Making Everything Easier!™

2nd Intel® Special Edition

FPGAs FOR DUMMIES®

内容：

- FPGA の仕組み
- FPGA、ASSP、ASIC の違い
- FPGA を機能ブロックとして システムで使用する方法

提供：

intel

Andrew Moore
with Ron Wilson

FPGAs FOR DUMMIES®

2nd Intel® Special Edition

Andrew Moore著

**インテル® プログマブル
ソリューションズ ・グループ編集長、
Ron Wilson 協力**

WILEY

FPGAs For Dummies®, 2nd Intel® Special Edition
出版：
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-45262-1 (pbk); ISBN: 978-1-119-45259-1 (ebk)

製作：アメリカ合衆国

10 9 8 7 6 5 4 3 2 1

謝辞

本書の出版にあたりご協力いただきました皆様に心より御礼申し上げます。

プロジェクト・エディター：Jennifer Bingham
アキジション・エディター：Katie Mohr
エディトリアル・マネージャー：Rev Mengle
事業開発担当：Karen Hattan
プロジェクト・コーディネーター：Magesh Elangovan
スペシャル ヘルプ：Intel Corporationからの多大な協力を得て製作されています

目次

はじめに

フィールド・プログラマブル・ゲート・アレイ（Field Programmable Gate Array：FPGA）はカスタマイズされたデジタルロジックを設計者がフィールドでプログラム化するための集積回路です。FPGAは1980年代から存在しており、当初はすべての設計チームにカスタムロジックを作成する能力を与えるために開発されました。初期の頃は、設計にFPGAを組み込むと、簡単な機能をFPGAに実行させるためだけに多くのプログラミングを行わねばならず、ほとんどの設計者はそれを避けてきました。大学時代の研究以降にFPGAについて調べたことがない方は、もう一度見直してみるとよいでしょう。

FPGAは有用ではあるが質素なインターフェイス・デバイスから、独自のマイクロプロセッサー、メモリーブロック、およびインターフェイスを備えたシステムレベルの集積回路（IC）へと進化を遂げました。今では次なる目玉と目されています。

手ごろな価格の開発キットを購入し、無料ツールをダウンロードしてFPGAの世界を探り始めるには、今がチャンスです。本書はFPGAの実用的な使用方法を誰もが理解できるように製作されました。

本書の概要

システムデザイナー、経験豊富なエンジニア、または工業学校の授業以来FPGAに触れたことがない方々にとって本書は最適です。

また、本書はIntel Corporationの一部となったAltera Corporationの協力を得て製作されています。

本書で使用するアイコン

本書では、必要に応じて以下のアイコンを使用し、重要な情報に注意を促します。よく見るスタンプや絵文字はありませんが、注意して読んでみてください！それぞれのアイコンの違いは以下のとおりです。

このアイコンは誕生日や電話番号を保存するアドレス帳のように、脳内のキャッシュやメモリーに保存しておくべき情報を示します。

もしかすると、パーティーで友達を驚かせるような役立つ情報をゲットできるかもしれませんよ！

ウェイターやバーテンダーにチップをあげろとは言っていません！ただ少し時間を取って情報を確認すれば、後でイライラすることが少なくなります。

さらに知りたいなら

本書には情報が溢れていますが、48ページですべてを語ることはできません！FPGAの詳細が知りたい方は、`www.intel.com/altera`をご覧ください。インテル® FPGAの詳細をご確認いただけます。またその他にも、ビデオやウェビナーを見たり、デモをダウンロードしたり、データシートやホワイトペーパーを読んだりすることもできます！

第1章

誰もが利用できるFPGA

本章の内容

- FPGAのご紹介
- FPGAの仕組み
- FPGAとASICの違いを検討する

ようこそ！これを読んでいるということは、あなたはフィールド・プログラマブル・ゲート・アレイ（FPGA）について聞いたことがあり、もっと詳しく知りたいと考えているエンジニアでしょうか。FPGAをご自身の設計でどのように使えるのかを知りたいのかもしれませんね。本章では、FPGAについて紹介し、どのような課題を解決できるのかについて説明します。また、FPGAの仕組みや設計でのトレードオフ、操作についてもお話します。

FPGAには、他のハードウェア構築方法と異なる大きな特長が2つあります。まず、競合他社も使用しているASSP（Application Specific Standard Product、特定用途向け汎用IC）や、設計するのに時間とコストがかり、リスクのあるASIC（Application Specific Integrated Circuit、特定用途向け集積回路）を使わずに、必要なハードウェアを構築できる点です。

また、FPGAはカスタマイズが可能なので、ASSPのマイクロプロセッサー・コアよりも簡単で素早く、電源効率よく演算を行うことができることも重要な点です。

FPGAが必要な理由

喜んで説明します！FPGAとは、製造の後でも回路を書き換えることができる半導体デバイスです。FPGAを使用すると、製品を出荷した後でも製品の特徴や機能を現場（フィールド）でプログラミングできるので、新しい規格に適応し、特定の用途向けにハードウェアを再構築できます。これがフィールド・プログラマブルという名称の所以です。また、ゲートアレイとは、論理ゲートが格子状に配列されていることを意味します。これらをうまく活用すれば、簡単な計算で重要な演算の実行が可能です。

簡単に言えば、FPGAは設計に柔軟性をもたらすものであり、莫大なコストや設計スケジュールの遅れなく、システムの各部分の仕組みを変更する方法の1つです。

自動車に搭載された後方確認用カメラがよい例です。現在イメージセンサーが画像をとらえた瞬間からカメラシステムがイメージフレームをディスプレイに表示するまでに250ミリ秒かかっているとして、政府規制の変更により、遅れまたはレイテンシーが100ミリ秒以内でなくてはならなくなったとします。この場合も、FPGAで画像信号処理パイプラインを調整すれば、レイテンシー要件を遵守できます。これはマイクロプロセッサー・ベースのシステムではほぼ不可能です。上記の例では、FPGAを使用することでパーツを再設計したり、新しいプロセッサーを購入する必要がなくなり、企業は大きな利益を得ることができます。

FPGA回路は当初、大きすぎて単一のチップに多数搭載することができませんでした。設計者はFPGAを使用してインターフェイスを構築し、顧客がインターフェイスを再プログラミングして何か別のこと（キーボード入力で稼動するインターフェイスからタッチスクリーン・デバイスによる入力に対応するインターフェイスへの変更など）をするしかなかったのです。しかし、設計者たちは間もなく、FPGAを使用してサブシステム全体の構築が可能なことに気づきました。つまり、設計者は、サブシステムの実装にASICを使用しなくてもよくなったのです。回路の部品が小さくなっていくにつれ、設計者は、1つのチップにより多くのデバイスを搭載できるようになりました。その結果、より複雑な機能と高速演算が可能になり、計算の高速化や消費電力の削減につながったのです。

現代のFPGAは構成可能なスタティック・ランダム・アクセス・メモリー（SRAM）と高速入出力ピン（I/O）、論理ブロック、およびルーティングの組み合わせでできています。さらに詳しく説明すると、FPGAにはロジックエレメント（LE）と呼ばれるプログラム可能な論理素子と、LE間の物理的な接続を可能にする再構成可能なインターコネクトから成る階層が含まれています。複雑な機能を実行するためにLEを構成することも、ANDやORといった基本的な論理ゲートを実行することも可能です。また、ほとんどのFPGAにはメモリーブロックが含まれています（このトピックに関する詳細については、「FPGAの基盤」をご覧ください）。

ASICとASSP

ASIC（特定用途向け集積回路）は、トランジスターやコンデンサー、抵抗などの電気部品でできた集積回路で、特定の用途向けにカスタマイズされたシリコンや他の半導体でできたウエハー上に組み立てられています。ASICの例としてはボイスレコーダーと高性能ビットコイン・マイナーの2つが挙げられます。ICで使用される部品のサイズは年々小さくなっており、面積の大きさは変わらず、より複雑な回路を作成することが可能になりました。この部品の縮小により、複数のマイクロプロセッサーやその他の複雑なサブシステムを搭載してもサイズが十分なASICも出てきました。

一方、ASSP（特定用途向け汎用IC）は特定の用途専用に開発された汎用ICで、単独の顧客向けに設計、販売されるASICとは異なり、複数のユーザーに販売されます（つまり標準品）。ASSPの例としては、多くのスマホやタブレットのコア部品であるマイクロコントローラーやシステムチップが挙げられます。

ASICもASSPも特定の機能専用に特別に設計されています。構成が厳密に制限されるため、ASICもASSPもとてもコンパクトで価格も手ごろであり、高速で消費電力も少なく、電子機器の設計には理想的な条件を備えています。しかし、それぞれの機能は製造時に回路化されているため、例えほんの一部であっても簡単に機能や回路を変更することはできません。また、回路がシリコンのウエハー上に加工されているため、回路を外して別の何かと取り換えることもできません。設計に何らかの変更が必要な場合は、チップ全体を破棄し、最初から作り直す必要があります。

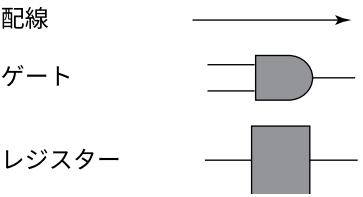
それが今では、ハードIP（ハードの設計成果物：Intellectual Property）をFPGAファブリックに構成できるようになったため、消費電力やコストを抑えつつ、豊富な機能を提供できるようになりました。現代のFPGAに含まれているハードIPの例としては、メモリーブロック、演算回路、トランシーバー、プロトコル・コントローラー、中央処理装置（Central Processing Unit：CPU）が挙げられます。しかし、重要なことは、ハードIPは他のFPGAと違ってカスタマイズできないことです。ハードIP（デジタル信号プロセッサー：DSPなど）を統合すれば、設計者はシステムに共通部品の再設計をしなくて済みます。これらの機能はコモディティー化されており、ほとんどの電子装置でほぼ統一されているため、FPGAメーカーはハードIPコンポーネントをFPGAに組み込み、手間を省くことができます。

FPGAを評価する

以下のセクションでは、最初にFPGAについて学んだときから随分時間が経ってしまい、その詳細を忘れてしまった方のために、FPGAのコンポーネントについて詳しく解説し、FPGAとは何かを分かりやすく説明します。

FPGAの基盤

デジタルなものは何でも配線、論理ゲート、レジスターという3つのコンポーネントで構築できます。（図1-1参照）。レジスターは他の何かを記録するように指示されるまで情報の一部を記憶します。論理ゲートはシンプルな信号の論理演算を行い、配線は他のコンポーネントを接続します。



提供：Intel Corporation

図1-1： デジタルシステムの基盤

論理ゲート

論理ゲートはデジタル回路のコア機能を実行します。つまり、「0」か「1」を表すためにコンピューターが使用する電子パルス（以下で詳述）である入力信号においてシンプルな論理を実行します。これらのシンプルな演算一つひとつにより大きな成果が出るわけではありませんが、数千、数百万と演算を組み合わせれば、とても大きな力を発揮します。お持ちのコンピューターのCPUは数十億もの論理ゲートでできており、そのおかげでコンピューターはさまざまな素晴らしい機能を実行できるのです。

論理ゲートは機能回路として稼動するため、ブール代数と呼ばれる算術を使用します。ブール代数は1854年、ジョージ・ブールにより初めて紹介されました。変数の値が数字であり、主な演算が加算と乗算である初等代数と違って、ブール代数は主に「AND演算（論理積）」、「OR演算（論理和）」、「NOT演算（論理否定）」を実行します。ブール代数での変数の値は真理（truth）値であり真（true）と偽（false）です。

基本的なブール代数の演算は以下のとおりです。

- **And** 演算（論理積）は「x AND y」というかたちで表し、xもyも「真（true）」である場合に「真（true）」となり、それ以外は「偽（false）」となります。
- **Or** （論理和）は「x OR y」というかたちで表し、xまたはyのいずれかが「真（true）」である場合に「真（true）」となり、xまたはyのどちらも「真（true）」でない場合に「偽（false）」となります。
- **Not** （論理否定）は「NOT x」というかたちで表し、xが「偽（false）」である場合に「真（true）」となり、xが「真（true）」の場合に「偽（false）」となります。

ブール代数の最も一般的な用例としては、デジタル論理設計が挙げられます。ブール代数は入力が「0」および「1」または「偽（false）」および「真（true）」のデジタル回路に直接マッピングします。「0」と「1」を使ったシンプルなブール代数で演算を行う論理ゲートの接続数を増やすことで、高度な機能を実行できるシステムが手に入ります。論理ゲートを使用すれば、数十億ものデバイスをGPSと接続し、火星探査車をガイドすることも、お持ちのモバイルデバイスでお好きなゲームを稼動させることも可能です。

レジスター

レジスターは後から使えるようにデータの一部を保管する、シンプルなデバイスです。電話をかける前に電話番号を記録しておくアドレス帳のようなもので、素早くアクセス可能な、データの一時保管場所と考えることができます。約束している時間など、何か他のことを思い出さなければならなくなった際は、前に記録した電話番号が約束の開始時間で上書きされます。レジスターはどんな情報でも、その情報を忘れて新しい情報を記録するよう指示されるまで保管します。

配線

デジタルに必要な要素の3つ目はレジスターや論理ゲートをすべて接続する配線です。1+2のようなシンプルなタスクから、Blu-rayディスクを読み込む青いLEDのパルスをテレビに映し出される明確で解像度の高い画像に変換するような複雑なタスクまで、実行したいタスクを行うには、これらの要素をシステム全体にわたり接続する必要があります。適切な数の論理ゲートとレジスターを配線でつなげることにより、お好きなデジタルシステムを構築できます。

FPGAをクリエイティブに説明する

ちょっと待ってください！論理ゲートやレジスター、配線の説明はどれも技術的で難解なようです。経験豊富な電気技師でなければFPGAを理解することはできないのでしょうか。そんなことはありません！以下の2つの例えを使用してFPGAを説明しましょう。最初の例えはビーズと糸。2つ目の例えは子供の頃に遊んだおもちゃのブロックです。

ビーズとブロックは論理要素を使用して電気システムを構築する2つの手法に似ているのです。

ビーズと糸

ビーズと糸で例えられる方法を使用すると、小さなビーズと細い糸ですべてをつないで模様を緻密にコントロールすることができます。その結果、非常に複雑で美しい模様ができるのです。ただし、この緻密なコントロールにはお金がかかります。模様を少しでも変更しようと思ったら、すべてをほどいて最初からやり直さなければならず、ほぼ不可能です。このビーズと糸をデジタル電

子機器の設計に当てはめると、ASICやASSPによく似た設計となります。

さまざまな色のビーズをさまざまな模様に配置し、糸でつないだビーズ作品の模様を想像してみてください。これらのシンプルなコンポーネントを使用することで、ビーズの数や色、配置により、最もシンプルなものから最も複雑なものまで、ありとあらゆる模様を作り出せます。

では、ビーズがレジスターと論理ゲート、糸が配線であると想像してみてください。これらの要素もビーズと糸のように、あるシステムを生み出します。そのシステムはとてもシンプルなものから非常に複雑なものまで、さまざまな計算を実行します。さまざまなビーズの色はさまざまな種類の論理ゲート（AND、OR、またはNOT）を表すと思ってください。シンプルな色のビーズも、糸を通すことで入り組んだ模様になるように、シンプルな算術演算も非常に複雑な計算になり得るのです。

ビーズを並べて模様にすると美しいデザインができあがります。では、ビーズの配置を変えたり、色を変えたりして模様を変更したいときはどうしたらいいのでしょう。そう、ここから先が問題なのです！ 模様を変えるには、すべての糸をほどき、再度配置し直さなければなりません。しかし、糸がしっかりつながっているため、デザインの一部だけをやり直すことはできないのです。そのとおり。少しでも模様を変えたいと思ったらデザイン全体をほどいてしまう以外にありません。これでは柔軟とは言えませんよね。

おもちゃのブロック

おもちゃのブロックは、お互いに連結できる部分が決まっている比較的大きな塊です。デザイン自体はビーズと糸により作られた模様ほど洗練されておらず、入り組んでもいません。ただし、すべてを壊して作り直さなくても、部分的な変更が可能です。これがデジタル電気設計に対する2つ目のアプローチ、FPGAです。

子供のころ、または子供と一緒に何時間も夢中でブロックの塔や消防車、宇宙船などを作ったことは誰でもあるでしょう。同様に、ブロックを使用して、論理ゲート、レジスター、すべてをつなぐために使用される配線の部分から成るテーブルを作成することで、デジタルシステムを表すことができるのです。

では、誰かに右下側のテーブルの模様（ブロックの色など）を変更するよう頼まれたとしましょう。ブロックは相互に連結しているため、右下側のブロックを簡単に外して違うブロックと取り換えることができます。テーブル上の他のブロックはそのままの状態を保ち、一部を変更するためにデザイン全体を作り直す必要はありません。

FPGAとASICの比較

FPGAは一般的にASICよりも柔軟で費用効果が高くなっています。その理由は以下で説明します。

コストと柔軟性

FPGAはASICが実行できる論理機能をどれでも実行できるうえに、チップ製造後でも機能を変更できるという、多くの用途で求められる明確なメリットがあります。また、顧客側でFPGAのプログラミングが可能なため、ニーズを満たすためにASICの設計や構築をベンダーに依頼する必要がなく、FPGAはASICよりも費用効果が高くなります。

スピードを落とさずにタイムリスクの削減を図る

世界最先端の半導体プロセスを使用したい場合、コストを考慮しなければ、最速のFPGAよりも速いASICを設計できます。しかし、最先端のプロセスを使用している人はほぼいません。なぜなら最先端プロセスの使用は非常に困難でリスクを伴い、破壊的な費用がかかるからです。事実、利用可能になった新プロセスに飛びつくASSP企業はひと握りしか存在しないのです。他の企業は1世代、2世代または3世代分も古いプロセスを使用しているのです。そして、現段階で最速のFPGAは、そのような旧バージョンのASICプロセスと渡り合えるのです。また、FPGAを使用すれば、設計の手間が減り、リスクも大幅に抑えられます。

例えば、決まった電力効率とパフォーマンス要件を持ったシステムを設計しており、65ナノメートル（nm）のASICを使用する計画であるとしましょう。実は、現行の20nmのFPGAでも同じような結果が出せるのです。

また、FPGAを使用することで、設計時間が短縮され、設計エラーのリスクを抑えられるうえに、ASICよりも総保有コスト（TCO）が低くなります。多くの用途において、FPGAの消費電力はニーズを満たすものとなるはずです。TCOの低さと柔軟性の高さにより、FPGAは多くのケースで最高の技術的選択となるのです。

ナノメートル（nm）はチップのトランジスターのサイズを表す単位です。トランジスターは過去数十年で大幅に小さくなりました。第2章をご覧ください。

FPGAを使用してシステムを設計すれば、ブロックの例で表されるように、設計全体に影響を与える必要なくFPGAの一部を変更できるため、コンフィグラビリティーが高まるうえに開発スケジュールに影響を与えるリスクが減少します。

驚くほど簡単なFPGAの使用

設計者の中には、現代的なFPGAを使用してシステムを構築し、何か有益なことをするには、数百万もの論理ゲートと膨大な量の接続に手を加える必要があるという間違った印象を持っている人もいます。もしそれが本当であれば、FPGAの使用はこれほど普及していません。逆に、FPGAのユーザーは半数ほどに減っていたでしょう。

嬉しいお知らせとしては、FPGA設計者がすでに、クロック・ジェネレーター、ダイナミック・ランダム・アクセス・メモリー（DRAM）コントローラー、ペリフェラル・コンポーネント・インターコネクト・エクスプレス（PCI）・コントローラー、およびすべてのマルチコア・マイクロプロセッサーといった一般的に必要とされるコンポーネントを追加するという大変な仕事をほとんど終えており、あとは用途に合わせて機能をカスタマイズするだけでかまいません。

ハードIP

本章の前半でも触れたハード IPはDRAMコントローラー、PCIeコントローラー、クロック・ジェネレーター、大きなメモリーブロックなど、FPGA内に構築されたIPです。事実、今日のFPGAには数多くのハードIPがあり、システム・オン・チップ（SoC）に対応しています。

多くのシステム設計者がFPGAのハードIPに組み込まなくてはならない一般的な機能だけでなく、レーダー用または通信用の高速シリアル・トランシーバーや信号処理用のデジタル・シグナル・プロセッサー（DSP）積和演算などの一般的にあまり必要とされない機能でさえも組み込むことが可能です。今日では、デュアルコアARM（マイクロプロセッサーのブランド）CPUサブシステムでさえも組み込むことができ、ハイエンドFPGAではダイ領域の半分をプログラム可能な論理、もう半分をハードIPで構成することが可能です。現代の設計者は一般的に、必要なIPをすでに備えているFPGAから始めて、その後、プログラム可能な論理を使用して、特定の用途向けにFPGAをカスタマイズします。

並列処理と次数低減

現代の設計者は、複雑な演算を1つの簡単な演算に置き換えることができ（次数低減）、一連の命令により、複雑な演算を同時に実行する並列処理能力も持つスマートなツールを上手く作成できるようになりました。

聞こえはいいですが、一体どういうことなのでしょう。マイクロプロセッサーは命令を実行することで何でもできます。プログラムが乗算を指示すると、マイクロプロセッサーはその命令をメモリーに読み込み、命令をデコードし、数字をロードし、乗算して結果を保存します。これらのプロセスはどれも時間と電力を消費します。では、数字を2倍にしたいだけの場合はどうでしょう。これは単なるシフト演算です。一般的に、乗算の係数のいずれかが定数である場合、複雑な演算を1つの簡単な演算に置き換え、処理時間と電力を節約することができます。マイクロコントローラーはFPGAと違い、可能な場合に乗算を加算に削減する賢さを持ち合わせていないため、乗算を実行する必要があり、より多くの電力消費および実行速度が遅れる原因となります。

FPGAはベクトルの計算などを実行する際に力を発揮します。ベクトルの計算は物理の授業に出てくるだけではありません。プログラマーは大きな数字の組にそれぞれに同じ演算を実行したい場合にこれを使用します。この点でFPGAが持つ大きなメリットは、マイクロプロセッサーはそれぞれの数字を別々に（最高でも一度に数個の数を）処理する必要がありますが、FPGAをプログラミングすれば、多くの演算を同時に（並行して）行えるという点です。128の要素を持つ行列があるとしたら、128の演算（パイプライン）を構築でき、同時にすべての演算を実行することでパフォーマンスを高めて消費電力を抑えられます。

ASICやASSPが最適な選択でないこともよくあるのです！

第2章

FPGAに含まれるもの

本章の内容

- プログラマブル・ファブリックとI/Oを探る
- スケールアップについて学習する
- ハードIPと統合CPUの組み込み
- 近年の設計フローに取り組む

本章を読んでいるということは恐らく、FPGAについてご存じなのでしょう。ただし、FPGAについて学んでから時間が経っているなら、ここ数年でプログラマブル論理ゲートアレイ以上の存在に進化したFPGAの姿に驚くことでしょう。近年のFPGAは良く使われる機能を簡単に実現できるハードウェアを組み込んでいます。本章ではFPGAとは何か、今後のスケールアップに加えて、設計フローについても解説します。

基本要素ープログラマブル・ロジックとI/O

フィールドプログラマブル・ゲート・アレイ（FPGA）という名前のとおり、本質的にはFPGAは単にいくつかの論理ゲートとI/O回路を含むICです。I/O回路はソースからデータを取り込み、他のシステムやサブシステムへデータを出力します。

第1章では、電気システムの構成要素である論理ゲート、配線、レジスターについて説明しました。電気システムの核となるのは、表面にエッチングされた配線とトランジスターを備えたフラットな長方形のシリコンです。これらのシリコンは集積回路（IC）として知られています。

トランジスターとは通常シリコンで作られる半導体デバイスです。トランジスターは電気信号の切り替え、および/または増幅に使用され、回路と接続するためのコネクター（または端子）を3つ以上備えています。トランジスターは次のようにして電気信号をスイッチします。2つの端子間に電気的な位置エネルギー（電圧）を加えておき、別の電位を3つ目の端子に加えることで、最初の2つの端子間で電流がトランジスターの片端からもう片端へ流れます。この電位が除去されると回路の電流は止まります。またトランジスターは入力よりも出力が大きくなるように電圧または電流を加えることで、入力電力を増幅させることができます。

トランジスターは配管で使用されるばね仕掛けのバルブに似ています。バルブに力を加えるとパイプ（この例では配線）を通して水が流れます。バルブに対する力を除去すると、バルブが閉じ、パイプへの水の流れが止まります。トランジスターも同じです。トランジスターに電位（力）を加えると、トランジスターからもう一方の端へ、ひいては回路を接続する配線まで電気が流れます。この電位を除去すると、電気は流れなくなります。

ブール代数は入力値「真（true）」または「偽（false）」、もしくは数字の「1」または「0」で表される数値演算です。論理ゲートはさまざまなブール代数演算を入力値「0」または「1」を使用して実行するデバイスです。トランジスターは電位を加えた

トランジスターの歴史

1947年、トランジスターはジョン・バーディーン、ウォルター・ブラッテン、ウィリアム・ショックレーにより開発されました。トランジスターが優れた発明である理由は、固体電子工学と集積回路の時代を招いたことです。トランジスターはそれまでトランジスターの代わりに使用されていた真空管よりも極めて小さく、電力の消費も少なかったのです。サイズの縮小によりデバイスの小型化が可能になりました。そのおかげで、現在、フランス料理とベトナム料理を融合させたお気に入りのレストランまでの運転ルートをナビで案内してもらいながら、YouTubeで最新のネコのビデオを再生できるスマートフォンを外出先に携帯できるようになったのです！オタク気分を味わいたいなら、トランジスターという名前はジョン・R・ピアースが「transfer（転送）」と「resistor（レジスター）」を組み合わせて作った造語であると説明して、パーティーで友達を驚かせましょう。この豆知識があれば、「ジェパディ！（アメリカのクイズ番組）」で賞金をゲットできるかもしれません。

り、除去したりすることで電気信号を切り替えるため、「AND」、「OR」、「NOT」（第1章のブール代数演算を参照）といったブール代数演算のいずれかを実行する論理ゲートをいくつかのトランジスターで構成することができます。

FPGAのコアはこれら論理ゲートの固まりと集積回路にエッチングされた配線が、再構成可能となったものです。第1章では、FPGAをおもちゃのブロックに例えて説明しています。よろしければご覧ください。

つまりFPGAは最も簡単に言えば、持ち主（FGPAの場合はプログラマー）が好きに並べ替えられる数多くの色とりどりの四角形が、長方形の台に並べられたようなものです。

これまで、設計者が描くFPGAの使用方法はシンプルなコンピューターのインターフェイスや基本的な論理機能の作成など、シンプルなものばかりでした。設計者は、フィールド・プログ

ムーアの法則

ムーアの法則は1965年4月19日発行の『Electronics Magazine』に掲載された論文、「Cramming More Components onto Integrated Circuits（集積回路により多くのコンポーネントを詰め込む）」に由来します。この記事でムーアは、ウエハー（集積回路が構築されるフラットなシリコンディスク）は変わらない一方で、回路の複雑性は2年おきに2倍になると正確に予測しました。1970年代の終わりには、ムーアの法則で最も有名な公式は最も複雑なチップ上のトランジスター数の限界として知られるようになりました。素晴らしいことに、ムーアの予測は最初の発表から50年ほど経った今でも当たっているのです！

簡単に言うと、ムーアの法則は、集積回路に組み込む部品のサイズが18〜24カ月おきに30%縮小できるため、トランジスターの数を2倍に増やせるというものです。ただし、ムーアの法則は物事を複雑で困難にします。例えば、これらの配線とトランジスターすべてをより面積の小さいシリコンのエリアに追加すると、プロセス後のエッチングされたシリコンウエハーでは元の回路設計の配置の一貫性を維持するのが非常に難しくなります。できあがった回路の配置像には、設計よりも線の幅が広い/狭いなどのおかしな点や、角が丸まっているなどのゆがみがウエハー上に見られます。

ラミング・インターフェイス用のデバイス以上の使い方をしてこなかったのです。

スケールアップ

設計者は、回路をエッチングして小さなシリコンウエハー上に数百万もの配線やトランジスターを形成する際に発生し得る問題を排除します。

数百万、数十億ものトランジスターでチップ設計をスケールアップする際に発生し得るもう1つの問題は、トランジスターがとても小さいため、ピンチオフ（電位を絶つこと）が難しくなることでした。チップの設計がウエハー上のトランジスターの数を倍増させるにつれ、設計はより複雑化し、トランジスターの漏れ電流も増えるため、回路が動作していない場合でも電力を消費してしまいます。また、トランジスターが弱くなり、チップの処理が遅くなるのを防ぐための作業が増えてしまいます。

スケールアップの限界が近いのでしょうか。設計者はすでに多大な電力をチップ上の小さなエリアに詰め込んでおり、チップが自身の配線を溶かしてしまうことすら起こり得ます！設計者はトランジスターの設計を平面の配置からFinFETへ抜本的に変更する段階に来ているのです。FinFETはトランジスターが占める表面の面積を削減するためにシリコンウエハー上に構築された非平面のマルチゲート電界効果トランジスター（Field-Effect Transistor：FET）です。FinFETの名称はトランジスターの伝導チャネルを形成する薄いシリコン、「fin」から来ています。

チップ設計者の今後はどうなるのでしょう。つい最近まで、1つのチップが抱えるトランジスターの数は20ほどでした。今後数年で、チップに構築されるトランジスターの数は数百億にもなるでしょう。このようなチップの設計が可能になる前、設計者の間では最先端のトランジスター開発や新種の配線、その他多くの変更を余儀なくされると考えられていました。チップ設計は前代未聞の領域に入ろうとしています。チップは自身で診断や補足、修正を行えるようになるのです。ムーアの法則を今後も試し続けることができるなんて、チップ設計者にとっては楽しい時代ですね！

ムーアの法則の適用は集積回路にとって大成功でした。今日の集積回路は数百万、数十億ものトランジスターを備えることができ、高速データ・ネットワーキングや高度な3Dグラフィックスの計算、インターネット上での高解像度のストリーミングや映画再生など、とても複雑な演算を実行することができます！ムーアの法則の対象は集積回路だけではありません。その適用はFGPAにも革新をもたらしました。

ハードIPと統合CPU

現在のところ、同じサイズのハードウェアで同じ設計を実装した場合、FPGAとASICのエネルギー効率と速度は同じであることが多くなっています。これはある程度真実です、なぜならFPGAのベンダーは標準インターフェイスからマイクロコンピューター全体まで、頻繁に必要とされる機能を実行するために事前に決められた大きなハードウェア・ブロックをFPGAに組み込んでいるからです。

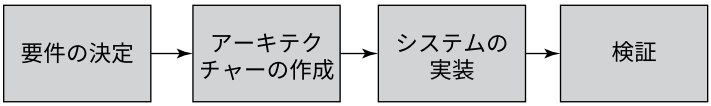
ただし、FPGAがフィールド・プログラマブルであるということは設計変更に対応して同じハードウェアを再プログラミングできるということであり、ASICよりも優れているといえます。ASICを使用した設計では、設計変更の際にはそれまでのハードウェアを廃棄して、設計の変更を反映する新しいハードウェアを構築する必要があります。多くの設計者がASICよりもFPGAを選択するようになりました。FPGAを使用した設計を試してみたいのであれば、組込みハードウェアのメリットを最大限に生かしつつ、FPGAを使用して設計するための設計フローについて読み進めてください。

近年のFPGA設計フロー

FPGAのデザインは、誰かがコピーすることを前提としたシステムの技術的な基本設計図となることを意味し、リファレンス・デザインから始まることが多くなっています。リファレンス・デザインにはシステムの重要な要素が含まれています。リファレンス・デザインは通常、販売サポート業務の一環としてアプリケーション・エンジニアによって行われますが、顧客の要望は変わりました。リファレンス・デザインは販売用のツールではなく、製品そのものになりつつあるのです！

機能ブロック図の作成

では、システム設計フローはどうでしょう。図2-1は、システム設計フローの概要を表す簡易的なブロック図を示しています。

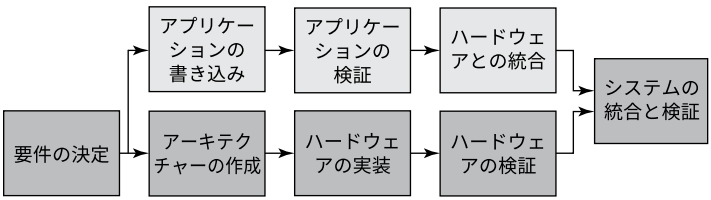


提供：Intel Corporation

図 2-1： システム設計フロー

システム設計フローはあなたが思い描くとおりになります。まず、要件を定義し、定義するシステムのアーキテクチャーを作成します。ここでは、設計を実装するうえで必要なコンポーネントを定義します。次に、綿密に計画したアーキテクチャーを使用してシステムを実装します。最後に、システムがすべての要件を満たしていることを確認します。

図2-1では、簡単なシステム設計フローを示しています。アーキテクチャーの作成とシステムの実装が作業を行うポイントです。ここでシステムのアーキテクチャーがどのように見えるかを定義し、システム設計の実装に必要なハードウェア・アプリケーションとソフトウェア・アプリケーションを構築します。また、要件の決定から検証までの手順をソフトウェア・アプリケーション・フローと呼ばれる別々の手順に分けることもできます。図2-2は図2-1のシステム設計フローにソフトウェア・アプリケーション・フローに加えます。



提供：Intel Corporation

図 2-2： ソフトウェア・アプリケーション・フローを加えたシステム設計フロー

要件の決定からシステムの統合と検証までの色が薄いブロックは設計のアプリケーション・フローを表します。この手順では、アプリケーションのソフトウェアを書き込み、検証して、ハードウ

ェアに統合します。アプリケーションをハードウェアに統合した後、システムが統合され、設計要件を満たしていることを確認します。

アプリケーション・フローはすべて、システムを実装するために必要なアプリケーションを開発することです。設計者は、システムが展開されるアプリケーションの種類（例えば自動車や通信など）に基づいて、さまざまなプラットフォームで、異なるシステムの実行方法を検討しなければならないことが多くあります。また、これらのシステム向けに開発されたアプリケーションが共通の機能を実行でき、お互いに連携できるよう、異なるアプリケーション・ドメインがそれぞれ確立したソフトウェア標準やハードウェア標準を持っていることが多くなっています。

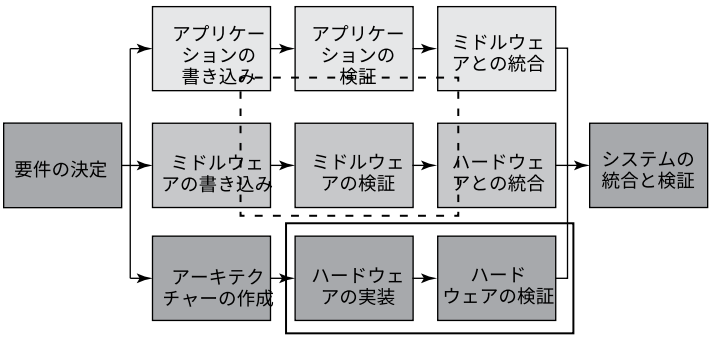
例えば、Android*ベースのスマートフォンを考えてみてください。Android*のオペレーティング・システムには、Android*向けに開発されたすべてのアプリケーションで使用できる共通機能が含まれており、カメラへのアクセスやアプリケーション間でデータを共有する際に、そのような共通機能のプラットフォーム規格を使用します。設計者はしばしば、一般にミドルウェアと呼ばれるものをアプリケーションに備えます。ミドルウェアは製品のコア機能を実行しないソフトウェア層ですが、業界規格またはプロトコルを実行する層を提供します。設計者は、自分の設計するアプリケーションを特定の規格または開発プラットフォーム（Android*やiOSなど）から分類するためにミドルウェアを作成します。ミドルウェアはさまざまなアプリケーションで再利用されることがよくあります。また、ミドルウェアをアプリケーションに追加することで、将来の規格やプラットフォームを簡単にアプリケーションに統合することができます。図2-3では、ミドルウェアの開発を組み込んだシステム設計フローを示しています。

提供：Intel Corporation

図 2-3： ミドルウェアを加えたシステム設計フロー

機能ブロックを既存のIPと置き換える

ブロック図を見ると、アプリケーションを構築し、システムに組み込む際に、あなたの仕事があなたのために省略されているようです。しかし、FPGAのメーカーは、多くのシステムが同じ機能を必要としていることを何年もかけて学習してきました。ネットワーク・データI/O、グラフィックス処理、マイクロプロセッサーなどの機能は共通して必要なため、システム設計者それぞれがこういったコンポーネントを設計し、構築するというのは理にかないません。このような機能は難しい設定などは一切なしで使用できるようにすべきです。近年、FPGAの製造者はこのような共通機能または設計成果物（IP）を製品に搭載するようになりました。このIPはチップに組み込まれたハードウェアやユーザーに提供されるソフトウェア、またはユーザーがプログラマブルな論理に落としこめるハードウェア設計（FPGAのみ）という形で提供されます。今では、自身で作業をする必要なく、ブロック図の中の一部を既存のIPで置き換えることができます。図2-4は既存のIPが設計図のどこに当てはまるのかを説明しています。



提供：Intel Corporation

図 2-4： 機能ブロックを既存のIPで置き換え

実線の長方形はハードウェアやプログラマブル論理IPを使用した場合の影響を示しています。ハードウェアの実装と検証の手順の多くは自動で実行されます。また、図2-4の点線の長方形は設計図において既存のIPが当てはまる場所を示しています。ここでは、アプリケーションやミドルウェアの書き込み作業の一部を実装済みの機能で置き換えます。

バグの中のバグ

語の起源がお好きな方なら、「バグ」という言葉を初めてソフトウェアで使用したのはコンピューターの先駆者であるグレース・ホッパーであることを知り興味をそそられることでしょう。1947年、電子機械コンピューターの中継装置に蛾が引っかかっているのを目にした彼女は、プログラム実行時の誤作動を「バグ（虫）」と呼ぶようになったのです！

足りないブロックのコード化

既存のIPに共通の機能を実装してもナビゲーション・システムのGPSデータへのアクセスなどが関の山です。FPGAにおける他のフィールド・プログラミング作業は設計者の手にかかっています。最終的には、用途にピッタリ合うようにしたいのではないでしょうか？

現代的なFPGAのプログラミングは思っているよりもずっと簡単です。FPGAのプログラミング手順には、自分で設計したい設計ブロックの特定、ハイレベルまたはハードウェア記述言語（Hardware Description Language：HDL）の選択、テキストエディターへのコードの書き込み、設計の統合（後ほど詳述）、設計の配置とルーティング、FPGAそのものに対する設計の読み込みが含まれます。FPGAに設計を読み込んだ後、機能のエラーを修正するためのデバッグサイクルが必要になる場合があります。

ソフトウェアやハードウェアの開発において、バグという技術用語を耳にする機会があるかと思います。バグとは、コンピューターのソフトウェアで発生する不可解な不具合で、不正確または予期しない結果を生み出します。デバッグという用語は必要な機能に沿って設計全体が稼動するまで不具合を除去することを意味します。

設計が満足に稼動することを確認したら、プログラムをドキュメントにして顧客に発送します。

では、ライブラリーからインポートできないブロックはどのようにプログラミングするのでしょう。FPGAが開発された当時はハードウェア記述言語（HDL）しか選択肢がなく、名前の

とおり、ブロックをハードウェア用語（配線とゲート）で記述していました。これにより、FPGAのプログラミングはハードウェア・エンジニアか意思の固い愛好家に限られていました。しかし今日では、OpenCLのようなハイレベルのソフトウェア・プログラミング・ツールにより、ソフトウェア設計者が特別なハードウェア・スキルを持つ必要なく、サブシステムの機能を指定し、コードをコンパイルして、普通のFPGA設計フローに取り込めるハードウェア記述ファイルを作成できます。このようなツールにより、ソフトウェア設計者はハードウェア・エンジニアのサポートを最小限（または全くなし）に抑えつつ、ハードウェア・アクセラレーターを作成できるようになり、現代的なFPGAのユーザーを大幅に広げたのです。

より詳細な設計が必要なブロックに関しては、設計者はHDLを使用します。Verilogは、FPGA向けの設計を作成する際に使用される一般的なHDLです。Verilogは一般的に使用されている汎用プログラミング言語、C言語によく似た構文を備えています。ただし、Verilog、VHDL、およびその他のハードウェア記述言語はコンピューターで稼動するプログラムを定義する代わりに、設計者がFPGAで作成したいハードウェア（ゲート、レジスター、配線で相互に接続されたネットワーク）を記述します。Verilogプログラムはシンプルなテキストエディターを使用し、適切な構文で書き込みます。

HDL設計を書き込んだら、次はコンパイルを行います。FPGAのプログラミングでは、統合ツールがHDL設計を入力として捉え、HDLが記述する機能を実行するために構成されたゲート、レジスター、配線のネットワークに変換します。その後、追加プロセスにより、FPGAで使用するゲート、レジスター、配線を選択し、FPGAがパワーアップした際にそれを設定するプログラミング・ファイルを作成します。

HDLコードは指定したFPGAデバイスで使用できる物理ハードウェア要素に直接マッピングされます。マイクロプロセッサーのプログラミングでは、プロセッサーが実行しなくてはならないプロセッサー命令リストにプログラム論理がマッピングされます。論理をシリコンゲートに直接変換して実行できるという機能の違いは大きく、素晴らしいものです。

このプロセスの間、設計ツールはFPGAに組み込み済みの事前に定義されたハードウェア・ブロックであるハードIPの設計と連携することもできます。現代のツールフローでは、使用するIPブロックがハードかソフトか、また、それらを接続する方法を

指定するだけでかまいません。書き込みが必要なのは、IPとして使用できないブロックのHDLコードのみです。

システム設計の検証

コードをコンパイルしたら、FPGAで展開する前にテストします。以前は、よりシンプルでプログラマブルな論理チップの設計が機能するかどうかを、使用して確認するという単純な方法で設計者がテストしていました。しかし、現代のFPGAが複雑になったことから、初期のデバッグツールのような電源を入れて試すといった方法では通用しなくなりました。

FPGA設計のデバッグは通常、シミュレーション環境で行われます。大体想像がつくと思いますが、シミュレーターは、デザインの動作をシミュレートするソフトウェア・アプリケーションです。ただし、シミュレーションは、設計をFPGAに実装する前に個別のレジスターの動作を確認できるソフトウェアを使用して行われます。

コードのデバッグと検証は通常、HDLコードが期待どおりに機能していることを確認するまで繰り返されます。ほとんどの開発者はテストベンチと呼ばれるツール使用してFPGAが実稼動環境に実装された際に機能するかどうかを検証します。テストベンチでは設計に合わせて、FPGAを備えたシステムモデルを形成するよう、ソフトウェアのシミュレーションと実際のハードウェアを組み合わせることも可能です。多くのFPGAは数万、数十万ものゲートを備えており、すべてをテストするのは不可能です。代わりに、テストベンチは設計の重要な部分を占める最も重要なゲートに集中します。シミュレーション環境は特定の領域を分離して、その部分にデバッグ支援機能を追加することができます。

良いソフトウェア・アプリケーションにするには、顧客やエンドユーザーにアプリケーションの定義を説明し、注意や警告などを与える豊富なドキュメントが必要です。FPGAのドキュメント要件は他のマイクロコントローラー・ベースのプログラミングと同じですが、内容は大きく異なります。

システムをFPGAハードウェアにマッピング

最後に、システムのゲートを実装するには合成されたビットデータをFPGAにロードする必要があります。

他のシステムと同様に、ハードウェアが正しければ、設計はバグの修正や機能の強化などを備えるよう進化できます。HDLコードを編集できる機能により、同じ環境での設計、デバッグ、および検証が可能になり、FPGAを使用した市場投入時間の短縮に役立ちます。

システム内で設計を試用する

設計をハードウェアでプログラム化したら、すべてが期待どおりに機能するか確認します。ところで、FPGAにとって機能するとはどういうことなのでしょう。この段階はクロージャーと呼ばれることもあり、どのハードウェア・デバイスにおいても、一定のパフォーマンス条件が期待されます。多くのアプリケーションでは、電力消費量が設計における重要な条件になります。スマートフォンを想像してみてください。スマートフォンには十分なバッテリー寿命を維持できるように厳格な電力要件があります。スマートフォンに電力を大量消費されてしまうと、短時間の使用でバッテリーが切れてしまうからです。また、動作速度も重要な条件の1つです。テストでは、ネット（ゲート間の配線）が時間制限を守るかどうかを確認します。最後に、すべてのクロックとパワーピンがFPGAに接続されていることを確認します。

FPGAの設計ツール環境では、設計を入力し、備えるIPブロックを選択して、設計を実際にFPGAに存在するハードウェア要素に変換できます。その後、まだ設計がソフトウェアにあり、テストを行いやすい内に、設計が求められる動作速度で期待どおりに機能するかを検証でき、消費電力を予測することすら可能です。

その後、奇跡の瞬間がやってきます。テスト済みのデザインをプロトタイプ・ボード上の対象のFPGAにロードし、システムをパワーアップさせ、すべてが期待どおりに機能しているか検証できます。これで要件にピッタリ合ったカスタム・ハードウェアの用意ができました。ASICでは、初のサンプルチップを受け取るのも数カ月先になってしまうところでしょう。

第3章

FPGAシステム

本章の内容

- 機能ブロックとしてのFPGAを理解する
- SoC FPGAを使用してシステムを実現する

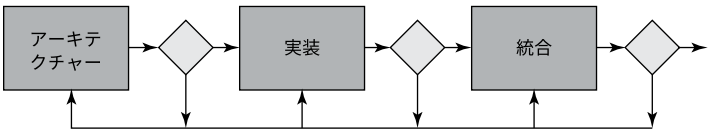
本章では、FPGAが実社会でどのように使用されているかを説明します。FPGAはシステムの機能ブロックであり、大きくなるにつれ、デジタルシステム全体を実現して、システム・オン・チップ（SoC）を形成します。

本章では、電子システムのコンセプトを探り、極めて複雑な高性能自動車でのSoC FPGAについて説明します。

システム設計でのFPGA

このセクションではシステムの設計プロセスと、FPGAがいかに重要な役割を担っているかを深く掘り下げます（基本については第2章で部分的に解説）。

図3-1では、従来のシステム設計モデルを示しています。各ブロック間のひし形は、プロセス内での意思決定ポイントを示しています。



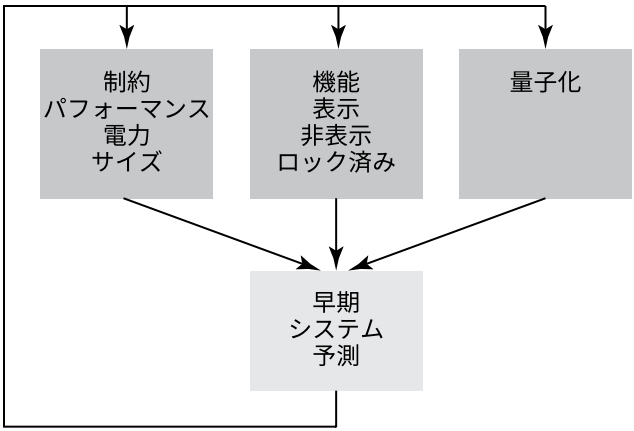
提供：Intel Corporation

図3-1： システム設計における意思決定ポイント

以下のような質問を投げかけるとき、それがシステム設計における意思決定ポイントです。

- **システムは何を実現しなければならないか？**この質問は、要件を決める段階で出てくる、最も根本的な質問です。多くの場合プロダクト・マネージャーが顧客と協力して、この質問に対する答えを導き、それから、要件を集める段階に進みます。
- **今ある設計を変更して使用できるか？**現在のシステム設計に変更を加えることで、システムの要件を満たせることはよくあります。
- **システムのどれくらいをソフトウェアで実現できるか？**これは設計および実装の段階で問われる重要な質問です。ソフトウェアから切り離す量によって、使用できるハードウェアの種類が決まってくるからです。ソフトウェアのプログラミングでは、FPGAとマイクロコントローラーを使用できます。
- **既製のハードウェアはどれくらい購入できるか？**システム設計の機能ブロックがすでに市販のハードウェア・デバイス（既製品）に実装されていることもよくあります。この場合、ソフトウェアを実装したり、ハードウェアを設計したりするより、既製品のデバイスを買ったり、IPとしてライセンスで購入した方が安いかもしれません。
- **動作するか？**これは統合段階での根本的な質問であり、システムを展開する前に、「動作する」と答えなければならないでしょう。答えが「機能しない」である場合、機能するまで設計と実装を反復する必要があります。

図3-2では、要件を決める際に設計者がパフォーマンス、消費電力、サイズといった制約を考慮しなければならないことを示しています。また、どの機能を明示するか、明示しないか、または固定するか、も含めてシステムの機能は重要となります。最後に、設計者は設計を試験運用します。このプロセスにより早期システム予測が行われ、システムの実サイズと範囲、および実装に必要な条件が示されます。



提供：Intel Corporation

図 3-2： システム要件の決定

FPGAの設計プロセスは反復式であり、システムの構想からスタートして、その構想を磨き上げ、トランザクションを定義します。トランザクションには入力、処理、出力が含まれます。トランザクションは銀行の取引のようなものであり、窓口でお金を渡すと、窓口担当がお金を受け取り、お金を口座に追加します。要するに、トランザクションは、システムの2つのコンポーネント間で情報を共有する場所なのです。

トランザクションを定義したら実装して、機能するかどうか、また、最初に設定した機能や制約を満たすかどうかを確認します。これもハードウェアやソフトウェアで実行する機能の決定、および既存のIPで実行できるのか、新しく設計しなければならないかを決定する反復プロセスです。

FPGAを使用した自動車用電子システム

このセクションでは実社会におけるシステムの用途を探ります。ひょっとすると、あなたも今朝会社に来るまでにこのシステムを運転したかもしれません。SoC（その多くはFPGAである可能性あり）が現代の自動車で使用される方法を見てみましょう。

動力伝達装置

自動車の動力伝達装置を構成するコンポーネントと、それが規制や安全性、コスト、および機能的なプレッシャーの下で電子化された過程を考えてみましょう。

- **エンジン：**エンジンに搭載されている電子装置は、パワー要求、排気ガス、スムーズネス、始動サイクル、およびストラテジーに基づいて燃料、点火装置、およびバルブを制御します。
- **トランスミッション：**現代のトランスミッションはギア比、変速シーケンス、速度に基づいた信号、パワー要求、および１分間当たりの回転数（RPM）で測定したエンジン回転数を制御するため、電子システムを搭載しています。
- **ブレーキ：**安全のため、電子システムはブレーキペダルだけではなく、制動力も制御します。
- **ステアリング：**高性能自動車には、たくさんの入力に基づいてステアリング比、フィードバック、角度を制御する洗練されたパワーステアリング機能が搭載されており、電子装置はステアリングの制御を引き継ぐことができます。
- **タイヤ：**近年、タイヤの圧力をモニターすることで、運転者にいつタイヤに空気を入れるかを知らせ、燃費向上とタイヤの寿命を高めています。

インフォテインメント

インフォテインメントとは、自動車の情報娯楽システムを意味するシャレた言葉です。多くの自動車は以下のような洗練された電子インフォテインメント機能を備えています。

- **表示と制御：**最近の自動車には、電子制御による速度計などの表示装置が搭載されています。
- **エンターテインメント：**現代の自動車はデジタルAM/FMラジオや衛星ラジオ、CD、および音楽ライブラリーをすべて保存できるデジタル・オーディオ・プレーヤーなどの先進機能を備えています。また、長距離のドライブでも子供を飽きさせないデジタル・ビデオ・システムを備えているものもあります！

- **快適性：**今では、運転手と乗客それぞれが照明やマルチゾーン車内温度調節システムにアクセスでき、他の乗員に影響を与えることなく好きなように温度を上げたり下げたりできます。
- **アクセス制御：**現代の自動車にはパワーロックやパワードア、パワーウィンドウ、およびセキュリティー・システムが備わっています。また、運転中に子供がドアや窓を開けられないようにウィンドウロックやドアロックなどの一般的な安全装置もついています。
- **受動安全性：**安全装置は自動車のどこに何人が乗っているかを認識し、衝突しそうになると適切な対策を行います。

運転補助

運転補助機能は、最近の自動車設計に搭載されているものの中でも最も素晴らしいテクノロジーをいくつか搭載しています。これにより、自動車はさらに安全になりました！運転補助システムには以下が含まれます。

- **ライト、後方確認、車線逸脱、衝突回避機能：**自動車は、最先端のライトシステム、表示機器、および車線を逸脱した際や他の車両または物に衝突しそうな際に運転手に警告を与えるためのヘッドアップ・ディスプレイを備えていることが多々あります。
- **カメラ、レーザー、レーダーなどのセンサー：**これらのセンサーは運転手が後方発進や車線変更の際に死角を確認するために使用され、事故の確率を大幅に減らします。

FPGAの重要性

今日、ほとんどの自動車システムはセンサー感知や何らかの動きがあった時点で作動する、コストが低いマイクロコントローラーに頼っています。自動車設計のトレンドはシステムの統合であり、システムをより自律させることです。システムが洗練されるにつれ、処理やメモリーの要件は跳ね上がりました。カルマンフィルターを使用したセンサー融合について考えてみましょう。

センサー・フュージョンは、異なるセンサーからのデータを組み合わせたシステムです。例えば、ステレオカメラは、少し視点が異なった2つのカメラによる二次元画像で、奥行情報を求めることができます。カルマンフィルターは、ノイズ（不規則変動）を含むさまざまな種類のセンサーを長時間観察することで得た一連の測定結果を使用するアルゴリズムであり、1つの測定結果のみに基づいた予測よりも正確な予測を生成します。カルマンフィルターは通常、ガイダンスやナビゲーション、車両制御で使用されます。

自動車システムが統合される際、マイクロコントローラーはSoCの実装に併合されます。これらのシステムがよりスマートになり、自律するにつれて、SoCはマルチコア・プロセッサー/DSPクラスターに進化していきます。爆発的に増えるモデル数やモデル年内の変更、バス・アーキテクチャーでの進化、およびセキュリティー強化に対する継続的な要求をコントロールするため、設計上の課題や頻繁な更新要求に対する唯一の実行可能な解決策であるSoC FPGAがトレンドになりつつあります。

自動車は、システムがより電子装置に依存していること、また、電子装置がより複雑になり、急速に変化していること、さらに、モデル年内でも変更可能なSoCへのニーズが高まっていることを示す一例に過ぎません。同じことは、航空機や電車、電力網から家電まで、複雑な動作を伴う幅広い製品に見られます。そう、トースターすらもこのパターンを備えているのです。

第4章

今後の展開：ヘテロジニアス・コンピューティングとOpenCL

本章の内容

- ヘテロジニアス・コンピューティングについて学ぶ
- OpenCLを考察する

FPGAは業界のトレンドに後押しされて、ヘテロジニアス・コンピューティングのパラダイムで大きな役割を担うようになりました。Open Computing Language（OpenCL）はFPGAをヘテロジニアス環境でプログラミングする際に使用される業界標準の開発プラットフォームです。

本章では、ヘテロジニアス・コンピューティングが必要な理由と、それを実行するソフトウェアを作成するための新言語の台頭について説明します。

ヘテロジニアス・コンピューティング

データセンターでの大きなトレンドの1つにマルチコアCPUからヘテロジニアス・コンピューティングへのコンピューティング・アーキテクチャーの移行があります。ヘテロジニアス・コンピューティングとは、特定の処理機能を実行するうえで複数の種類のプロセッサーを使用するシステムを指します。ヘテロジニアス・コンピューティング・システムの例としては、CPU

とグラフィックス・プロセッシング・ユニット（GPU）を使用して3Dグラフィックスの描画をコンピューター上で行うグラフィックス・レンダリング・システムが挙げられます。GPUは3Dシーンの表示と大規模なデータセットにおける演算量の多い処理の実行に特に適しています。CPUはオペレーティング・システムとデータ管理を実行するためにバックグラウンドで使用されます。ヘテロジニアス・コンピューティングは整備されたシステムとして標準になりつつあり、使用するにはいくつかの異なるプロセッサー・アーキテクチャーを組み込む必要があります。

パラレル・コンピューティングとはコンピューターが「大きな問題は小さな問題に細分化して同時に（並列で）解決できる」という原則に基づいて、多くの計算を同時に実行する能力です。パラレル・コンピューティングにはビットレベル、命令レベル、データレベル、タスクベースなど、さまざまな形式があります。また、IBMの通称であるチェス名人「ビッグブルー」のような、ただの高性能コンピューティングの領域ではありません。組み込み電子装置においては消費電力が設計要素になるため、パラレル・コンピューティングはコンピューター・アーキテクチャーにおける主要パラダイムになり、マルチコア・ プロセッサーというかたちが最も一般的になっています。

データ並列処理は複数のプロセッサーにデータを分散し、並列して実行するアイデアに特化しています。マルチコア・プロセッサーはプログラムの複数のインスタンスを各プロセッサーに請け負わせることで、こういった命令を同時に実行します。タスク並列処理はスレッドと呼ばれるコンピューター・コード・ブロックをさまざまなプロセッサーに請け負わせて並行で実行できるようにするプロセッサーに関連しています。

FPGAでOpenCLを使用する理由

ヘテロジニアス・コンピューティングに対するニーズは、新しいハードウェアを利用するための新しいプログラミング言語の台頭につながりました。例として、Apple, Inc.により初めて開発されたOpenCLが挙げられます。OpenCLはヘテロジニアス・プラットフォームで実行するプログラムを書き込むための枠組みで、CPU、GPU、DSP、FPGA、およびその他の種類のプロセッサーで構成されています。OpenCLはカーネル（ハードウェア・

デバイスで実行する機能）を開発するための言語と、メインプログラムがカーネルを制御できるようにするアプリケーション・プログラミング・インターフェイス（Application Programming Interface：API）を備えています。OpenCLはタスクベースおよびデータベースの並列性を使用して並列処理を実現します。

過去10年余りで、プロセッサー・ハードウェア動作周波数がいわゆるに当たり、プロセッサーの周波数が上がらなくなってしまいました。CPUのメーカーは急速に上がり続けるクロック周波数の代わりに、プロセシング・コアをCPUに追加し続け、複数の命令を同時に実行できるように命令セットを強化しており、クロック周波数の高速化を要求することなく、プログラムの実行速度を上げています。また、ソフトウェア企業もスレッドと呼ばれる大量のコンピューター・コードを本当の意味で並行して実行できる並列ソフトウェアの開発に忙しい状況です。スレッドはこれまでのような並列処理まがいの処理（スレッドを異なるコア上で実行するのではなく、オペレーティング・システムによるタイム・スライシングで並列処理のように見せる処理）を行うのではなく、異なるプロセッサー・コア上で実行することもできます。

FPGAには元々並列処理機能が備わっているため、OpenCLの並列処理機能にピッタリです。FPGAは一般的なデータ並列処理またはタスク並列処理の代わりにパイプライン並列処理をサポートします。これにより、各タスクは互いにプッシュプル構成で接続され、ホストの介在なしに各タスクから別のタスクへデータを受け渡すことができます。OpenCLを使用すると、おなじみのC言語でコードを開発できます。その後、OpenCLが提供する追加機能により、コードを普通のソフトウェアとカーネルに分散でき、並行して実行できます。これらのカーネルはFPGAに適用されます。ここではFPGA設計者に必要な低レベルなHDLコーディングを学ぶ必要はありません。一般的に、OpenCLを使用してFPGAのコードを開発することは、ソフトウェア開発者とシステム設計者にとって以下のようなメリットを提供します。

- **シンプルで簡単な開発：**多くのソフトウェア開発者はC言語に精通していますが、低レベルなHDL言語については詳しくありません。OpenCLでは高レベルなプログラミングを維持でき、お持ちのシステムの対象をより多くのソフトウェア開発者に広げることができます。

- **コードのプロファイル：**OpenCLを使用すると、コードをプロファイルして、FPGAでカーネルとして加速されたハードウェアなど、パフォーマンスに影響される部分を特定できます。
- **パフォーマンス：**消費電力当たり性能はシステム設計における究極の目標です。FPGAを使用すると、エネルギー効率の良いソリューションでのハイパフォーマンスのバランスを調整できます。
- **効率性：**FPGAは、きめ細かい並列処理アーキテクチャーを備えており、OpenCLを使用することで、ハイパフォーマンスを実現するために必要な論理のみを生成できます。
- **ヘテロジニアス・システム：**OpenCLを使用すれば、FPGA、CPU、GPU、およびDSPを対象としてカーネルをシームレスに開発でき、本当の意味でヘテロジニアスなシステム設計が可能です。
- **コードの再使用：**ソフトウェア開発における究極の目標はコードの再使用を実現することです。コードの再使用は、ソフトウェア開発者やシステム設計者にとって達成しづらい目標の1つです。OpenCLカーネルは、ポータブルコードを可能にし、1つのプロジェクトから次のプロジェクトへ、異なるファミリーや世代のFPGAを対象にすることができ、コードの寿命を伸ばします。

今日では、OpenCLはテクノロジー団体であるクロノスグループにより開発、管理されています。

多くのFPGAメーカーは、FPGA上でのOpenCL開発向けにソフトウェア開発キット（Software Development Kit：SDK）を提供しています。

第5章

5つのFPGAの用途

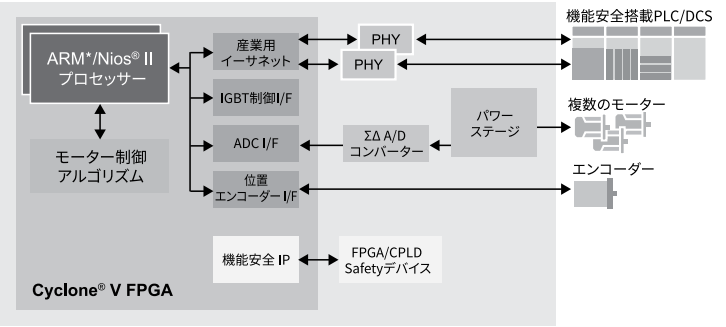
本章の内容

▶ FPGAが実社会でどう使用されるかに目を向ける

FPGAは、FPGAを使用して複雑なシステムを実装する際に膨大な論理ゲートのプログラミングが必要だった昔と比べて大きく進化しました。現代のFPGAにはネットワーク・インターフェイス、メモリーブロック、ARMコアまでさまざまな機能が組み込まれています。ARMコアが組み込まれたインテルのFPGAはこれらの強力なチップが担う役割を評価し、SoC FPGAと呼ばれています。今では、フィールドでプログラマブルできる論理回路がチップ面積の半分以下になることもあります。本章では、さまざまな業界やテクノロジーにおいてFPGAがどのように使用されているかをまとめています。

シングルデバイス・モーター・コントロール

モーターとモーター制御はどのような工業デザインにおいても一般的です。工場や工業地帯に行くと、モーターで動くという共通の特徴を除いて大きく異なるさまざまな機械を目にするでしょう。大抵のモーター制御システムはマイクロコントローラーを使用して設計されています。ただし、マイクロコントローラーは直接トルク制御（Direct Torque Control：DTC）やセンサーレス・ベクトル制御（Sensorless Field Oriented Control：SFOC）など、洗練されたモーター制御アルゴリズムのパフォーマンス・ニーズには性能不足の可能性があります。これまでは、そのような問題を解決するためにDSPが使用されていましたが、ハイパフォーマンスにおける費用効果ではFPGAにかないません。柔軟で、拡張可能なハイパフォーマンス・モーター制御システムを単一のSoC FPGAに構築することもできます（図5-1参照）。



提供：Intel Corporation

図 5-1： モーター制御

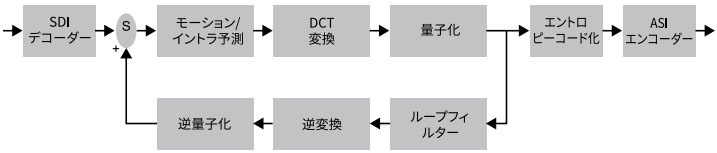
SoC FPGAを別々に制御された2つのDCモーターとシンプルな光フィードバック・システムを備えたモーター制御モジュールに接続する例について考えてみましょう。SoC FPGAにはフィードバックを管理し、信号を制御することで2つのモーターを別々に動かすことが可能な組み込みプロセッサーが搭載されています。プロセッサーはフィードバック・システムからデータを読み込み、モーターの動きを同期するとともに回転速度を制御するアルゴリズムを実行します。SoC FPGAを使用することで、他のモーター制御装置で機能するように簡単にカスタマイズできる独自のIPを構築できます。モーター制御にマイクロコントローラーではなくSoC FPGAを使用するメリットは以下のとおりです。

- **システムの統合：**産業ネットワーク、安全装置、パワーステージ・インターフェイス、およびDSP制御アルゴリズムを単一のデバイスに統合することで部品が減り、その結果、材料費の減少や必要電力の低下につながり、信頼性に関する課題も少なくなります。
- **拡張可能なパフォーマンス：**すべての製品ラインに対し、単一の拡張可能なプラットフォームを使用できます。SoC FPGAは、より高速かつ高度な制御ループにより効率を上げ、マシンの寿命を延ばすことができ、ハイパフォーマンスを達成できます。
- **機能安全：**オートメーションが危険な可能性がある装置を稼動する役目を担う機会が多くなるにつれ、監督官庁は害が及ばないように保証する機械制御の電子装置を要求するようになりました。SoC FPGAと正しい設計フローを使用すれば、これらの政府や業界における安全規制に準拠するための時間と手間を省くことができます。

テレビ放送

テレビ放送はシリアル・デジタル・インターフェイス（Serial Digital Interface：SDI）規格を使用して、圧縮されていないデジタルビデオを75オームの同軸ケーブル（ケーブル/衛星受信機またはアンテナとテレビを接続しているものと同じ）で送信します。ビデオ画像の改善により、規格の容量は増やさざるを得なくなりました。最新の規格は12-Gbps（12G）-SDIと呼ばれ、4K ultraHDシグナルをスタジオ中に張り巡らすことができます。これらさまざまな変更により、FPGAの力を発揮できる分野がまた広がりました！FPGAソリューションは同じトランシーバー上で5つのSDIレートすべて（SD SDI、HD SDI、3G-SDI、6G-SDI、および12G-SDI）で機能するコア・トランシーバーを備えています。

また、スタジオでもいろいろなものが変化しました。新しいデジタル技術により、ビデオストリームの編集、映像の質の改善や修正、ケーブルや衛星回線で送信する画像の圧縮が容易になったのです。最新の圧縮規格、H.265（「High-Efficiency Video CoDec（高効率ビデオ・コーディング）」とも呼ばれる）は映画やテレビ番組のエンコードに必要なビット数を大幅に削減します。しかしこれには膨大なコンピューティングが必要とされます。装置ベンダーの多くは、急速な進化へのプレッシャーに対応しつつ、SoCにパワーを与える最高のソリューションはFPGAであると考えています。（図 5-2参照）。



提供：Intel Corporation

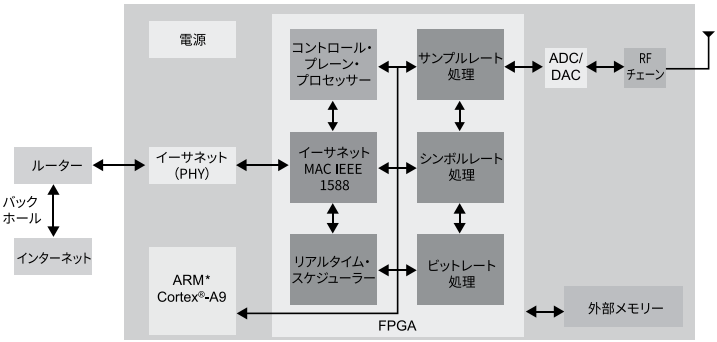
図 5-2： 放送

ワイヤレスデータ

4Gワイヤレス・テクノロジーの出現以上に私たちの生活と仕事を変えたものはありません。電話を持ち運び、ただ通話するだけでなく、外出先からウェブを閲覧したり、近況を投稿できるようになったのです！最新のテクノロジーは4G通信規格から5Gへ移行しています。

ネットワークを拡大し、アップグレードしながら経費を削減しようと努力している基地局や携帯電話会社の主なニーズは、フォームファクターが拡張可能で、消費電力が低く、コストが安くて、プログラム可能であることです。メーカーも売れる製品をリリースし競争力を高めるために、生産性を上げ差別化にかかる時間を短縮する方法を探しています。

今ではFPGAに高度なネットワーク用に作り込まれた低レイテンシーIPと生産性向上のための高度なツールが備わっています。これによりメーカーはFPGAのパフォーマンス、電力、価格、および生産性におけるメリットを活用して、ワイヤレス・インフラストラクチャーの基本的な設計ではなく、製品の差別化に集中できるようになりました。(図 5-3 参照）。



提供：Intel Corporation

図 5-3： ワイヤレス・インフラストラクチャー

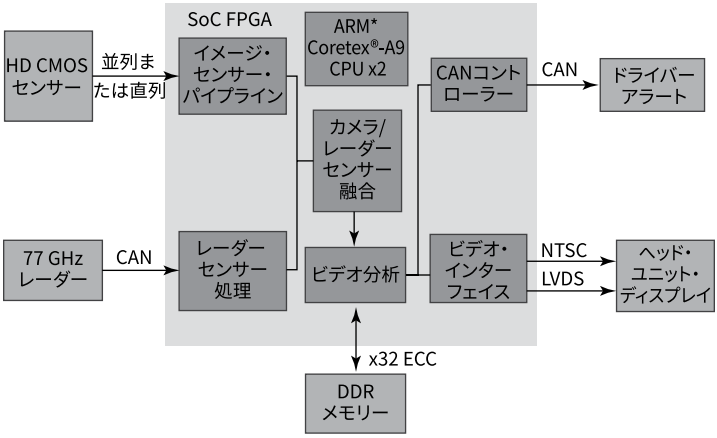
自動車用運転支援カメラ

自動車業界で最も成長した分野の1つに、テクノロジー機能の爆発的普及があります。安価な自動車ですらナビゲーション・システムやビデオ・エンターテインメント・システム、カメラといったオシャレなガジェットが備わっています。

運転支援とバックカメラは最も重要な安全性を高めるためのイノベーションで自動車をこれまでになく安全なものにするために貢献しました。バックしているときでも車の後ろに何もないことを確認できる安心感は何物にも代えられません。

前方カメラシステムは高速ビデオ処理、複雑なセンサー・フュージョン、リアルタイム・データ分析で構成されており、これにより、自動車は、運転手がうとうとして車線を逸脱してしまった場合などに補正動作を実行できます。前方カメラはレーダーやレーザーセンサーなど、さまざまなセンサーと統合することで機能します。センサーのデータ提供方法はそれぞれ異なっており、複数アーキテクチャー向けの設計における課題となっています。

従来のDSPプロセッサーまたはマイクロコントローラーはリアルタイムでビデオ処理や分析を同時に行うパワーを備えていません。また、カメラが暗いところも明るいところも平等に見えるようにするうえで必要なHDR（高ダイナミック・レンジ：High Dynamic Range）は、正確なビデオ分析に欠かせません。HDR処理は従来の非HDRカメラに比べてビデオ信号処理力が3倍にもなり、そのパフォーマンス要件は最も高価なDSPでない限り及びません。DSPやマイクロコントローラーの代わりとして、カメラシステム全体を単一の安価なSoC FPGAに統合できます。また、FPGAロジックの使用、およびSoC FPGAのハード・プロセッサー・システムで稼動しているソフトウェア・アルゴリズムの統合によって、ハードウェア並列処理エンジンを開発することでシステム・パフォーマンスを最適化できます。図5-4は自動車ビジョンシステムの一部としてのSoC FPGAの図を示しています。



提供：Intel Corporation

図 5-4： 自動車ビジョンシステムにおけるFPGA

ハイパフォーマンス・コンピューティング

ハイパフォーマンス・コンピューティング（HPC）市場は今日のコンピューティングにおいて最も急速に成長している分野の1つです。これは金融、医療画像、生物科学、軍事、およびFPGAのロジックやメモリーリソースからメリットを得てアプリケーション指定のコプロセッサーを開発できるその他の分野を含め、多くの業界で非常に重要です。例えば、金融市場と取引や予測、価格の計算などのために移動し続ける、気が遠くなるようなデータ量を想像してみてください。これらの取引においては1セントの上昇や下降が重要になるため、高速で正確な浮動小数点演算が必要不可欠になります。

HPCでは、浮動小数点はひと続きの数字またはビットを実数で表す数値表現になります。用途が整数の計算ではじき出す結果よりも正確な結果を得るには、浮動小数点のデータタイプが必要です。浮動小数点演算にはより多くのプロセッサー論理が必要であり、その結果、より多くの電力が必要になります。一般的な浮動小数点の用途には以下が含まれます。

- 高速フーリエ変換（Fourier transform：FFT）
- レーダー
- 生物科学
- 有限インパルス応答（Finite Impulse Response：FIR）フィルター
- 金融オプション取引
- 行列の数値演算（3Dグラフィックスと画像処理で広範囲にわたって使用）
- 分子力学
- 地震探査と医療画像

コプロセッサーはプライマリー・プロセッサーまたは中央処理装置（CPU）の機能を補足するために使用されます。コプロセッサーは通常、浮動小数点演算、信号処理、文字列処理、暗号化、または周辺機器へのI/Oインターフェイシングを処理するために使用されます。コプロセッサーは大きな計算量と伴う処理を行い、コンピューターの核となる機能で使用できるようCPUリソースを開放します。

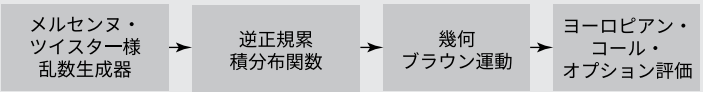
すべてのHPC市場が生産性、パフォーマンス、および電力におけるアドバンテージを提供するためにコプロセッサーを必要とします。

嬉しいニュースと言えば、最新のインテル FPGAはDSP機能だけでなく、浮動小数点ハードウェアも備えているため、プログラマーはFPGAで加速されたサーバーで稼動させる前に、浮動小数点形式から整数形式へとプログラムを変換する必要がありません。この機能は以下のような分野で大きな恩恵をもたらします。

- **データ・アプライアンス：**データベースと金融市場の加速
- **機能：**金融市場向けの乱数発生器、軍事向けの100万ポイントFFTと信号処理アプリケーション
- **アルゴリズム：**SRC CARTE、Impulse、およびAutoESLシステム生成アルゴリズム

ケーススタディー：モンテカルロ ブラックショールズ式

金融市場で最も重要な指標の1つはモンテカルロ・ブラックショールズ式を介したオプション価格の計算です。金融用語でオプションとは、任意の日付前に指定の価格を支払って資産を購入、または売却するための権利を買い手に与えるものの義務は与えない契約のことです。モンテカルロ・ブラックショールズ法は基盤となる株価のランダム・シミュレーションの実行と数百万もの異なるパスにおける期待利得の平均化に基づいています。以下の図はこの方式をグラフィックで示したものです。



提供：Intel Corporation

多くの計算シミュレーションでは、設計者がある種の乱数発生器を使用してデータ入力をシミュレーションし、実際のシステムの乱数度をモデリングします。モンテカルロ・ブラックショールズ式では通常、モデラーがメルセンヌツイスターと呼ばれる乱数発生器を使用します。メルセンヌツイスター乱数発生器はとても速く、質の高い疑似乱数の生成器で、シミュレーションにピッタリです。この一連の乱数は普通に分布された数列を生成するため、逆正規累積分布関数（乱数の分布を指定する確率関数）に送られます。これらの乱数は幾何ブラウン運動（一般的に株価予測に使用されるアルゴリズム）を使用した株価変動のシミュレーションに使用されます。各シミュレーションパスの終わりには、コール・オプション・ペイオフが記録され、ペイオフに期待される値を生成するために平均化されます。FPGAからCPUおよびGPUに移植できるOpenCLコードのおよそ300ラインにアルゴリズム全体を実装できます。FPGAソリューションは以下の図で示すとおり、電力面、パフォーマンス面、および効率面でCPUとGPUの両方をしのぎます。

この図では、消費電力、秒あたりのシミュレーション数、および秒あたりの電力効率という3つの条件においてFPGAをCPUおよ

プラットフォーム	パワーワット (W)	秒あたりのパフォーマンスシミュレーション (Bsims/s)	ワットあたりの効率シミュレーション (Msims/s/W)
CPU	130	0.032	0.0025
GPU	212	10.1	48
FPGA ソリューション	45	12.0	266

提供：Intel Corporation

びGPUと比較しています。FPGAは元々並列性を備えているため、複雑な計算を並列で行える計算に細分化します。FPGAはCPUやGPUよりも多くの演算を並行して行うことができ、実行速度を上げ、電力効率を高めます。この機能は電子システム設計において必要不可欠です。

システム設計で FPGA のパワーと柔軟性を活かす

FPGA のうわさを聞いたエンジニアの方ですか？FPGA について大学で習ったけど、うろ覚えでしょうか。大丈夫！本書を読めば、FPGA の使用について理解した上で判断できるようになります。

- ***FPGA とその仕組みを理解する*** *— FPGA とその仕組みについて学びます*
- ***プログラマブル・ファブリックについて探る*** *— FPGA のプログラマブル・ファブリックとそれを使用して機能を設計、開発する方法を学びます*
- ***FPGA を機能ブロックとしてシステムに統合*** *— システムをシステム・オン・チップ（SoC）として FPGA に併合する方法を探ります*
- ***ヘテロジニアス・コンピューティングの世界へ一歩踏み出す*** *— ヘテロジニアス・コンピューティングがどのようにシステムの開発方法とコミュニケーション方法を変えたかを探ります*

本書から学べること：

- FPGA の現代的な設計フロー
- FPGA を設計に追加するうえでの長所と短所
- 自動車など、多くの業界のシステム設計で FPGA を応用する方法
- ヘテロジニアス・コンピューティングや OpenCL など、システム設計における FPGA の展望

Dummies.comでは、さまざまなビデオや順を追った説明、ハウツー記事を閲覧していただけるだけでなく、商品のご購入も可能です！

Andrew Mooreは15年のソフトウェア設計経験を持つソフトウェア・エンジニアです。**Ron Wilson**はインテルのメッセージング・ストラテジストです。彼はエンジニアリングとテクノロジーのジャーナリズムに携わっていたこともあります。

WILEY

ISBN 978-1-119-45262-1

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.