

FPGA 混在の HM-SoC に対する モデルベース並列化設計開発環境の検討

山本 椋太^{1,a)} 小川 真彩高¹ 生沼 正博^{1,2} 近藤 真己³ 本田 晋也⁴ 枝廣 正人¹

概要: 近年、組込み制御システムの多様化・複雑化が進行している。そのため、1つのシステムに CPU に加えて、FPGA や GPU といった複数種類の処理要素 (PE) を搭載することで様々な特性を併せ持つ HM-SoC の利用が増加している。また、設計抽象化によって開発コストを低減するべく、MATLAB/Simulink などによるモデルベース開発の導入も増えている。しかし、ヘテロジニアスマルチコア SoC (HM-SoC) 向けのアルゴリズムレベルの並列化や、異なる PE 間での通信や同期制御の実装は高コストである。そこで、本稿では FPGA 搭載の HM-SoC に対して、MATLAB/Simulink で設計したモデルから並列化コードを生成する統合型設計環境である HS-MBP を提案する。提案環境が持つツールでは、異なる PE 間での通信をサポートしている。

An Environment of Model-Based Development for Heterogeneous Multicore SoC

RYOTA YAMAMOTO^{1,a)} MASATAKA OGAWA¹ MASAHIRO OINUMA^{1,2} MASAKI KONDOU³
SHINYA HONDA⁴ MASATO EDAHIRO¹

Abstract: In recent years, it is increasing that embedded systems has heterogeneous multiprocessors due to increasing the complexity and variety of the systems. Additionally, automotive developers tends to use model-based development (MBD) decrease development cost by abstracting a parallel application development of embedded systems to decrease development cost. Unfortunately, it is difficult and high cost to implement parallel applications to heterogeneous multiprocessors. In this paper, we propose an integrated development environment that can generate parallelized code and executables for heterogeneous multiprocessors. The environment supports the inter-core/CPU-FPGA communication.

1. はじめに

近年、組込み制御システムでは、大規模化や複雑化と共に Time To Market の短縮に対する要求も同時に求められる。加えて、処理負荷の高い機械学習処理の活用といった、

性能面での要求もあり、より高速に動作するハードウェアプラットフォームを利用する需要が高まってきている。

実行性能が要求される組込み制御システムでは、シングルコアでは十分な性能を発揮することが困難となっており、マルチコアによる性能向上が行われている。しかしながら、自動運転等の更に高い性能が要求される応用に対しては、マルチコアだけでは性能要件や消費電力要件を満たせないケースがあり、GPU や FPGA などの、演算ユニット (PE) を持つ図 1 に示すようなヘテロジニアスマルチコア SoC (HM-SoC) が必要となる [1]。

しかし、HM-SoC 向けの実装課題として、各 PE 向けの最適化に加えて、PE 間の通信 I/Fなどを始めとした同期機構の実装が必要となり、PE 間の最適化を行う必要があ

¹ 名古屋大学
Nagoya University, Nagoya, Aichi 464-8603, Japan
² ガイオ・テクノロジー株式会社
GAIO TECHNOLOGY Co., Ltd., Shinagawa, Tokyo 140-0002, Japan
³ NEC ソリューションイノベータ株式会社
NEC Solution Innovators, Ltd., Kawasaki, Kanagawa 213-8511, Japan
⁴ 南山大学
Nanzan University, Nagoya, Aichi 466-8673, Japan
a) muku@ertl.jp

るなど、高難度かつ開発期間の長期化が挙げられる。

組込み制御システムの開発期間の短縮する方法として設計抽象度の向上があり、その方法の1つとしてモデルベース開発が注目されている。モデルベース開発では、ブロックを用いてグラフィカルにアルゴリズムを設計することができ、アルゴリズムレベルでの検証が容易である。

モデルベース開発の代表的なツールとして、MATLAB/Simulink[2], [3]が存在する。Simulinkモデルは、処理を図記号で表現することができ、ブロックの単位で記述されている。MATLAB/Simulinkから、モデルからシングルコア向けのC言語ソースコード（逐次コード）を生成する、Embedded Coderと呼ばれるツールを利用可能である。

モデルベース開発によってHM-SoC向けの実装が可能となれば、開発期間の短縮と性能向上を両立できる可能性がある。しかしながら、これら両方を同時に活かすための問題として、Embedded Coderで生成された逐次コードを、HM-SoC向けに書き換え、並列化する必要がある。この書き換えには、通信部分の作成や、PE向けの最適化、同期機構も含まれる。

この問題を解決するため、我々の研究グループでは、ヘテロジニアスマルチコアやGPU混在環境における、モデルベース並列化（Model-Based Parallelization, MBP）の研究を進めてきた[4]。MBPを推進するために、MBP環境の研究開発を進めており、たとえば、Simulinkモデルから、マルチコア環境やGPU混在のHM-SoC向けの並列実行向けのソースコード（並列化コード）を生成してきた[4]。しかし、現状のMBP環境では、FPGA混在のHM-SoCをターゲットとはしていない。

そこで、本研究では、FPGA混在のHM-SoCをターゲットとしたMBP環境である、HS-MBP（Heterogeneous System - MBP）の検討を行う。提案する環境では、異なる演算ユニット間の通信機能を提供しており、また、FPGAに対しては、ハードウェア・ソフトウェア協調設計環境のSystemBuilder[5]を利用し、高位合成を利用することで逐次コードから並列化コードを変換する。これによって、すべてCコードを元として開発を進めることができる。

本研究において開発した貢献は、以下の通りである。

- Simulinkモデルから、FPGA混在のHM-SoC向けの並列化コードを生成し、実行可能ファイルの生成を行う一連の開発を支援できる環境HS-MBPを開発した。
- 異なる種類のPE間の通信機構を用意し、上記の並列化コード内において通信APIを自動的に埋め込むことで、開発者は容易に様々なHMPE（Heterogeneous Multicore Processing Element）間の通信を、意識することなく実現することができる。

2. モデルベース並列化

モデルベース開発では、ブロックを用いてグラフィカル

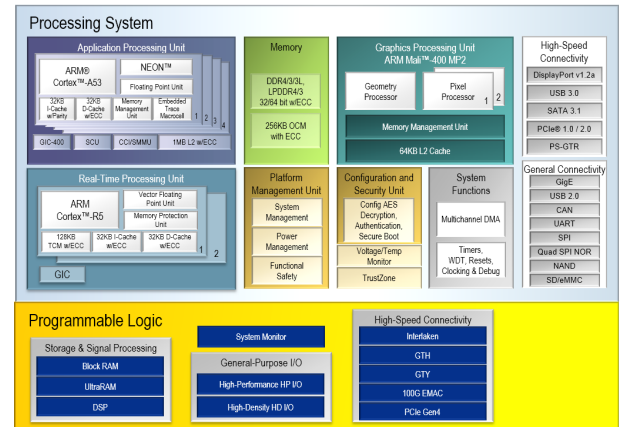


図 1: HM-SoC の構成 [6] より引用

にアルゴリズムを設計する。モデルベース開発の代表的なツールとして、MATLAB/Simulinkが存在する。

従来より、制御分野におけるフィードバック制御などの制御アルゴリズムは、Simulink上でアルゴリズムが設計されてきた。制御アルゴリズムは徐々に複雑化してきており、モデルの規模も増大している。そのため、Cコードを自動生成することが必要となり、MATLAB/Simulinkでは、逐次コード自動生成ツールEmbedded Coderを利用可能である。

生成された逐次コードをベースとして、ヘテロジニアスマルチコアSoCをターゲットとして並列化する場合には、ハードウェア・ソフトウェアプラットフォームの両方の側面で困難な点がある。

まず、ハードウェアの側面では、生成された逐次コードを並列化するために高い専門性と実装コストが必要となる。たとえば、通信機構を用意した上で、必要な箇所に通信機能の呼び出しを追加する必要がある。また、ハードウェアアーキテクチャに対する理解が必要である。

次いで、ソフトウェアの側面では、ソフトウェアプラットフォームに対する対応が必要である。具体的には、コアごとに搭載されるOSが異なる場合に、マルチプラットフォーム対応の通信APIを用意する必要がある。

これに加えて、FPGAが混在するHM-SoCでも、同様の問題が発生し、開発コストが大きくなる。

以上の観点から、モデルベース開発と並列化を組み合わせた開発は難度が高く、また、開発コストがかかる。

しかし、1でも述べたとおり、HM-SoCを利用することで、性能面で得られる恩恵も多く、Simulinkモデルの開発に注力して、実装にかかるコストを低減できれば、開発全体の工数低減につながる。

そのため、これまでに我々の研究グループでは、MBP環境の研究開発を進めてきた。Simulinkモデルから、マルチコア環境やGPU混在環境への並列化コードの生成を行い[4], [7], [8], また、この開発フローをサポートするツールの開発も進めてきた[9], [10], [11], [12], [13]。

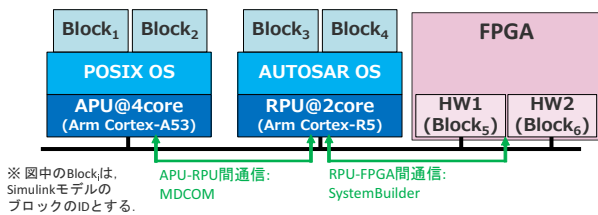


図 2: HM-SoC への実装時の構成

3. FPGA 混在の HM-SoC

ヘテロジニアスマルチコアは、異なるアーキテクチャを持つ複数のプロセッサで構成される。同じアーキテクチャを持つ複数のプロセッサで構成されるホモジニアスマルチコアと比べ、用途に適したアーキテクチャを利用することで、実行時間や消費電力を最適化できる。

本稿では、HM-SoC として、Xilinx 社製の Zynq UltraScale+ MPSoC(ZynqMP) を用いる。ZynqMP は Application Processing Unit(APU), Realtime Processing Unit(RPU) からなるヘテロジニアスマルチコアと FPGA から構成される。ZynqMP の構成を図 1 に示し、実装時の構成を図 2 に示す。ZynqMP に存在する各ヘテロジニアスマルチコア PE (HMPE) のうち、本稿で用いるものについて、以下で説明する。

APU は、4 コアの ARM Cortex-A53 から構成される。L2 キャッシュやパイプラインを持ち、最大 1.5GHz で動作する。Linux が搭載されることを想定している。

RPU は、2 コアの ARM Cortex-R5 から構成される。APU と比べて最大 600MHz と低速だが、リアルタイムシステムのために特化しており、割込みコントローラやローカル RAM へのアクセスレイテンシが低い。AUTOSAR 準拠の車載リアルタイム OS (RTOS) が搭載されることを想定している。また、ロックステップ機能を搭載しており、機能安全系での使用も想定されている。

FPGA は、従来、ハードウェア記述言語 (HDL) による開発が主であり、タイミング制約などといった、ハードウェア設計特有の困難な点が存在しており、開発をする上での専門性が高い。画像処理などのアクセラレータとして使用することや、近年では深層学習のアクセラレータとして利用されることも増えてきている。

4. HS-MBP

HS-MBP 環境は、HM-SoC 上で、Simulink モデルから生成されたモデルが図 1 のような構成で動作するようなシステムの開発を支援する。4.1 では、Simulink ブロックが、各 PE で実行される際の実装について説明し、4.2 では、このときに利用する通信機構について説明する。

また、HS-MBP では、図 3 のようなフローによって、各 HMPE 向けの実行可能ファイルが生成される。実行可能

ファイルが生成されるまでの流れの詳細について、4.3 で説明する。

4.1 HMPE ごとの実装

HMPE によって、Simulink ブロックの実装は異なる。まず、Embedded Coder が生成する逐次コードは、ブロックと関数が必ずしも 1 対 1 で対応するわけではない。そのため、1 つの関数内で複数のブロックがまとめられていることがあり、1 つのブロックに対応した実装とは限らない。

以下、各 HMPE における実装を説明するが、前提として、マルチレート実装、すなわち 1 つのモデル内で複数の実行周期がある場合において、プロセスやスレッド、タスクに分割されることとする。

APU アプリケーションは、Linux 上 (今回は PetaLinux[14] を使用) のスレッドとして、RPU アプリケーションは、AUTOSAR[15] 準拠の OS である TOPPERS/ATK2 カーネル^{*1}の Linux 上のスレッドとして実装される。FPGA では、我々が提案した、システムレベル設計環境の SystemBuilder[5], [16] におけるプロセスの粒度で実装される。

4.2 通信機構

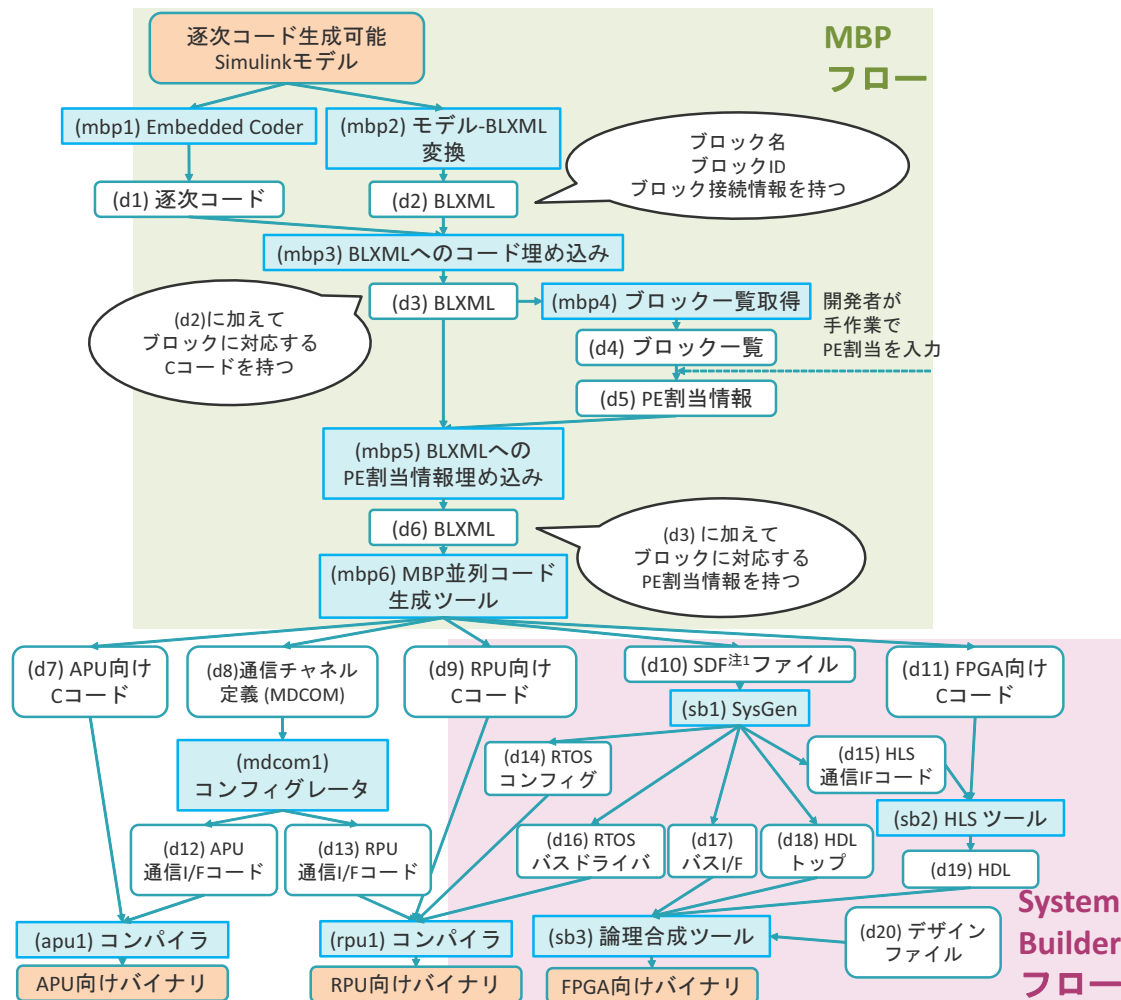
異なる HMPE 間での通信機構について説明する。また、3 でも述べたとおり、ハードウェアアーキテクチャだけではなく、搭載される OS も異なっている。

検討すべき通信のパターンは、APU-RPU 間、RPU-FPGA 間、および FPGA-APU 間の通信であるが、FPGA-APU 間の直接的な通信については、時間概念が大きく異なることから、今回対象外としたそのため、現在は APU-RPU 間、RPU-FPGA 間の 2 つの通信を行うことで APU にデータを送信する。

APU-RPU 間、RPU-FPGA 間の通信について、方針を説明する。これらに共通した通信方式として、マルチレートの実装を考慮している。送信側のタスク・プロセス・スレッドの実行周期を T_s 、受信側のタスク・プロセス・スレッドの実行周期を T_r とするとき、 $T_s \leq T_r$ の場合にはブロッキング通信、 $T_s > T_r$ の場合にはノンブロッキング通信を行う。このアプローチは、中野らによるマルチレート実装の検討に基づいている [17]。

$T_s > T_r$ の場合に、ブロッキング通信だと、深さ 1 のバッファを用意した場合には、送信待ちが生じるため、短周期側は長周期側の処理を待つこととなるため、マルチレートを実現することが困難となる。また、深さが 2 以上の FIFO としてブロッキング通信を実行した場合も、古いデータが蓄積され続け、やがて FIFO がフルとなることで送信側が待ち状態となる。そのため、 $T_s > T_r$ の場合にはノンブロッキング通信を採用して送信側が待ち状態になら

^{*1} <https://www.toppers.jp/atk2.html> (2020 年 7 月 13 日参照)



注 1: システム定義ファイル (System Definition File)

図 3: HS-MBP における開発フロー

ないようにしている。また、ダブルバッファを採用しており、必ず受信側は送信側の 1 周期前のデータを受信することができ、ジッタに対するロバスト性を考慮している。

$T_s \leq T_r$ の場合には、ブロッキング通信を用いる。受信側の受信タイミングに合わせて、送信側は送信を行う。すなわち、 $T_s = 3T_r$ の場合には、送信側が 3 度実行する度に 1 度データを送信する。このようにすることで、送信にかかるオーバーヘッドを低減しつつ、1 周期前のデータを受信することができる。

4.2.1 APU-RPU 間通信

大竹らによって提案され [18], TOPPERS プロジェクトによって開発されている^{*2}, ヘテロニアスマルチコア向け通信ライブラリ MDCOM を用いる。MDCOM によって、TOPPERS/ATK2 カーネルと、POSIX-OS 間の通信が可能となる。MDCOM は、通信機構として以下の 2 つを用意している。

- SMEM チャンネル: 通信を行うすべてのドメインが、い

ずれもアクセス可能な共有メモリを用いたドメイン間通信である。アクセス時にロックを取得することで排他制御を実現している。単一データ領域のみを提供する。

- FIFO チャンネル: メッセージを Block (固定長の共有メモリ) に配置し、FIFO キューに Block の ID を与える。複数のデータ領域を提供する。

マルチレート実装に対しては、 $T_s \leq T_r$ の場合には、FIFO チャンネル、 $T_s > T_r$ の場合には、SMEM チャンネルを利用する。

4.2.2 RPU-FPGA 間通信

SystemBuilder が生成する通信 IF を利用する [5], [16]. SystemBuilder は、HW-SW 間通信、HW-HW 間通信および SW-SW 間通信の各 IF を提供している。SW-SW 間の通信については、現状、RPU 内部のコア間通信をサポートしており、前述の通り、異なるコア間の通信については、MDCOM によってサポートされている。通信 IF は、SDF (System DeFinition) ファイルにおいて定義し、高位合成

^{*2} <https://www.toppers.jp/mdcom.html> (2020 年 7 月 13 日参照)

向けの C コード中, また, RPU 向けの RTOS アプリケーションの C コード中で, API として呼び出すことで使用することができる. SystemBuilder の通信 IF は, 以下の 3 つが用意されている.

- BC (Blocking Channel) : FIFO に相当する. FIFO にデータが存在しなければ, 読み込み側プロセスはデータの書き込み待ちとなり, FIFO の空きがなければ, 書き込み側プロセスはデータの読み込み待ちとなる.
- NBC (Non-Blocking Channel) : レジスタに相当する. アクセス時, BC における待ちを行わず, データを読み書きする.
- MEM (Memory Access Channel) : 任意のメモリに対してアクセスを行う. スタートアドレスを静的に決定しておき, ランタイムではスタートアドレスに対するオフセットで特定のアドレスにアクセスできる.

マルチレート実装に対しては, $T_s \leq T_r$ の場合には, BC チャンネル, $T_s > T_r$ の場合には, NBC チャンネルを利用する.

4.3 設計フロー

図 3 に基づいて説明する. HS-MBP は, 逐次コードを生成可能な Simulink モデルがすでに存在していることを仮定し, これを入力とする. 以下, APU, RPU および FPGA をターゲットとする場合を想定し, まず, MBP 環境の利用手順の例を示す.

手順 1-1 図 3 中の (mbp1) において, 並列化可能な Simulink モデルから, MATLAB/Simulink から利用可能な Embedded Coder を用いて, 逐次コード (d1) を Simulink モデルから自動生成する.

手順 1-2 (mbp2) において, 並列化可能な Simulink モデルから, ブロックレベル構造情報 XML (BLXML) (d2) を生成する.

手順 1-3 (mbp3) では, (d2) の BLXML に対して, ブロックに対応する逐次コードの断片を付与した BLXML(d3) を生成する.

手順 1-4 (mbp4) では, (d3) の BLXML から, ブロック名とブロック ID の一覧を持つ CSV 形式のファイル (d4) を生成する.

手順 1-5 (mbp5) では, (d4) に対して, PE (Pricessing Element) の割当 (HMPE の指定) を手作業により追記したファイル (d5) を, (d3) の BLXML に埋め込んだ BLXML(d6) を生成する.

手順 1-6 (d6) の BLXML かと逐次コードから, 各 HMPE 向けの並列化コードや必要な IF 構成ファイル ((d7) から (d11)) を自動生成する.

MBP 環境では, Embedded Coder は Mathworks 者によって開発されているが, その他のツールは, 我々の研究グループで研究開発を進めてきた. そのツール群の中には自動 PE 割当も含まれているが, 今回の環境に向けての自

動 PE 割当は現在検討を進めているところであるため, 本研究では手作業で PE 割当を行った. 今後, この作業も自動化されれば, 開発者はコマンドを実行するだけで, コード生成までを自動的に行うことができる.

次いで, 各 HMPE 向けの実行可能ファイルの生成までについて述べる. 以降の作業はすべて自動化されている.

まず, APU について, 以下の手順で進める.

手順 2-1 図 3 中の (d7) および (d8) において, ビルドに必要なファイルが生成されている.

手順 2-2 (mdcom1) において, (d8) をもとに, 通信 I/F コード (d12) および (d13) を生成する.

手順 2-3 (apu1) では, (d7) および (d8) から, 実行可能ファイルを生成する.

次いで, RPU については, 通信部分が MDCOM か SystemBuilder かで手順が異なるが, MDCOM 部分については, APU と同様の流れ (手順 2-2 まで) であるため, 省略する. 以下, SystemBuilder 部分 (RPU と FPGA) について説明する.

手順 3-1 図 3 中の (d9) から (d11) において, ビルドに必要なファイルが生成されている.

手順 3-2 まず, (sb1) において, (d10) のシステム定義ファイル (SDF ファイル) をもとに, 通信 I/F やバス I/F 利用のためのファイルや, HDL トップなどのファイル ((d14) から (d18)) を生成する.

手順 3-3 RPU 部分については, 手順 3-2 で生成されたファイルや, MDCOM のコンフィグレータが生成したファイル (d13) を (rpu1) でビルドするだけで良い.

手順 3-4 FPGA (HW) 部分については, (d11) と (d15) からなる C ソースコードを高位合成する. このとき, 高位合成ツールには, NEC 社製の CyberWorkBench (CWB) や, Xilinx 社製の Vivado HLS を利用でき, HDL ファイル (d19) が生成される.

手順 3-5 あらかじめ, SystemBuilder 用の HW デザイン・プロジェクトファイル (d20) は用意されており, これまでに生成された HDL ファイル ((d16) から (d19)) とともに論理合成しビットストリームを生成する. 本稿では, Xilinx 社製の ZynqMP を用いるため, 同社の Vivado を利用することのみ考慮している.

以上のように, 開発者は, HMPE 間の通信などの利用方法に詳しくなくとも, HMPE を利用することができる. また, 生成されたバイナリについて, ZynqMP で実機実行する際には, SD カードに書き込むだけの状態であるため, 実機実行も容易である.

ここまでの手順によってバイナリファイルを生成・実機実行することができるが, シミュレーション環境も用意している. SW 部分には, エミュレータである QEMU を用いることができ, HW と SW のコシミュレーションには QEMU と HDL シミュレータを連携させた環境を用いるこ

とができる。

5. ケーススタディ

本稿では、ケーススタディとして、Simulink モデルの振る舞いを、様々な HMPE 上で動作させ、Simulink モデルのシミュレーション結果と同一の結果を取得できることを確認する。

5.1 入力モデル

今回、ターゲットとする Simulink モデルを図 4 に示す。

このモデルは、PID 制御とプラントからなるマルチレートモデルであり、図 4c に示す P 制御部分は 30ms 周期、図 4d に示す I 制御部分は 20ms 周期、そして、図 4e に示す D 制御部分は 10ms 周期で実行され、それ以外の箇所はすべて D 制御と同様に 10ms 周期で実行される。

また、図 4b、図 4c、図 4d および図 4e は Atomic SubSystem であり、図 4f は SubSystem である。Atomic SubSystem は展開されず、1 つのブロックとして親のモデルから扱われ、SubSystem は、親モデルに展開される。

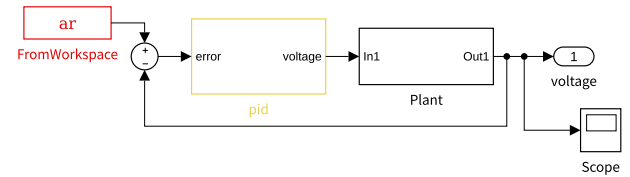
図 4a の入力値はあらかじめ用意してされており、double 型の 0 または 2 が周期的に設定され、出力値も double 型で得られる。

5.2 作業手順と結果

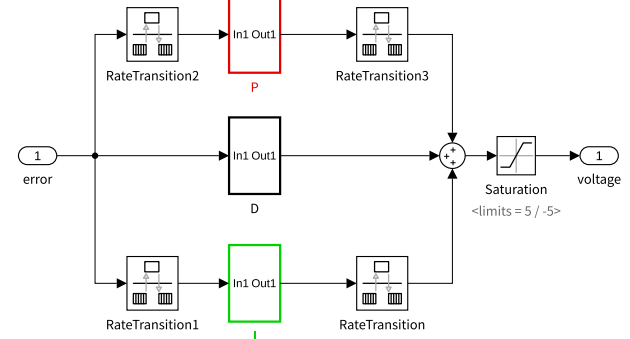
図 4 を入力として、4.3 に示した手順を適用した。図 3 における、(d4) から (d5) の作成を除き、GUI 上から機能を実行する、またはコマンドを打ち込むだけで実行することができる。

このとき、図 5 に示す CSV ファイルを作成している。この CSV ファイルの各行は、ブロック名、ブロック ID、そして PE 割当情報を示している。コア情報を除く情報は、ツールによって自動生成することができるため、開発者は PE 割当情報を追記する。この CSV ファイルを入力として、BLXML に追記するツールも用意されているため、開発者は PE 割当情報をこの CSV ファイルに追記するだけで良い。図 5 中で言えば、rpu0、apu0、および hw0 が該当する。これらは、それぞれ RPU のコア 0、APU のスレッド 0、HW (FPGA) のプロセス ID0 に相当する。ただし、本稿では、APU にはスレッドとして割当を行っており、コア割当を行っていない。

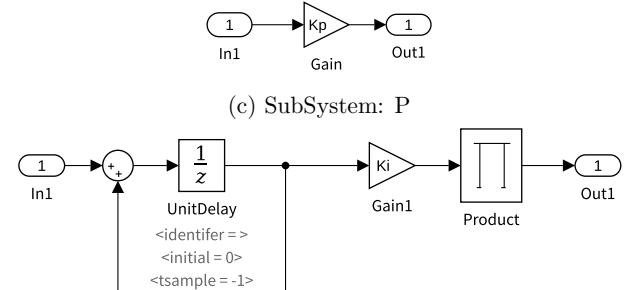
ケーススタディでの目的は、入力モデルを様々なパターンで実行可能かどうか調査することが目的である。そこで、表 1 に示すパターンにおいて、実機実行を行い、出力結果を確認する。また、表 1 は、図 5 において、指定する際の設定を示している。表 1 では、以下のパターンを確認し、マルチレート実装時の通信や実行が正常に実行されることを確認することが目的である。既述の通り、マルチレート実装時は、送受信のブロックの実行周期の長短に応じて、



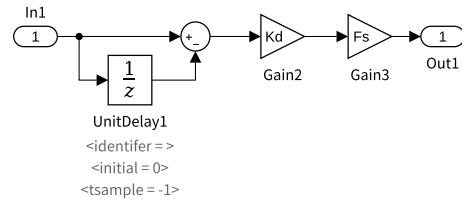
(a) モデルトップ



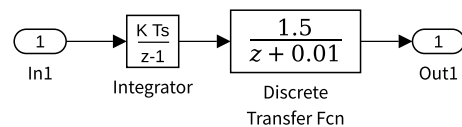
(b) SubSystem: pid



(d) SubSystem: I



(e) SubSystem: D



(f) SubSystem: Plant

図 4: ケーススタディで用いる Simulink モデル

利用する通信機構が変化するため、割当を変えている。また、HW は演算結果を標準出力に出力することが困難であるため、出力値を RPU で確認するようにしている。そのため、10ms のブロックを RPU に割り当てないこととしても、結果の出力に必要な部分のみ、RPU に割り当てられていることがある。

- C1 RPU1 コアに割り当てる。
- C2 RPU2 コアに割り当てる。
- C3 短周期ブロック、中周期・長周期ブロックを、それぞれ RPU1 コア、APU1 スレッドに割り当てる。

multirate2_FromWorkspace,1,rpu0		
ブロック名	ブロックID	対象
multirate2_Plant_DiscreteTransferFcn,4,rpu0		
multirate2_Plant_Integrator,5,rpu0		
multirate2_Sum,7,rpu0		
multirate2_pid_D,10,rpu0		RPUのコア0を指定
multirate2_pid_I,17,apu0		
multirate2_pid_P,24,hw0		HWプロセス0を指定
multirate2_pid_RateTransition,28,apu0		
multirate2_pid_RateTransition1,29,apu0		APUのスレッド0を指定
multirate2_pid_RateTransition2,30,hw0		
multirate2_pid_RateTransition3,31,hw0		
multirate2_pid_Saturation,32,rpu0		
multirate2_pid_Sum2,33,hw0		
multirate2_voltage,35,rpu0		

図 5: マッピングを指定する CSV ファイル

表 1: 実行パターン

通し番号	マルチレート周期		
	10ms(D 制御)	20ms(I 制御)	30ms (P 制御)
C1	RPU0	RPU0	RPU0
C2	RPU0	RPU1	RPU1
C3	RPU0	APU0	APU0
C4	RPU0	APU0	APU1
C5	APU0	RPU0	RPU0
C6	RPU0	HW0	HW0
C7	RPU0	HW0	HW1
C8	HW0	RPU0	RPU0
C9	RPU0	APU0	HW0

- C4** 短周期ブロック, 中周期・長周期ブロックを, それぞれ RPU1 コア, APU2 スレッドに割り当てる.
- C5** 短周期ブロック, 中周期・長周期ブロックを, それぞれ APU1 スレッド, RPU1 コアに割り当てる.
- C6** 短周期ブロック, 中周期・長周期ブロックを FPGA (HW) とし, RPU1 コア, HW1 プロセスに割り当てる.
- C7** 短周期ブロック, 中周期・長周期ブロックを, それぞれ RPU1 コア, HW2 プロセスに割り当てる.
- C8** 短周期ブロック, 中周期・長周期ブロックを, それぞれ RPU1 コア, HW1 プロセスに割り当てる.
- C9** 短周期ブロック, 中周期ブロック, 長周期ブロックを, それぞれ RPU1 コア, APU1 スレッド, HW1 プロセスに割り当てる.

C1 から C9 のすべてのパターンを実行し, それぞれが Simulink モデルのシミュレーション結果と一致することを確認した. また, これらは, 第 1 著者によって実施されたが, コア割当を行う時間も 1 分から 2 分程度であり, FPGA を利用する場合を除けば, 3 分以内に実行を行うことができた. FPGA を利用する場合は, HW の論理合成があるため, 論理合成に 10 分程度を要した.

6. 関連研究

林らは, FE-GA (Flexible Engine/Generic ALU array) と呼ばれる動的再構成プロセッサをもつヘテロジニアスマルチコア RP-X に対する, ヘテロジニアスマルチコア向けソフトウェア開発フレームワークおよび API を開発した [19]. 彼らのフレームワークでは, 逐次 C ソースコードを入力とし, PE 割当も自動で行っている. 我々の提案手法では, 入力, Simulink モデルであり, 現状では PE 割当が自動ではない. また, 彼等もヘテロジニアスマルチコアに加えて FPGA を持つ HM-SoC 対象としているが, 入力, Simulink モデルである分, HS-MBP の方が開発の抽象度は高いと考えられ, PE 割当などの自由度はある.

Xilinx は, SoC 向けソフトウェア・ハードウェア協調開発環境の Vitis (バージョン 2019.1 までは, SDSoC)*3を開発している. Vitis によって協調設計を行う前に, PS (Processing Systems) の設定や, メモリ空間の設定, 信号の設定などを行う. これらの作業を実施してから, 協調設計を始めることができる. そのため, ハードウェアの構成や周辺機能の利用のためには, 柔軟に設計を行うことができる. また, 機能として, ハードウェアレベルのシミュレーションやソフトウェアレベルのシミュレーション, 実機実行時プロファイルなどがある. 実機実行時の分析として, たとえば, メモリ転送のレイテンシやサイズなどのプロファイリング結果や, 実行時間のタイミングチャートをデバイスごとに得ることができる.

HS-MBP で用いるシステムレベル設計環境の System-Builder も, Vivado および Vivado HLS を利用しているため, コーディング部分については, ほぼ同様である. しかし, ハードウェア設計の面で差がある. SystemBuilder では, 基本的には事前に用意されたデザインを用いて協調設計を進めるため, ハードウェアの構成は隠蔽されている. つまり, ハードウェアの知識が少なくとも, ソフトウェア・ハードウェアの協調設計が可能である. ただし, 周辺機能を用いる場合など, ハードウェアの構成を変更する場合には, ハードウェア構成の知識が必要となる.

また, シミュレーション機能などについては, System-Builder における, コシミュレーション機能や, QEMU を利用したシミュレーションを可能としており, また, プロファイリング機能も SystemBuilder では用意しているため, 機能としては, 同等に近い機能がある.

7. おわりに

本稿では, Simulink モデルを入力とする, FPGA 混在の HM-SoC に対するモデルベース並列化設計開発環境

*3 <https://japan.xilinx.com/products/design-tools/vitis.html> (2020 年 7 月 13 日参照)

HS-MBP を検討した。

今回対象としたモデルは、マルチレートのモデルであり、データの取得タイミングが異なると、出力結果が異なるはずである。そのため、マルチレート実装ができていると考える。

また、作業時間について、今回は比較的小規模なモデルを対象としており、手作業による PE 割当は容易であった。しかし、モデルが大規模化すると、手作業による PE 割当の作業コストは増大し、割当ミスを生じる可能性も増大する。そのため、PE 割当については、自動化されることが望ましいと考えている。既述のとおりだが、MBP では、自動 PE 割当ツールを用意しており、実行性能見積もりに基づいて、ヘテロジニアス環境であっても自動的に PE 割当ができる。現在、自動 PE 割当を行うための作業フローを含めて検討を進めている。

また性能について、今回は、小規模なモデルが対象のため、FPGA 向けの高位合成コードの最適化は不必要であった。Embedded Coder が出力する逐次コードでは、FPGA による高速な実行のための記述がなされていないため、高速化するべき処理がある場合には、手作業で部分的にコードを変更することが、開発フローとして発生しうると考えている。

今後の課題として、現状、RPU や APU がマルチコアで動作しているかなど、詳細な動作状況の確認をできていないことがあげられる。そのため、シミュレータの導入が必要である。すでに、本環境に対応したエミュレータの QEMU と、RTL シミュレータの Questa は存在しており、SystemBuilder は、QEMU と Questa を用いたコシミュレーションを提供しているが、HMPE によって実行方法が変化するなど、開発者が容易に使用可能となっていない。開発者が容易に使用可能とした上で、このシミュレーション環境を用いて、より詳細な実行時情報を取得し、動作確認を行う必要がある。

謝辞 本研究の一部は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務の結果得られたものです。

参考文献

- [1] Schmidt, A. G., Weisz, G., French, M., Flatley, T. and Villalpando, C. Y.: SpaceCubeX: A framework for evaluating hybrid multi-core CPU/FPGA/DSP architectures, *2017 IEEE Aerospace Conference*, IEEE, pp. 1–10 (2017).
- [2] MathWorks: Simulink, <https://jp.mathworks.com/products/simulink.html> (2020 年 7 月 30 日参照)。
- [3] MathWorks: MATLAB, <https://jp.mathworks.com/products/simulink.html> (2020 年 7 月 30 日参照)。
- [4] 鍾 兆前, 枝廣正人: 組込み制御システムに対するマルチコア向けモデルレベル自動並列化手法, *情報処理学会論文誌*, Vol. 59, No. 2, pp. 735–747 (2018).
- [5] 本田晋也, 富山宏之, 高田広章: システムレベル設計環

- 境: SystemBuilder, *電子情報通信学会論文誌*, Vol. 88, No. 2, pp. 163–174 (2005).
- [6] Xilinx: Zynq UltraScale+ MPSoC, <https://japan.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html> (2020 年 7 月 30 日参照)。
- [7] 油谷 創, 枝廣正人: 階層構造を持つメニーコアアーキテクチャへのタスクマッピング, *情報処理学会論文誌*, Vol. 56, No. 8, pp. 1568–1581 (2015).
- [8] Zhaoqian, Z. and Masato, E.: Model-Based Parallelization for Simulink Models on Multicore CPUs and GPUs, *INTERNATIONAL JOURNAL OF COMPUTERS and TECHNOLOGY*, Vol. 20, pp. 1–13 (2020).
- [9] eSOL: メニーコアプロセッサ対応スケーラブルリアルタイム OS, available from <https://www.esol.co.jp/embedded/emcos.html>.
- [10] 山口滉平, 竹松慎弥, 池田良裕, 李 瑞徳, 鍾 兆前, 近藤真己, 枝廣正人: Simulink モデルからのブロックレベル並列化, 組込みシステムシンポジウム 2015 論文集, Vol. 2015, pp. 123–124 (2015).
- [11] 竹松慎弥, 鍾 兆前, 井上雅理, 横山静香, 小島流石, 近藤真己, 中本幸一, 安積卓也, 道木慎二, 本田晋也, 枝廣正人: モデルベース開発におけるクロスレイヤ設計手法のマルチコア上モータ制御実装への適用, 組込みシステムシンポジウム 2017 論文集, Vol. 2017, pp. 110–111 (2017).
- [12] 佐合 惇, 枝廣正人: ハードウェア抽象化記述 SHIM と SHIMulator によるソフトウェア動的性能見積手法, *情報処理学会研究報告*, Vol. 2020-EMB-53, No. 14 (2019).
- [13] 鳥越 敬, 枝廣正人: ハードウェア抽象化記述 SHIM による性能見積のための LLVM-IR 命令実行時間計測手法, *情報処理学会研究報告*, Vol. 2020-EMB-53, No. 41 (2020).
- [14] Xilinx: PetaLinux ツール, <https://japan.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html> (2020 年 7 月 30 日参照)。
- [15] AUTOSAR Foundation: AUTOSAR Classic Platform Working Groups (CP), <https://www.autosar.org/working-groups/classic-platform/> (2020 年 2 月 12 日参照)。
- [16] Ando, Y., Honda, S., Takada, H. and Eda Hiro, M.: System-level design method for control systems with hardware-implemented interrupt handler, *Journal of Information Processing*, Vol. 23, No. 5, pp. 532–541 (2015).
- [17] 中野友貴, 本田晋也, 枝廣正人, 鈴木 均: モデルベース並列化 (MBP) におけるマルチレートモデルの車載 RTOS 向けランタイムとコード生成, *情報処理学会研究報告*, Vol. 2017-SLDM-179, No. 4, pp. 1–6 (2017).
- [18] 大竹史紘, 本田晋也, 高田広章: ヘテロジニアスプロセッサ向け通信ライブラリ MDCOM, Vol. 2017-EMB-44, No. 34, pp. 1–6 (2017).
- [19] 林 明宏, 和田康孝, 渡辺岳志, 関口 威, 間瀬正啓, 白子 準, 木村啓二, 笠原博徳: ヘテロジニアスマルチコア向けソフトウェア開発フレームワークおよび API, *情報処理学会論文誌*, Vol. 5, No. 1, pp. 68–79 (2012).