Xcell journal

87 号 2014

SOLUTIONS FOR A PROGRAMMABLE WORLD

ザイリンクスの SDNet 環境で「Softly」Defined Network を実現

UltraScale アーキテクチャが無線通信アプリケーションを促進

Zynq SoC の割り込み使用方法

低速ソフトウェアを Vivado HLS により高速化

ザイリンクスが Tcl ストアをオープン

MATLAB の支援により、モーター ドライブを Zynq SoC に移行

ページ 28

XILINX
ALL PROGRAMMABLE™

japan.xilinx.com/xcell/

Xcell journal

発行人 Mike Santarini
mike.santarini@xilinx.com
+1-408-626-5981

編集 Jacqueline Damian

アートディレクター Scott Blair

デザイン/制作 Teie, Gelwicks & Associates

日本語版統括 秋山 一雄
kazuo.akiyama@xilinx.com

制作進行 周藤 智子
tomoko.suto@xilinx.com

日本語版 制作・広告 有限会社エイ・シー・シー

Xcell PUBLICATIONS

japan.xilinx.com/xcell/

Xcell Journal 日本語版 86 号

2014 年 6 月 20 日発行

Xilinx, Inc
2100 Logic Drive
San Jose, CA 95124-3400

ザイリンクス株式会社
〒 141-0032
東京都品川区大崎 1-2-2
アートヴィレッジ大崎セントラルタワー 4F

30 年におよぶ革新の集大成、そしてさらに数十年先へ

ザイリンクスは 今年 2 月に、創立 30 周年を迎えました。2008 年に入社した比較的新参者の私でさえ、ザイリンクスとザイリンクス社製デバイスがこのたった 6 年でいかに飛躍的に成長してきたかを知っています。サンノゼの Hamilton Avenue でわずかな人数で会社を立ち上げ、多くの人が常識外れと思っていたテクノロジを、さらに常識外れのビジネス モデルで提供していた設立当初からザイリンクスと共に歩んできた従業員の目には、それがどのように映ったのか想像がつきません。

FPGA とファブレス半導体ビジネス モデルは今でこそ業界の主力になっていますが、1984 年当時は違いました。数年前に私は、Bill Carter にインタビューする機会を得ました。ザイリンクスの中では生きた伝説のような存在であり、業界初の FPGA である XC2064 を設計し、後にザイリンクスの初代 CTO になっています。彼がザイリンクスで経験した多くの素晴らしい出来事の中で最も印象的なものの 1 つが、ザイリンクスの共同創立者 Ross Freeman (FPGA の考案者)、Bernie Vonderschmitt、Jim Barnett との就職面接の話でした。彼らは再プログラム可能なデバイスのアイデアを思いつき、セイコーエプソン社に製造を委託する計画を立てていました。

「私はアーキテクチャを一目見て、難題だと思いました」と Carter は述べています。「当時、シリコンの面積は貴重で、新しい回路構造をインプリメントするには非常に多くのトランジスタが必要でした。しかし、ハードウェアを再プログラムできるという考えは革新的でした」。

ビジネス モデルもその当時としては難題でした。1984 年当時、チップはそれぞれ自社で作製しており、事業参入への最大の障壁は工場を建設するための資金を調達することでした。Carter は、Vonderschmitt が RCA 当時から製造を理解し、さらにセイコーエプソン社の友人 (RCA 時代にシリコン製造の秘訣を教えていた) にチップの製造を依頼できることを思い出しました。

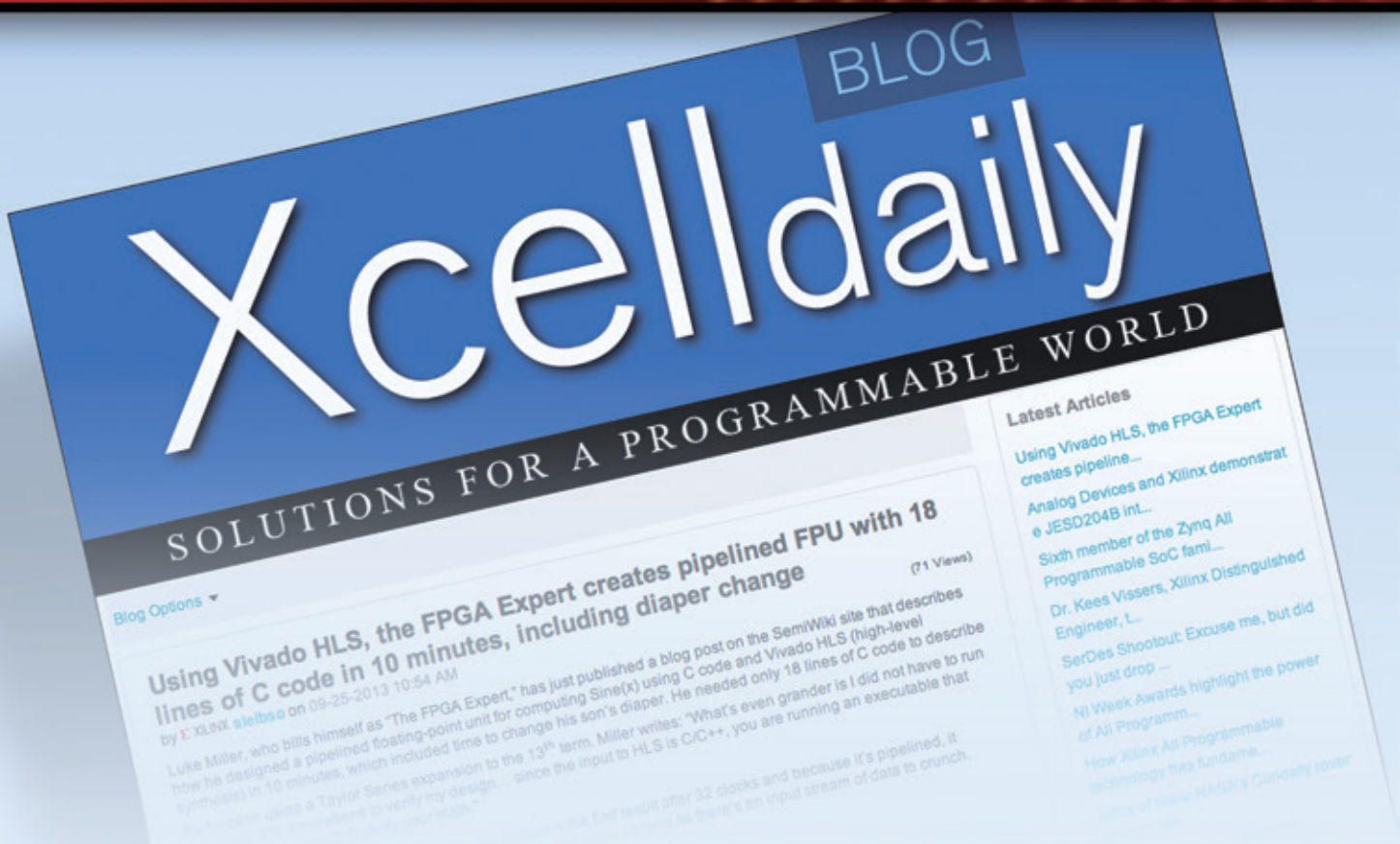
ここで Bill Carter の登場です。当時彼は未成年の家族とローンの支払いを抱え、Z80 の製造で定評のある Zilog 社で堅実に働いていましたが、ザイリンクスでチャンスを掴むために、同社からザイリンクスに転職してきました。その後のことはご存じのとおりです。ザイリンクスは 1985 年に FPGA を市場に送り出し、その数年後 TSMC 社のリーダーシップの下でファブレス モデルがチップ製造の主流になりました。

それから 30 年経ち、私が実感しているのは、多くの人物がやって来ては去り、ザイリンクスの社風を形作ってきましたが、創業当時からのリスクに立ち向かう革新的な精神が今なお健在であるということです。この 6 年、私は、ザイリンクスが革新的なシリコン (7 シリーズ All Programmable FPGA、SoC、および 3D IC) および各種ツール (Vivado® Design Suite) により、28 ナノメートル ノードの競争で一世代先へリードしてきた様子を目撃してきました。ザイリンクスは FPGA 市場初の 20nm UltraScale™ デバイスでさらにこのリードを広げています。また、FinFET 時代の到来と共に、さらに大きな革新の登場も期待できます。このパイオニア精神は、ザイリンクスのお客様がこの 30 年間で当社のチップで築いてきた目覚ましい革新に支えられています。私は、この先数十年間もさらに多くのお客様の革新を目にすることになると確信しています。

編集者注： ザイリンクスおよびファブレス業界の初期の先駆的時代に関心がおありの場合は、Xcell Daily の編集者である Steve Leibson による力作「ザイリンクスとファブレス半導体業界の誕生」(英語) をお勧めします。今後のザイリンクスの革新に関する最新情報をご希望の場合は、26 周年を迎えた Xcell Journal を継続的に購読し、技術記事を寄稿してください。

Mike Santarini
発行人

Xcell Journal を拡充。新たに Daily Blog を追加



ザイリンクスは、数々の受賞歴がある Xcell Journal をさらに拡充し、エキサイティングな Xcell Daily Blog（英文）を始めました。このブログでは、コンテンツを頻繁に更新し、技術者の皆様がザイリンクスの製品とエコシステムの多岐にわたる機能が活用でき、All Programmable システムおよび Smarter System の開発に役立つ情報を提供します。

Recent（最近の記事）

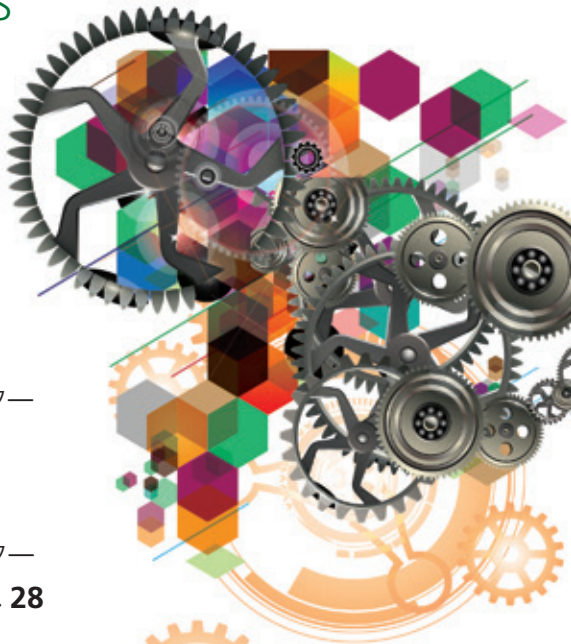
- *Avnet opens registration for International X-fest 2014 training events in nearly 40 cities worldwide*
- *Low-cost Artix-7 50T FPGA Eval Kit now ready to order from Avnet*
- *New Zynq-based RazerCam is one smart industrial machine vision camera with three different sensor options*
- *Video: UltraScale integrated 100G Ethernet MAC passes 970 billion error-free frames overnight*
- *Linux + Zynq + Hardware Image Processing = Fused Driver Vision Enhancement (fDVE) for Tank Drivers*

ブログ : www.forums.xilinx.com/t5/Xcell-Daily/bg-p/Xcell

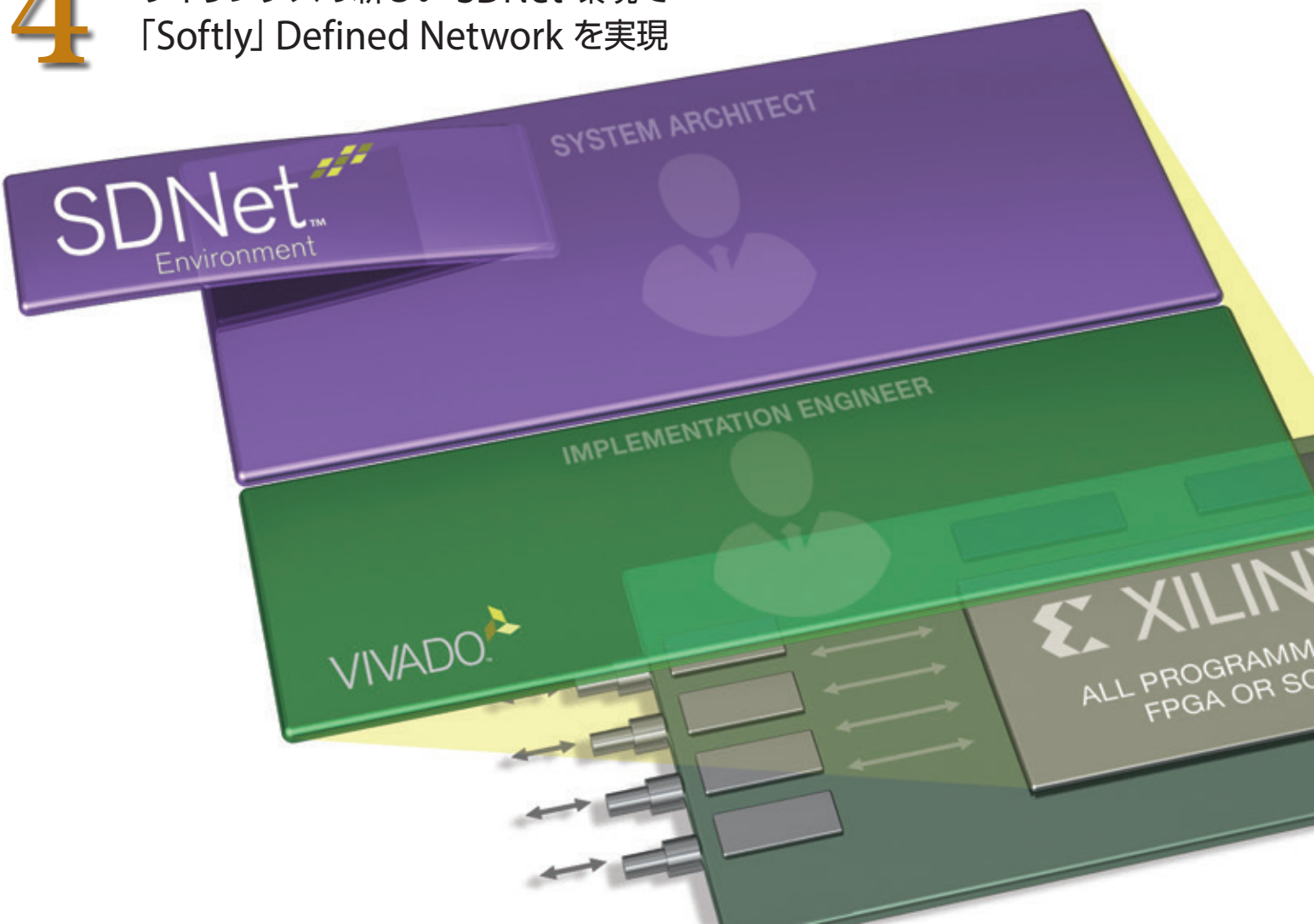
XILINX
ALL PROGRAMMABLE™

VIEWPOINTS

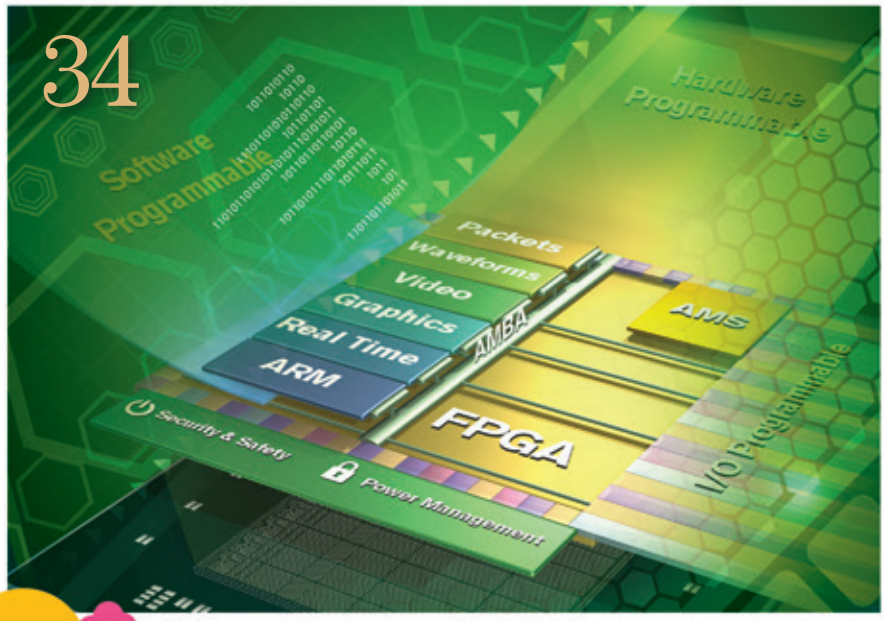
XCELLENCE BY DESIGN APPLICATION FEATURES

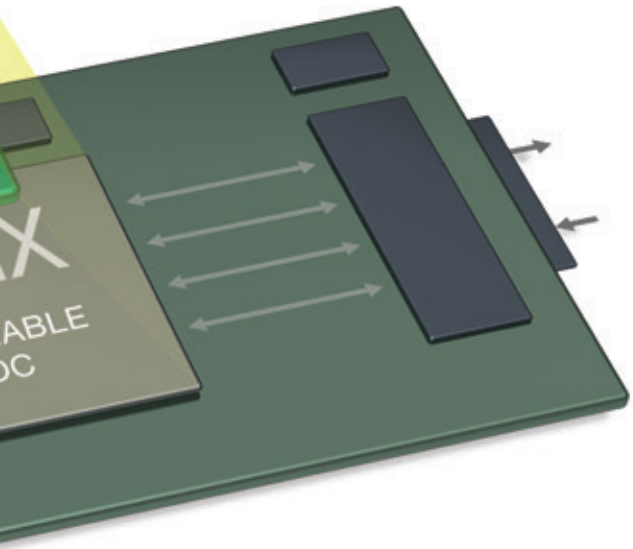


Cover Story



THE XILINX XPERIENCE FEATURES



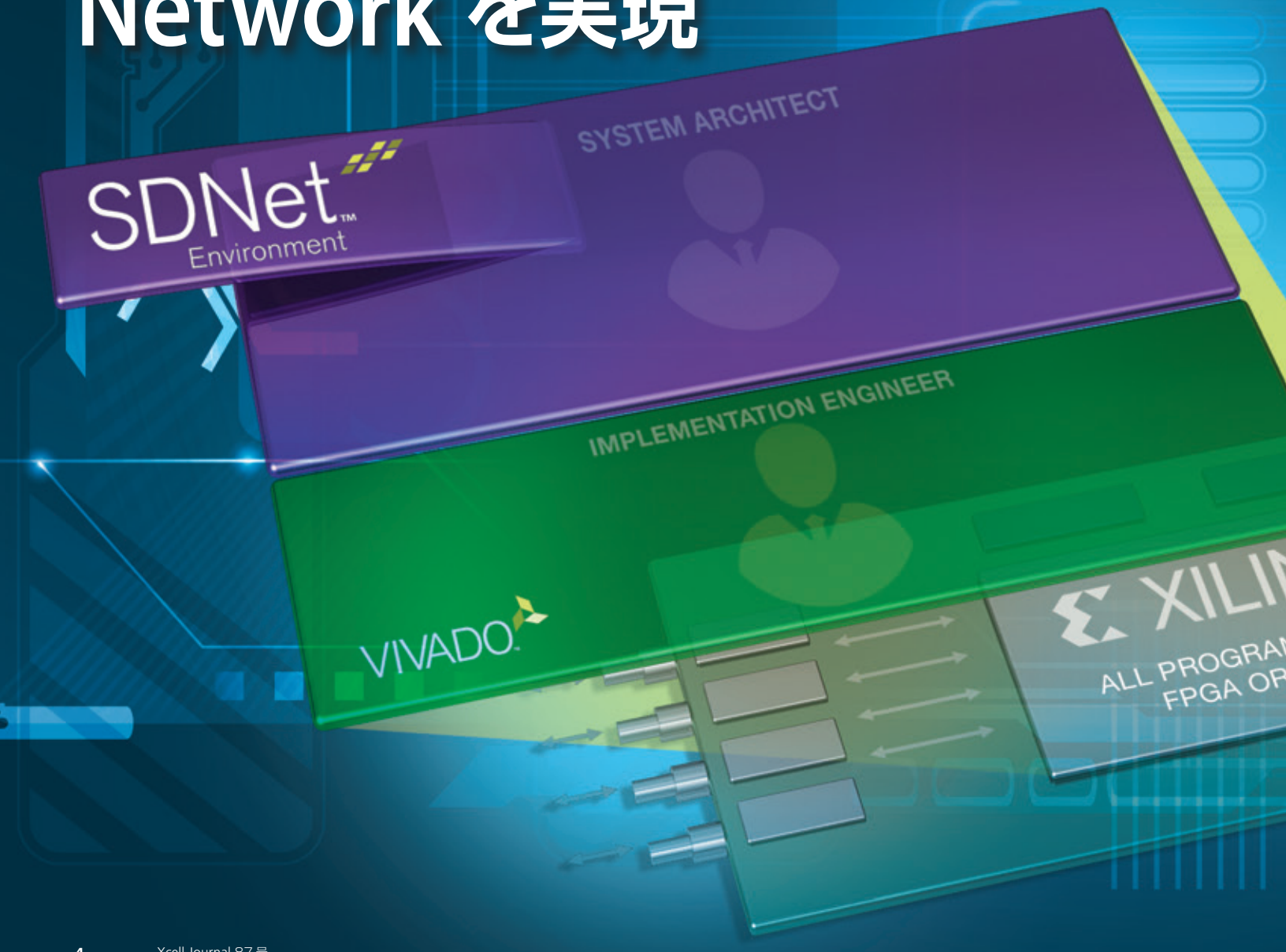


APEX® AWARDS FOR PUBLICATION EXCELLENCE
Excellence in Magazine & Journal Writing
2010, 2011

APEX® AWARDS FOR PUBLICATION EXCELLENCE
Excellence in Magazine & Journal Design
2010, 2011, 2012

Xilinx's New SDNet Environment Enables 'Softly' Defined Networks

ザイリンクスの新しい SDNet 環境で「Softly」Defined Network を実現



ザイリンクスのテクノロジにより、設計チームはチップ上にライン カードを構築し、特定のネットワーク サービスおよびアプリケーションに対してハードウェアを調整することができます。

Mike Santarini
Publisher
Xcell Journal
Xilinx, Inc.
mike.santarini@xilinx.com

通信アーキテクチャが急速に進化している中で、より広い帯域幅、信頼性および安全性が向上した優れたサービスに対する顧客要求を受け、ザイリンクスは市場の流れを一変させるテクノロジおよびデザイン手法を導入しました。これにより、データセンター用だけでなく、有線ネットワークや無線ネットワーク用の次世代ライン カードを迅速に製造およびアップグレードできるようになります。この新しいテクノロジは SDNet と呼ばれるソフトウェア定義の仕様環境です。SDNet をザイリンクス All Programmable FPGA および SoC と一緒に使用することで、通信デザイン グループはザイリンクスが「Softly」Defined Network と称する革新的な手法を次世代のソフトウェア定義のネットワーク アーキテクチャ用ライン カードのデザインおよびアップグレードに適用できます。

固定ネットワークから SDN へ

過去 20 年間の通信アーキテクチャを主に構成していたのは特定仕様のコントロール プレーンとデータ プレーンであり、ネットワーク要件が変化しても拡張できなかった、とザイリンクスの通信 IP およびサービス担当副社長である Nick Possley は述べています。この柔軟性に欠けるアーキテクチャでは、ネットワーク機能を拡張し、全体的な帯域幅を拡大する場合、通信事業者は頻繁に装置を交換する必要がありました。このようなシステムの中核であるライン カードは、非常に特殊な ASIC、ASSP、メモリ IC の組み合わせに大きく依存していました。FPGA は、ライン カード上のチップ間の通信の高速化および橋渡しの役割を果たしました。

その要求がペースアップしたため、それらのサービスを提供する通信事業者および通信システム企業は、より優れた代替手法を模索しました。この数年、それらはソフトウェア定義ネットワーク (SDN) およびネットワーク機能仮想化 (NFV) に移行しています。これらのアーキテクチャでは、コントロール プレーンとデータ プレーンを切り離し、コントロール プレーンにより多くのソフトウェア仮想化を加えています。この結果、通信事業者は新しいアプリケーションを迅速に展開でき、従来のネットワークよりもネットワーク装置のアップグレードが簡単です。これによって、寿命 (および

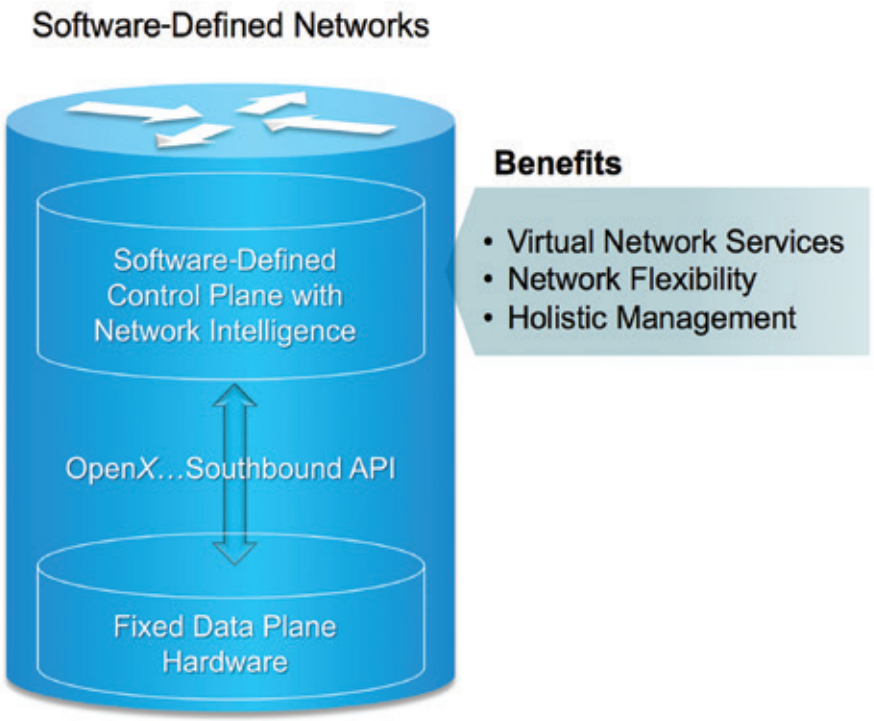


図 1 – 今日のソフトウェア定義ネットワークは、コントロール プレーンとデータ プレーンを切り離してはいるものの、特定仕様のデータ プレーンがあるため、差別化は最小限にとどまり、ライフ サイクルも短期です。

利益性) が向上し、ネットワークの管理が容易になります (図 1)。

しかし、最新の SDN および NFV アーキテクチャでさえ柔軟性に欠け、データ プレーンはプログラムできず、通常既製の ASSP を使用してデザインされていると Possley は述べています。ネットワークの中核であるライン カードでは、コプロセッサと外部メモリのほか、オプティクスに接続した個別の既製のパケット プロセッサおよびトラフィック マネージャー ASSP を使用します。また、カードには、これらすべてのチップ間の通信を高速化するために FPGA が含まれます。

さまざまなチップ メーカーが SDN および NFV アーキテクチャに対して考案している最新バージョンの ASSP は、勿論 SDN 仕様に準拠しています。しかし、サプライヤーはすべてのネットワーク システム企業に同じ ASSP を提供するため、これらのチップには競争力のある製品の差別化や機能の拡充は期待できません。この結果、ネットワーク システム ベンダーは、通信事業者に対して競い合って最低の価格設定を提示しなければならない状況に迫られます。

一見すると、通信事業者は装置のこの安い価格設定を好んでいるように思うかもしれません。しかし、実際には、ASSP ベースの SDN アーキテクチャでさえ特定仕様のデータ プレーン デザインが拡張性に欠けるため、ASSP の特定仕様のハードウェア機能が変化の激しいアプリケーション、プロトコル アップデートおよび新機能要件に対応できなくなった場合、通信事業者は高価なライン カードの交換を現場で実施することになります。このようなライン カードの交換では、技術者が古いカードを取りはずし、新しいカードを取り付けている間ネットワークをシャットダウンする必要があります。さらに、ASSP ベンダーは 1 つのデバイスで多くのマーケットに対応しようとして、デザインを機能過剰に構築しがちです。この結果、こうした ASSP ベースのライン カードは消費電力が大きく、過熱しやすくなる傾向があり、通信事業者は装置を冷却するための対策がさらに必要になります。当然ですが、冷却コストは事業費を圧迫し、通信事業者の最終的な収益が低減することになります。

より優れたソリューション : SOFTLY DEFINED NETWORK

SDNet およびザイリンクスの革新的な Softly Defined Network 手法により、通信システム企業は SDN アーキテクチャで必要とされるネットワーク インテリジェンスを備えたソフトウェア定義のコントロール プレーンをはるかに凌ぐ、統合型の低消費電力 All Programmable ライン カードを開発できます。また、ベンダーはこの新しいテクノロジを使用することにより、コンテンツ インテリジェンスを備えたソフトウェア定義のデータ プレーン ハードウェアでシステムの差別化を図ることができます。つまり、設計チームはシステムで必要とされるまさにそのネットワーク サービスおよびアプリケーションに対してハードウェアを調整することができます (図 2)。

伝統的に、通常ハードウェア デザインのバックグラウンドを持たないネットワーク設計者は、特定プロトコルの要件をインターネットの RFC や ISO 規格文書などのように英語で記述します。

その後、ターゲット デバイスの基盤となるアーキテクチャに精通した専門のエンジニアが、これらの要件を低レベルのインプリメンテーションに特化した記述に変換します (通常、高度な専門マイクロコードを使用)。このハードウェア エンジニアは汎用プロセッサや専用ネットワーク プロセッサでどのようにパケット処理を実行するのかを指定するか、またはカスタム ASIC 向けに機能を設計します。

さらに、ネットワーク設計チームが、ハードウェアが設計者の本来のデザイン意図を満たしているか、または少なくともカードで使用するプロトコルの最新バージョンに対応できることを検証します。ライン カードが要件を満たさない場合は、正常に動作するまでデザイン プロセスを繰り返します。所望の仕様とマイクロコードの関係が解り難く、基盤となるアーキテクチャには性能の限界があり、企業がターゲットとするサービスによって異なる機能があるため、このプロセスは複雑です。

SDNet の Softly Defined Network 手法はこの問題の根底に迫るもので、ネットワーク システム設計チームはこの手法を使用することにより、正しさを検証しながらライン カードを迅速に開発できます。特に、SDNet はライン カード デザインの最も複雑な面である、今日のライン カードでのパケット プロセッサおよびトラフィック マネージャー機能の設計とプログラミングの自動化に着目しています (図 3)。

これらの機能を処理する 2 つの個別の ASSP を装備する代わりに、ネットワーク システム チームはパケット処理とトラフィック管理をその他のライン カード機能と共に 1 つのザイリンクス All Programmable FPGA または SoC に統合できます。彼らは対象アプリケーションに最適なインプリメンテーションを作成できます。多くのチップの機能を 1 つの All Programmable デバイスに統合するだけでなく、SDNet はライン カードのハイレベルのビヘイビアー仕様の作成を簡素化し、ザイリンクスの All Programmable デバイス、ファームウェア、および検証テストベンチでのインプリメンテーションのための RTL ブロックを自動生成します。

「SDNet では、システム設計者は『どのように』ではなく『何を』を指定します」と Possley は述べています。「システム設計者は、基盤となるシリコンにどのように配置されるかは気にせず、配置しようとしているサービスを正確に指定します」。

SDNet フローでは、システム設計者はハイレベルのファンクション仕様を使用してライン カード機能を定義します (図 4)。SDNet を使用することで、構文解析、編集、検索、およびサービス品質 (QoS) ポリシー エンジンなど、多様なパケット処理エンジンに必要な動作を記述できます。設計者は、インターコネクトし、パケット データ フローにまとめることができる、よりシンプルなサブエンジンに関してエンジンを階層的に記述することも可能です。これらのサブエンジンは、ユーザー提供のエンジンを含むことができます。SDNet 仕様環境には、インプリメンテーションの詳細は一切含まれません。これにより、カスタマーは基盤となるアーキテクチャの詳細を理解しなくても、デザインの性能やリソースを自由に拡張できます。また、SDNet 仕様は、特定のネットワーク プロトコルに限定されません。

Possley によれば、SDNet はシンプルで、ザイリンクスと一緒にベータ テストを実施した一部のカスタマーは非常に直観的で使いやすいものであることを認めています。「シンプルで直観的な仕様に変換する必要があるコードの量が大幅に低減されるため、ネットワーク プロセッサのマイクロコーディングと比較すると、作業量は桁違いに軽減されます」と言います。

設計者は SD Net 仕様環境でシステム エンジンとフローの定義を終えると、スループット

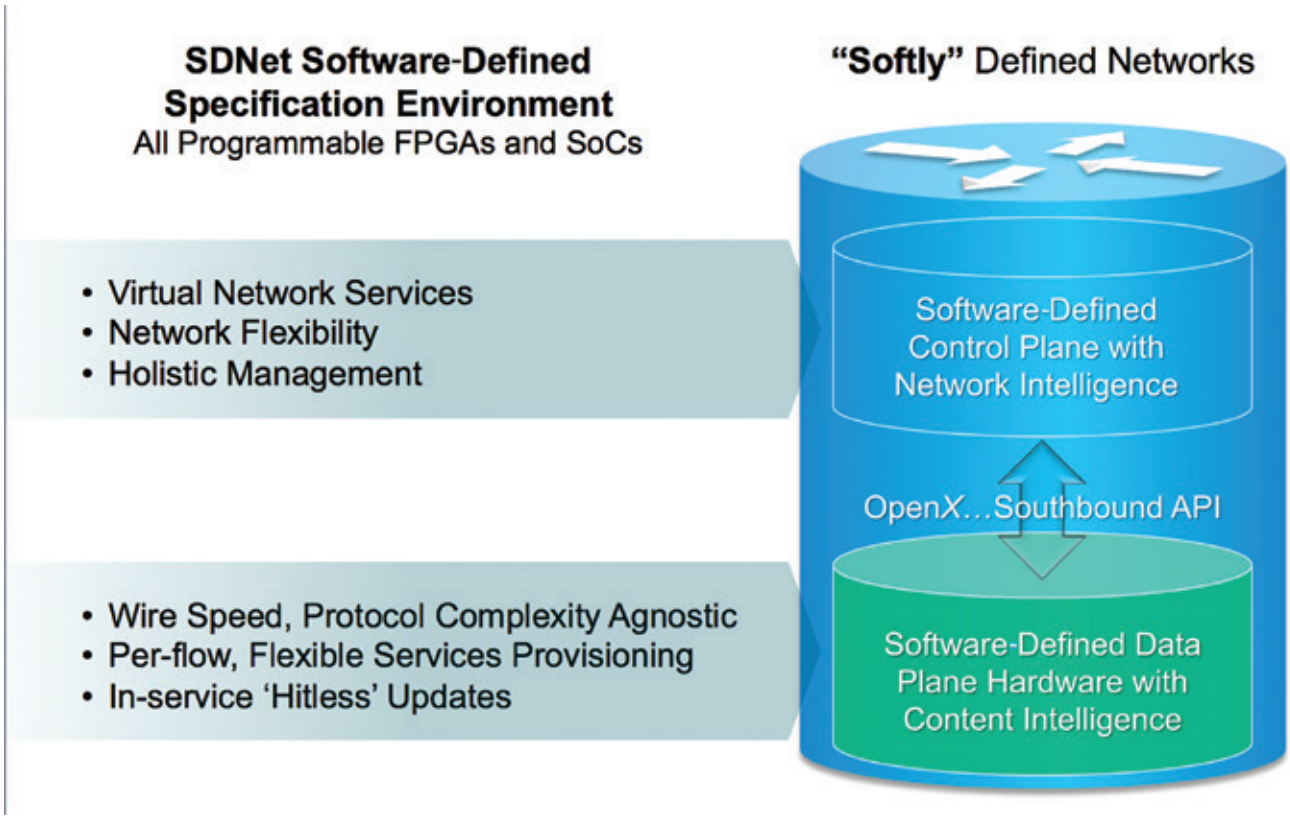


図 2 – SDNet はデータ プレーンに柔軟性とオートメーションをもたらし、次世代ネットワークのデザインおよびアップグレードに対する Softly Defined Network 手法を可能にします。

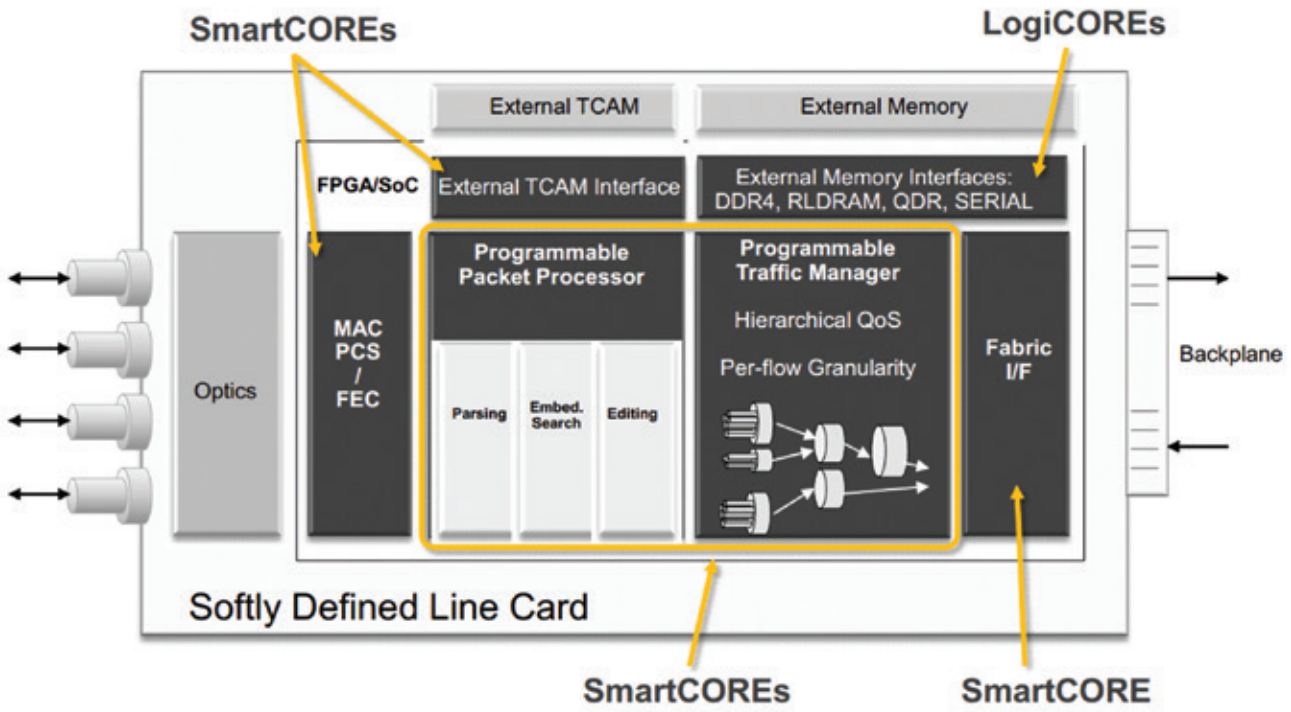


図 3 – SDNet により、高度に統合化された All Programmable ライン カードを作成できます。

とレイテンシ要件およびランタイム プログラマビリティ要件を SDNet コンパイラに指定します。これらは、コンパイラが生成する最適化されたハードウェア アーキテクチャに影響を及ぼします。その後、コマンドを実行すると、デザインで必要となるハードウェア ブロックの RTL を SDNet コンパイラが自動生成します。また、コンパイラはファームウェアと検証 / 妥当性確認テストベンチも生成します。SDNet デザイン環境には、ザイリンクスで最適化されたネットワーキング向け SmartCORE とコネクティビティ向け LogiCOREs ™、外部メモリ制御、エンベデッド プロセッサとの統合が含まれています。

コンパイルが終了したら、ネットワーク エンジニアは IP Integrator (IPI) ツールを使用して、Vivado® Design Suite でデザインのインプリメンテーションを完成することができます。まず、Vivado ツールと IPI を使用して、SDNet コンパイラが生成した RTL アーキテクチャ記述を、最適化されたザイリンクス FPGA インプリメンテーションに変換します。その後、選択したデバイスに十分なリソースがあれば、付加的なライン

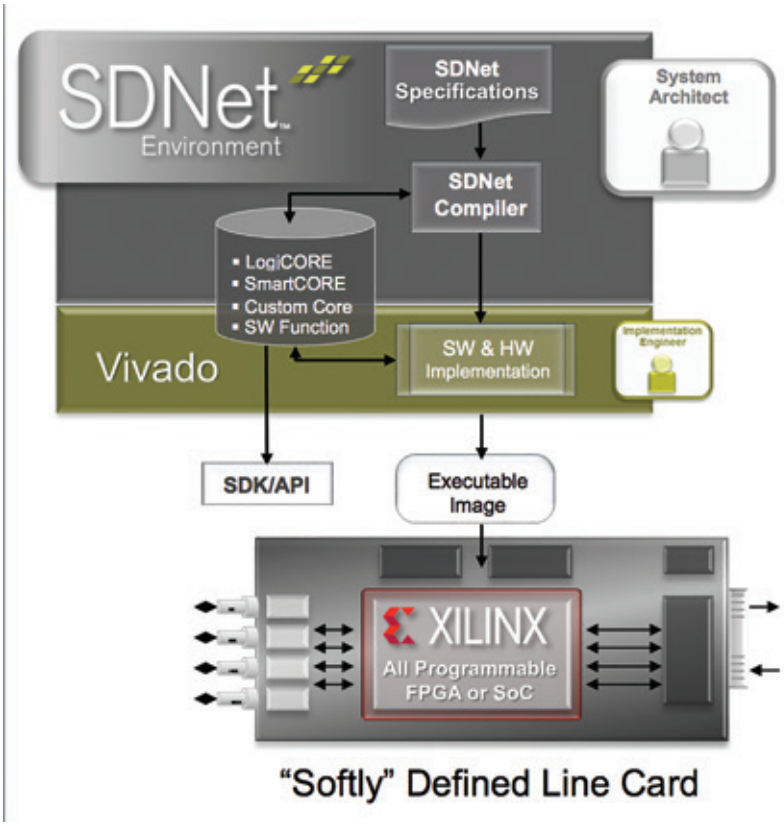


図 4 – SDNet ベースのインプリメンテーション フローにより、
正しさを検証しながら All Programmable ライン カードを設計できます。

カード機能を FPGA に統合し、実質的に All Programmable ライン カードをチップ上に作成できます。

さらに、SDNet は正しさを検証しながら設計できるよう、機能検証・確認用のデータを生成します。具体的には、SDNet コンパイラはデザインの入出力をテストするための大量のテスト パケットを受け入れます。設計者は、プロセスの仕様定義段階でこのパケットを使用することで、SDNet 記述の正確な解釈を作成できます。ネットワーク エンジニアは、SDNet コンパイラで生成された RTL 記述のシミュレーション時にテスト パケットを使用できます。大事なことを言い忘れていましたが、パケットはネットワーク テスト装置を使用したデザインの最終的なインプリメンテーションのハードウェア検証に役立ちます。さらに、SDNet は対応する検索エンジン ルックアップ テーブルの内容を生成します。この検証・確認機能により、デザイン時間が大幅に短縮されると共に、システム設計者とネットワーク ハードウェア エンジニアの間で反復が解消され、高度に差別化された製品をより迅速に市場に投入できます。

ザイリンクスの著名なエンジニアである Gordon Brebner は、アーキテクチャ内の個々のコンポーネントに対しコンパイラがカスタムファームウェア動作およびそのバイナリ エンコードを自動生成すると述べています。「これは、処理を十分に制御できるという感覚を設計者に与えます」。SDNet には、実行の記録を保持し、生成されたアーキテクチャとそのファームウェアの詳細を保存するユーティリティがあります。ユーザーがアップデートした SDNet 記述を入力して再実行すると、コンパイラは新たなハードウェアを生成せず、ファームウェア アップデートのみで変更に対応可能か、またはハードウェア（およびファームウェア）の再生成が必要かを判断します。通常、ライン カードが処理するプロトコルの追加や削除などの中規模アップデートは、ファームウェア アップデートのみで対応可能です。

「どちらも SDNet コンパイラによって生成されるファームウェアとアーキテクチャの間の親密な関係は、ユーザーがヒットレス アップグレードを実行でき、これによりパケットの流れを中断することなく、ファームウェアを変更し、サービスを使用できることを意味しています」(Brebner)。「このため、企業はサービスを中断することなく、重要なサービス アップグレードを実行できます。この革新的な開発は、SDNet テクノロジの特殊性およびそのハイレベル仕様とザイリンクスの All Programmable デバイスの結合によって実現されます」(図 5)。

「サービス提供中のヒットレス アップデートをサポートするデータパス処理機能を生成する SDNet の機能はほかにはありません」と Possley は述べています。「通信事業者は、標準 SDNet API を用いたソフトウェア コントローラーから、ライン カード コンポーネントに対して新しい機能や性能をアップデートできます。アップデート用ソフトウェアは、組み込み型ソフト プロセッサまたは外部プロセッサ上で実行できます」。当然ながら、ザイリンクスの Zynq®-7000 All Programmable SoC にデザインをインプリメントしている場合、デバイスのエンベデッド ARM® プロセッサ上でソフトウェアを実行できると、彼は付け加えました。

「SDNet は、ソフトウェア制御下での詳細なハードウェアのフル プログラマビリティを提供します。これこそが私達が「Softly」Defined Networking と呼ぶ理由です」と Possley は述べています。

SDNet の動作を実演するビデオ デモを含め、SDNet 仕様環境の詳細は、japan.xilinx.com/sdnet を参照してください。同じサイトに、「SDNet (Software Defined Specification Environment for Networking)」(SDNet の背景説明) というタイトルの詳細を解説したホワイト ペーパーが掲載されています。

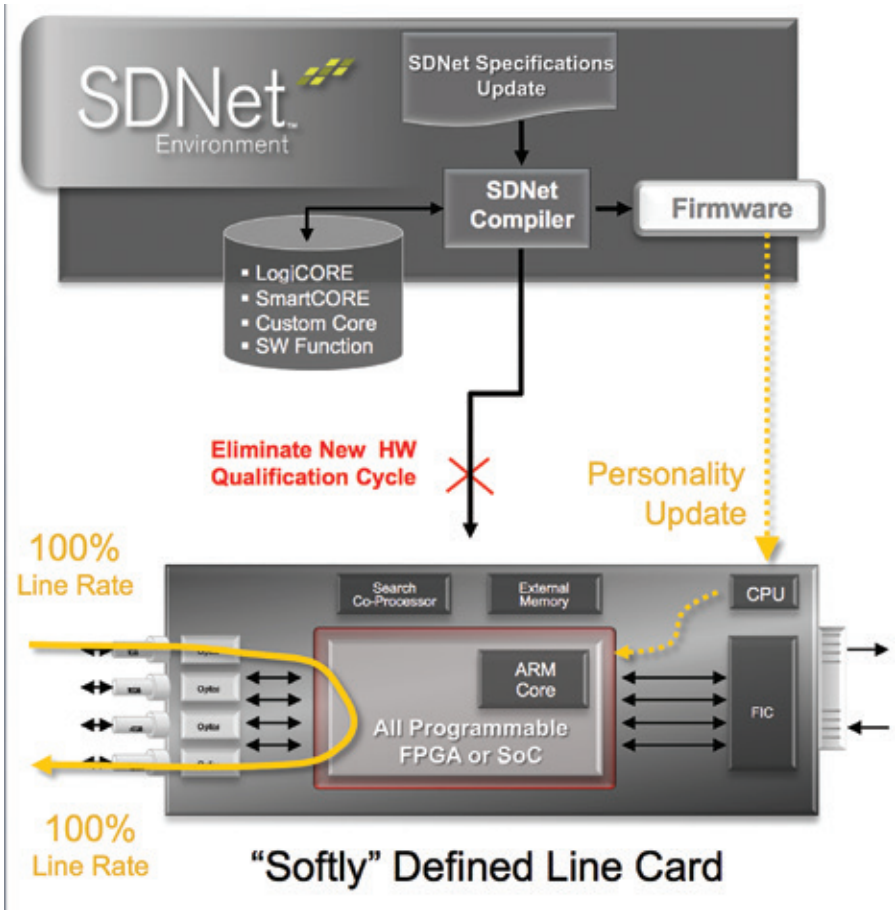
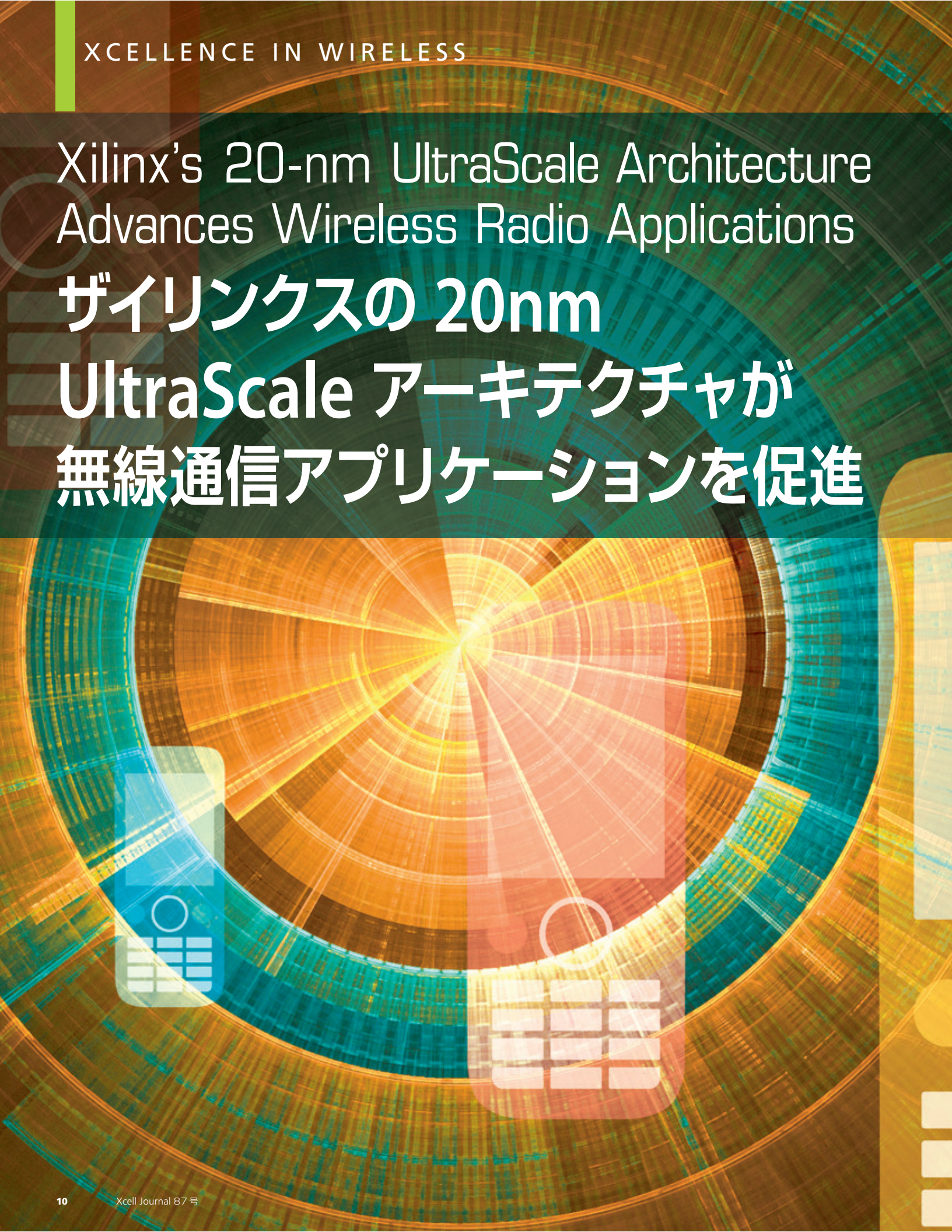


図 5 – 展開後、SDNet により、ベンダーはサービスを中断することなく、ライン カードのプロトコルをアップデートできます。

Xilinx's 20-nm UltraScale Architecture Advances Wireless Radio Applications

ザイリンクスの 20nm UltraScale アーキテクチャが無線通信アプリケーションを促進



次世代 5G システムでは設計が複雑になります。UltraScale デバイスには、この作業を容易にする機能が組み込まれています。

Michel Pecot
Wireless Systems Architect
Xilinx, Inc.
michel.pecot@xilinx.com

次の 5G ワイヤレス通信システムでは、現在使用している 4G システムよりも広い帯域幅 (200MHz 以上) のサポートが求められます。キャリア周波数を高くすることで大きなアンテナ アレイに対応できるだけでなく、より小さいアンテナ エレメントの構築が可能になります。こうしたいわゆる大規模 MIMO アプリケーションと、レイテンシ要件の厳格化が相まって、デザインは桁違いに複雑になります。

昨年末にザイリンクスは 20 ナノメートル UltraScale™ ファミリを発表し、現在最初のデバイスを出荷中です [1,2,3]。この新しいテクノロジは、旧世代の 28nm の 7 シリーズと比べ、特にワイヤレス通信により多くの利点をもたらします。実際には、この新しいシリコンと Xilinx® Vivado® Design Suite [4, 5] のツールの組み合わせは、次世代無線通信アプリケーションなどの高性能信号処理デザインに最適です。

こうしたデザインに対する UltraScale デバイスの利点について、アーキテクチャ面を中心に見てみましょう。具体的には、無線デジタル フロントエンド (DFE) アプリケーションで一般的に使用される機能のインプリメンテーションに対する DSP48 スライスおよびブロック RAM の機能強化の利点です。UltraScale ファミリは、7 シリーズ デバイスと比べると配線の密集度およびクロッキング リソース数がはるかに向上し、特に高速デザインに対しデバイスの使用率を改善できます。ただし、こうした機能は通常デザイン アーキテクチャに直接影響を与えることがないため、ここでは取り上げません。

ULTRASCALE ファブリックへの機能強化の概要

20nm への移行は、ジオメトリ ノードのマイグレーションに伴う集積機能の高度化、ファブリック性能の向上、消費電力の低減を実現するだけではありません。UltraScale 20nm アーキテクチャでは、DFE アプリケーションを直接サポートする強力な新機能がいくつか組み込まれました。これは特に UltraScale Kintex® デバイスに当てはまり、ザイリンクスはこのタイプのデザイン ニーズへの対応に力を入れています。

まず、これらのデバイスには最大 5,520 の DSP48 スライスが含まれています。これは、7 シリーズ FPGA で提供される最大数 1,920 (Zynq®- 7000 All Programmable SoC の場合は 2,020) のほぼ 3 倍です。このため、高度な統合が可能です。たとえば、瞬時帯域幅が 80 ～ 100MHz の 8Tx/8Rx DFE システム全体を 1 つの中規模の UltraScale FPGA にインプリメントできます。これに対し、7 シリーズ アーキテクチャでは各チップが事実上 4x4 システムをサポートする 2 チップ ソリューションが必要です。このようなデザインの詳しい機能説明については、ザイリンクスのホワイト ペーパー WP445「ザイリンクス All Programmable FPGA および SoC での高速無線通信デザインの実現」をお読みください [6]。

受動冷却される遠隔無線装置によって熱制約が課されるため、複合デザインを 1 つのデバイスに統合するには、消費電力を大幅に低減し、放熱を可能にする必要があります。

UltraScale ファミリは、同サイズの 7 シリーズ デバイスと比べてスタティック消費電力を 10 ～ 15 パーセント低減し、類似デザインのダイナミック消費電力を 20 ～ 25 パーセント低減した状態で、このような機能を提供します。さらに、ザイリンクスは UltraScale 製品ラインの SerDes 消費電力も大幅に低減させています。

性能上の利点もあります。最低速グレードの UltraScale デバイスでクロック レートが 500MHz を超えるデザインをサポートしますが、7 シリーズ デバイスでは中速グレードが必要です。そうは言っても、タイミングの観点からブロック RAM は要求が厳しく、この種の性能を実現するには WRITE_FIRST モードまたは NO_CHANGE モードを選択する必要があります。READ_FIRST は、約 470MHz に制限されるため 使用できませんが、ほかの 2 つのモードでは 530MHz は実現可能です。消費電力も低減されるため、可能であれば NO_CHANGE が最善の選択です。

同様に、SerDes は最低速グレードの UltraScale で最大 12.5Gbps のスループットをサポートできるため、最高スピードの JESD204B インターフェイスが実現可能になります。これは、ほとんどの DAC および ADC でまもなく利用できるはずです。同じく、最低速グレードで、2 つの最高 CPRI レート (それぞれスループットが 9.8304Gbps と 10.1376Gbps のレート 7 とレート 8) および DFE システムで通常使用される 10GE インターフェイスもサポートできます。

加えて、UltraScale Kintex リソース の組み合わせはさらに無線アプリケーションに適し、ロジック リソースの最適な利用ができます。特に、DSP 対ロジック比は、DFE デザインで通常要求されるレベルに密接です。より正確には、UltraScale Kintex デバイスは 1K ルックアップ テーブル (LUT) あたり 8 ～ 8.5 個の DSP48 スライスを持つのに対し、7 シリーズ デバイスではこれはわずか 6 つ程度です。

ザイリンクスは UltraScale アーキテクチャのクロッキングおよび配線リソースも著しく向上させました。この向上により、特に高クロック レート デザインでデバイスの使用効率を向上できます。実際、配線密集度が軽減され、設計者はデザインの圧縮および LUT 使用効率の向上を実現できます。特に LUT/SRL の圧縮は効率的です。これは優れたファブリック機能であり、この機能を使用することで、デザインの圧縮率を向上させ、リソースの使用率を最適化できると共に、ダイナミック消費電力についても、関連ロジックに対し最大 1.7 倍の削減が可能です。LUT/ SRL 圧縮の原理は、LUT6 の 2 つの出力を使用して 2 つの異なる機能を 1 つの LUT に圧縮することです。これにより、同じ入力またはメモリに対する読み出し / 書き込みアドレスを共有するのであれば、ロジック ファンクションまたはメモリをインプリメントする 2 つの LUT5 を 1 つの LUT6 に圧縮できます。同様に、2 つの SRL16 を 1 つの LUT6 に圧縮できます。

この機能は、一般的に同じアドレスを共有する多くの小さいメモリ (たとえば、フィルター係数を格納する ROM) と異なる信号経路の時間を合わせるための多くの短い遅延ライン (16 サイクル未満) を統合するデジタル無線デザインに有効です。データ多重化機能、特に 2 入力マルチプレクサーの場合もこの機能が効果的です。ただし、高いクロック レートを対象とする場合は、LUT/SRL 圧縮は慎重に使用する必要があります。まず、タイミング問題を解消するために、O6/ O5 LUT 出力に接続した 2 つのフリップフロップを使用します。同じ理由から、この機能は関連ロジックのみに適用することをお勧めします。この戦略には配線の密集を制限するという利点もあります。

また、クロッキング アーキテクチャとコンフィギャラブル ロジック ブロック (CLB) も UltraScale デバイスの使用効率の向上に貢献します。CLB は 7 シリーズ アーキテクチャをベースにしていますが、CLB あたりのスライスは 2 つから 1 つになり、8 つの 6 入力 LUT と 16 個のフリップフロップを統合しています。その結果、キャリー チェーンは 8 ビット長になり、より幅広い出力を持つマルチプレクサーが構成可能です。さらに、ザイリンクスは制御用リソース (つまり、CLB 内の記憶素子によって共有されるクロック信号、クロック イネーブル信号、およびリセット信号) も向上させました。

ただし、基本的には、無線デザインのアーキテクチャに最も影響を及ぼすのは DSP48 スライスとブロック RAM の機能強化です。これらについてもう少し詳しく見てみましょう。

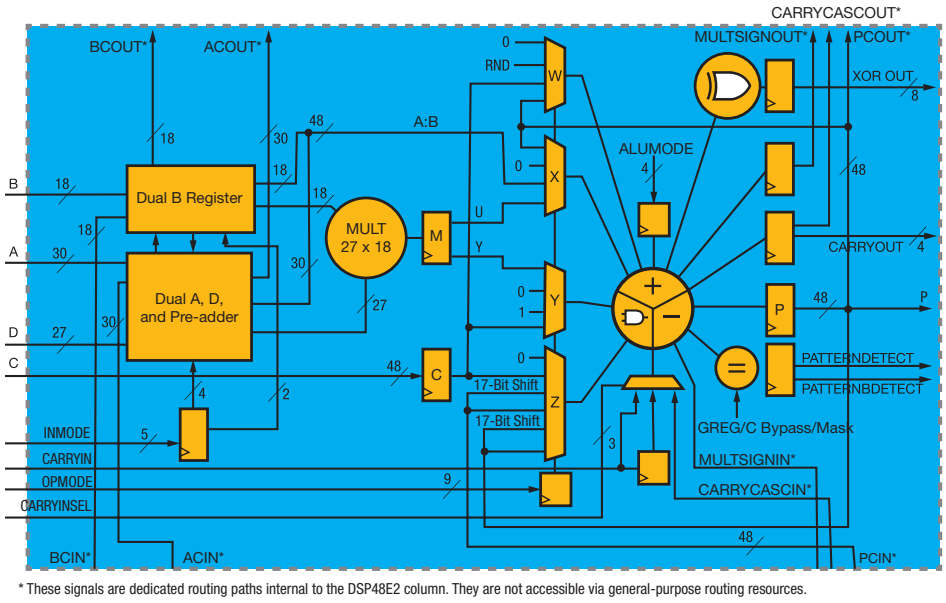
ULTRASCALE DSP48 スライス アーキテクチャの利点

図 1 に UltraScale DSP48 スライス (DSP48E2) を示します。上図 (「a」) は詳細なアーキテクチャを表し、下図 (「b」) では 7 シリーズのスライス (DSP48E1) と比べた場合の 強化機能に焦点を当てています。

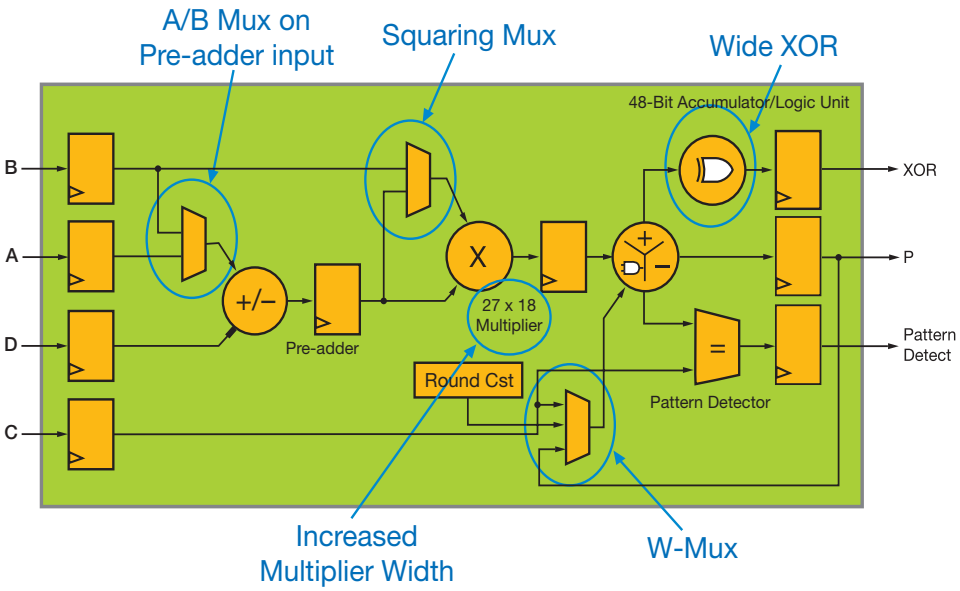
ザイリンクスのユーザー ガイド UG579 で、DSP48E2 機能の包括的な説明を提供します [7]。UltraScale アーキテクチャの主要な強化機能は次の通りです。

- 乗算器の幅を 25x18 から 27x18 に広げ、それに応じて前置加算器の幅が 27 ビットに 広がっています。
- 前置加算器入力として A または B を選択できると共に、多重化ロジックが出力に統合され、いずれかの乗算器入力 (27 ビットまたは 18 ビット入力) に D±A または D±B を入力できます。
- 前置加算器出力は両方の乗算器入力 に供給され (18 ビット入力では適宜 MSB 切り捨て)、最大 18 ビットのデータに対し $(D\pm A)^2$ または $(D\pm B)^2$ の計算が可能です。
- W-mux マルチプレクサーを加えることによって、論理演算ユニット (ALU) に 4 つ目のオペランドが入力されます。C、P、または定数値 (FPGA コンフィグレーション時に定義) を入力として使用できます。これにより、A*B+C+P や A*B+P+PCINなど、乗算器の使用時に 3 入力演算を実行できます。なお、W-mux 出力は ALU 内で加算のみが可能です (減算はできません)。
- X、Y、Z マルチプレクサー出力のいずれか 2 つの 96 ビット間で多入力の XOR を実行できるよう、付加ロジックを統合しました。実際には 4 種類のモードが可能で、1x 96 ビット、2x 48 ビット、4x 24 ビット、8x 12 ビットの XOR 演算を提供します。

乗算器のサイズを 25x18 から 27x18 に大きくすることは、DSP48 スライスのシリコン 面積に対する影響は最小限で、浮動小数点演算のサポートを大幅に向上します。まず、指摘すべきは、DSP48E2 が実際には最大 28x18 ビットまたは 27x19 ビットの符号付き乗算をサポートできる点です。これは、28 ビット オペランド X と



(a) Detailed DSP48E2 architecture



(b) DSP48E2 high-level functional view

図 1 – UltraScale DSP48 スライスアーキテクチャ

18 ビット オペランド Y の乗算を示す図 2 で説明されているように、C 入力を使用して付加ビットを処理することで実現されます。

46 ビット出力の上位 45 ビット (MSB) は次のように計算します。

Z[45:1]＝X[27:1]*Y[17:0]＋X[0]*Y[17:1]

X の MSB 27 ビットと Y の 18 ビットは DSP48E2 乗算器入力に直接入力されるのに対し、X[0]*Y[17:1] は外部の 17 ビット AND 演算子から導き出され、DSP48E2 レイテンシに合わせて 1 段のパイプライン後に C 入力に送られます。実際には、X[0] でリセット ピンを制御し、Y[17:1] をレジスタに直接入力することにより、AND 演算子を省略できます。同様に、外部の 1 ビット AND 演算子および レイテンシを調整するための 3 クロック サイクル遅延を使用して、Z の LSB である Z[0] を計算します。

このため、1 つの DSP48E2 スライスと 18 個の LUT/ フリップフロップ ペアで 28x18 ビット乗算器をインプリメントできます。同じことが 27x19 ビット乗算器にも当てはまり、さらに 27 個の LUT/ フリップフロップ ペアを使用することでインプリメントできます。どちらの場合も、W-mux により、結果の偶数丸めをサポートできます。

倍精度の浮動小数点乗算は、両方の演算子の 53 ビット符号なし仮数の整数積を用います。52 ビット値 (m) は倍精度の浮動小数点表示で格納されますが、これは符号なし仮数の小数部を表し、実際には正規化された 1+m 値で、値を掛け合わせる必要があります。このため、乗算では追加のビットが必要です。両方の 53 ビット オペランドの MSB が 1 であることを考慮し、DSP48E2 26x17 ビットの符号なし乗算器とその強化された機能 (W-mux によって可能になる真の 3 入力 48 ビット加算器など) を有効に利用するために乗算を適宜分割すると、たった 6 つの DSP48E2 スライスと最小限の外部ロジックで 53x53 ビットの符号なし乗算を構築できることがわかります。このようなインプリメンテーションの詳細をすべて提供することはこの記事の範囲外ですが、同様の手法を、旧世代の 7 シリーズ デバイスで実現すると 10 個の DSP48E1 スライスが必要になります。したがって、UltraScale アーキテクチャによって 40 パーセントの改善がもたらされます。

DSP48E2 の 27x18 乗算器は、融合型データ パスをベースとするアプリケーションにも非常に有用です。近頃、融合型積和演算子のコンセプトが IEEE 浮動小数点規格に追加されました [8]。基本的には、乗算器と加算器の間で明示的にデータの丸め、正規化、非正規化を行わずに、浮動小数点演算 A*B+C を構築 します。こうしたファンクションは実際、従来の浮動小数点演算を使用した場合、コスト高になり、レイテンシの最大の原因になります。このコンセプトを一般化し、線形代数 (行列積、コレスキー分解) で一般的な積和 (SOP) 演算子を構築できます。したがって、このような手法は、浮動小数点表現の精度やダイナミック レンジを必要としながらもコストやレイテンシが重要なアプリケーションで非常に効率的です。これは、非線形フィルター係数のアップ

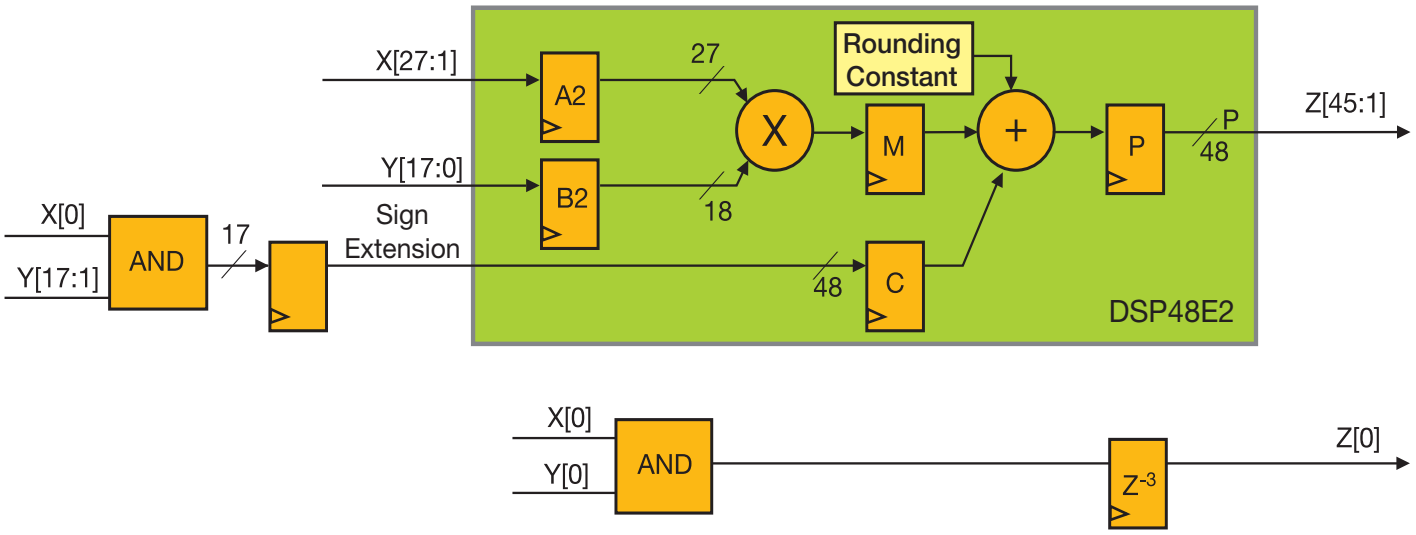


図 2 – 出力の偶数丸め機能を伴う 28x18 ビット符号付き乗算

デート レートを向上させるために、デジタル プリディストーション機能で通常ハードウェア アクセラレーションのサポートが必要となる無線 DFE アプリケーションの場合です。このような場合、FPGA ファブリックに 1 つまたは複数の浮動小数点 MAC エンジンを構築して、ソフトウェアで実行される係数予測アルゴリズムを支援できます (Zynq SoC の ARM® Cortex™-A9 コアの 1 つなど)。

このような演算構造では、仮数の幅を 23 ビットから 26 ビットにわずかに広げただけでも、真の単精度浮動小数点インプリメンテーションと比べてさらに精度は向上し、レイテンシと占有面積は低減することが判明しました。これにも UltraScale アーキテクチャは十分適合し、単精度の融合型乗算器を構築するのに必要な DSP48 スライスはわずか 2 つで済みますが、7 シリーズ デバイスでは付加的なファブリック ロジックを伴い、3 つも必要です。

乗算器前段の DSP48 スライス内に前置加算器を統合することで、デジタル アップコンバーター (DUC) およびデジタル ダウンコンバーター (DDC) 機能を実現するために通常 DFE デザインで使用される対称型フィルターを効率的にインプリメントできます。N タップ対称型フィルターでは、出力サンプルは次のように計算します。

$$y(n)=\sum_{k=0}^{\left[\frac{N-1}{2}\right]} h(k).(x(n-k)+x(n-N+1+k)$$

ここで、x(n) は入力信号を表し、h(n) はフィルター インパルス応答で、h(n)= h(N − 1 − n) です。

このため、入力サンプルのペアが前置加算器に入力され、出力に適切なフィルター係数が乗算されます。7 シリーズ アーキテクチャでは、前置加算器は DSP48E1 の 30 ビット入力 (A) を 25 ビット入力 (D) と一緒に使用する必要があり、その出力が乗算器の 25 ビット入力と接続され、B 入力は 18 ビットの乗算器入力に接続されます。したがって、対称型フィルターを構築する場合、18 ビット以上では係数を量子化できず、帯域除去減衰量が約 85 ～ 90dB に制限されます。これは、干渉レベルが非常に高い環境で動作し、減衰量が大きいフィルターが必要となると思われる次世代の 5G 無線通信システムでは問題となる可能性があります。

UltraScale アーキテクチャはこの問題を克服し、前置加算器入力として A か B を選択でき、いずれかの乗算器入力 (27 ビット入力または 18 ビット入力) へ D ± A または D ± B を入力できるように多重化ロジックが出力に統合されています。この結果、最大 27 ビット係数の対称フィルターをサポートできます。

もう 1 つ DSP48E2 スライスに追加されたのは、前置加算器出力を乗算器の両方の入力に接続できる機能です (18 ビット入力では適宜 MSB 切り捨て)。これにより、最大 18 ビット データに対して $(D \pm A)^2$ や $(D \pm B)^2$ などの演算を行うことができ、二乗誤差項の和の評価時に効率的に使用できます。こうした演算は、モデム内のイコライザーの係数を導出したり、2 つの信号の時間を合わせたりするために最小二乗ソリューションをインプリメントする場合など、最適化問題ではきわめて一般的です。

W-mux マルチプレクサーによる ALU への 4 つ目の入力オペランドの追加が無線アプリケーションに最も大きな利点をもたらすのは疑問の余地がありません。通常、このオペランドは 7 シリーズ デバイスの同じインプリメンテーションと比べ、こうしたデザインに対する DSP48 要件の 10 ～ 20 パーセントを軽減できます。

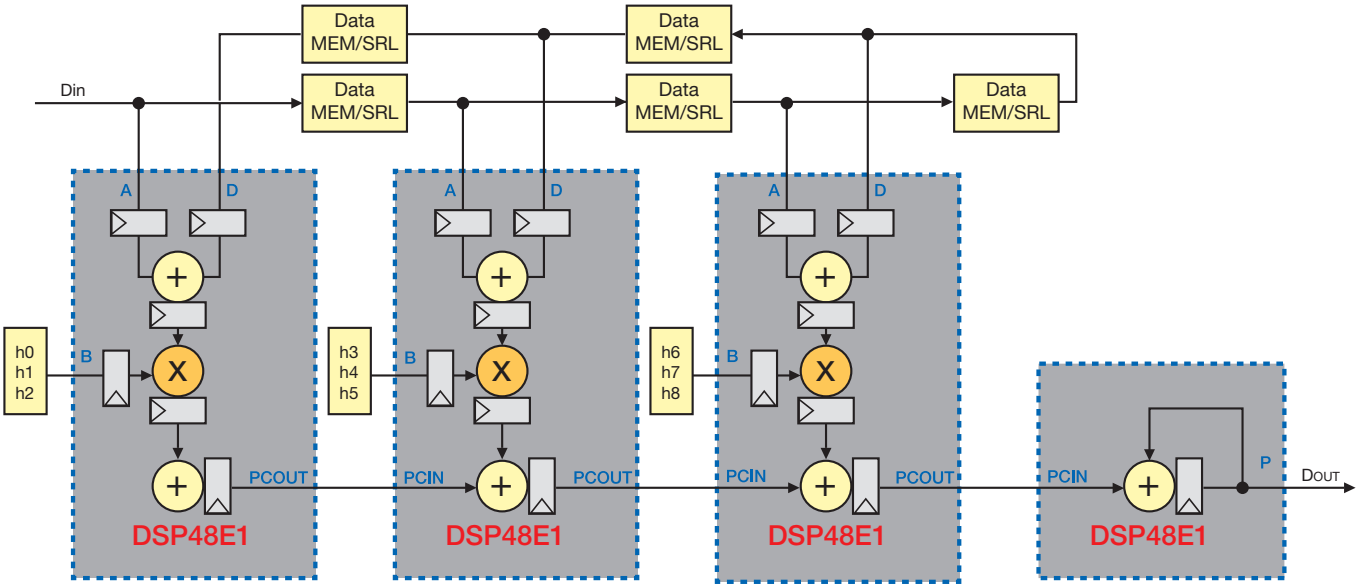
W-mux 出力は ALU 内で加算のみが可能で (減算はできません)、C または P レジスタの内容をダイナミックに設定するか、定数値 (DSP48 出力の偶数丸めまたは対称丸めのために追加する定数など) を FPGA コンフィグレーション時に定義するか、または強制的に 0 にできます。これ により、A*B+C+P、A*B+C+PCIN、A*B+P+PCIN など、乗算器の使用時に真の 3 入力演算を実行できます。これは 7 シリーズ アーキテクチャでは不可能であったことです。 実際、乗算器段は最後の 2 つの部分積出力を生成し、これが ALU 内で加算され、演算が完了します (図 1 を参照)。このため、乗算器はイネーブルであるとき、ALU の 2 つの入力を使用し、7 シリーズ デバイスでは 3 入力演算を実行できません。

この付加的な ALU 入力の恩恵を受ける最も顕著な例は、セミパラレル フィルターと複素 MAC (Multiply and Accumulate) 演算子の 2 つです。この両方についてもう少し詳しく見てみましょう。

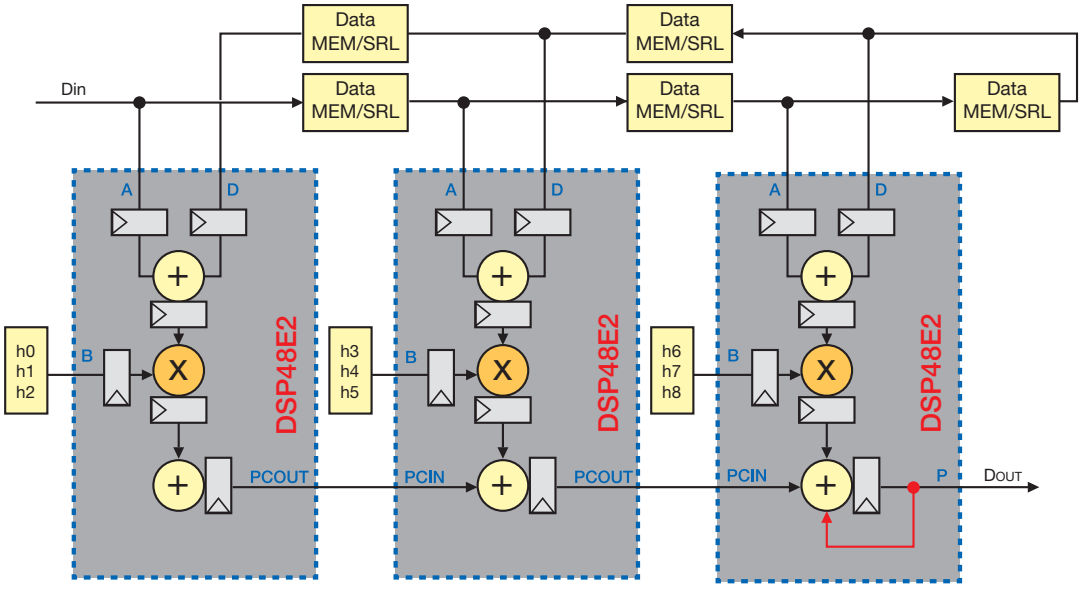
フィルターと MAC

リニア フィルターは、DFE アプリケーションの最も一般的な処理ユニットです。ザイリンクス FPGA でこのような機能を統合する場合は、できる限り、複合サンプリング レート (チャネルの数と各チャネルの共通の信号サンプリング周波数の積によって定義) がデザインの動作クロック レートと同じマルチチャネル フィルターをインプリメントすることをお勧めします [6]。いわゆるパラレル アーキテクチャでは、各 DSP48 スライスがデータ チャネルごとに 1 つのフィルター係数をサポートし、制御ロジックが大幅に簡易化され、これによりデザイン リソースの使用率が低減されます。

ただし、高速クロック レート機能 (最低速グレードの UltraScale デバイスで 500MHz 以上など) では、比較的低いサンプリング レートで動作するフィルターの場合、通常、クロック レートとして複合サンプリング レートの倍数を選択できます。さらにデザインの

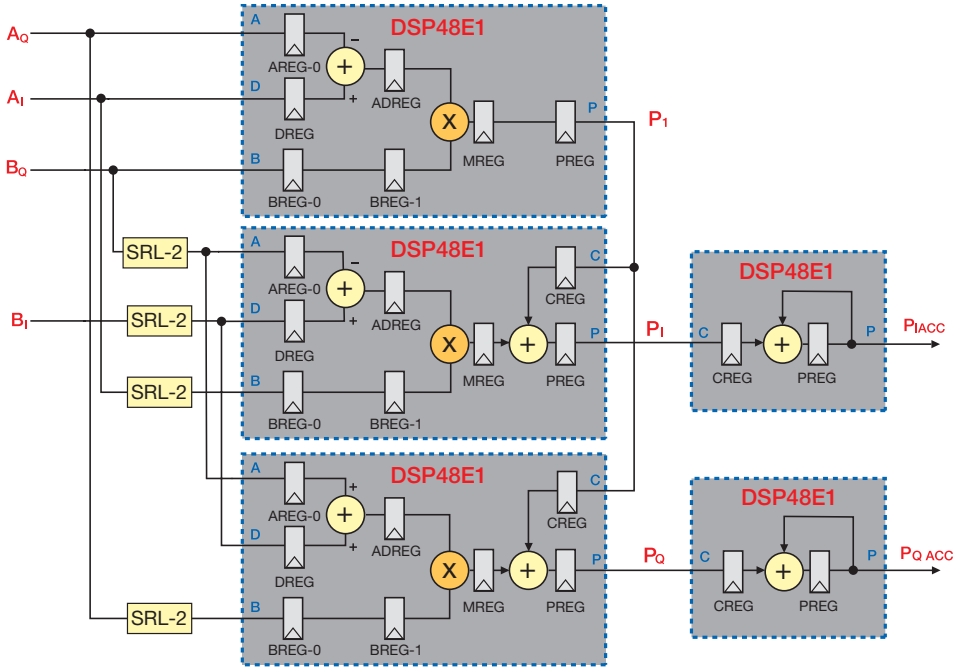


(a) 7 Series implementation

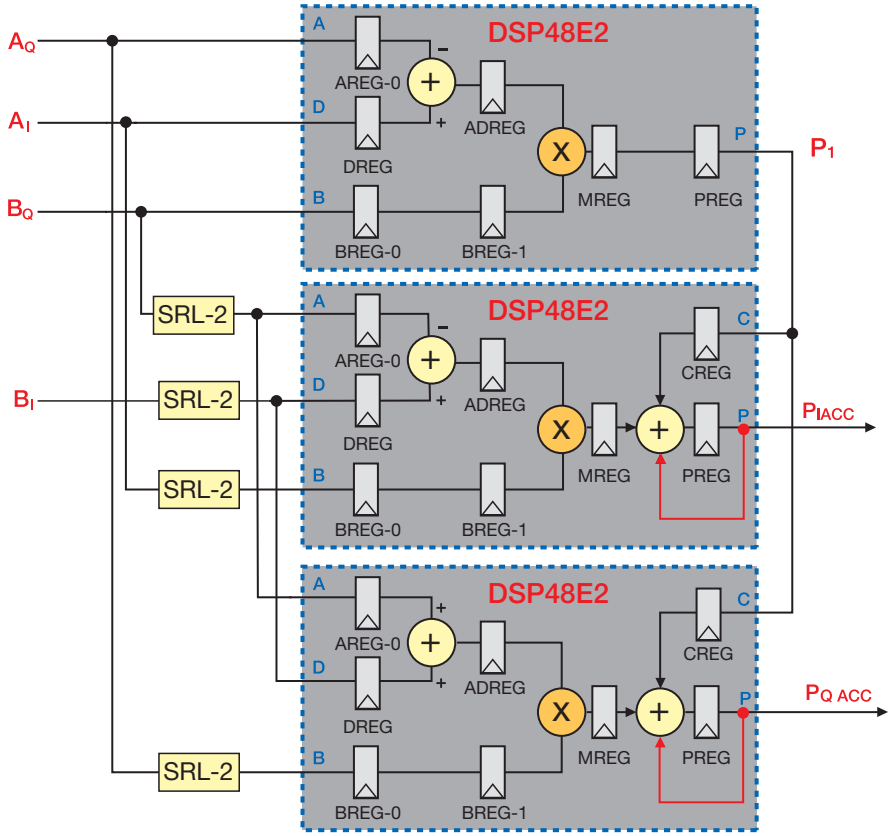


(b) UltraScale implementation

図 3 - 7 シリーズおよび UltraScale アーキテクチャのセミパラレル フィルターインプリメンテーション



(a) 7 Series implementation



(b) UltraScale implementation

図 4 – 7 シリーズおよび UltraScale アーキテクチャの複素 MAC インプリメンテーション

占有面積および消費電力を低減するためにできるだけクロック レートを上げることが望まれます。このような場合、各 DSP48 がチャネルあたり K 個の係数を処理するセミパラレル アーキテクチャが構築されます (ここで、K はクロック レートと複合サンプリング レートの比率)。したがって、最も効率的なインプリメンテーションは、フィルターを K 個の位相に分割し、各 DSP48 が K 個の位相の個々の係数を処理することです。

各クロック サイクルで、フィルター出力の連続位相が計算され、これを累積して出力サンプルを形成する必要があります (K サイクルごとに 1 回)。したがって、パラレル インプリメンテーションと比較した場合、フィルター出力にさらにアキュムレータが必要になります。この全精度のアキュムレータは、$b_S+b_C+b_F$ に相当する広いデータ幅で動作します (ここで、b_S と b_C はそれぞれデータ サンプルと係数のビット幅、$b_F=Log_2N$ はフィルターのビットグロスで、N は係数の合計数)。このため、通常は、占有面積と消費電力を低減しながら、最高のクロック レートをサポートできるように、DSP48 スライス内にアキュムレータをインプリメントします。

注目すべき点は、セミパラレル アーキテクチャがどのタイプのフィルター (シングル レート、整数または分数レート補間および間引き) についても得られることです。図 3 に、7 シリーズと UltraScale 両方のインプリメンテーションの簡略ブロック図を示します。W-mux 機能により位相アキュムレータが最後の DSP48 スライスに吸収され、UltraScale ソリューションの利点が明白になっています。

次に、クロック サイクルごとに出力を 1 つ生成するフルパラレルの複素 MAC 演算子のインプリメンテーションを考えてみましょう。複素積の方程式 $P_I + j.P_Q = (A_I + j.A_Q).(B_I + j.B_Q)$ を、実数乗算を 3 つだけ使用するように、次のように書き換えることができることはよく知られています。

- $P_I = P_1 + A_I.(B_I - B_Q)$
- $P_Q = P_1 + A_Q.(B_I + B_Q)$

ここで、$P_1 = B_Q.(A_I - A_Q)$ です。

したがって、内蔵の前置加算器を利用することで、3 つの DSP48 だけで複素乗算器をインプリメントできます (1 つは P_1 の

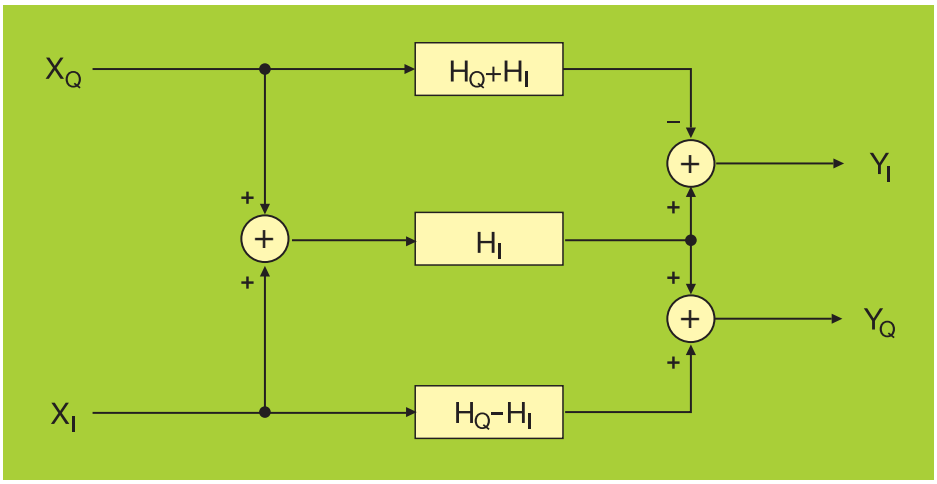


図 5 – 複素フィルターのインプリメンテーション アーキテクチャ

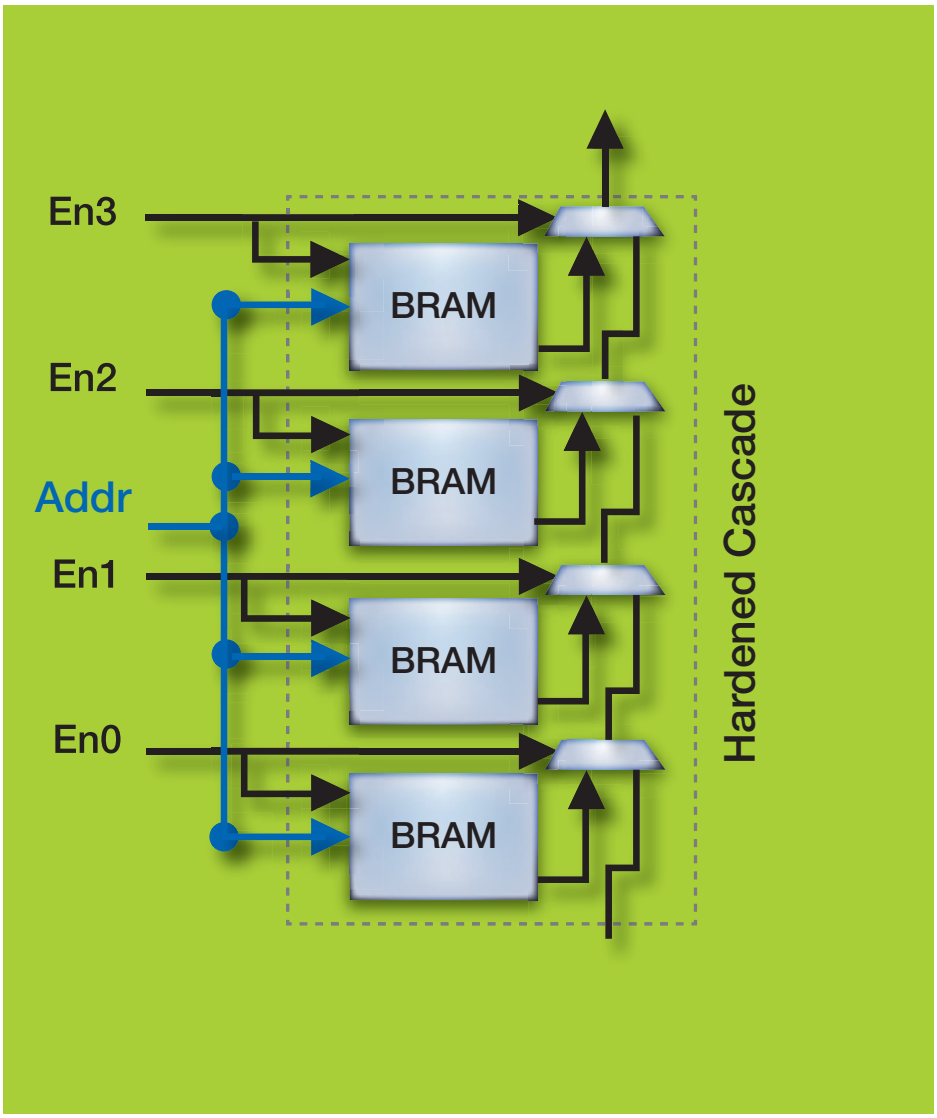


図 6 – UltraScale デバイスの BRAM カスケード

計算用、後の 2 つは P_I および P_Q 出力の処理用)。速度性能にも影響を及ぼすレイテンシ要件によっては、異なるデータ パス間の遅延を調整するために、何らかのロジックを追加する必要があります。最大限の速度サポートを実現するには、DSP48 を完全にパイプライン化して、演算子の総レイテンシは 6 サイクルにする必要があります。したがって、実数部と虚数部のデータ パスを正しく合わせるために、各入力で 2 サイクル遅延ラインが追加されます。これは入力ビットごとに 4 つの SRL2 でインプリメントされますが、実際には SRL 圧縮機能を利用して、2 つの LUT に圧縮されます。

最後に P_I 出力と P_Q 出力のそれぞれでアキュムレータを追加することにより、複素 MAC は完成です。繰り返しますが、このアキュムレータは広いデータ幅で動作するため、DSP48 スライス内により適切に統合されます。7 シリーズと UltraScale デバイスに対応するインプリメンテーションを図 4 に示します。ここでも、W-mux 統合の利点が実証されています。P_I と P_Q の DSP48E2 スライスがアキュムレータを吸収するため、リソースを 40 パーセント削減できます。また、レイテンシも低減され、一部のアプリケーションで効果を発揮する可能性があることも言及に値します。

同じような構造を使用して、図 5 に示すような、3 つの実数フィルターから構成される複素フィルター (複素データと係数) を構築できます。入力信号の実数部と虚数部が 2 つの実数フィルターに供給され、使われる係数はフィルター係数の虚数部と実数部の差と和としてそれぞれ導き出されます。3 番目のフィルターは係数の実数部を使用して、入力の実数部と虚数部の和を並行して処理します。

この 3 つのフィルターの出力が最終的に結合され、出力の実数コンポーネントと虚数コンポーネントが生成されます。この場合も、典型的に DFE アプリケーションで使用されるイコライザーなどのように、パラレルフィルターを構築する必要がある場合、W-mux の恩恵を受けることができます。

ULTRASCALE メモリ アーキテクチャの利点

UltraScale デバイスに搭載されているブロック RAM は基本的には 7 シリーズと

同じですが、新しいアーキテクチャではダイナミック消費電力のゲーティング機能のほか、ハードウェアによるデータ カスケード手法が導入されています。図 6 はこのカスケードを示したもので、同じカラム内で上段および下段に隣接する各ブロック RAM の間にデータ マルチプレクサーが組み込まれています。このため、ロジック リソースをさらに使用することなく、ボトムアップ形式でより大きなメモリを構築できます。

デバイスの各カラム全体がカスケードの対象になりますが、クロック スキューを回避し、最適なタイミング性能を実現するために、使用は 1 つのクロック領域 (つまり、12 個連続した BRAM) に制限するほうが良いでしょう。また、さまざまなカスケード機能のインプリメンテーションに対応する柔軟性を備えています。実際、ブロック RAM データ入力またはオプション レジスタ前後の出力にマルチプレクサーを適用できます。

カスケードは、7 シリーズ デバイスでは実現できない、占有面積の低減、最高速クロック レート、消費電力の削減をサポートしながら、複数の BRAM を必要とする大きなメモリの構築の可能性を切り開きます。たとえば、16 ビット データを格納する 16K メモリをインプリメントする場合、ロジック リソースとレイテンシが増えてタイミングや配線の密集度に影響を及ぼす可能性がある外部のデータ多重化を回避するために、7 シリーズ デバイスでは、8 つの BRAM (36K) を 16Kx2 ビットとして構成します。読み出しまたは書き込み操作時に 8 つのブロック RAM がイネーブルになるため、ダイナミック消費電力の観点では、残念ながらこれはあまり効率的な方法ではありません。 最適なソリューションは、2Kx16 ビット構成を使用することです。こうすれば、1 つの BRAM だけがイネーブルになるので、ダイナミック消費電力を 8 分の 1 に低減できます。これこそがそもそもカスケード機能が、ダイナミック消費電力のゲーティング機能と共に、UltraScale デバイスで実現することです。

ブロック RAM カスケードのもう 1 つの正確なアプリケーションは、通常 DFE システムのベースバンド CPRI インターフェイスと統合されている、I/Q データスイッチング機能のインプリメンテーションに関連します。図 7 に、基本的に NxM メモリ アレイから構成されるハイレベルスイッチングアーキテクチャを示します。N 個の入力ストリームの連続データが出力先に応じて縦列の適切なブロック RAM に書き込まれ、M 個の出力ストリームがカラム内の適切なブロック RAM から読み出されます。したがって、BRAM カスケードで各カラムを効果的にインプリメントできます。

20nm UltraScale ファミリの詳細については、http://japan.xilinx.com/products/silicon-devices/fpga/index.htm を参照してください。

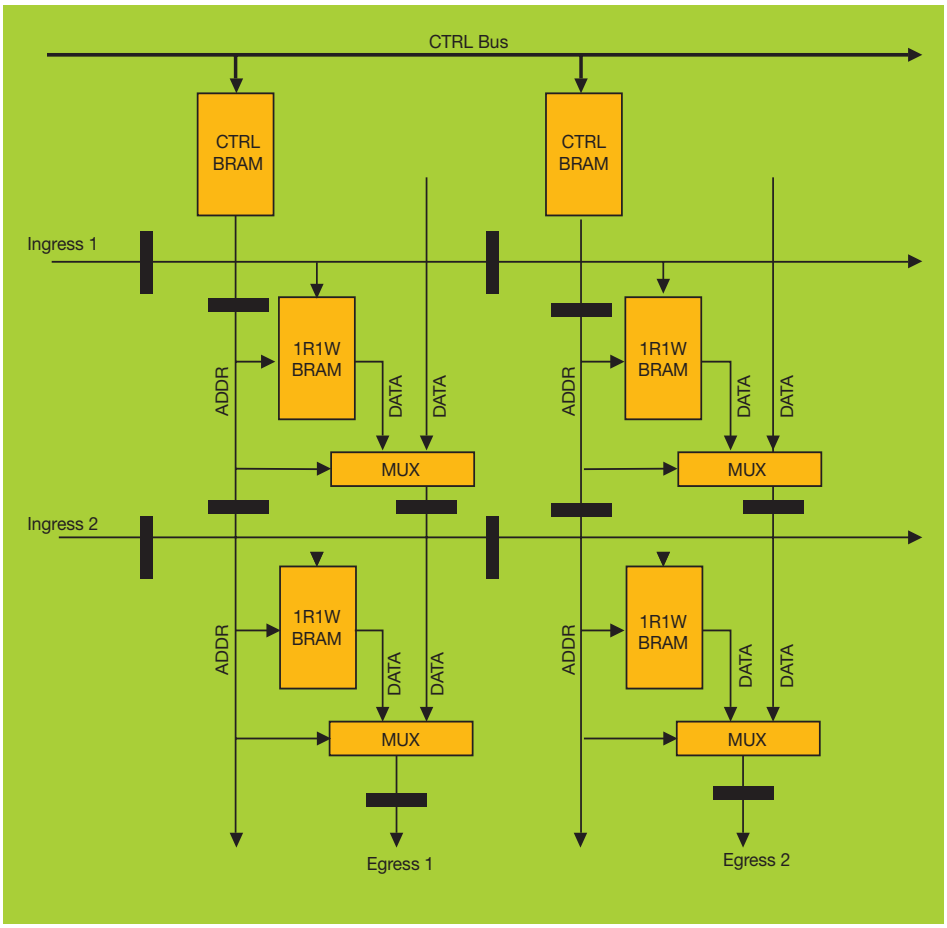


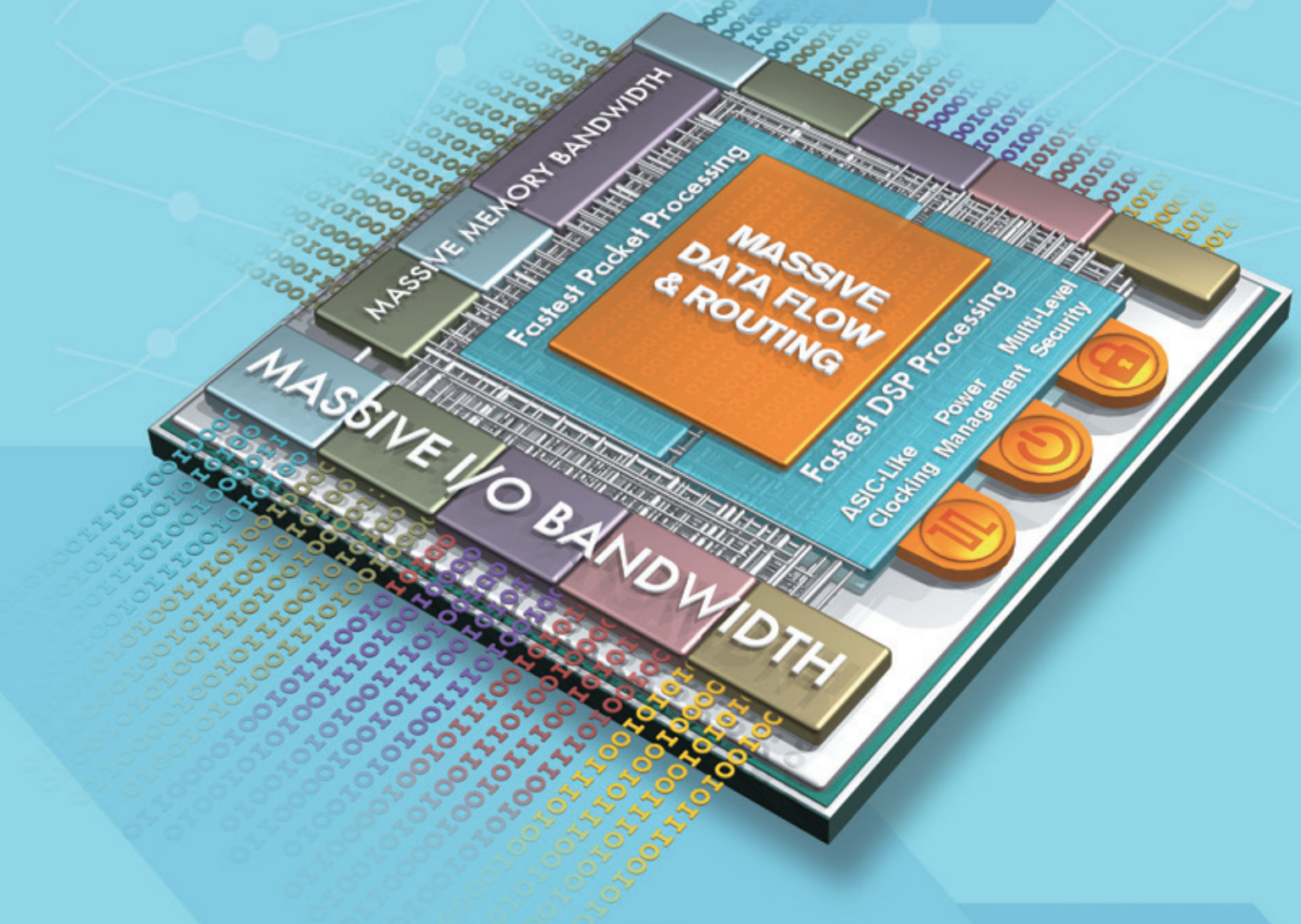
図 7 – データスイッチングのハイレベル アーキテクチャ

参考資料

1. Xilinx backgrounder, “Introducing UltraScale Architecture: Industry’s First ASIC-Class All Programmable Architecture,” July 2013
2. Xilinx white paper WP435, “Xilinx UltraScale: The Next Generation Architecture for Your Next Generation Architecture,” July 8, 2013
3. Xilinx datasheet DS890, “UltraScale Architecture and Product Overview,” Feb. 6, 2014
4. Xilinx backgrounder, “9 Reasons why the Vivado Design Suite Accelerates Design Productivity,” July 2013
5. Xilinx user guide UG949, “Design Methodology Guide for the Vivado Design Suite,” July 5, 2013
6. Xilinx white paper WP445, “Enabling High-Speed Radio Designs with Xilinx All Programmable FPGAs and SoCs,” Jan. 20, 2014
7. Xilinx user guide UG579, “UltraScale Architecture—DSP Slice, Advance Specification User Guide,” Dec. 10, 2013
8. IEEE Computer Society, “IEEE Standard for Floating Point Arithmetic, IEEE Std 754-2008,” Aug. 29, 2008

A Generation Ahead

業界初、ASIC クラスのプログラマブル アーキテクチャ



UltraSCALE™ Architecture

詳細はこちら

XILINX®
ALL PROGRAMMABLE

ザイリンクス株式会社
製品のお問い合わせは下記の販売代理店へどうぞ

■東京エレクトロン デバイス(株) TEL(045)443-4016 x2web@teldevice.co.jp ■アヴネット・インターニックス(株) TEL(03)5792-8210 EVAL-KITS-JP@avnet.com
■(株)PALTEK TEL(045)477-2005 nfo_pal@paltek.co.jp ■新光商事(株) TEL(03)6361-8086 X-Pro@shinko-sj.co.jp

Angle Measurement Made Easy with Xilinx FPGAs and a Resolver-to-Digital Converter

ザイリンクス FPGA およびレゾルバー - デジタル コンバーターにより、角度測定が容易に

アングル トランスデューサーを FPGA と適切に組み合わせることで、今までよりも優れた機械システムを考案できます。

N. N. Murty
Scientist "F"

S. B. Gayen
Scientist "F"

Manish Nalamwar
Scientist "D"
nalamwar.manishkumar@rcilab.in

K. Jhansi Lakshmi,
Technical Officer "C"
Radar Seeker Laboratory
Research Centre IMARAT
Defense Research Development Organization
Hyderabad, India

人間が車輪を発明して以来、正確さの程度に違いこそあれ、われわれは車輪をより効率的に回転させる方法を追求してきました。過去数世紀にわたって、科学者と技術者はこの目標を達成するための多くの方法を研究および考案し、基本的な車輪と車軸のシステムの原理を自動車からステレオのノブ、あらゆる機械の歯車、手押し車まで、ほぼすべての機械システムに適用してきました [1]。

このように長い年月の末、車輪を効率よく回転させる上で最も重要な要素は車輪自体ではなく (Reinventing the wheel「車輪の再発明」)、車輪の軸角度であることを突き止めました。そして現在、軸角度を測定および最適化するために最も効果的なのは、アングル トランスデューサーを使用する方法です。車軸の監視および改良を通じて車輪効率の最適化に役立つアングル トランスデューサーには多くの種類がありますが、この作業に FPGA を適用することで、素晴らしい結果を達成し、幅広いアプリケーションで車軸 / 車輪の回転効率を向上できます。

この作業を、ザイリンクス® FPGA を使用することによって技術者がどのように最適化しているかを詳しく検討する前に、アングル トランスデューサーの基本原理を簡単に確認しましょう。現在、エンコーダーとレゾルバーの 2 つのタイプが広く使用されています。

エンコーダーとレゾルバーのタイプ

エンコーダーはインクリメンタルとアブソリュートの 2 つの基本カテゴリーに分類されます。インクリメンタル エンコーダーは車軸の 2 つの位置を監視し、車軸がこれらの位置を通過するたびに

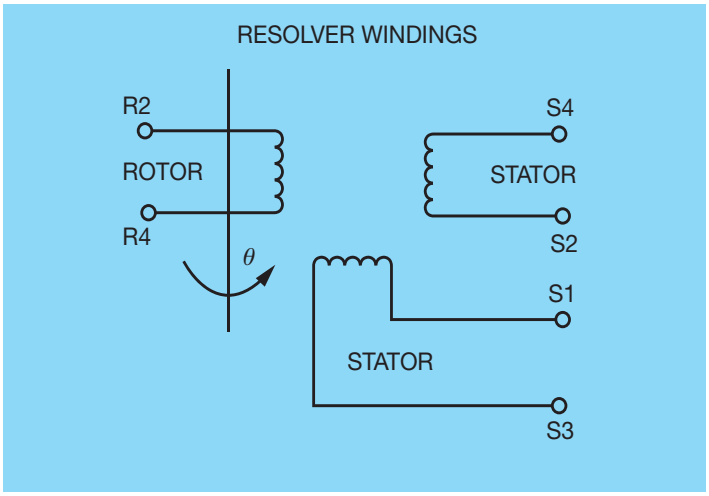


図 1 – レゾルバーの回転子に対する励起

A パルスまたは B パルスを生成します。その後、別の外部電気カウンターがこのパルスから速度と回転方向を読み取ります。インクリメンタル カウンターは多くのアプリケーションで有効ですが、欠点もいくつかあります。たとえば、車軸の電源が切断されると、まずインクリメンタル エンコーダーは運転開始前の所定の校正点に戻り、校正を実行しなければなりません。また、インクリメンタル カウンターは電気的干渉の影響を受けやすく、システムに送信するパルスの精度低下を引き起こしたり、回転数の正確さも得られなくなる可能性があります。さらに、インクリメンタル エンコーダーの多くは光電デバイスであり、そのことが対象アプリケーションで問題となる場合、放射線危険地域では使用できません。

アブソリュート エンコーダーは、車軸の回転数と方向を監視するセンサー システムです。アブソリュートエンコーダー ベースのシステムでは、電気的接触点や光電式基準器を持つ車軸に車輪を取り付けるのが一般的です。車軸が作動している場合、アブソリュート エンコーダー ベースのシステムは回転と作動の方向を記録し、簡単にコード (通常はバイナリ コードまたはグレイ コード) に変換できるパラレル デジタル出力を生成します。アブソリュート エンコーダーには、校正は一度で済み (通常は工場で行われます)、使用のたびに校正する必要がないというメリットがあります。さらに、他のエンコーダーよりも信頼性が高いのが一般的です。ただし、アブソリュート エンコーダーは概して高価であり、特にエンコーダーが測定を行う電子システムから遠く離れている場合、パラレル データ転送は適していません。

レゾルバーは回転トランスで、監視している入力車軸角度と出力電圧が一意に関連するアナログ デバイスです。これは回転角度が 0 ～ 360° の絶対位置トランスデューサーで、車軸と直結し、速度および位置をレポートします。レゾルバーはエンコーダーと比べていくつかの利点があります。堅牢なデバイスとして、粉塵、油分、極端な温度、衝撃、放射線といった過酷な環境に耐えることができます。レゾルバーはトランスとして、信号絶縁や電気的干渉の固有の同相信号除去を提供します。これらの機能に加え、レゾルバーの場合、角度データの伝送に必要なワイヤはわずか 4 本で、重工業から小型システム、航空宇宙産業で使用されるものまであらゆる用途に適合します。

さらなる改良点は、回転子に対してスリップ リング接続が必要ないブラシレス レゾルバーであることです。このため、このタイプのレゾルバーは信頼性が高く、ライフ サイクルが長くなります。

レゾルバーは、車軸角度に関連する出力電圧を取得するために、2 つの方法を使用します。第 1 の方法では、図 1 に示すように、

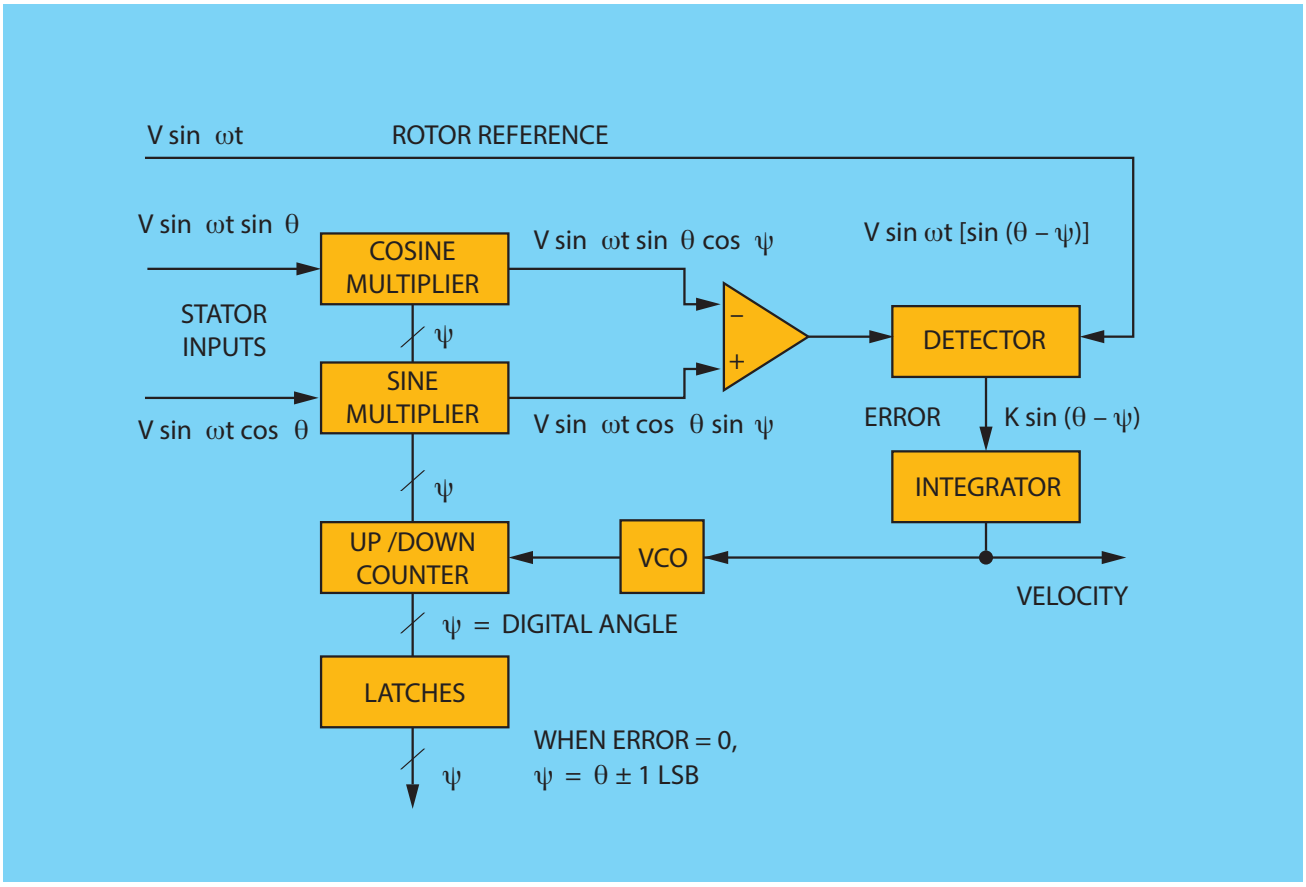


図 2 – レゾルバー-デジタル コンバーター (RDC) のブロック図

交流信号によって回転子巻線が励起され、2 つの固定子巻線から出力が取り込まれます。固定子巻線は機械的に直角に配置されるため、出力信号の振幅は車軸角度の三角関数の正弦関数および余弦関数によって求められます。正弦波信号と余弦波信号はどちらも本来の励起信号と同じ位相を持ちます。車軸の回転に合わせて、振幅だけが正弦関数および余弦関数によって変調されます。

第 2 の方法では、固定子巻線が交流信号で励起され、互いに直交位相になります。この状態で、回転子巻線で電圧が誘起されます。巻線の振幅と周波数は固定ですが、位相は車軸角度に応じて変わります。

レゾルバーは、角度を測定する必要がある場所に配置できます [2]。電子機器は通常レゾルバー - デジタル コンバーター (RDC) で、デジタル出力の測定が必要な場所に配置できます。レゾルバーからのアナログ出力には車軸の角度位置情報が含まれ、RDC を使用してデジタル形式に変換されます。

一般的な RDC の機能

通常、レゾルバーの 2 つの出力は、RDC の正弦波器と余弦波乗算器に適用されます [3]。これらの乗算器は正弦関数と余弦関数のルックアップ テーブルを内蔵し、乗算デジタル - アナログ コンバーターとして機能します。図 2 にこの原理を示します。

まず、アップ / ダウン カウンターの現在のステートが試験用の角度 ψ を表すデジタル数値であるとします。コンバーターは、測定しているアナログ角度 θ を追跡し、これと等しくなるようにデジタル角度 ψ を継続的に調整しようとします。

レゾルバーの固定子出力電圧は次のとおりです。

$$V1 = V \sin\omega t \sin\theta \quad \text{Eq. 1}$$

$$V2 = V \sin\omega t \cos\theta \quad \text{Eq. 2}$$

ここで、θ はレゾルバーの回転子の角度です。デジタル角度 ψ が余弦波乗算器に入力され、その余弦関数に V1 が乗算され、次の項が生成されます。

$$V \sin\omega t \sin\theta \cos\psi. \quad \text{Eq. 3}$$

デジタル角度 ψ は正弦波乗算器にも入力され、V2 が乗算され、次の項が生成されます。

$$V \sin\omega t \cos\theta \sin\psi. \quad \text{Eq. 4}$$

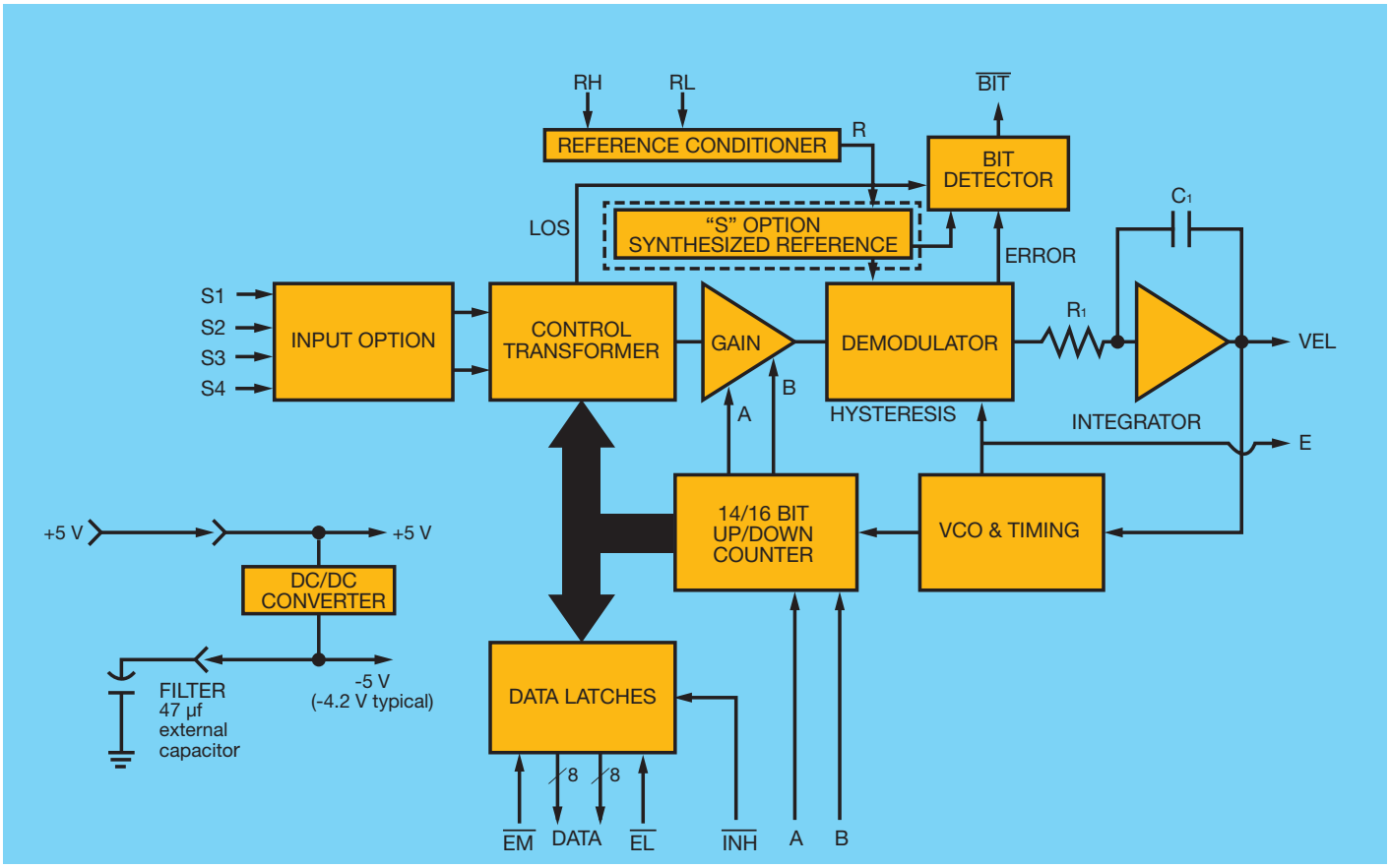


図 3 – SD-14620 のブロック図 (1 チャネル)

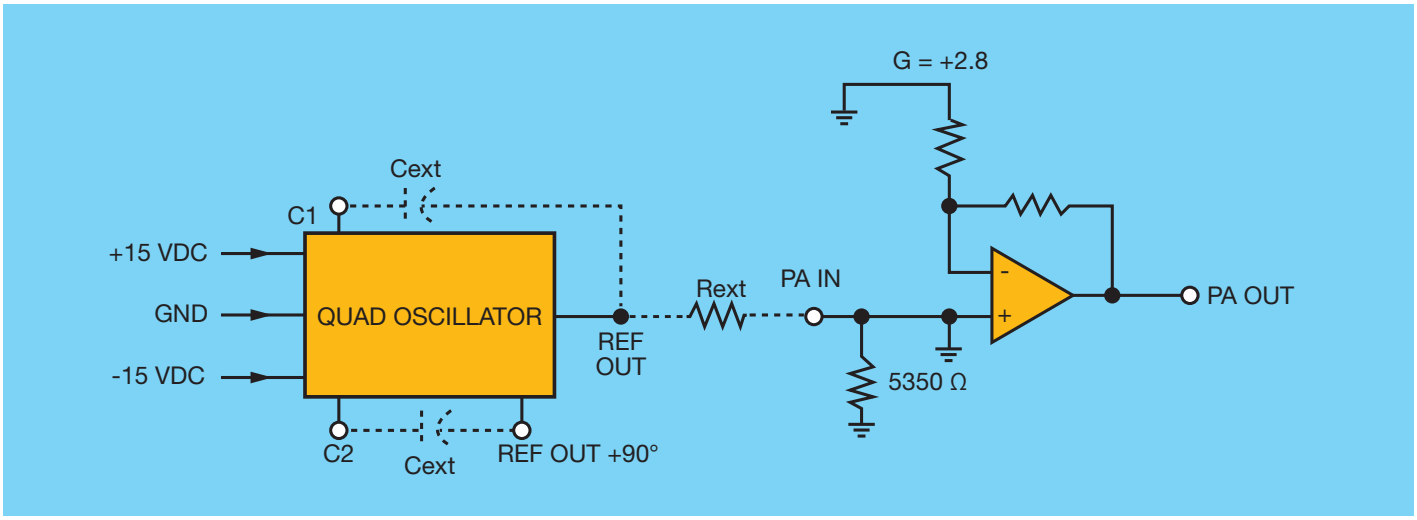


図 4 – OSC-15802 基準オシレーターのブロック図

この 2 つの信号が誤差増幅器により互いに減算され、次の形式の誤差信号が生成されます。

$$(V \sin\omega t \ \sin\theta\cos\psi \ - V \sin\omega t \ \cos\theta \sin\psi) \qquad \text{Eq. 5}$$

$$V \sin\omega t \ (\sin\theta \cos\psi - \cos\theta \sin\psi) \qquad \text{Eq. 6}$$

三角関数の公式から、これは次のように変換されます。

$$V \sin\omega t \ [\sin (\theta - \psi)] \qquad \text{Eq. 7}$$

検出器はレゾルバーの回転子電圧を基準として使用し、この ac 誤差信号を同期検波します。このため、dc 誤差信号は sin (θ -ψ) に比例します。

dc 誤差信号は積分器に供給され、その出力が電圧制御型オシレーターを駆動します。この結果、VCO によって、アップ/ダウン カウンターは適切な方向でカウントを行い、1 回のカウントで次のようになります。

$$\sin (\theta - \psi) \rightarrow 0. \qquad \text{Eq. 8}$$

この結果が得られた場合、次のようになり、

$$\theta - \psi \rightarrow 0, \qquad \text{Eq. 9}$$

したがって、次のようになります。

$$\theta = \psi \qquad \text{Eq. 10}$$

したがって、カウンターのデジタル出力 ψ は角度 θ を表します。ラッチにより、ループの追跡を中断することなくこのデータを外部に転送できます。

この回路は実質的には 2 つの乗算器を持つのと同様で、Type 2 サーボ ループと等価です。1 つはカウンターで、パルスを累算します。もう 1 つは検出器の出力側の乗算器です。定数による回転速度入力を持つ Type 2 サーボ ループでは、外部で生成される変換信号を必要とすることなく、入力を追跡し、連続的に出力デジタル ワードが生成されます (入力が追跡されます)。

RDC の典型例 : SD-14621

SD-14621 は Data Device Corp. (DDC) の小型の低コスト RDC です。チャネルが 2 つあり、分解能制御はプログラム可能です。分解能のプログラミングにより、10 ビット、12 ビット、14 ビット、16 ビットのいずれかのモードを選択できます [4]。この機能によって、分解能を低くすれば追跡を迅速化でき、分解能を高くすれば精度を上げることができます。このコンバーターはサイズ、コスト、精度、汎用性により、高性能の軍事システム、商業システム、および位置制御システムに適しています。

デバイスの動作には、単一の +5V が必要です。コンバーターはアナログ グランドを基準にして電圧レンジが ± 4V の速度出力 (VEL A、VEL B) を持ち、回転速度計と置き換え可能です。信号の損失 (LOS) を示すために、2 つのチャネル (/BIT A と /BIT B) に 2 つのビルトイン テスト出力が提供されています。

このコンバーターは、入力フロント エンド、エラー処理装置、デジタル インターフェイスの 3 つの主要セクションから構成されます。フロント エンドは、同期入力、レゾルバー入力、直接入力で異なります。同期入力には電子 Scott-T を使用し、レゾルバー入力にはレゾルバー コンディショナーを、直接入力にはサイン コサイン ボルテージ フォロワーを使用します。これらの増幅器が高精度の制御変圧器 (CT) に入力を与えます。CT のもう 1 つの入力は 16 ビットのデジタル角度 ψ で、この 2 つの入力の差であるアナログ誤差角度または角度差が出力になります。CT は、精度比率において、アンプ、スイッチ、ロジック、キャパシタを使用して、

SIN θ COS ψ - COS θ SIN ψ = Sin(θ - ψ) のレシオメトリックな三角関数計算を行います。

従来の高精度抵抗器と比べ、これらのキャパシタは高い精度を得るために、精度比で使用されます。さらに、これらのキャパシタ (オペアンプと共に計算要素として使用) は、ドリフトやオペアンプのオフセットを解消するために、高いレートでサンプリングします。

DC 誤差の処理が統合され、電圧制御型オシレーターを駆動する速度電圧が出力されます。この VCO は速度積分器と組み合わせると増分積分器 (Type 2 サーボ フィードバック ループ) になります。

基準オシレーター

弊社のデザインの電力オシレーターも DDC 社の OSC-15802 です。このデバイスは RDC、同期、LVDT、RVDT、インダクトシン アプリケーションに適しています [5]。周波数および振幅の出力はそれぞれキャパシタと抵抗器でプログラム可能です。出力周波数レンジは 400Hz ～ 10kHz、出力電圧は 7Vrms です。図 4 にデバイスのブロック図を示します。

レゾルバーと RDC に与えられるオシレーター出力は基準信号の役割を果たします。

VIRTEX-5 FX30T FPGA と RDC のインターフェイス

私達のデザインでは、ザイリンクス Virtex®-5 FX30T FPGA を使用しました [6]。FPGA の I/O 電圧は 3.3V であるのに対し、RDC の電圧は 5V です。電圧変換トランシーバーを使用して、2 つのデバイス間の電圧互換性を実現しました。FPGA との内部接続は、図 5 に示すように、ザイリンクスが提供する GPIO IP コアによって確立されます。

わかりやすくするために、図 5 では単一のレゾルバー インターフェイスを持つ 1 つのチャネルだけを示します。

また、ザイリンクスの XBD ファイル作成については、Xcll 86 号 (日本語版) の「ザイリンクス　デザイン用のカスタム XBD のファイルの作成」をご覧ください。

デバイス ドライバーの詳細

このケースでは、FPGA に 20MHz の外部入力クロックを使用しました。この FPGA は、200MHz 周波数で動作するハード PowerPC® 440 コアを持ちます。RDC のタイミング ダイアグラムを図 6 および図 7 に示します。

RDC のタイミング ダイアグラムに従い、開発およびテストを行い、実際のハードウェアで機能が正しいことを確認しました [4]。デバイス ドライバーの実際のコードは、別途提供の XBD ファイルに収録されています。タイミング ダイアグラムに従い、ループに必要な遅延を生成しました。処理速度が 200MHz の場合、各カウントは 5 ナノ秒の遅延に相当します。

デバイス ドライバーには、RDC の初期化、制御信号の生成と RDC のチャネル A からの読み出し、制御信号の生成とチャネル B からの読み出しの 3 つのコード セクションがあります。RDC の初期化では、信号の方向とデフォルト値が設定されます。たとえば、次の文では、方向は FPGA から RDC へ「out」に設定されます。

```
XGpio_WriteReg(XPAR_RESOLUTION_CNTRL_CH_A_
   BASEADDR,XGPIO_TRI_OFFSET,0x000);
```

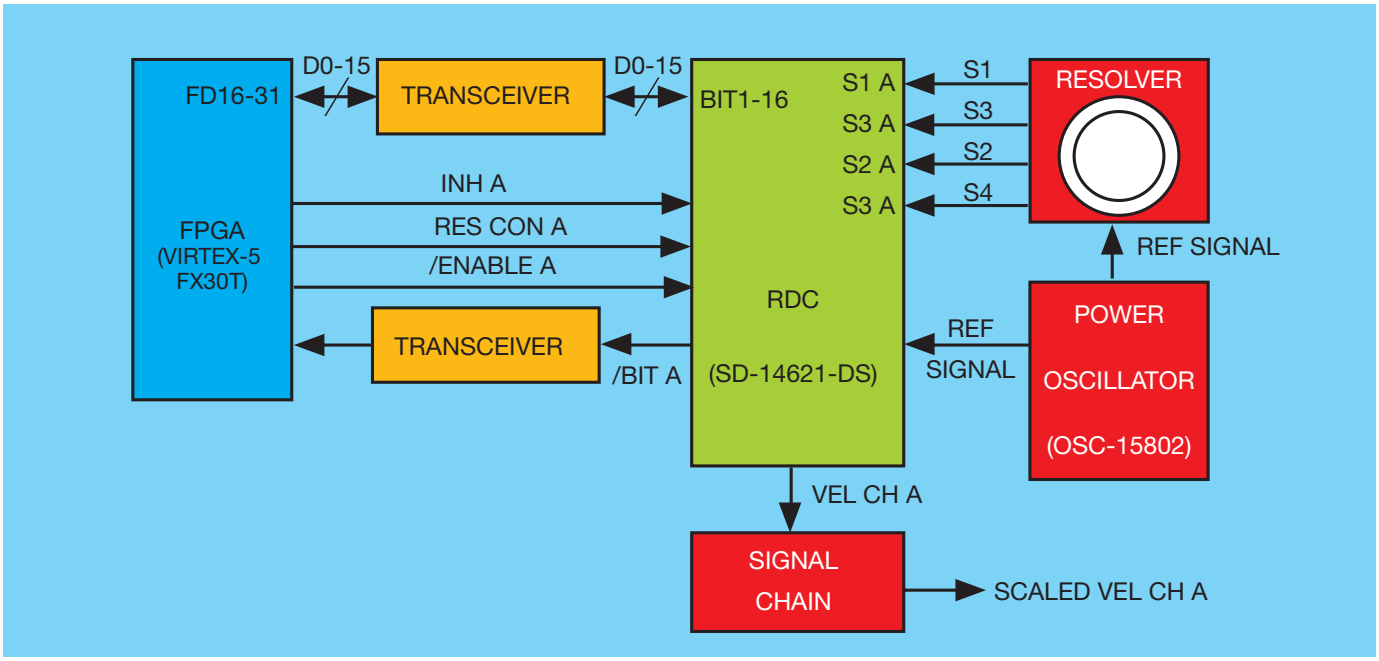


図 5 – RDC と Virtex-5 FPGA のインターフェイス (単一チャネル)

次の文では、「0x3」（つまり、high にプルアップ）と指定することで、16 ビットの分解能が設定されます。

```
XGpio_WriteReg(XPAR_RESOLUTION_CNTRL_CH_A_
   BASEADDR,XGPIO_DATA_OFFSET,0x03);
```

図 8 にコーディングのスナップショットを示します。注記：わかりやすくするために、チャネルが 1 つだけのコードを取り込みました。

これでおわかりのように、アングル トランスデューサーを使用することで、エンジニアは、優れた車輪をはじめ、多くの効率的な機械システムを作り出すことができます。レゾルバーはアングルトランスデューサーの特に有用なタイプで、FPGA と適切に組み合わせ、制御すると、さらに素晴らしい機械システムを考案できます。

参考資料

1. “Synchro/Resolver Conversion Handbook,” Data Device Corp.
2. John Gasking, “Resolver-to-Digital Conversion: A Simple and Cost-Effective Alternative to Optical Shaft Encoders,” AN-263, Analog Devices
3. Walt Kester, “Resolver to Digital Converter,” MT-030, Analog Devices
4. SD-14620 Series Data Sheet, Data Device Corp.
5. OSC-15802 Data Sheet, Data Device Corp.
6. “Virtex-5 Family Overview,” Xilinx

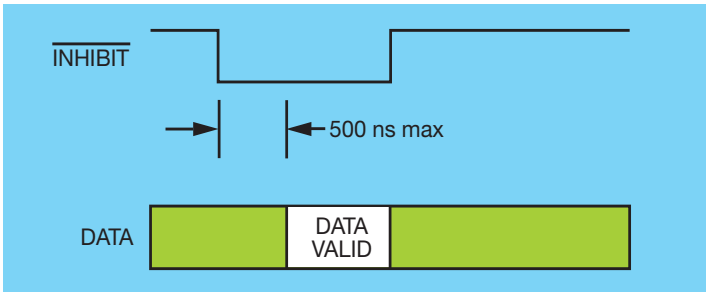


図 6 – INHIBIT タイミング

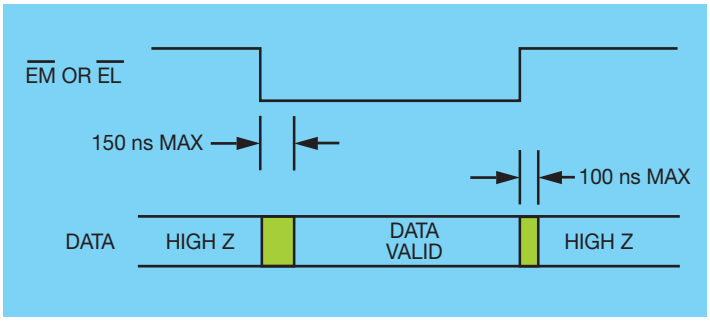


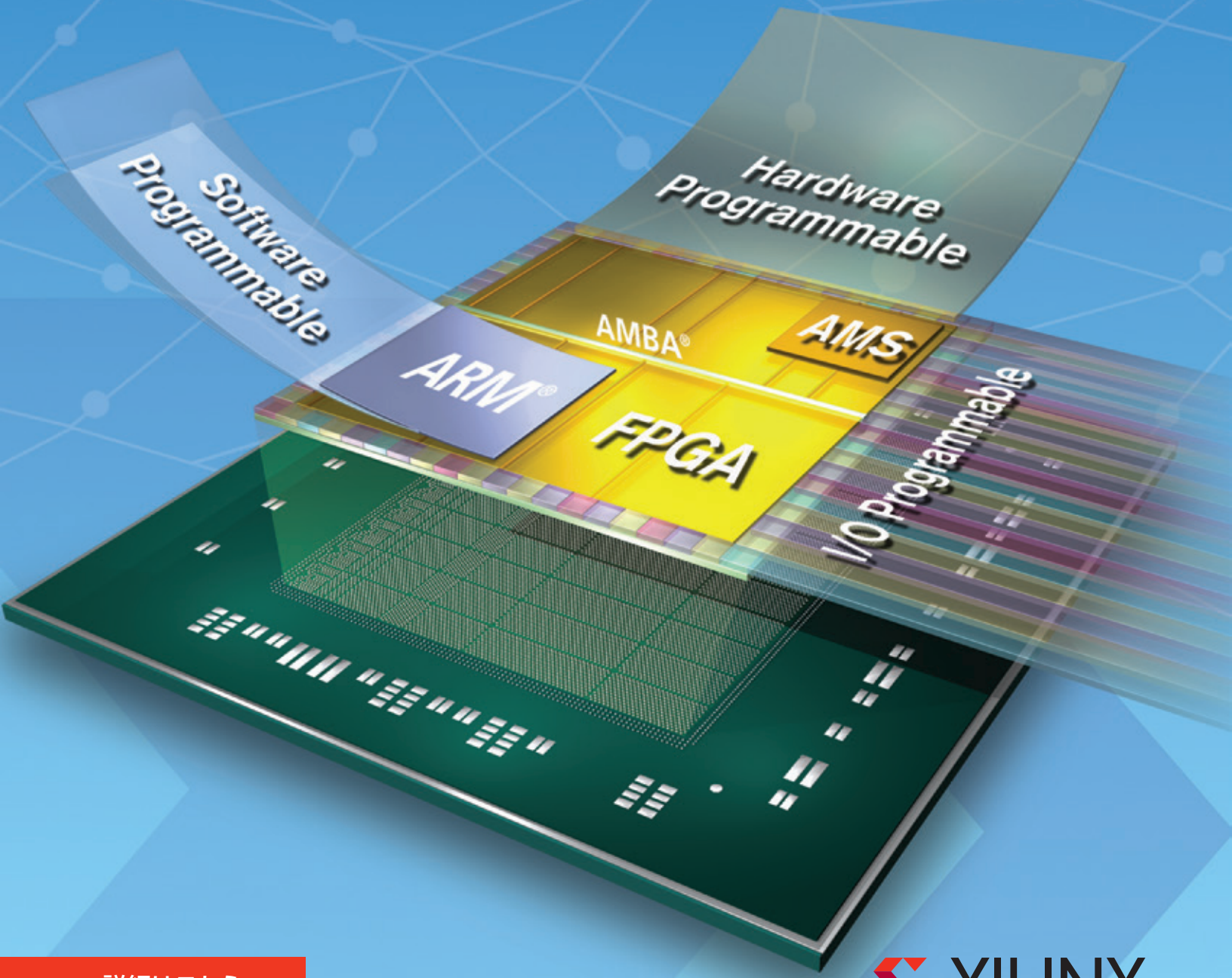
図 7 – ENABLE タイミング

```
seXGpio_WriteReg(XPAR_INHIBIT_CH_A_
BASEADDR,XGPIO_DATA_OFFSET,0x01);
for(i=0;i<=5;i++); //gives delay of 25 ns
          XGpio_WriteReg(XPAR_ENABLE_LSB_
CH_A_BIT_BASEADDR,XGPIO_DATA_OFFSET,0x01);

for(i=0;i<=5;i++);
XGpio_WriteReg(XPAR_INHIBIT_CH_A_BASEAD-
DR,XGPIO_DATA_OFFSET,0x00);
for(i=0;i<=2;i++);
XGpio_WriteReg(XPAR_ENABLE_LSB_CH_A_BIT_
BASEADDR,XGPIO_DATA_OFFSET,0x00);

for(i=0;i<=2;i++);
lsb_val=XGpio_ReadReg(XPAR_RDC_DATA_15_
TO_0_PINS_BASEADDR,XGPIO_DATA_OFFSET);

XGpio_WriteReg(XPAR_INHIBIT_CH_A_BASEAD-
DR,XGPIO_DATA_OFFSET,0x01);
      for(i=0;i<=5;i++);
XGpio_WriteReg(XPAR_ENABLE_LSB_CH_A_BIT_
BASEADDR,XGPIO_DATA_OFFSET,0x01);
for(i=0;i<=25;i++);

XGpio_WriteReg(XPAR_INHIBIT_CH_A_BASEAD-
DR,XGPIO_DATA_OFFSET,0x01);
for(i=0;i<=5;i++);
XGpio_WriteReg(XPAR_ENABLE_MSB_CH_A_BIT_
BASEADDR,XGPIO_DATA_OFFSET,0x01);
for(i=0;i<=5;i++);

XGpio_WriteReg(XPAR_INHIBIT_CH_A_BASEAD-
DR,XGPIO_DATA_OFFSET,0x00);
      for(i=0;i<=2;i++);
XGpio_WriteReg(XPAR_ENABLE_MSB_CH_A_BIT_
BASEADDR,XGPIO_DATA_OFFSET,0x00);
for(i=0;i<=2;i++);
msb_val=XGpio_ReadReg(XPAR_RDC_DATA_15_
TO_0_PINS_BASEADDR,XGPIO_DATA_OFFSET);

  lsb_val=lsb_val & 0x00ff;

  msb_val=msb_val & 0xff00;

  rdccount_cha = msb_val | lsb_val;

XGpio_WriteReg(XPAR_INHIBIT_CH_A_BA-
SEADDR,XGPIO_DATA_OFFSET,0X01);
for(i=0;i<=5;i++);

XGpio_WriteReg(XPAR_ENABLE_MSB_CH_A_
BIT_BASEADDR,XGPIO_DATA_OFFSET,0x01);
for(i=0;i<=20;i++);
```

図 8 – RDC デバイス ドライバー コードのスナップショット

A Generation Ahead

ハードウェア、ソフトウェア、I/O がプログラマブルなSoC



詳細はこちら

XILINX®
ALL PROGRAMMABLE

ザイリンクス株式会社

製品のお問い合わせは下記の販売代理店へどうぞ

■東京エレクトロン デバイス(株) TEL(045)443-4016 x2web@teldevice.co.jp ■アヴネット・インターニックス(株) TEL(03)5792-8210 EVAL-KITS-JP@avnet.com
■(株)PALTEK TEL(045)477-2005 nfo_pal@paltek.co.jp ■新光商事(株) TEL(03)6361-8086 X-Pro@shinko-sj.co.jp

Motor Drives Migrate to Zynq SoC with Help from MATLAB

MATLAB の支援により、モーター ドライブを Zynq SoC に移行

Tom Hill
Senior Manager, DSP Solutions
Xilinx, Inc.
tom.hill@xilinx.com

工業デザイナーは、ラピッド プロトタイプとモデルベース デザインを使用してモーター制御アルゴリズムを Zynq SoC 環境に移行できます。

1990 年代以降、モーター ドライブの開発者は、マルチチップ アーキテクチャを使用してモーター制御および処理要件をインプリメントしています。このアーキテクチャでは、個別のデジタル信号処理 (DSP) チップがモーター制御アルゴリズムを実行し、FPGA が高速 I/O およびネットワーキング プロトコルをインプリメントし、個別のプロセッサが実行制御機能を処理します。ザイリンクス® Zynq®-7000 All Programmable SoC が出現したことで、設計者はこうした機能を 1 つのデバイスに集約しながら、これ以外の処理タスクも統合できるようになりました。部品点数の削減と複雑さの解消により、性能と信頼性を向上させながら、システム コストを削減できます。

では、ドライブ開発者は、Zynq SoC を利用するために、確立された設計手法をどのように発展させたらいいのでしょうか。

工業デザイナーは長い間、シミュレーションと C コード生成を使用することで、DSP チップでのカスタム モーター アルゴリズムの開発にモデルベースのデザインを採用してきました。ザイリンクスと共同開発した MathWorks の新しいワークフローでは、モデルベース デザインを Zynq-7000 All Programmable SoC で提供されるプロセッシング システムとプログラマブル ロジックに拡張します。

モーター制御用の ZYNQ SOC

現在、最先端のモーター制御システムは、EtherCAT、Profinet、Powerlink、Sercos III などの産業用ネットワークと制御アルゴリズムの組み合わせで、計算リソースから処理帯域幅を引き出します。さらに、動作制御層、PLC 層、診断層、およびコミッショニング、保守、遠隔監視のためのユーザー インターフェイスなど、その他の要件も制御システムに集約されています。こうした要件は論理パーティションと物理パーティションに変換され、そのままプロセッシング システムに組み込まれる要素もあれば、ハードウェアを使用したオフロードやアクセラレーションに組み込まれる要素もあります。

ハードウェア プラットフォームは、堅牢でスケーラブルなシステムを選択する必要があります。ザイリンクスの Zynq SoC はこうした要件を満たし、ネットワーキング、動作、ソフト PLC、診断、遠隔保守機能に対応するための高性能プロセッシング システムと、ハードウェアによる高性能な機能の処理速度向上のためのプログラマブル ロジックを提供します。プロセッシング システムの面では、Zynq SoC はデュアルコア ARM® Cortex™ -A9 プロセッシング システムと NEON コプロセッサおよび

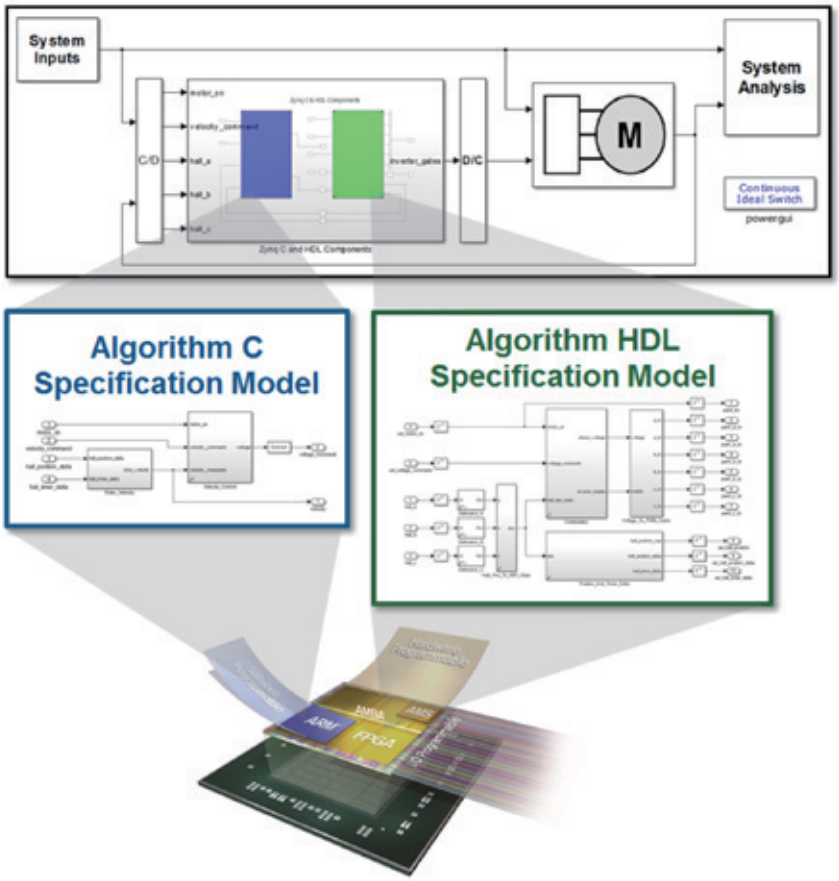


図 1 – C および HDL コード生成を使用した、
ZynqSoC に対する MathWorks 社のワークフロー

浮動小数点の拡張を組み合わせ、ソフトウェアの実行処理速度を高速化します。プログラマブル ロジックについては、デバイスは最大 444,000 のロジック セルと 2,200 の DSP48 スライスを持ち、膨大な処理帯域幅を供給します。5 つの高スループット AMBA®- 4 AXI の高速インターコネクトが、効果的帯域幅の 3,000 以上のピンと同等のものを使用して、プログラマブル ロジックをプロセッシング システムに緊密に連携させます。

表 1 に、Zynq SoC デバイスが実現可能な処理性能をリストします。

SIMULINK と Control System Toolbox を使用したプラントおよびモーターのモデリング

今日の制御アルゴリズムでは、システム時間およびシステム変数が数桁にも及び、ハードウェア / ソフトウェアのパーティショニングは面倒で繰り返しの多い時間のかかる作業です。図 2 に一般的なエレクトリカル ドライブを示します。電源は通常 50 ～ 60Hz で、平流電圧 (DC) を実現するために整流されます。この DC 電圧は可変周波数に変換され、モーター端子に供給する電力段を制御します。また、コントローラーは電流や電圧を含むモーターの基本的な変数を読み取る必要があります。同様に、速度を含む車軸の位置の読み取りや設定、および通信ネットワークや監視コントローラーから出されるコマンドの処理の必要もあります。

Simulink® は、マルチドメイン システム シミュレーションおよびモデルベースのデザインに対してブロック図環境を提供します。これは、制御アルゴリズムおよびプラントモデルを含むシステムのシミュレーションに適しています。Control System Toolbox などの MathWorks 社製品は、Simulink でモデリングされた制御システムを体系的に解析、設計、調整するために広く使用されている方法に基づき、多様な「アプリケーション」を提供します。Simulink でシステム モデリングを行うと、次のように、リスクを低減しながら、モーター制御システムの開発期間を短縮できます。

- 破損リスクの低減 : シミュレーションにより、製品段階のハードウェアでテストする前に新しい制御システム アルゴリズムを詳細に検査でき、ドライブの電子機器やモーター、その他システム コンポーネントを破損するリスクが低減されます。
- システム統合の促進 : サポート スタッフは新しい制御システム アルゴリズムをプロダクション システムに統合する必要が

Elements	Performance (up to)
Processors (each)	1 GHz
Processors (aggregate)	5,000 DMIPs
DSP (each)	741 MHz
DSP (aggregate)	2,662 GMACs
Transceivers (each)	12.5 Gbps
Transceivers (aggregate)	200 Gbps
Software acceleration	10x

表 1 – Zynq SoC の処理性能

あります。新しいコントローラーの展開はスタッフの貴重な時間を消費し、長い時間を要する可能性があります。

- 装置の可用性に対する依存性の低減：カスタム ドライブの電子機器や電気モーターが開発中の場合や、制御システム設計者がアクセスできる場所に配置されていない場合など、プロダクション環境自体が使用できないことがあります。

これらを踏まえ、シミュレーションが製品段階のハードウェアでのテストに代わる優れた選択肢を提供します。Simulink などのシミュレーション環境が、電気機械コンポーネントの既存の基本単位のライブラリからプラント モデルを作成するためのフレームワークとなり、新しい制御システム アーキテクチャをプラント モデルと照合して評価できます。

さらに、システム モデルを最終的なプロダクション システムだけでなく、ラピッド プロトタイプ環境にリンクさせることで、スケジュールに対するリスクを低減します。ラピッド プロトタイプ フローにより、アルゴリズムの開発者はハードウェア設計者に依存することなく製品試作を実行できます。高度に自動化されたプロセスでプラットフォーム固有のサポート パッケージを使用して、システムのハードウェアとソフトウェアのコンポーネントをデザイン テンプレートに展開し、これを固有のハードウェア開発プラットフォームにコンパイルできます。ハードウェアおよびソフトウェアのデザイン チームは、これと同じハードウェアおよびソフトウェア コンポーネントを一切変更することなく、最終的なプロダクション システムで再利用でき、開発時間を短縮し、エラーの発生を減らすことができます。

AVNET INTELLIGENT DRIVES KIT を使用したラピッド プロトタイプ

設計者は Avnet Zynq-7000 AP SoC / Analog Devices Intelligent Drives Kit と Simulink および Zynq SoC ワークフローの組み合わせにより、モーター制御アプリケーション用のラピッドプロトタイプ システムを実現できます。Zynq SoC に Analog Devices 社の最新の高精度データ コンバーターとデジタル アイソレーションを組み合わせたこのキットでは、高性能モーター制御とデュアル ギガビット イーサネットの産業用ネットワーク接続の構築が可能です (http://www.xilinx.com/products/boards-and-kits/1-490M1P.htm 参照)。

キットには、Avnet ZedBoard 7020 ベースボード、24 ボルトの外部電源 (キットに同梱) でブラシレス DC およびステッパー モーターを駆動できる Analog Devices 社の AD-FMCMOTCON1-EBZ モジュール、定格回転数が 4,000RPM でホール効果センサーと 1,250-CPR インデックス付きエンコーダーを装備した 24V BLDC モーターが付いています。また、フィールド オリエンテッド コントロール (FOC) の Zynq SoC リファレンス デザインと Analog Devices 社のドライバー、アプリケーション ソフトウェア、ソース コードを含む Ubuntu Linux フレームワークも含まれます。

例：台形モーター制御

Simulink のシミュレーションを使用して、図 1 の台形モーター制御システムにこのワークフローを適用し、シミュレーションしたプラントでコントローラーを評価した後、Intelligent Drives Kit を使用してコントローラーを試作します。最後のステップとして、ハードウェア テストの結果を使用して Simulink モデルを検証します。

この例では、キットを使用し、基本的な台形コントローラーで、アルミニウム ディスクの形で慣性負荷を駆動します。コントローラーの主要コンポーネントは次のとおりです。

- ホール効果センサー：モーターの位置を検出する。
- 速度エスティメーター：センサー信号に基づいて回転子の速度を計算する。
- 6 ステップ整流器：回転子の位置と速度に基づいて、位相電圧およびインバーター イネーブル信号を計算する。
- パルス幅変調 (PWM)：駆動回路を介してコントローラー出力を駆動する。

まず、制御ループ解析に適したシステムのビヘイビアー制御ループ モデルを使用します。最初に、シミュレーションのモデルを評価するために、パルス テストを行い、1 秒あたり 150 ラジアンの回転レートを 2 秒間指令した後、停止に戻します。制御ループの比例積分 (PI) コントローラー ゲインの調整により、ごくわずかのオーバーシュートで 1.2 秒の整定時間を実現できます (制御ループのシミュレーション結果は図 3 で紫色の網掛けになっている信号として表示されます。この例の詳細は mathworks.com/zidk で提供します)。

次に、制御ループ ゲインを設定した状態で、

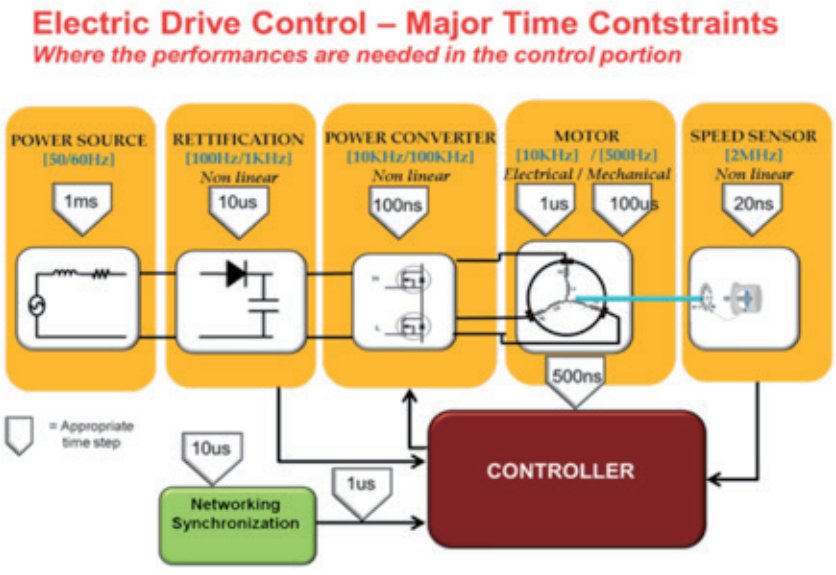


図 2 – エレクトリカルドライブ コントローラーの主な時間制約

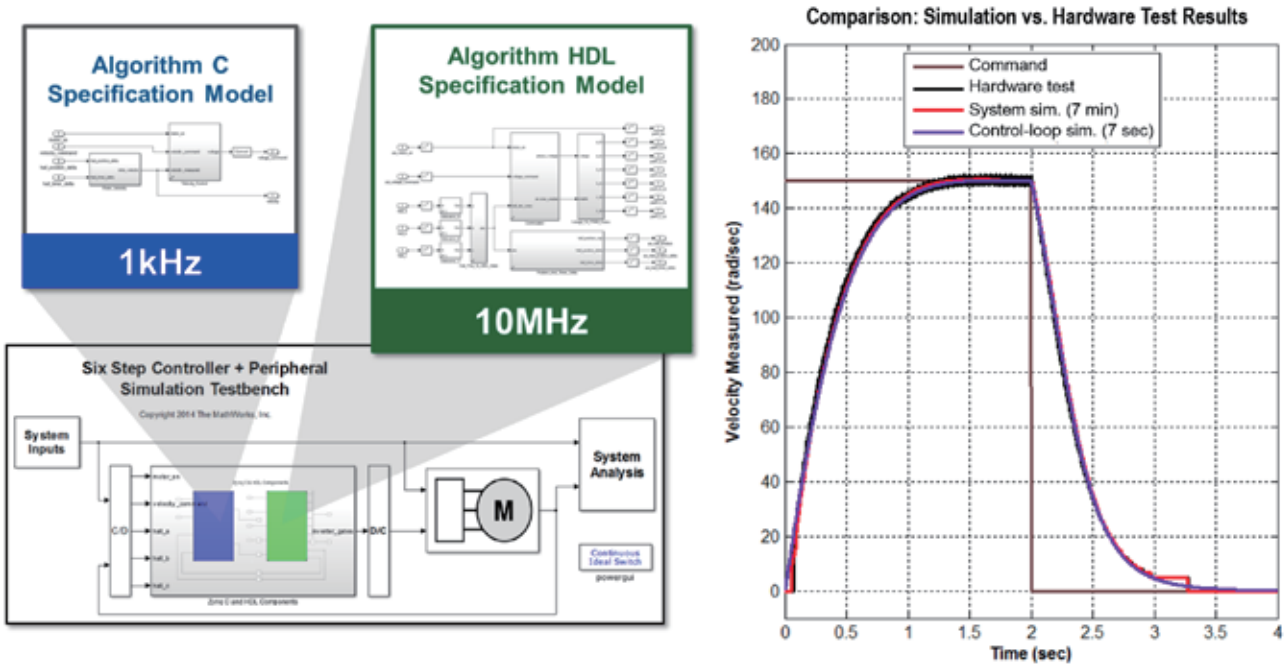


図 3 – ハードウェア結果と照合・検証するために使用されるハードウェアおよびソフトウェアのシミュレーション モデル

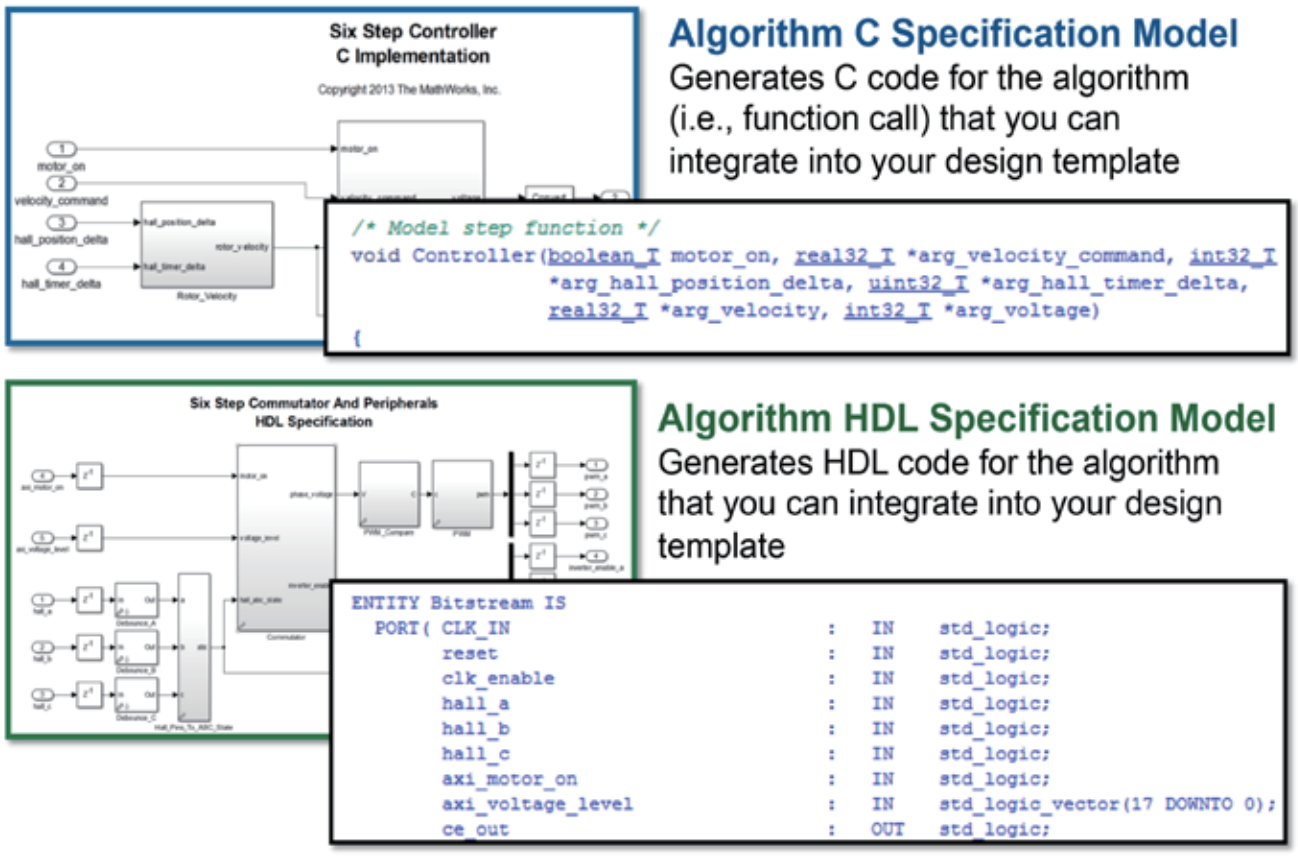


図 4 – パーティションされた Simulink モデルから生成される C および HDL コード

コントローラーのより正確なシステム モデルをテストします。制御ループ モデルと対照的に、システム モデルにはより詳細なドライブの電子機器モデルが組み込まれています。さらに重要なことに、PWM およびホール効果センサーの処理のためのタイミング精度が高いモデルなど、コントローラーやペリフェラルのインプリメンテーションを指定する詳細モデルが含まれています。

Zynq SoC 用のコントローラーを分割し、1kHz で ARM コアで動作する速度コントローラーと速度エスティメーター、および Zynq SoC のプログラマブル ロジックで動作する整流器、ホール センサー、PWM にしました。

制御ループとシステム モデルのシミュレーション結果を比較できます (システム モデル結果は図 3 で赤色の信号として表示されます)。通常、モーターの速度がゼロに近づいている場合を除き、波形間で非常に良好な一致が得られます。モーターの速度がゼロに近づくと、ホール センサーの粗粒度が顕著になり、モーターの車軸の 1 回転あたりのインデックス パルス数はわずか 6 です。忠実度が低い制御ループ モデルの実行時間がわずか 7 秒であるのに対し、この再現性の高いシステム モデルは 4 秒のシミュレーションを 7 分かけて実行します。制御システムの設計者にとって、ここでのポイントは、コントローラーの選択肢をさらに評価できるだけの十分な精度を制御ループ モデルが有し、システム モデルを使用してハードウェア テストを行う前に検証可能であるという確信がシミュレーション結果から与られることです。

こうしたことから、Intelligent Drives Kit でコントローラーをプロトタイプする準備が整いました。Zynq SoC ガイド ワークフローから、ARM コアとプログラマブル ロジックをターゲットとしたサブシステムに対して、パーティションした Simulink モデルから C および HDL コードを生成できます (図 4)。

このワークフローで、MathWorks 社の HDL Coder を使用して、ARM コアで動作する実行ファイルを構築し、AXI バスを介してコアと実行ファイルの間のインターフェイスを確立するために、Zynq SoC デバイスのプログラマブル ロジックで動作する IP コアを生成します。

ビットストリームをプログラマブル ロジックにロードし、実行ファイルを ARM コアで走らせた状態で、hardware-in-the-loop(HIL) テストを実行できます。このテストでは、シミュレーションしたプラント モデルではなく hardware-in-the-loop を使用しているため、ドライブの電子機器、モーター、センサーのモデルを削除した Simulink テストベンチ モデルを使用します。テスト結果のチェック (シミュレーション結果との比較) を容易にするために、モーターの車軸速度の計測結果などのデータを ARM コアのメモリ内に格納しておくよう Zynq SoC をセットアップできます (図 3 の黒い網掛け信号がハードウェア テストの結果を示します)。こうすることで、テストベンチにパルス入力を印加することにより、結果をテストの最後に処理および視覚化する MATLAB セッションにアップロードできます。これで、シミュレーションで行ったテストをハードウェアで正確に再現できます。プロトタイプの結果は、ホール センサーによるモーター速度計測のばらつきも含め、シミュレーション結果と完全に一致します。

この概要では、Zynq SoC の MathWorks ワークフローにより、シミュレーションおよび試作で使用するモデルベース デザインがどのようにして実行可能になるかを示しています。引き続き製品に移行するには、生成された C および HDL コードを Vivado® Design Suite にインポートし、システムの全体的なインプリメンテーションに必要とされる実行ルーチン、ネットワーク用 IP、その他デザイン コンポーネントと統合できます。

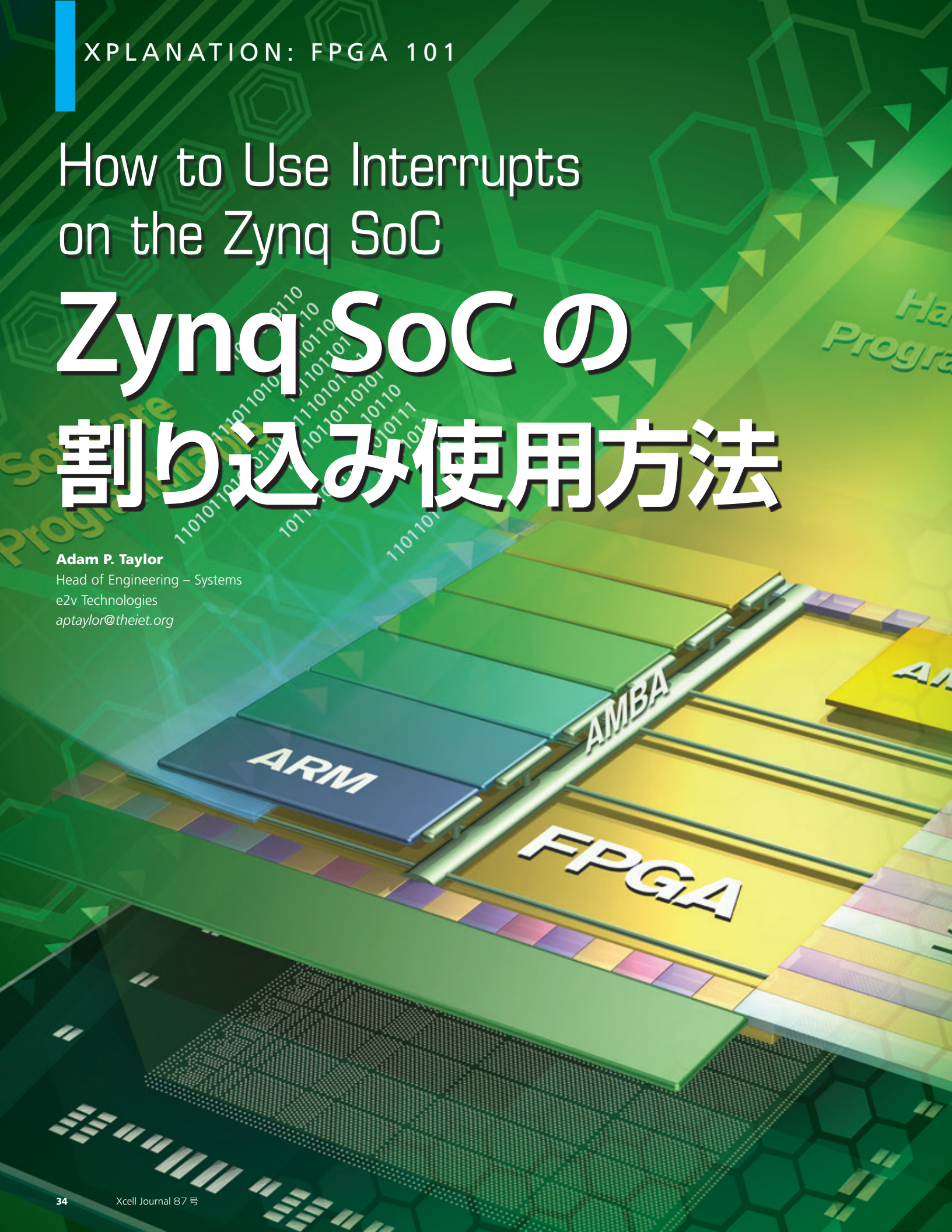
本稿に示すモデルのダウンロード、および Zynq-7000 All Programmable SoC / Avnet 社の Analog Devices Intelligent Drives Kit でのモデルベース デザインの詳しい使用方法については、mathworks.com/zidk を参照してください。また、このページから、完全な フィールド オリエンテッド コントロール (FOC) モデルを Zynq SoC デバイスにインプリメントする Simulink モデルを参照でき たり、この例をより詳しく紹介するビデオを再生できます。

MathWorks 製品がザイリンクス Zynq-7000 All Programmable SoC ファミリをいかにサポートするかの情報については、mathworks.com/zynq を参照してください。

How to Use Interrupts on the Zynq SoC

Zynq SoC の割り込み使用方法

Adam P. Taylor
Head of Engineering – Systems
e2v Technologies
aptaylor@theiet.org



リアルタイム コンピューティングでは
しばしば、イベントに対して割り込みの
迅速な応答が要求されます。
Zynq SoC の割り込み構造の原理を
把握すれば、割り込み駆動システムの
設計も難しくありません。

エンベデッド プロセッシングにおいて、割り込みはプロセッサの現在のアクティビティを一時的に中断させる信号です。プロセッサは現在の状態を保存し、割り込みサービス ルーチンを実行して、割り込みの理由に対処します。割り込みの発生源として、次の 3 つのソースがあります。

- ハードウェア ：プロセッサに直接接続される電子的手段で発生する信号
- ソフトウェア ：プロセッサによって読み込まれるソフトウェア命令
- 例外 ： エラーまたは例外的イベントが発生したときにプロセッサによって生成される例外処理

また、どのソースの場合も、割り込みはマスカブルとノンマスカブルのいずれかに分類されます。マスカブル割り込みは、割り込みマスク レジスタの当該ビットをセットすることで安全に無視することができます。一方、ノンマスカブル割り込みは通常タイマーやウォッチドッグに使用されるため無視できません。

割り込みはエッジトリガで発生する場合とレベルトリガで発生する場合があります。ザイリンクス® Zynq®-7000 All Programmable SoC では、後述するようにどちらの割り込み設定もサポートします。

割り込み駆動方式を使用する理由

リアルタイム デザインではしばしば割り込み駆動方式が必要になります。理由は簡単で、多くのシステムには随時処理を必要とするさまざまな入力 (キーボード、マウス、プッシュボタン、センサーなど) があるためです。通常、こうしたデバイスからの入力は現在実行されているプロセスまたはタスクと非同期であるため、イベントがいつ発生するかを常に予測できるとは限りません。

割り込みを使用することで、プロセッサはイベントが発生するまで処理を続行でき、イベントが発生した時点でイベントに対処できます。また、この割り込み駆動方式を採用することで、プログラムが同期方式で外部デバイスのステータスをアクティブにサンプリングするポーリング方式よりもイベントに対する応答時間を短縮できます。

ZYNQ SOC の割り込み構造

プロセッサの高度化に伴い、割り込みの発生源となりうるソースも増えました。図 1 に示すように、Zynq SoC は汎用割り込みコントローラー (GIC) を使用して割り込みを処理します。GIC が処理する割り込みのソースは次のとおりです。

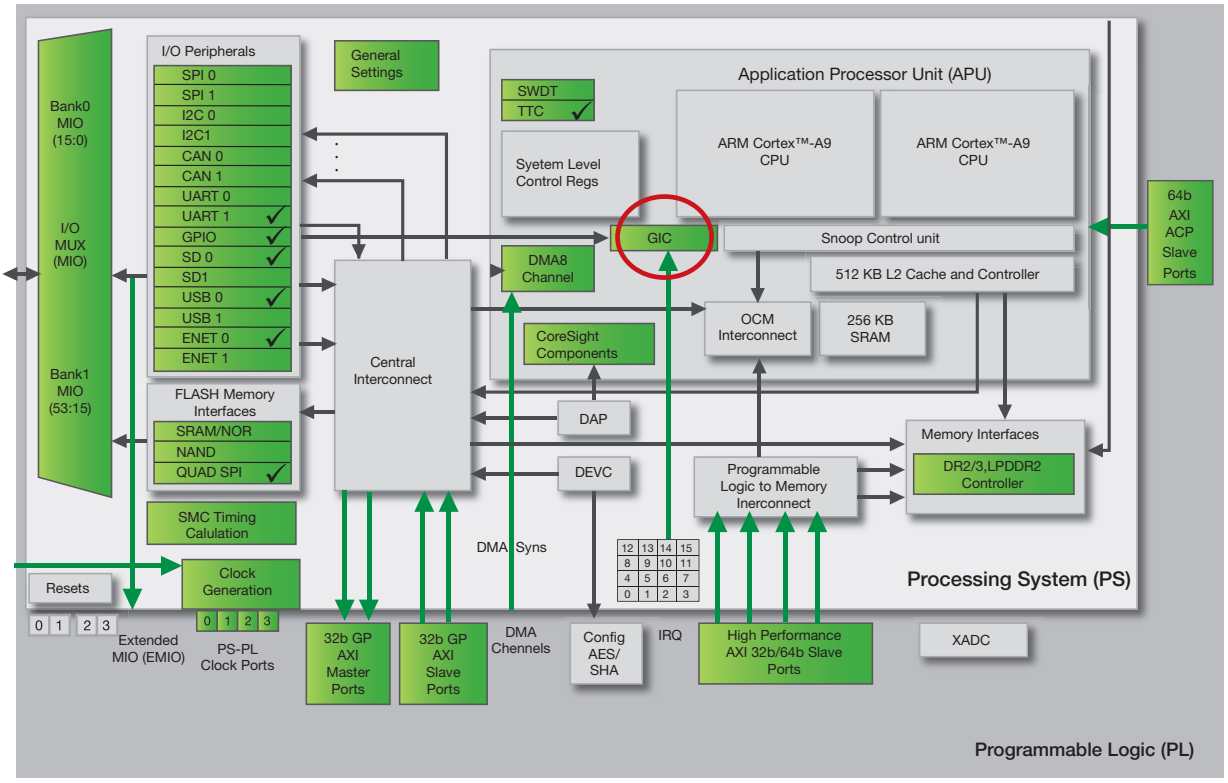


図 1 – 汎用割り込みコントローラーは赤丸で囲ってあります。

- ソフトウェア生成割り込み ：このような割り込みは各プロセッサに 16 あります。Zynq SoC の一方または両方の ARM® Cortex™-A9 プロセッサ コアに対して割り込みを実行できます。
- 共有ペリフェラル割り込み ：この割り込みは合計 60 で、I/O 周辺機器から、またはデバイスのプログラマブル ロジック (PL) 側との間で発生します。Zynq SoC の 2 つの CPU 間で共有されます。
- プライベート ペリフェラル割り込み：このカテゴリの 5 つの割り込みは各 CPU (たとえば、CPU タイマー、CPU ウォッチドッグ タイマー、専用の PL-CPU 割り込み) にプライベートな割り込みです。

共有ペリフェラル割り込みは非常に柔軟であり魅力があります。I/O 周辺機器 (合計 44 の割り込み) または FPGA ロジック (合計 16 の割り込み) から、いずれかの CPU に配線できます。ただし、図 2 に示すように、I/O 周辺機器からデバイスのプログラマブル ロジック側に割り込みを配線することもできます。

ZYNQ SOC の割り込み処理

Zynq SoC 内で割り込みが発生した場合、プロセッサは次のアクションを行います。

1. 割り込みが保留中として示される。
2. プロセッサは現在のスレッドの実行を停止する。
3. プロセッサは割り込み処理が完了した時点で処理を続行できるよう、スレッドの状態をスタックに保存する。
4. プロセッサは割り込みの処理方法が定義されている割り込みサービス ルーチンを実行する。
5. プロセッサは中断したスレッドをスタックから復元し、処理を再開する。

割り込みは非同期イベントであるため、複数の割り込みが同時に発生する可能性があります。この問題に対処するために、プロセッサは割り込みに優先順位を付け、保留中の割り込みの中で最も優先順位が高い割り込みを最初に処理します。

この割り込み構造を正しくインプリメントするには、割り込みが発生したときに実行するアクションを定義するための割り込みサービス ルーチンと割り込みを設定するための割り込みセットアップの 2 つのファンクションを構築する必要があります。割り込みセットアップは再利用可能なルーチンで、さまざまな割り込みを構築できます。このルーチンはシステム内のすべての割り込みに汎用的であり、汎用 I/O (GPIO) の割り込みを構築して有効にできます。

SDK での割り込みの使用

割り込みはザイリンクス ソフトウェア開発キット (SDK) 内でサポートされ、スタンドアロンのボード サポート パッケージ (BSP) を使用して仮想化 システム上にインプリメントできます。BSP には、この割り込み駆動 システムの構築作業を大幅に軽減するファンクションが複数含まれ、次のヘッダー ファイル内で提供されています。

- Xparameters.h ：このファイルには、プロセッサのアドレス空間とデバイス ID が含まれます。
- Xscugic.h ： このファイルには、GIC をコンフィギュレーションおよび使用するためのドライバーが格納されています。
- Xil_exception.h ：このファイルには、Cortex-A9 の例外ファンクションが含まれます。

ハードウェア 周辺機器に対処するために、使用するデバイス (つまり、GIC) のアドレス範囲およびデバイス ID を把握しておく必要があります。これは、ほとんど BSP ヘッダー ファイル xparameters 内で提供されます。ただし、割り込み ID は xparameters_ps.h から提供されます (xparameters.h ファイルに含まれているため、このヘッダー ファイルをソース コード内に宣言する必要はありません)。次に示すように、この「ID」というラベルの割り込み (GPIO_Interrupt_ID) をソース ファイル内で使用できます。

```
#define GPIO_DEVICE_ID    XPAR_XGPIOPS_0_DEVICE_ID
#define INTC_DEVICE_ID  XPAR_SCUGIC_SINGLE_DEVICE_ID
#define GPIO_INTERRUPT_ID    XPS_GPIO_INT_ID
```

このシンプルな例では、ボタン 押下に続き、割り込みを生成するように Zynq SoC の GPIO を構成します。割り込みをセットアップするには、2 つの静的グローバル変数と上記で定義した割り込み ID が必要になります。

```
static XScuGic Intc; // Interrupt Controller Driver
static XGpioPs Gpio; //GPIO Device
```

割り込みセットアップ ファンクション内で、Zynq SoC の例外の初期化、GIC のコンフィギュレーションと初期化、割り込み処理ハードウェアへの GIC の接続を行う必要があります。Xil_exception.h および Xscugic.h ファイルでは、このようなタスクに必要なファンクションを提供します。結果として、次のようなコードが生成されます。

```
//GIC config
XScuGic_Config *IntcConfig;
Xil_ExceptionInit();

//initialize the GIC
IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);

XScuGic_CfgInitialize(GicInstancePtr, IntcConfig,
IntcConfig->CpuBaseAddress);

//connect to the hardware
Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IN-
T,(Xil_ExceptionHandler)XScuGic_InterruptHandler,
       GicInstancePtr);
```

同じ割り込みコンフィギュレーション ルーチン内で割り込みとして機能するように GPIO をコンフィギュレーションする場合は、バンクまたは個別ピンを設定できます。このタスクは、たとえば次のように、xgpiops.h 内で提供されているファンクションを使用して実現できます。

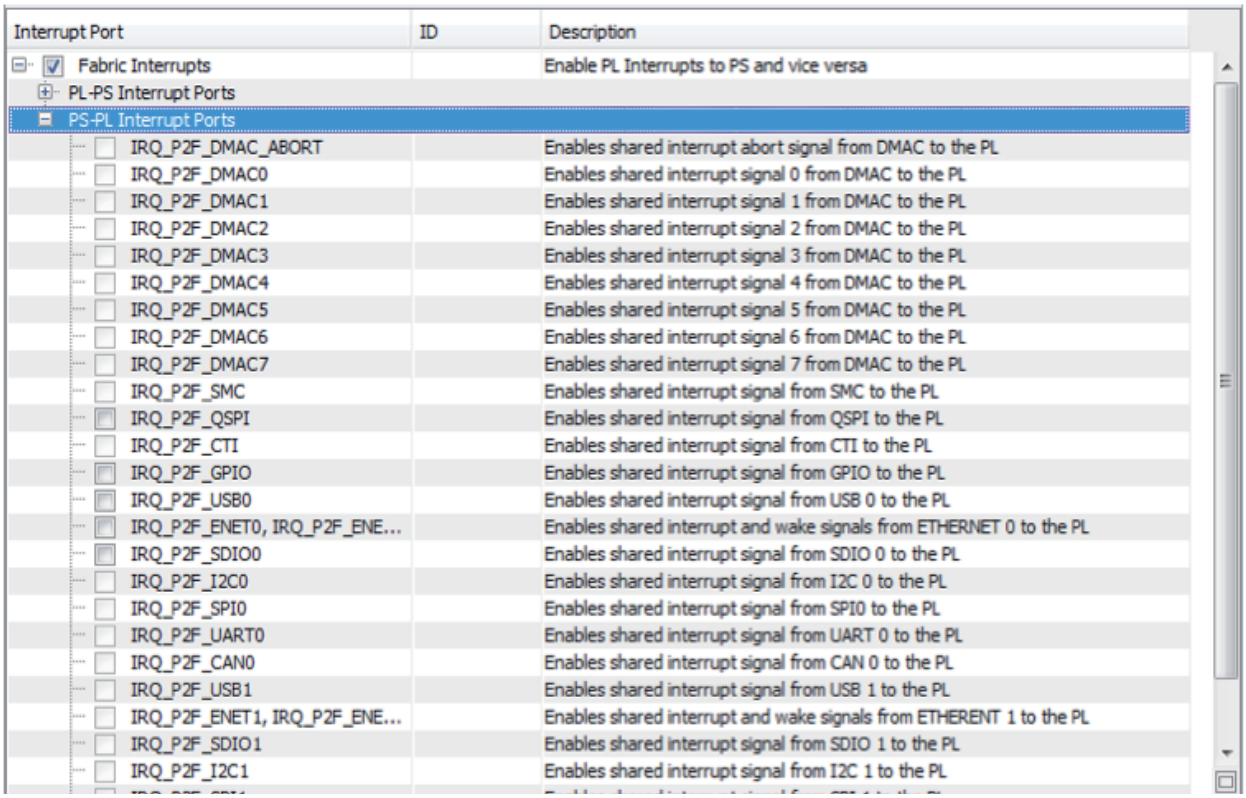


図 2 – プロセッシング システムとプログラマブル ロジックの間で使用可能な割り込みです。

```
void XGpioPs_IntrEnable(XGpioPs *InstancePtr, u8
    Bank, u32 Mask);
void XGpioPs_IntrEnablePin(XGpioPs *InstancePtr,
    int Pin);
```

当然ですが、割り込みを正しく設定する必要もあります。たとえば、エッジトリガにするのかレベルトリガにするのか。さらに、ファンクションを使用して、どのエッジおよびレベルを実現できるのかを設定します。

```
void XGpioPs_SetIntrTypePin(XGpioPs *InstancePtr,
int Pin, u8 IrqType);
```

ここで、IrqType は xgpiops.h 内の次の 5 つの定義のいずれかで定義されます。

```
#define XGPIOPS_IRQ_TYPE_EDGE_RISING 0 /**<
    Interrupt on Rising edge */
#define XGPIOPS_IRQ_TYPE_EDGE_FALLING 1 /**<
    Interrupt Falling edge */
#define XGPIOPS_IRQ_TYPE_EDGE_BOTH 2 /**<
    Interrupt on both edges */
#define XGPIOPS_IRQ_TYPE_LEVEL_HIGH 3 /**<
    Interrupt on high level */
#define XGPIOPS_IRQ_TYPE_LEVEL_LOW 4 /**<
    Interrupt on low level */
```

バンク イネーブルを使用する場合は、割り込みをイネーブルにするピンがどのバンクにあるのか把握する必要があります。Zynq SoC は、最大 118 の GPIO をサポートします。このコンフィギュレーションでは、EMIO (64 ピン) と共に、すべての MIO (54 ピン) が GPIO として使用されます。このコンフィギュレーションを 4 つのバンクに分割でき、各バンクに最大 32 ピンを割り当てることができます。

このセットアップ ファンクションは、ファンクションを使用する割り込みが発生した場合に呼び出される割り込みサービス ルーチンも定義します。

```
XGpioPs_SetCallbackHandler(Gpio,
    (void *)Gpio, IntrHandler);
```

アプリケーションの定義に応じて、割り込みサービス ルーチンは単純にも複雑にもなります。この例では、ボタンが押されるたびに、LED のステータスがオンとオフの間で切り替わります。また、割り込みサービス ルーチンは、ボタンが押されるたびに、メッセージをコンソールに出力します。

```
static void IntrHandler(void *CallBackRef, int
Bank, u32 Status)
{
        int delay;
          XGpioPs *Gpioint = (XGpioPs *)
          CallBackRef;
          XGpioPs_IntrClearPin(Gpioint, pbsw);
        printf("****button pressed****\n\r");
        toggle = !toggle;
        XGpioPs_WritePin(Gpioint, ledpin, toggle);
        for( delay = 0; delay < LED_DELAY; delay++)
        //wait
        {}
}
```

プライベート タイマーの例

Zynq SoC では、いくつかのタイマーとウォッチドッグを使用できます。ある CPU 専用のものと、両方の CPU が利用できる共有リソースの場合があります。これらのコンポーネントをデザイン内で効率的に使用する場合、割り込みが必要になります。次のようなタイマーおよびウォッチドッグがあります。

- CPU 32 ビット タイマー (SCUTIMER)。CPU の半分のクロック周波数で駆動できます。
- CPU 32 ビット ウォッチドッグ (SCUWDT)。CPU の半分のクロック周波数で駆動できます。
- 共有型 64 ビット グローバル タイマー (GT)。CPU の半分のクロック周波数で駆動できます (CPU ごとに固有の 64 ビット コンパレータがあり、各 CPU 専用の割り込みを駆動する GT と一緒に使用されます)。
- システム ウォッチドッグ タイマー (WDT)。CPU クロックまたは外部ソースから駆動できます。
- トリプル タイマー カウンター (TTC) のペア。それぞれのカウンターに 3 つの独立したタイマーがあります。TTC は、CPU クロックによって駆動するか、またはプログラマブル ロジックの MIO または EMIO からの外部ソースによって駆動できます。

提供されているタイマーおよびウォッチドッグの利点を最大限に引き出すために、Zynq SoC の割り込みを利用できるようにする必要があります。最もコンフィギュレーションが簡単なのはプライベート タイマーです。Zynq SoC のほとんどの周辺機器と同様、このタイマーでは、リソースを効率的に使用するためのいくつかのファンクションおよびマクロがあらかじめ定義されています。これらは次の中に含まれます。

```
#include "xscutimer.h"
```

このファイルには、初期化やセルフテストなどの機能を提供するファンクション (マクロ) が格納されています。また、このファイル内のファンクションはタイマーの開始と停止、タイマーの管理 (リスタート、タイマーが切れたかどうかの確認、タイマーのロード、オート ローディングのイネーブル / ディスエーブル) も行います。もう 1 つのジョブは、タイマー割り込みのセットアップ、イネーブル、ディスエーブル、解除、および管理です。最後に、これらのファンクションはプリスケーラの呼び出しと設定も行います。

タイマー自体は次の 4 つのレジスタで制御されます。

- プライベート タイマー ロード レジスタ : このレジスタはオート リロード モードで使用されます。オート リロードがイネーブルである場合にプライベート タイマー カウンター レジスタにリロードされる値が含まれます。
- プライベート タイマー カウンター レジスタ : 実際のカウンターです。イネーブルである場合、このレジスタがゼロに達すると、割り込みイベント フラグが設定されます。
- プライベート タイマー制御レジスタ :制御レジスタはタイマー、オート リロード モード、割り込み生成をイネーブルまたはディス

エーブルにします。タイマー用のプリスケーラも含みます。

- プライベート タイマー割り込みステータス レジスタ：このレジスタには、プライベート タイマー割り込みステータス イベント フラグが含まれます。

GPIO の使用に関しては、タイマーのセットアップに必要なタイマー デバイス ID とタイマー割り込み ID は XParameters.h ファイル内に含まれます。紹介する例では、以前開発したプッシュボタン式割り込みを使用します。ボタンを押すと、タイマーがロードし、動き始めます（オート リロード モードではありません）。タイマーが切れると、割り込みが生成され、STDOUT にメッセージが書き出されます。その後、割り込みは解除され、次にボタンが押されるまで待機します。この例では、必ず同じ値をカウンターにロードします。そのため、ファイル上部での宣言で、次のようにタイマー カウント値が定義されます。

```
#define TIMER_LOAD_VALUE    0xFFFFFFFF
```

次のステージでは、プライベート タイマーを設定および初期化し、タイマーのカウント値をロードします。

```
 //timer initialisation
 TMRConfigPtr = XScuTimer_LookupConfig
   (TIMER_DEVICE_ID);
 XScuTimer_CfgInitialize(&Timer,
   TMRConfigPtr,TMRConfigPtr->BaseAddr);
//load the timer
 XScuTimer_LoadTimer(&Timer, TIMER_LOAD_VALUE);
```

また、割り込みセットアップ サブルーチンをアップデートして、タイマー割り込みを GIC に接続し、タイマー割り込みをイネーブルにする必要があります。

```
 //set up the timer interrupt
  XScuGic_Connect(GicInstancePtr, TimerIntrId,
(Xil_ExceptionHandler)TimerIntrHandler,
    (void *)TimerInstancePtr);
 //enable the interrupt for the Timer at GIC
 XScuGic_Enable(GicInstancePtr, TimerIntrId);
 //enable interrupt on the timer
 XScuTimer_EnableInterrupt(TimerInstancePtr);
```

ここで、TimerIntrHandler は割り込みが発生したときに呼び出されるファンクションの名前であり、タイマー割り込みを GIC およびタイマー内部でイネーブルにする必要があります。

タイマー割り込みサービス ルーチンは非常にシンプルです。保留中の割り込みを解除して、次のように STDOUT にメッセージを書き出すだけです。

```
static void TimerIntrHandler(void *CallBackRef)
{

  XScuTimer *TimerInstancePtr =
     (XScuTimer *) CallBackRef;
  XScuTimer_ClearInterruptStatus(TimerInstancePtr);
  printf("****Timer Event!!!!!!!!!!!!!****\n\r");
```

このアクションが完了すると、最後に GPIO 割り込みサービス ルーチンを変更して、次のようにボタンが押されるたびにタイマーをスタートするようにします。

```
 //load timer
 XScuTimer_LoadTimer(&Timer, TIMER_LOAD_VALUE);
 //start timer
 XScuTimer_Start(&Timer);
```

このためには、まずタイマー値をタイマーにロードし、タイマー スタート ファンクションを呼び出します。これで、図 3 からわかるように、再度プッシュボタン割り込みを解除し、処理を再開できます。

最初は、多くのエンジニアが割り込み駆動 システムのデザインに不安を抱いています。しかし、Zynq SoC のアーキテクチャおよび SDK で提供されるドライバーと汎用割り込みコントローラーの組み合わせにより、割り込み駆動 システムを非常に迅速かつ効率的に稼働できるようになります。

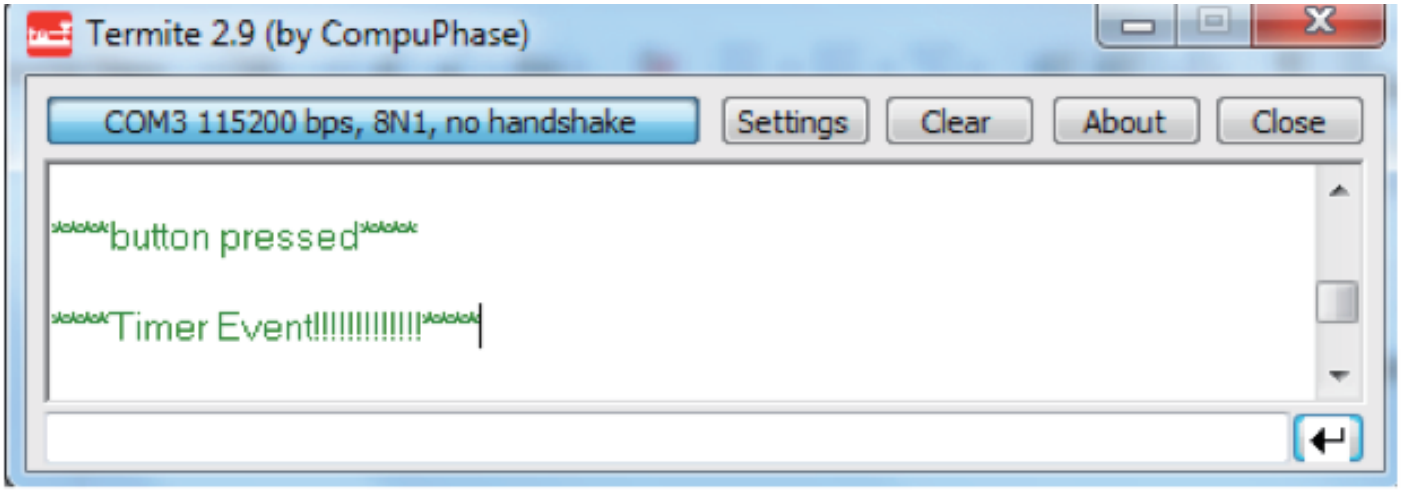
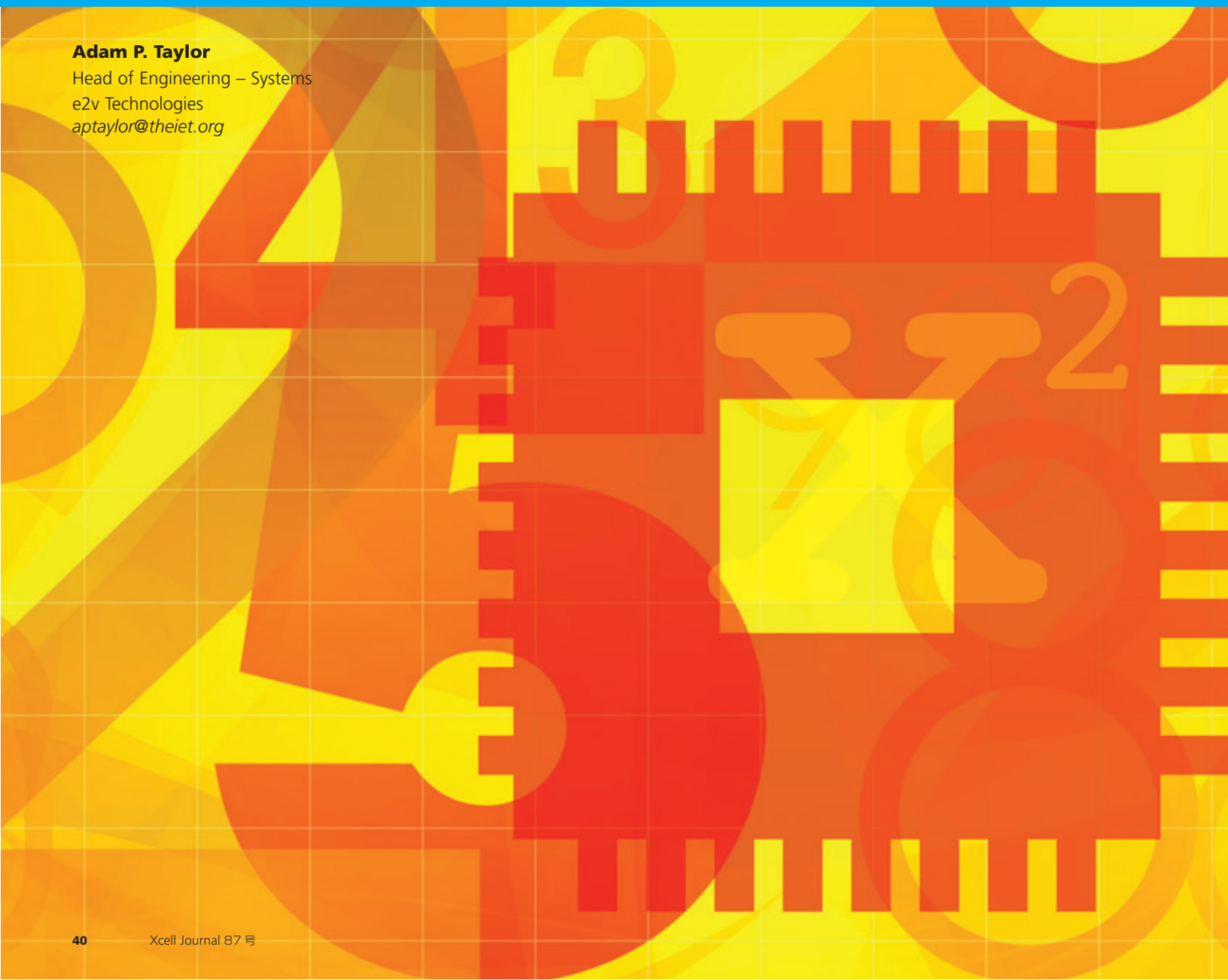


図 3 – この画面は、GPIO およびタイマー割り込みイベント出力の例です。

Calculating Mathematically Complex Functions

数学的に複雑な関数の計算

Adam P. Taylor
Head of Engineering – Systems
e2v Technologies
aptaylor@theiet.org



FPGA の大きな利点の 1 つとして、エンベデッド DSP ブロックを使用して、最も複雑な数学的伝達関数にも対処できます。多項式近似はそのための 1 つの良い手段です。

FPGA はその柔軟性と性能により、産業、科学、軍事など、複雑な数学の問題や伝達関数の計算を必要とする多くのアプリケーションに進出してきました。さらにクリティカルなアプリケーションでは、厳しい精度や算出遅延時間が求められることは珍しくありません。

FPGA を使用して数学関数をインプリメントする場合、通常エンジニアは固定小数点演算を選択します (Xcell Journal 英語版 80 号「The Basics of FPGA Mathematics」を参照。http://issuu.com/xcelljournal/docs/xcell80/44?e=2232228/2002872)。また、超越関数の計算に使用できる CORDIC などの多くのアルゴリズムがあります (Xcell Journal 日本語版 79&80 合併号「FPGA デザイン内での CORDIC アルゴリズムの使用方法」(P-42) を参照。http://issuu.com/xcell-journal-japanese/docs/xcell_79-80)。

しかし、数学的に非常に複雑な関数に直面した場合、要求の厳しい関数をそのまま FPGA 内にインプリメントするよりも効率的な対処方法があります。そのような代替手法 (特に、その中の 1 つである多項式近似) を理解するために、まず問題を明確にしてみましょう。

問題の説明

このような数学的に複雑な伝達関数の 1 つの例は FPGA 内部にあり、白金抵抗温度計 (PRT) を監視し、PRT の電気抵抗を温度に変換するというものです。この変換には、通常 Callendar-Van Dusen 方程式を使用します。次に示すような簡易式の場合、この式で 0 ～ 660℃ の温度を求めることができます。

$$R = R_0 \times (1 + a \times t + b \times t^2)$$

ここで、R_0 は 0℃ での抵抗、a と b は PRT の係数、t は温度です。

ただし、実際には、抵抗を温度に変換する必要があります。このため、与えられた抵抗に対する温度が結果となるように、この式を書き換えます。PRT を使用するほとんどのシステムでは、電子回路を使用して PRT の抵抗を測定し、その後次のように書き換えられた式を使用して FPGA に温度を計算させるよう、電子機器を設計します。

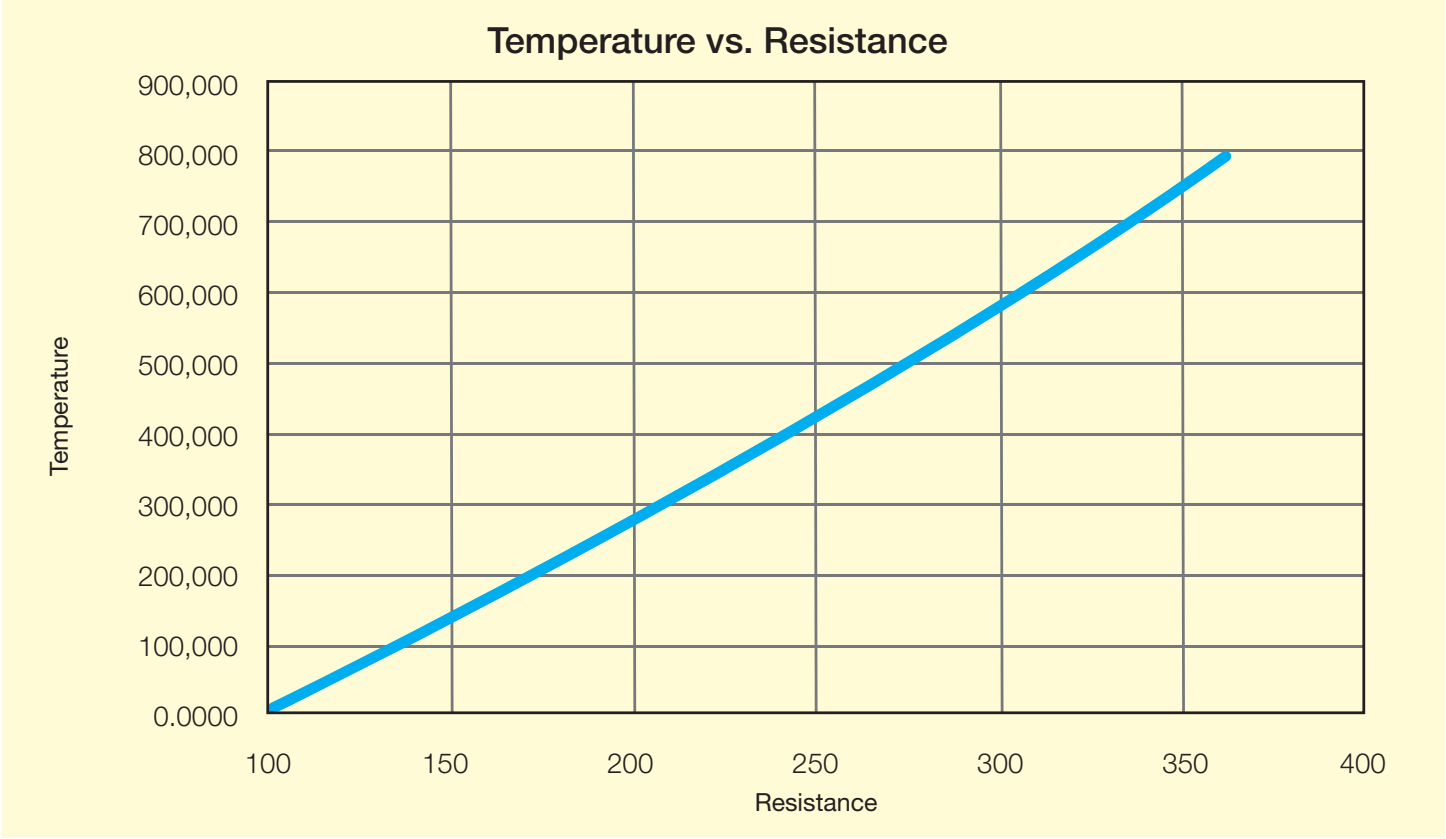


図 1 – プロットした伝達関数

$$t = \frac{-R_0 \times a + \sqrt{R_0^2 \times a^2 - 4 \times R_0 \times a \times (R_0 - R)}}{2 \times R_0 \times b}$$

FPGA 内でこの式をインプリメントする作業は、たとえ経験豊富な FPGA エンジニアであっても厄介なことです。得られた抵抗と温度をプロットすると、図 1 に示すようなグラフになります。このグラフから、応答の非線形性は明確です。

書き換えた伝達関数を FPGA に直接インプリメントすることは、実際に必要となるデザイン作業と検証（バウンダリおよびコーナーケース全体での精度および機能の確認）の両面でかなりの難題になる可能性があります。多くのエンジニアが、プロジェクトのタイム スケールを守るために、デザインおよび検証作業の負担を軽減しながら関数をインプリメントするためのさまざまな方法を模索します。その 1 つとして、ルックアップ テーブルを使用して、LUT 内のポイント間を線形補間してカーブ上のいくつかのポイントを格納する手法が考えられます。

この手法は、精度要件やルックアップ テーブル内に格納される要素数によっては、要求を満たすかもしれません。とは言え、線形インターポレーター関数をデザイン内に取り込む必要が残ります。この関数は数学的に高度であり、2 のべき乗以外の除算が含まれることが多く、より複雑になります。

FPGA リソースの活用

こうしたタイプの伝達関数をインプリメントするために使用可能な方法がもう 1 つあります。FPGA の性質そのものを利用する方法です。Xilinx® Spartan®-6 や 7 シリーズ Artix®、Kintex®、Virtex® などの今日の FPGA には、従来のルックアップ テーブルやフリップフロップをはるかに超えるものが含まれています。また、DSP スライス、ブロック RAM、分散 RAM が組み込まれており、さらに PCIe® やイーサネット エンドポイント、高速シリアル リンクなど多くの最先端ハード IP コアが搭載されています。

DSP スライスは、48 ビットのアキュムレータを提供するため通常 DSP48 と呼ばれます。ただし、25 x 18 ビット幅の乗算器、加算 / 減算機能なども提供します。こうした内部 RAM 構造および DSP スライスを使用することで、非常に簡単に伝達関数をインプリメントできます。

多項式近似

FPGA の DSP および RAM に富むアーキテクチャを利用する 1 つの方法が多項式近似です。この方法を使用するには、まず MATLAB® や Excel などの数学プログラムで入力値範囲を対象にして、数学関数をプロットします。次に、近似曲線方程式が精度要件を満たしていれば、該当するデータセットに多項式近似曲線を追加して、数学的に複雑な関数の代わりに近似曲線の式を FPGA 内にインプリメントできます。

多項式近似曲線の追加機能を備えたほとんどの数学プログラムで、次数（または多項式の項の数）を選択できます。次数が大きいほど、近似精度が向上します。ただし、

FPGA 内にインプリメントする項は多くなります。現在使用している伝達関数の例に Microsoft Excel でこのプロセスを実行したところ、図 2 に示すような近似曲線および方程式が得られました。この例では、多項式次数を 4 にしました。

インプリメントする伝達関数の多項式近似が得られたら、同じ解析ツール (この例では Excel) を使用して、本来の伝達関数に対する精度を再確認します。温度を監視している現在のケースでは、特に精度要求が厳しいわけではなく、測定結果の精度が +/-1℃ になれば問題ありません。とは言え、測定の範囲やインプリメントする伝達関数によっては、1 つの多項式を使用するだけでは実現が困難な場合もあります。この問題に対処するには、どうしたらいいでしょうか。

入力値によって選択される複数近似曲線

1 つの多項式で伝達関数の入力範囲全体にわたって十分な精度を実現できない場合は、多項式を増やすだけです。入力範囲全体で使用する複数の多項式定数を生成する限り、この方法を使用できます。つまり、入力値が所定の境界を超えると、新しい定数のセットが読み込まれます。

温度の例を続けると、最初の多項式は 0 ～ 268℃ の間で +/-1℃ の精度を実現します。多くのアプリケーションでは、これで十分すぎるほどです。しかし、もし広い動作範囲と 300℃ までのトレランスが必要である場合は、当初の手法では、デザイン要件を満たすことができません。分割型手法を使用して、269 ～ 300℃ の範囲をプロットし、この出力範囲により高い精度を実現できる異なる多項式を取得することで、この問題に対処できます (図 3 を参照)。

要するに、インプリメンテーションでは、事前に計算した 268℃ に相当する範囲を入力値が上回るまで、最初の多項式の定数を使用します。この範囲を超えると、精度要件を維持するために、2 番目の定数のセットを使用します。

このように、目的の精度を実現するために、伝達関数をいくつかのセグメントに分割できます。伝達関数全体で均一にセグメントを分割できます (つまり、同じ値 X の 10 個のセグメントに分割するなど)。または、目的の精度を実現するために、伝達関数の中で精度の実現がより困難な部分に重点を置き、非均一な間隔で必要なセグメントに分割することもできます。

インプリメンテーションを決定する際に考慮すべきトレードオフの中で、均一に分割する方が非均一な場合よりもメモリの占有面積がより多く必要になる可能性があるということを頭に入れて置いてください。インプリメントする伝達関数によっては、非均一に分割した方が、かなりの節約になる可能性があります。

比較

当然ですが、前述したように伝達関数をインプリメントするために他の方法を使用することもできます。多項式近似を除き、ソフト

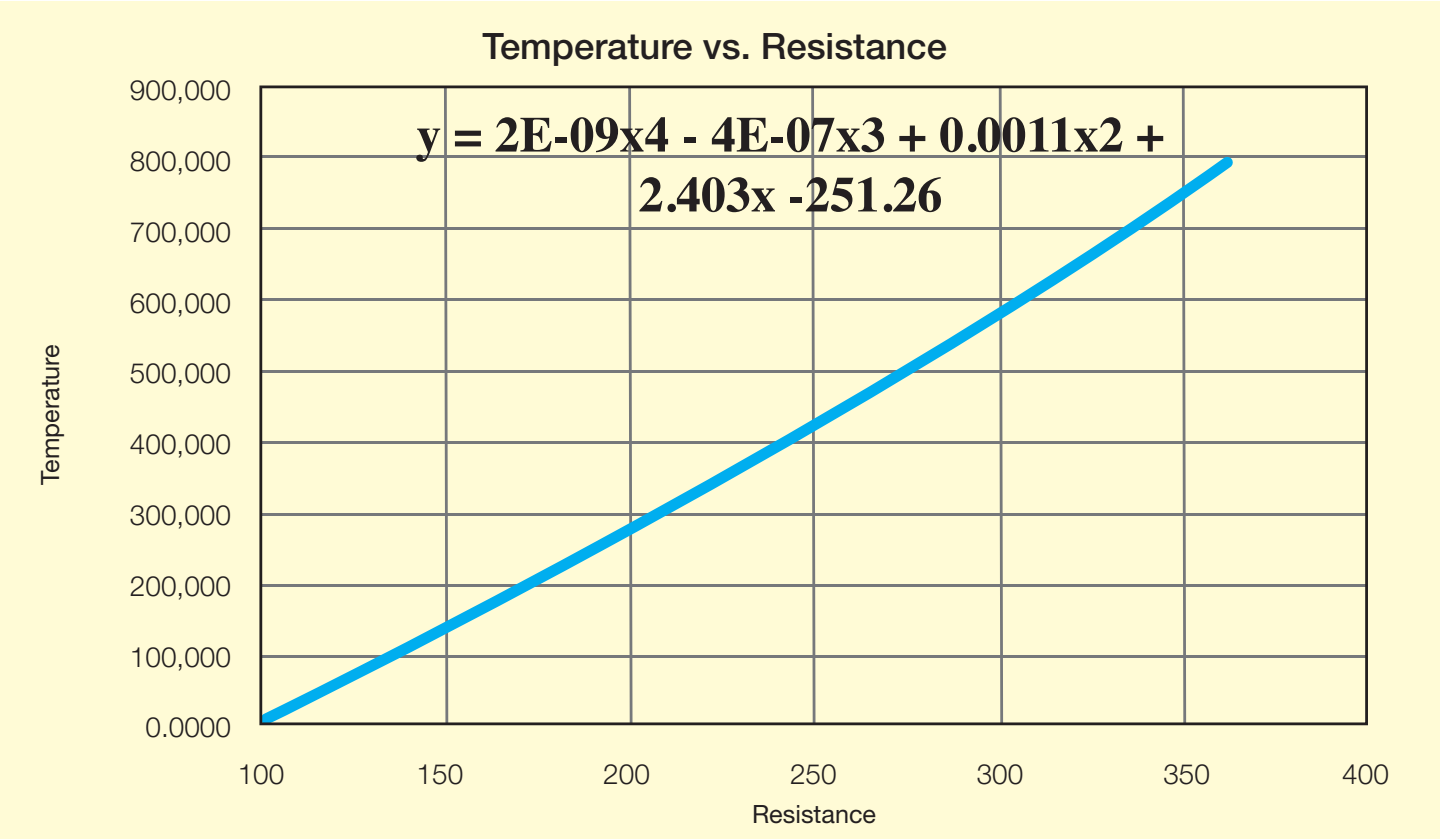


図 2 – 温度伝達関数の近似曲線および多項式

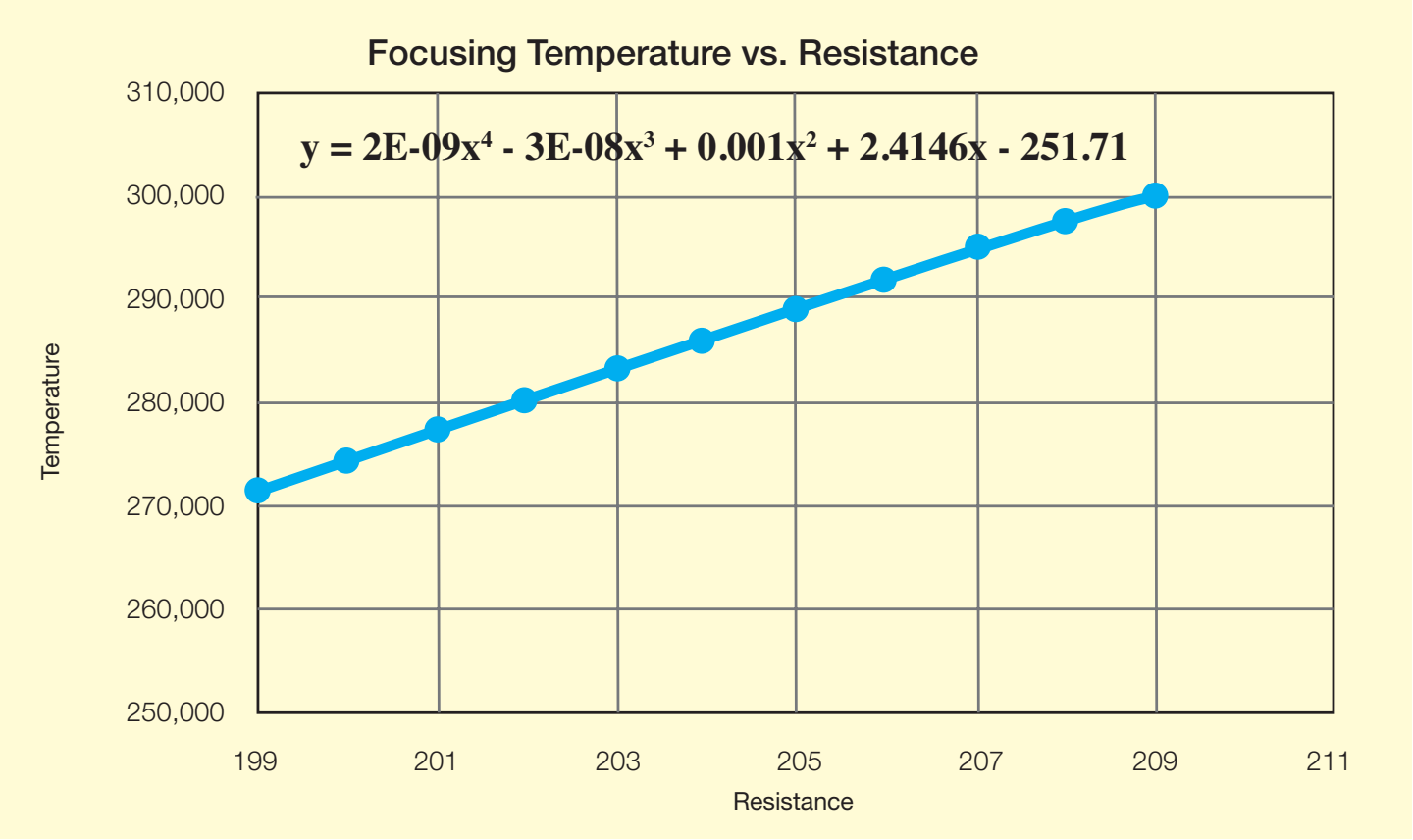


図 3 – 269～300℃ の範囲のプロットでより正確な結果を実現

ウェア ルーチン、ルックアップ テーブル、補間型ルックアップ テーブル、CORDIC の 4 つの方法が通常使用されます。

伝達関数の計算にソフトウェアを使用すると、プロセッサを追加する必要があるため、システム アーキテクチャが複雑になります (デザインが複雑化、BOM コストが増大化するなど)。たとえデザイン チームがザイリンクス Zynq®-7000 All Programmable SoC などのシステム オン チップを使用してこの欠点を克服したとしても、問題は残ります。第一に、ソフトウェアで伝達関数を計算するためにかかる時間はロジックで実現するよりもずっと長くなり、システムの応答時間が低下します。実際、サンプル デザインで使用するものなどの伝達関数の計算は、Zynq SoC のプログラマブル ロジック側に処理をオフロードすべき典型的な例です。

2 番目の事前に計算された入力値を含むルックアップ テーブルを使用する手法の有効性は、入力値の範囲および幅に依存して変化する可能性があります。結果として、あっという間に大きな LUT となり、FPGA 内に多くの RAM が必要となる場合があります。この方法を使用した場合、FPGA によっては、使用可能なリソースを上回るリソース数が必要になったり、デザイン内の他のモジュールの要件と競合したりする可能性があります。ただし、もちろん、この方法を使用すると、非常に迅速に結果が「計算」されるというメリットがあります。

3 つ目の補間型ルックアップ テーブルを使用する手法は、すでに検討したもので、完全 LUT 手法で必要なメモリ ロケーションの数を減らすことができます。この方法では、エンジニアは FPGA 内に線形補間関数を作成しなければならず、多少複雑になります。それでも、最後のオプションである CORDIC よりははるかに簡単です。

CORDIC アルゴリズムでは、正弦、余弦、乗算、除算、平方根などの超越関数をインプリメントできます。このため、CORDIC アルゴリズムと基本的な数学ブロックの組み合わせを使用して、伝達関数を正確にインプリメントできます。この方法を使用すると、より精度を上げることができます。しかし、複雑な伝達関数の場合、精度が向上する代わりに、デザインや検証により多くの時間がかかります。当然、この方法でインプリメントしたデバイスの動作周波数に影響が生じます。

このため、多項式近似は 4 つのオプションの妥協点を提示し、性能、精度、インプリメンテーションの占有面積における最適なトレードオフを実現します。

インプリメンテーションの容易さ

エンジニアはだれでも、デバイス リソースの最適な使用を実現する FPGA の生成を望みます。多項式近似を使用すると、FPGA で提供される乗算器および RAM に富む環境の利点を活かしながら、最初は数学的に非常に複雑な伝達関数であると思われるものを、こうしたリソースを使用して簡単にインプリメントできます。

XILINX® ウェブセミナー Web Seminar

All Programmable FPGA、SoC、3D IC の世界的なリーディング プロバイダーの
ザイリンクスが提供するプログラマブル ロジックからプログラマブル システム
インテグレーションのさまざまな機能と活用方法をご紹介します。
コストを抑え、最大のパフォーマンスを実現するための最新情報を手に入れてください。

ニーズに合わせたプログラムを各種取り揃えて好評配信中!!

New!! 新セミナー登場
ザイリンクス All Programmable ソリューションで実現する機能安全

FPGA入門編

FPGA の基本を理解したい方へ FPGA の全体概要を解説した入門編と、ものづくりにチャレンジする経営者、技術管理者の方へ FPGA を採用する利点をご説明します。

- ▶ 30分で判る! FPGA入門
- ▶ 15分で判る! FPGA採用理由

FPGA/SoC 活用編

ザイリンクス FPGA/SoC を使った最先端デザインの設計手法や、さまざまなアプリケーション設計に求められるデザイン チャレンジに対するソリューションをご紹介・解説します。

- ▶ Zynq SoC を使用したマルチチャンネル リアルタイム ビデオ プロセッサの設計
- ▶ Zynq SoC を使用した最先端 エンベデッド システムの設計 ～アクセレータでのソフトウェア ボトルネックの解消方法～
- ▶ システムレベル セキュリティの強化: All Programmable SoC で OS を超える
- ▶ 7 シリーズ ターゲット デザイン プラットフォーム

開発ツール編

プログラマブルデバイスである FPGA の設計には開発ツールがキーになります。ザイリンクスが提供するユーザー フレンドリーな開発ツールの特徴や使い方、先端設計メソドロジについて解説します。

- ▶ 次世代FPGA設計手法セミナー PlanAhead デザイン解析ツール ～ 第1部、第2部、第3部、デモ ～
- ▶ AMBA AXI4 テクニカルセミナー

FPGA/SoC 概要編

FPGA の世界トップシェアを誇るザイリンクスが提案するソリューションや、ザイリンクスの最先端 FPGA の詳細を解説します。

- ▶ Zynq-7000 SoC アーキテクチャとエコシステム
- ▶ 28nm ザイリンクス 7 シリーズ FPGA のアジャイル ミックスド シグナル テクノロジ

セミナー内容の詳細／ご視聴は今すぐこちらから ▶▶▶ http://japan.xilinx.com/webseminar/

Make Slow Software Run Fast with Vivado HLS

低速ソフトウェアを Vivado HLS により高速化

コーディング ボトルネックの悩みを一気に解決したい場合、非常に効果的な組み合わせの高位合成と Zynq SoC を検討すべきです。

David C. Black
Senior Member of Technical Staff
Doulos
david.black@doulos.com

コーディングに最善を尽くしたのに、期待したソフトウェアの実行速度が得られなかったという経験はありませんか。私はあります。「コードの一部でも、それほど高くない複数のカスタム プロセッサまたはカスタム ハードウェアに移植できさえすれば」と思ったことはありませんか。しかし結局は、あなたのアプリケーションは数ある中の 1 つにすぎず、カスタム ハードウェアの構築には時間とお金がかかる、という結論に至ってしまうのです。

ザイリンクス® の高位合成ツール Vivado® HLS の話を耳にしたとき、私はこの問題をもう一度考えてみました。デュアルコア ARM® Cortex™-A9 プロセッサと FPGA ファブリックから構成される Zynq®-7000 All Programmable SoC と高位合成の組み合わせは、デザインの新しい可能性を開きます。このツール クラスは、C、C++、または SystemC のソース コードから高度に調整された RTL を生成します。そして、このテクノロジには多くの提供者が存在し、近年普及率が増加しています。

でも、本当に Vivado HLS を使用するだけで要求の厳しい計算を行うことができるとしても、あれほど低速のコードをハードウェアに移行するのはどれほど大変なものなのか。結論としては、私は通常 C++ でコードを作成し、Vivado HLS も C/C++ を入力として使用していたので、ARM プロセッサ コアは、私が従来の環境のままでソフトウェアの大部分を実行できることを意味していました。事実、ザイリンクスはこのためにソフトウェア開発キット (SDK) と PetaLinux も提供しています。

アーキテクチャ上の問題

ソフトウェアの観点からこの変換を考え始めると、ソフトウェア インターフェイスが気になり出しました。結局、ハードウェア インターフェイス処理に特化したハードウェアを HLS が生成します。ソフトウェアを高速化するために、コプロセッサやハードウェア アクセラレータのような簡単にアクセスできるものが必要でした。また、新しいコンパイラも作成したくなかったのです。残りのソフトウェアとデータを簡単に交換するために、入力を取り込み、後でその結果を読み出すことができる単純なメモリ ロケーションのようなインターフェイスが必要でした。

ここで私は発見しました。Vivado HLS は、比較的少ない作業で AXI スレーブを作成するという目的をサポートしています。この機能は、アクセラレータの作成は結局のところそれほど難しくないのではないかという思いを起こさせ、私は可能性を探るために単純な例をコーディングしました。その結果は私にとって嬉しい驚きでした。

私が使用した手法を確認し、その結果について考えてみましょう。

私の例に関して言えば、加算や乗算など単純な行列演算のモデリングを選択しました。固定のサイズに制限したくなかったので、入力アレイとそれぞれのサイズの両方を指定する必要がありました。理想的なインターフェイスは、図 1 のコードに示すように、すべての値を関数の単純な引数として取り込むことです。

ハードウェアへのインターフェイスでは、関数の引数をメモリ ロケーションにマッピングするための簡単な方法が必要になります。図 2 に、このマッピングをサポートするためのメモリ レイアウトを示します。レジスタは、行列のレイアウトと必要な演算に関する情報を格納します。コマンド レジスタは必要な演算を示します。これによって、複数の単純な演算を 1 つのハードウェアに統合できます。ステータス レジスタは単に、演算が処理中なのか正常に終了したのかを知る手段です。デバイスが割り込みもサポートしていると理想的です。

ハードウェア デザインに話を戻すと、Vivado

```
Matrix operand1(5,10), operand2(10,5), product(10,10);
int status;
status = matrix_op(MUL, operand1, operand2, product); // product = operand1 * operand2;
if (status != 0) cout << "ERROR: multiplication failed" << endl;
```

図 1 – アクセラレータの呼び出し例

Addr	Register name	Dir	Bits	Contents	
0	Matrix0_ptr	RW	32	Address of matrix 0 data	
4	Matrix0_shape	RW	32	Rows matrix 0	Cols matrix 0
8	Matrix1_ptr	RW	32	Address of matrix 1 data	
12	Matrix1_shape	RW	32	Rows matrix 1	Cols matrix 1
16	Matrix2_ptr	RW	32	Address of matrix 2 data	
20	Matrix2_shape	RW	32	Rows matrix 2	Cols matrix 2
24	Matrix3_ptr	RW	32	Address of matrix 3 data	
28	Matrix3_shape	RW	32	Rows matrix 3	Cols matrix 3
32	-reserved-	-	32		
36	-reserved-	-	32		
40	Command	RW	32	0	enum
44	Status	RW	32	0	enum
8192 x 32 memory					

図 2 – レジスタ概要表

```
int Accelerator(int registers[16], int memory[8192]);
```

図 3 – アクセラレータ関数の API

HLS により配列引数で小型メモリを指定できることがわかりました。つまり、図 3 に示すような関数でこの機能を記述します。

AXI スレーブが合成可能なことを前提として、これはソフトウェアとどのように適合するのでしょうか。私のコーディング環境は通常 Linux です。幸いザイリンクスは PetaLinux を提供し、好都合なことに PetaLinux は User I/O デバイスとして知られるメカニズムを提供しています。UIO は、新しいハードウェアをユーザー メモリ空間に簡単にマッピングすることができると共に、割り込みを待つ機能も提供します。つまり、デバイス ドライバを記述する厄介な時間とプロセスを回避できます。図 4 にシステムを示します。

もちろん、この手法にはいくつか欠点もあります。たとえば、UIO デバイスは DMA と共に使用できないので、デバイス メモリで行列を構築し、手作業でコピーする必要があります。将来的には、必要に応じて、カスタムデバイス ドライバがこの問題に対処できるようになります。

VIVADO HLS でのハードウェアの合成

AXI スレーブの合成に話を戻しましょう。これはどれほど難しいのでしょうか。コーディング制約はかなり妥当であることがわかりました。メモリの動的な割り当てを除き、C++ 言語の大半を使用できました。

結論を言うと、演算時にハードウェアが自動生成されることはありません。このため、動的な割り当てを多用する Standard Template Library (STL) 関数の使用は制限されます。しかし、データがスタティックなままである限り、ほとんどの機能は使用可能です。最初このタスクは厄介に思えましたが、それほどでもないことがわかりました。また、Vivado HLS では、C++ クラス、

テンプレート、関数、演算子のオーバーロードが可能です。私の行列演算はカスタム行列クラスに簡単にラップできました。

AXI スレーブを作成するために I/O を追加することは簡単でした。どのポートが関係し、どのようなプロトコルを使用するかを示すプラグマをいくつか追加するだけです。

また、すべてのノブを押したりしない限り、合成ツールの実行もかなり簡単でした。ここでは詳しく説明しませんが、図 5 に関係する全体的なステップを示します。ターゲットのテクノロジとクロック スピードに関して、Vivado HLS は多少の指示を必要とします。それさえすれば、後はポリシー違反のレポートを監視し、解析レポートから Vivado HLS が期待した通りの動作をしたかを確認するだけです。ツールのユーザーはハードウェアの側面を多少把握する必要がありますが、この問題をカバーするテクノロジ クラスが存在します。また、期待された動作なのかを確認するために、合成前と合成後の両方でシミュレーションを実行するという問題もあります。

Vivado IP Integrator は、AXI スレーブの Zynq SoC ハードウェアへの接続を簡単にし、信号が間違って接続されるという懸念を解消しました。ザイリンクスはさらに私の開発システムである ZedBoard のプロファイルを持ち、IP Integrator がソフトウェア開発キットのデータをエクスポートします。

ボトルネックの解消

私は結果に十分満足し、このチップとツールセットの組み合わせでもっと多くのことをしたいと思っています。まだすべての可能性を検討したわけではありませんが。たとえば、Vivado HLS は AXI マスター インターフェイスもサポートしています。AXI を使用することで、アクセラレータは外部メモリから行列をコピーできます。ただし、このケースではセキュリティの問題が残る可能性があります。それでも、ソフトウェアのコーディング ボトルネックに直面しているのならば、このツールセットを検討してみることを強くお勧めします。Doulos 社が提供するものなど、立ち上げを早くするための豊富なトレーニング クラス、リソース、資料が用意されています。詳細については、www.doulos.com を参照してください。

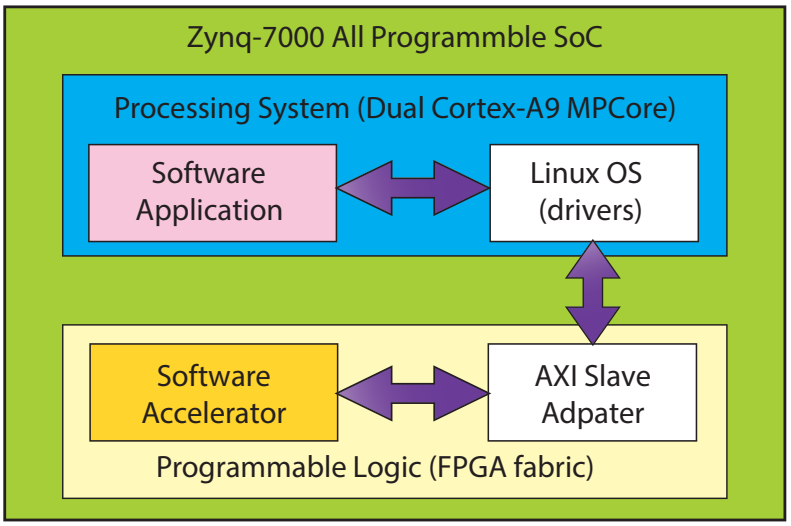


図 4 – システム ダイアグラム

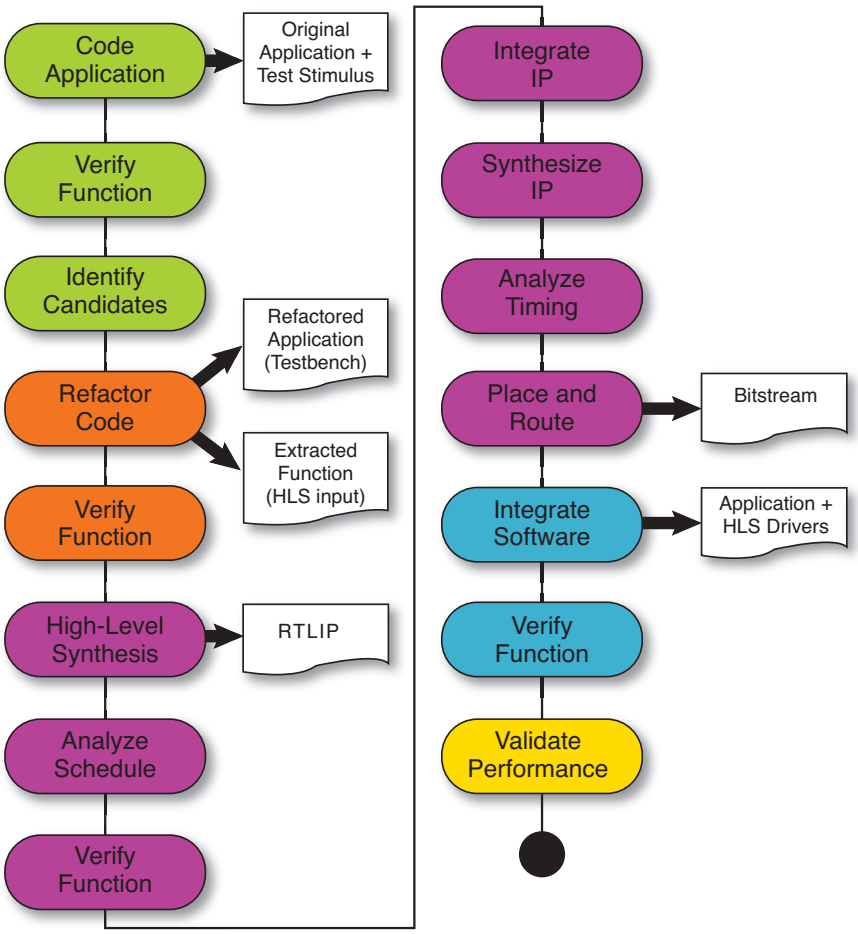


図 5 – デザイン フローのステップ

Xilinx Opens a Tcl Store

ザイリンクスが Tcl ストアをオープン

Greg Daughtry
Director of Product Marketing
Xilinx, Inc.
greg.daughtry@xilinx.com

ツール コマンド言語スクリプトのオープン ソース共有レポジトリ（データベース）が GitHub.com に提供されています。

この 5 年間、ザイリンクスは業界最先端の包括的な開発環境を提供することで、生産性の向上、デザイン サイクルの短縮化、製品の早期市場投入を目指した設計手法やツールの開発を戦略的目標にしてきました。

次世代 Vivado® Design Suite の生産性向上と包括的な UltraFast™ Design Methodology の組み合わせを使用したとしても、今日の All Programmable デバイスのデザインはチャレンジングです。設計者は、数百もの高度にパラメーター化された IP コア、数十万もの配置可能なオブジェクト、数百万ものロジック セルを、ザイリンクス® All Programmable FPGA、3D IC、SoC に統合する必要があります。設計者が複雑なデザインで境界を動かすために試すべき順列は無数にあります。

4 月の Vivado 2014.1 のリリースに伴い、ザイリンクスはツール コマンド言語 (Tcl) コードを共有するためのオープン ソース レポジトリを提供することにより、設計者の生産性をさらに向上させています。Xilinx Tcl Store と呼ばれるこのレポジトリを使用することで、他のエンジニアが開発した Tcl (「ティックル」と発音) スクリプトを非常に簡単に検索および共有できます。Tcl を使用することによって、これらのスクリプトで、Vivado Design Suite の核となる機能を拡張でき、生産性と使用性がが向上します。Tcl Store はユーザー コミュニティに開放され、他のユーザーに大変便利だと思われる Tcl コードを公開することにより、すべての設計者により大きな利益をもたらします。

複雑化するデザイン

Vivado Design Suite はオープンでスケーラブルなデータ モデルを基盤にしています。オープン システムとして、生産性を向上させるために、ツールを使いやすくし、より多くのカスタマイズ オプションおよび解析機能を提供します。これにより、より適した情報を設計者に与え、設計者は最適なインプリメンテーションを提供するツールを使用できます。

2012 年の Vivado Design Suite のリリース以降、大小両方のタスクを実行するための Tcl スクリプトが急増しています。Tcl は Vivado の XDC 制約言語の基盤であるため、設計者は Tcl を理解し使用することがますます重要になっています。

Tcl コマンドを使用することで、タイミング制約をインタラクティブに作成でき、コンパイル時間とデバッグ作業を短縮します。コア コマンドにより、カスタム レポートに使用したり、非常に複雑なツール制御を実行するためのオブジェクト クエリを作成できたりします。また、Vivado Design Suite を使用することによって、固有の DRC およびリント チェックを開発できると共に、高度にカスタマイズされたフローで結果の精度を上げたり、実行時間を短縮したりできます。さらに、Tcl により、設計者はエンジニアリング チェンジ オーダー (ECO) 操作で目的の設計変更を行うことができます。

Tcl による生産性の向上、作成の容易性、可読性により、有益なコードを共有する上で Tcl は重要です。これまでのところ、この共有は主に電子メールやユーザー フォーラムを介して暫定的に行われてきました。社内専用の Tcl ライブラリを確立し、プロジェクト内で使用している会社もあります。

ザイリンクスは新しい Xilinx Tcl Store の提供によって、Tcl の共有を次の段階に発展させつつあります。

TCL STORE へようこそ

Xilinx Tcl Store では、カスタム レポートの作成方法、特定のツール動作の制御方法、カスタムネットリスト変更方法、およびシミュレーション、合成、タイミング、消費電力解析、リント ツールなどのサードパーティ製 EDA ツールの統合方法の例が提供されています。

Tcl Store は Vivado 統合設計環境 (IDE) から本来アクセスでき、直接「アプリ」と呼ばれる Tcl スクリプトの中から選択してインストールできます。インストールが完了すると、これらのアプリはビルトインの Vivado Design Suite コマンドと同じように表示され、すぐにヘルプ インフラストラクチャを開くことができます。Vivado Design Suite は、Tcl のスタンダード パッケージ機能を使用してアプリのバージョンを管理するため、新しいバージョンがリリースされた場合には、ユーザーはマウスをクリックするだけでアップグレードできます。

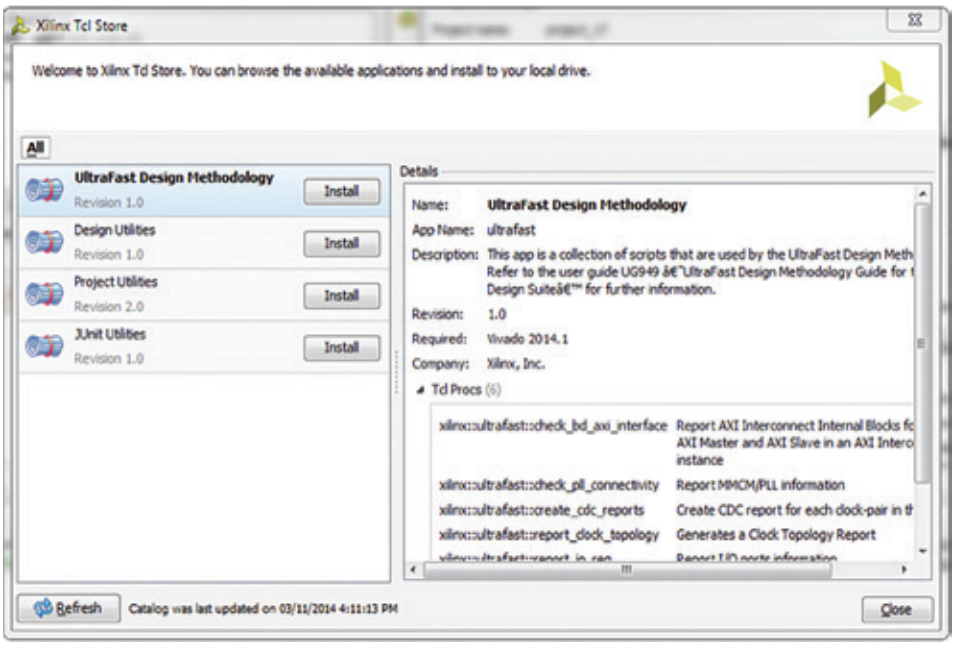


図 1 – Vivado IDE の [Tcl Store] ダイアログ ボックスで、アプリのインストールおよびコマンドの参照ができます。

Xilinx Tcl Store は、ユーザー コミュニティが開発およびサポートする、優れた Tcl スクリプトを簡単に検索して使用できるようにしたものであり、Linux とよく似たシステムです。Tcl スクリプトは、IDE ボタンを選択するより少々先進的ですが、使用方法は簡単です。Tcl API の特定コマンドに関する詳細は、各資料およびユーザー ガイドに記載されており、japan.xilinx.com/support から入手できます。

Xilinx Tcl Store から Tcl アプリをインストールして使用するためのインフラストラクチャを詳しく見てみましょう。

インストールと使用

設計者は、初めて Vivado IDE を起動したときに [Getting Started] ページのアイコンから Xilinx Tcl Store にアクセスできます。または、[Tools] メニューから [Xilinx Tcl Store] メニュー オプションを選択することもできます。これによって、レポジトリ ダイアログ ボックスが開かれ、インストール可能なアプリのリストが示されます (図 1)。

アプリのリストを参照すると、各アプリ内に、実行可能なコマンド (Tcl では「プロシージャ」と呼ばれます) のリストがあります。各アプリの説明およびアプリ内の各プロシージャの説明を見ると、そのアプリやプロシージャの機能を把握できます。[Install] ボタンをクリックすると、アプリがインストールおよび登録され、ネイティブな Vivado Design Suite コマンドと同じように表示されるようになります。一度アプリをインストールすれば、Vivado Design Suite を起動するたびに、自動的に読み込まれます。新しいセッションを開始するたびにアプリをインストールする必要はありません。

プロシージャには、Tcl で名前空間と呼ばれる機能を使用する命名規則があります。コマンドの名前は通常の Tcl コマンドよりも多少複雑で、「::」文字がコマンドに埋め込まれています。たとえば、xilinx::ultrafast::check_pll_connectivity はザイリンクス デバイスのクロック調整ブロックの接続チェックを実行します。命名規則は、Tcl コードが一意であることを保証し、あるアプリ内のプロシージャが別のアプリ内の同じ名前の別のプロシージャと競合しないようにする働きがあります。名前空間は Tcl の標準機能です。

アプリ コマンドを実行するには、名前空間を含めプロシージャの完全修飾名を入力し、オプションで必要な引数を他の Tcl コマンドと同じように受け渡します。また、コマンドは標準の名前空間を使用するので、コマンドをグローバル名前空間にインポートすることもできます。他のコマンド名と競合がなければ、このストラテジは効果を発揮します。この場合、名前空間の修飾子を省略して、プロシージャ名だけを使用できます。上記の例で、UltraFast アプリをグローバル名前空間にインポートした場合、名前空間修飾子を付けず check_pll_connectivity コマンドを直接呼び出すことができます。

アプリの [Details] セクション内にある [Uninstall App] ハイパーリンクをクリックすれば、アプリをアンインストールできます。カタログをアップデートする [Refresh] ボタンもあります。Vivado Design Suite のリリースとは関係なく、アプリのリビジョンに対するアップデートをプッシュするサードパーティのウェブサイト上で Tcl Store カタログが公開されます。カタログが更新された場合、Vivado ツールはアプリのリストについて重要性の少ない同期化を実行します。インストールされているアプリのアップデート バージョンが提供された場合は、[Update] ボタンを使用して取得します。Vivado Design Suite は、最新バージョンのアプリをコピーおよび同期化して、インストールします。コンフィギュレーションの制御問題を回避するために、設計者の要求があった場合に限り、アップグレードがインストールされます。セキュリティが懸念される場合、Vivado Design Suite をネットワーク ファイアウォールの外部と同期させないようにするために、カタログの同期化を無効にするパラメーターがあります。

Xilinx Tcl Store では、Tcl アプリは簡単かつ容易に使用できるようになっています。世界中の開発チームが使用および共有することで生産性を向上させることがザイリンクスの目標です。所定のアプリの最新バージョンだけが表示され、設計者は最新のサポート バージョンだけをインストールまたはアップグレードできます。当然ながら、最善の方法は、有益なコードの豊富なライブラリを確保することです。ザイリンクスは役に立つユーティリティや統合スクリプトをレポジトリに提供しているため、独自の再利用可能な Tcl スクリプトを構築する良い例として活用できます。

TCL STORE への投稿

Tcl Store に投稿し、すべての Vivado Design Suite ユーザーにスクリプトを公開する方法は 2 つあります。1 つ目は、既存の

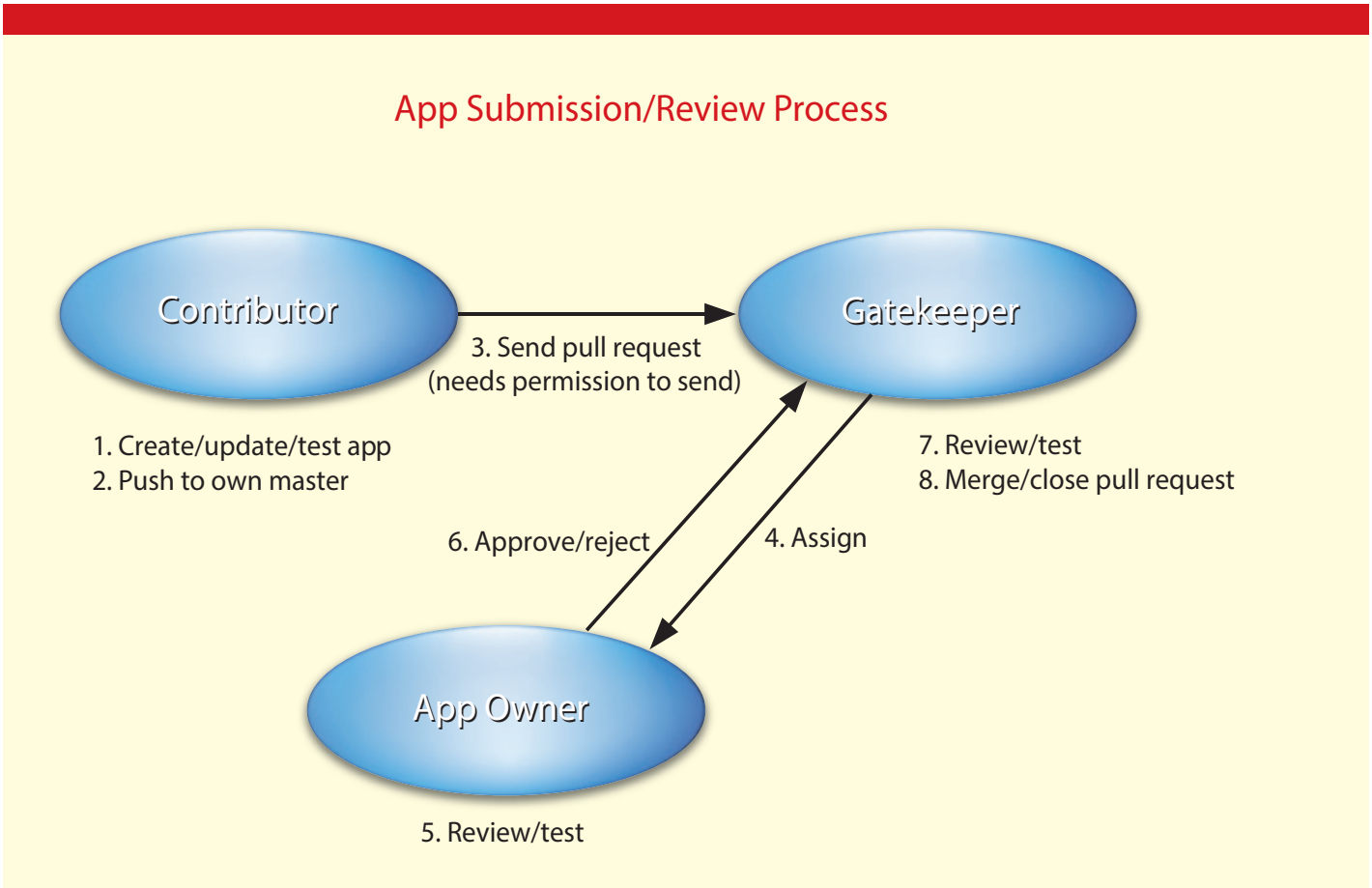


図 2 – Xilinx Tcl Store の投稿プロセスのワークフローは複数の個別ステップから構成されます。

アプリを修正する方法です。2 つ目は、新しいアプリを開発して、その要求を提出する方法です。レポジトリにコードを投稿するには、ソフトウェア開発ツールの改訂管理にある程度の経験があるか、または少なくともその学習意欲が必要です。

各アプリは一名（通常、コードの大半を作成した人で、「アプリ所有者」と呼ばれます）が管理します。ザイリンクス レポジトリ全体はザイリンクスが管理し、アプリ内で基本的な一貫性を保つために、アプリをパブリックドメインに公開するプロセスを保守します。ザイリンクスの従業員が品質を保証するための「ゲートキーパー」の役割を果たします。「投稿者」は、提出のために、ゲートキーパーとアプリ所有者の協力の下で、他のオープンソース プロジェクトのプロセスと一貫性を保ちながら、既存のアプリを変更または新しいアプリを追加します。コードが提供されるサイトの wiki にこのプロセスが記載されます。

すべてのコードの投稿に基本的な要件が適用されます。ザイリンクスは妥当なユーザー エクスペリエンスを確保しながらも、このリスト（変更の可能性があります）をできるだけ少なくするように努力しています。守るべき基本的なアプリ要件のリストは次のとおりです。

- グローバル変数を使用またはアクセスしない引数のプロシージャを使用することで、基本的なコーディング スタイル ガイドラインに従います。
- どのようなプロシージャで、引数はどれで何を返すかを示す基本的な説明をプロシージャ内に含めます。
- コードに基本的な構文チェックを適用すると共に、Vivado Design Suite の一部として提供されるリント ツールを実行します。
- コードが最低限動作し、期待する機能を果たすことを確認するための基本テストを各プロシージャで最低 1 回は実行します。

GITHUB 上の TCL STORE

Xilinx Tcl Store は GitHub.com というサードパーティのウェブサイトで提供されます。改訂管理ツールを使用することで、管理された方法で分散開発が行われることが保証されます。このプロセスのキーとなるのが Git です。これは、一般的なオープン ソースの分散型リビジョン管理ツールで、通常 Linux に使用されます。投稿およびテストのためにレポジトリにアクセスするには、GitHub.com の無償アカウントを取得し、Git をインストールおよびセットアップします。GitHub では、Windows PC 用の Git ツールのインストールを提供します。Linux マシンの場合は、通常装備済みですが、スタンダード パッケージからインストールすることもできます。Git を簡単に使用できるよう、GitHub ではチュートリアルが用意されています。

GitHub アカウントを取得したら、次は Tcl Store レポジトリに投稿するステップです。

1. Xilinx Tcl Store マスター レポジトリを複製します。これにより、サンドボックスのローカル コピーが作成され、他に影響を及ぼすことなく、ローカルに開発およびテストできます。
2. アプリ名と会社名または GitHub ユーザー名に基づき、既定のガイドラインに従い、新しいコードを所定のディレクトリに配置します。標準的な Git の add コマンドを使用します。
3. ローカル レポジトリで Vivado Design Suite を使用し、コードの登録に必要ないくつかのコマンドを呼び出し、catalog.xml ファイルを生成します。このファイルは、必要となる 3 つのファイルの内の 1 つです。残りの 2 つはパッケージ インデックスと Tcl インデックスです。
4. 別の場所で Vivado Design Suite を開き、ローカル レポジトリを指定し、アプリをテストします。すべてが正しく動作し、問題がなくなるまで、リンターおよびローカル テストを実行します。
5. 変更内容をコミットし、それを簡単に説明するメッセージを用意します。
6. tclstore@xilinx.com へ投稿許可を要求する電子メールを送信します。新しいアプリを作成するのか、何を呼び出すのかを明記します。既存のアプリの変更または投稿をする場合は、その旨を明記します。この場合、アプリ所有者の許可が必要になります。
7. ウェブ ブラウザーを使用して GitHub.com にアクセスし、プル リクエストを送ります。これで、レポジトリに投稿をマージするプロセスが正式に開始されます。問題が生じた場合は、適宜ゲートキーパーやアプリ所有者と協力し、GitHub および電子メールで問題を解決します。
8. 以上で完了です。これで、仲間の設計者を助けることができます。

図 2 のワークフローの基本ダイアグラムに、投稿プロセスを示します。

細則

Xilinx Tcl Store はオープン ソースであり、投稿も投稿の利用も無償です。Tcl Store に投稿されたアプリは、オープン ソース プロジェクトで 通常使用される BSD ライセンスを介して無償で再配布可能です。レポジトリへの投稿には、投稿内容が広く容認され、公開できるよう、各アプリの BSD ライセンスのバージョンが含まれます。会社またはユーザーが知的設計資産をパブリック ドメインに公開したくないという場合は、Vivado Design Suite は プル リクエストの送信前テストに使用されるものと同じメカニズムにより、レポジトリのローカル バージョンをサポートします。

さらに、プロジェクトの提供にサードパーティのサービスである GitHub を使用するため、投稿者はアカウントの取得時に GitHub のサービス利用規約に同意する必要があります。

アプリ ストア内のアプリはユーザー コミュニティによって開発およびサポートされます。つまり、ザイリンクスのテクニカル サポートはこの機能についてトレーニングを受けていないため、Tcl コードに関する質問にはお答えできません。これらのアプリのサポートに関する質問はザイリンクスのユーザー フォーラムにお寄せください。コードにバグや問題点を見つけた場合は、GitHub.com プロジェクトに直接提起し、監視できます。これはオープン ソースの開発モデルなので、Linux と同様、他のユーザー全体の利益のために、各ユーザーがこうした問題を解決し、エクスペリエンスを向上させてください。

ロード マップ

Vivado 2014.1 の Tcl Store の導入は始まったばかりです。今年、ザイリンクスは Tcl Store を強化するために、アプリやプロシージャの説明を検索するための機能をインプリメントし、簡単にファンクションを見つけることができるようにする予定です。アプリをインストールすることなく、ソース コードを参照および表示するための方法を提供します。さらに、ユーザーが 5 段階評価、そしてオプションで書き込み評価できるレビュー機能も提供する予定です。これにより、より一般的な投稿のフィードバック 方法が実現されます。

また、シミュレーション、合成、インプリメンテーション、プロジェクト、ネットリスト ユーティリティなど、アプリのカテゴリに基づくフィルタリング機能を実装し、カテゴリ化を向上させます。レポジトリの増大に伴い、グループを増やし、投稿を反映するためのアプリの分類を拡張します。できる限りアプリを簡単に投稿できるようにするために、サポートの負担を最小限に抑え、GitHub を経由することなく、電子メールで Tcl スクリプトを投稿するための方法を模索します。このようなプロセスには管理する者が存在しないという特徴があり、現在のインストール体系とは相いれず、おそらく例として扱うのには適しています。

世界で数千人の設計者が Vivado Design Suite を使用し、数百の会社が UltraFast Design Methodology を採用しています。Xilinx Tcl Store は今後もザイリンクス、そのパートナー、および Tcl スクリプトの共有を目的とする当社顧客との間で新しいオープン ソース プロジェクトを提供することにより、設計者の生産性を向上させていきます。

情報およびリソース

GitHub アカウント アクセスの取得および Git や GitHub のチュートリアルを参照するには、次のサイトにアクセスします。
https://github.com/

Xilinx Tcl Store コード レポジトリおよび投稿方法が記載された wiki は次の場所にあります。
https://github.com/Xilinx/XilinxTclStore

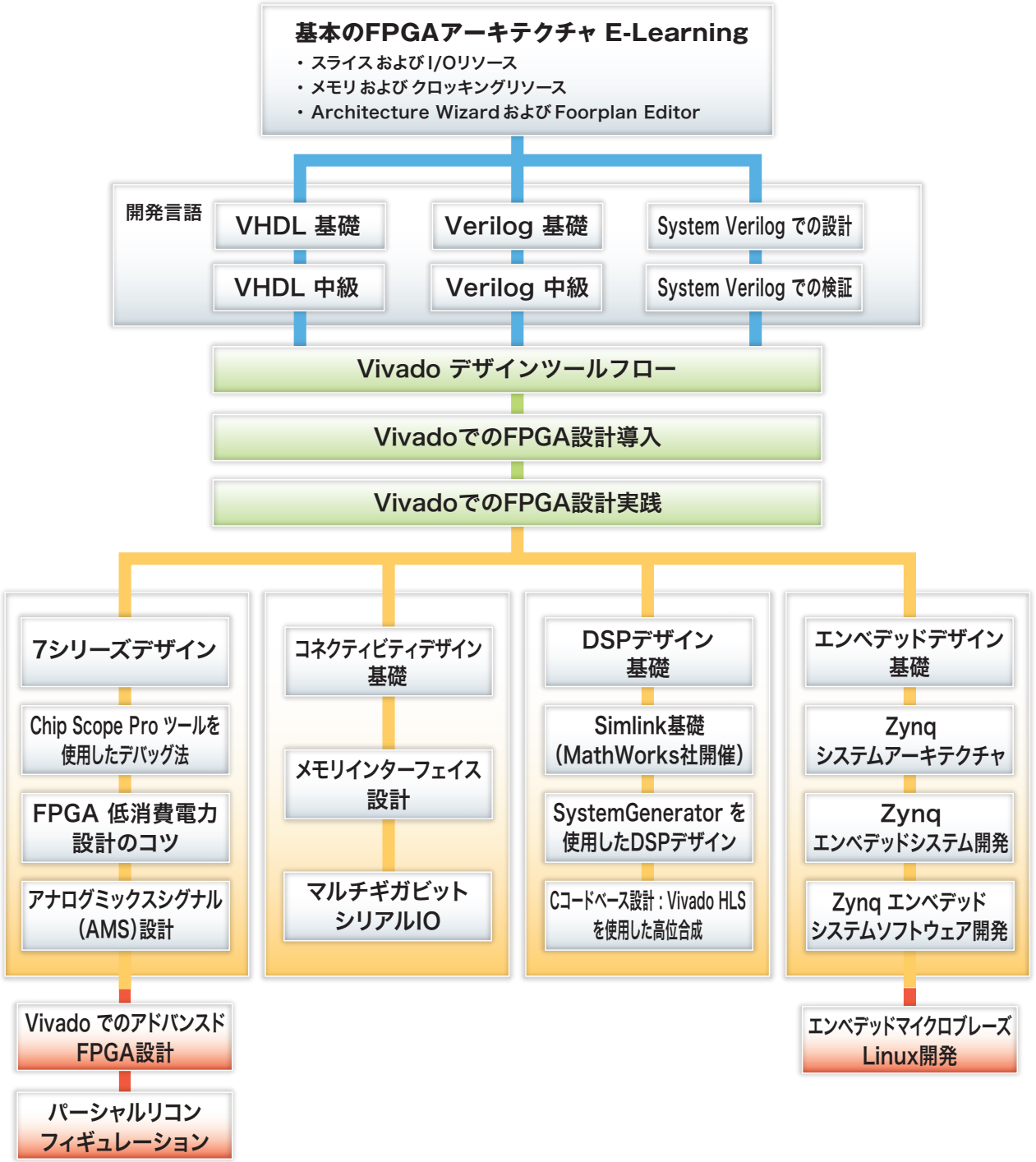
Xilinx Tcl Store wiki には、投稿プロセスに関する詳しい情報が含まれています。
https://github.com/Xilinx/Xilinx-TclStore/wiki/Xilinx-Tcl-Store-Home

UG 894『Using Tcl Scripting Guide』には、Vivado Design Suite のスクリプト機能全般に関する情報が記載されています。
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx2013_4/ug894-vivado-tclscripting.pdf

UG 835『Tcl Command Reference』には、Vivado Design Suite で使用可能なすべてのネイティブ Tcl コマンドに関する情報が記載されています。
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx2013_4/ug835-vivado-tcl-commands.pdf

ザイリンクス トレーニング カリキュラムパス

XILINX



ザイリンクス販売代理店 / 認定トレーニングプロバイダ オリジナル トレーニング

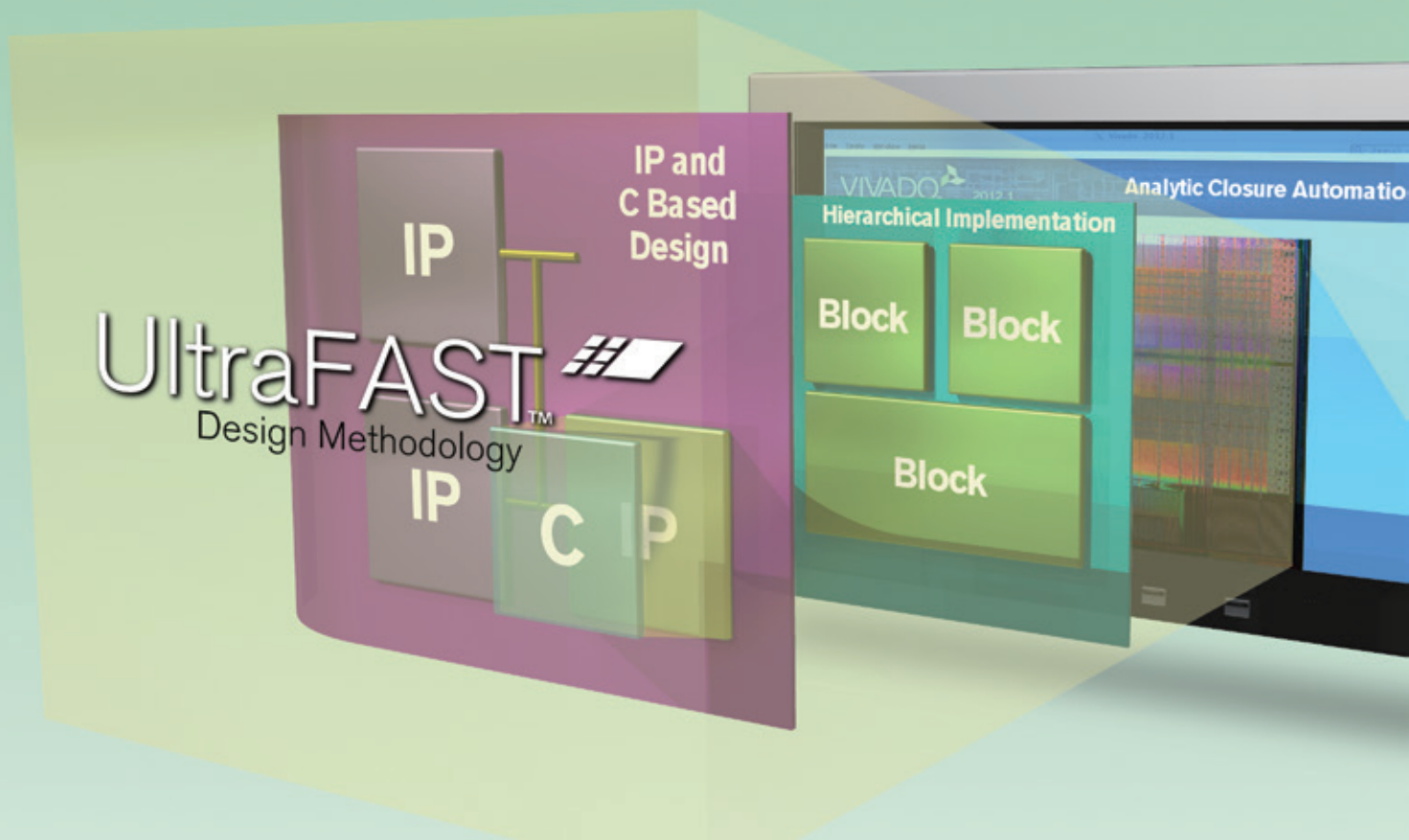
オリジナルトレーニングの内容およびスケジュールは、各社の Web サイトをご覧ください。

AVNET Internix	アヴネット・インターニックス	avnetinternix.co.jp/training.aspx
新光商事 Shinko Shoji Co.,Ltd.	新光商事	xilinx.shinko-sj.co.jp/training/index.html
東京エレクトロン デバイス	東京エレクトロンデバイス	ppg.teldevice.co.jp/
PALTEK	パルテック	www.paltek.co.jp/seminar/index.htm
hdLab	エッチ・ディー・ラボ	www.hdlab.co.jp/web/x500x/

詳細とご登録はこちらから **Japan.xilinx.com/training/**

Xilinx Introduces

Vivado® Design Suite 向け
UltraFast™ 設計手法



ザイリンクス UltraFast 設計手法は迅速で予測可能な設計サイクルを可能にします。

VIVADO

XILINX
ALL PROGRAMMABLE™

ザイリンクス株式会社
製品のお問い合わせは下記の販売代理店へどうぞ

■東京エレクトロン デバイス(株) TEL(045)443-4016 x2web@teldevice.co.jp ■アヴネット・インターニックス(株) TEL(03)5792-8210 EVAL-KITS-JP@avnet.com
■(株)PALTEK TEL(045)477-2005 nfo_pal@paltek.co.jp ■新光商事(株) TEL(03)6361-8086 X-Pro@shinko-sj.co.jp

詳細はこちら： japan.xilinx.com/ultrafast