CatapultとTLM高位検証を用いたRISC-V Processorでのシステムデザインの加速

Siemens EDA
TSS Calypto Japan
Consultant Application Engineer
酒井　健一

SIEMENS

アジェンダ

- プロセサを取り巻くトレンド
- プロセサモデルを含むシステムレベル検証の分類と目的
  - MatchLibの紹介
- Catapultが提供する最適解の探求とその事例
- まとめ

SIEMENS

プロセサを取り巻くトレンド

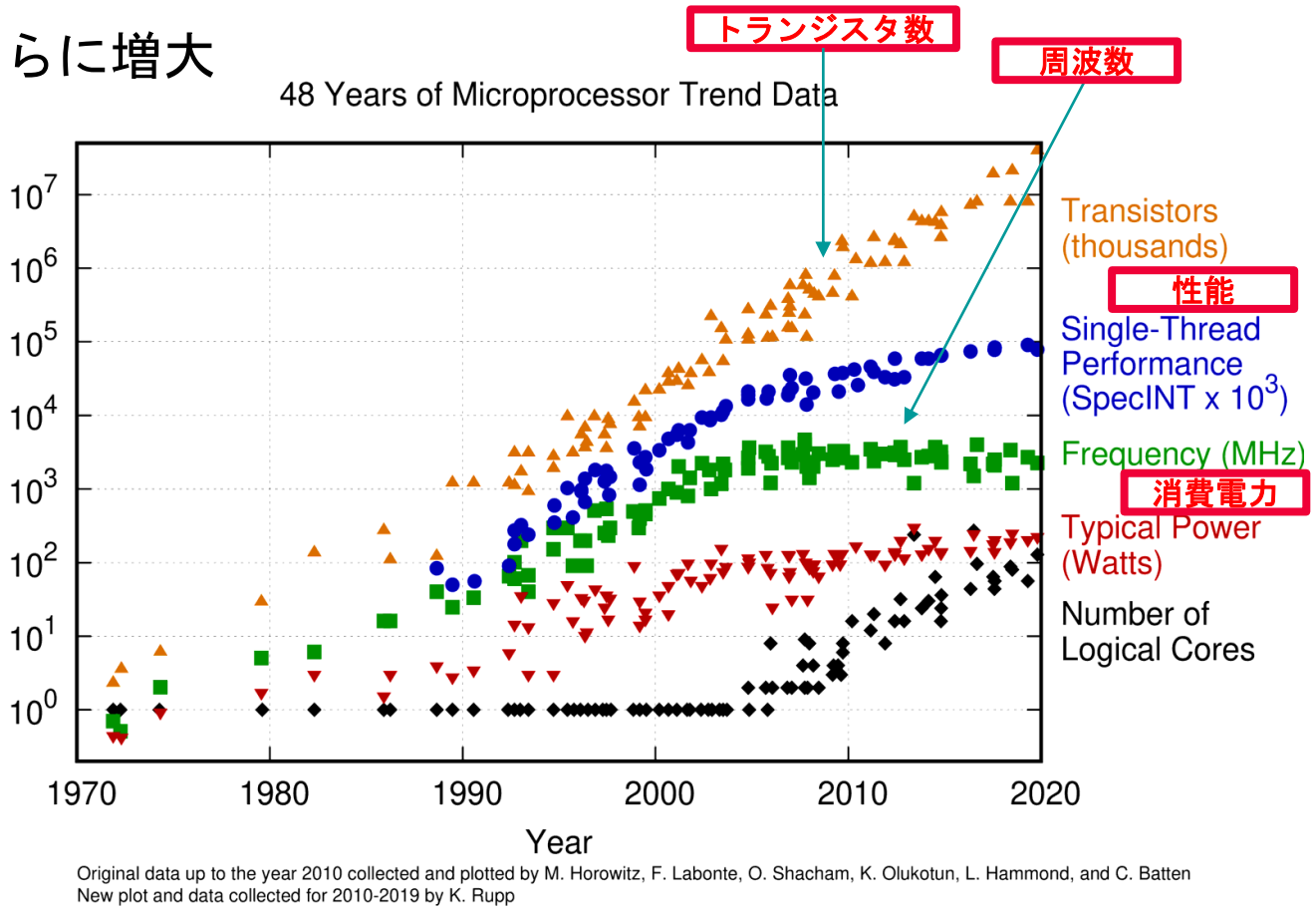
SIEMENS

システム全体の最適化へのプロセサSW要所の性能改善要求

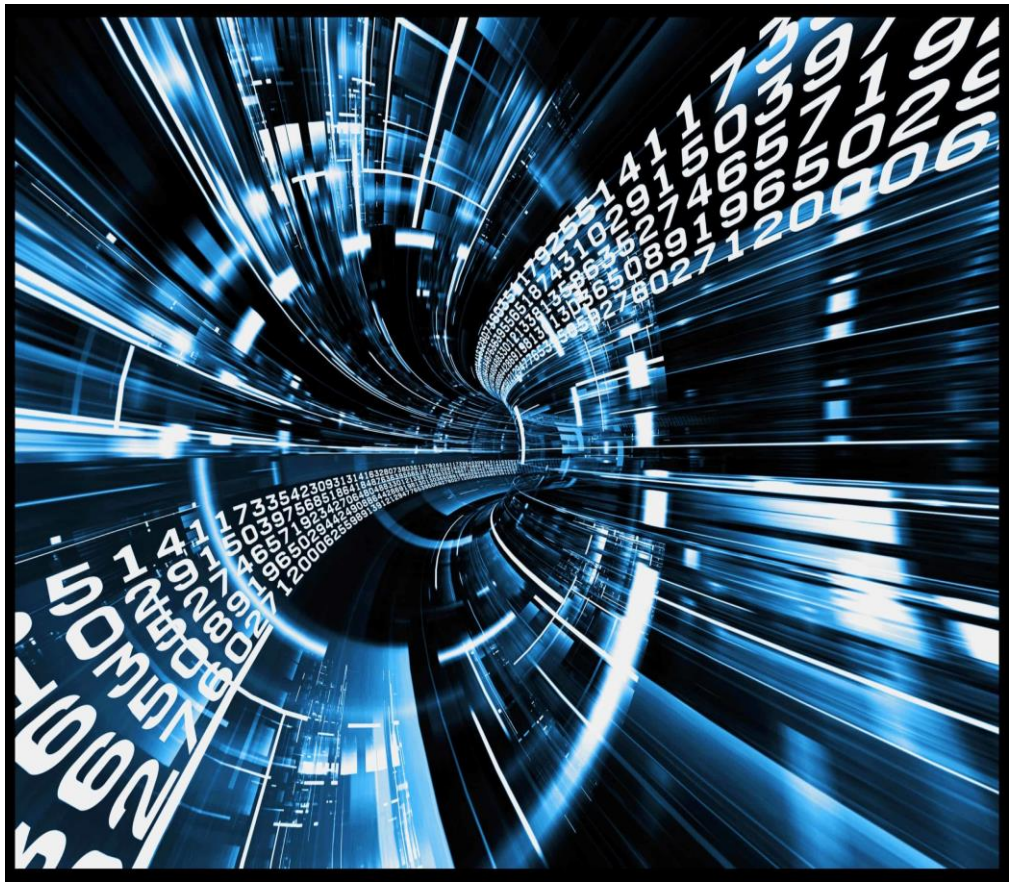
- ムーアの法則の終焉により、微細加工技術の進歩が鈍化
  プロセサの処理能力、エネルギー効率も鈍化

- 要求される処理能力、エネルギー効率は、さらに増大

このギャップを埋めるには？
理想論ではなく、現実的な解決策は？



HWを用いたアクセラレーション



SWはフレキシブル、プログラムが容易、迅速に更新

- 性能とエネルギの面で非効率

HWへの機能の移動は、
性能とエネルギ面で著しい改善をもたらす

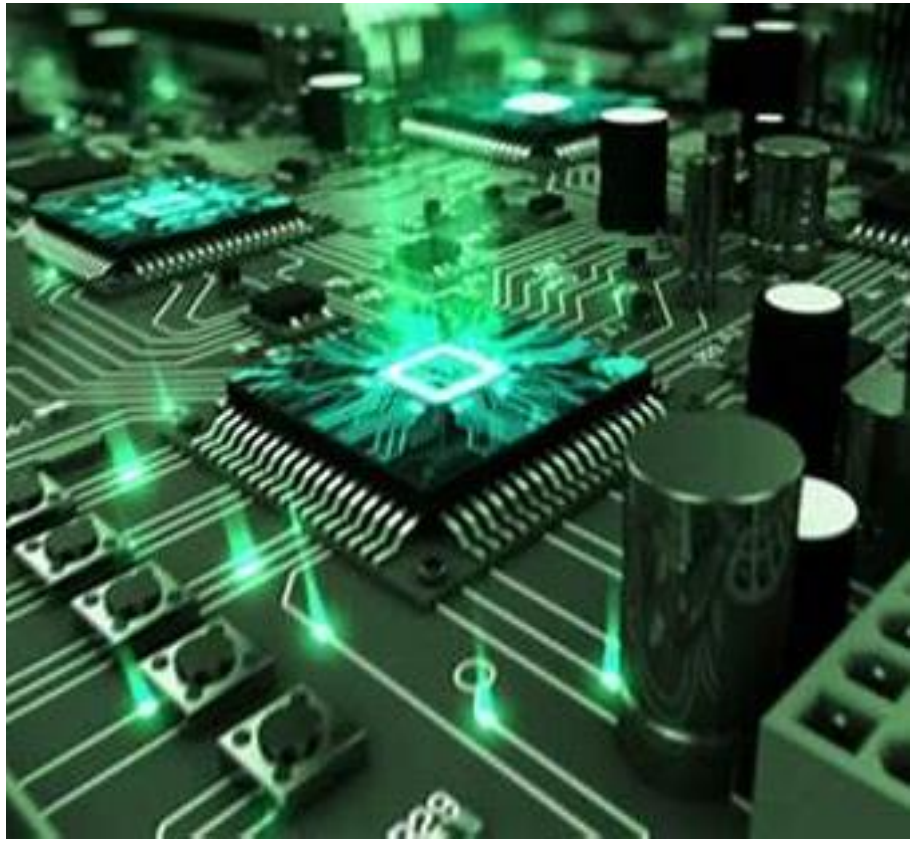
- 10X ～　50Xの改善、 >80%のエネルギ削減

Video, 5G, AI アルゴリズムでは、
CPUが実行する以上にプロセッシングパワーを
必要とする

- これらの分野のアルゴリズムは、SWとして設計
- アルゴリズムは急な変更を伴い
  RTL設計はリスクあり

SIEMENS

HWへの機能のポーティング



HWが高速でより効率が良いか？

- プロセサもHWで構成される
- CPUでの乗算機能　＝　HWの乗算機能

性能改善目的で、並列化を導入

- HWでは並列化の操作を実行することで、SW以上の速度を達成できる

効率のために、回路を調整する必要がある

- HWに必要なものだけを実装することでSWを上回る効率を提供できる

プロセサモデルを含むシステムレベル検証の分類と目的

＋　MatchLibの紹介

SIEMENS

プロセサモデルを含むシステムレベル検証の分類と目的

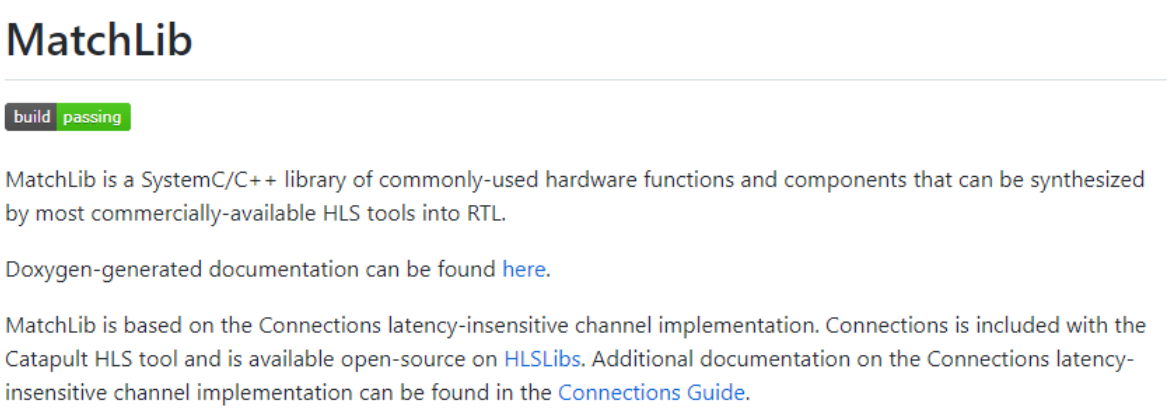
プロセサモデル	インタコネクト	Catapultのコンポーネント	検証の目的・タスク
Native C++	N/A	C++アルゴリズム or C++アーキテクチャ	機能面だけの検証 バスを介さない ポインタ制御でデータ移動
Host Code Execution（HCE）	TLM/MatchLib*	C++アーキテクチャ	インタフェースとバスの **データ移動の考慮**
Fast ISS	TLM/MatchLib*	C++アーキテクチャ	**SWをターゲットCPUにクロスコンパイル実装し、プロセサ性能を考慮したインタフェースとバスのデータ移動の考慮** オープンソースや商用で利用可能 **- > TLM 2.0への接続の鍵**
RTL	RTL	高位合成後のRTL	クロック精度 最終製品での機能・性能検証

MatchLib： 次ページにて解説

MatchLibとは？

- **M**odular **A**pproach **T**o **C**ircuits and **H**ardware **Lib**rary
- MLアクセラレータを作成する際に、NVIDIA Labで開発
  - システムの挙動をシミュレーションするための**高抽象度**からのニーズ
  - (正確でなくとも)モデルの性能　≒　RTL実装時の**性能**　に極力近づけたいニーズ
- 性能と消費電力に対し、多くの異なるアーキテクチャ評価が必要
  - RTLで全てをデザインする余裕は無し
  - 重大な間違いが許容できる余裕は無し

- MatchLibは、オープンソースコード
  - https://github.com/NVlabs/matchlib

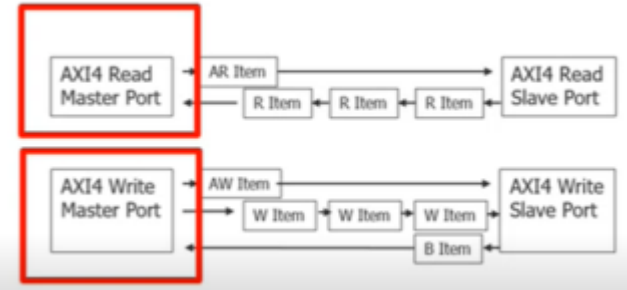


合成可能なMatchLib AXI4 model　と　TLM Wrapper

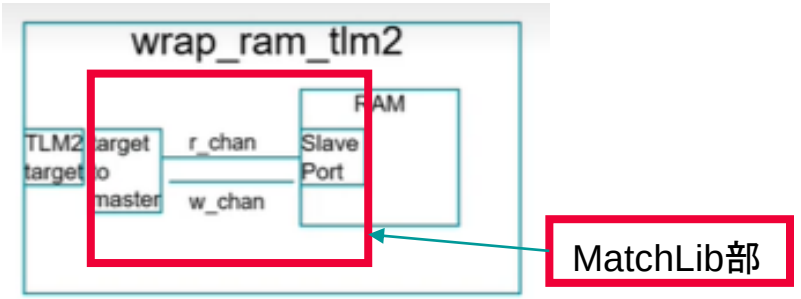
- **AXI4の全5チャネル（ar, r, aw, w, b）のtransactionを正確にモデル化**
  - チャネルでは、ビット精度とビート精度をモデル化
- 高いQoRを保ち高位合成Catapultで合成可能
- スループット精度（ACCURATE_SIM)とFAST_SIMモードをサポート

- TLM < - > Catapult Wrapper Modelで、システム性能検証が早期から実現可能

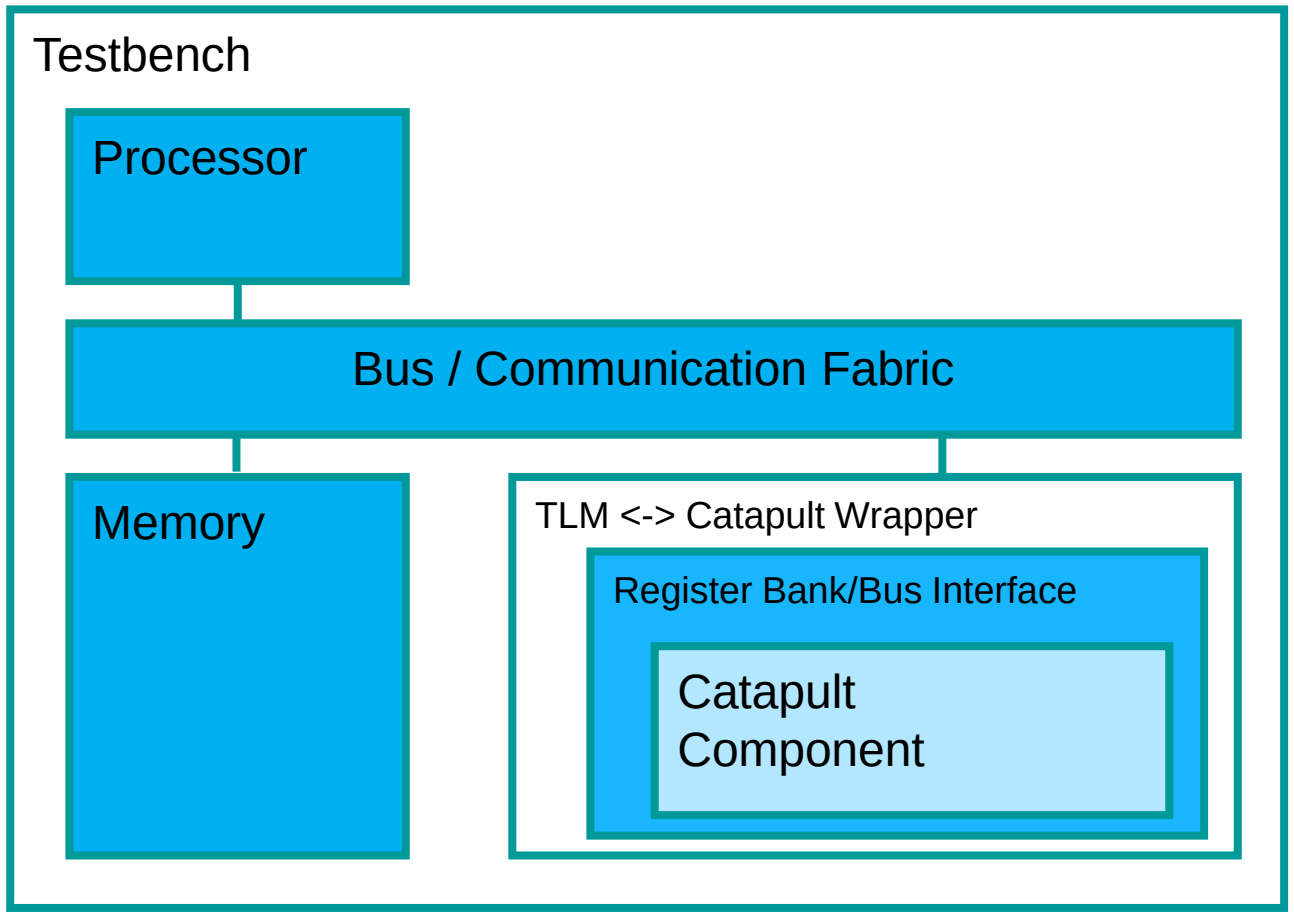
MatchLib　Model



TLM Wrapper model for MatchLib Ram Model



Catapult向けのC++/SystemCのTLM環境への検証実装法



- ビット精度を扱うac_ type から
  TLMへの変換を行うトランザクタで
  Catapultのコンポネントをラッピング
- それ以外のシステムは
  TLM-2のコンポーネントで実装
  - コンカレントな実行動作

C++/SystemC

TLM-2

SIEMENS

システムレベルシミュレーションの結果 (Yolo Tiny Inference)

検証目的	プロセサ	インタコネクト	アクセラレータ	RunTime (sec)	検証内容	速度	消費電力効率
機能検証 データと オペレータサイジング	Native C++	N/A	C++ Algorithmic	12	コアの アルゴリズム検証		
データ移動を伴う SWインタフェース/ 同期検証	Host Code	CONNECTIONS_FAST_SIM & TLM/MatchLib	C++ Architecture	1,027	インタフェース、 データフロー検証		
	Host Code	CONNECTIONS_ACCURATE_SIM & TLM/MatchLib	C++ Architecture	6,455			
ターゲットCPU SWインタフェース 性能検証 (approximate)	Fast ISS	CONNECTIONS_ACCURATE_SIM & TLM/MatchLib	C++ Architecture	7,272	クロスコンパイル コードでの SWドリブン検証	データ移動に依存 桁違いの性能差 システムの中間 データ格納・アクセスに大きく依存	
最終性能と機能検証	RTL	CONNECTIONS_ACCURATE_SIM	C++ Architecture	97,329			
	RTL	RTL	RTL	(est) 2,600,000		クロック 精度	PowerPro +/- 15%
	Gate, post P&R	Gate, post P&R	Gate, post P&R	不明 アクセラレータが必要			PowerPro +/- 2-5%

検証の目的に依るが、SWドリブンの検証では時間が掛かる

システムレベル検証からの考察

1．Host Code / ISS or QEMUモデル　＋ MatchLib TLM高位検証により、
ソフトウエア（プロセッサ処理）/データ移動のボトルネックを早期の時点で探ることが重要

2. 上記ボトルネックを探ることで、アクセラレーションの候補を見つけ出せる

3．アクセラレーションの手段として、高位合成によりアルゴリズムやアーキテクトの探求により、
システムの最適解を見つけ出せる可能性が最大限に高まる

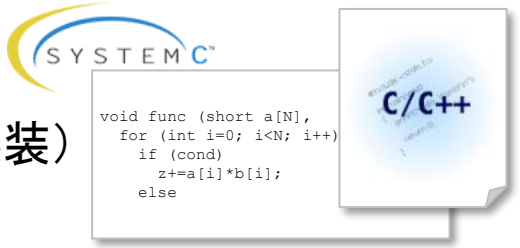
Catapultが提供する最適解の探求とその事例

SIEMENS

Catapult HLSが提供する最適解の探求のための機能

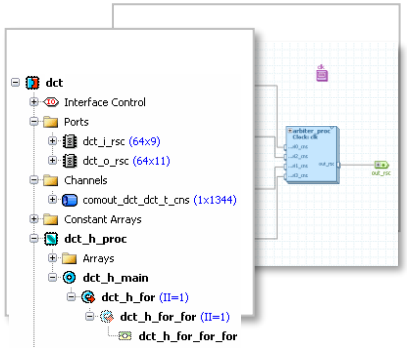
ユーザへのアーキテクチャ制御

- 並列化、スループット、面積、レイテンシ（ループの展開やパイプライン実装）
- メモリ vs レジスタ (リソース配置)



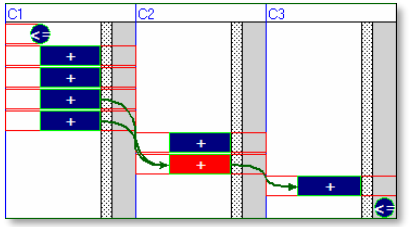
適用した制約でのアーキテクチャ探索と実装

- ソースコードの変更を伴わない



自動化された算術最適化とビット幅最適化

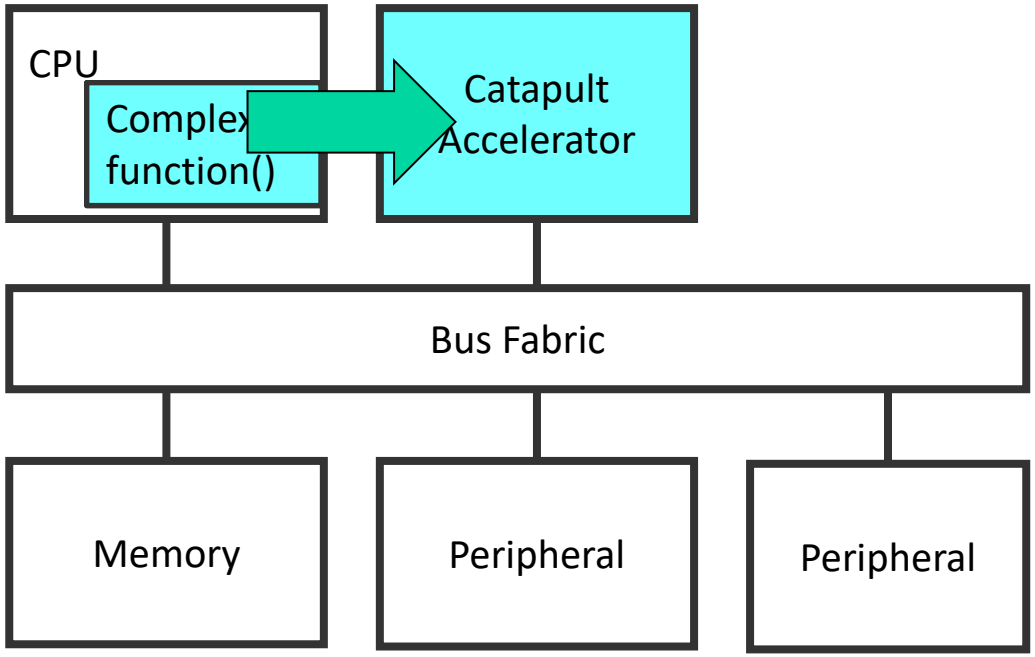
- ビット精度タイプにより、出力に算術精度の伝搬を可能に



FPGA とASICの双方にプロセスアウェアなスケジューリングを提供

- Area/Latency重視のデータパススケジューリングを実行
- I.P. 再利用のRTLテクノロジのペナルティを排除

アクセラレーションの狙い



- CPUからHWに負荷のある機能の負荷を低減
- 性能の改善
- エネルギ消費の低減

プロファイルの取得と理解は大切！　アクセラレータ化前に…

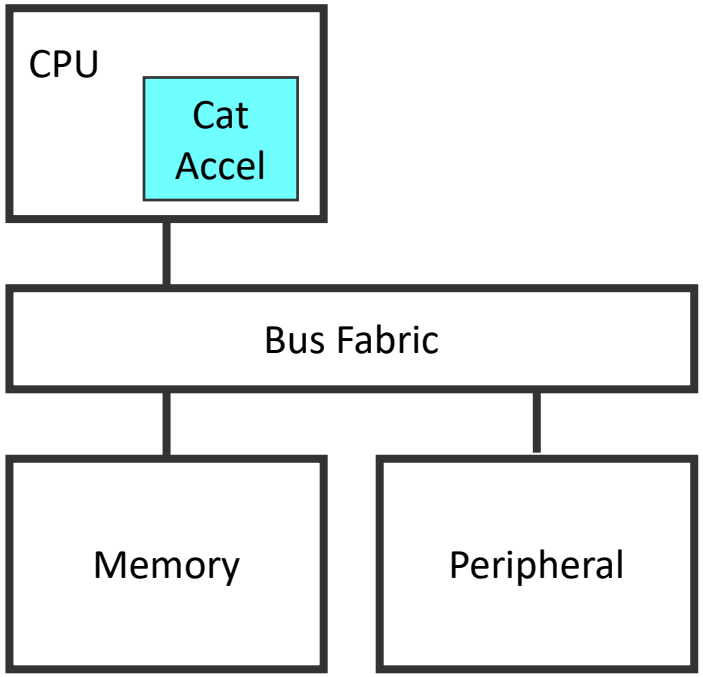
Time Profiler 〉 Profile 〉 Root

Weight	Self Weight	Symbol Name
5.23 s 100.0%	0 s	▼yolo (48794)
5.23 s 100.0%	0 s	▼Main Thread 0x461821
5.23 s 100.0%	0 s	▼start libdyld.dylib
5.23 s 100.0%	0 s	▼main yolo
5.23 s 100.0%	0 s	▼yolo_tiny yolo
5.21 s 99.5%	4.99 s	▶conv_2d_sw yolo
8.00 ms 0.1%	8.00 ms	leaky_relu_sw yolo
8.00 ms 0.1%	8.00 ms	max_pool_sw yolo
6.00 ms 0.1%	6.00 ms	bias_add_sw yolo

何処？
何が？

- 計算上ボトルネックになる箇所を理解する
  - 小規模だが頻繁な繰り返しのある関数は、
    新規のインストラクションまたはco-processorに変更可能な候補
  - 大規模な関数は、バスベースのペリフェラルに変更可能な候補
    ∵　**中間データの保持やデータ移動**を伴うことになる

アクセラレータの接続例 – 新規のインストラクション



- ❑ CPUと密にインテグレート
  - バスファブリック上のトラフィックが無い
  - 最速のオペランドアクセス
- ❑ **RISC-V では、インストラクションセットの拡張をサポート**
- ❑ 小規模だが複雑なオペレーション
  - 数サイクルに限定
  - 内部ステートを持たない

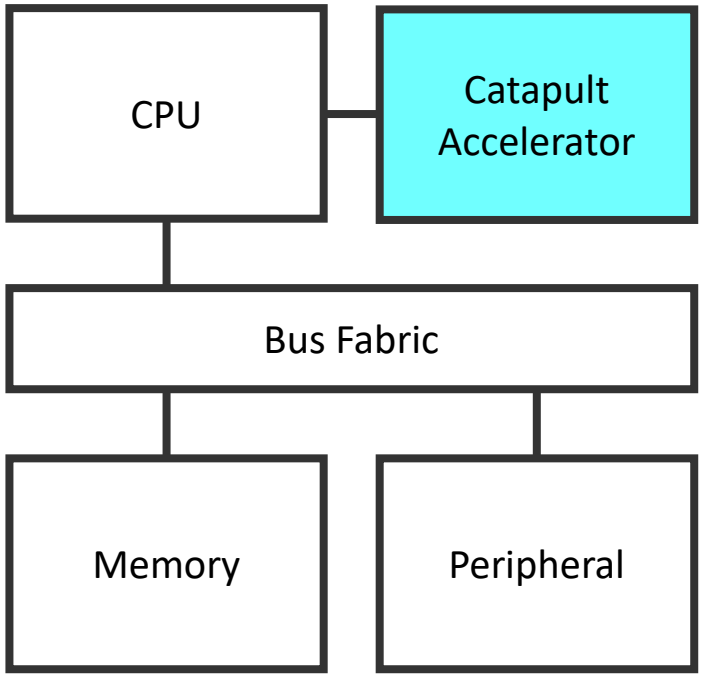
例:

- 複雑なSIMDインストラクション
- 複素数データタイプ演算

新規opcodeでのアクセスは、通常はC ”asm”ディレクティブを使用

```
asm(“CX3 %2, %0, %1”
            : “r” var_in_0, var_in_1
            : “=r” var_out_2)
```

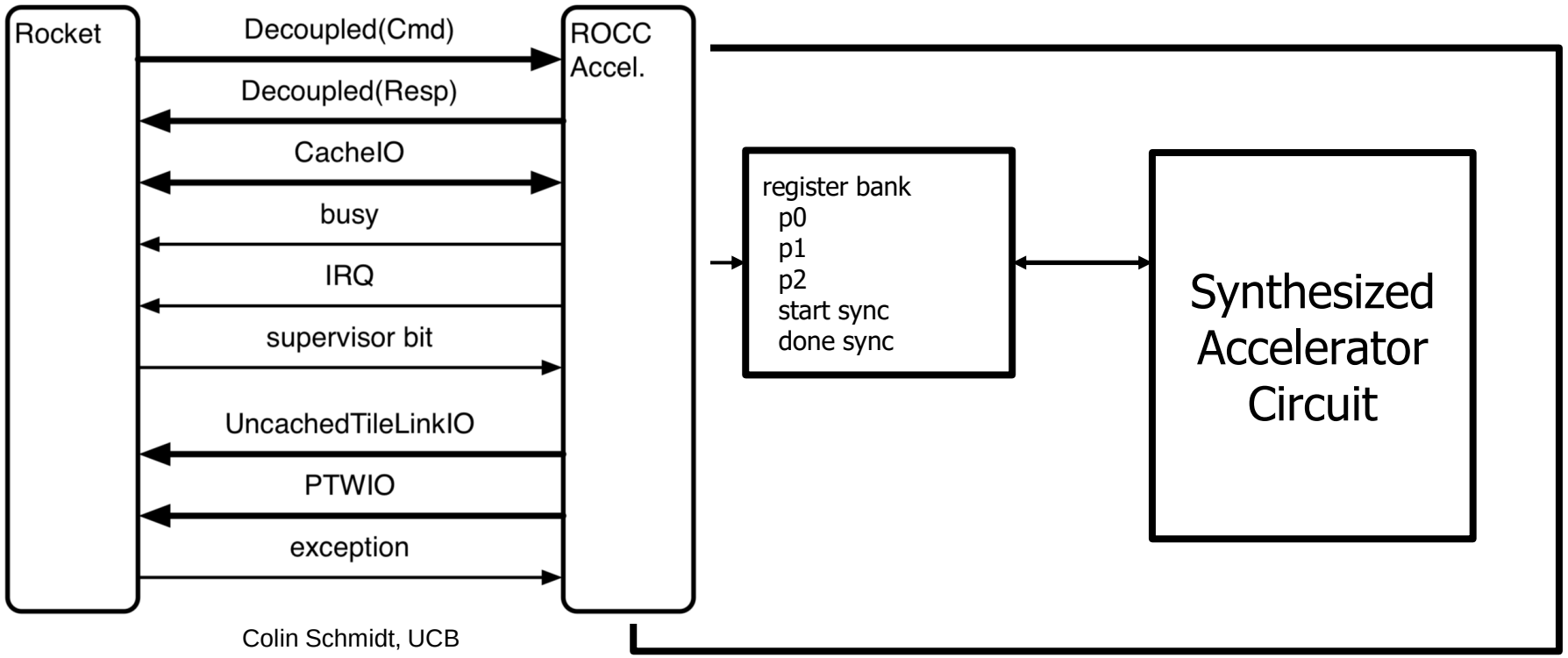
アクセラレータの接続例– Co-processor (RoCC)



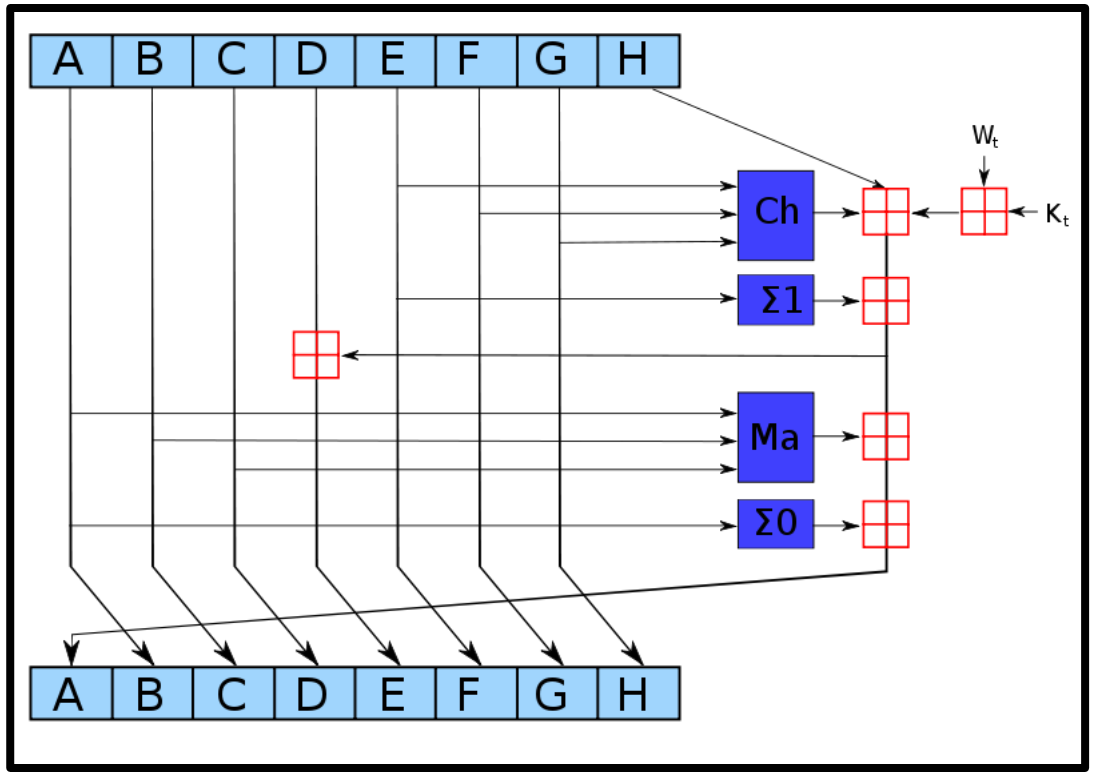
- ❑ CPUからco-processorインタフェースへデータを渡す
  - バスファブリック上のトラフィックが無し
    高速のアクセス
  - CPUだけがアクセラレータにアクセスできる。
- ❑ Rocket Core Custom Coprocessor Interfaceを使用
  - メモリの一貫性の問題を避けるため、メモリアクセスの実行は行わない
- ❑ 中程度の複雑度のアルゴリズムでは、SWと組み合わせたデータを使用

  例:
  - フィルタ
  - エンクリプション
  - ハッシュ

アクセラレータの接続例– Co-processor (RoCC)

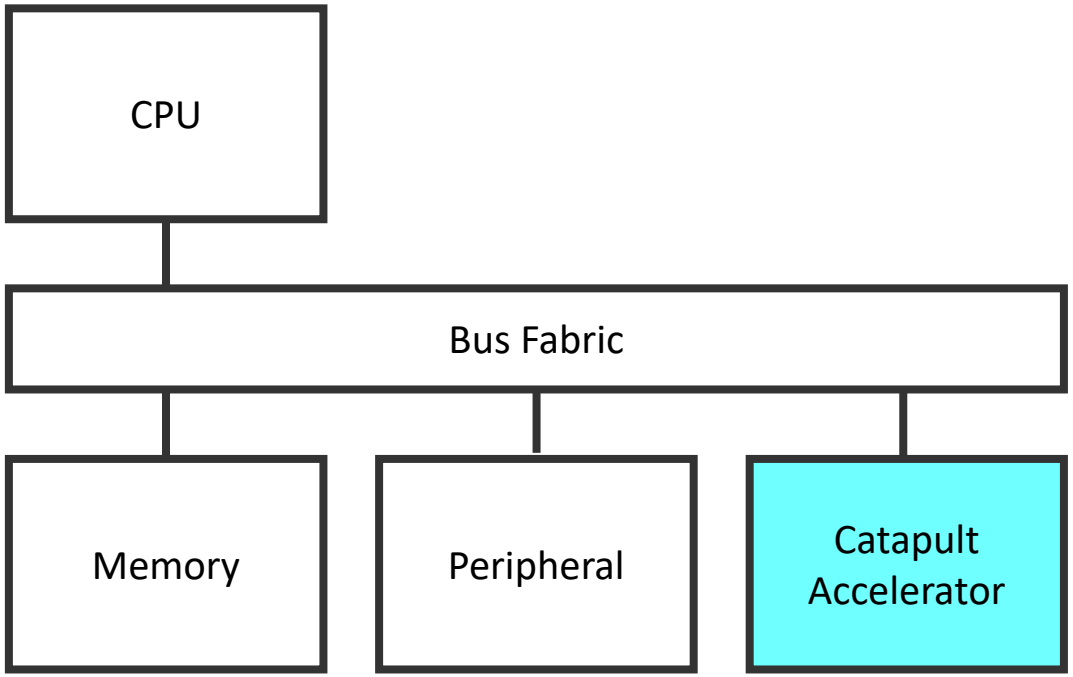


SHA-256 Hash Algorithm



- ❑ HW化により演算の複雑性を緩和
  - 数100万のビット演算
  - 高並列化
  - データ移動は制限
  - Hashのサイズ依存の性能
- ❑ SWと比べたHW性能比
  - 25X 高速
  - 49X の低エネルギ（hash当たり）

アクセラレータの接続例- Slave

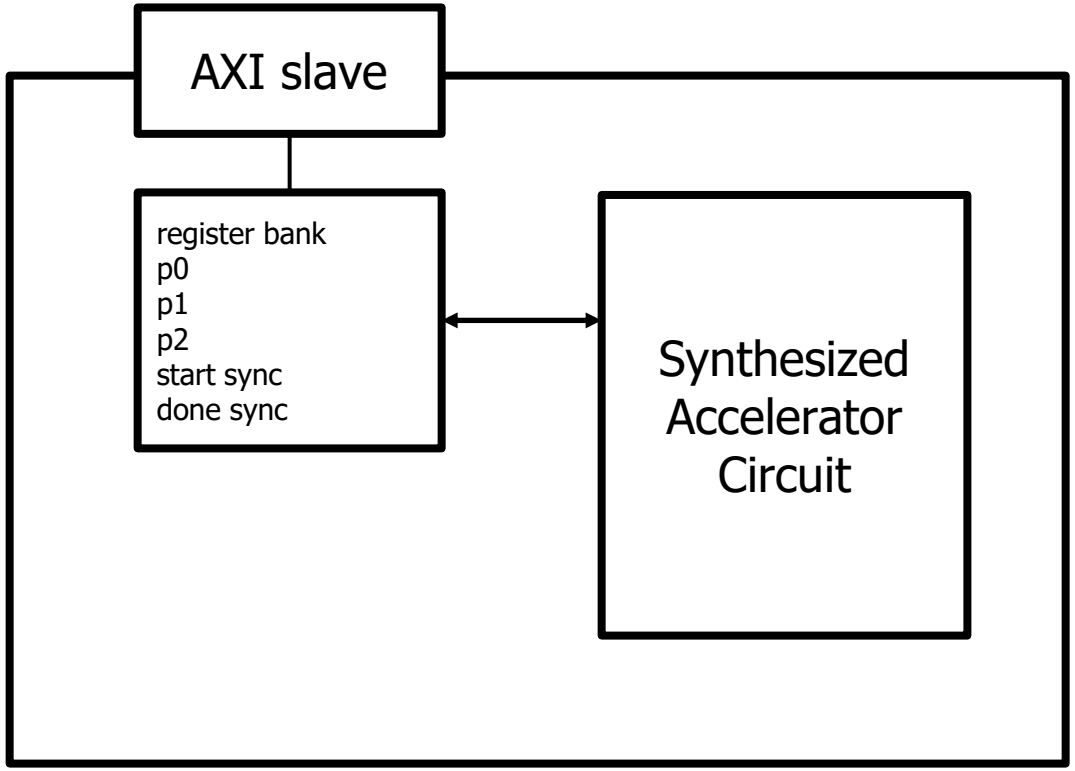


- ❑ データはCPUからアクセラレータへ渡される
  - ローカルメモリも実装可
  - キャッシュと仮想/物理アドレスの問題は無し
  - 低い複雑度のアルゴリズムでより高速
  - メモリマップドレジスタインタフェースを介してアクセス
- ❑ 低い複雑度の演算を軽減

例

- フィルタ
- エンクリプション
- ハッシュ

アクセラレータの接続例- Slave



複雑なAIアプリケーション　2D Convolution Algorithm

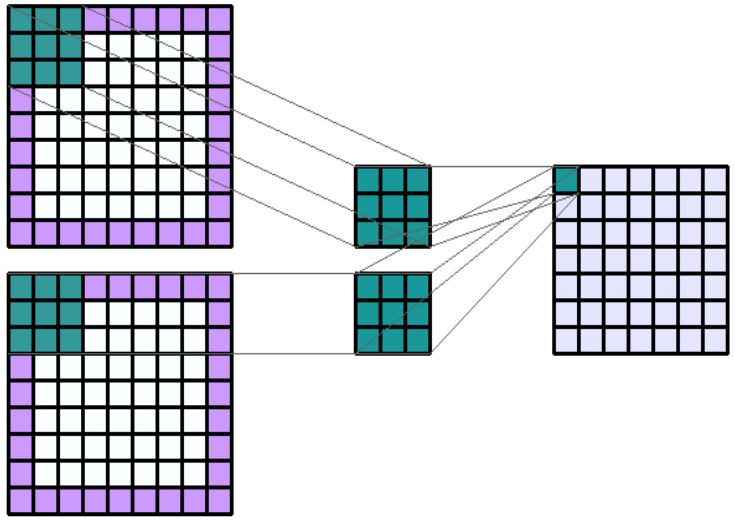
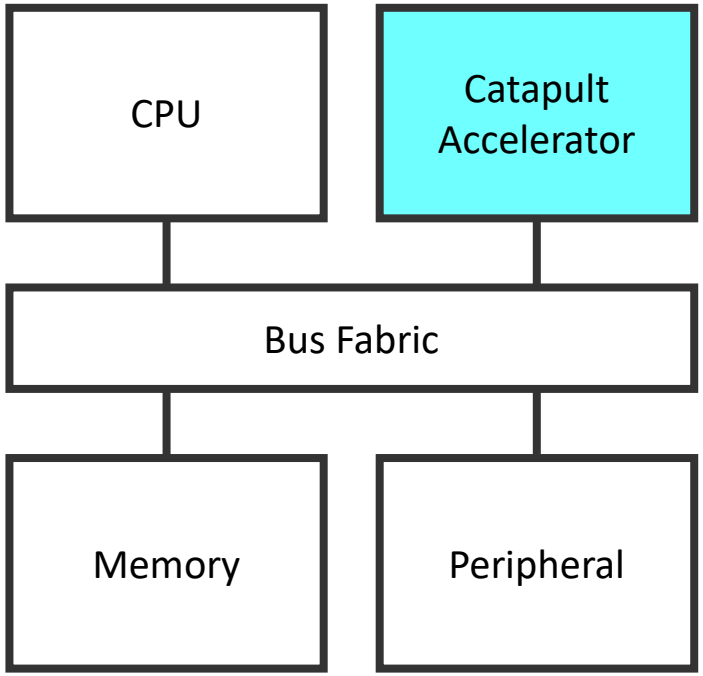


image source: towarddatascience.com

- ❑ 多くのAIと画像処理アプリケーションで使用
  - データ要素当たり数千の演算
  - 高並列性
  - データ量に依存する性能
  - 配列サイズとFeatureマップの数とフィルタの数に大きく依存

- ❑ SWと比べたHW性能比
  - 340X 高速
  - 480X の低エネルギ（演算当たり）

アクセラレータの接続例- Master

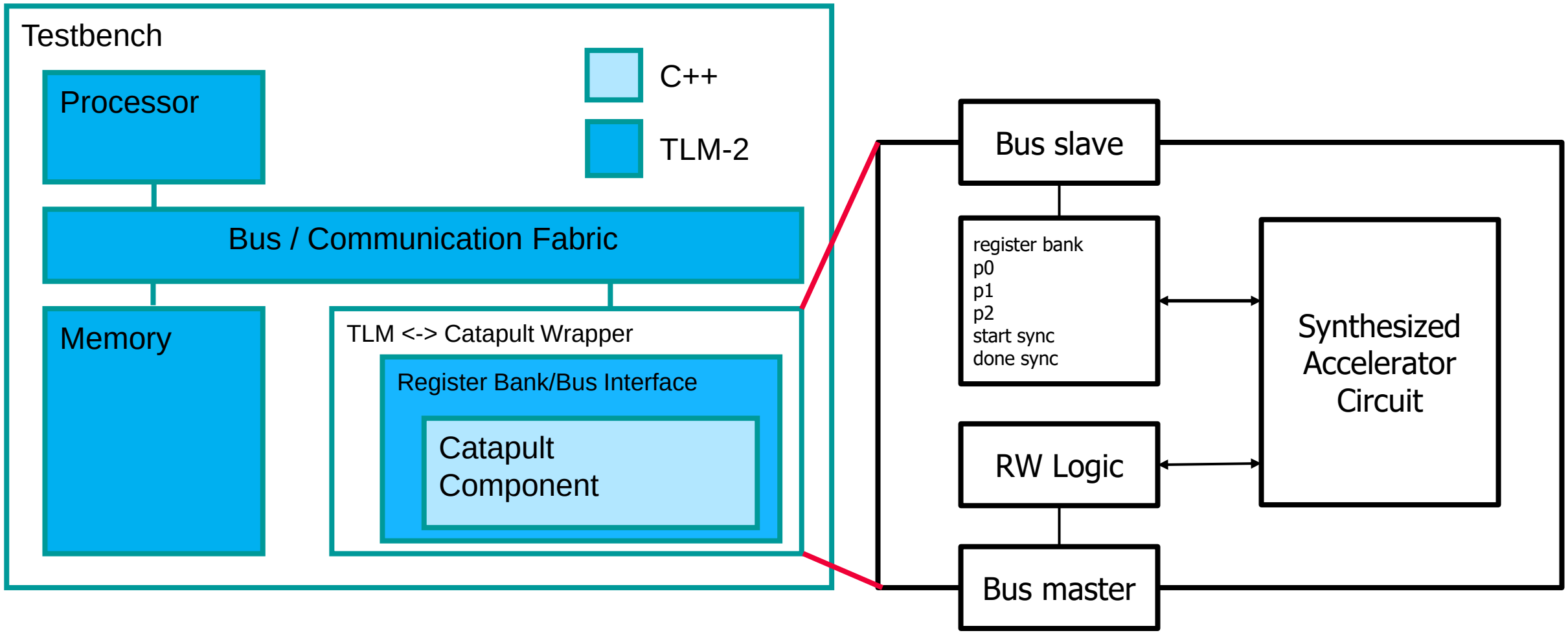


- ❑ アクセラレータがプロセサのデータメモリに直接アクセス可能
  - 入力の読み込み、出力の書き出し
  - 仮想/物理アドレスのキャッシュを考慮する必要あり
  - データの移動は性能を制限する
- ❑ コントロールと同期化のためのスレーブインタフェースを実装
  - メモリマップドレジスタインタフェースを介してアクセスされる
- ❑ 大規模で複雑な演算の負荷を低減する

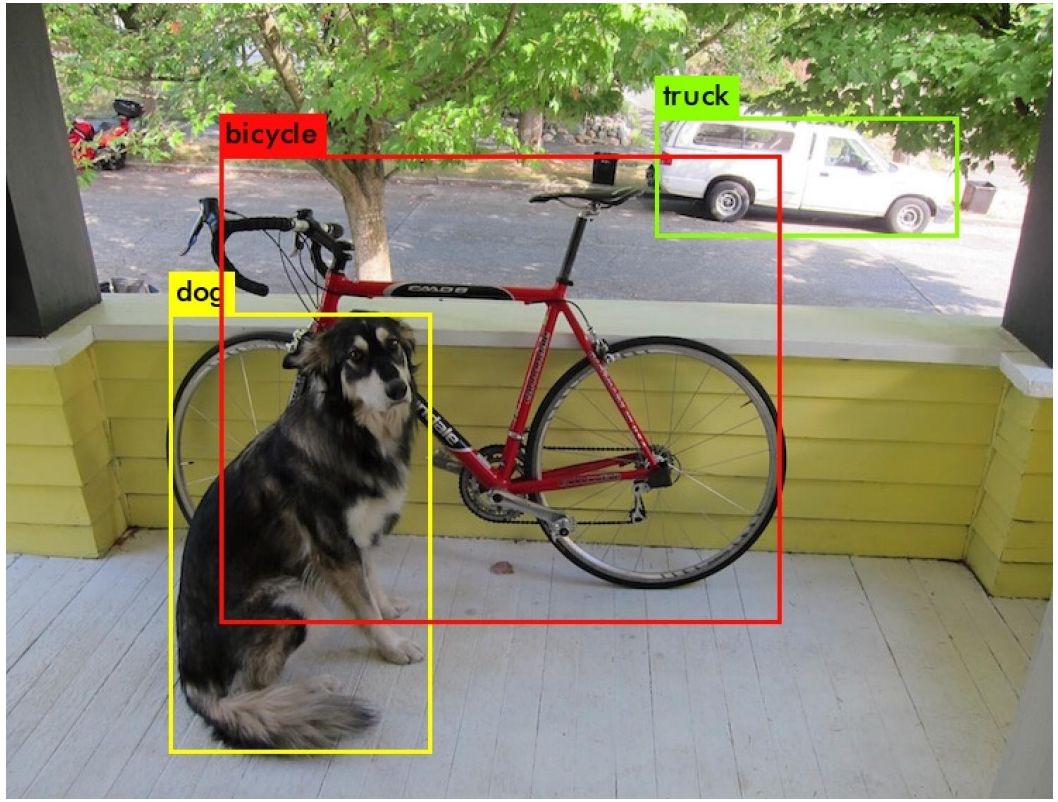
例:

- 推論アルゴリズム
- 画像 or ビデオプロセッシング
- パケットフィルタリング

アクセラレータの接続例 - Master （+ Slave）



Yolo Tiny 物体認識アルゴリズム



- ❑ 高程度の演算の複雑性
  - 推論に~34億の MAC
  - 高並列性
  - 大きな中間データセット
- ❑ SWとの性能比較
  - 1,123X 高速
  - 1,480Xの低エネルギ（推論当たり）

各実装方法での性能・エネルギ比較

Algorithm	Interface	SW perf/ HW perf	SW energy/ HW energy
bfloat	Instructions	N/A	N/A
SHA-256 hash	Co-Processor	25	49
conv2d	Bus slave	340	480
Yolo Tiny	Bus master	1,123	1,480

高位検証での検討と高位合成によるHW化の効果

Performanceの数値では
プロセサと同期のデータ移動のon/offのオーバーヘッドを含む

- ❑ 大規模の関数で、大きな性能の改善が得られる
  - 並列化が必須　シリアルのアルゴリズム処理では効果が出ない

まとめ　– Catapultを用いたシステムデザインの加速 –

- プロセサとSWの影響を理解するため、Host Code / ISS or QEMUモデルを用いた高位検証が重要
- プロセサモデルに、CatapultのMatchLib + TLMコンポーネントを組み合わせることで高位合成前後の双方で、機能的な検証、性能、面積、電力をシステムレベルの全体で検証が可能
  - 検証のゴールに合う適切なレベルの抽象度の選択が鍵
- Catapult HLS は、システムデザインを加速するための必要不可欠なソリューションを提供
  - SWからHWへアクセラレーションでは、Catapultがベストなソリューション

Early Birdウェビナーシリーズ - 高位設計ソリューション
シーメンスEDAジャパン　https://www.mentorg.co.jp/events/early_bird/

Contact

Siemens EDA
TSS Calypto Japan
Consultant Application Engineer
酒井　健一
E-mail: Kenichi_sakai@mentor.com

SIEMENS