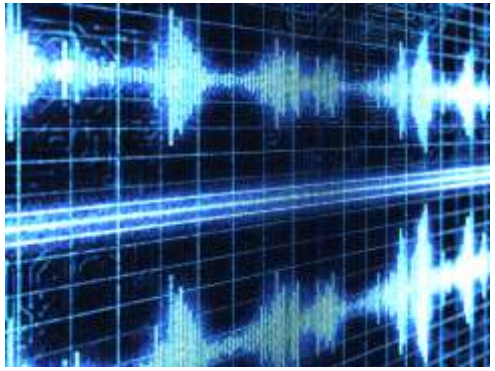
MATLAB EXPO 2019

# HW/SWのパフォーマンス解析・最適化および協調設計

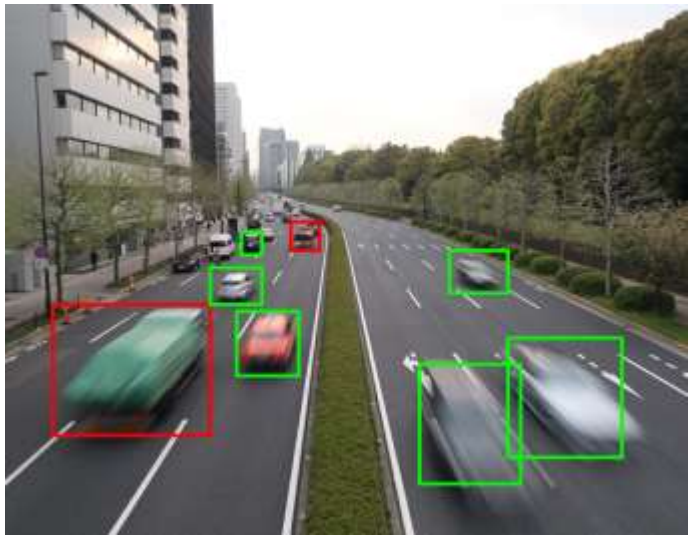
MathWorks Japan
アプリケーションエンジニアリング部
松本 充史

# よくある質問@コード生成ツール



- 最適化されたC/HDLコードが生成されますか？
  ⇒各種アルゴリズム沢山詰め込みたい

- マルチコアCPUでマルチタスクの
  シミュレーションできますか？
  ⇒モータ制御における複数のタスク



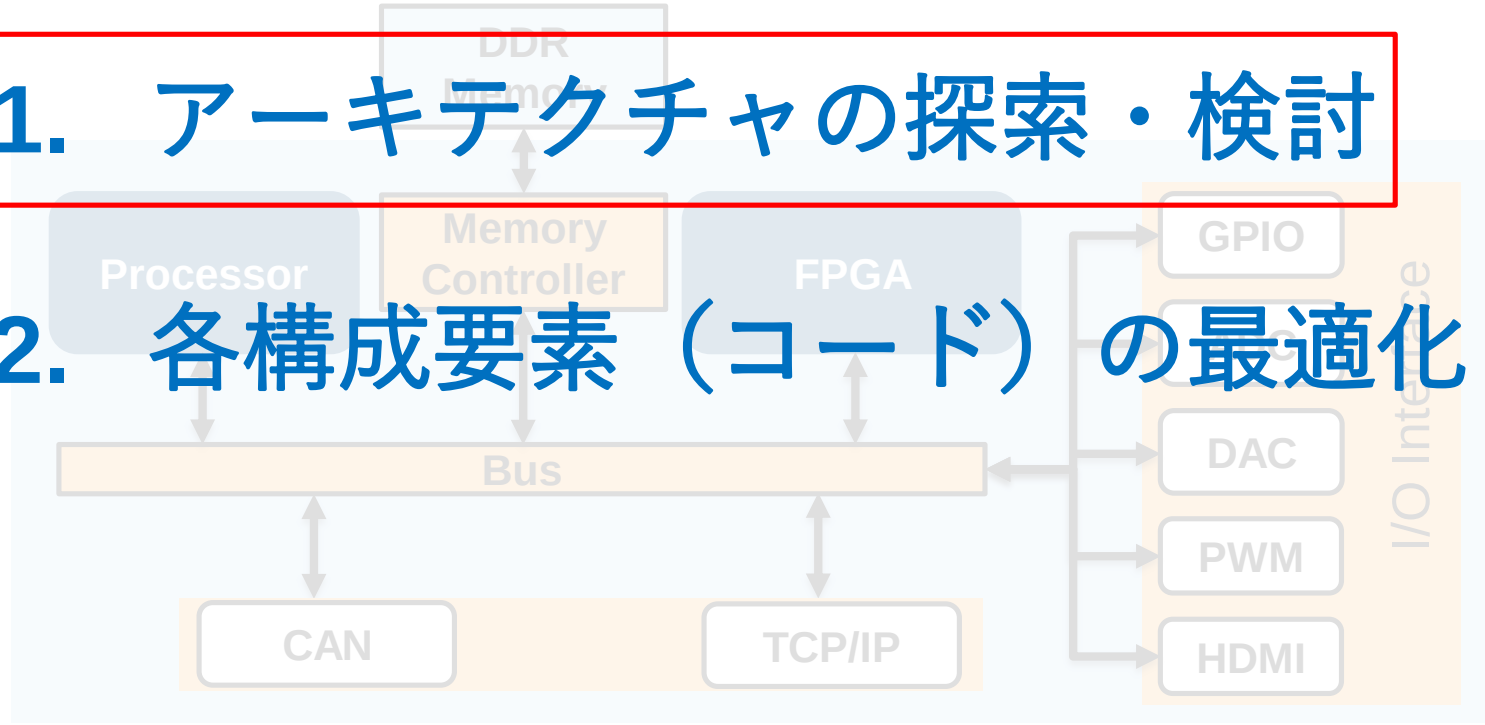
- DDR Memory使えますか？
  ⇒FPGA・Processor間のデータ転送
  ⇒画像処理システムにおけるフレームバッファ

# アプリケーションをSoC FPGAに実装する際の設計要素

- 構成要素
  - Processor
  - FPGA
  - I/O
- 接続要素
  - 内部バス
  - 共有メモリー
- 要素間の通信方法
  - レジスタアクセス
  - 共有メモリ、プロトコル

**1. アーキテクチャの探索・検討**

**2. 各構成要素（コード）の最適化**

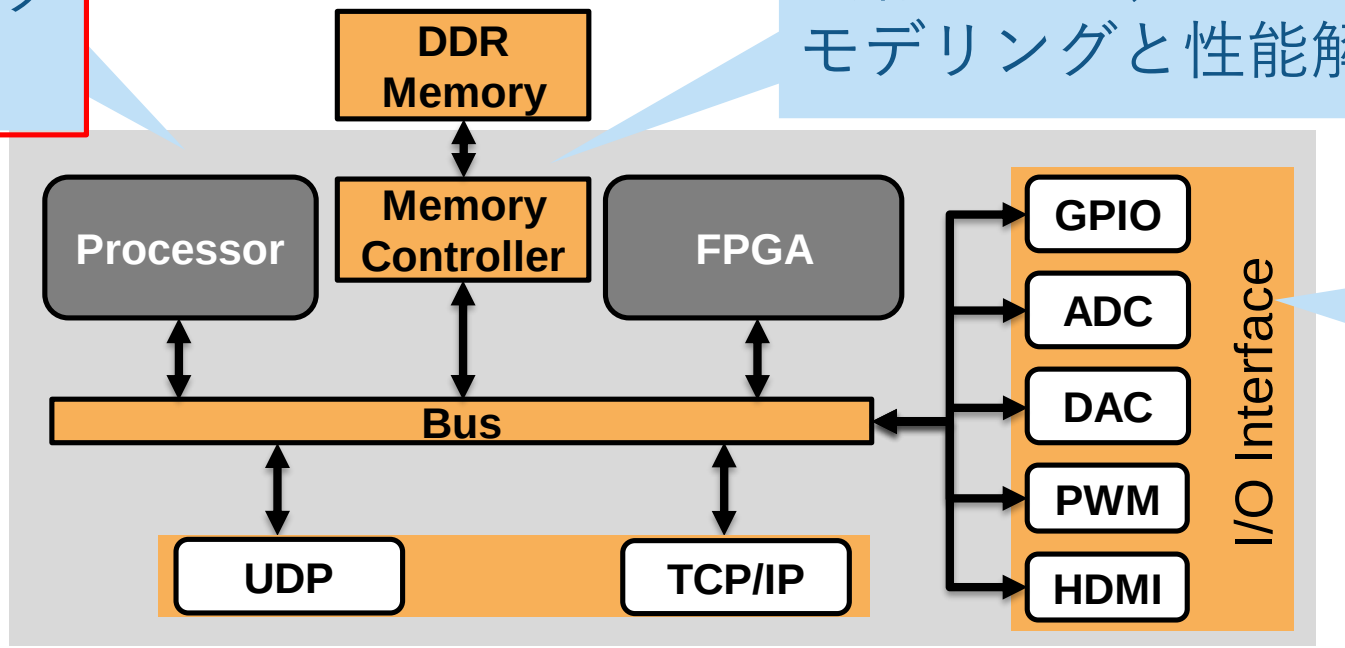


# Processorタスク、共有メモリを含めたSoC設計、解析、実装

*SoC Blockset™* 提供機能 R2019a

Processorのタスク
実行タイミング

共有メモリ、レジスタの
モデリングと性能解析




I/Oブロック
(AD/DA, HDMI, SW,
TCP/IPなど)

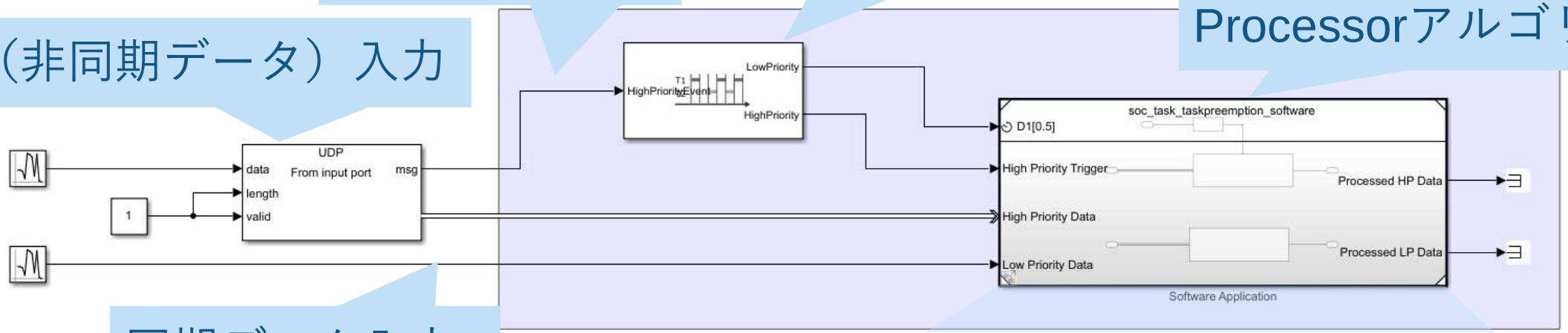
# *SoC Blockset* モデル例 (1/3) ：Processorマルチタスクモデル

非同期イベント

タスクマネージャ

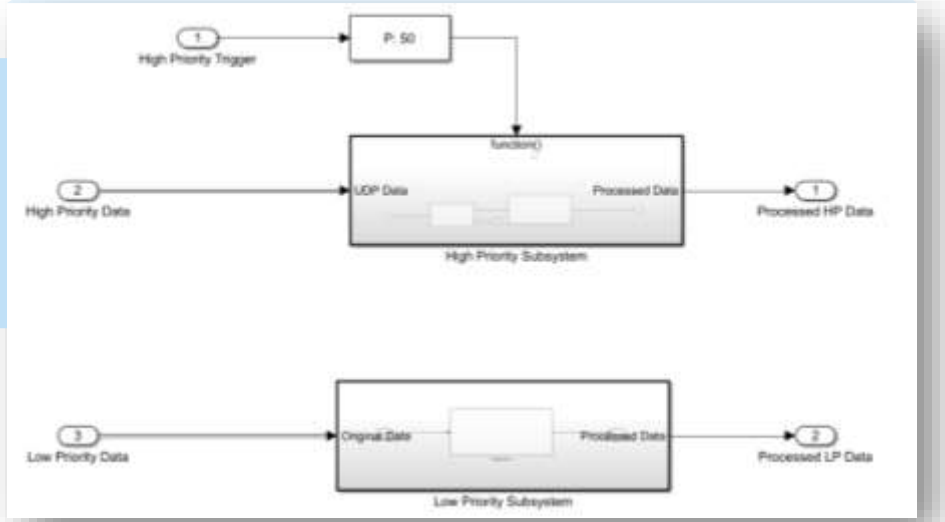
UDP（非同期データ）入力

Processorアルゴリズム

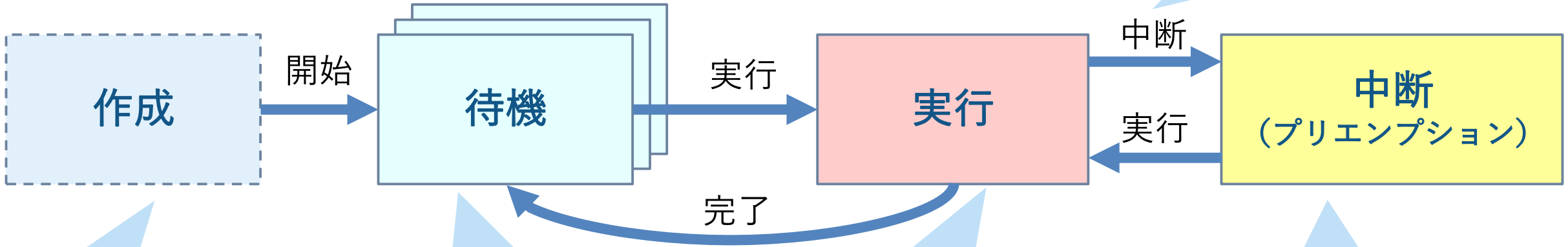



同期データ入力

2つのタスク
1. 非同期タスク
2. 同期タスク



# Processorのタスク実行の概念




タスクを実行中に、それよりも優先度の高いタスクがトリガされると中断に移行。

アプリケーションが起動して作成された状態

タイマーやイベントなど実行を待機している状態
優先度が高いタスクが実行状態になる

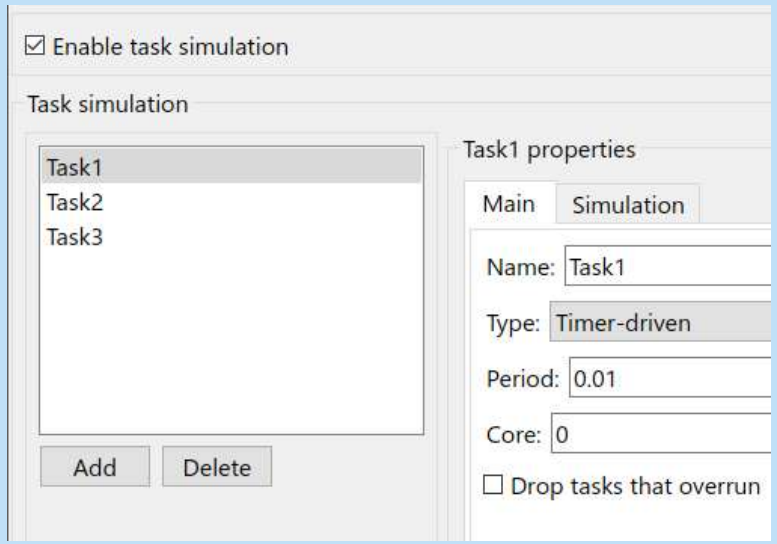
コードを実行している状態
完了すると待機状態に移行

タスクが横取りされ実行待ち状態。

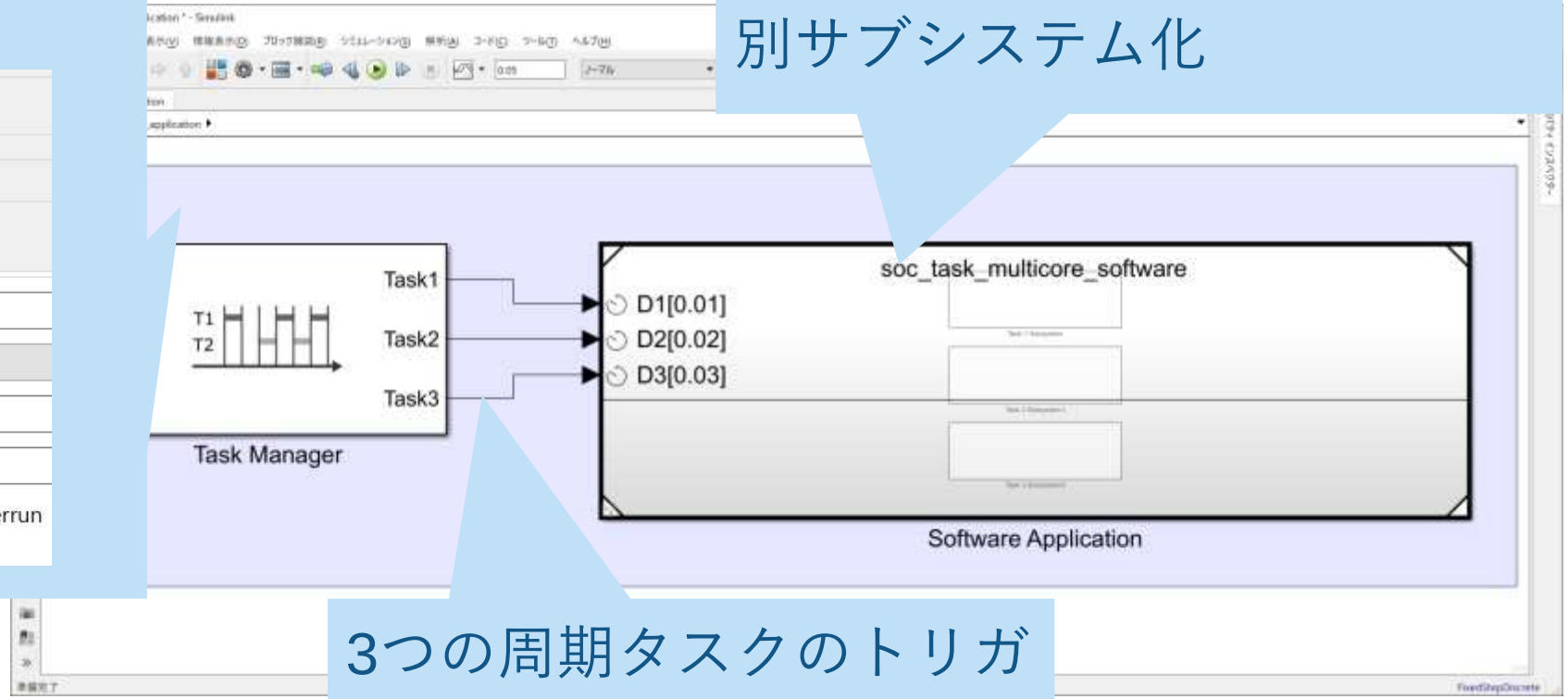
# Processorのタスク実行タイミングの解析例

Task Managerブロックで
タスクの実行タイミングを管理

- 周期、イベント（非同期/周期）
- 実行時間（平均、最小/大、標準偏差）
- 実行するコア

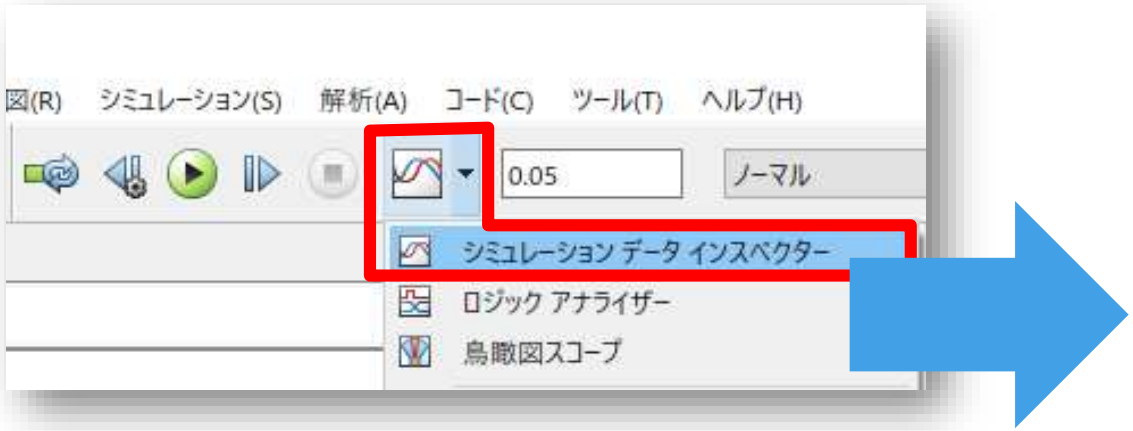



各タスクで実行される処理を
別サブシステム化



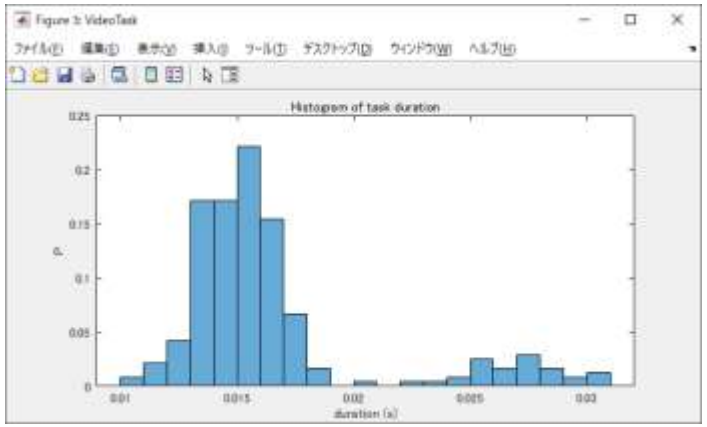

3つの周期タスクのトリガ

# タスク実行タイミングの可視化

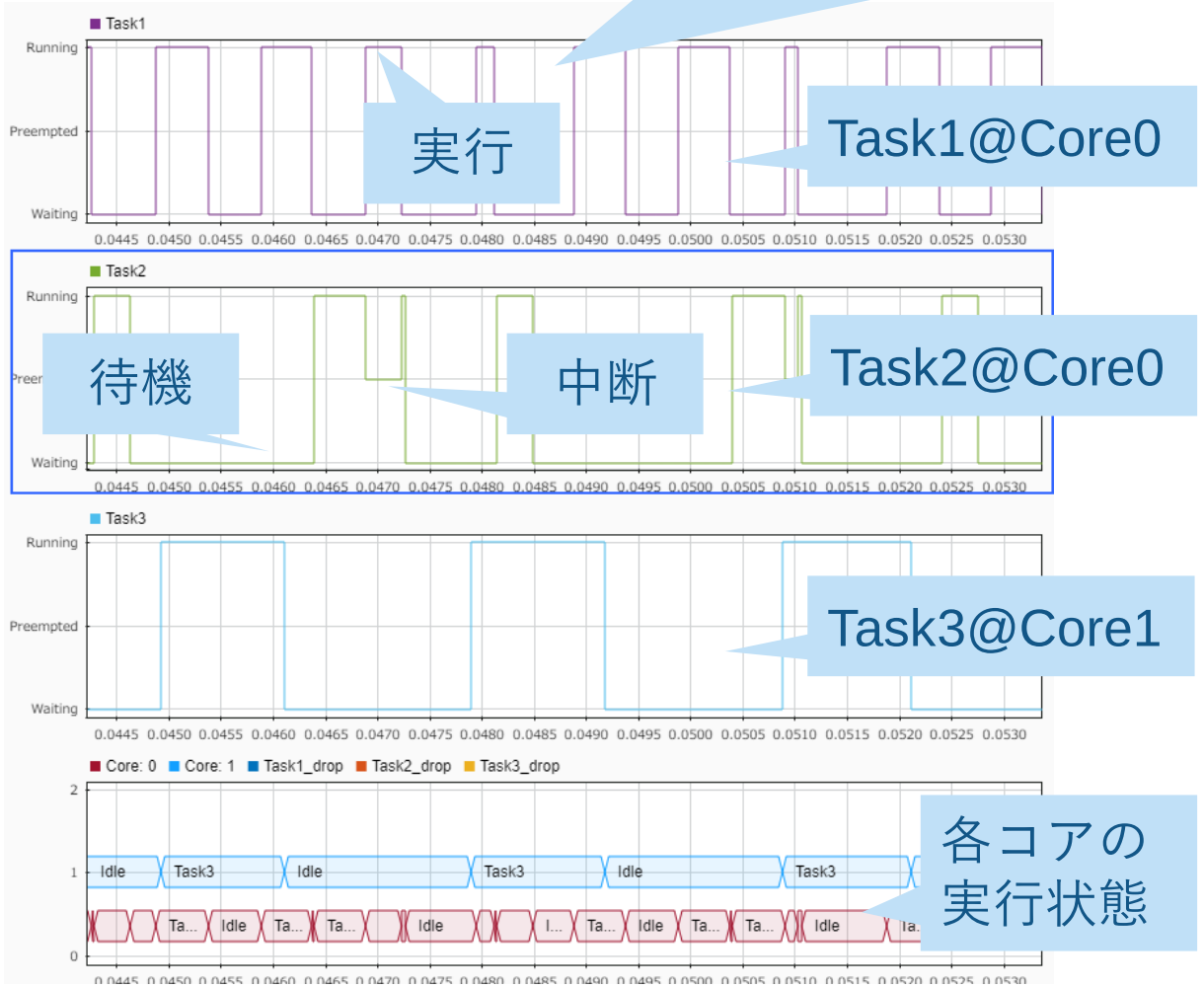



- タスクの実行タイミング（右図）
- 実行時間のヒストグラム（下図）

  **>>socTaskTimes**




ノーマルモード：シミュレーション結果
エクスターナルモード：実機実行結果

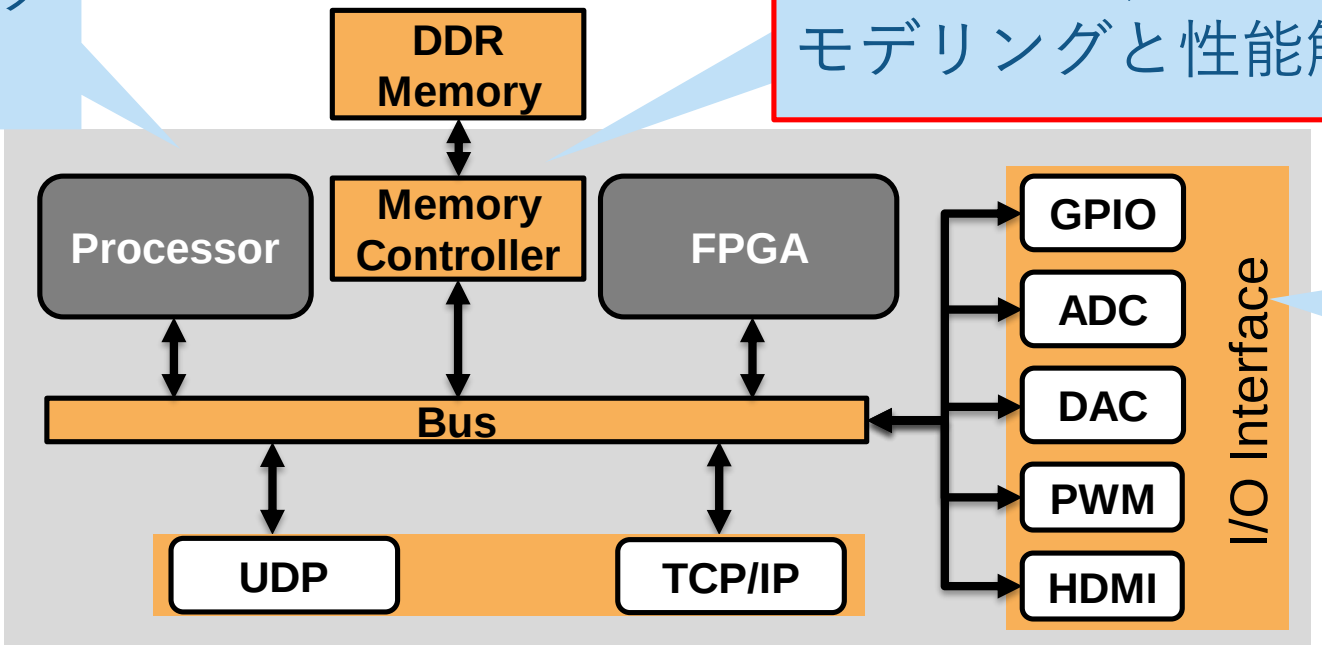


# Processorタスク、共有メモリを含めたSoC設計、解析、実装

*SoC Blockset™* 提供機能 R2019a

Processorのタスク
実行タイミング

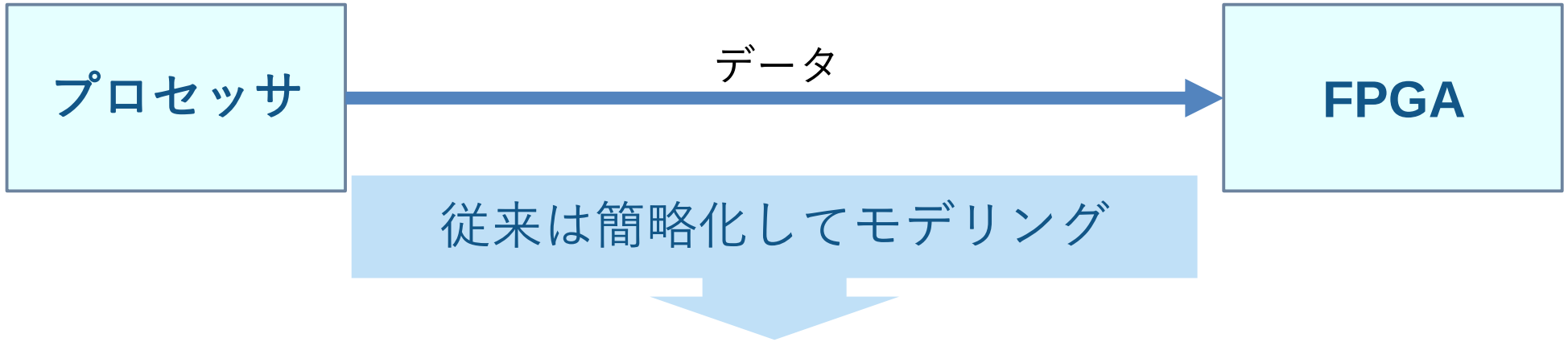
共有メモリ、レジスタの
モデリングと性能解析



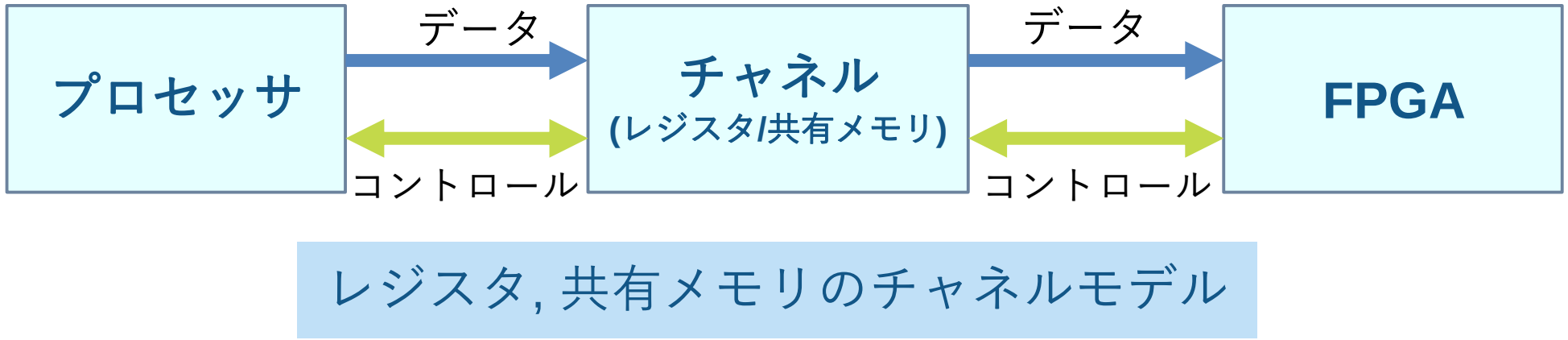

I/Oブロック
(AD/DA, HDMI, SW,
TCP/IPなど)

# メモリ/レジスタ・チャネルモデル

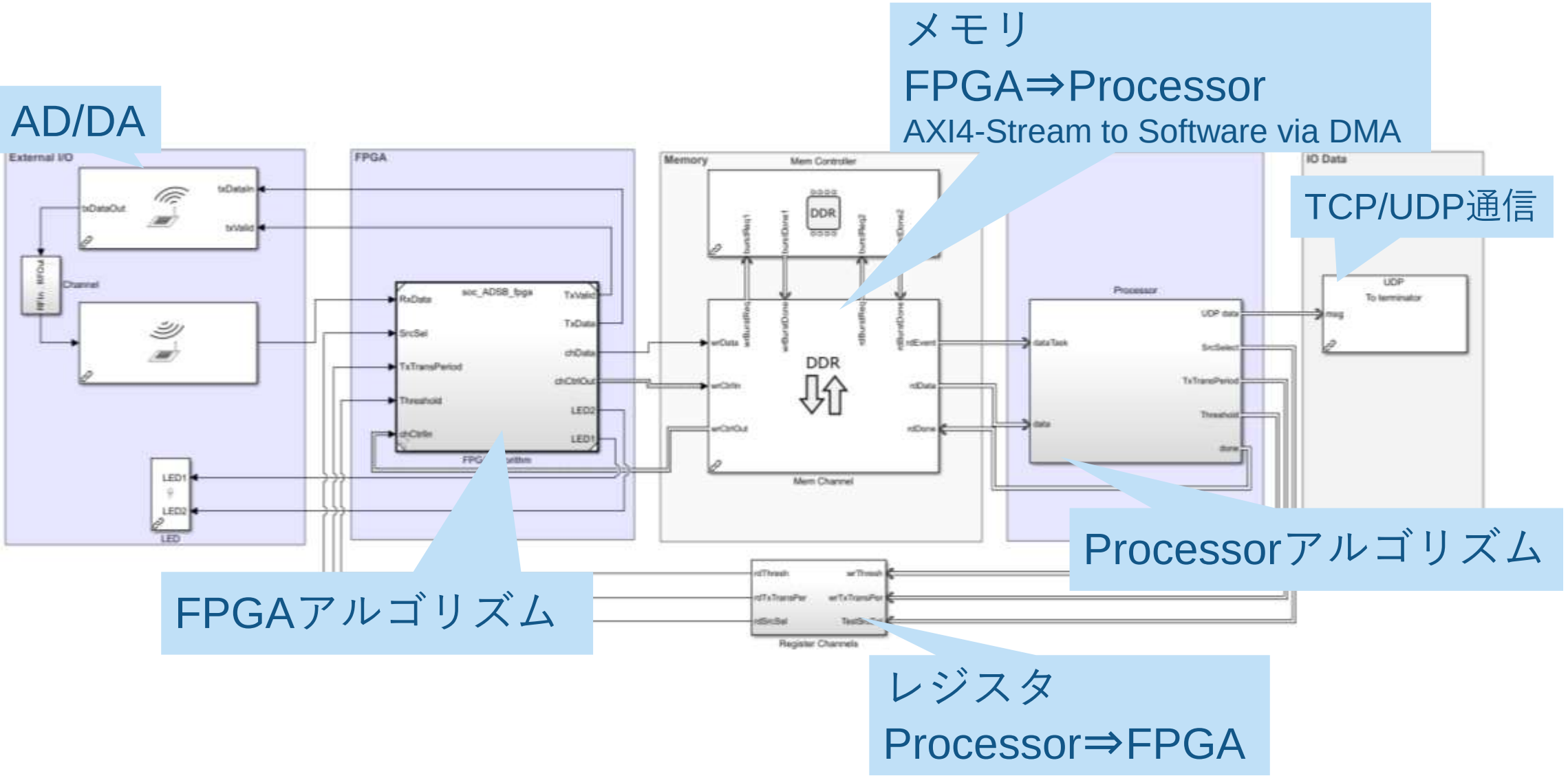
従来モデル



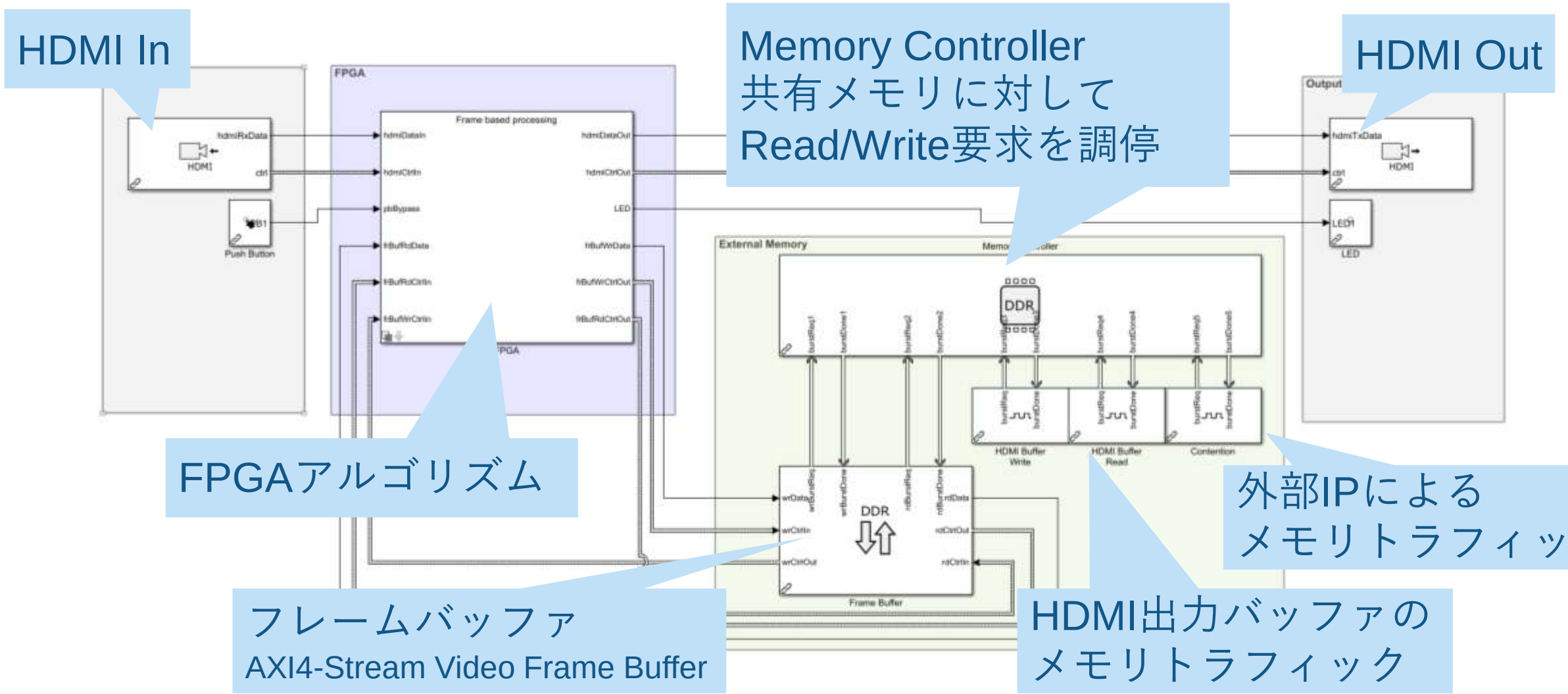

*SoC Blockset*



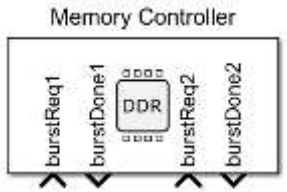
# *SoC Blockset* モデル例 ：信号処理・通信システム



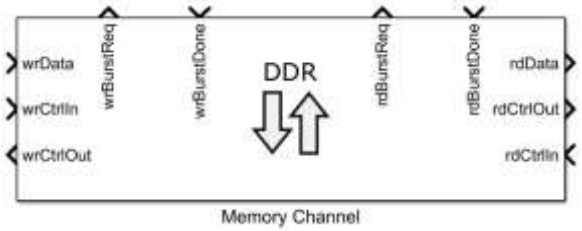
# *SoC Blockset* モデル例：画像処理システム



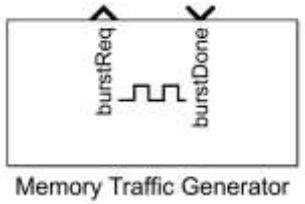
# チャネルブロックの機能



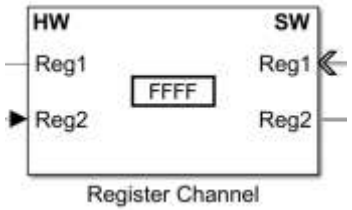

Memory Controller：1つまたは複数のMemory Channelブロックと接続し、メモリトランザクションの調停を行う。ログを取ってパフォーマンス解析する機能を持つ。




Memory Channel：共有メモリを介してI/O, Processor, FPGA間のストリーミングでデータ転送を行うためのブロック。ストリームデータをMemory Controllerブロックへのバースト信号に変換する。




Memory Traffic Generator：パフォーマンス解析のために、HDMI I/Oなどのモデル化していないメモリトランザクションをシミュレーションで発生させる。




Register Channel： FPGA⇒ProcessorまたはProcessor⇒FPGA通信のためのレジスタRead/Writeを行うためのブロック。

# 共有メモリのプロトコルの種類（1/2）

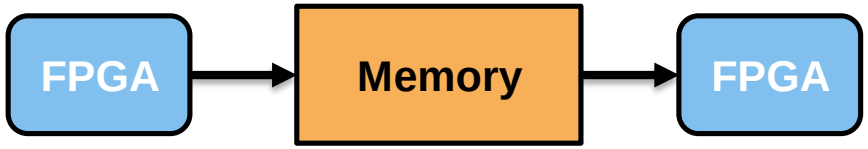
## プロトコル

### ストリームでデータ転送


**AXI4-Stream to Software via DMA**

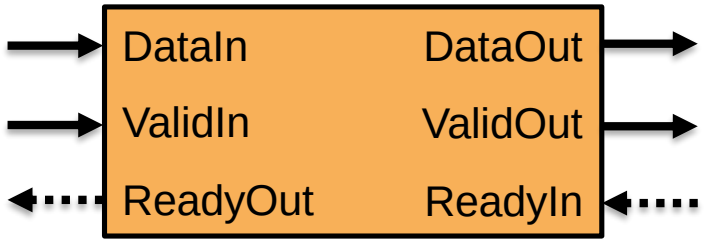

**AXI4-Stream FIFO**

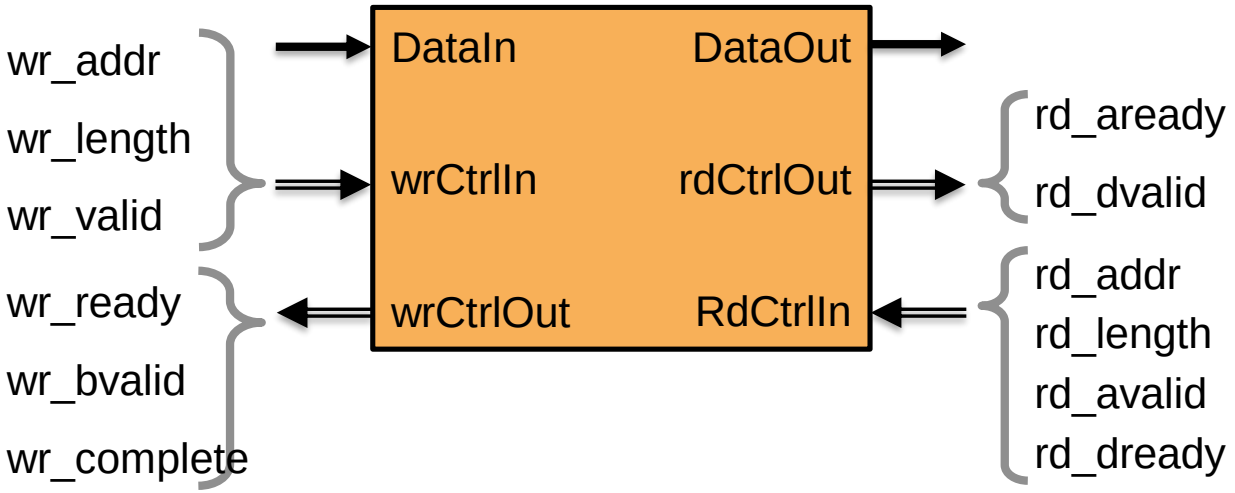
### ランダムアクセスでデータ転送




**AXI4-Random Access**

## インターフェース仕様

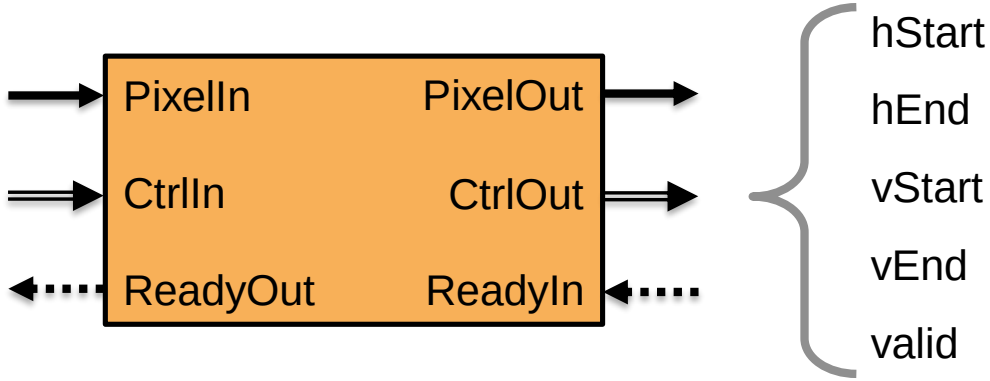


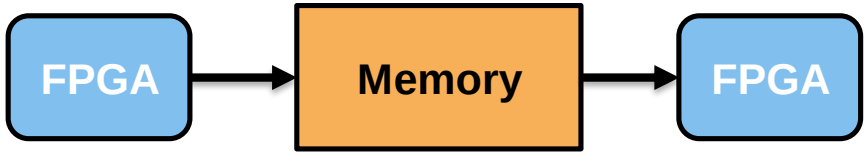
# 共有メモリのプロトコルの種類（2/2）

| プロトコル | インターフェース仕様 |
|---|---|
| ストリームでビデオデータ転送 | |


**AXI4-Stream Video FIFO**

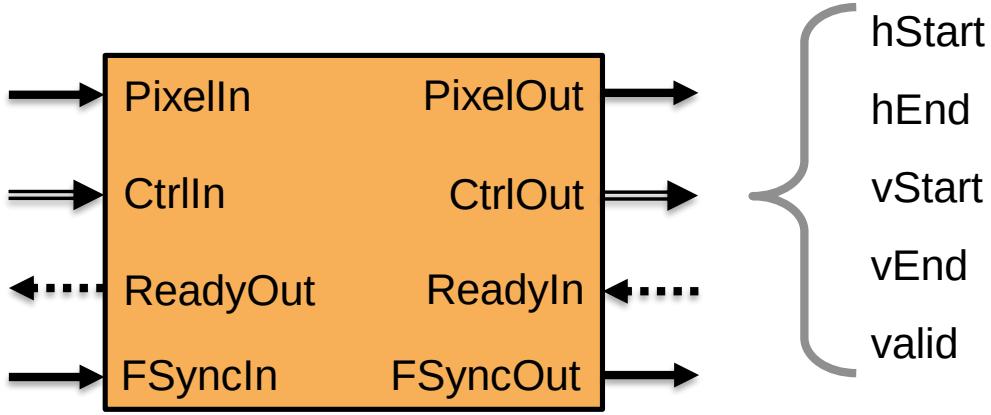



| ストリームでビデオデータ転送 | |
|---|---|



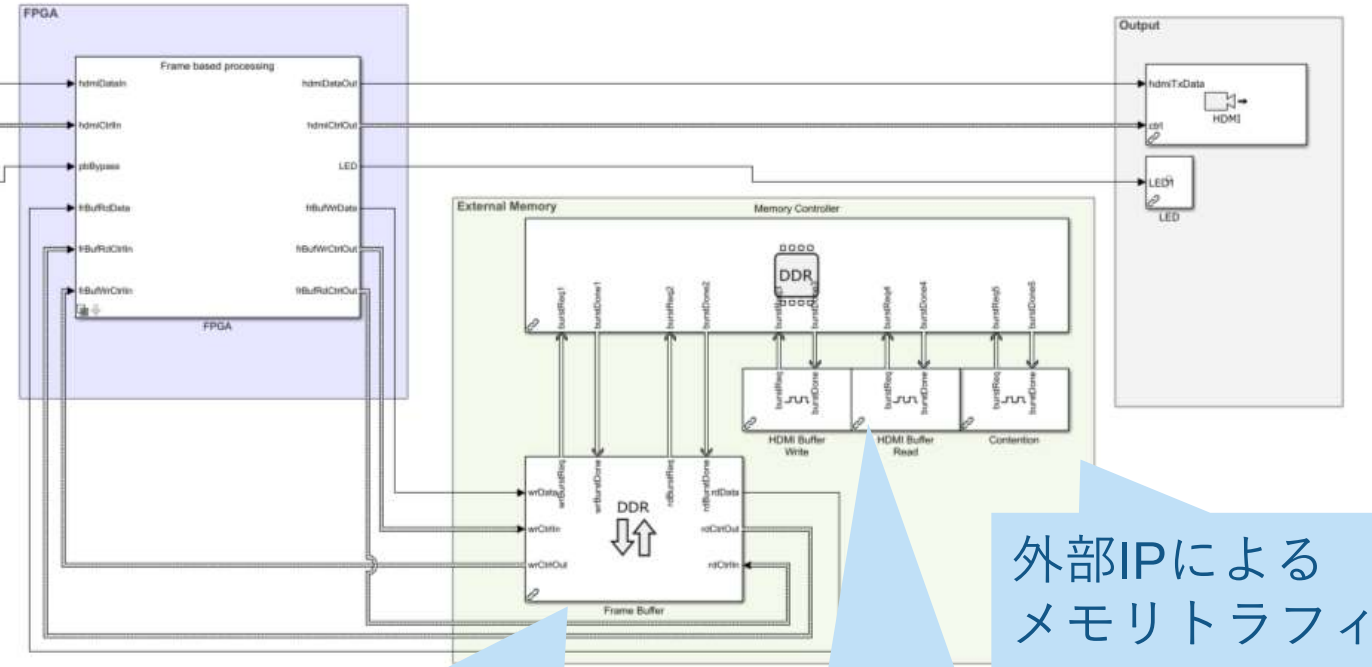

**AXI4-Stream Video Frame Buffer**

FSyncIn/Outはフレーム同期信号



# 画像処理システムのメモリトラフィックの解析

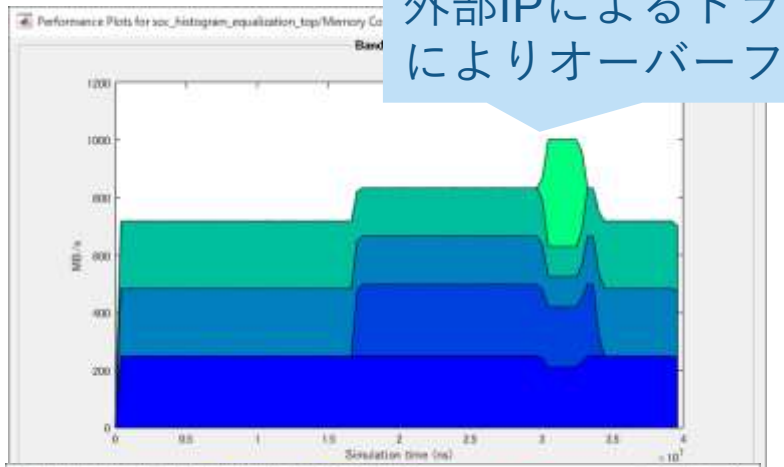
- シミュレーションで複数トラフィックの解析
  ⇒メモリの最大帯域幅、許容レイテンシを超えないか？



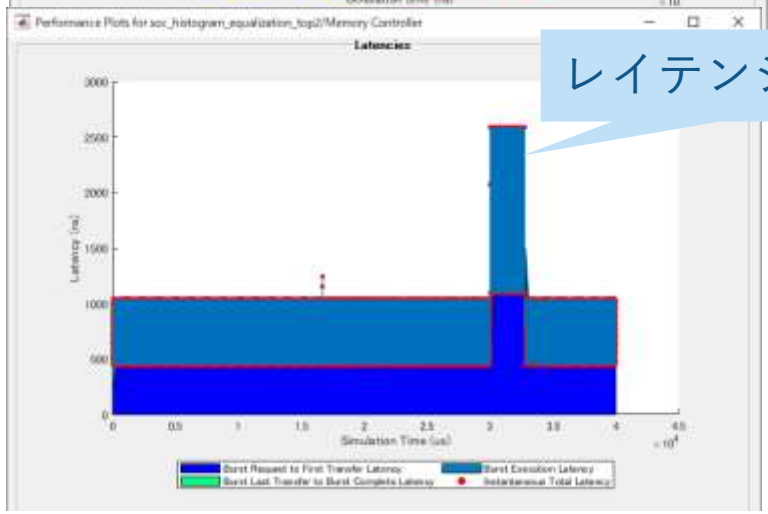

外部IPによる
メモリトラフィック

FPGAのフレームバッファ

HDMI出力バッファの
メモリトラフィック



外部IPによるトラフィック
によりオーバーフロー

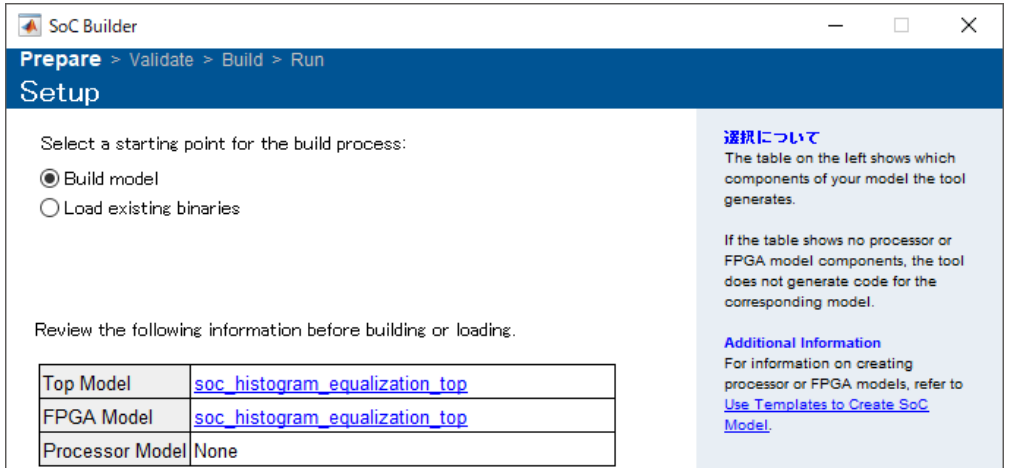


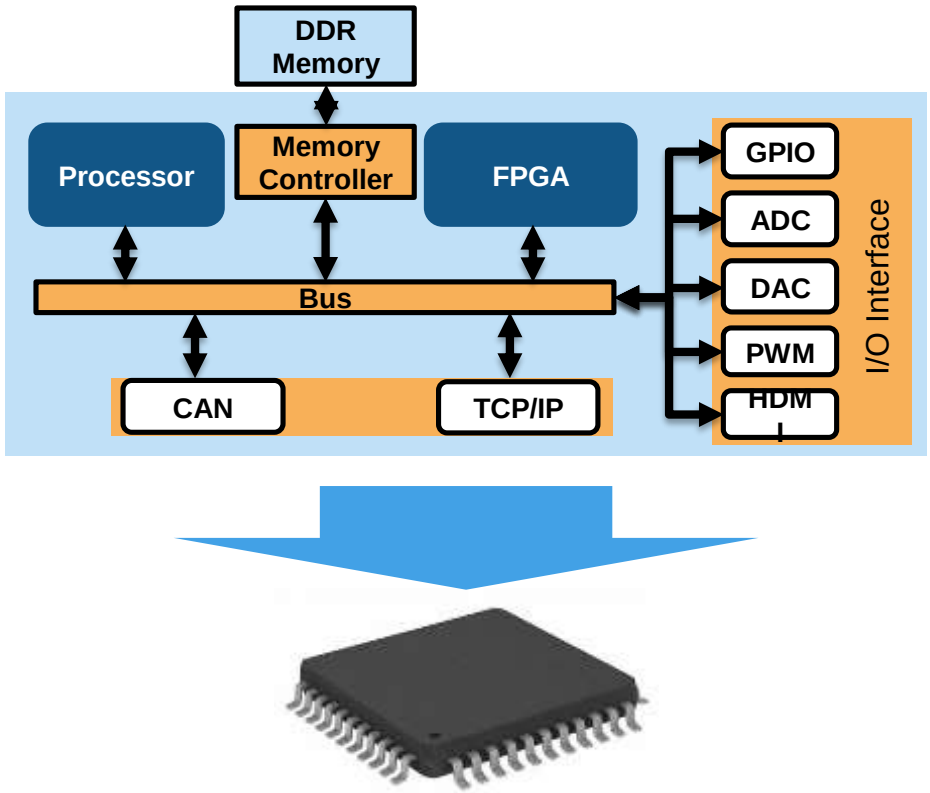

レイテンシが急増

# SoCへのフルチップ実装：SoC Builder

SoC FPGAボードにビルド、ロード、実行するツール（*SoC Blockset* 機能）

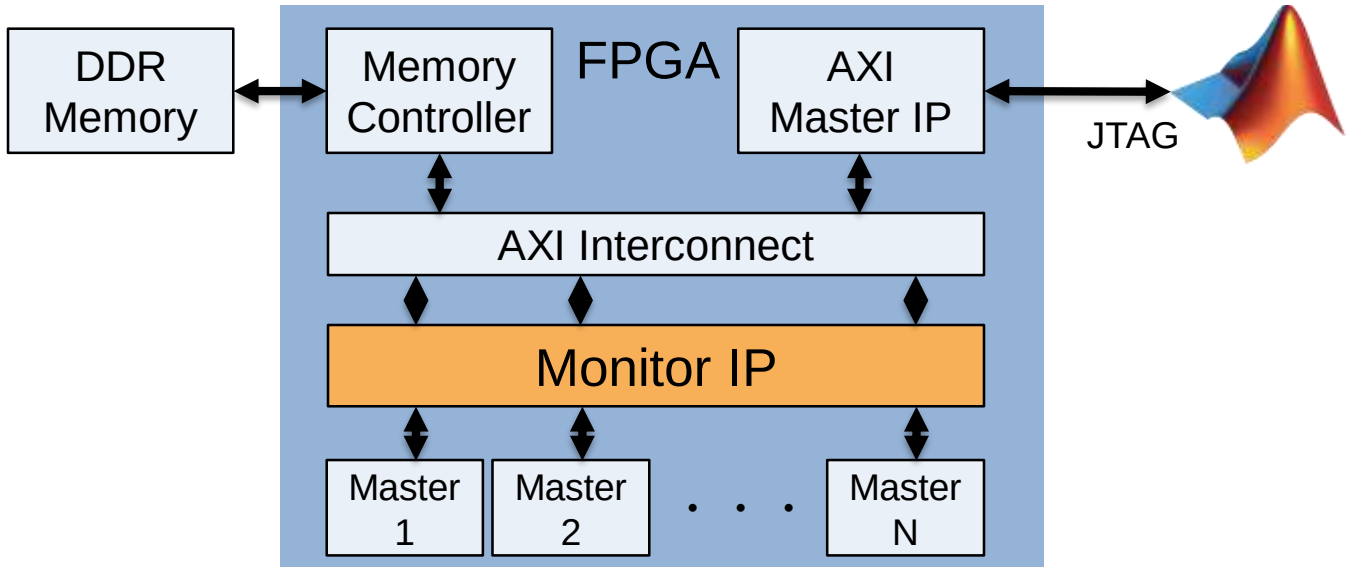
- モデルチェック
- メモリマップのレビュー
- C/HDLコード生成、コンパイル
  - *Embedded Coder/HDL Coder*が必要
- 実行ファイルのダウンロード



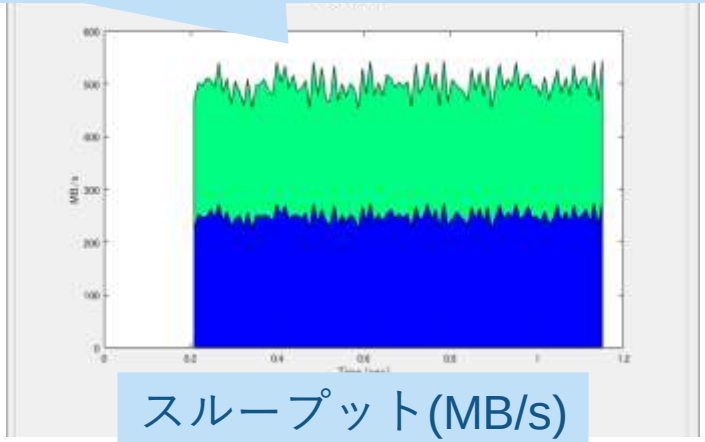



# FPGA実機のメモリパフォーマンスの解析

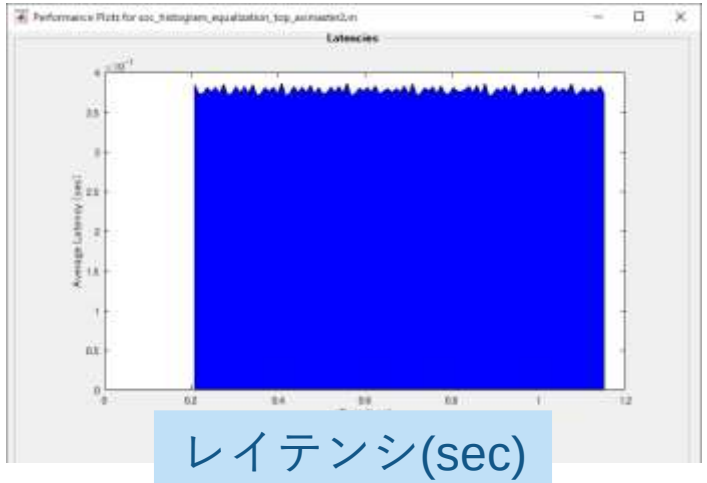
- FPGA実機動作の帯域解析
  ⇒実機動作ログ
  - スループット (MB/s)
  - レイテンシ (sec)




FPGA実機の
メモリトラフィック解析
（フレームバッファのRead/Write）



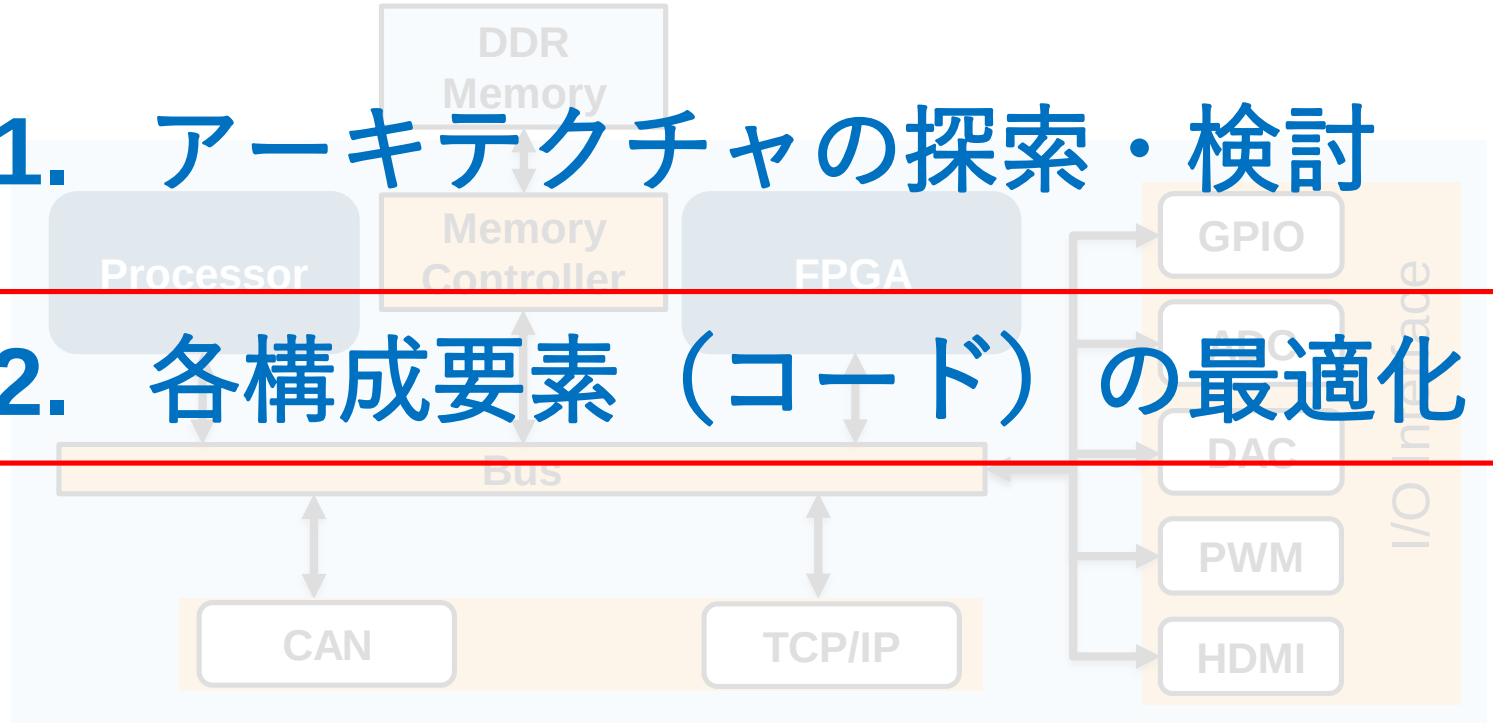
スループット(MB/s)



レイテンシ(sec)

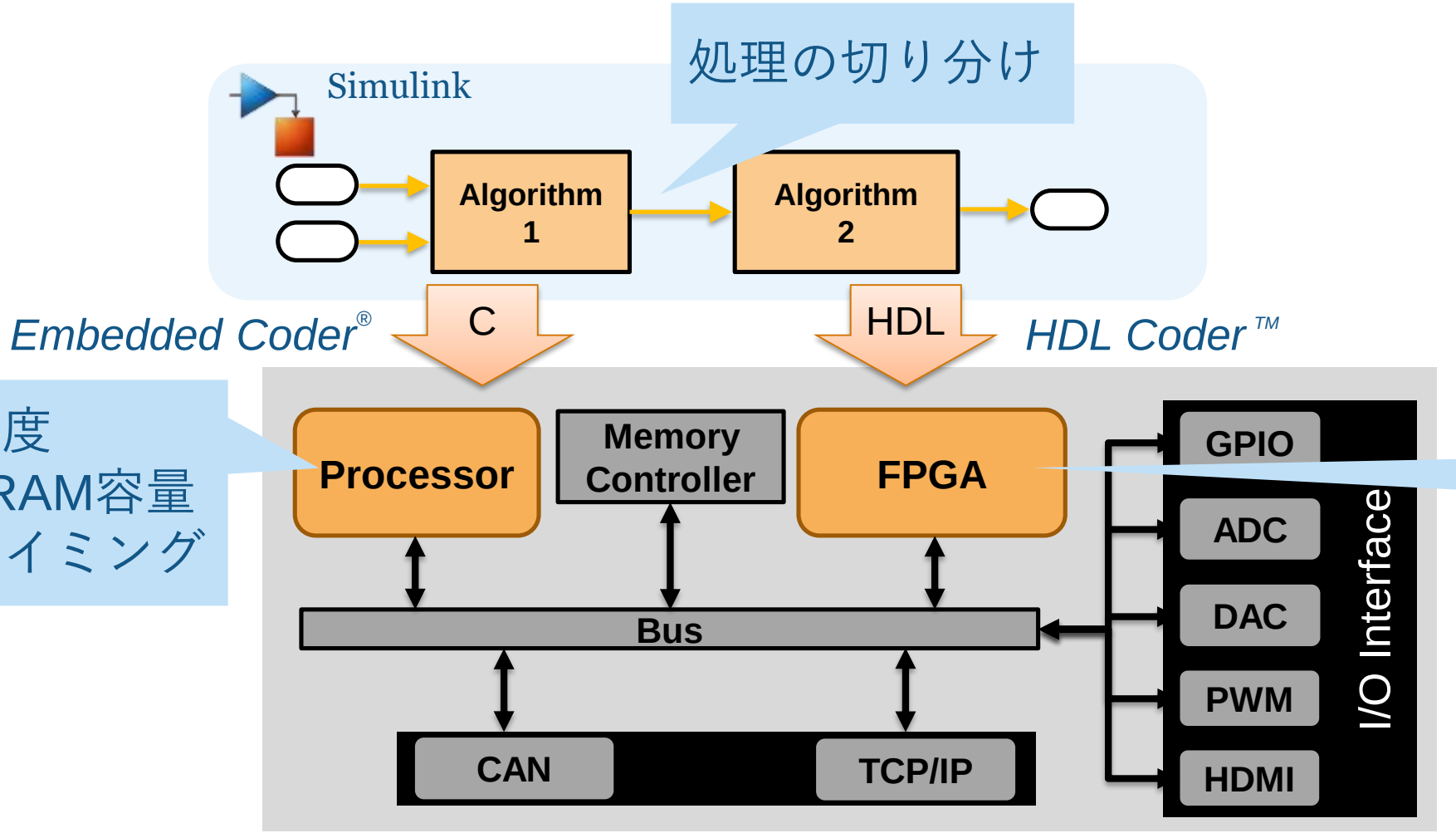
# アプリケーションをSoC FPGAに実装する際の設計要素

- 構成要素
  - Processor
  - FPGA
  - I/O
- 接続要素
  - 内部バス
  - 共有メモリー
- 要素間の通信方法
  - レジスタアクセス
  - 共有メモリ、プロトコル




1. アーキテクチャの探索・検討
2. 各構成要素（コード）の最適化

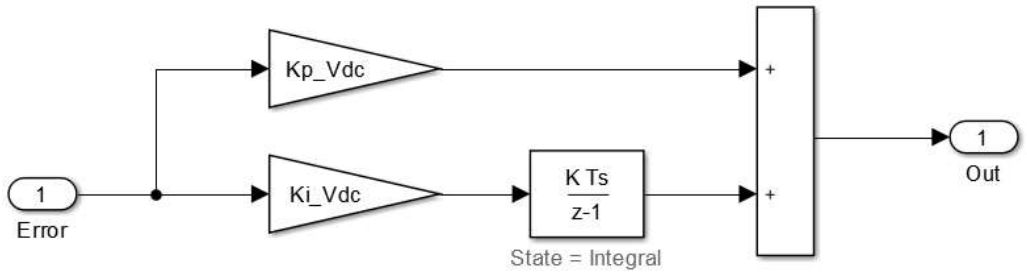
# 各コンポーネントのコード生成で考慮すること



# C vs. HDLコード生成における基本モデリングパターン

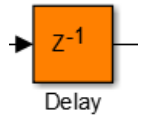
## Cモデリング

- 処理内容を記述
- モデル周期はProcessorの1サイクル周期
  ⇒Delayも1サイクル遅延



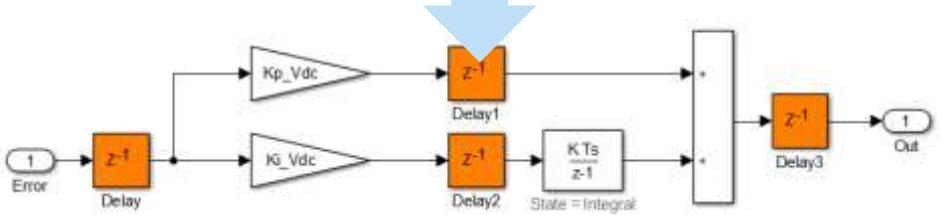

## HDLモデリング

- クロック同期用パイプライン設計
  Delayブロック⇒ Flip Flop回路




- モデル周期 = Clock周期 >> Processor周期
  ⇒Delayは1CLK遅延
- クロック レート パイプライン設定
  ⇒モデル周期 ≠ Clock周期

HDLプロパティ：DistributedPipelining
でDelayを自動挿入

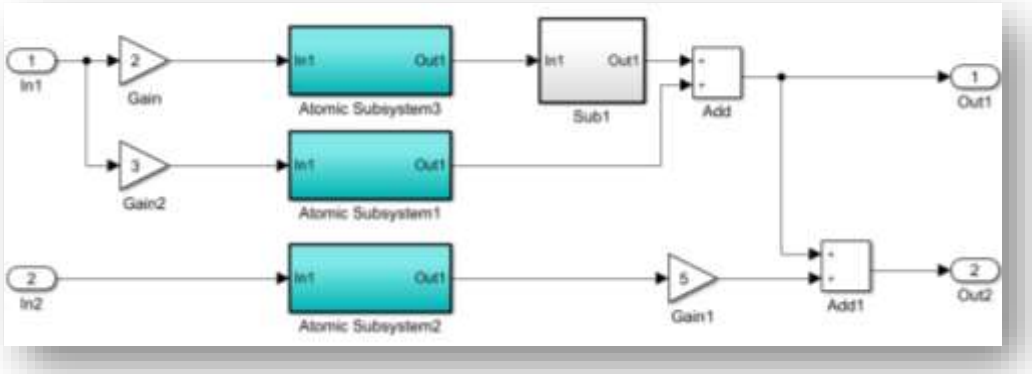


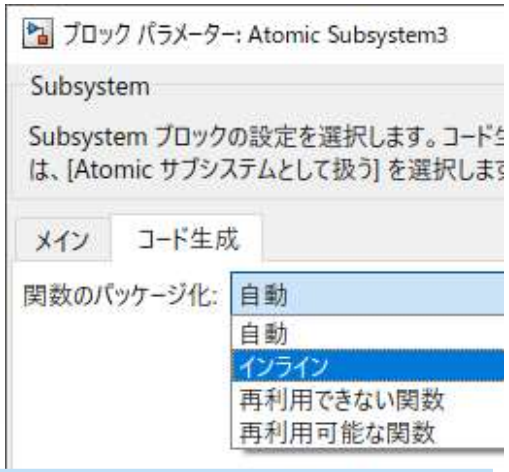

パイプライン設計の省力化

# Cコード生成における同一処理の最適化：処理速度向上

## Cモデリング

- 複数の同一処理
  デフォルト設定：再利用可能関数化されROM削減



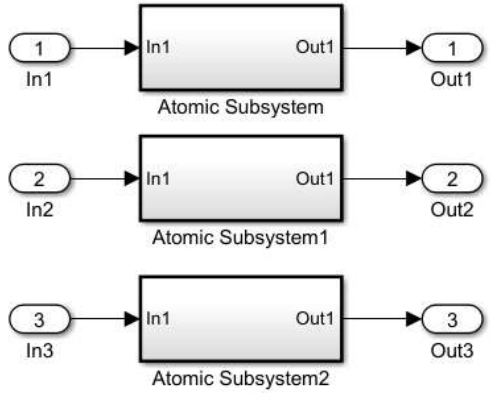

サブシステムパラメータ：関数のパッケージ化
⇒インライン

**ROM増大 / 処理速度向上**

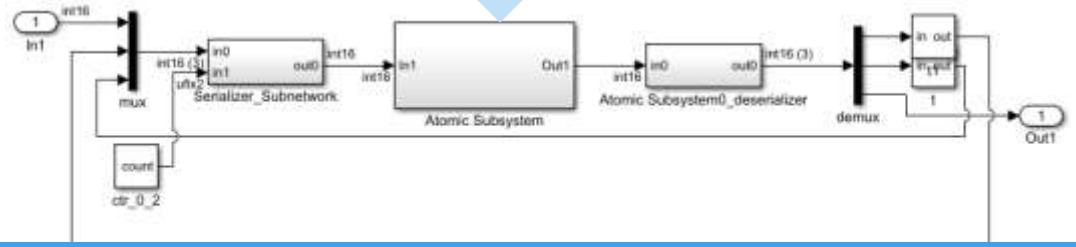
# HDLコード生成における同一処理の最適化：回路リソース削減

## HDLモデリング

- カスケード接続 or 並列接続の同一処理
  ⇒処理回数分の回路リソース消費


HDLプロパティ：SharingFactor
でリソースを時分割で共有化

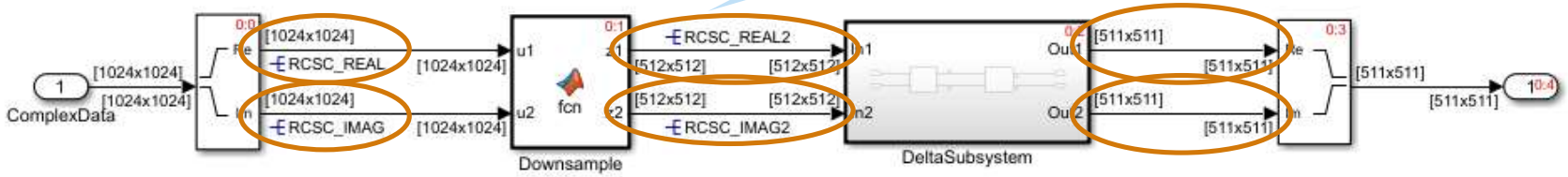



**スループット低下 / 回路リソース削減**

# Cコード生成におけるメモリ消費削減

## Cモデリング

- Simulink.Signalオブジェクトで変数の再利用
  - 同一信号・状態
  - サブシステム間の信号




Signalストレージクラス＞Reusable

**RAM容量削減**

例：Reusable無し

```
out = (2.0 * in + 1.0) + ud;
ud = out;
```

Reusable適用

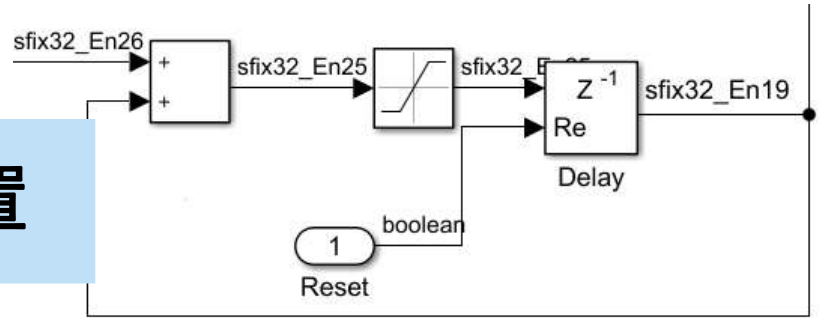
```
out = (2.0 * in + 1.0) + out;
```

# コード生成アドバイザーを使ってモデリングとコンフィグのチェック

## Cモデリング

- コード生成アドバイザーにより目的に応じた最適化
  コンフィグ＞コード生成＞コード生成の目的＞優先順位が設定された目的

# HDLコード生成における固定小数点設計




符号の有無

ビット幅

小数点位置

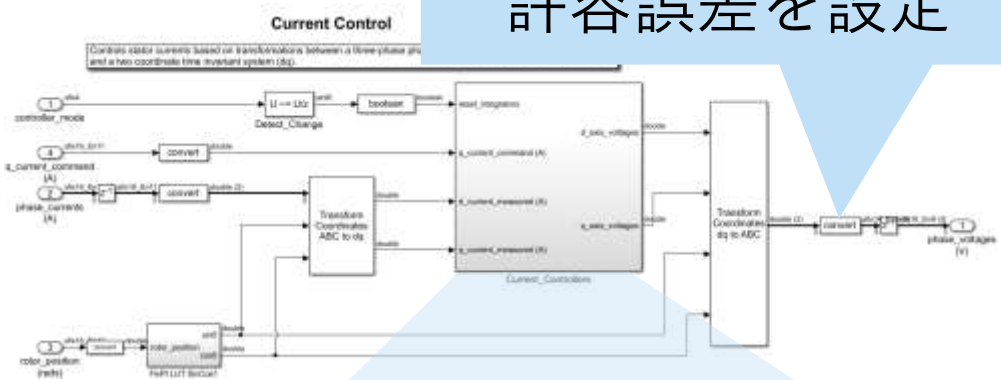
| オーバーフロー | S | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 量子化誤差 |
|---|---|---|---|---|---|---|---|---|---|

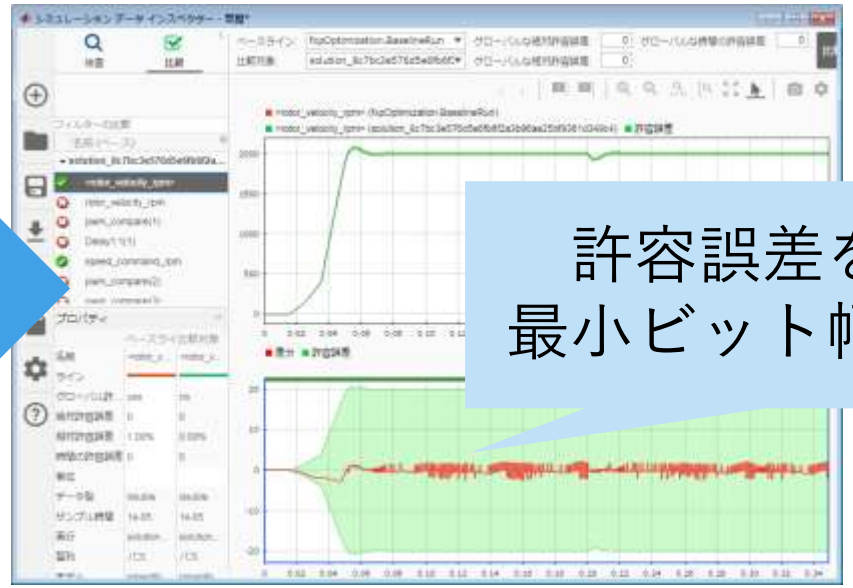
許容誤差範囲で最小のビット幅設計

*Fixed-Point Designer 機能：fxpopt*

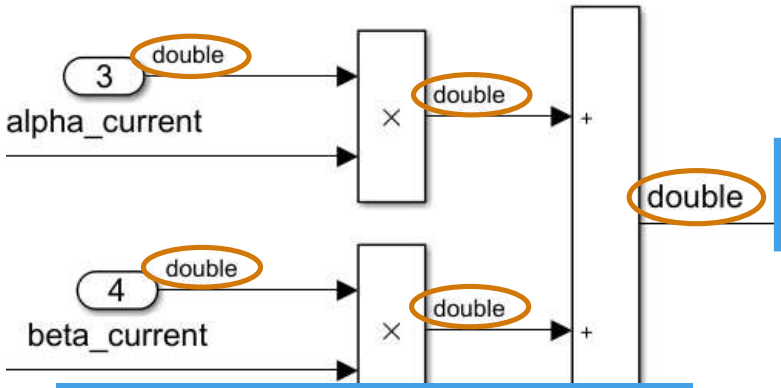
# 許容誤差ベースの固定小数点設定最適化：*fxpopt*

任意ポイントで
許容誤差を設定

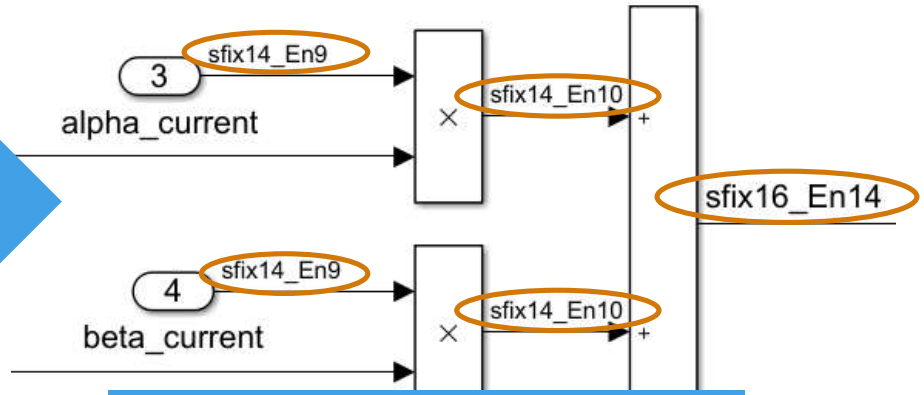


許容誤差を満たす
最小ビット幅に最適化

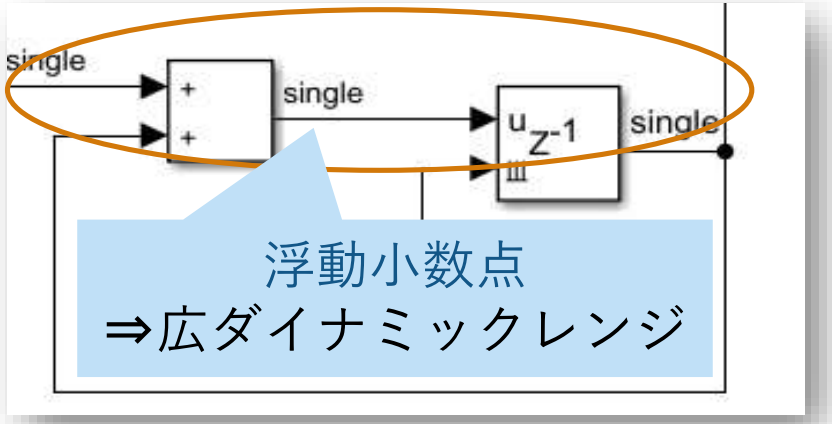



浮動・固定小数点モデル




固定小数点最適化モデル

# ネイティブ浮動小数点でのHDL生成
## *HDL Coder*機能


**ネイティブ浮動小数点**

- ターゲット依存しないHDL生成
- Double / Singleデータ型対応
- 多数の算術演算をサポート
  - exp, log, power, atan/sin/cos…

➢ プロトタイピング開発期間の短縮

➢ 広ダイナミックレンジ、精度

ユーザ事例： DEMCON社はプロト開発期間を1/7に短縮
https://www.mathworks.com/company/user_stories/demcon-reduces-development-time-for-fpga-controlled-surgical-instrument.html

# C / HDL生成モデリングパターンを示したガイドライン

## Cモデリング

- MAABモデリングガイドライン
  - Simulinkヘルプドキュメント
    >> web(fullfile(docroot, 'simulink/modeling-guidelines.html'))
  - https://www.mathworks.com/help/releases/R2019a/simulink/modeling-guidelines.html
- Embedded Coder Tips集
  - 技術サポート宛てにお問い合わせ下さい。

## HDLモデリング

- HDL Modeling Guideline
  - R2019a以降のHDL Coderヘルプドキュメント
    >> web(fullfile(docroot, 'hdlcoder/modeling-guidelines-for-hdl-code-generation.html'))
  - https://www.mathworks.com/help/releases/R2019a/hdlcoder/modeling-guidelines-for-hdl-code-generation.html

※旧バージョンR2014a版は技術サポート宛てにお問い合わせ下さい。

# まとめ

- Processorのタスク、メモリなど含めたアーキテクチャの早期検討
- Cコード、HDLコードのコンポーネントの最適化
- アルゴリズムおよびアーキテクチャのSoC FPGA実装

⇒*HDL Coder, Embedded Coder, SoC Blockset*で実現できます。