93 号 2015

Xcell journal

SOLUTIONS FOR A PROGRAMMABLE WORLD

ザイリンクス、最新医療機器の Time-to-Market を短縮化

FPGA ベースの
ファジー コントローラーで、
サトウキビの搾汁プロセスを管理

Zynq MPSoC と
Xen ハイパーバイザーのサポート

XDC タイミング制約の活用

非対称型 マルチプロセッシング
システムのソフトウェア開発を
容易化

ザイリンクス FPGA による
先進の自律的群衆モニタリング
システム

ページ 12

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 神保 直弘
naohiro.jinbo@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 93 号

2015 年 12 月 21 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

Zynq UltraScale+ の「Hello World」

予定より 3 ヵ月も前倒しで、最初の Zynq MPSoC XCZU9EG を出荷開始した Zynq® UltraScale+™ MPSoC 設計チームに心からの賛辞を送ります。これで、ザイリンクスは三連覇を達成しました。28nm では 7 シリーズ、20nm では UltraScale™ ファミリ、そして今回、16/14nm FinFET では UltraScale+ と、プログラマブル ロジック メーカーとして各先進プロセス ノードで初となるデバイスを出荷し、3 回連続で市場競争に勝利したことになります。

2015 年 9 月 30 日、ザイリンクスは、2 社の顧客にこのデバイスのサンプル出荷を開始したことを発表しました。テープアウトから出荷までは 2 ヵ月半でした。この早さは、ザイリンクスの Zynq MPSoC デザイン チームおよび品質チーム、そしてファウンドリ TSMC社 の皆さんの技術力を強固に裏付けるものです。

ザイリンクスは、9 月後半に特性評価およびテスト用の最初のシリコンを入手後、わずか数時間でアップストリーム Linux カーネルを新たなデバイス上でブートすることができました (デモ動画をご覧ください)。

ザイリンクスの UltraScale+ ポートフォリオの出荷第 1 弾である Zynq UltraScale+ MPSoC デバイス上の「アップストリーム」Linux カーネルのブートのデモ動画

本誌では、Xcell Journal 90 号のカバー ストーリーで、この画期的な新製品のアーキテクチャ的特長と単位ワット当たりの性能について特集しました。TSMC社 の 16nm FinFET+ プロセス テクノロジに実装された Zynq MPSoC は、クワッドコア 64 ビット ARM® Cortex™-A53 アプリケーション プロセッサ、32 ビット ARM Cortex-R5 リアルタイム プロセッサ、ARM Mali-400MP グラフィックス プロセッサ、16nm FPGA ロジック (UltraRAM を含む)、ペリフェラルのホスト、セキュリティおよび信頼性に関する各種機能、革新的な消費電力制御テクノロジをシングル デバイス上に搭載しています。この Zynq UltraScale+ MPSoC の採用は、28nm Zynq SoC で設計した場合に比べ、5 倍の単位ワット当たり性能を持つシステムを設計できます。

Xcell Publications の発行人として、ザイリンクスの 28nm Zynq-7000 All Programmable SoC で実現された驚くべきイノベーションについて、読者の皆様からの寄稿を楽しんで拝読してきました。これからも、新たな Zynq MPSoC と、その多数のマルチプロセッシング機能によってどのようなことが可能になるか、皆様からのご寄稿を心待ちにしております。本製品は、今後数四半期の間に提供を予定しています。本誌およびソフトウェア開発者向けに最近創刊された「Xcell Software Journal」にぜひご寄稿頂き、Zynq MPSoC 設計経験を共有しませんか。皆様からの素晴らしいデザインを楽しみにしています。

Mike Santarini
発行人

This year's best release.

Xcell Publications

Solutions for a Progammable World

ザイリンクス SDx IDE およびデバイスで C/C++、OpenCL コーディングの高速化を追求するソフトウェア開発者向けの決定版

受賞歴のあるザイリンクスのパブリケーション グループは、プログラマブル FPGA のソフトウェア業界に特化した新しい業界誌を四半期ごとに発行します。ザイリンクス SDx™ 開発環境や高位の入力方法を用いて、ザイリンクスの All Programmable デバイスのプログラミングを行うユーザーが対象になります。

広告掲載について

Xcell Software Journal では、この美しくデザインされた新しい誌面への広告掲載の予約を受け付けています。最も関心の高い人々に、製品やサービスについてお知らせする絶好の機会です。

広告に関するお問い合わせは、有限会社 エイ・シー・シー(sohyama@acc-j.com)へお気軽にお問い合わせください。

XILINX
ALL PROGRAMMABLE™

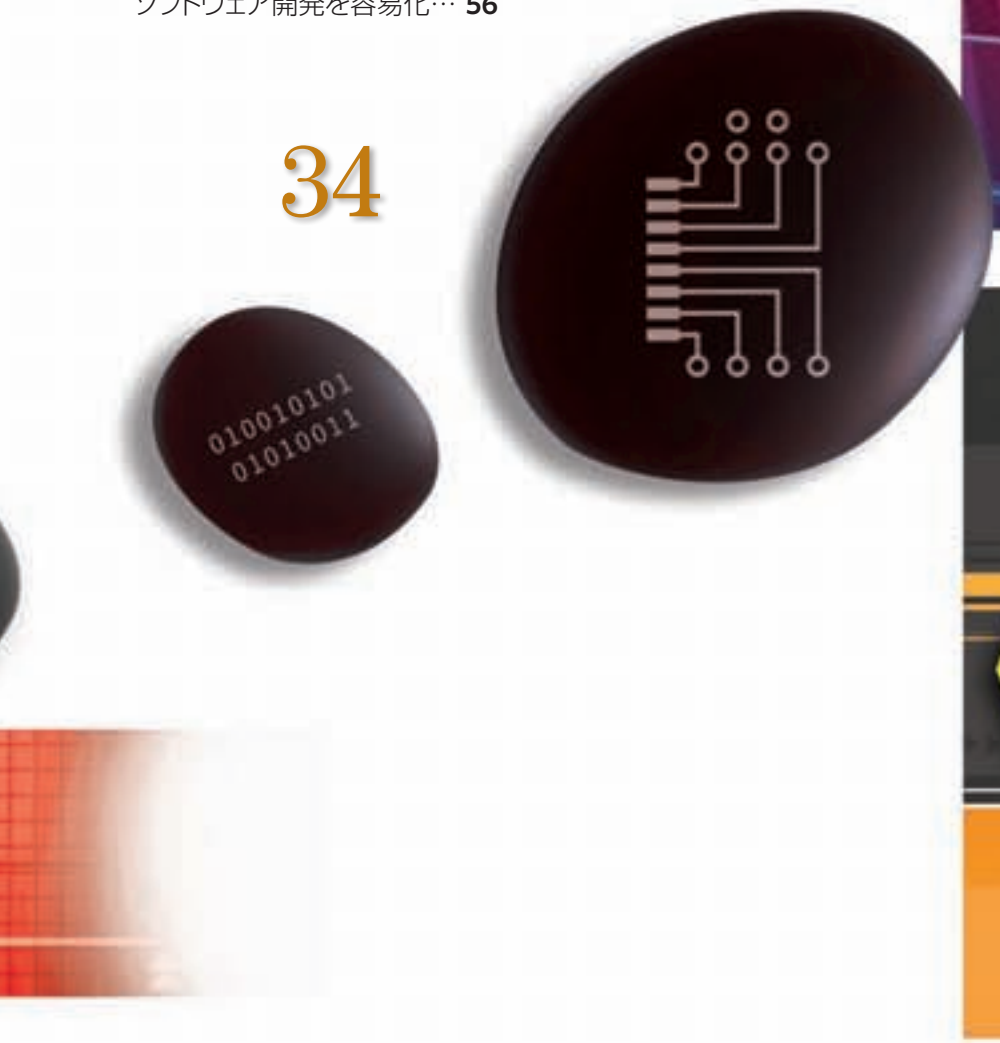
VIEWPOINTS

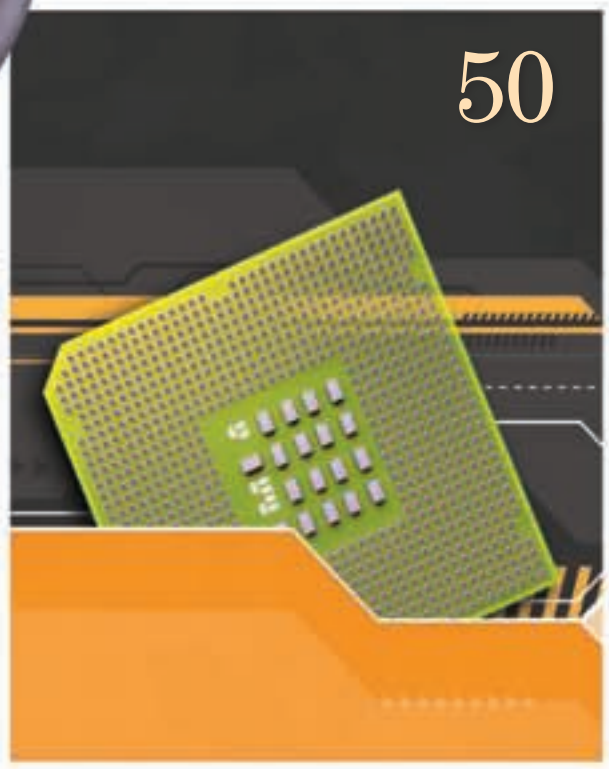
XCELLENCE BY DESIGN APPLICATION FEATURES

Cover Story

ザイリンクス、
最新医療機器の
Time-to-Market を短縮化

THE XILINX XPERIENCE FEATURES





Xilinx Speeds Customer Medical Innovations to Market

ザイリンクス、最新医療機器のTime-to-Market を短縮化

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

電子機器は、私たちの生活にさまざまな影響を与えていますが、文字通り、生か死か、に影響をおよぼす分野が医療機器産業です。医療技術の継続的な発達により長寿命化が進んでいますが、そこに大きく貢献しているのが医療用電子機器の急速な進化です。ザイリンクスの All Programmable デバイスは、これまでの 30 年間、新たな医療用電子機器を開発するために使用されてきました。

現在、ザイリンクスの FPGA (最近では Zynq®-7000 All Programmable SoC などのシステム オン チップも含む) は、増え続ける医療システムの中核に使用されています。最終製品としては、手術用ロボット (英文)、生体情報モニター、人工呼吸器、医用画像処理 (CT、MRI、超音波診断)、X 線装置から、除細動器、内視鏡検査機器、輸液ポンプ、アナライザに至るまで、多岐にわたります。FPGA は、研究者がゲノム配列を高速に解読したり、科学者や製薬会社が病気症状を治療する医薬品をよりスピーディーに開発、改良したりするために必要な高性能コンピューティング システムの中心となっています。

FPGA が医療機器市場でこのように重要な位置を占めるようになってきた理由は容易に理解できます。なぜなら医療機器メーカーは、FPGA (最近では Zynq SoC) の使用により、開発した機器の故障リスクを低減して、法規制の承認手続きにかかる時間を短縮できます。All Programmable デバイスにより、高い信頼が要求されるシステム機能を抽出してザイリンクス デバイスのロジックでハードウェア実装を行い、それほど重要度の高くない機能はソフトウェアで実行できます (図 1)。

ザイリンクスの医療機器グループ、シニア プロダクト マーケティング エンジニアの Kamran Khan は、次のように述べています。「医療テクノロジは、急速に進化しています。スマート化、接続性の向上、統合化、コンパクト化が進み、患者の病状回復を早くするためにより低侵襲 (検査・治療において可能な限り患者の身体への負担を減らす) になってきています。さらに重要なことに、高機能化しています。」この半世紀の医療技術の進化の結果でもある、世界の人口増加と長寿命化が進むにつれ、高機能な医療機器の需要はさらに増大しています。

より長く、より健康に生きる

医療業界、ひいては医療機器市場の最大の促進要因は、今後見込まれる世界人口の増加にあります。現在、世界の人口は、約 73 億人で、2050 年までに、97 億人に増加すると予測されています。一方で、現在、65 歳以上の人口は世界全体の人口の約 23% ですが、2050 年までには 32% (31 億人) に増加すると見込まれています。高齢者は、最も定期的な医療サービスが必要とされる可能性が高い人口区分となっています。

「生活の質が向上し、薬品への理解が進んだことで、寿命が伸びてきました。寿命が伸びると、高齢者に向けたより優れた医療に加え、副作用の少ない低侵襲のテクニックで提供することが要求されます。」と、Khan は指摘します。

人口の増加と高齢化は、世界の医療コミュニティ、そして、国による医療政策の重要度が増している各国政府にとって、大きな課題です。しかし、この人口増加は、医療用電子機器やその他の医療分野のイノベーションにとっては、大きなチャンスでもあります。Khan によれば、医療市場は 2019 年までに 2,120 億ドルに拡大し、そのうち半導体への支出は 60 億ドルに達すると予測されています。

Khan は、また、次のように述べています。「医療用電子機器業界は、かっては、非常に閉ざされたもので、それにより、Siemens 社や General Electric 社などの企業が、自社のシステムの新バージョンを自社ペースで開発することができました。現在、X 線および超音波診断はコモディティ化しています。今では、多くの人々が、法規制の負担を考慮しても、成熟した低リスクのアプリケーションに参入することはそれほど難しくないことに気づいています。実際、医療機器のベンチャー企業が盛り上がりを見せています。これは世界的なブームで、中国や南米から多数の新しい企業が市場に参入しています。中国やブラジルなどの一部の国々では、自国の製品を優遇する措置が取られているため、この新市場に新しい企業が台頭してきています。」

Khan は、この新しい競争により、当然、市場のあらゆるプレイヤーにとって、Time-to-Marketや価格へのプレッシャーが高まっていると指摘しています。同時に、医療業界により大きな価値とイノベーションが、より迅速にもたらされることは、あらゆるプレイヤーにとって好ましいことです。

Khan は、さらに、次のように述べています。「機能の一体化と、移植性の高さが市場に求められています。医療施設では、従来、特定の作業に特化したマシンを作業ごとに 1 台配備していました。どのマシンも大型化し、多くの異なるマシンが必要な場合は病室のかなりの面積が機器よって占有されてきました。」

「システムは、必ずしもシステム間で通信ができるわけでなく、互換性があるわけでもなかったため、さらに複雑になりました。そのため、現在では、複数のタスクを 1 台の装置で行うことが求められております。同様に、省スペースで部屋間の移動もしやすくなる小型化へのニーズも高まりつつあります。さらに、機器がバッテリ駆動型であれば、電気が供給されていない場所や救急車でも使用できます。同時に、たとえば、患者のバイタルサインの変化に合わせて投薬量を調整するなど、各システムと装置の他の部分とのリアルタイム通信への需要も高まっています。」

「たとえば、生体情報モニターは、従来、非常に単純でしたが、現在では異なります。以前は、数チャネルのアナログ バイオテレメトリーを入力とし、極めてシンプルな何らかの処理を行って、モニターに情報を表示するだけでした。とてもシンプルなため、その多くは TI 社や Freescale 社などのエンベデッド プロセッサで実行されていました。しかし、現在では、医療機器はより統合され、スマート性が必要になってきています。また、システム間の通信や協調動作も求められています。」

たとえば、生体情報モニターは人工呼吸器や輸液ポンプとの通信が必要である、と

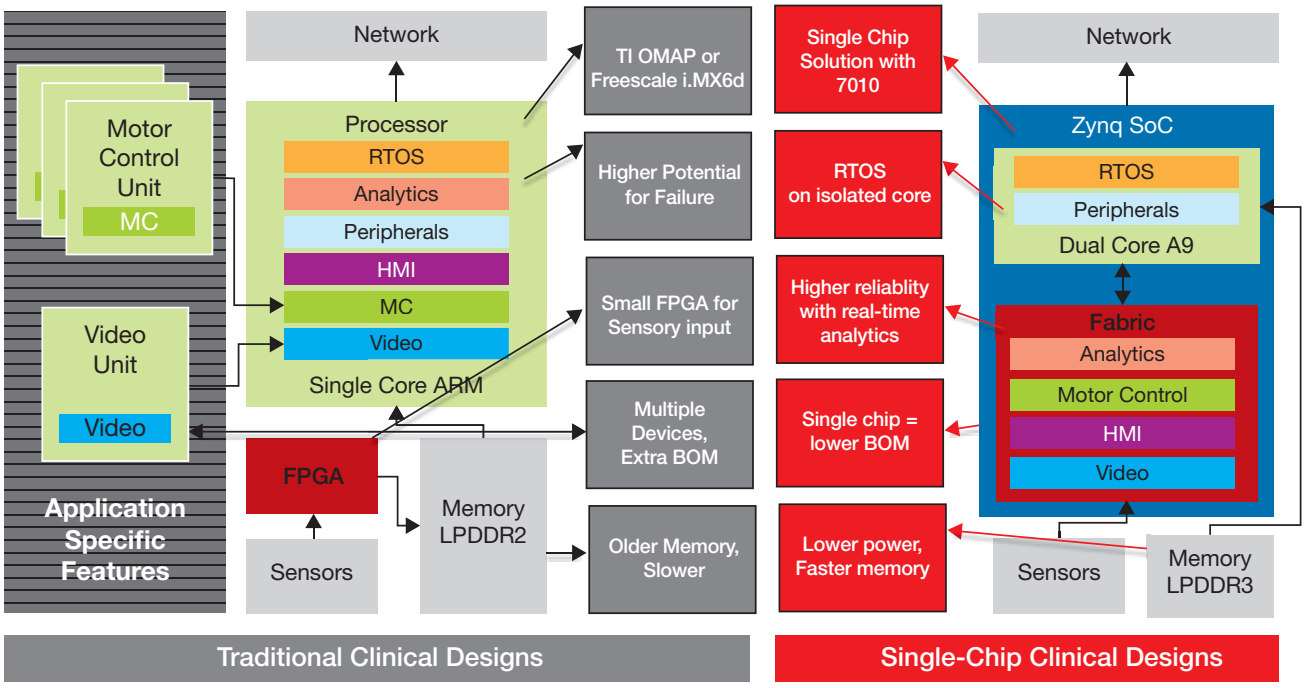


図 1 – 医療機器メーカーは、Zynq SoC プラットフォームにより、革新的で最適化されたシステムを迅速に開発し、市場に投入できます。

Khan は言います。患者のバイタルサインの変化に合わせて、どちらのシステムも適切に対処しなければなりません。つまり、呼吸ポンプは酸素混合気を調整し、輸液ポンプは投与量を調整する必要があります。「また、病院のメイン ネットワークとも通信して、緊急事態の発生をスタッフに知らせる必要があります。」と Khan は述べています。さらに、その情報は患者の長期的なカルテの一部として保管する必要があります。

Khan によると、モノのインターネット、クラウド コンピューティング、そして、現代のネットワーク インフラストラクチャも医療業界に受け入れられつつあるといえます。これらの技術を融合させることで、外来患者の健康状態をモニターし、健康状態のパターンを遠隔装置に記録し、緊急事態にリアルタイムで対応することが可能になります。ちょうど、ホーム セキュリティ会社が各特性をモニターするのと似たような方法で、患者の健康状態をリアルタイムでモニターする、という急成長中の分野です。

ますます増加する法的課題

業界の成長と、スマートに接続され、統合された医療システムへの動きが活発化する中で、ベンダーは、ますます厳格さを増し、規制当局の数も世界中で増え続けている安全性および信頼性規制基準に自社の装置を準拠させる必要に迫られています。

変化し続ける法規制環境は、現在の医療機器メーカーが直面する唯一かつ最大の課題と言われることがよくあります。企業は、比較的厳しい法規制ガイドラインおよびテストを通過するまでは、法的に製品を販売することができません。医療機器の故障は、医療責任を問う訴訟にまで発展します。

多くの場合、規制の度合いは、市場に投入しようとする装置の種類によって変わり、評価に必要な期間も変わります。一般的に、患者の身体に接触しない非クリティカルな装置の場合は市場化までに約半年の法規制サイクル、患者の身体に接触してよりクリティカルな機能を行う装置の場合は約 2 年間の承認サイクルが必要になります。

Khan は、次のように述べています。「生体情報モニタリングや超音波診断などの確立した市場においても、一般的には、必要な文書とテストの数が膨大であるために、1 年から 1 年半もの法規制の承認期間が必要です。メーカーは製品をサポートするテクニカル ファイルを作成し、その文書を規制機関に提出しなければなりません。長い評価サイクルを経た後に、製品の合格 / 不合格が決まります。どのメーカーも、初回で合格することを目指します。なぜなら、もし合格しなかった場合、2 回目の評価はさらに細かいものとなり、サイクルがさらに長くなる可能性があるためです。これは税務調査と似ています。」

Khan は、真の目標は、その医療システムがもたらすリスクを理解し、定量化することだと言います。「医療機器は故障してはならない、という印象がありますが、規制当局はすべての機器は故障するということを理解しています。規制当局が OEM メーカーに求めているのは、故障リスクを理解して、リスクを下げる努力をするとともに、機器が故障し得るすべてのケースと、その場合に何が起こるかを把握することです」と Khan は述べます。

世界各国のさまざまな規制当局がハードウェアとソフトウェアの両方を精査しますが、ソフトウェアの方を特に厳しく調査する傾向にある、と Khan は指摘します。それは、ソフトウェアのエラーの方が、システムを未知の状態にする可能性が高いとみなされているためです。Khan は、続けてこのように述べています。「PC や携帯端末上でソフトウェア バグを経験した人は多いですが、実ハードウェアの障害を経験した人は多くありません。そのため、医療機器デバイスにおいて、一般的に、ソフトウェアの方が厳しい調査を受けます。」

「市場に投入される多くの医療機器デバイスには、エンベデッド プロセッサが必要です。演算処理能力はそれほど必要とされませんが、エンベデッド処理のためにソフトウェアが存在します。問題は、そのデバイスが安全で、患者を傷つけないということを、どのようにして規制当局および顧客に実証するかということです。たとえば、毎日薬剤を供給してくれる輸液ポンプが、時間どおりに正しい投与量を供給してくれて、夜中に止まったりしないと、どうして分かるでしょうか」

デバイスの複雑性が増し、複数のタスクを実行するようになるにつれ、コードの複雑性も高まり大規模化していきます。Khan は、次のように述べています。「『信頼性の高い』ソフトウェアは、依然として、あまり理解されていません。ソフトウェアはとても複雑で、デバッグによりそのリスクを把握するのは困難です。このように、故障する可能性が高いことは、生命に影響のない日常的な医療システムにおいても、高いリスクにつながります。」

FPGA、そして最近では Zynq SoC が医療機器メーカーに普及してきた理由はここにあります。All Programmable デバイスを使用すると、メーカーは故障リスクを低減し、法規制の承認にかかるプロセスを短縮できるのです。基本的に、高い信頼性が求められるシステム機能を抽出してザイリンクス デバイスのロジックで実装し、その他の重要度の高くない機能はソフトウェアで実行できます。

医療イノベーションを促進するザイリンクス デバイス

ザイリンクスは、数十年にわたる医療用電子機器セクターにおける取り組みの末、ザイリンクスの All Programmable FPGA および SoC で構成された総合的な医療向けツールボックスを開発した、と Khan は述べます。このツールボックスには、高い品質、信頼性、冗長性を提供する認証済みの設計ツールやメソドロジや、ザイリンクスおよびザイリンクス アライアンス プログラムのメンバーによる、信頼性の高さが実証されたシリコン IP およびソフトウェア スタックも含まれています。ザイリンクスの新しい開発環境 SDSoC™ の利用により、クリティカルな機能は Zynq SoC に実装し、余りクリティカルでない機能は Zynq SoC の ARM® プロセッシング システムで実行します。そうすることにより、最適化された医療システムをさらに素早く構築できるようになります (図 2)。

ザイリンクスが医療市場に貢献してきた 30 年間の歴史のうち、この 10 年で、医療機器に採用の ASIC や ASSP は、急速に FPGA

に置き換わってきました。医療機器は、世界中で販売される台数としては比較的少ないため、コストと厳格で時間のかかるテストおよび法規制プロセスによって、ASIC や ASSP は使用できなくなったのです。その結果、現在では、大多数の医療機器に、ザイリンクスのデバイスが採用されています。

Khan が述べるには、1980 年代後半から 1990 年代前半にかけて、ザイリンクスのスマートな FPGA が医療機器のセンサー インターフェイスとして使用され始めました。しかし、次第に、FPGA が ASIC や ASSP に取って代わるようになるとともに、よりクリティカルな機能がデバイスに盛り込まれるようになってきました。最新の機器 (特に、Zynq SoC、並びに、最近出荷開始し、Zynq SoC よりも安全性やセキュリティ機能が向上した、Zynq UltraScale+™ MPSoC) では、ザイリンクス デバイスが医療システムの中核で重要な役割を果たしています。

Khan は、次のように述べています。「ザイリンクスの Zynq SoC ポートフォリオは、リスクを低減し、医療分野の革新的製品の市場投入期間を短縮できます。デザインのソフトウェア部分を新しいツールの SDSoC で設計し、ソフトウェアではなくプログラマブル ロジック ファブリックに実装した後、冗長性のレイヤーに追加し、システムの信頼性を高めることができます。」

Khan が述べるには、たとえば、輸液ポンプを設計する場合、システムの一部は、正確な量の薬剤を指定された時間どおりに供給し、メトリクスが医師の設定したとおりに保たれるようにモーターを制御します。また、輸液ポンプの別の一部分は、患者をモニターして健康状態に問題がないことを確認するバイオテレメトリーを構成します。

Khan は、次のように述べます。「ザイリンクスのアイソレーション デザイン フローにより、システムをクリティカルな機能と非クリティカルな機能に分離し、クリティカルな機能をロジックに実装することで、システム内のクリティカルな機能間に物理的バリアを設けることができます。故障条件が発生した場合に追加の安全対策を設計して、安全かつ予測可能な方法でシャットダウンします。さらに、規制当局には、デザインの信頼性が高いザイリンクス ファブリック上で構築されていることを報告できます。ザイリンクスの IDT ツールが発行するレポートを使用して、信号パス、予測可能な結果、およびフェールセーフについて報告することもできます。」

ザイリンクスの新しい開発環境SDx (FPGA 開発用 C、C++、OpenCL™ によるデザイン エントリを行う SDAccel™や、Zynq SoC 開発用 C/C++ によるデザイン エントリを行う SDSoC) により、医療機器メーカーの開発者は、システムのプロトタイプを C 言語で開発ができ、どの機能 (クリティカルと非クリティカル両方) がハードウェアもしくはソフト

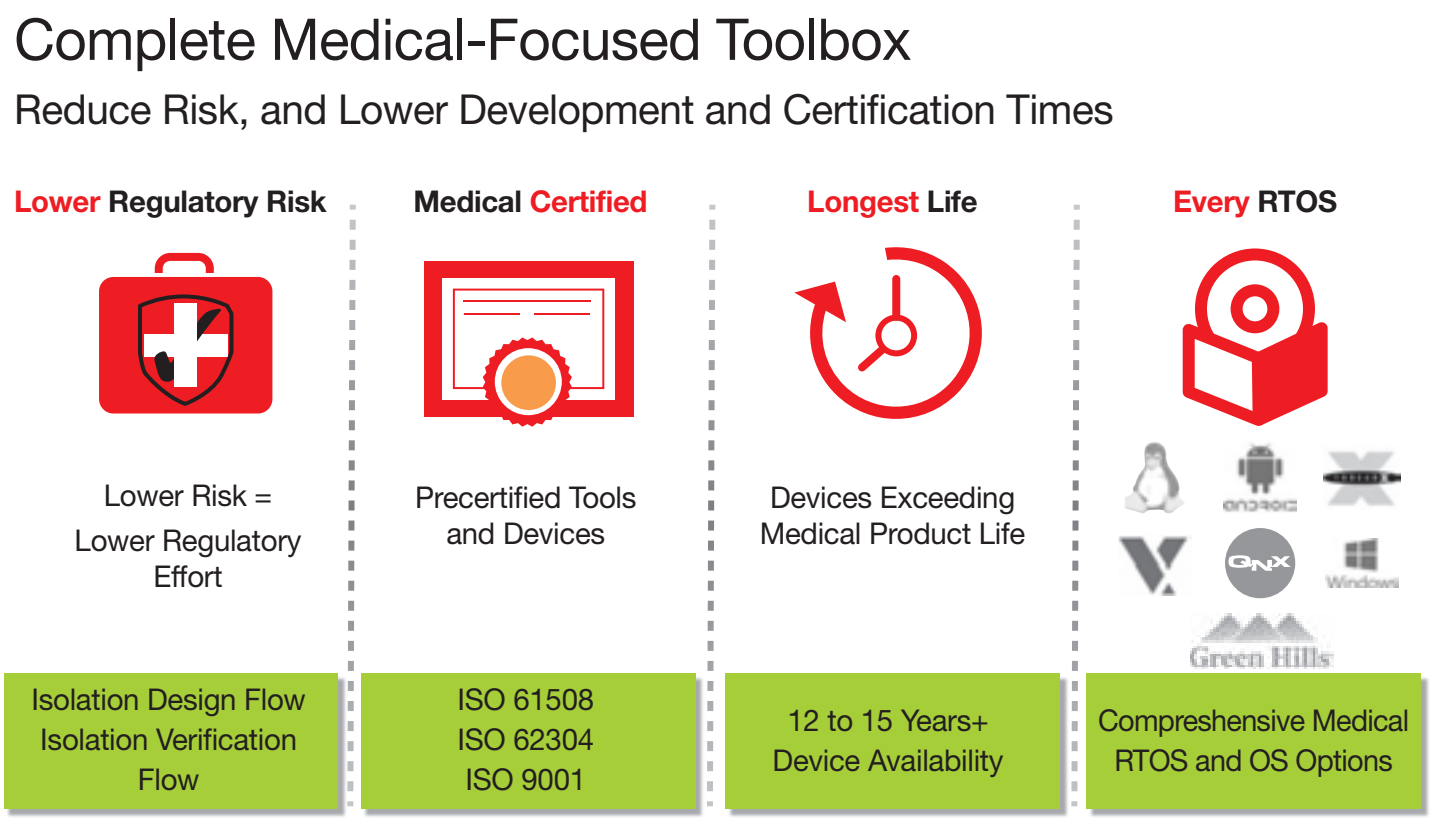


図 2 – ザイリンクスには、リスクを軽減し、法規制プロセスにかかる時間を短縮する、実証済みのデザイン ツールがあります。

図 3 – TOPIC Embedded Products 社の Dyplo IP を使用により、
医療用製品の設計と開発がスピードアップします。

ウェアでの実行に適しているかを判断し、アイソレーション デザイン フローを使用してハードウェア機能を詳細に実装し、冗長性レイヤーを追加して信頼性を高めることができます。「システム レベル メソドロジを使用することで、バックエンドのデザイン サイクルを数ヵ月短縮できることもあります」と Khan は指摘しています。

Khan によると、ザイリンクスの軍事および航空宇宙産業における豊富な経験から、ザイリンクスの医療アプリケーション向け民生用デバイスは、すでに放射線耐性について必要な要件を満たしているとのことです。また、ザイリンクスは、自社のデバイス、Vivado® Design Suite、および IPコア (自社 IPコアおよび医療機器市場のアライアンス メンバーのコア) が厳格な品質や安全性基準に準拠するよう、熱心に取り組んでいます。一例として、ISO 60601 第 3 版、ISO 13485 医療機器設計基準のほか、ICE 61508 (機能安全) および ICE 62304 (RTOS 統合) などの国際規格があります。その結果として、顧客による最終製品の設計が法規制の手続きを迅速に通過できるようになります。

Khan は、次のように述べています。「顧客は個々のデバイスではなく、最終製品についての認証を得る必要がありますが、私たちの役目は、個々のデバイス、ツール、IPコアを同規格に準拠させることです。たとえば、ICE 62304 は RTOS 統合のための規格です。アライアンス メンバーである QNX 社の RTOS は、すでに ICE 62304 の認証済みであるため、Zynq SoC で使用することで、6 ヵ月の認証プロセスが短縮されます。同様に、アライアンス メンバーの TOPIC Embedded Products 社は、プロトタイプや設計をさらにスピードアップする素晴らしい IPコア を提供しています。その IPコア、デザイン フロー、SOM ボードは、ISO 13485 品質管理基準に準拠することが認証されています。これにより顧客は、法規制プロセスにかかる時間をさらに短縮できます (図 3 のビデオを参照)」。

プラットフォーム戦略

法規制の負担や、Time-to-Marketの重圧が増大する中で、現代の医療機器メーカーの多くは、Zynq SoC を中心としたプラットフォーム デザイン ビジネス戦略を取っています。

Khan は、次のように述べています。「各製品を一から設計することができない場合、製品ラインにスケーラブルなアプローチを取る必要がある、という認識が市場に急速に広がっています。たとえば、製品ラインに携帯可能バージョン、下位バージョン、中位バージョン、ハイエンド バージョンがある医用画像処理の分野などでは、プラットフォーム ベースの設計とコスト サイジングは大きいものになります。Zynq SoC をベースにハイエンド用の 1 つのプラットフォームを設計すれば、同じハードウェアを各レベルで使用して、各最終製品の市場ニーズに合わせて機能を制限することで、スケーリングできます。」

Khan によれば、Zynq SoC を中心とするプラットフォーム アプローチは、複数のディスクリート部品で構成するプラットフォームよりも多くのメリットがあります。Khan は、次のように続けます。「医療機器は、通常、他の消費者向け機器(通常、2 ～ 3 年) よりも長く、製造してから10 ～ 15 年程度販売されます。医療用デバイスは、一般に、設計に 3 年間、法規制の承認にさらに 1 ～ 3 年かかり、そこから 10 年以上市場に出回るのです。しかし、現在のほとんどのエンベデッド プロセッサは約 5 年で寿命が来て、新しいバージョンのデバイスに置き換わります。これは、ほとんどのデザインのターゲットが、主に消費者市場であるためです。しかし、医療機器業界では、旧バージョンが入手できなくなったために新しいバージョンのチップに置き換える必要が生じた場合、製品はもう一度法規制の手続きを経なければなりません。」

Khan によれば、Zynq SoC と MPSoC ファミリを使用することで、顧客のデザインは、エンベデッド プロセッサまたは複数プロセッサの性能メリットに加え、フレキシビリティ、製品の差別化、安全なプログラマビリティを享受できます。さらに、幅広いプロトコル、センサー、ビデオ構成に対応する I/O フレキシビリティがあります。「複数のシステム機能を Zynq SoC ファミリと MPSoC ファミリで統合することで、マルチチップ プラットフォームと比べて面積、BOM (部材) コスト、消費電力が劇的に削減でき、新しい医療機器のTime-to-Marketが短縮されます。」と Khan は述べています。

ザイリンクスの医療機器アプリケーションの詳細については、http://japan.xilinx.com/applications/medical.html をご覧ください。

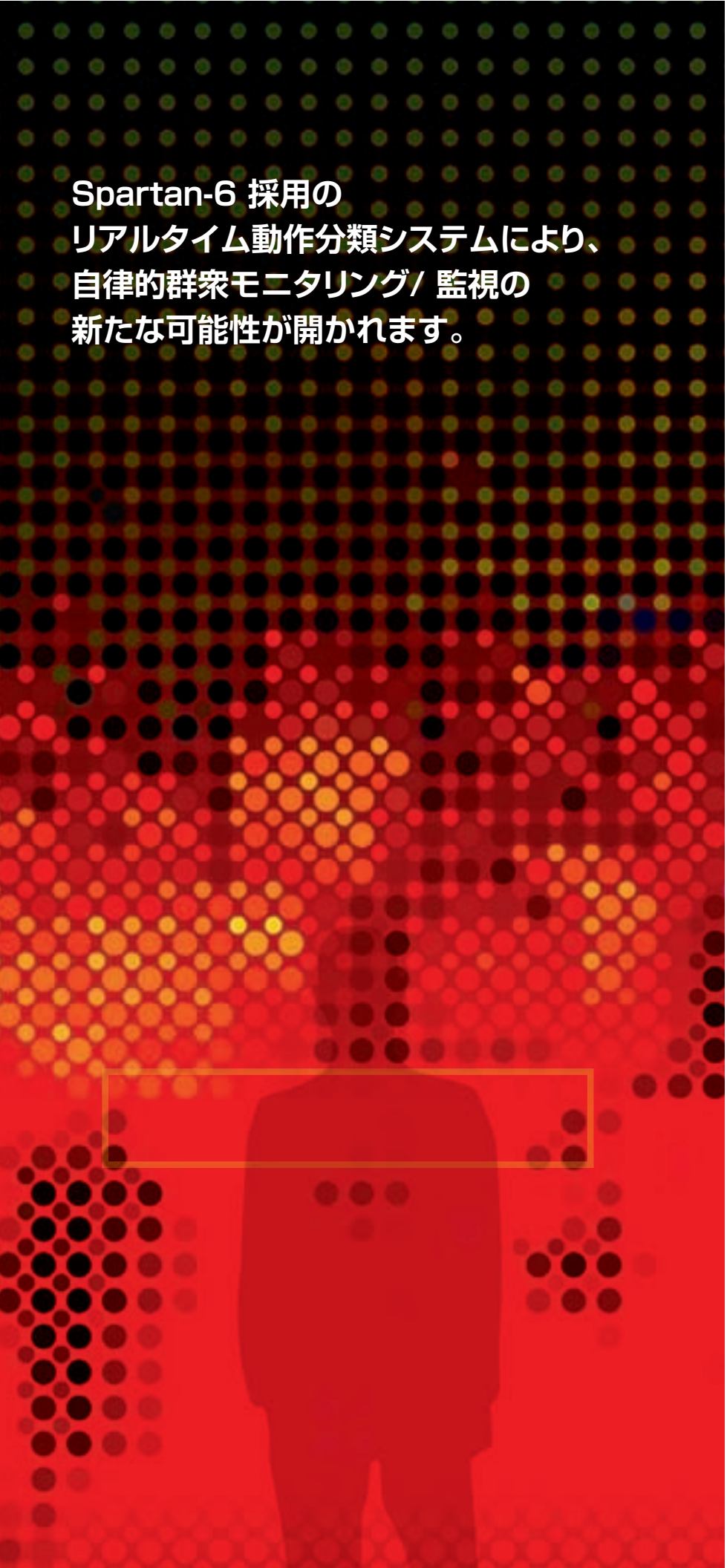
Xilinx FPGAs Advance Autonomous Monitoring of Crowds

ザイリンクス FPGA による先進の自律的群衆モニタリング システム

Muhammad Bilal
MSc Candidate
Center for Advanced Studies in Engineering
Islamabad, Pakistan
bilal.case.edu@gmail.com

Dr. Shoab. A. Khan
Professor
Center for Advanced Studies in Engineering (www.case.edu.pk)
Islamabad, Pakistan
shoab@case.edu.pk

Spartan-6 採用のリアルタイム動作分類システムにより、自律的群衆モニタリング/ 監視の新たな可能性が開かれます。



近年、群衆の監視およびモニタリングの重要性が高まってきています。政府やセキュリティ関連部門は、公共の場の人混みをインテリジェントにモニタリングし、手遅れにならないうちに異常なアクティビティを検出するための高度な手法を探し始めました。しかし、その目標を効果的に達成するには、まだいくつかの壁があります。たとえば、都市全体で発生し得るすべての群衆の活動を同時に 24 時間モニターする際に、数千台の CCTV カメラが設置されている場合は、完全手動のモニタリングではもちろん不可能です。

この問題を解決するには、先進的なビデオ解析手法を使用して、群衆の活動を自律的にモニターし、異常な事象が発生したら中央制御ステーションに即座に通知する、インテリジェントな新しいカメラ / ビジョン システムを開発する必要があります。

インテリジェントなカメラ / ビジョン システムを設計するには、通常のイメージング センサーと光学系に加えて、ビデオ解析のために高性能なビデオ プロセッサが必要です。そのようなビデオ プロセッサが必要な理由は、高度なビデオ解析手法のほとんどで計算集約型のビデオ処理アルゴリズムが使用されており、高い処理能力が要求されるからです。

このような性能重視のアプリケーションには、FPGA が最適です。現在では、ザイリンクスの Vivado® Design Suite の高位合成 (HLS) によって可能になった UltraFast™ 設計手法により、最適かつ高性能な FPGA デザインを容易に作成できるようになりました。さらに、FPGA のリコンフィギュレーション可能なロジック内にザイリンクス MicroBlaze™ などのエンベデッド プロセッサを統合することで、複雑な制御フローを持つアプリケーションを容易に FPGA に移植できます。

このテーマを念頭に置いて、今回、Vivado HLS、ザイリンクスのエンベデッド開発キット (EDK)およびISE® Design Suite ソフトウェア ベースEDA ツールを使用して、群衆の動作分類やモニタリング システムのプロトタイプを設計しました。その設計は、「ソフトウェア制御、ハードウェア アクセラレーション アーキテクチャ」と筆者らが

みなす手法に基づいています。設計のターゲットは、ザイリンクスの低コスト Spartan®-6 LX45 FPGA としました。システム設計全体は 短期間での完成、リアルタイム性能、低コスト性、設計の高い柔軟性のそれぞれについて期待が持てる結果を示しました。

システムの設計

全体のシステム設計は、2 つの段階で行いました。第 1 段階では、群衆の動作分類アルゴリズムを開発しました。このアルゴリズムを検証した後、第 2 段階として、このアルゴリズムを FPGA に実装しました。この第 2 段階で焦点となったのは、FPGAベースのリアルタイム ビデオ処理アプリケーションのアーキテクチャの設計でした。これには、リアルタイム ビデオ パイプライン処理の開発、ハードウェア アクセラレータの開発、その後にそれらを統合してアルゴリズム制御とデータ フローを実装してシステム設計を完成させる作業が含まれています。

では、これらの開発段階を 1 つずつ見ていきましょう。まず、アルゴリズムの設計について簡単に説明した後、FPGA プラットフォームへのインプリメンテーションについて詳細に説明します。

アルゴリズムの設計

群衆の監視およびモニタリングについて、さまざまなアルゴリズムが文献等に提案されています。ほとんどのアルゴリズムにおいて、最初に、群衆シーン内に注目点を検出 (または配置) し、経時的にトラッキングして、動きの統計を収集します。次に、この動きの統計を事前計算された複数の動きモデルに当てはめ、異常な活動を予測します [1]。改良版の 1 つとして、注目点をクラスター化し、個々の注目点の代わりに、これらのクラスターをトラッキングする方法があります [2]。

今回使用した群衆の動作分類アルゴリズムは、動作推定を行うために KLT (Kanade-Lucas-Tomasi) 特徴トラッカーなどの従来のアプローチの代わりに、テンプレートマッチング方式を使用した点を除き、同じ概念に基づいています。コントラストが低い場合もしくは変動する場合は、テンプレートマッチング方式の方が、計算量は若干増化するものの、優れた動き推定を行えることが分かりました。

テンプレートマッチング方式の動き推定を行うため、ビデオ フレームを、複数の小さい四角形から成るグリッドに分割し、絶対差の加重和 (SWAD) ベースの手法を使用して、各四角形内の現在の画像と直前の画像の動きを計算しました。各四角形は、順に、その特定の位置における 2 つのフレーム間の動きの大きさと方向を示す 1 つの動きベクトルを構成します。その結果、画像全体で 900 を超える動きベクトルが計算されます。動きベクトルの計算に必要なステップを図 1 に示します。

画像内のオクルージョン領域やゼロ コントラスト領域に対する堅牢性を実現するため、加重ガウス カーネルも使用しました。また、動きベクトルの計算において、ある四角形の処理は、他の四角形の処理に依存しないため、このアプローチは FPGA での並列インプリメンテーションに適しています。

画像全体の動きベクトルが計算された後、その統計的特性が計算されます。統計的特性には、動きベクトルの平均長、動きベクトルの数、動きの主要な方向、その他の同様なメトリックスが含まれます。

動きベクトルの方向の 360 度ヒストグラムも計算され、標準偏差、平均値、変動係数などの特性がさらに解析されます。これらの統計的特性は、事前計算された動きモデルに当てはめられ、現在の動きがいくつかのカテゴリの 1 つに分類されます。複数のフレームにわたってこれらの統計的特性を出すことで、分類結果を確認します。

事前計算された動きモデルは、統計的特性を考慮して観察対象の動きを分類する加重決定木分類器という形で構築されます。たとえば、動きが多く、シーン内の運動量が突然変化し、動きの方向がランダムもしくは画像平面にない場合、パニック状態の可能性として分類されます。アルゴリズムの開発には、Microsoft Visual C++ と OpenCV ライブラリが使用されました。このアルゴリズムのデモの詳細については、本稿の末尾に記載する Web サイト リンクからご覧ください。

FPGA のインプリメンテーション

システム設計の次の段階は、アルゴリズムの FPGA への実装でした。このようなインプリメンテーションならではの設計課題として、ビデオ入出力とフレーム バッファーが FPGA ベースの設計の一部になることなどが挙げられます。また、リソース ファイルと性能が制限されるために、適宜デザインの最適化が必要な場合があります。

これらのデザイン上や、その他のアーキテクチャ上の検討事項を踏まえて、FPGAベースのインプリメンテーション全体を 3 つのセクションに分割しました。1 つ目のセクションでは、必要なビデオ入出力とフレーム バッファーを行うため、一般的なリアルタイム ビデオ パイプライン処理を FPGA 内に開発しました。2 つ目のセクションでは、アルゴリズム固有のハードウェア アクセラレータを開発しました。最後に、3 つ目の設計段階では、それらを統合して、アルゴリズム制御とデータフローを実装しました。これで、FPGAベースのシステム設計全体が完了です。

では、プロセスのそれぞれのセクションを詳細に確認してみましょう

リアルタイム ビデオ パイプライン処理

リアルタイム ビデオ パイプライン処理は、FPGA プラットフォーム用のビデオ処理アプリケーションを開発する上で、最も重要な構成要素です。パイプライン処理を行うことで、ビデオ入出力およびフレーム バッファーに関する複雑なメモリ管理がユーザーから見えなくなり、処理対象のビデオ フレーム データにアクセスするためのシンプルなインターフェイスが提供されます。

これについては、高機能で商用認可を受けたビデオ パイプライン処理 [3] がいくつかありますが、この目的にはカスタムのビデオ パイプライン処理を構築することを選びました。このパイプライン処理には、ザイリンクス EDK を、ビデオ入出力データを扱うカスタム ビデオ キャプチャ ポート / 表示ポートと合わせて使用しました。他のザイリンクスFPGA ファミリに対しても、パイプライン処理を簡単にコンフィギュレーション可能です。

ビデオ キャプチャ ポートは、ビデオ ADC からの入力ビデオ ストリーム データをデコードして、ローカルにバッファーします。その後、メイン メモリに転送され、ビデオ フレームが構築されます。同様に、ビデオ表示ポートは、ローカル バッファーにあるビデオ フレーム データをエンコードし、表示のためにビデオ DAC に転送します。ビデオ入力ポートと出力ポートは、このようなメイン メモリとのビデオ データ トラフィックを処理する MicroBlaze ホスト プロセッサのメイン ペリフェラル バスに接続されています。

両ビデオ ポートは、割り込みを生成でき、新しいデータがビデオ入力ポートに入ってきたとき、もしくはビデオ出力ポートにデータを送る必要があるときに MicroBlaze プロセッサに通知できます。MicroBlaze プロセッサがすぐにビデオ ポートにサービスを提供できない

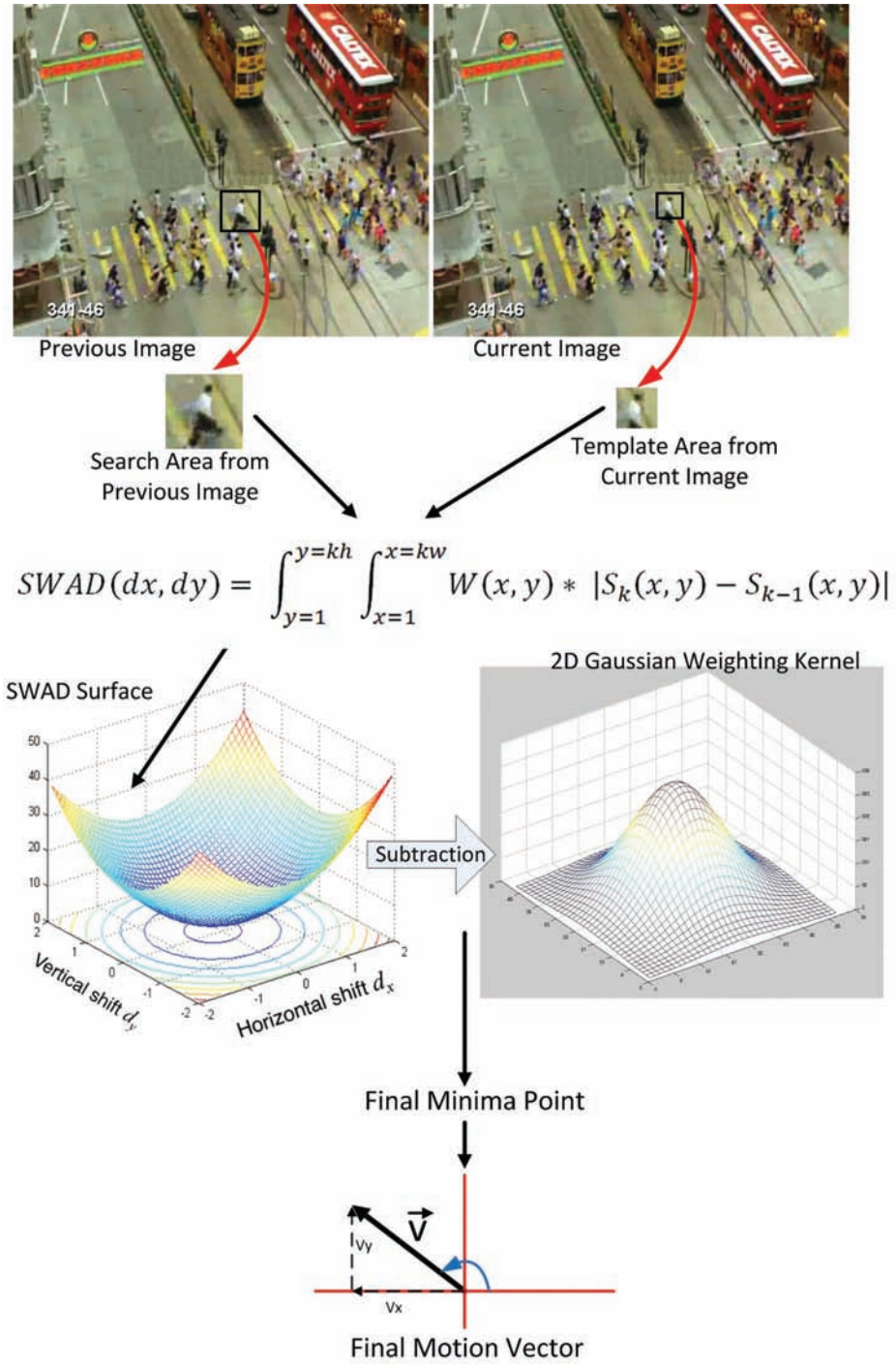


図 1 – 画像のキャプチャ (最上部) から始まる、動きベクトル計算の各ステップ

場合にバッファー オーバーフローやアンダーランが発生しないように、両ビデオ ポートは、「ピンポン」バッファー管理方式を用います。図 2 は、両ビデオ ポートと MicroBlaze プロセッサ間のインターコネクトを示しています。

両ビデオ ポートは、ビデオ入出力ストリーム内のビデオ ライン番号、フィールド ID (インターレース ビデオの場合)、およびその他の制御情報を検出および生成するよう設計されています。ビデオ入力ポートによって十分な量のビデオ データがバッファーされるか、ビデオ表示ポートによって要求されると、これらの情報は、ビデオ ポートの割り込みサービス ルーチン (ISR) によって、MicroBlaze プロセッサに受け渡されます。ISR はさらに、ビデオ ポートのローカル メモリとメイン メモリ間で DMA によるビデオ データ転送を行います。

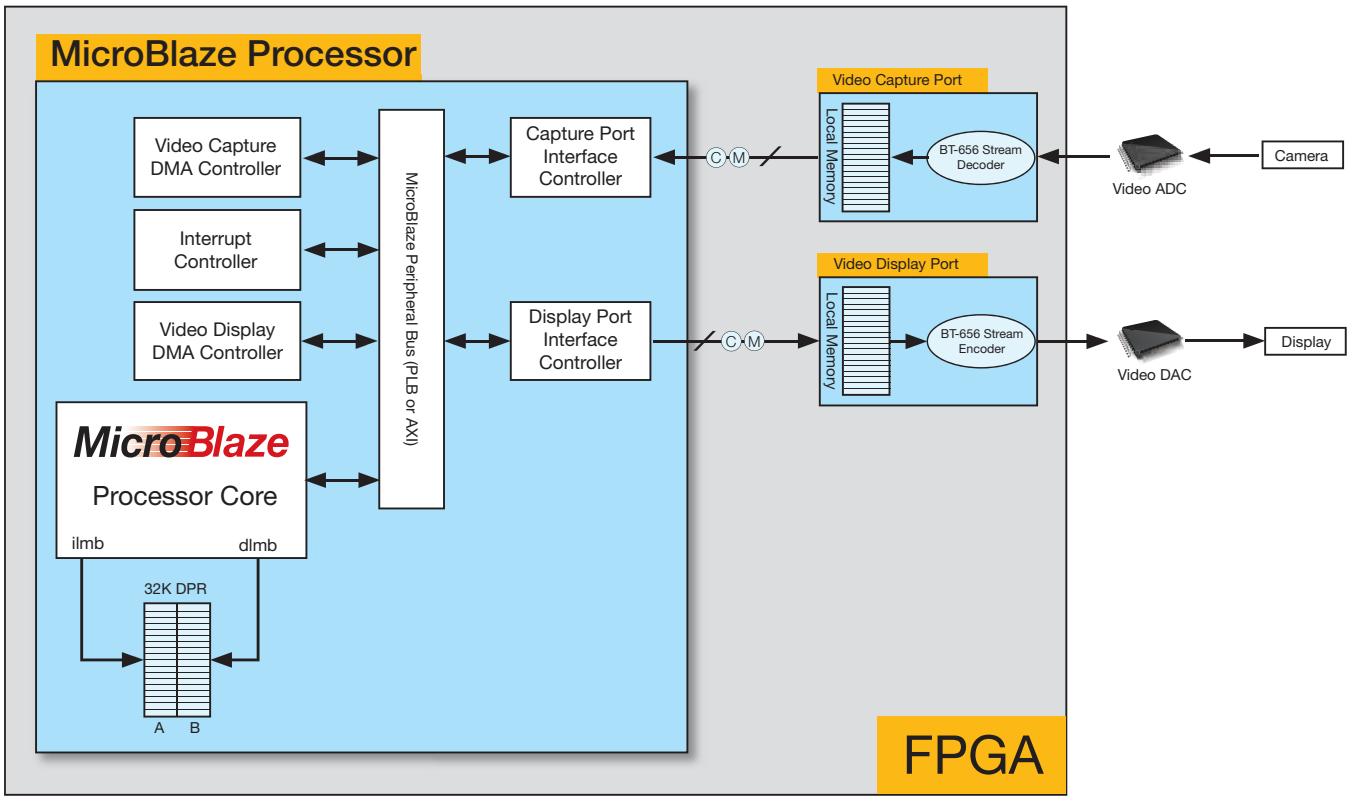
ビデオ ポート ISR に加えて、一連の高レベルなビデオ フレーム キュー管理関数 (ビデオ フレーム キュー API と呼ぶ) が、ビデオ ポート ISR とユーザー レベル アプリケーションの間で動作します。この API は、複数のキャプチャおよび表示フレームのキューを管理し、ダブル / トリプル フレーム バッファー方式に対応します。MicroBlaze プロセッサ上で実行中のユーザー アプリケーションは、ビデオ フレーム キュー API 関数を使用することで、簡単にビデオ キャプチャ フレームを取得したり、ビデオ表示フレームを送出したりできます。各階層レベルにおける、これらの関数を図 3 に示します。

システム内のさまざまな構成要素を接続するホスト プロセッサとして MicroBlaze を使用することで、いくつかのメリットがあります。たとえば、ビデオ フレーム データをビデオ ポートにロードまたは保存するために、MicroBlaze プロセッサと広範囲の外部メモリ (SRAM、SDRAM など) のインターフェイスを簡単に構築できます。同様に、ビデオ ポートとメイン メモリ間のビデオ データ転送には EDK に含まれる DMA コントローラーを使用できます。また、MicroBlaze プロセッサと同じ方法で、カスタムのハードウェア アクセラレータとのインターフェイスを構築しました。

これらのビデオ フレーム キュー API 関数と、ビデオ ポート ISR、ビデオ入出力ポートを合わせて、今回のデザインのビデオ パイプライン処理が構成されます。図 4 は、FPGA でこのビデオ パイプライン処理を使用してキャプチャ、処理し、表示した実際のビデオ フレームを示しています。また、計算された動きベクトルのズームアウト表示を行う Picture in Picture 機能も示されています。

Vivado HLSベースの ハードウェア アクセラレータ

上述した群衆の動作分類アルゴリズムにおいて、最も時間がかかる、計算集約型のタスクは、動きベクトルの計算です。もう一方のシステム タスク (分類の実行) には、ピクセル レベルの処理が含まれていないため、シンプルで簡単に終わります。このような設計上の



Note: (C) (M) Donate 'Control' and 'Memory' interfaces respectively.

図 2 – ビデオ ポートとそのインターコネクト

検討事項を考慮して、動きベクトルを計算するためにハードウェア アクセラレータを構築しました。アクセラレータは、C/C++ 言語で設計し、テストした後、ザイリンクスの Vivado HLS を使用して RTL 合成しました。

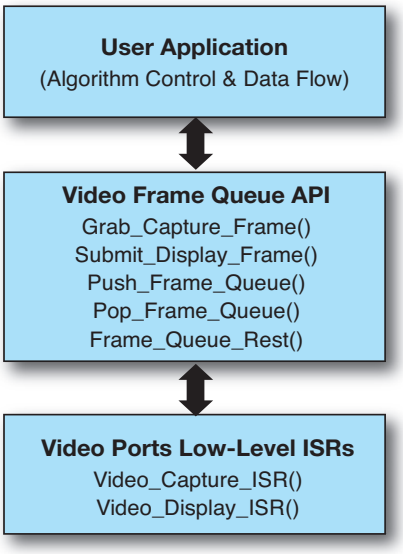


図 3 – ビデオ ポート ISR と
ビデオ フレーム キュー API 関数

Vivado HLS で生成した RTL コードの重要な特長の 1 つは、大部分が最適なコードであることです。Vivado HLS は、配列アクセス (配列に格納されたピクセル データなど) をメモリ インターフェイスに合成し、コードを解析して必要なアドレスを自動生成します。また、いわゆる「ストライド」方式のメモリ アクセスを高速化するため、事前計算が可能なオフセットや定数を解析します。ストライド メモリ アクセスは、画像の複数のラインからデータにアクセスするときに発生します (2D たたみ込みなど)。

Vivadoベースの アクセラレータを設計する際に特に考慮したのは、動きベクトルの計算を並列化し、メイン メモリからのデータの読み出しを最大化することでした。そのため、8 つのブロック RAM を使用してビデオ フレーム データを並列にロードおよび保存しました。ハードウェア アクセラレータ コアは、4 つの動きベクトルを並列に計算でき、そのために、これら 8 つのブロック RAM がすべて使用されます。メイン メモリからこれらのブロック RAMへのデータ ポンプは、MicroBlaze プロセッサによって DMA 制御されます。

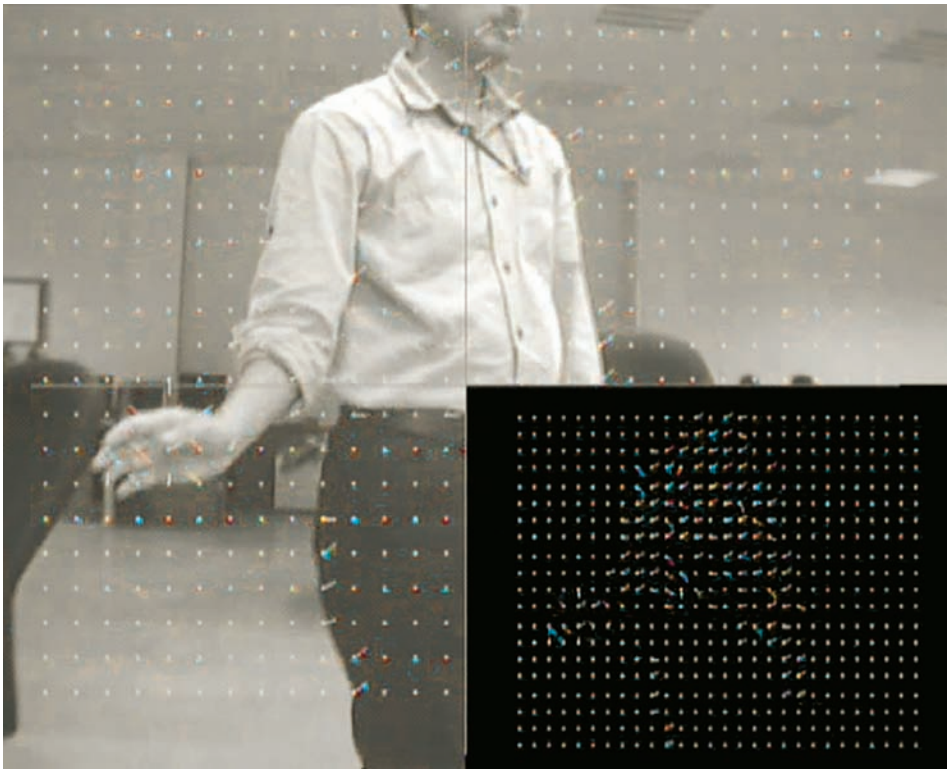


図 4 – 実際に FPGA で処理したフレーム (右下に動きベクトル グリッドを重ねて表示)

Vivado HLS で生成したハードウェア アクセラレータには、ハードウェア アクセラレータを開始および停止するために必要な自動生成のハンドシェイク信号が含まれています。これらのハンドシェイク信号には、「開始」、「ビジー」、「アイドル」、「完了」のフラグがあります。これらのフラグは、ハンドシェイクを行えるように、MicroBlaze プロセッサに GPIO を介して配線されます。ハードウェア アクセラレータ、8 つのブロック RAM、MicroBlaze プロセッサのメイン ペリフェラル バス間のインターコネクトを図 5 に示します。

図 5 において、SA1、TA1 ～ SA4、TA4 と名付けられたブロック RAM のサイズは、それぞれ 16 KBです。SA1、TA1 ～ SA4、TA4 の各ペアは、1 行全体の動きベクトルを計算するのに十分なデータを保持できます。したがって、実行完了後、ハードウェア アクセラレータは、4 行分の動きベクトルを出力し、同じブロック RAM メモリに書き戻します。MicroBlaze プロセッサは、計算されたこれらの動きベクトルを読み戻し、結果を動きベクトルのグリッドとしてメイン メモリにコピーします(図 4 は、実際のフレームにこのハードウェア アクセラレータで計算された動きベクトルのグリッドを重ねて表示したものです)。

ハードウェア アクセラレータは、200MHz で動作し、イメージ全体の動きベクトルを計算するために必要なすべての処理を完了するのにかかる時間は、メモリとのデータ転送をすべて含めて 10 ミリ秒未満です。

アルゴリズム制御とデータ フロー

ビデオ パイプライン処理とハードウェア アクセラレータの開発が終わると、システム設計の最後のステップは、これらの 2 つの要素を MicroBlaze ホスト プロセッサと統合し、ザイリンクスのソフトウェア開発キット (SDK) を使用して、アルゴリズム制御とデータ フローをユーザー レベル アプリケーションに C/C++ で実装することです。アルゴリズム制御とデータ フローをザイリンクス SDK で実装すると、設計に大きな柔軟性が生まれます。これは、新しいハードウェア アクセラレータを同じ方法で設計および統合し、必要な

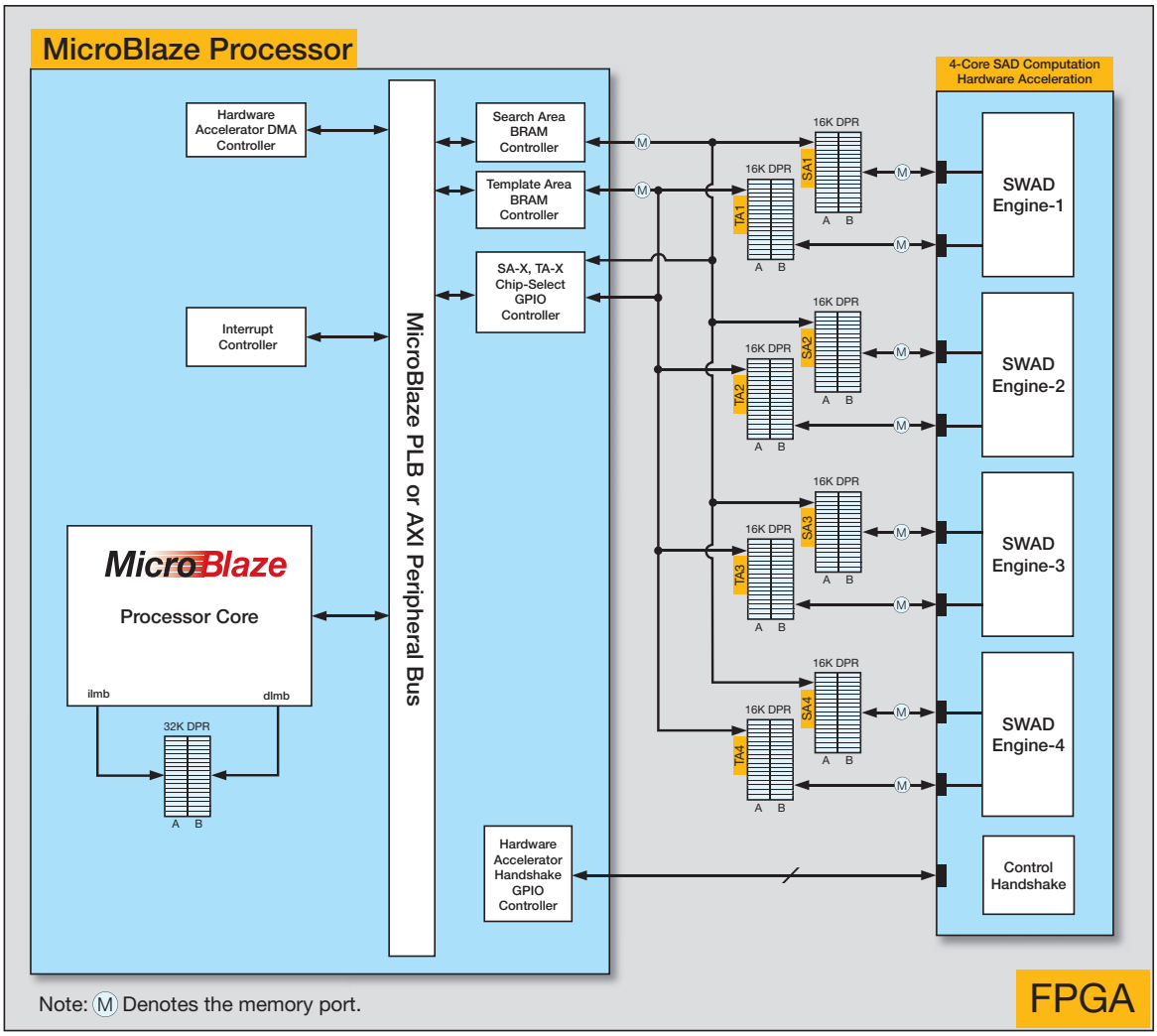


図 5 – Vivado HLSベースのハードウェア アクセラレータとそのインターコネクト

制御とデータ フローを変更して新しいハードウェア アクセラレータを組み込めるためです。その結果、フルソフトウェア インプリメンテーションのようにフレキシブルで、フルハードウェア インプリメンテーションのように高性能な、一種のソフトウェア制御ハードウェア アクセラレーション デザインが可能になります。

群衆の動作分類アルゴリズムの制御とデータ フローでは、まず、ビデオ フレーム キュー API 関数を通してビデオ フレームをキャプチャします。フレームを取得すると、ユーザー アプリケーションは、現在と直前のビデオ フレーム データをハードウェア アクセラレータに転送し、動きベクトルを計算させます。

この時点で、動きベクトルの統計的特性と分類結果がソフトウェアによって計算されます。その理由は、これらのステップにはピクセル レベルの処理が含まれておらず、処理のオーバーヘッドが極めて少ないためです。分類結果が計算されると、オンスクリーン ディスプレイ (OSD) 関数によって、その結果と動きベクトルが処理済みフレームに表示されます。OSD 関数も、ザイリンクス SDK で、C/C++ で実装されます。

これらすべての構築ブロック (リアルタイム ビデオ パイプライン処理、ハードウェア アクセラレータ、アルゴリズム制御 / データ フロー) が完成すると、全体のシステム設計は完了です。結果の精度を確認するため、今回の FPGAベースのインプリメンテーションを、既存のデスクトップ PC ベースのインプリメンテーションと比較したところ、2 つの結果は同じであることが確認できました。システムのテストには、ミネソタ大学のデータベース (http://mha.cs.umn.edu/proj_recognition.shtml) と www.gettyimages.com のさまざまなテスト用ビデオを使用しました。

インプリメンテーション結果

設計全体で、Spartan-6-LX45 FPGA に搭載されているスライス LUT のわずか 30%、ブロック RAM の 60%、DSP48E 乗算器リソースの 12% が使用されます。図 6 は、ハードウェア セットアップ (最上部) と実際のシステム出力を示しています。ハードウェア セットアップは、Digilent Atlys Spartan-6 FPGA ボードと、ビデオ ADC および DAC を

使用してこの FPGA にビデオ入出力機能を付与するカスタムのビデオ インターフェイス カードで構成されます。このシステムの詳しいデモは、次の Web サイトでご覧いただけます。

http://www.dailymotion.com/video/x2av1wo_fpga-based-real-time-human-crowd-motion-classification-demo_school

http://www.dailymotion.com/video/x23icxj_real-time-motion-vectors-computation-on-fpga_news

http://www.dailymotion.com/video/x28sq1c_crowd-motion-classification-using-motion-vectors-statistical-features_school

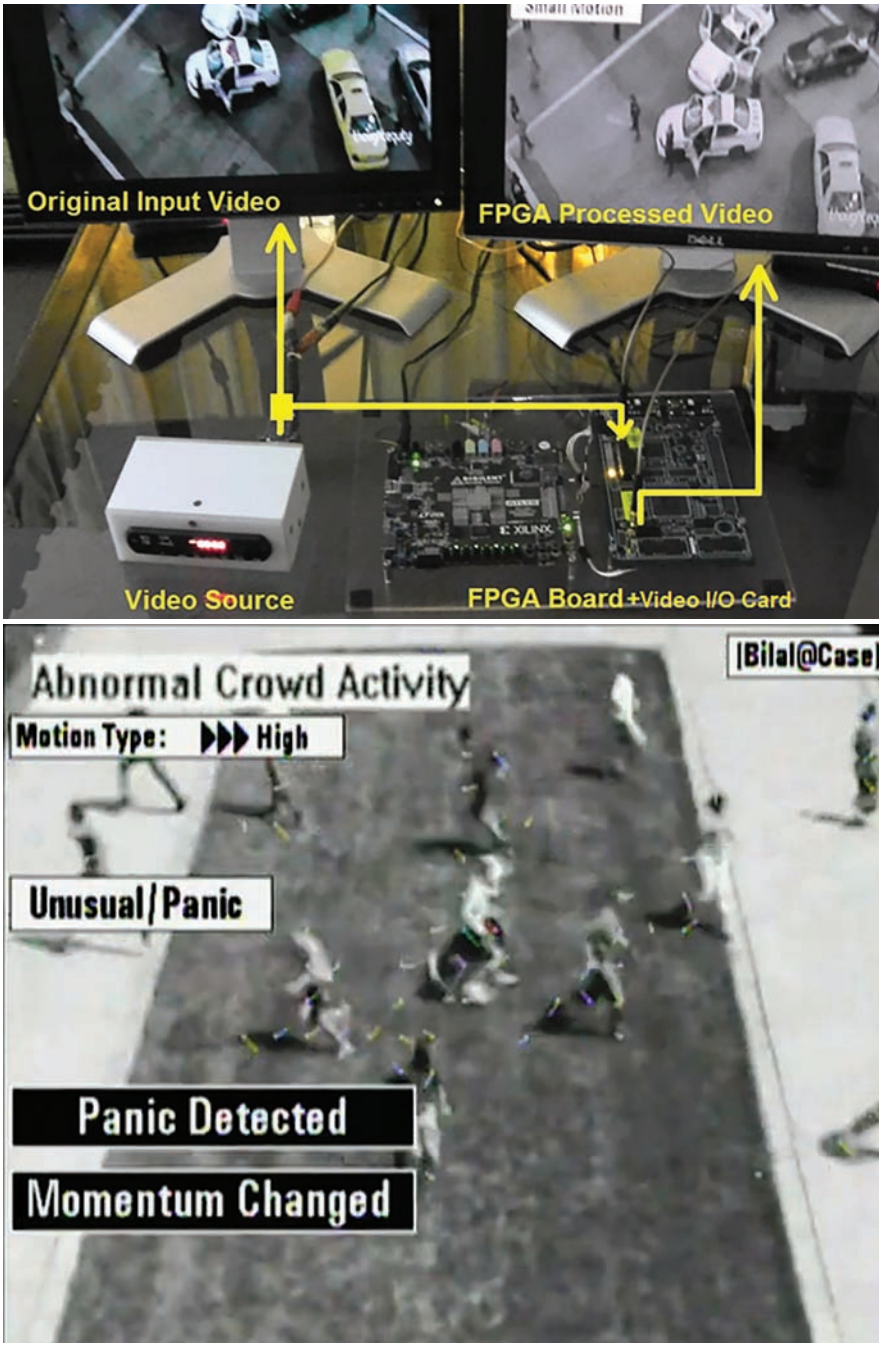


図 6 – ハードウェア セットアップ (最上部) と、実際に FPGA で処理されたフレームで、あるシーンをパニックの一種と分類している様子

大きな将来性

FPGA は、リアルタイム ビデオ プロセッシングなど、性能重視のアプリケーションにおいても理想的なプラットフォームです。このようなアプリケーションを開発するには、選定した FPGA の性能を最大限引き出すために、いくつかのアーキテクチャ上の検討事項があります。さらに、EDK や Vivado HLS などの最新ツールを使用することで、より効率的に、以前よりもかなり短い開発時間でシステム全体を設計できます。

したがって、このプロジェクトで実証されたように、性能重視のアプリケーションを上述のツールを使って FPGA に実装する手法には、大きな可能性があります。実用可能なプラットフォームがすでに存在していることから、この取り組みをさらに進めて、交通状況の自動モニタリング、病院患者の自動モニタリング、その他の多くのアプリケーションのより多くの技術的な問題に対処できるようにしたいと願っています。

参考文献

1. Ramin Mehran, Mubarak Shah, “Abnormal Crowd Behavior Detection Using Social Force Model,” IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), Miami, 2009
2. Duan-Yu Chen, Po-Chung Huang, “Motion- based unusual-event detection in human crowds,” Journal of Visual Computation and Image Representation, Vol. 22 Issue 2 (2011), pages 178–186
3. OmniTek OSVP: *http://omnitek.tv/sites/default/files/OSVP.pdf*

謝辞

共著者の Shoab A. Khan 教授とその素晴らしいアイデアに感謝します。また、Dr. Darshika G. Parera と Dr. Umair Ahsun から大いなるインスピレーションをいただきました。

Test New Memory Technology Chips Using the Zynq SoC

新世代テクノロジを使用したメモリ チップのテスト環境を Zynq SoC で構築

ザイリンクスの ZC706 評価キットを用いたプラットフォームは、Qualcomm 社の MRAM テスト システムで十分な高速性と柔軟性を実証しました。

Botao Lee
Senior Staff Engineer
Qualcomm Technologies, Inc.
blee@qti.qualcomm.com

Baodong Liu
Staff Engineer
Qualcomm Technologies, Inc.
baodongl@qti.qualcomm.com

Wah Hsu
Senior Staff Engineer
Qualcomm Technologies, Inc.
wahh@qti.qualcomm.com

Bill Lu
Staff Engineer
Qualcomm Technologies, Inc.
xiaolu@qti.qualcomm.com

エレクトロニクス業界では、PRAM、MRAM、RRAM などの新しいメモリ テクノロジの開発に精力的に投資を行っています。新しいメモリ テクノロジを使用したテスト チップの性能は急速に向上していますが、従来のメモリと競合、もしくは従来のメモリを置き換えて本格的に普及するまでには、まだ課題が残ります。

一般的に、新しいメモリ テクノロジのテスト チップは、入手可能になった時点で、縮退故障、遷移故障、アドレス デコード故障といったメーカーに関する問題について、基本的なテストがすでに実施されています。しかし、チップが確実にアクセス可能な最大速度や、チップのアクセス速度がコンピューティング システム全体のパフォーマンスに与える影響など、性能関連のテストという形で、別種のテストも行う必要があります。

計画された性能テストを適切に実施するには、チップにアクセスするためにコンフィギュレーション可能なデジタル波形を生成できるテスト環境が必要です。また、チップのアクセス速度がシステムに与える影響を測るために、テスト環境においてコンピューティング環境全体を構築できる必要があります。このようなニーズを満たすテスト環境は、さまざまな方法を用いて作成もしくは購入することができます。しかし、当社 のチームでは、ザイリンクスの Zynq®- 7000 All Programmable SoC ZC706 評価キットを使用して、独自の環境を作成することにしました。

メモリの入出力

DRAM、SRAM、フラッシュなどの従来のメモリ テクノロジは、電荷を使用して 1 と 0 を各メモリ セルに格納します。DRAM は、PC やモバイル コンピューティング デバイスにおいて、プログラムを実行したり、一時的なデータを保存したりするのに広く使われています。SRAM は、マイクロプロセッサのキャッシュ メモリやレジスタ ファイルとしてよく使われています。また、消費電力が特に重視されるエンベデッド システムでの使用にもよく見られます。一方、フラッシュ メモリは、DRAM や SRAM とは異なり、システムの電源を切断した後もデータを保持する永続的なメモリです。フラッシュ メモリの実行速度は DRAM や SRAM よりも遅く、プログラミング サイクルが極めて多い場合、損耗することがあります。

従来のメモリ テクノロジが電荷をベースとしているのに対し、新しいメモリ テクノロジは、記憶素子のその他の

物理的属性をベースとしています。たとえば、磁気抵抗 RAM (MRAM) のメモリ素子は、薄い絶縁物の層を 2 枚の強磁性体プレートで挟んだ構造をしています。各プレートは磁化を保持します。そのうち一方は永続的で、もう一方は外部の磁界によって変更可能で、データの保存に使用できます。保存したデータは、素子の電気抵抗を測定することで読み出されます。MRAM は、SRAM 並みの速度と、DRAM 並みの集積度を持ちます。MRAM は、フラッシュ メモリと比べると非常に高速で、プログラミングによる劣化がありません。

必要条件の分析

MRAM テスト チップの評価スキームを設計するにあたり、次の事項を検討した上で、Zynq SoC アプローチに決定しました。

- ZC706 ボードでは、FPGA メザニン カード (FMC) インターフェイスにより、FMC ドーターカードを介して、メモリ テスト チップとの信号のやりとりを高速に行えること。
- Zynq SoC のプログラマブル ロジック (PL) 部分を使用すると、パラメーター変更可能なメモリ コントローラー コアを作成できること。このことは、テスト チップのアクセス速度を可変にするという要件を満たすために必須です。
- 2 つの ARM® A9 コアから成る Zynq SoC のプロセッシング システム (PS) を使用して、テスト チップのアクセス速度をソフトウェアで変更できること。
- また、PS を使用することで、完全なコンピューティング システムを構築可能です。このことは、チップのアクセス速度がコンピューティング環境全体に与える影響を測定するという本テスト システムの要件を満たすために必須です。

ハードウェア アーキテクチャとシステム アーキテクチャ

チップのテスト環境のハードウェア アーキテクチャを図 1 に示します。ソフトウェアは Zynq SoC の ARM A9 プロセッサ上で実行され、メモリ コントローラー コアはプログラマブル ロジックで作成されます。PS とコントローラー コア間には DMA チャネルを配し、大きなデータ ブロックを簡単に移動できるようにしました。メモリ テスト チップは、FMC ドーターボード上にあり、FMC インターフェイスを介して、メモリ コントローラー コアと通信します。

システム アーキテクチャを図 2 に示します。下の 3 つのレイヤーはハードウェア レイヤーで、上の 3 つのレイヤーはソフトウェア レイヤーです。オペレーティング システムに Linux を選択したのは、Linux がオープン ソースで、必要に応じてソース コードを改変できるためです。開発の現段階では改変は行っていませんが、将来的に、新しいメモリ

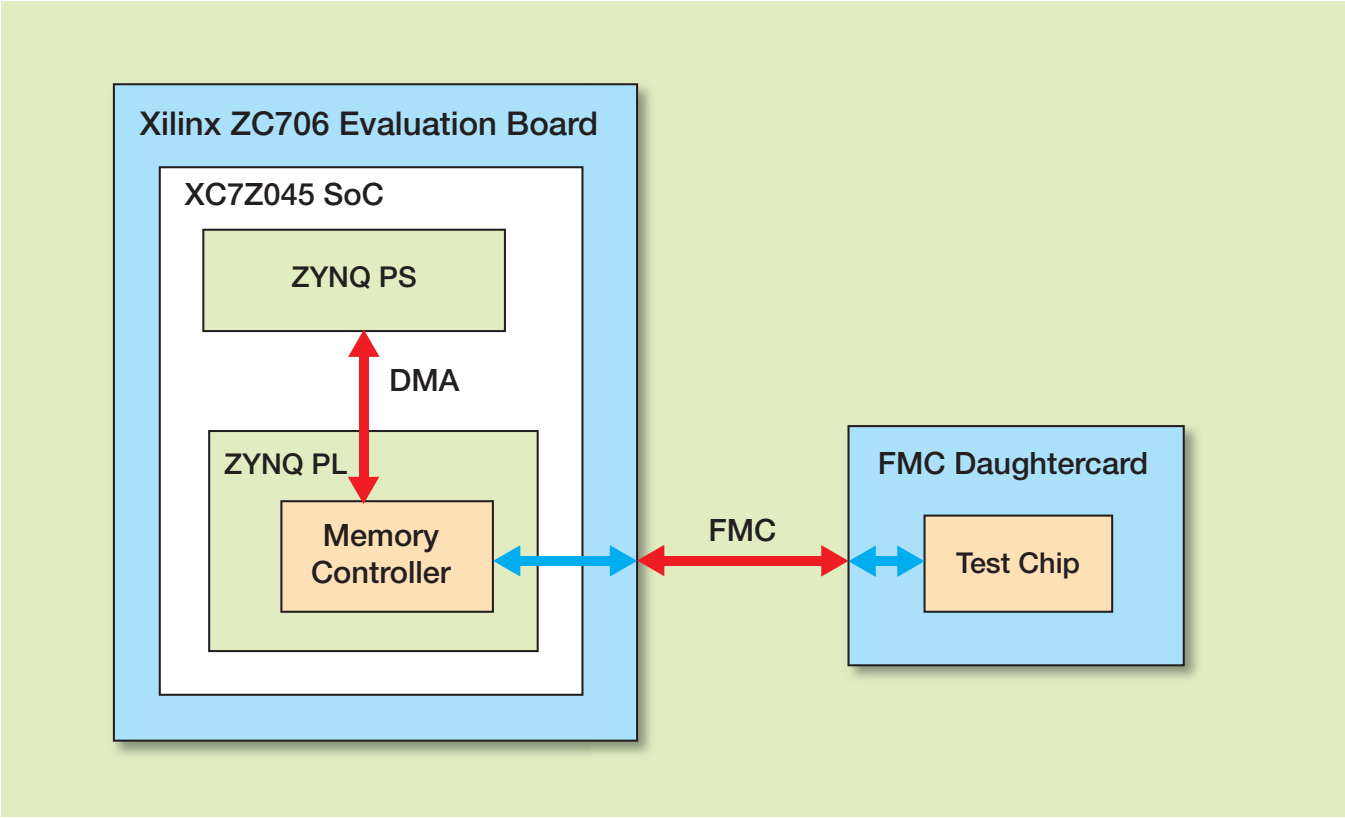


図 1 – テスト環境のハードウェア アーキテクチャ

Application	Test Configuration Performance Profiling		
OS	Linaro Linux		
Device Driver	Memory Controller Device Driver		
Core	2x ARM A9	Memory Controller	
Device	XC7Z045 SoC		Memory Test Chip
Board	ZC706 Evaluation Board		FMC Daughtercard

図 2 – テスト環境のシステム アーキテクチャ

チップならではの属性を活かすために、改変が必要になる可能性があります。

アプリケーション レイヤーで作成したソフトウェアは、2 つのカテゴリーに分けられます。1 つはメモリ コントローラーで、コアのコンフィギュレーション用、もう 1 つはメモリ チップの性能とシステム全体の性能のプロファイリング用です。

ハードウェアとソフトウェアの容易な移行

ザイリンクスFAE のサポートにより、1 ヵ月未満でテスト環境を構築できました。という訳で私たちは、ソフトウェア レイヤーとハードウェア レイヤー間のインターフェイスを設計し実装する作業に注力できました。これは、私たちが Zynq SoC を積極的に採用する理由の 1 つでもあります。つまり、マイクロプロセッサとプログラマブル ロジックが 1 つのデバイス上に集積されており、ハードウェアとソフトウェア間の機能の移行が非常に容易に行えるのです。今回は、ソフトウェア / ハードウェアの分割方法を幾度も微調整しながら、望ましいデザインを完成させることができました。Zynq SoC ベースのシステムで快適に作業するには、ハードウェアとソフトウェアの両方についての適度な知識が必要です。

もう 1 つ気に入っている点として、Vivado® Design Suite ツール チェーンが挙げられます。Vivado ツール環境では、デザイン ブロックが直感的に表示され、レジスタ アドレスを自動的に割り当て、ハードウェア情報をソフトウェア開発プロセスにエクスポートする前にエラーのチェックが可能です。また、Vivado Design Suite に搭載されているインシステム信号レベル デバッグ機能は、RTL の問題の根本原因を特定するためには必要不可欠な機能です。

最後に触れておきたいのは、Linux OS についてです。アプリケーション レベルで作成したソフトウェアは、ほぼ GUI ベースです。Linux OS が普及していることから、Linux GUI 開発に関する既存の経験を有効に活用し、テスト プログラムを迅速にに完成させることができました。

スピードと高い費用対効果

Zynq-7000 All Programmable SoC ZC706 評価キットにより、新しいメモリ テクノロジのチップをテストするための完全なコンピューティング環境を、最小限のコストで、しかも迅速に構築することができました。いずれ、同様の設計手法により、他の目的に対してもこのようなシステムを構築したいと考えております。

Unleashing High-Performance USB Devices with Artix-7 FPGA

Artix-7 FPGA で、高性能 USB デバイスを実現

ザイリンクスの低消費電力 FPGA ファミリにより、バスパワー USB デバイスの設計が容易になります。

Tom Myers
Senior Hardware Engineer
Anritsu Company
tom.myers@anritsu.com

USB (ユニバーサル シリアル バス) は、市場でのポート数が数十億個に上り、ホスト デバイスとペリフェラル デバイス間のギガビット未満の接続では定番のインターフェイスです。しかし、USB 規格には厳しい突入電流や定常状態動作電流の制約があることから、バスパワー デバイス アプリケーションに関しては、FPGA よりも、性能も柔軟性も低いマイクロコントローラー ソリューションに軍配が上がっていました。

ザイリンクスの低消費電力デバイス ポートフォリオに新たに Artix®-7 が加わったことによって、この状況は一変しました。システム レベルの電力変換効率と起動順序に注意を払い、Vivado® Design Suite の消費電力見積もりツールおよび最適化ツールを使用することで、厳しい制約を乗り越えて、高集積のカスタマイズされた高性能バスパワー デバイスを設計できるようになりました。

本稿では、Artix-7 MicroBlaze™ ベースのプラットフォームを使用して USB 2.0 ハイスピード バスパワー デバイスを設計する方法を説明します。Anritsu Company (アンリツ) では、最近、マイクロ波電力計測製品をこのアプローチにより開発しました。新しいデザインの USB 2.0 ハイスピード インターフェイスでは、前世代の製品に使用されていたマイクロコントローラー ベースの USB フルスピード ソリューションと比べて、計測スループットが大きく向上しています。計測スループットが向上すると、生産試験アプリケーションの試験時間が短縮され、顧客のコストを削減できます。

システムの設計

当社プロジェクトの重要な課題は、定常状態での電流引き込み値 500mA (公称 5V) の制限があることでした。そのため、システム設計は、消費電力値の改善に注力しました。データシートの数値から、標準および最大の電流引き込み値を計算して、消費電力バジェット スプレッドシートを作成しました。

消費電力の大部分は、200MBオフチップ メモリの最小要件 によるもので、この要件に最適なのは、標準の 4 ギガビット (Gb) LPDDR2 デバイスであることが分かりました。ベンダーのアプリケーション ノートに記載されていた詳細な手法を使用して、推定されるデータ フロー プロファイルを適用し、このデバイスの引き込み電流値の見積もりを生成しました。また、さまざまなプログラマブル デバイスやその他のソリューションについても、Xilinx Power Estimator などのツールを使用して、機能、クロック スピード、トグル レートなどの推定値を評価しました。

MicroBlaze、メモリ コントローラー (メモリ インターフェイス ジェネレーター (MIG) の使用) により全システムのサブセットを作成し、Vivado の IP インテグレーター ツールを使用して各ペリフェラルへのインターフェイス ブロックを追加することで、候補となるいくつかのデバイスを特定し、消費電力、デバイス サイズ、I/O 見積もりを改良していきました。Vivado の消費電力レポートにより、合成可能なターゲットを素早く取得し、消費電力値を改善できました。

MIG は、LPDDR2 デバイスへのネイティブな AXI 接続を提供していないため、後日社内で開発しました。当社の AXI Shim が完成するまでは、初期的な消費電力見積もりおよびサイズ見積もりビルドとして、MIG が生成した LPDDR2 サンプル デザインを使用していました。図 1 は、その結果得られたシステム アーキテクチャを示しています。

『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907) に記載されているように、デバイスのダイ温度を下げると、リーク電力も少なくなります。私たちのストラテジの 1 つは、デバイスのダイ サイズを最小にし、アプリケーションの厳しいボード面積制約の中で、できる限り大きな物理デバイス パッケージを選ぶことでした。

電圧レールの数を削減することで、変換ロスとレギュレータ回路のコストを最小限にしました。デバイスの消費電力要件を確実に制御した後、定格の USB 5V バス電圧から電圧レールに降圧する電圧変換回路を設計しました。

ここまでは、定常状態の電流引き込み値に注目していました。しかし、突入電流引き込みについても考慮しなければなりません。突入電流を最小化する方法の 1 つは、ソフトスタート機能を持つレギュレータを選定して、シーケンスすることです。FPGA 側の起動順序および電圧上昇時間要件と、USB 側の要件とのバランスに注意する必要があります。

予測不可能を予測可能に

USB デバイスを正常にシャットダウンして取り外すには、さまざまなメカニズムが用意されていますが、現実には、多くのユーザーがいきなり取り外しています。このことは、

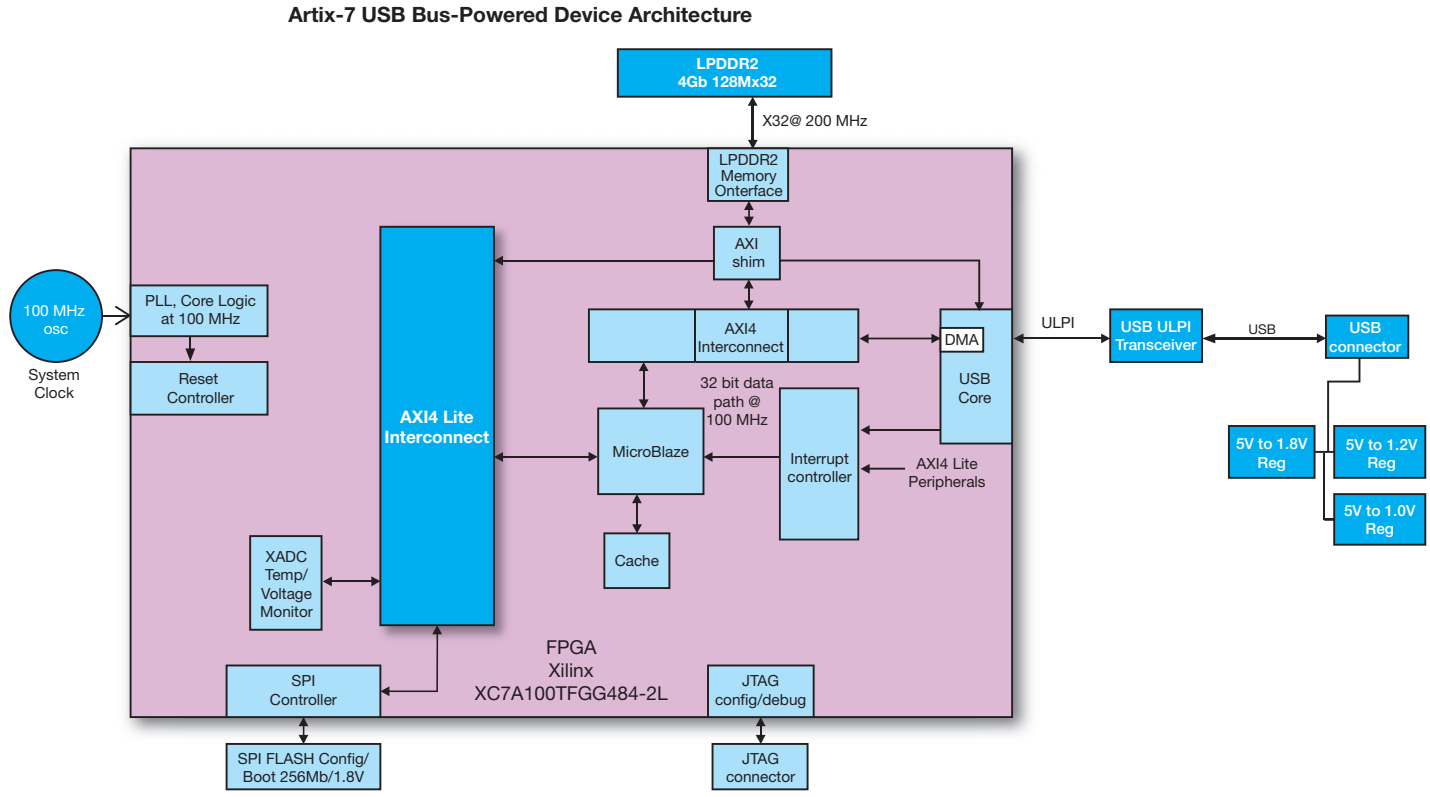


図 1 – Artix-7 ベースの設計のシステム アーキテクチャ

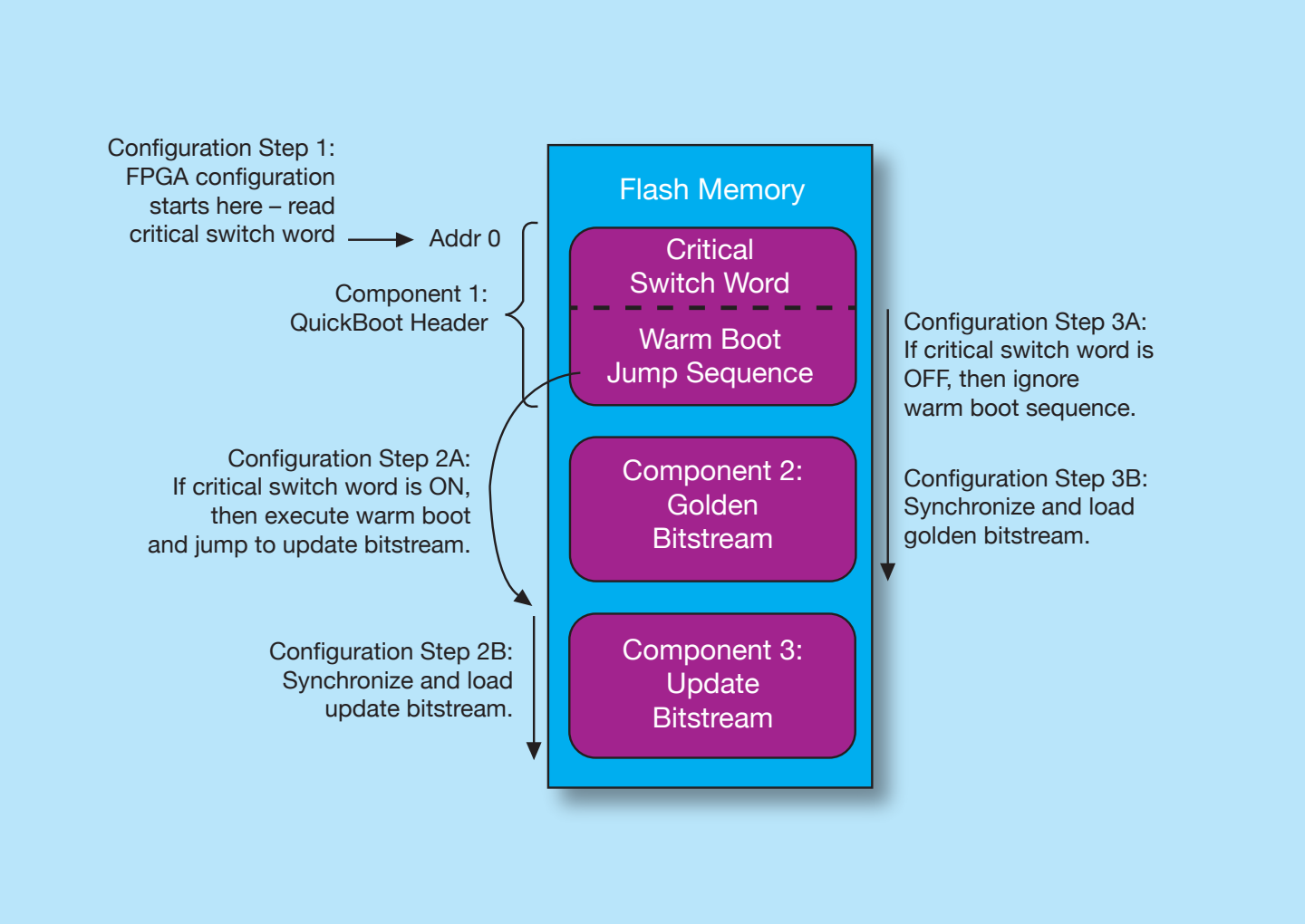


図 2 – QuickBoot フラッシュ メモリの構成要素と設定方法

ファームウェアの更新プロセスの堅牢性が不十分な場合に問題になることがあります。つまり、ファームウェアが不完全状態に陥り、デバイスが無反応な「文鎮」状態となり、顧客の気分を害し、ファームウェア リカバリのためにお金もかけてデバイスを返却してもらう必要性が出てきます。当社の強みは、量産時のテストにおける信頼性とスピードにあり、起動の速さと短いファームウエア更新時間は主な必要条件となります。

私たちは、ザイリンクスのアプリケーション ノート XAPP1081 に紹介されている QuickBoot ゴールデン イメージ ファームウェア更新アーキテクチャおよびプロセスを実装することで、この問題を解決しました。これを図 2 にまとめます。従来の 7 シリーズ フォールバック マルチブート ソリューションでは、コンフィギュレーション フラッシュ メモリ内にビットストリームなどの既知の良好な「ゴールデン」イメージを保持する起動プロセスが提供されています。更新プロセス中、更新済みの「ワーキング」イメージは、ゴールデン イメージの後にメモリにロードされます。更新プロセスが失敗した場合、または、ワーキング イメージに何らかの破損がある場合、FPGA は自動的にエラーを検出し、ゴールデン イメージにフォールバックします。XAPP1081 の QuickBoot メソッドは、この手順を拡張したもので、設定時間が改良され、ゴールデン イメージ更新機能が加わっています。

本プロジェクトが成功を収めたことで、ザイリンクスの次世代デバイスによって、私たちの製品にどのような機能を付け加えていくか検討中です。たとえば、消費電力の大部分は、外付けの SDRAM インターコネクトによって消費されています。新しい 16nm UltraScale+ ラインナップの UltraRAM を使用してこの負荷を削減もしくは解消し、ARM7 対応 Zynq®-7000 All Programmable SoC 製品ラインを当社のアプリケーションに使用できるか、調査することを楽しみにしています。

詳細については、*tom.myers@anritsu.com* までお問い合わせください。

FPGA-Based Fuzzy Controller Manages Sugarcane Extraction

FPGA ベースの ファジー コントローラーで、 サトウキビの搾汁プロセスを管理

Deepali Vyas
Master's Candidate
Mody University of Science and Technology
Lakshmangarh, Rajasthan, India
deepalivyas100@yahoo.com

Yogesh Misra
Research Scholar
Mewar University
Chittorgarh, Rajasthan, India
yogeshmisra@yahoo.com

H. R. Kamath
Director
Malwa Institute of Technology
Indore, Madhya Pradesh, India
rskamath272@gmail.com

ザイリンクスのVirtex-6 FPGA にインプリメントした 3 入力ファジー コントローラーにより、製糖プロセス中のサトウキビの長さを調整

砂糖は、日常で幅広く使われている重要な食材です。世界の粗糖供給量の半分以上がサトウキビから生産されています。インドは、ブラジル に次ぐ世界第 2 位の砂糖生産国で、サトウキビ栽培に携わる農業者、関係者は 6,000 万人で、120 億ドルのビジネスとなっています。

サトウキビの搾汁は非線形的なプロセスになっています。そのため、当チームは、このフローを改良する方法としてファジー ロジックに目を向けることにしました。MITS (Mody Institute of Science and Technology) の分析よると、ザイリンクスの FPGA で設計、実装した当チームのファジー ベースのコントローラーの性能は、従来のコントローラーより、高性能であることが確認されました。1 日当たり 2,500t のサトウキビを粉砕するための特定のパラメーターを使用して、26.6kg/秒の流量が必要です。

3 入力ファジー コントローラーの実装方法について詳しく説明する前に、砂糖生産の基本について理解しておく必要があります。

サトウキビの搾汁方法

図 1は、サトウキビ汁搾汁プロセスの回路図を示しています。サトウキビ農家から製糖工場に送られてくるサトウキビ ビレットは、原料ヤードで重量測定され、搬入されます。そのサトウキビは、クレーンでサトウキビ キャリアに載せられます。動き続けるサトウキビ キャリアにより、サトウキビは製糖工場のフロアに搬送されます。

サトウキビは、まず、2 種類の回転ナイフで処理されます。サトウキビ ナイフは、サトウキビを細かくカットし、シュレッダー ナイフは繊維を処理します。約 1 ～ 2cm に小さくなった繊維は、レーク キャリアによって Donnelly シュートに投入されます。2 本か 3 本のロールで構成された圧搾機で繊維を粉砕し、サトウキビ汁を絞り出します。このプロセスを、5 台または 6 台の圧搾機で繰り返し行います。「バガス」と呼ばれるサトウキビの絞りかすは、ボイラーに運び、燃料として使用します。絞り汁は清浄化の工程に送られた後、絞り汁から砂糖を作るパン セクションに送られます。

処理されるサトウキビ材料は非常に不均一です。搾汁中、サトウキビが不均一であるために、圧搾機の効率が下がり、圧搾機が壊れたり、装置が停止したり、詰まったりします。理想的な搾汁のためには、Donnelly シュート内のサトウキビのレベルを必要な長さに維持する必要があります。

私たちは、ファジー ロジックを使用し、従来のコントローラーよりも適切にレーク キャリアの速度を調整することで、サトウキビの不均一性をなくし、サトウキビの長さを必要な程度に維持できるのではないかと考えました。これが、ファジー ロジックの概念を製糖に応用しようと考えた理由です。

2014 年、第 1 段階として、レーク キャリア上のサトウキビの重さと、Donnelly シュート内のサトウキビの長さという 2 つのばらつきが生じる項目を正確にモニタリングする 2 入力 ファジー コントローラー [1] を設計しました。このコントローラーの目的は、シュート内のサトウキビの長さを一定に保ち、必要な流量の 26.6 kg/秒を維持することです。その結果を従来のコントローラーのものと比較したところ、2 入力ファジー コントローラーによる性能向上が確認できました。サトウキビはロール間で粉砕されるため、実験として、3 つ目のパラメーターとしてロール速度に同じアルゴリズムを導入することにしました。この 3 つ目のパラメーターを追加してみると、ロール速度は、他の 2 つのパラメーターと同じくらい重要な変数であることが分かりました。

従って、2014 年後半、ロール速度を 3 つ目のパラメーターとして導入しました [2]。

3 つ目のパラメーターを追加してアルゴリズムを再設計し、MATLAB® により実装しました。この新しい 3 入力 コントローラーのソフトウェア インプリメンテーションが完了した後 [3]、次のステップとして、ザイリンクスの FPGA にアルゴリズムを実装して、ファジー システム全体を開発しました。FPGA は、電子回路のリアルタイム ハードウェア インプリメンテーションを可能にする、再プログラム可能なシリコン チップです。FPGA は非常に信頼性が高く、費用対効果に優れ、製造工程前に回路性能を確認するための手段が提供されます。ザイリンクスの Virtex®-6 FPGA は、今回のハードウェア インプリメンテーションにとって理想的なソリューションでした。

ハードウェアの設計

図 2 は、3 入力ファジー コントローラーのアルゴリズムを示しています。制御の原理は、2 入力バージョンと変わりませんが、3 入力に合わせて修正し、MATLAB で実装されています。制御の原理は、重さ、長さ、ロール速度の 3 つの条件 (Roll Low (RL): 12cm/秒、Roll Medium (RM): 14.3cm/秒、Roll High (RR): 16.6cm/秒) をコントロールします。

コントローラーを MATLAB で設計した後、入力パラメーターを測定するために必要なハードウェアを設計しました。ロード セルで、レーク キャリアに載せられているサトウキビの量を計測します。シュート内のサトウキビの長さを測るため、長さセンサーを追加しました。また、タコジェネレータ センサーで、ロールの回転速度を計測します。

ロード セル、長さセンサー、タコジェネレータ センサーの出力は、マイクロボルト単位です。これらのメトリクスをプロセスの次のステップで使うためには、単位を増幅して、計測可能な値にする必要がありました。具体的には、マイクロボルト単位からミリボルト単位に増幅しました。増幅は、PSpice の信号コンディショニング システムを使って行いました。その後、コンディショニング システムと直列に接続したアナログ デジタル コンバーター (ADC) を使用して、結果をデジタル値に変換しました。このようにして、増幅済みの値をコントローラーの入力としました。

5 段階のプロセス

ザイリンクスのハードウェアを使用した、ファジー コントローラーの VHDL インプリメンテーションは、5 段階に分けられます。入力のファジー化、ルール評価、IF-THEN 演算、集計、そして、非ファジー化です。

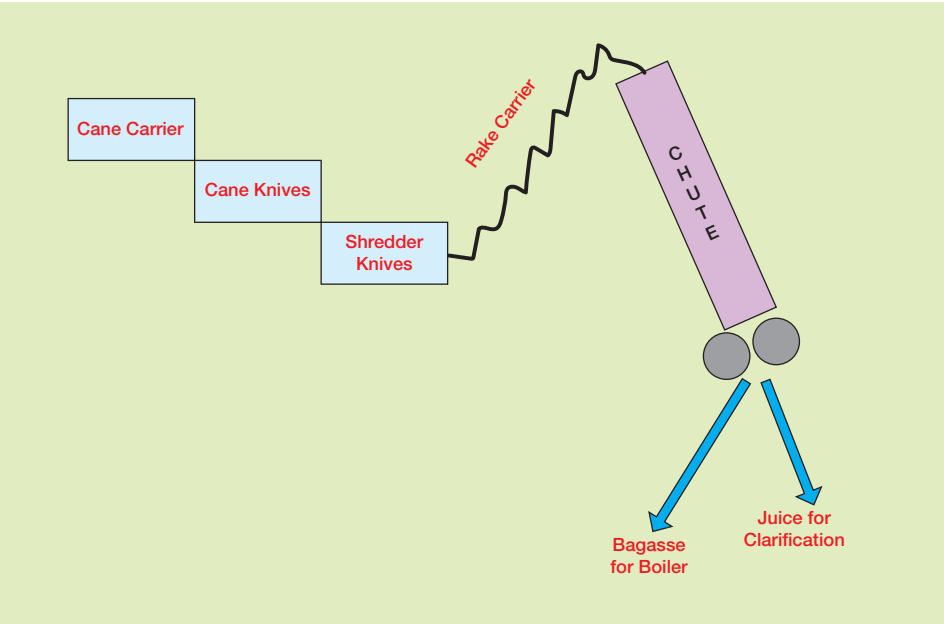


図 1 – サトウキビ汁搾汁プロセスの回路図

ファジー ロジック コントローラーの設計手法として、Mamdani と Sugeno の 2 つの手法があります。Mamdani 手法は、難しく非常に複雑です。研究によると、Mamdani 手法は、連続的に変化する関数を積分することで、2 次元形状の重心を必要とします。そのため、この手法は、演算効率が良くありません。一方、Sugeno 設計手法は、非常にシンプルです。そのため、Sugeno 手法をインプリメンテーションの方法として採用しました。

最初のステップであるファジー化では、切りの良い値をファジーな値に変換し、メンバーシップ関数で表現します。切りの良い値とは、特定の集合に属する値ですが、ファジーな値は、特定の範囲にあるもので、特定の集合には収まりません。

3 つの入力変数は、重さ、長さ、ロール速度です。三角形のメンバーシップ関数を使用して、これらの 3 つの入力変数を表現しました。入力パラメーター「WEIGHT」(重さ) の対象領域は 500kg ～ 1,000kg の範囲で、11 個の三角形の言語変数 (LV) にファジー化されました。入力パラメーター「HEIGHT」(長さ) の対象領域は 0 ～ 180cm の範囲で、7 つの三角形 LV にファジー化されました。入力パラメーター「ROLLSPEED」(ロール速度) の対象領域は 12cm/秒～ 16.6 cm/秒の範囲で、3 つの三角形 LV にファジー化されました。

VHDL コードにおけるファジー化は、次のように行いました。図 3 に示すように、3 つの点と 2 つの勾配で各メンバーシップ関数を定義しました。上向きの勾配 (勾配 1) と下向きの勾配 (勾配 2) は、次の式で評価できます。

S1= (y2-y1/ポイント 2-x1)
S2= (y2-y1/x2- ポイント 2)

ファジー化の次のステップは、メンバーシップ度 (DOM) 関数 (μ) を使用します。今回のアルゴリズムでは、メンバーシップ関数を 4 つのセグメントに分けます。それぞれ、Segment1 (μ=0)、Segment2 {(入力 - ポイント 1)* 勾配 1}、Segment3 {(入力 - ポ

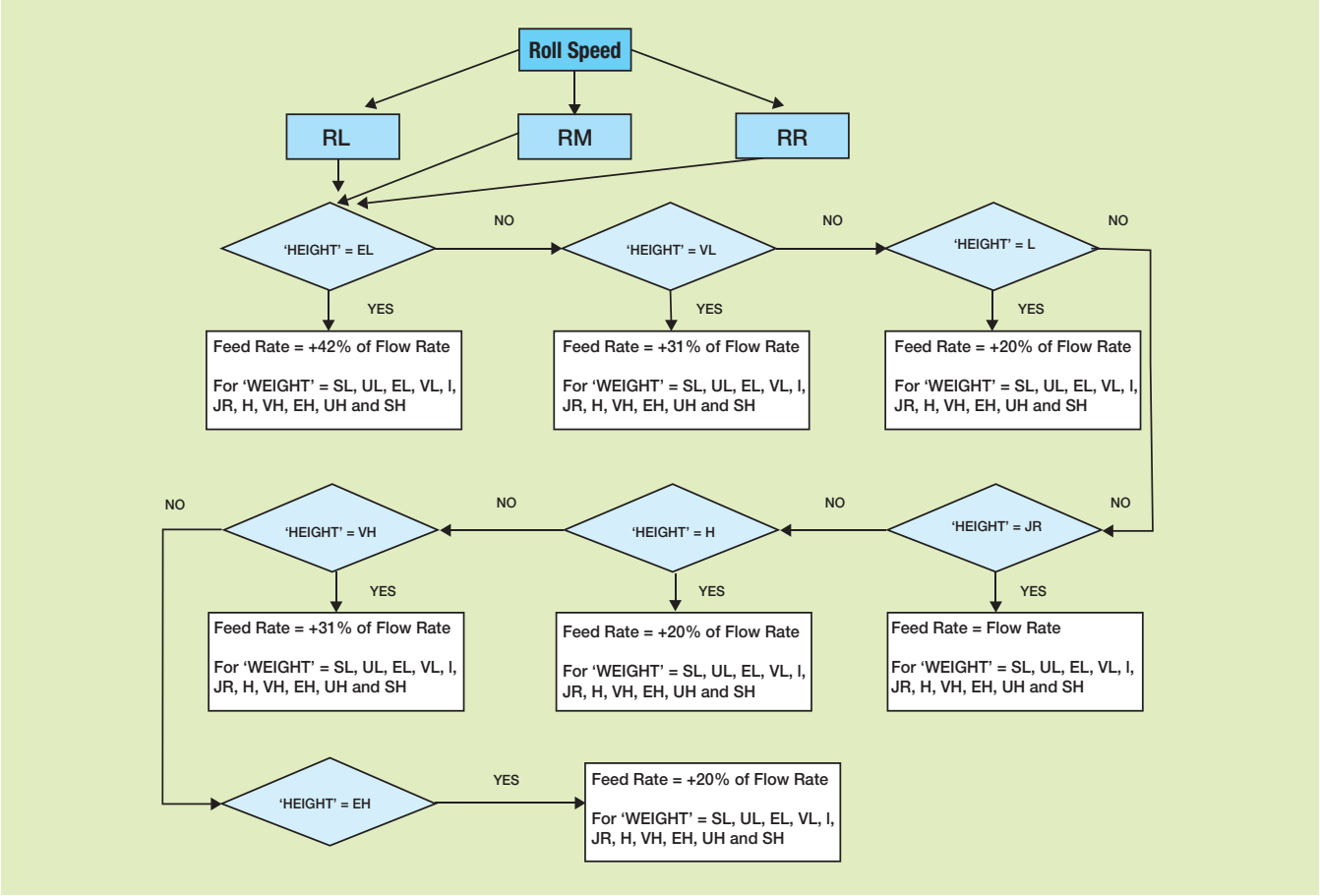


図 2 – 3 入力ファジー コントローラーの開発アルゴリズム

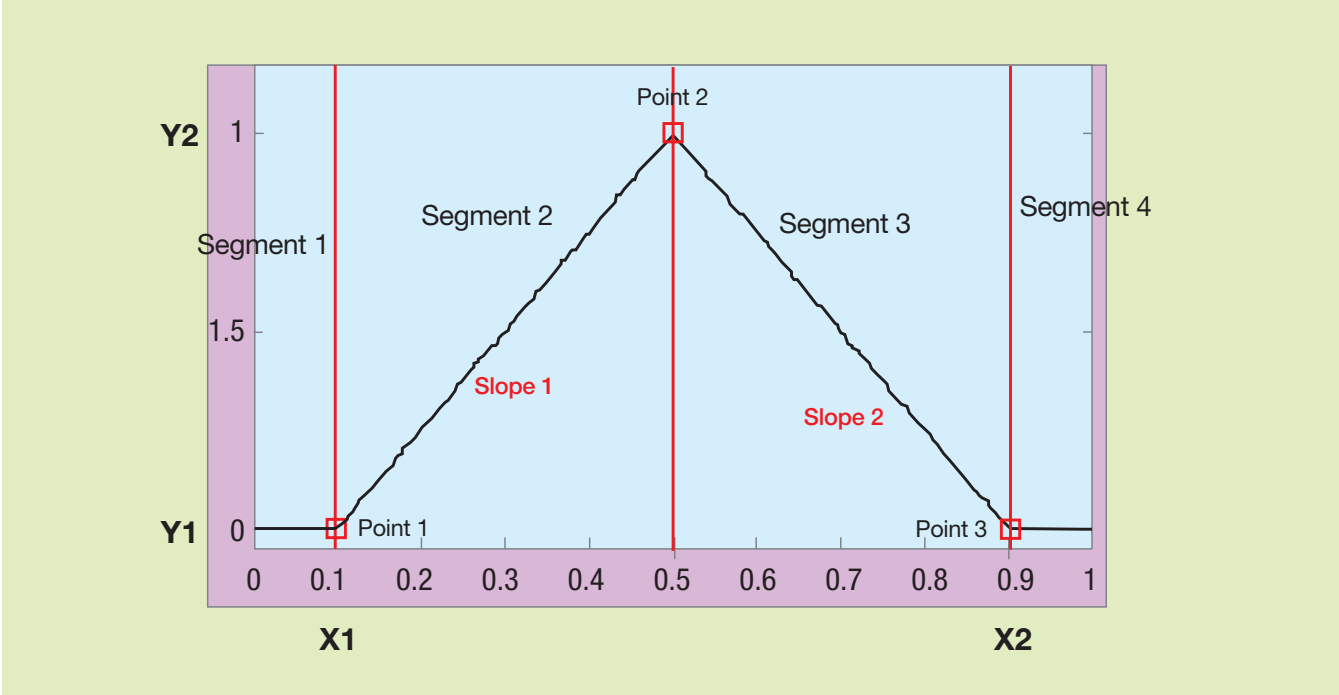


図 3 – 3 つのポイントと 2 つの勾配で定義されたメンバーシップ関数

イント 2)* 勾配 2}、Segment4 (μ=0) です。DOM の値は、次のように計算されます。

- 入力値 < ポイント 1 の場合 (Segment 1)、DOM =0
- 入力値 ≤ ポイント 2 かつ≥ ポイント 1 の場合 (Segment 2)、DOM = (入力値 - ポイント 1) * 勾配 1
- 入力値 ≤ ポイント 3 かつ ≥ ポイント 1 の場合 (Segment 3)、DOM = FF- (入力値 -ポイント 2) * 勾配 2
- 入力値 ≥ ポイント 3 の場合 (Segment 4)、DOM = 0

異なるメンバーシップ度

次のステップは、メンバーシップ関数の度合いに応じてアクションを特定するためのルールの設計でした。シンプルな If-Then 条件を使用して、すべての前件部が後件部を持つファジー ルールを構成しました。MATLAB の Fuzzy Logic Toolbox には、複数の前件部を結びつけるためのさまざまな演算子があります。複数の前件部間の最小動作を表すため、AND 演算子で 3 つの前件部を結ぶことにしました。3 入力コントローラーについては、合計 231 個のルールを生成しました。これらのルールについて、ルール マトリックスを設計しました。最小関数は、3 つの値のうちの最小を見つけます。つまり、3 つの入力変数間の DOM の最小値が計算されます。

また、多くのルールの後件部が同じになることも分かりました。同じ後件部を持つルールはすべてまとめて、最大関数を使用してこれらの値の最大値を計算しました。その次のステップでは、同じ後件部を持つすべてのルールをまとめました。LV 全体の最大値を評価するため、複数の最大関数をエンコードしました。

各ルールの出力を特定した後の最後のステップは、すべての出力を 1 つの値にすることです。別の言い方をすれば、切りの良い数値に変換することです。これは、非ファジー化で行います。

非ファジー化は、ファジー システム設計の最後の重要なステップです。非ファジー化された値から、1 つの切りの良い数値 (すなわち、レーク モーターの速度) が生成されます。今回使用した Sugeno 非ファジー化手法は加重平均法で、集計によって得られたファジー出力に、対応するシングルトン値を掛け、それらの値の合計を、ルール評価によって得られたすべてのファジー出力の合計で割りました。つまり、集計によって値を求めました。

Virtex-6 インプリメンテーション

上記のステップを行った後、三角形のメンバーシップ関数と重心法による非ファジー化を使用して、3 入力ファジー コントローラーを設計できました。プログラム コード

Parameters		Cane level (cm)	Cane weight (kg)	Motor speed (rpm)	Carrier speed (cm/s)	Cane in carrier (kg/cm)	Feed rate (kg/s)	Data for next sampling		Cane level *(VHDL)	Cane (MATLAB) ** (cm)
Time (sec)	Roll speed (cm/s)							kg	cm		
0	15.4	90.0	750	47.0	24.6	0.938	23.1	-16.0	-6.4	83.6	85.7
10	15.8	83.6	729	52.0	27.2	0.911	24.8	-5.0	-2.0	81.6	84.3
20	15.0	81.6	792	50.0	26.2	0.990	25.9	+19.0	+7.6	89.2	90.4
30	16.2	89.2	908	42.0	22.0	1.135	25.0	-9.0	-3.6	85.6	86.5
40	16.6	85.6	965	44.0	23.0	1.206	27.7	+11.0	+3.9	89.5	90.5
50	13.4	89.5	720	49.0	25.7	0.900	23.1	+16.0	+6.4	95.9	95.9
60	13.8	95.9	760	39.0	20.4	0.950	19.4	-27.0	-9.6	86.3	86.3
70	13.4	86.3	790	44.0	23.0	0.988	22.7	+12.0	+4.8	91.1	91.3
80	15.4	91.1	820	46.0	24.1	1.025	24.7	0.0	0.0	91.1	93.4
90	16.2	91.1	555	73.0	38.2	0.694	26.5	-4.0	-1.6	89.5	93.4
100	13.0	89.5	609	51.0	26.7	0.761	20.3	-5.0	-2.0	87.5	92.3
110	14.3	87.5	578	62.0	32.5	0.723	23.5	+6.0	+2.4	89.9	90.2
120	14.6	89.9	598	57.0	29.8	0.748	22.3	-11.0	-4.4	85.5	87.0
130	12.3	85.5	700	44.0	23.0	0.875	20.1	+4.0	+1.6	87.1	88.8
140	12.6	87.1	679	48.0	25.1	0.849	21.3	+11.0	+4.4	91.5	91.7
150	15.4	91.5	800	46.0	24.1	1.000	24.1	-6.0	-2.4	89.1	91.3
160	12.0	89.1	845	32.0	16.8	1.056	17.7	-15.0	-6.0	83.1	84.2
170	14.3	83.1	835	45.0	23.6	1.044	24.6	+17.0	+6.1	89.2	90.3
180	14.6	89.2	874	42.0	22.0	1.093	24.0	+6.0	+2.4	91.6	92.1
190	15.0	91.6	900	41.0	21.5	1.125	24.2	+2.0	+0.8	92.4	92.1
200	15.4	92.4	924	40.0	20.9	1.155	24.1	-6.0	-2.4	90.0	91.4

* Cane level of FPGA-implemented system after each sampling ** Cane level of MATLAB-implemented system after each sampling

表 1 - サトウキビの長さが 90cm、ロール速度はサンプルごとに変化

は、著者から入手できます。MATLAB の Fuzzy Toolbox バージョン 7.11.0.584 (R2020b) を使用して 3 入力ファジー コントローラーのシミュレーションを行い、ザイリンクスの ISE® Design Suite 14.5 を使用して、VHDL でザイリンクス Virtex-6 FPGA に実装しました。サンプリング周期は 10 秒、合計シミュレーション時間は 200 秒でした。

6 つの異なるケースにおける計 756 種の入力パラメーター条件を調査しましたが、ここでは、シミュレーションの初期段階において、サトウキビの長さが 90cm、キャリア上の重さが 750kg であるケースに注目します。ロール速度は、サンプリングごとに変化させます。このシミュレーション結果を表 1 に示します。

ハードウェア インプリメンテーションの工程は、VHDL モデリング、シミュレーション、合成、FPGA インプリメンテーションで構成され、MITS キャンパスの私たちの研究室で行いました。今回、3 入力ファジー コントローラーの VHDL モデルを設計するにあたり、ビヘイビアー モデリング、構造モデリングなど、いくつかのタイプのモデリングを合わせて使用しました。シミュレーションは、ザイリンクスの ISim シミュレータで行いました。ISim で生成された波形は、コントローラーの機能を裏付けるものでした。図 4 は、レーク キャリア上のサトウキビの重さが 750kg、Donnelly シュート内のサトウキビの長さが 90cm、ロール速度が 16.6 cm/秒の場合のシミュレーション波形を示しています。このような条件下では、推定レーク モーター速度は 54.2rpm (MATLAB) です。非ファジー化したシミュレーション結果は 36H、つまり、54rpm となり、これは MATLAB 結果と一致し、デザインを証明するものです。

シミュレーション後、デザインを合成し、テクノロジ回路図を生成し、概算デバイス使用率レポートを生成しました。今回のデザインでは、Virtex-6 のスライス ルックアップ テーブル (LUT) の 78% 以上、占有スライスの 93%、スライス レジスタの 1%、LUT フリップフロップの 1% が使用されていることが分かりました。

その後、VHDL 結果を従来のコントローラーの結果と比較しました (表 2 を参照)。この比較により、ファジー -ロジック システムの方が従来のコントローラーよりも効率が良いのが分かります。

MITS の研究室には、研究用に Spartan®-6 FPGA があります。しかしながら、必要な LUT ブロック数がターゲット デバイスの容量を超えていたため、Virtex-6 を選択しました。リソース ファイルがなく、研究室でのリアルタイム インプリメンテーションは行えませんでした。

次のステップとして、インドの National Sugar Institute と連携して、システム全体を開発し、実環境で結果を確認したいと考えています。すでにプレゼンテーションを National Sugar Institute に送り、好反応をいただいています。ファジー ロジックの概念は、砂糖業界の未来を変えると信じています。

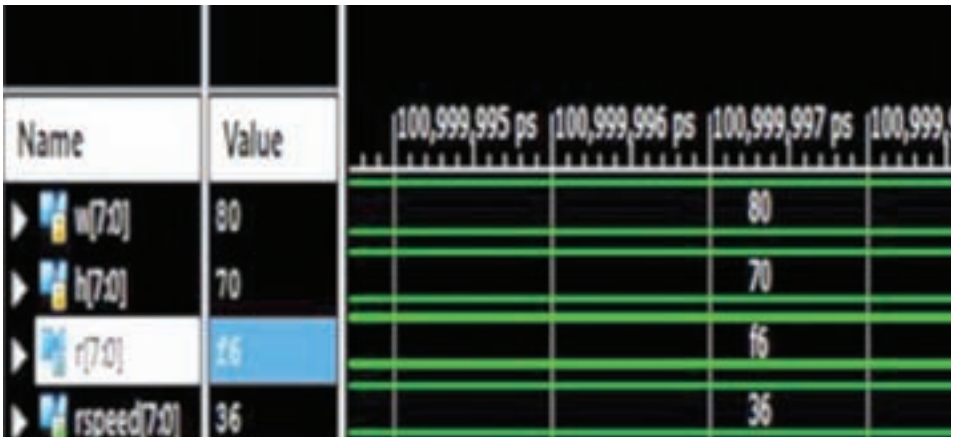


図 4 – 重さが 750 kg、長さが 90 cm、ロール速度が 16.6 cm/s の場合のシミュレーション波形

	Three-input conventional controller	Three-input fuzzy controller (MATLAB)	Three-input fuzzy controller (Xilinx VHDL)
Percentage of time cane is in between 85 cm-95 cm (% Time)	45.8	94.7	88.0
Lowest level of cane in chute (cm)	61.7	84.2	81.6
Highest level of cane in chute (cm)	103.5	95.9	95.9

表 2 – 結果の比較

参考文献

1. Y. Misra and H.R. Kamath, “Design Algorithm of Conventional and Fuzzy Controller for Maintaining the Cane Level During Sugar Making Process,” *International Journal of Intelligent Systems and Applications*, ISSN: 2074-9058,Vol.7 No.1, December 2014
2. Y. Misra and H.R. Kamath, “Implementation and Performance Analysis of a Three Inputs Conventional Controller to Maintain the Cane Level During Cane Crushing in FPGA using VHDL,” *International Journal of Engineering Research & Technology* (IJERT), ISSN: 2278-0181, Vol. 3 Issue 9, September 2014
3. Y. Misra and H.R. Kamath, “Analysis and design of a three input fuzzy system for maintaining the cane level during sugar manufacturing,” *Journal of Automation and Control* (accepted), ISSN: 2372-3041

Zynq MPSoC Gets Xen Hypervisor Support

Zynq MPSoC と Xen ハイパーバイザーのサポート

Steven H. VanderLeest
Chief Operating Officer
DornerWorks, Ltd.
Steve.VanderLeest@DornerWorks.com

**ザイリンクスの
最新Zynq デバイスにより、
Xen ハイパーバイザーが
パワーアップします。
しかし、このオープンソースの
仮想化アプローチを選ぶには、
サポートが鍵となります。**

オープンソースのハイパーバイザーの Xen は、多機能な仮想化テクノロジです。これは以前からクラウド コンピューティングに使用されていましたが、最近ではエンベデッド システムにも導入されるようになりました。DornerWorks 社は、新しい Zynq® UltraScale+ MPSoC デバイス上で Xen サポートを提供しています。これは、ザイリンクス ユーザーにとって複数のメリットがあります。Xen Zynq ハイパーバイザーにより、ソフトウェアの統合が高速化し、システムの安全性が高まりセキュリティが向上するのみならず、エンタープライズ向けクラウド コンピューティングのメリットをエンベデッド システムで得られるようになります。

ハイパーバイザーは、設計を厳密にコンパートメント化することができるため、同一のコンピューティング デバイス上に、(OS 全体を含む) 新しいソフトウェアを迅速に統合できるようになります。同時に、この分離によって、各ソフトウェア機能間の予期せぬ干渉が低減もしくは解消されます。

さらに、分離により、機能間の予期せぬ相互作用が減り、脅威にさらされる攻撃対象領域が小さくなることによって、システムの安全性とセキュリティが大幅に向上し、安全性またはセキュリティの特性をより容易に実証できるようになります。また、ソフトウェアの変更を (ほとんど) せずに、最新のハードウェア上に旧型のソフトウェアを展開できるなど、エンタープライズ向けクラウド コンピューティングのメリットの多くがエンベデッド システム上でも活用できるようになります。

Zynq MPSoC 上で動作するオープンソース Xen ハイパーバイザー、Xen Zynq の詳細に入る前に、ハイパーバイザーについて簡単に説明します。

ハイパーバイザーとは

ハイパーバイザーは、仮想化を可能にする基本的なソフトウェア層です。OS が同時に実行する複数のアプリケーションを (OS が管理するマシン リソースに対してアクセス可能なプロセスにアプリケーションを分割して割り当てることで) 管理するのと同じように、ハイパーバイザーも、同時に実行される複数の OS を (ハイパーバイザーが管理するマシン リソースに対してアクセス可能な仮想マシンに分割することで) 管理します。

仮想化のアイデアは、1960 年代にさかのぼります。1974 年、Popek と Goldberg が、次の 3 つの定義的特性を持つ仮想マシン モニター (VMM) のアイデアを正式に発表しました。

- VMM は、オリジナル マシン (物理マシン) と本質的に同じランタイム環境 (仮想) をプログラムに提供する。
- VMM がパフォーマンスに与える影響は無視できるほど小さい。
- VMM は、システムのリソースを管理する。

ハイパーバイザーは、主として基本的なマシン管理タスクに特化した VMM です。つまり、ファイル システム、GUI、ネットワーク プロトコル スタックなどの一般的に搭載されるはずのサービスをこの層では実装せずに、上位層 (ハイパーバイザーがホストする仮想化マシン内で実行されるゲスト OS など) に任せています。

上述したようなハードウェア上でネイティブに動作するハイパーバイザーを、Type 1 ハイパーバイザーと呼びます。一方、Type 2 ハイパーバイザーは、ソフトウェア レイヤーの最下層ではなく、OS 上にホストされます。この種のハイパーバイザーは、一般的に、ある OS から別の OS を動作させる際に使用されます。たとえば、Mac ユーザーが MacBook 上で Parallels を使用して Windows を実行するときや、Windows ユーザーが VirtualBox を使用して仮想マシン内で Linux を起動するときなどです。

エンタープライズ向けハイパーバイザーとエンベデッド向けハイパーバイザーにも、重要な違いがあります。エンタープライズ向けハイパーバイザーの典型的な使用事例は、クラウド コンピューティングとビッグ データです。エンベデッド業界においてハイパーバイザーが使われるようになったのはごく最近で、十分な性能と許容できる消費電力を兼ね備えたプロセッサの登場とともに導入が始まっています。

エンベデッド向けハイパーバイザーの採用事例には、「複数の複雑な機能を、ある程度の分離を保ちながら、単一のコンピューティング プラットフォームに統合する」という共通のテーマがあります。航空宇宙アプリケーションでは、ハイパーバイザーは、統合されたモジュール型アビオニクスをサポートするためによく使用されます。従来、連結された (独立した) アビオニクス ハードウェア上で実行されていたソフトウェアが、単一のコンピューティング プラットフォームに統合されます。その機能には、操縦系統、ナビゲーション、飛行管理システム、アビオニクス衝突防止などがあります。米国連邦航空局 (FAA) では、従来、別々のハードウェア上で実行されていたソフトウェアの機能を組み合わせた場合に、それぞれが互いに影響を与えないことを要求しています。この分離は、DO-248C などの規格に規定された厳しいパーティショニングで可能です。

FAA の要求は、機能の統合による民間旅客機の安全性の確保を目的としたものですが、軍事用アビオニクスでは、セキュリティを確保するため、分離へのニーズは倍増します。1 つのシステム上で、厳密に分けられた複数のレベルをサポートするアプローチには、MILS (Multiple Independent Levels of Security) と呼ばれるアーキテクチャが使用されます。

医療機器アプリケーションでは、ハイパーバイザーを使用して、MRI スキャナー、手術用ロボット (またはロボット支援手術装置)、CT スキャン マシンなどのハイエンド医療機器 (これらすべてに、現在、複数の独立したプロセッシング システムが組み込まれている) に同様な統合が見込まれています。統合の対象となる機能としては、医師用グラフィカル ユーザー インターフェイス、画像処理、リアルタイム モーター制御、患者情報データベース、システム管理機能などが挙げられます。

車載機器アプリケーションにおいて、ハイパーバイザーは、車両に組み込まれている多数の独立したマイクロプロセッサやマイクロコントローラーを統合する、魅力的な手法です。ほとんどすべての車載 OEM 製品において、ハイパーバイザーに移行して、情報および娯楽サービス、運転者および同乗者の操作機器、先進的な運転支援システム (ADAS)、計器類、カー ナビゲーション システム、インターネット接続、そしていずれはリアルタイム制御などの機能を統合する方向性が検討されています。

仮想化ソリューションを検討する際、「性能への影響が無視できるほど小さい」という VMM の特性について評価することは重要です。ハイパーバイザーは、すべてのハードウェア リソース (CPU、メモリ、および I/O) をコントロールするため、そのいずれかの性能に影響を与えることがあります。CPU の場合、重要なメトリクスの 1 つは、ある仮想マシンを実行しているコアを、別の仮想マシンの実行に使用するために切り替える際にかかる時間です。これを「コンテキスト スイッチ時間」と呼ぶことがありますが、OS が行うプロセス間の切り替えについての類似の概念と区別するために、「パーティション スイッチ時間」または「ドメイン スイッチ時間」と呼ぶこともあります。関連するメトリクスはジッターです。ジッターはスイッチ時間のばらつきの尺度で、決定性や予測性に影響を与えます。

リアルタイム設計をする場合、スケジュール可能な最小タイム スライスの計測値も重要な情報です。これによって、CPU スケジュールの最高周波数、言い換えれば、ある周期内に実行可能な仮想マシンの最大数が制約されます。メモリへの影響を測定する際、ハイパーバイザー カーネルのフットプリントは、一定のベース部分と、ゲスト (仮想マシン) が 1 台加わるごとに増加するインクリメンタル部分から構成されます。仮想マシンの最大数は、累積的なフットプリントによって制約されます。I/O について、各対象デバイスで測定すべき主要測定項目は、帯域幅とレイテンシです。ただし、全体の割り込みレイテンシや RAW 通信帯域幅などの一般的なメトリクスから見積もりを立てることもできます。

多くのハイパーバイザーは、I/O について、排他と共有の 2 つのアプローチをサポートしています。排他的 I/O の場合、ハイパーバイザーは 1 台の仮想マシンに特定の I/O デバイス (一般に「パススルー」デバイスと呼ぶ) への直接かつ単独のアクセスを提供するため、多くの場合、オーバーヘッドが低下します。共有 I/O の場合、ハイパーバイザーは共有スキームを実現するためのメカニズムを実行する必要があるので、オーバーヘッドが増加します。

オープンソースの側面

「オープンソース」という用語は、「フリー」なソフトウェアを指しますが、これは、「自由」なソフトウェアという意味であって、必ずしも「無償」のソフトウェアを意味するものではありません。オープンソース ソフトウェアは、その自由が保たれるように注意深く作成されたライセンスの下で、ソース コードを自由に変更して共有することができます。特に有名なオープンソース ライセンス契約には、GNU General Public License (現行バージョンは GPLv2 と GPLv3)、GNU Lesser General Public License、Apache License、BSD ライセンス (複数の種類) などがあります。

オープンソースは、必ずしも無償ではありません。オープンソース製品をベースにして構築されたビジネスは、通常、従来のソフトウェア ベンダーとは異なる種類の収益モデルを持ち、製品サポート、付属品 (印刷されたユーザー マニュアルなど)、研修、カスタマイズ 設計サービスなどを販売しています。最も有名な例の 1 つは Red Hat 社で、オープンソースの Linux OS をベースに 10 億ドル単位のビジネスを築いています。

Xen の 新しい Zynq へのマッピング

ザイリンクスの新しい Zynq UltraScale+ MPSoC は、Xen ハイパーバイザーを実行するパワフルなプラットフォームです。本デバイスは、ARMv8 命令セットにハードウェア仮想化拡張機能と 64 ビット 機能が搭載されたクワッドコア ARM® Cortex™-A53 です。パワフルなハードウェアの機能と性能を最大限引き出すために、豊かなソフトウェア エコシステムが必要です。ザイリンクスは新しい Zynq MPSoC を開発中に、航空宇宙業界、軍事業界、医療業界、電気通信業界、オートモーティブ業界を含むさまざまな業界の主要顧客にアンケート調査を行いました。その結果、ほとんどの顧客が新しいデバイスにハイパーバイザー オプションを希望しており、その半数がオープンソースのハイパーバイザーを望んでいました。そこでザイリンクスは、Xen をオープンソース ハイパーバイザーとして選択しました。また、DornerWorks は新製品 Xen Zynq のサポート サービス提供企業として選ばれました。

Xen ハイパーバイザーは、仮想マシン内にゲスト OS をホストし、下位のマシンの仮想化ビューを提供します。ゲスト OS とそのアプリケーションは、仮想化された CPU、メモリ、および I/O を使用し、Xen は、それらの仮想化リソースと物理リソースのマッピングを管理します。

Xen では、各仮想マシンを「ドメイン」と呼びます。ハイパーバイザー カーネルをできる限り小さくするため、Xen はある 1 つのドメインに特別な権限を与えています。このシステム ドメインを dom0 と呼びます。dom0 は、

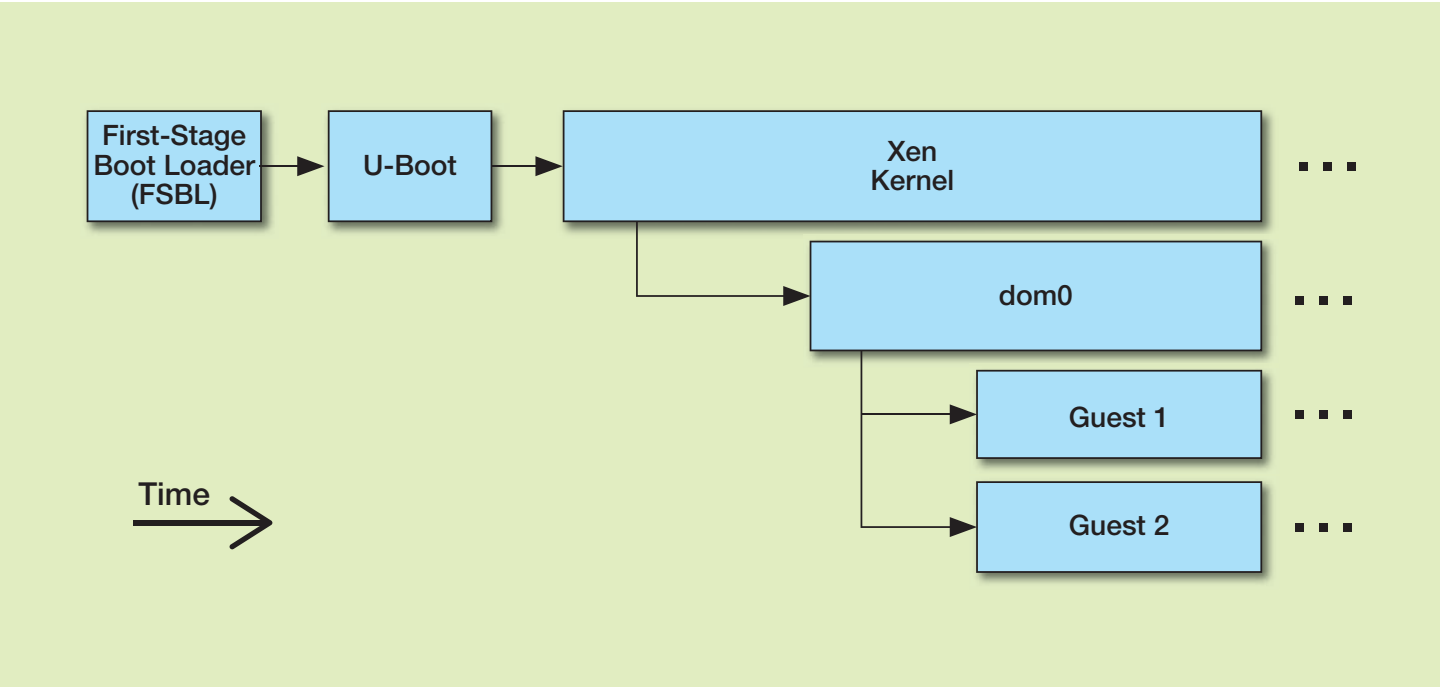


図 1 – ゲスト OS が実行されるまでの段階を示す一般的な起動シーケンス

他のゲスト ドメイン (domU と呼ぶ) を起動したり、カーネルのスケジュール設定やメモリ マップ設定を行ったり、I/O アクセス権限を管理したりします。もう少し詳しく説明するため、起動シーケンス、ARM 例外レベル、実行スケジュール、そしてリソース管理など、ハイパーバイザー環境の側面を考えてみましょう。

電源をオンにした時点から始まる Zynq MPSoC の起動シーケンスは、どちらのプロセッサが先に起動するか (Cortex-A53 または Cortex-R5) を含めて、さまざまな方法で構成できます。ほとんどの採用事例では、2 つのプロセッサの独立性がかなり維持されるため、標準的な Xen Zynq ハイパーバイザー ディストリビューションは Cortex-A53 上のみで実行されます。一般的な起動シーケンスを図 1 に示します。Cortex-R5 で独立した非仮想のセキュア OS をホスティングしている場合、通常、FSBL (第 1 段階ブート ローダー) によって R5 が最初に起動されます。R5 が起動すると、R5 は A53 (自身の FSBL を使用して起動) を起動します。U-Boot などの第 2 段階ブート ローダーは、通常、ハイパーバイザー カーネル イメージの整合性チェックなどの、幅広い起動機能を提供するために使用されます。

この段階で、Xen ハイパーバイザー カーネルが起動します。カーネルの起動中、有効な dom0 が存在するかチェックされます。dom0 はゲスト ドメイン用に有効なイメージがあるかどうかをチェックした後、ゲスト ドメインを起動し、1 つまたは複数のコアにスケジュールします。多くの場合、dom0 は継続実行されて、システムをモニタリングしたり、共有リソースの管理を行ったり、特定のシステム故障に対応したりします。ハイパーバイザー カーネルは、各ドメイン コンテキスト スイッチ中に実行されます。また、ハイパーコールによって起動されることもあります。ハイパーコールは、アプリケーションが OS サービスを呼び出すことができるシステム コールと類似していますが、この場合、ハイパーバイザー サービスを呼び出します。デフォルトでは、dom0 は任意のハイパーバイザーコールを発行できますが、domU には制限があります。開発者は Xen モジュール XSM-FLASK を使用することで、より詳細なハイパーコール アクセス制御をインプリメントできます。

プロセッサ ハードウェアは、ARM 例外レベル モデルで定義される各カテゴリの権限に従って動作します。Cortex-A53 が使用する ARMv8 アーキテクチャには、図 2 に示すように、4 つの例外レベルが定義されています。この図の最も下の層が最も高い権限を持ち、1 層上がるごとに権限が減少していきます。例外レベル EL3 には完全なアクセス権を与えられ、ARM TrustZone Monitor に使用されます。ハイパーバイザーは EL2 で実行され、ゲスト ドメインの仮想化を行います。ホストされた各仮想マシン内において、ホストされた OS が EL1 で実行されます。最後に、ユーザー アプリケーションは、EL0 において最も小さい権限で動作します。低い権限を持つ例外レベルに変更するとき、仮想化マシンのレジスタは、ビット数が自身と同じか自身より小さくなければなりません。つまり、64 ビット ハイパーバイザー と 32 ビットのゲストは可能ですが、その逆は不可能です。Xen Zynq は、ARMv8 アーキテクチャの AArch64 例外モードを使用して、64 ビットまたは 32 ビットのゲストをサポートします。

特権ドメインの dom0 がスケジュールを設定し、ドメインがいつ、どのコア上で実行されるかを決定します。その後、ハイパーバイザー カーネルが設定されたスケジュールを実行します。特定の種類の決定した論理を実現するため、ゲスト ドメインがあるタイム スロット中、マシンに単独のアクセス権を持つようなスケジュールを設定することもできます。図 3 に例を示します。ゲスト 1 が (dom0 とともに) 複数のコア上で 1 タイムスロットの間実行されていますが、ゲスト 2 と 3 はこの制限が不要なため、他のタイムスロットではより幅広い組み合わせのロード バランシング スキームでスケジュールが可能です。

ハイパーバイザーは、マシン上のすべてのリソースを管理します。CPU コアは、上述

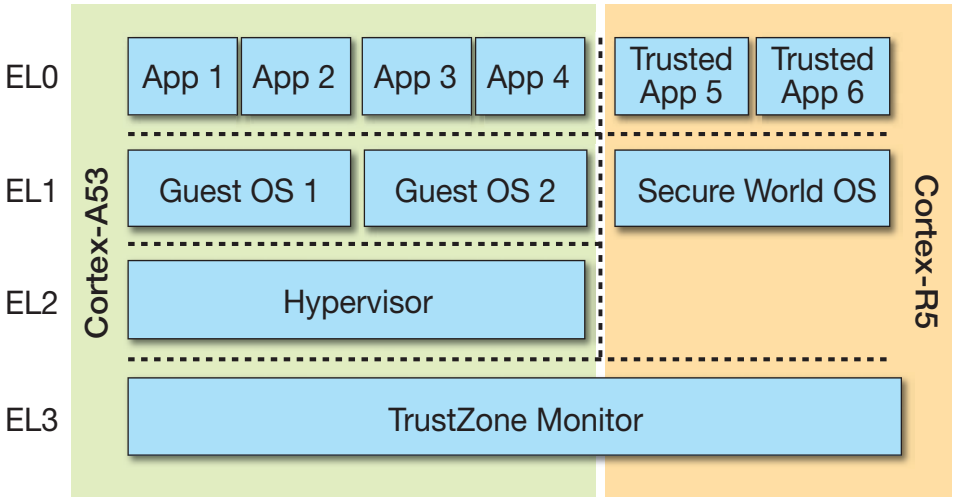


図 2 – ARM 例外レベルの図。ハイパーバイザーは EL2 にマッピングされている。

したように、主に時分割方式で管理されます。ハイパーバイザーはハードウェア タイマーを使用してスケジュールを実行します。メモリは、時分割ではなく、空間を分割してメモリの一部を各ゲスト ドメインに割り当てることによって共有されます。ハイパーバイザーは、ハードウェア メモリ管理装置 (MMU) を使用して、メモリ レイアウトを管理します。I/O の管理には、デバイスのタイプに応じてさまざまな方法があります。Cortex-A53 に直接マッピングされる I/O デバイスや、FPGA プログラマブル ファブリックを介して接続する必要がある I/O デバイスなどがあります。

I/O デバイスへのゲスト アクセスは、dom0 により設定および管理され、適切なハイパーコールが Xen カーネルに発行されてデバイスへのメモリ マップが確立されます。dom0 は、必要に応じてゲスト ドメイン アクセスを特定の I/O デバイスに付与することも、共有メカニズムを実施するゲートウェイとして、dom0 が共有 I/O を管理することもできます。Xen のドメイン間通信 (I/O を含む) では、一般的に、Xen イベント チャネルはデータを渡すための通知と共有メモリに使用されます。Xen の共有 I/O デバイス ドライバーには、スプリット ドライバー モデルが使用されています。このモデルでは、ゲスト ドメインの上半分がゲスト OS に API を提供し、dom0 とのデータ転送機能を実行します。dom0 の下半分のドライバーがデバイスとの実際の I/O 処理を行います。

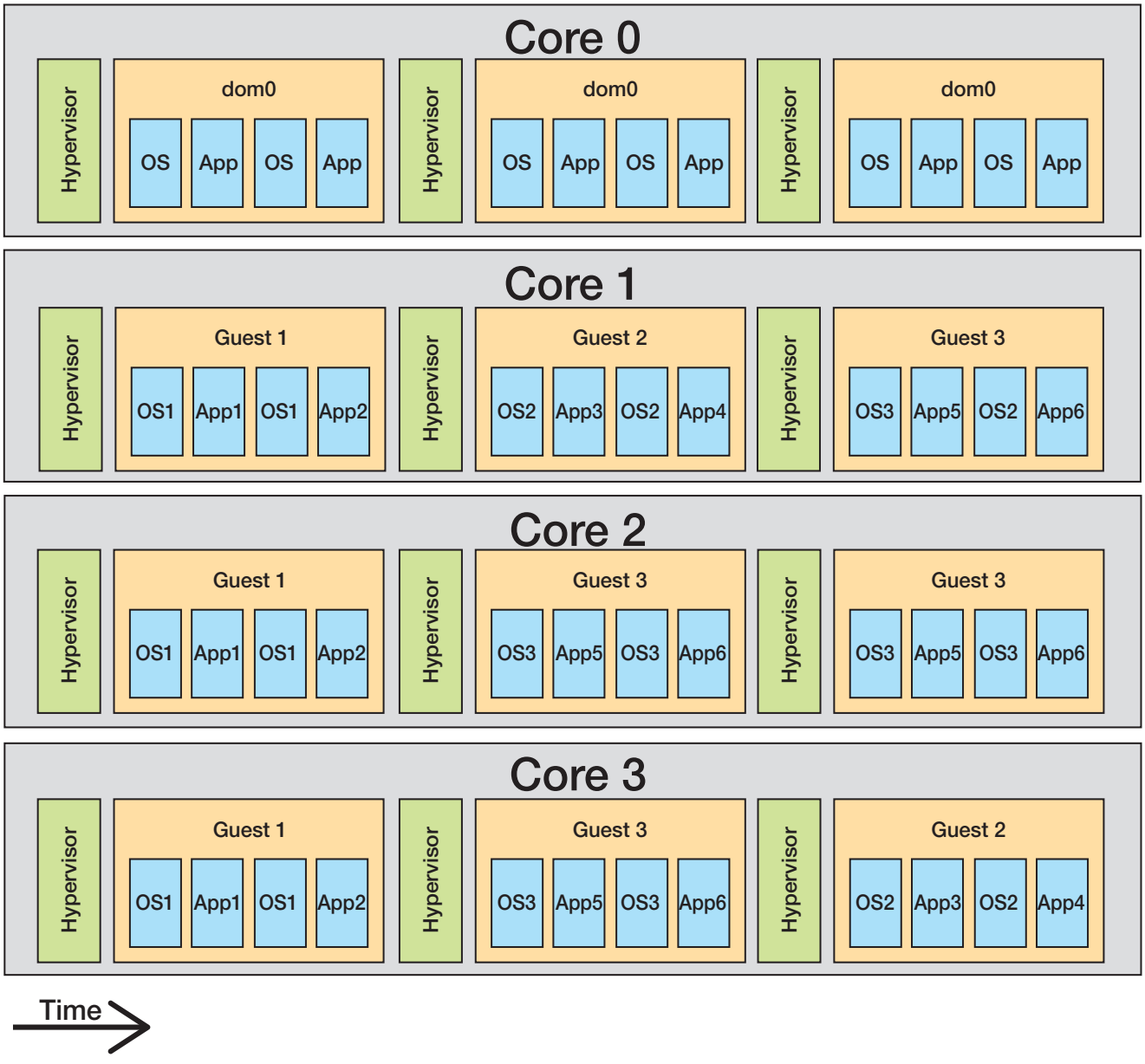


図 3 – マルチコア スケジューリングで、ゲスト 1 を排他的タイム スロットに入れ、
ゲスト 2 および 3 は混在

Xen Zynq のサポート

ザイリンクスが次世代の Zynq SoC デバイスに期待をよせる顧客の声を調査したとき、多くの方がハイパーバイザーに関する手厚いサポートを希望しており、そのうちの半数がオープンソース オプションを望んでいることが分かりました。このサポートは、単純なヘルプデスク方式のサービス以上のものでなければなりません。サポート オプションは、厳しい要件 (高帯域幅、低レイテンシ、低消費電力、高信頼性など) のバランスを取り、エンベデッド環境の多種多様なシステム デバイスをつなぐエンベデッド システムを設計するための支援を提供できる、より幅の広いものでなければなりません。ザイリンクスが DornerWorks を選んだ理由は、Xen ハイパーバイザーに関する経験、エンベデッド エンジニアリング設計の経験があり、ザイリンクス アライアンス プログラムのプレミア メンバーであることに加え、システムの FPGA デザイン部分についてのサポートを求めるお客さまに追加のオプションを提供できるからです。

DornerWorks は、ザイリンクスと共同で Xen の新しい Zynq MPSoC への移植を完了し、検証および動作確認テストを行って、妥当性を確認しました。当社の検証作業では、Xen ハイパーバイザー カーネルがハードウェア上で正常に実行できるかの検証のほか、特権ドメイン dom0 (Linux を実行) とゲスト ドメインについて、多種多様なサポート対象ゲスト オペレーティング システムでの検証を行いました。こうして出来上がったソフトウェア パッケージは、Xen Zynq ディストリビューションと名付けられました。

実際のハードウェアが完成する前に、さらなる検証課題が待ち受けていました。当社が使用したハードウェア代用モデルは、QEMU オープンソース マシン エミュレーター ソフトウェアで、個々のデバッグおよびテストは x86 開発者システム上で、継続的インテグレーション テストはチームのビルド サーバーで実行しました。さらに、開発は、Zynq MPSoC を 6 つのザイリンクス Virtex®-7 FPGA でエミュレートする Remus という名のエミュレーション ボード (同名の Xen ライブ移行ツールとは別物です) に対して行いました。

図 4 は、ビルド/テスト サーバーを中心にした継続的インテグレーション アプローチを示しています。サーバーは、ソース コードのリポジトリに定期的にクエリを発行します。何か変更が検出されると、サーバーは、ビルド イメージの依存部分について、インクリメンタル ビルドを実行します。その後、ターゲット ファームの各デバイスに、各テストに必要なイメージをロードし、テスト スクリプトを呼び出します。一部のテスト ケースでは、ターゲットに外部スティミュラスが与えられます。テスト サーバーは結果を収集および照合し、テスト スイートの全体的な状態を示すサマリ ダッシュボードを表示するか、解決すべき問題点を指摘します。

また、DornerWorks では、新しい Zynq MPSoC 上で Xen ハイパーバイザーを使用するザイリンクスの顧客に総合的なサポートを提供するためのインフラストラクチャを開発しました。基本的なサポートは、ユーザーがコメントを比較し、情報共有することができる

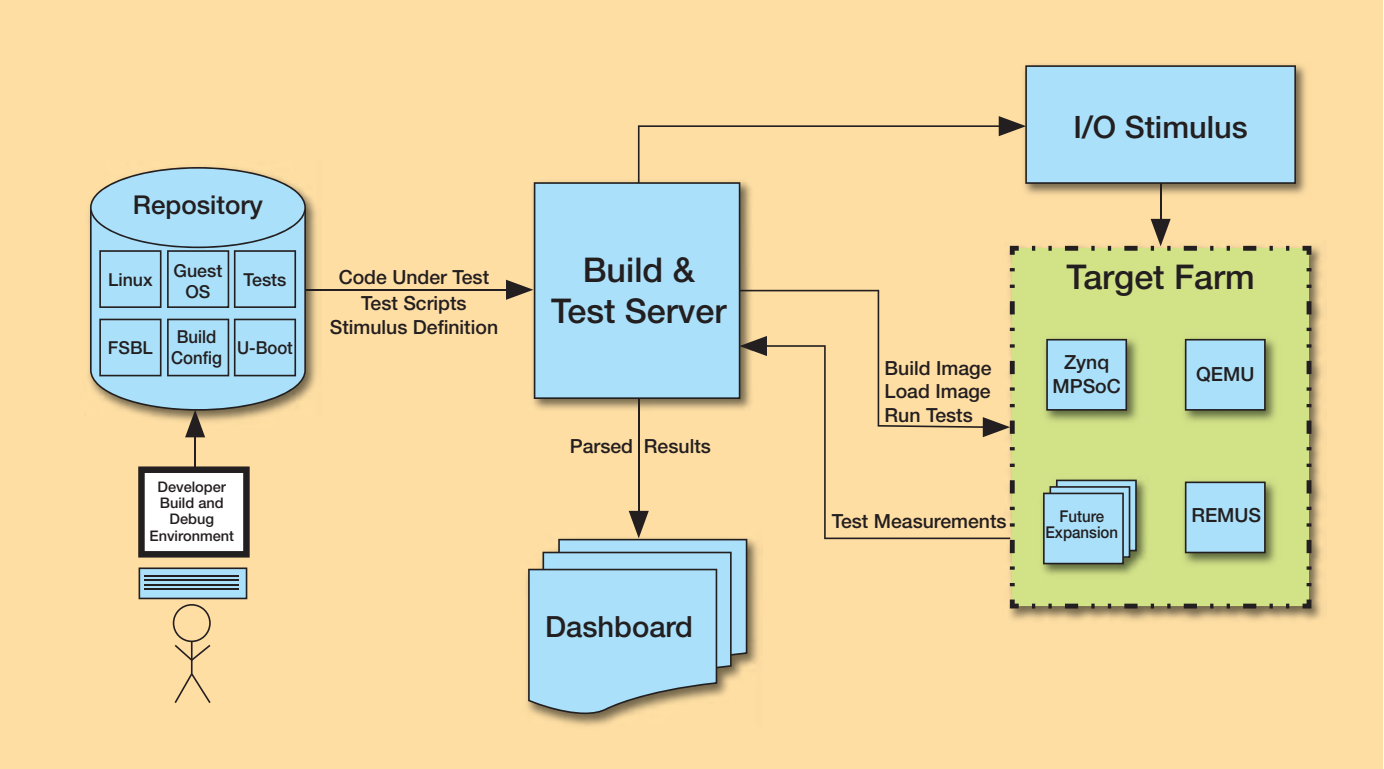


図 4 – 継続的インテグレーションにより Xen Zynq のビルドとテストを自動化

オープンソース コミュニティの活動をベースとします。DornerWorks は、フォーラムをホストし、コミュニティの問題を収集します。ザイリンクスが示す問題、DornerWorks 社内で発見された問題、顧客 (コミュニティまたは有料購読者) が見つけた問題をトラッキングするツールとして、Jira を使用しています。DornerWorks では、Xen に関する取り組みを支えるため、ビジネス リスクを下げ、ニーズに迅速に対応できるよう、多くのお客さまが希望するビジネス クリティカルなサポートを契約によって提供する、有料購読サービスとカスタム デザイン サポート サービスを提供しています。サポート オプションの詳細は、http://xen.world をご覧ください。

Xen 試用の勧め

新しい Zynq MPSoC は、来年初めに出荷を予定されていますが、それを待つ間にも、Xen の調査を始めることができます。Xen は、標準の x86 PC で実行できます。Type 1 ハイパーバイザーとしてネイティブに使用することも、開発用に Windows 上の VirtualBox 上にホストすることも可能です。エンベデッド Xen を試用するには、実際の ARM ハードウェアまたはエミュレーターが必要です。仮想化拡張機能を持つ ARM プロセッサを選択します (理想は Cortex-A53 ですが、Cortex-A15 など、他のプロセッサでもほぼ類似の環境を作成できます)。エンベデッド ターゲット向けに完全なハイパーバイザー ベースのシステムを構築するためのワークフローを図 5 に示します。Xen の情報や、dom0 の Linux イメージや、さまざまなゲスト OS イメージを構築するための情報については、http://www.xenproject.org/ を参照してください。

DornerWorks の新しい Zynq MPSoC 用の Xen Zynq ディストリビューションは、http://dornerworks.com/services/XilinxXen でダウンロードできます。ゲスト OS イメージを追加するだけで、独自のエンベデッド仮想化システムを構築できます。

新しい Zynq MPSoC と Xen の組み合わせにより、クラウド コンピューティングを自由に活用できるようになります。

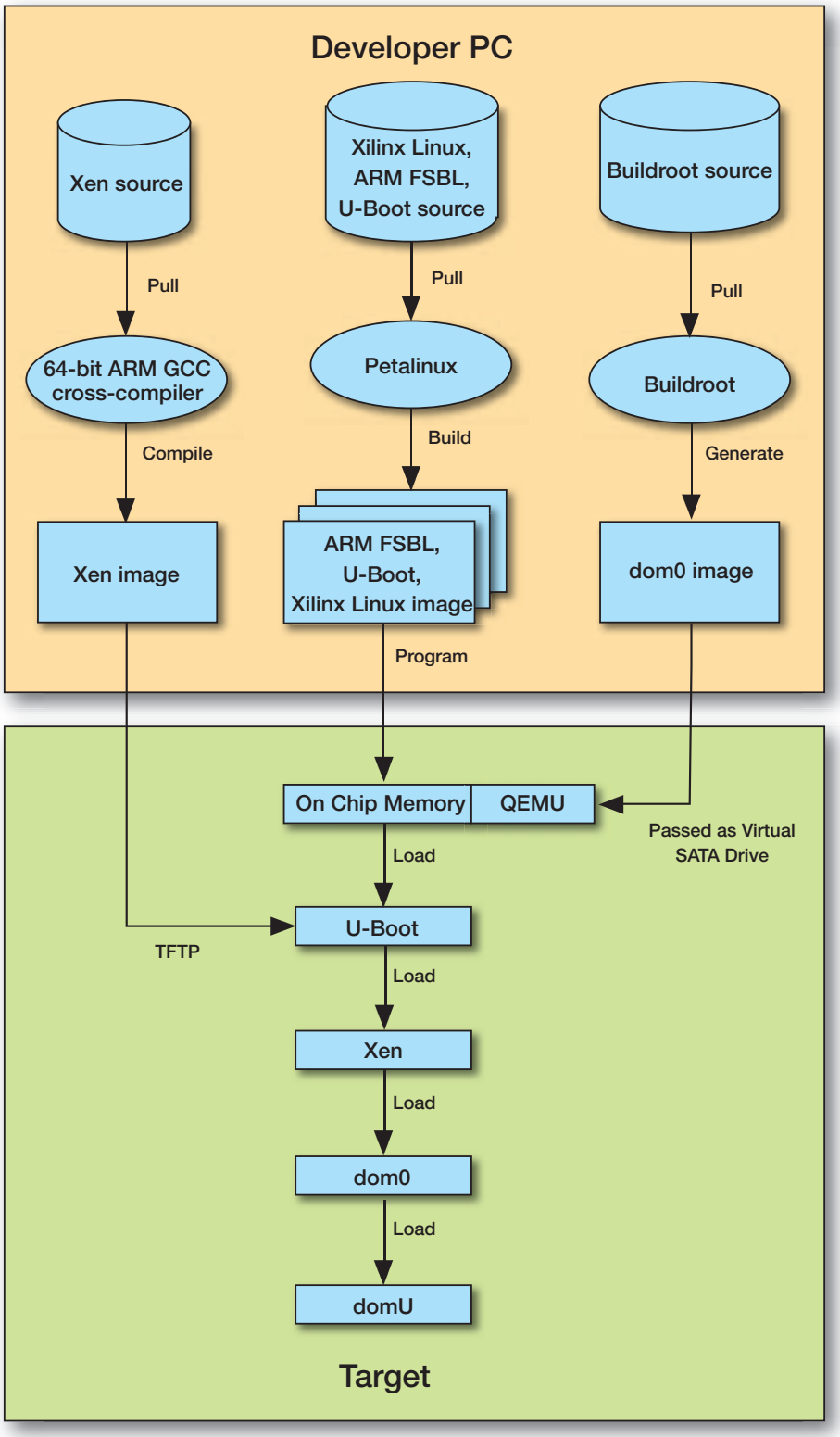


図 5 – Xen 開発ワークフロー

Vivado IPI Opens FPGA Shareable Resources for Aurora Designs

Vivado IPI による Aurora 共有可能 FPGA リソースの実現

K Krishna Deepak
Senior Design Engineer
Xilinx, Inc.
kde@xilinx.com

Dinesh Kumar
Senior Engineering Manager
Xilinx, Inc.
dineshk@xilinx.com

Jayaram PVSS
Senior Engineering Manager
Xilinx, Inc.
jayaram@xilinx.com

Ketan Mehta
Senior IP Product Manager
Xilinx, Inc.
ketanm@xilinx.com

ザイリンクスの IP インテグレーター ツールにより、マルチコア Aurora 使用時のデザイン入力の効率化とリソースの最適化を実現

1 つの FPGA に収める大規模なデザインを、複数のIP (Intellectual Property) インスタンスで構成する際に直面する大きな問題の 1 つは、システム全体でリソースを効率的に共有することです。ザイリンクスの Aurora シリアル通信コアの共有ロジック機能を使用して設計すると、複数のインスタンス間で共有のリソースを使用できるようになります。この共有リソースの効果を最大限に引き出すために欠かせないのが、Vivado® Design Suite の IP インテグレーター ツール (IPI) です。

電子機器業界では、パラレル通信規格から高速シリアル通信規格への移行が急速に進んでいます。業界標準のシリアル通信プロトコルは、伝送速度やレーン幅が固定されており、ギガビット シリアル トランシーバーの性能を十分に活用できないことがありました。

Aurora はザイリンクスの高速シリアル通信プロトコルで、他の業界プロトコルでは複雑すぎることや、リソース消費が大きすぎるなどの理由で実装が困難なアプリケーションによく用いられており、業界でも高い評価を受けています。そしてAurora は、低コストで高速かつスケーラブルな IP コアのソリューションを可能にし、高速シリアル通信データ チャネルを柔軟に設計する手段として利用できます。

伝送速度とチャネル幅の両面においてスケーラビリティが要求される高性能システムやアプリケーションにおいて、Aurora が注目を集めています。また、ASIC 設計や、数ギガビットのデータを伝送するバックプレーンと複数の FPGA デバイスで

構成されるシステム設計においても、その存在感を増しています。Aurora では、シンプルなフレーミング構造、プロトコル拡張フロー制御機能によって、既存のプロトコルからのデータをカプセル化できます。Aurora の電気的要件は、一般の電子機器と互換性があります。ザイリンクスは、Vivado Design Suite の IP カタログの一部として、Aurora 64b66b コアと Aurora 8b10b コアを提供しています。

Vivado の IP インテグレーター (IPI) は、複雑なマルチコア システムのリソース最適化の鍵を握るツールです。IPI は、Aurora 64b66b コアと Aurora 8b10b コアの共有可能リソース、特に、「共有ロジック」機能を最大限活用するのに役立ちます。説明を分かりやすくするために、本稿では Aurora 64b66b IP を取り上げますが、Aurora 8b10b コアについても、同じテクニックを活用できます。

一目で分かる Aurora の共有可能リソース

Aurora 64b66b コアの代表的なブロック図を図 1 に示します。オレンジ色で示されているのは、ミックスド モード クロック マネージャー (MMCM)、BUFG、IBUFDS などのクロック リソースと、GT コモンや GT チャネルなどのギガビット トランシーバー (GT) リソースです。ザイリンクス 7 シリーズ デバイスをベースとする 2 レーン デザインとして、GT1 および GT2 と表記しています。

Kintex®-7 FPGA KC705 評価キットで使用されている、一般的な 16 レーンの Aurora 64b66b コアのクロック リソースと GT リソースの要件を表 1 にまとめます。

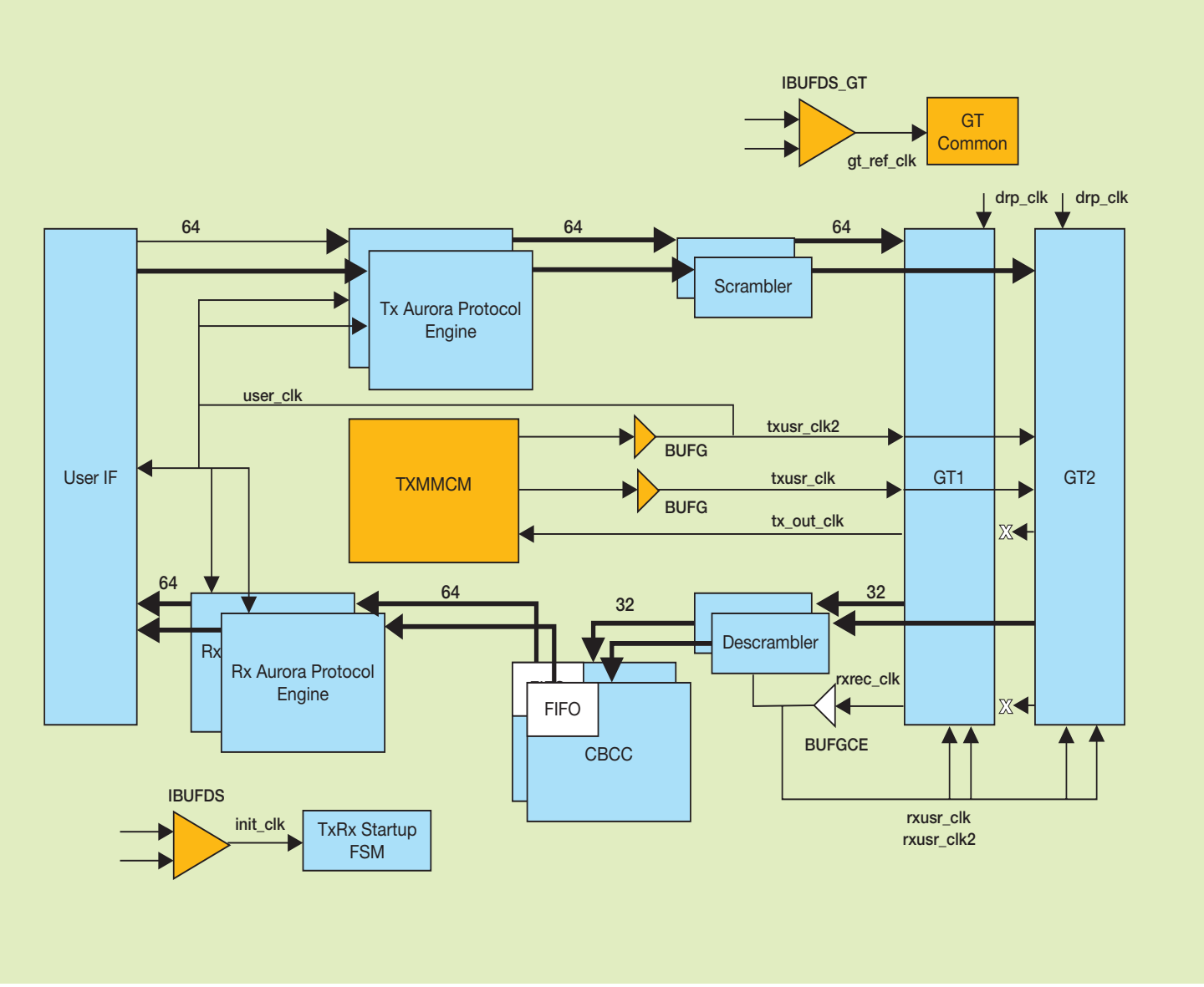


図 1 – Aurora 64b66b コアの共有可能リソースをオレンジ色で表示

16-Lane Aurora Design		
Resource	**Availability in Device**	**Used by Aurora**
MMCME2_ADV	10	1
IBUFDS_GTE2	8	2
GTE2_COMMON	4	4
GTXE2_CHANNEL	16	16

表 3 – Kintex-7 FPGA KC705 評価キットにおけるクロック リソースと GT リソースの使用率

FPGA のクロック リソースと GT リソースは、デバイスと選択されたパッケージによって異なります。システム レベルで使用するためのリソースが、複数の IP コアによって要求されることはよくあります。そのため、貴重なリソースの使用を最適化することは、システム コストと消費電力を削減するために必須となります。

AURORA のリソース共有

共有ロジック機能は、複数の GT ベースのザイリンクス コアによりサポートされています。Aurora コアは、共有ロジック機能の一部として、「コア内の共有ロジック (マスター)」と「サンプル デザイン内の共有ロジック (スレーブ)」のいずれかにて構成できます。この 2 つの構成を組み合わせることで、システム レベルでインスタンシエートした際に、マスターとスレーブでクロック リソースと GT リソースを共有することができます。

共有ロジック機能の適用対象となるアプリケーションでは、複数の IP コア間の接続を手作業で行うと、ミスが起こり易く、かえってデザイン入力にかかる時間が増大します。この問題を解消する方法としては、ツール支援によるデザイン入力が有効です。それが、ザイリンクスの IP インテグレーター ツール (IPI) です。

IPI ツールは、コアを最上位ブロックとして可視化します。また、標準インターフェイス ポート間の接続を、より直感的に、インテリジェントに、場合によっては自動的に行えます。適切なデザイン ルール チェックがツールや IP コアに組み込まれており、不適切な接続は強調表示され、デザイン入力の段階でエラーに気づくことができます。最上位ラッパー ファイルと適切なピン レベル I/O 要件の推論が自動的に行えるため、システム設計の生産性が向上します。カスタムのサブブロックの設計経験がある場合は、ザイリンクス アプリケーション ノート 1168「Vivado IP インテグレーターへのカスタム AXI IPの組み込み」(XAPP1168) に従ってデザインを組み込み、IPI でサブブロックを使用することをお勧めします。

Aurora の共有ロジック機能には、複数のインスタンス間の共有リソースの提供以外にも利点があります。GT コモン、PLL、クロッキング、その他の関連するモジュールを編集することなく、同一の GT クワッドで GT チャネルをそのまま使用できます。唯一の制約は、「共有」コアの伝送速度を同一にする必要があることです (クロック リソースの変更が可能であれば逓倍したクロックを使用することも可能)。

一般的な共有ロジック デザインでは、1 つのクワッド内に、1 つのマスターと 1 つまたは複数のスレーブ インスタンスが存在します。他の多くの通信 IP とは異なり、Aurora では、共有が 1 クワッドに制限されません。Aurora コアの共有ロジック定義は、サポート対象のレーンにいくつでも拡張できます。

以下に、Aurora の共有ロジック機能を使用したアプリケーション例をいくつか挙げます。

複数のシングル レーンで構成されたデザイン

1 つの FPGA に複数のシングル レーンを設計するのは、チャネル ボンディングが必要になる点で、マルチレーン デザインとは異なります。直感的に、複数のシングル レーンで設計した場合、必要なリソースがシステム レベルで直線的に増加するのは明らかだと考えられます。それでは、いくつかのシナリオについて、共有ロジック機能がそれぞれどのように役立つかを検証してみましょう。

まず、4 つのシングル レーンで構成されたデザインを考えます。このようなデザインは、4 つのシングルレーン Aurora コアをインスタンシエートするだけで、容易に作成できます。実際に、インプリメンテーションを行ってみると、Aurora デザインごとに 1 つの GT コモン インスタンスができます。そのため、このデザインの配置およびリソース使用は、4 つの GT クワッドに分散してしまいます。これはリソースを多く消費するため、実用的なソリューションとはなりません。消費電力およびリソース使用の観点から、最適化したソリューションを実現するには、配置を改善して選択した 4 つの GT を同じ GT クワッドに配置することが重要です。

共有ロジック機能がなければ、生成済みのデザインをこの要件に合うように手作業で調整するには、集中的な取り組みが必要となるでしょう。共有ロジック機能の効果的な使用には、1 つの Aurora コアをマスター モードで生成し、他の 3 つの Aurora コア

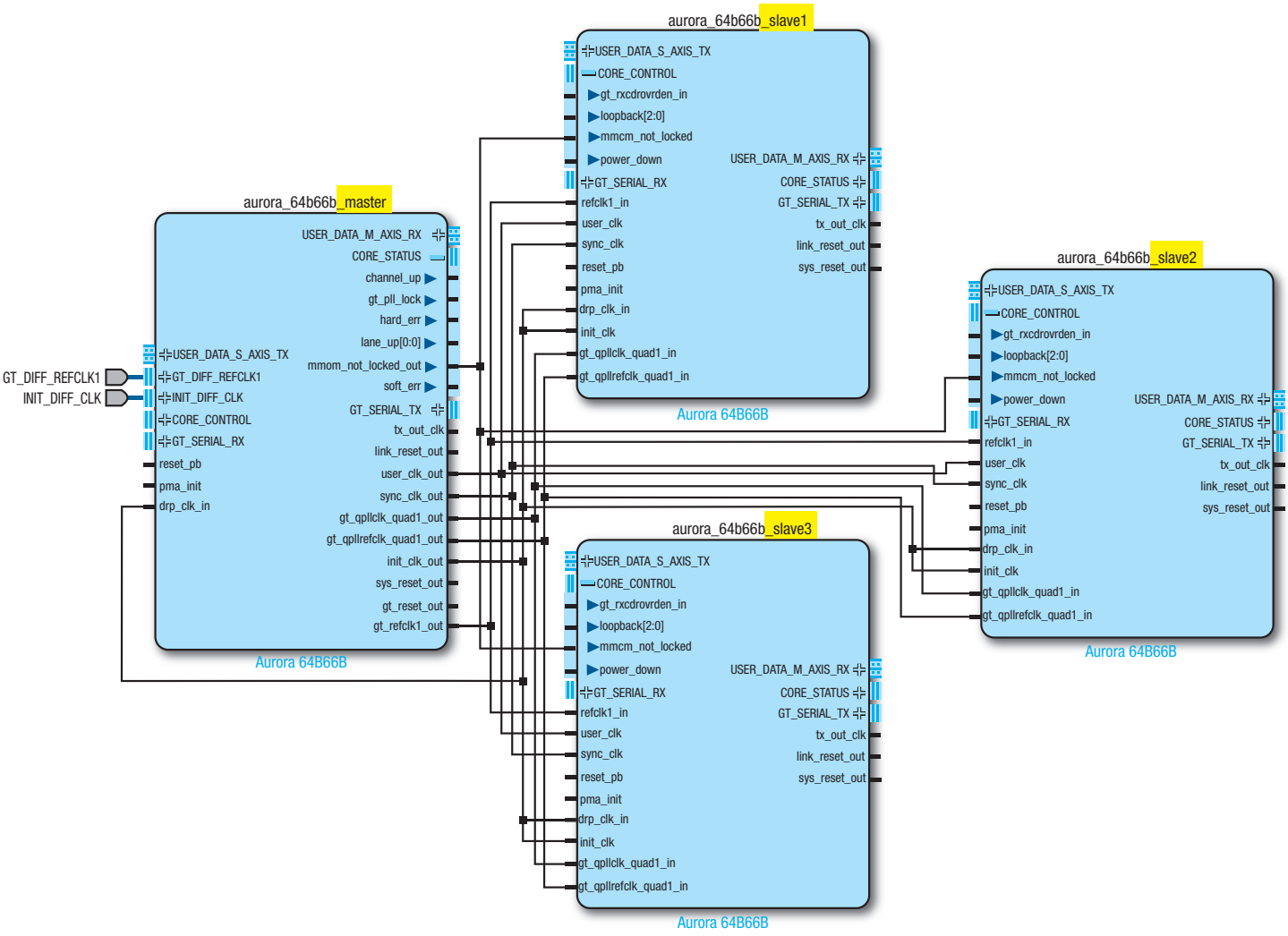


図 2 – 1 つのマスター Aurora コア (左) と 3 つのスレーブを使用した共有ロジック デザイン

Design with Four Single Lanes		
Resource	**Without Shared Logic**	**With Shared Logic**
MMCME2_ADV	4	1
IBUFDS_GTE2	4	1
GTE2_COMMON	4	1
GTXE2_CHANNEL	4	4

表 4 – 4 つのシングル レーンを共有ロジックで設計した場合のリソース メリット

12-Single-Lane Designs			
Resource	Without Shared Logic	With Shared Logic (default)	With Shared Logic (using single-ended master input clocks)
MMCME2_ADV	12	3	3
IBUFDS_GTE2	12	3	1
GTE2_COMMON	12	3	3
GTXE2_CHANNEL	12	12	12

表 5 – 12 のシングルレーンを共有ロジック機能で設計した場合のリソース メリット

をスレーブ モードで生成する必要があります (図 2)。マスター コアがスレーブ コアへのクロック供給を制御することによるコアのリセットなど、その他のシステム レベルの検討事項があります。このような構成とリソースの最適化は、Aurora コアがすべて同じ伝送速度で構成されている場合に限り、容易に実行できます。表 2 は、4 つのシングルレーン デザインで構成されたシステムで共有ロジック機能を使用するメリットを数値で表しています。

12 の GT チャネルを占めるデザイン

7 シリーズ FPGA において、ノース サウス クロッキングに基づく GT の要件は、ミドル クワッドの場合に、1 つの基準クロック ソースで最大 12 の GT チャネルをサポートできることです。

最小限のクロック リソースを使用して、12 のシングルレーンを設計するケースを考えてみましょう。図 2 に示す 1 マスター + 3 スレーブ構成を拡張することで、クロック リソースを節約できます。しかし、この 1+3 構成を 3 クワッド分拡張すると、合計 6 つの差動クロック リソースが必要になります。そこで、マスター デザインから 2 つを選択して、シングルエンド INIT_CLK と GT 基準クロック

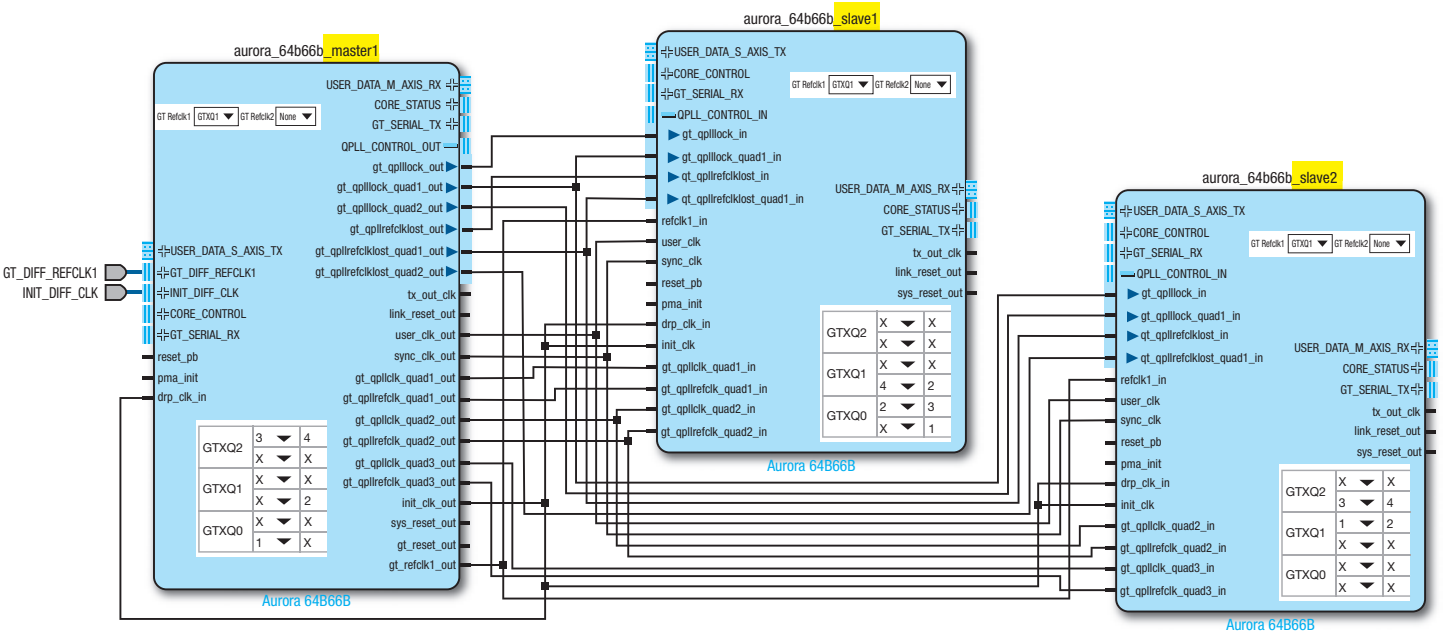


図 3 – 3 つの連続するクワッドにまたがる、1 マスター、2 スレーブ構成による、4 レーン Aurora デザイン

Optimized Lane Selection for 3 x 4-Single-Lane Designs		
GTQ2	Master1_3	Master1_4
	Slave2_3	Slave2_4
GTQ1	Slave2_1	Slave2_2
	Slave1_4	Master1_2
GTQ0	Slave1_3	Slave1_2
	Master1_1	Slave1_1

表 6 – 3 x 4 レーン デザインにおけるレーン選択の最適化

を受け入れるよう構成することで、さらなるリソースの節約が可能です。この方法により、システムの差動クロック入力要求を 6 から 2 に削減できる可能性があります。したがって、IBUFDS/IBUFDS_GTE2 リソースの要件が減少します (表 3)。IBUFDS_GTE2 のリソースが減少すると、実際には、外部クロック リソースや、デザインのピン数も減少します。MMCM についても同様な最適化が見込めます。

3 x 4 レーン デザイン

3 つの 4 レーンデザインという要件がある場合、共有ロジック機能がなければ、マスター モードで 3 つの 4 レーン Aurora コアを作成し、生成したデザインを手作業で調整を加え、クロック リソースの使用を最適化することになります。実は、同じことを、簡単に行う方法があります。それは、図 3 のように 1 つのマスター コア、2 スレーブ コア構成をカスタマイズする方法です。

シングルレーン Aurora デザインを大規模化 (16 以上) していくと、共有ロジック機能の必要性はさらに高まります。中には、48

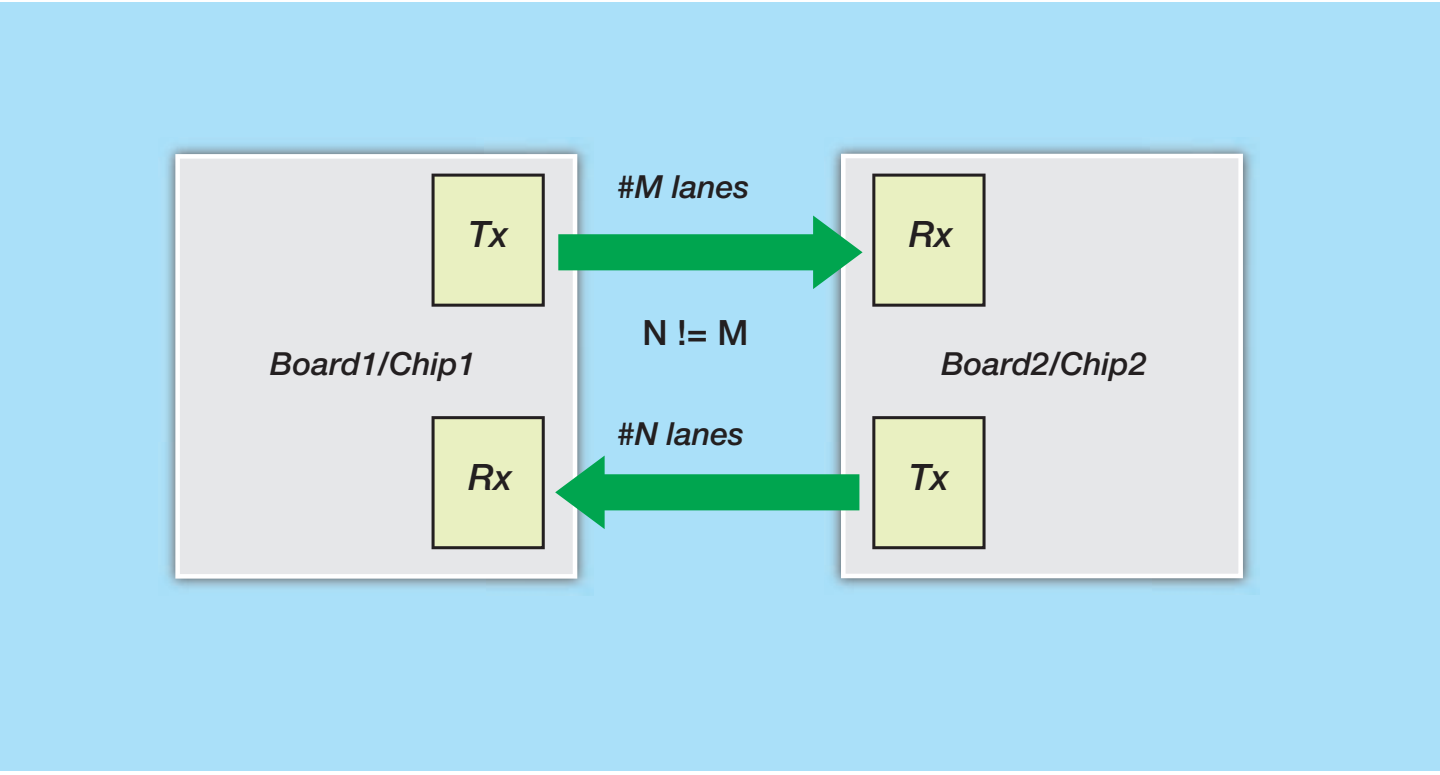


図 4 – Aurora で実現したリンク間の非対称データ転送

48-Single-Lane Designs		
Resource	**Without Shared Logic**	**With Shared Logic**
MMCME2_ADV	48	4
IBUFDS_GTE2	48	4
GTE2_COMMON	48	12
GTXE2_CHANNEL	48	48

表 7 – 48 のシングルレーンを共有ロジック機能で設計した場合のリソース メリット

シングルレーンの独立した全二重リンクが要求される場合もあります。Aurora シングル レーン リンクの数は、選択したデバイスで利用できる GT リソース数のみに制限されます。このようなケースでは、共有ロジック機能を活用せずに設計するのは困難です。

このようなデザインは、12 クワッドにまたがることがあります。そこから想定される 2*12 の差動クロック リソースの要件は、ボード設計の観点からは大変な作業になります。12 のシングルレーン デザインの例で説明したテクニックで設計することで、システム全体の差動クロックと MMCM 要件を低減できます (表 5)。

非対称レーンとその他のカスタム最適化

ビデオ プロジェクタなどのアプリケーションでは、一方向にメインストリーム データが高スループットで流れ、もう一方向では補助情報や制御情報が低スループットで流れます。このようなアプリケーションでは、完全な全二重リンクを設計しても、帯域幅の使用率が低下し、システム デザインの ROI が低下します。このような課題に対する理想的なソリューションは、図 4 に示すように、ある方向 (高スループット) のレーン数がもう片方の方向 (低スループット) よりも多い、GT リソース使用に最適化された非対称リンク幅を設計することです。

Aurora コア で現在提供されているデータ フロー モード (単方向/全二重) では、TX と RX が同じレーン数のコアのみを構成できます。方向によってレーンの数を変えるには、計 2 つの Aurora 単方向コアを各方向に生成します。7 シリーズ FPGA でこのような非対称レーンを設計する 1 つの方法が、ザイリンクス アプリケーション ノート 1227「Aurora 64B/66B IP コアを使用した非対称レーンの設計」(XAPP1227) にまとめられています。

もう 1 つの役立つ設計ストラテジが、BUFG リソースの最適化です。多くの場合、伝送速度が同じか異なる、複数の Aurora コアをインプリメントするには、デバイス固有のクロック要件と制限を把握している必要があります。多数の Aurora リンクをインプリメントする場合、各リンクに対してクロックの生成が必要になります。クロック リソースを節約すると、システムのコスト パフォーマンスが上がります。システム デザインに複数のブロックがあり、クロック リソース (BUFG) が不足する場合、BUFG を BUFR/BUFH に置き換えることを検討すると良いでしょう。ただし、GT コアの両方の TX パス ユーザー クロックは、同じバッファー タイプで駆動することをお勧めします。

7 シリーズ Aurora コアには、通常 1 つの BUFG を使用する、追加のダイナミック リコンフィギュレーション ポート (DRP) クロック入力が必要です。Aurora のフリーランニング クロック周波数を DRP クロックの許容範囲内で選択すると、Aurora の出力フリーランニング クロックの空きを再利用して、DRP クロックに戻すことができます。その結果、生成するデザインの BUFG 数を削減できます。

複数の Aurora デザインの伝送速度を選択する際、伝送速度が整数倍であれば、リンク間のクロックの派生や共有が簡単になることを考慮してください。共有ロジック機能を高調波伝送速度に拡張する場合は、いくつかのクロック分周器を追加で設計することで、スレーブ Aurora コアに必要な入力周波数を生成できます。

将来の可能性

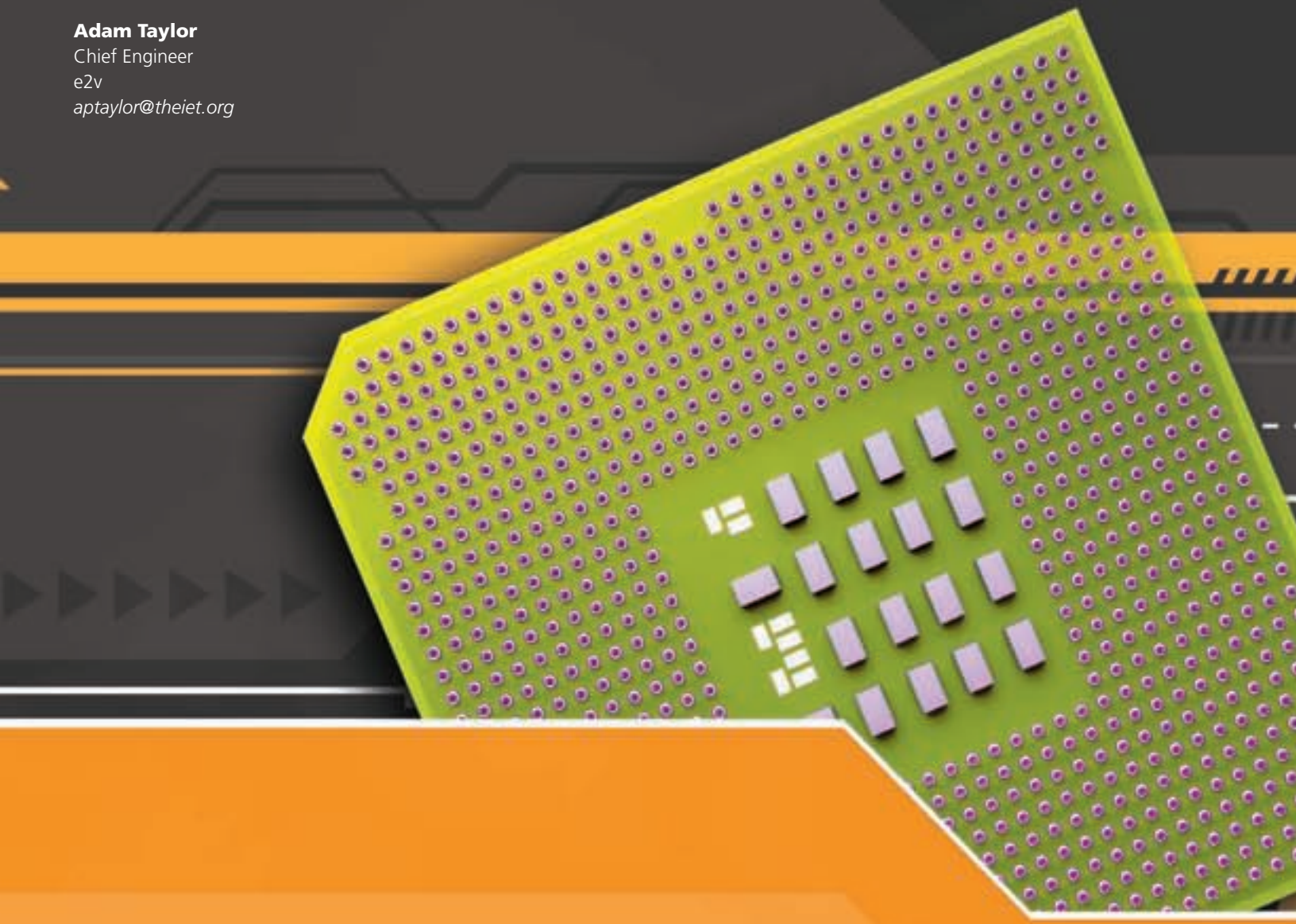
Aurora の柔軟な機能によって、さまざまなシステム構成とアプリケーションを開発する可能性が開かれます。ザイリンクスの Vivado IP インテグレーターなどのパワフルなツールと組み合わせることで、デザイン入力の生産性とシステム レベルのリソース共有が向上します。その結果、All Programmable アプリケーションのイノベーションが促進されます。ザイリンクスの UltraScale™ アーキテクチャでは、さらに多くの GT チャネルと高速な GT 伝送速度をサポートするデバイスが提供されるため、今後の可能性とリソース使用の効率化はさらに広がります。

Aurora コアを評価するには、IP カタログ、IPI、Aurora 製品のウェブ ページ (http://japan.xilinx.com/products/design_resources/conn_central/grouping/aurora.htm) を参照してください。

Making XDC Timing Constraints Work for You

XDC タイミング制約の活用

Adam Taylor
Chief Engineer
e2v
aptaylor@theiet.org



タイミング制約と配置制約は、デザイン要件を達成するために不可欠な要素です。各制約の活用方法の基本を解説します。

RTL デザインの完成は、FPGA デザインの完成に至るプロセスの一部です。その後には、シリコンのタイミング要件およびパフォーマンス要件を満たす作業が待ち受けています。その中で、多くの場合、タイミング制約と配置制約の両方を定義する必要があります。

本稿では、ザイリンクスの FPGA および SoC でシステムを設計する際に、これらの両制約を作成し、活用する方法を解説します。

タイミング制約

タイミング制約は、最も基本的なレベルでは、設計するシステムのクロックの動作周波数を決定します。より詳細なタイミング制約では、クロック パス間の関係を定義します。タイミング制約は、クロック パスの解析が必要であるか、無視するか (クロック パス間に有効なタイミング関係がない場合) を判断するために使用されます。

ザイリンクスの Vivado® Design Suite は、デフォルトではすべての関係を解析します。しかし、デザイン内のすべてのクロックが正確に解析可能なタイミング関係を持っているわけではありません。その 1 つの例は、非同期クロックです。非同期クロックでは、その位相を正確に特定することができません (図 1 を参照)。

クロック パス間の関係を管理するには、制約ファイルでクロック グループを宣言します。クロック グループを宣言すると、グループ内で定義されたクロック間のいずれの方向についても、Vivado ツールによるタイミング解析は実行されません。

Vivado ツールでは、タイミング制約を容易に作成できるよう、クロックが「同期クロック」、「非同期クロック」、「共通周期のないクロック」の 3 つのカテゴリに分類されます。

- 同期クロック – タイミング/位相の関係が予測可能なクロックです。通常、プライマリ クロックとその派生クロックがこれに分類されます。これらは、同じルートから発生しており、同じ周期を持っているからです。
- 非同期クロック – 予測可能なタイミング/位相の関係性を持たないクロックです。通常、異なるプライマリ クロック同士 (およびその派生クロック) がこれに分類されます。非同期クロックは、異なるルートを持ちます。

- 共通周期のないクロック – 2 つのクロック間に、1,000 サイクルを超えても共通周期を見つけることができなかった場合、これらのクロックは共通周期がないクロックとみなされます。その場合、その 1,000 サイクルのワーストケース セットアップ関係が使用されます。しかし、現実にワーストケースである保証はありません。

クロックの種類を知るには、Vivado ツールで生成されるクロック レポートを使用します。このレポートで、非同期クロックと、共通周期のないクロックを特定できます。

非同期クロックと、共通周期のないクロックを特定したら、「クロック グループ」制約を使用して、クロック間でタイミング解析を無効化できます。Vivado スイートでは、ザイリンクス デザイン制約 (XDC) が使用されます。XDC は、広く使用されている Tcl ベースの制約フォーマットである Synopsys デザイン制約 (SDC) に基づく制約です。XDC でクロック グループを定義するには、次のコマンドを使用します。

```
set_clock_groups
–name –logically_exclusive
–physically_exclusive
–asynchronous –group
```

–name は、グループの名前です。–group オプションには、グループのメンバー (タイミング関係がないクロック) を定義します。–logically_exclusive オプションと –physically_exclusive オプションは、クロック ツリーの駆動源として複数のクロック ソースを選択できる場合に使用します (BUFGMUX と BUFGCTL を含む)。これらのクロックは、クロック ツリー上に同時に出現することができません。相互排他的であるこれらのクロック間の関係を Vivado ツールで解析をしないよう設定します。また、–asynchronous オプションは、非同期クロック パスを定義します。

タイミング関係の定義における最後の検討事項は、クロックの非理想的関係、特に、ジッターを考慮することです。ジッターは、入力ジッターとシステム ジッターの 2 つに分けて考える必要があります。入力ジッターは、プライマリ クロックの入力時に存在しているジッターで、実際の遷移のタイミングと理想的な条件下でのタイミングとの差です。システム ジッターは、デザインに存在するノイズ等によって発生するジッターです。

set_input_jitter 制約を使用すると、各プライマリ入力クロックにジッターを定義できます。また、set_system_ jitter 制約で、デザイン全体 (すべてのクロック) に対してシステム ジッターの定義ができます。

タイミング例外

定義したクロック グループ内において、例外が発生した場合の挙動についても注意する必要があります。ところで、例外とは何でしょうか。

一般的なタイミング例外の例として、クロック サイクル 1 つおきに結果を取得する場合があります。また、同期されている 2 つのクロック間で、低速クロックから高速クロック (もしくはその逆) にデータを転送している場合もあります。実は、この 2 つは、一般に「マルチサイクル パス」と呼ばれるタイミング例外の例です (図 2 を参照)。

このようなパスにマルチサイクル パスを宣言すると、条件を緩和して適切にタイミング解析を行い、他のクリティカルなパスにタイミング エンジンを割くことができます。したがって、品質の向上が可能になります。

XDC ファイルでマルチサイクル パスを宣言するには、次の XDC コマンドを使用します。

```
set_multicycle_path path_
multiplier [-setup|-
hold] [-start|-end]
[-from <startpoints>] [-to
<endpoints>] [-through
<pins|cells|nets>]
```

マルチサイクル パスを宣言すると、実質的には、セットアップ解析またはホールド解析 (もしくはその両方) の要件を path_multiplier で乗算することになります。たとえば、上述のクロック サイクル 1 つおきに出力する例では、セットアップ タイミング のpath_mutiplier が 2 になります。

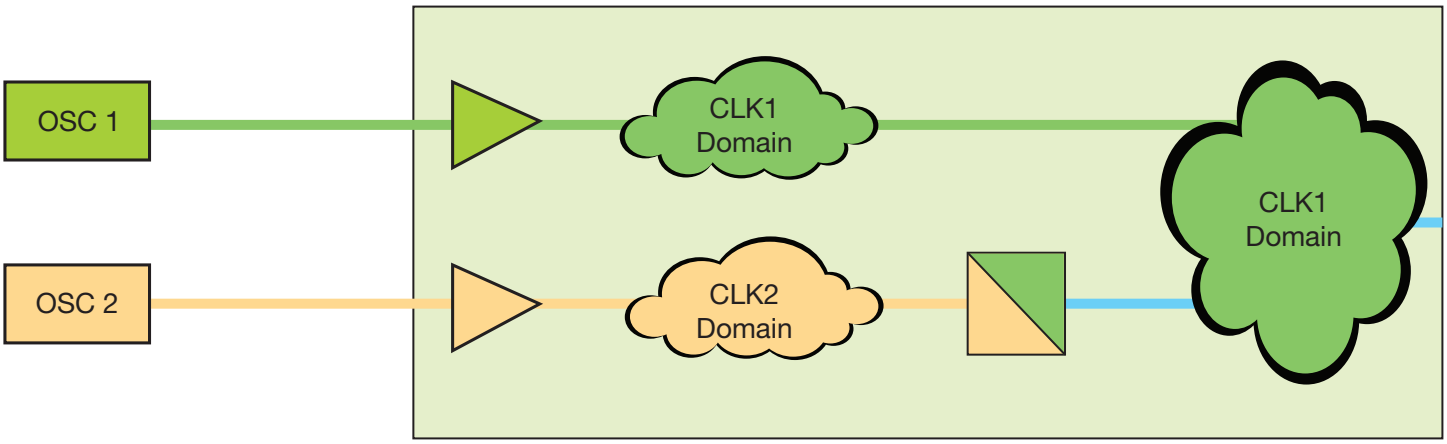


図 1 – CLK1 ドメインと CLK2 ドメインは非同期

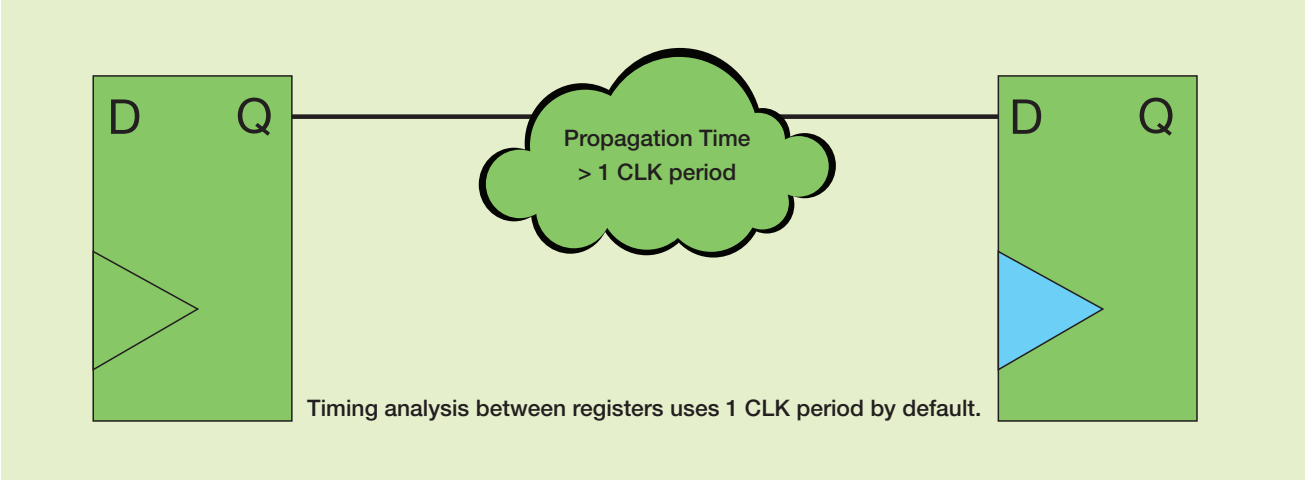


図 2 – タイミング例外の一例であるマルチサイクル パス

マルチサイクル パスは、セットアップとホールドのいずれに適用するか、そして、どこに適用するかを選択できます。一般に、セットアップ乗数を宣言した場合は、次の論理式を使用して、ホールド タイム乗数も宣言しておくことをお勧めします。

ホールド サイクル数 = セットアップ乗数 - 1 - ホールド乗数

ここから、上述のシンプルな例において、ホールド乗数は、次の論理式によって定義されることが分かります。

ホールド乗数 = セットアップ乗数 - 1 (共通クロック使用時)

マルチサイクル パスの重要性を示す簡単なサンプル ファイルを、こちらからダウンロードいただけます。この XDC ファイルには、セットアップとホールド両方の 2 つのマルチサイクル パスを宣言したサンプルが 1 つ含まれています。

物理制約

最もよく使用される物理制約は、I/O ピンの配置と、標準駆動電流値などのI/O ピン関連のパラメーターの定義です。物理制約には、他にも、配置制約、配線制約、I/O 制約、コンフィギュレーション制約などがあります。配置制約ではセルの配置を定義できます。配線制約では信号の配線を定義できます。I/O 制約では I/O の配置とそのパラメーターを定義できます。コンフィギュレーション制約では、コンフィギュレーションの方法を定義できます。

ただし、これらの分類以外に属するいくつかの制約があります。Vivado Design Suite には、このような制約が 3 つあり、主にネットリストで使用されています。

- DONT_TOUCH – 最適化を回避します。セーフティ クリティカルなシステム、高信頼性システムの実装に非常に役立ちます。
- MARK_DEBUG – デバッグ用に RTL ネット名を保持します。
- CLOCK_DEDICATED_ROUTE – クロックの配線を指定します。

最も一般的に使用されているのは、I/O 配置と I/O コンフィギュレーションに関連する制約です。I/O を FPGA 上に配置するには、物理ピンを配置するための配置制約と、I/O 規格、スルー レートなどの I/O 特性をコンフィギュレーションする I/O 制約の両方を使用する必要があります。

最近の FPGA では、多種のシングルエンド I/O 規格や差動 I/O 規格がサポートされています。これらは、I/O 制約を介して定義されます。ただし、最終的なピン配置によって定まる I/O バンク規則に合致しているかどうかに注意する必要があります。

ところで、I/O バンク規則とは何でしょうか。FPGA には多くのユーザー I/Oがあり、多数のバンクにグループ化されています。幅広い種類の I/O 規格に対応するため、バンクごとに電圧源は独立しています。Zynq®-7000 All Programmable SoC (およびその他の 7 シリーズ デバイス) では、I/O バンクはさらに、High Performance (高性能、HP) と High Range (広範囲、HR) の 2 つのクラスに大別されます。クラスによって、性能がさらに制約されるため、各インターフェイスに適したクラスを使用する必要があります。

HP クラスは、高いデータ レートに最適化されています。そのため、動作電圧は低く、LVCMOS 33 と LVCMOS 25 はサポートしていません。一方、HR クラスは、HP がサポートしていない幅広い I/O 規格に対応するよう最適化されています。HR は、従来の 3.3V および 2.5V インターフェイスをサポートします。これらのバンクを図 3 に示します。

信号の駆動電流とスルー レートは、その信号で使用するバンクを決定した後でも変更できます。これらは、ボードのシグナル インテグリティを最適化するためにはハードウェア設計チームにとって重要なメトリクスです。その

選択は、ボード設計のタイミングにも影響します。そのため、SI (シグナル インテグリティ) ツールを使用できます。

SI ツールを利用するには、IBIS モデルが必要です。IBIS モデルを Vivado ツールから抽出するには、インプリメントしたデザインを開いて、[File] → [Export] → [Export IBIS model] を選択します。このファイルを SI ツールに読み込むことで、システム レベルの SI 問題を解決し、最終 PCB レイアウトのタイミング解析を完了できます。

システム全体の SI 性能とタイミングの最適化が完了すると、I/O について、次のような多数の制約が作成されます。

```
set_property PACKAGE_PIN G17
[get_ports {dout}]
set_property IOSTANDARD
LVCMOS33 [get_ports {dout}]
set_property SLEW SLOW [get_
ports {dout}]

set_property DRIVE 4 [get_
ports {dout}]
```

HP I/O バンクでは、デジタル コントロール インピーダンス (DCI) の使用により、外部終端について考慮せずに正確なI/O 終端を施し、システムの SI を向上することもできます。たとえば、外部コネクタに接続されている場合など、駆動している信号がない場合の I/O への影響も考える必要があります。その場合、I/O 制約でプルアップ抵抗またはプルダウン抵抗を実装して、システムの問題の原因となり得る FPGA 入力信号のフロート状態を防ぐことができます。

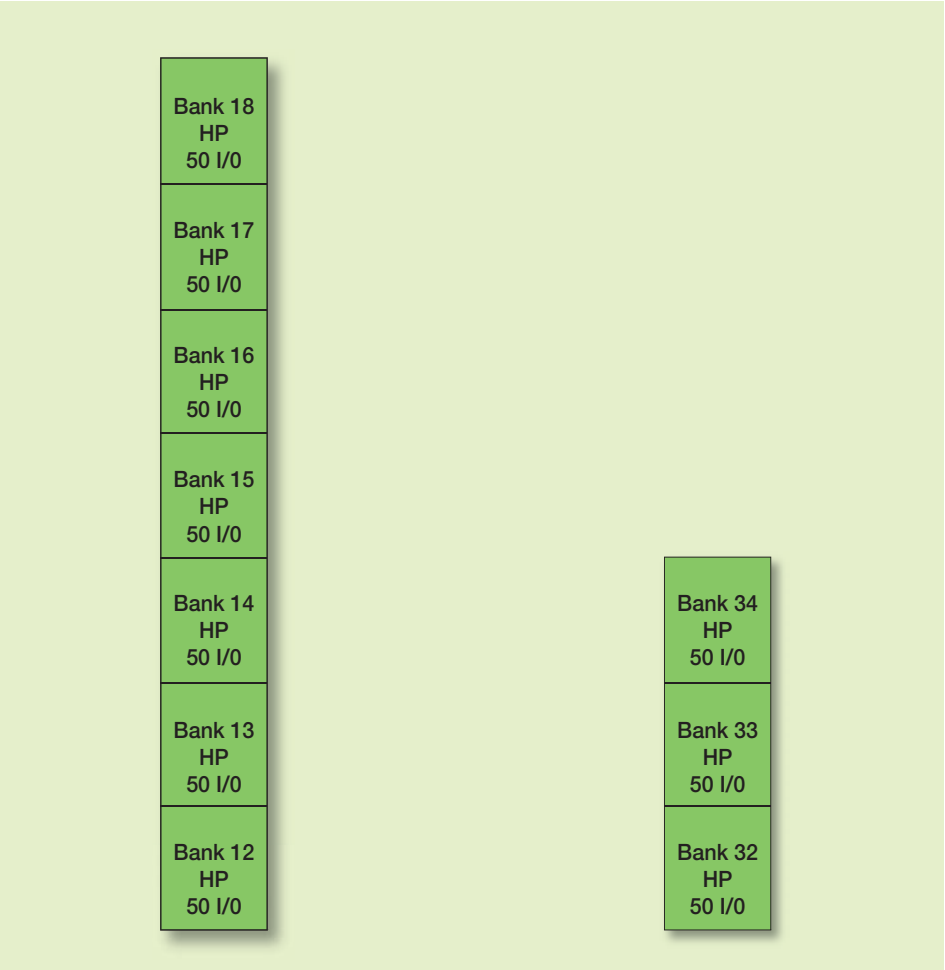


図 3 – ザイリンクス 7 シリーズ デバイスの HP (左) と HR I/O バンク

もちろん、物理制約を使用してI/O ブロック内部の最終出力前段のフリップフロップをインプリメントすることによりデザインのタイミングを向上することもできます。この場合、クロックから出力へのタイミングが短縮されます。入力信号でも同じことができ、ピン間のセットアップおよびホールドのタイミング要件を満たすことができます。

物理制約は配置から

配置制約を行う理由については、多くのことが考えられます。タイミング目標を達成するため、もしくは、デザインをセクションごとに分離するためなどがあります。配置制約では、次の 3 種類の制約が重要になります。

- BEL (Basic Element of Logic)– ネットリストの要素を、あるスライス内に配置できます。
- LOC (Location)– ネットリストの要素を、デバイス内のある位置に配置します。
- PBlock (物理 (P) ブロック)– 論理ブロックを FPGA のある領域に制約できます。

このように、LOC 制約では、デバイス内のスライスまたはその他の位置を定義できますが、BEL 制約では、そのスライス内で使用するフリップフロップをより細かく指定できます。PBlock 制約を使用すると、ロジックをグループ化して、デザインの大きな領域をセグメント化できます。また、PBlock は、パーシャル リコンフィギュレーションを行うロジック領域の定義にも使用できます。

ケースによっては、タイミングを最適化するために、小規模なロジック ファンクションをグループ化することがあります。PBlock でも可能ですが、そのような場合は、RPM (相対配置マクロ) を使用する方が一般的です。

RPM を使用すると、DSP、フリップフロップ、LUT、RAM などのデザイン要素を配置内でグループ化できます。RPM は、PBlock とは異なり(特に指定しない限り) 各要素の配置がデバイスの特定の領域に制約されません。RPM では、配置時に要素がグループ化されます。

```
SIGNAL ip_sync : STD_LOGIC_VECTOR(1 DOWNTO 0) :=(OTHERS =>'0');
SIGNEL shr_reg : STD_LOGIC_VECTOR(31 DOWNTO 0) :=(OTHERS =>'0');

ATTRIBUTE RLOC : STRING;
ATTRIBUTE HU_SET : STRING;
ATTRIBUTE HU_SET OF ip_sync : SIGNAL IS "ip_sync";
ATTRIBUTE HU_SET of shr_reg : SIGNAL IS "shr_reg";
```

図 4 – ソース コード内の制約

デザイン要素を近くに配置すると、2 つの目標を達成できます。具体的には、リソース効率の向上と、インターコネクト長の微調整による、タイミング性能の向上になります。

デザイン要素を近くに配置するには、次の 3 種類の制約を使用できます。これらは、HDL ソース ファイルで定義できます。

- U_SET – 階層に関わらず、RPM のセルのセットを定義します。
- HU_SET – 階層に従って、RPM のセルのセットを定義します。
- RLOC – SET に対する相対配置を割り当てます。

RLOC 制約では、RLOC = XmYm の定義が使用されます。ここで、X と Y は、FPGA アレイの座標に関連します。RLOC を定義すると、RPM_GRID 属性の有無によって、相対座標か絶対座標のいずれかに指定できます。RPM_GRID 属性を付加すると、相対座標ではなく絶対座標として定義されます。

これらの制約は、図 4 に示すように HDL 内で定義するため、多くの場合、配置を正しく定義するには、HDL ファイルに制約を追加する前に、配置配線イタレーションを実行しておくことが必要になります。

以上のように、タイミング制約と配置制約を理解し、正しい使用方法を学ぶことは、ザイリンクス ベースのプログラマブル ロジック デザインで最高の品質を得るための鍵となります。

Toward Easier Software Development for Asymmetric Multiprocessing Systems

非対称型マルチプロセッシング システムのソフトウェア開発を容易化

Arvind Raghuraman
Staff Engineer
Mentor Graphics Corp.
arvind_raghuraman@mentor.com

Mentor Embedded Multicore Framework は、ヘテロジニアス ハードウェアおよびソフトウェア環境の複雑性をカバーし、SoC システムの設計を容易化します。

現在、エンベデッド アプリケーションにおいて、ヘテロジニアス マルチプロセッシングの重要性が高まっています。ザイリンクスの Zynq® UltraScale+™ MPSoC などの SoC (システム オンチップ) アーキテクチャは、クワッド ARM® Cortex®-A53 コアとデュアル ARM Cortex-R5 コアで構成されるパワフルなヘテロジニアス マルチプロセッシング インフラストラクチャを提供します。SoC には、コアの演算インフラストラクチャの他に、ハードウェア化されたペリフェラル IP および FPGA ファブリックが豊富に含まれており、高性能マルチプロセッシング システムを設計するためのフレキシブルな設計モデルを適用できます。

Zynq MPSoC などの SoC のマルチプロセッシング性能を活用する場合には、さまざまなソフトウェア開発モデルを使用できます。対称型マルチプロセッシング (SMP) OS は、マルチプロセッシング システム内の複数のホモジニアス コア間で、アプリケーション ワークロードのバランスを対称的、もしくは非対称的に調整するために必要なインフラストラクチャを備えています。しかし、システム内のヘテロジニアス プロセッサの演算帯域幅を活用するには、非対称型マルチプロセッシング (AMP) ソフトウェア アーキテクチャが必要です。

AMP アーキテクチャでは、一般的に、SoC 内に存在する異種のプロセッシング コア上で実行される異種のソフトウェア環境のあらゆる組み合わせ (Linux、リアルタイム オペレーティング システム (RTOS)、ベアメタル ソフトウェアなど) がエンド アプリケーションの設計目標を実現するために協調動作します。一般的なデザインでは、マスター コア上のソフトウェア コンテキストが、リモート コア上のリモート ソフトウェア コンテキストをオンデマンドで起動して演算負荷を分散します。マスター プロセッサ、リモート

プロセッサ、そしてそれぞれのソフトウェア コンテキスト (すなわち、その OS 環境) は、性質として、ホモジニアスもしくはヘテロジニアスの両方が可能です。異種 (または同種) のプロセッサ上のいくつかのオペレーティング システムのライフ サイクル管理の複雑性にうまく対処しながら、演算負荷分散用にプロセッサ間通信 (IPC) インフラストラクチャを提供するには、改良された新しいソフトウェア性能および手法が必要になります。

Mentor Graphics の Mentor Embedded Multicore Framework は、AMP システム開発者に 2 つの重要な機能を提供するソフトウェア フレームワークです。その 1 つは、リモート プロセッサおよびそのソフトウェア コンテキストのライフ サイクル管理を行う remoteproc コンポーネントおよび API です。もう 1 つは、AMP 環境内の OS コンテキスト間の IPC を実現する rpmsg コンポーネントおよび API です。Mentor Embedded Multicore Framework は、ヘテロジニアス ハードウェアおよびソフトウェア環境を管理する際の複雑な作業をカバーし、アプリケーション レベルのシンプルなインターフェイスを提供します。

この新しい開発フレームワークで AMP システムにおけるヘテロジニアス演算を行う方法を詳しく見ていきましょう。

互換性と歴史

オープン規格に準拠していることと、Linux コミュニティに採用されていることは、Mentor Embedded Multicore Framework に適した API を選択する上で重要な条件でした。Mentor Graphics が選択したのは、Linux 3.4.x 以降のカーネルに搭載されている remoteproc および rpmsg API です。Linux の remoteproc および rpmsg インフラストラクチャは、元々、Texas Instruments によって開発され、Linux カーネルに搭載されたものです。このインフラストラクチャによって、マスター プロセッサ上の Linux OS でライフ サイクルの管理や、リモート プロセッサ上のリモート ソフトウェア コンテキストとの通信の管理を行えるようになりました。

しかし、Linux ベースのインフラストラクチャにはいくつかの制限がありました。まず、Linux rpmsg は、Linux が常にマスター OS であることを暗黙的に前提としており、Linux をリモート OS とする AMP コンフィギュレーションはサポートされていませんでした。さらに、remoteproc API と rpmsg API は、Linux カーネル空間からしか使用できず、他の OS やランタイムから使用できる同等の API やライブラリは存在しませんでした。

Mentor Embedded Multicore Framework は、C 言語で記述されたスタンドアロン ライブラリです。RTOS やベアメタル ソフトウェア環境で使用できる remoteproc および rpmsg 機能をクリーンルーム実装しています。これらは、Linux 版と API レベルの互換性を持ち、同等の機能を持ちます。図 1a は、Mentor Embedded Multicore Framework のソフトウェア スタック図と、RTOS またはベアメタル環境における使用法を示しています。この図から分かるように、本フレームワークの十分に抽象化された Porting Layer (移植層) は、ハードウェア インターフェイス層 (HIL) と OS 抽象化層で構成されているため、他のプロセッサや OS に容易に移植できます。

図 1b は、Linux カーネル上の remoteproc および rpmsg インフラストラクチャを示して

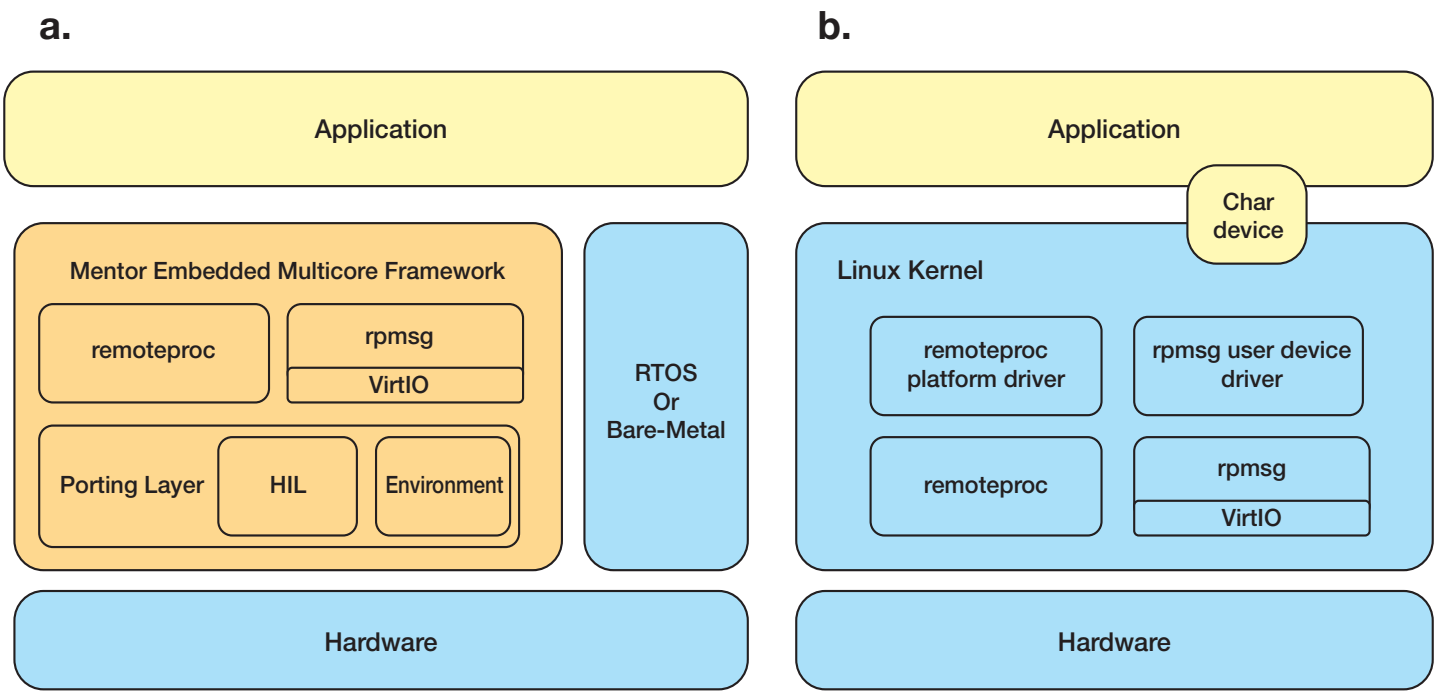


図 1 – Mentor Embedded Multicore Framework: RTOS およびベアメタル環境 (a) とLinux カーネル内の remoteproc および rpmsg (b)

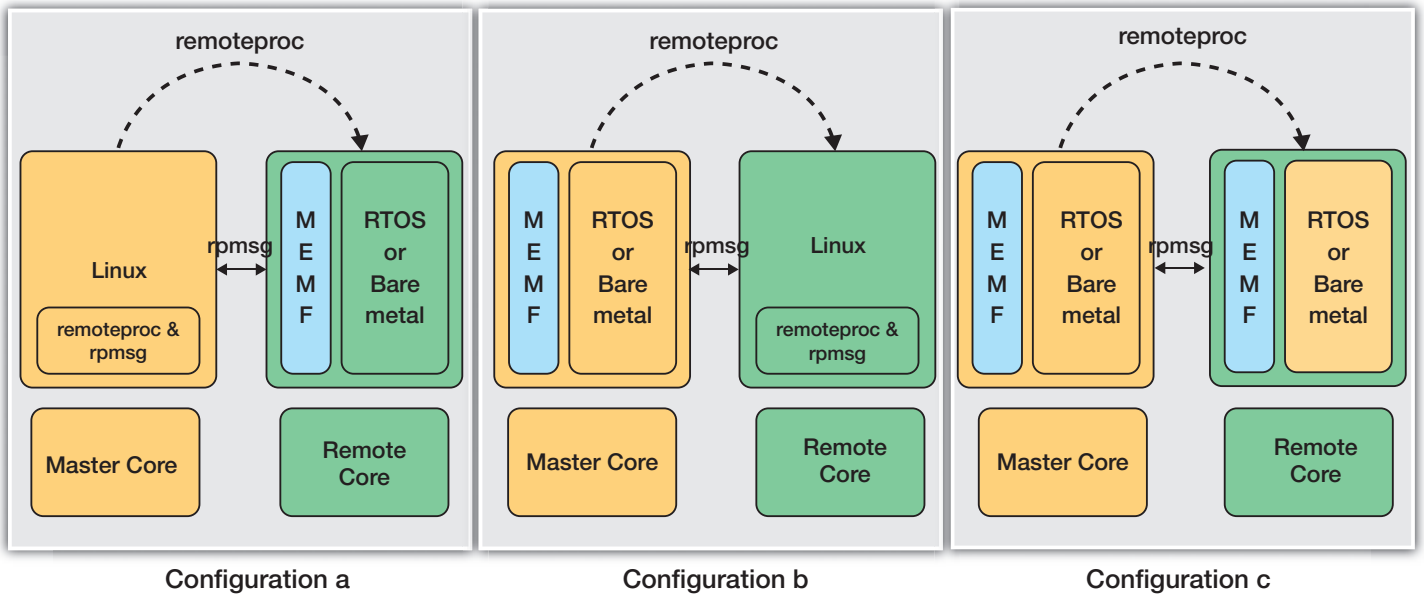


図 2 – Mentor Embedded Multicore Framework がサポートするAMP コンフィギュレーション

います。remoteproc および rpmsg カーネル空間ドライバーが、remoteproc プラットフォーム ドライバーや、rpmsg ユーザー デバイス ドライバーにサービスを提供します。remoteproc プラットフォーム ドライバーは、リモート ライフ サイクル管理を行い、rpmsg ユーザー デバイス ドライバーは、ユーザー空間アプリケーションに IPC サービスを提供します。

Mentor Embedded Multicore Framework は、AMP アーキテクチャにおいて RTOS 環境およびベアメタル環境を Linux の remoteproc/rpmsg インフラストラクチャと相互運用可能にするだけではありません。本フレームワークは、Linux を AMP コンフィギュレーションのリモート OS としてパッケージし、起動するワークフローおよびランタイム インフラストラクチャを備えています。本フレームワークでサポートされる各種の AMP コンフィギュレーションを図 2 に示します。

使用事例とアプリケーション

Mentor Embedded Multicore Framework は監視あり、あるいは監視なし両方の AMP アーキテクチャに適しています。

監視なしの AMP (uAMP) アーキテクチャは、関与する OS コンテキスト間で強固な分離が必要ないアプリケーションで便利です。この場合、関与する OS は、システム内のプロセッサ上でネイティブに実行されます。図 3a に示すように、Mentor Embedded Multicore Framework は、マスター (ブート) プロセッサ上のマスター ソフトウェア コンテキストがライフ サイクルを管理し、SoC 内の他の演算リソースに演算を分散させる、シンプルで効果的なインフラストラクチャを提供します。

監視ありの非対称型マルチプロセッシング (sAMP) アーキテクチャは、ソフトウェア コンテキストの分離とシステム リソースの仮想化が必要なアプリケーションに最適です。sAMP では、関与するゲスト OS は、ハイパーバイザー (仮想マシン モニターとも呼ぶ) によって管理およびスケジュールされるゲスト仮想マシン内で実行されます。ハイパーバイザーは、仮想マシンに分離および仮想化サービスを提供します。Mentor Embedded Multicore Framework により、sAMP アーキテクチャで SoC 内のヘテロジニアス演算リソースの演算を管理できるようになります。

図 3b に示すように、このフレームワークは 2 つの方法で使用できます。まず、ゲスト OS コンテキストからは、ヘテロジニアス演算リソースの監視なしの管理のために使用できます。一方、ヘテロジニアス演算リソースの監視ありの管理を行うハイパーバイザー内からは、ゲスト OS と関連するリモート コンテキスト間の通信をハイパーバイザーで監視できるようになります。

一般的に、Mentor Embedded Multicore Framework は、オンデマンドで、演算機能をマルチプロセッシング チップ上の特定用途コアに負荷分散する必要のあるアプリケーションに適しています。省電力が求められるデバイスは本フレームワークで設計すると、演算リソースをオンデマンドで起動 / シャットダウンして消費電力を最適化できます。

また、本フレームワークで設計すると、旧来のシングルコア エンベデッド システムを、パワフルかつ多機能化なマルチプロセッシング SoC に容易に統合できます。シングルコア シリコン用に開発された旧型のソフトウェアにおいても、少ない労力で移植し、パワフルな新しいマルチプロセッシング チップ上で開発された高度なシステム機能と容易に相互運用できます。

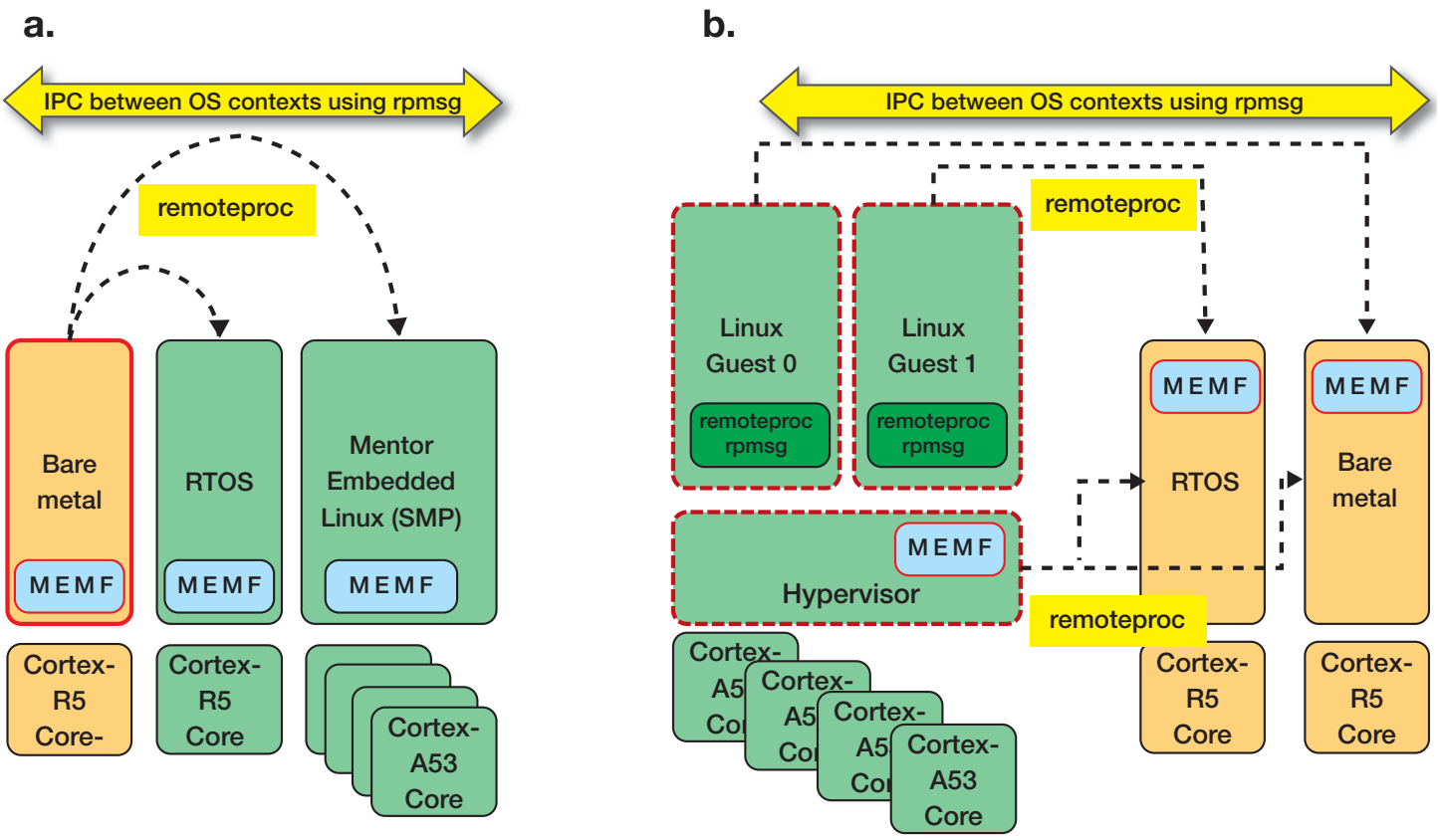


図 3 – Mentor Embedded Multicore Framework の使用例: uAMP (a) および sAMP (b) アーキテクチャ

最後に、Mentor Embedded Multicore Framework では、耐障害性システム アーキテクチャの実装も容易です。たとえば、クリティカルなシステム機能を担う RTOS コンテキスト (マスター) によって、非クリティカルなシステム機能を担う Linux コンテキストを管理することができます。Linux ベースのサブシステムに障害が発生した場合、クリティカルなシステム機能にまったく影響を与えることなく、RTOS は障害が発生したサブシステムを再起動できます。

システム レベルの検討事項

Mentor Embedded Multicore Framework の API により、AMP システムの演算を管理するために必要なソフトウェア インフラストラクチャが提供されます。ただし、AMP システムを設計する際には、API を使用してアプリケーション ソフトウェアを開発する前に、一定のシステム レベルの検討事項について考慮する必要があります。

設計段階の初めに、AMP トポロジを決定します。本フレームワークでは、スター型トポロジ (1 つのマスターで複数のリモートを管理)、チェーン型トポロジ (マスター ノードとリモート ノードを鎖状に接続) が使用できます。

トポロジを選択したら、次は、メモリ配置を決定します。関与する各 OS ランタイムにメモリ領域を割り当て、OS インスタンス間の IPC 用に共有メモリ領域を割り当てます。メモリ配置が完了したら、フレームワークで提供されるプラットフォーム固有のコンフィギュレーション データをアップデートして、選択したメモリ アーキテクチャを反映します。

一般に、既製の OS は SoC 全体の所有を前提としているので、そのままでは監視なしの AMP 環境における動作には適しません。それは共有リソースの共同使用と非共有リソースの排他的使用が重要な要件であるからです。AMP システムに関与する各 OS の設定を変更し、共有リソースを共同使用するように設定する必要があります。たとえば、リモート OS は、マスター コンテキストですでに使用されている可能性がある共有グローバル割り込みコントローラーをリセット、再初期化しないでください。また、競合の原因となるため、共有クロック ツリーや共有ペリフェラルに変更を加えないでください。この設定変更を行うには、通常、関与する OS カーネルまたは BSP ソース、もしくはその両方に対する変更作業が必要です。

次のステップは、システムのパーティショニングです。メモリや非共有 I/O デバイスなどのシステム リソースを、関与する OS 間でパーティショニングして、各 OS が各自に割り当てられたリソースのみを認識し、アクセスできるようにします。そのため、関与する OS が持つプラットフォーム固有のデータ (デバイスおよびメモリ定義) を変更します。たとえば、Linux OS の場合、Linux デバイス ツリー

ソース (DTS) ファイルでメモリおよびデバイス定義を変更します。Nucleus RTOS の場合は、プラットフォーム定義ファイルを変更します。ベアメタル環境の場合、通常、プラットフォーム固有ヘッダーを変更します。

remoteproc による ライフ サイクル管理

システム レベルの設計を決定し、関与する OS の設定を変更した後、アプリケーション ソフトウェアから Mentor Embedded Multicore Framework を使用できるようになります。本フレームワークは、Linux、RTOS、またはベアメタル ベースのソフトウェア イメージを、必要なブートストラップ ファームウェアとともにパッケージし、リモート ファームウェア イメージをELF フォーマットで生成するためのワークフローを提供します。

リモート ファームウェア ELF イメージには、リソース テーブルと呼ばれる特別なセクションがあります。リソース テーブルは静的なデータ構造で、そのリモート ファームウェアが要求するリソースをユーザー指定可能な定義済みの制約を持ちます。リソース テーブルに含まれる主な定義の中には、リモート ファームウェアが要求するメモリや、リモート ファームウェアがサポートする IPC 性能などがあります。マスター ソフトウェア コンテキスト内の remoteproc コンポーネントは、リソース テーブル定義を使用して、リソースを割り当て、リモート コンテキストとの通信を確立します。

フレームワーク マスターは、remoteproc_init API を使用してリモート プロセッサ コンテキストを初期化します。remoteproc マスターは、起動すると、リモート ファームウェア イメージを読み込み、デコードし、リソース テーブルを取得および解析して、リモート ファームウェアのリソース要件を把握します。remoteproc は、リソース テーブル定義に基づいて、リモート ファームウェアに要求される物理メモリを分割し、rpmsg/VirtIO ベースの IPC 固有の初期化機能を実行します。

remoteproc の初期化後、remoteproc_boot API を使用して、関連するソフトウェア コンテキストとともにリモート プロセッサを起動できます。起動すると、ファームウェア イメージがメモリ内に配置され、リモート プロセッサがリセット状態から解放されてイメージを実行します。remoteproc_shutdown API により、アプリケーションはリモート プロセッサをシャットダウンできます。remoteproc_deinit API により、アプリケーションはリモート プロセッサを非初期化できます。(図 5 の擬似コード ブロックは、マスター コンテキストからの remoteproc API 使用例を示しています。)

リモート コンテキストでは、ブート API とシャットダウン API は無関係です。remoteproc コンポーネントを初期化するには remoteproc_resource_init API、非初期化するには remoteproc_resource_deinit API を使用する必要があります。Linux コンテキストにおける remoteproc の使用方法については、Linux カーネルのドキュメンテーションをご覧ください。

RPMSG とプロセッサ間通信 (IPC)

リモート ファームウェアがリモート プロセッサで稼働開始すると、マスター ソフトウェア コンテキストとリモート ソフトウェア コンテキスト間のプロセッサ間通信に rpmsg API を使用できるようになります。rpmsg を使用する上で重要な抽象化と概念を以下にまとめます。

- マスター側から見て、rpmsg デバイスはリモート プロセッサを表す。
- rpmsg チャネルは、マスターとリモート プロセッサ (rpmsg デバイス) 間の双方向通信チャネルである。
- rpmsg エンドポイントは、rpmsg チャネルの任意の側における、論理的抽象化である。
- エンドポイントは、マスター コンテキストとリモート コンテキスト間で対象メッセージを送信するインフラストラクチャを提供する。
- エンドポイントを作成するとき、ユーザーが独自のエンドポイント インデックスを指定するか、rpmsg コンポーネントでエンドポイント用のインデックスを割り当てる。また、ユーザーは、作成したエンドポイントに関連付けられたアプリケーション定義のコールバックを設ける。
- rpmsg は、あるエンドポイント インデックス宛のメッセージを受信すると、受信したデータ ペイロードを参照する、関連付けられた受信コールバックを起動する。
- ユーザーは、rpmsg チャネルの任意の側に、エンドポイントをいくつでも作成できる。
- 特定のエンドポイント インデックスを明示的にあて先としないメッセージは、rpmsg チャネルに関連付けられたデフォルト エンドポイントに到達する。
- rpmsg コンポーネントは、初期化中に登録されたユーザー提供のコールバックを使用して、チャネルの作成や削除などのイベントをユーザー アプリケーションに通知する。

rpmsg チャネルとエンドポイント抽象化とその使用方法を図 4 に示します。rpmsg コンポーネントは、remoteproc と協調動作し、マスター コンテキストとリモート コンテキスト間の rpmsg 通信チャネルを確立、管理します。マスター上の remoteproc がリモート コンテキストを起動すると、リモート コンテキスト上の rpmsg は名前サービス宣言を送信します。名前サービス宣言を受信すると、マスターは、宣言された rpmsg デバイスを登録し、rpmsg チャネルを確立します。チャネルが確立されると、rpmsg チャネルで作成されたコールバックが両側で呼び出され、マスターとリモート アプリケーションにチャネルの作成を通知します。

このとき、マスターとリモート コンテキストは、rpmsg_sendxx API (ブロッキング送信リクエスト) と rpmsg_ trysendxx API (非ブロッキング送信リクエスト) を使用して互いにデータを送信できます。リモート コンテキストが remoteproc_resource_deinit を呼び出すと、rpmsg チャネル削除コールバックによりマスター アプリケーションにイベントが通知され、rpmsg ベースの

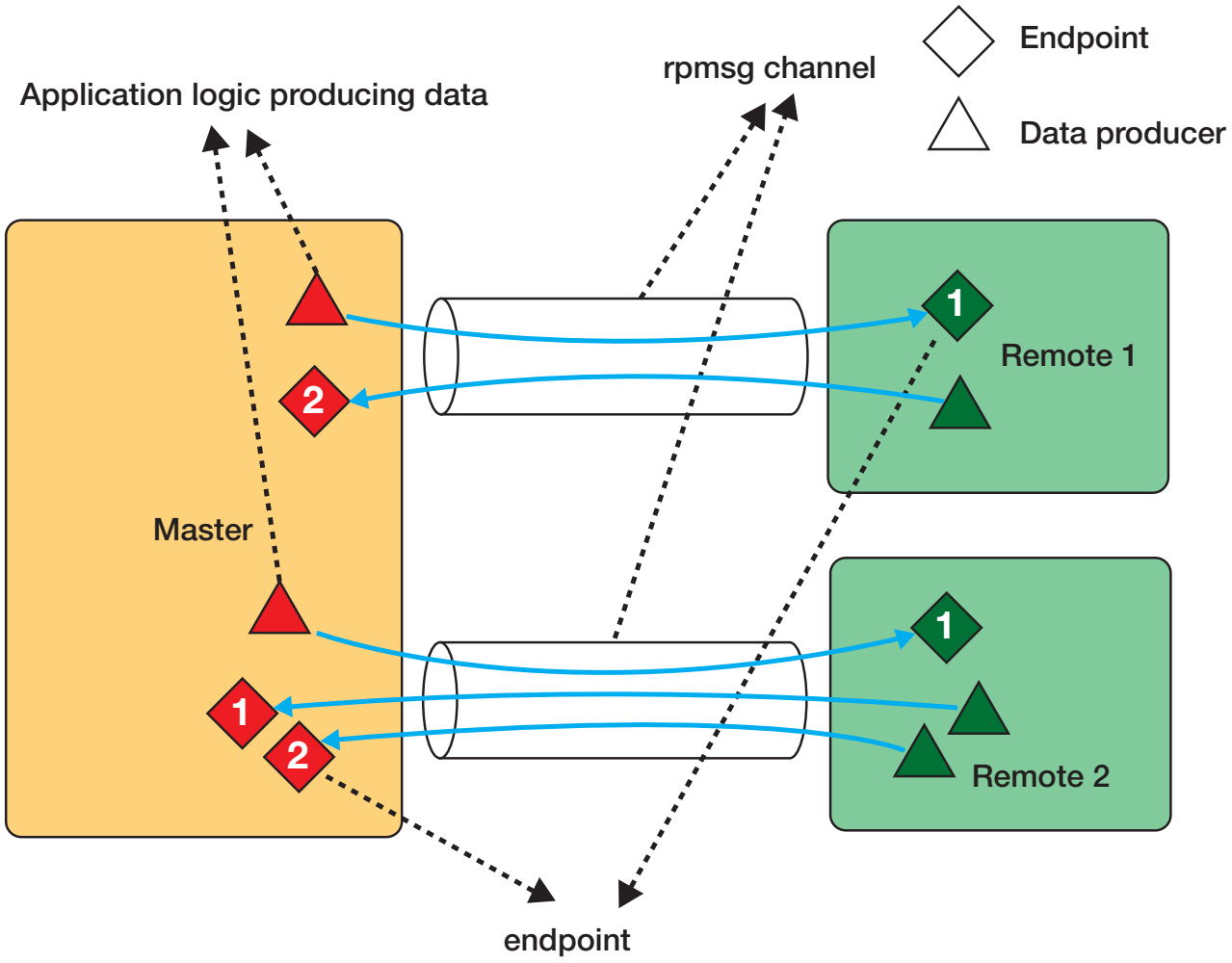


図 4 – rpmsg チャネルとエンドポイント抽象化

通信リンクを正常に遮断できます。リモートコンテキストが応答しなくなった場合、マスターは remoteproc_shutdown API を使用して、リモート プロセッサを非同期でシャットダウンできます。図 5 の擬似コードは、マスター コンテキストにおける、remoteproc API と rpmsg API の使用方法を示しています。

rpmsg コンポーネントは、共有メモリ ベースの伝送の抽象化に VirtIO を使用します。VirtIO は、元々、lguest、KVM、Mentor Embedded Hypervisor におけるゲスト ホスト間通信に使用されていた I/O 仮想化規格でした。rpmsg ドライバーは、VirtIO 層が提供するサービスを、相手側との共有メモリ ベースの通信に使用します。rpmsg ドライバーは、rpmsg VirtIO デバイスをインスタンシエートし、VirtQueue インターフェイスを介して通信先とのデータをプッシュおよび消費します。

AMP システムの開発ツール

AMP アプリケーション ソフトウェアの開発には、AMP システムならではの課題があります。通常、ヘテロジニアス SoC 上の異種のプロセッサ上にインストールされた異なる OS 環境を同時にデバッグする必要が生じます。関与する各 OS を認識可能な統合デバッグ環境があれば、デバッグの環境が向上するだけでなく、生産性も上がります。Mentor Embedded Sourcery CodeBench ツールは、サポート対象のすべての OS 環境 (Mentor Embedded Linux と Nucleus RTOS を含む) を認識可能な、統合 IDE を提供します。Sourcery CodeBench は、さまざまなデバッグ オプションもサポートしています。Linux カーネル空間、Nucleus RTOS、ベアメタル コンテキストをデバッグするための JTAG ベースのデバッグ、Linux ユーザー空間と Nucleus RTOS ベースのアプリケーションをデバッグするための GDB ベースのデバッグなどです。

AMP システムの開発において、ソフトウェアのプロファイリングは、ヘテロジニアス OS 上に配置したさまざまなアプリケーションのランタイム中の相互作用を予測するのに役立つツールです。各 OS インスタンスは、通常、独立したクロック基準を使用するため、ある

```
/* rpmsg channel created callback - invoked on channel creation */
void rpmsg_channel_created(struct rpmsg_channel *rp_chnl) {
    ..
      /* Use RTOS provided primitives (ex., semaphores) to
    release the application context blocked on rpmsg
    channel creation */
}

/* rpmsg channel deletion callback - invoked on channel deletion */
void rpmsg_channel_deleted(struct rpmsg_channel *rp_chnl) {
    ..
      /* Use RTOS provided primitives (ex., semaphores) to
    release the application context blocked on rpmsg
    channel deletion */
}

/* rpmsg receive callback - invoked when data received on
default endpoint */
void rpmsg_rx_cb(struct rpmsg_channel *rp_chnl, void *data,
     int len, void * priv, unsigned long src) {
    ..
      /* Copy received data to application buffer and use
      RTOS provided primitives (ex., semaphores) to release
      the application IPC data processing context */
}

/* Initialize remote context */
int Initialize_Remote_Context(..) {
    ...
    /* Initialize remote context */
    remoteproc_init(remote_fw_info,
        rpmsg_channel_created, rpmsg_channel_deleted,
        rpmsg_rx_cb, &proc);

      /* Boot remote context */
    remoteproc_boot(proc);
      ...
}

/* Send data to remote context after rpmsg channel creation */
int Send_Data_To_Remote(..) {
    ..
      rpmsg_send(app_rp_chnl, user_buff, sizeof(user_buff));
      ..
}

/* Finalize remote context after rpmsg channel deletion */
int Finalize_Remote_Context(..) {
    ..
      /* Shut down and finalize the remote processor */
      remoteproc_shutdown(proc);
      remoteproc_deinit(proc);
      ..
}
```

図 5 – マスター コンテキストからの主要な remoteproc API と rpmsg API の使用方法を示す擬似コード

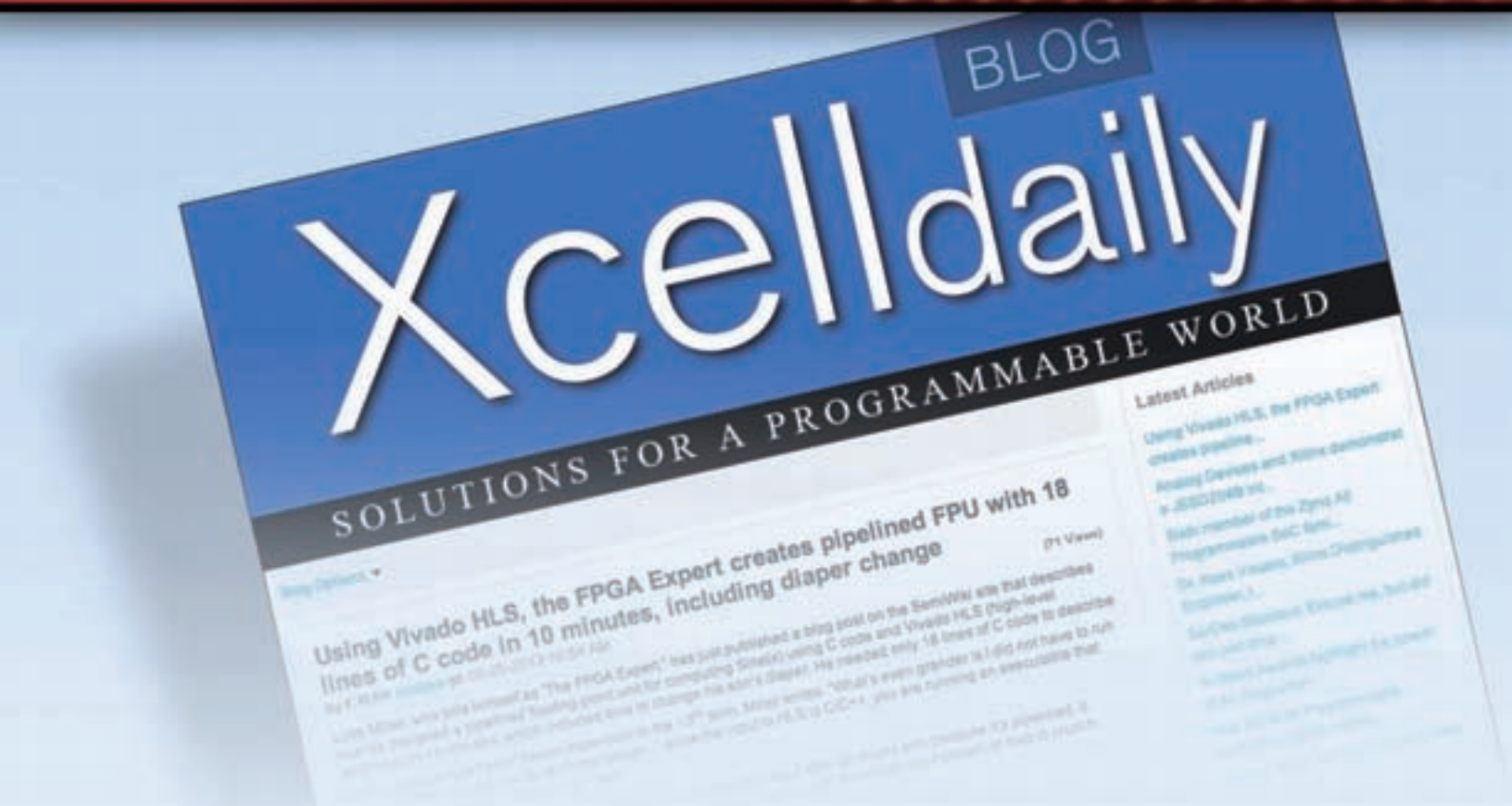
OS のコンテキストで収集したプロファイリング データは、その OS のローカル タイムが基準になります。Mentor Embedded Sourcery Analyzer のホスト ベースのツールと Mentor の OS は、異種の OS ソースから収集したトレース データを、統一した時間基準でグラフィカルに表示して解析可能なアルゴリズムを内蔵しています。この機能により、AMP ソフトウェア開発中によくある、複雑な相互作用や見つけづらいタイミング問題について、興味深い情報を得ることができます。

オープンソースおよびランタイム コンポーネント

Mentor Embedded Multicore Framework は、Mentor の開発ツールおよび OS と緊密に統合されています。また、幅広い ARM ベースの SoC およびプラットフォームをサポートしています。このフレームワークを Mentor のツールおよび OS と組み合わせて使用することで、AMP システムをゼロから設計する必要はなくなります。つまり、「システム レベルの検討事項」のセクションに記載した作業は不要になります。リファレンス コンフィギュレーションから任意のデザインを選んで AMP アプリケーション開発を開始し、後からニーズに合わせてシステム コンフィギュレーションをカスタマイズできます。

AMP システム設計には、オープンソース Linux コミュニティに採用されているインターフェイスと相互運用可能な RTOS またはベアメタル ソフトウェアを開発可能な、規格ベースのソフトウェア フレームワークへのニーズが明らかに存在しています。このニーズに応え、業界への普及を促進するため、Mentor Graphics とザイリンクスは共同で、Zynq-7000 All Programmable SoC プラットフォームをサポートする Mentor Embedded Multicore Framework のランタイム コンポーネントを OpenAMP オープンソース プロジェクトの下でオープンソース化しました。 このプロジェクトは、現在、Mentor Graphics とザイリンクスが共同で管理しています。

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）

- *Use C/C++ to Offload Image Processing to Programmable Logic*
- *EE Journal's Kevin Morris writes some choice words about high-level synthesis and the Xilinx Vivado HLx introduction*
- *Xcell Software Journal Issue #2 now online, ready for you to read and download*
- *Bunnie Huang's Novena Open-Source Linux Laptop incorporates a User-Programmable Spartan-6 FPGA*
- *Auviz offers free eval version of its high-performance OpenCV Computer Vision Library for Xilinx All Programmable devices*

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

XILINX
ALL PROGRAMMABLE™