WP540 (v1.0)

Kria ロボティクス スタック:

ロボティクスにおける ROS 2 中心の設計アプローチによるハードウェア アクセラレーション

著者: Víctor Mayoral-Vilches

ハードウェア アクセラレーションがもたらすロボティクスの革命は、ロボットの応答の高速化、消費電力の削減、アーキテクチャのセキュリティ強化を可能にしています。SDK に ROS 2 を使用して構築されるロボット チップと、SoC を使用して構築されるロボット専用回路の未来について説明します。

概要

Kria™ ロボティクス スタック (KRS) には、ハードウェアを使用して、インダストリアル グレードのロボット ソリューションの開発、メンテナンス、商用展開を加速するためのロボット ライブラリおよびユーティリティが統合されています。KRS はソフトウェア開発キット (SDK) として ROS 2 を採用し、計算グラフの作成から、ザイリンクス アプリ ストアで提供する ROS 2 オーバーレイ ワークスペースの商用化まで、ROS 2 を中心とする開発アプローチを提案します。Kria SOM をターゲットとする KRS は、量産対応、高スループット、低レイテンシ、リアルタイム性でロボットの価値を向上させます。ロボット技術者は、KRS が提供する機能を使って独自のセキュアなコンピューティング アーキテクチャを開発でき、それらをザイリンクス アプリ ストアで商用化することもできます。KRS を入口として、ROS 2 ロボット技術者はハードウェア アクセラレーションの世界を体験できます。

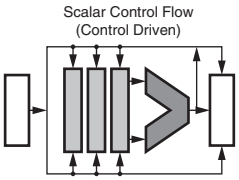
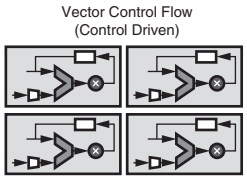
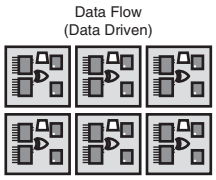
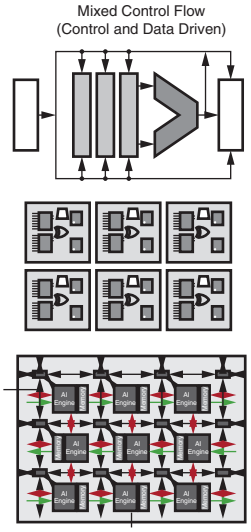
ザイリンクスは、従業員、顧客、パートナーそれぞれが認められ、受け入れられていると実感できる環境作りに取り組んでいます。その一環として、ザイリンクス製品資料およびその他関連資料に含まれる非包括的な用語/表現を排除していきます。ザイリンクス ソフトウェアおよび IP に組み込まれている用語を含め、人を差別、疎外したり、歴史的な偏見を増長する可能性のある表現をなくすための社内的取り組みが始まっています。該当表現を改め、進化する業界標準に則った取り組みを進めていますが、旧製品資料には配慮に欠ける表現が残っている可能性があります。詳細は、このリンクを参照してください。

はじめに

ムーアの法則が限界を迎えると共に、ハードウェア アクセラレーションに特化した演算ユニットがロボットの性能向上の鍵となることが実証されています [参照 1]。このような専用コンピューティング アーキテクチャは、「高速演算」(ひとたび開始されれば処理を迅速に完了する) 動作と「リアルタイム」(各タスクに設定された期限を守る) 動作を得るためのロボット アルゴリズムに合わせた特定のハードウェア、すなわち、適応性のあるシステム オン チップに依存します。これはハードウェア アクセラレーションがもたらす結果です。中心となるアイディアは、ロボティクスに使用される従来の制御駆動型アプローチと、ハードウェア リソース量を最適化するデータ駆動型アプローチを組み合わせることで性能が向上するということです [参照 2]。またこれにより、より電力消費効率が高く、確定的で、セキュアなハードウェア構造が可能となります。

表 1 では、ロボットのさまざまな演算モデルを比較し、制御駆動型とデータ駆動型を統合したモデルによってどのように最良のトレードオフが得られるかを示します。ただし、専用ハードウェアの開発プロセスには、複雑性というコストが伴います。

表 1: ロボットのハードウェア演算モデルの比較

演算モデル	スカラー制御フロー	ベクトル制御フロー	データフロー	複合型 (制御駆動とデータ駆動)
				
テクノロジ	CPU	DSP、GPU、AI エンジン	FPGA	アダプティブ SoC (CPU + FPGA)、 ACAP (CPU + FPGA + AI エンジン)
定義	フォンノイマン型スカラー プロセッサ。従来型の演算。制御トークンが、ステートメントをいつ実行する必要があるかを示す。	フォンノイマン型ベクトル プロセッサ。ベクトル化された単一の命令を実行するための制御トークン。固定されたプロセッシング アーキテクチャおよびメモリ アーキテクチャ。	柔軟な処理構造。先行評価。ステートメントはデータが利用可能になるとすぐに実行される。	適応型の処理構造。各種のプロセッサが混在する、制御駆動とデータ駆動を組み合わせた演算モデルを構築できる
プログラミング性能	SW でプログラム可能	SW でプログラム可能	HW レベルでプログラム可能	SW でも HW レベルでもプログラム可能
長所	1. 完全な制御 2. データと制御の複合構造を簡単に実装できる。	1. 完全な制御、 2. ドメイン特化型の並列処理 (数値演算、ビデオ、画像処理など)	1. さまざまに応用できる並列処理性能 2. 高スループット 3. 確定的: 副次的作用がない	1. 完全な制御機能 2. データと制御の複合構造を簡単に実装できる 3. さまざまに応用できる並列処理性能 4. 高スループット 5. 確定的 6. ドメイン特化型の並列処理
短所	1. 非同期イベント (割り込みによって演算サイクルが切替わる) の管理効率が低い 2. 制御パスが固定されているため、並列処理のスケーラビリティ (CPU コア数) と実行性能が制限される	1. 固定されたメモリ アーキテクチャと演算アーキテクチャ (効率のボトルネック) 2. GPU はロボットなどの物理的システムに特有のリアルタイム データ ストリームを直接処理できない 3. DSP は専用の命令を実装するが、周波数/性能のトレードオフがある	1. 開発に時間がかかる 2. プログラムと設計が難しい 3. データフローが制御フローよりも優先して扱われる	1. 性能をフルに引き出すには、さまざまなタスクを各種演算ユニットに (通常は手動で) 割り当てる必要がある 2. 各種演算ユニット間の協調が必要 (いつ、どのように、どこでデータを取得するか)

複雑性は、ロボットには古くからある問題です。平均で、博士課程の学生が基本的な概念に慣れるには 1 ～ 2 カ月かかり、ロボット アプリケーション開発の事実上の標準であるロボット オペレーティング システム (ROS) [参照 3] を使用して複雑な計算グラフを構築するには数年かかります。ROS の重要性と、ROS を使用したハードウェア アクセラレーションの過去の試みについては、『ロボティクスにおけるアダプティブ コンピューティング: ROS 2 に対応した FPGA 用ソフトウェア定義ハードウェアの利用』(WP537) [参照 2] で説明しています。

複雑性に加えて、ロボット アーキテクチャの構築で重要なもう 1 つの要素はリアルタイム性です。ロボットは本質的に確定的なマシンです。ロボットはネットワークのネットワークです。これらのネットワークは全体として、確定的なタイミングでの情報交換を容易にする、ロボットの神経系として理解されます。人の神経系と同じように、これらすべてのネットワークのリアルタイム性は、ロボットが整合性のある動作をするための基本条件です。したがって、プロセス内 (同じプロセス内のロボット計算ノード)、プロセス間 (同じコンピューター内のプロセス間の計算ノード)、およびネットワーク内の各レベルで、リアルタイム性を確保する必要があります。CPU と GPU は固定されたフォン ノイマン型のアーキテクチャを持つため、制御フローの処理は得意ですが、確定性の確保には苦戦します。さらに、これらの固定された演算ユニットを使用する場合、設計者は一般的に高スループットと低レイテンシの間で選択を強いられます。

このホワイト ペーパーでは、Kria™ ロボティクス スタック (KRS) を紹介します。KRS には、インダストリアル グレードのロボット ソリューションの開発、メンテナンス、商用展開を加速するためのロボット ライブラリおよびユーティリティが ROS 2 と緊密に統合されています。KRS は、ハードウェア アクセラレーションによって低レイテンシ (高速演算)、確定性 (予測性)、リアルタイム (オンタイム)、セキュリティ、高スループットを提供するように設計されています。KRS は ROS 2 へのバインド構造を提供し、CPU と GPU のヘテロジニアス プラットフォームでプログラムを実行できるようにします。これには演算アクセラレーション用の FPGA 高位合成 (HLS) も含まれます。また、Kria SOM ポートフォリオおよびリファレンス開発ボードをプラットフォームとして利用し、独自のセキュアなコンピューティング アーキテクチャを作成できます。

KRS は、ロボットのネットワーク全体にわたるリアルタイムの必要性を意識しながら、ROS 2 でのハードウェア アクセラレーションを容易にするように設計されています。ROS 2 に関するこれまでの研究で、OSI スタックの時間的ボトルネックおよび制限が特定されています [参照 4] [参照 5] [参照 6] [参照 7] [参照 8]。これらすべての課題をまとめて解決し、分散された ROS 2 計算グラフの確定性を確保するために、KRS のコンポーネントは次の 3 つのカテゴリで提供され、設計者はアプリケーションの要件を満たすために必要なコンポーネントを選択できます。

- **ROS 2 アプリケーションおよびライブラリ**: このグループは、ROS 2 上の OSI L7 アプリケーションまたはライブラリを加速するアクセラレーション カーネルに対応します。ROS 2 上のあらゆる演算は、このカテゴリの良い候補となります。例として、図 1 に示すように、ナビゲーション スタック、マニピュレーション スタック、パーセプション スタック、または制御スタック内の一部のコンポーネントが挙げられます。
- **ROS 2 コア**: これには、サポートされる各通信ミドルウェアについて、OSI L7 ROS 2 コンポーネントおよびツールを (たとえば FPGA またはアダプティブ SoC 内の) ロボット専用回路で高速化またはオフロードするカーネルが含まれます。`rclcpp`、`rcl`、`rmw`、および対応する `rmw_adapters` に焦点が当てられます。(1) 例として、FPGA でオフロードされる、より確定的な動作のための ROS 2 エグゼキューター [参照 9] [参照 10] や、完全にハードウェアでオフロードされる ROS 2 ノード [参照 11] が挙げられます。
- **ROS 2 下層**: このカテゴリは、ROS 2 コア レイヤーより下層にある、OSI L2 ～ L7 に属するすべてのアクセラレータ (通信ミドルウェアを含む) をまとめたものです。これらのタイプのアクセラレータの例として、FPGA でアクセラレーションされる DDS 実装の全体または一部、オフロードされるネットワーク スタック、またはリアルタイムの確定性、低レイテンシ、および高スループットの相互作用を実現するデータ リンク層が挙げられます。

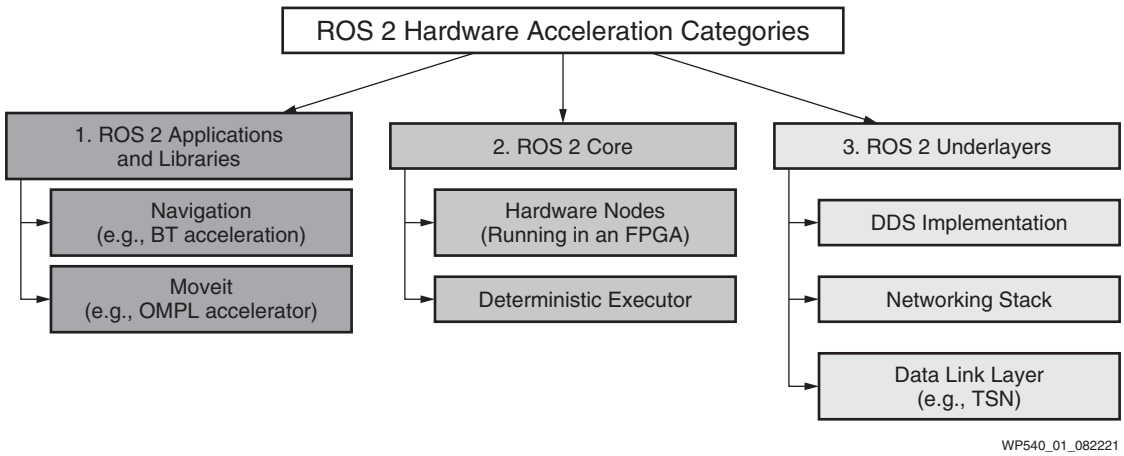


図 1: ROS 2 のハードウェア アクセラレーション パスと例

1. 通信ミドルウェアそれ自体は、ROS 2 コア アクセラレータのカテゴリの一部とは見なされません。これは、下位の通信ミドルウェアには依存しないという ROS 2 デザインの原則との整合性を維持するためです。したがって、任意の通信ミドルウェア (DDS など) は、次のカテゴリ (ROS 2 下層) に属します。

このホワイト ペーパーはこれ以降、次のような内容で構成されています。「Kria ロボティクス スタック」では、デザインにおける選択と機能を含めて、KRS についてより詳しく説明します。これには、ROS 2 を中心とする開発手法、ROS 2 アクセラレータの商用化を容易にする KRS 複合ソース エコシステム、およびリアルタイム性に関する考慮事項が含まれます。「ユース ケース」では、KRS の 2 つのユース ケースとして、ROS 2 アプリケーションおよびライブラリのアクセラレーションと、ROS 2 下層の最適化による確定性の確保について説明します。「ザイリンクス アプリ ストア」では、ロボット技術者が開発したアクセラレーション ROS 2 パッケージの保護と収益化を、KRS がどのように支援するかについて詳しく説明します。

Kria ロボティクス スタック

KRS は、産業用の ROS 2 スーパーセットで、ソフトウェア定義ロボティクス ソリューションの開発を支援するツールとライブラリを提供します。KRS はロボット技術者とザイリンクスとの橋渡しとなり、ハードウェア アクセラレーションを簡単に使えるようにします。KRS のコンポーネントは、アプリケーションと AI、ミドルウェア、OS とハイパーバイザー、シリコン アーキテクチャ、および安全性(1) とセキュリティの 5 つのグループに分類されます。(2) 図 2 に KRS スタックを示します。以降のサブセクションでは、KRS ベースのデザインにおける重要な選択について説明します。

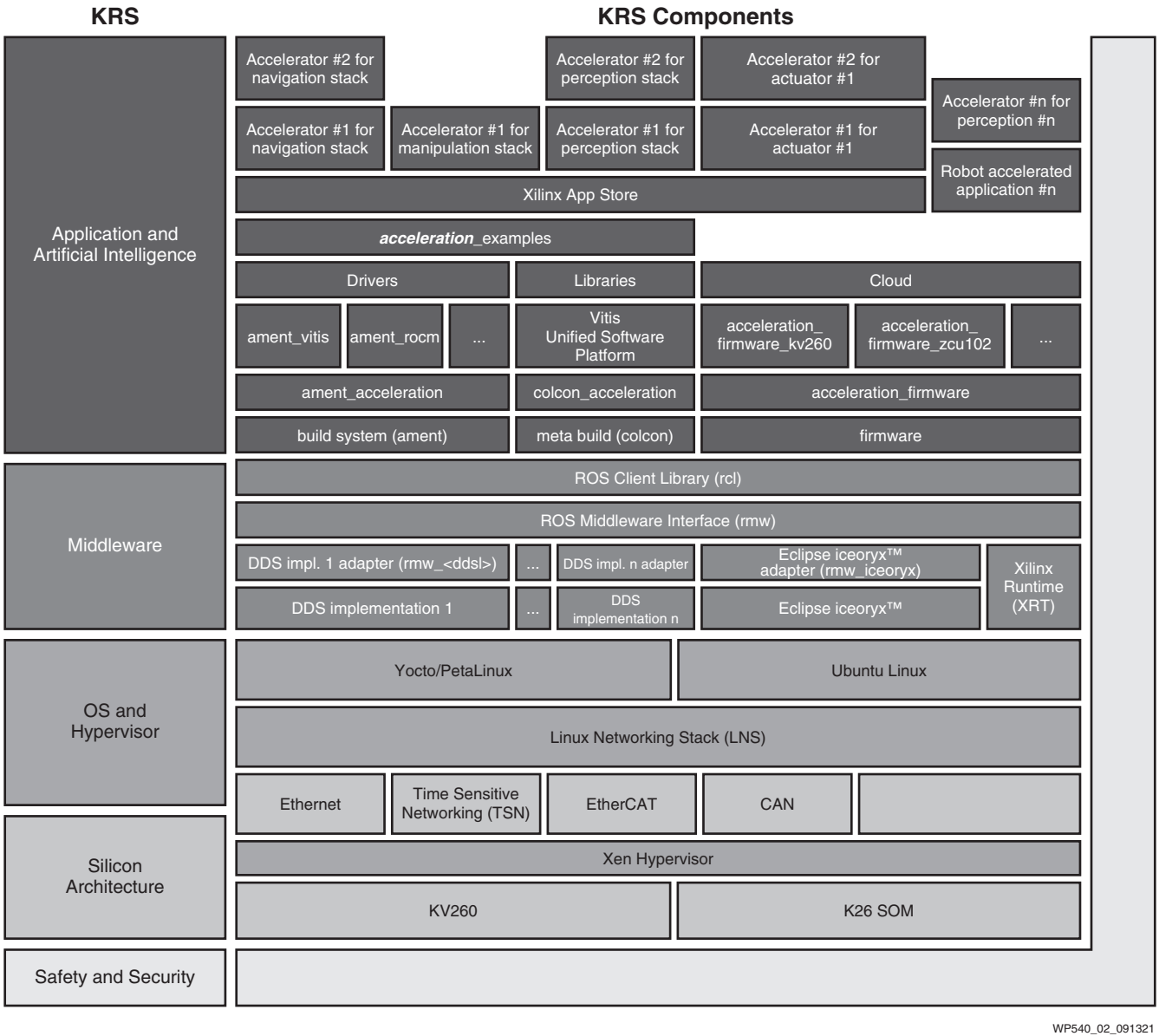


図 2: Kria ロボティクス スタックのツールとコンポーネント

1. https://www.xilinx.com/publications/solution-briefs/xilinx-functional-safety-solution-brief.pdf
2. 安全性とセキュリティの詳細は、[参照 12] [参照 13] を参照してください。

すべてのロボット技術者のための ROS 2 中心型アプローチ

ロボット技術者にとっての ROS は、多くのコンピューター科学者やソフトウェア開発者にとっての Linux と同じです。ROS 2 は ROS の機能を強化し、ロボットの動作を量産対応システム用に改善します。KRS を使用すれば、ライブラリの複製や類似のシミュレータが多数作成されるような手法でアーキテクチャを新規に再設計しなくても、ROS ロボティクス コミュニティの要求を満たすことができます。KRS は、KRS に緊密に接続されるロボティクス シミュレータ Gazebo [参照 14] と共に、ROS 2 上に構築されます。

図 2 に、ROS 2 レイヤー (`rcl`、`rmw` など) がどのようにミドルウェア グループ内で KRS アーキテクチャの中心部に位置するかを示します。その周囲には、ROS 2 の性能を向上させてロボットの相互作用の確定性と性能を高めるレイヤーが配置されます。

ROS 2 エコシステムに対するザイリンクスの取り組みの一環として、ザイリンクスが主導する ROS 2 ハードウェア アクセラレーション ワーキング グループ (HAWG) が発足しました。このワーキング グループの目的は、各種の演算ユニット (FPGA、アダプティブ SoC、GPU など) 上での ROS 2 および Gazebo の相互作用の最適化のために、オープン規格 (C++ および OpenCL) でのアクセラレーション カーネルの開発、メンテナンス、テストを促進することです。コミュニティの発表で具体的に述べたように、このグループは次のハードウェア アクセラレーションをターゲットにしています。

- エンベデッド (エッジ) デバイス
- ワークステーション
- データセンター
- クラウド

HAWG の計画では、まず最初に、ロボット技術者が自分のデザインのブループリントとして使用できるアプリケーション層の簡単な例を使用して、エッジ デバイス上のアクセラレーションのデモを示します。次に、下層をターゲットとして、ROS 2 計算グラフ内のノード間の相互作用を最適化します。その後は方向転換して、アプリケーション レベルを高速化するカーネルを開発します。最初はパーセプション、アクチュエーション、ナビゲーション、マニピュレーションを考慮に入れます。

図 3 に、ザイリンクスによる HAWG への貢献として ROS 2 の機能強化を示します。この作業では、ROS 2 ビルド システム (`ament`) の中核となるハードウェア アクセラレーションとメタ ビルド ツール (`colcon`) をテクノロジに依存しない方法で統合し、将来は AI エンジンや GPU を含むほかのテクノロジにも拡張できるようにします。ロボット専用回路の生成プロセスは、ホスト コードのクロスコンパイル、合成、およびアクセラレータの実装を自動化する特定のフラグを持つ `colcon` を使用して、ROS 2 ワークスペースから直接制御およびトリガーできます。したがって、ROS 2 ロボットのカスタム ハードウェア アクセラレータを生成する際に、ベンダー固有のツール (Vitis® または Vivado® ツールなど) との相互作用は不要です。生成されたカーネルは、ROS 2 オーバーレイ ワークスペース内に含まれます。これをパッケージにして、ロボットへ直接組み込むことができます。

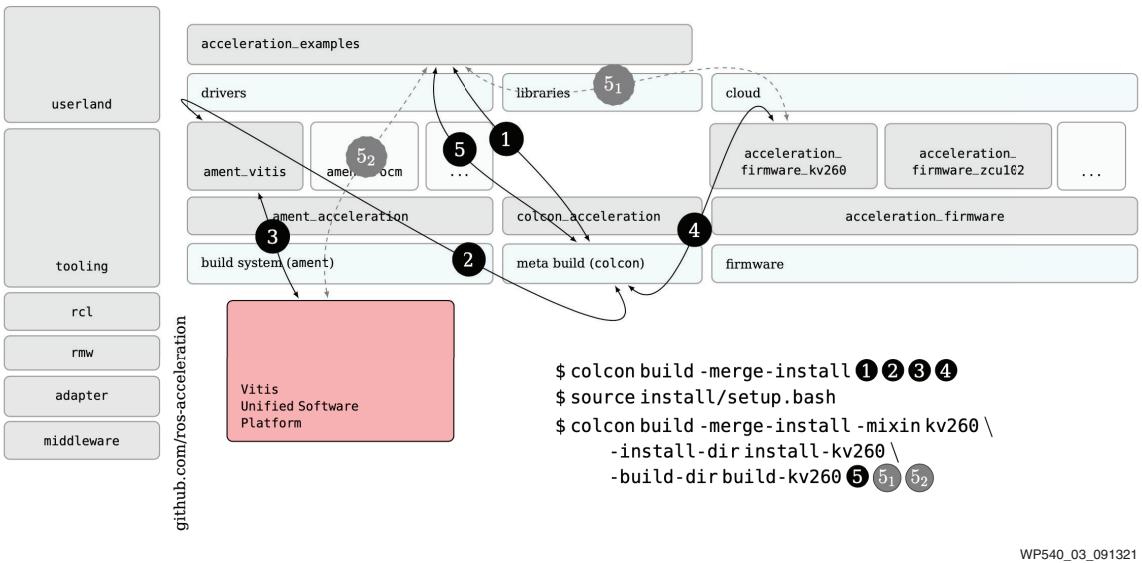


図 3: HAWG の初期アーキテクチャ内での ROS 2 パッケージとの相互作用

ament_vitis ROS 2 パッケージにより、ロボット技術者は使い慣れたソフトウェア駆動型フローでアクセラレータを実装できます。ザイリンクスのテクノロジは、次の 3 つのビルド ターゲットを使用して生産性を向上させます。そのうち 2 つは、開発を迅速化する QEMU エミュレーションに対応しています。

- **ソフトウェア エミュレーション (sw_emu):** カーネル コードがホスト プロセッサ上で動作するようにコンパイルします。これを使用すると、ビルドと実行を短時間で反復して、アルゴリズムを段階的に改良できます。このターゲットは、構文エラーを特定したり、アプリケーションと共に実行しながらカーネル ソース コードのソース レベル デバッグを実行したり、システムの機能を検証したりする場合に役立ちます。このビルド ターゲットが提供する変換機能により、すべてのコードは、アクセラレータが存在しないかのように、ターゲット ハードウェア (KV260 など) に合わせてエミュレートされるプロセッサ上で動作します。
- **ハードウェア エミュレーション (hw_emu):** カーネル コードをハードウェア モデルにコンパイルした後、専用のシミュレータで実行します。この場合、ビルドと実行の反復にはやや時間がかかりますが、カーネルの動作をサイクル精度で詳細に確認できます。アダプティブ SoC のプログラマブル ロジックの機能をテストしたり、最初の性能を見積もる場合に役立ちます。ターゲット ハードウェア (KV260 など) のエミュレーション環境で、アダプティブ SoC のシミュレーションができます。
- **ハードウェア (hw):** カーネル コードをハードウェア モデルにコンパイルした後、ターゲットとなるアダプティブ SoC にインプリメントします。これにより、実際のアダプティブ SoC で動作するバイナリが生成されます。

acceleration_examples メタパッケージで提供されるいくつかの ROS 2 パッケージ サンプル (sw_emu など) は、ビルドに約 1 分かかります。hw_emu は 4 分、hw は約 23 分かかります。(1)

KRS は SDK に ROS 2 を使用して構築されていますが、すべてのロボット技術者に役立つように、ほかのロボティクス フレームワークを使用した代替フローや、ロボティクス フレームワークを使用しないフロー (ザイリンクス Vitis 統合ソフトウェア プラットフォームを直接使用する場合など) もサポートします。

専門的用途での利用を促進する複合ソース エコシステム

ROS エコシステムは、ROS の抽象化を使用しながらロボット アプリケーションの開発に取り組んでいる数千人のロボット技術者の世界的なコミュニティを 1 つにまとめます [参照 2]。ある意味で、ROS はロボット技術者がロボットの動作を構築するときに使用する共通の API であり、ロボティクスのリファレンス SDK となっています。ロボティクスにおける複合ソース テクノロジ エコシステムが登場し [参照 15]、コミュニティに貢献しながらオープンソース パッケージに関連する価値を提供している企業のさまざまな例が既に見られます。図 4 に、KRS で提供される複合ソース エコシステムと、ザイリンクスの貢献によるさまざまな成果を示します。

1. AMD Ryzen 5 PRO 4650G 搭載ワークステーション上でビルドを実行した場合です。

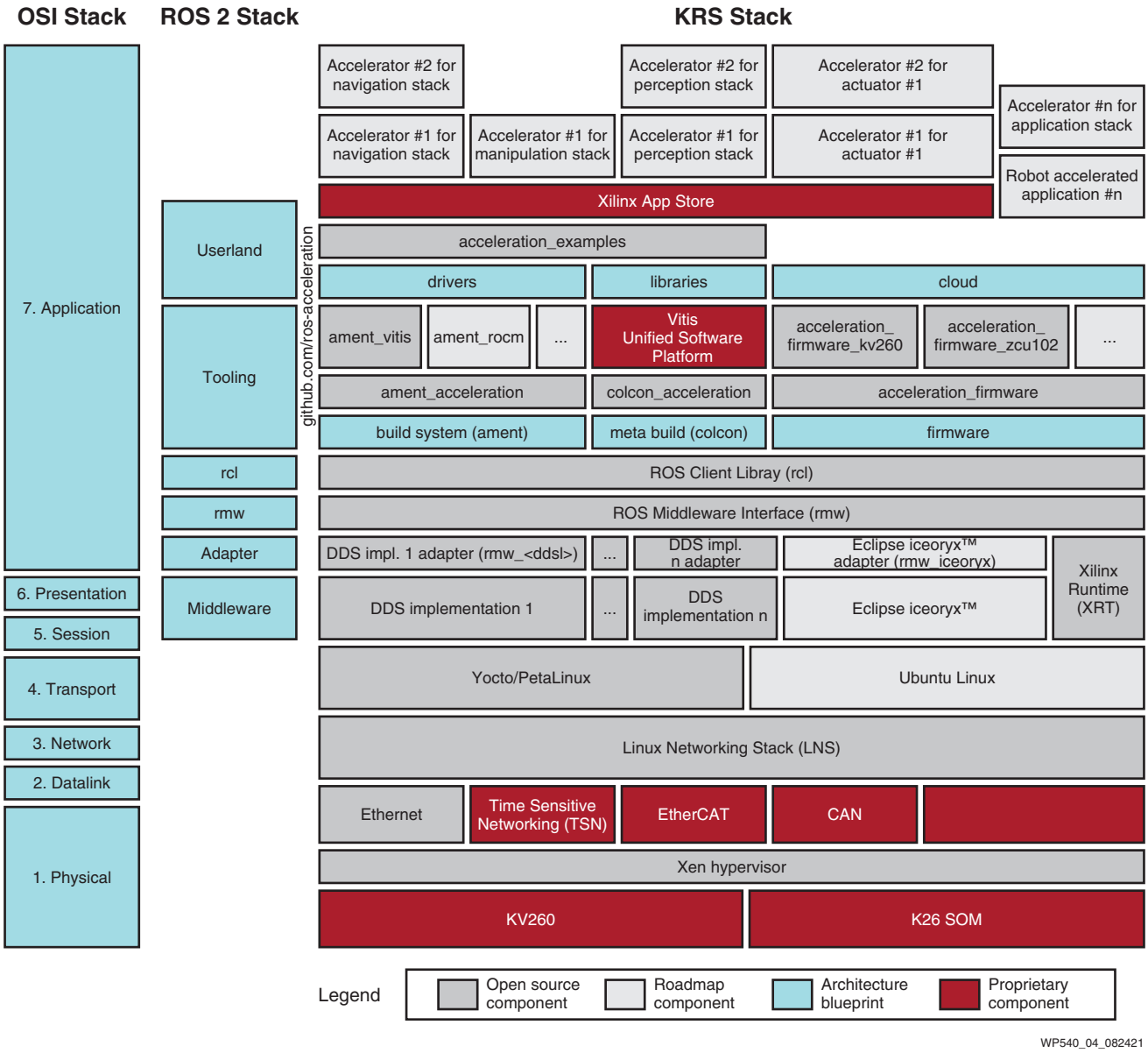


図 4: OSI および ROS 2 スタックに関連する KRS 複合ソース エコシステム

KRS は、ROS ロボット アクセラレーション アプリケーションの商用化のためのレイヤーを提供します。ザイリンクス アプリストアは、暗号化と認証による FPGA ビットストリームのセキュリティ確保に関するザイリンクスの経験を応用して、適切に管理された、使いやすい、セキュアなデジタル著作権管理 (DRM) 用インフラストラクチャを提供します。アプリ ストアでアプリケーションを商用化し、世界中のカスタマーの目に留まるようにすることができます。ザイリンクス アプリ ストアを通じて、KRS は ROS 2 オーバーレイ ワークスペースをロボット アクセラレーション アプリケーション内にコンテナー化し、パートナーが開発した ROS 2 アクセラレーション パッケージの保護と収益化を可能にします。

KRS の中核となるリアルタイム性

リアルタイム性は、ロボット システムのエンド ツー エンドの特性です。スカラー プロセッサ (すなわち、CPU) 上で動作する ROS 2 アプリケーションには、さまざまな不確定性の発生源があります。図 5 に、OSI スタック内の不確定性の発生源を示します。ロボットが確定的なタイミングで応答するには(1)、スタック (L1 ～ L7) 内のすべてのレイヤーが整合性のある応答をすしなければなりません。不確定性の発生源には、それぞれのレベルごとに対処する必要があります。

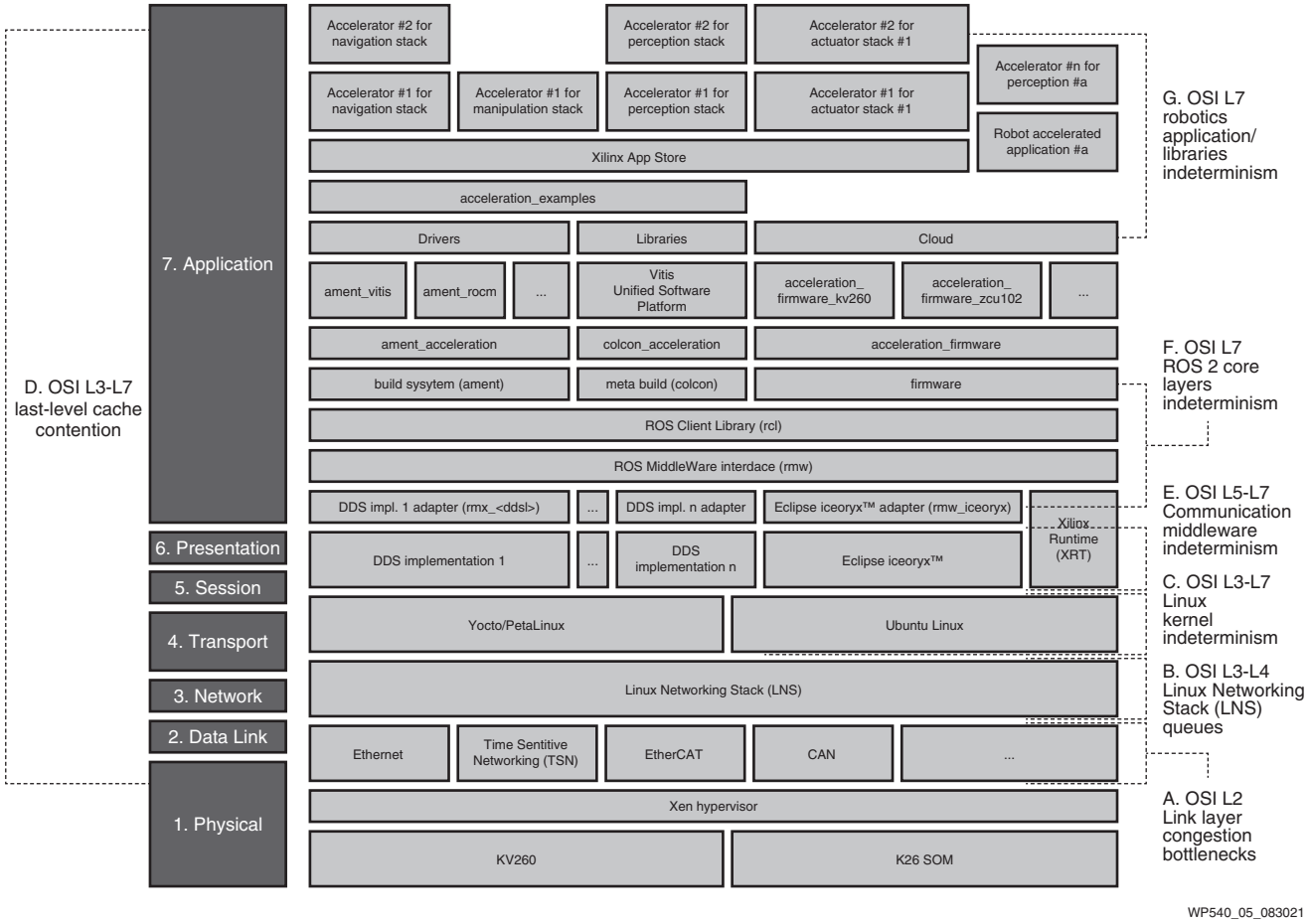


図 5: ROS 2 スタック内の不確定性の発生源

一般的に、CPU のリアルタイム性の問題に対する解決策は、1) 対応する抽象化層で適切な優先度を設定する、2) サービス品質 (QoS) の手法を適用する、の 2 つに大きく分類されます。各レイヤーに独自の QoS 手法があります。レイヤー 2 は、既存の IEEE Std 802.1Q 規格で規定された QoS 手法と、TSN (Time-Sensitive Network) 規格などの新しい手法を使用します。Linux ネットワーク スタックでは、トラフィック制御を使用して QoS 手法を設定します。同様に、リアルタイム性能を実現するには、Linux カーネルから上位のアプリケーション ライブラリまでの各レイヤーを、ロボット システムの最大レイテンシの制約条件に応じて設定する必要があります。

KRS の目的は、モジュラー アプローチによってこれらの不確定性の問題を軽減するメカニズムを提供することです。一部のモジュールに高い優先度を与え、それらを使用して目的の不確定性の発生源を解消して、各ユース ケースのタイミング要件を緩和できます。以降のセクションでは、不確定性の発生源について分析し、KRS に統合されたソリューションについて簡単に説明します。図 5 には、次に示す原因と紐付けて A、B、C などの文字を示しています。

1. リアルタイム性 (確定的なタイミング、時間どおりであること) と高速演算 (低レイテンシ) の違いに注意してください。

A. リンク層の輻輳によるボトルネック: データ リンク層のボトルネック (OSI L2) に関連するリアルタイム性の問題は、ロボット システムにおける不確定性の発生源です。最も一般的なシナリオは、優先度が異なるさまざまなトラフィック ソースが同じネットワーク セグメント内に存在する場合です。たとえば、環境の点群データを生成している各種の深度センサーのデータがスイッチド イーサネット ネットワークで送信され、より優先度の高いトラフィック (アクチュエータに命令するトラフィックなど) と混在する場合、輻輳の問題が発生する可能性があります。これが原因で望ましくない不確定性が生じ、タイミングがクリティカルなパッケージでパケットの欠落につながることもあります。KRS は、イーサネット上で確定的なリアルタイム通信を可能にする、タイム センシティブなリンク層コンポーネント (TSN など) を提供することにより、これらの問題に対処します。

データ リンク層の不確定性のもう 1 つの発生源は、異なる通信バス間での相互運用中に発生します。たとえば、イーサネットで 認識したセンシング データをEtherCAT フィールド バスへ中継する場合や、アノテーション付きのデータをイーサネットから CAN へ渡す場合がこれにあたります。これらのネットワーク間のインターフェイスは、スカラー プロセッサでは通常は上位層 (OSI L7 など) で処理されます。そのため、追加の処理サイクルを多数消費するのに加え、データが上位抽象化層の不確定性の影響をさらに受けます。アダプティブ SoC を利用することで、KRS によってデータ リンク層でデータを処理しながらネットワーク間の通信を橋渡しする専用回路を作成する機能が提供されます。

図 5 に、ROS 2 スタックにおける不確定性の発生源を示します。KRS は、ロボット アプリケーションにおける無制限のレイテンシの発生源を軽減するためのメカニズムを提供します。

B. Linux ネットワーク スタック (LNS) キューの輻輳: LNS 内のリアルタイム性の問題は、同じデバイス内の ROS 2 パブリッシャーとサブスクライバー間で共有されるネットワーク リソース (キューなど) に輻輳の問題が発生し、望ましくないレイテンシにつながる場合に起こります。これらの遅延は、ネットワーク メッセージを処理中の複数のスレッドが CPU 時間を得ようとして競合している場合に特に明らかになります。これにより、Linux カーネルの不確定性によるリアルタイム性の問題がさらに大きくなる可能性があります。KRS は、Linux カーネルのトラフィック制御 (TC) 機能を利用して、LNS の不確定性を軽減します。

C. Linux カーネルの不確定性: これはオペレーティング システム (OS) 内部の不確定性に関連するリアルタイム性の問題 (より厳密には、ドライバーを含むカーネル レベルでプリエンプションが適切なタイミングで実行されないこと) の原因となります。たとえば、カーネル ドライバーにプリエンプション機能がない場合、メッセージの処理、配信、または取得時に CPU サイクルの到着が遅れるために、通信の遅延につながることがあります。KRS は、完全にプリエンプティブな Linux カーネル (`PREEMPT_RT`) を提供し、確定性とプリエンプション用に最適化されたドライバーを備えるザイリンクス独自の Linux カーネル fork を維持することにより、これらのリアルタイム性の問題に対処します。(1)

D. ラスト レベル キャッシュ (LLC) の競合: ラスト レベル キャッシュも、マルチコア スカラー プロセッサ内でリアルタイム干渉の大きな発生源になります。マルチコア CPU 内で、LLC はすべてのコアによって共有されます。したがって、隣接するコアが LLC を頻繁に使用すると、ほかのコア上のソフトウェアのワースト ケースの実行時間 (WCET) に大きな影響を与える可能性があります。過去の研究 [参照 16] によると、これによって処理速度は 340 分の 1 に低下し、クワッド コア マルチコア プロセッサで想定される理論上 4 倍の高速化は得られません。KRS は、Xen ハイパーバイザー レベルでのキャッシュ カラーリングによってこの問題に対処します。キャッシュ カラーリングとは、L2 キャッシュを分割し、ハイパーバイザー内の各仮想マシン (VM) にキャッシュ エントリを割り当てることです [参照 17]。この手法により、リアルタイム アプリケーションは、確定的な IRQ を実行でき、時間のずれの縮小とキャッシュ干渉の影響の低減を実現し、1 つのアダプティブ SoC 上に複数のクリティカルなワークロードを混在させることができます。

E. 通信ミドルウェアの不確定性: 通信ミドルウェアは、多くの場合 (確定性ではなく) データのコネクティビティとスループットを重視して設計およびプログラムされるため、不確定性の発生源となります。通信ミドルウェアを実装する際には (上位の抽象化層を使用する際にも)、リアルタイム性の安全性を確保するコーディング プラクティスを適用する必要があります。さらに、通信ミドルウェアは OSI スタックの L1 ～ L4 上に構築されるため、エンド ツー エンドのリアルタイム性能を全体として確保するには、通信ミドルウェアの実装が依存する下位層も確定的でなければなりません。KRS は現在、ROS 2 の分野で広く利用されている通信ミドルウェアをサポートしています。将来の機能強化では、これらの通信ミドルウェアの実装が依存する下位層もリアルタイムで確定的になる予定です。

F. ROS 2 コア レイヤーの不確定性: LLC の競合と同様に、ROS 2 レイヤーは、リアルタイム性の安全性を確保するプリミティブとリアルタイム対応の下層を使用して構築する必要があります。KRS は、当初は公式の ROS 2 `rclcpp`、`rcl`、`rmw`、および `rmw_adapters` をサポートします。その他のアーキテクチャについても、ROS 2 抽象化層の一部をハードウェアにオフロードする研究が進められています [参照 11]。

G. ロボット アプリケーションおよびライブラリの不確定性: アプリケーション レベルでは、すべてのライブラリを含むロボティクス コードもリアルタイム性の安全性を確保する必要があり、コードが使用するすべての下位プリミティブおよびレイヤーにも確定性を確保する必要があります。

1. 詳細は、https://github.com/Xilinx/linux-xlnx/ を参照してください。

ユース ケース

C++ で記述して高位合成した ROS 2 アプリケーションの高速化、最初のロボット デバイスの構築

2 つのベクトルを加算し、選択した ROS 2 トピックに 10 Hz の周波数で結果をパブリッシュしようとする、ダブル ベクトル加算 ROS 2 パブリッシャーの場合を考えます。ハードウェア アクセラレーション分野では、このベクトル加算演算 (VADD) は hello world の例としてよく知られています。シンプルさで選ばれたこの例と ROS 2 を組み合わせることで、ザイリンクスは、ロボット設計者がどのように高性能と確定性の向上を実現できるかを考察し、説明することができました。

ROS 2 パブリッシャーを図 6 のコード リスト 1 に示し、ベクトル加算演算を実行する関数をコード リスト 2 に示します。アクセラレーションの実例を示すために、VADD 演算は意図的に高コストにしています。演算負荷の高いベクトル加算は、ROS 2 パッケージによく見られることに注意してください。最初にこのソース コードを KV260 開発ボードのクワッド Arm® Cortex®-A53 CPU で実行すると、10 Hz の目標レートを満たせず、2 Hz 前後にとどまります。この値は、設計者が演算を最適化する際に基準とするベースラインになります。

ベースラインを求めた後、ロボット設計者がこの ROS 2 パブリッシャーを最適化するために実行する 2 番目の手順は、スカラー プロセッサ (CPU) から、アダプティブ SoC 内に作成されるロボット専用回路へ、VADD 演算をオフロードすることです。簡単に言うと、「ロボット チップ」を作成します。これにより、設計者は、SoC が備える並列処理性能を利用して、より高いスループット、より回復力に優れたセキュリティ体制、強力な確定的応答を実現できます。最終的に、VADD 演算はハードウェアに実装されます。

Code listing 1 vadd–publisher ROS 2 package host code running in the Processing System (PS), the scalar CPU processors.

```
1  int main( int argc , char * argv [ ] ) {
2    rclcpp : : i n i t ( argc , argv ) ;
3    auto node = rclcpp : : Node : : make_shared("
           vadd_publisher") ;
4    auto publisher = node–>create_publisher<
5           std_msgs : :msg: : String>("vector" , 10) ;
6    auto publish_count = 0;
7    std_msgs : :msg: : String message;
8    rclcpp : : WallRate loop_rate(100ms) ;
9
10   / /  Application variables
11   unsigned int in1 [DATA_SIZE] ;
12   unsigned int in2 [DATA_SIZE] ;
13   unsigned int out [DATA_SIZE] ;
14
15   while (rclcpp : : ok ( ) ) {
16     / / randomize the vectors used
17     for ( int i = 0 ;  i < DATA_SIZE ;  i++) {
18       in1 [ i ] = rand ( ) % DATA_SIZE;   / /  NOLINT
19       in2 [ i ] = rand ( ) % DATA_SIZE;   / /  NOLINT
20       out [ i ] = 0;
21     }
22
23     / / Add vectors
24     vadd( in1 , in2 , out , DATA_SIZE) ;   / / function
25                                            / /  subject to
26                                            / /  be
                                               / /  accelerated
27     / /  Validate operation
28     check_vadd(in1 , in2 , out) ;
29
30     / /  Publish publish result
31     message . data = "vadd finished, iteration: " +
32         std : : to_string (publish_count++);
33     RCLCPP_INFO(node–>get_logger ( ) , "Publishing:
34                 '%s'" , message . data . c_str ( ) ) ;
35
36     try {
37       publisher–>publish (message) ;
38       rclcpp : : spin_some(node) ;
39     } catch (const rclcpp : : exceptions : : RCLError & e)
           {
40       RCLCPP_ERROR(
41         node–>get_logger ( ) ,
42         "unexpectedly failed with %s" ,
43         e . what ( ) ) ;
44     }
45     loop_rate . sleep ( ) ;
46   }
47   rclcpp : : shutdown ( ) ;
48   return 0;
49 }
```

Code listing 2 vadd function of the **vadd–publisher** ROS 2 package. The operation is executed also in the Processing System (PS).

```
1  void vadd(
2          const unsigned int     *in1 ,   / /  Read–Only
3                                          / /  Vector  1
4          const unsigned int     *in2 ,   / /  Read–Only
5                                          / /  Vector  2
6          unsigned int *out ,             / /  Output  Result
7          int size                        / /  Size in
8                                          / /  integer
9          )
10 {
11       for ( int j = 0;  j < size ; ++j ) {     / /  stupidly
12                                                / /  iterate
13                                                / /  to
14                                                / /  generate
                                                       load
15         for ( int i = 0;  i < size ; ++i ) {
16           out [ i ] = in1 [ i ] + in2 [ i ] ;
17         }
18       }
19 }
```

WP540_06_091321

図 6: コード リスト 1 および 2

VADD 演算をハードウェア実装するには、(FPGA との相互作用を容易にする KRS のコンポーネントである) ザイリンクス ランタイム (XRT) ツールを利用し、KV260 CPU と FPGA 間の通信に OpenCL を使用して ROS 2 パブリッシャーを変更する必要があります。コード リスト 3 (図 7) は、設計者がカーネル ソース コード上で元の ROS 2 パッケージに加える必要がある変更を示しています。コード リスト 4 (図 8) は、カーネル構築用の `colcon` ビルドの `CMakeLists.txt` ファイルに追加される内容を示しています。KRS はこのフローの大部分を自動化し、ROS 2 中心型の体験を設計者に提供します。これにより、設計者は個々のロボット演算に集中でき、エンベデッドおよびハードウェアの詳細の大部分を意識せずに済みます。

Code listing 3 Kernel for the accelerated_vadd_publisher ROS 2 package annotated with HLS pragmas.

```
1  void vadd(
2          const unsigned int  *in1,  //  Read-Only Vector 1
3          const unsigned int  *in2,  //  Read-Only Vector 2
4          unsigned int  *out,        //  Output Result
5          int  size                  //  Size in integer
6          )
7  {
8  #pragma HLS INTERFACE m_axi port=in1 bundle=aximm1
9  #pragma HLS INTERFACE m_axi port=in2 bundle=aximm2
10 #pragma HLS INTERFACE m_axi port=out bundle=aximm1
11
12     for  (int  j  +  0;  j  <size;  ++j)  {  //  stupidly iterate over
13                                            //  it  to  generate  load
14 #pragma HLS loop_tripcount min = c_size max = c_size
15     for  (int  i  =  0;  i  <size;  ++i)  {
16 #pragma HLS loop_tripcount min = c_size max = c_size
17           out  [i]  i  =  in1 [i] + in2 [i];
18       }
19   }
20 }
```

The HLS INTERFACE pragma helps specify how RTL ports are created. In here, we have two vectors as inputs that we would like to read simultaneously. We thereby map arguments to ports and request an additional **m_axi** port.

HLS loop_tripcount pragma informs the HLS compiler about how many iterations (in total) the loop will execute. This is then used by the compiler while at synthesis to estimate the overall computation time.

WP540_07_091321

図 7: コード リスト 3: `accelerated_vadd_publisher` ROS 2 パッケージのカーネル、HLS プラグマによるアノテーション付き

Code listing 4 ROS 2 package extension on its **CMakeLists.txt** file to generate a kernel of the type selected.

```
1  vitis_acceleration_kernel(
2     NAME vadd
3     FILE src/vadd.cpp
4     CONFIG src/kv260.cfg
5     INCLUDE
6        include
7     TYPE
8        # sw_emu
9        # hw_emu
10       hw
11    PACKAGE
12 )
```

ament_vitis macro, helps define acceleration kernels directly from CMakeList.txt

kernel name

platform configuration

source code

build (kernel) type

invoke Vitis compiler package flag

WP540_08_091321

図 8: コード リスト 4: 選択したタイプのカーネルを生成する、`CMakeLists.txt` 上の ROS 2 パッケージの拡張

デフォルトでは、カーネルを作成するとき、すべての引数が単一の AXI インターフェイスに割り当てられます。このことはデバイス リソースの節約に役立ちますが、CPU と FPGA 間のすべてのメモリ転送が 1 つのポートを通過するため、性能が制限される可能性があります。さらに、`m_axi` ポートには互いに独立した READ チャネルと WRITE チャネルがありますが、各チャネルには同時に 1 つのソースしか割り当てることができません。つまり、1 つの `m_axi` ポートが 2 つのソースからの READ を同時に実行することはできません。引数 (in1 と in2) として 2 つの入力ソースがあるため、追加の AXI インターフェイスを要求し、入力読み出しを 2 つの異なるポートに分離することにより、全体的なデータフローを最適化できます。

これらの変更とカーネルの構築後、ロボット設計者は、ROS 2 パブリッシャーが Kria SOM KV260 ボード上で動作しながら約 6.2 Hz のレートで計算グラフにパブリッシュできるようになったことを観察できます。コード リスト 3 でのデータフローの小さな変更後 (図 7)、初期ベースラインと比較して 3 倍以上高速化しましたが、目標値に設定した 10 Hz には達していません。この変更は CPU と FPGA 間のデータフローを最適化するだけで、ソース コード内のさらなる並列性を活用していません。

コード リスト 5 (図 9) に、これを行う方法を示します。PL-PS の相互作用での入力引数と出力引数間のデータフローの最適化以外に、行 16 でプラグマを利用して内周 `for` ループを係数 16 で展開し、同じクロック サイクルで 16 個の和の計算を並列で実行しています。16 は任意に選択した値ではなく、各クロック サイクルで `m_axi` ポートの全帯域幅 (512 ビット) を消費するように特に選択した値です。512 ビットを埋めるために、16 個の符号なし `int` 入力 (各 4 バイト、16 個の符号なし `int`) がまとめてパックされます。

これらの変更により、最も最適化された形式の VADD カーネルが得られます。このカーネルは 10 Hz のパブリッシング目標を満たし、5 倍の高速化を実現します。(1)

Code listing 5 Kernel for the faster_vadd_publisher ROS 2 package modified and annotated with HLS pragmas.

```
1  void vadd(
2              const unsigned int  *in1,  //  Read-Only Vector 1
3              const unsigned int  *in2,  //  Read-Only Vector 2
4              unsigned int  *out,        //  Output Result
5              int  size                  //  Size in integer
6              )
7  {
8  #pragma HLS INTERFACE m_axi port=in1 bundle=aximm1
9  #pragma HLS INTERFACE m_axi port=in2 bundle=aximm2
10 #pragma HLS INTERFACE m_axi port=out bundle=aximm1
11
12       for  (int  j  +  0;  j  <size;  ++j)  {  //  stupdly iterate over
13                                               //  it  to  generate  load
14 #pragma HLS loop_tripcount min = c_size max = c_size
15       for  (int  i  =  0;  i  <size;  ++i)  {
16 #pragma HLS UNROLL factor=16
17 #pragma HLS loop_tripcount min = c_size max = c_size
18             out  [i]  i  =  in1 [i] + in2 [i];
19           }
20      }
21 }
```

The HLS INTERFACE pragma helps specify how RTL ports are created. In here, we have two vectors as inputs that we would like to read simultaneously. We thereby map arguments to ports and request an additional m_axi port.

HLS loop_tripcount pragma informs the HLS compiler about how many iterations (in total) the loop will execute. This is then used by the compiler while at synthesis to estimate the overall computation time.

Unrolls the inner for loop by a factor of 16, executing them all in parallel within the same clock cycle. The value of 16 is not arbirary, but the one that allows to consume the whole bandwidth of the AXI interface on the platform at each clock cycle (512-bit).

WP540_09_091321

図 9: コード リスト 5: faster_vadd_publisher ROS 2 パッケージのカーネル、HLS プラグマによる変更およびアノテーション付き

同じ原則はその他の ROS 2 パッケージにも適用でき、専用回路を作成することで、C++ で記述した ROS 2 アプリケーションおよびライブラリのアクセラレーションを、HLS(2) の助けを借りて実現できます。

ミックスト クリティカリティおよび ROS 2 下層のアクセラレーション

「KRS の中核となるリアルタイム性」で示したように、リアルタイム性はロボット システムのエンド ツー エンドの特性です。ROS 2 アプリケーションが確定的なタイミングでリアルタイムで応答するには、下位の抽象化層 (ROS 2 下層) もすべて確定的なタイミングで応答する必要があります。KRS が備える複数のビルド済みコンポーネントは、ROS 2 メタ ビルド ツール (colcon) から直接選択でき、設計者が確定的なタイミングで動作するロボット システムを簡単に構築できるように支援します。図 10 の 2 つの例は、KRS で colcon がどのように (colcon_acceleration によって) 拡張され、完全にプリエンティティブな Linux カーネルと、3 つの仮想マシンを備える Xen ハイパーバイザーを含む同じ KV260 ボード内のミックスト クリティカリティ セットアップを選択できるかを示しています。

1. 実際には 5 倍以上高速化します。ただし、ROS 2 WallRate インスタンスが 100 ms に設定されているため、カーネルは新しいデータ到着の待機中にアイドル状態になり、それ以上の高速化の機会は放棄されます。
2. 先頃のザイリンクスの Silexica 社買収により、同社のテクノロジは Vitis エコシステムに統合されます。この結果、必要な C++ プログラムのコード変更が大幅に減少するため、製品化までの期間が短縮され、ソフトウェア開発者が HLS を利用しやすくなります。Silexica のテクノロジには、並列性の自動識別、設計空間の徹底的な検討による性能/エリアのトレードオフの評価、ユーザーのソース コードへの HLS プラグマの自動的な挿入などがあります。詳細は、https://www.xilinx.com/about/xilinx-ventures/silexica.htm を参照してください。

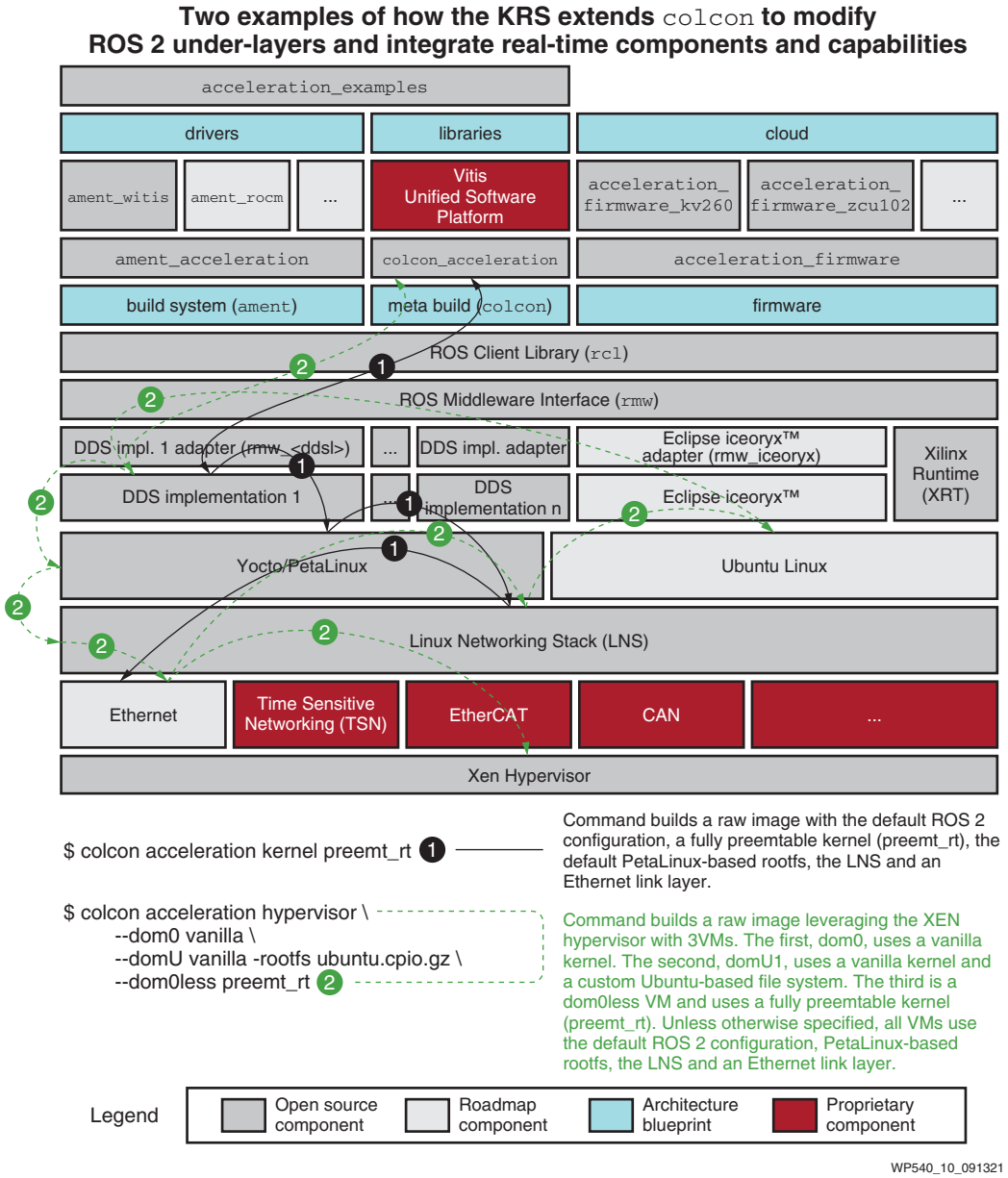


図 10: colcon を使用した ROS 2 下層の変更と、リアルタイム コンポーネントおよび機能の統合例

ザイリンクス アプリ ストア

ROS エコシステムは、パッケージ単位で構成される一連の抽象化層(1) を基盤として構築されています。各 ROS 2 パッケージは、ロボティクス ソフトウェアの 1 つのユニットと見なされます。各パッケージには、ソース コード、ビルド システム ファイル、資料、テスト、品質宣言、ライブラリ、コンフィギュレーション ファイルやその他の関連するリソースが含まれ、有用なモジュールを論理的に構成します。各パッケージは、機能が不十分だと利便性が損なわれ、機能が多すぎるとパッケージが重くなってほかのソフトウェアから使いにくくなるというゴルディロックスの原理に従って、使いやすく再利用可能な機能を提供します。ロボット技術者は、さまざまなパッケージを組み合わせることによって計算グラフを構築し、それらを使用してロボットの動作を構築します。

図 11 に、ザイリンクス アプリ ストア向けに ROS 2 アクセラレーション パッケージを商用化するプロセスを示します。この図は ROS 2 の開発フローを示しています。このフローでは、最初に ROS 2 のインストール時に生成されたファイルを souce コマンドで実行して、その後でオーバーレイ ワークスペースを作成してアクセラレーション カーネルを構築します。その後、作成したオーバーレイを商用化のためにパッケージングして、ザイリンクス アプリ ストアがら入手可能にします。

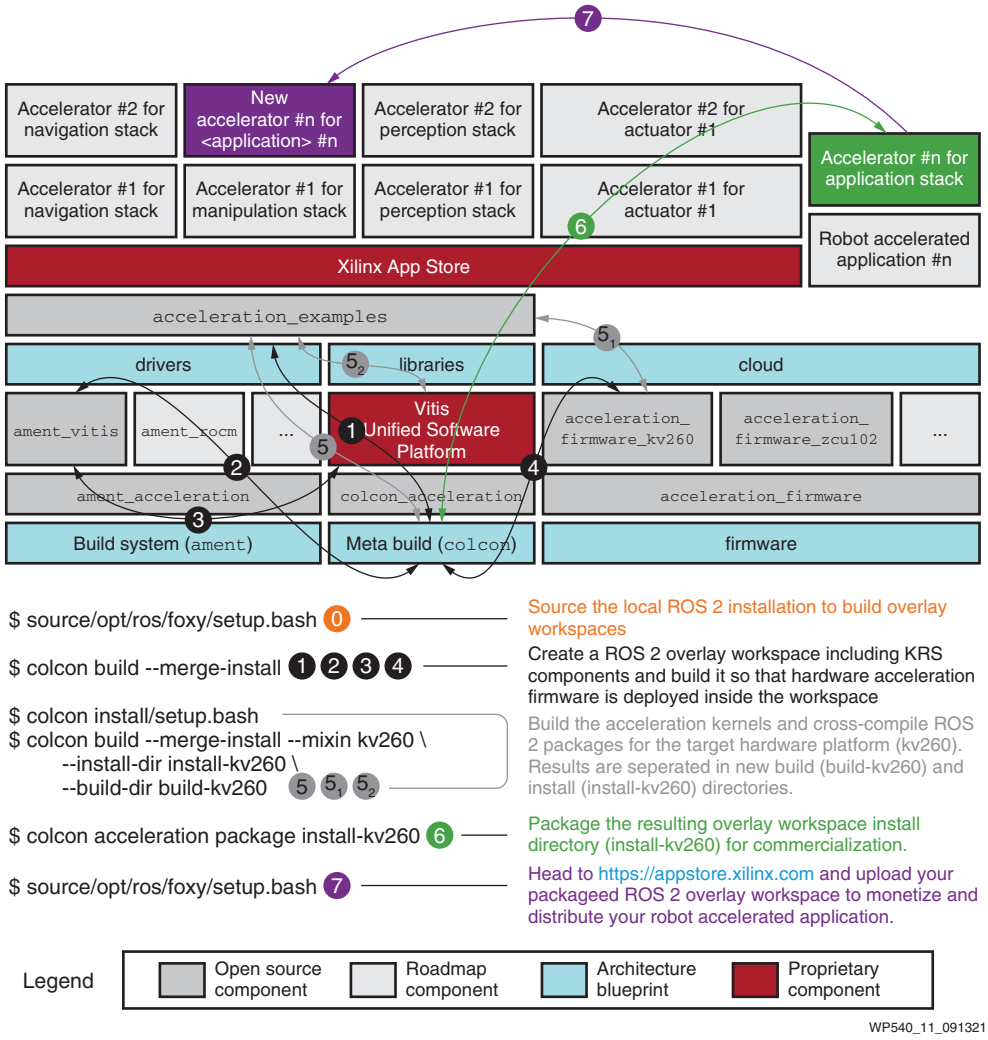


図 11: Kria アプリ ストアで ROS 2 アクセラレーション パッケージを商用化するプロセス

1. ROS 2 の基本概念の詳細は、『ロボティクスにおけるアダプティブ コンピューティング: ROS 2 ソフトウェア定義ハードウェア』(WP537) を参照してください。

パッケージはまとめてメタ パッケージにパックされ、さまざまなサイトの git リポジトリ (最も一般的には GitHub) でサービスが提供されます。すべての利用可能な ROS 2 パッケージのサブセットが (それぞれ独自のバージョンで) ディストリビューションとしてリリースされます (1)。ROS 2 ディストリビューションについては、https://docs.ros.org/en/galactic/Releases.html にまとめられています。ロボット技術者は通常、ロボット開発用のリファレンス SDK として ROS 2 ディストリビューションを使用しますが、多くの場合はベースとなる ROS 2 インストールのオーバーレイとして ROS 2 ワークスペース内に独自のパッケージを作成します。これらの ROS 2 オーバーレイ ワークスペースに、最終的なロボット アプリケーションが含まれます (2)。

ROS 2 パッケージとオーバーレイ ワークスペースの概念は、ザイリンクス アプリ ストアによく適合します。ザイリンクス アプリ ストアは、適切な管理の下、使いやすくセキュアなデジタル著作権管理 (DRM) インフラストラクチャを使用して、個々のアクセラレーション ソリューションのホスティング、マーケティング、販売用の強力なプラットフォームを提供します。世界中のカスタマーに開かれたザイリンクス アプリ ストアは、ロボティクス パートナーが発明した技術や製品を、ROS 2 オーバーレイ ワークスペースの形式で、自社の開発環境とまったく同じように、セキュリティを確保して販売できる商用化ハブです。ザイリンクス アプリ ストア内で ROS 2 アクセラレーション パッケージを商用化するプロセスは、図 11 に示しています。

まとめ

このホワイト ペーパーでは、Kria ロボティクス スタック (KRS) について紹介しました。KRS には、インダストリアル グレードのロボット ソリューションの開発、メンテナンス、商用展開を加速するためのロボット ライブラリおよびユーティリティが統合されています。KRS はロボット開発用 SDK に ROS 2 を利用し、ROS 2 を中心とするアプローチに従ってロボティクス コミュニティと Gazebo シミュレータなどのツールを基盤とする価値の構築を実現します。KRS は Kria SOM ハードウェア ポートフォリオをターゲットとして、量産対応、低レイテンシ (高速演算)、確定性 (予測可能性)、リアルタイム性 (時間どおり)、セキュリティ、高スループットでロボットの価値を向上させます。ROS 2 ロボット技術者は、KRS が提供する機能を活用して独自のセキュアなコンピューティング アーキテクチャを作成できます。KRS はまた、パートナーがザイリンクス アプリ ストアを介してこれらのアーキテクチャを商用化する機会を提供します。

このホワイト ペーパーでは、ロボットのさまざまな演算モデルを紹介し、KRS の基盤となるデザイン原則について説明した後、KRS の 2 つの使用モデルを紹介しています。第 1 に、計算を最適化するロボット設計者の立場で、C++ と HLS で作成した小さな ROS アプリケーションのアクセラレーション カーネルをどのように構築するかを示しています。このカーネルが提供する専用回路では処理速度が 5 倍に向上します。第 2 に、KRS のミックスト クリティカリティおよび ROS 2 下層の最適化機能を利用して、colcon を直接使用してターゲット ハードウェア用のイメージをどのように生成するかの例を示しています。最後に、ROS 2 ワークスペースとオーバーレイについて分析し、これらの概念を基盤として構築された KRS が、どのようにザイリンクスのロボティクス パートナーとザイリンクス アプリ ストアを直接結び付けて、ROS 2 ワークスペースを直接商用化する機能でパートナーを支援しているかを説明しました。

今後、特に ROS 2 アプリケーションおよびライブラリ、ROS 2 コアのアップグレード、ROS 2 下層の 3 つのカテゴリに関連して、より多くのロボット ライブラリおよびアクセラレータで KRS をアップグレードしていきます。同様に、ROS 2 のディストリビューション、ビルド システム、ツールを通じて、KRS と ROS 2 の統合を引き続き改善していきます。また、KRS には、Kria SOM を使用した開発作業を容易にする追加ツールがリリースされる予定です。これには、ネットワーク ブート、追加のセキュリティ メカニズムや、アクセラレータ用のベンチマーク機能が含まれます。KRS を使用して開発を始める際には、https://xilinx.github.io/KRS を参照してください。

1. ROS 2 ディストリビューションに含まれる ROS 2 共通パッケージの詳細は、REP-2005 を参照してください。
2. ROS ワークスペースとオーバーレイの詳細は、REP-0128 を参照してください。これらの概念は catkin を対象としますが、colcon にも同じように適用できます。

参考資料

1. Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, "A survey of FPGA-based robotic computing," IEEE Circuits and Systems Magazine, vol. 21, no. 2, pp. 48-74, 2021.
2. V. Mayoral-Vilches、G. Corradi 『ロボティクスにおけるアダプティブ コンピューティング: ROS 2 に対応したソフトウェア定義ハードウェアの利用』、ザイリンクス (WP537)、2021 年。
3. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in ICRA workshop on open source software, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
4. V. Mayoral-Vilches, A. Hernández, R. Kojcev, I. Muguruza, I. Zamalloa, A. Bilbao, and L. Usategi, "The shift in the robotics paradigm-the hardware robot operating system (h-ros); an infrastruc- ture to create interoperable robot components," in 2017 NASA/ESA Conference on Adaptive Hard- ware and Systems (AHS). IEEE, 2017, pp. 229-236.
5. C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. Mayoral-Vilches, "Time-sensitive networking for robotics," arXiv preprint arXiv:1804.07643, 2018.
6. C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. Mayoral-Vilches, "Real-time Linux communications: an evaluation of the linux communication stack for real- time robotic applications," arXiv preprint arXiv:1808.10821, 2018.
7. C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. Mayoral-Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communi- cations for real-time robotic applications," arXiv preprint arXiv:1809.02595, 2018.
8. C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, I. M. Goenaga, L. A. Kirschgens, and V. Mayoral- Vilches, "Time synchronization in modular collaborative robots," arXiv preprint arXiv:1809.07295, 2018.
9. Y. Yang and T. Azumi, "Exploring real-time executor on ros 2," in 2020 IEEE International Conference on Embedded Software and Systems (ICESS). IEEE, 2020, pp. 1-8.
10. J. Staschulat, I. Lütkebohle, and R. Lange, "The rclc executor: Domain-specific deterministic scheduling mechanisms for ros applications on microcontrollers: work-in-progress," in 2020 International Conference on Embedded Software (EMSOFT), 2020, pp. 18-19.
11. C. Lienen and M. Platzner, "Design of distributed reconfigurable robotics systems with reconros," 2021.
12. ザイリンクス『ザイリンクス ソリューションで IEC 62443 準拠の製品を実現』、ザイリンクス (WP513)、2019 年。
13. ザイリンクス『医療機器のエンベデッド システムにおけるリスク管理』、ザイリンクス (WP511)、2019 年。
14. N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, vol. 3. IEEE, 2004, pp. 2149-2154.
15. R. Casadesus-Masanell and G. Llanes, "Mixed source," Management Science, vol. 57, no. 7, pp. 1212-1230, 2011.
16. M. Bechtel and H. Yun, "Denial-of-service attacks on shared cache in multicore: Analysis and prevention," in 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2019, pp. 357-367.
17. G. Corradi、S. Sabellini 『Zynq® UltraScale+™ MPSoC ベースで干渉のないリアルタイムの仮想化を実現する XEN』2020 年 (オンデマンド)。

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	内容
2021 年 9 月 13 日	1.0	初版

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとします。また、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、https://japan.xilinx.com/legal.htm#tos で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。https://japan.xilinx.com/legal.htm#tos で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとします。セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品の責任の制限を規定する適用法令および規則にのみ従うものとします。

© Copyright 2021 Xilinx, Inc. Xilinx、Xilinx のロゴ、Alveo、Artix、Kintex、Kria、Spartan、Versal、Vitis、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他各国のザイリンクス社の商標です。AMBA、AMBA Designer、Arm、ARM1176JZ-S、CoreSight、Cortex、PrimeCell、Mali、および MPCore は、EU およびその他各国の Arm Limited の商標です。PCI、PCIe、および PCI Express は PCI-SIG の商標であり、ライセンスに基づいて使用されています。すべてのその他の商標は、それぞれの保有者に帰属します。

この資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。