codasip

貴方だけのRISC-Vストーリー
競合に勝つための差別化設計

#DesignForDifferentiation

2022/11/17 | Tokyo, Japan

コダシップ , ジャパン カントリーマネージャー
明石 貴昭

MNIST(手描き数字認識)速度比較 [標準RISC-V 対 カスタマイズRISC-V]

コダシップ グループ

プロセッサ開発を
よりシンプルに
より速く
より安く

コダシップとは？

- RISC-VプロセッサIPとプロセッサ設計自動化ツールのリーディング プロバイダー
- 2014年にチェコ共和国（EU）ブルノで設立
  - 10年にわたるブルノ工科大学でのプロセッサ設計自動化の研究成果をもとに起業
  - 2015年 RISC-V International創設メンバー (以前の RISC-V Foundation), www.riscv.org
  - 2015年11月に世界初のライセンサブルRISC-VプロセッサIPを発表
  - 20億個以上のCodasip RISC-V プロセッサが出荷済み
- 本社、R&Dセンターは、 EU圏
  - 約180名の社員（昨年度比200%）
  - 世界に営業拠点有

コダシップのソリューション

codasip
RISC-V PROCESSORS

選択:

アプリケーションに適した
RISC-Vプロセッサ コアを

- 低消費電力、高性能な組み込み型
- アプリケーション・プロセッサ
- RISC-V 準拠
- コンフィギュラブル

カスタム:

Codasip RISC-Vプロセッサを
ベースとしてCodasip Studioで

- カスタム命令を追加して
  アルゴリズムを高速化
- 既存のRISC-Vプロセッサを
  カスタマイズのベースとして使用

codasip
STUDIO

作成:

高レベル記述を用い
プロセッサをゼロから

- 性能、面積、消費電力を考慮した
  設計のファインチューニング
- ソフトウェアとハードウェアの開発
  キットを自動生成

コダシップを選んだお客様

codasip
RISC-V PROCESSORS

TRINAMIC
Now part of Maxim Integrated

LATTICE SEMICONDUCTOR

vidtoo 微迪兔

VITEC

Kameleon

MYTHIC

DONGWOON
(주)동운아나텍

analogix
Bring Vision to Life

Microsemi

UBILITE

NeuLinker

alhua
TECHNOLOGY

codasip
STUDIO

MOBILEYE

SIGMA DESIGNS

Rambus

LUMISSIL Microsystems
A Division of ISSI

codasip

RISC-V PROCESSORS

拡張可能なコア・ファミリー

既製プロセッサのポートフォリオ

- **直ぐ**に利用可能
- 検証済み、**テープアウト品質のIP**
  - お客様によるIPの検証は不要
- すべてのコダシップ プロセッサはRISC-Vに準拠しています
  - RISC-V特権仕様を実装
  - RISC-Vデバッグ仕様の実装
- **業界標準**インターフェース
  - 命令バス、データバス用AMBA
  - デバッグ用JTAG (4ピン/2ピン)

1 SERIES

電力効率と面積効率の良い**3段パイプライン**

圧縮命令で最小のコードサイズを実現

16レジスタ内蔵命令セット (RV32E)

最小面積の**シーケンシャル乗算器**

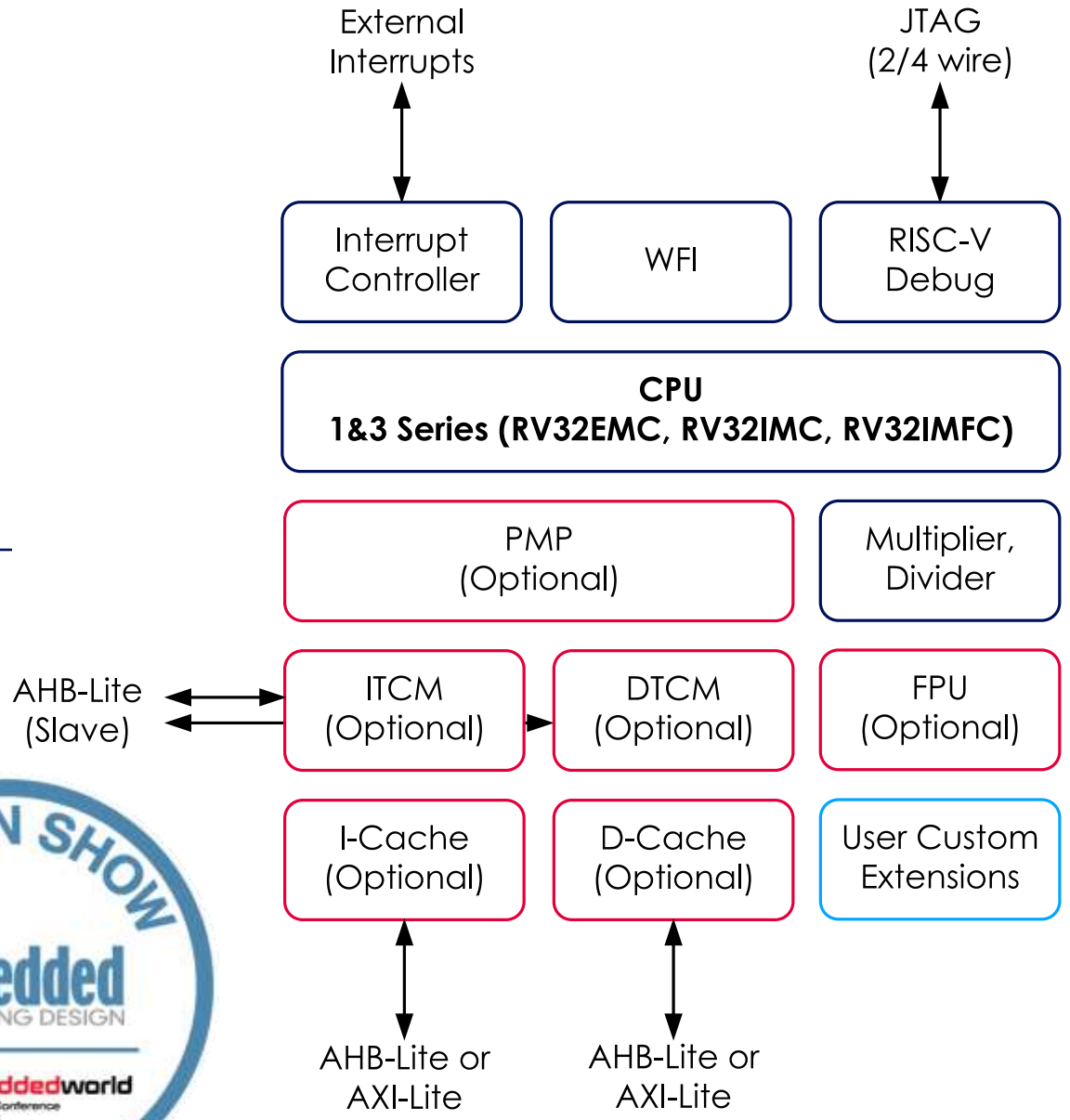
3 SERIES

電力効率と面積効率の良い**3段パイプライン**

圧縮命令で最小のコードサイズを実現

標準的32レジスタ内蔵命令セット (RV32I)

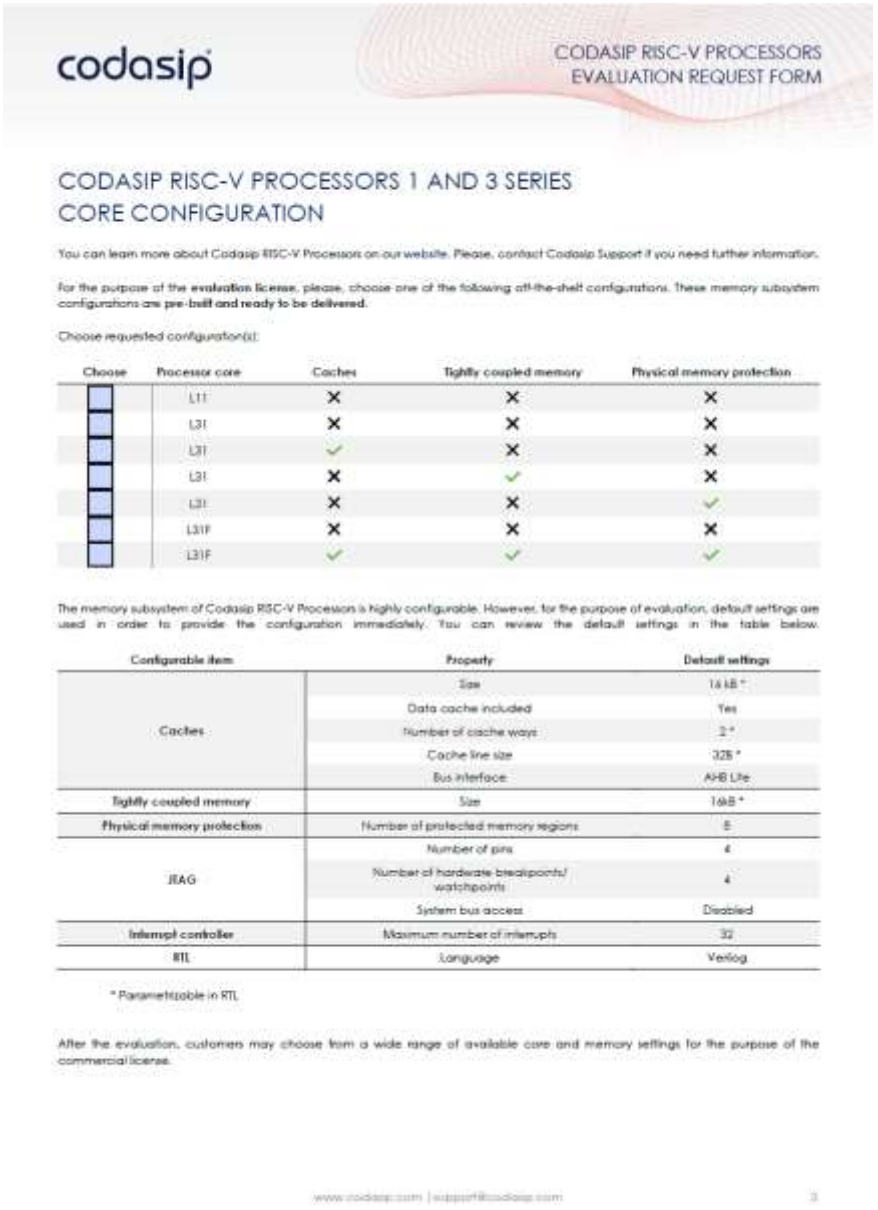
高性能な**並列乗算器**



幅広いコンフィギュレーション

- 3シリーズには全てのメモリオプション有

- コンフィギュラブル L1 キャッシュ
  - サイズ, キャッシュウェイ数,
    キャッシュラインのサイズ
- 密結合メモリ
  - 最大2MBまでカスタマイズ可能
- RISC-V PMP (Physical Memory Protection)
  - 最大16個



codasip

CODASIP RISC-V PROCESSORS
EVALUATION REQUEST FORM

CODASIP RISC-V PROCESSORS 1 AND 3 SERIES
CORE CONFIGURATION

You can learn more about Codasip RISC-V Processors on our website. Please, contact Codasip Support if you need further information.

For the purpose of the evaluation license, please, choose one of the following off-the-shelf configurations. These memory subsystem configurations are pre-built and ready to be delivered.

Choose requested configuration(s):

Choose	Processor core	Caches	Tightly coupled memory	Physical memory protection
☐	L11	✗	✗	✗
☐	L31	✗	✗	✗
☐	L31	✓	✗	✗
☐	L31	✗	✓	✗
☐	L31	✗	✗	✓
☐	L31F	✗	✗	✗
☐	L31F	✓	✓	✓

The memory subsystem of Codasip RISC-V Processors is highly configurable. However, for the purpose of evaluation, default settings are used in order to provide the configuration immediately. You can review the default settings in the table below.

Configurable item	Property	Default settings
Caches	Size	16 kB *
	Data cache included	Yes
	Number of cache ways	2 *
	Cache line size	32B *
	Bus interface	AHB Lite
Tightly coupled memory	Size	16kB *
Physical memory protection	Number of protected memory regions	8
JTAG	Number of pins	4
	Number of hardware breakpoints/ watchpoints	4
	System bus access	Disabled
Interrupt controller	Maximum number of interrupts	32
RTL	Language	Verilog

* Parametrizable in RTL

After the evaluation, customers may choose from a wide range of available core and memory settings for the purpose of the commercial license.

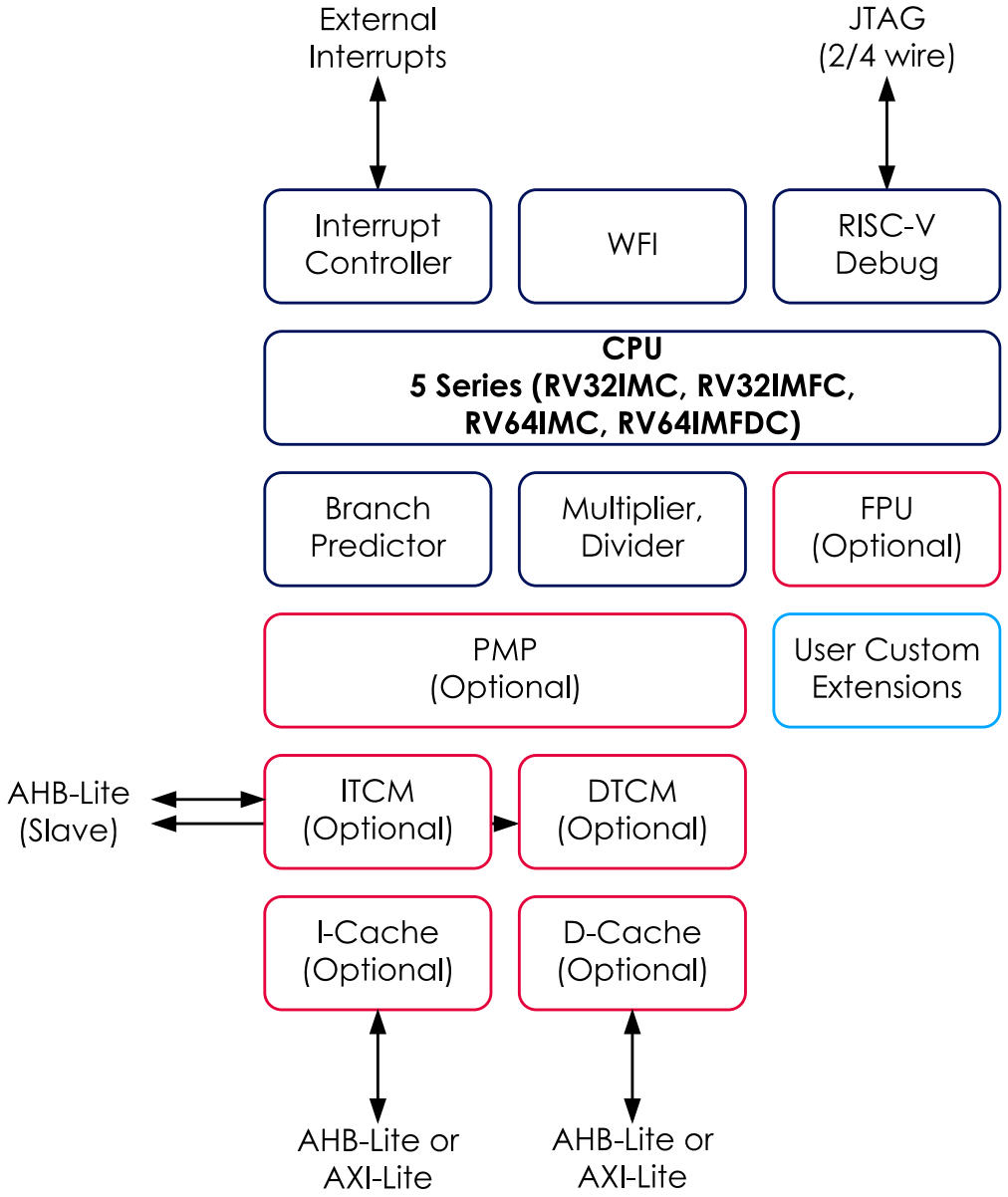
Example config, some items subject to change

5 SERIES

バランスの良い**5段パイプライン**

圧縮命令で最小のコードサイズを実現

動的な**分岐予測**

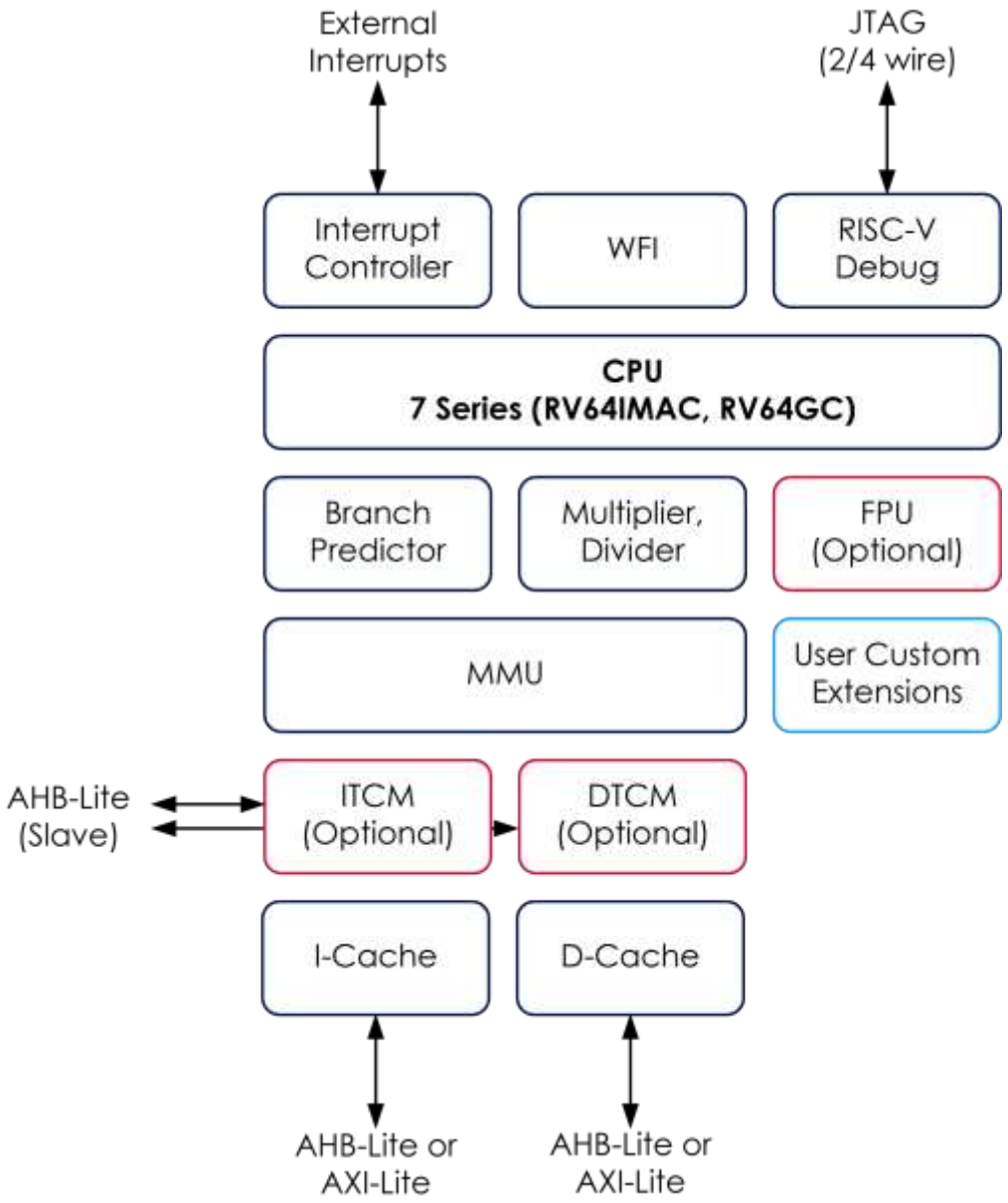


7 SERIES

複雑な**7段パイプライン**

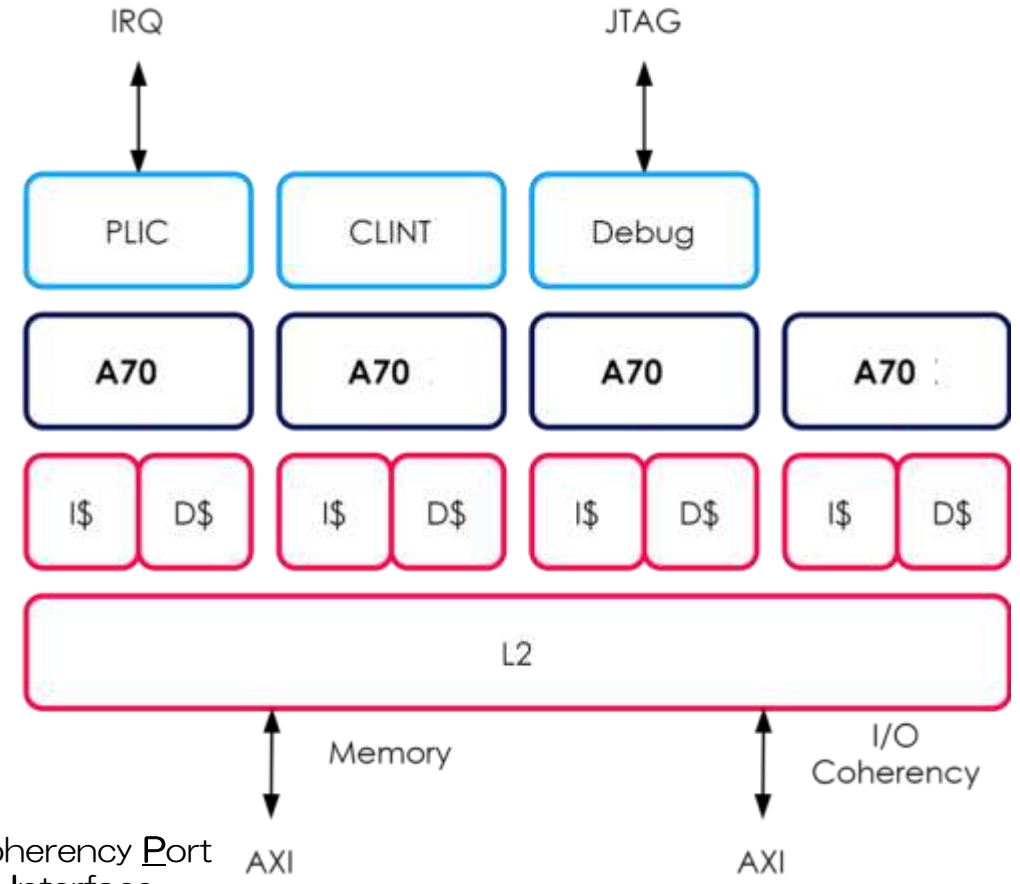
圧縮命令で最小のコードサイズを実現

動的な**分岐予測**



マルチコアによる圧倒的なパフォーマンス

- 最大4つのA70またはA70Pプロセッサをクラスタに搭載
- コヒーレントL1/L2キャッシュを搭載したスケーラブルなマイクロアーキテクチャ
  - キャッシュはコンフィギュラブル
  - ACPポート付きAXIインターフェース（I/Oコヒーレンシ）の提供を開始
  - CHIプロトコル（フルシステムコヒーレンシー）は近日中に利用可能
- RISC-V標準の割り込みコントローラ
  - CLINT (コア レベル)
  - PLIC (プラットフォーム レベル)
- A70-MP、A70P-MPがマルチコア対応



* ACP: Accelerator Coherency Port
* CHI: Coherence Hub Interface
* CLINT: Coreplex Local INTerrupts
* PLIC: Platform-Level Interrupt Controller

アプリケーションに最適なコアを選択

ALL CORES Standard RISC-V debug JTAG (4pin/2pin) Compressed instructions AMBA buses	LOW POWER EMBEDDED 32-bit Up to 128 interrupts	HIGH PERFORMANCE EMBEDDED 64-bit Up to 256 interrupts	APPLICATION 64-bit FPU Linux support
7 SERIES 7-stage pipeline IMC instruction set 32 registers Branch predictor Parallel multiplier			Codasip A70 Codasip A70-MP Codasip A70P Codasip A70P-MP
5 SERIES 5-stage pipeline IMC instruction set 32 registers Branch predictor Parallel multiplier	Codasip L50 Codasip L50F	Codasip H50 Codasip H50F	
3 SERIES 3-stage pipeline IMC instruction set 32 registers Parallel multiplier	Codasip L31 Codasip L31F Codasip L30* Codasip L30F*		
1 SERIES 3-stage pipeline EMC instruction set 16 registers Sequential multiplier	Codasip L11 Codasip L10*		

* Not recommended for new designs
F = Floating Point Unit, P = RISC-V P Packed SIMD Extension, MP = Multiprocessing

+

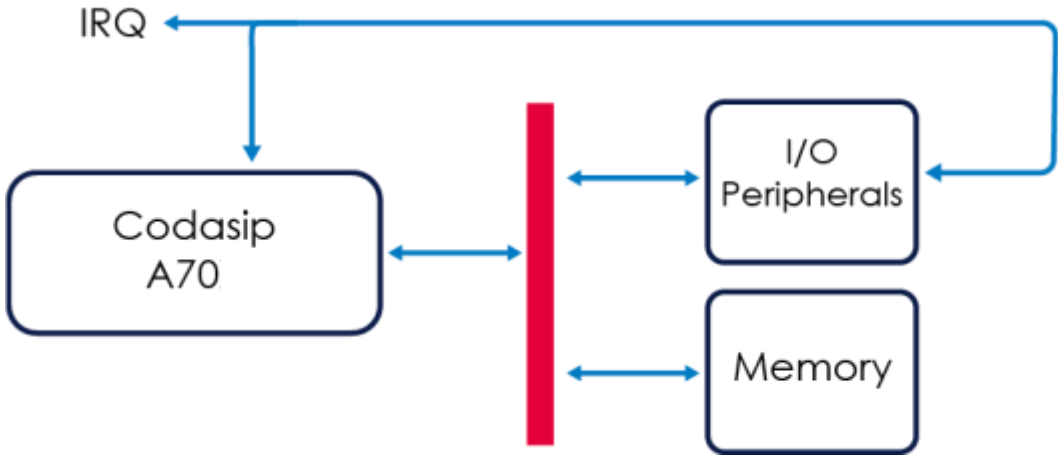
キャッシュ
命令とデータ
サイズのカスタマイズ
キャッシュウェイ数
キャッシュラインのサイズ

密結合メモリ
TCM: Tightly Coupled Memories

PMP: Physical Memory Protection

Codasip FPGA評価プラットフォーム

- RTOSまたはLinuxが動作するプロセッササブシステム
  - Codasip RISC-Vプロセッサを簡単に評価可能
  - ソフトウェアのRISC-Vへの移植に有用
- FPGAプラットフォームの推奨設定
  - Digilent Nexys-A7-100T FPGAボード
  - Digilent JTAG-HS2プログラミングケーブル
    - SeggerやIARのケーブルを使用することも可能
- 15分以内に起動可能
- 推奨FPGAボードのステップバイステップなクイック・スタート・ガイドを提供



codasip

RISC-V PROCESSORS CUSTOMIZATION

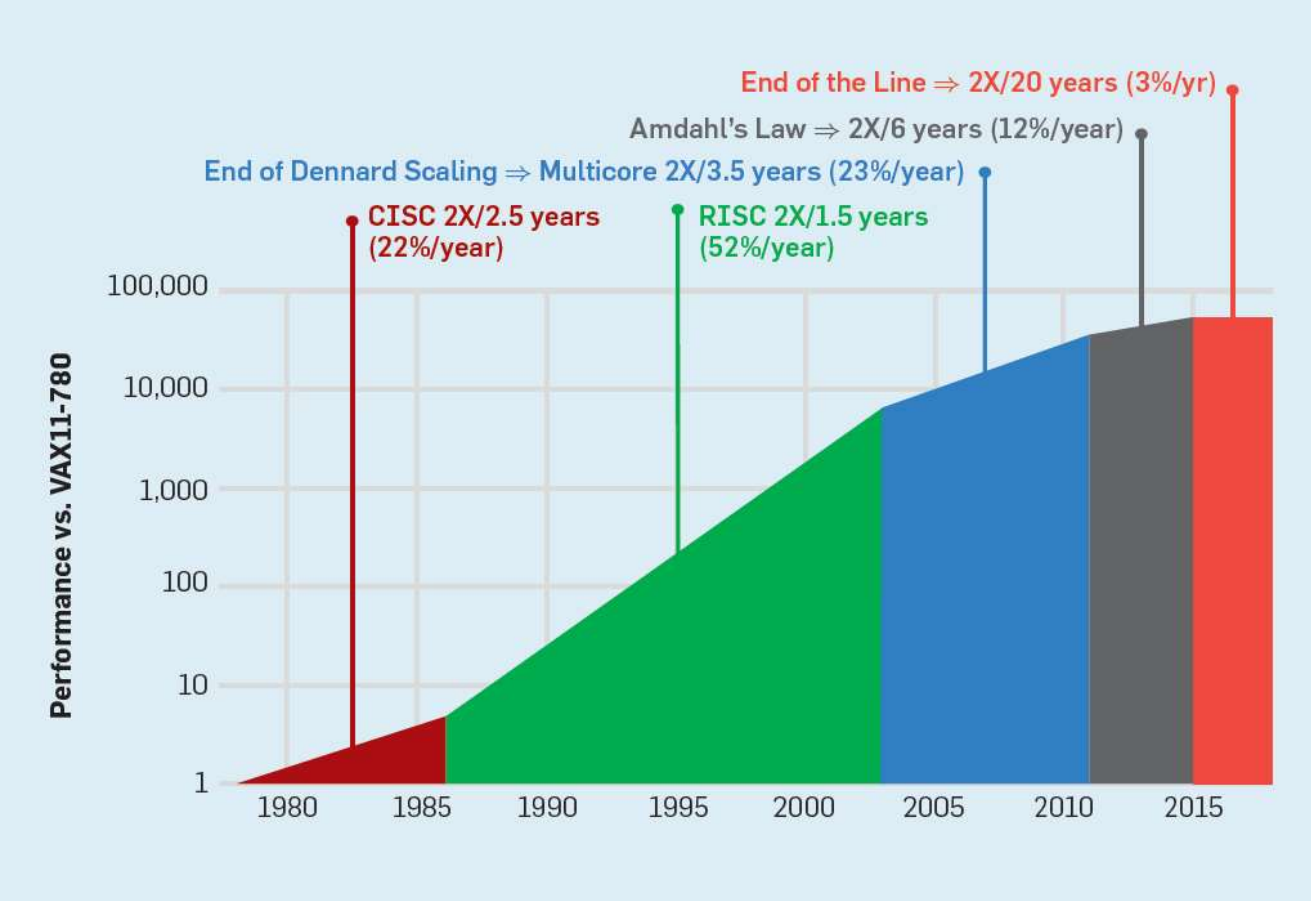
競合に差をつけたくありませんか？

#DesignForDifferentiation

お探しのプロセッサが見つかりませんか？

- オープン標準命令セットアーキテクチャRISC-Vは
  カスタマイズを前提に設計されています
  - RISC-Vエコシステムの利点を維持したまま、製品を差別化する
  - ISAを最適化し、より良いPPAを実現する
    - 1タスクあたりの命令数の削減 → 性能アップ
    - 総サイクル数の削減 → 低消費電力化（低周波数化）
    - コードおよびデータサイズの削減 → 小面積化（ロジックおよびROM）
- Codasip RISC-Vプロセッサをカスタマイズのベースラインとして使用できます
- RISC-Vプロセッサを自分でカスタマイズできる!

汎用プロセッサ性能の飽和



出典: Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson,
"A Domain-Specific Architecture for Deep Neural Networks"
https://cacm.acm.org/magazines/2018/9/230571-a-domain-specific-architecture-for-deep-neural-networks/fulltext

成長の余地: ムーアの法則が通用しなくなったときにコスト・パフォーマンスを大きく向上させる唯一の道は、**特定のドメイン向け命令を追加**することである。例えばディープ・ラーニング、拡張現実、組み合せ最適化、グラフィックスなどのドメインが考えられる。

出典: RISC-V原典（日本語版）P9

*In our current research, we are **especially interested** in the **move towards specialized and heterogeneous accelerators**, driven by the power constraints imposed by the **end of conventional transistor scaling**. We wanted a **highly flexible and extensible base ISA** around which to build our research effort.*

出典: RISC-V Spec Volume I: Unprivileged ISA 20191213, Section 28.1
https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

31 26	25 15	14 12	11 7	6 0	
funct6	*custom*	funct3	*custom*	opcode	Recommended Purpose
6	11	3	5	7	
100011	*custom*	0	*custom*	SYSTEM	Unprivileged or User-Level
110011	*custom*	0	*custom*	SYSTEM	Unprivileged or User-Level
100111	*custom*	0	*custom*	SYSTEM	Supervisor-Level
110111	*custom*	0	*custom*	SYSTEM	Supervisor-Level
101011	*custom*	0	*custom*	SYSTEM	Hypervisor-Level
111011	*custom*	0	*custom*	SYSTEM	Hypervisor-Level
101111	*custom*	0	*custom*	SYSTEM	Machine-Level
111111	*custom*	0	*custom*	SYSTEM	Machine-Level

Figure 3.30: SYSTEM instruction encodings designated for custom use.

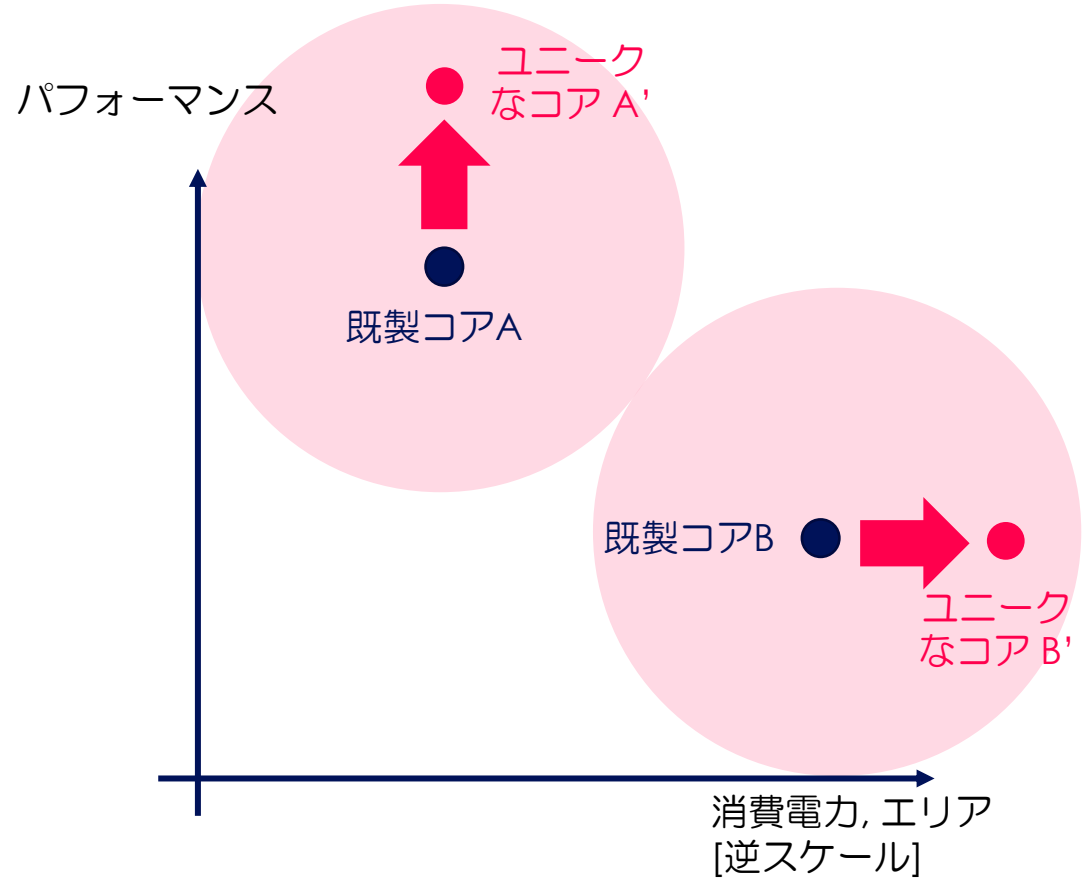
出典: RISC-V Spec Vol II: Privileged Architecture 20211203, Section 3.3.4
https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

- RV32I – 最も基本的なRISC-Vの実装
- RV32IMAC
  – 整数 + 乗算 + アトミック命令 + 圧縮命令
- RV32IMAC_X[ext]
  – MAC + 非標準ユーザー拡張

考慮されている「非標準ユーザー拡張」

既製コアIPは汎用向けであり、貴方に最適ではない

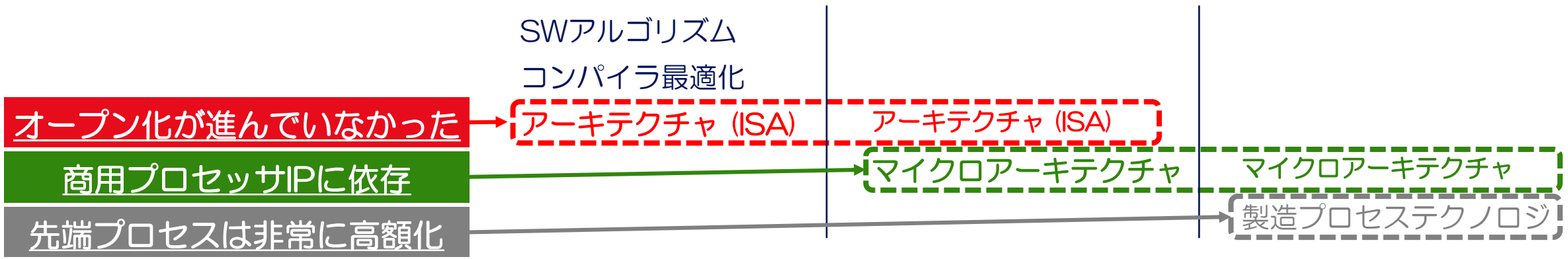
- **既製コアのPPAは1つ**
  - 典型的ボリューム要件に寄せた設計
  - 一方お客様のアプリケーションは千差万別
  - 既製コアを選択するということは、妥協することを意味します
- **あなたの要件に適したPPAを得る**
  - Codasip Studioによるカスタマイズで、お客様のアプリケーションに最適なPPAを実現します



プロセッサパフォーマンスの鉄則 (Iron law of processor performance)

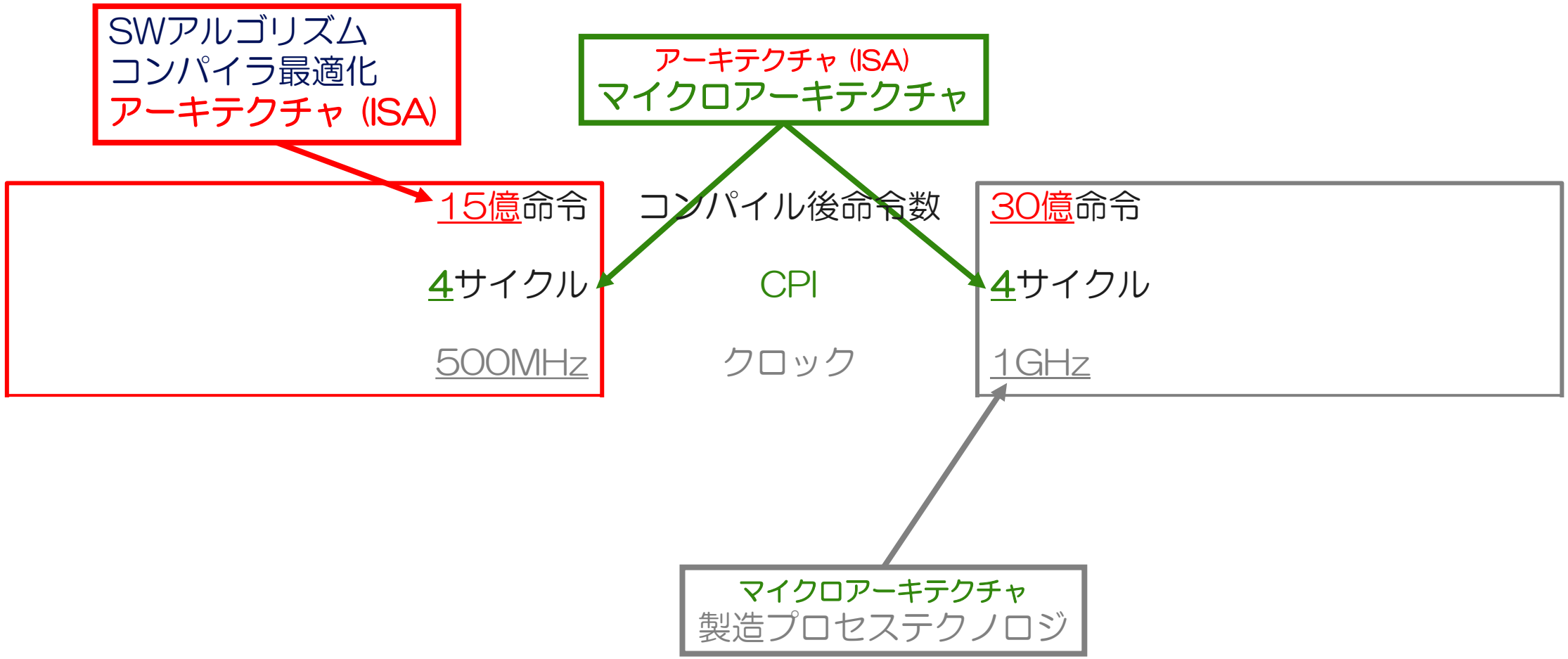
$$\text{パフォーマンス} = \frac{\text{時間}}{\text{プログラム}} = \frac{\text{命令数}}{\text{プログラム}} \times \frac{\text{ClockCycles}}{\text{命令数}} \times \frac{\text{時間}}{\text{ClockCycles}}$$

(プログラム->命令数) (CPI) (Cycle Time)



出典: A Characterization of Processor Performance in the VAX-11/780, Joel S. Emer, Douglas W. Clark, 1984, IEEE

Iron law を用いた考察



何がカスタマイズ可能か？

- ISAアーキテクチャレベルでは:
  - スカラ命令
  - SIMD/ベクター命令
  - 単精度/倍精度 浮動小数点演算 命令
  - 複数の入出力オペランドを持つ命令
  - DSP 命令
    - ゼロ・オーバーヘッド・ループ
    - ロード/ストア x プリ/ポストインクリメント
  - 複数レジスタファイル
- マイクロアーキテクチャレベルでは:
  - パイプライン段数の定義
    - ステージ難読化
  - メモリサブシステムへの接続定義
    - プロトコル
    - Bit幅
  - データ/制御/構造 ハザードのハンドリング
  - リソースシェアリング
  - MMU
  - 物理メモリプロテクション

その他いろいろ...

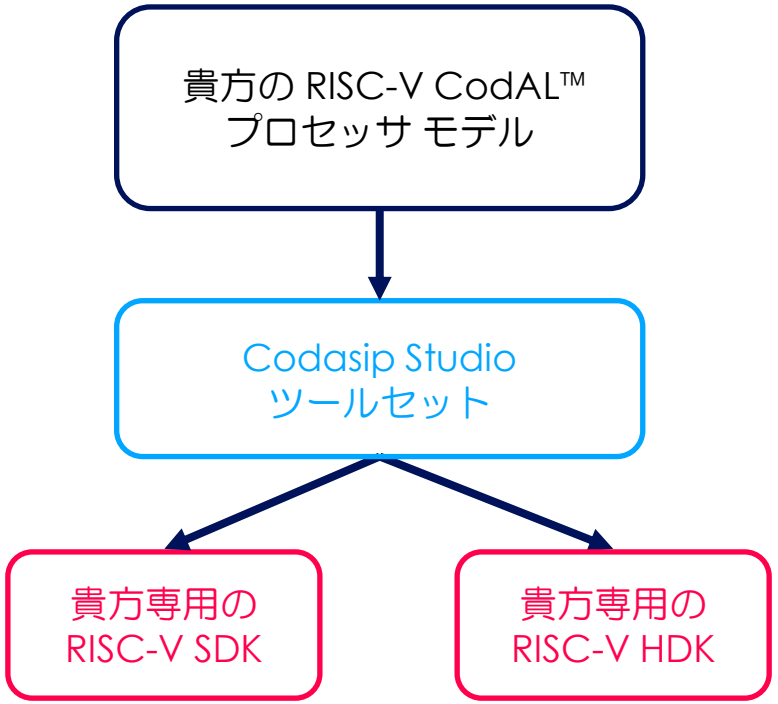
codasip STUDIO

私たちの技術基盤

Codasip Studioとは？

RISC-Vプロセッサを高速かつ容易に改造するためのユニークなツール群
オールインワン、高度な自動化。2014年発売、大手ベンダーでシリコン実証済

- プロセッサを高位アーキテクチャ記述言語で記述
- 下記のカスタマイズを可能に:
  - 命令セット アーキテクチャ (ISA: Instruction set architecture)
  - マイクロ アーキテクチャ
- いずれも可能です:
  - RISC-V標準拡張のコンフィグレーション
  - 既製Codasip RISC-Vプロセッサの**カスタマイズ**
  - 新しいプロセッサを**ゼロから設計**



カスタマイズのハードルを下げる!

カスタマイズしたプロセッサには、カスタマイズに追従するSDKが必要...

普通のカスタマイズ
（カスタム命令を手動で追加）：

1. 新しい命令をモデル化し、シミュレーション
2. コンパイラの修正
3. アセンブラを修正
4. デバッガでの対応追加
5. 検証, 検証, 検証…

>>挑戦的、時間がかかる、費用がかかる…

カスタマイズ自動生成ツールの利点:

- カスタムプロセッサの開発コスト削減
- ツールの改造に要する時間の短縮
- 追加命令をコンパイラが自動推定
  - インライン アセンブラによるコーディング不要
- 実績あるオープンソースのテクノロジと
  フレームワークにより、RISC-Vエコシステム
  に容易に統合可能

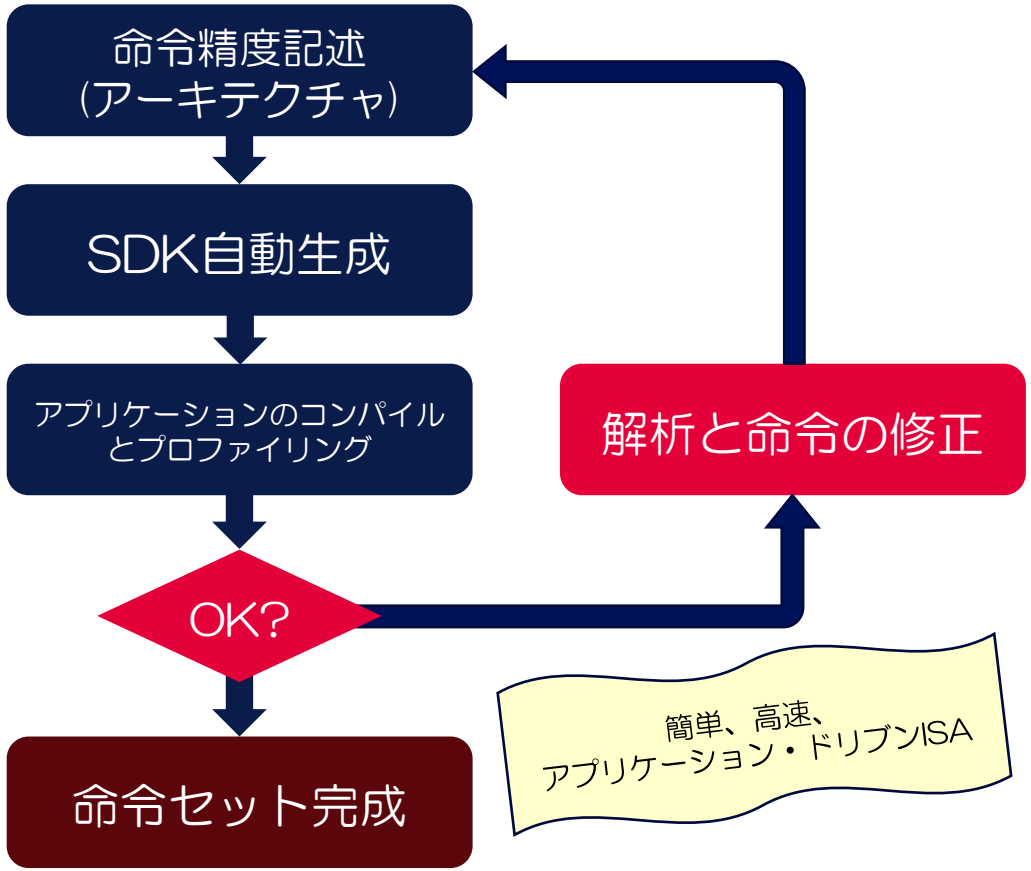
コダシップは、カスタマイズを自動化するための独自のツールセットを提供: Codasip Studio.

SDKとHDKの自動生成

- Software Development Kit (SDK)
  - C/C++ LLVM コンパイラ
    (コダシップ改良版)
  - C/C++ ライブラリ (newlib)
  - アセンブラ、逆アセンブラ、リンカー
  - 命令精度(IA)シミュレータ
  - サイクル精度(CA)シミュレータ
  - デバッガとプロファイラー
  - ドキュメントとISAの可視化
  - 検証時に使用するランダムプログラム
- Hardware Development Kit (HDK)
  - RTL (Verilog/VHDL/SystemVerilog)
    - 高い可読性
    - CodALソースへのリンク
  - SystemVerilog UVM検証環境
  - 統合テストベンチ
  - 標準EDAツールのサンプルスクリプト
  - SystemCコシミュレーション モデル

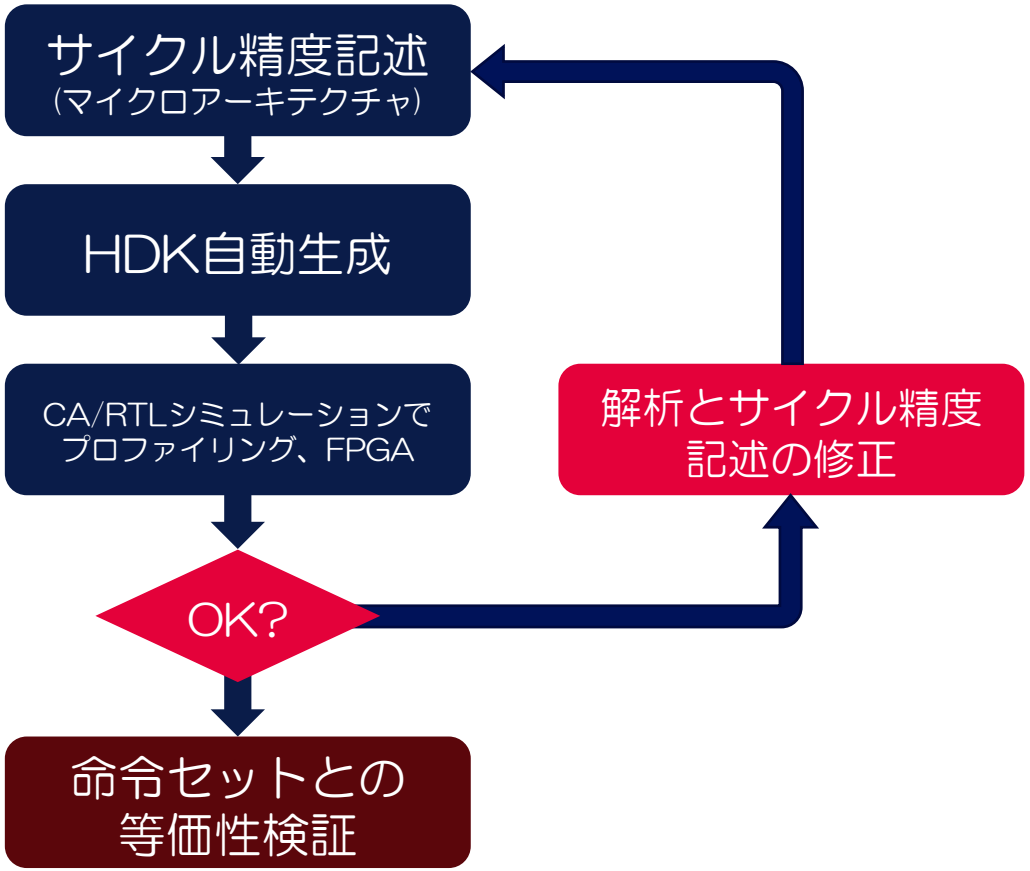

命令セット(ISA)探求に!!!

SDK in the loop



ハードウェア開発に!!!

HDK in the loop



Codasip Architectural Language

CodALは、HDLのような汎用的設計に使われる言語とは異なります

- プロセッサ開発に特化した言語
- 1つのCodALモデルで
  複数のマイクロアーキテクチャの
  実装が可能

理解し易いC言語ライク言語、高抽象度でプロセッサのアーキテクチャ、構造を表現し、自動生成可能にする


```
/*    Multiply and accumulate: semantics
                        dst += src1 * src2
*/

element i_mac {
            use reg as dst, src1, src2;
            assembly { “mac” dst “,” src1 “,” src2 };
            binary { OP_MAC dst src1 src2 0:bit[9] };
            semantics {
                        rf[dst] += rf[src1] * rf[src2];
            };
};
```

可読性の高いHDLを生成

- CodAL記述から生成されるRTLは読みやすい
  - 前ページの手書きのコードと、右の自動生成コードを比べてみてください
- 生成されるRTLには、デバッグに有用な情報が含まれます
  - 生成されるRTLには、ソースであるCodALコードへのリンク

```
module rf_gpr(
    input  wire CLK,
    input  wire RST,
    input  wire r0_RE,
    input  wire [4:0] r0_RA,
    input  wire r1_RE,
    input  wire [4:0] r1_RA,
    input  wire w0_WE,
    input  wire [4:0] w0_WA,
    input  wire [31:0] w0_D,
    output wire [31:0] r0_Q,
    output wire [31:0] r1_Q
);
    localparam integer SIZE = 32'h00000020;
    localparam [31:0] DEFAULT_VALUE = 32'h00000000;
    // memory storage
    reg [31:0] RAM[0:SIZE-1];

    generate
        genvar ii;
        for ( ii = 32'sd0; ii < SIZE; ii = ii + 32'sd1 ) begin : WRITE_PROC
            always @( posedge CLK or negedge RST ) begin
                if ( RST == 1'b0 ) begin
                    RAM[ii] <= DEFAULT_VALUE;
                end else if ( (w0_WE == 1'b1) && (w0_WA == ii) ) begin
                    RAM[ii] <= w0_D;
                end
            end
        end
    endgenerate

    assign r0_Q = (r0_RE == 1'b1) ? RAM[r0_RA] : 32'h00000000;
    assign r1_Q = (r1_RE == 1'b1) ? RAM[r1_RA] : 32'h00000000;
endmodule
```

まとめ

codasip
RISC-V PROCESSORS

商用品質のRISC-V IP
を提供するリーディングカンパニー

- 包括的なオフザシェルフ（既製品）ポートフォリオ
- 完全な事前検証済みIP
- 専門性の高い専任のカスタマーサポートスタッフが対応

RISC-Vのカスタマイズ
を簡単に自動化

- カスタマイズで、圧倒的な真の差別化を実現
- コダシップは、カスタマイズのための完全なツールやリソースのセットを提供

codasip
STUDIO

カスタムアーキテクチャ
を設計するためのユニークな手法

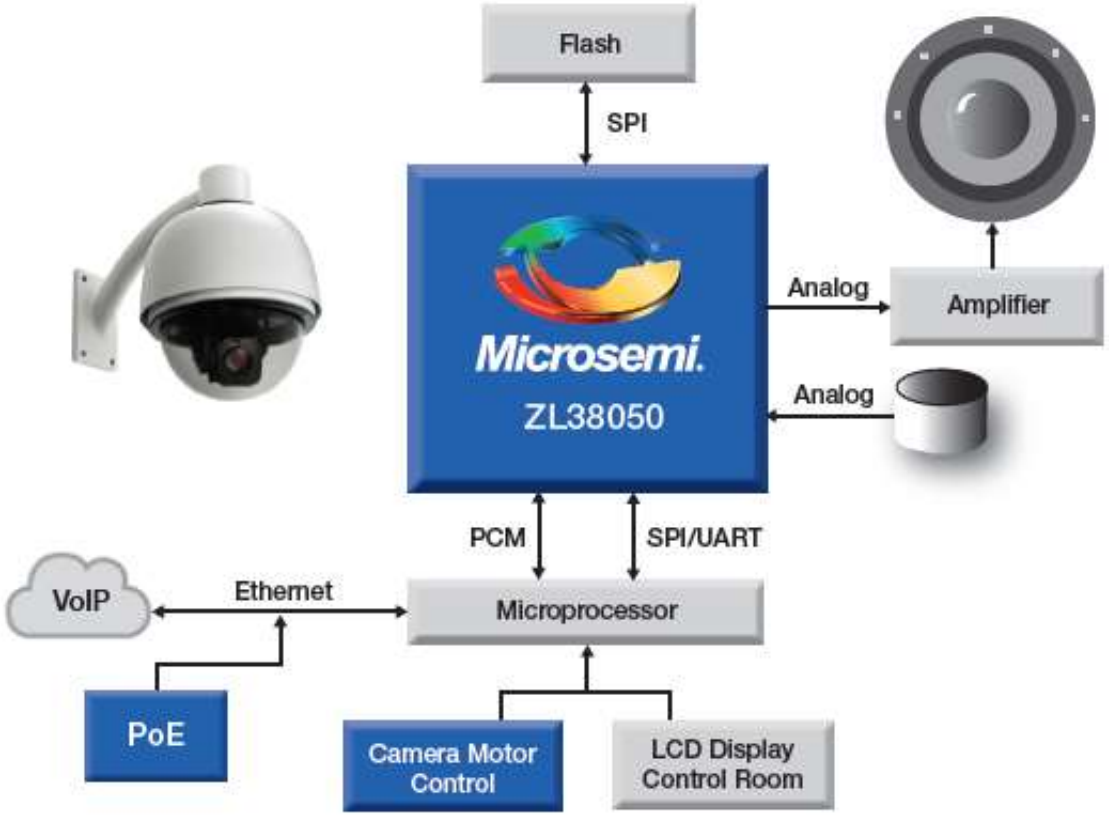
- Codasip Studioを用いた高度な自動化プロセス
- お客様や社内のプロセッサ開発で幾度も使用されている高水準の設計フロー

ケーススタディ

サクセスストーリー

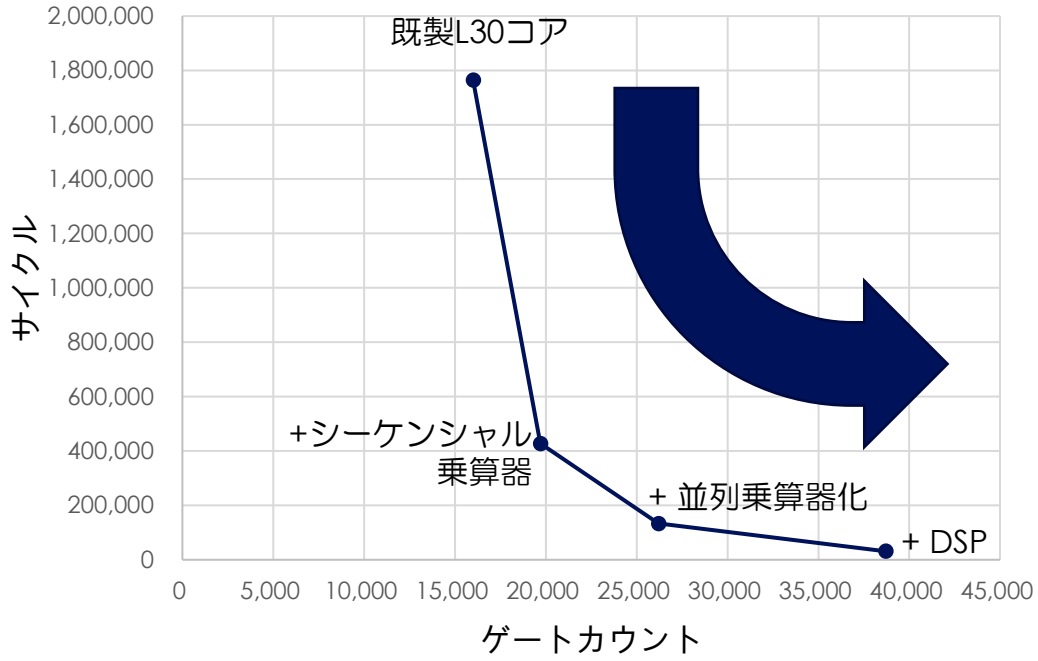
ケーススタディ：Microsemi社（現Microchip社）

- オーディオ・イコライザー・アルゴリズム
- **要求事項**
  - 低価格化
    - 最適ROIの製造プロセスノード選択
    - IPライセンス料、ロイヤリティ等
  - 低消費電力（低動作周波数化等）
  - 多様な要件に対応する派生デザイン開発容易化
  - Time-to-market
    - デザインサイクル9-12ヶ月
  - Cortex-Mの置換え
- **Codasip Studioでのデザイン探求**
  - RV32Iから開発スタート
  - M拡張命令追加、 後にカスタムDSP命令追加



カスタム命令を追加するメリット

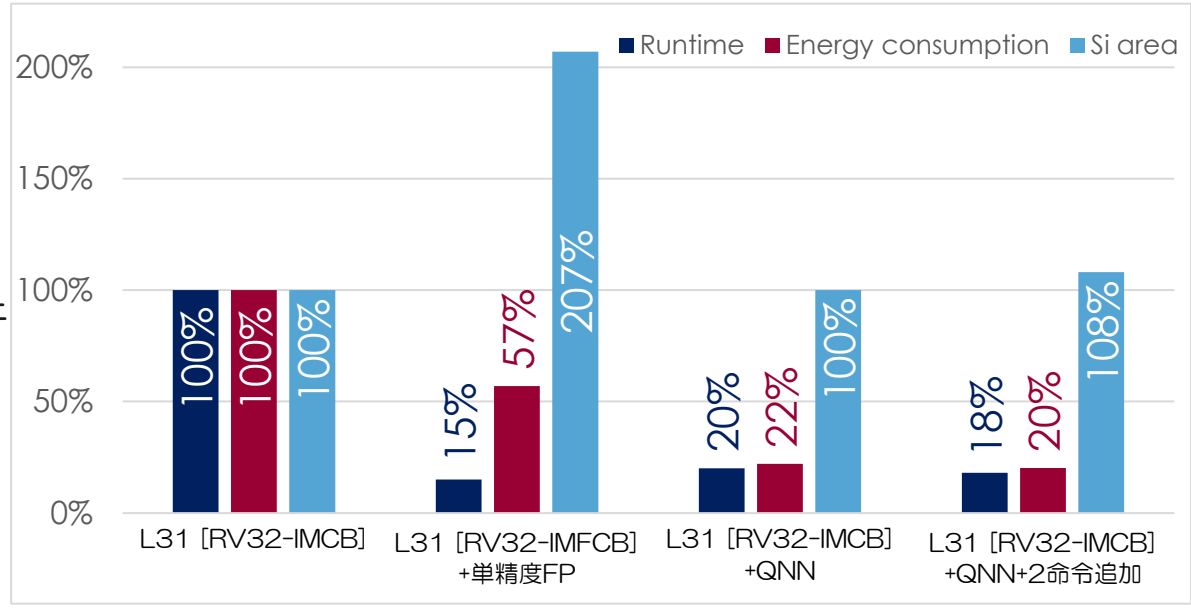
- スループット **56.24倍 向上**
- コードサイズ **3.62倍 圧縮**
- ゲートカウント **2.43倍 増加**
  - プロセッサコアではなく、命令メモリが領域で支配的であることに注意
  - いずれもベースのL30コアとの比較
- 古い製造プロセスノードでシリコン化することで大幅なコスト削減を実現

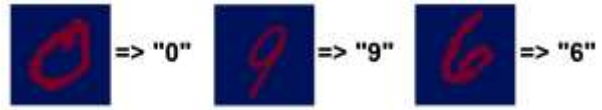


シンプルに、速く、正確に、デザインスペースの探求
… 数日で完了

ケーススタディ: AI/ML

- TensorFlow Lite for MicrocontrollersをL31コアで実行
- IoT/エッジ・アプリケーションに最適化
  - 低消費電力化 / 限定的なメモリ量
- AIによる手書き文字認識 (MNIST)
  - ランタイムの削減 **80%**
  - 低消費電力化 **78%**
  - エリア増加**インパクト無**





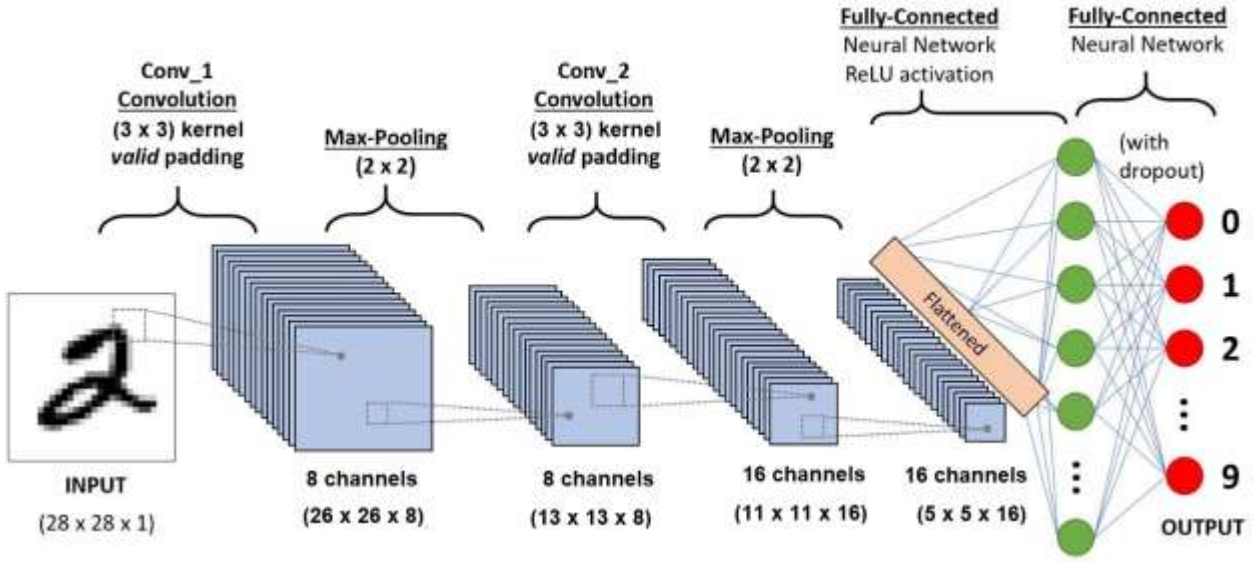
MNIST "Handwritten Digits recognition" benchmark intelligent label assignment to grayscale 28x28 image.

Symbol	Address	Instructions	Instructions Percent	Cycles	Cycles Percent
tflite::reference_integer_ops::ConvPerChannel	36fa6	6572379	86.3 %	9340321	83.9 %
tflite::reference_integer_ops::MaxPool	45e60	412255	5.4 %	710898	6.4 %
tflite::reference_integer_ops::FullyConnected	3e388	158370	2.1 %	236154	2.1 %

ホワイトペーパー（日本語有）
https://codasip.com/2022/02/24/embedded-ai-on-l-series-cores/

Codasip Studioが生成するシミュレータを用いて、
どのアルゴリズムが重要で、どこを最適化すべきかを簡単に確認できます

Profiling TFLite Image classification model



Source Code Coverage

Symbol	Address	Instructions	Instructions Percent	Cycles	Cycles Percent
tflite::reference_integer_ops::ConvPerChannel( )	2940c	8878981	89.9 %	15388368	90 %
tflite::reference_integer_ops::MaxPool( )	2b7e8	412616	4.2 %	710990	4.2 %
tflite::reference_integer_ops::FullyConnected( )	2a2de	157058	1.6 %	207101	1.2 %
...etc	26a60	95618	1 %	143346	0.8 %

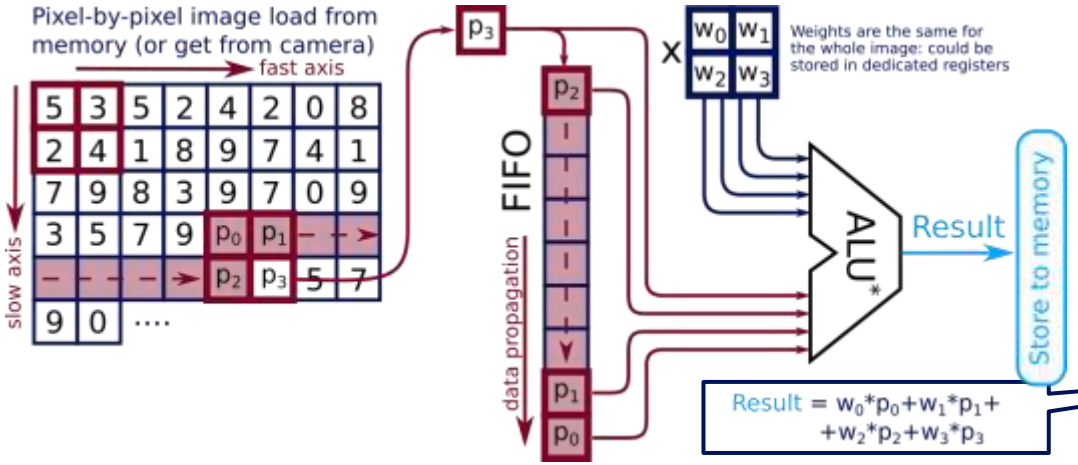
```
   430   0.004%   for (int out_y = 0; out_y < output_height; ++out_y) {
                    const int in_y_origin = (out_y * stride_height) - pad_height;
  9092   0.092%    for (int out_x = 0; out_x < output_width; ++out_x) {
                      const int in_x_origin = (out_x * stride_width) - pad_width;
 46142   0.467%     for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
 52460   0.531%       auto group = out_channel / filters_per_group;
 19562   0.198%       int32_t acc = 0;
596782   6.044%       for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
                        const int in_y = in_y_origin + dilation_height_factor * filter_y;
390154   3.951%       for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
132192   1.339%         const int in_x = in_x_origin + dilation_width_factor * filter_x;
                        // Zero padding by omitting the areas outside the image.
                        const bool is_point_inside_image =
                            (in_x >= 0) && (in_x < input_width) && (in_y >= 0) &&
                            (in_y < input_height);
435865   4.414%       if (!is_point_inside_image) {
                          continue;
                        }
491681   4.980%       for (int in_channel = 0; in_channel < filter_input_depth;
188064   1.905%            ++in_channel) {
                          int32_t input_val =
                              input_data[Offset(input_shape, batch, in_y, in_x,
                                                in_channel + group * filter_input_depth)];
376128   3.809%         int32_t filter_val = filter_data[Offset(
                              filter_shape, out_channel, filter_y, filter_x, in_channel)];
790788   8.009%         acc += filter_val * (input_val + input_offset);
                      }
                    }
                  }
```

- Image convolution (>89%) has a major impact on overall performance

CONV accelerator in <200 lines of CodAL code

element comprises convolution instruction assembly, binary and semantics in dedicated sections

push_conv instruction pushes image pixel from **src** to ***fifo[]***, calculates the convolution result and stores it to the **dst** address



fifo[] is a hidden register file,
load and **store** addresses are incremented automatically, thanks to **fifo_counter** and **out_counter** that contain the number of input image pixels loaded and output image pixels processed

Single instruction gets the convolution result (2x2 window)

```
element inst_push_conv
{
    use gpr_all as dst, src;                    // src - image pixel data, dst - address to store the result

    assembly { "push_conv" dst "," src };       // Assembly format
    binary { OPC_PUSH_CONV UNUSED:bit[5] dst src UNUSED:bit[9] };   // Instruction binary pattern
    semantics
    {
        uint32 p0,p1,p2,p3, result;

        p3 = rf_gpr_read(src);

        if ((fifo_counter>=FIFO_DEPTH) && (fifo_counter%(FIFO_DEPTH - 1) != 0))   // Image edge condition handling
        {
            uint32 addr_dst;
            p0 = fifo[fifo_counter % FIFO_DEPTH];
            p1 = fifo[(fifo_counter - FIFO_DEPTH + 1)%FIFO_DEPTH];
            p2 = fifo[(fifo_counter - 1)%FIFO_DEPTH];

            result = p3 * weights[3] + p2 * weights[2]                  // ALU*
                   + p1 * weights[1] + p0 * weights[0];

            addr_dst = rf_gpr_read(dst) + ARRAY_STEP * out_counter;     // Storage and destination address increment
            store(OPC_ST, addr_dst, result);
            out_counter = (out_counter < CONV_RES_LEN - 1) ? out_counter + 1 : 0;
        }

        fifo[w_index_0] = val;                  // Push current pixel data to FIFO, pointer increment
        fifo_counter = (fifo_counter < IMAGE_LEN - 1) ? fifo_counter + 1 : 0;
    };
};
```

ケーススタディ：SecureRF社（現Veridify社）

SECURE RF
Securing the Internet of Things®

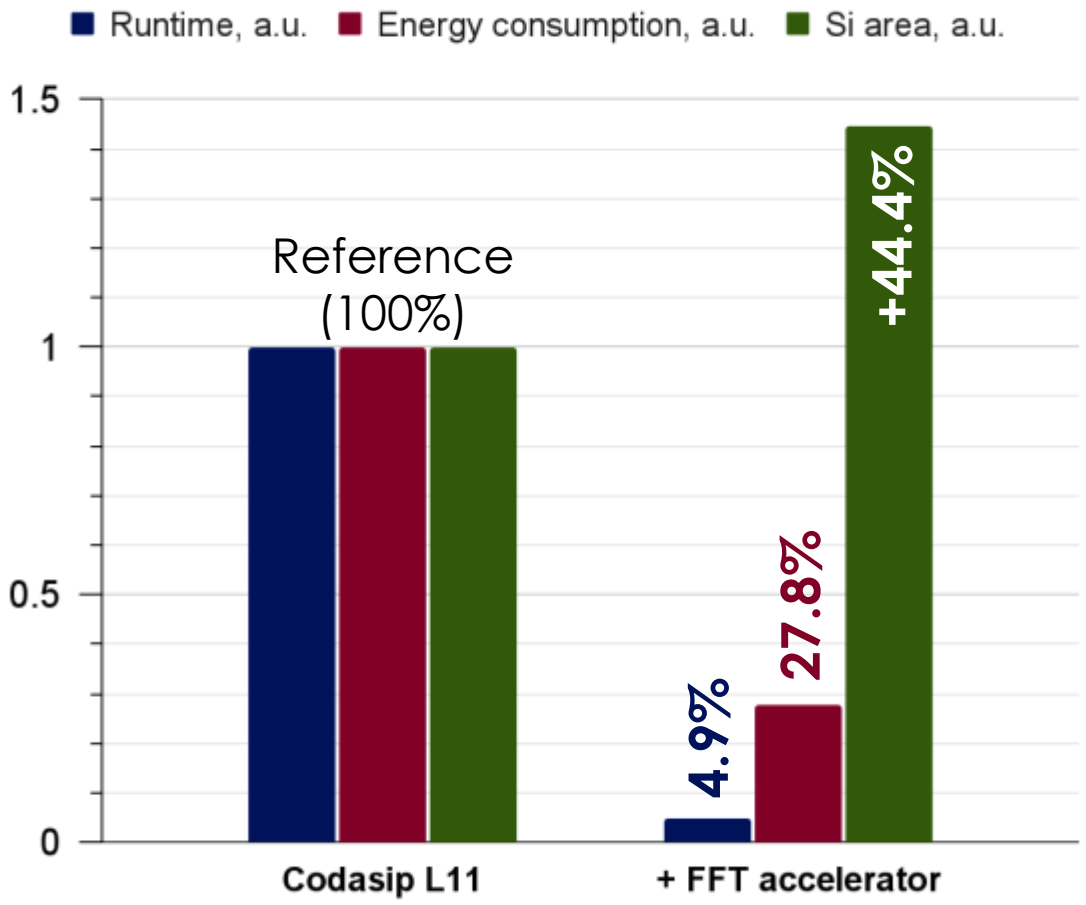
	HWアクセラレータ無	HWアクセラレータ有	改善結果
検証時間	10.7 ms	3.8 ms	2.8x 高速化
署名の検証時間/秒（50MHz）	93	263	
コードサイズ	4,504 bytes	3,052 bytes	32% 小型化

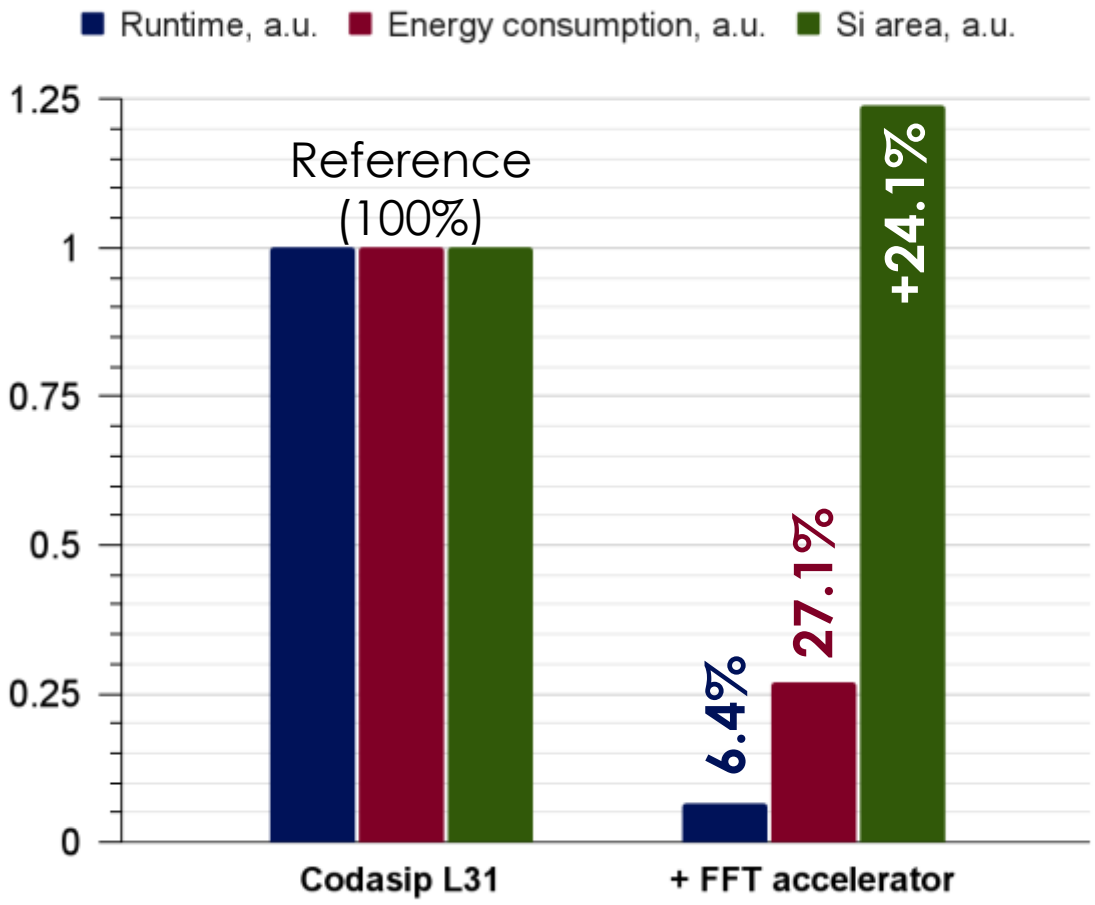
命題は、Codasip RISC-Vプロセッサで実行されている
WalnutDSA（デジタル署名アルゴリズム）の高速化

結果は、わずか2%の追加HWリソースで、実行時間は3倍改善

わずかな努力で、大きな改善が可能

ケーススタディ：FFTアクセラレータ

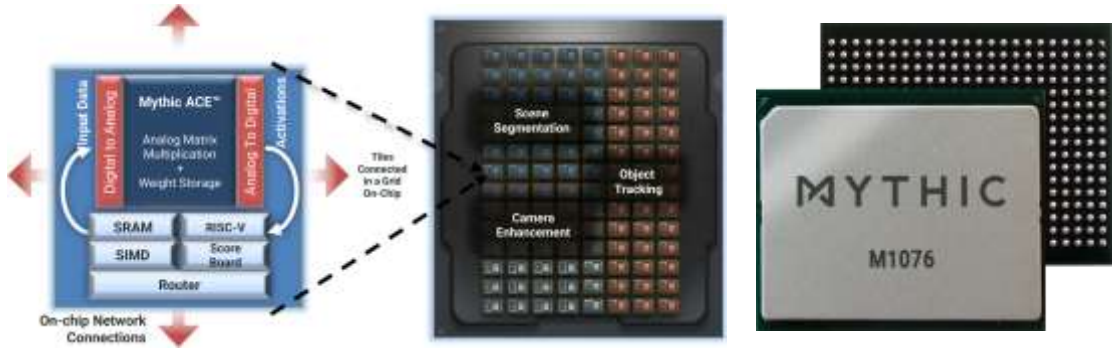




ケーススタディ: Mythic社

MYTHIC

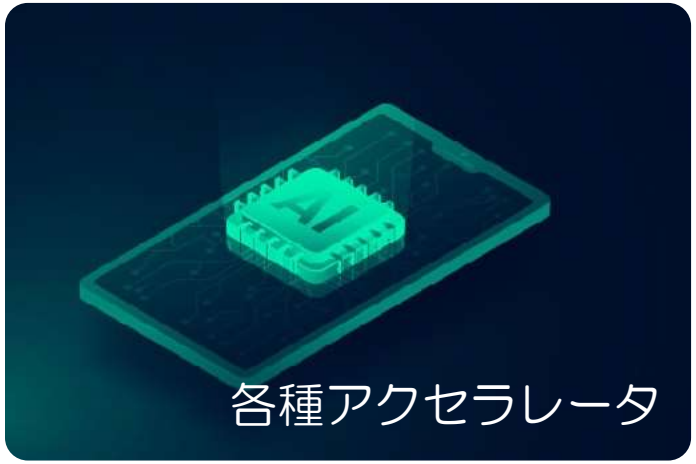
- Mythic社はAnalog Matrix Processor (AMP) M1076に独自のコアを必要とした
- いずれの既存コアも要求を満たせず: Mythic社はCodasip L30をベースにカスタマイズを実施
  - DSP、ビット操作、ゼロオーバーヘッドループ、
    コプロセッサ制御命令、カスタムコプロセッサインタフェース
- 従来設計より大幅な小型化・高性能化を実現
- 従来比、半分以下の時間で設計

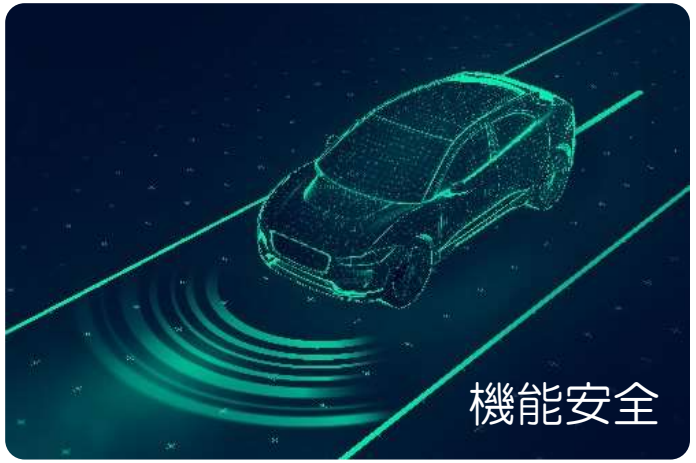


"コダシップのソリューションは、我々のニーズに特化した、真にユニークなRISC-Vプロセッサを柔軟に開発することを可能としました。これにより、自社でプロセッサを一から作る手間が省け、製品開発における他の重要な分野に集中することができました。"

Ty Garibay, VP of Hardware Engineering at Mythic

Codasip Lab – 注力イノベーション分野





Thank you.

Now, it’s your turn!

www.codasip.com

お問い合わせは、takaaki.akashi@codasip.com まで