2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Differentiating with Custom Compute and Use Case Intro

Shigehiko Ito, Field Application Engineer
Codasip

Codasip

accellera SYSTEMS INITIATIVE

今 カスタム・コンピュートが最善の道です

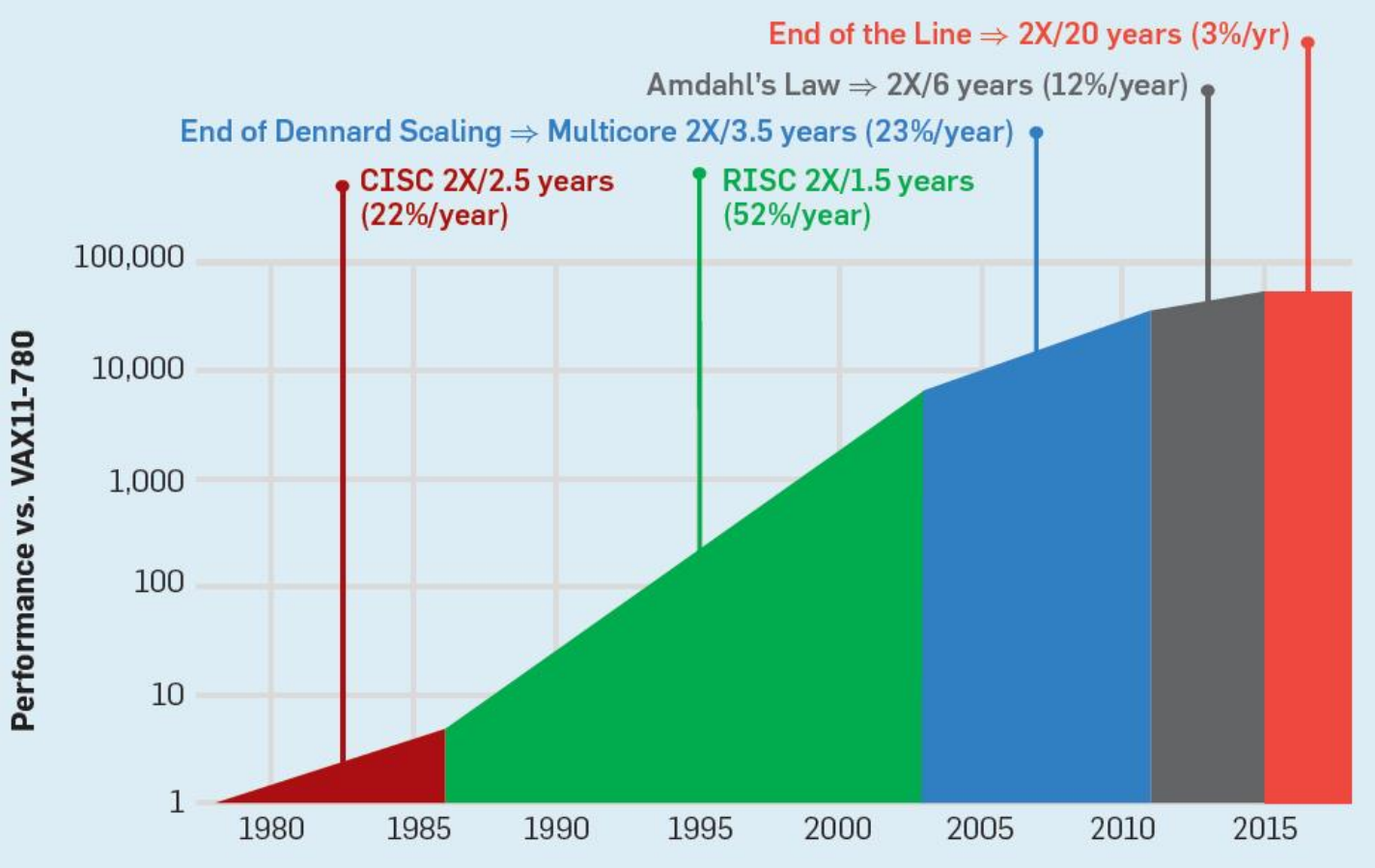
- シリコンプロセス技術の進歩（ムーアの法則）は減速しています
- 差別化が重要

カスタムコンピュートは

10x ... 100x

HW/SW協調最適化による性能向上をもたらす

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

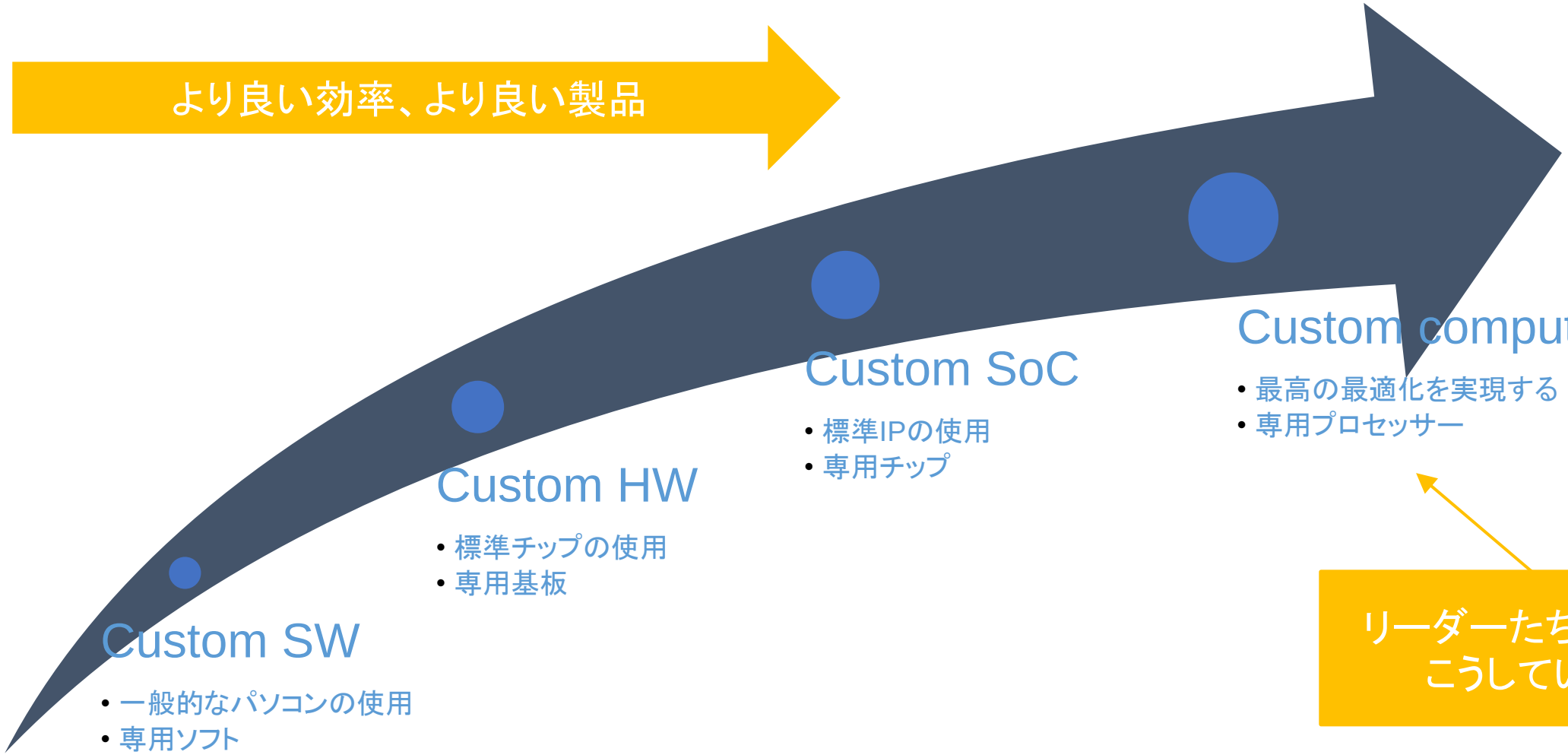
Figure 3.30: SYSTEM instruction encodings designated for custom use.

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」

さまざまなレベルの製品カスタマイズ



カスタム・コンピュートのメリット

性能の向上

- 古いプロセスの活用
- 限界への挑戦

効率性の向上

- 電力の削減
- 面積の縮小
  - 例えば、コンパニオンチップの削減、コードの圧縮など

差別化

- 競合より抜きんでる
- コピーから守る
  - 独自のプロセッサは真似することが困難

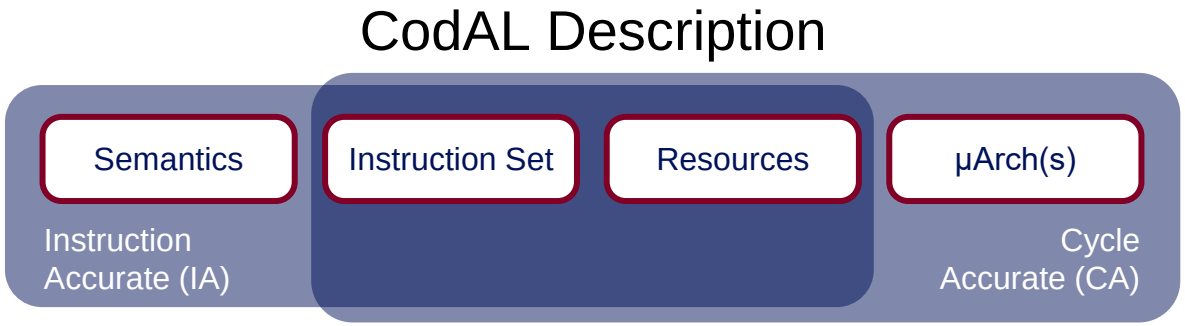
どうやってカスタム・コンピュートを実装する？

CodAL language

高い抽象度でのプロセッサの記述

CodAL - processor description at the high level

- CodALはC/C++のような言語で、プロセッサの豊富な機能をモデリングすることに重点を置いています。
- アーキテクチャとマイクロアーキテクチャの両方をカバー。
- **命令は、構文、バイナリ符号化、実装を表す「エレメント」という形で記述される。**
- CodALの記述をRTLに変換したり、サイクル精度の高いシミュレータとプロファイラを備えたC/C++コンパイラの生成に使用することができます。




```
/*    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];
          };
};
```

CodALによるコード縮小化


```
module rf_gpr #(parameter xlen = 64, parameter size = 32,
  parameter resetval = 32'b0, localparam aw = $clog2(size))
( input wire clk, input wire rst, input wire w0_we,
  input wire [aw-1:0] w0_wa, input wire [xlen-1:0] w0_d,
  input wire r0_re, input wire [aw-1:0] r0_ra,
  output wire [xlen-1:0] 0_q, input wire r1_re,
  input wire [aw-1:0] r1_ra, output wire [xlen-1:0] r1_q );

reg [xlen-1:0] mem[size-1:0];
integer i;

always @(posedge clk or negedge rst)
if (~rst) begin
  for (i = 0; i < size; i = i + 1)
    mem[i] <= resetval;
end else if (w0_we) begin
  mem[w0_wa] <= w0_d;
end

assign r0_q = r0_re ? mem[r0_ra] : (xlen)'(0);
assign r1_q = r1_re ? mem[r1_ra] : (xlen)'(0);

endmodule
```

```
arch register_file bit[32] rf_gpr
{
    dataport r0, r1 {flag = R;};
    dataport w0 {flag = W;};
    size = 32;
    reset = true;
    default = 0;
};
```

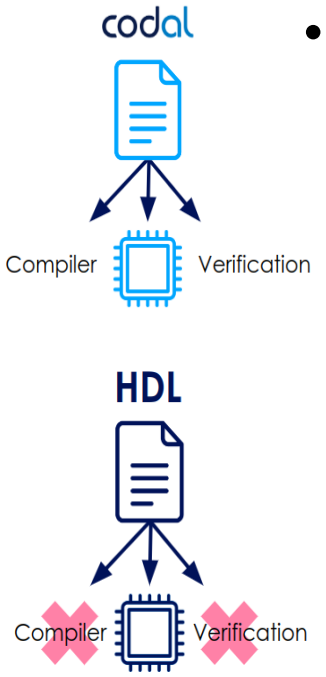
CodALは、標準的なプロセッサの機能設計を容易にする多くの構造体を提供します：

- レジスタファイル
- メモリ (cache, TCM)
- オンチップデバッガ, トレース, etc

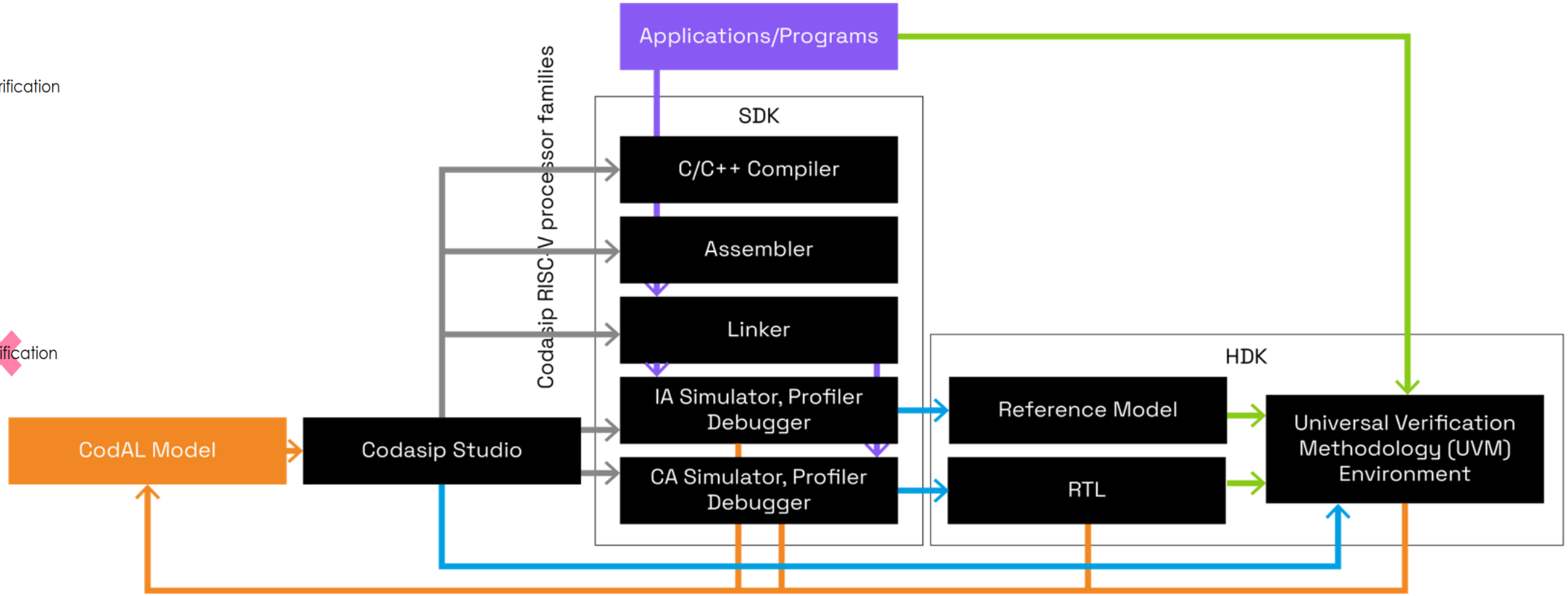
CodALを使用すると、多くの作業が自動化されます。

- 自動モジュール相互接続、デコーダ生成

CodALベースのプロセッサ設計フロー



- CodALはRISC-Vに限らず、RISC-Vが提供するオープンソースISAやCodALがサポートするカスタマイズ機能の恩恵を受けられるところです。



Software Analysis | Processor Modeling | SDK Generation | HDK Generation | Verification

CodAL:生成されるものは？

Software Development Kit (SDK)

- C/C++ LLVM コンパイラ
- C/C++ ライブラリ
  - 例）newlib、標準Cライブラリ
- アセンブラ、ディスアセンブラ、リンカ
- 高性能な命令セット精度及びサイクル精度シミュレータ
- デバッガとプロファイラ
- ドキュメンテーションとISAの可視化
- 検証に使用するランダムプログラム

Hardware Development Kit (HDK)

- RTL (Verilog/VHDL/SystemVerilog)
  - 可読性高い
  - CodALのソースへのリンク
- SystemVerilog UVMのテスト環境
- 統合されたテストベンチ
- EDA ツール用のサンプルスクリプト
- SystemCコ・シミュレーションモデル

ユースケースの紹介

4x4 Systolic array of MACs

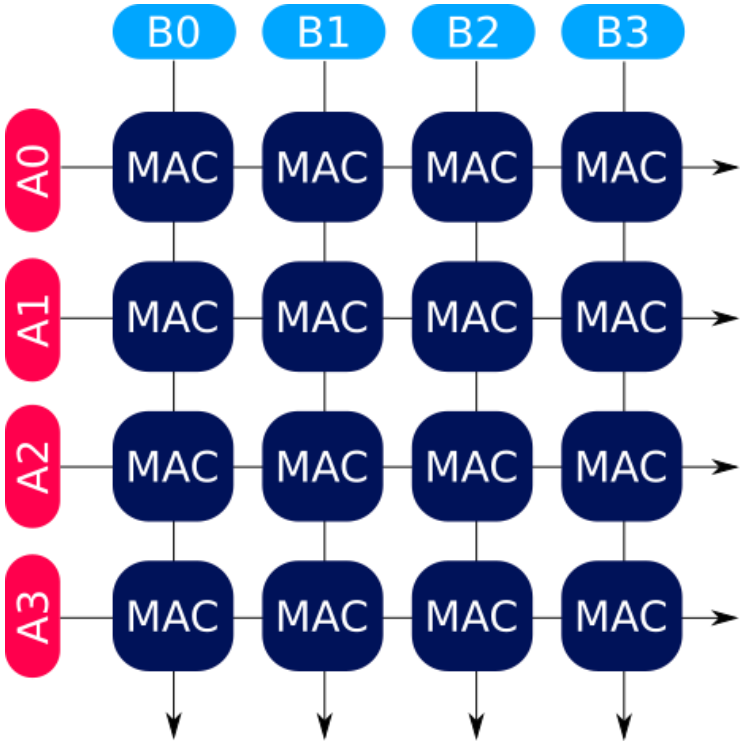
IA model

```
element i_sysarray_math
{
    use sysarray_math as opc;
    use reg_any as dst, src;

    assembly { opc "," dst "," src };
    binary { 0:bit[12] src opc dst OPC_X };
    semantics
    {
        int i;
        uint32 val, result;
        codasip_compiler_unused();
        codasip_compiler_builtin();

        val = rf_gpr_read(src);

#pragma simulator
        {
            switch (opc)
            {
                case OPC_SYS_ARR_FLUSH:
                    for (i=0; i<SYS_LENGTH; i++)
                    {
                        S0[i] = 0;
                        S1[i] = 0;
                        S2[i] = 0;
                        S3[i] = 0;
                    }
                    break;
                case OPC_SYS_ARR_SETA:
                    for (i=0; i<SYS_LENGTH-1; i++)
                    {
                        A[i] = A[i+1];
                    }
                    A[SYS_LENGTH-1] = val;
                    break;
                case OPC_SYS_ARR_SETB:
                    for (i=0; i<SYS_LENGTH-1; i++)
                    {
                        B[i] = B[i+1];
                    }
                    B[SYS_LENGTH-1] = val;
                    break;
                case OPC_SYS_ARR_CALC:
                    for (i=0; i<SYS_LENGTH; i++)
                    {
                        S0[i] = (uint32)((int32)(S0[i]) + (int32)(A[i]) * (int32)(B[0]));
                        S1[i] = (uint32)((int32)(S1[i]) + (int32)(A[i]) * (int32)(B[1]));
                        S2[i] = (uint32)((int32)(S2[i]) + (int32)(A[i]) * (int32)(B[2]));
                        S3[i] = (uint32)((int32)(S3[i]) + (int32)(A[i]) * (int32)(B[3]));
                    }
                    break;
                default: break;
            }
        }
#pragma compiler
        {
            result = (opc==OPC_SYS_ARR_FLUSH || opc==OPC_SYS_ARR_SETA || opc==OPC_SYS_ARR_SETB || opc==OPC_SYS_ARR_CALC) ? val : (uint32)0;
            rf_gpr_write(dst, result);
        }
    };
};
```

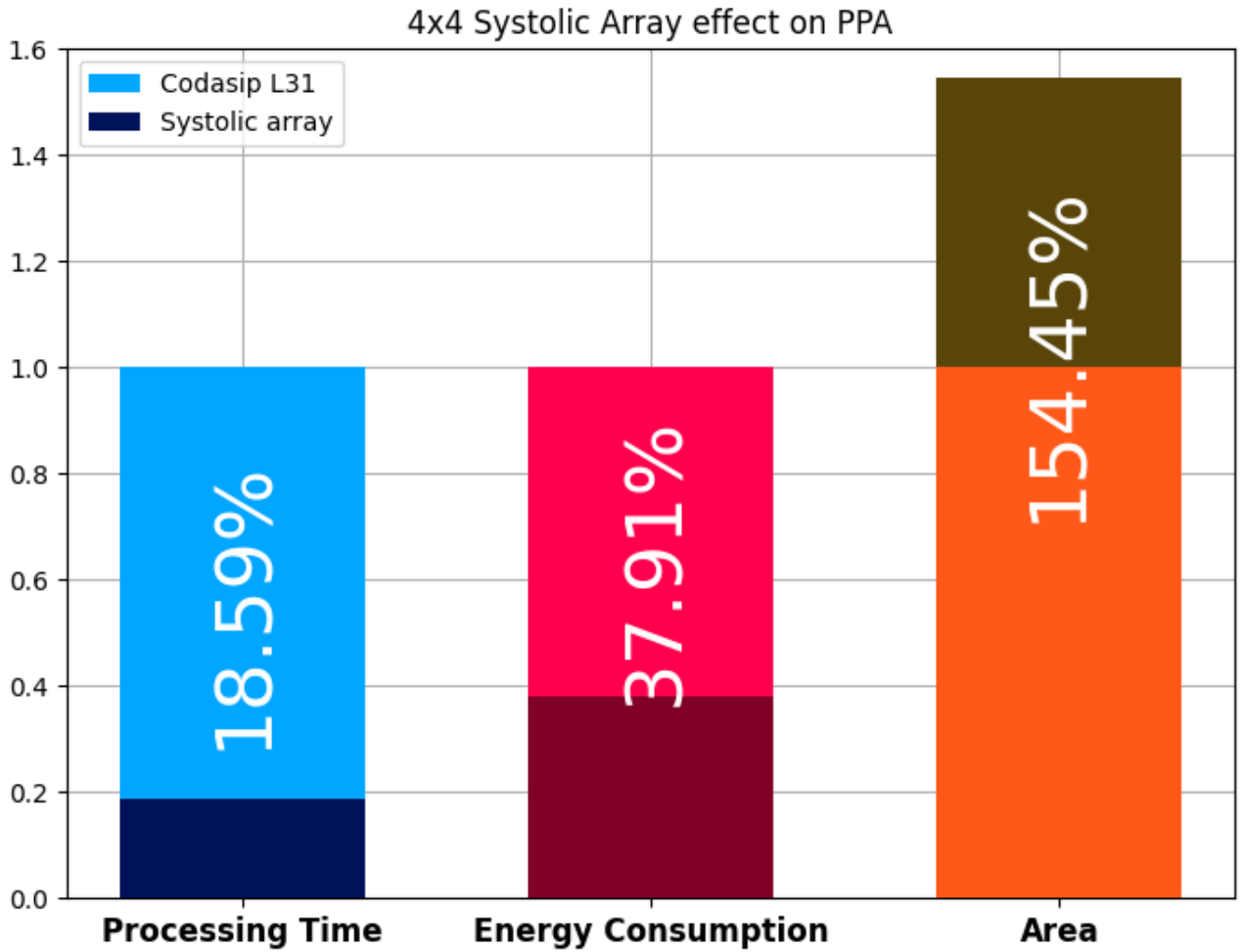
CA model

```
if (ex_fu == FU_SYS_MATH)
    {
        switch (ex_op)
        {
            case OP_SYS_ARR_FLUSH:
                for (i=0; i<SYS_LENGTH; i++)
                {
                    S0[i] = 0;
                    S1[i] = 0;
                    S2[i] = 0;
                    S3[i] = 0;
                }
                break;
            case OP_SYS_ARR_SETA:
                for (i=0; i<SYS_LENGTH-1; i++)
                {
                    A[i] = A[i+1];
                }
                A[SYS_LENGTH-1] = rs1_data;
                break;
            case OP_SYS_ARR_SETB:
                for (i=0; i<SYS_LENGTH-1; i++)
                {
                    B[i] = B[i+1];
                }
                B[SYS_LENGTH-1] = rs1_data;
                break;
            case OP_SYS_ARR_CALC:
                for (i=0; i<SYS_LENGTH; i++)
                {
                    S0[i] = (uint32)((int32)(S0[i]) + (int32)(A[i]) * (int32)(B[0]));
                    S1[i] = (uint32)((int32)(S1[i]) + (int32)(A[i]) * (int32)(B[1]));
                    S2[i] = (uint32)((int32)(S2[i]) + (int32)(A[i]) * (int32)(B[2]));
                    S3[i] = (uint32)((int32)(S3[i]) + (int32)(A[i]) * (int32)(B[3]));
                }
                break;
            default: break;
        }
    }
```

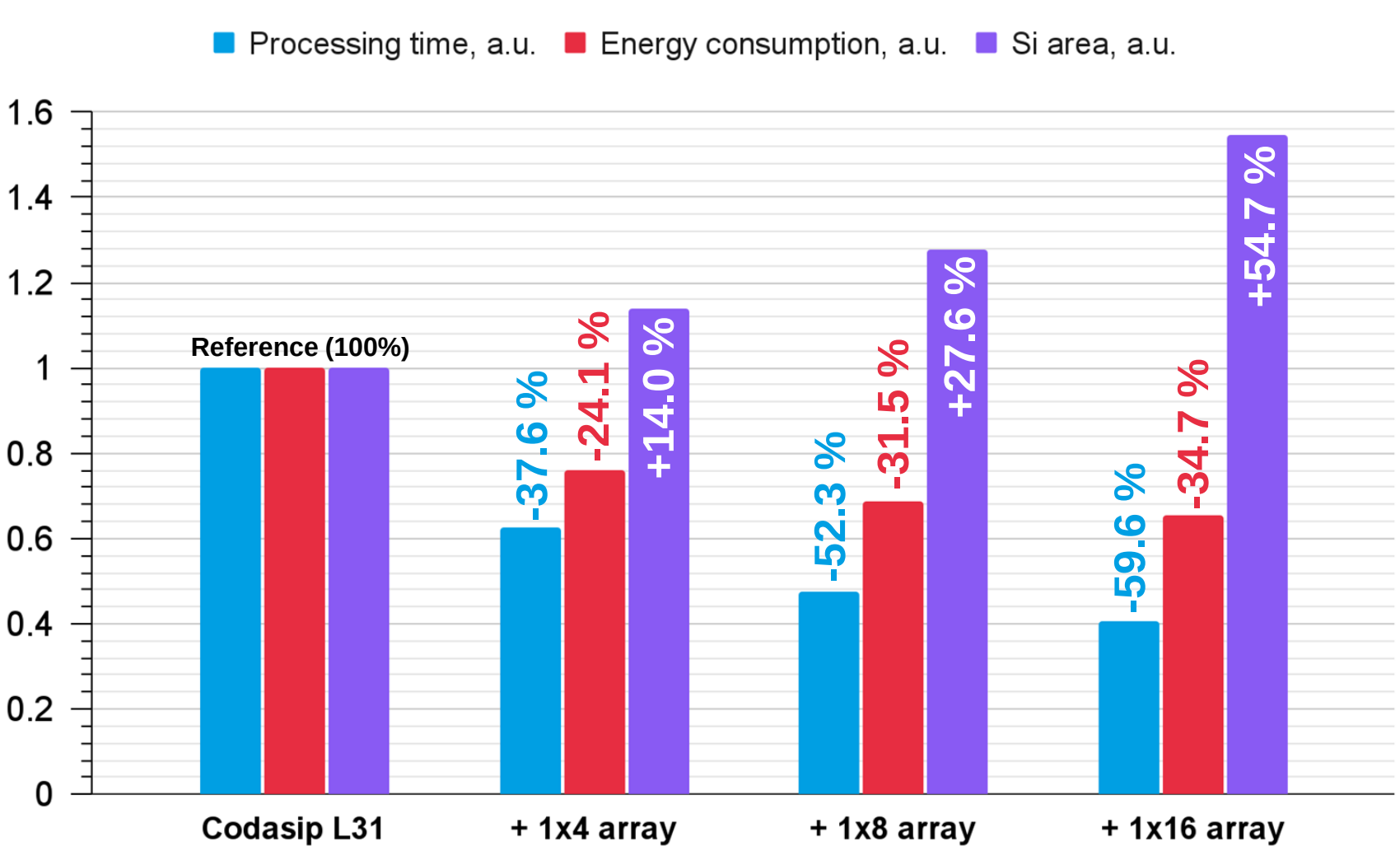
4x4 Systolic array of MACs

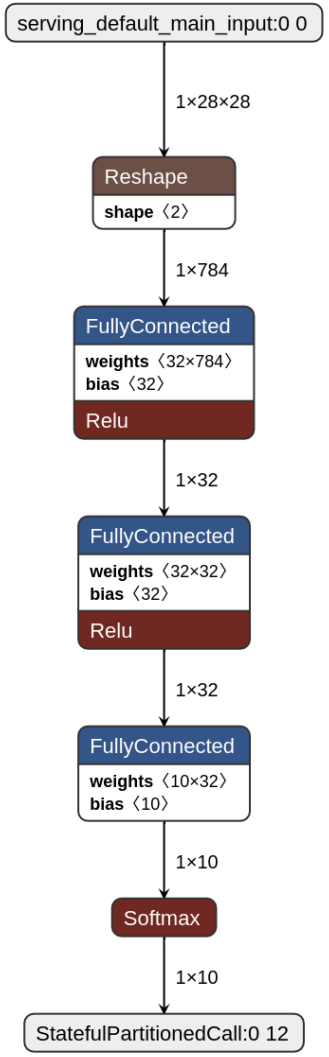



```
case OPC_SYS_ARR_CALC:
    for (i=0; i<SYS_LENGTH; i++) {
        S0[i] = (int32)(S0[i]) + (int32)(A[i]) * (int32)(B[0]);
        S1[i] = (int32)(S1[i]) + (int32)(A[i]) * (int32)(B[1]);
        S2[i] = (int32)(S2[i]) + (int32)(A[i]) * (int32)(B[2]);
        S3[i] = (int32)(S3[i]) + (int32)(A[i]) * (int32)(B[3]);
    } break;
```



Systolic arrays tests on TF-Lite application





Multicycle custom CRC32 instruction

IA model

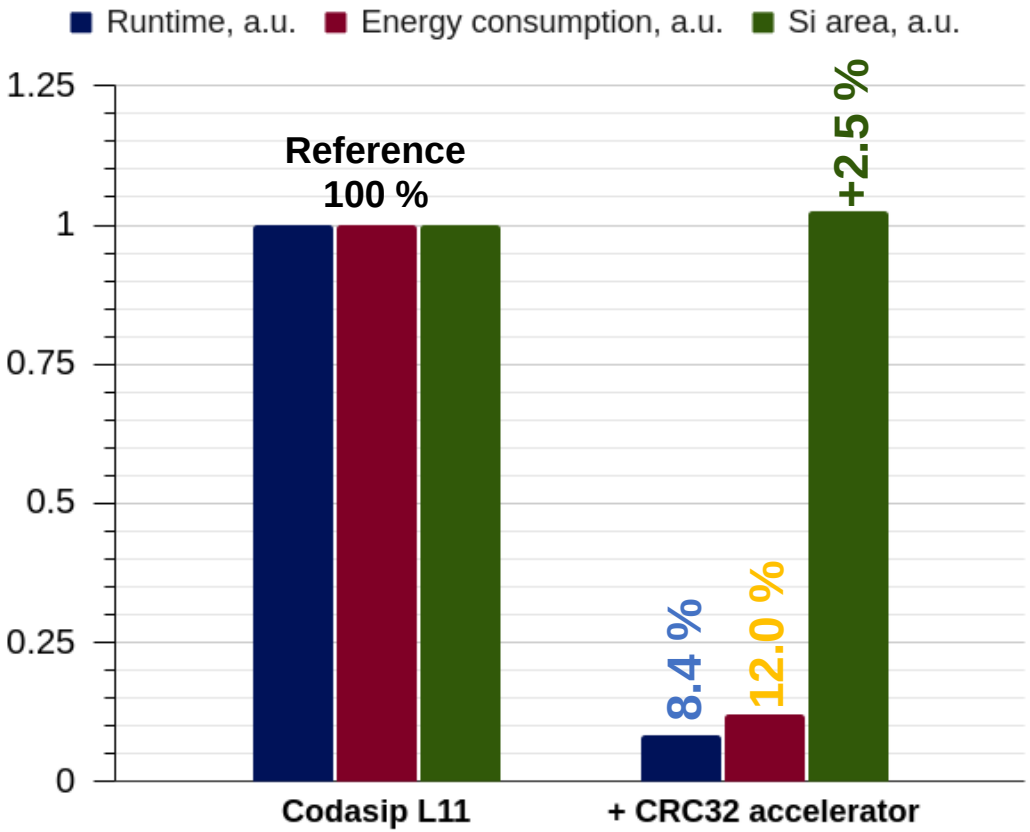
```
element i_ext_crc32
{
    use opc_crc32_i as opc;
    use reg_any as rd, rs1, rs2;

    assembly { opc rd "," rs1 "," rs2};
    binary { 0:bit[7] rs2 rs1 opc rd OPC_X };

    semantics
    {
        uXlen result, src1, src2;
        uint32 crc32_bit;
        uint8 crc32_byte;
        int crc32_bitcounter;

        src1 = rf_gpr_read(rs1);
        src2 = rf_gpr_read(rs2);

        result = src2;
        crc32_byte = (uint8)(src1 & 0x000000FF);
        for (crc32_bitcounter=0; crc32_bitcounter<8; crc32_bitcounter++)
        {
            crc32_bit = (crc32_byte ^ result) & 0x00000001;
            result >>= 1;
            if (crc32_bit)
            {
                result = result ^ 0xEDB88320u;
            }
            crc32_byte >>= 1;
        }

        rf_gpr_write(rd, result);
    };
};
```

CA model

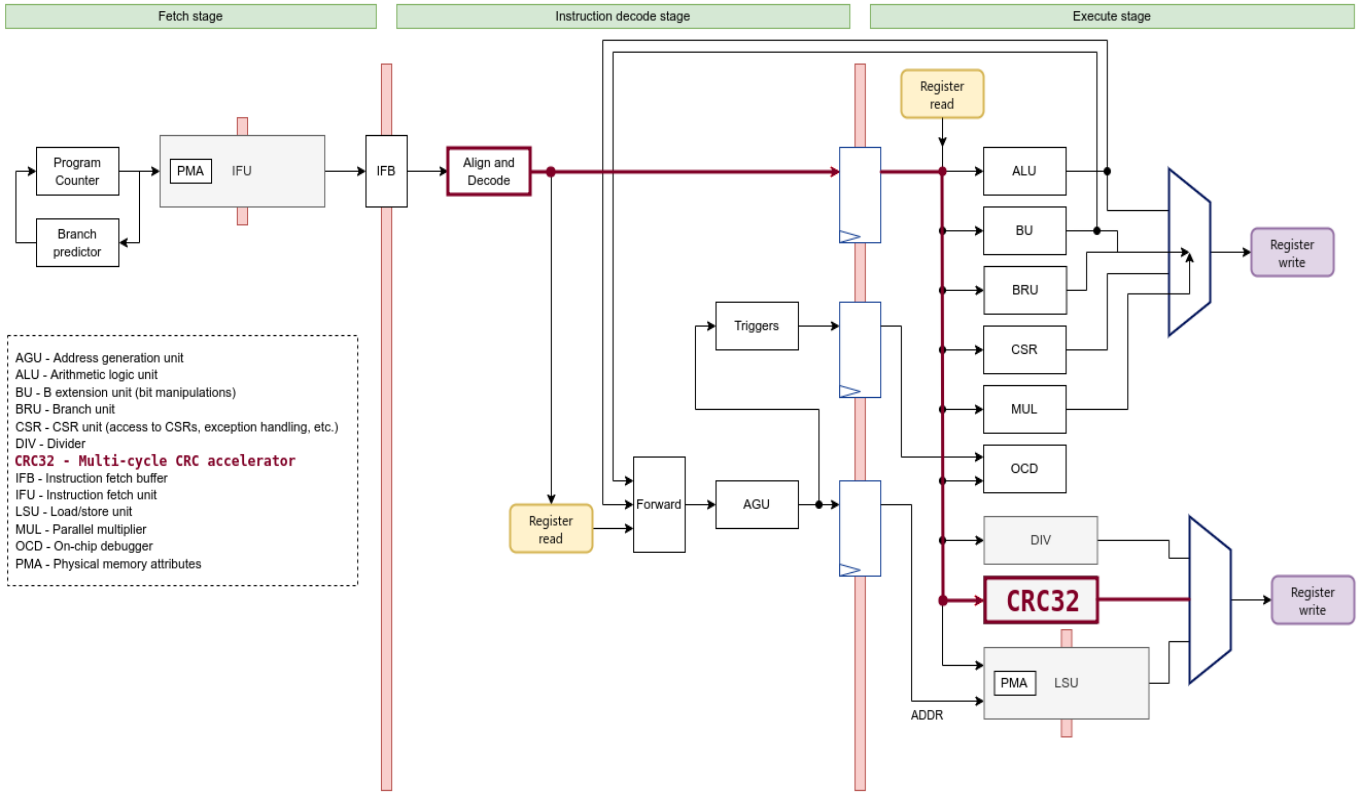
```
event compute
{
    semantics
    {
        uint_<32> s_crc32_bit;
        bool crc32_op;

        switch (r_state)
        {
            case CRC32_IDLE:
                crc32_op = (s_op_i == OP_CRC32);

                r_counter = 8;
                r_op = s_op_i;
                r_crc32_byte = s_s1_data_i[7..0];
                r_val = s_s2_data_i;

                r_state = s_flush_i || !crc32_op ? CRC32_IDLE : CRC32_COMPUTE;
                break;
            case CRC32_COMPUTE:
                s_crc32_bit = (r_crc32_byte ^ r_val) & 0x00000001;
                if (s_crc32_bit)
                {
                    r_val = (r_val>>1) ^ 0xEDB88320u;
                }
                else r_val = r_val>>1;
                r_crc32_byte = r_crc32_byte>>1;

                r_counter = r_counter - (uint_<CNT_W>)1;
                r_state = s_flush_i || r_counter == 0 ? CRC32_IDLE : CRC32_COMPUTE;
                break;
```

Multicycle custom CRC32 instruction

PPA result



CRC32 implementation



結論

- MACを格子状に配置（Systolic）して、行列演算やTensor Flow Lite for Microcontrollerを実行したところ、カスタマイズしていないプロセッサと比較して
  - 実行時間：約5倍（4x4 Systolic）、約1.6~2.4倍（TFLite）
  - 消費電力：約63%削減（4x4 Systolic）、約24~34%削減（TFLite）
  - エリア：約54%増加（4x4 Systolic）、約14~54%増加（TFLite）

- CRC32を加速するための命令を追加して、ベースのプロセッサと比較したところ
  - 実行時間：約12倍
  - 消費電力：約88%削減
  - エリア：2.5%の増加

- Custom Computeのための高水準言語CodALのメリット
  - 高速なデザイン探索
  - シストリック配列の実装でコードサイズが小さい
  - カスタムインストラクションのSDKサポート