RISC-V ベースのプロセッサを自動生成するシステムの開発
- FPGA 開発の新しいアプローチ “Logic as Software” -

1　背景

FPGA（**F**ield **P**rogrammable **G**ate **A**rray）とは，論理回路を集積したデバイスであり，開発者の要求に応じて自由に配線を変更することができるものである．開発者は，デバイスに搭載された論理回路の規模の制限内において自由に回路を構成することができる．実際のハードウェア上に回路を構成するため，高速動作を望むことができるのが特徴である．他にも，回路の構成にあたっては，HDL（Hardware Description Language）という一般的なプログラミング言語に似た構文をもつ言語を用いて行えることが特徴的である．このような点を活かし，FPGA は音声処理や画像処理などの分野において，特定の処理を高速に行いたいというニーズに応えてきた．

近年では，生成 AI などの発展により，FPGA の利活用についての関心がさらに高まりつつある．これは FPGA を用いた高速なデータ処理や，FPGA を用いた高速な機械学習などが期待されているためである．また，近年「自作 CPU」や「自作 OS」といったキーワードを含む書籍が多く発行されていたり，それに関連するイベントに多くの参加者がいたりすることから，FPGA を用いたプロセッサの開発にも関心が高まっている．生成 AI や自作 CPU といった分野での利用は企業での利用が多いと考えられがちであるが，各種 SNS を観察すると学生をはじめとする個人開発者からの関心が高いことが分かる．

しかし，FPGA を用いた開発には論理回路・ハードウェア開発に関連する知識・経験や，構成した回路を利用するために低レイヤと呼ばれる分野のソフトウェア知識が要求されることから，通常の開発者にとっては敷居の高いものとなってしまっている．加えて，開発した回路の検証が難しく，関連分野の知識・経験があったとしても扱いきれない事例が多い．

2　目的

本プロジェクトでは，学生など個人開発者をターゲットとして FPGA 開発に関する問題を解決しようとするアプローチを取る．FPGA 開発に関する問題や HDL による回路設計に関する問題を解決するためには，次にあげる事柄を達成する必要がある．現状では，これらを達成するための環境は，企業に属さない個人開発者にとって十分に整っているとは言えない．

- FPGA 開発に関する知識や経験を有している指導者の代替の実現
- FPGA 開発を学べる環境の実現
- HDL に関する深い知識を要求しない開発環境の実現

本プロジェクトでは図 1 に示すような環境を実現することを目指す．個人開発者をターゲットとして「選択」「合成」「実装」の 3 つのステップを経て，FPGA 上で動作する RISC-V プロセッサを生成することができる環境を提供する．「選択」はソフトウェア開発においてライブラリを選択するようにして，FPGA 上で動作する回路の構成を選択するというステップのことを指す．「合成」は選択された回路を合成するというステップのことを，「実装」は合成された結果として得られる回路を FPGA に実装するというステップのことを指す．この 3 ステップからなる環境を個人開発者に向けて提供することで，FPGA 開発に関する問題を解決し，FPGA 開発を利活用できる環境を実現することを目指す．

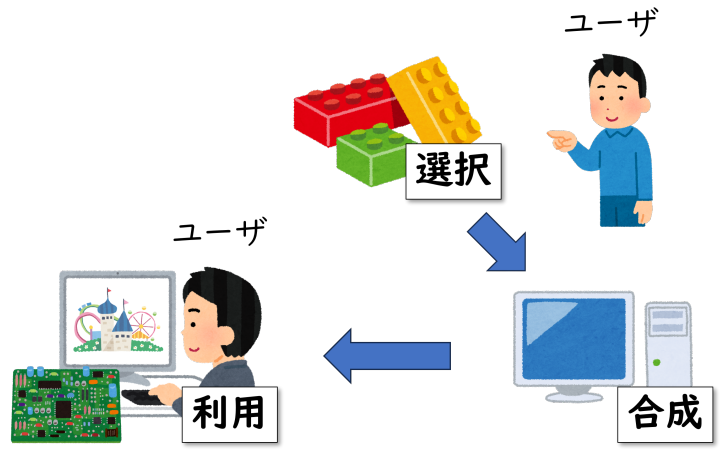

図 1: プロジェクトが実現すること

3　開発の内容

概要

本プロジェクトでは FPGA 開発に関する開発を容易に行えるようになることを目指し，「選択」「合成」「実装」の 3 つのステップを経て，FPGA 上で動作する RISC-V プロセッサを生成することができるシステム “Sasanqua（さ ざ ん か）” を開発した．具体的には，FPGA 上で動作するプロセッサの核となる「プロセッサコア」，プロセッサコアを拡張するための回路である「コプロセッサ」，プロセッサコアとコプロセッサを適切に合成するための「プロセッサ生成ツール」を開発した．Sasanqua システムに含まれる回路およびソフトウェアの関係性を表した図を図 2 に示す．Sasanqua システムを使用することによって，暗号化や数値計算といった処理に特化した命令やユーザオリジナルの回路を制御する命令を持つように，ユーザが自由にプロセッサを構成し生成することができる．

さらに，本プロジェクトでは独自に “Logic as Software” なる語を作成した．これは既存の語である IaaS（Infrastructure-as-a-Service）や PaaS（Platform-as-a-Service）を基に作成した語であり，回路をまるでソフトウェアのように扱うことができるという意味を持つ．ソフトウェアの特性である再利用性や拡張性，変更の容易さを回路にも持ち込むことを目指し，本プロジェクトでは，回路およびソフトウェアの開発を行った．

本プロジェクトの開発は，Digilent 社が開発および提供する ZYNQ Zybo-Z710 評価ボードを用いて行った．ZYNQ Zybo-Z710 評価ボードは，Xilinx 社の ZYNQ-7000 シリーズを搭載した評価ボードであり，FPGA と ARM プロセッサを 1 つのチップに搭載するものである．ZYNQ Zybo-Z710 評価ボードの外観を図 3 に示す．

プロセッサコア，コプロセッサ

プロセッサコアは ISA として RISC-V を採用し，基本命令として RV32I を実装した．RISC-V には様々な拡張命令が存在するが，本プロジェクトでは基本命令のみを実装した．プロセッサコアは，RV32I の基本命令を実装することで汎用的なプロセッサとしての機能の実装のみに注力させ，拡張命令は後述するコプロセッサで実装する．プロセッサコアの設計にあたっては，基本的な RISC プロセッサおよび既存実装として設計が公開されている RISC-V プロセッサの設計を参考にした．また，本プロジェクトの根底にある “Logic as Software” のコンセプトに基づき，プロセッサコアを構成する各部のモジュール化およびインタフェースの統一化を実施することで，プロセッサコアの設計においてもソフトウェアの特性を持たせることを意識した．プロセッサコアの実装は HDL の一種である Verilog HDL を用いて行った．

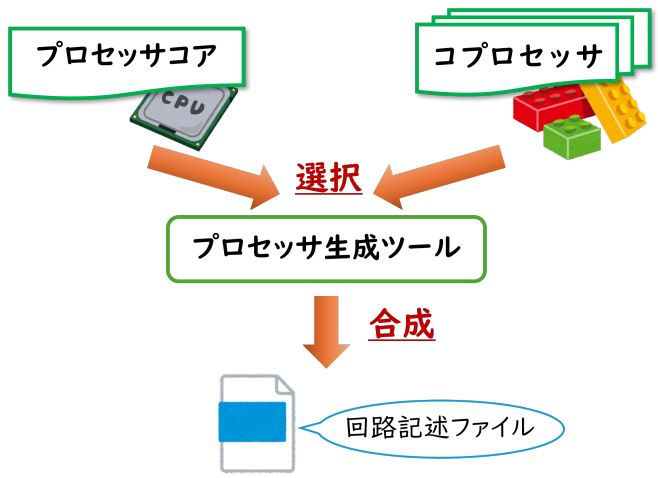

図 2: Sasanqua システム

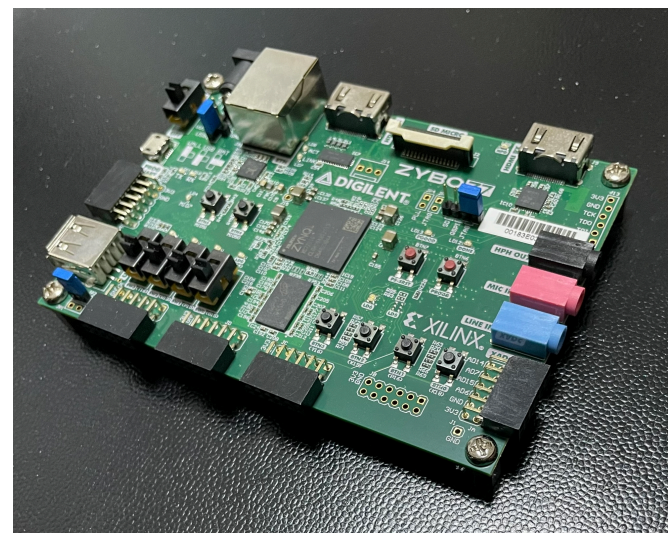

図 3: ZYNQ Zybo-Z710 評価ボード

コプロセッサは，プロセッサコアが持つ RV32I の基本命令に対して拡張命令を提供するためのモジュールである．コプロセッサはプロセッサコアと同様に，RISC-V の ISA に基づくものとし，この制限において自由に拡張命令を実装できるようにした．また，RISC-V は独自命令用にカスタム命令フィールドを持つため，このフィールドを使用して拡張命令を実装することができる．

コプロセッサの利用については「RISC-V 拡張としての実装」,「並列動作を前提とした実装」,「専用データの入力による実装」の 3 つのパターンが考えられる．RISC-V 拡張として実装する場合，プロセッサコアが持たない命令（RV32I に規定されない拡張命令）をコプロセッサ側に実装することで，プロセッサコアが持つ命令セットを拡張することができる．並列動作を前提として実装する場合，コプロセッサが持つ命令をある種の制御コマンドのような構成とすることで，コプロセッサが持つ機能であったりさらに追加で接続される回路を制御することができる．専用データの入力による実装をする場合，命令実行開始時にあるまとまったデータを渡すことによって，それに対してコプロセッサ側でデータの加工処理などを行うことができる．

プロセッサ生成ツール

プロセッサ生成ツールは，先に述べたプロセッサコアとコプロセッサを組み合わせて，FPGA 上で動作するプロセッサを生成するためのツールである．具体的には，プロセッサ生成ツールはプロセッサコアとコプロセッサの設定ファイルを入力とし，これに基づいてプロセッサコアとコプロセッサを組み合わせた回路を生成する．なお，プロセッサコアに対してコプロセッサを複数の接続を行えるように設計を行っている．ユーザは，自身の要求に応じてプロセッサ生成ツールに組み合わせを指示することによって，専用の構成を持つプロセッサを生成する．生成される回路は実際には Verilog で記述されたプログラムファイルの形で出力されるため，これを Vivado などの FPGA 開発ツールで読み込み，合成および配置配線を行うことで FPGA 上でプロセッサを動作させることができる．

プロセッサコアおよびコプロセッサは接続のための専用インタフェースを持つため，プロセッサ生成ツールはこれを適切に接続するように回路を生成する．このとき，プロセッサコアとコプロセッサを単にワイヤを用いて接続するだけではなく，間に「調停回路」という専用の回路を挟むことで，プロセッサコアとコプロセッサの間での独自の通信制御を行うようにする．調停回路はプロセッサコアからの信号を受け取り，これを適切にコプロセッサに伝え，またその逆の働きもする．例えば，プロセッサコアに複数のコプロセッサが接続されている場合，プロセッサコアから渡される命令を分配する必要があるほか，コプロセッサの応答を適切に選択してプロセッサコアに返す必要がある．これらの制御を調停回路が担当する．

プロセッサ生成ツールは Rust 言語（以下，Rust）を用いて実装した．プロセッサ生成ツールは，インタ

フェースとして Rust の内部 DSL を提供する．そのため，ユーザはプロセッサ生成ツールが提供するトレイトや構造体，関数を使用することによって，プロセッサコアおよびコプロセッサの組み合わせ指示を行う．内部 DSL として提供されることから，ユーザはエディタ上での補完や，型システムによる静的検証といった機能を活用しながら構成系を使用することができるという利点がある．実際に，プロセッサ生成ツールを使用するためのコード例を以下に示す（一部を「...」で省略している）．

```
#[derive(Debug)]
pub struct Rv32iMini;

impl CopProfile for Rv32iMini {
    fn opcodes(&self) -> Vec<(&'static str, OpCode)> {
        vec![("INST_ADD", OpCode::new(0b0110011, 0b000, 0b0000000)), ...]
    }

    fn body(&self) -> CopImpl {
        CopImplTemplate::from(&Rv32iMini)
            .set_ready(include_str!("ready.v"))
            .set_exec(include_str!("exec.v"))
    }
}
```

4　従来の技術との相違

ハードウェアレイヤの隠蔽

本プロジェクトでは，このアプローチおよび考え方のことを “Logic as Software” と銘打ち，これをプロジェクトのコンセプトとして掲げ，開発を実施した．このコンセプトは C や Java といった一般的に使用されるプログラミング言語に親しんできた開発者にとって易しく，FPGA 開発に対するハードルを下げることができる．このような既存言語ではモジュールやクラスといった概念が標準で存在し，開発者は自然とこのような概念を習得している．本プロジェクトでは，この習得度をそのまま FPGA 開発の文脈に持ち込むことができるように，プロセッサコアやプロセッサ生成ツールの設計を工夫している．

具体的には，FPGA の回路を記述するための HDL とそれを管理するための Rust プログラムを組み合わせるアプローチを採用した．このアプローチにより，FPGA 開発におけるハードウェアレイヤの隠蔽が実現される．従来の FPGA 開発では，ハードウェアの詳細に深く関与する必要があり，これが開発の複雑さと時間の増大を招いていた．しかし，本プロジェクトにおいて採用した方法では，Rust プログラムがハードウェアレイヤを抽象化するような回路を自動生成するため，開発者がより高いレベルでの設計と制御に集中できるようになった．これにより、開発プロセスが簡素化され効率化が図られる．

FPGA 開発に必要な工程の削減

これまで，FPGA 開発としていえば一から HDL を記述し，さらにそれを検証するためのテストベンチを HDL で記述し，そしてシミュレーションを実行し，という長い工程が必要であった．ここで触れた工程の先にも数多くのやらなければいけない工程が存在し，それらを通過してようやく実機での動作を確認できる．ただし，この実機での動作確認がすぐに完了するという事例は少なく，多くの場合は何度もさらに修正を繰り返す必要がある．

IP コアを使用することでこのような工程を短縮できる場合もあるが，それでもなお多くの工程が必要である．また，IP コアを使用する場合は，その IP コアが提供する機能に制限があるため，自由度が低いという問題もある．IP コアのカスタマイズのために公開されているパラメータ数が少なかったり，公開されているパラメータ数が多くてもそれらのパラメータが自分の求めるものと一致しなかったりと，IP コアを使用することで得られる利点が少ない場合もある．

本プロジェクトの成果を使用することで，これらの問題を解決できると考えている．本プロジェクトで開発したプロセッサ生成ツールを使用することで，ユーザは自分の求めるプロセッサを自由に設計できる．IP コアの利用とは異なり，回路の部分を自由に変更できるため，自由度が高い．このツールで生成できるプロセッサの多くは自動生成されるため，ユーザに対して要求される工程は少ない．しかも，自動生成されるプロセッサの多くは検証済みであることから，不具合の箇所を見つけるための工程も少なくて済む．

5　期待される効果

本プロジェクトでは “Logic as Software” というコンセプトを新規に発案した．このコンセプトに基づく HDL と Rust プログラムの組み合わせによる FPGA 開発におけるハードウェアレイヤの隠蔽というアプローチは，新しいものである．本プロジェクトで採用したこのアプローチは，FPGA 開発の効率化と簡素化に貢献するものであり，今後の FPGA 開発における一手法として提案したい．

本プロジェクトで開発したプロセッサは，RISC-V を ISA として採用した．RISC-V はこれまでに定義されていた数多くの ISA が抱えていた課題が解決されると言われ，また ISA 設計がオープンソースかつロイヤリティフリーとなっていることから，近年注目を集めている．しかし，個人開発者が RISC-V を採用したプロダクトを開発するには，参考となる資料や実装例が少なく，高い壁が立ちはだかっている状況である．本プロジェクトの成果によって，誰もが自由に，かつ容易に，RISC-V を ISA として採用したプロセッサを利用，あるいは組み込んだプロダクトを開発できる未来が実現されることを期待する．

6　普及の見通し

本プロジェクトの成果は全て GitHub 上で公開しているため，Sasanqua は誰でも自由に利用することができる．今後は成果を利活用するためのドキュメントを準備し，併せて，関連コミュニティにも積極的に情報を発信していく．多くの人に認知されることで，本プロジェクトの成果が広く利用されることを期待する．

現在，本プロジェクトの成果である Sasanqua をより直感的に扱うための「ツールキット」に開発に着手している．ツールキットの開発によって，プロセッサコアやコプロセッサの組み合わせ指示，プロセッサ生成ツールの実行，生成された回路の合成および配置配線，FPGA への書き込みといった一連の作業を GUI で提供することを目指している．他にも多くの機能を検討および実装することでより扱いやすいツールキットを探求し，多くの人に使用されるプロダクトとして成長させていく．

7　クリエータ名（所属）

- 中神 悠太（筑波大学情報学群情報メディア創成学類）

（参考）関連 URL

- Sasanqua：`https://github.com/SasanquaProject`