dwm デザイン ウェーブ ムック DESIGN WAVE MOOK

LSI/FPGAの 回路アーキテクチャ 設計法

アルゴリズムの高性能ハードウェア化

森岡澄夫 著

- 重要さが増している回路アーキテクチャ設計
- 設計例題の概要と高位検証環境
- アーキテクチャ設計で用いる部品とコスト感覚
- アーキテクチャ設計のあらまし
- 最初のアーキテクチャ案の構成
- データの流れに着目したメモリ配置の最適化
- 並列化による処理速度と回路規模の最適化
- 全体動作制御と部品間のデータ伝送の検討
- 各部品の内部構成の検討とRTL設計
- データ演算処理の設計の最適化
- CPUやパソコンとのインターフェースの初歩

内容・購入方法については，下記のWebサイトをご覧ください．
内容：https://shop.cqpub.co.jp/hanbai/books/52/52891.html

CQ出版社

まえがき

本書では，システム・クラスのディジタル回路の全体（高位）アーキテクチャ設計を説明します．実務における設計事例そのままではなく，そのエッセンスだけを抜き出した小さな例題を使いますが，それを実務の設計と全く同じ思考方法・手順で作ります．

本書には，従来にはない試みが二つあります．

一つ目は，実務設計通りの思考法・センスを紹介しようという試みです．ディジタル回路の入門書は，どうしてもごく単純な機能のサンプル回路を示して，後はその組み合わせで何でも作れますというストーリ展開になりがちです．紙面が限られているのでやむを得ない面もあります．しかし，ステート・マシンやカウンタなどの細かい部品の作り方が分かれば，「組み合わせるだけ」で100万ゲートや1000万ゲートのシステム回路が作れるのかというと，それは不可能でしょう．実用的な回路は，回路化可能なアルゴリズム，データの流れ方の解析，処理の展開法，メモリ量の最適化，データ伝送プロトコル，プロセッサとの組み合わせなど，細かい部品レベルでは存在していなかった概念・考え方をいろいろ持ち込んで総合的に使わなければ，作ることができません．単に部品を寄せ集めて「組み合わせるだけ」，「つなげるだけ」では作れないのが，システムです．

二つ目は，アーキテクチャ設計の説明をしようという試みです．イメージとしては，数十万～数百万ゲートの大規模IPコアや中小SOCを，100個以上のユニットを組み合わせて作るシーンを想像してください（本書の例題ははるかに小規模にしてありますが）．単に機能的に正しく動くようにするだけではなく，狙った目標性能を計画的に達成するのがアーキテクチャ設計です．このテーマは従来ほとんど出版物がありませんでしたが，それはそのはずで，形式知化が難しいのです．アーキテクチャよりずっと下位，例えば論理設計における論理簡単化などは，手順をアルゴリズムとして明確に説明できます．しかし，実用・現実のアーキテクチャ設計を整理して説明するのは，あたかも将棋の名人の思考方法をアルゴリズム化しようと挑戦するようなものです．あいまい・直感的な説明が多々残っていると思いますが，それは筆者の非力・非才によるものです．

本書執筆の動機は，LSIやFPGAの回路アーキテクチャ設計の重要性が高まってきたにもかかわらず，解説書がちまたにほとんどない，という点に尽きます．実はちょうど10年前，拙書「HDLによる高性能ディジタル回路設計」のまえがきに，筆者はこう記しました．

> 『それほど遠くない将来，C/C＋＋言語を使った設計へ移る可能性が高いでしょう．誰もがCベース設計をするようになったとき，いかにハードウェアを意識した設計をするかで決定的な差がつくようになるはずです．』

その前半は，残念ながら当たりませんでした．現在，経済状況などの理由で，動作合成（C/SystemC言語からのRTL合成）を誰しも使う状況には遠く至っていません．しかし後半は予想以上に早く進展しました．実務における設計物は10年前よりもはるかに高度になっており，単体のIPコアでさえ100万ゲート越えが珍しくなくなりました．それ自体が一つのシステムであり，少し前のSOC並みです．これは，動作合成で自動合成すれば済むレベルをはるかに超えています．

また，ASICであれFPGAであれ回路は，初級ソフトウェアにありがちな「言語をコンパイルすればよい」，「性能は，作ってから測りつつ改善していけばよい」というセンスでは，実用として使えるものを作れません．どのような言語やツールを使うかは重要な議論ではあるのですが，何を使うにしても「どのようなハードウェア構成にすれば目標性能をクリアできるのか？」という根本の設計，つまりアーキテクチャ設計は残っています．アーキテクチャ設計がでたらめでは，どのようなツールでも価値ある回路を作れません．回路業界ではいまだに「これからはソフトウェアがハードウェアになる時代だ」などといわれることがありますが，今はマルチコア・プログラミングなど，逆にソフトウェアの方でアーキテクチャを意識する必要性が高まっている時代です．

本書の目的は新しい設計手法の提案ではないので，仮説・机上の話はありません．幸いにも，筆者は企業研究所に所属しつつも量産製品設計を自らの手で行う機会に多く恵まれてきたので，全ては，世の中で広く使われている回路に実際に適用された話です（もちろん，特許などで保護された話や製品固有の機密事項は記載していませんので，安心して利用してください）．個人の限られた経験の中で書いていますので，あらゆる回路設計をカバーできてはいませんが，読者の方々に役立つ部分が少しでもあれば光栄です．アーキテクチャ設計の知識を世に出す必要性は深く認識していたのですが，その重要性が認知されるまで待ち，筆者自身が知見を整理するのに，10年もの時間が流れてしまいました．ただ，どうやら時期は熟したようです．

最後に，筆者とともに回路を作り苦労と喜びを分かち合ってきた多くの仲間の方々と，本書執筆の機会と粘り強いご協力をいただいたCQ出版の西野直樹氏，そして連日深夜まで仕事に熱中している筆者をサポートし続けてくれた妻の和香子，長女の美月，長男の和希に，深く感謝いたします．

2012年春　森岡澄夫

設計例題の概要と高位検証環境

本書では各章で同じ設計例題を使うので，その内容について説明します．専門家でなくてもすぐ分かる簡単な画像加工処理です．また，実務では，処理が正しく動くかを必ず回路作成前に確かめるので（高位検証），本章でもやってみましょう．高位検証環境にはいろいろありますが，アルゴリズムの開発に広く使われているMathWorks社の「MATLAB」と，標準的なC言語を使います．

2-1　設計例題はシンプルな画像処理回路

● 写真の明るさ調整をしてみよう

ディジタル・スチル・カメラで写真を撮影したとき，逆光などで撮影対象が暗くなってしまい，困った経験はないでしょうか．最近のデジカメは，いろいろな画像処理を施すことで悪条件下でもきれいな写真が撮れるようになりましたが，そのために必要な画像処理を回路化してみることにしましょう．

例題として用いる処理の概要を**図2-1**に示します．これは，入力された画像のうち，人の部分が適度な明るさになるよう，画像全体の明るさを調整する処理です．

詳細は後述しますが，これは原画像から人の部分の明度を測る処理と，その結果に基づいて全体の明度を変更する処理の2段階からなります．人の部分は肌色領域を抽出することで判定します（ただし実用では，色で判定する方法はあまり使わない）．

● 画像処理は回路化の要求が高い

こうした画像処理は，回路化の要求が高い分野で，実務で回路設計を行う場合にもよく登場します．特に，家電製品や組み込み機器では，画像処理をマイクロプロセッサではなく，専用回路によって行う場合が非常に多くなります．その理由は，大量のデータを高速・低電力・低コストで扱わないといけないためです．

図2-2に，家電製品などで扱うデータが，どのくらいの量であるかを示します（圧縮などを施す前の生データの量）．**図2-2**（a），（b）が画像，**図2-2**（c）～（e）がそれ以外のデータですが，画像のデータ量は桁違いです．

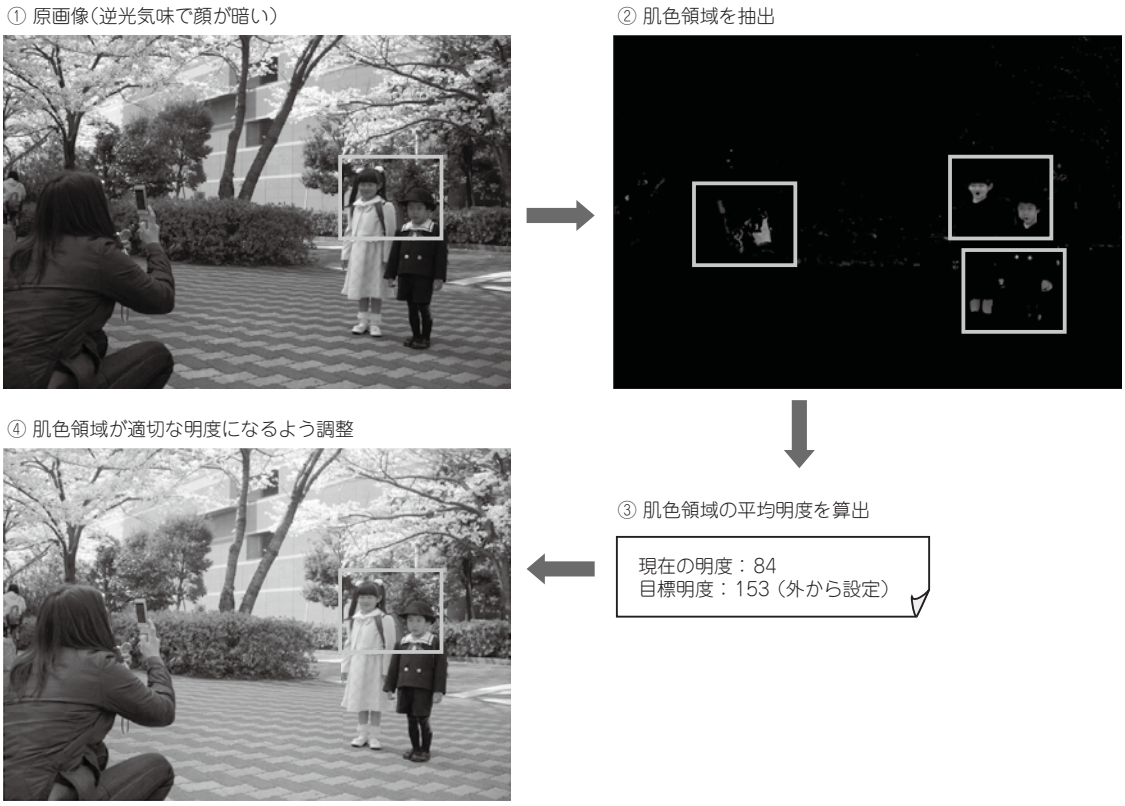


図2-1　例題「人に合わせた写真明度調整」の処理
肌色で人を検出する処理を例題とする．分かりやすい方法ではあるが，実用では色で判定する方法はあまり使われない．

本書の執筆時点(2012年)において，図の上段に示したような量のデータを組み込みプロセッサや1チップ・マイコンを使ってリアルタイム処理することは困難であり，高速なパソコン用プロセッサでもあまり余裕がない状況です(図の下段のような低速データについては，マイコンなどでも処理可能)．そのため，ディジタル・ビデオ・カメラやディジタル・スチル・カメラの内部では，画像の加工や圧縮などを行うために専用回路が用いられています．

将来，組み込みプロセッサの能力が上がれば，画像処理のソフトウェア処理は十分可能になるかもしれません．しかしそのときにも，さらに重い別の処理をしたり，低消費電力化を図ったり，あるいは決まりきった処理からプロセッサを解放したりする目的で，専用回路は使われ続けるでしょう．今でも1チップ・マイコンやSOC(System on a Chip)の内部では，I/O制御処理や頻繁に使われる計算などをプロセッサでやれないわけではないものの，専用回路を使って処理しています．同じ理由からです．

アーキテクチャ設計で用いる部品とコスト感覚

例題の画像処理について，回路化の検討に入ります．まず，何を目的に回路を作るのかに立ち返りつつ，アーキテクチャ設計に登場する部品やコスト感覚を紹介します．こうした知識がないと，アーキテクチャを検討しても，その良否が判断できないからです．最も重要なのは，処理をただハードウェア化することには大して意味がない，という現実です．

3-1　なぜハードウェアのコスト感覚が必要なのか

● 回路化すれば速くなるという暗黙の認識

一般的な技術者が回路（LSI）に対して持っている共通の認識は，「ソフトウェアは遅くてハードウェアは速い．だけどハードウェアは作るのにばくだいなお金や時間，専門知識がいるし，直せないのでリスクが高い」というものです（**図3-1**）．電子情報系の技術者であれば，回路化することで低電力化の効果があったり，FPGAを使えば回路の修正ができたりする（プログラマブルである）ことまで知っているでしょう．

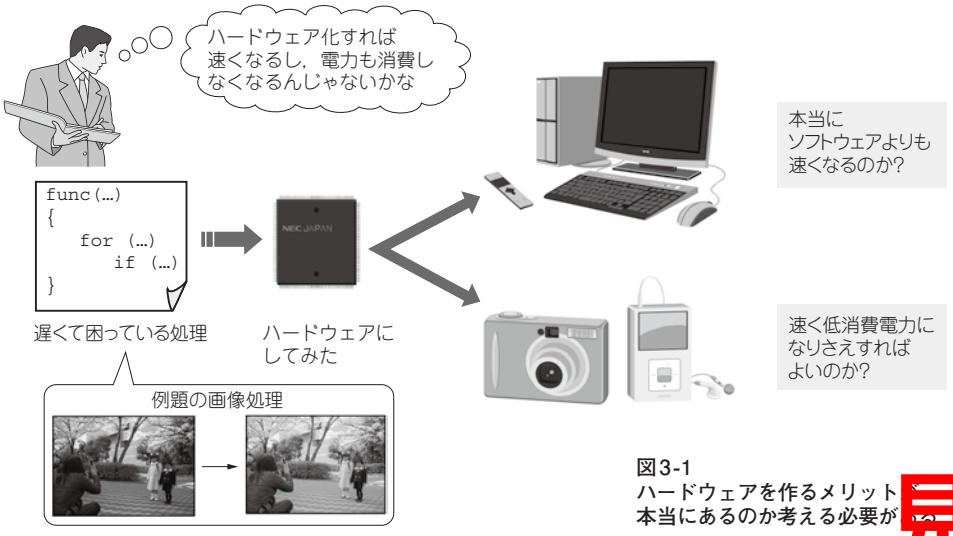


図3-1
ハードウェアを作るメリットが本当にあるのか考える必要がある

回路を組むときは，だいたい高速化などの効果を期待しているはずです．もし回路化によって1/100の速度になったり100倍の電力を消費したりするのであれば，回路化など最初から考えもしないことでしょう．

また，回路設計ツールの宣伝文句でしばしば「ソフトウェア（やプログラミング言語）から回路が作れる」といわれたりするのも，暗黙のうちに，回路化によって速くなることを想定しています．

● 漠然と回路を作っても決して速くならない

しかし実際には，回路にすれば速くなるほど単純ではありません．何も考えずに回路化すると，逆に遅くなる可能性が十二分にあります．本章で幾つかの例を示しますが，速いか遅いかを議論するときには，次の条件によく注意する必要があります．

- ソフトウェアと回路を，それぞれどのような環境やデバイスを使って動かすのか．「ソフトウェアは3GHz動作のパソコン，回路は100MHz動作のFPGA」というように土俵が全然違ってしまうと（しかもそれが普通），ハードウェアが勝つのは容易ではありません．
- 製作にどれだけのお金と時間をかけられるのか．無尽蔵のお金と時間を投入して凝りに凝った専用回路を作れるのであれば勝つのは当たり前です．そうはいかないから悩みが生じるわけです．
- ソフトウェアと回路で同じアルゴリズムや入出力方式なのか，それぞれに特化した別方式なのか．一般的には，ソフトウェア用に高速化されたアルゴリズムは回路に向いておらず，逆も同様です．「向いていない」程度では済まず，「回路化（またはソフトウェア化）ができない」という場合もしばしばあります．また，データの入出力方法もかなり違います．ソフトウェアではデータがメモリやハード・ディスク上にまとまってある場合が多いのですが，回路ではそうとは限らず，ストリームである場合が大半です．

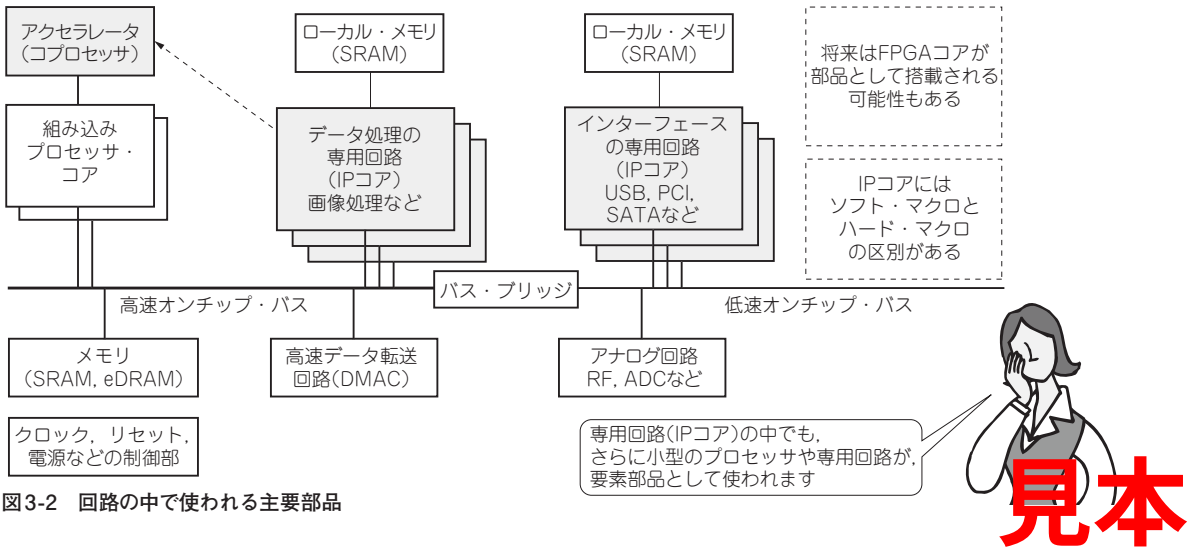


図3-2　回路の中で使われる主要部品

アーキテクチャ設計のあらまし

本章では例題の画像処理回路のアーキテクチャ設計において，何を行うかについて，簡単に要約して説明します．具体的な検討は第５章以降で行いますが，各章で何をしているのかがつかめなくなったとき，本章に立ち戻ってみてください．

4-1　回路化をうまく進めるための経験的コツ

● アーキテクチャ設計はマニュアル化しにくい

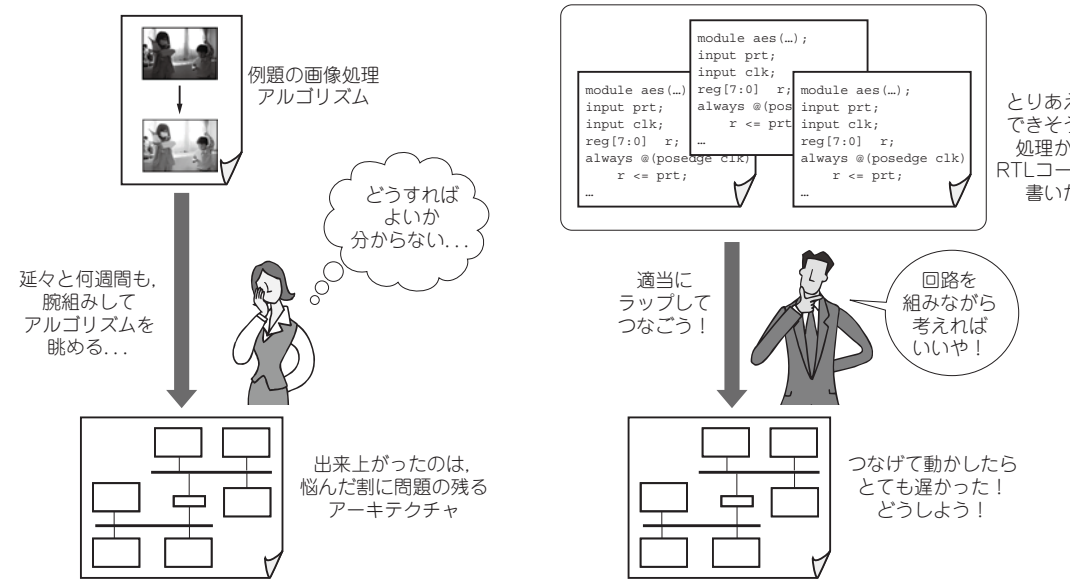
回路のアーキテクチャ設計は，ほかのいろいろなシステム設計と同じで，「このような手順でこう作業を進めていけば誰でも間違いなくできる」ということを明文化したマニュアルを作るのが難しい部分です（こう書くと，本書の意義を否定してしまうが…）．実務でシステムやアーキテクチャの設計をすると，「性能を50％上げて回路規模を半分に減らすのをあと１週間のうちに３人でやらないといけない」というように相反する要求がもぐらたたきのように出てきて，あの手この手で落としどころを見つけるかじ取りをする必要があります．いくら形式知化が大事とはいえ，このような作業は，個別の事例や人的な側面に強く依存するので，あいまいさなくマニュアル化することができません．

ただし，本書で試みているように技術上の基本手順ならば整理できますし，「こうすると明らかにまずいことになった」という筆者の失敗経験は事実としてあるので，まずはそこから紹介しましょう．

● 考え込むだけ，作るだけと極端に走らない

筆者が実務レベルの回路設計に慣れていなかった初心者の頃，陥ったのが**図4-1**の二つのパターンです．

まず，やりたい処理は理解していても何をどう考えれば回路に到達できるかが分からないので，延々と考え込んでいるだけの状態になってしまったのが**図4-1**（a）です．このパターンの特徴は，とにかく頭の中だけで設計をしようとして，手を動かさないことです．実質的には何も考えていないようなもので，時間はかかっているのですが設計はさっぱり進みません．



（a）回路化における筆者の失敗例（1）
「頭の中だけで設計しようとする」

（b）回路化おける筆者の失敗例（2）
「部品のRTLコードから作りはじめ，アーキテクチャは後付け」

図4-1　処理の回路化でやりがちなパターン

一方，考えるのを一切止めてしまって，分かる部品や処理からRTLコード（回路）を直接作り始めてしまったのが，**図4-1**（b）です．回路の実装能力に自信が付いてきて「力業で何でもねじ伏せられる」という気がしはじめた頃に陥ったパターンです．多くの部品を統合したときに，性能が全く出ないような問題が発覚してきます．パッチを当てて対処できればラッキーですが，根本的な原因があった場合は全面作り直しになってしまいます．

どちらのパターンも，「どんなアーキテクチャにしてよいか分からない」となったときに考え込むだけになるか，考えるのを放棄するかであり，非常に極端です．

● アーキテクチャの作り直しをいとわない

回路化に当たり，筆者自身や筆者が関係してきた幾つもの設計チームが採っているのが**図4-2**の方法です．もちろんプロジェクト状況や組織体制などによる差異はあるのですが，おおむね共通しているのは次の点です．

- 処理が複雑で高性能になればなるほど，回路アーキテクチャは一発で決まらず，繰り返し修正をしながら固めていくものである．
- 全ての回路（RTLコード）を作ってから一気に結合して性能を測る，ということはしない．現実の実務は「アーキテクチャを完全に決めてからRTL記述を始める」というほどきれいには進まないが，それでもアーキテクチャ案や性能面の見通しなどが何もないままRTLコードを記述するよ[illegible]とは

第11章

CPUやパソコンとのインターフェースの初歩

どのような回路を作っても，必ず外界との接続が必要になってきます．接続相手となるのは，オンチップCPU（のバス），パソコン，外部デバイスなどです．本書では実用オンチップ・バスや個別デバイスの詳細は解説しませんが，それらの理解に欠かせない初歩的な知識を説明します．理解を深めるのに役立つとともに，実用度も高い軽量CPUの設計事例も示します．

11-1　オンチップCPUとのインターフェース

● CPUとの接続は頻繁に行われる

画像処理などのIPコアを作成したとき，それをLSI/FPGA内部で単独で使う場合もありますが，オンチップCPUと接続する場合も頻繁にあります．ある程度以上の規模のシステムであれば，まず間違いなく行うといってよいでしょう．接続の目的としては，

- そのIPコアをCPUのアクセラレーション（高速化）に使う
- 主記憶上にある入出力データをCPUに転送してもらう
- IPコアの起動・停止をCPUに制御してもらう

などいろいろあります（**図11-1**）．

前章までに何度か触れてきていますが，IPコアはオンチップ・バスを介してCPUにつながります．接続のためにIPコアをバス・インターフェース回路でくるむのですが，それをラッパと呼びます（**図11-2**）．ラッパの回路構成については，後ほど述べます．

● バス接続のためのポートの検討

IPコアをバスに接続するとき，一つのポートしか使えないわけではありません．同じIPコアに複数のポートを設け，それぞれをバスに接続することもできます．よく見られるのは，**図11-3**のようにスレーブ1本か，またはスレーブとマスタを1本ずつ持つ形態です[注1]．

注1：一般のオンチップ・バスでは，接続個所がむやみに増えると信号のドライブやアービトレーションに困るので，ポート数はできるだけ少なくしてほしく．マスタや制限の数にも制限が設けられている場合がある．

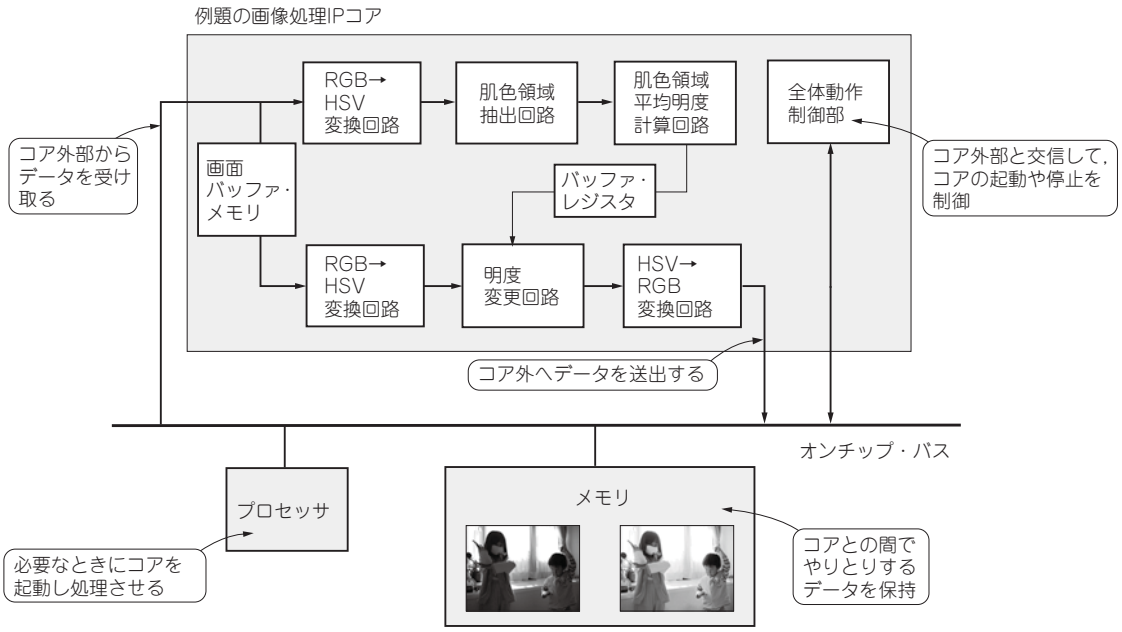


図11-1　上位階層との接続の具体的な方法を考える

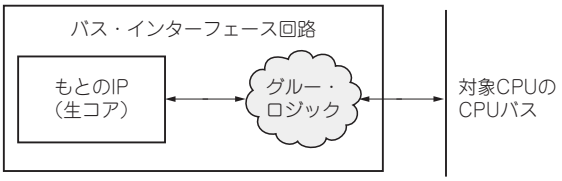


図11-2　CPUバスとの接続の基本

例題の画像処理回路であれば，データ入出力部や全体動作制御部にバス・インターフェースを設けるのがよいでしょう（**図11-4**）．全体としてスレーブ・インターフェースを1個だけ持たせる案も考えられますが，回路へのデータ転送速度は遅くなります．

バスにつなぐのは処理全体の入出力だけ，と決まっているわけでもありません．例題の画像処理回路の場合，**図11-5**のように前半処理と後半処理を切り分けて，それぞれをバスにつなぐ構成もあり得ます．その場合，後半処理の直前にあった画面バッファは廃止し，主記憶メモリで代用させます．主記憶～IPコア間で2回データを転送しなければならないというデメリットがありますが，巨大なバッファ・メモリを削減できるのはメリットです．

このように，ポートの設置数やバスとの接続の仕方はいろいろ自由にアレンジできるので，システム全体の動きを想像しつつ，パターンにこだわらないで検討します．

ISBN978-4-7898-5289-0
C3055 ￥3700E
CQ出版社

定価４,070円（本体3,700円）⑩

dwm デザイン ウェーブ ムック DESIGN WAVE MOOK

LSI/FPGAの 回路アーキテクチャ 設計法