学士論文

マルチ ALU プロセッサのデバッガ用 GUI の設計と実現

氏　　名　：　石川　陽章
学籍番号　：　2260090005-0
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

本論文では、本研究室で開発したハード/ソフト協調学習システムのプロセッサ設計支援ツールにおけるプロセッサデバッガのユーザーインターフェースに関する問題点を検討している。本研究では、プロセッサデバッガのデバッグコマンドが適当であるかを検討し最適化した。また、従来の CUI ベースのコマンド入力の問題点を解消するためにプロセッサデバッガの GUI を設計し、その処理系作成した。さらに、研究室で開発された SARIS プロセッサと MAP プロセッサを GUI ベースのデバッガで実行し、正しく動作することを確認した。

目次

図目次

表目次

1. はじめに

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。特に、プロセッサを中心とするシステムでは、ハードウェアの内部構造からその上で動作するソフトウェアのプログラミングスキルまで、ハード/ソフト協調設計で必要な要素が集約されている。ハードウェアとソフトウェアそれぞれの性能を両立して引き上げることは、システム設計者にとって非常に重要な技術であり、両者の関係を深く理解できる学習システムの開発が渇望されている。

本研究室ではプロセッサにおけるハードウェアとソフトウェアの設計を通じて、両者のトレードオフを学習するハード/ソフト協調学習システム(HSCS)の研究を進めてきた。現在では、FPGA ボード上での実行を支援することを目的としたツールとして、プロセッサモニタとプロセッサデバッガが開発されており、本システムの学習効果の考察と評価が行われてきた[1]。また、HSCS システムを用いて、複数の ALU による並列処理可能なマルチ ALU プロセッサ(MAP)の設計を行い、FPGA ボード上での動作検証と並列性の評価を行うことを目的とする研究が進められている[2]。MAP は名前のとおり、複数の ALU を有し、32 ビットのプロセッサである。MAP では ALU 同士の並列演算と連鎖演算を可能として、高速化を図っている。

MAP を FPGA ボードに実装させた際に使用されるプロセッサモニタが CUI ベースのアプリケーションであったため様々な問題があった。コマンド入力時にキーボードからコマンドを入力しなければならない問題やデバッグコマンドを記憶していなければならない問題、コマンド入力とデータ出力が同じ画面内で文字が乱雑に表示される問題があり学生が学習する上で非常に理解しづらい画面表示となっていると判断した。本研究では、視認性、操作性に優れ、直感的な操作が可能なため、広く普及し、現在では主流のインタフェースになっている GUI を設計することでコマンドを GUI 画面上にボタンで表示しボタン操作でコマンドを入力できるようにし、コマンド入力する場所とデータが出力される場所を分けることで画面が見づらい問題を解消し学生の HSCS のよる学習効果をより向上させることを目的としている。

2. マルチ ALU プロセッサ(MAP)システム

2.1 ハード/ソフト協調学習システム

ハード/ソフト協調学習システムとは、プロセッサを通してハードウェアとソフトウェアの両方の知識を学習していく為に考案されたシステムである。 ソフトウェアを学習する面では、アーキテクチャが可変な命令セットシミュレータを用いてプロセッサのアーキテクチャの仕組みの理解し、アセンブリ言語で書かれたプログラムを評価する。ハードウェアを学習する面では、シミュレータで理解したプロセッサの知識を基に、HDL によるプロセッサ設計を行う。そして学習者が設計したプロセッサを検証、評価することによってプロセッサ設計能力を習得する。次に、ハードウェア学習の際に使用するプロセッサ設計支援ツールについて説明する。命令セット定義ツール、汎用アセンブラ、汎用シミュレータにより、命令セットを独自に定義することができる。またプロセッサモニタとプロセッサデバッガは、学習者が設計したプロセッサを FPGA ボード上で検証する際に使用する。これらのプロセッサ設計支援ツールを使用し、ハードとソフトの両方の学習を進めていくことがこのシステムの目的である。

2.2 MAP の構成と特徴

MAP は複数 ALU を有する 32 ビットのプロセッサである。命令メモリとデータメモリが分離したハーバードアーキテクチャである。図 1 に MAP の構成を示す。

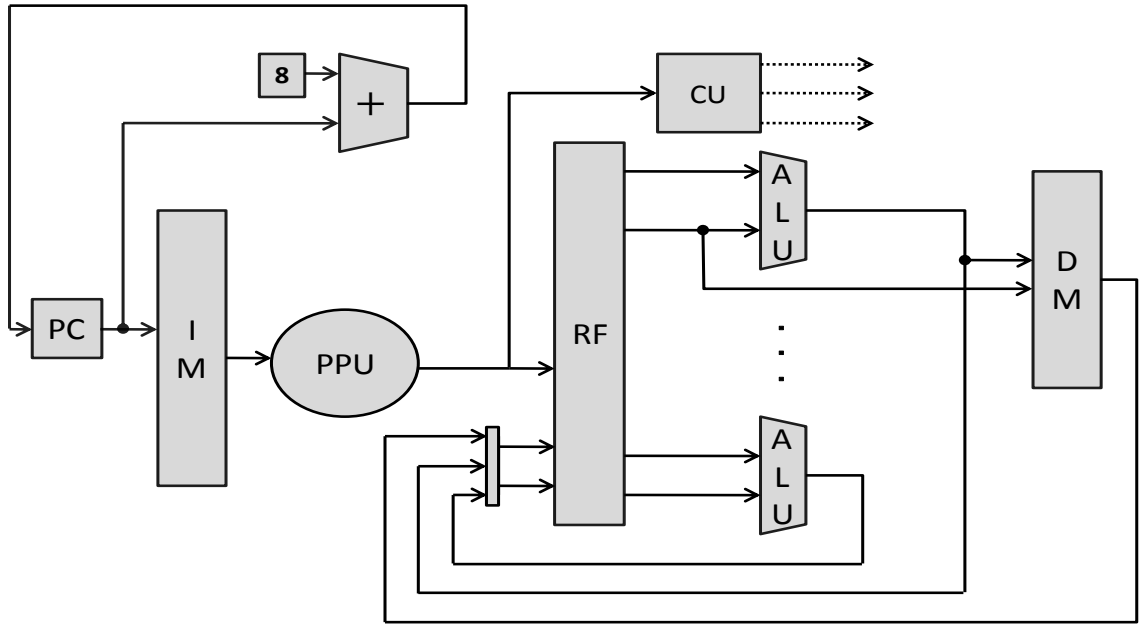

図 1　MAP の構成

MAP の特徴は以下の 3 点に要約できる。

① 複数 ALU による並列処理が可能で、ALU 数は 2,4,8

② レジスタファイルは 32 個で、全 ALU で共有

③ ALU による並列演算と連鎖演算を行う

例えば、レジスタ間演算命令では、IM から PPU を通し CU にオペコードとファンクションコードを送る。PPU では並列実行を判断する。IM から 2 つのオペランドのアドレスと、演算結果を格納するアドレスを RF に出力する。ALU は、CU から演算命令の種類を受け取り、2 つの演算データを行い、指定したアドレスに演算結果を格納する。PC は加算機によって 8 加算される。

2.3 MAP の命令セットアーキテクチャ

MAP には、Jump 形式（J 形式）、Register 形式（R 形式）、Immediate 形式（I 形式）、Long 形式（L 形式）の 4 つの命令形式を用意した。J 形式には無条件分岐命令の JUMP、プログラム終了命令となる HALT を定義している。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。L 形式と J 形式では DM へのアクセスと PC に 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 の命令形式を示す。また、表 2 に命令セットの内容を示す。

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	Rt	imm		
L 形式	Op	Rs	Rt	address/immediate		
J 形式	Op	address				

表 2　命令セットの内容

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,SGTI, SLEI,SGEI,SEQI,SNEI,LDHI, LDLI
L 形式	BEQZ,BNEZ,LD,ST,JAL
J 形式	JUMP,HALT

2.4　並列演算と連鎖演算

(a)　並列演算

ALU 並列演算は、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。

MAP の並列実行において図 2 のプログラムにおける並列関係を示す。並列演算においては、一方の ALU で{ADD $3 $3 $1}のレジスタ演算を行い、もう一方の ALU では{BNEZ $3 LOOP}の分岐命令を行う。

(b)　連鎖演算

ALU 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。PPU ではフェッチした 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。次に連鎖判定で並列演算と連鎖演算の判定を行う。連鎖判定で命令内のレジスタ指定フィールドから一方の ALU の演算結果を他方の ALU で演算を行おうとしているのか判断する。つまり、演算に使われるレジスタが格納するレジスタと等しければ連鎖演算が可能と判断する。連鎖演算が可能であると判断したとき、ALU の前にあるマルチプレクサで ALU の結果を演算に使用できるように指定する。

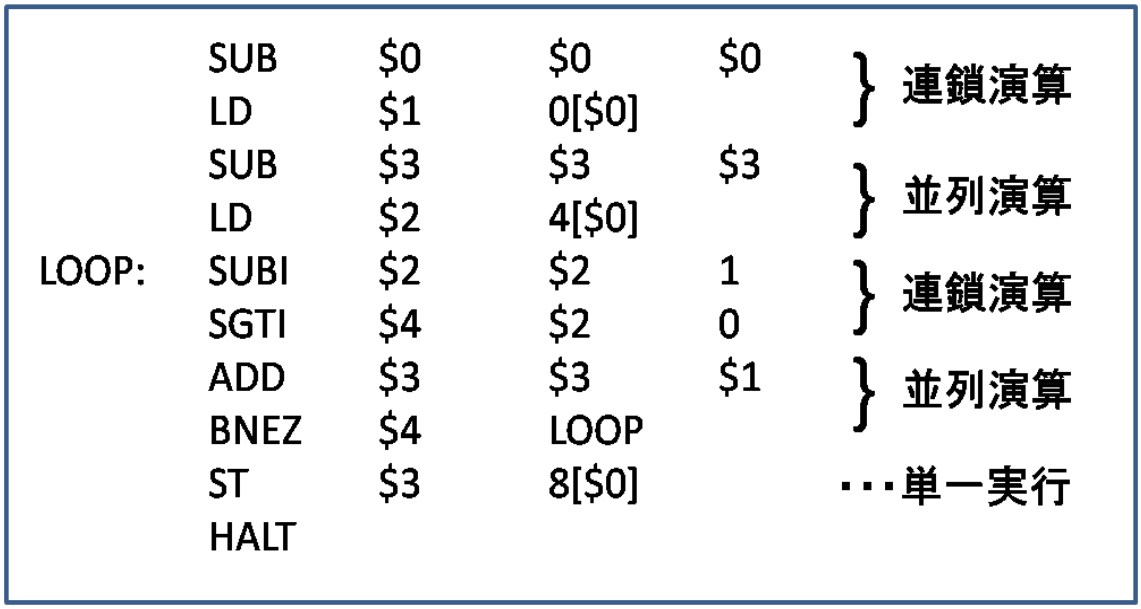

図 2　MAP の乗算プログラム例

3.　プロセッサデバッガ用 GUI の設計

3.1　プロセッサモニタとプロセッサデバッガ[1][3]

MAP を FPGA ボードに実装する際には、HSCS で開発されたプロセッサデバッガ・モニタを用いる。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。プロセッサモニタは、設計したプロセッサを FPGA 上で動作検証する際に、ホスト PC 上から操作するツールとして利用する。プロセッサモニタの役割は 2 つある。1 つは、ユーザからのデバッグ要求を解釈し、プロセッサデバッガに向け指示を与えること。もう 1 つは、ホスト PC 上に展開した仮想メモリバッファとプロセッサデバッガ内のメモリの同期をとることである。図 3 にプロセッサモニタとプロセッサデバッガの構成を示す。

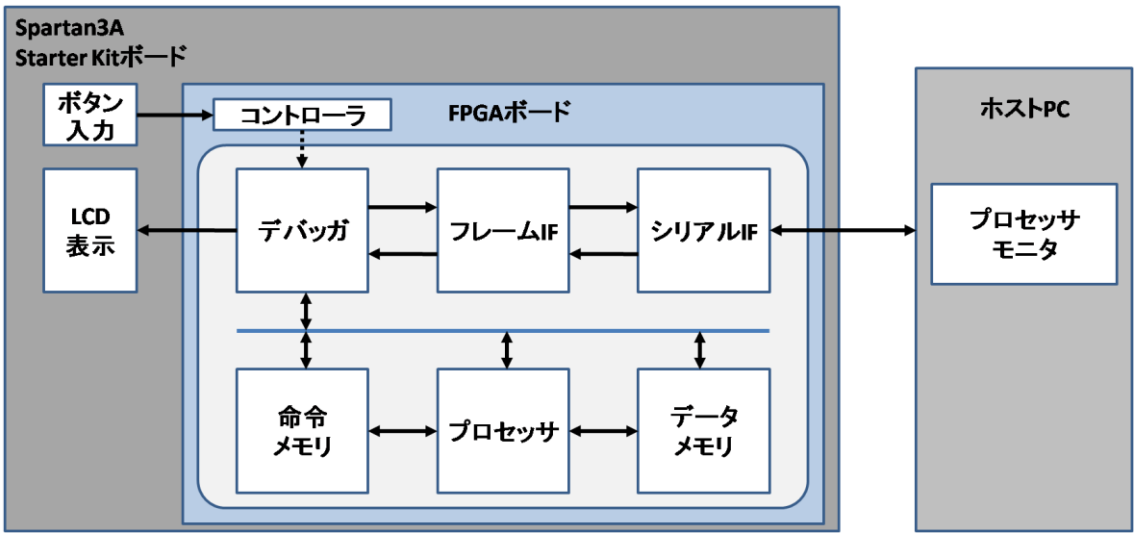

図 3　プロセッサデバッガとプロセッサモニタの構成

プロセッサモニタでは、FPGA 上のメモリ内容を read コマンドによって受信し、内部の Memory モジュールに保持する。ユーザはこのデータを各種デバッグ用のコマンドで自由に編集でき、編集後、送信コマンドによって再度 FPGA 上に展開する。このようにメモリの情報をホスト PC 上で再現することで、実機上のデータを自由に扱える仮想デバッグ環境を提供する。プロセッサモニタの導入により、FPGA ボード上の限られた入出力によらず、柔軟なデバッグ操作が可能となる。

プロセッサモニタの処理の流れは、1 つのユーザコマンドに対し 1 つ以上のアプリケーションコマンドの発行というスタイルで行う。まず、プロセッサモニタ上でユーザのコマンドを受け付け解釈する。受信コマンドの場合は、受信用のバッファを生成し、データの受信に備える。次に、プロセッサデバッガにデータの読み出し要求を送信する。要求はコマンドフレームにパッキングされたアプリケーションコマンドによって伝えられる。プロセッサデバッガは受信したアプリケーションコマンドを実行し、連続して内部のデータを送

信する。プロセッサモニタでは、これらのデータを一時的な受信バッファに保管し、全データ受信後にデータの抽出を行う。受信データにはヘッダや CRC 等の情報も含まれているので、一度 Frame オブジェクトに代入し、メンバ関数によって生データを読み出す。

データメモリの読み出し以外のコマンドも同様に、プロセッサモニタがアプリケーションコマンドを送信することで実行が可能である。

プロセッサデバッガは、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。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。プロセッサ設計支援ツールは、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。

3.2　デバッグコマンド

プロセッサモニタでは、ユーザが入力するユーザコマンドと、プロセッサモニタがプロセッサデバッガに指示を与えるアプリケーションコマンドがあり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っている。GUI においてもこのコマンド体系を継承している。GUI を作成するに際しデバッグコマンドを再検討し一部のコマンドの名称の変更と不要と思われるコマンドの削除を行うことで最適化を図った。GUI 用のデバッグコマンドを表 3 に示す

表 3　デバッグコマンド

コマンド	ターゲット	入力	意味
reset	-	-	RF、PC の初期化
load	dm/im/rf/pc/bp	ファイルパス	ファイルからプロセッサモニタにロード
send	dm/im/rf	開始番地/終了番地	メモリ、レジスタ書き込み
set	dm/rf/pc/bp	対象番地/値	DM、RF、 PC、BP の設定
read	dm/im/rf/pc	開始番地/終了番地	メモリ、レジスタ読み出し
save	dm/im/rf/pc/bp	ファイルパス	メモリ、レジスタの内容を保存
delete	bp	-	BP の削除
run	-	-	通常実行
run clk N	-	-	N クロック実行
run bp	-	-	ブレイク実行

プロセッサモニタは、Microsoft 社提供の Visual Studio 2005 を用いて C++でコマンドライン入力による CUI ベースのアプリケーションで実装されている。CUI ではまず、ユーザが入力したコマンドを正しく解析する字句解析器を作成する。ユーザコマンドにはコマンド名といくつかの引数の組で入力され、コマンドによっては引数が省略されることがある。図 4 に CUI の画面を載せる。

```
input command > load im kake.bin
input command is : 2
Complete loading kake.bin to IM.
Complete to load.
input command > send im 0 5
READ
input command is : 1
trg 1
[送信中. 0x02 0x00 0x02 0x00 0x05 0x00 0x00 0x00 0x00 0x07 ]
Send; address: 00 [送信中. 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0xd8 0x01 0xd9 ]
Send; address: 01 [送信中. 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x42 0x92 0xd4 ]
Send; address: 02 [送信中. 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0xd8 0x0a 0xe2 ]
Send; address: 03 [送信中. 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x43 0x6d 0xb0 ]
Send; address: 04 [送信中. 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x90 0x10 0xa0 ]
Complete to send.
input command > _
```

図 4　CUI の画面

3.3　GUI の設計

GUI は windows アプリケーションを作るのに適していると判断した C#で実装を行なった。また、開発環境として Microsoft 社提供の Visual Studio2010 を用いた。図 5 に GUI の画面構成を示す。

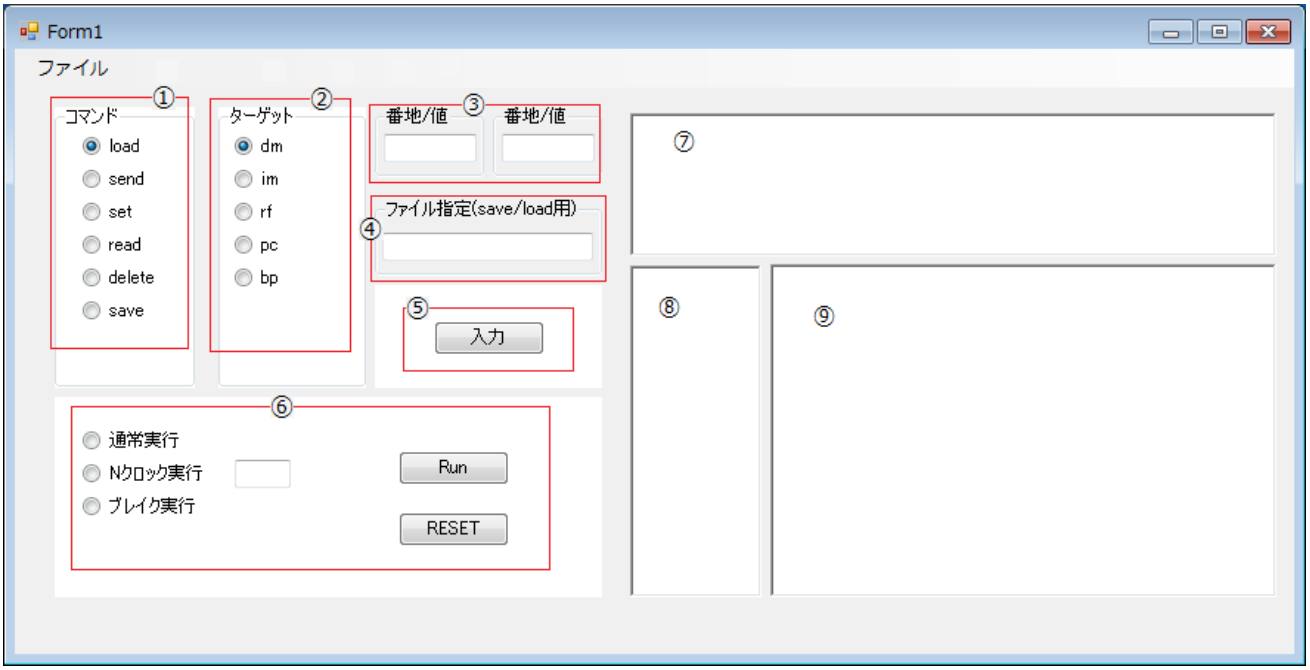

図 5　GUI の画面構成

① 入力したいコマンドを選択する
② 入力したいターゲットを選択する
③ ターゲット番地を入力する。また、set 命令時には左に番地、右に値を入力する
④ load 命令時に使用するファイル名を入力する。また save 命令時には実行結果を保存するファイルを指定する
⑤ ①~④で行いたいコマンドを選択し、このボタンをクリックすることで CUI に送信し、命令が実行される
⑥ 3 種の実行方法から 1 つ選択し、RUN ボタンをクリックすることで run 命令 CUI に送信され、CUI 上で run 命令が実行されプログラムが実行される。また、N クロック実行時は横にクロック数を書き込むことによって N の値を指定することができる。また、RESET ボタンは FPGA ボード上のレジスタを初期化する
⑦ 入力したコマンドを表示
⑧ データメモリまたは命令メモリのアドレスを表示
⑨ 実行結果を表示

4. GUI の実現

4.1 GUI のクラス構成

プロセッサモニタは CUI ベースのアプリケーションであった。そのため、CUI ではコマンド入力時にキーボードからコマンドを入力しなければならない問題やデバッグコマンドを記憶していなければならない問題、コマンド入力とデータ出力が同じ画面内で文字が乱雑に表示される問題があった。そこで本研究では視認性、操作性に優れ、直感的な操作が可能なため、広く普及し、現在では主流のインタフェースになっている GUI を設計することでコマンドを GUI 画面上にボタンで表示しボタン操作でコマンドを入力できるようにし、コマンド入力する場所とデータが出力される場所を分けることで画面が見づらい問題を解消した。

GUI は起動すると同時に CUI をバックグラウンドで起動させることによって、GUI の画面上から CUI に文字を入力するアプリケーションとして開発した。GUI には Form クラスが存在し、それによってアプリケーション画面が起動しボタン入力や値を入力する画面が表示される。Form クラスから従来のプロセッサモニタの Monitor クラスにコマンド入力を行い、命令を実行させる。処理結果はプロセッサモニタ側でテキストファイルに出力し、そのファイルを GUI 側で読み込むことで GUI 画面上での出力を実現した。図 6 に GUI のクラス構成を示す。

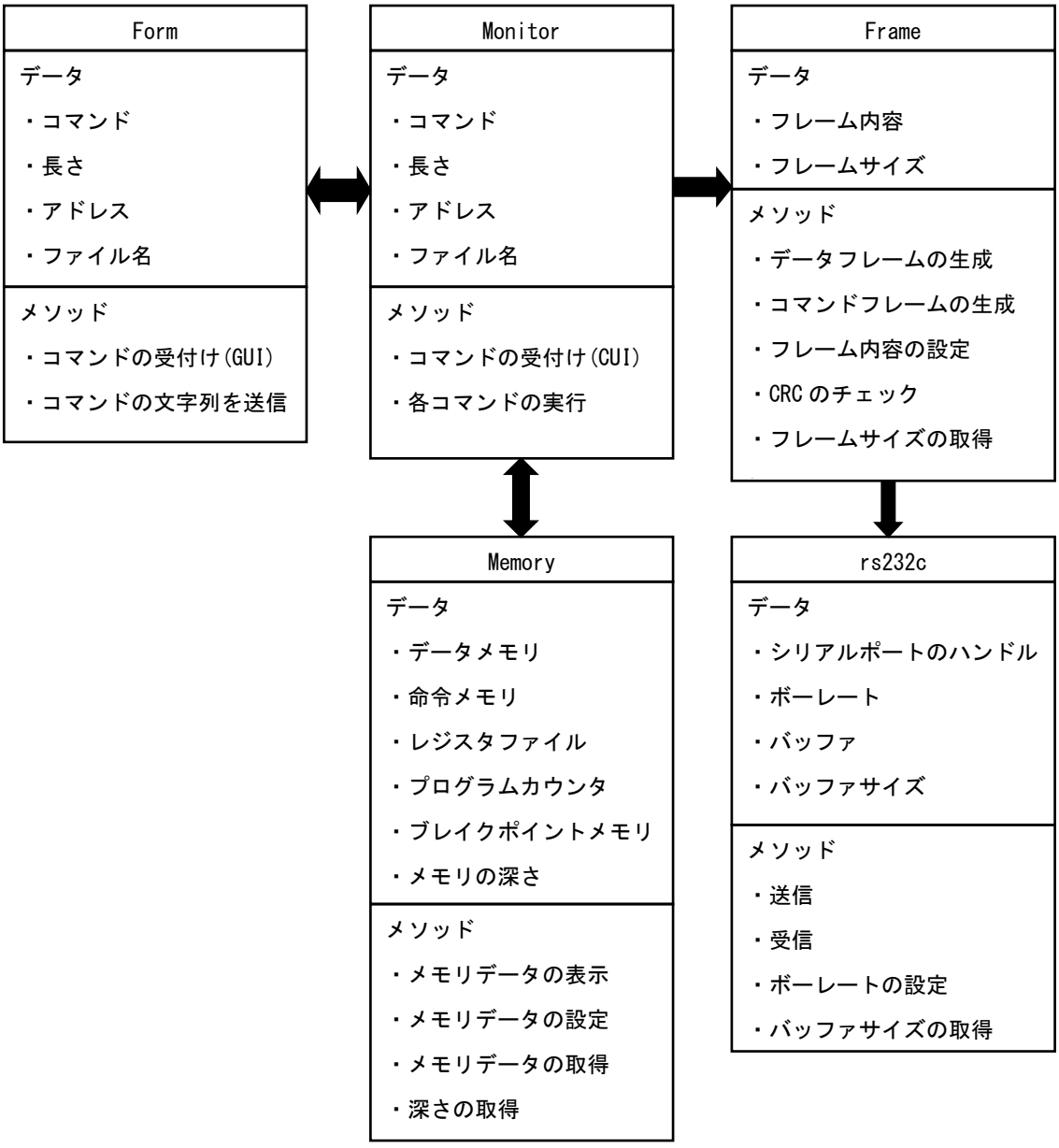

図 6　GUI のクラス構成

4.2　GUI と CUI の関連付け

今回は GUI を起動すると同時に CUI をバックグラウンドで起動させることによって、GUI の画面上から CUI に文字を入力できるプログラムを作成することで実現している。まず、GUI で実行する命令をラジオボタンやボックスに値を打ち込み、入力ボタンをクリックすると GUI は選択されたボタンと値を読み取り、それをもとに従来の CUI で使用されていた命令コマンドを文字列で作成し、CUI 側に送信する。CUI 側で受信した文字列が入力され命令が実行される。実行された結果は従来の CUI ではコンソール画面で表示していたが今回はテキストファイルに出力する。そして、GUI 側で生成された出力ファイルが読み込むことで GUI 側で実行結果の出力を可能とした。

GUI 上で命令コマンドやターゲットをラジオボタンでの選択方式にしたことによって命令コマンドが画面上に表示されているので CUI での問題であった命令コマンドを覚える必要がなくなった。また、CUI では 1 命令ずつキーボートから入力しなければならなかったが GUI ではラジオボタンで選択し値を入力することで命令を実行できる。実行結果の出力が CUI ではコマンド入力画面と同じなので文字が多く乱雑になりやすく見づらかったが GUI では入力したコマンドの表示と実行結果の出力を違うボックスで分けることによって乱雑になることがなくなった。図 7 に GUI と CUI の関係を示す。

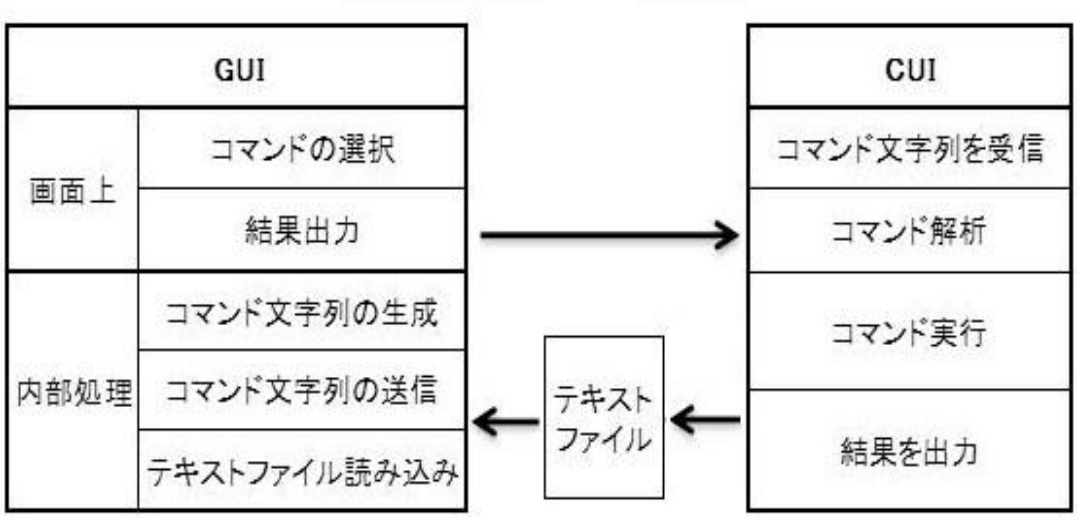

図 7　GUI と CUI の関係

5. GUI を用いたプロセッサ実行

5.1 テスト手法

(1) GUI の動作検証

GUI 上ではデバッグコマンドをラジオボタンによる選択、ボックスに番地の値を打ち込むことで入力する。GUI の動作検証ではラジオボタンや値の解析を行いプロセッサモニタへ送信する文字列が正確に生成されているかどうかを確認した。

(2) CUI へのコマンド入力

GUI でコマンド入力を行うと GUI で生成されたコマンド文字列をプロセッサモニタに送信することでプロセッサデバッガを動作させる。ここでは GUI から送られてきた文字列がプロセッサモニタ側で正しく受信できているかどうかを確認した。

(3) CUI 上での実行結果の出力

従来のプロセッサモニタは CUI に実行結果を出力していたが今回は GUI を実装することにより出力先が GUI 上となる。GUI 上での出力方法はプロセッサモニタ側でデータメモリや命令メモリ、レジスタファイルの内容をテキストに出力し GUI 側で生成されたテキストファイルを読み込むことで GUI 画面上に表示するように設計した。ここではプロセッサモニタ側で正しく結果が出力されているかどうか、また GUI 上で出力されるかを確認した。

5.2 SARIS の実行

従来のプロセッサモニタは本研究室で開発された SARIS プロセッサ等に対応している。このプロセッサモニタを元に GUI を作成するとしたので、まずは SARIS 用のプロセッサデバッガの GUI を作成した。

SARIS プロセッサは命令長 16 ビット固定で 3 オペランド命令形式、全命令は 22 命令、JUMP 形式（J 形式）、Register 形式（R 形式）、Immediate5 形式（I5 形式）、Immediate8 形式（I8 形式）の 4 つの命令形式といった特徴持つプロセッサである[4]。SARIS のプロセッサ構成を図 8 に示す。

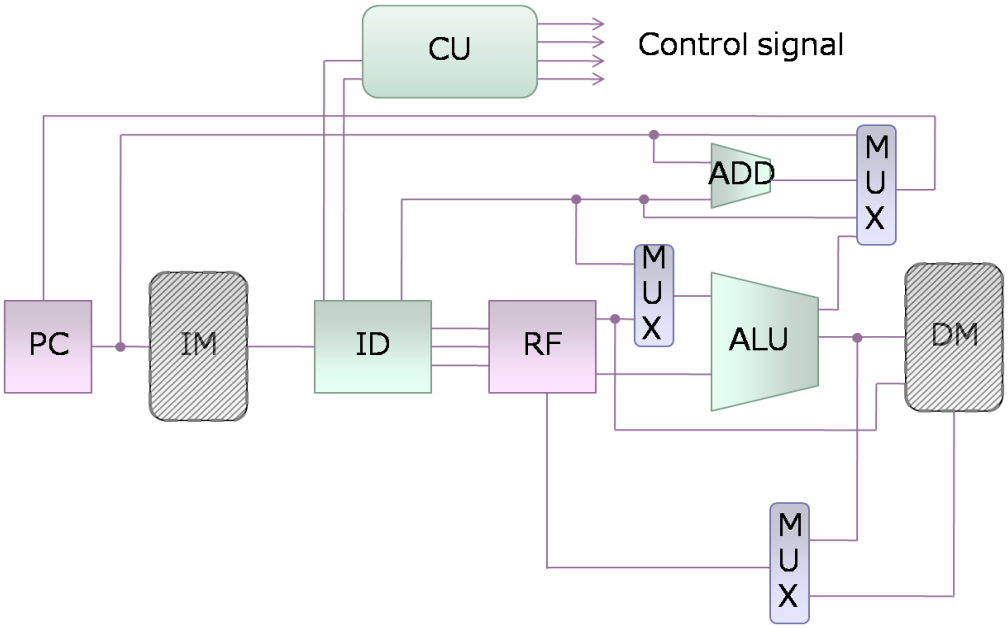

図 8　SARIS のプロセッサ構成

SARIS プロセッサでのテストを行うことで、GUI の基本的な動作を確認することができた。GUI 上でのコマンド選択や番地の指定、ファイル指定を行った。また、コマンド入力後に入力されたコマンドに対応した文字列を生成しプロセッサモニタに送信した。プロセッサモニタ側で生成された文字列の受信、解析を行い、データメモリ、命令メモリの読み書き、プログラムの実行とレジスタの書き込み、演算結果のテキストファイルのへの出力を行った。これにより GUI でプロセッサデバッガを動作させることが可能と示せた。

5.3　MAP の実行

MAP は使用する命令語長やレジスタファイルが従来のプロセッサとは違う。そこでプロセッサモニタ側で命令語長やレジスタファイルを調整することで従来のプロセッサモニタから MAP に適応させた。なお、GUI 側での変更はない。

MAP でのテストも SARIS プロセッサ時のテストと同様に GUI の基本的な動作を確認した。GUI 上でのコマンド選択や番地の指定、ファイル指定を行い、コマンド入力後に入力されたコマンドに対応した文字列を生成しプロセッサモニタに送信した。プロセッサモニタ側で生成された文字列の受信、解析を行い、データメモリ、命令メモリの読み書き、プログラムの実行とレジスタの書き込み、演算結果のテキストファイルのへの出力を行った。GUI で生成されたテキストファイルを読み込み画面上に出力されることを確認した。図 9～図 11 に命令実行時の出力結果を示す。これにより GUI で MAP 用プロセッサデバッガ・モニタを動作させることが可能と示せた。

図 9　send 命令実行結果

図 10　run 命令実行結果

図 11　read 命令実行結果

5.4　考察

プロセッサデバッガのユーザーインターフェースとしてGUIを実装することによってデバッグコマンドがGUI上で明確に表示されているので使用者がコマンドを覚える必要がなくなった。また、コマンドをラジオボタンでの選択式にすることでスムーズにコマンド入力することができ、プログラム実行が素早くできる。GUI画面上でデータメモリや命令メモリ、レジスタファイルの中身の出力することで文字が整理されて表示されるので非常に読み取りやすくなった。また、従来のプロセッサモニタからMAP用プロセッサモニタへと修正することができたことで、HSCSを利用して新たなプロセッサが開発された場合、デバッグコマンドが同じであればプロセッサモニタ側で命令語長やレジスタファイル容量などを調整することで修正することでGUIはそのまま使用することが可能であると考える。従来のプロセッサモニタのCUI画面上での出力内容とGUIの出力内容でデータメモリや命令メモリに格納されている値は同じであるがGUI側では16進表示で行われる値の2桁目に格納されている0がスペースとなってしまっていた。プロセッサモニタ側でのテキストファイルへの出力の際に問題が生じていると考えられる。

6. おわりに

本研究では、プロセッサ設計支援ツールのプロセッサモニタの問題点を解決するために GUI の開発を行った。開発した GUI を SARIS プロセッサでの検証、プロセッサモニタを MAP へ対応させることで GUI を MAP での使用を可能とし FPGA ボード上での動作検証を行った。

現在の GUI はプロセッサデバッガを動作させる最低限の機能しか実装されていない。また、従来のプロセッサモニタの CUI 画面上での出力と GUI の出力が少し違って表示される。今後の課題としては、GUI 上での出力の改善、windows アプリケーションとしての機能向上が挙げられる。

謝辞

本研究の機会を与えてくださり、ご指導を頂きました山崎勝弘教授に深く感謝いたします。また、本研究に関して様々な相談に乗って頂き、貴重な助言を頂いた孟助手や境直樹氏をはじめ、様々な面で貴重な助言や励ましを下さった研究室の皆様に深く感謝いたします。

参考文献

[1] 難波翔一郎：プロセッサ設計支援ツールの実装とハード/ソフト協調学習システムの評価、立命館大学理工学研究科修士論文、2007.

[2] 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

[3] 難波翔一郎・志水建太・山崎勝弘・小柳滋：プロセッサ設計支援ツールの設計・実装とハード/ソフト協調学習システムの評価、FIT2007.LC002.2007.

[4] 井手 純一：ハード/ソフト協調学習システム を用いたプロセッサ設計と評価、立命館大学理工学部情報学科卒業論文、2008

[5] 境直樹・山崎勝弘：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

付録

付録として GUI 起動時に PM を起動する記述と GUI から CUI に文字を送信する記述についての説明をします。

・GUI 起動時に PM を起動する記述

```
namespace PDGUI
{
    // 開始直後の動作
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // PM起動
            Process.Start(@"PM.exe");

            // 500ms待機
            System.Threading.Thread.Sleep(500);

            richTextBox1.Text = "Ready.¥n";

            // 承諾ボタンを[入力]ボタンにセットする
            this.AcceptButton = this.button2;
        }
```

図 12　PM を起動する記述

今回作成した GUI では図 12 の赤い枠内に記述されている C#の Process メソッドを使用することで外部プログラムを呼び出しています。

・GUI から CUI に文字を送信する記述

```
// dm
if (radioButton7.Checked)
{
    string loaddm = "load dm " + value1;
    richTextBox1.AppendText(loaddm);

    SendKeys.SendWait(loaddm);
}
```

図 13　GUI から CUI に文字を送信する記述

GUI から CUI に文字を送信する方法は C#の Sendkeys メソッドを使用することで実現しています。string 型の変数にコマンド命令を格納し、その変数を Sendkeys メソッドで CUI に渡すことで CUI に命令を入力することができます。