一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2025-12,DC2025-12,RECONF2025-12(2025-06)

RISC-V ソフトプロセッサでカスタム命令を簡単に扱うフレームワーク CFU Proving Ground の設計と実装

藤野　蒼羽†　　吉瀬　謙二†

† 東京科学大学 情報理工学院　〒152–8550 東京都目黒区大岡山 2 丁目 12–1
E-mail: †fujino.a.85bc@m.isct.ac.jp, ††kise@comp.isct.ac.jp

あらまし　ASIP (Application-Specific Instruction-set Processor) の開発において，ハードウェアとソフトウェアの不整合による開発サイクルの長期化が課題になっている．ハードウェアとソフトウェア (HW/SW) の協調設計の導入により開発環境の改善が進んでいるが，柔軟性や拡張性の点で改善すべき点が多い．本研究では，FPGA をターゲットとして，CFU (Custom Function Unit) と呼ばれるアクセラレータを用いて高性能化を狙う HW/SW 協調設計環境の CFU Proving Ground を提案する．CFU Proving Ground は，RTL ベースの設計フローと，RISC-V のカスタム命令を処理する資源効率の高い CFU をアクセラレータとして採用することで，ASIP の迅速な開発を支援する．また，軽量なライブラリ，明確なファイル依存関係，柔軟なメモリマップを提供することで，ユーザービリティの向上を狙う．Arty A7 FPGA ボードを用いた評価結果から提案フレームワークの有用性を議論する．
キーワード　FPGA，RISC-V，CFU，フレームワーク，カスタム命令，アクセラレータ

CFU Proving Ground: a framework for efficiently utilizing custom instructions in RISC-V soft processors

Aoba FUJINO† and Kenji KISE†

† Institute of Science Tokyo, School of Computing　2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8550 Japan
E-mail: †fujino.a.85bc@m.isct.ac.jp, ††kise@comp.isct.ac.jp

Abstract　In ASIP (Application-Specific Instruction-set Processor) development, mismatches between hardware and software often prolong the development cycles. While hardware/software (HW/SW) co-design approaches have improved development environments, challenges remain regarding flexibility and scalability. This research proposes CFU Proving Ground, an HW/SW co-design environment targeting FPGAs that improve performance using CFU (Custom Function Unit) as an accelerator. CFU Proving Ground supports agile ASIP development through RTL-based design flows and a resource-efficient CFU that processes RISC-V custom instructions. It enhances usability by providing lightweight libraries, explicit file dependencies, and flexible memory mapping. We discuss the framework's effectiveness based on evaluation results using the Arty A7 FPGA board.
Key words　FPGA, RISC-V, CFU, Framework, Custom Instructions, Accelerators

1. はじめに

ASIP (Application-Specific Instruction-set Processor) は，低コストで高効率に，特定のアプリケーションの高速化を目指すプロセッサであり，組込みシステムなどのエッジデバイスで注目を集めている [1]．ASIP の基本的な実装方法では，既存のプロセッサにアクセラレータを統合する [2] [3] [4] [5]．アクセラレータには，主に，大規模な専用ハードウェアとして実現するコプロセッサベースの手法と，プロセッサパイプラインに組み込まれる ALU ベースの方法の 2 種類がある．後者の ALU ベースのアクセラレータは，命令セットアーキテクチャが提供するカスタム命令を実行する機能ユニットであり，**Custom Function Unit (CFU)** と呼ばれる．本研究では，命令セットアーキテクチャとして RISC-V を採用する．

CFU の制御や演算に必要なデータのやりとりは，カスタム命令を通じて行われる．カスタム命令は 2 個の入力オペランドと 1 個の出力オペランドを持ち，RISC-V 命令セットアーキテクチャの仕様に準拠している．ユーザーがカスタム命令を自由に定義することが可能であり，アプリケーション固有の最適化に活用できる．しかしながら，特定のアプリケーションの高速

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2025 by IEICE

化に効果的な ASIP の構成を見つけ出すことは困難な課題である [6]．一般的な設計手法では，ハードウェアとソフトウェアを独立に設計するため，ASIP の構成がアプリケーション特化ではなく，汎用的な構成に近づく可能性がある．このため，アプリケーションのボトルネックを十分に改善できず，結果として期待する性能向上を達成できないことがある [7]．

この課題の解決策として，FPGA を活用する HW/SW 協調設計フレームワークが注目されている．FPGA は再構成可能なハードウェアプラットフォームとして，迅速な開発と柔軟なハードウェアの設計を可能にするため，ASIP を実装するプラットフォームとして理想的である [8]．特に，ハードウェアを再構成できるという特徴には，アプリケーション固有の演算を効率的に実装し，性能向上とエネルギー効率を同時に最適化するという利点がある [9]．

本研究では，効率的かつ柔軟な ASIP の設計を実現する FPGA ベースの HW/SW 協調設計フレームワークである **CFU Proving Ground** を提案する．このフレームワークは，RTL (Register Transfer Level) のユーザーフレンドリーな設計手法を提供し，高性能な ASIP を容易に設計・実装するための HW/SW 協調設計フレームワークである．また，CFU を採用することで，アクセラレータのインターフェースが統一され，ASIP の設計のための労力を改善できる．そのソースコードは MIT ライセンスの下で，GitHub[1]で公開しており，ユーザーが独自の要件に応じてカスタマイズできる柔軟性を備えている．

本研究の貢献は以下の通りである．

- FPGA をターゲットとする HW/SW 協調設計フレームワークである CFU Proving Ground を提案する．このフレームワークでは，アプリケーションのボトルネックを解析する手段を提供して，高性能な ASIP の設計と実装を支援する．
- 幾つかのハードウェア設計を用いて，提案手法の CFU Proving Ground を評価して，その有効性を明らかにする．
- MIT ライセンスの下でオープンソースとして公開し，オープンソースコミュニティや産業界での応用に貢献する．我々が知る限り，CFU Proving Ground はシンプルなライセンスの依存関係の初の HW/SW 協調設計フレームワークである．
- CFU Proving Ground はファイルの依存関係を簡潔にすることで計 23 ファイル，2681 行のコードで実装されている．これは競合するフレームワークより格段に軽量で，全体の理解を素早くでき，ユーザーがシステムの要件に応じてセンサーを追加したり，独自の評価ボードで実装することを容易にする．

2. CFU Proving Ground の提案

図 1 に，本研究で提案する CFU Proving Ground の設計フローを示す．このフレームワークは，C 言語によるアプリケーション開発，RISC-V のカスタム命令の設計と Verilog HDL による実装，ASIP の設計と実装，そして，ハードウェアの性能評価と検証といった作業を統合的に支援する．

図 1 の設計フローは，その左側に示す設計・実装フローと右側に示すシミュレーションフローを持つ．設計・実装フローでは，ユーザーが記述した C 言語プログラム (`main.c`) を起点とする．このプログラムを，フレームワークに付属するミニディスプレイ用ライブラリ (st7789 library) およびパフォーマンス測定用ライブラリ (performance library) と共に，RISC-V コンパイラによってコンパイルする．コンパイルによって生成されるファイルを入力として，フレームワークが提供するスクリプトによってプロセッサの命令メモリとデータメモリを初期化するためのファイル (`memi.txt`, `memd.txt`) を生成する．また，後述するカスタム命令と CFU の定義に従って，Verilog HDL のファイル (`cfu.v`) に CFU のハードウェアを記述する．これらのファイルとプロセッサを含む SoC を記述する Verilog HDL のファイルを用いて，AMD 社の Vivado により，FPGA のビットストリームファイル (`main.bit`) を生成する．このビットストリームファイルを用いて，ユーザーは FPGA をコンフィギュレーションし，FPGA の動作を確認する．

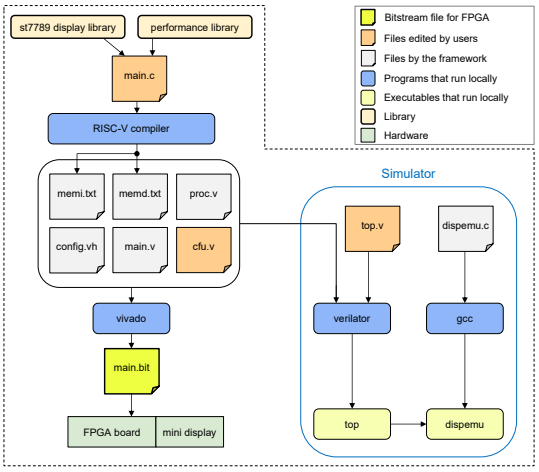

図 1: 提案する CFU Proving Ground の設計フロー

右側に示すシミュレーションフローでは，テストベンチのトップモジュールの記述 (`top.v`) とプロセッサを含む SoC を記述する Verilog HDL ファイルを Verilator でコンパイルし，SoC をシミュレーションするためのプログラムを生成する．さらに，SoC の出力を表示するミニディスプレイのエミュレータ (`dispemu.c`) のコードをホストコンピュータの GCC でコンパイルする．トップモジュールからディスプレイのエミュレータに画面描画のためのデータを送りながら，ミニディスプレイを含むシステム全体のシミュレーションを実現する．このシミュレーションフローを用いてアプリケーションのボトルネックを特定し，その特定したボトルネックから，カスタム命令と CFU を策定する．

2.1 SoC (System on Chip) の構成

CFU Proving Ground が提供する SoC は，RISC-V プロセッサの RVProc，CFU，命令メモリ，データメモリ，ビデオメモリ，ディスプレイコントローラを持つ．図 2 に、RV32IM をサポートする RISC-V プロセッサの RVProc のデータパスを示す．

このプロセッサは，我々が Verilog HDL で開発したものであり，命令フェッチ (If)，デコード (Id)，実行 (Ex)，メモリアク

(注1)：https://github.com/archlab-sciencetokyo/CFU-Proving-Ground

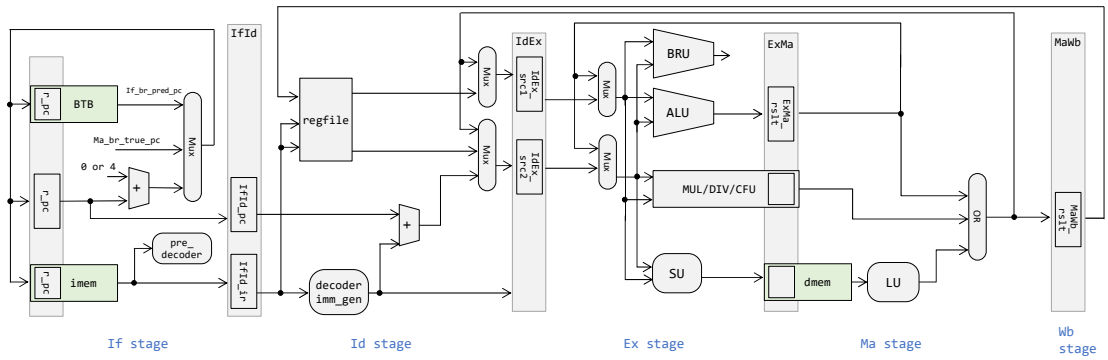

図 2: RVProc のデータパス

```
module cfu (
    input  wire        clk_i,
    input  wire        en_i,
    input  wire [ 2:0] funct3_i,
    input  wire [ 6:0] funct7_i,
    input  wire [31:0] src1_i,
    input  wire [31:0] src2_i,
    output wire        stall_o,
    output wire [31:0] rslt_o
);
    assign stall_o = 0;
    assign rslt_o = (en_i) ? src1_i | src2_i : 0;
endmodule
```

図 3: モジュール CFU の記述例

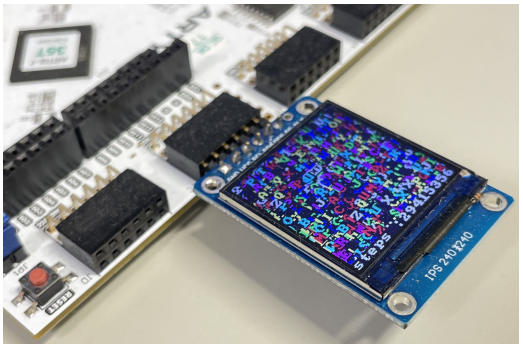

図 4: Arty A7-35T FPGA ボードと ST7789 ミニディスプレイ

セス (Ma)，ライトバック (Wb) という 5 段のパイプライン構成のスカラプロセッサである．分岐予測器として bimodal を搭載して，分岐命令のオーバヘッドを低減する．また，RISC-V のカスタム命令を処理する CFU は Ex ステージに配置され，命令フィールドの funct3 と funct7 によって CFU の動作を制御する．また，通常の ALU と同様に，32 ビットの 2 個のオペランドを入力として，1 個のオペランドを生成する．命令メモリ (imem) とデータメモリ (dmem) は FPGA の BRAM によって実現する．これらの容量の標準設定はそれぞれ 32 KiB と 16 KiB であるが，これらの容量はユーザーによって適切に設定する．

フレームワークが提供するスクリプトによって生成される `memi.txt` と `memd.txt` というファイルによって命令メモリとデータメモリの内容を初期化する．

2.2 カスタム命令と CFU

RISC-V 命令セットアーキテクチャは拡張性を重視しており，標準の命令セットでは定義されない独自の演算や機能を実装するためのカスタム命令を提供する．CFU Proving Ground では，このカスタム命令を活用し，ソフトウェアから CFU の制御やデータのやりとりが可能となり，ハードウェアとソフトウェアの協調設計を実現する．カスタム命令は R 形式に基づいており，最大で 2 個の入力オペランドと，1 個の出力オペランドを指定できる．

CFU の記述例を図 3 に示す．デコーダーから funct7 と funct3 を受け取り，カスタム命令の動作を制御する．ただし，この例ではこれらの値を利用せずに，常に，2 個の入力オペランドの論理和を処理する記述としている．CFU では，複数サイクルを用いて演算をすることが可能であり，その場合は `stall_o` をアサートして，プロセッサをストールさせる．プロセッサの既存のデータパスを拡張する方針で CFU を追加しているため，制御ハザードやデータハザードの問題は既存のパイプライン制御によって適切に処理される．

2.3 ミニディスプレイと MMIO

提案するフレームワークは，240 × 240 ピクセルのミニディスプレイをサポートする．このために，24 KiB のビデオメモリを備えて，その読み書きのために Memory Mapped I/O (MMIO) のアドレスを定義し，各アドレスには RGB 形式の 3 ビットカラーデータを格納する．

図 4 に，フレームワークが標準で提供するベンチマークプログラムを実行する様子を示す．ここでは，ランダムな場所にランダムな文字を表示しながら，表示した文字の数を表示している．このミニディスプレイを利用することで，簡単に，グラフィカルな出力や性能測定の結果を視覚化できる．

性能評価などで利用する 64 ビットカウンタ mcycle を実装し，停止やリセットなどの制御を行うための MMIO のアドレスを設けた．そして，シミュレーションの停止を要求する MMIO のアドレスを設け，デバック作業の効率化を図った．

これらのパフォーマンスカウンタとビデオメモリの実装により，アプリケーションのボトルネックについて視覚的なフィードバックを容易に得ることができる．

表 1: CoreMark のスコア

Processer	F_{max} [MHz]	CoreMark/MHz	CoreMark
ESP32-C3 [11]	160	2.55	408
VexRiscv full mex perf [12]	200	2.57	514
RVProc (proposal)	235	2.65	622

3. 評　　価

3.1 評 価 環 境

評価には，Arty A7-35T FPGA ボードを用いる．コンパイルなどは Intel Core i9-12900KF CPU と 128GiB DDR4 メモリを搭載するコンピュータを用いる．OS は Ubuntu 22.04.2 LTS である．論理合成，配置・配線，ビットストリームの生成には AMD Xilinx Vivado 2024.2 を使用する．プログラムの実行サイクル数の測定には，CFU Proving Ground が提供するパフォーマンスライブラリを用いる．

CoreMark ベンチマーク [10] を用いて RVProc の性能を定量的に評価し，他のプロセッサと比較する．また，CFU Proving Ground の有効性を実証するため，本フレームワークの設計フローに沿った CFU 実装による性能向上を分析する．指標として，ベンチマークの実行に必要となるサイクル数とハードウェア使用量を用いる．ベンチマークとして，バックトラック法を用いた N-Queen，非再帰 3 重ループ構造で実装した高速フーリエ変換を用いる．この 2 種類のアルゴリズムを採用することで，異なる計算量に対する CFU Provoing Ground の有効性を評価する．

3.2 CoreMark の評価結果

CoreMark はプロセッサの性能を測定するためのベンチマークである．ベンチマークのコンパイルには，GCC14.2.0 を使用した．イテレーションは 10000，コンパイラオプションは`-O2 -march=rv32im -mabi=ilp32 -nostartfiles` を用いた．RVProc は命令メモリ 32 KiB, データメモリ 16 KiB の構成を使用した．測定環境を VexRiscv に合わせ，選択可能な最大のスピードグレードを選択した．この構成の，5 MHz 単位で計測した最大の動作周波数 (F_{max}) は 235 MHz である．

表 1 に CoreMark の測定結果を示す．ESP32-C3 は，販売されている RISC-V プロセッサを搭載する ASIC の SoC である．VexRiscv では，FPGA フレンドリーな 32 ビット RISC-V プロセッサを実装している．ESP32-C3 の CoraMark スコアや動作周波数は，文献から引用した．VexRiscv の CoraMark スコアや動作周波数は，そのウェブページに記載されている値を用いた．表 1 の結果から，CFU Proving Ground が用いるプロセッサ RVProc のスコアは 622 であり，関連する RISC-V プロセッサより優れた性能であることがわかる．

3.3 アプリケーション高速化

3.3.1 N-Queen の評価結果

バックトラック法を用いた N-Queen をベンチマークとして，本フレームワークの設計フローを用いて CFU の設計を行った．カスタム命令を使用しないプログラムの性能を，比較のための

```
1  void nqueen_kernel() {
2    for (int i=0; i<(n/2 + n%2); i++) {
3        int h = 1;      /* height or level  */
4        int r = 1 << i; /* candidate vector */
5        a[h].col = (1<<n)-1;
6        a[h].pos = 0;
7        a[h].neg = 0;
8        long long ret = qn(n, h, r, a);
9        answers += ret;
10       if(i!=n/2) answers += ret;
11   }
12 }
```

図 5: ベースラインの N-Queen カーネル

```
1  void cfu_nq_ini(int i) {
2      asm volatile ("cfu.nq.ini %0" : : "r"(i));
3  }
4
5  int cfu_nq_ex() {
6      int ret;
7      asm volatile("cfu.nq.ex %0" : "=r"(ret));
8      return ret;
9  }
10
11 void nqueen_kernel() {
12     for (int i=0; i<(n/2 + n%2); i++) {
13         cfu_nq_ini(i);
14         long long ret = cfu_nq_ex();
15         answers += ret;
16         if(i!=n/2) answers += ret;
17     }
18 }
```

図 6: カスタム命令を使う N-Queen カーネル

ベースラインとして用いる．ベースラインのカーネルを図 5 に示す．

これに対し，提案手法では図 5 のカーネルの処理の一部を 2 個のカスタム命令で置き換え，CFU で高速化する．図 6 に，カスタム命令に置き換えたプログラムを示す．

カスタム命令はインラインアセンブラを利用して使用する．これを行っているのが関数 `cfu_nq_ini()` と `cfu_nq_ex()` である．

ベースラインではループの最初に変数の初期化を行っている．これをカスタム命令を呼び出す関数 `cfu_nq_ini()` に割り当て，初期化した値を CFU 上のレジスタに記録している．ベースラインにおける関数 `qn()` は `cfu_nq_ex()` に対応する．

以上の演算を行う CFU を実装したプロセッサの F_{max} は 175 MHz であった．本稿の著者がこの CFU の設計と実装に要した期間は約 3 日間と短かった．

問題サイズ (N) を 12 〜 16 に変化させて測定した結果を図 7 に示す．性能の指標として実行時間と速度向上を用いた．提案手法ではベースラインと比較して，平均 18 倍という大幅な高速化を実現できている．

3.3.2 高速フーリエ変換の評価結果

もう一つのベンチマークとして，Q16.16 フォーマットの固

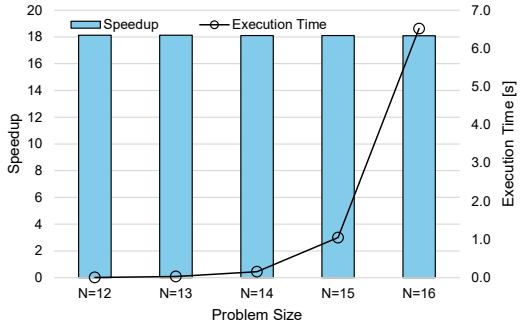

図 7: N-Queen のサイズと実行時間 (秒)

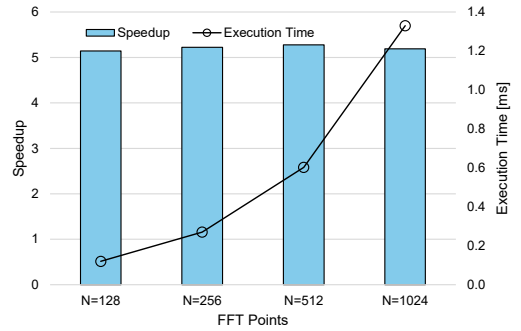

図 8: FFT のポイント数と実行時間 (ms)

定小数を用いて，3 重ループで実装する高速フーリエ変換を用いた．カスタム命令を使用しないプログラムの性能を比較するベースラインとして用いる．これに対し，提案手法では高速フーリエ変換のバタフライ演算をカスタム命令で置き換え，その命令を処理する CFU で高速化した．CFU を実装したプロセッサの F_{max} は 180 MHz であった．本稿の著者がこの CFU の設計と実装に要した期間は約 5 日間と短かった．

FFT ポイント数 (N) を 128 〜 1024 に変化させて測定した結果を図 8 に示す．性能の指標として実行時間と速度向上を用いた．バタフライ演算をカスタム命令に置き換えることによって，ベースラインと比較して，平均 5.2 倍の性能向上を達成できた．

3.4 ハードウェア使用量の評価

CFU を使用することで生じる実装面積のオーバヘッドを N-Queen と高速フーリエ変換のケースで測定した．ベースラインとしてカスタム命令や CFU のインターフェースをサポートしない SoC を用いた．

表 2 に，デフォルトの構成のハードウェア使用量を示す．この RVProc では，命令メモリを 32 KiB，データメモリを 16 KiB とした．このとき，F_{max} である 180 MHz で動作させた．CFU Usage は Vivado の `report_utilization -hierarchical` コマンドで得られる値を記した．Total Usage は SoC 全体のリソース量を記したものである．

表 3 に，N-Queen のハードウェア使用量を示す．この構成では，命令メモリを 8 KiB，データメモリを 4 KiB とした．この時の F_{max} は 175 MHz である．N-Queen の実装では，LUT と FF の使用量が大きく増加した．これは，複雑な論理演算を用いているため，保持しなければならないデータが多いからである

表 2: デフォルトの構成のハードウェア使用量

Resouces	CFU Usage	Total Usage	Available	Total Usage Rate
LUT	0	1771	20800	8.51%
FF	0	872	41600	2.10%
BRAM	0	19	50	38.00%
DSP	0	4	90	4.44%

表 3: N-Queen のハードウェア使用量 (N=16)

Resouces	CFU Usage	Total Usage	Available	Total Usage Rate
LUT	1153	2819	20800	13.55%
FF	1279	2753	41600	6.62%
BRAM	0	10	50	20%
DSP	0	4	90	4.44%

表 4: FFT のハードウェア使用量 (N=1024)

Resouces	CFU Usage	Total Usage	Available	Total Usage Rate
LUT	537	2236	20800	10.75%
FF	598	1508	41600	3.63%
BRAM	0	23	50	46.00%
DSP	16	20	90	22.22%

と考えられる．

表 4 に，高速フーリエ変換のハードウェア使用量を示す．この構成では，命令メモリ 8 KiB, データメモリ 32 KiB とした．このとき F_{max} は 180 MHz である．FFT の実装では，BRAM と DSP が大きく増加した．この増加は，FFT のための回転因子テーブルで BRAM を使用したためと考えられる．また，バタフライ演算は複数の乗算を行うので，それが原因で DSP が増加したと考えられる．

どちらの構成においても，FPGA の利用率 (Total Usage Rate) に余裕があり，無理なくこれらの CFU が実装できていることがわかる．

4. 関 連 研 究

Chipyard [13] は，Rocket コアおよび BooM コアを用いた SoC 自動生成フレームワークである．RoCC インターフェースを介して典型的なアクセラレータを追加することで，一定の HW/SW 協調設計の課題に対応している．しかし，習得にかかるコストの大きさとアーキテクチャ上の非効率性により，依然として HW/SW 協調設計における顕著な障壁が存在している．

ASIP Designer [14] は，ASIP 設計を多様な SoC に対応させるソリューションを提供している．この IP は，アーキテクチャの選択・変更に自動対応する C/C++ コンパイラベースのソフトウェア開発キットの生成，および消費電力と面積に最適化された合成可能な RTL の自動生成を実現している．HW/SW 協調設計における優れたスケーラビリティを提供するものの，商用 IP であるため高額な初期導入コストが必要となる点が大きな制約である．

OpenASIP [15] は RISC-V ベースの迅速なカスタマイズを可能にする ASIP 協調設計ツールである．RTL 生成機能に加え，

カスタム命令を活用した RISC-V プロセッサの高級言語プログラミングをサポートしており，少ない労力での命令セットカスタマイズを実現している．しかしながら，XML ベースのアーキテクチャ定義方式を採用しているため，プロセッサの柔軟性に制約がある．

CFU Playground [16] は，スケーラビリティ向上のために，SoC 生成フレームワークである LiteX とプロセッサ生成フレームワークである VexRiscv を統合したアプローチを提案している．また，限られたリソース環境でレイテンシを重視する TinyML アプリケーション向けに，アクセラレータを反復的に設計・評価する手法も提示している．しかしながら，その機能は LiteX に大きく依存している．さらに，複雑な抽象化による多層的な依存関係により，CFU 以外のカスタマイズが困難となっている．

SawareruSys [17] は，サーバーと FPGA 間の I/O のインターフェースを固定として，ユーザーが記述するモジュールに動的再構成を用いたフレームワークである．CFU にはインターフェースが不変であるという特徴があり，このような動的再構成の利用に適している．この特徴を利用することで，ビットストリームを生成するまでの時間を短縮し，CFU の設計と評価のサイクルを短縮できる可能性がある．

5. ま と め

本稿では，FPGA 開発のための柔軟な HW/SW 協調設計フレームワークである CFU Proving Ground について述べた．CFU Proving Ground は，ALU ベースのアクセラレータである CFU を採用し，アプリケーションからの明示的なアクセラレータへのアクセスを可能とする．さらに，軽量な設計フローにより，高性能な ASIP の迅速な開発を支援する．

評価結果から，CFU を利用しないベースラインと比較して，N-queen のケースで平均 18 倍，高速フーリエ変換のケースで平均 5.2 倍の高速化を達成した．

6. 謝　　辞

本研究は，三菱電機株式会社 情報技術総合研究所からの委託研究として実施しました．関係各位に感謝いたします．

文　　献

[1] Q. Dinh, D. Chen, and M.D.F. Wong, “Efficient asip design for configurable processors with fine-grained resource sharing,” Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays, pp.99–106, FPGA ’08, Association for Computing Machinery, New York, NY, USA, 2008. https://doi.org/10.1145/1344671.1344687

[2] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D.A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, “The Rocket Chip Generator,” April 2016. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y.S. Shao, “Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration,” 2021 58th ACM/IEEE Design Automation Conference (DAC), pp.769–774, IEEE Press, 2021. https://doi.org/10.1109/DAC18074.2021.9586216

[4] M. Chen, X. Li, W. Zhou, Y. Li, and F. Deng, “Looppara: an architecture- transparent acceleration framework for loops by exploiting data-level parallelism,” 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS), pp.22–27, 2021.

[5] S. Kim, J. Zhao, K. Asanović, B. Nikolić, and Y.S. Shao, “Aurora: A full-stack solution for scalable and virtualized accelerator integration,” IEEE Micro, vol.44, no.4, pp.97–105, July 2024. https://doi.org/10.1109/MM.2024.3409546

[6] Q. Si and B. Carrion Schaefer, “Advice: Automatic design and optimization of behavioral application specific processors,” Proceedings of the Great Lakes Symposium on VLSI 2023, pp.327–332, GLSVLSI ’23, Association for Computing Machinery, New York, NY, USA, 2023. https://doi.org/10.1145/3583781.3590214

[7] L. Liu, O. Morozov, Y. Han, J. Gutknecht, and P. Hunziker, “Automatic soc design flow on many-core processors: a software hardware co-design approach for fpgas,” Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp.37–40, FPGA ’11, Association for Computing Machinery, New York, NY, USA, 2011. https://doi.org/10.1145/1950413.1950424

[8] C. Su, L. Du, T. Liang, Z. Lin, M. Wang, S. Sinha, and W. Zhang, “Graflex: Flexible graph processing on fpgas through customized scalable interconnection network,” Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp.143–153, FPGA ’24, Association for Computing Machinery, New York, NY, USA, 2024. https://doi.org/10.1145/3626202.3637573

[9] 原佑輔，福間慎治，“積分画像による重心計算を用いた物体追跡システムの fpga 実装,” 信学技報，vol.124，no.330，pp.23–28，2025.

[10] EEMBC, “CoreMark,” https://github.com/eembc/coremark, 2009. Accessed: March 17, 2025.

[11] ESPRESSIF, “ESP32-C3 Series Datasheet Version 2.0,” 2024.

[12] SpinalHDL, “VexRiscv,” https://github.com/SpinalHDL/VexRiscv, 2025. GitHub repository. Accessed: March 7, 2025.

[13] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y.S. Shao, K. Asanović, and B. Nikolić, “Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs,” IEEE Micro, vol.40, no.4, pp.10–21, 2020.

[14] Synopsys, Inc., “ASIP Designer,” https://www.synopsys.com/dw/ipdir.php?ds=asip-designer, 2024. Accessed: March 7, 2025.

[15] K. Hepola, J. Multanen, and P. Jääskeläinen, “OpenASIP 2.0: Co-Design Toolset for RISC-V Application-Specific Instruction-Set Processors,” 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp.161–165, 2022.

[16] S. Prakash, T. Callahan, J. Bushagour, C. Banbury, A.V. Green, P. Warden, T. Ansell, and V.J. Reddi, “CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (TinyML) Acceleration on FPGAs,” 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp.157–167, 2023.

[17] Fujieda, Naoki and Okuchi, Atsuki, “A Novel Remote FPGA Lab Platform Using MCU-based Controller Board,” 2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), pp.1–6, 2023.