修士論文の和文要旨

研究科	・専攻	大学院 情報システム学研究科ネットワー	ークシステム	学専攻 博士前期課程
氏	名	佐藤 遼	学籍番号	1252018
論 文	題目	カスタムプロセッサ構築用 FPGA プラッ	トフォームの	開発と評価

要 旨

プロセッサの性能向上は今なお求められており,更なる処理性能向上にはプロセッサアーキテ クチャの改善が必要である.プロセッサアーキテクチャの研究ではアイディアの検証はソフトウ ェアシミュレーションによって行われることが多い.しかしながら,回路規模の増大や処理の複 雑化によって評価にかかる時間が増大するという問題が顕在化している.

そこで注目すべき解決策が, Field Programmable Gate Array(FPGA) を用いたエミュレーション方法である. FPGA はハードウェアチップを設計するより,簡単に所望の回路を実現することができ,かつハードウェアの動作をソフトウェアよりも高速に模倣させることができる. また,何度でも内容を変更・修正できるため,動作確認のテストを容易に行うことができる. このため FPGA を用いたアーキテクチャ研究の高速化は有用な手段の一つと考えられる.

そこで本論文では ARM の ISA をベースとするカスタムプロセッサ構築用 FPGA プラットフォ ームを独自に実現することを最終的な目的として、プロセッサを独自に実装し、動作検証と評価 を行う.

本稿ではカスタムプロセッサ構築用FPGA プラットフォームの実現を最終的な目的として FPGAプラットフォームの実装と評価を行った.第一に,設計したプロセッサが想定通りに実装 できていることを検証した.第二に,様々なプログラムに対してプロセッサが動作可能であるこ とを確認した.第三に,シリアル通信でプロセッサのメモリにアクセスする機構を追加実装した. 第四に,PCIe 通信でプロセッサの性能評価指数を出力する機構を追加実装した.最後に,シミ ュレータとFPGAで,プロセッサの検証にかかる実行時間を比較した.

これらの検証結果からFPGA による拡張性,高速動作性を確認し,カスタムプロセッサ構築用 FPGAプラットフォームとして有用であることを確認した.加えて今後の研究の方向性として提 案システムの改善点をまとめた.

平成25年度修士論文

カスタムプロセッサ構築用 FPGA プラットフォームの開発と評価

大学院情報システム学研究科 情報ネットワークシステム学専攻

学籍番号: 1252018
氏名: 佐藤遼
主任指導教員: 吉永努教授
指導教員: 入江英嗣准教授
指導教員: 大坐畠智准教授
提出年月日: 平成26年1月27日

(表紙裏)

目 次

第1章	序論	1
第2章	関連研究	3
2.1	異種命令混合実行プロセッサ OROCHI	3
2.2	CoreSymphony	3
2.3	関連製品	4
2.4	その他の関連研究	5
第3章	提案システムの概要と実装手順	6
3.1	概要	6
3.2	実装手順	7
	3.2.1 事前学習と準備	7
	3.2.2 カスタムプロセッサの実装と検証	8
	3.2.3 シリアル通信によるメモリアクセスの実装と検証	9
	3.2.4 PCIe 通信による性能評価指数出力の実装と検証	10
第4章	実装したプラットフォームの検証と評価	11
4.1	様々なプログラムに対するカスタムプロセッサの動作検証	11
4.2	シリアル通信のメモリアクセス検証..............................	27
4.3	PCIe 通信の性能評価指数出力検証.................................	31
4.4	性能評価	34
第5章	結論	35
5.1	本研究のまとめ	35
5.2	今後の課題	36
参考文南	伏	38
付録		40

図目次

3.1.1 提案システムの構成図	6
3.2.1 カスタムプロセッサブロック図	8
3.2.2 シリアル通信ブロック図/Buf 有り	9
3.2.3 シリアル通信ブロック 図/Buf 無し	9
3.2.4 PCIe 通信ブロック図	10
4.1.1 Hillo シミュレーション結果	11
4.1.2 Hillo 全体波形図	12
4.1.3 Hillo 拡大波形図	12
4.1.4 Gusu シミュレーション結果	13
4.1.5 Gusu 全体波形図	14
4.1.6 Gusu 拡大波形図	14
4.1.7 Foradd シミュレーション結果	15
4.1.8 Foradd 全体波形図	16
4.1.9 Foradd 拡大波形図	16
4.1.10 Fibonacci シミュレーション結果	17
4.1.11 Fibonacci 全体波形図	18
4.1.12 Fibonacci 拡大波形図	18
4.1.13 FizzBuzz シミュレーション結果	19
4.1.14 FizzBuzz 全体波形図	20
4.1.15 FizzBuzz 拡大波形図	20
4.1.16 Sort シミュレーション結果	21
4.1.17 Sort 全体波形図	22
4.1.18 Sort 拡大波形図	22
4.1.19 Sosu シミュレーション結果	23
4.1.20 Sosu 全体波形図	24
4.1.21 Sosu 拡大波形図	24
4.1.22 Himeno シミュレーション結果	25
4.1.23 Himeno 全体波形図	26
4.1.24 Himeno 拡大波形図	26
4.2.1 Hillo メモリデータ入出力	27
4.2.2 Gusu メモリデータ入出力	27
4.2.3 Foradd メモリデータ入出力	28
4.2.4 Fibonacci メモリデータ入出力	28
4.2.5 FizzBuzz メモリデータ入出力	29
· _ · · · · · · · · · · · · · · · · · ·	-

4.2.6	Sort メモリデータ入出力	29
4.2.7	Sosu メモリデータ入出力	30
4.2.8	Himeno メモリデータ入出力	30
4.3.1	PCIe 出力/Hillo	32
4.3.2	PCIe 出力/Gusu	32
4.3.3	PCIe 出力/Foradd	32
4.3.4	PCIe 出力/Fibonacci	32
4.3.5	PCIe 出力/FizzBuzz	33
4.3.6	PCIe 出力/Sort	33
4.3.7	PCIe 出力/Sosu	33
4.3.8	PCIe 出力/Himeno	33

表目次

3.1	シミュレーション環境....................................	7
4.1	PCIe 通信出力詳細	1
4.2	動作周波数と使用リソース量 3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3-3	4
4.3	実行時間比較	4

ソースコード

4.1	Hillo	12
4.2	Gusu	13
4.3	Foradd	15
4.4	Fibonacci	17
4.5	FizzBuzz	19
4.6	Sort	21
4.7	Sosu	23
5.1	Himeno	40

第1章 序論

プロセッサの性能向上は今なお求められている.プロセッサの性能を向上させる方法には周波 数の向上,アーキテクチャの改善およびマルチコア化がある.しかし熱や消費電力,配線遅延の 相対的な増加によって,周波数の向上は限界に達してきている.またマルチコア化は,複数のタ スクを同時に処理する事でシステム全体の処理性能を向上させる手法であり,シングルスレッド の処理性能を向上させるわけではない.そのためプロセッサの更なる処理性能向上にはプロセッ サアーキテクチャの改善が必要である.特に,従来研究されてきたプロセッサ単体の計算処理性 能を向上させる研究に加え,プリフェッチやキャッシュなどのプロセッサシステム全体を考慮した アーキテクチャの研究が重要になってきている.

プロセッサアーキテクチャの研究ではアイディアを検証し性能を評価する必要がある.時間的 および金銭的コストの問題で,ソフトウェアシミュレーションによる動作検証及び性能評価が行 われることが多い.しかしながら,プリフェッチやキャッシュの研究ではアプリケーションレベル での評価が必要になるため,従来のアーキテクチャ単体のシミュレーションに比べて,評価にか かる時間が増大するという問題が顕在化している.

ソフトウェアによるシミュレーションに時間がかかるとはいえ,実際に研究のアイディアを実 ハードウェアで作って検証・評価することは,金銭的にも開発コスト的にも現実的ではない.そこ で注目すべき解決策が,Field Programmable Gate Array(FPGA)を用いたエミュレーション方法で ある.FPGA はアプリケーションに合わせた実ハードウェアをユーザが設計できるデバイスであ リ,実際にハードウェアチップを設計するよりは,簡単に所望の回路を実現することができ,か つハードウェアそのものであることから,ハードウェアの動作をソフトウェアよりも高速に模倣 させることができる.また,何度でも内容を変更・修正できるため,動作確認のテストを容易に 行うことができる.このためFPGA を用いたアーキテクチャ研究の高速化は有用な手段の一つと 考えられる.

アーキテクチャの研究では Instruction Set Architecture(ISA)を決める方がやりやすい.パソコン では x86,携帯機器では ARM,研究教育用では MIPS がよく用いられる.もちろん,オリジナル の ISA をベースにアーキテクチャの研究を行うという選択も考えられるが,今回は,ARM を対象 にしたアーキテクチャの研究を想定する.その理由は,ARM は小型かつ消費電力あたりの処理性 能が高いプロセッサであることから,携帯電話をはじめとして最近特に注目を集めているからで ある.すなわち,ARM の ISA をベースとすることで,新しい研究上のアイディアを実用的な環境 で評価しやすいというメリットがあると考えられる.しかし,ARM は IP コアで商売を行ってい るため,ASIC 開発や FPGA 上に構成可能な RTL 記述が公開されていない.そのため,FPGA を 用いて ARM をベースとしたアーキテクチャ研究を行うことができない.プロセッサアーキテク チャの研究で,あるアイディアを検証しようとしても,当然,ベースとなる ISA に沿った振る舞 いをするプロセッサ部分のエミュレーションができなければ,その評価はできない.

そのため本研究ではARMのISAをベースとするカスタムプロセッサ構築用 FPGA プラットフォームを独自に実現することを最終的な目的として,カスタムプロセッサを独自に実装し,動作検証

と評価を行う.

本論文の構成は次の通りである.まず,第二章で関連研究について述べる.次に,第三章では 実装の対象であるプロセッサアーキテクチャについて実装までの手順を説明し,第四章で,実装 した回路の動作検証の結果を,また,ソフトウェアシミュレータを使ってシミュレーションをす るのにかかる時間と FPGA 上でエミュレーションするのにかかる時間を比較し, FPGA による高 速化の有効性を示す.最後に本論文の成果をまとめ,今後の課題を述べる.

第2章 関連研究

プロセッサの性能を向上させる方法のひとつとしてマルチスレッド・マルチコア化がある.関連 研究 OROCHI は単一コアで複数命令セットを実行可能なプロセッサを,関連研究 CoreSymphony は複数のコアを協調動作させ,複数のキャッシュや演算器を利用し,逐次処理の高速化を実装して いる.以下に各関連研究の詳細を示す.

2.1 異種命令混合実行プロセッサ OROCHI

異なる命令セットアーキテクチャのプロセッサを1チップに混載した商用マルチコア型プロセッ サがあり,今後複数の命令セットを同時に実行できるプロセッサが徐々に一般化していくと考えら れる.しかし複数種類のプロセッサコアを並置する方式を用いると,全体の回路規模が大きくな り,コア間通信のためにある程度複雑な調停機構が必要になる.そのため単一コアで同時実行する マルチスレッディングプロセッサ OROCHIを設計する.ここで OROCHI は奈良先端科学技術大学 院大学が開発した異なる命令セットを同時に実行できるよう,一般的な SMT プロセッサを拡張し たプロセッサモデルのことを指す,OROCHI は汎用命令セットにより記述されたソフトウェア資 産をスーパスカラ方式により高速実行する部分と,マルチメディア処理用の命令セットに特化し, コンパイラ等によりスケジューリングが完了している命令列を VLIW 方式により効率よく実行す る部分から構成される.そしてシミュレータの観点からソフトウェアシミュレーションと RTL ソー スの記述量とシミュレーション実行速度について分析し,実用性の観点から FPGA と ASIC それ ぞれを対象とした論合成の結果から遅延時間や回路規模等についての評価している[1][2][3][4][5].

2.2 CoreSymphony

近年,1チップに複数のコアを集積する CMP が周流のアーキテクチャとなっている.CMP はスレッドレベル並列性を利用し,複数のスレッドを複数のコアで並列実行することで性能向上を得る.しかし並列プログラム中の逐次処理をなくすことは難しいため,CMP においても逐次処理の高速化が重要な課題となっている.そこで CoreSymphony という技術が提案されている.CoreSymphony は発行幅の狭いプロセッサコアをいくつか協調動作させることで,より発行幅の大きな仮想コアを形成する手法である.そのため,複数コアのキャッシュや演算器を利用することで,逐次処理を高速実行可能にしている.そして性能評価はシミュレータを利用し行われているが,ハードウェアリソース量の面から見ても標準的なアウトオブオーダ実行プロセッサの2倍程度のハードウェア量で実装可能であることを検証している[6][7][8][9][10].

2.3 関連製品

年々プロセッサに要求される回路規模は増大し,処理が複雑化している.そのためハードウェ アとソフトウェアの開発をより密接に連携させる必要である.そこでハードウェア/ソフトウェア 協調検証を行えるシステムが多く製品化されてきた.関連製品 Palladium XP もその内のひとつで 4 つのプラットフォームを実装しつつ,各プラットフォームはデータベースやフローにおいて互換 性を持っている.そのためプラットフォーム間でデータの再コンパイルなどが不必要となってい る.以下に各関連製品の詳細を示す

• Cadence Palladium XP

ケイデンスの最先端名ハードウェア,ソフトウェアで実現した論理検証用コンピューティン グ・プラットフォームである.プロセッサ・ベースの計算エンジンとUXE(Unified Xccelerator Emulator)ソフトウェアにより,高速かつ柔軟な拡張性を実現するとともに従来のエミュレー ションでは困難だった多様な用途に応えることができる[11]

• Showa&Sophia Technologies DS-5

ARM プロセッサー搭載プラットフォーム向けの ARM 者推奨次世代開発ツールである.業界 標準ともいえる Eclipse ベースの GUI で直感的な操作が可能となっている.サードパーティ 性プラグ印との親和性も高く, Android SDK の ADT プラグインを追加インストールするこ とで java と C/C++のシームレスな開発が可能となる.各開発担当が同じ操作系を共有でき るため交流も円滑に進む.[12]

• ALDEC HES-DVM

HES-DVM は完全自動化機能とスクリプト環境を備えた SoC/ASIC デザインのハイブリッド バリデーション・検証環境である.ハードウェとソフトウェアの協調検証を可能にしてい て,協調エミュレーションを活用することでハードウェアとソフトウェアの設計者は最新の FPGA テクノロジが利用可能となり,互いに同時並行で開発しながら RTL で開発・検証でき る.[13]

2.4 その他の関連研究

プロセッサの性能向上が求められているため,性能検証の高速化は依然として求められている. 性能検証の方法としてシミュレーション検証とエミュレーション検証がある.基本的にエミュレー ション検証の方が高速であることがわかっている.そのためソフトウェアシミュレーションによ る検証を高速化する様々な手法が提案されている.以下に各種関連研究の詳細を示す.

• ARM アーキテクチャ用仮想マシンモニタの実装

今日,組み込み機器は広く普及し重要な要素となったいるが,その組み込み機器に要求される機能が非常に高度なものとなっている.また企業への新製品開発サイクル短縮への要求も同様に増加しており,これが原因となって組み込み機器製品の不具合や脆弱なセキュリティが問題となりかねない.それらを解決する手段として仮想マシンモニタ(VMM)の導入が考えられる.VMMはハードウェア上に複数の仮想マシンを構築できるため,セキュリティレベルによって個別の仮想マシンとOSを提供した場合,組み込み機器の安定性向上と強固なセキュリティの実現が可能となる.VMMは主にIA-32アーキテクチャ専用だが,将来的な組み込み機器への利用を考え,ARMアーキテクチャ用のVMMを開発する[14].

• 組込みシステム向けマルチコア・プロセッサのためのソフトウェア開発支援

高機能アプリケーションや汎用 OS を利用可能とするために,高性能・省電力型プロセッサ への需要が増大している.これに対して組込み機器向けプロセッサ・ベンダはマルチコア・ プロセッサの開発を進めている.しかし,プロセッサのソフトウェア開発は煩雑なものとな りがちである.そこで携帯端末で広く使われている OMAP プロセッサを例に,ソフトウェ ア開発の問題点を明らかにし,解決し,これまで面倒だったマルチコアプロセッサを用いた システム開発を可能とするランタイム環境を提案する[15].

第3章 提案システムの概要と実装手順

3.1 概要

前章で述べられている OROCHI や CoreSymphony は異種命令混合実行単一プロセッサであった リマルチコアプロセッサであるため, ARM の ISA ベースの単一コア・プロセッサの評価を得るこ とはできない.またハードウェア/ソフトウェア協調開発システムや OROCHI は独自のシミュレー タもしくは Xilinx の ISE Design Suite(ISE) などの波形シミュレータを用いて性能評価を出力して いる.そこで FPGA 上で動作可能な ARM の ISA ベースの単一コア・プロセッサ (カスタムプロ セッサ)とカスタムプロセッサのデータをやり取りする HostPC と FPGA 間のデータ通信機構を持 つ FPGA プラットフォームを開発することでこれらの問題を解決できる.そのためカスタムプロ セッサを実装しシリアル通信を用いたメモリアクセスや PCIe 通信による性能評価指数出力を追加 実装することで, FPGA の拡張性を持ち実機に近い環境での動作検証と性能評価を得ることがで きるプラットフォームを開発する.以下の図 3.1.1 にシステム全体の構成図を示す.



図 3.1.1: 提案システムの構成図

3.2 実装手順

プロセッサを実装するには,アーキテクチャの作成が必要不可欠である.しかし最初から全てを 作るには命令セットの選択,デコーダや演算器などの機構,キャッシュやプリフェッチなどの理論 と様々な事柄を理解し設計する必要がある.そのため多大な時間がかかってしまう.そこで,奈良 先端科学技術大学院大学より異種命令セット同時実行プロセッサ OROCHI と OROCHI シミュレー タ OSIM を参考にカスタムプロセッサを実装する.その後,シリアル通信,PCIe 通信を追加実装 する.以降にカスタムプロセッサや各種通信機能を実装する際に行った手順を示す.

3.2.1 事前学習と準備

OROCHI シミュレータ OSIM の実行

事前準備として初めに OROCHI プロセッサの動作や出力結果等を調べるため,最終的に FPGAの エミュレーションと比較するため, OROCHI シミュレータ OSIM の動作を確認する.そこで OSIM の実行環境として以下の表 3.1 の Linux 環境を用意した.そしてソースコードと幾つかのサンプル プログラムの実行結果から OSIM の使用方法と構成を理解した.

表 3.1: シミュレーション環境

オペレーティングシステム (OS)	Windows XP Professional Version 2002 Service Pack2
CPU	Intel(R) Core(TM)2 Quad CPU Q9450 @2.66GHz
Memory	3.25GB RAM
仮想マシン	VMware Player Version 4.0.4.30409
仮想 OS	FreeBSD6.2R

OROCHI プロセッサの解析

次に, Verilog ソースの OROCHI プロセッサを解析・実行することで OROCHI の構成を理解し, カスタムプロセッサ構築のための知識を得た [4][5][16].ここで, Verilog ソース解析から定義を書 き換えることで様々な構成に変更可能であることやテストベンチを除くと主記憶,キャッシュやプ ロセッサ制御が FPGA 外にある構成で,論理合成には各機構を新しく追加する必要があることが 確認できた.また実際に ISE でサンプルプログラムを実行し,構成の違いにより動作や出力結果 にどのような差異が発生するかを確認した.

3.2.2 カスタムプロセッサの実装と検証

OROCHIの知識を元にカスタムプロセッサを実装し,ISE14.7による動作検証を行った[17][18][19]. OROCHIの構成を変更するため定義の書き換えを行い単一アーキテクチャプロセッサへと分離し, OROCHIと同様のテストベンチの元でサンプルプログラムを実行した.その後,既存モジュール の書き換えやメモリ/プロセッサコントローラーや新規モジュールの追加を行い,論理合成可能な カスタムプロセッサを実装した.そしてプロセッサへの入力をクロック/リセットのみとしてテス トベンチ新たに作成し,サンプルプログラムが同様に実行可能であることを確認した.

以下の図 3.2.1 に実装したカスタムプロセッサのブロック図を示す.



図 3.2.1: カスタムプロセッサブロック図

各種モジュールは以下の役割を行う. PC はプログラムカウンタを, IF は命令読み込みを, L2 は L2 キャッシュのアクセス制御を.decoder, map は命令解析と物理レジスタのマッピングを, read reg は各種演算器の利用状況管理と命令発行を, alu, sfm, eag, me1 は算術論理演算を, Load/Store と brc はロード・ストア命令と分岐命令を MemoryController と ProcessorController はメモリとプ ロセッサシステム全体の制御を担当する.

3.2.3 シリアル通信によるメモリアクセスの実装と検証

次に,カスタムプロセッサにシリアル通信機構を実装し,メモリに対してシリアル通信でデー タのやり取りを可能にした[20][21].その後,FPGA 評価ボード(ML605)上でカスタムプロセッサ とシリアル通信が正しく実機動作可能であることを確認した.前述のカスタムプロセッサは ISE 上でのシミュレーションによる動作確認は可能であるが,FPGA に実装したとしても外部と通信す ることができないため,プログラムデータを入力することも実行結果を出力することもできない. そこでデータ通信機能を追加実装する必要がある.しかしいきなりシリアル通信をを直接繋いで しまうと,回路のどの箇所で問題が発生しているか判別することができない.問題箇所を確認す るためシリアル通信機構とカスタムプロセッサの間にバッファを接続し動作確認を行った.不必 要なバッファとスイッチを排除し軽量化と手順の簡略化を行い,ML605へ実装を行った.その結 果,シリアル通信でメモリに直接プログラムを入力,実行後のメモリを出力として通信可能なこ とを確認した.なお,シリアル通信のデータ入力や出力は Cygwin 上で PySerial を使用した.以下 の図 3.2.2 にバッファ有りのシリアル通信のブロック図を,図 3.2.3 に最終的に実装したバッファ 無しのシリアル通信のブロック図を示す.



図 3.2.2: シリアル通信ブロック図/Buf 有り



図 3.2.3: シリアル通信ブロック図/Buf 無し

3.2.4 PCIe 通信による性能評価指数出力の実装と検証

最後に,より高速な通信方法として PCIe 通信を実装した.前述のシリアル通信では直接メモリ の読み書きを行えるため,メモリに記入されている事柄については確認することができるが,記 入されていないことは確認できない.またシリアル通信でデータのやり取りを行うことは実行速 度の面から見て,非常に遅く合理的ではない.そのためより高速な通信方法として PCIe が考えら れた.そこで PCIe の IPcore を用いてプロセッサのサイクル数やキャッシュのヒット/ミス数などの 性能評価指数を高速かつプロセッサの状態によらず,確認可能に設計し ML605 に実装した.その 後,正しく結果が出力されていることを確認した.以下の図 3.2.4 に PCIe 通信のブロック図を示 す図の Custom Processor 部分が今回実装した書き換え可能なカスタムプロセッサで, PCIe 通信の GPIO Port へ性能評価指数を出力している.



図 3.2.4: PCIe 通信ブロック図

第4章 実装したプラットフォームの検証と評価

4.1 様々なプログラムに対するカスタムプロセッサの動作検証

ISE14.7を用いて,カスタムプロセッサ上で様々なプログラムをシミュレーション実行することで,各プログラムが実装したカスタムプロセッサで実行可能であることを確認する.またシミュレーション結果や出力される波形図からカスタムプロセッサが設計した通りに動作していることを検証する.以降に ISE の出力結果と各プログラムのソースコードを示す.

Hillo

Hillo は Hello を出力後, e を i に書き換えて出力するプログラムである. Hillo の ISE シミュレー ション結果を図 4.1.1 に, ソースコードをソースコード 4.1 に全体波形図を図 4.1.2 に, 一部拡大 波形図を図 4.1.3 に示す.

まず図 4.1.1 を見ると,初めに Hello が出力され,その後 Hillo が出力されている.この結果と ソースコード 4.1 から正しくプログラムが動作していることがわかる.次に図 4.1.2 を見ると,3 列目から順番に [242c,24e,14,3e,7,b,6] と出力されている.これはプログラム実行後に出力 される性能評価指数を表していて,図 4.1.1 の最後に同様の値が出力されている.最後に図 4.1.3 を見ると,最下段の信号 char の赤枠内に Hillo が出力されている.これは出力結果を格納してい る信号であり,他の値も同様に出力されている.これらの結果からカスタムプロセッサ上で Hillo が実行され,回路が想定通り動作していることがわかる.

> Hello Syscall : Puts Hillo Syscall : Exit :0x000000000000242c cycle_counter run_cycle_counter(cycle - swi) :0x000000000000024e arm_terminal_instruction_counter :0x000000000000014 armcache hit :0x0000003e armcache miss :0x00000007 op1cache_hit :0x0000000b :0×00000006 op1cache_miss

Syscall : Puts

図 4.1.1: Hillo シミュレーション結果

- 1 char a[1024] = "Hello\n";;
 2 main(){
 3 int i;
 4 for (i=1; i<=1; i++){
 5 puts(a);
 6 a[1] = 'i';
 7 puts(a);</pre>
- 8 }}

										1,858,46	30000 us
Name	Value	Pus	00 us	400 us	600 us	800 us	1.000 us	1.200 us	1.400 us	1,600 us	1.800 u
🐻 clk	1										
16 rst	0										
data0[63:0]	00000000000242c										
data1[63:0]	000000000000024e		0000000	0000000000		00000000	0000001 🕫 💥 000000000	0000203	0000000000	0024e	
data2[63:0]	000000000000014		00000	00000000000		χχχχ 00000000	оссооссь 💥 сооссоо	0000012	0000000000	00014	
data3[31:0]	0000003e		00	000000		X()_X()_X(0000 0000	025	000000	3e	
data4[31:0]	00000007		000	000000		00000	004 00000	05	0000000	9	
data5[31:0]	000000b			0000000		XXXX 000	00008		00000006		
data6[31:0]	00000006		00	000000			04)))		0000006		
lle rst_pro	0	1									
1 ser_on	0										
Ug pro_end	1										
char[7:0]	-						Х.	X			

図 4.1.2: Hillo 全体波形図

Name	Value		1,269,500 ns	1,270,000 ns	1,270,500 ns	1,271,000 ns	1,271,500 ns	1,272,000
16 clk	1 0							
 data0[63:0] data1[63:0] 	00000000000001859	X		<u>, X000 X000 X</u>		<u>, 2000 2000 2</u> 0	000000000000000000000000000000000000000	03
 data2[63:0] data3[31:0] 	00000000000000000000000000000000000000							0000002b
 data4[31:0] data5[31:0] 	00000005 0000000b							00
data6[31:0] 1 rst_pro	0000006							00
Lig ser_on Lig pro_end	0		_					
char[7:0]	-	<u> </u>	· (н)(і		<u>'</u> Х	•)_	-)

図 4.1.3: Hillo 拡大波形図

Gusu

Gusu は 100 までの偶数を出力するプログラムである. Gusu の ISE シミュレーション結果を図 4.1.4 に, ソースコードをソースコード 4.2 に全体波形図を図 4.1.5 に, 一部拡大波形図を図 4.1.6 に示す.

まず図 4.1.4を見ると、2,4,6,…98,100と出力されている.この結果とソースコード 4.2 から 正しくプログラムが動作していることがわかる.次に図 4.1.5を見ると、3 列目から順番に [215e4, 133f6,34c9,3f60,b0,13c4,27e]と出力されている.これはプログラム実行後に出力される性 能評価指数を表していて、図 4.1.4の最後に同様の値が出力されている.最後に図 4.1.6を見ると、 最下段の信号 char の赤枠内に 98 が出力されている.これは出力結果を格納している信号であり、 他の値も同様に出力されている.これらの結果からカスタムプロセッサ上で Gusu が実行され、回 路が想定通り動作していることがわかる.

Syscall : Write	Syscall : Write	Syscall : Write	Syscall : Write	
2	30	58	86	
Syscall : Write	Syscall : Write	Syscall : Write	Syscall : Write	
4	32	60	88	
Syscall : Write	Syscall : Write	Syscall : Write	Syscall : Write	
6	34	62	90	
Syscall : Write	Syscall : Write	Syscall : Write	Syscall : Write	
8	36	64	92	
Syscall : Write	Syscall : Write	Syscall : Write	Syscall : Write	
10	38	66	94	
Syscall : Write	Syscall : Write 40	Syscall : Write 68	Syscall : Write 96	
Syscall : Write	Syscall : Write 42	Syscall : Write 70	Syscall : Write 98	
Syscall : Write	Syscall : Write 44	Syscall : Write 72	Syscall : Write 100	
Syscall : Write	Syscall : Write 46	Syscall : Write 74	Syscall : Exit cycle_counter	:0×00000000000215e4
Syscall : Write	Syscall : Write	Syscall : Write	run_cycle_counter(cycle - swi)	:0x00000000000133f6
	48	76	arm_terminal_instruction_counter	:0x0000000000034c9
Syscall : Write	Syscall : Write	Syscall : Write	armcache_hit	:0×00003f60
	50	78	armcache_miss	:0×000000b0
Syscall : Write	Syscall : Write	Syscall : Write	op1cache_hit	:0x000013c4
	52	80	op1cache_miss	:0x0000027e
Syscall : Write	Syscall : Write 54	Syscall : Write 82		
Syscall : Write 28	Syscall : Write 56	Syscall : Write 84		

図 4.1.4: Gusu シミュレーション結果

ソースコード 4.2: Gusu

```
1 #include <stdio.h>
2 main() {
3 int i;
4 for (i=1;i<=100;i++){
5 if(i%2==0) printf("%d_",i);
6 }}</pre>
```

							27.341.660000 ms
Name	Value	0 ms	5 ms	10 ms	15 ms	20 ms	25 ms
🐻 clk	1						
🔓 rst	0						
data0[63:0]	00000000000215e4						
data1[63:0]	0000000000133f6		billomomomomomomomomo				000000000000000000000000000000000000000
data2[63:0]	0000000000034c9		billomomomomomomomomo				000000000000000000000000000000000000000
data3[31:0]	00003f60		billomomore cilication como en como en Como en como en Como en como en c				000000000000000000000000000000000000000
data4[31:0]	00000060						···· X··· X··· X00····
data5[31:0]	000013c4						000000000000000000000000000000000000000
data6[31:0]	0000027e	00					
1@ rst_pro	0						
1@ ser_on	0						
14 pro_end	1						
char[7:0]							

図 4.1.5: Gusu 全体波形図

Name	Value	26,213 us 26,214 us 26,215 us
<pre> [] clk [] rst [] data0[63:0] [] data1[63:0] [] data2[63:0] [] data3[31:0] </pre>	1 0 00000000001fffc 000000000012d48 0000000000033a0 00003dd3	
adata4[31:0] adata5[31:0] adata6[31:0] ata6[s1:0] ser_pro ser_on pro_end	000000ae 00001358 0000026d 0 0 0	00001355
char[7:0]		

図 4.1.6: Gusu 拡大波形図

Foradd

Foradd は 1 ~ 99 までの数を合計値を出力するプログラムである. Foradd の ISE シミュレーションした結果を図 4.1.7 に,ソースコードをソースコード 4.3 に全体波形図を図 4.1.8 に,一部拡大 波形図を図 4.1.9 に示す.

まず図 4.1.7 を見ると,4950 と出力されている.この結果とソースコード 4.3 から正しくプロ グラムが動作していることがわかる.次に図 4.1.8 を見ると,3列目から順番に [2d1a,dc2,11b, e4,36,61,e] と出力されている.これはプログラム実行後に出力される性能評価指数を表して いて,図 4.1.7 の最後に同様の値が出力されている.最後に図 4.1.9 を見ると,最下段の信号 char の赤枠内に 495...が出力されている.これは出力結果を格納している信号であり,他の値も同様に 出力されている.これらの結果からカスタムプロセッサ上で Foradd が実行され,回路が正常に動 作していることがわかる.

Syscall : Write	
4950	
Syscall : Exit	
cycle_counter	:0x000000000002d1a
run_cycle_counter(cycle - swi)	:0x00000000000000dc2
arm_terminal_instruction_counter	:0x00000000000011b
armcache_hit	:0x000000e4
armcache_miss	:0×00000036
op Icache_hit	:0x00000061
op1cache_miss	:0x0000000e

図 4.1.7: Foradd シミュレーション結果

ソースコード 4.3: Foradd

- 1 #include <stdio.h>
- 2 #include <stdlib.h>
- 3 int main(void){
- 4 **int** i;
- 5 **int** kekka=0;
- 6 **for** (i=0; i<100; i++){
- 7 kekka = kekka + i;
- 8 printf("%d",kekka);
- 9 **return** 0;}

						2,315,66000 us
Name	Value	0 us	500 us	1.000 us	1.500 us	2,000 us
🐻 clk	1					
🔓 rst	0					
data0[63:0]	000000000002d1a					
data1[63:0]	000000000000de2	000000000000000000000000000000000000000	o	C	000 00000000000000000000000000000000000	000000004c2
data2[63:0]	00000000000011b	000000000000000000000000000000000000000	00 XXXXX	0	000 00000000000000000000000000000000000	00000000116
data3[31:0]	000000e4	00000000	XXXXXX		000000c5 XXXXXX	000000e4
data4[31:0]	0000036	00000000	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Omoro militario di manda di concerne co	00000031 💥	0000036
data5[31:0]	00000061	00000000	XXXXXX8808		0000005.	00000061
data6[31:0]	0000000e	00000000	2000000	XXX 0000000 X 000000	o » 🔬 🤇	0000000
🕼 rst_pro	0					
Use ser_on	0					
1 pro_end	1					
char[7:0]	0				Х	4



Name	Value	1.678,500 ns 11.679,000 ns 11.679,500 ns 11.580,000 ns
<pre> clk clk rst data0[63:0] data1[63:0] data2[63:0] data3[31:0] data5[31:0] data5[31:0] data6[31:0] data6[31:0] lig rst_pro lig ser_on lig pro end </pre>	1 0 0000000000000000000000000000000000	
char[7:0]	o	

図 4.1.9: Foradd 拡大波形図

Fibonacci

Fibonacci はフィボナッチ数列を 10 項目まで出力するプログラムである. Fibonacci の ISE シミュ レーション結果を図 4.1.10 に,ソースコードをソースコード 4.4 に全体波形図を図 4.1.11 に,一 部拡大波形図を図 4.1.12 に示す.

まず図 4.1.10 を見ると,0,1,1,2,3,5,8,13,21,34 と出力されている.この結果とソー スコード 4.4 から正しくプログラムが動作していることがわかる.次に図 4.1.11 を見ると,3 列目 から順番に [2d1a,dc2,11b,e4,36,61,e] と出力されている.これはプログラム実行後に出力 される性能評価指数を表していて,図 4.1.10 の最後に同様の値が出力されている.最後に図 4.1.12 を見ると,最下段の信号 char の赤枠内に34 が出力されている.これは出力結果を格納している信 号であり,他の値も同様に出力されている.これらの結果からカスタムプロセッサ上で Fibonacci が実行され,回路が正常に動作していることがわかる.

Syscall : Write		
0	Syscall : Write	
Syscall : Write		
	13	
1	Syscall : Write	
Syscall : Write		
	21	
1	Syscall : Write	
Syscall : Write		
	34	
2	Syscall : Exit	
Syscall : Write	cycle counter :0x000000000	07eed
	run cycle counter(cycle - swi) :0x00000000000	03bc7
3	arm_terminal_instruction_counter :0x00000000000	0096a
Syscall : Write	armcache_hit :0x00000abe	
	armcache_miss :0x00000048	
5	op1cache_hit :0x000003a1	
Syscall : Write	op1cache_miss :0x00000072	
9		

図 4.1.10: Fibonacci シミュレーション結果

ソースコード 4.4: Fibonacci

1	<pre>#include <stdio.h></stdio.h></pre>
2	<pre>int main(void){</pre>
3	int a,b=0,c=1,i;
4	for(i=1;i<=10;i++){
5	<pre>printf("%d\n", b);</pre>
6	a = b+c;
7	b = c;
8	$c = a;$ }
9	return 0;}

							SISUSUSUU .
Name	Value	0 ms	1 ms	2 ms	β ms	4 ms	5 ms 6 ms
🔚 cik	1						
🔚 rst	0						
data0[63:0]	0000000000007eed						
data1[63:0]	000000000003bc7	000000000000000000000000000000000000000	00	00	00)	00	000000000000000000000000000000000000000
data2[63:0]	00000000000096a	000000000000000000000000000000000000000	00	00	00)	00	000000000000000000000000000000000000000
▶ 🦷 data3[31:0]	00000abe	00000000	00	····)/////////////////////////////////	00	00	00000abe
▶ 🦷 data4[31:0]	00000048	00000000	00	···· X 0···· X 000··· X 0··· X 000··	· X00 X 00000034 X 0000003	occocc41	4 X 00000046 X000… X 00000048
data5[31:0]	000003a1	00000000	() (000	00	00	(00)	000003a1
▶ 🌄 data6[31:0]	00000072	00000000	0000000 хили (хоо	···)//////////////////////////////////	(000···)((000····)((000····)((000····)((000····)((000····)((000····)((000····)((000····)((000····)((000····)((000····)((000····)((000·····)((000·····)((000·····)((000·····))))))))	() 000···)	00000 ···)/////////////////////////////
1 rst_pro	0						
1 ser_on	0						
🕼 pro_end	1						
🕨 📷 char[7:0]	_	(χ.		K X X	· X · X	<u> </u>

図 4.1.11: Fibonacci 全体波形図

Name	Value	5,889 us 5,890 us 5,891 us 5,892 us
16 clk	0	
16 rst	0	
data0[63:0]	00000000000072fb	
data1[63:0]	00000000003b1e	00000000000000000000000000000000000000
data2[63:0]	00000000000095d	000000000000000000000000000000000000000
data3[31:0]	00000a8e	00000488
data4[31:0]	00000047	
data5[31:0]	0000039a	00000397 00000398 00000399 0
data6[31:0]	0000006e	
1 rst_pro	o	
1 ser_on	0	
14 pro_end	0	
char[7:0]	-	- 3 4

図 4.1.12: Fibonacci 拡大波形図

FizzBuzz

FizzBuzz は 15 までの数値の内,3 の倍数に Fizz を 5 の倍数に Buzz を 15 の倍数に FizzBuzz を 出力するプログラムである.FizzBuzz の ISE シミュレーション結果を図 4.1.13 に,ソースコード をソースコード 4.5 に,全体波形図を図 4.1.14 に,一部拡大波形図を図 4.1.15 に示す.

まず図 4.1.13 を見ると,1,2, Fizz,...,13,14, Fizz,Buzz と出力されている.この結果とソー スコード 4.5 から正しくプログラムが動作していることがわかる.次に図 4.1.14 を見ると,3列 目から順番に [a7c1,50bf,c4c,e0f,48,51c,93]と出力されている.これはプログラム実行後 に出力される性能評価指数を表していて,図 4.1.13 の最後に同様の値が出力されている.最後に 図 4.1.15 を見ると,最下段の信号 char にの赤枠内 Fizz,Buzz が出力されている.これは出力結果 を格納している信号であり,他の値も同様に出力されている.これらの結果からカスタムプロセッ サ上で FizzBuzz が実行され,回路が正常に動作していることがわかる.

Syscall : Write	Syscall : Write
Syscall : Write	Buzz Syscall : Write
2	-,
Syscall : Write	11 Syscall : Write
Fizz	
Syscall : Write	Fizz Syscall : Write
4	
Syscall : Write	13 Syscall : Write
Buzz	
Syscall : Write	14 Svscall : Write
Fizz	
Syscall : Write	Fizz,Buzz Svecall : Exit
7	cycle counter :0x00000000000002c1
Syscall : Write	run_cycle_counter(cycle - swi) :0x000000000000050bf arm terminal instruction counter :0x000000000000064c
8	armoache hit :0x00000000000000000000000000000000000
Syscall : Write	armcache_miss :0x00000048
Fizz	op lcache_miss :0x0000093

図 4.1.13: FizzBuzz シミュレーション結果

ソースコード 4.5: FizzBuzz

1	#include <stdio.h></stdio.h>
2	<pre>int main(void){</pre>
3	int i;
4	for (i = 1; i <= 15; i++) {
5	if (i % $3 == 0 \&\& i \% 5 == 0$)
6	<pre>printf("Fizz,Buzz\n");</pre>
7	else if (i % 3 == 0)
8	<pre>printf("Fizz\n");</pre>
9	else if (i % 5 == 0)
10	<pre>printf("Buzz\n");</pre>
11	else
12	printf("%d\n", i);}
13	return 0;}

									8.58	95460000 ms
Name	Value	10 ms	1 ms	2 ms	3 ms	4 ms	5 ms	6 ms	7 ms 8 m	ms
16 cik	1									
1a rst	0									
data0[63:0]	000000000000a7c1									
data1[63:0]	0000000000050bf	000000000000000000000000000000000000000	0	0	•	0	0	··	0	00000000
data2[63:0]	0000000000000c4c	00000000000000000	0	0)	·	0)	0		0	0000000
data3[31:0]	00000e0f	00000000	0	0)	·	0	0		0	00000e0f
data4[31:0]	00000048	00000000	0····)(···· <u>X00000 ··· X</u> 0··· X	0000003c	<u>%0000003e %</u>	00000040	00000044 💥 000	0000048	8
data5[31:0]	0000051c	00000000	() (0	····) (0···) (0···) (0···	· ////////////////////////////////////	0	0	··) (0···) (···)	(IIII) (III) (IIII) (III) (III	0000051c
data6[31:0]	00000093	00000000	00000000	0	00	0	00 (00) (0.	· /00/00····)	(000)(·)(00000····)
🔓 rst_pro	0									
🔓 ser_on	0									
pro_end	1									
char[7:0]	-	· ·	χ.		<u>x · x · x</u>			X		.)

図 4.1.14: FizzBuzz 全体波形図

Name	Value	7	982 us	7,983 us	17,984 us	7,985 us	7,986 us
18 clk	1						
16 rst	0						
data0[63:0]	bbde00000000000	··· X··· X					
data1[63:0]	000000000005010						0000000000
data2[63:0]	000000000000c41						
data3[31:0]	00000def						
data4[31:0]	0000048						
data5[31:0]	00000513			00000510		∕00000511	00000512
data6[31:0]	0000008£						
1 rst_pro	0						
1 ser_on	0						
1 pro_end	0						
char[7:0]	_				τ χ.	X B (u)	z

図 4.1.15: FizzBuzz 拡大波形図

Sort

Sort はランダムに並んだ数値を一度出力し,パブルソート後ソートされた数値を出力するプロ グラムである.SortのISEシミュレーション結果を図 4.1.16 に,ソースコードをソースコード 4.6 に,全体波形図を図 4.1.17,一部拡大波形図を図 4.1.18 に示す.

まず図 4.1.16を見ると,80,5,36,...,9,1,78 と出力された後,1,5,9,...,78,80,100 と出力されている.この結果とソースコード4.6から正しくプログラムが動作していることがわか る.次に図 4.1.17を見ると,3列目から順番に[fd99,83ed,17de,47ab,94,8e7,102]と出力 されている.これはプログラム実行後に出力される性能評価指数を表していて,図 4.1.16の最後 に同様の値が出力されている.最後に図 4.1.18を見ると,最下段の信号 char の赤枠内に80が出力 されている.これは出力結果を格納している信号であり,他の値も同様に出力されている.これ らの結果からカスタムプロセッサ上でSortが実行され,回路が正常に動作していることがわかる.

Syscall : Write 80, Syscall : Write 5,	Syscall : Write 1, Syscall : Write 5,	Syscall : Exit cycle_counter run_cycle_counter(cycle - swi) arm terminal instruction counter	:0×0000000000000fd99 :0×00000000000083ed :0×00000000000017de	
Syscall : Write 36.	Syscall : Write 9	armcache_hit armcache_miss	:0x000047ab	
Syscall : Write 23,	Syscall : Write 12,	op 1cache_hit op 1cache_miss	:0x000008e7 :0x00000102	
Syscall : Write 12,	Syscall : Write 23.	op lodollo_illise		
Syscall : Write 100.	Syscall : Write 36			
Syscall : Write 45,	Syscall : Write			
Syscall : Write 9,	Syscall : Write 78.			
Syscall : Write 1.	Syscall : Write 80,			
Syscall : Write 78,	Syscall : Write 100,			
Syscall : Write	Syscall : Write			

図 4.1.16: Sort シミュレーション結果

ソースコード 4.6: Sort

1	<pre>#include <stdio.h></stdio.h></pre>
2	<pre>int main(){</pre>
3	int data[MAX] = { 80,5,36,23,12,100,45,9,1,78 };
4	int n,i,w;
5	for (i=0; i <max;){<="" i++="" td=""></max;>
6	printf("%d,",data[i]);
7	<pre>printf("\n");</pre>
8	for (n=MAX; n>1; n){
9	for (i=0; i <n-1;){<="" i++="" td=""></n-1;>
0	if $(data[i] > data[i+1]) $
1	w=data[i];
2	data[i] = data[i+1];
3	$data[i+1]=w;}\}$
4	for (i=0; i<10; i++){
5	<pre>printf("%d,",data[i]);}</pre>
6	printf("\n");}

							l l	2.330000000 ms
Name	Value	10 ms	2 ms	4 ms	6 ms	8 ms	10 ms	12 ms
🔓 clk	1							
16 rst	0							
Interpretation [63:0]	000000000000fd99							
data1[63:0]	0000000000083ed	00000000						00000
data2[63:0]	00000000000017de	00000000						···· \00000
data3[31:0]	000047aa	00000000				······		
data4[31:0]	00000094	0000000	000… ※000… ※000… ※000…	<u> </u>	00)	X000 X000 X000 X000	·· X 000··· X 000··· X 000··· X	00000094
data5[31:0]	000008e7	00000000						X00000
data6[31:0]	00000102	00000000						
1 rst_pro	0							
1 ser_on	0							
1 pro_end	1							
char(7:0)	-	· X						

図 4.1.17: Sort 全体波形図

Name	Value	11,500,500 ns 11,501,000 ns 111,501,500 ns
16 clk	1	
1 rst	0	
data0[63:0]	00000000000e088	···· X000··· X
data1[63:0]	000000000007a7a	يستحسني فسنعدف وشحيتها في ا
data2[63:0]	000000000001662	
data3[31:0]	00003241	···· \(\doc{000···· \(\doc{000···· \(\doc{000···· \(\doc{000···· \(\doc{000···· \(\doc{000··· \(\doc{00··· \doc{00··· \(\doc{00··· \(\doc{00··· \(\doc{00··· \(\doc{00··· \(\doc{00··· \(\doc{00··· \doc{00··· \doc{00··· \doc{00·· \doc{00··· \doc{00··· \doc{00·· \doc{00·} \doc{00·· \doc{00·} \doc{00·} \doc{00·} \doc{00·} \doc{00} \doc{0} \do
data4[31:0]	00000092	
data5[31:0]	00000837	0000837
data6[31:0]	000000eb	
1 rst_pro	0	
1 ser_on	0	
1 pro_end	0	
char[7:0]	*	. <mark>8 X 0)</mark>

図 4.1.18: Sort 拡大波形図

Sosu

Sosu はエラトステネスの篩で 50 までの素数を出力するプログラムである. Sosu のソースコー ドをソースコード 4.7 に, ISE のシミュレーション結果を図 4.1.19 に, 全体波形図を図 4.1.20 に, -部拡大波形図を図 4.1.21 に示す.

まず図 4.1.19 を見ると,2,3,5,7,...41,43,47 と出力されている.この結果とソースコー ド 4.7 から正しくプログラムが動作していることがわかる.次に図 4.1.20 を見ると,3 列目から順 番に [c913, 6eb4, 1364, 15ea, 6b, 6b4, e2] と出力されている.これはプログラム実行後に出力 される性能評価指数を表していて,図4.1.19の最後に同様の値が出力されている.最後に図4.1.21 を見ると,最下段の信号 char にの赤枠内4...が出力されている.これは出力結果を格納している 信号であり,他の値も同様に出力されている.これらの結果からカスタムプロセッサ上で Sosu が 実行され,回路が正常に動作していることがわかる.

Syscall : Write	Syscall : Write	
Syscall : Write 3	Syscall : Write 31	
Syscall : Write 5	Syscall : Write 37	
Syscall : Write 7	Syscall : Write 41	
Syscall : Write 11	Syscall : Write 43	
Syscall : Write 13	Syscall : Write 47	
Syscall : Write 17	cycle_counter run cycle counter(cycle - swi)	:0x0000000000000c913 :0x00000000000006eb4
Syscall : Write 19	arm_terminal_instruction_counter armcache hit	:0x0000000000001364 :0x000015ea
Syscall : Write 23	armcache_miss op1cache hit	:0x0000006b :0x000006b4
	op1cache_miss	:0x000000e2

図 4.1.19: Sosu シミュレーション結果

ソースコード 4.7: Sosu

- 1 **#include** <stdio.h>
- 2 **#include** <math.h>
- 3 #define NUM 50
- 4 int main(void){
- unsigned i, j; 5
- **unsigned** sq_num = (**int**)sqrt((**double**)NUM); 6
- unsigned prime[NUM]; 7
- **for** (i = 0; i < NUM; i++) 8

```
prime[i] = 1;
9
```

```
prime[0] = 0;
10
```

```
for (i = 1; i < sq_num; i++) {
11
```

```
if (prime[i] == 1)
12
```

```
for (j = (i+1); (i+1) * j \le NUM; j++)
13
                 prime[(i+1) * j - 1] = 0;
```

```
14
```

```
for (i = 0; i < NUM; i++)
15
```

if (prime[i] == 1) 16

```
printf("%3d", i + 1);
17
```

```
putchar('\n');
18
        eturn (0)
19
```

											1 0 3 01 460000 ms
Name	Value	10 ms	1 ms	2 ms	β ms	4 ms	5 ms	6 ms	17 ms	8 ms	9 ms 10 ms
🔚 cik	1										
16 rst	0										
data0[63:0]	000000000000000000000000000000000000000										
data1[63:0]	000000000006eb4	000000000000000000000000000000000000000	0								· XXXXXXX (··· X 000000····)
data2[63:0]	000000000001364	000000000000000000000000000000000000000	0								·);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
data3[31:0]	000015ea	00000000									·)////////////////////////////////////
data4[31:0]	0000006b	00000000 X	0	<u> </u>	0)/ 0000)// 0	000… 💥 0000004f%	00000 🗶 00000056	00000058 000000	5c)X0000005f)X000	00063 🗸 000000 ··· 💥 0	0000069 🗙 00000066
data5[31:0]	000006b4	00000000	0								·)////////////////////////////////////
data6[31:0]	000000e2	00000000		0	0	0			0	0	·
🕼 rst_pro	0										
læ ser_on	0										
le pro_end	1										
char[7:0]	7			2 3	5 1 7	χ 1 χ 3	XTX	<u>χ 3 χ</u>	9 χ 1	(7)(1)	χ 3 χ 7

図 4.1.20: Sosu 全体波形図

Name	Value		9,119 us	9,120 us	9,121 us
18 clk	1				
1 rst	0				
data0[63:0]	0000000000b20e	X X X X X			
data1[63:0]	0000000000066£8			00	0000000006618
▶ 駴 data2[63:0]	00000000000120e				000
▶ 🛃 data3[31:0]	00001432				00001 432
▶ 🚮 data4[31:0]	00000069				
data5[31:0]	0000063e		00000635		X0000063c
data6[31:0]	000000ce				
1 rst_pro	0				
1 ser_on	0				
1 pro_end	0				
▶ 📷 char[7:0]	3	1	x x x		

図 4.1.21: Sosu 拡大波形図

Himeno

Himeno は姫野ベンチマークの行列計算 (5 × 5 × 9)を指定回数行うプログラムである. Himeno の ISE のシミュレーション結果を図 4.1.22 に,全体波形図を図 4.1.23 に,一部拡大波形図を図 4.1.24 に示す. Himeno のソースコードは非常に長文のため付録に添付する.

まず図 4.1.19 を見ると, mimax=5 mjmax=5 mkmax=9 … Process end と出力されている.この 結果とソースコード 5.1 から正しくプログラムが動作していることがわかる.次に図 4.1.20 を見 ると,3列目から順番に [d12a9,cc186,4aecb,5c305,446,8b43,1c8] と出力されている.こ れはプログラム実行後に出力される性能評価指数を表していて,図 4.1.19 の最後に同様の値が出 力されている.最後に図 4.1.21 を見ると,最下段の信号 char の赤枠内に Process end が出力されて いる.これは出力結果を格納している信号であり,他の値も同様に出力されている.これらの結 果からカスタムプロセッサ上で Himeno が実行され,回路が正常に動作していることがわかる.

Syscall : Write mimax = 5 mjmax = 5 mkmax = 9

Syscall : Write imax = 4 jmax = 4 kmax =8

Syscall : Write Start rehearsal measurement process.

Syscall : Write Measure the performance in 10 times.

Syscall : Write process end

 Syscall : Exit
 :0x000000000012a9

 run_cycle_counter(cycle - swi)
 :0x00000000000c186

 arm_terminal_instruction_counter
 :0x00000000004aecb

 armcache_hit
 :0x00005c305

 armcache_hit
 :0x0000000446

 op1cache_hit
 :0x00008b43

 op1cache_miss
 :0x00001c8

図 4.1.22: Himeno シミュレーション結果

									1741	355450000 ms
Name	Value	0 ms	20 ms	140 ms	60 ms	80 ms	1 00 ms	120 ms	140 ms	160 ms
🔚 cik	1									
16 rst	0									
16 sw1	0									
🔓 sw2	0									
data0[63:0]	00000000000d12a9									
data1[63:0]	00000000000cc186									
data2[63:0]	000000000004aecb	Ō								
▶ 🍓 data3[31:0]	0005c305	Ō								
▶ 🍓 data4[31:0]	00000446	000								
▶ 🍓 data5[31:0]	00008b43	$\overline{(}$								
▶ 🔣 data6[31:0]	000001c8	0X()		· · · · · · · · · · · · · · · · · · ·		X X X X X X X X X X X X X X X X X X X)))))))))))))))))))))))))))))))))))))))	0000	0000	
🕼 rst_pro	0									
1 ser_on	0									
🔓 pro_end	1									
🕨 📷 char[7:0]	0a	00				0a				X



	Nam	ie	Value	 170,751 us	170,752 us	170,753 us	170,754 us	170,755 us
	16	j clk	0					
	٦.	rst	0					
	٦.	sw1	0					
	16	sw2	0					
b	-	data0[63:0]	0000000000d06f4					
þ	- 📝	data1[63:0]	00000000000cc101				000000000	00cc1 01
b	-	data2[63:0]	000000000004aec3					000000000004aec
b	-	data3[31:0]	0005c2db					0005c2db
h	-	data4[31:0]	00000446					00000446
Б	-	data5[31:0]	00008b3a	00008637		X00008638 X0000	8b39 X	
Б	-		000001c7					000001 c7
ľ	U	rst_pro	0					
	10	ser_on	0					
	10	pro_end	0					
1	-	char[7:0]		P X T X O X	с <u>х</u> е <u>х</u>	s X X	e X n X d	

図 4.1.24: Himeno 拡大波形図

4.2 シリアル通信のメモリアクセス検証

カスタムプロセッサに追加実装したシリアル通信の入力データとプログラム実行後の出力デー タを比較し,二つの差異を見ることでシリアル通信が正しく実装されていることを確認する. 図 4.2.1 から図 4.2.8 に各プログラムごとのデータの入出力の結果とデータ差異を示す.各図は上 が入力データを下が出力データを表している.また,赤く塗られている部分が上下のデータで差 異がある部分である.

各プログラムが正しく動作していることは確認されているため,どの図においても最後に出力 した結果がメモリに格納されていることがわかる,もし,シリアル通信の設計が正しくできてい ない場合,アドレスのずれやビット欠けによるデータの欠落などが起こる.しかし,どの図をとっ てみてもアドレスのずれやデータの欠落はしていない.この結果から入力データと出力データが 正しくやり取りされていることが証明でき,シリアル通信を含む回路全体が正しく実装されてい ることがわかる,

Hillo.bin																		[0			8
ADDRESS	00 01	02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	012	345	678	9A	BCDE	EF	*
0001F260	00 00	0D	00	0E	00	0F	00	10	00	00	00	48	65	60	60					.He	11	
0001F270	6F 0/	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0			••		• •	+
output-Hille	o.bin																		-			83
ADDRESS	o.bin 00 01	02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	012	345	678		BCDE	D)(EF	83
Determine output-Hille	o.bin 00 01 0C 00	02 0D	<mark>03</mark> 00	04 0E	<mark>05</mark> 00	<mark>06</mark> 0F	<mark>07</mark> 00	<mark>08</mark> 10	<mark>09</mark> 00	<mark>0A</mark> 00	<mark>0B</mark> 00	0C 48	0D 69	0E 6C	OF 6C	012	345	678	9A	BCDE	5)(EF 11	8

図 4.2.1: Hillo メモリデータ入出力

🖻 Gusu.bin																	(0			83
ADDRESS	00 01	02 0	3 04	05	06	07	08	09	0A	0B	00	0D	0E	0F	01	2345	67	894	BC	DEF	^
000FF710	00 00	00 00	00 (00	00	00	00	00	00	00	00	00	00	00							
000FF720	00 00	00 00	00 (00	00	00	00	00	00	00	00	00	00	00					••		
000FF730	00 00	00 00	00	00	00	00	00	00	00	00	00	00	00	00			• •		•••		
000FF740	00 00	00 00	00 (00	00	00	00	00	00	00	00	00	00	00			••		••		
000FF750	00 00	00 00	00 (00	00	00	00	00	00	00	00	00	00	00			••		••		
000FF760	00 00	00 00	00	00	00	00	00	00	00	00	00	00	00	00			••	•••	••	•••	-
w output-Gus	su.bin																(0			×
ADDRESS	su.bin 00 01	02 0	3 04	05	06	07	08	09	0A	0B	00	0D	0E	0F	01	2345	67	89/	BC	□)(DEF	×
Determination of the second se	su.bin 00 01 00 00	02 03 00 01	<mark>3 04</mark>) 00	<mark>05</mark> 00	<mark>06</mark> 00	<mark>07</mark> 00	<mark>08</mark> 00	<mark>09</mark> 00	<mark>0A</mark> 31	0B 30	<mark>0C</mark> 30	<mark>0D</mark> 00	0E 00	<mark>0F</mark> 00	01	<mark>2345</mark>	67	89/	BCI 00	□)(DEF	*
output-Gus ADDRESS 000FF710 000FF720	su.bin 00 01 00 00 00 00	02 0 00 0 00 0	<mark>3 04</mark>) 00) 00	<mark>05</mark> 00 00	<mark>06</mark> 00 00	<mark>07</mark> 00 00	<mark>08</mark> 00 00	<mark>09</mark> 00 00	<mark>0A</mark> 31 00	0B 30 00	0C 30 00	0D 00 00	0E 00 00	0F 00 00	01	2345	67 	89/ 1	BCI 00	©)(DEF ···∸	× ^
DOUT CONTRACT CONTRACTICA TERECT CONTRACTICA TERCONTRACT CONTRACT CONTRACT CONTRACT	su.bin 00 01 00 00 00 00 00 00	02 0 00 0 00 0 00 0	<mark>3 04</mark>) 00) 00) 6C	05 00 00 F7	06 00 00 FF	<mark>07</mark> 00 00 07	08 00 00 00	09 00 00 00	<mark>0A</mark> 31 00 00	0B 30 00 00	0C 30 00 00	0D 00 00 00	0E 00 00 00	0F 00 00 00	01	2345 	67 67	89/ 1	BCI 00	□ (DEF · · · <u>-</u>	*
■ output-Gus ADDRESS 000FF710 000FF720 000FF730 000FF740	su.bin 00 01 00 00 00 00 00 00 00 00	02 03 00 01 00 01 00 01 00 01	3 04) 00) 00) 60) 60	05 00 00 F7 00	06 00 00 FF 00	07 00 00 07 00	08 00 00 00 00 00	09 00 00 00 00	0A 31 00 00 00	0B 30 00 00 00	0C 30 00 00 00	0D 00 00 00 00	0E 00 00 00 00	0F 00 00 00 00	01	2345 	67 	89/	BCI 00	▣)(DEF · · <u>·</u> · · ·	~
output-Gus ADDRESS 000FF710 000FF720 000FF730 000FF740 000FF750	su.bin 00 01 00 00 00 00 03 00 03 00 00 00 08 50	02 0 00 0 00 0 00 0 00 0 00 0	<mark>3 04</mark>) 00) 00) 60) 00) 00	05 00 00 F7 00 00	06 00 00 FF 00 00	07 00 00 07 00 00 00	08 00 00 00 00 00	09 00 00 00 00 00	0A 31 00 00 00 00	0B 30 00 00 00 00	0C 30 00 00 00 18	0D 00 00 00 00 00	0E 00 00 00 00 00	0F 00 00 00 00 00	01 	2345	67 	89/	BCI 00	▣)(DEF · · <u>·</u> · · ·	×

図 4.2.2: Gusu メモリデータ入出力

Foradd.bin																	23
ADDRESS	00 0	1 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF	*
000FF710	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF720	0 00	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF730	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF/40	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00	•••••	
000FF750	00 0	0 00	00	00	00	00	00	00	00	00	00	ÛÛ	00	<u>00</u>	00	•••••	
000FF760	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF770	UU U	U UU	υu	UU	00	00	UU	00	00	UU	UU	UU	00	00	UU		-
output-For	add.bir	n					-			_							23
Dutput-For	add.bir	n 1 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF	23
Dutput-For ADDRESS 000FF710	add.bir 00 0 00 0	n <mark>1 02</mark> 0 00	<mark>03</mark> 00	<mark>04</mark> 00	<mark>05</mark> 00	<mark>06</mark> 00	<mark>07</mark> 00	<mark>08</mark> 00	<mark>09</mark> 00	<mark>0A</mark> 00	0B 00	0C 00	0D 34	0E 39	0F 35	0123456789ABCDEF 495	23
DUTPUT-For ADDRESS 000FF710 000FF720	add.bir 0000 000 3000	1 02 0 00 0 00	03 00 00	04 00 00	05 00 00	<mark>06</mark> 00 00	<mark>07</mark> 00 00	<mark>08</mark> 00 00	<mark>09</mark> 00 00	<mark>0A</mark> 00 00	0B 00 00	0C 00 00	0D 34 00	0E 39 00	0F 35 00	0123456789ABCDEF 	83
DUTE: 1000000000000000000000000000000000000	add.bir 00 0 00 0 00 0 00 0	1 02 0 00 0 00 0 00	03 00 00 00	04 00 00 00	05 00 00 10	06 00 00 00	<mark>07</mark> 00 00 00	08 00 00 70	09 00 00 F7	0A 00 00 FF	0B 00 00 07	00 00 00 00	0D 34 00 00	0E 39 00 00	0F 35 00 00	0123456789ABCDEF 	83
	add.bir 0000 0000 0000 0000 0000	1 02 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00	04 00 00 00 00	05 00 00 10 00	06 00 00 00 00	07 00 00 00 00	08 00 00 70 00	09 00 00 F7 00	0A 00 00 FF 00	0B 00 00 07 07	00 00 00 00 00	0D 34 00 00 00	0E 39 00 00 00	0F 35 00 00 00	0123456789ABCDEF 	83
DOFF710 000FF710 000FF720 000FF730 000FF740 000FF750	add.bir 0 0 0 0 00 0 0 0 00 0 00 0 00	1 02 0 00 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00	04 00 00 00 00 00	05 00 00 10 00 5D	06 00 00 00 00 00	07 00 00 00 00 00	08 00 00 70 00 00	09 00 00 F7 00 00	0A 00 00 FF 00 00	0B 00 00 07 00 00	00 00 00 00 00	0D 34 00 00 00 00	0E 39 00 00 00 00	0F 35 00 00 00 00	0123456789ABCDEF 	23
■ output-For ADDRESS 000FF710 000FF720 000FF730 000FF740 000FF750 000FF760	add.bir 00 0 00 0 00 0 00 0 00 0 00 0	1 02 0 00 0 00 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00 00	04 00 00 00 00 00 00	05 00 00 10 00 5D 00	06 00 00 00 00 00 01 00	07 00 00 00 00 00 00 00	08 00 00 70 00 00 00	09 00 00 F7 00 00 00	0A 00 00 FF 00 00 00	0B 00 00 07 00 00 00	0C 00 00 00 00 00	0D 34 00 00 00 00 00	0E 39 00 00 00 00 00	0F 35 00 00 00 00 00 00	0123456789ABCDEF 	

図 4.2.3: Foradd メモリデータ入出力

Fibonacci.	bin																
ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF •
000FF710	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF720	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF730		00	00	00	00	00	00	00	00	00	UU	00	UU	00	00	00	
000FF740	00	00	00	00	00	00	00	00	00	00	00	00		00	00	00	
000FF750	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00011700	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	•••••••••••••••••••••••••••••••••••••••
7				_	_									_	_		
output-Fil	oonac	ci.bi	n														
ADDRESS	oonac 00	ci.bi	n 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF ^
DOUT CONTRACT IN CONTRACT.	oonac 00 00	ci.bi 01 00	n 02 00	03 33	04 34	<mark>05</mark> 00	<mark>06</mark> 00	<mark>07</mark> 00	<mark>08</mark> 00	<mark>09</mark> 00	<mark>0A</mark> 00	<mark>0B</mark> 00	<mark>0C</mark> 00	<mark>0D</mark> 00	0E 00	0F 00	0123456789ABCDEF ^ 34
ADDRESS 000FF710 000FF720	00000000000000000000000000000000000000	ci.bi 01 00 00	n 02 00 00	03 33 00	04 34 00	05 00 00	06 00 00	07 00 00	08 00 03	09 00 00	00 00 00	0B 00 00	0C 00 64	0D 00 F7	OE 00 FF	0F 00 07	0123456789ABCDEF 34d
output-Fil ADDRESS 000FF710 000FF720 000FF730 000FF730	00000000000000000000000000000000000000	ci.bi 01 00 00 00	n 00 00 00	03 33 00 00	04 34 00 00	05 00 00 00	06 00 00 00	07 00 00 00	08 00 03 00	09 00 00 00	<mark>0A</mark> 00 00 00	0B 00 00 00	0C 00 64 00	0D 00 F7 00	0E 00 FF 00	0F 00 07 00	0123456789ABCDEF 34d 7d
Dependence of the second secon	00 00 00 00 87 00	ci.bi 01 00 00 00 00	n 00 00 00 00 00	03 33 00 00 00	04 34 00 00 00	05 00 00 00 00	06 00 00 00 00	07 00 00 00 00	08 00 03 00 08	09 00 00 00 50	0A 00 00 00 00	0B 00 00 00 00	00 00 64 00 37	0D 00 F7 00 00	0E 00 FF 00 00	0F 00 07 00 00	0123456789ABCDEF 34d 7 y]7
■ output-Fil ADDRESS 000FF710 000FF720 000FF730 000FF730 000FF740 000FF750 000FF750	00 00 00 00 00 00 00	ci.bi 01 00 00 00 00 00 00	02 00 00 00 00 00 00	03 33 00 00 00 00 00	04 34 00 00 00 00	05 00 00 00 00 00 00 00	06 00 00 00 00 00	07 00 00 00 00 00 00	08 00 03 00 00 00	09 00 00 00 5D 00	0A 00 00 00 00 00	0B 00 00 00 00 00 00	00 00 64 00 37 18 00	0D 00 F7 00 00 00 00	0E 00 FF 00 00 00	0F 00 07 00 00 00 00	0123456789ABCDEF 34d 7y]7

図 4.2.4: Fibonacci メモリデータ入出力

FizzBuzz.bi	in															
ADDRESS	00 01	02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF
000FF710	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF720	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF730	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF740	00 00	00	00	00	00	00	00	00	00	00	00	00	00	_00	00	
000FF750	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF760	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF770	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	• • • • • • • • • • • • • • • • • • • •
output-Fiz	zBuzz.bi	n														
ADDRESS	zBuzz.bi	n 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF
ADDRESS 000FF710	zBuzz.bi 00 01 00 00	n 02 00	<mark>03</mark> 00	<mark>04</mark> 00	<mark>05</mark> 00	<mark>06</mark> 00	07 31	08 34	<mark>09</mark> 00	<mark>0A</mark> 00	<mark>0B</mark> 00	0C 00	0D 00	0E 00	0F 00	0123456789ABCDEF
ADDRESS 000FF710 000FF720	zBuzz.bi 00 01 00 00 00 00	n 02 00 00	03 00 00	04 00 00	05 00 00	06 00 00	07 31 00	08 34 00	09 00 00	<mark>0A</mark> 00 00	0B 00 00	0C 00 00	0D 00 00	0E 00 00	0F 00 00	0123456789ABCDEF
output-Fiz: ADDRESS 000FF710 000FF720 000FF730	zBuzz.bi 00 01 00 00 00 00 68 F7	02 00 00 FF	03 00 00 07	04 00 00 00	05 00 00 00	06 00 00 00	07 31 00 00	08 34 00 00	09 00 00 00	<mark>0A</mark> 00 00 00	0B 00 00 00	00 00 00 00	0D 00 00 00	0E 00 00 00	0F 00 00 00	0123456789ABCDEF
output-Fiz ADDRESS 000FF710 000FF720 000FF730 000FF740	zBuzz.bi 00 01 00 00 00 00 68 F7 00 00	n 02 00 00 FF 00	03 00 00 07 00	04 00 00 00 00	05 00 00 00 00	06 00 00 00 00	07 31 00 00 00	08 34 00 00 00	09 00 00 00 00	<mark>A0</mark> 00 00 00 00	0B 00 00 00 00	00 00 00 00	0D 00 00 00 5D	0E 00 00 00	0F 00 00 00 00	0123456789ABCDEF
output-Fiz: ADDRESS 000FF710 000FF720 000FF730 000FF740 000FF750	zBuzz.bi 00 01 00 00 00 00 68 F7 00 00 0F 00	n 00 00 FF 00 00	03 00 00 07 00 00	04 00 00 00 00 00	05 00 00 00 00 00	06 00 00 00 00 00	07 31 00 00 00 00	08 34 00 00 00 00	09 00 00 00 00 00	00 00 00 00 00 00	0B 00 00 00 00 00	0C 00 00 00 00 00	0D 00 00 00 5D 00	0E 00 00 00 00 01	0F 00 00 00 00 00	0123456789ABCDEF
output-Fize ADDRESS 000FF710 000FF720 000FF730 000FF740 000FF750 000FF760	zBuzz.bi 00 01 00 00 00 00 68 F7 00 00 0F 00 00 00	02 00 00 FF 00 00 00	03 00 00 07 00 00 00	04 00 00 00 00 00 00	05 00 00 00 00 00 00	06 00 00 00 00 00 00	07 31 00 00 00 00 00	08 34 00 00 00 00 46	09 00 00 00 00 00 00 69	0A 00 00 00 00 00 00 7A	0B 00 00 00 00 00 7A	00 00 00 00 00 18 20	0D 00 00 00 5D 00 42	0E 00 00 00 01 01 75	0F 00 00 00 00 00 7A	0123456789ABCDEF 14 h

図 4.2.5: FizzBuzz メモリデータ入出力

Sort.bin																	3
ADDRESS	00 0	1 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF	*
000FF6E0	00 00	00 0	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF6F0	00 00	00 0	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF700	00 00	00 0	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF710	00 00	00 0	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF720	00 00	00 0	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF730		J 00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF740	00 00	J UU	00	00	00	00	00	00	00	00	00	00	00	00	00		
000FF750	00 00	J UU	00	00	00	00	00	00	00	00	00	00	00	00	00		_
000FF760	00 0	J UU	00	00	00	00	00	00	00	00	00	00	00	00	00	••••••	
000FF770	00 00	1 00	00	UU	UU	00	ŲŬ	00	00	00	00	ŲŲ	00	00	UU		Ŧ
output-Sor	rt.bin		_														3
ADDRESS	rt.bin	1 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F		3
Determination of the second se	rt.bin 00 0 00 01	1 02 0 00	<mark>03</mark> 00	<mark>04</mark> 00	<mark>05</mark> 00	<mark>06</mark> 00	<mark>07</mark> 00	<mark>08</mark> 00	<mark>09</mark> 00	<mark>0A</mark> 00	<mark>0B</mark> 00	<mark>0C</mark> 00	<mark>0D</mark> 00	0E 31	0F 30	0123456789ABCDEF 10	3
DOUTPUT-Son ADDRESS 000FF6E0 000FF6F0	rt.bin 00 0 00 01 30 01	<mark>1 02</mark> 0 00 0 00	03 00 00	04 00 00	05 00 00	06 00 00	<mark>07</mark> 00 00	<mark>08</mark> 00 00	<mark>09</mark> 00 00	<mark>0A</mark> 00 00	0B 00 00	0C 00 00	0D 00 00	0E 31 00	0F 30 00	0123456789ABCDEF 10 0	3
ADDRESS 000FF6E0 000FF6F0 000FF700	rt.bin 00 0 00 00 00 00 00 00	<mark>1 02</mark>) 00) 00) 00	03 00 00 00	04 00 00 03	05 00 00 00	06 00 00 00	07 00 00 00	08 00 00 40	09 00 00 F7	<mark>0A</mark> 00 00 FF	0B 00 00 07	00 00 00 00	0D 00 00 00	0E 31 00 00	0F 30 00 00	0123456789ABCDEF 	3
■ output-Sor ADDRESS 000FF6E0 000FF6F0 000FF700 000FF710	rt.bin 00 0 00 00 00 00 00 00 00 00	1 02 0 00 0 00 0 00 0 00	03 00 00 00 00	04 00 00 03 00	05 00 00 00 00	06 00 00 00 00	07 00 00 00 00	08 00 00 40 00	09 00 00 F7 00	0A 00 00 FF 00	0B 00 00 07 00	0C 00 00 00 00	0D 00 00 00 00	0E 31 00 00 00	0F 30 00 00 00	0123456789ABCDEF 10 0	3
output-Sor ADDRESS 000FF6E0 000FF6F0 000FF700 000FF710 000FF720	t.bin 00 0 00 00 00 00 00 00 00 00 00 00	1 02 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00	04 00 00 03 00 08	05 00 00 00 00 00 5D	06 00 00 00 00 00	07 00 00 00 00 00	08 00 00 40 00 00	09 00 00 F7 00 00	0A 00 00 FF 00 00	0B 00 00 07 00 00	0C 00 00 00 00 00	0D 00 00 00 00 00 00	0E 31 00 00 00 00	0F 30 00 00 00 00	© © ∞ 0123456789ABCDEF 10 0	3
output-Sor ADDRESS 000FF6E0 000FF700 000FF710 000FF720 000FF720 000FF720 000FF720	t.bin 00 00 00 00 00 00 00 00 00 00 00 00 00 00	1 02 0 00 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00 00	04 00 00 03 00 08 09	05 00 00 00 00 5D 00	06 00 00 00 00 00 01 00	07 00 00 00 00 00 00	08 00 00 40 00 00 00 00	09 00 00 F7 00 00 00	0A 00 00 FF 00 00 00	0B 00 00 07 00 00 00	00 00 00 00 00 01 74	0D 00 00 00 00 00 00 F7	0E 31 00 00 00 00 FF	0F 30 00 00 00 00 00 00	© © © © © 0123456789ABCDEF 10 0	3
output-Sor ADDRESS 000FF6E0 000FF6F0 000FF700 000FF710 000FF720 000FF730 000FF740 000FF740 000FF740	rt.bin 00 0 00 00 00 00 00 00 00 00 00 00 00 00 00 00	1 02 0 00 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00 00 00	04 00 03 00 03 00 03	05 00 00 00 00 5D 00 00	06 00 00 00 00 00 00	07 00 00 00 00 00 00 00	08 00 00 40 00 00 00 04 00	09 00 00 F7 00 00 00 00	0A 00 00 FF 00 00 00 00	0B 00 00 07 00 00 00 00	00 00 00 00 00 01 74 00	0D 00 00 00 00 00 F7 00	0E 31 00 00 00 00 FF 00	0F 30 00 00 00 00 00 00 00	© © © © © © 0123456789ABCDEF 10 0	3
■ output-Sor ADDRESS 000FF6E0 000FF6F0 000FF700 000FF710 000FF720 000FF730 000FF740 000FF750	t.bin 00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	1 02 0 00 0 00 0 00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00 00 00 00	04 00 03 00 03 00 00 00 00	05 00 00 00 00 5D 00 00 00	06 00 00 00 00 00 00 00 00	07 00 00 00 00 00 00 00 00	08 00 40 00 00 00 00 00 00 00 00 00 00	09 00 00 F7 00 00 00 00 00 5D	0A 00 00 FF 00 00 00 00 00	0B 00 00 07 00 00 00 00 00	00 00 00 00 00 01 74 00 00	0D 00 00 00 00 00 F7 00 00	0E 31 00 00 00 00 FF 00 00	0F 30 00 00 00 00 00 00 00 00	© © © © © 0123456789ABCDEF 10 0	
■ output-Sor ADDRESS 000FF6E0 000FF6F0 000FF700 000FF710 000FF720 000FF730 000FF740 000FF750 000FF760	t.bin 00 0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	1 02 0 00 0 00 0 00 0 00 0 00 0 00 0 00	03 00 00 00 00 00 00 00 00 00	04 00 03 00 09 00 00 00	05 00 00 00 50 00 00 00 00 00	06 00 00 00 00 00 00 00 00	07 00 00 00 00 00 00 00 00 00	08 00 00 40 00 00 04 00 04 00 00	09 00 F7 00 00 00 00 00 50 00	0A 00 00 FF 00 00 00 00 00 00	0B 00 00 00 00 00 00 00 00 00	0C 00 00 00 00 01 74 00 00 00	0D 00 00 00 00 00 F7 00 00 00	0E 31 00 00 00 00 FF 00 00 00	0F 30 00 00 00 00 00 00 00 00 00	© © © © © 0123456789ABCDEF 10 0	3

図 4.2.6: Sort メモリデータ入出力

Sosu.bin			• 8
ADDRESS 00 01 02 0	<u>3 04</u> 05 06 07 08 09 0	0A 0B 0C 0D 0E 0F 0123456789AB0	DEF ^
000FF650 00 00 00	0 00 00 00 00 00 00 00 0	00 00 <u>00 00 00 00 </u>	
000FF660 00 00 00 0) 00 00 00 00 🛄 00 0	00 00 <u>00 00 00 00</u>	
000FF670 00 00 00 0) 00 00 00 00 00 00 00 0		
000FF680 00 00 00 00 00			
			••••
			•••
🖻 output-Sosu.bin			
ADDRESS 00 01 02 0	<u>3 04 05 06 07 08 09 0</u>	0A 0B 0C 0D 0E 0F 0123456789AB0	DEF ^
000FF650 00 00 00	<u>4 37</u> 00 00 00 00 00 0	00 00 00 00 00 0047	··· 2
000FF660 00 00 00 0	0 00 00 00 00 00 00 00 00 0	$00 \ 00 \ A4 \ F6 \ FF \ 07 \ \dots$	
000FF670 00 00 00 0			
000FF680 00 00 00 00 00			J
000FF6A0 01 00 00 0	0 20 34 37 00 00 00 C	00 00 00 00 00 00 00 47	

図 4.2.7: Sosu メモリデータ入出力

Himeno.bi	n															
ADDRESS	00 (01 02	03	04	05	06	07	08	09	0A	0B	00	0D	0E	0F	0123456789ABCDEF
000FF710	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF720	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF730	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF740	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF750	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF760	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF770	00	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000FF780	00 (00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	
and output-Hir	neno.l	bin														
Detroit output-Hin	neno.t	bin 01 02	03	04	05	06	07	08	09	0A	OB	00	OD	0E	0F	0123456789ABCDEF
ADDRESS	neno.t	bin 01 02 00 00	03 31	04 30	<mark>05</mark> 00	<mark>06</mark> 00	<mark>07</mark> 00	<mark>08</mark> 00	<mark>09</mark> 00	<mark>0A</mark> 00	0B 00	0C 00	0D 00	0E 00	0F 00	0123456789ABCDEF
ADDRESS 000FF710 000FF720	neno.t	bin 01 02 00 00 00 00	03 31 00	04 30 00	05 00 00	06 00 00	<mark>07</mark> 00 00	08 00 00	<mark>09</mark> 00 00	<mark>0A</mark> 00 00	0B 00 00	0C 00 64	0D 00 F7	0E 00 FF	0F 00 07	0123456789ABCDEF
	neno.l 00 (00 (00 (bin 01 02 00 00 00 00 00 30	03 31 00 00	04 30 00 00	05 00 00 00	<mark>06</mark> 00 00 00	07 00 00 00	08 00 0 <u>0</u> 00	<mark>09</mark> 00 00 00	<mark>0A</mark> 00 00 00	0B 00 00 00	00 00 64 00	0D 00 F7 00	0E 00 FF 00	0F 00 07 00	ロロン 0123456789ABCDEF 10d 靂<
	neno.t	bin 01 02 00 00 00 00 00 30 00 00	03 31 00 00 00	04 30 00 00 00	05 00 00 00 00	06 00 00 00 00	07 00 00 00 00	08 00 0A 00 08	09 00 00 00 00 5D	0A 00 00 00 00	0B 00 00 00 00	0C 00 64 00 0A	0D 00 F7 00 00	0E 00 FF 00 00	0F 00 07 00 00	□ □ ∞ 0123456789ABCDEF 10d 虚<
■ output-Hin ADDRESS 000FF710 000FF720 000FF730 000FF740 000FF750	neno.l 00 (00 (00 (00 (00 (bin 01 02 00 00 00 00 00 30 00 00 00 00	03 31 00 00 00 00	04 30 00 00 00 00	05 00 00 00 00 00	06 00 00 00 00 00	07 00 00 00 00 00	08 00 0A 00 08 00	09 00 00 00 5D 00	0A 00 00 00 00 01 00	0B 00 00 00 00 00	00 00 64 00 0A 18	0D 00 F7 00 00 00	0E 00 FF 00 00 00	0F 00 07 00 00 00 00	ロロン 0123456789ABCDEF 10d 虚く
Imit output-Hin ADDRESS 000FF710 000FF720 000FF720 000FF730 000FF740 000FF750 000FF760	neno.l	bin 01 02 00 00 00 00 00 00 00 00 00 00 00 00	03 31 00 00 00 00 00	04 30 00 00 00 00 70	05 00 00 00 00 00 72	06 00 00 00 00 00 6F	07 00 00 00 00 00 00 63	08 00 0A 00 08 00 65	09 00 00 00 5D 00 73	0A 00 00 00 00 01 00 73	0B 00 00 00 00 00 00 20	0C 00 64 00 0A 18 65	0D 00 F7 00 00 00 6E	0E 00 FF 00 00 01 64	0F 00 07 00 00 00 00 00	ロロズ 0123456789ABCDEF 10d 虚く
Imit output-Hin ADDRESS 000FF710 000FF720 000FF720 000FF730 000FF740 000FF750 000FF760 000FF770	neno.l 00 (00 (00 (00 (00 (00 (00 (00	bin 01 02 00 00 00 00 00 00 00 00 00 00 70 65	03 31 00 00 00 00 00 72	04 30 00 00 00 00 70 66	05 00 00 00 00 00 72 6F	06 00 00 00 00 00 6F 72	07 00 00 00 00 00 63 6D	08 00 04 00 08 00 65 61	09 00 00 00 5D 00 73 6E	0A 00 00 00 01 00 73 63	0B 00 00 00 00 00 00 20 65	0C 00 64 00 0A 18 65 20	0D 00 F7 00 00 00 6E 69	0E 00 FF 00 00 01 64 6E	0F 00 07 00 00 00 00 00 20	ロロズ 0123456789ABCDEF 10d 靂く

図 4.2.8: Himeno メモリデータ入出力

4.3 PCIe 通信の性能評価指数出力検証

実装した PCIe 通信と動作検証用のプログラムを使用し,性能評価指数がプロセッサの状態にかかわらず出力できることを確認する.また,PCIe 通信を用いた性能評価指数の出力結果と ISE14.7 のシミュレーションの出力結果を比較し,動作を確認する.以降の図 4.3.1 から図 4.3.8 に各プログラムごとの性能評価指数の出力を示す.

各プログラムに対する PCIe の出力結果と ISE14.7 のシミュレーション結果を比較すると,各所 に一致もしくは類似する数値が入っていることから PCIe 通信によって正しく性能評価指数が出力 されていることがわかる.出力結果を比較して数値が違う箇所は動作環境の違いによるものであ ると考えられる.シミュレーション検証に比べてエミュレーション検証が実機に近い環境である ことやシミュレーションとエミュレーションによる不定値の扱いなどの点から PCIe 通信側の出力 結果がより正しい出力と考えられる.出力結果の行番号と詳細内容を以下の表 4.1 にまとめる.

行番号	出力詳細
0	ボードサイクル数
1,2	実行サイクル数
3,4	実行サイクル数 – システムコールサイクル数
5,6	ARM 命令実行数
7	命令キャッシュ/ヒット数
8	命令キャッシュ/ミス数
9	データキャッシュ/ヒット数
10	データキャッシュ/ミス数
13	プロセッサリセット
14	シリアル通信 on/off
15	プログラム終了フラグ

表 4.1: PCIe 通信出力詳細

GPIOs
[0] 29fbf04b
[1] 00000000
[2] 0000242c
[3] 00000000
[4] 0000024e
[5] 00000000
[6] 00000014
[7] 0000003e
[8] 00000007
[9] 000000b
[10] 00000006
[11] 00000000
[12] 00000000
[13] 00000000
[14] 00000001
[15] 00000001

図 4.3.1: PCIe 出力/Hillo

GPIOs
[0] 246b4b1f
[1] 00000000
[2] 000215ea
[3] 00000000
[4] 000133fb
[5] 00000000
[6] 000034c9
[7] 00003f0d
[8] 000000c7
[9] 000013c4
[10] 0000027c
[11] 00000000
[12] 00000000
[13] 00000000
[14] 00000001
[15] 00000001

図 4.3.2: PCIe 出力/Gusu

GPIOs
[0] 6aba2ca5
[1] 00000000
[2] 00002d1a
[3] 00000000
[4] 00000dc2
[5] 00000000
[6] 0000011b
[7] 000000e4
181 00000036
191 00000061
[10] 0000000e
[11] 00000000
[12] 00000000
[13] 00000000
[14] 00000000
[15] 00000001

図 4.3.3: PCIe 出力/Foradd

GPIOs
[0] 05127fc6
[1] 00000000
[2] 00007eed
[3] 00000000
[4] 00003bc7
[5] 00000000
[6] 0000096a
[7] 00000aba
[8] 00000049
[9] 000003a1
[10] 00000072
[11] 00000000
[12] 00000000
[13] 00000000
[14] 00000000
[15] 00000001

図 4.3.4: PCIe 出力/Fibonacci

GPIOs
[0] c2b97296
[1] 00000000
[2] 0000a7c1
[3] 00000000
[4] 000050bf
[5] 00000000
[6] 00000c4c
[7] 00000e0b
[8] 00000049
[9] 0000051c
[10] 00000093
[11] 00000000
[12] 00000000
[13] 00000000
[14] 00000001
[15] 00000001

図 4.3.5: PCIe 出力/FizzBuzz

[0] ab2824c9 [1] 00000000 [2] 0000c913 [3] 00000000 [4] 00006eb4 [5] 00000000 [6] 00001364 [7] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 00000002 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001	GPIOs
[1] 00000000 [2] 0000c913 [3] 00000000 [4] 00006eb4 [5] 00000000 [6] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001	[0] ab2824c9
[2] 0000c913 [3] 00000000 [4] 00006eb4 [5] 00000000 [6] 00001364 [7] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 00000002 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001	[1] 00000000
[3] 00000000 [4] 00006eb4 [5] 00000000 [6] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 00000002 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001	[2] 0000c913
[4] 00006eb4 [5] 00000000 [6] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001	[3] 00000000
[5] 00000000 [6] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001	[4] 00006eb4
[6] 00001364 [7] 000015d6 [8] 00000070 [9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[5] 00000000
[7] 000015d6 [8] 00000070 [9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[6] 00001364
[8] 00000070 [9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[7] 000015d6
[9] 000006b4 [10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[8] 00000070
[10] 000000e2 [11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[9] 000006b4
[11] 00000000 [12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[10] 000000e2
[12] 00000000 [13] 00000000 [14] 00000001 [15] 00000001	[11] 00000000
[13] 00000000 [14] 00000001 [15] 00000001	[12] 00000000
[14] 00000001	[13] 00000000
[15] 00000001	[14] 00000001
	[15] 00000001

図 4.3.7: PCIe 出力/Sosu

GPIOS
[0] 565bebaf
[1] 00000000
[2] 0000fd90
[3] 00000000
[4] 000083e4
[5] 00000000
[6] 000017de
[7] 00004799
[8] 00000095
[9] 000008e6
[10] 00000102
[11] 00000000
T121 00000000
[13] 00000000
[14] 00000001
[15] 00000001

図 4.3.6: PCIe 出力/Sort

GPIOs	
[0]	14cf44bc
[1]	000000000
[2]	000d125e
[3]	00000000
[4]	000cc13b
[5]	000000000
[6]	0004aecb
[7]	0005c297
[8]	00000446
[9]	00008b4c
[10]	000001c9
[11]	00000000
[12]	00000000
[13]	00000000
[14]	00000001
[15]	00000001

図 4.3.8: PCIe 出力/Himeno

4.4 性能評価

第一に今回実装した FPGA プラットフォームが十分な拡張性を持っていることを確認する.論 理合成の結果からプラットフォームの各機構の動作周波数と使用リソース量を以下の表4.2 に示す

	動作周波数	BRAM	LUTs	Registers
Processor	5MHz	63%	34%	6%
Serial	10MHz	1%	1%	1%
PCIe	100MHz	3%	1%	1%

表 4.2: 動作周波数と使用リソース量

この結果から新たに機能拡張を行うのに十分なリソース量が残されていることがわかる.

第二に作成したプラットフォームのエミュレーションにかかる時間を評価する.まず各プログラムの FPGA 実行時間は以下の式によって得ることができる.

実行時間 *T*[s] = サイクル数 プロセッサの動作周波数 [Hz]

なお,サイクル数は前述の PCIe 出力結果から確認でき,動作周波数はプロセッサ設計時に行う設定から 5[MHz] であるため, FPGA 実行時間は下式のようになる

実行時間 T[s] = サイクル数×200[ns]

次にソフトウェアシミュレータのシミュレーションにかかる時間を評価する. OSIM でシミュ レーション開始から終了までの時間を10回測定し,その平均をOSIM 実行時間とする. 以上か ら求められる各プログラムの FPGA 実行時間とOSIM 実行時間を表 4.3 に示す.

プログラム名	FPGA[ms]	OSIM[ms]
Hillo	1.86	35500
Gusu	27.40	36335
Foradd	2.31	35562
Fibonacci	6.50	36651
FizzBuzz	8.59	37133
Sort	13.0	36674
Sosu	10.3	36841
Himeno	171	48206

表 4.3: 実行時間比較

この結果から FPGA の実行時間が OSIM の実行時間に比べて最小で 282 倍,最大で 19086 倍高速 であることがわかる.また論理合成の結果から FPGA プラットフォーム全体の動作周波数 5[MHz] に対して,最大動作周波数は 24.639[MHz] とであるため,今回の回路構成でも更に約 5 倍の高速 化が望める.

第5章 結論

5.1 本研究のまとめ

本稿ではカスタムプロセッサ構築用 FPGA プラットフォームの実現を最終的な目的として FPGA プラットフォームの実装と評価を行った.命令セットには組み込み用プロセッサの業界標準であ る ARM 命令セットを採用した.

第一に, XIlinx Design Suite14.7 のシミュレータである ISim を使用し,設計したカスタムプロ セッサが想定通りに実装できていることを検証した.

第二に,シミュレータ上で単純な文字の出力のみだけでなく,数字の出力・単純な演算・繰り返し処理・複雑な演算・それらの組み合わせなどの様々なプログラムに対して設計したカスタムプロセッサが動作可能であることを確認した.

第三に,シリアル通信でメモリへの読み書きする機構を追加し,直接メモリに読み書き可能で あること,またシリアル通信を含めて FPGA 上で様々なプログラムに対してカスタムプロセッサ が正しく動作することを確認した.

第四に,より高速な通信方法として PCIe 通信を追加することで,カスタムプロセッサの状態に依らず,かつ高速に性能評価指数を出力できることを確認した.

最後に, FPGAの使用リソースの観点から十分な拡張性を持っていることを検証した.またシ ミュレータと FPGAのそれぞれで,設計した回路のプログラム実行にかかる時間を求めた.その 結果,拡張に対して十分なリソースが確保されていること,シミュレータを使用した動作検証と 比べて,FPGAを使用した動作検証の方がおよそ数百倍~数万倍高速であることを確認した.これ はプロセッサアーキテクチャを研究する上で FPGAを用いた高速化が有用であることを意味する.

これらの結果からこの FPGA プラットフォームは拡張性,高速動作性,様々なプログラムを動作させることができるプロセッサを保持し,カスタムプロセッサ構築用のプラットフォームとして有用であることが確認できた.

5.2 今後の課題

第一に,シミュレータと FPGA のそれぞれで設計した回路の実行時間を比較しているが,今回 はプログラムの実行時間のみの比較であり,通信時間を除外している.理由はシリアル通信では データのやり取りに膨大な時間がかかるためである.その改善の一環として PCIe による性能評価 指数の検出を行っているが,メモリへの読み書きは依然としてシリアル通信によって行われてい る.より高速な動作のために,シリアル通信で行われているメモリへの読み書きを PCIe 通信,若 しくは他の高速な通信方法に置き換えること.

第二に, PCIe の IPcore と作成したプロセッサの伝送遅延と動作周波数の関係からプロセッサ側の動作周波数を非常に押さえている.そのため, FPGAの実行時間が非常に遅くなっている.なので,プロセッサ本体の回路構成の変更や IP コアを使わず通信方法を確立することでプロセッサを含む全体の性能を向上させること.

第三に,今回の検証では FPGA 上の動作検証と通信方法の確立は行っているが他の動作検証評価は行っていない.そこで,現回路構成以外の新しい回路構成を実装し,正しく動作するかを確認すること.

以上の三点が今後の課題としてあげられる.

謝辞

本研究を進めるにあたり,ご指導をいただいた修士論文指導教員の吉永努教授,吉見真聡助教 に感謝いたします.また常日頃から助言や励まし,心配のお言葉をいただいた吉永・入江研究室 の皆様に感謝します.

参考文献

- [1] 小島知也, 中島康彦. OROCHI 評価用集中命令ウィンドウ型スーパスカラの設計. 奈良先端科 学技術大学院大学情報科学研究科, 2006.
- [2] 吉村和浩, 中田尚, 中島康彦. 異種命令 SMT プロセッサ OROCHI の実装と分析. 奈良先端科学 技術大学院大学情報科学研究科, 2008.
- [3] 山原幹雄,中田尚,中島康彦. 異種命令混在実行プロセッサにおけるプロセススケジューリン グ手法. 奈良先端科学技術大学院大学,2008.
- [4] 小島知也. 異種命令セット同時実行プロセッサ OROCHI における命令分解機構の設計と評価. 奈良先端科学技術大学院大学情報科学研究科, 2007.
- [5] 市來亮人. SMT プロセッサ向けの正確な記憶下位層モデルの構築. 奈良先端科学技術大学院 大学情報科学研究科, 2009.
- [6] 若杉祐太, 坂口嘉一, 三好健文, 吉瀬謙二. CoreSymphony アーキテクチャの高効率化. 東京工 業大学大学院情報理工学研究科, 2009.
- [7] 坂口嘉一, 松村貴之, 永塚智之, 吉瀬謙二. CoreSymphony 実現に向けたコアアーキテクチャの 検討. 東京工業大学大学院情報理工学研究科, 2011.
- [8] 永塚智之, 坂口嘉一, 松村貴之, 吉瀬謙二. CoreSymphony の実現に向けた高性能フロントエン ドアーキテクチャ. 東京工業大学工学部情報工学科, 2011.
- [9] 坂口嘉一. CoreSymphony アーキテクチャの実装に関する研究. 東京工業大学大学院情報理工 学研究科, 2012.
- [10] 上野貴廣. CoreSymphony における命令ステアリングの高性能化. 東京工業大学工学部情報工 学科, 2012.
- [11] Cadence. Palladium XP. http://www.cadence.co.jp/topics/2010/palladium_xp.html.
- [12] Sohwa&Sophia Technologies. DS-5. http://www.ss-technologies.co.jp/service/arm/ds5/index.html.
- [13] ALDEC. HES-DVM. http://www.aldec.com/jp/products/emulation/hes-dvm.
- [14] 鈴木章浩, 及川修一. ARM アーキテクチャ用仮想マシンモニタの実装. 筑波大学, 2010.
- [15] 高橋清隆, 柴山悦哉. 組込みシステム向けマルチコア・プロセッサのためのソフトウェア開発 支援. 東京工業大学大学院情報理工学研究科, 2007.
- [16] 小林優. ハードウェア記述言語の速習 & 実践 入門 Verilog-HDL 記述. CQ 出版社, 2001.

- [17] David Money Harris, Sarah L Harris. Digital Design and Computer Architecture. 翔泳社, 2009.
- [18] Steve Furber. ARM プロセッサ. CQ 出版社, 1999.
- [19] ARM Information Center. RealView Compilation Tools アセンブラガイド, 2014.
- [20] 中野巧. VHDL によるマイクロプロセッサ設計入門. CQ 出版社, 2002.
- [21] 三好健文. Inter face ソフトウェア技術者のための FPGA 入門. CQ 出版社, 2009.

ソースコード 5.1: Himeno

```
#include <stdio.h>
1
   #define MIMAX 9
2
3
   #define MJMAX 9
   #define MKMAX 17
4
5
  double second();
6
7
  float jacobi();
   void initmt();
8
   double fflop(int,int,int);
9
   double mflops(int,double,double);
10
11
   static float p[MIMAX][MJMAX][MKMAX];
12
   static float a[4][MIMAX][MJMAX][MKMAX],
13
14
                 b[3][MIMAX][MJMAX][MKMAX],
                  c[3][MIMAX][MJMAX][MKMAX];
15
   static float bnd[MIMAX][MJMAX][MKMAX];
16
   static float wrk1[MIMAX][MJMAX][MKMAX],
17
                  wrk2[MIMAX][MJMAX][MKMAX];
18
19
   static int imax, jmax, kmax;
20
   static float omega;
21
22
  int main(){
23
24
      int i,j,k,nn;
      float gosa;
25
      double cpu,cpu0,cpu1,flop,target;
26
27
      target = 60.0;
28
      omega= 0.8;
29
30
      imax = MIMAX-1;
      jmax = MJMAX-1;
31
32
      kmax = MKMAX-1;
33
34
      /*
35
       * Initializing matrixes
       */
36
      initmt();
37
      printf("mimax_=_%d_mjmax_=_%d_mkmax_=_%d\n",MIMAX, MJMAX, MKMAX);
38
      printf("imax_=_%d_jmax_=_%d_kmax_=%d\n",imax,jmax,kmax);
39
40
```

```
nn= 10;
41
       printf("_Start_rehearsal_measurement_process.\n");
42
       printf("_Measure_the_performance_in_%d_times.\n\n",nn);
43
44
       gosa= jacobi(nn);
45
46
       flop= fflop(imax,jmax,kmax);
47
       printf("measurement_process_end\n");
48
       return (0);}
49
50
51
   void initmt(){
       int i,j,k;
52
53
54
       for (i=0 ; i<MIMAX ; i++)
55
         for (j=0 ; j<MJMAX ; j++)
            for (k=0 ; k<MKMAX ; k++){
56
57
              a[0][i][j][k]=0.0;
              a[1][i][j][k]=0.0;
58
              a[2][i][j][k]=0.0;
59
              a[3][i][j][k]=0.0;
60
              b[0][i][j][k]=0.0;
61
              b[1][i][j][k]=0.0;
62
              b[2][i][j][k]=0.0;
63
              c[0][i][j][k]=0.0;
64
65
              c[1][i][j][k]=0.0;
              c[2][i][j][k]=0.0;
66
              p[i][j][k]=0.0;
67
68
              wrk1[i][j][k]=0.0;
              bnd[i][j][k]=0.0;}
69
70
       for (i=0 ; i<imax ; i++)
71
         for (j=0 ; j<jmax ; j++)
72
            for (k=0 ; k<kmax ; k++){
73
              a[0][i][j][k]=1.0;
74
              a[1][i][j][k]=1.0;
75
              a[2][i][j][k]=1.0;
76
77
              a[3][i][j][k]=1.0/6.0;
78
              b[0][i][j][k]=0.0;
79
              b[1][i][j][k]=0.0;
              b[2][i][j][k]=0.0;
80
81
              c[0][i][j][k]=1.0;
              c[1][i][j][k]=1.0;
82
              c[2][i][j][k]=1.0;
83
              p[i][j][k] = (float)(i*i)/(float)((imax-1)*(imax-1));
84
              wrk1[i][j][k]=0.0;
85
86
              bnd[i][j][k]=1.0;
            }}
87
88
   float jacobi(int nn){
89
90
       int i,j,k,n;
```

```
float gosa, s0, ss;
91
92
        for (n=0 ; n<nn ; ++n){
93
94
           gosa = 0.0;
95
        for (i=1; i < imax-1; i++)
96
           for (j=1 ; j<jmax-1 ; j++)
97
               for (k=1 ; k<kmax-1 ; k++){
98
                    s0 = a[0][i][j][k] * p[i+1][j][k]
99
                       + a[1][i][j][k] * p[i][j+1][k]
100
101
                       + a[2][i][j][k] * p[i][j][k+1]
                       + b[0][i][j][k] * (p[i+1][j+1][k] - p[i+1][j-1][k])
102
                                           -p[i-1][j+1][k] + p[i-1][j-1][k])
103
                       + b[1][i][j][k] * (p[i][j+1][k+1] - p[i][j-1][k+1]
104
105
                                         - p[i][j+1][k-1] + p[i][j-1][k-1])
                       + b[2][i][j][k] * ( p[i+1][j ][k+1] - p[i-1][j ][k+1]
106
                                         -p[i+1][j][k-1] + p[i-1][j][k-1])
107
                       + c[0][i][j][k] * p[i-1][j][k]
108
                       + c[1][i][j][k] * p[i][j-1][k]
109
                       + c[2][i][j][k] * p[i][j][k-1]
110
                       + wrk1[i][j][k];
111
112
                    ss = (s0 * a[3][i][j][k] - p[i][j][k]) * bnd[i][j][k];
113
114
115
                    gosa+= ss*ss;
116
                    wrk2[i][j][k] = p[i][j][k] + omega * ss;
               }
117
        for (i=1 ; i<imax-1 ; ++i)
118
           for (j=1 ; j<jmax-1 ; ++j)
119
               for (k=1; k < kmax - 1; ++k)
120
                  p[i][j][k] = wrk2[i][j][k];
121
        } /* end n loop */
122
        return(gosa);
123
124
    }
125
    double fflop(int mx,int my, int mz){
126
127
        return((double)(mz-2)*(double)(my-2)*(double)(mx-2)*34.0);
128
   }
```