_tsk();
            }
            rtcd = USB_HID_GetStatus( pPipe, (unsigned char*)bBuff, len, timeout );
            if( rtcd >= 0 ){
                DBGPRINTF(("umseTest Get Data x=%d, y=%d, z=%d, button=%x ¥n", bBuff[1], bBuff[2], bBuff[3], bBuff[0] ));
            } else if( rtcd == -18 ){
                DBGPRINTF(("umseTest Get Data (Time Out)¥n"));
                break;
            } else {
                DBGPRINTF(("umseTest Get Data (Error) ¥n"));
                break;
            }
            MSEC_WAIT( pUsbHidInfo->Interval );
        }
    }

}

/*************************************************************************/
/*  name     = CallBack                                                  */
/*  function = Call Back Routine                                         */
/*  input    = usbhidPIPE *pUsbHidInfo                                   */
/*  output   = int ReternCode : 正常終了 : OK(0)                         */
/*                              異常終了 : ERROR(-1)                     */
/*************************************************************************/
signed short kbdTaskId = 0;
signed short mseTaskId = 0;
USB_HID_INFO KbdInfo;
USB_HID_INFO MseInfo;

int HID_CallBack( int DeviceState, USB_HID_INFO *pUsbHidInfo )
{
    if( DeviceState == USB_HID_DEVICE_INSERT ){
    /* デバイス挿入時処理 */
        if( pUsbHidInfo->Protocol == USB_HID_DEVICE_KEYBOARD ){
        /* 挿入デバイスはキーボード */
            memcpy( &KbdInfo, pUsbHidInfo, sizeof(USB_HID_INFO) );
            HID_KBDTaskTerminate = 0;
            kbdTaskId = TASK_CREATE( "ukbdeTask", 33, TA_HLNG, 0x00001000, (FP)KeyboardTask, (int)&KbdInfo );
        } else if( pUsbHidInfo->Protocol == USB_HID_DEVICE_MOUSE ){
        /* 挿入デバイスはマウス */
            memcpy( &MseInfo, pUsbHidInfo, sizeof(USB_HID_INFO) );
            HID_MSETaskTerminate = 0;
            mseTaskId = TASK_CREATE( "umseTask", 33, TA_HLNG, 0x00001000, (FP)MoseTask, (int)&MseInfo );
        } else {
            return( -1 );
        }
    } else if( DeviceState == USB_HID_DEVICE_REMOVE ){
    /* デバイス抜去時処理 */
        if( pUsbHidInfo->Protocol == USB_HID_DEVICE_KEYBOARD ){
        /* 抜去デバイスはキーボード */
            HID_KBDTaskTerminate = 1;
        } else if( pUsbHidInfo->Protocol == USB_HID_DEVICE_MOUSE ){
        /* 抜去デバイスはマウス */
            HID_MSETaskTerminate = 1;
        } else {
            return( -1 );
        }
    } else {
        return( -1 );
    }
    return( 0 );
}


ER_ID TASK_CREATE( char *pName, PRI pri, ATR atr, SIZE stksz, FP task, VP_INT exinf )
{
T_CTSK ctsk;

    ctsk.tskatr = atr | TA_NAME | TA_PATH | TA_ACT;
    ctsk.itskpri = pri;
    ctsk.stksz = stksz;
    ctsk.task = task;
    ctsk.exinf = exinf;
    ctsk.stk = NULL;

    ctsk.name = pName;
    ctsk.path = "¥¥¥¥con2";

    return acre_tsk( &ctsk );
}
```

付録A コンパイラの設定

1. Visual Studio でのコンパイルとリンク

Ｗｉｎｄｏｗｓアプリケーションを作成する場合、本ＳＤＫで用意しているリモートコマンドのインクルードファイルパスの設定とリモートコマンドのＤＬＬインポートライブラリのリンク設定が必要です。

リモートコマンドＡＰＩは、シングルボード構成やマルチボード構成でシステムに対応するため、VisualC/C++で次の２種類のインタフェースを用意しています。

(1)ボードを識別するためのボード識別子（デバイスＩＤ）無しのＡＰＩ

(2)ボードを識別するためのボード識別子（デバイスＩＤ）付きのＡＰＩ

シングルボード構成のシステムには、ボードを識別するためのデバイスＩＤ無しのＡＰＩを使用します。リモートコマンドＡＰＩのベーシックな形式で、オンボードシステムのモジュールは、常にこの形式のＡＰＩで動作します。ボードを識別するためのデバイスＩＤ付きのＡＰＩは、マルチボード構成システムで常に複数のボードを切換えながら画像処理を行う場合に使用します。

VisualC でプログラムをコンパイル、リンクする場合、下記の要領でインクルードファイルのディレクトリパスの設定とライブラリのプロジェクトへの追加を行って下さい。なお、以下の”プロジェクト名”とは、ユーザが開発するプロジェクトの名称です。

1.1 インクルードファイルのディレクトリパスの設定

Ｗｉｎｄｏｗｓアプリケーションプログラムをコンパイルする場合、Visual C++でインクルードファイルのディレクトリパスを設定します。

［プロジェクト］メニューから［”プロジェクト名”のプロパティ］を選択します。［構成プロパティ］から［Ｃ／Ｃ＋＋］を選択後、[全般］を選択すると”追加のインクルードディレクトリ”という項目が表示されます。ここに、インストールしたリモートコマンドライブラリのインクルードファイルディレクトリを入力します。インストール時にディレクトリの変更がなければ

Ｃ：￥ＶＰ２３０ＳＤＫ￥ＳＤＫ￥ＶＣ￥ＩＮＣ

と入力します。

1.2 プリプロセッサの設定

ボードを識別するためのデバイスＩＤ付きのＡＰＩを使用する場合、上記のインクルードファイルパス以外に、VisualC上でプリプロセッサを指定する必要があります。

［プロジェクト］メニューから［”プロジェクト名”のプロパティ］を選択します。［構成プロパティ］から［Ｃ／Ｃ＋＋］を選択後、［プリプロセッサ］を選択すると”プリプロセッサの定義”という項目が表示されます。ここに、

ＭＵＬＴＩ＿ＢＯＡＲＤ＿ＣＯＮＦＩＧ

定義を入力します。デバイスＩＤ無しのＡＰＩを使用する場合は、追加しないで下さい。

1.3 ライブラリのプロジェクトへの追加

リモートコマンドのＤＬＬインポートライブラリをリンクする場合、VisualC上でライブラリをプロジェクトに追加することでビルド時にリンクされます。
［プロジェクト］メニューから［”プロジェクト名”のプロパティ］を選択します。［構成プロパティ］から［リンカ］を選択後、［全般］を選択すると、”追加のライブラリディレクトリ”という項目が表示されます。ここに、リモートコマンドのＤＬＬインポートライブラリをインストールしたディレクトリを入力します。インストール時にディレクトリの変更がなければ

Ｃ：￥ＶＰ２３０ＳＤＫ￥ＳＤＫ￥ＶＣ￥ＬＩＢ

と入力します。
また、［プロジェクト］メニューから［”プロジェクト名”のプロパティ］を選択し、［構成プロパティ］から［リンカ］を選択後、［入力］を選択すると、”追加の依存ファイル”という項目が表示されます。ここに、以下のリモートコマンドのＤＬＬインポートライブラリを入力します。デバイスＩＤ無しのＡＰＩを使用する場合 ”Ａｘ２００ＳＧＬ.ＬＩＢ”、 デバイスＩＤ付きのＡＰＩを使用する場合 ”Ａｘ２００ＭＵＬ.ＬＩＢ”をそれぞれ指定してください。

表A-1-1 VisualCのプロパティ設定

設定内容	デバイスID無し	デバイスID有り
インクルードファイルのパス	[インストールフォルダ]¥SDK¥VC¥INC 設定例： C:¥VP230SDK¥SDK¥VC¥INC	[インストールフォルダ]¥SDK¥VC¥INC 設定例： C:¥VP230SDK¥SDK¥VC¥INC
プリプロセッサ定義	設定しない	MULTI_BOARD_CONFIG
ライブラリファイルのパス	[インストールフォルダ]¥SDK¥VC¥LIB 設定例： C:¥VP230SDK¥SDK¥VC¥LIB	[インストールフォルダ]¥SDK¥VC¥LIB 設定例： C:¥VP230SDK¥SDK¥VC¥LIB
ＤＬＬインポートライブラリ	Ax200SGL.LIB	Ax200MUL.LIB

2. Hew-SHCのコンパイルとリンク

オンボードＣＰＵで動作するダウンロードモジュールアプリケーションを作成する場合、SuperH RISC engine C/C++ コンパイラパッケージ Ver9.xを使用します。本ＳＤＫで用意しているオンボードアプリケーション用のコマンドライブラリのインクルードファイルパスの設定とライブラリのリンク設定が必要です。
High-performance Embedded Workshop(Hew)でプログラムをコンパイル、リンクする場合、以下の要領でインクルードファイルのディレクトリパスの設定とライブラリのプロジェクトへの追加を行って下さい。

2.1 インクルードファイルのディレクトリパスの設定

オンボードアプリケーションプログラムをコンパイルする場合、Ｈｅｗ上でインクルードファイルのディレクトリパスを設定します。

［ビルド］メニューから［SuperH RISC engine Standard Toolchain］を選択しダイアログを表示します。[コンパイラ]タブを選択し、カテゴリのコンボボックスで[ソース]、オプション項目のコンボボックスで［インクルードファイルディレクトリ］を選択後、[追加］ボタンをクリックすると[インクルードファイルディレクトリ追加]ダイアログが表示されます。ここにインストールしたオンボードＣＰＵライブラリのインクルードファイルディレクトリを入力します。インストール時にディレクトリの変更がなければ

Ｃ：￥ＶＰ２３０ＳＤＫ￥ＳＤＫ￥ＳＨ￥ＩＮＣ

と入力します。

2.2 プリプロセッサの設定

デバイスＩＤ付きのリモートコマンドで作成された画像処理アプリケーションのソースをそのままオンボードアプリケーションとして構築する場合は、プリプロセッサで以下の定義が必要です。

［ビルド］メニューから［SuperH RISC engine Standard Toolchain］を選択しダイアログを表示します。[コンパイラ]タブを選択し、カテゴリのコンボボックスで[ソース]、オプション項目のコンボボックスで［マクロ定義］を選択後、[追加］ボタンをクリックすると[マクロ定義追加]ダイアログが表示されます。ここに、

ＭＵＬＴＩ＿ＢＯＡＲＤ＿ＣＯＮＦＩＧ

定義を入力します。[MULTI_BOARD_CONFIG]を定義することで、デバイスＩＤを削除するマクロが有効になります。デバイスＩＤ無しのＡＰＩの場合は、追加しないで下さい。

2.3 ライブラリのプロジェクトへの追加

オンボードアプリケーションを構築する場合、Ｈｅｗ上でライブラリをプロジェクトに追加することでビルド時にリンクされます。なお、オンポードライブラリおよびリロケータブルオブジェクトファイルは、デバイスＩＤ無しのＡＰＩのライブラリのみです。

［ビルド］メニューから［SuperH RISC engine Standard Toolchain］を選択しダイアログを表示します。[最適化リンカ]タブを選択し、カテゴリのコンボボックスで[入力]、オプション項目のコンボボックスで[ライブラリファイル]選択後、[追加］ボタンをクリックすると[ライブラリファイル追加]ダイアログが表示されます。ここにインストールしたオンボードライブラリファイルを入力します。オンボードライブラリは複数あり、全てのライブラリを追加して下さい。

また、同様にオプション項目のコンボボックスで[リロケータブルファイル/オブジェクトファイル追加]選択後、[追加］ボタンをクリックすると[リロケータブルファイル/オブジェクトファイル追加]ダイアログが表示されます。ここにインストールしたリロケータブルオブジェクトファイルを入力します。

コマンド名の重複等により不具合が発生する場合には、ライブラリを削除することで解決することがあります。

表A-2-1 オンボードライブラリ

ライブラリファイル名	内容	備考
ipxcmd2.lib	画像認識コマンドＡＰＩ	
ipxctl2.lib	ボード制御コマンドＡＰＩ	
knlsvc2.lib	μＩＴＲＯＮサービスコールＡＰＩ	
fmdisk2.lib	ファイルシステムＡＰＩ	
tcpip2.lib	TCP/IPプロトコルスタック(BSDソケット)ＡＰＩ	
usbapi2.lib	ＵＳＢ　ＨＩＤ制御ＡＰＩ	

表A-2-2 リロケータブルオブジェクトファイル

ライブラリファイル名	内容	備考
stdfnc2.rel	Ｃ言語標準入出力関数	

2.4 セクションの設定

オンボードアプリケーションでは、アプリケーションを構築する場合、アプリケーションをどこに配置するかユーザーが設定する必要があります。ＮＶＰ－Ａｘ２３０の場合はユーザー領域として

０ｘＢ０００：０００００　～　０ｘＢ３ＦＦ：ＦＦＦＦ

までの６４Ｍバイトが割り当てられてられており、この範囲内に全てのユーザアプリケーションが収まるようにして下さい。

［ビルド］メニューから［SuperH RISC engine Standard Toolchain］を選択しダイアログを表示します。[最適化リンカ]タブを選択し、カテゴリのコンボボックスで[セクション]、オプション項目のコンボボックスで[セクション]選択後、[追加］ボタンをクリックすると[セクション追加]ダイアログが表示されます。アドレスに上記アドレス範囲内のユーザアプリケーションの任意アドレスを設定し、セクションに［Ｐ，Ｃ，Ｄ，Ｂ]からアプリケーションに応じたセクションを入力してください。

表A-2-3 Ｈｅｗのコンパイラオプション設定

設定内容		デバイスID無し	デバイスID有り
インクルードファイルのパス		[インストールフォルダ]¥SDK¥SH¥INC 設定例： C:¥VP230SDK¥SDK¥SH¥INC	[インストールフォルダ]¥SDK¥SH¥INC 設定例： C:¥VP230SDK¥SDK¥SH¥INC
マクロ定義		設定しない	MULTI_BOARD_CONFIG
ライブラリファイル		ipxcmd2.lib ipxctl2.lib stdfnc2.rel knlsvc2.lib fmdisk2.lib tcpip2.lib usbapi2.lib	
セクション	アドレス	0xB000:0000 ～ 0xB3FF:FFFF	
	セクション	P,C,D,B（アプリケーションに応じて必要なセクション）	

付録B 環境設定

1. SW設定

表B-1-1 SW1設定

SW1	出荷設定	機能
1	OFF	リセット直後の起動モード設定 (詳細は表B-1-3参照)
2	OFF	
3	OFF	
4	OFF	
5	OFF	未サポート(出荷設定のまま使用ください)
6	OFF	
7	OFF	
8	OFF	

表B-1-2 SW2設定

SW2	出荷設定	機能
1	OFF	シリアル通信設定 (詳細は表B-1-4参照)
2	OFF	
3	OFF	
4	OFF	未サポート(出荷設定のまま使用ください)
5	OFF	
6	OFF	
7	OFF	
8	OFF	

表B-1-3 SW1-4～1 起動モード設定

SW1				機能
4	3	2	1	
OFF	OFF	OFF	OFF	ノーマルモード(カレントドライブ fmd:)
OFF	ON	OFF	OFF	セーフモード(*1)
OFF	OFF	ON	OFF	ノーマルモード(カレントドライブ ata:)
OFF	ON	OFF	ON	ノーマルモード(カレントドライブ usb:)

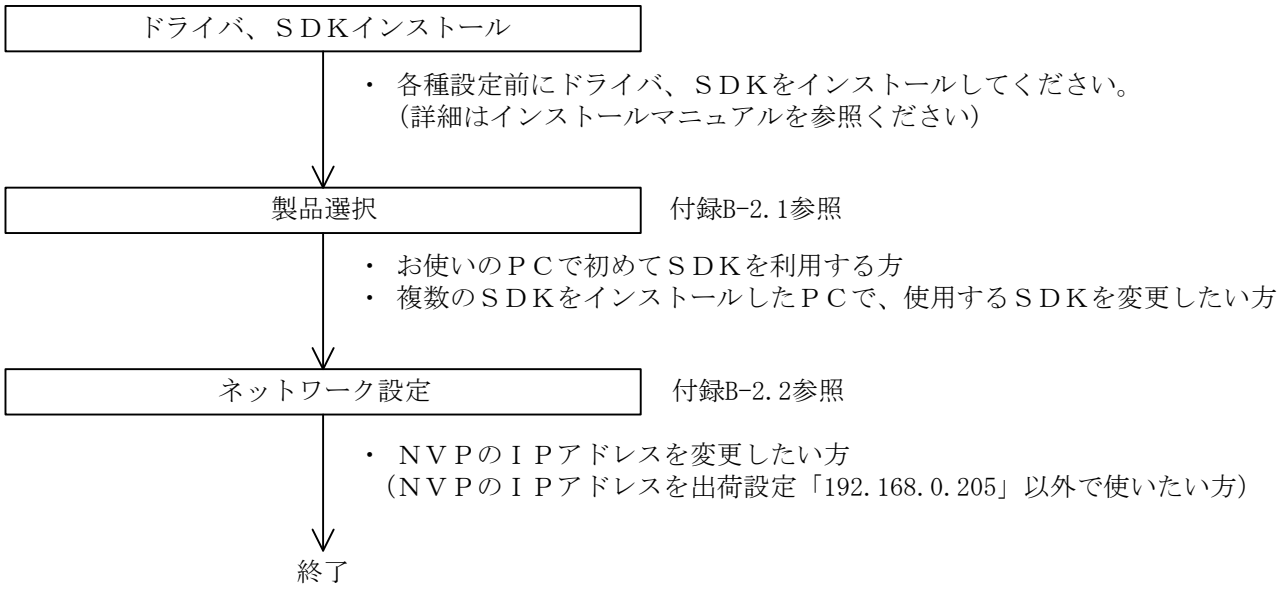
※ その他設定は未サポートです。
(*1) セーフモードはシステム復旧時に使用します。詳細は環境設定マニュアルを参照ください。

表B-1-4 SW2-3～1 シリアル通信設定

SW2			機能
3	2	1	
OFF	OFF	OFF	COM2接続の外部機器とNVPシリアルコントローラ間でシリアル通信
ON	ON	ON	COM2接続の外部機器とカメラ(CH1)間でシリアル通信
ON	ON	OFF	COM2接続の外部機器とカメラ(CH2)間でシリアル通信
ON	OFF	OFF	COM2接続の外部機器とカメラ(CH3)間でシリアル通信
OFF	ON	OFF	COM2接続の外部機器とカメラ(CH4)間でシリアル通信

2. 環境設定

以下に環境設定フローを示します。



表B-2-1 環境設定フロー

2.1 製品選択

ＶＰシリーズＳＤＫをＰＣにインストール後、そのＰＣで使用する製品を選択する必要があります。
また、複数のＶＰシリーズＳＤＫがインストールされるＰＣで、同時に使用できるＳＤＫは１製品のため、使用する製品を変更したい場合、製品を選択する必要があります。

製品選択はVPSetReg2にて行います。VPSetReg2起動して使用する製品名を選択して[書き込み]ボタンをクリックしてください。詳細は環境設定マニュアルを参照ください。

2.2 ネットワーク設定

ＮＶＰのＩＰアドレスは、出荷時に「192.168.0.205」に設定されています。ＮＶＰのＩＰアドレスを任意の値に設定可能ですが、ＬＡＮ経由で設定変更するため必ず一回は「192.168.0.205」のＮＶＰに接続する必要があります。以下にＮＶＰのＩＰアドレスを出荷設定「192.168.0.205」から変更する手順を示します。

① ＰＣのＩＰアドレスを「192.168.0.xxx」(xxxは0,255,205以外)に設定します。
② VPSetReg2でＮＶＰのＩＰアドレスを「192.168.0.205」にします(ＰＣ側設定)。
③ ＮＶＰを起動して、ＮＶＰとＰＣのＬＡＮ接続を確認します(pingコマンド)。
④ VPSetUp2でＮＶＰのＩＰアドレスを任意のＩＰアドレスに設定します(ＮＶＰ側設定)。
ＮＶＰ再起動で設定したＩＰアドレスが適用されます。
⑤ VPSetReg2でＮＶＰのＩＰアドレスを④設定にします(ＰＣ側設定)。
⑥ ＰＣのＩＰアドレスを④設定のＮＶＰに接続可能な設定に変更します。
⑦ ＰＣとＮＶＰとの接続を確認します。

詳細は環境設定マニュアルを参照ください。

画像認識ユニット NVP-Ax230
Software Development Kit
ユーザーズマニュアル（第5版）

開発元

マクセルシステムテック株式会社

電子機器部　〒992-0021 山形県米沢市花沢3091-6

外販営業部　〒244-0801 神奈川県横浜市戸塚区品濃町549-2 三宅ビル

技術サポート窓口　URL http://www.systemtech.maxell.co.jp
E-mail vp-support@maxell.co.jp