学士論文

マルチ ALU プロセッサの並列プログラミングと動作検証

氏　　名　：　杵川　大智
学籍番号　：　2260090017-3
担当教員　：　山崎 勝弘 教授

提 出 日　：　2013 年 2 月 19 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本研究ではアセンブリ言語を用いたプログラムを作成し、本研究室のプロジェクトの一つであるマルチ ALU プロセッサ（MAP）システムの動作検証と並列性の評価を行った。今回は検証用プログラムとして、Booth（ブース）のアルゴリズムを用いて符号付き整数の乗算を行うプログラムと、与えられた 2 つの座標点（符号付き整数）を通る直線の方程式を出力するプログラムの 2 種類を用意し、シミュレータ及び FPGA ボード上での動作確認を行う。シミュレータでの検証には、あらかじめ用意されていた MAP シミュレータ・論理シミュレーションを利用する。FPGA ボード上での動作検証に際しては、あらかじめ用意されていた CUI モードと、新たに開発した GUI モードを用いての検証を行った。

本論文では、複数の ALU を有するプロセッサであるマルチ ALU プロセッサ（MAP）について解説した後､MAP の動作検証を目的として作成したアセンブリプログラムを説明し、シミュレータ及び FPGA ボード上での動作と並列性の検証を行う。

目次

図目次

表目次

1. はじめに

近年、FPGA の集積度、性能が著しく向上し、マイクロプロセッサを中心としたソフトウェアとハードウェアを搭載したシステム LSI が半導体業界の主流となっている。FPGA の集積度、性能の向上により、LSI による可能性が拡大する反面、システムの大規模化によるコストの上昇、設計の複雑化による設計期間の長期化が問題となってきており、これまでより短期間かつ低コストで信頼性高く設計・製造することが要求されている[1]。そこで、デジタル機器向けの組み込みシステム開発の現場では、多くの製品で共通して使用できるモジュールは、以前の設計資産やベンダーが提供する IP を利用し、設計の効率化を図っている。また、半導体メーカや電機メーカ各社では、予めどの製品でも使用できる汎用的なプラットフォームを用意しておくプラットフォームベース設計の手法を提案している。LSI 設計においては、これらの手法を用いることで、ソフトウェア設計量の削減と開発期間の短縮を実現しながらも、要求を満たす組み込みシステムの開発を行っているが、この手法は、ハードウェアとソフトウェアの性能のトレードオフを完全に理解していることが前提となっている[2]。

このような背景から、大学教育にはハードウェアとソフトウェアを深く理解し、その上でシステム設計を提案できる人材を育成することが求められている。本研究室では、プロセッサ設計を中心として、ハードとソフトの両方の知識を習得するためのハード/ソフト協調学習システム（HSCS）の研究を進めており、複数の ALU による並列処理が可能なマルチ ALU プロセッサ（MAP）が開発されている。MAP システムは、HSCS を用いて設計された 32 ビットのプロセッサであり、複数の命令間の依存関係を判別し、並列演算または連鎖演算を行う。現在は、2 つの命令を 1 命令サイクルで同時実行する 2ALU による MAP が設計され、さまざまな処理を行うプログラムを実行した場合でも正しい動作を行えるかどうかを検証する段階である。

そこで、本研究ではアセンブリプログラムを作成し、本研究室で開発された MAP システムの動作検証と並列性の評価を行う。今回は検証用プログラムとして、Booth（ブース）のアルゴリズムを用いて符号付き整数の乗算を行うプログラムと、与えられた 2 つの座標点（符号付き整数）を通る直線の方程式を出力するプログラムの 2 種類を用意し、シミュレータ及び FPGA ボード上での動作確認を行う。また、FPGA ボード上への MAP の実装を行う際には、本研究と関連して新たに開発したプロセッサデバッガ用 GUI の動作の確認も兼ねて、従来の CUI を用いた方法に加えて、GUI を用いて正しく実行できるかどうかの検証も行う。今回開発した GUI は、従来の CUI によるデバッガのコマンドライン制御をマウス操作によって行えるように改良したものである。ボード上に MAP の実装を行ってプログラムを動作させる際に GUI を用いることで、正しくデバッガを制御できることを確認する。

2. マルチ ALU プロセッサ（MAP）システム[3][4][5][6]

2.1 MAP の設計方針

MAP は複数 ALU を有する 32 ビットのプロセッサである。複数 ALU による並列処理が可能で、並列演算と連鎖演算を行う。現在は ALU を 2 個搭載した MAP が設計されており、プログラムから 2 命令ずつ取り出して実行する。図 1 に MAP のブロック構成を示す。

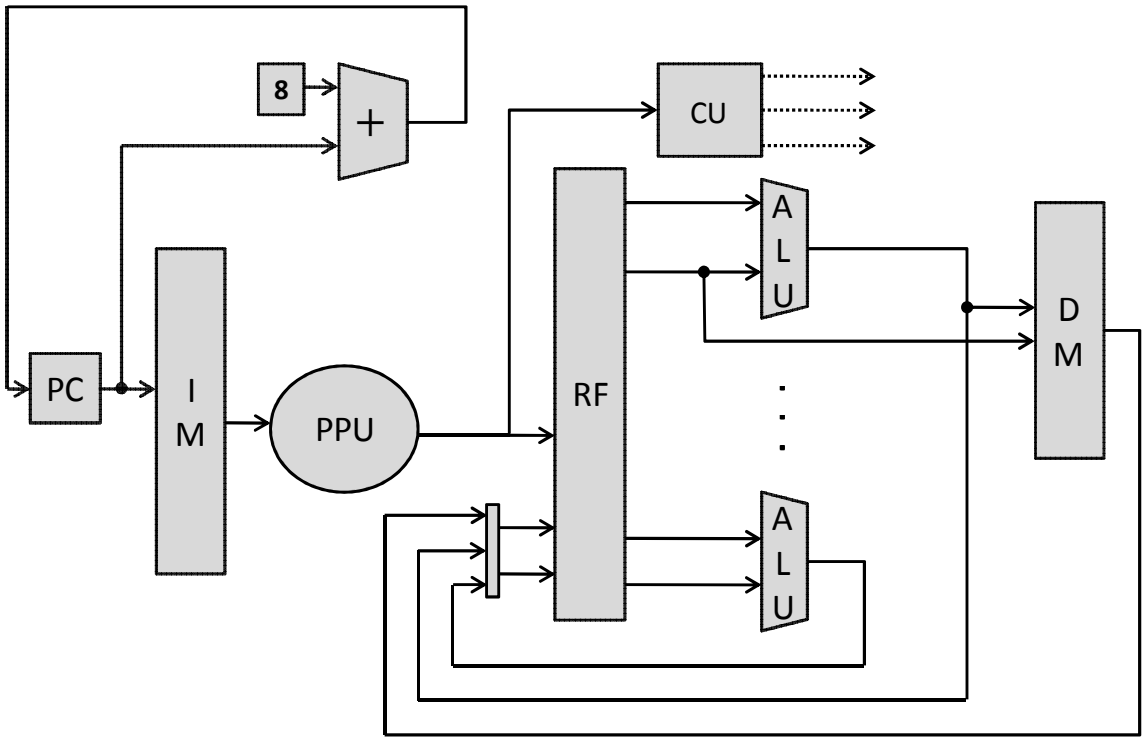

図 1：MAP のブロック構成

PPU(Parallel Processing Unit)は並列実行を判断するユニットである。PPU では IM(命令メモリ)からフェッチした 2 つの演算のオペコードを読み取り、2 演算が単一実行しか動作できないと判断した場合は 1 演算のみ動作させ、並列実行可能と判断した場合には連鎖判定にフェッチした 2 演算を送る。そして PPU での並列・連鎖の判定をもとにして 2 つの ALU が 1 命令サイクルで 2 つの演算を実行する。

2.2 命令セットアーキテクチャ

MAP には、Register 形式（R 形式）、Immediate 形式（I 形式）、Long 形式（L 形式）、Jump 形式（J 形式）、の 4 つの命令形式が用意されている。R 形式にはレジスタ間の演算を行う命令を定義している。I 形式にはレジスタ値と即値演算を行う命令を定義している。L 形式には条件分岐命令とメモリ・レジスタへのデータ転送命令を定義している。J 形式には無条件分岐命令の JUMP、プログラム終了命令の HALT を定義している。R 形式では、Fn で命令を詳細に識別しているため、Op1 つに対して 64 種類まで拡張が行える。L 形式と J 形式では DM へのアクセスと PC へ JUMP する範囲を多く取ることにより、従来のプロセッサにはできなかった大規模な計算を行うことができる。表 1 に MAP の命令形式を、表 2 に命令セット内容を示す。

表 1：MAP の命令形式

命令語長	32					
命令形式	6	5	5	5	5	6
R 形式	Op	Rs	Rt	Rd	Shamt	Fn
I 形式	Op	Rs	Rd	Imm		
L 形式	Op	Rs	Rd	Address/immediate		
J 形式	Op	address				

表 2：命令セット内容

R 形式	ADD, SUB, AND, OR, XOR, NOT, NOP, SLT, SGT, SLE, SGE, SEQ, SNE, SLL, SRL, SRA, JR
I 形式	ADDI, SUBI, ANDI, ORI, XORI, SLTI, SLEI, SGEI, SEQI, SNEI, LDHI, LDHI, LDLI
L 形式	BEQZ, BNEZ, LD, ST, JAL
J 形式	JUMP, HALT

命令フィールドの意味を以下に示す。

・Op　：命令を識別
・Rs　：演算レジスタ
・Rt　：演算レジスタ
・Rd　：演算結果を格納するレジスタ
・Shamt：シフト量
・Fn　：R 形式の命令を詳細に識別
・Imm　：アドレスと即値

2.3 並列演算と連鎖演算

MAP には、表 2 に示すような 37 種類の命令が存在する。これらの命令を、プログラマが 1 演算ずつ順に並べて逐次プログラムを記述する。プログラマは、連鎖・並列性を意識することなくプログラムを記述し、MAP が実行時に 2 演算ずつフェッチして連鎖演算、並列演算、単一実行を判定する。図 2 に MAP プログラム例を示す。

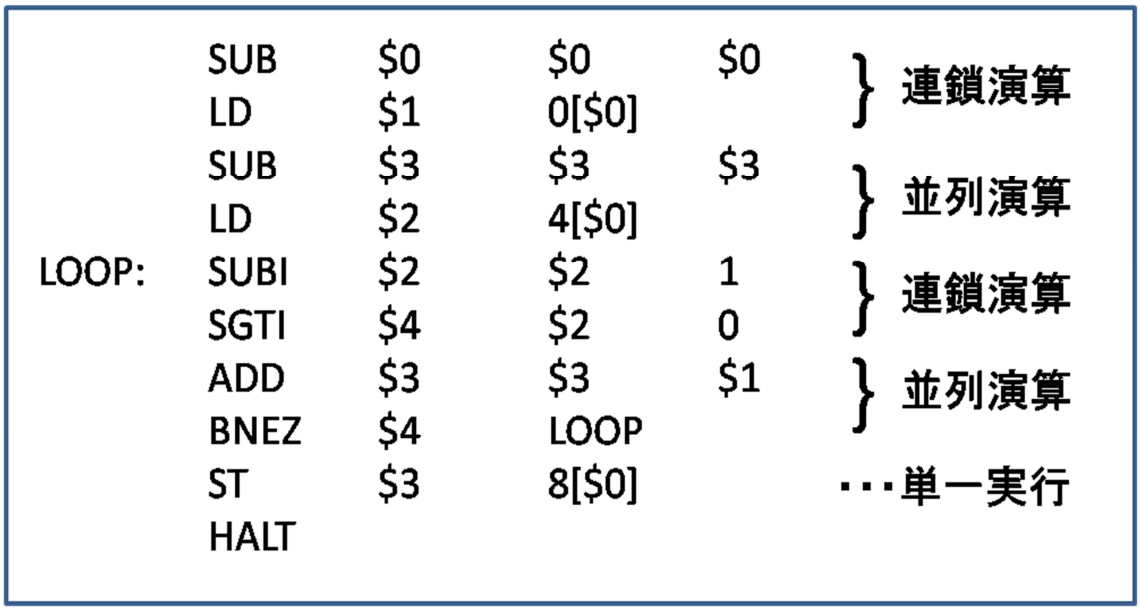

図 2：乗算のプログラム例

MAP を用いた並列実行には、2 つの ALU 間でデータ依存関係がなく独立した演算を行う場合の並列演算と、データ依存関係があり連鎖させて演算を行う場合の連鎖演算がある。並列演算は、互いにデータの依存関係がない演算を 2 つの ALU によって 1 命令サイクルで同時実行する。連鎖演算では、ある ALU での演算結果を他の ALU の入力とすることにより、依存関係のある 2 つの演算を 1 命令サイクルで実行する。

2.3.1　ALU 並列演算

ALU 並列演算は、データ依存関係のない 2 つの演算を、1 命令サイクルで同時実行する。命令メモリからフェッチしてきた 2 演算で命令の判別を行い、並列実行できるか判断する。また、2 演算のオペランドに依存関係がないか判断し、なければ並列演算を行う。図 3 に並列演算のデータパスを示す。

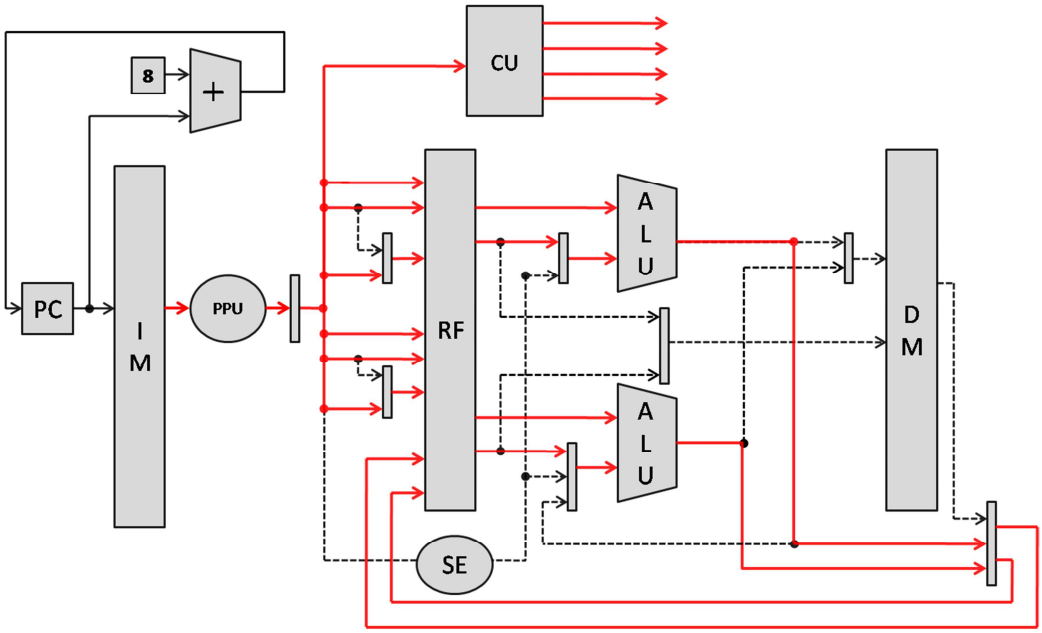

図 3：並列演算のデータパス

2.3.2　ALU 連鎖演算

ALU 連鎖演算とは、ALU の結果を次の命令で処理を行う手順がある場合において、同時にその処理を行おうとするものである。ALU の結果をデータ依存関係のある命令の場合において、次の命令まで待機させることなく同時に演算を行うことが可能である。ある ALU での演算結果を他の ALU の入力とすることにより、データ依存関係のある 2 つの演算を 1 命令サイクルで実行する。図 4 に連鎖演算のデータパスを示す。

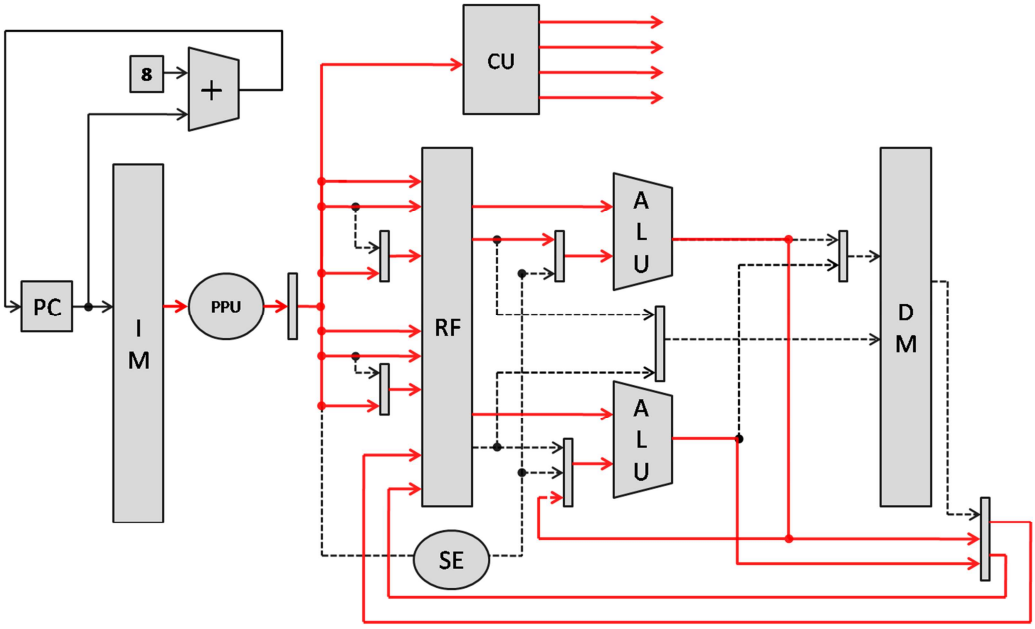

図 4：連鎖演算のデータパス

3. MAP の並列プログラミング

3.1 Booth の乗算

3.1.1 Booth の乗算のアルゴリズム

Booth の乗算のアルゴリズムは、2 値 A、Sのうちの 1 つを積 Pに繰り返し加算し、Pを右方向に算術シフトするという形で実装できる。ここで、m を被乗数、r を乗数とし、m のビット数を x、r のビット数を yとする。

1. Aと S の値を決定し、Pを初期状態にする。いずれも長さは $(x+y+1)$ ビットとする。

A) A: 最上位のビット列に m の値を格納する。残る $(y+1)$ ビットは全て 0 にする。
B) S: 最上位のビット列に (−m) の 2 の補数表現を格納する。残る $(y+1)$ ビットは全て 0 にする。
C) P: 最上位の x ビットを全て 0 にする。その右のビット列に r の値を格納する。最下位ビットは 0 とする。

2. Pの最下位 2 ビットを調べる。

A) その中身が 01 の場合、$P+A$ を計算し、オーバーフローは無視する。
B) その中身が 10 の場合、$P+S$を計算し、オーバーフローは無視する。
C) その中身が 00 または 11 の場合、現在の Pをそのまま次ステップで使用する。

3. ステップ 2 で得られた値を右に 1 ビット算術シフトし、新たな P の値とする。

4. ステップ 2 と 3 を y回反復し、最後に Pの最下位（右端）ビットを除去する。

3.1.2 テストデータ

Booth の乗算プログラムを用いた動作検証で使用する入力値と、その値を用いた場合に出力値を表 3 に示す。

表 3：Booth の乗算で使用する入力値と正しい出力値

入力値（乗数/被乗数）		出力値（演算結果）
3	4	12
-3	4	-12

Booth の乗算アルゴリズムでは正の数・負の数を問わず乗算を行うことができるため、動作検証を行うにあたっては出力値が正の場合と負の場合の二通りについての検証を行った。今回は、負の場合については-3×4=-12 の計算を、正の場合については 3×4=12 の計算を検証用データとする。

3.2 直線の方程式

3.2.1 直線の方程式のアルゴリズム

直線の方程式のプログラムでは、x 軸-y 軸上の異なる 2 点の座標を入力値として、その 2 点を通る直線の方程式を求める。今回作成したプログラムは、求める直線の方程式を y = ax + b としたときの傾き a と切片 b を求め、符号付き整数で出力する。また、a, b が整数とならない場合にはその値を有理数$\frac{q}{p}$で表し、p, q の値を出力する。

1. x 軸-y 軸上の異なる 2 点を(x1, y1)、(x2, y2)、求める方程式を y = ax + b とする。
2. 傾き a、切片 b を、(x1, y1)、(x2, y2)を用いて a, b の順で以下のように求める。

A) $a = \frac{y2-y1}{x2-x1}$

B) $b = y1 - a \times x1$

3.2.2 テストデータ

直線の方程式のプログラムを用いた動作検証で使用する入力値と、その値を用いた場合に期待される出力値を表 4 に示す。

表 4：直線の方程式で使用する入力値と正しい出力値

入力値（座標 A）		入力値（座標 B）		出力値（傾き）	出力値（切片）
1	1	2	3	2	-1
0	2	2	1	$-\frac{1}{2}$	2

傾きと切片がそれぞれ正の場合と負の場合、整数の場合と分数の場合を検証するため、上記のような 2 通りのテストデータを用いて検証を行った。今回は、(1, 4)(3, 1)の 2 点を通る直線 $y = \frac{-3}{2}x + \frac{11}{2}$、(1, 1)(2, 3)の 2 点を通る直線 y = 2x + 1 の 2 種類を検証用データとする。なお、分数の出力値についてはアセンブリ言語の特性上、-3, 2, 11, 2 と分けて出力する。

4. シミュレーションによる動作検証

4.1 MAP シミュレータ[7]

MAP シミュレータは、アセンブリプログラムのデバッグと命令の動的並列・連鎖性を調べ、複数 ALU による並列・連鎖演算、単一実行判別の有効性を示すことを目的として設計された MAP 検証用のシミュレータである。アセンブラによってアセンブリプログラムが変換されたオブジェクトコードを入力とし、プログラムの実行を行い結果としてデータメモリに演算結果が格納される。

4.1.1 Booth の乗算

Booth の乗算を MAP シミュレータで動作させ、表 3 で示した値で実行した結果、正常な出力結果が得られていることを確認した。図 5、図 6 に、MAP シミュレータでの Booth の乗算の実行結果を示す。

```
DM[0] = 3          DM[20] = 0
DM[4] = 4          DM[24] = 0
DM[8] = 0          DM[28] = 0
DM[12] = 12        DM[32] = 0
DM[16] = 0         DM[36] = 0
```

図 5 : MAP シミュレータでの Booth の乗算の実行結果 1

```
DM[0] = -3         DM[20] = 0
DM[4] = 4          DM[24] = 0
DM[8] = 0          DM[28] = 0
DM[12] = 244       DM[32] = 0
DM[16] = 0         DM[36] = 0
```

図 6 : MAP シミュレータでの Booth の乗算の実行結果 2

データメモリの割り当てを以下に示す。

・DM[0]　：乗数（入力値）
・DM[4]　：被乗数（入力値）
・DM[12]　：演算結果（出力値）

4.1.2　直線の方程式

直線の方程式を MAP シミュレータで動作させ、表 4 で示した値で実行した結果、正常な出力結果が得られていることを確認した。図 7、図 8 に、MAP シミュレータでの直線の方程式の実行結果を示す。

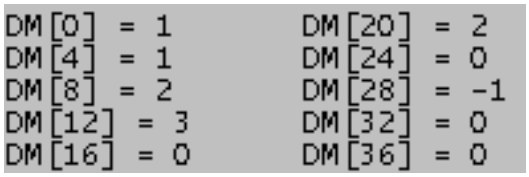

図 7 : MAP シミュレータでの直線の方程式の実行結果 1

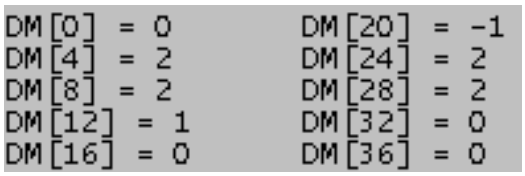

図 8 : MAP シミュレータでの直線の方程式の実行結果 2

データメモリの割り当てを以下に示す。

・DM[0], DM[4]　　：座標 A（入力値）
・DM[8], DM[12]　：座標 B（入力値）
・DM[20], DM[24]　：傾き（出力値）
・DM[28], DM[32]　：切片（出力値）

4.2 論理シミュレーション

論理シミュレーションでは論理回路の機能やタイミングを検証し、設計者が意図したとおりに論理回路が動作するかどうかの確認を行う。Xilinx 社から提供されている”ISE Design Suite”というツールを用いて MAP を動作させ、シミュレーションを行うことで出力される信号波形を読み取って演算結果を確認する。

Booth の乗算プログラムと直線の方程式のプログラムを MAP で動作させたところ、複数の ALU が動作し演算を行っていることが出力波形から確認できた。また、実行後にデータメモリに出力される値を読み取ることで、正常な出力結果が得られていることが確認できた。

4.2.1　Booth の乗算

Booth の乗算プログラムを用いて MAP の論理シミュレーションを行った。表 3 で示した値で実行した結果、正常な出力結果が得られていることを確認した。図 9、図 10 に実行後のデータメモリの内容を示す。

	0	1
0x0	00000003	00000004
0x2	00000000	0000000C
0x4	00000000	00000000

図 9：Booth の乗算プログラム実行後のデータメモリの内容 1

	0	1
0x0	11111101	00000004
0x2	00000000	11110100
0x4	00000000	00000000

図 10：Booth の乗算プログラム実行後のデータメモリの内容 2

4.2.2　直線の方程式

直線の方程式のプログラムを用いて MAP の論理シミュレーションを行った。表 4 で示した値で実行した結果、正常な出力結果が得られていることを確認した。図 11、図 12 に実行後のデータメモリの内容を示す。

	0	1
0x0	00000001	00000001
0x2	00000002	00000003
0x4	00000000	00000002
0x6	00000000	FFFFFFFF
0x8	00000000	00000000

図 11：直線の方程式プログラム実行後のデータメモリの内容 1

	0	1
0x0	00000000	00000002
0x2	00000002	00000001
0x4	00000000	FFFFFFFF
0x6	00000002	00000002
0x8	00000000	00000000

図 12：直線の方程式プログラム実行後のデータメモリの内容 2

5. FPGA ボード上での動作検証

5.1 プロセッサデバッガの概要

プロセッサデバッガは、HSCS を利用して設計したプロセッサを、実機上で動作検証するためのハードウェア IP である。本モジュールと接続して論理合成をかけることで様々なプロセッサが FPGA に実装でき、ホスト PC と通信することで実行が可能である。本モジュールが担当する処理は、ホスト PC 上のプロセッサモニタから要求される様々なデバッグコマンドを FPGA 内部で処理し、必要に応じてデータの送受信を行うことである。

本ツールは、シリアル通信を行うシリアル IF、フレームの生成・解釈を行うフレーム IF、デバッグを行うデバッガ本体、命令・データメモリと、FPGA 上で動作検証したいユーザ設計のプロセッサによって構成される。図 13 にプロセッサデバッガの構成を示す。

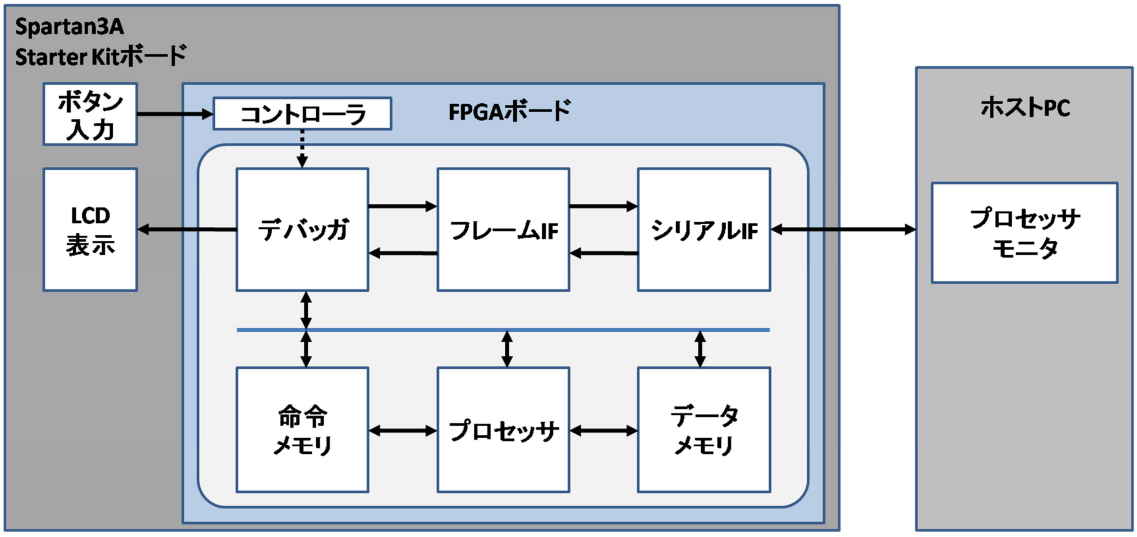

図 13：プロセッサデバッガの構成

プロセッサモニタは、ホスト PC 上からプロセッサデバッガを制御するために設計された CUI である。ユーザ側とプロセッサデバッガ側で異なるコマンド体系を扱うため、モジュールの設計において 2 つインタフェースを基軸に置く。ユーザがコマンドラインを打ち込むことによってユーザコマンドを受け付け、要求されたデバッグ処理を複数のアプリケーションコマンドによって実現する。

また今回新たに開発した GUI は、プロセッサモニタの操作の簡略化を目的として設計したものである。CUI の操作に慣れていない者でも、キーボードとマウスを用いた直感的な操作を可能にする。

5.2 CUI モードでの検証

作成したアセンブリプログラムを用いて、FPGA ボード上での MAP の動作検証を、プロセッサデバッガの CUI にて行った。図 14、図 15 に Booth の乗算プログラムの実行結果を、図 16、図 17 に直線の方程式プログラムの実行結果を示す。

```
input command > read dm 0 5
READ
input command is : 0
[送信中. 0x02 0x00 0x06 0x00 0x05 0x00 0x00 0x00 0x00 0x0b ]
Read; address: 00 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 01 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 03 0x0c 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Complete to read.
```

図 14：Booth の乗算プログラムの実行結果 1

```
input command > read dm 0 5
READ
input command is : 0
[送信中. 0x02 0x00 0x06 0x00 0x05 0x00 0x00 0x00 0x00 0x0b ]
Read; address: 00 0xfd 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 01 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 03 0xf4 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 04 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Complete to read.
```

図 15：Booth の乗算プログラムの実行結果 2

```
input command > read dm 0 10
READ
input command is : 0
[送信中. 0x02 0x00 0x06 0x00 0x0a 0x00 0x00 0x00 0x00 0x10 ]
Read; address: 00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 03 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 05 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 06 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 07 0xff 0xff 0xff 0xff 0x00 0x00 0x00 0x00
Read; address: 08 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 09 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Complete to read.
```

図 16：直線の方程式プログラムの実行結果 1

```
input command > read dm 0 10
READ
input command is : 0
[送信中. 0x02 0x00 0x06 0x00 0x0a 0x00 0x00 0x00 0x00 0x10 ]
Read; address: 00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 01 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 02 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 03 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 05 0xff 0xff 0xff 0xff 0x00 0x00 0x00 0x00
Read; address: 06 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 07 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 08 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Read; address: 09 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Complete to read.
```

図 17：直線の方程式プログラムの実行結果 2

5.3 GUI モードでの検証

プロセッサデバッガ用の GUI を新たに実装し、FPGA ボード上での MAP の動作検証を行った。図 18、図 19 に Booth の乗算プログラムの実行結果を、図 20、図 21 に直線の方程式プログラムの実行結果を示す。

図 18：Booth の乗算プログラムの実行結果 1

図 19：Booth の乗算プログラムの実行結果 2

図 20：直線の方程式プログラムの実行結果 1

図 21：直線の方程式プログラムの実行結果 2

5.4 並列性の検証

MAP で実行した結果から並列演算、連鎖演算、単一実行の回数を調べ、並列性の検証を行った。表 5 に、MAP での静的・動的時の演算数を示す。

表 5：MAP での静的時・動的時の演算数

	静的			動的					
	並列	連鎖	単一	並列	連鎖	単一	実行サイクル数	全命令数	$\frac{\text{実行サイクル数}}{\text{全命令数}}$
Booth の乗算 1	10	13	1	6	22	11	39	67	58.21%
Booth の乗算 2				7	23	11	41	71	57.75%
直線の方程式 1	17	33	6	9	10	5	24	43	55.81%
直線の方程式 2				10	6	3	19	35	54.29%

表 5 に示すように、プログラムを静的に見たときと動的に実行したときで、演算数に違いが見られる。これはプログラムにループ処理等が含まれており、動的に実行した場合にはループ処理により演算を行う回数が多くなるためだと考えられる。Booth の乗算の場合、入力値によって並列演算の数が増えており、ループ回数が増加していることが考えられる。直線の方程式の場合、入力値によって並列演算、連鎖演算、単一演算の回数が変わっているが、これは分岐命令により実行するプログラムの範囲が変わるためである。

検証用プログラムを動的に実行した場合、単一実行よりも並列演算と連鎖演算が多く行われており、命令サイクル数の削減が実現されている。また、並列実行を行うことにより、いずれの場合も演算量を 40%以上削減できていることがわかる。このことから、2ALU の MAP の有効性が評価できた。

6. おわりに

本研究では MAP の動作を検証するためのアセンブリプログラムを作成し、MAP システムの動作検証と並列性の評価を行った。動作検証では Booth の乗算のプログラム、直線の方程式のプログラムを実行し、入力した数値に対して正しい結果が得られることを確認することができた。並列性の評価では FPGA ボード上で動作させた際の並列演算、連鎖演算の回数を調査し、ALU を 2 つにすることによる演算の効率化を評価した。また、新たに開発したプロセッサデバッガ用 GUI が正しくデバッガを制御できることを確認した。この GUI は従来の CUI によるデバッガのコマンドライン制御をマウス操作によって行えるように改良したものであり、ボード上に MAP の実装を行ってプログラムを動作させる際に GUI を用いて実行したところ、デバッガの制御を正しく行えることを確認することができた。

謝辞

本研究の機会を与えてくださり、ご指導いただきました山崎勝弘教授に深く感謝します。また、本研究に関して貴重な助言、ご意見をいただきました境直樹様、孟助手、そのほか高性能計算研究室の皆様に心より感謝いたします。

参考文献

[1] 末吉敏則：「FPGA/PLD 最前線」－歴史から最新動向までー、5 都市 FPGA カンファレンス 2005 pp 9-60 2005.

[2] 難波翔一朗：プロセッサ設計支援ツールの実装とハード/ソフト強調学習システムの評価 立命館大学理工学研究科電子システム学専攻修士論文 2007

[3] 境直樹：デザインパターンを用いたマルチ ALU プロセッサの設計 立命館大学理工学部電子情報デザイン学科卒業論文 2010

[4] 境直樹・山崎勝弘：FPGA ボード上でのマルチ ALU プロセッサの設計と実装 平成 23 年度情報処理学会関西支部 支部大会 A-02

[5] 境直樹：演算レベル並列処理用マルチ ALU プロセッサの設計と実現 立命館大学理工学研究科修士論文 2013

[6] 境直樹：MAP 仕様書 2011

[7] 高松良太：マルチ ALU プロセッサにおけるシミュレータの設計と試作 立命館大学理工学部電子情報デザイン学科 2008