1998 年度 卒業論文

FPGA を用いた行列計算専用プロセッサの設計

9510191　山岡 寛明

電気通信大学 電気通信学部 電子工学科 木村・齋藤研究室

指導教官 齋藤 理一郎 助教授

提出日 平成 11 年 2 月 10 日

概要

物性の計算には、多くの場合膨大な計算量を要する。例えば、分子軌道計算には行列の固有値・固有ベクトルを求める計算を必要とするが、その計算量は行列の次数を N とすると $O(N^3)$ となり、$N \sim 10^4$ に及ぶ大規模な計算を実用時間内に行うことは非常に困難となっている。物性計算においてその計算時間を短縮することが大きな課題である。

計算時間短縮のためにはスーパーコンピュータをはじめとする並列コンピュータを用いた計算の並列化が考えられるが、演算に並列化できない部分があるとどんなにプロセッサ数を増やしても並列処理の効果は期待できない、プロセッサ間の通信が多い場合、そこで時間がかかる等の問題があるため、計算時間短縮にはならない場合が多い。

そこで、近年、汎用並列コンピュータにかわるアーキテクチャとして、ある問題に特化したコンピュータである専用コンピュータを用いる手法が $O(N)$ 法に代表される新アルゴリズムの開発とともに一般的になってきた。本研究では行列計算を対象とし、専用プロセッサを設計することによって計算時間短縮を試みる。

また従来、対象のアーキテクチャの性能評価を行うには大別して、ソフトウェアでエミュレーションを行う方法、あるいは実際に対象のアーキテクチャをハードウェアとして製作する等の手法が取られているが、前者では評価に膨大な時間を要する、後者では製作に多額の費用がかかる、開発から実際に評価を行うまでに相当な期間を要する、ハードウェアを変更して複数のアーキテクチャを試行することが困難である等の問題がある。

これらの問題を解決するシステムとして、近年、デジタル回路設計手法として一般的になってきたハードウェア記述言語 HDL(Hardware Description Language) の 1 つである VHDL を用いてプロセッサの機能設計を行い、これをプログラマブルデバイスである FPGA(Field Programmable Gate Array) を用いて実装評価するシステムを構築した。実際に、10 万ゲート相当の FPGA 2 個を用いた基盤 [3] において、行列固有値・固有ベクトル計算専用プロセッサを実装した。プロセッサは 4 つの単精度浮動小数点数演算器を含み、同時処理で最大 16MFLOPS の性能を持つ。

目次

第 1 章

序論

1.1　背景、研究目的

物性の計算には、多くの場合膨大な計算量を要する。例えば、分子軌道計算には行列の固有値・固有ベクトルを求める計算を必要とするが、その計算量は行列の次数を N とすると $O(N^3)$ となり、$N \sim 10^4$ に及ぶ大規模な計算を実用時間内に行うことは非常に困難となっている。物性計算においてその計算時間を短縮することが大きな課題である。

計算時間短縮のためにはスーパーコンピュータをはじめとする並列コンピュータを用いた計算の並列化が考えられる。これにより、演算を独立なものに分割し、並列に計算を行えば計算時間の短縮が期待できる。しかし、大きな問題点が 2 つある。1 つめは、演算に並列化できない部分があるとどんなにプロセッサ数を増やしても並列処理の効果は期待できないということである。2 つめは、プロセッサ間の通信が多い場合、そこで時間がかかり、結局計算時間の短縮にはならないということである。また、現在の汎用コンピュータは幅広い問題に対応できるように設計されているため、ある問題に特化した場合には計算効率が悪い場合が多い。

そこで、汎用並列コンピュータにかわるアーキテクチャとして、ある問題に特化したコンピュータである専用コンピュータを用いる手法が $O(N)$ 法に代表される新アルゴリズムの開発とともに一般的になってきた。汎用コンピュータにおいて効率の悪い計算を、専用の計算システムを考えることによって計算時間の短縮が期待できる。

本研究は、行列計算に特化した専用プロセッサを設計し、評価をすることが目的で

ある。

従来では、コンピュータアーキテクチャの研究に際し、対象のアーキテクチャの性能評価を行うには大別して、ソフトウェアでエミュレーションを行う方法、あるいは実際に対象のアーキテクチャをハードウェアとして製作する等の手法が取られているが、ソフトウェアエミュレーションは実行速度が非常に遅い場合が多く、サンプルプログラムをシミュレーションにかけて評価するには多大な時間を要する。

一方、ハードウェアで対象アーキテクチャを製作する方法では製作に多額の費用がかかる、設計を開始してから製作、デバッグ、調整を経て最終的に評価に入るまでに多大な時間を要する、そして一旦製作したハードウェアを変更して複数のアーキテクチャを試行することが困難であるという問題がある。

このようなことから、十分な演算速度を持つ、アーキテクチャの設定、変更が容易である、アーキテクチャを設定してから評価を行うまでの時間を短縮できる、過大でない予算で十分実現可能であるといった要素を満たすシステムを実現するために、プログラマブルなデバイスを使用してアーキテクチャを評価するシステムについて検討した。

現在までに、このような要件を満たすシステムにおけるコンピュータアーキテクチャの研究が試みられてきてはいるが、本研究で扱うような数値計算を目的としたものは依然として大きく発展するには至っていない。その大きな理由として、一般に数値計算において必要とされる浮動小数点数演算器の回路規模は非常に大きく、これを実装するのは困難であった。

しかし、最近ではLSI技術の急速な進歩により、高速であり、10万ゲート以上の規模の大容量プログラマブルデバイスが提供されてきている。これにより、本研究の設計思想が明確なものとなってきた。

本研究では、近年、デジタル回路設計手法として一般的になってきたハードウェア記述言語HDL(Hardware Description Language)の1つであるVHDLを用いてプロセッサの機能設計を行い、これをプログラマブルデバイスであるFPGA(Field Programmable Gate Array)を用い、実際にプロセッサをハードウェアとして実現し、評価を行う。

1.2　今までの研究成果と本年度の課題

本研究は、科学技術計算における線形計算を高速に行う専用計算機の開発を目的として、1996 年度 4 月より開始した研究である。ここでは、今までの研究成果と本年度の課題を述べる。

まずはじめに、本研究では、科学技術計算における線形計算の中でも物性計算等で多用される行列計算 (行列固有値・固有ベクトル計算) を計算対象とすることにし、これを高速に計算する専用計算機の開発を見据え、適切な計算アルゴリズムを探索した。計算アルゴリズムの評価には、計算量、ハードウェア化の可能性を考慮し、その結果、ハウスホルダー法 (ハウスホルダー変換、二分法、逆反復法、ハウスホルダー逆変換) を採用することに決定した。ハウスホルダー法は行列の次数が大きくなるほど、他のアルゴリズムより計算効率が良い、また、並列計算が可能であるという特徴があり、専用計算機を並列計算機として設計することが期待できる。実際に、ソフトウェアシミュレーションにより、ハウスホルダー法の計算アルゴリズムの妥当性の検証を行い [1]、また、計算アルゴリズムの並列化の一つのモデルを作成した [2]。

専用計算機の計算対象、計算アルゴリズムが決定したところで、次は計算アルゴリズムを実際にハードウェア化することが課題である。ハードウェア化に際して、まず検討しなければならないことは設計手段と使用するテクノロジーであるが、前節で述べたように設計の容易性、開発費用を考慮する必要がある。このようなことから、本研究では、近年、デジタル回路のハイレベル設計手法として一般的になってきたハードウェア記述言語 HDL の一つである VHDL を設計手段として採用し、また、VHDL により設計した機能をハードウェアとして実現するためのテクノロジーとして、プログラマブルデバイスである FPGA を用いることが適切であると考えた。

我々の研究室では、このような開発環境を得るために、HDL 設計ツールとして、(株) インターリンクよりアカデミックな価格で PeakVHDL/FPGA を導入し、また、(株) 日本アルテラ の行っているユニバーシティ・プログラムに参加することによって、FPGA の配置・配線ツールとして MAX+PLUS II の無償提供を受けた。FPGA は (株) 日本アルテラ から提供される FLEX 10K シリーズの一つである EPF10K100GC503-3 (公称値 10 万ゲート)2 個を導入した。

このようにして、設計ツール、FPGA を導入したが、実際に専用計算機を構築する

ためには、FPGA を載せるための基盤が必要であり、また、FPGA への機能のダウンロードと実際の動作検証を行うために、基盤とパソコンとの通信を行うインターフェースが必要である。

本研究では、導入した FPGA 2 個が搭載可能であり、かつ SRAM、DRAM を計算アルゴリズムに適した配置で FPGA とともに搭載できる基盤を製作し、そして、パソコンの ISA バスよりパラレルインターフェース PPI8255 を通じて基盤との通信が行えるインターフェースを製作した [3]。

また、この基盤の製作と並行して VHDL により計算アルゴリズムを動作レベルで記述し、機能シミュレーションを行うことにより計算アルゴリズムを VHDL で記述する際の一つのモデルを作成した [4]。

以上が本年度までの研究成果である。これで概ね開発環境は整ったといえる。そこで、本年度の課題としては、FPGA 搭載基盤において実際に動作するデジタル回路を VHDL で設計することである。まずは、計算に必要である浮動小数点演算器やメモリコントローラ等のモジュールを設計し、そしてこれらを用いてハウスホルダー法の計算アルゴリズムを論理合成でき、かつ FPGA の容量を越えないような形式で設計し、FPGA へ機能を実装して、行列固有値・固有ベクトル計算専用システムを構築することが本年度の目標である。

1.3　本論文の構成

まず、第 2 章において本研究で用いる行列の固有値・固有ベクトルを求めるアルゴリズムであるハウスホルダー法について説明する。

第 3 章ではハードウェア記述言語 HDL を用いた LSI 設計手順を示すとともに、設計した計算システムを実装評価するための FPGA 搭載計算システム評価ボードについて説明する。

第 4 章では設計した数値計算プロセッサについて説明する。

第 5 章では 4 章で挙げた数値計算プロセッサによる計算結果を示すとともに、プロセッサの性能評価を行う。

第 6 章では本研究の結論を述べる。

付録には VHDL で作成した数値計算プログラム及び C 言語で作成した評価ボードで

の行列固有値・固有ベクトル計算実行プログラムを示し、また、FPGA 搭載計算システム評価ボードの詳細を説明する。

第 2 章

計算方法

行列の固有値・固有ベクトルを求めるための効率の良いアルゴリズムとして、行列の相似変換を利用したハウスホルダー法 (Householder method) がよく知られている。本研究では行列の固有値・固有ベクトルを求める専用プロセッサをハウスホルダー法を用いて設計する。ここでは、ハウスホルダー法による行列固有値・固有ベクトルの計算手順を説明する。

2.1　ハウスホルダー法

行列 $\boldsymbol{A}$ の固有値・固有ベクトルを直接計算するのではなく、まず相似変換によって三重対角行列に変換したあとで、その三重対角行列の固有値・固有ベクトルを計算するのがハウスホルダー法である。これはハウスホルダー変換、二分法、逆反復法、ハウスホルダー逆変換の 4 つの手法により構成される。ここで、三重対角行列とは、対角要素および副対角要素以外は 0 であるような行列をいう。

計算手順としては、まずハウスホルダー変換により、与えられた行列を三重対角行列へ変換し、二分法によりこの三重対角行列の固有値を求め、逆反復法により三重対角行列とその固有値を用いて固有ベクトルを求め、最後にハウスホルダー逆変換により三重対角行列の固有ベクトルをもとの行列の固有ベクトルに変換する。

以下、順に 4 つの手法の計算手順を説明する。

2.1.1 ハウスホルダー変換

ハウスホルダー変換 (Householder transformation) は一般の対称行列を三重対角行列に変換するアルゴリズムである。

$n \times n$ 行列 $\boldsymbol{A}^{(0)}$ が与えられたとすると、第 1 回目の変換によって、$\boldsymbol{A}^{(0)}$ の第 1 列目の要素 $a_{i1}(i \geq 3)$ を 0 にする。$\boldsymbol{A}^{(0)}$ が対称行列であれば、第 1 行目の $a_{1j}(j \geq 3)$ も同時に 0 となる。その結果できた行列を $\boldsymbol{A}^{(1)}$ とすると、$\boldsymbol{A}^{(1)}$ の第 1 行、第 1 列を除いた行列に対して、第 1 回目の変換と同様の変換をする。この操作を続けていけば、結局 $(n-2)$ 回の変換によって三重対角行列に変換することができる。

$(r-1)$ 回の変換により、左上の $(r-1)$ 行 $(r-1)$ 列がすでに三重対角行列になっている行列を $\boldsymbol{A}^{(r-1)}$ とすると、

$$\boldsymbol{A}^{(r)} = \boldsymbol{P}^{(r)} \boldsymbol{A}^{(r-1)} \boldsymbol{P}^{(r)}, \qquad r = 1,\ 2, \cdots,\ n-2 \tag{2.1}$$

によって $\boldsymbol{A}^{(r)}$ が計算される。この変換行列$\boldsymbol{P}^{(r)}$ は

$$\boldsymbol{P}^{(r)} = \boldsymbol{I} - c_r \boldsymbol{u}^{(r)} \boldsymbol{u}^{(r)T} \tag{2.2}$$

と書ける。ここで、

$$\boldsymbol{u}^{(r)} = (\underbrace{0, \cdots, 0}_{r},\ a_{r+1,r}^{(r-1)} \ +\ \mathrm{sgn}(a_{r+1,r}^{(r-1)}) s_r, a_{r+2,r}^{(r-1)}, \cdots, a_{n,r}^{(r-1)})^T \tag{2.3}$$

$$s_r = \sqrt{\sum_{i=r+1}^{n} (a_{i,r}^{(r-1)})^2} \tag{2.4}$$

$$c_r = \frac{1}{s_r^2 + |a_{r+1,r}^{(r-1)}| s_r} \tag{2.5}$$

である。ただし、 $s_r = 0$ のときには $\boldsymbol{P}^{(r)} = \boldsymbol{I}$ とする。 $\boldsymbol{u}^{(r)}$ の $(r+1)$ 成分に $\mathrm{sgn}(a_{r+1,r}^{(r-1)})$ [1]があるのは、この計算で桁落ちが起こらないようにするためである。

$$\begin{aligned} \boldsymbol{u}^{(r)T} \boldsymbol{u}^{(r)} &= \{a_{r+1,r}^{(r-1)} \ +\ \mathrm{sgn}(a_{r+1,r}^{(r-1)}) s_r\}^2 \ +\ \sum_{i=r+2}^{n} (a_{i,r}^{(r-1)})^2 \\ &= s_r^2 + 2|a_{r+1,r}^{(r-1)}| s_r + s_r^2 \\ &= \frac{2}{c_r} \end{aligned} \tag{2.6}$$

[1] $\mathrm{sgn}(x) = \begin{cases} +1 & (x \geq 0) \\ -1 & (x < 0) \end{cases}$

であるから、

$$\boldsymbol{P}^{(r)}\boldsymbol{P}^{(r)} = \boldsymbol{I} - 2c_r\boldsymbol{u}^{(r)}\boldsymbol{u}^{(r)T} + c_r^2\boldsymbol{u}^{(r)}(\boldsymbol{u}^{(r)T}\boldsymbol{u}^{(r)})\boldsymbol{u}^{(r)T} = \boldsymbol{I} \tag{2.7}$$

すなわち、

$$\boldsymbol{P}^{(r)-1} = \boldsymbol{P}^{(r)} \tag{2.8}$$

である。また、明らかに

$$\boldsymbol{P}^{(r)T} = \boldsymbol{P}^{(r)} \tag{2.9}$$

であるから、$\boldsymbol{P}^{(r)}$ は対称な直交行列であることがわかる。

$\boldsymbol{P}^{(r)}$ の左上の r 行 r 列は単位行列と同じであるから、$\boldsymbol{A}^{(r-1)}$ の左上の $r \times r$ 小行列は変化をうけない。また、$\boldsymbol{A}^{(r-1)}$ の r 列目のベクトルを$\boldsymbol{a}_r^{(r-1)}$ とすると、$\boldsymbol{A}^{(r-1)}\boldsymbol{P}^{(r)}$ の r 列目も$\boldsymbol{a}_r^{(r-1)}$ となる。したがって、$\boldsymbol{A}^{(r)}$ の r 列目のベクトル$\boldsymbol{a}_r^{(r)}$ は、

$$\begin{aligned}\boldsymbol{a}_r^{(r)} &= \boldsymbol{P}^{(r)}\boldsymbol{a}_r^{(r-1)} = (\boldsymbol{I} - c_r\boldsymbol{u}^{(r)}\boldsymbol{u}^{(r)T})\boldsymbol{a}_r^{(r-1)} \\ &= \boldsymbol{a}_r^{(r-1)} - (c_r\boldsymbol{u}^{(r)T}\boldsymbol{a}_r^{(r-1)})\boldsymbol{u}^{(r)} \\ &= \boldsymbol{a}_r^{(r-1)} - \boldsymbol{u}^{(r)}\end{aligned} \tag{2.10}$$

となるから、$\boldsymbol{a}_r^{(r)}$ のはじめの r 成分は変化せず、$(r+1)$ 成分から先は、

$$\begin{aligned}&\boldsymbol{a}_{r+1,r}^{(r)} = -\mathrm{sgn}(a_{r+1,r}^{(r-1)})s_r, \\ &\boldsymbol{a}_{i,r}^{(r)} = 0, \qquad r+2 \le i \le n\end{aligned} \tag{2.11}$$

となって、$\boldsymbol{A}^{(r)}$ の左上の r 行 r 列は三重対角行列となる。

$\boldsymbol{A}^{(r)}$ の右下の $(n-r)\times(n-r)$ 小行列は、行列の対称性を利用して次のようにして計算する。

$$\boldsymbol{A}^{(r)} = \boldsymbol{A}^{(r-1)} - \boldsymbol{u}^{(r)}\boldsymbol{q}^{(r)T} - \boldsymbol{q}^{(r)}\boldsymbol{u}^{(r)T} \tag{2.12}$$

ここで、

$$\boldsymbol{q}^{(r)} = \boldsymbol{p}^{(r)} - \frac{1}{2}c_r\alpha_r\boldsymbol{u}^{(r)} \tag{2.13}$$

$$\boldsymbol{p}^{(r)} = c_r\boldsymbol{A}^{(r-1)}\boldsymbol{u}^{(r)} \tag{2.14}$$

$$\alpha_r = \boldsymbol{u}^{(r)T}\boldsymbol{p}^{(r)} \tag{2.15}$$

である。

ハウスホルダー変換による三重対角化に要する演算回数は乗算が約 $\frac{2}{3}n^3$ 回、平方根の計算が $(n-2)$ 回である。

2.1.2 二分法

ハウスホルダー変換により、行列 $\boldsymbol{A}$ と相似な三重対角行列

$$\boldsymbol{T} = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ 0 & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{2.16}$$

が得られたとする。$\boldsymbol{T}$ に対応して $\boldsymbol{T} - \lambda \boldsymbol{I}$ を考える。

$$\boldsymbol{T} - \lambda \boldsymbol{I} = \begin{pmatrix} \alpha_1 - \lambda & \beta_1 & & & \\ \beta_1 & \alpha_2 - \lambda & \beta_2 & & \\ & \cdots & \cdots & & \\ & & \cdots & \cdots & \\ 0 & & \beta_{n-2} & \alpha_{n-1} - \lambda & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n - \lambda \end{pmatrix} \tag{2.17}$$

$\boldsymbol{T} - \lambda \boldsymbol{I}$ の左上からとった k 番目の主小行列式 $p_k(\lambda)$ は、次の漸化式により求められる。

$$\begin{aligned} &p_0(\lambda) = 1, \quad p_1(\lambda) = \alpha_1 - \lambda, \\ &p_k(\lambda) = (\alpha_k - \lambda) p_{k-1}(\lambda) - \beta_{k-1}^2 p_{k-2}(\lambda) \end{aligned} \tag{2.18}$$

$k = n$ のときには、

$$p_n(\lambda) = |\boldsymbol{T} - \lambda \boldsymbol{I}| \tag{2.19}$$

であり、この根が求める固有値である。

この関数列 $\{p_k(\lambda)\}$ は、$\beta_k \neq 0$ であれば次の 3 つの条件を満たすからスツルム (Sturm) 関数列であることがわかる。

1. $p_0(\lambda)$ は定符号である。

2. 引き続く二つの多項式は同時に 0 にならない。

3. $p_k(\lambda) = 0$ となる λ に対して、$p_{k-1}(\lambda) p_{k+1}(\lambda) < 0$ となる。

ある λ_0 に対して、$p_0(\lambda_0)$、$p_1(\lambda_0)$、$\cdots$、$p_n(\lambda_0)$ の値を計算し、隣同士の符号が一致する回数を $N(\lambda_0)$ とする。ただし、$p_k(\lambda_0) = 0$ となったときは、$p_k(\lambda_0)$ は $p_{k+1}(\lambda_0)$ と同符号であると考える。そうすると、$N(\lambda_0)$ は $\boldsymbol{T}$の固有値のうち λ_0 以上のものの個数を与える。

実際には、(2.18) の漸化式を計算するかわりに、

$$q_k(\lambda) = p_k(\lambda)/p_{k-1}(\lambda) \tag{2.20}$$

として、

$$\begin{aligned} q_1(\lambda) &= \alpha_1 - \lambda, \\ q_k(\lambda) &= \alpha_k - \lambda - \beta_{k-1}^2/p_{k-1}(\lambda) \end{aligned} \tag{2.21}$$

によって $q_k(\lambda_0)$ を計算すれば、正または 0 となる $q_k(\lambda_0)$ の個数が $N(\lambda_0)$ となる。$q_k(\lambda_0)$ が 0 のときは、ε を適当に小さい整数として、

$$q_{k+1}(\lambda_0) = \alpha_{k+1} - \lambda_0 - |\beta_k|/\varepsilon \tag{2.22}$$

で置き換えて計算を続ければよい。この置き換えによる固有値の変動は $\varepsilon|\beta_k|$ の程度である。実際には、ε を非常に小さくとれば、$q_{k+1}(\lambda_0)$ は絶対値の非常に大きい負の数になるから、

$$\begin{aligned} q_{k+2}(\lambda_0) &= \alpha_{k+2} - \lambda_0 - \beta_{k+1}^2/q_{k+1}(\lambda_0) \\ &\simeq \alpha_{k+2} - \lambda_0 \end{aligned} \tag{2.23}$$

となり、$q_{k+1}(\lambda_0)$ の計算は省略して、$q_{k+2}(\lambda_0)$ を上式で計算すればよい。

副対角要素 $\beta_k = 0$ がある場合には、$\boldsymbol{T}$ はいくつかの主小行列に分けて考えることになる。ところが、(2.21) の漸化式を用いれば、たとえば $\beta_m = 0$ のときには、

$$q_{m+1}(\lambda) = \alpha_{m+1} - \lambda \tag{2.24}$$

となるから、

$$q_{m+i}(\lambda) = p_{m+i}(\lambda)/p_{m+i-1}(\lambda), \quad p_m(\lambda) = 1 \tag{2.25}$$

と考え直すことによって、行列全体に適用することができる。

この性質を利用して、$\boldsymbol{T}$ の固有値を計算する方法が二分法 (bisection method) である。すなわち、大きいほうから k 番目の固有値 λ_k を求めるときには、$N(a) \geq k$ 、

$N(b) < k$ となる a 、 b を選び、区間 (a,b) の中点 $c = (a+b)/2$ に対して $N(c)$ を計算し、 $N(c) \geq k$ ならば a を c で置き換え、 $N(c) < k$ ならば b を c で置き換える。これを繰り返すことによって、 λ_k の存在する区間の幅を毎回 $1/2$ に狭めることができる。したがって、所要の精度の範囲まで区間の幅が狭くなったところで計算を打ち切れば λ_k が求められる。

なお、$\boldsymbol{T}$ の固有値は ゲルシュゴーリン (Gerschgorin) の定理により、

$$d = \max_{2 \leq i \leq n-1} \{(|\alpha_1| + |\beta_1|),\ (|\beta_{i-1}| + |\alpha_i| + |\beta_i|),\ (|\beta_{n-1}| + |\alpha_n|)\} \tag{2.26}$$

としたとき、すべて区間 $(-d,d)$ の間に存在することがわかっているから、この区間から計算を始めれば、任意の固有値を求めることができる。

2.1.3 逆反復法

三重対角行列 $\boldsymbol{T}$ の固有値を λ_k とし、二分法で計算した近似固有値を $\tilde{\lambda}_k$ とするとき、任意のベクトル $\boldsymbol{x}^{(0)}$ から始めて、

$$(\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I})\boldsymbol{x}^{(r)} = \boldsymbol{x}^{(r-1)} \tag{2.27}$$

によって、 $\boldsymbol{x}^{(r)}$ を繰り返し計算すれば λ_k に対する固有ベクトルが求められる。この方法を逆反復法 (inverse iteration method) という。

初期ベクトル $\boldsymbol{x}^{(0)}$ を $\boldsymbol{T}$ の固有ベクトル $\boldsymbol{u}_i$ で展開して

$$\boldsymbol{x}^{(0)} = \sum_{i=1}^{n} a_i \boldsymbol{u}_i \tag{2.28}$$

とすると、

$$\begin{aligned} \boldsymbol{x}^{(r)} &= (\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I})^{-r} \boldsymbol{x}^{(0)} = \sum_{i=1}^{n} \frac{a_i}{(\lambda_i - \tilde{\lambda}_k)^r} \boldsymbol{u}_i \\ &= \frac{1}{(\lambda_k - \tilde{\lambda}_k)^r} \left\{ a_k \boldsymbol{u}_k + \sum_{i \neq k} \left(\frac{\lambda_k - \tilde{\lambda}_k}{\lambda_i - \tilde{\lambda}_k} \right)^r a_i \boldsymbol{u}_i \right\} \end{aligned} \tag{2.29}$$

となり、ここで $\tilde{\lambda}_k$ が λ_k の十分よい近似値であれば、

$$\left| \frac{\lambda_k - \tilde{\lambda}_k}{\lambda_i - \tilde{\lambda}_k} \right| \ll 1 \tag{2.30}$$

であるから、たいていの場合、$r = 2$ 程度で $\boldsymbol{x}^{(r)}$ は真の固有ベクトル $\boldsymbol{u}_k$ に収束する。このとき連立 1 次方程式 (2.27) の係数行列 $(\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I})$ の行列式が 0 に近いために、解は真の解にはならず、その定数倍になるが、固有ベクトルを求める場合には定数倍は差し支えない。

実際の計算手順は次のようになる。まず $(\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I})$ に対して、行の変換 (ピボットの選択) を伴うガウスの消去法を適用して、右上三角行列 $\boldsymbol{R}$ に変換する。変換行列を $\boldsymbol{L}$ と書くことにすると、

$$\boldsymbol{L}(\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I}) = \boldsymbol{R} \tag{2.31}$$

となる。行列 $\boldsymbol{L}$ 全体を記憶しておくかわりに、行交換に関する情報 (i 行と $i+1$ 行を交換したか否か) と乗数 m_i (i 行の m_i 倍を $i+1$ 行から引く) を記憶しておけばよい。初期ベクトルを $\boldsymbol{x}_0$ とすると、式 (2.27) より、

$$(\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I})\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} \tag{2.32}$$

この両辺に $\boldsymbol{L}$ をかけると

$$\boldsymbol{R}\boldsymbol{x}^{(1)} = \boldsymbol{L}\boldsymbol{x}^{(0)} \tag{2.33}$$

となる。ここで、$\boldsymbol{x}^{(0)}$ は任意のベクトルであるから、$\boldsymbol{x}^{(0)}$ を与えるかわりに $\boldsymbol{L}\boldsymbol{x}^{(0)}$ を与えてもよい。このベクトルとしては、$(1,\ 1,\ \cdots,\ 1)^T$ とするのがよい。式 (2.33) は、後退代入によって解くことができ、$\boldsymbol{x}^{(1)}$ が求まる。次に、

$$\boldsymbol{R}\boldsymbol{x}^{(2)} = \boldsymbol{L}\boldsymbol{x}^{(1)} \tag{2.34}$$

を解く。まず、$\boldsymbol{L}\boldsymbol{x}^{(1)}$ を $(\boldsymbol{T} - \tilde{\lambda}_k \boldsymbol{I})$ の変換の際に記憶しておいた情報を用いて計算する。$\boldsymbol{x}^{(2)}$ は、前と同様に後退代入によって計算できる。この $\boldsymbol{x}^{(2)}$ を λ_k に対する固有ベクトルとして採用する。

多重固有値に対する固有ベクトルについては注意を要する。m 重固有値に対しては、m 個の独立な固有ベクトルが存在し、それらの任意の線形結合も固有ベクトルとなる。ところが、逆反復法で計算すると、固有値が等しければ固有ベクトルも等しくなるので、固有ベクトルは一つしか求められない。互いに独立な固有ベクトルを計算するには、異なる初期ベクトルを用いる方法と、固有値をわずかに異なる値にして計算する方法がある。いずれの場合にも、これらのベクトルは互いに直交しているとは限らな

いので、互いに直交するベクトルが必要ならば、グラム・シュミット (Gram-Schmidt) の直交化を行なわなければならない。

2.1.4 ハウスホルダー逆変換

ハウスホルダー変換により得られた三重対角行列 $\boldsymbol{T}$ は、

$$\boldsymbol{T} = \boldsymbol{P}^{(n-2)} \cdots \boldsymbol{P}^{(1)} \boldsymbol{A} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \tag{2.35}$$

と表される。行列 $\boldsymbol{T}$ の固有値を λ、固有ベクトルを $\boldsymbol{x}$ とすると、

$$\boldsymbol{T}\boldsymbol{x} = \lambda \boldsymbol{x} \tag{2.36}$$

である。これに (2.35) を代入して、

$$\boldsymbol{P}^{(n-2)} \cdots \boldsymbol{P}^{(1)} \boldsymbol{A} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} = \lambda \boldsymbol{x} \tag{2.37}$$

ここで、両辺に左から $\boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)}$ をかける。

$$\boldsymbol{P}^{(r)-1} = \boldsymbol{P}^{(r)} \quad (r = 1,\ 2, \cdots, n-2) \tag{2.38}$$

であることを用いると、

$$\boldsymbol{A} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} = \lambda \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} \tag{2.39}$$

となる。すなわち、

$$\boldsymbol{y} = \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} \tag{2.40}$$

が、固有値 λ に対応する行列 $\boldsymbol{A}$ の固有ベクトルである。

実際の計算手順は次のようになる。 $\boldsymbol{x}^{(n-2)} = \boldsymbol{x}$ として、

$$\begin{aligned} \boldsymbol{x}^{(r-1)} &= \boldsymbol{P}^{(r)} \boldsymbol{x}^{(r)} \\ &= (\boldsymbol{I} - c_r \boldsymbol{u}^{(r)} \boldsymbol{u}^{(r)T}) \boldsymbol{x}^{(r)} \\ &= \boldsymbol{x}^{(r)} - (c_r \boldsymbol{u}^{(r)T} \boldsymbol{x}^{(r)}) \boldsymbol{u}^{(r)} \\ &\qquad (r = n-2,\ n-3, \cdots, 1) \end{aligned} \tag{2.41}$$

とすると、 $\boldsymbol{x}^{(0)}$ が求める固有ベクトル $\boldsymbol{y}$ となる。

第 3 章

設計方法

3.1　設計の概要

本研究では、近年、デジタル回路設計手法として一般的になってきたハードウェア記述言語 HDL(Hardware Description Language) の 1 つである VHDL を用いてプロセッサの機能設計を行う。また、プログラマブルデバイスである FPGA(Field Programmable Gate Array) に設計したプロセッサを実装し、実際の動作を検証する。

3.2　設計の手順

本研究における HDL を用いた LSI 設計の流れを図 3.1に示す。

まず、HDL により回路の機能を記述した HDL ファイルを作成する。そして、機能検証を行い、機能が正しくなければ再び HDL ファイルを作成し直す段階へ戻り、同様の手順に従う。機能が正しければ論理合成を行うことにより HDL ファイルを EDIF(Electronic Design Interchange Format) ファイルに変換する。EDIF ファイルとはデジタル回路を表すフォーマットであり、回路を実際のデバイスで実装するための情報が含まれている。次に、この EDIF ファイルから TTF(Tabular Text File) 形式で表されたデバイスへの配置配線ファイルを生成し、この配置配線ファイルを FPGA へダウンロードする。最後に機能を実装した FPGA により実装検証をし、正しく動作しなければ再び HDL ファイルを作成する段階に戻る。このようにして実際の機能動作が正当なものとなるまで繰り返し設計を行うことになる。

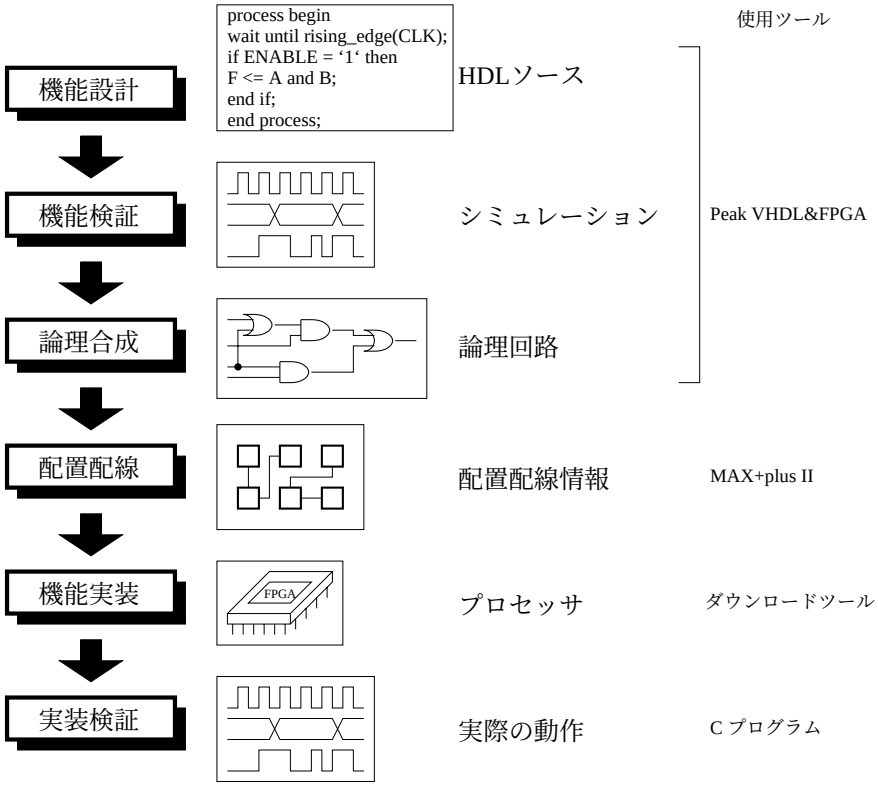


図 3.1　LSI 設計の流れ [1]

3.3　計算システム評価ボード

HDL により設計したプロセッサの機能が実際にハードウェアとして動作するかを検証するため、FPGA を 2 個搭載した評価ボード (松尾 製作) [3] を用いて実装検証を行う。評価ボードの外観を図 3.2に、ブロック図を図 3.3に示す。

本設計では米 ALTERA 社の開発した SRAM 型 FPGA である FLEX10K シリーズの 1 つである EPF10K100 を使用する。ゲート容量は 10 万ゲート (公称値) であり、本研究で想定している浮動小数点数演算器を単精度で単体であれば十分実装できる回路規模である。ボードには FPGA 2 個を中央の上下に配置し、その左右両側にはそれぞれ 1M ビット SRAM が 4 個づつ合計 16 個、72 ピン SIMM DRAM が 1 個づつ合計 4 個配置配線されており、計算においてはこの 2 種類のメモリを主記憶装置として用いることができる。FPGA 同士の通信には FPGA 間に接続されている 56bit のバスを用い

[1] ファイル名：./fig/lsi-flow3.eps

る。また、それぞれの FPGA にはクロックオシレータにより共通のクロックが供給されており、 FPGA 同士で同期設計をすることが可能である。ボード外部へのインターフェース部分は PPI8255 を通してパソコンと接続されており、これを通してパソコンと FPGA 間の通信を行う。

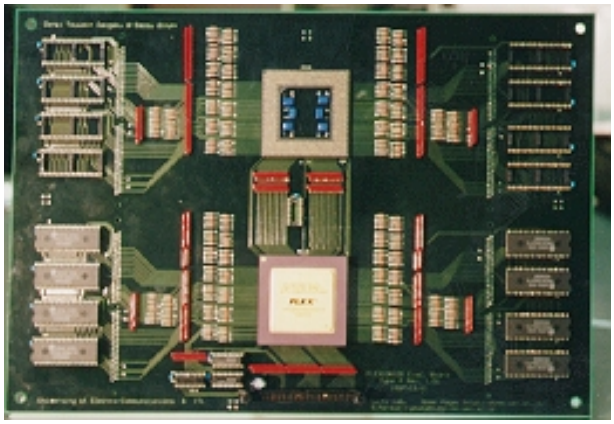


図 3.2　計算システム評価ボード [2]
(下側には FPGA、メモリを積んである状態、上側には積んでいない状態を示す。)

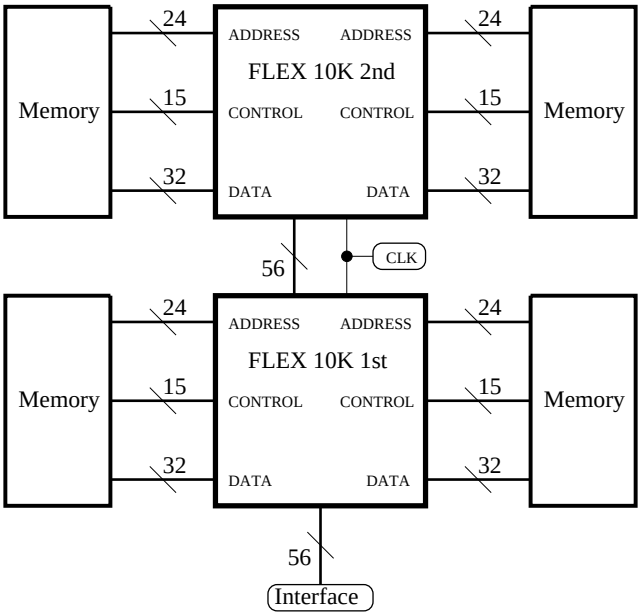


図 3.3　評価ボードのブロック図 [3]

[2]ファイル名：./fig/eval1.eps

[3]ファイル名：./fig/ev-block.eps

3.4 設計方針

本設計では行列の固有値・固有ベクトルを求めるシステムの構築を目的とし、以下のことを念頭に置き設計を行う。

専用コンピュータの設計において重要なことは、頻繁に行われる計算を高速化する、各デバイス間のデータの通信、データ待ちを極力抑えるということである。この設計思想に基づき、また評価ボードの構造を最大限に利用するシステムを検討する。

行列の固有値・固有ベクトルを求めるに際して、行列の次数が大きくなるにしたがい計算量が顕著に増大するのは行列積、行列・ベクトル積である。これらはすべてベクトル積、すなわち積和の計算に帰着されるのであるから積和に着目し、これを高速化するシステムを実現する。

第 4 章

設計モジュール

4.1　メモリコントローラ (SRAM)

数値計算において、多くのデータを扱うためにはメモリが必要である。FPGA にメモリ空間を作ることも可能ではあるが、浮動小数点形式のデータを格納するためには多くのロジックを必要とするため実用的ではない。

そこで、記憶空間には評価ボード上の FPGA の両側に配置配線されている RAM を用いる。ここでは高速である SRAM を使用するためのメモリコントローラを設計した。entity の構成を表 4.1に、 VHDL ソースを付録 I.1に示す。

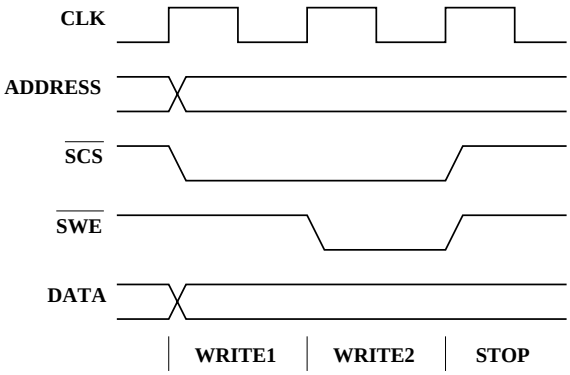
表 4.1　entity の構成

信号名	用途	方向	型
CLK	クロック	in	std_logic
RESET	リセット	in	std_logic
ADRS	アドレスバス	out	std_logic_vector(16 downto 0)
ADRS_BUF	アドレスレジスタ	in	std_logic_vector(16 downto 0)
DATA	データバス	inout	std_logic_vector(31 downto 0)
WR_DATA	書き込みデータ	in	std_logic_vector(31 downto 0)
RD_DATA	読み込みデータ	out	std_logic_vector(31 downto 0)
SCS	チップセレクト	out	std_logic_vector(3 downto 0)
SOE	アウトプットイネーブル	out	std_logic
SWE	ライトイネーブル	out	std_logic
MEM_STATE_SEL	動作選択	in	std_logic_vector(1 downto 0)
WR_CYCLE	ライト終了	out	std_logic
RD_CYCLE	リード終了	out	std_logic

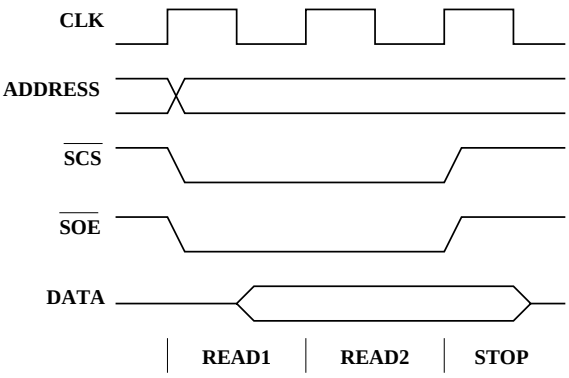
VHDL ソースは 2 つの process から成り立っており、動作はステート・マシンによりシステムクロックに同期して行われる。

最初の process はステート変数に従って制御信号の上げ下げを行い、次の process はステート変数の更新と外部からの動作選択信号のデコードを行い、リードサイクル及びライトサイクルを開始する働きを持つ。

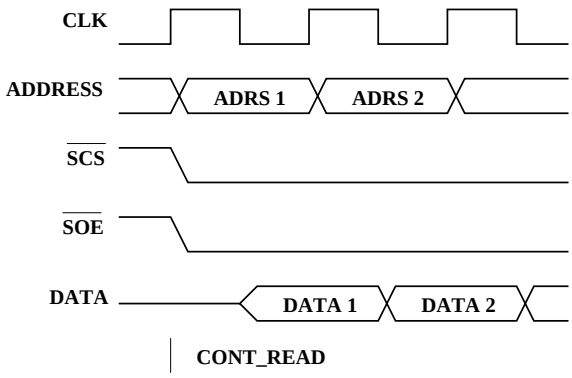
ステートは、ライトサイクルが WRITE1、 WRITE2、 STOP の 3 ステート、リードサイクルが READ1、 READ2、 STOP の 3 ステートにより成り立っている。また、毎クロックごとにアドレスを与え、毎クロックごとにデータを読み込むための CONT_READ のステートを作成した。リードサイクル及びライトサイクルのタイミングを図 4.1に示す。



(a) ライトサイクル



(b) リードサイクル



(c) 連続リードサイクル

図 4.1 リード・ライトサイクル [1]

[1]ファイル名 : (a):./fig/write.eps, (b):./fig/read.eps, (c):./fig/contread.eps

4.2 浮動小数点数演算器

数値計算を行うことは、主に浮動小数点数の四則演算を組み合わせて行うことに帰着される。そこで、浮動小数点数演算器として加算器、乗算器、除算器を設計した。また、ハウスホルダー法を行う際に必要な平方根計算器もあわせて設計した。

計算はIEEE-754の単精度浮動小数点規格(符号1bit, 指数部8bit, 仮数部23bit)に準拠したデータタイプで行う。

また、加算器、乗算器、除算器はcomponentとして使用できるように、平方根計算器はfunctionとして使用できるように設計した。ここで、componentとは回路において新しい階層を形成するものであり、一方、functionとは回路の一部分になるという特徴をもつ。

4.2.1 加算器

entityの構成を表4.2に、VHDLソースを付録I.2.1に示す。

表 4.2 entityの構成

信号名	用途	方向	型
CLK	クロック	in	std_logic
FA	被加数	in	std_logic_vector(31 downto 0)
FB	加数	in	std_logic_vector(31 downto 0)
Q	和	out	std_logic_vector(31 downto 0)

演算は3ステージで構成される同期パイプライン方式を用いている。ステージ構成を図4.2に示す。ここで、入力におけるE1、M1、E2、M2はそれぞれFAの指数部、仮数部、FBの指数部、仮数部を表す。

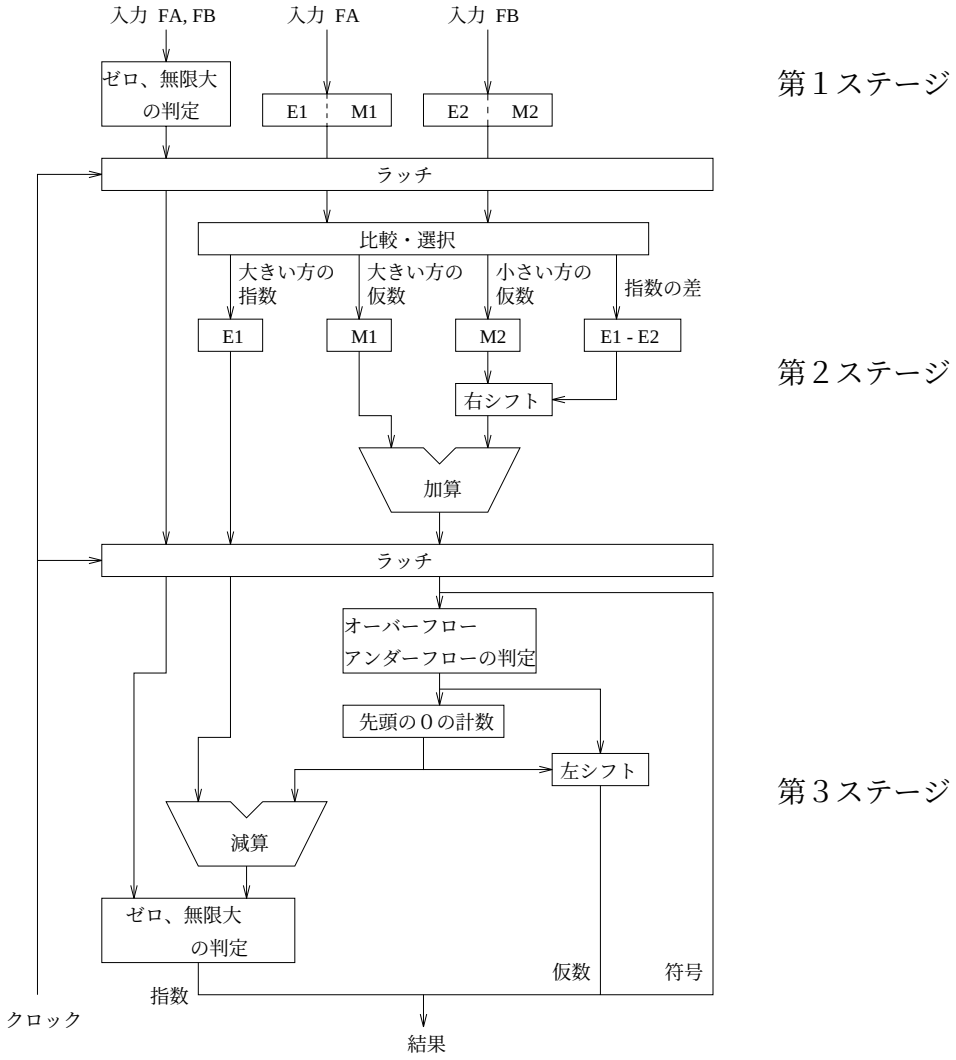


図 4.2　浮動小数点数加算 [2]

VHDL ソースでは上から順にステージが構成されており、図 4.2の構成に対応している。演算動作は組み合わせ回路で記述されており、また、ステージ間のラッチは process 文でシステムクロックに同期して行うように記述されている。

まず、第 1 ステージではFA と FB のゼロ、無限大の判定を行う。第 2 ステージではFA と FB の比較・選択をし、絶対値の小さい方の数の仮数部 (M2) を指数部の差 (E1-E2) だけ右ビットシフトし、 M1 と M2 の加算を行う。第 3 ステージでは仮数部の加算結果のオーバーフロー、あるいはアンダーフローの判定をして結果を絶対値表現に戻し、正規化し、それに基づき指数部を調整して計算結果を外部に出力する。

[2]ファイル名：./fig/add-flow.eps

4.2.2 乗算器

entity の構成を表 4.3に、 VHDL ソースを付録 I.2.2に示す。

表 4.3 entity の構成

信号名	用途	方向	型
CLK	クロック	in	std_logic
FA	被乗数	in	std_logic_vector(31 downto 0)
FB	乗数	in	std_logic_vector(31 downto 0)
Q	積	out	std_logic_vector(31 downto 0)

演算は 3 ステージで構成される同期パイプライン方式を用いている。ステージ構成を図 4.3に示す。ここで、入力における E1、 M1、 E2、 M2 はそれぞれ FA の指数部、仮数部、 FB の指数部、仮数部を表す。

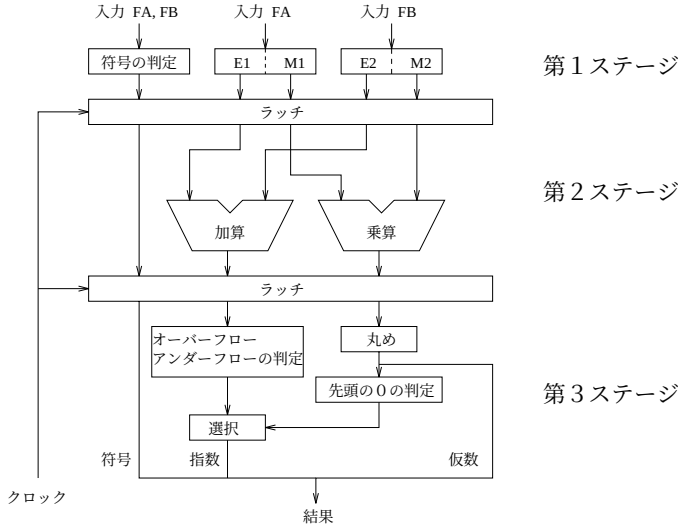


図 4.3 浮動小数点数乗算 [3]

VHDL ソースでは上から順にステージが構成されており、図 4.3の構成に対応している。演算動作は組み合わせ回路で記述されており、また、ステージ間のラッチは process 文でシステムクロックに同期して行うように記述されている。

まず、第 1 ステージでは指数部と仮数部を計算形式にし、また積の符号を判定する。第 2 ステージでは仮数部の乗算をし、不要な下位 22 ビットを切り捨て、また、並行し

[3]ファイル名：./fig/mul-flow.eps

て指数部の加算を行う。第 3 ステージでは仮数部の乗算結果の丸めをし、指数部の加算結果のオーバーフロー、あるいはアンダーフローの判定をして結果を調整し、最後に正規化して計算結果を外部に出力する。

4.2.3　除算器

entity の構成を表 4.4に、 VHDL ソースを付録 I.2.3に示す。

表 4.4　entity の構成

信号名	用途	方向	型
CLK	クロック	in	std_logic
FA	被除数	in	std_logic_vector(31 downto 0)
FB	除数	in	std_logic_vector(31 downto 0)
Q	商	out	std_logic_vector(31 downto 0)

演算は 3 ステージで構成される同期パイプライン方式を用いている。ステージ構成を図 4.4に示す。ここで、入力における E1、 M1、 E2、 M2 はそれぞれ FA の指数部、仮数部、 FB の指数部、仮数部を表す。

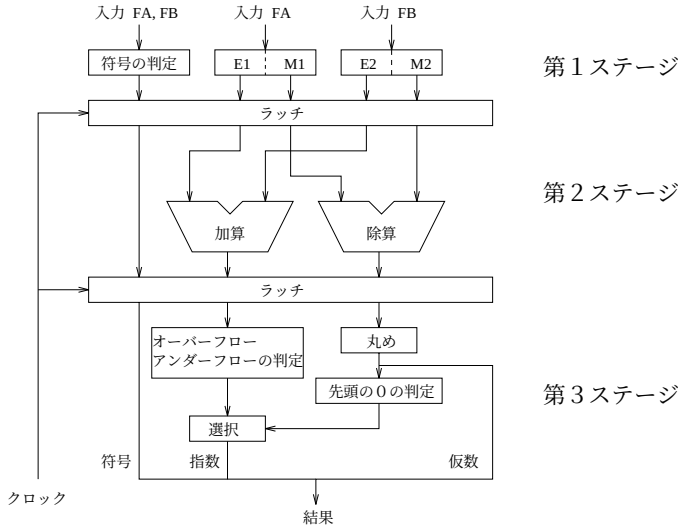


図 4.4　浮動小数点数除算 [4]

[4]ファイル名：./fig/div-flow.eps

VHDL ソースでは上から順にステージが構成されており、図 4.4の構成に対応している。演算動作は組み合わせ回路で記述されており、また、ステージ間のラッチは process 文でシステムクロックに同期して行うように記述されている。

ステージ構成はほぼ乗算器と同様であり、第 2 ステージにおける仮数部と指数部の演算のみが異なる。第 2 ステージでは仮数部の除算をし、また、並行して指数部の減算を行う。

4.2.4 平方根計算器

入出力の構成を表 4.5に、 VHDL ソースを付録 I.2.4に示す。

表 4.5 入出力の構成

信号名	用途	方向	型
FA	被計算数	in	std_logic_vector(31 downto 0)
Q	平方根	out	std_logic_vector(31 downto 0)

VHDL ソースは、入力を FA、出力を Q として function 形式で記述した。ソース中の EQ は指数部の計算結果を表しており、指数部のげた表現をはずして右 1 ビットシフトし、再度げた表現に戻すという操作により指数部を半分にしている。また、 MQ は仮数部の計算結果を表しており、 for 文によって仮数部の開平算を実行して計算を行う。計算結果は必ず正の数として出力し、 $\sqrt{-1}$ などは対応していない。

4.3 ハウスホルダー法

メモリコントローラ、浮動小数点数演算器を用いて行列の固有値・固有ベクトルを求めるアルゴリズムであるハウスホルダー法を行うプロセッサを設計した。

1 つの FPGA にすべての機能を実装すると使用可能なロジックセル数の上限を越えてしまうので、 2 つの FPGA に機能を分散して計算システムを構成した。 FPGA 間は 56bit のバスで接続されており、これにより通信を行う。

主要な機能の実装構成としては、 FLEX 1st には除算器、平方根計算器、ハウスホルダー法アルゴリズムとし、 FLEX 2nd には加算器と乗算器を組み合わせた積和器、

メモリコントローラとする。メモリは主にFLEX 2ndの両側のSRAMを使用する。また、FLEX 1stのハウスホルダー法アルゴリズムを一度にすべて実装することもロジックセル数の制限により困難なので、4つのアルゴリズムを1つずつ実装し計算結果をSRAMに格納しながら計算を進めていく。つまり、ハウスホルダー変換を実装し、計算を実行し、得られた三重対角行列をSRAMに格納する。次に、二分法をを実装し、SRAMに格納されている三重対角行列を利用して計算を実行し、得られた固有値をSRAMに格納するといった操作を4つのアルゴリズムに対して行う。この際、SRAMに格納されたデータは評価ボードに供給されている電源を切らない限り消去しないので、4つのアルゴリズムによる計算を有機的に行うことができる。各アルゴリズムの動作はステート・マシンにより構成され、すべてシステムクロックに同期する。また、本設計モデルではFPGAの左右のSRAMのうち、片方のSRAM容量1Mビット $\times$ 4を利用して最大255 $\times$ 255の行列まで扱うことができる。

4.3.1 積和器

FLEX 2ndに実装するモジュールとして、メモリコントローラに加え、加算器と乗算器を組み合わせた積和器を設計した。entityの構成を表4.6に、VHDLソースを付録I.3.1に示す。

表 4.6 entityの構成

信号名	用途	方向	型
CLK	クロック	in	std_logic
DATA_BUS	FPGA間のデータバス	inout	std_logic_vector(31 downto 0)
ADRS_BUS	FPGA間のアドレスバス	in	std_logic_vector(16 downto 0)
CTRL_BUS	FPGA間のコントロールバス	in	std_logic_vector(4 downto 0)
CALC_DONE	計算終了信号	out	std_logic
OE_ALU	FPGA間のデータバスの方向制御	in	std_logic
R_DATA	右メモリのデータバス	inout	std_logic_vector(31 downto 0)
R_ADRS	右メモリのアドレスバス	out	std_logic_vector(16 downto 0)
R_SCS	右メモリのチップセレクト	out	std_logic_vector(3 downto 0)
R_SOE	右メモリのアウトプットイネーブル	out	std_logic
R_SWE	右メモリのライトイネーブル	out	std_logic
L_DATA	左メモリのデータバス	out	std_logic_vector(31 downto 0)
L_ADRS	左メモリのアドレスバス	out	std_logic_vector(16 downto 0)
L_SCS	左メモリのチップセレクト	out	std_logic_vector(3 downto 0)
L_SOE	左メモリのアウトプットイネーブル	out	std_logic
L_SWE	左メモリのライトイネーブル	out	std_logic

3.4節において説明したように、行列の固有値・固有ベクトルを求める際、頻繁に行われるのは積和計算である。本設計システムでは積和器において積和演算を行う。積和計算でのデータの流れを図 4.5に示す。

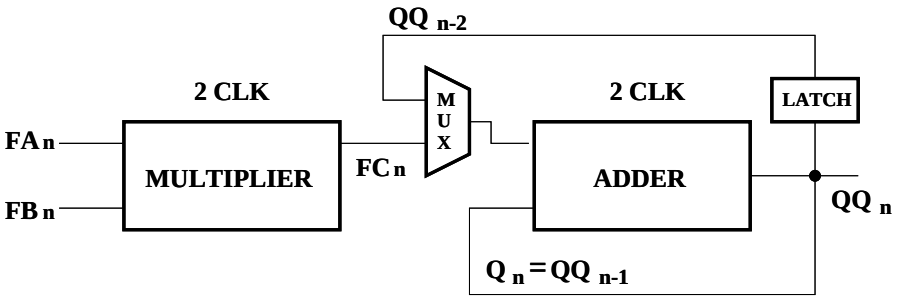


図 4.5 積和計算 [5]

ここではメモリコントローラの連続読み込み機能を使用して、クロック毎にアドレスをメモリに与え、クロック毎に出てくるデータを読み込む。この動作を FPGA の左右両方のメモリについて同時に行い、同時に 2 つのデータ (FA と FB) を得る。そして、FA と FB をまず乗算器に入力し、結果 (FC) を得る。次に、FC を加算器に入力し、今までの積和結果 (Q) をもう一つの入力とする。このようにして積和計算を行うわけだが、加算器、乗算器はデータを入力してから計算結果を得るまでに 2 クロックを要する。このことから、偶数番目に入力したデータと奇数番目に入力したデータの 2 つの積和計算結果が 1 クロック毎に交互に QQ に現れる。よって、計算の最後に偶数番目、奇数番目の積和結果を加算するため、どちらかの結果をラッチによりレジスタに格納しておき、1 クロック後に加算器の入力をこのレジスタへマルチプレクサにより切り替える。そして、この加算結果が最終的な積和結果となる。計算結果はそのままメモリ、あるいは FLEX 1st に送る。このような操作により、各デバイス間のデータの通信、データ待ちを極力抑えることができる。

また、メモリや FLEX 1st から送られてくるデータの加算、乗算等の演算や、メモリのリード、ライト動作も行う。このような動作は FLEX 1st によりすべて制御される。FLEX 1st からの制御信号を表 4.7に示す。この信号は FLEX 1st から FLEX 2nd へ FPGA 間の 5bit のコントロールバス (CTRL_BUS) により伝えることにより、積和器でこの命令を判断し、表 4.7の動作を行う。この動作が終了したならば FLEX 2nd か

[5] ファイル名：./fig/inpro-flow.eps

らFLEX 1stへ計算終了信号(CALC_DONE)を伝える。積和器はハウスホルダー法のすべてのアルゴリズム(ハウスホルダー変換、二分法、逆反復法、ハウスホルダー逆変換)において共通に用いる。

表 4.7 制御信号

信号名	用途
R_MEM_WR	右メモリの書き込み
R_MEM_RD	右メモリの読み込み
L_MEM_WR	左メモリの書き込み
L_MEM_RD	左メモリの読み込み
LR_MEM_WR	左右メモリの書き込み
MEM_STOP	メモリ制御信号のリセット
RR_INPRO	右メモリデータの積和
LL_INPRO	左メモリデータの積和
LR_INPRO	左右メモリデータの積和
INPRO_F	積和結果をFPGAへ転送
INPRO_R	積和結果を右メモリへ転送
INPRO_L	積和結果を左メモリへ転送
INPRO_LR	積和結果を左右メモリへ転送
FF_MUL	FPGAデータの乗算
FR_MUL	FPGA,右メモリデータの乗算
FL_MUL	FPGA,左メモリデータの乗算
RR_MUL	右メモリデータの乗算
LL_MUL	左メモリデータの乗算
LR_MUL	左右メモリデータの乗算
FF_ADD	FPGAデータの加算
FR_ADD	FPGA,右メモリデータの加算
FL_ADD	FPGA,左メモリデータの加算
RR_ADD	右メモリデータの加算
LL_ADD	左メモリデータの加算
LR_ADD	左右メモリデータの加算
CALC_F	計算結果をFPGAへ転送
CALC_R	計算結果を右メモリへ転送
CALC_L	計算結果を左メモリへ転送
CALC_LR	計算結果を左右メモリへ転送

4.3.2 ハウスホルダー変換

FLEX 1stの実装構成は、除算器、平方根計算器、ハウスホルダー変換アルゴリズムである。entityの構成を表4.8に、アルゴリズムのステートの構成を表4.9に、VHDLソースを付録I.3.2に示す。

表 4.8 entity の構成

信号名	用途	方向	型
CLK	クロック	in	std_logic
A	PPI ポート A	inout	std_logic_vector(15 downto 0)
BL	PPI ポート B 下位	in	std_logic_vector(7 downto 0)
BH	PPI ポート B 上位	out	std_logic_vector(5 downto 0)
B_CONF	FPGA コンフィグレーション用バス	in	std_logic_vector(1 downto 0)
CL	PPI ポート C 下位	in	std_logic_vector(5 downto 0)
DATA_BUS	FPGA 間のデータバス	inout	std_logic_vector(31 downto 0)
ADRS_BUS	FPGA 間のアドレスバス	in	std_logic_vector(16 downto 0)
CTRL_BUS	FPGA 間のコントロールバス	in	std_logic_vector(4 downto 0)
CALC_DONE	計算終了信号	out	std_logic
OE_ALU	FPGA 間のデータバスの方向制御	in	std_logic
OBF	PPI の OBF	in	std_logic_vector(1 downto 0)
ACK	PPI の ACK	out	std_logic_vector(1 downto 0)
STB	PPI の STB	out	std_logic_vector(1 downto 0)
IBF	PPI の IBF	in	std_logic_vector(1 downto 0)

表 4.9 ステートの構成

状態名	動作
HsStop	停止、または行列のメモリ書き込み
HsDimWrite	行列の次元のメモリ書き込み
HsVarIni	変数の初期化
HsS2Calc	s^2 の計算
HsSCalc	s の計算
HsAkRead	$a_{k+1,k}$ のメモリ読み込み
HsSNorm	$a_{k+1,k}$ と s の符号を合わせる
HsAkxSCalc	$a_{k+1,k} \times s$ の計算
HsS2pAkxSCalc	$s^2 + a_{k+1,k} \times s$ の計算
HsCCalc	c の計算
HsCWrite	c のメモリ書き込み
HsViceWrite	副対角成分のメモリ書き込み
HsAkpSCalc	$a_{k+1,k} + s$ の計算
HsAxUCalc	$\boldsymbol{Au}$の計算
HsPCalc	$\boldsymbol{p}$の計算
HsAlphaCalc	α の計算
HsAlphaxCCalc	$\alpha \times c$ の計算
HsAlphaxCd2Calc	$\frac{\alpha \times c}{2}$ の計算
HsAlphaxCd2xUCalc	$\frac{\alpha \times c}{2}\boldsymbol{u}$ の計算
HsQCalc	$\boldsymbol{q}$の計算
HsUxQtCalc	$\boldsymbol{uq}^T$ の計算
HsQxUtCalc	$\boldsymbol{qu}^T$ の計算
HsUxQtpQxUtCalc	$\boldsymbol{uq}^T + \boldsymbol{qu}^T$ の計算
HsACalc	$\boldsymbol{A}$の計算
HsResRead	三重対角行列のメモリ読み込み

4.3.3 二分法

FLEX 1st の実装構成は、除算器、二分法アルゴリズムである。entity の構成はハウスホルダー変換と同じである。アルゴリズムのステートの構成を表 4.10に、VHDL ソースを付録 I.3.3に示す。

表 4.10 ステートの構成

状態名	動作
BisStop	停止
BisDimRead	行列の次元のメモリ読み込み
BisAlphaRead	α_k のメモリ読み込み
BisBetaRead	β_k のメモリ読み込み
BisAlphapBetaCalc	$\alpha_k + \beta_k$ の計算
BisMaxCalc	$\alpha_k + \beta_k + \beta_{k-1}$ の計算
BisMaxComp	Gerschgorin の定理による最大値比較
BisMaxSet	固有値存在範囲の設定
BisApBCalc	$a + b$ の計算
BisCCalc	c の計算
BisAlphamCCalc	$\alpha_k - c$ の計算
BisBeta2Calc	β_{k-1}^2 の計算
BisBeta2dQCalc	$\frac{\beta_{k-1}^2}{q_{k-1}}$ の計算
BisQCalc	q_k の計算
BisQComp	q_k の符号比較
BisNewRange	新しい範囲の設定
BisEvWrite	固有値のメモリ書き込み
BisResRead	固有値のメモリ読み込み

4.3.4 逆反復法

FLEX 1st の実装構成は、除算器、逆反復法アルゴリズムである。entity の構成はハウスホルダー変換と同じである。アルゴリズムのステートの構成を表 4.11に、VHDL ソースを付録 I.3.4に示す。

表 4.11 ステートの構成

状態名	動作
InvStop	停止
InvDimRead	行列の次元のメモリ読み込み
InvLambdaRead	固有値の読み込み
InvAlphamLambdaCalc	$\alpha_k - \lambda$ の計算
InvViceRead	β_k のメモリ読み込み
InvViceWrite	β_k のメモリ書き込み
InvAlpha1Read	α_k のメモリ読み込み
InvAlpha2Read	α_{k+1} のメモリ読み込み
InvBeta1Read	β_k のメモリ読み込み
InvBeta2Read	β_k のメモリ読み込み
InvBeta3Read	β_{k+1} のメモリ読み込み
InvAbsComp	ピボットの選択
InvMCalc	m の計算
InvMWrite	m のメモリ書き込み
InvAlpha1Write	α_k のメモリ書き込み
InvBeta2Write	β_k のメモリ書き込み
InvBeta3Write	β_{k+1} のメモリ書き込み
InvMxBeta3Calc	$m \times \beta_{k+1}$ の計算
InvMxBeta2Calc	$m \times \beta_k$ の計算
InvAlpha2mMxBeta2Calc	$\alpha_{k+1} - m \times \beta_k$ の計算
InvRRange	後退代入の範囲の設定
InvRxXCalc	$\boldsymbol{Rx}$の計算
InvXRead	$\boldsymbol{x}$のメモリ読み込み
InvXmRxXCalc	$\boldsymbol{x}$ - $\boldsymbol{Rx}$の計算
InvAlphaRead	α_k のメモリ読み込み
InvXCalc	$\boldsymbol{x}$の計算
InvXWrite	$\boldsymbol{x}$のメモリ書き込み
InvX1Read	x_k のメモリ読み込み
InvX2Read	x_{k+1} のメモリ読み込み
InvX1X2Comp	x_k と x_{k+1} の交換
InvX1Write	x_k のメモリ書き込み
InvMxX1Calc	$m \times x_k$ の計算
InvEvCalc	x_{k+1} の計算
InvResRead	$\boldsymbol{x}$のメモリ読み込み

4.3.5 ハウスホルダー逆変換

FLEX 1st の実装構成は、除算器、平方根計算器、ハウスホルダー逆変換アルゴリズムである。 entity の構成はハウスホルダー変換と同じである。アルゴリズムのステートの構成を表 4.12に、 VHDL ソースを付録 I.3.5に示す。

表 4.12　ステートの構成

状態名	動作
HsInvStop	停止
HsInvDimRead	行列の次元のメモリ読み込み
HsInvVarIni	変数の初期化
HsInvUtxXCalc	$\boldsymbol{u}^T\boldsymbol{x}$ の計算
HsInvUtxXxCCalc	$c\boldsymbol{u}^T\boldsymbol{x}$ の計算
HsInvUtxXxCxUCalc	$c\boldsymbol{u}^T\boldsymbol{x}\boldsymbol{u}$ の計算
HsInvXmUtxXxCxUCalc	$\boldsymbol{x} - c\boldsymbol{u}^T\boldsymbol{x}\boldsymbol{u}$ の計算
HsInvX2Calc	$\sum x_k^2$ の計算
HsInvNormCalc	$\sqrt{\sum x_k^2}$ の計算
HsInvInvNormCalc	$\frac{1}{\sqrt{\sum x_k^2}}$ の計算
HsInvXNorm	$\boldsymbol{y}$の計算
HsInvResRead	固有ベクトルのメモリ読み込み

4.4　ロジックセル使用率

設計した各モジュールのロジックセル使用率を表 4.13に示す。これより、ハウスホルダー法の 4 つのアルゴリズムを一度に実装することはできないが、それぞれ単体で実装可能である。

表 4.13　ロジックセル使用率

モジュール名	ロジックセル数 (最大 4992)
メモリコントローラ	106 (2%)
加算器	852 (17%)
乗算器	1879 (37%)
除算器	1444 (28%)
平方根計算器	658 (13%)
積和器 (メモリコントローラ, 加算器, 除算器を含む)	3973 (79%)
ハウスホルダー変換 (除算器, 平方根計算器を含む)	3725 (74%)
二分法 (除算器を含む)	2868 (57%)
逆反復法 (除算器を含む)	4056 (81%)
ハウスホルダー逆変換 (除算器, 平方根計算器を含む)	2914 (58%)

第 5 章

結果、考察

5.1　行列固有値・固有ベクトル計算

ハウスホルダー法を実装したプロセッサにより行列固有値・固有ベクトルを求めた結果を示す。この結果と直接計算した結果が一致することを確認した。

4×4 行列$\boldsymbol{A}$に対してハウスホルダー変換、二分法、逆反復法、ハウスホルダー逆変換を行った結果を以下に示す。

$$\boldsymbol{A} = \begin{pmatrix} 4.000000 & 3.000000 & 2.000000 & 1.000000 \\ 3.000000 & 3.000000 & 2.000000 & 1.000000 \\ 2.000000 & 2.000000 & 2.000000 & 1.000000 \\ 1.000000 & 1.000000 & 1.000000 & 1.000000 \end{pmatrix}$$

(1) ハウスホルダー変換

$$\text{三重対角行列 } \boldsymbol{T} = \begin{pmatrix} 4.000000 & -3.741657 & 0 & 0 \\ -3.741657 & 5.000002 & 0.4629099 & 0 \\ 0 & 0.4629099 & 0.6666666 & -0.08908703 \\ 0 & 0 & -0.08908703 & 0.3333332 \end{pmatrix}$$

(2) 二分法

$$\text{固有値 } \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{pmatrix} = \begin{pmatrix} 8.290861 \\ 1.000000 \\ 0.4260224 \\ 0.2831185 \end{pmatrix}$$

(3) 逆反復法

$$\begin{aligned} \boldsymbol{T}\text{の固有ベクトル } \boldsymbol{x} &= \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \boldsymbol{x}_3^T \\ \boldsymbol{x}_4^T \end{pmatrix} \\ &= \begin{pmatrix} -1369225 & 1570201 & 9534870 & -1067458 \\ 4332763 & 3473938 & 5003037 & -6685582 \\ -1181856 & -1128892 & 1601669 & -1539424 \\ -1445330 & -1435759 & 2947399 & 5229063 \end{pmatrix} \end{aligned}$$

(4) ハウスホルダー逆変換

$$\begin{aligned} \boldsymbol{A}\text{の固有ベクトル } \boldsymbol{y} &= \begin{pmatrix} \boldsymbol{y}_1^T \\ \boldsymbol{y}_2^T \\ \boldsymbol{y}_3^T \\ \boldsymbol{y}_4^T \end{pmatrix} \\ &= \begin{pmatrix} -0.6565384 & -0.5773503 & -0.4285250 & -0.2280134 \\ 0.5773506 & -0.0000005309556 & -0.5773500 & -0.5773500 \\ -0.4285250 & 0.5773512 & 0.2280118 & -0.6565381 \\ -0.2280124 & 0.5773493 & -0.6565392 & 0.4285259 \end{pmatrix} \end{aligned}$$

6×6 行列$\boldsymbol{A}$に対してハウスホルダー変換、二分法、逆反復法、ハウスホルダー逆変換を行った結果を以下に示す。

$$\boldsymbol{A}=\begin{pmatrix} 10.000000 & 20.000000 & 31.000000 & 4.000000 & 5.000000 & 60.000000 \\ 20.000000 & 7.000000 & 8.000000 & 9.000000 & 10.000000 & 11.000000 \\ 31.000000 & 8.000000 & 12.000000 & 13.000000 & 14.000000 & 15.000000 \\ 4.000000 & 9.000000 & 13.000000 & 16.000000 & 17.000000 & 18.000000 \\ 5.000000 & 10.000000 & 14.000000 & 17.000000 & 19.000000 & 20.000000 \\ 60.000000 & 11.000000 & 15.000000 & 18.000000 & 20.000000 & 21.000000 \end{pmatrix}$$

(1) ハウスホルダー変換

三重対角行列 $\boldsymbol{T}$

$$=\begin{pmatrix} 10.00000 & -70.72481 & 0 & 0 & 0 & 0 \\ -70.72481 & 43.00419 & 35.21627 & 0 & 0 & 0 \\ 0 & 35.21627 & 29.23690 & 0.8454500 & 0 & 0 \\ 0 & 0 & 0.8454500 & 1.218373 & 0.2622349 & 0 \\ 0 & 0 & 0 & 0.2622349 & 0.4411387 & -0.1083632 \\ 0 & 0 & 0 & 0 & -0.1083632 & 1.093406 \end{pmatrix}$$

(2) 二分法

$$\text{固有値 } \lambda=\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix}=\begin{pmatrix} 109.0463 \\ 25.59799 \\ 1.283560 \\ 1.101837 \\ 0.3499792 \\ -52.37977 \end{pmatrix}$$

(3) 逆反復法

$$\boldsymbol{T}\text{の固有ベクトル } \boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \boldsymbol{x}_3^T \\ \boldsymbol{x}_4^T \\ \boldsymbol{x}_5^T \\ \boldsymbol{x}_6^T \end{pmatrix}$$

$$= \begin{pmatrix} -1256402 & 1759526 & 7764629 & 6088069 & 1470305 & -1475.893 \\ 4445266 & -9803808 & 9411994 & 3264309 & 3404087 & -1505342 \\ -3608139 & -4446829 & -6719407 & 2406883 & 8148499 & -4643529 \\ -1778606 & -2237734 & -3305715 & 1193293 & -4236720 & 5447930 \\ 2649438 & 3615015 & 4883011 & -1818983 & 5865281 & 8549355 \\ 7133338 & 6291654 & -2715190 & 4283008 & -2126425 & -4309.194 \end{pmatrix}$$

(4) ハウスホルダー逆変換

$$\boldsymbol{A}\text{の固有ベクトル } \boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1^T \\ \boldsymbol{y}_2^T \\ \boldsymbol{y}_3^T \\ \boldsymbol{y}_4^T \\ \boldsymbol{y}_5^T \\ \boldsymbol{y}_6^T \end{pmatrix}$$

$$= \begin{pmatrix} -0.5469132 & -0.2516926 & -0.3647585 & -0.2666515 & -0.2959788 & -0.5880318 \\ 0.4249735 & -0.04796060 & -0.02780368 & -0.5961965 & -0.6373863 & 0.2336929 \\ -0.01396184 & 0.9227973 & 0.09775608 & -0.1101848 & -0.1046093 & -0.3400149 \\ -0.003179897 & -0.2302671 & 0.8417600 & 0.1673426 & -0.3039847 & -0.3435061 \\ 0.004273052 & 0.09146816 & -0.2899872 & 0.7160429 & -0.6163230 & 0.1222741 \\ 0.7211520 & -0.1463091 & -0.2529213 & 0.1434740 & 0.1514299 & -0.5924933 \end{pmatrix}$$

5.2 計算機性能

ハウスホルダー法により行列固有値・固有ベクトルを求める計算システムにおいて、システムクロック 4MHz で動作することを確認した。本設計システムにおいて加算器、乗算器、除算器、平方根計算器の浮動小数点数演算器は同時に使用することが可能であるので、計算システムのピーク性能は 16MFLOPS である。

しかし、固有値・固有ベクトルが正しく計算されない場合があり、動作が不安定であるという問題が発生した。また、システムクロックを 10MHz としたときには動作しなかった。

動作が不安定である要因としては、評価ボードとパソコン間のデータの送受信にあると考える。行列のデータをパソコンから評価ボードに送信し、パソコンに返信する操作を行ったが、不安定な動作となった。このため、評価ボードとパソコンのインターフェース部を改良する必要がある。

また、性能を決める一つの要因であるシステムクロック速度は、主に浮動小数点数演算器の仮数部演算に左右される。特に除算器の仮数部演算は単精度計算の場合、他の演算器に比べ 5 倍から 10 倍程度時間がかかるので、除算器をなるべく使用しない、演算方式を変えるなどして加算器、乗算器に処理速度を合わせる必要がある。さらに、それぞれの演算器は 3 ステージで構成されるパイプライン方式を用いているが、処理時間の大部分は仮数部演算であり、それぞれのステージの処理時間が均一ではないので、これを均一化し、また、ステージ数を増やすことによって各ステージの処理時間を減少させれば、システムの性能向上が期待できると考える。

第 6 章

結論

本研究ではプログラマブルデバイスである FPGA(容量 10 万ゲート相当)2 個を用いた基盤において、単精度浮動小数点数演算機能を含み、同時処理で最大 16MFLOPS の性能を持つ数値計算プロセッサが実装できることを示した。また、使用した評価ボードにおいて、 FPGA の再構成可能論理を利用し、行列の固有値・固有ベクトルを求めるための動的な計算システムを構築した。

従来はコンピュータアーキテクチャの研究に際して、専門のベンダなどでなければ自らの研究に向いたアーキテクチャのマシンを製作したり、提案したアーキテクチャを容易に実機にすることは困難であった。しかし、大容量のプログラマブルデバイスが提供されるようになり、本研究で実践したような計算システムの評価はその容易性、高速性から有効であると考える。

また、本研究では各単精度浮動小数点数演算器を、デバイスの容量の制限により単一で実装するにとどまったが、これからの LSI 技術の進歩によるプログラマブルデバイスの高速化、大容量化に伴い、多くの物性計算に必要な倍精度浮動小数点数演算器の実装、さらには複数実装を可能にし、オンチップマルチプロセッサシステムとしてのコンピュータアーキテクチャの研究が発展していくものと思われる。

謝 辞

本研究及び論文作成にあたり、終始懇切なる御指導、御助言をして頂きました指導教官である齋藤理一郎 助教授に心より御礼を申し上げます。

また、本研究を進めるにあたり貴重な御意見、御検討をして頂きました木村忠正 教授、湯郷成美 助教授、一色秀夫 助手に深く感謝申し上げます。

また、本研究において必要不可欠な評価ボードを製作し、そして多くの御助言をして頂きました松尾竜馬氏に感謝の意を表します。

また、研究活動を共にし、多くの援助をして頂いた八木将志氏、沼知典氏、安藤泰夫氏、田代哲正氏をはじめ木村・齋藤研究室、湯郷研究室の皆様に感謝致します。

そして、多くの技術支援をして頂きました (株) 日本アルテラ 浮谷光明氏、 (株) インターリンク 倉重克己氏に感謝致します。

最後に、事務業務をして頂きました山本純子さんに感謝致します。

参 考 文 献

[1] 中島 瑞樹, “超高速行列演算チップの開発”, 1996 年度 卒業論文.

[2] 八木 将志, “大行列の対角化プログラムの並列化”, 1996 年度 卒業論文.

[3] 松尾 竜馬, “行列計算専用大規模集積回路の開発”, 1997 年度 卒業論文.

[4] グェン ドゥック ミン, “ハードウェア記述言語を用いた行列計算専用プロセッサの設計”, 1997 年度 卒業論文.

[5] 高田 勝, 春海 佳三郎 編著, “数値計算の手順と実際” , コロナ社, 1984.

[6] Ben Cohen, “VHDL Coding Styles and Methodologies”, Kluwer Academic Publishers, 1997.

[7] 尾形, 山本, 水尾, 木村, 笠原, “FPGA を用いたマルチプロセッサシステムテストベッドの実装”, 情報処理学会 ARC-128-14, pp. 79-84, 1997.

[8] 尾形, 水尾, 村上, “EPF10K250 を用いたオンチップマルチプロセッサエミュレータの実装”, ALTERA PLD WORLD’98 技術論文集, pp. 239-248, 1998.

付録 I

プログラムソース

I.1　メモリコントローラ (SRAM)

```
------------------------------------------------------------
-- Memory Controller  (FLEX10k)
-- < memctrl.vhd >
-- 1999/01/21 (Thu)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity memctrl is
port (  CLK           : in std_logic;
        RESET         : in std_logic;
        ADRS          : out std_logic_vector(16 downto 0);
        ADRS_BUF      : in std_logic_vector(16 downto 0);
        DATA          : inout std_logic_vector(31 downto 0);
        WR_DATA       : in std_logic_vector(31 downto 0);
        RD_DATA       : out std_logic_vector(31 downto 0);
        SCS           : out std_logic_vector(3 downto 0);
        SOE           : out std_logic;
        SWE           : out std_logic;
        MEM_STATE_SEL : in std_logic_vector(1 downto 0);
        WR_CYCLE      : out std_logic;
        RD_CYCLE      : out std_logic
);
end memctrl;
architecture RTL of memctrl is

type MEM_STATE_TYPE is (
  STOP, WRITE1, WRITE2,
  READ1, READ2, CONT_READ
  );

signal CURRENT_STATE  : MEM_STATE_TYPE;
signal NEXT_STATE     : MEM_STATE_TYPE;
signal NEXT_MEM_CYCLE : std_logic;
signal OE             : std_logic;
signal DATA_BUF       : std_logic_vector(31 downto 0);

constant WRITE_CYCLE     : std_logic_vector(1 downto 0) := "00";
constant READ_CYCLE      : std_logic_vector(1 downto 0) := "01";
```

```
constant CONT_READ_CYCLE : std_logic_vector(1 downto 0) := "10";

begin

DATA <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE = '1' else DATA_BUF;

process ( RESET, CLK ) begin
if RESET = '1' then
        SOE <= '1';
        SWE <= '1';
        SCS <= "1111";
        OE <= '1';
        ADRS <= ( others => '0' );
        DATA_BUF <= ( others => '0' );
        RD_DATA <= ( others => '0' );
        WR_CYCLE <= '0';
        RD_CYCLE <= '0';
        NEXT_MEM_CYCLE <= '0';
        NEXT_STATE <= STOP;
elsif rising_edge( CLK ) then
case CURRENT_STATE is

    when STOP =>
        SOE <= '1';
        SWE <= '1';
        SCS <= "1111";
        OE <= '1';
        WR_CYCLE <= '0';
        RD_CYCLE <= '0';
        NEXT_MEM_CYCLE <= '0';

    when WRITE1 =>
        SCS <= "0000";
        ADRS <= ADRS_BUF;
        DATA_BUF <= WR_DATA;
        OE <= '0';
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= WRITE2;

    when WRITE2 =>
        SWE <= '0';
        WR_CYCLE <= '1';
        NEXT_STATE <= STOP;

    when READ1 =>
        SCS <= "0000";
        SOE <= '0';
        ADRS <= ADRS_BUF;
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= READ2;

    when READ2 =>
        RD_DATA <= DATA;
        RD_CYCLE <= '1';
        NEXT_STATE <= STOP;

    when CONT_READ =>
        SOE  <= '0';
        SWE  <= '1';
        SCS  <= "0000";
        OE   <= '1';
        ADRS <= ADRS_BUF;

    when others =>
        NEXT_STATE <= STOP;

end case;
end if;
end process;

process ( RESET, CLK ) begin
if RESET = '1' then
        CURRENT_STATE <= STOP;
elsif falling_edge( CLK ) then
        if NEXT_MEM_CYCLE = '0' then
                case MEM_STATE_SEL is
```

```
                        when WRITE_CYCLE =>
                                CURRENT_STATE <= WRITE1;
                        when READ_CYCLE =>
                                CURRENT_STATE <= READ1;
                        when CONT_READ_CYCLE =>
                                CURRENT_STATE <= CONT_READ;
                        when others =>
                                CURRENT_STATE <= STOP;
                end case;
        else
                 CURRENT_STATE <= NEXT_STATE;
        end if;
end if;
end process;
end RTL;
```

I.2 単精度浮動小数点数演算器

I.2.1 加算器

```
------------------------------------------------------------
-- Floating Point Number Adder  (FLEX10k)
-- < fpadd.vhd >
-- 1998/11/17 (Tue)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fpadd is
  port ( CLK  : in std_logic;
         FA   : in std_logic_vector(31 downto 0);
         FB   : in std_logic_vector(31 downto 0);
         Q    : out std_logic_vector(31 downto 0)
      );

end fpadd;
architecture RTL of fpadd is

signal INF1, INF2, INF3, ZA1, ZA2, ZA3, ZB1, ZB2, ZB3 : std_logic;
signal SA1, SA2, SA3, SB1, SB2, SB3 : std_logic;
signal EA1, EA2, EB1, EB2 : std_logic_vector(8 downto 0);
signal EA3, EA4, EQ, ED1, ED2 : std_logic_vector(7 downto 0);
signal MA1, MA2, MA3, MB1, MB2, MB5 : std_logic_vector(22 downto 0);
signal MB3 : std_logic_vector(23 downto 0);
signal MA4, MB4 : std_logic_vector(25 downto 0);
signal MQ1, MQ2 : std_logic_vector(25 downto 0);
signal MQ3, MQ4 : std_logic_vector(24 downto 0);
signal MQ5 : std_logic_vector(22 downto 0);
signal V0, V1, V2, V3, V4, V5 : std_logic_vector(24 downto 0);
signal VV0, VV1, VV2, VV3, VV4 : std_logic_vector(24 downto 0);
signal VES1, VES2 : std_logic_vector(8 downto 0);

begin

INF1 <= '1' when FA(30 downto 23) = "11111111" or
FB(30 downto 23) = "11111111" else '0';
ZA1 <= '1' when FA(30 downto 23) = "00000000" else '0';
ZB1 <= '1' when FB(30 downto 23) = "00000000" else '0';

EA1 <= '0' & FA(30 downto 23);
EB1 <= '0' & FB(30 downto 23);

process begin
wait until rising_edge( CLK );
```

```
        SA1 <= FA(31);
        SB1 <= FB(31);
        EA2 <= EA1;
        EB2 <= EB1;
        MA1 <= FA(22 downto 0);
        MB1 <= FB(22 downto 0);
        INF2 <= INF1;
        ZA2 <= ZA1;
        ZB2 <= ZB1;
end process;

VES1 <= EA2 - EB2;
VES2 <= EB2 - EA2;

SA2 <= SA1 when VES1(8) = '0' else SB1;
SB2 <= SB1 when VES1(8) = '0' else SA1;

EA3 <= EA2(7 downto 0) when VES1(8) = '0' else EB2(7 downto 0);
ED1 <= VES1(7 downto 0) when VES1(8) = '0' else VES2(7 downto 0);

MA2 <= MA1 when VES1(8) = '0' else MB1;
MB2 <= MB1 when VES1(8) = '0' else MA1;

ZA3 <= ZA2 when VES1(8) = '0' else ZB2;
ZB3 <= ZB2 when VES1(8) = '0' else ZA2;

V0 <= "1" & MB2 & "0" when ED1(0) = '0' else "01" & MB2(22 downto 1) & "0";
V1 <= V0 when ED1(1) = '0' else "00" & V0(24 downto 2);
V2 <= V1 when ED1(2) = '0' else "0000" & V1(24 downto 4);
V3 <= V2 when ED1(3) = '0' else "00000000" & V2(24 downto 8);
V4 <= V3 when ED1(4) = '0' else "0000000000000000" & V3(24 downto 16);
V5 <= V4 + "00000000000000000000000001";
MB3 <= V5(24 downto 1) when ED1(7 downto 5) = "000" else "000000000000000000000000";

MA4 <= "00000000000000000000000000" when ZA3 = '1' else
       "001" & MA2 when SA2 = '0' else
       "00000000000000000000000000" - ("001" & MA2);

MB4 <= "00000000000000000000000000" when ZB3 = '1' else
       "00" & MB3 when SB2 = '0' else
       "00000000000000000000000000" - ("00" & MB3);

MQ1 <= MA4 + MB4;

process begin
wait until rising_edge( CLK );
        EA4 <= EA3;
        MQ2 <= MQ1;
        INF3 <= INF2;
end process;

MQ3 <= MQ2(24 downto 0) when MQ2(25) = '0' else
       "0000000000000000000000000" - MQ2(24 downto 0);

MQ4 <= MQ3 + "00000000000000000000000001";

ED2 <=  "00000000" when MQ3(24) = '1' else
        "00000001" when MQ3(23) = '1' else
        "00000010" when MQ3(22) = '1' else
        "00000011" when MQ3(21) = '1' else
        "00000100" when MQ3(20) = '1' else
        "00000101" when MQ3(19) = '1' else
        "00000110" when MQ3(18) = '1' else
        "00000111" when MQ3(17) = '1' else
        "00001000" when MQ3(16) = '1' else
        "00001001" when MQ3(15) = '1' else
        "00001010" when MQ3(14) = '1' else
        "00001011" when MQ3(13) = '1' else
        "00001100" when MQ3(12) = '1' else
        "00001101" when MQ3(11) = '1' else
        "00001110" when MQ3(10) = '1' else
        "00001111" when MQ3( 9) = '1' else
        "00010000" when MQ3( 8) = '1' else
        "00010001" when MQ3( 7) = '1' else
        "00010010" when MQ3( 6) = '1' else
        "00010011" when MQ3( 5) = '1' else
```

```
        "00010100" when MQ3( 4) = '1' else
        "00010101" when MQ3( 3) = '1' else
        "00010110" when MQ3( 2) = '1' else
        "00010111" when MQ3( 1) = '1' else
        "00011000" when MQ3( 0) = '1' else
        "10000000";

VV0 <= MQ3 when ED2(0) = '0' else MQ3(23 downto 0) & "0";
VV1 <= VV0 when ED2(1) = '0' else VV0(22 downto 0) & "00";
VV2 <= VV1 when ED2(2) = '0' else VV1(20 downto 0) & "0000";
VV3 <= VV2 when ED2(3) = '0' else VV2(16 downto 0) & "00000000";
VV4 <= VV3 when ED2(4) = '0' else VV3( 8 downto 0) & "0000000000000000";
MQ5 <= VV4(23 downto 1) when MQ4(24) = '0' else MQ4(23 downto 1);

EQ <= "11111111" when INF3 = '1' else
      EA4 - ED2 + "00000001" when ED2(7) = '0' else
      "00000000";

Q <= MQ2(25) & EQ & MQ5;
end RTL;
```

I.2.2 乗算器

```
------------------------------------------------------------
-- Floating Point Number Multiplier (FLEX10k)
-- < fpmult.vhd >
-- 1999/01/19 (Tue)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fpmult is
  port ( CLK  : in std_logic;
         FA   : in std_logic_vector(31 downto 0);
         FB   : in std_logic_vector(31 downto 0);
         Q    : out std_logic_vector(31 downto 0)
       );

end fpmult;
architecture RTL of fpmult is

signal MA1, MA2, MB1, MB2 : std_logic_vector(23 downto 0);
signal EA1, EA2, EB1, EB2 : std_logic_vector(9 downto 0);
signal MQ1, MQ2, MQ3      : std_logic_vector(25 downto 0);
signal EQ1, EQ2, EQ3, EQ4 : std_logic_vector(9 downto 0);
signal EQ5, EQ6, EQ       : std_logic_vector(7 downto 0);
signal MQ                 : std_logic_vector(22 downto 0);
signal S1, S2, SQ         : std_logic;

begin

MA1 <= '1' & FA(22 downto 0);
MB1 <= '1' & FB(22 downto 0);
EA1 <= "00" & FA(30 downto 23);
EB1 <= "00" & FB(30 downto 23);
S1  <= FA(31) xor FB(31);

process begin
wait until rising_edge( CLK );
        MA2 <= MA1;
        MB2 <= MB1;
        EA2 <= EA1;
        EB2 <= EB1;
        S2  <= S1;
end process;

process( MA2, MB2 )
        variable TMQ1 : std_logic_vector(47 downto 0);
```

```
begin
        TMQ1 := MA2 * MB2;
        MQ1 <= TMQ1(47 downto 22);
end process;

EQ1 <= "0011111111" when EA2(7 downto 0) = "11111111" or
EB2(7 downto 0) = "11111111" else
       "0000000000" when EA2(7 downto 0) = "00000000" or
EB2(7 downto 0) = "00000000" else
       EA2 + EB2 - "0001111111";

EQ2 <= "0011111111" when EA2(7 downto 0) = "11111111" or
EB2(7 downto 0) = "11111111" else
       "0000000000" when EA2(7 downto 0) = "00000000" or
EB2(7 downto 0) = "00000000" else
       EA2 + EB2 - "0001111110";

process begin
wait until rising_edge( CLK );
        SQ <= S2;
        MQ2 <= MQ1;
        EQ3 <= EQ1;
        EQ4 <= EQ2;
end process;

MQ3 <= MQ2 + "00000000000000000000000001" when MQ2(25) = '0' else
       MQ2 + "00000000000000000000000010";

EQ5 <= "00000000" when EQ3(9 downto 8) = "11" else
       "11111111" when EQ3(9 downto 8) = "01" else
       EQ3(7 downto 0);

EQ6 <= "00000000" when EQ4(9 downto 8) = "11" else
       "11111111" when EQ4(9 downto 8) = "01" else
       EQ4(7 downto 0);

MQ <= MQ3(23 downto 1) when MQ3(25) = '0' else
      MQ3(24 downto 2);

EQ <= EQ5 when MQ3(25) = '0' else
      EQ6;

Q <= SQ & EQ & MQ;
end RTL;
```

I.2.3 除算器

```
------------------------------------------------------------
-- Floating Point Number Divider   (FLEX10k)
-- < fpdiv.vhd >
-- 1999/01/18 (Mon)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fpdiv is
  port ( CLK  : in std_logic;
         FA   : in std_logic_vector(31 downto 0);
         FB   : in std_logic_vector(31 downto 0);
         Q    : out std_logic_vector(31 downto 0)
      );
end fpdiv;

architecture RTL of fpdiv is

signal MA1, MA2, MB1, MB2 : std_logic_vector(23 downto 0);
signal EA1, EA2, EB1, EB2 : std_logic_vector(9 downto 0);
signal MQ1, MQ2, MQ3      : std_logic_vector(25 downto 0);
signal EQ1, EQ2, EQ3, EQ4 : std_logic_vector(9 downto 0);
```

```
signal EQ5, EQ6, EQ       : std_logic_vector(7 downto 0);
signal MQ                 : std_logic_vector(22 downto 0);
signal S1, S2, SQ         : std_logic;

begin

MA1 <= '1' & FA(22 downto 0);
MB1 <= '1' & FB(22 downto 0);
EA1 <= "00" & FA(30 downto 23);
EB1 <= "00" & FB(30 downto 23);
S1  <= FA(31) xor FB(31);

process begin
wait until rising_edge( CLK );
        MA2 <= MA1;
        MB2 <= MB1;
        EA2 <= EA1;
        EB2 <= EB1;
        S2  <= S1;
end process;

process( MA2, MB2 )
        variable MA3, MB3, REMAIN : std_logic_vector(24 downto 0);
begin
        MA3 := '0' & MA2;
        MB3 := '0' & MB2;
        for i in 25 downto 0 loop
                REMAIN := MA3 - MB3;
                 if REMAIN(24) = '1' then
                         MQ1(i) <= '0';
                       MA3 := MA3(23 downto 0) & '0';
                  else
                         MQ1(i) <= '1';
                       MA3 := REMAIN(23 downto 0) & '0';
                  end if;
        end loop;
end process;

EQ1 <= "0011111111" when EA2(7 downto 0) = "11111111" or
EB2(7 downto 0) = "00000000" else
       "0000000000" when EA2(7 downto 0) = "00000000" or
EB2(7 downto 0) = "11111111" else
       EA2 - EB2 + "0001111110";

EQ2 <= "0011111111" when EA2(7 downto 0) = "11111111" or
EB2(7 downto 0) = "00000000" else
       "0000000000" when EA2(7 downto 0) = "00000000" or
EB2(7 downto 0) = "11111111" else
       EA2 - EB2 + "0001111111";

process begin
wait until rising_edge( CLK );
        SQ <= S2;
        MQ2 <= MQ1;
        EQ3 <= EQ1;
        EQ4 <= EQ2;
end process;

MQ3 <= MQ2 + "00000000000000000000000001" when MQ2(25) = '0' else
       MQ2 + "00000000000000000000000010";

EQ5 <= "00000000" when EQ3(9 downto 8) = "11" else
       "11111111" when EQ3(9 downto 8) = "01" else
       EQ3(7 downto 0);

EQ6 <= "00000000" when EQ4(9 downto 8) = "11" else
       "11111111" when EQ4(9 downto 8) = "01" else
       EQ4(7 downto 0);

MQ <= MQ3(23 downto 1) when MQ3(25) = '0' else
      MQ3(24 downto 2);

EQ <= EQ5 when MQ3(25) = '0' else
      EQ6;
```

```
Q <= SQ & EQ & MQ;
end RTL;
```

I.2.4 平方根計算器

```
-----------------------------------------------------------------
-- Square Root Calculator   (FLEX10k)
-- < fpsqrt.vhd >
-- 1998/11/11 (Wed)
-- yamaoka@tube.ee.uec.ac.jp
-----------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

package MATH is
        function sqrt( FA : std_logic_vector ) return std_logic_vector;
end MATH;

package body MATH is
        function sqrt( FA : std_logic_vector ) return std_logic_vector is
        variable MA : std_logic_vector(25 downto 0);
        variable MQ : std_logic_vector(24 downto 0);
        variable EQ : std_logic_vector(7 downto 0);
        variable TMP1, TMP2, REMAIN : std_logic_vector(27 downto 0);
begin

EQ := FA(30 downto 23) - "01111111";
EQ := ( EQ(7) & EQ(7 downto 1) ) + "01111111";

if FA(23) = '1' then
        MA := "01" & FA(22 downto 0) & '0';
else
        MA := '1' & FA(22 downto 0) & "00";
end if;

TMP1 := "0000000000000000000000000000" & MA( 25 downto 24 );
TMP2 := "0000000000000000000000000001";

for I in 0 to 24 loop
        REMAIN := TMP1 - TMP2;
        if REMAIN(27) = '0' then
                MQ(24-I) := '1';
                if I <= 11 then
                        TMP1( (I+3) downto 0 )
:= REMAIN( (I+1) downto 0 ) & MA( (23-(2*I)) downto (22-(2*I)));
                else
                        TMP1( (I+3) downto 0 ) := REMAIN( (I+1) downto 0 ) & "00";
                end if;
                TMP2( (I+3) downto 0 ) := TMP2( (I+2) downto 1 ) & "01";
                TMP2(2) := '1';
        else
                MQ(24-I) := '0';
                if I <= 11 then
                        TMP1( (I+3) downto 0 )
:= TMP1( (I+1) downto 0 ) & MA( (23-(2*I)) downto (22-(2*I)));
                else
                        TMP1( (I+3) downto 0 ) := TMP1( (I+1) downto 0 ) & "00";
                end if;
                TMP2( (I+3) downto 0 ) := TMP2( ( I+2 ) downto 1) & "01";
        end if;
end loop;
MQ := MQ + "00000000000000000000000001";
return '0' & EQ & MQ(23 downto 1);
end sqrt;
end MATH;
```

I.3 ハウスホルダー法

I.3.1 積和器

```
------------------------------------------------------------
-- ALU for Householder Method  (upper FLEX10k)
-- < alu3.vhd >
-- 1999/01/22 (Fri)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity alu3 is
  port (CLK    : in std_logic;

                DATA_BUS  : inout std_logic_vector(31 downto 0);
                ADRS_BUS  : in std_logic_vector(16 downto 0);
                CTRL_BUS  : in std_logic_vector(4 downto 0);
                CALC_DONE : out std_logic;
                OE_ALU    : in std_logic;

                R_DATA : inout std_logic_vector(31 downto 0);
                R_ADRS : out std_logic_vector(16 downto 0);
                R_SCS  : out std_logic_vector(3 downto 0);
                R_SOE  : out std_logic;
                R_SWE  : out std_logic;

                L_DATA : inout std_logic_vector(31 downto 0);
                L_ADRS : out std_logic_vector(16 downto 0);
                L_SCS  : out std_logic_vector(3 downto 0);
                L_SOE  : out std_logic;
                L_SWE  : out std_logic
);

attribute pinnum of CLK : signal is  "AY22";

attribute pinnum of DATA_BUS  : signal is "BC5,BB6,BC7,BB8,BC9,BB10,BC11,
BB12,BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20,BC23,BB24,BC25,BB26,BC27,
BB28,BC29,BB30,BC31,BB32,BC33,BB34,BC35,BB36,BC37,BB38";
attribute pinnum of ADRS_BUS  : signal is
"AU15,AV18,AU19,AV20,AU21,AV22,AU23,AV24,AU25,AV28,AU29,AV30,AU31,AV32,
AU33,AV34,AU35";
attribute pinnum of CTRL_BUS  : signal is  "AV10,AU11,AV12,AU13,AV14";
attribute pinnum of CALC_DONE : signal is  "AU9";
attribute pinnum of OE_ALU    : signal is  "AV8";

attribute pinnum of R_ADRS : signal is "T38,W37,Y38,AA37,AB38,AC37,AD38,
AE37,AF38,AJ37,AK38,AL37,AM38,AN37,AP38,AR37,AT38";
attribute pinnum of R_DATA : signal is "F42,G43,H42,J43,K42,L43,M42,N43,
T42,U43,V42,W43,Y42,AA43,AB42,AC43,AF42,AG43,AH42,AJ43,AK42,AL43,AM42,
AN43,AT42,AU43,AV42,AW43,AY42,BA43,BB42,BC43";
attribute pinnum of R_SCS  : signal is  "AG39,AH40,AJ39,AM40";
attribute pinnum of R_SOE  : signal is  "AP40";
attribute pinnum of R_SWE  : signal is  "AN39";

attribute pinnum of L_ADRS : signal is "T6,W7,Y6,AA7,AB6,AC7,AD6,AE7,AF6,
AJ7,AK6,AL7,AM6,AN7,AP6,AR7,AT6";
attribute pinnum of L_DATA : signal is "F2,G1,H2,J1,K2,L1,M2,N1,T2,U1,V2,
W1,Y2,AA1,AB2,AC1,AF2,AG1,AH2,AJ1,AK2,AL1,AM2,AN1,AT2,AU1,AV2,AW1,AY2,BA1,BB2,BC1";
attribute pinnum of L_SCS  : signal is  "AG5,AH4,AJ5,AM4";
attribute pinnum of L_SOE  : signal is  "AP4";
attribute pinnum of L_SWE  : signal is  "AN5";

end alu3;
architecture RTL of alu3 is
```

```
--------------< Memory Controller >------------
component memctrl is
port (  CLK           : in std_logic;
                RESET         : in std_logic;
                ADRS          : out std_logic_vector(16 downto 0);
                ADRS_BUF      : in std_logic_vector(16 downto 0);
                DATA          : inout std_logic_vector(31 downto 0);
                WR_DATA       : in std_logic_vector(31 downto 0);
                RD_DATA       : out std_logic_vector(31 downto 0);
                SCS           : out std_logic_vector(3 downto 0);
                SOE           : out std_logic;
                SWE           : out std_logic;
                MEM_STATE_SEL : in std_logic_vector(1 downto 0);
                WR_CYCLE      : out std_logic;
                RD_CYCLE      : out std_logic
);
end component;

-------------< 32-bit Floating point Number Multiplier >-------
component fpmult is
port (  CLK : in std_logic;
                FA  : in std_logic_vector(31 downto 0);
                FB  : in std_logic_vector(31 downto 0);
                Q   : out std_logic_vector(31 downto 0)
      );
end component;

-------------< 32-bit Floating point Number Adder >-------
component fpadd is
port (  CLK : in std_logic;
                FA  : in std_logic_vector(31 downto 0);
                FB  : in std_logic_vector(31 downto 0);
                Q   : out std_logic_vector(31 downto 0)
      );
end component;

signal MUL_A : std_logic_vector(31 downto 0);
signal MUL_B : std_logic_vector(31 downto 0);
signal MUL_Q : std_logic_vector(31 downto 0);

signal ADD_A : std_logic_vector(31 downto 0);
signal ADD_B : std_logic_vector(31 downto 0);
signal ADD_Q : std_logic_vector(31 downto 0);
signal ADD_A_BUF  : std_logic_vector(31 downto 0);
signal ADD_B_BUF  : std_logic_vector(31 downto 0);

signal RESET : std_logic;

signal R_ADRS_BUF : std_logic_vector(16 downto 0);
signal L_ADRS_BUF : std_logic_vector(16 downto 0);
signal R_MEM_STATE_SEL : std_logic_vector(1 downto 0);
signal L_MEM_STATE_SEL : std_logic_vector(1 downto 0);
signal R_RD_CYCLE : std_logic;
signal L_RD_CYCLE : std_logic;
signal R_WR_CYCLE : std_logic;
signal L_WR_CYCLE : std_logic;

signal R_WR_DATA : std_logic_vector(31 downto 0);
signal R_RD_DATA : std_logic_vector(31 downto 0);
signal L_WR_DATA : std_logic_vector(31 downto 0);
signal L_RD_DATA : std_logic_vector(31 downto 0);

signal DCNT : std_logic;
signal DCNT1 : std_logic;
signal DCNT2 : std_logic_vector(3 downto 0);
signal DCNT3 : std_logic;
signal DCNT4 : std_logic_vector(2 downto 0);
signal DCNT5 : std_logic;
signal CALC1_ACK : std_logic;
signal CALC3_ACK : std_logic;
```

```
signal CALC4_ACK : std_logic;

signal DATA_BUF : std_logic_vector(31 downto 0);
signal CALC_CNT : std_logic;
signal FB_Q : std_logic_vector(31 downto 0);

type DATA_BUS_SEL_TYPE is (
        dR_MEM_RD, dL_MEM_RD,
        dINPRO_F,
        dMUL, dADD,
        dSTOP
    );

type LATCH_SEL_TYPE is (
        RR_MEM, LR_MEM, LL_MEM,
        FR_MEM, FL_MEM, FF_MEM,
        lSTOP
    );

signal DATA_BUS_SEL : DATA_BUS_SEL_TYPE;
signal LATCH_SEL : LATCH_SEL_TYPE;

constant WRITE_CYCLE     : std_logic_vector(1 downto 0) := "00";
constant READ_CYCLE      : std_logic_vector(1 downto 0) := "01";
constant CONT_READ_CYCLE : std_logic_vector(1 downto 0) := "10";
constant STOP_CYCLE      : std_logic_vector(1 downto 0) := "11";

constant cR_MEM_WR  : std_logic_vector(4 downto 0) := "00001";
constant cR_MEM_RD  : std_logic_vector(4 downto 0) := "00010";
constant cL_MEM_WR  : std_logic_vector(4 downto 0) := "00011";
constant cL_MEM_RD  : std_logic_vector(4 downto 0) := "00100";
constant cLR_MEM_WR : std_logic_vector(4 downto 0) := "00101";
constant cMEM_STOP  : std_logic_vector(4 downto 0) := "00110";
constant cRR_INPRO  : std_logic_vector(4 downto 0) := "00111";
constant cLL_INPRO  : std_logic_vector(4 downto 0) := "01000";
constant cLR_INPRO  : std_logic_vector(4 downto 0) := "01001";
constant cINPRO_F   : std_logic_vector(4 downto 0) := "01010";
constant cINPRO_R   : std_logic_vector(4 downto 0) := "01011";
constant cINPRO_L   : std_logic_vector(4 downto 0) := "01100";
constant cINPRO_LR  : std_logic_vector(4 downto 0) := "01101";
constant cFF_MUL    : std_logic_vector(4 downto 0) := "01110";
constant cFR_MUL    : std_logic_vector(4 downto 0) := "01111";
constant cFL_MUL    : std_logic_vector(4 downto 0) := "10000";
constant cRR_MUL    : std_logic_vector(4 downto 0) := "10001";
constant cLL_MUL    : std_logic_vector(4 downto 0) := "10010";
constant cLR_MUL    : std_logic_vector(4 downto 0) := "10011";
constant cFF_ADD    : std_logic_vector(4 downto 0) := "10100";
constant cFR_ADD    : std_logic_vector(4 downto 0) := "10101";
constant cFL_ADD    : std_logic_vector(4 downto 0) := "10110";
constant cRR_ADD    : std_logic_vector(4 downto 0) := "10111";
constant cLL_ADD    : std_logic_vector(4 downto 0) := "11000";
constant cLR_ADD    : std_logic_vector(4 downto 0) := "11001";
constant cCALC_F    : std_logic_vector(4 downto 0) := "11010";
constant cCALC_R    : std_logic_vector(4 downto 0) := "11011";
constant cCALC_L    : std_logic_vector(4 downto 0) := "11100";
constant cCALC_LR   : std_logic_vector(4 downto 0) := "11101";

begin

RESET <= '1' when CTRL_BUS = "00000" else '0';

DATA_BUS <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_ALU = '0' else
                ADD_Q     when DATA_BUS_SEL = dADD or DATA_BUS_SEL = dINPRO_F else
                MUL_Q     when DATA_BUS_SEL = dMUL else
                R_RD_DATA when DATA_BUS_SEL = dR_MEM_RD else
                L_RD_DATA when DATA_BUS_SEL = dL_MEM_RD else
                ( others => '0' );

process ( RESET, CLK ) begin
if RESET = '1' then
```

```
        DATA_BUS_SEL <= dSTOP;
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                when cMEM_STOP | cCALC_F | cCALC_R | cCALC_L | cCALC_LR =>
                        null;
                when cR_MEM_RD =>
                        DATA_BUS_SEL <= dR_MEM_RD;
                when cL_MEM_RD =>
                        DATA_BUS_SEL <= dL_MEM_RD;
                when cFF_MUL | cFR_MUL | cFL_MUL | cRR_MUL | cLL_MUL | cLR_MUL =>
                        DATA_BUS_SEL <= dMUL;
                when cFF_ADD | cFR_ADD | cFL_ADD | cRR_ADD | cLL_ADD | cLR_ADD =>
                        DATA_BUS_SEL <= dADD;
                when cINPRO_F =>
                        DATA_BUS_SEL <= dINPRO_F;
                when others =>
                        DATA_BUS_SEL <= dSTOP;
        end case;
end if;
end process;

process ( RESET, CLK ) begin
if RESET = '1' then
        LATCH_SEL <= lSTOP;
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                when cRR_INPRO | cRR_MUL | cRR_ADD =>
                        LATCH_SEL <= RR_MEM;
                when cLL_INPRO | cLL_MUL | cLL_ADD =>
                        LATCH_SEL <= LL_MEM;
                when cLR_INPRO | cLR_MUL | cLR_ADD =>
                        LATCH_SEL <= LR_MEM;
                when cFR_MUL | cFR_ADD =>
                        LATCH_SEL <= FR_MEM;
                when cFL_MUL | cFL_ADD =>
                        LATCH_SEL <= FL_MEM;
                when cFF_MUL | cFF_ADD =>
                        LATCH_SEL <= FF_MEM;
                when cINPRO_F | cINPRO_R | cINPRO_L | cINPRO_LR | cCALC_F |
cCALC_R | cCALC_L | cCALC_LR =>
                        null;
                when others =>
                        LATCH_SEL <= lSTOP;
        end case;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        MUL_A <= ( others => '0' );
        MUL_B <= ( others => '0' );
elsif rising_edge( CLK ) then
        if DCNT1 = '1' or CALC_CNT = '1' then
                case CTRL_BUS is
                        when cRR_INPRO =>
                                MUL_A <= R_DATA;
                                MUL_B <= R_DATA;
                        when cLL_INPRO =>
                                MUL_A <= L_DATA;
                                MUL_B <= L_DATA;
                        when cLR_INPRO =>
                                MUL_A <= R_DATA;
                                MUL_B <= L_DATA;
                        when cFF_MUL =>
                                MUL_A <= DATA_BUF;
                                MUL_B <= DATA_BUS;
                        when cINPRO_F | cINPRO_R | cINPRO_L| cINPRO_LR |
cCALC_F | cCALC_R | cCALC_L | cCALC_LR =>
                                case LATCH_SEL is
                                        when RR_MEM =>
                                                MUL_A <= R_DATA;
                                                MUL_B <= R_DATA;
                                        when LL_MEM =>
                                                MUL_A <= L_DATA;
                                                MUL_B <= L_DATA;
```

```
                                        when LR_MEM =>
                                                MUL_A <= R_DATA;
                                                MUL_B <= L_DATA;
                                        when FR_MEM =>
                                                MUL_A <= DATA_BUF;
                                                MUL_B <= R_DATA;
                                        when FL_MEM =>
                                                MUL_A <= DATA_BUF;
                                                MUL_B <= L_DATA;
                                        when others =>
                                                null;
                                end case;
                        when others =>
                                MUL_A <= ( others => '0' );
                                MUL_B <= ( others => '0' );
                end case;
        else
                MUL_A <= ( others => '0' );
                MUL_B <= ( others => '0' );
        end if;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        ADD_A_BUF <= ( others => '0' );
        ADD_B_BUF <= ( others => '0' );
elsif rising_edge( CLK ) then
        if DCNT1 = '1' or CALC_CNT = '1' then
                case CTRL_BUS is
                        when cFF_ADD =>
                                ADD_A_BUF <= DATA_BUF;
                                ADD_B_BUF <= DATA_BUS;
                        when cCALC_F | cCALC_R | cCALC_L | cCALC_LR =>
                                case LATCH_SEL is
                                        when RR_MEM =>
                                                ADD_A_BUF <= R_DATA;
                                                ADD_B_BUF <= R_DATA;
                                        when LL_MEM =>
                                                ADD_A_BUF <= L_DATA;
                                                ADD_B_BUF <= L_DATA;
                                        when LR_MEM =>
                                                ADD_A_BUF <= R_DATA;
                                                ADD_B_BUF <= L_DATA;
                                        when FR_MEM =>
                                                ADD_A_BUF <= DATA_BUF;
                                                ADD_B_BUF <= R_DATA;
                                        when FL_MEM =>
                                                ADD_A_BUF <= DATA_BUF;
                                                ADD_B_BUF <= L_DATA;
                                        when others =>
                                                null;
                                end case;
                        when others =>
                                ADD_A_BUF <= ( others => '0' );
                                ADD_B_BUF <= ( others => '0' );
                end case;
        else
                ADD_A_BUF <= ( others => '0' );
                ADD_B_BUF <= ( others => '0' );
        end if;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        DATA_BUF <= ( others => '0' );
        CALC_CNT <= '0';
        DCNT <= '0';
elsif rising_edge( CLK ) then
        if ( CTRL_BUS = cFF_MUL or CTRL_BUS = cFR_MUL or CTRL_BUS = cFL_MUL or
CTRL_BUS = cFF_ADD or CTRL_BUS = cFR_ADD or CTRL_BUS = cFL_ADD ) and
CALC_CNT = '0' then
                DATA_BUF <= DATA_BUS;
```

```
        end if;
        if ( ( CTRL_BUS = cFF_MUL or CTRL_BUS = cFF_ADD ) and CALC_CNT = '0' ) or
DCNT = '1' then
                CALC_CNT <= '1';
        else
                CALC_CNT <= '0';
        end if;
        if CTRL_BUS = cFR_MUL or CTRL_BUS = cFL_MUL or CTRL_BUS = cRR_MUL or
CTRL_BUS = cLL_MUL or CTRL_BUS = cLR_MUL or CTRL_BUS = cFR_ADD or
CTRL_BUS = cFL_ADD or CTRL_BUS = cRR_ADD or CTRL_BUS = cLL_ADD or
CTRL_BUS = cLR_ADD then
                DCNT <= '1';
        else
                DCNT <= '0';
        end if;
end if;
end process;

process ( RESET, CLK ) begin
if RESET = '1' then
        DCNT1 <= '0';
        CALC1_ACK <= '0';
elsif rising_edge( CLK ) then
        if CTRL_BUS = cRR_INPRO or CTRL_BUS = cLL_INPRO or CTRL_BUS = cLR_INPRO then
                CALC1_ACK <= '1';
        else
                CALC1_ACK <= '0';
        end if;
        if CALC1_ACK = '1' then
                DCNT1 <= '1';
        else
                DCNT1 <= '0';
        end if;
end if;
end process;

process ( RESET, CLK ) begin
if RESET = '1' then
        DCNT2 <= ( others => '0' );
elsif rising_edge( CLK ) then
        if CTRL_BUS = cINPRO_F or CTRL_BUS = cINPRO_R or CTRL_BUS = cINPRO_L or
CTRL_BUS = cINPRO_LR then
                if DCNT2 < "1001" then
                        DCNT2 <= DCNT2 + '1';
                end if;
        else
                DCNT2 <= ( others => '0' );
        end if;
end if;
end process;

process ( RESET, CLK ) begin
if RESET = '1' then
        DCNT3 <= '0';
        CALC3_ACK <= '0';
elsif rising_edge( CLK ) then
        if ( CTRL_BUS = cFF_MUL or CTRL_BUS = cFF_ADD ) and CALC3_ACK = '0' then
                CALC3_ACK <= '1';
        end if;
        if ( CTRL_BUS = cCALC_F or CTRL_BUS = cCALC_R or CTRL_BUS = cCALC_L or
CTRL_BUS = cCALC_LR ) and CALC3_ACK = '1' then
                DCNT3 <= '1';
                CALC3_ACK <= '0';
        else
                DCNT3 <= '0';
        end if;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        DCNT4 <= ( others => '0' );
        CALC4_ACK <= '0';
elsif rising_edge( CLK ) then
        if CTRL_BUS = cFR_MUL or CTRL_BUS = cFL_MUL or CTRL_BUS = cRR_MUL or
```

```
CTRL_BUS = cLL_MUL or CTRL_BUS = cLR_MUL or CTRL_BUS = cFR_ADD or
CTRL_BUS = cFL_ADD or CTRL_BUS = cRR_ADD or CTRL_BUS = cLL_ADD or
CTRL_BUS = cLR_ADD then
                CALC4_ACK <= '1';
        end if;
        if ( CTRL_BUS = cCALC_F or CTRL_BUS = cCALC_R or CTRL_BUS = cCALC_L or
CTRL_BUS = cCALC_LR ) and CALC4_ACK = '1' then
                if DCNT4 < "100" then
                        DCNT4 <= DCNT4 + '1';
                else
                        CALC4_ACK <= '0';
                end if;
        else
                DCNT4 <= ( others => '0' );
        end if;
end if;
end process;

process ( RESET, CLK ) begin
if RESET = '1' then
        DCNT5 <= '0';
elsif rising_edge( CLK ) then
        if ( CTRL_BUS = cCALC_R or CTRL_BUS = cCALC_L or CTRL_BUS = cCALC_LR ) and
( DCNT3 = '1' or DCNT4 = "011" ) then
                DCNT5 <= '1';
        else
                DCNT5 <= '0';
        end if;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        CALC_DONE <= '0';
elsif rising_edge( CLK ) then
        if ( CTRL_BUS = cINPRO_F and DCNT2 = "0111" ) or
( CTRL_BUS = cCALC_F and DCNT3 = '1' ) or
( CTRL_BUS = cCALC_F and DCNT4 = "011" ) or
R_RD_CYCLE = '1' or R_WR_CYCLE = '1' or L_RD_CYCLE = '1' or L_WR_CYCLE = '1' then
                CALC_DONE <= '1';
        else
                CALC_DONE <= '0';
        end if;
end if;
end process;


----------------< Adder Feedback >------------
process ( RESET, CLK ) begin
if RESET = '1' then
        FB_Q <= ( others => '0' );
elsif rising_edge( CLK ) then
        if DCNT2 = "0101" then
                FB_Q <= ADD_Q;
        elsif DCNT2 = "0111" then
                FB_Q <= ( others => '0' );
        end if;
end if;
end process;

ADD_A <= FB_Q  when DCNT2 = "0110" or DCNT2 = "0111" else
         MUL_Q when CTRL_BUS = cRR_INPRO or CTRL_BUS = cLL_INPRO or
CTRL_BUS = cLR_INPRO or ( ( CTRL_BUS = cINPRO_F or CTRL_BUS = cINPRO_R or
CTRL_BUS = cINPRO_L or CTRL_BUS = cINPRO_LR ) and DCNT2 < "1000" ) else
         ADD_A_BUF;

ADD_B <= ADD_Q when CTRL_BUS = cRR_INPRO or CTRL_BUS = cLL_INPRO or
CTRL_BUS = cLR_INPRO or ( ( CTRL_BUS = cINPRO_F or CTRL_BUS = cINPRO_R or
CTRL_BUS = cINPRO_L or CTRL_BUS = cINPRO_LR ) and DCNT2 < "1000" ) else
         ADD_B_BUF;

----------<< Memory Controller >>---------
process ( RESET, CLK ) begin
if RESET = '1' then
```

```
        R_MEM_STATE_SEL <= STOP_CYCLE;
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                 when cR_MEM_WR | cLR_MEM_WR =>
                        R_MEM_STATE_SEL <= WRITE_CYCLE;
                when cR_MEM_RD =>
                        R_MEM_STATE_SEL <= READ_CYCLE;
                when cMEM_STOP | cINPRO_F | cCALC_F =>
                        R_MEM_STATE_SEL <= STOP_CYCLE;
                when cRR_INPRO | cLR_INPRO | cFR_MUL | cRR_MUL | cLR_MUL |
cFR_ADD | cRR_ADD | cLR_ADD =>
                        R_MEM_STATE_SEL <= CONT_READ_CYCLE;
                when cINPRO_R | cINPRO_LR =>
                        if DCNT2 = "1000" then
                                R_MEM_STATE_SEL <= WRITE_CYCLE;
                        else
                                R_MEM_STATE_SEL <= STOP_CYCLE;
                        end if;
                when cCALC_R | cCALC_LR =>
                        if DCNT5 = '1' then
                                R_MEM_STATE_SEL <= WRITE_CYCLE;
                        else
                                R_MEM_STATE_SEL <= STOP_CYCLE;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        R_WR_DATA <= ( others => '0' );
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                 when cR_MEM_WR | cLR_MEM_WR =>
                        R_WR_DATA <= DATA_BUS;
                when cINPRO_R | cINPRO_LR =>
                        if DCNT2 = "1000" then
                                R_WR_DATA <= ADD_Q;
                        end if;
                when cCALC_R | cCALC_LR =>
                        if DCNT5 = '1' then
                                if DATA_BUS_SEL = dMUL then
                                        R_WR_DATA <= MUL_Q;
                                elsif DATA_BUS_SEL = dADD then
                                        R_WR_DATA <= ADD_Q;
                                end if;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        R_ADRS_BUF <= ( others => '0' );
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                 when cR_MEM_WR | cR_MEM_RD | cLR_MEM_WR | cRR_INPRO |
cLR_INPRO | cFR_MUL | cRR_MUL | cLR_MUL | cFR_ADD | cRR_ADD | cLR_ADD =>
                        R_ADRS_BUF <= ADRS_BUS;
                when cINPRO_R | cINPRO_LR =>
                        if DCNT2 = "1000" then
                                R_ADRS_BUF <= ADRS_BUS;
                        end if;
                when cCALC_R | cCALC_LR =>
                        if DCNT5 = '1' then
                                R_ADRS_BUF <= ADRS_BUS;
                        end if;
                when others =>
                        null;
        end case;
```

```
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        L_MEM_STATE_SEL <= STOP_CYCLE;
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                 when cL_MEM_WR | cLR_MEM_WR =>
                        L_MEM_STATE_SEL <= WRITE_CYCLE;
                when cL_MEM_RD =>
                        L_MEM_STATE_SEL <= READ_CYCLE;
                when cMEM_STOP | cINPRO_F | cCALC_F =>
                        L_MEM_STATE_SEL <= STOP_CYCLE;
                when cLL_INPRO | cLR_INPRO | cFL_MUL | cLL_MUL | cLR_MUL |
cFL_ADD | cLL_ADD | cLR_ADD =>
                        L_MEM_STATE_SEL <= CONT_READ_CYCLE;
                when cINPRO_L | cINPRO_LR =>
                        if DCNT2 = "1000" then
                                L_MEM_STATE_SEL <= WRITE_CYCLE;
                        else
                                L_MEM_STATE_SEL <= STOP_CYCLE;
                        end if;
                when cCALC_L | cCALC_LR =>
                        if DCNT5 = '1' then
                                L_MEM_STATE_SEL <= WRITE_CYCLE;
                        else
                                L_MEM_STATE_SEL <= STOP_CYCLE;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;



process ( RESET, CLK ) begin
if RESET = '1' then
        L_WR_DATA <= ( others => '0' );
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                 when cL_MEM_WR | cLR_MEM_WR =>
                        L_WR_DATA <= DATA_BUS;
                when cINPRO_L | cINPRO_LR =>
                        if DCNT2 = "1000" then
                                L_WR_DATA <= ADD_Q;
                        end if;
                when cCALC_L | cCALC_LR =>
                        if DCNT5 = '1' then
                                if DATA_BUS_SEL = dMUL then
                                        L_WR_DATA <= MUL_Q;
                                elsif DATA_BUS_SEL = dADD then
                                        L_WR_DATA <= ADD_Q;
                                end if;
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;


process ( RESET, CLK ) begin
if RESET = '1' then
        L_ADRS_BUF <= ( others => '0' );
elsif rising_edge( CLK ) then
        case CTRL_BUS is
                 when cL_MEM_WR | cLR_MEM_WR | cL_MEM_RD | cLL_INPRO |
cFL_MUL | cLL_MUL | cFL_ADD | cLL_ADD =>
                        L_ADRS_BUF <= ADRS_BUS;
                when cLR_INPRO | cLR_MUL | cLR_ADD =>
                        L_ADRS_BUF <= DATA_BUS(16 downto 0);
                when cINPRO_L =>
                        if DCNT2 = "1000" then
```

```
                                L_ADRS_BUF <= ADRS_BUS;
                        end if;
                when cINPRO_LR =>
                        if DCNT2 = "1000" then
                                L_ADRS_BUF <= DATA_BUS(16 downto 0);
                        end if;
                when cCALC_L =>
                        if DCNT5 = '1' then
                                L_ADRS_BUF <= ADRS_BUS;
                        end if;
                when cCALC_LR =>
                        if DCNT5 = '1' then
                                L_ADRS_BUF <= DATA_BUS(16 downto 0);
                        end if;
                when others =>
                        null;
        end case;
end if;
end process;


------------< Memory Controller >--------------
R_MEM : memctrl port map (
                        CLK => CLK, RESET => RESET,
                        ADRS => R_ADRS, ADRS_BUF => R_ADRS_BUF,
                        DATA => R_DATA,
                        WR_DATA => R_WR_DATA, RD_DATA => R_RD_DATA,
                        SCS => R_SCS, SOE => R_SOE, SWE => R_SWE,
                        MEM_STATE_SEL => R_MEM_STATE_SEL,
                        WR_CYCLE => R_WR_CYCLE, RD_CYCLE => R_RD_CYCLE
                        );

L_MEM : memctrl port map (
                        CLK => CLK, RESET => RESET,
                        ADRS => L_ADRS, ADRS_BUF => L_ADRS_BUF,
                        DATA => L_DATA,
                        WR_DATA => L_WR_DATA, RD_DATA => L_RD_DATA,
                        SCS => L_SCS, SOE => L_SOE, SWE => L_SWE,
                        MEM_STATE_SEL => L_MEM_STATE_SEL,
                        WR_CYCLE => L_WR_CYCLE, RD_CYCLE => L_RD_CYCLE
                        );

------------< Floating Point Number Multiplier and Adder>-------------
multiplier : fpmult port map ( CLK => CLK, FA => MUL_A, FB => MUL_B, Q => MUL_Q );
adder      : fpadd  port map ( CLK => CLK, FA => ADD_A, FB => ADD_B, Q => ADD_Q );

end RTL;
```

I.3.2 ハウスホルダー変換

```
------------------------------------------------------------
-- Householder Transform  (Lower FLEX10k)
-- < hshld10.vhd >
-- 1999/03/15 (Mon)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.MATH.all;

library metamor;
use metamor.attributes.all;

entity hshld10 is
  port (CLK  : in std_logic;
                A    : inout std_logic_vector(15 downto 0);
                BL   : in std_logic_vector(7 downto 0);
                BH   : out std_logic_vector(5 downto 0);
                B_CONF  : in std_logic_vector(1 downto 0);
```

```
                CL   : in std_logic_vector(5 downto 0);

                DATA_BUS  : inout std_logic_vector(31 downto 0);
                ADRS_BUS  : out std_logic_vector(16 downto 0);
                CTRL_BUS  : out std_logic_vector(4 downto 0);
                CALC_DONE : in  std_logic;
                OE_ALU    : out std_logic;

                OBF  : in std_logic_vector(1 downto 0);
                ACK  : out std_logic_vector(1 downto 0);
                STB  : out std_logic_vector(1 downto 0);
                IBF  : in std_logic_vector(1 downto 0)
                );

attribute pinnum of CLK : signal is  "D22";
attribute pinnum of A   : signal is "BC23,BB24,BC25,BB26,BC27,BB28,BC29,BB30,
BC31,BB32,BC33,BB34,BC35,BB36,BC37,BB38";
attribute pinnum of BL  : signal is  "BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20";
attribute pinnum of BH  : signal is  "BC7,BB8,BC9,BB10,BC11,BB12";
attribute pinnum of B_CONF  : signal is "BC5,BB6";
attribute pinnum of CL   : signal is  "AU23,AV24,AU25,AU33,AV34,AU35";

attribute pinnum of DATA_BUS  : signal is "A5,B6,A7,B8,A9,B10,A11,B12,A13,B14,
A15,B16,A17,B18,A19,B20,A23,B24,A25,B26,A27,B28,A29,B30,A31,B32,A33,B34,A35,
B36,A37,B38";
attribute pinnum of ADRS_BUS  : signal is "F16,G19,F20,G21,F22,G23,F24,G25,
F26,G29,F30,G31,F32,G33,F34,G35,F36";
attribute pinnum of CTRL_BUS  : signal is  "G11,F12,G13,F14,G15";
attribute pinnum of CALC_DONE : signal is  "F10";
attribute pinnum of OE_ALU    : signal is  "G9";

attribute pinnum of OBF : signal is  "AV18,AV28";
attribute pinnum of ACK : signal is  "AU19,AU29";
attribute pinnum of STB : signal is  "AU21,AU31";
attribute pinnum of IBF : signal is  "AV20,AV30";

end hshld10;
architecture RTL of hshld10 is

-------------< 32-bit Floating point Number Divider >-------
component fpdiv is
 port (CLK : in std_logic;
                FA  : in std_logic_vector(31 downto 0);
                FB  : in std_logic_vector(31 downto 0);
                Q   : out std_logic_vector(31 downto 0)
    );
end component;

signal DIV_A : std_logic_vector(31 downto 0);
signal DIV_B : std_logic_vector(31 downto 0);
signal DIV_Q : std_logic_vector(31 downto 0);
signal DIV_ACK   : std_logic;
signal DIV_DONE  : std_logic;
signal DCNT : std_logic;
signal DCNT2 : std_logic_vector(3 downto 0);
signal DCNT3 : std_logic_vector(3 downto 0);

signal A_REG : std_logic_vector(15 downto 0);
signal ACK_BUF : std_logic_vector(1 downto 0);
signal STB_BUF : std_logic_vector(1 downto 0);
signal IN_CNT : std_logic;
signal OUT_CNT : std_logic;
signal OUT_ACK : std_logic;
signal OUT_ACK2 : std_logic;

signal N : std_logic_vector(7 downto 0);
signal ROW : std_logic_vector(7 downto 0);

signal WA : std_logic;
signal MA_WR : std_logic;
signal MA : std_logic_vector(31 downto 0);
```

```
--------------------------------

type CALC_STATE_TYPE is (
                FIRST_DATA, SECOND_DATA
                );

signal CALC_STATE : CALC_STATE_TYPE;

type ALU_STATE_TYPE is (
                R_MEM_WR, R_MEM_RD,
                L_MEM_WR, L_MEM_RD,
                LR_MEM_WR,
                MEM_STOP,
                RR_INPRO, LL_INPRO, LR_INPRO,
                INPRO_F, INPRO_R, INPRO_L, INPRO_LR,
                FF_MUL, FR_MUL, FL_MUL, RR_MUL, LL_MUL, LR_MUL,
                FF_ADD, FR_ADD, FL_ADD, RR_ADD, LL_ADD, LR_ADD,
                CALC_F, CALC_R, CALC_L, CALC_LR,
                aSTOP
                );

signal ALU_STATE : ALU_STATE_TYPE;
signal DATA_BUS_BUF : std_logic_vector(31 downto 0);
signal DATA_BUF1 : std_logic_vector(31 downto 0);
signal DATA_BUF2 : std_logic_vector(31 downto 0);
signal OE_BUF : std_logic;
signal ADRS_BUF1 : std_logic_vector(16 downto 0);
signal ADRS_BUF2 : std_logic_vector(16 downto 0);
signal ALU_ACK : std_logic;

--------------------------------

type HS_STATE_TYPE is (
                HsDimWrite,
                HsVarIni, HsS2Calc,
                HsSCalc, HsAkRead, HsSNorm,
                HsAkxSCalc, HsS2pAkxSCalc, HsCCalc, HsCWrite, HsViceWrite,
                HsAkpSCalc, HsAxUCalc, HsPCalc,
                HsPxUCalc, HsAlphaCalc, HsAlphaxCCalc, HsAlphaxCd2Calc,
                HsAlphaxCd2xUCalc, HsQCalc,
                HsUxQtCalc, HsQxUtCalc, HsUxQtpQxUtCalc, HsACalc,
                HsResRead,
                HsStop
                );

signal HS_STATE : HS_STATE_TYPE;
signal S2, S, Ak, AkxS, S2pAkxS, C, AxU, Alpha, AlphaxC, AlphaxCd2, AlphaxCd2xU,
 UxQt, QxUt, UxQtpQxUt    : std_logic_vector(31 downto 0);

signal RES    : std_logic_vector(31 downto 0);

constant FP_ONE : std_logic_vector(31 downto 0)
:= "00111111100000000000000000000000";
constant FP_TWO : std_logic_vector(31 downto 0)
:= "01000000000000000000000000000000";

constant cR_MEM_WR  : std_logic_vector(4 downto 0) := "00001";
constant cR_MEM_RD  : std_logic_vector(4 downto 0) := "00010";
constant cL_MEM_WR  : std_logic_vector(4 downto 0) := "00011";
constant cL_MEM_RD  : std_logic_vector(4 downto 0) := "00100";
constant cLR_MEM_WR : std_logic_vector(4 downto 0) := "00101";
constant cMEM_STOP  : std_logic_vector(4 downto 0) := "00110";
constant cRR_INPRO  : std_logic_vector(4 downto 0) := "00111";
constant cLL_INPRO  : std_logic_vector(4 downto 0) := "01000";
constant cLR_INPRO  : std_logic_vector(4 downto 0) := "01001";
constant cINPRO_F   : std_logic_vector(4 downto 0) := "01010";
constant cINPRO_R   : std_logic_vector(4 downto 0) := "01011";
constant cINPRO_L   : std_logic_vector(4 downto 0) := "01100";
constant cINPRO_LR  : std_logic_vector(4 downto 0) := "01101";
constant cFF_MUL    : std_logic_vector(4 downto 0) := "01110";
constant cFR_MUL    : std_logic_vector(4 downto 0) := "01111";
constant cFL_MUL    : std_logic_vector(4 downto 0) := "10000";
```

```
constant cRR_MUL    : std_logic_vector(4 downto 0) := "10001";
constant cLL_MUL    : std_logic_vector(4 downto 0) := "10010";
constant cLR_MUL    : std_logic_vector(4 downto 0) := "10011";
constant cFF_ADD    : std_logic_vector(4 downto 0) := "10100";
constant cFR_ADD    : std_logic_vector(4 downto 0) := "10101";
constant cFL_ADD    : std_logic_vector(4 downto 0) := "10110";
constant cRR_ADD    : std_logic_vector(4 downto 0) := "10111";
constant cLL_ADD    : std_logic_vector(4 downto 0) := "11000";
constant cLR_ADD    : std_logic_vector(4 downto 0) := "11001";
constant cCALC_F    : std_logic_vector(4 downto 0) := "11010";
constant cCALC_R    : std_logic_vector(4 downto 0) := "11011";
constant cCALC_L    : std_logic_vector(4 downto 0) := "11100";
constant cCALC_LR   : std_logic_vector(4 downto 0) := "11101";

begin

A <= "ZZZZZZZZZZZZZZZZ" when ACK_BUF = "00" else A_REG;

ACK_BUF <= OBF;
ACK <= ACK_BUF;

----------<< Input Matrix Data from PC >>-------------
process ( BL(0), OBF ) begin
if BL(0) = '1' then
        IN_CNT <= '0';
        MA <= ( others => '0' );
elsif OBF'event and OBF = "11" then
        if WA = '0' then
                if IN_CNT = '0' then
                        MA(15 downto 0) <= A;
                        IN_CNT <= '1';
                else
                        MA(31 downto 16) <= A;
                        IN_CNT <= '0';
                end if;
        end if;
end if;
end process;

---------<< Matrix Row Counter >>---------
process ( BL(0), BL(1), IN_CNT ) begin
if BL(0) = '1' or BL(1) = '1'  then
        ROW <= ( others => '0' );
elsif rising_edge( IN_CNT ) then
        ROW <= ROW + '1';
end if;
end process;

---------<< Matrix Dimension Counter >>---------
process ( BL(0), BL(1) ) begin
if BL(0) = '1' then
        N <= ( 0 => '1', others => '0' );
elsif rising_edge( BL(1) ) then
        if WA = '0' then
                N <= N + '1';
        end if;
end if;
end process;

---------<< Detect End of Matrix Data >>---------
process ( BL(0), BL(2) ) begin
if BL(0) = '1' then
        WA <= '0';
elsif rising_edge( BL(2) ) then
        WA <= '1';
end if;
end process;

-----<< Generate Memory Write Signal of Matrix >>-----
process ( BL(0), ALU_ACK, WA, IN_CNT ) begin
if BL(0) = '1' or ALU_ACK = '1' or WA = '1' then
        MA_WR <= '0';
```

```
elsif falling_edge( IN_CNT ) then
        MA_WR <= '1';
end if;
end process;

-----------<< Output Result Data to PC >>------------
process ( BL(0), BL(3), IBF ) begin
if BL(0) = '1' then
        STB <= "11";
        OUT_CNT <= '0';
        A_REG <= ( others => '0' );
elsif rising_edge( BL(3) ) then
        STB <= "00";
        if OUT_CNT = '0' then
                A_REG <= RES(15 downto 0);
                OUT_CNT <= '1';
        else
                A_REG <= RES(31 downto 16);
                OUT_CNT <= '0';
        end if;
end if;
if IBF = "11" then
        STB <= "11";
end if;
end process;

------------<< Generate Memory Read Signal of Result Data >>------------
process ( BL(0), BL(4), OUT_ACK2 ) begin
if BL(0) = '1' or OUT_ACK2 = '1' then
        OUT_ACK <= '0';
elsif rising_edge( BL(4) ) then
        OUT_ACK <= '1';
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        OUT_ACK2 <= '0';
elsif rising_edge( CLK ) then
        if OUT_ACK = '1' then
                OUT_ACK2 <= '1';
        else
                OUT_ACK2 <= '0';
        end if;
end if;
end process;

-----------<< Selecter of ALU Operation >>-------
DATA_BUS <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_BUF = '1' else DATA_BUS_BUF;
OE_ALU <= OE_BUF;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CTRL_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR =>
                        CTRL_BUS <= cR_MEM_WR;
                when R_MEM_RD =>
                        CTRL_BUS <= cR_MEM_RD;
                when L_MEM_WR =>
                        CTRL_BUS <= cL_MEM_WR;
                when L_MEM_RD =>
                        CTRL_BUS <= cL_MEM_RD;
                when LR_MEM_WR =>
                        CTRL_BUS <= cLR_MEM_WR;
                when MEM_STOP =>
                        CTRL_BUS <= cMEM_STOP;
                when RR_INPRO =>
                        CTRL_BUS <= cRR_INPRO;
                when LL_INPRO =>
                        CTRL_BUS <= cLL_INPRO;
                when LR_INPRO =>
                        CTRL_BUS <= cLR_INPRO;
                when INPRO_F =>
```

```
                        CTRL_BUS <= cINPRO_F;
                when INPRO_R =>
                        CTRL_BUS <= cINPRO_R;
                when INPRO_L =>
                        CTRL_BUS <= cINPRO_L;
                when INPRO_LR =>
                        CTRL_BUS <= cINPRO_LR;
                when FF_MUL =>
                        CTRL_BUS <= cFF_MUL;
                when FR_MUL =>
                        CTRL_BUS <= cFR_MUL;
                when FL_MUL =>
                        CTRL_BUS <= cFL_MUL;
                when RR_MUL =>
                        CTRL_BUS <= cRR_MUL;
                when LL_MUL =>
                        CTRL_BUS <= cLL_MUL;
                when LR_MUL =>
                        CTRL_BUS <= cLR_MUL;
                when FF_ADD =>
                        CTRL_BUS <= cFF_ADD;
                when FR_ADD =>
                        CTRL_BUS <= cFR_ADD;
                when FL_ADD =>
                        CTRL_BUS <= cFL_ADD;
                when RR_ADD =>
                        CTRL_BUS <= cRR_ADD;
                when LL_ADD =>
                        CTRL_BUS <= cLL_ADD;
                when LR_ADD =>
                        CTRL_BUS <= cLR_ADD;
                when CALC_F =>
                        CTRL_BUS <= cCALC_F;
                when CALC_R =>
                        CTRL_BUS <= cCALC_R;
                when CALC_L =>
                        CTRL_BUS <= cCALC_L;
                when CALC_LR =>
                        CTRL_BUS <= cCALC_LR;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DATA_BUS_BUF <= ( others => '0' );
        OE_BUF <= '0';
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR | L_MEM_WR | LR_MEM_WR | FR_MUL | FL_MUL |
FR_ADD | FL_ADD =>
                           DATA_BUS_BUF <= DATA_BUF1;
                        OE_BUF <= '0';
                when MEM_STOP | INPRO_F | CALC_F =>
                        DATA_BUS_BUF <= ( others => '0' );
                        OE_BUF <= '1';
                when LR_INPRO | INPRO_LR | LR_MUL | LR_ADD | CALC_LR =>
                        DATA_BUS_BUF(16 downto 0) <= ADRS_BUF2;
                        OE_BUF <= '0';
                when FF_MUL | FF_ADD =>
                        OE_BUF <= '0';
                        if CALC_STATE = FIRST_DATA then
                                DATA_BUS_BUF <= DATA_BUF1;
                        else
                                DATA_BUS_BUF <= DATA_BUF2;
                        end if;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CALC_STATE <= FIRST_DATA;
elsif rising_edge( CLK ) then
```

```
        case ALU_STATE is
                when FF_MUL | FF_ADD =>
                         CALC_STATE <= SECOND_DATA;
                when others =>
                        if CALC_DONE = '1' then
                                CALC_STATE <= FIRST_DATA;
                        end if;
        end case;
end if;
end process;


process ( BL(0), CLK ) begin
if BL(0) = '1' then
        ADRS_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when MEM_STOP | INPRO_F | FF_MUL | FF_ADD | CALC_F =>
                        ADRS_BUS <= ( others => '0' );
                when others =>
                        ADRS_BUS <= ADRS_BUF1;
        end case;
end if;
end process;

------------<< Householder Transform >>-------------
process( BL(0), CLK )
        variable i, j, k : std_logic_vector(7 downto 0);
begin
if BL(0) = '1' then
        HS_STATE <= hsStop;
        ALU_STATE <= aStop;
        ALU_ACK <= '0';
        DIV_ACK <= '0';
        DATA_BUF1 <= (others => '0');
        DATA_BUF2 <= (others => '0');
        ADRS_BUF1 <= (others => '0');
        ADRS_BUF2 <= (others => '0');
--        DIV_A <= (others => '0');
--        DIV_B <= (others => '0');
--        S2 <= (others => '0');
--        S <= (others => '0');
--        Ak <= (others => '0');
--        AkxS <= (others => '0');
--        S2pAkxS <= (others => '0');
--        HsC <= (others => '0');
--        AxU <= (others => '0');
--        AL <= (others => '0');
--        ALxC <= (others => '0');
--        ALxCd2 <= (others => '0');
--        ALxCd2xU <= (others => '0');
--        UxQT <= (others => '0');
--        QxUt <= (others => '0');
--        UxQtpQxUt <= (others => '0');
        i := ( others => '0' );
        j := ( others => '0' );
        k := ( 0 => '1', others => '0' );
        BH <= ( others => '0' );
--        RES <= ( others => '0' );
--        OUT_ACK <= '0';
--        DCNT2 <= "0000";
elsif rising_edge( CLK ) then
        case HS_STATE is
                when HsStop =>
                        if WA = '1' then
                                HS_STATE <= HsDimWrite;
                        elsif MA_WR = '1' then
                                ALU_ACK <= '1';
                                DATA_BUF1 <= MA;
                                ADRS_BUF1 <= '0' & N & ROW;
                                ALU_STATE <= LR_MEM_WR;
                        elsif ALU_ACK = '1' then
                                ALU_ACK <= '0';
                                ALU_STATE <= MEM_STOP;
                        end if;
```

```
when HsDimWrite =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= "00000000000000000000000000" & N;
                ADRS_BUF1 <= "10000000000000000";
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                HS_STATE <= HsVarIni;
        end if;

when HsVarIni =>
        i := k;
        j := k + '1';
        HS_STATE <= HsS2Calc;

when HsS2Calc =>
        if i < N then
                ALU_STATE <= RR_INPRO;
                i := i + '1';
                ADRS_BUF1 <= '0' & i & k;
        else
                ALU_STATE <= INPRO_F;
        end if;

        if CALC_DONE = '1' then
                S2 <= DATA_BUS;
                i := k;
                HS_STATE <= HsSCalc;
        end if;

when HsSCalc =>
        S <= sqrt(S2);
        HS_STATE <= HsAkRead;

        --if DCNT2 = "1111" then
        --        DCNT2 <= "0000";
        --        HS_STATE <= HsAkRead;
        --else
--                DCNT2 <= DCNT2 + '1';
--        end if;

when HsAkRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '0' & ( k + '1' ) & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Ak <= DATA_BUS;
                ALU_ACK <= '0';
                HS_STATE <= HsSNorm;
        end if;

when HsSNorm =>
        S(31) <= Ak(31);
        HS_STATE <= HsAkxSCalc;

when HsAkxSCalc =>
        ALU_STATE <= FF_MUL;
        DATA_BUF1 <= Ak;
        DATA_BUF2 <= S;

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
```

```
                AkxS <= DATA_BUS;
                HS_STATE <= HsS2pAkxSCalc;
        end if;

when HsS2pAkxSCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= S2;
        DATA_BUF2 <= AkxS;

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                S2pAkxS <= DATA_BUS;
                HS_STATE <= HsCCalc;
        end if;

when HsCCalc =>
        if DIV_ACK = '0' then
                DIV_A <= FP_ONE;
                DIV_B <= S2pAkxS;
                DIV_ACK <= '1';
        --else
        --        DIV_A <= ( others => '0' );
        --        DIV_B <= ( others => '0' );
        end if;

        if DIV_DONE = '1' then
                C <= DIV_Q;
                DIV_ACK <= '0';
                HS_STATE <= HsCWrite;
        end if;

when HsCWrite =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= C;
                ADRS_BUF1 <= "100000000" & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                HS_STATE <= HsViceWrite;
        end if;

when HsViceWrite =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_WR;
                DATA_BUF1 <= ( not S(31) ) & S(30 downto 0);
                ADRS_BUF1 <= '0' & k & ( k + '1' );
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                HS_STATE <= HsAkpSCalc;
        end if;

when HsAkpSCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= Ak;
        DATA_BUF2 <= S;

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_L;
                ADRS_BUF1 <= '0' & ( k + '1' ) & k;
        end if;

        if CALC_DONE = '1' then
                HS_STATE <= HsAxUCalc;
```

```
        end if;

when HsAxUCalc =>
        if i < N then
                ALU_STATE <= LR_INPRO;
                i := i + '1';
                ADRS_BUF1 <= '0' & j & i;
                ADRS_BUF2 <= '0' & i & k;
        else
                ALU_STATE <= INPRO_F;
        end if;

        if CALC_DONE = '1' then
                AxU <= DATA_BUS;
                i := k;
                HS_STATE <= HsPCalc;
        end if;

when HsPCalc =>
        ALU_STATE <= FF_MUL;
        DATA_BUF1 <= AxU;
        DATA_BUF2 <= C;

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_R;
                ADRS_BUF1 <= '0' & j & k;
        end if;

        if CALC_DONE = '1' then
                if j < N then
                        j := j + '1';
                        HS_STATE <= HsAxUCalc;
                else
                        j := k + '1';
                        HS_STATE <= HsAlphaCalc;
                end if;
        end if;

when HsAlphaCalc =>
        if i < N then
                ALU_STATE <= LR_INPRO;
                i := i + '1';
                ADRS_BUF1 <= '0' & i & k;
                ADRS_BUF2 <= '0' & i & k;
        else
                ALU_STATE <= INPRO_F;
        end if;

        if CALC_DONE = '1' then
                Alpha <= DATA_BUS;
                i := k + '1';
                HS_STATE <= HsAlphaxCCalc;
        end if;

when HsAlphaxCCalc =>
        ALU_STATE <= FF_MUL;
        DATA_BUF1 <= Alpha;
        DATA_BUF2 <= C;

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                AlphaxC <= DATA_BUS;
                HS_STATE <= HsAlphaxCd2Calc;
        end if;

when HsAlphaxCd2Calc =>
        if DIV_ACK = '0' then
                DIV_A <= AlphaxC;
                DIV_B <= FP_TWO;
                DIV_ACK <= '1';
        --else
        --        DIV_A <= ( others => '0' );
        --        DIV_B <= ( others => '0' );
```

```
                        end if;

                        if DIV_DONE = '1' then
                                AlphaxCd2 <= DIV_Q;
                                DIV_ACK <= '0';
                                HS_STATE <= HsAlphaxCd2xUCalc;
                        end if;

                when HsAlphaxCd2xUCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FL_MUL;
                                DATA_BUF1 <= AlphaxCd2;
                                ADRS_BUF1 <= '0' & i & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_F;
                        end if;

                        if CALC_DONE = '1' then
                                AlphaxCd2xU <= DATA_BUS;
                                ALU_ACK <= '0';
                                HS_STATE <= HsQCalc;
                        end if;

                when HsQCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FR_ADD;
                                DATA_BUF1 <=
( not AlphaxCd2xU(31) ) & AlphaxCd2xU(30 downto 0);
                                ADRS_BUF1 <= '0' & i & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_R;
                                ADRS_BUF1 <= '0' & i & k;
                        end if;

                        if CALC_DONE = '1' then
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                        HS_STATE <= HsAlphaxCd2xUCalc;
                                else
                                        i := k + '1';
                                        HS_STATE <= HsUxQtCalc;
                                end if;
                        end if;

                when HsUxQtCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= LR_MUL;
                                ADRS_BUF1 <= '0' & i & k;
                                ADRS_BUF2 <= '0' & j & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_F;
                        end if;

                        if CALC_DONE = '1' then
                                UxQt <= DATA_BUS;
                                ALU_ACK <= '0';
                                HS_STATE <= HsQxUtCalc;
                        end if;

                when HsQxUtCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= LR_MUL;
                                ADRS_BUF1 <= '0' & j & k;
                                ADRS_BUF2 <= '0' & i & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_F;
                        end if;

                        if CALC_DONE = '1' then
                                QxUt <= DATA_BUS;
                                ALU_ACK <= '0';
```

```
                        HS_STATE <= HsUxQtpQxUtCalc;
                end if;

        when HsUxQtpQxUtCalc =>
                ALU_STATE <= FF_ADD;
                DATA_BUF1 <= UxQt;
                DATA_BUF2 <= QxUt;

                if CALC_STATE = SECOND_DATA then
                        ALU_STATE <= CALC_F;
                end if;

                if CALC_DONE = '1' then
                        UxQtpQxUt <= DATA_BUS;
                        HS_STATE <= HsACalc;
                end if;

        when HsACalc =>
                if ALU_ACK = '0' then
                        ALU_STATE <= FR_ADD;
                        DATA_BUF1 <=
( not UxQtpQxUt(31) ) & UxQtpQxUt(30 downto 0);
                        ADRS_BUF1 <= '0' & j & i ;
                        ALU_ACK <= '1';
        else
                        ALU_STATE <= CALC_LR;
                        ADRS_BUF1 <= '0' & j & i;
                        ADRS_BUF2 <= '0' & j & i;
                end if;

                if CALC_DONE = '1' then
                        ALU_ACK <= '0';
                        if i < N then
                                i := i + '1';
                                HS_STATE <= HsUxQtCalc;
                        elsif j < N then
                                j := j + '1';
                                i := k + '1';
                                HS_STATE <= HsUxQtCalc;
                        elsif k < ( N - "00000010" ) then
                                k := k + '1';
                                HS_STATE <= HsVarIni;
                        else
                                i := ( 0 => '1', others => '0' );
                                j := ( 0 => '1', others => '0' );
                                HS_STATE <= HsResRead;
                        end if;
                end if;

        when HsResRead =>
                if ALU_ACK = '0' then
                        ALU_STATE <= R_MEM_RD;
                        ADRS_BUF1 <= '0' & j & i;
                        ALU_ACK <= '1';
                else
                        ALU_STATE <= MEM_STOP;
                end if;

                if CALC_DONE = '1' then
                        BH <= ( 0 => '1', others => '0' );
                        RES <= DATA_BUS;
                end if;

                if OUT_ACK2 = '1' then
                        BH <= ( others => '0' );
                        ALU_ACK <= '0';
                        if i < N then
                                i := i + '1';
                        elsif j < N then
                                j := j + '1';
                                i := ( 0 => '1', others => '0' );
                        else
                                i := ( 0 => '1', others => '0' );
                                j := ( 0 => '1', others => '0' );
                        end if;
                end if;
```

```
                when others =>
                        null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DCNT <= '0';
--        DCNT3 <= "0000";
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then
--if DIV_ACK = '1' and DIV_DONE = '0' then
--        DCNT3 <= DCNT3 + '1';
--else
--        DCNT3 <= "0000";
--end if;
--
--if DCNT3 = "1111" then
--        DIV_DONE <= '1';
--else
--        DIV_DONE <= '0';
--end if;

        if DIV_ACK = '1' and DCNT = '0' and DIV_DONE = '0' then
                DCNT <= '1';
        else
                DCNT <= '0';
        end if;
        if DCNT = '1' then
                DIV_DONE <= '1';
        else
                DIV_DONE <= '0';
        end if;
end if;
end process;

-------------<< Floating Point Number Divider >>---------------
divider : fpdiv port map ( CLK => CLK, FA => DIV_A, FB => DIV_B, Q => DIV_Q );

end RTL;
```

I.3.3 二分法

```
------------------------------------------------------------
-- Bisection Method  (Lower FLEX10k)
-- < bisec.vhd >
-- 1999/03/15 (Mon)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity bisec is
  port (CLK  : in std_logic;
                A    : inout std_logic_vector(15 downto 0);
                BL   : in std_logic_vector(7 downto 0);
                BH   : out std_logic_vector(5 downto 0);
                B_CONF  : in std_logic_vector(1 downto 0);
                CL   : in std_logic_vector(5 downto 0);

                DATA_BUS  : inout std_logic_vector(31 downto 0);
                ADRS_BUS  : out std_logic_vector(16 downto 0);
                CTRL_BUS  : out std_logic_vector(4 downto 0);
                CALC_DONE : in  std_logic;
                OE_ALU    : out std_logic;
```

```
                OBF  : in std_logic_vector(1 downto 0);
                ACK  : out std_logic_vector(1 downto 0);
                STB  : out std_logic_vector(1 downto 0);
                IBF  : in std_logic_vector(1 downto 0)
);

attribute pinnum of CLK : signal is  "D22";
attribute pinnum of A   : signal is "BC23,BB24,BC25,BB26,BC27,BB28,BC29,BB30,
BC31,BB32,BC33,BB34,BC35,BB36,BC37,BB38";
attribute pinnum of BL  : signal is  "BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20";
attribute pinnum of BH  : signal is  "BC7,BB8,BC9,BB10,BC11,BB12";
attribute pinnum of B_CONF  : signal is  "BC5,BB6";
attribute pinnum of CL   : signal is  "AU23,AV24,AU25,AU33,AV34,AU35";

attribute pinnum of DATA_BUS  : signal is "A5,B6,A7,B8,A9,B10,A11,B12,A13,B14,
A15,B16,A17,B18,A19,B20,A23,B24,A25,B26,A27,B28,A29,B30,A31,B32,A33,B34,A35,
B36,A37,B38";
attribute pinnum of ADRS_BUS  : signal is "F16,G19,F20,G21,F22,G23,F24,G25,F26,
G29,F30,G31,F32,G33,F34,G35,F36";
attribute pinnum of CTRL_BUS  : signal is  "G11,F12,G13,F14,G15";
attribute pinnum of CALC_DONE : signal is  "F10";
attribute pinnum of OE_ALU    : signal is  "G9";

attribute pinnum of OBF : signal is  "AV18,AV28";
attribute pinnum of ACK : signal is  "AU19,AU29";
attribute pinnum of STB : signal is  "AU21,AU31";
attribute pinnum of IBF : signal is  "AV20,AV30";

end bisec;
architecture RTL of bisec is

-------------<< Floating point Number Divider >>-------
component fpdiv is
 port (CLK : in std_logic;
                FA  : in std_logic_vector(31 downto 0);
                FB  : in std_logic_vector(31 downto 0);
                Q   : out std_logic_vector(31 downto 0)
    );
end component;

signal DIV_A : std_logic_vector(31 downto 0);
signal DIV_B : std_logic_vector(31 downto 0);
signal DIV_Q : std_logic_vector(31 downto 0);
signal DIV_ACK   : std_logic;
signal DIV_DONE  : std_logic;
signal DCNT : std_logic;
signal DCNT3 : std_logic_vector(3 downto 0);

signal A_REG : std_logic_vector(15 downto 0);
signal ACK_BUF : std_logic_vector(1 downto 0);
signal STB_BUF : std_logic_vector(1 downto 0);
signal OUT_CNT : std_logic;
signal OUT_ACK : std_logic;
signal OUT_ACK2 : std_logic;

signal N : std_logic_vector(7 downto 0);

--------------------------------

type CALC_STATE_TYPE is (
                FIRST_DATA, SECOND_DATA
                );

signal CALC_STATE : CALC_STATE_TYPE;

type ALU_STATE_TYPE is (
                R_MEM_WR, R_MEM_RD,
                L_MEM_WR, L_MEM_RD,
                LR_MEM_WR,
                MEM_STOP,
                RR_INPRO, LL_INPRO, LR_INPRO,
                INPRO_F, INPRO_R, INPRO_L, INPRO_LR,
```

```
                FF_MUL, FR_MUL, FL_MUL, RR_MUL, LL_MUL, LR_MUL,
                FF_ADD, FR_ADD, FL_ADD, RR_ADD, LL_ADD, LR_ADD,
                CALC_F, CALC_R, CALC_L, CALC_LR,
                aSTOP
                );

signal ALU_STATE : ALU_STATE_TYPE;
signal DATA_BUS_BUF : std_logic_vector(31 downto 0);
signal DATA_BUF1 : std_logic_vector(31 downto 0);
signal DATA_BUF2 : std_logic_vector(31 downto 0);
signal OE_BUF : std_logic;
signal ADRS_BUF1 : std_logic_vector(16 downto 0);
signal ADRS_BUF2 : std_logic_vector(16 downto 0);
signal ALU_ACK : std_logic;

--------------------------------

type BIS_STATE_TYPE is (
                BisDimRead,
                BisAlphaRead, BisBetaRead, BisAlphapBetaCalc,
                BisMaxCalc, BisMaxComp, BisMaxSet,
                BisApBCalc, BisCCalc,
                BisAlphamCCalc, BisBeta2Calc, BisBeta2dQCalc, BisQCalc, BisQComp,
                BisNewRange, BisEvWrite, BisResRead,
                BisStop
                );

signal BIS_STATE : BIS_STATE_TYPE;
signal Alpha, Beta, OldBeta, AlphapBeta, tMax, Max, BisA, BisB, ApB, BisC,
AlphamC, Beta2, Beta2dQ, Q   : std_logic_vector(31 downto 0);

signal RES    : std_logic_vector(31 downto 0);

constant BISEC_R  : std_logic_vector(4 downto 0) := "11000"; -- 24

constant FP_ONE : std_logic_vector(31 downto 0)
:= "00111111100000000000000000000000";
constant FP_TWO : std_logic_vector(31 downto 0)
:= "01000000000000000000000000000000";

constant cR_MEM_WR  : std_logic_vector(4 downto 0) := "00001";
constant cR_MEM_RD  : std_logic_vector(4 downto 0) := "00010";
constant cL_MEM_WR  : std_logic_vector(4 downto 0) := "00011";
constant cL_MEM_RD  : std_logic_vector(4 downto 0) := "00100";
constant cLR_MEM_WR : std_logic_vector(4 downto 0) := "00101";
constant cMEM_STOP  : std_logic_vector(4 downto 0) := "00110";
constant cRR_INPRO  : std_logic_vector(4 downto 0) := "00111";
constant cLL_INPRO  : std_logic_vector(4 downto 0) := "01000";
constant cLR_INPRO  : std_logic_vector(4 downto 0) := "01001";
constant cINPRO_F   : std_logic_vector(4 downto 0) := "01010";
constant cINPRO_R   : std_logic_vector(4 downto 0) := "01011";
constant cINPRO_L   : std_logic_vector(4 downto 0) := "01100";
constant cINPRO_LR  : std_logic_vector(4 downto 0) := "01101";
constant cFF_MUL    : std_logic_vector(4 downto 0) := "01110";
constant cFR_MUL    : std_logic_vector(4 downto 0) := "01111";
constant cFL_MUL    : std_logic_vector(4 downto 0) := "10000";
constant cRR_MUL    : std_logic_vector(4 downto 0) := "10001";
constant cLL_MUL    : std_logic_vector(4 downto 0) := "10010";
constant cLR_MUL    : std_logic_vector(4 downto 0) := "10011";
constant cFF_ADD    : std_logic_vector(4 downto 0) := "10100";
constant cFR_ADD    : std_logic_vector(4 downto 0) := "10101";
constant cFL_ADD    : std_logic_vector(4 downto 0) := "10110";
constant cRR_ADD    : std_logic_vector(4 downto 0) := "10111";
constant cLL_ADD    : std_logic_vector(4 downto 0) := "11000";
constant cLR_ADD    : std_logic_vector(4 downto 0) := "11001";
constant cCALC_F    : std_logic_vector(4 downto 0) := "11010";
constant cCALC_R    : std_logic_vector(4 downto 0) := "11011";
constant cCALC_L    : std_logic_vector(4 downto 0) := "11100";
constant cCALC_LR   : std_logic_vector(4 downto 0) := "11101";
```

```
begin

A <= "ZZZZZZZZZZZZZZZZ" when ACK_BUF = "00" else A_REG;

ACK_BUF <= OBF;
ACK <= ACK_BUF;

------------<< Output Result Data to PC >>------------
process ( BL(0), BL(3), IBF ) begin
if BL(0) = '1' then
        STB <= "11";
        OUT_CNT <= '0';
        A_REG <= ( others => '0' );
elsif rising_edge( BL(3) ) then
        STB <= "00";
        if OUT_CNT = '0' then
                A_REG <= RES(15 downto 0);
                OUT_CNT <= '1';
        else
                A_REG <= RES(31 downto 16);
                OUT_CNT <= '0';
        end if;
end if;
if IBF = "11" then
        STB <= "11";
end if;
end process;

------------<< Generate Memory Read Signal of Result Data >>------------
process ( BL(0), BL(4), OUT_ACK2 ) begin
if BL(0) = '1' or OUT_ACK2 = '1' then
        OUT_ACK <= '0';
elsif rising_edge( BL(4) ) then
        OUT_ACK <= '1';
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        OUT_ACK2 <= '0';
elsif rising_edge( CLK ) then
        if OUT_ACK = '1' then
                OUT_ACK2 <= '1';
        else
                OUT_ACK2 <= '0';
        end if;
end if;
end process;

-----<< Selecter of ALU Operation >>-----
DATA_BUS <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_BUF = '1' else DATA_BUS_BUF;
OE_ALU <= OE_BUF;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CTRL_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR =>
                        CTRL_BUS <= cR_MEM_WR;
                when R_MEM_RD =>
                        CTRL_BUS <= cR_MEM_RD;
                when L_MEM_WR =>
                        CTRL_BUS <= cL_MEM_WR;
                when L_MEM_RD =>
                        CTRL_BUS <= cL_MEM_RD;
                when LR_MEM_WR =>
                        CTRL_BUS <= cLR_MEM_WR;
                when MEM_STOP =>
                        CTRL_BUS <= cMEM_STOP;
                when RR_INPRO =>
                        CTRL_BUS <= cRR_INPRO;
                when LL_INPRO =>
                        CTRL_BUS <= cLL_INPRO;
```

```
                when LR_INPRO =>
                        CTRL_BUS <= cLR_INPRO;
                when INPRO_F =>
                        CTRL_BUS <= cINPRO_F;
                when INPRO_R =>
                        CTRL_BUS <= cINPRO_R;
                when INPRO_L =>
                        CTRL_BUS <= cINPRO_L;
                when INPRO_LR =>
                        CTRL_BUS <= cINPRO_LR;
                when FF_MUL =>
                        CTRL_BUS <= cFF_MUL;
                when FR_MUL =>
                        CTRL_BUS <= cFR_MUL;
                when FL_MUL =>
                        CTRL_BUS <= cFL_MUL;
                when RR_MUL =>
                        CTRL_BUS <= cRR_MUL;
                when LL_MUL =>
                        CTRL_BUS <= cLL_MUL;
                when LR_MUL =>
                        CTRL_BUS <= cLR_MUL;
                when FF_ADD =>
                        CTRL_BUS <= cFF_ADD;
                when FR_ADD =>
                        CTRL_BUS <= cFR_ADD;
                when FL_ADD =>
                        CTRL_BUS <= cFL_ADD;
                when RR_ADD =>
                        CTRL_BUS <= cRR_ADD;
                when LL_ADD =>
                        CTRL_BUS <= cLL_ADD;
                when LR_ADD =>
                        CTRL_BUS <= cLR_ADD;
                when CALC_F =>
                        CTRL_BUS <= cCALC_F;
                when CALC_R =>
                        CTRL_BUS <= cCALC_R;
                when CALC_L =>
                        CTRL_BUS <= cCALC_L;
                when CALC_LR =>
                        CTRL_BUS <= cCALC_LR;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DATA_BUS_BUF <= ( others => '0' );
        OE_BUF <= '0';
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR | L_MEM_WR | LR_MEM_WR | FR_MUL | FL_MUL |
FR_ADD | FL_ADD =>
                        DATA_BUS_BUF <= DATA_BUF1;
                        OE_BUF <= '0';
                when MEM_STOP | INPRO_F | CALC_F =>
                        DATA_BUS_BUF <= ( others => '0' );
                        OE_BUF <= '1';
                when LR_INPRO | INPRO_LR | LR_MUL | LR_ADD | CALC_LR =>
                        DATA_BUS_BUF(16 downto 0) <= ADRS_BUF2;
                        OE_BUF <= '0';
                when FF_MUL | FF_ADD =>
                        OE_BUF <= '0';
                        if CALC_STATE = FIRST_DATA then
                                DATA_BUS_BUF <= DATA_BUF1;
                        else
                                DATA_BUS_BUF <= DATA_BUF2;
                        end if;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
```

```
if BL(0) = '1' then
        CALC_STATE <= FIRST_DATA;
elsif rising_edge( CLK ) then
        case ALU_STATE is
                when FF_MUL | FF_ADD =>
                         CALC_STATE <= SECOND_DATA;
                when others =>
                        if CALC_DONE = '1' then
                                CALC_STATE <= FIRST_DATA;
                        end if;
        end case;
end if;
end process;


process ( BL(0), CLK ) begin
if BL(0) = '1' then
        ADRS_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when MEM_STOP | INPRO_F | FF_MUL | FF_ADD | CALC_F =>
                        ADRS_BUS <= ( others => '0' );
                when others =>
                        ADRS_BUS <= ADRS_BUF1;
        end case;
end if;
end process;

------------<< Bisection Method ?>-------------

process( BL(0), CLK )
        variable i, j, k, nPos : std_logic_vector(7 downto 0);
        variable r : std_logic_vector(4 downto 0);
begin
if BL(0) = '1' then
        BIS_STATE <= BisStop;
        ALU_STATE <= aStop;
        ALU_ACK <= '0';
        DIV_ACK <= '0';
        DATA_BUF1 <= (others => '0');
        DATA_BUF2 <= (others => '0');
        ADRS_BUF1 <= (others => '0');
        ADRS_BUF2 <= (others => '0');
--        DIV_A <= (others => '0');
--        DIV_B <= (others => '0');
--        Alpha  <= (others => '0');
--        Beta <= (others => '0');
--        OldBeta <= (others => '0');
--        AlphapBeta <= (others => '0');
--        tMax <= (others => '0');
        Max <= (others => '0');
--        BisA <= (others => '0');
--        BisB <= (others => '0');
--        ApB <= (others => '0');
--        BisC <= (others => '0');
--        AlphamC <= (others => '0');
--        Beta2 <= (others => '0');
--        Beta2dQ <= (others => '0');
--        Q <= (others => '0');
        i := ( 0 => '1', others => '0' );
--        j := ( others => '0' );
        k := ( 0 => '1', others => '0' );
        r := ( others => '0' );
        nPos := ( others => '0' );
        BH <= ( others => '0' );
elsif rising_edge( CLK ) then
        case BIS_STATE is
                when BisStop =>
                        if BL(5) = '1' then
                                BIS_STATE <= BisDimRead;
                        else
                                ALU_STATE <= aSTOP;
                                BIS_STATE <= BisStop;
                        end if;
```

```
when BisDimRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_RD;
                ADRS_BUF1 <= "10000000000000000";
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                N <= DATA_BUS(7 downto 0);
                ALU_ACK <= '0';
                BIS_STATE <= BisAlphaRead;
        end if;

when BisAlphaRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '0' & i & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Alpha <= DATA_BUS;
                ALU_ACK <= '0';
                if i < N then
                        BIS_STATE <= BisBetaRead;
                else
                        BIS_STATE <= BisAlphapBetaCalc;
                end if;
        end if;

when BisBetaRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '0' & i & (i + '1');
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Beta <= DATA_BUS;
                ALU_ACK <= '0';
                BIS_STATE <= BisAlphapBetaCalc;
        end if;

when BisAlphapBetaCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= '0' & Alpha(30 downto 0);
        DATA_BUF2 <= '0' & Beta(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                if i = "00000001" or i = N then
                        tMax <= DATA_BUS;
                        BIS_STATE <= BisMaxComp;
                else
                        AlphapBeta <= DATA_BUS;
                        BIS_STATE <= BisMaxCalc;
                end if;
        end if;

when BisMaxCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= AlphapBeta;
        DATA_BUF2 <= '0' & OldBeta(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
```

```
        end if;

        if CALC_DONE = '1' then
                tMax <= DATA_BUS;
                BIS_STATE <= BisMaxComp;
        end if;

when BisMaxComp =>
        if tMax(30 downto 0) > Max(30 downto 0) then
                Max <= tMax;
        end if;

        if i < N then
                i := i + '1';
                OldBeta <= Beta;
                BIS_STATE <= BisAlphaRead;
        else
                i := ( 0 => '1', others => '0' );
                BIS_STATE <= BisMaxSet;
        end if;

when BisMaxSet =>
        BisA <= '1' & Max(30 downto 0);
        BisB <= '0' & Max(30 downto 0);
        BIS_STATE <= BisApBCalc;

when BisApBCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= BisA;
        DATA_BUF2 <= BisB;
        nPos := (others => '0');

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                ApB <= DATA_BUS;
                BIS_STATE <= BisCCalc;
        end if;

when BisCCalc =>
        if DIV_ACK = '0' then
                DIV_A <= ApB;
                DIV_B <= FP_TWO;
                DIV_ACK <= '1';
        else
                DIV_A <= ( others => '0' );
                DIV_B <= ( others => '0' );
        end if;

        if DIV_DONE = '1' then
                BisC <= DIV_Q;
                DIV_ACK <= '0';
                BIS_STATE <= BisAlphamCCalc;
        end if;

when BisAlphamCCalc =>
        if ALU_ACK = '0' then
                ALU_STATE <= FR_ADD;
                DATA_BUF1 <= ( not BisC(31) ) & BisC(30 downto 0);
                ADRS_BUF1 <= '0' & i & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                if i = "00000001" then
                        Q <= DATA_BUS;
                        BIS_STATE <= BisQComp;
                else
                        AlphamC <= DATA_BUS;
                        BIS_STATE <= BisBeta2Calc;
                end if;
```

```
        end if;

when BisBeta2Calc =>
        if ALU_ACK = '0' then
                ALU_STATE <= RR_MUL;
                ADRS_BUF1 <= '0' & ( i - '1' ) & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                Beta2 <= DATA_BUS;
                ALU_ACK <= '0';
                BIS_STATE <= BisBeta2dQCalc;
        end if;

when BisBeta2dQCalc =>
        if DIV_ACK = '0' then
                DIV_A <= Beta2;
                DIV_B <= Q;
                DIV_ACK <= '1';
        else
                DIV_A <= ( others => '0' );
                DIV_B <= ( others => '0' );
        end if;

        if DIV_DONE = '1' then
                Beta2dQ <= DIV_Q;
                DIV_ACK <= '0';
                BIS_STATE <= BisQCalc;
        end if;

when BisQCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= AlphamC;
        DATA_BUF2 <= ( not Beta2dQ(31) ) & Beta2dQ(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                Q <= DATA_BUS;
                BIS_STATE <= BisQComp;
        end if;

when BisQComp =>
        if Q(31) = '0' then
                nPos := nPos + '1';
        end if;

        if Q(30 downto 0) = "00000000000000000000000000000000" then
                i := i + '1';
        end if;

        if i < N then
                i := i + '1';
                BIS_STATE <= BisAlphamCCalc;
        else
                i := ( 0 => '1', others => '0');
                BIS_STATE <= BisNewRange;
        end if;

when BisNewRange =>
        if nPos >= k then
                BisA <= BisC;
        else
                BisB <= BisC;
        end if;

        if r < BISEC_R then
                r := r + '1';
                BIS_STATE <= BisApBCalc;
        else
                r := (others => '0');
```

```
                                BIS_STATE <= BisEvWrite;
                        end if;

                when BisEvWrite =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_WR;
                                DATA_BUF1 <= BisC;
                                ADRS_BUF1 <= "100000001" & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                ALU_ACK <= '0';
                                if k < N then
                                        k := k + '1';
                                        BisA <= '1' & Max(30 downto 0);
                                        BisB <= BisC;
                                        BIS_STATE <= BisApBCalc;
                                else
                                        BIS_STATE <= BisResRead;
                                end if;
                        end if;

                when BisResRead =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_RD;
                                ADRS_BUF1 <= "100000001" & i;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                BH <= ( 0 => '1', others => '0' );
                                RES <= DATA_BUS;
                        end if;

                        if OUT_ACK2 = '1' then
                                BH <= ( others => '0' );
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                else
                                        i := ( 0 => '1', others => '0' );
                                end if;
                        end if;

                when others =>
                        null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DCNT <= '0';
--        DCNT3 <= "0000";
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then
--if DIV_ACK = '1' and DIV_DONE = '0' then
--        DCNT3 <= DCNT3 + '1';
--else
--        DCNT3 <= "0000";
--end if;
--
--if DCNT3 = "1111" then
--        DIV_DONE <= '1';
--else
--        DIV_DONE <= '0';
--end if;

        if DIV_ACK = '1' and DCNT = '0' and DIV_DONE = '0' then
                DCNT <= '1';
        else
```

```
                DCNT <= '0';
        end if;
        if DCNT = '1' then
                DIV_DONE <= '1';
        else
                DIV_DONE <= '0';
        end if;
end if;
end process;


------------<< Floating Point Number Divider >>--------------
divider : fpdiv  port map ( CLK => CLK, FA => DIV_A, FB => DIV_B, Q => DIV_Q );

end RTL;
```

I.3.4 逆反復法

```
------------------------------------------------------------
-- Inverse Iteration Method  (Lower FLEX10k)
-- < invit.vhd >
-- 1999/03/15 (Fri)
-- yamaoka@tube.ee.uec.ac.jp
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity invit is
  port (CLK  : in std_logic;
                A    : inout std_logic_vector(15 downto 0);
                BL   : in std_logic_vector(7 downto 0);
                BH   : out std_logic_vector(5 downto 0);
B_CONF  : in std_logic_vector(1 downto 0);
                CL   : in std_logic_vector(5 downto 0);

                DATA_BUS  : inout std_logic_vector(31 downto 0);
                ADRS_BUS  : out std_logic_vector(16 downto 0);
                CTRL_BUS  : out std_logic_vector(4 downto 0);
                CALC_DONE : in  std_logic;
                OE_ALU    : out std_logic;

                OBF  : in std_logic_vector(1 downto 0);
                ACK  : out std_logic_vector(1 downto 0);
                STB  : out std_logic_vector(1 downto 0);
                IBF  : in std_logic_vector(1 downto 0)
);

attribute pinnum of CLK : signal is  "D22";
attribute pinnum of A   : signal is "BC23,BB24,BC25,BB26,BC27,BB28,BC29,BB30,
BC31,BB32,BC33,BB34,BC35,BB36,BC37,BB38";
attribute pinnum of BL  : signal is  "BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20";
attribute pinnum of BH  : signal is  "BC7,BB8,BC9,BB10,BC11,BB12";
attribute pinnum of B_CONF  : signal is  "BC5,BB6";
attribute pinnum of CL   : signal is  "AU23,AV24,AU25,AU33,AV34,AU35";

attribute pinnum of DATA_BUS  : signal is "A5,B6,A7,B8,A9,B10,A11,B12,A13,B14,
A15,B16,A17,B18,A19,B20,A23,B24,A25,B26,A27,B28,A29,B30,A31,B32,A33,B34,A35,B36,
A37,B38";
attribute pinnum of ADRS_BUS  : signal is "F16,G19,F20,G21,F22,G23,F24,G25,F26,
G29,F30,G31,F32,G33,F34,G35,F36";
attribute pinnum of CTRL_BUS  : signal is  "G11,F12,G13,F14,G15";
attribute pinnum of CALC_DONE : signal is  "F10";
attribute pinnum of OE_ALU    : signal is  "G9";

attribute pinnum of OBF : signal is  "AV18,AV28";
attribute pinnum of ACK : signal is  "AU19,AU29";
```

```
attribute pinnum of STB : signal is  "AU21,AU31";
attribute pinnum of IBF : signal is  "AV20,AV30";

end invit;
architecture RTL of invit is

-------------<< Floating point Number Divider >>-------
component fpdiv is
 port (CLK : in std_logic;
               FA  : in std_logic_vector(31 downto 0);
               FB  : in std_logic_vector(31 downto 0);
               Q   : out std_logic_vector(31 downto 0)
    );
end component;

signal DIV_A : std_logic_vector(31 downto 0);
signal DIV_B : std_logic_vector(31 downto 0);
signal DIV_Q : std_logic_vector(31 downto 0);
signal DIV_ACK   : std_logic;
signal DIV_DONE  : std_logic;
signal DCNT : std_logic;
signal DCNT3 : std_logic_vector(3 downto 0);

signal A_REG : std_logic_vector(15 downto 0);
signal ACK_BUF : std_logic_vector(1 downto 0);
signal STB_BUF : std_logic_vector(1 downto 0);
signal OUT_CNT : std_logic;
signal OUT_ACK : std_logic;
signal OUT_ACK2 : std_logic;

signal N : std_logic_vector(7 downto 0);

-------------------------------

type CALC_STATE_TYPE is (
               FIRST_DATA, SECOND_DATA
               );

signal CALC_STATE : CALC_STATE_TYPE;

type ALU_STATE_TYPE is (
               R_MEM_WR, R_MEM_RD,
               L_MEM_WR, L_MEM_RD,
               LR_MEM_WR,
               MEM_STOP,
               RR_INPRO, LL_INPRO, LR_INPRO,
               INPRO_F, INPRO_R, INPRO_L, INPRO_LR,
               FF_MUL, FR_MUL, FL_MUL, RR_MUL, LL_MUL, LR_MUL,
               FF_ADD, FR_ADD, FL_ADD, RR_ADD, LL_ADD, LR_ADD,
               CALC_F, CALC_R, CALC_L, CALC_LR,
               aSTOP
               );

signal ALU_STATE : ALU_STATE_TYPE;
signal DATA_BUS_BUF : std_logic_vector(31 downto 0);
signal DATA_BUF1 : std_logic_vector(31 downto 0);
signal DATA_BUF2 : std_logic_vector(31 downto 0);
signal OE_BUF : std_logic;
signal ADRS_BUF1 : std_logic_vector(16 downto 0);
signal ADRS_BUF2 : std_logic_vector(16 downto 0);
signal ALU_ACK : std_logic;

-------------------------------

type INV_STATE_TYPE is (
               InvDimRead,
               InvLambdaRead, InvAlphamLambdaCalc,
               InvViceRead, InvViceWrite,
               InvAlpha1Read, InvAlpha2Read,
               InvBeta1Read, InvBeta2Read, InvBeta3Read,
               InvAbsComp, InvMCalc, InvMWrite,
               InvAlpha1Write, InvBeta2Write, InvBeta3Write,
               InvMxBeta3Calc, InvMxBeta2Calc, InvAlpha2mMxBeta2Calc,
```

```
                InvRRange, InvRxXCalc, InvXRead, InvXmRxXCalc,
                InvAlphaRead, InvXCalc, InvXWrite,
                InvX1Read, InvX2Read, InvX1X2Comp, InvX1Write, InvMxX1Calc,
                InvEvCalc,
                InvResRead,
                InvStop
                );

signal INV_STATE : INV_STATE_TYPE;
signal Lambda, Vice, Alpha1, Alpha2, Beta1, Beta2, Beta3, M, MxBeta2, X, RxX,
XmRxX, Alpha, X1, X2, MxX1 : std_logic_vector(31 downto 0);

signal Ex     : std_logic_vector(254 downto 1);
signal Rrange : std_logic_vector(7 downto 0);
signal RES    : std_logic_vector(31 downto 0);

constant FP_ONE : std_logic_vector(31 downto 0)
:= "00111111100000000000000000000000";
constant FP_TWO : std_logic_vector(31 downto 0)
:= "01000000000000000000000000000000";

constant cR_MEM_WR  : std_logic_vector(4 downto 0) := "00001";
constant cR_MEM_RD  : std_logic_vector(4 downto 0) := "00010";
constant cL_MEM_WR  : std_logic_vector(4 downto 0) := "00011";
constant cL_MEM_RD  : std_logic_vector(4 downto 0) := "00100";
constant cLR_MEM_WR : std_logic_vector(4 downto 0) := "00101";
constant cMEM_STOP  : std_logic_vector(4 downto 0) := "00110";
constant cRR_INPRO  : std_logic_vector(4 downto 0) := "00111";
constant cLL_INPRO  : std_logic_vector(4 downto 0) := "01000";
constant cLR_INPRO  : std_logic_vector(4 downto 0) := "01001";
constant cINPRO_F   : std_logic_vector(4 downto 0) := "01010";
constant cINPRO_R   : std_logic_vector(4 downto 0) := "01011";
constant cINPRO_L   : std_logic_vector(4 downto 0) := "01100";
constant cINPRO_LR  : std_logic_vector(4 downto 0) := "01101";
constant cFF_MUL    : std_logic_vector(4 downto 0) := "01110";
constant cFR_MUL    : std_logic_vector(4 downto 0) := "01111";
constant cFL_MUL    : std_logic_vector(4 downto 0) := "10000";
constant cRR_MUL    : std_logic_vector(4 downto 0) := "10001";
constant cLL_MUL    : std_logic_vector(4 downto 0) := "10010";
constant cLR_MUL    : std_logic_vector(4 downto 0) := "10011";
constant cFF_ADD    : std_logic_vector(4 downto 0) := "10100";
constant cFR_ADD    : std_logic_vector(4 downto 0) := "10101";
constant cFL_ADD    : std_logic_vector(4 downto 0) := "10110";
constant cRR_ADD    : std_logic_vector(4 downto 0) := "10111";
constant cLL_ADD    : std_logic_vector(4 downto 0) := "11000";
constant cLR_ADD    : std_logic_vector(4 downto 0) := "11001";
constant cCALC_F    : std_logic_vector(4 downto 0) := "11010";
constant cCALC_R    : std_logic_vector(4 downto 0) := "11011";
constant cCALC_L    : std_logic_vector(4 downto 0) := "11100";
constant cCALC_LR   : std_logic_vector(4 downto 0) := "11101";

begin

A <= "ZZZZZZZZZZZZZZZZ" when ACK_BUF = "00" else A_REG;

ACK_BUF <= OBF;
ACK <= ACK_BUF;

------------<< Output Result Data to PC >>-------------
process ( BL(0), BL(3), IBF ) begin
if BL(0) = '1' then
        STB <= "11";
        OUT_CNT <= '0';
        A_REG <= ( others => '0' );
elsif rising_edge( BL(3) ) then
        STB <= "00";
        if OUT_CNT = '0' then
                A_REG <= RES(15 downto 0);
                OUT_CNT <= '1';
```

```
        else
                A_REG <= RES(31 downto 16);
                OUT_CNT <= '0';
        end if;
end if;
if IBF = "11" then
        STB <= "11";
end if;
end process;

------------<< Generate Memory Read Signal of Result Data >>-------------
process ( BL(0), BL(4), OUT_ACK2 ) begin
if BL(0) = '1' or OUT_ACK2 = '1' then
        OUT_ACK <= '0';
elsif rising_edge( BL(4) ) then
        OUT_ACK <= '1';
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        OUT_ACK2 <= '0';
elsif rising_edge( CLK ) then
        if OUT_ACK = '1' then
                OUT_ACK2 <= '1';
        else
                OUT_ACK2 <= '0';
        end if;
end if;
end process;

-----<< Selecter of ALU Operation >>-----
DATA_BUS <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_BUF = '1' else DATA_BUS_BUF;
OE_ALU <= OE_BUF;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CTRL_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR =>
                        CTRL_BUS <= cR_MEM_WR;
                when R_MEM_RD =>
                        CTRL_BUS <= cR_MEM_RD;
                when L_MEM_WR =>
                        CTRL_BUS <= cL_MEM_WR;
                when L_MEM_RD =>
                        CTRL_BUS <= cL_MEM_RD;
                when LR_MEM_WR =>
                        CTRL_BUS <= cLR_MEM_WR;
                when MEM_STOP =>
                        CTRL_BUS <= cMEM_STOP;
                when RR_INPRO =>
                        CTRL_BUS <= cRR_INPRO;
                when LL_INPRO =>
                        CTRL_BUS <= cLL_INPRO;
                when LR_INPRO =>
                        CTRL_BUS <= cLR_INPRO;
                when INPRO_F =>
                        CTRL_BUS <= cINPRO_F;
                when INPRO_R =>
                        CTRL_BUS <= cINPRO_R;
                when INPRO_L =>
                        CTRL_BUS <= cINPRO_L;
                when INPRO_LR =>
                        CTRL_BUS <= cINPRO_LR;
                when FF_MUL =>
                        CTRL_BUS <= cFF_MUL;
                when FR_MUL =>
                        CTRL_BUS <= cFR_MUL;
                when FL_MUL =>
                        CTRL_BUS <= cFL_MUL;
                when RR_MUL =>
                        CTRL_BUS <= cRR_MUL;
                when LL_MUL =>
```

```
                        CTRL_BUS <= cLL_MUL;
                when LR_MUL =>
                        CTRL_BUS <= cLR_MUL;
                when FF_ADD =>
                        CTRL_BUS <= cFF_ADD;
                when FR_ADD =>
                        CTRL_BUS <= cFR_ADD;
                when FL_ADD =>
                        CTRL_BUS <= cFL_ADD;
                when RR_ADD =>
                        CTRL_BUS <= cRR_ADD;
                when LL_ADD =>
                        CTRL_BUS <= cLL_ADD;
                when LR_ADD =>
                        CTRL_BUS <= cLR_ADD;
                when CALC_F =>
                        CTRL_BUS <= cCALC_F;
                when CALC_R =>
                        CTRL_BUS <= cCALC_R;
                when CALC_L =>
                        CTRL_BUS <= cCALC_L;
                when CALC_LR =>
                        CTRL_BUS <= cCALC_LR;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DATA_BUS_BUF <= ( others => '0' );
        OE_BUF <= '0';
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR | L_MEM_WR | LR_MEM_WR | FR_MUL | FL_MUL |
 FR_ADD | FL_ADD =>
                           DATA_BUS_BUF <= DATA_BUF1;
                        OE_BUF <= '0';
                when MEM_STOP | INPRO_F | CALC_F =>
                        DATA_BUS_BUF <= ( others => '0' );
                        OE_BUF <= '1';
                when LR_INPRO | INPRO_LR | LR_MUL | LR_ADD | CALC_LR =>
                        DATA_BUS_BUF(16 downto 0) <= ADRS_BUF2;
                        OE_BUF <= '0';
                when FF_MUL | FF_ADD =>
                        OE_BUF <= '0';
                        if CALC_STATE = FIRST_DATA then
                                DATA_BUS_BUF <= DATA_BUF1;
                        else
                                DATA_BUS_BUF <= DATA_BUF2;
                        end if;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CALC_STATE <= FIRST_DATA;
elsif rising_edge( CLK ) then
        case ALU_STATE is
                when FF_MUL | FF_ADD =>
                         CALC_STATE <= SECOND_DATA;
                when others =>
                        if CALC_DONE = '1' then
                                CALC_STATE <= FIRST_DATA;
                        end if;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        ADRS_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
```

```
        case ALU_STATE is
                when MEM_STOP | INPRO_F | FF_MUL | FF_ADD | CALC_F =>
                        ADRS_BUS <= ( others => '0' );
                when others =>
                        ADRS_BUS <= ADRS_BUF1;
        end case;
end if;
end process;

------------<< Inverse Iteration Method >>-------------
process( BL(0), CLK )
        variable i, j, k : std_logic_vector(7 downto 0);
        variable r : std_logic;
begin
if BL(0) = '1' then
        INV_STATE <= InvStop;
        ALU_STATE <= aStop;
        ALU_ACK <= '0';
        DIV_ACK <= '0';
        DATA_BUF1 <= (others => '0');
        DATA_BUF2 <= (others => '0');
        ADRS_BUF1 <= (others => '0');
        ADRS_BUF2 <= (others => '0');
--        DIV_A <= (others => '0');
--        DIV_B <= (others => '0');
--        Lambda  <= (others => '0');
--        Vice <= (others => '0');
--        Alpha1 <= (others => '0');
--        Alpha2 <= (others => '0');
--        Beta1 <= (others => '0');
--        Beta2 <= (others => '0');
--        Beta3 <= (others => '0');
--        M <= (others => '0');
--        MxBeta2 <= (others => '0');
--        X <= (others => '0');
--        RxX <= (others => '0');
--        XmRxX <= (others => '0');
--        Alpha <= (others => '0');
--        X1 <= (others => '0');
--        X2 <= (others => '0');
--        MxX1 <= (others => '0');
--        Res <= (others => '0');
--        Ex <= (others => '0');
--        N <= (others => '0');
        i := ( 0 => '1', others => '0' );
        j := ( 0 => '1', others => '0' );
        k := ( 0 => '1', others => '0' );
        r := '0';
        BH <= ( others => '0' );
elsif rising_edge( CLK ) then
        case INV_STATE is
                when InvStop =>
                        if BL(5) = '1' then
                                INV_STATE <= InvDimRead;
                        else
                                ALU_STATE <= aSTOP;
                                INV_STATE <= InvStop;
                        end if;

                when InvDimRead =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_RD;
                                ADRS_BUF1 <= "10000000000000000";
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                N <= DATA_BUS(7 downto 0);
                                ALU_ACK <= '0';
                                INV_STATE <= InvLambdaRead;
                        end if;

                when InvLambdaRead =>
```

```
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_RD;
                                ADRS_BUF1 <= "100000001" & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                Lambda <= DATA_BUS;
                                ALU_ACK <= '0';
                                INV_STATE <= InvAlphamLambdaCalc;
                        end if;

                when InvAlphamLambdaCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FR_ADD;
                                DATA_BUF1
<= ( not Lambda(31) ) & Lambda(30 downto 0);
                                ADRS_BUF1 <= '0' & i & i;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_L;
                                ADRS_BUF1 <= '0' & i & i;
                        end if;

                        if CALC_DONE = '1' then
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                else
                                        i := ( 0 => '1', others => '0' );
                                        INV_STATE <= InvViceRead;
                                end if;
                        end if;

                when InvViceRead =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= R_MEM_RD;
                                ADRS_BUF1 <= '0' & i & (i + '1');
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                Vice <= DATA_BUS;
                                ALU_ACK <= '0';
                                INV_STATE <= InvViceWrite;
                        end if;

                when InvViceWrite =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_WR;
                                DATA_BUF1 <= Vice;
                                ADRS_BUF1 <= '0' & i & (i + '1');
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                ALU_ACK <= '0';
                                if i < ( N - '1' ) then
                                        i := i + '1';
                                        INV_STATE <= InvViceRead;
                                else
                                        i := ( 0 => '1', others => '0' );
                                        INV_STATE <= InvAlpha1Read;
                                end if;
                        end if;

                when InvAlpha1Read =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_RD;
                                ADRS_BUF1 <= '0' & i & i;
```

```
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Alpha1 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvAlpha2Read;
        end if;

when InvAlpha2Read =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_RD;
                ADRS_BUF1 <= '0' & (i + '1') & (i + '1');
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Alpha2 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvBeta1Read;
        end if;

when InvBeta1Read =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '0' & i & (i + '1');
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Beta1 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvBeta2Read;
        end if;

when InvBeta2Read =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_RD;
                ADRS_BUF1 <= '0' & i & (i + '1');
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Beta2 <= DATA_BUS;
                ALU_ACK <= '0';
                if i < (N - '1') then
                        INV_STATE <= InvBeta3Read;
                else
                        INV_STATE <= InvAbsComp;
                end if;
        end if;

when InvBeta3Read =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_RD;
                ADRS_BUF1 <= '0' & (i + '1') & (i + "00000010");
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Beta3 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvAbsComp;
        end if;

when InvAbsComp =>
```

```
        if Alpha1(30 downto 0) < Beta1(30 downto 0) then
                Ex(conv_integer(i)) <= '1';
                Alpha1 <= Beta1;
                Beta1 <= Alpha1;
                Alpha2 <= Beta2;
                Beta2 <= Alpha2;
        else
                Ex(conv_integer(i)) <= '0';
        end if;
        INV_STATE <= InvMCalc;

when InvMCalc =>
        if DIV_ACK = '0' then
                DIV_A <= Beta1;
                DIV_B <= Alpha1;
                DIV_ACK <= '1';
        else
                DIV_A <= ( others => '0' );
                DIV_B <= ( others => '0' );
        end if;

        if DIV_DONE = '1' then
                M <= DIV_Q;
                DIV_ACK <= '0';
                INV_STATE <= InvMWrite;
        end if;

when InvMWrite =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= M;
                ADRS_BUF1 <= "100000010" & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                if Ex(conv_integer(i)) = '1' then
                        INV_STATE <= InvAlpha1Write;
                else
                        INV_STATE <= InvMxBeta2Calc;
                end if;
        end if;

when InvAlpha1Write =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= Alpha1;
                ADRS_BUF1 <= '0' & i & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                INV_STATE <= InvBeta2Write;
        end if;

when InvBeta2Write =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= Beta2;
                ADRS_BUF1 <= '0' & i & (i + '1');
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                if i < (N - '1') then
                        INV_STATE <= InvBeta3Write;
                else
```

```
                        INV_STATE <= InvMxBeta2Calc;
                end if;
        end if;

when InvBeta3Write =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= Beta3;
                ADRS_BUF1 <= '0' & i & (i + "00000010");
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                INV_STATE <= InvMxBeta3Calc;
        end if;

when InvMxBeta3Calc =>
        ALU_STATE <= FF_MUL;
        DATA_BUF1 <= M;
        DATA_BUF2 <= ( not Beta3(31) ) & Beta3(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_L;
                ADRS_BUF1 <= '0' & (i + '1') & (i + "00000010");
        end if;

        if CALC_DONE = '1' then
                INV_STATE <= InvMxBeta2Calc;
        end if;

when InvMxBeta2Calc =>
        ALU_STATE <= FF_MUL;
        DATA_BUF1 <= M;
        DATA_BUF2 <= Beta2;

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                MxBeta2 <= DATA_BUS;
                INV_STATE <= InvAlpha2mMxBeta2Calc;
        end if;

when InvAlpha2mMxBeta2Calc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= Alpha2;
        DATA_BUF2 <= ( not MxBeta2(31) ) & MxBeta2(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_L;
                ADRS_BUF1 <= '0' & (i + '1') & (i + '1');
        end if;

        if CALC_DONE = '1' then
                if i < (N - '1') then
                        i := i + '1';
                        INV_STATE <= InvAlpha1Read;
                else
                        i := N;
                        j := N;
                        X <= FP_ONE;
                        RxX <= (others => '0');
                        INV_STATE <= InvXmRxXCalc;
                end if;
        end if;

when InvRRange =>
        if Ex(conv_integer(i)) = '1' then
                Rrange <= i + "00000010";
        else
                Rrange <= i + "00000001";
        end if;
```

```
        if i = (N - '1') then
                Rrange <= N;
        end if;
        INV_STATE <= InvRxXCalc;

when InvRxXCalc =>
        if i < Rrange then
                ALU_STATE <= LR_INPRO;
                i := i + '1';
                ADRS_BUF1 <= '1' & i & k;
                ADRS_BUF2 <= '0' & j & i;
        else
                ALU_STATE <= INPRO_F;
        end if;

        if CALC_DONE = '1' then
                RxX <= DATA_BUS;
                if r = '1' then
                        INV_STATE <= InvXRead;
                else
                        X <= FP_ONE;
                        INV_STATE <= InvXmRxXCalc;
                end if;
        end if;

when InvXRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '1' & j & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                X <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvXmRxXCalc;
        end if;

when InvXmRxXCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= X;
        DATA_BUF2 <= ( not RxX(31) ) & RxX(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                XmRxX <= DATA_BUS;
                INV_STATE <= InvAlphaRead;
        end if;

when InvAlphaRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= L_MEM_RD;
                ADRS_BUF1 <= '0' & j & j;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                Alpha <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvXCalc;
        end if;

when InvXCalc =>
        if DIV_ACK = '0' then
                DIV_A <= XmRxX;
                DIV_B <= Alpha;
                DIV_ACK <= '1';
        else
```

```
                DIV_A <= ( others => '0' );
                DIV_B <= ( others => '0' );
        end if;

        if DIV_DONE = '1' then
                X <= DIV_Q;
                DIV_ACK <= '0';
                INV_STATE <= InvXWrite;
        end if;

when InvXWrite =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_WR;
                DATA_BUF1 <= X;
                ADRS_BUF1 <= '1' & j & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                if j > "00000001" then
                        j := j - '1';
                        i := j;
                        INV_STATE <= InvRRange;
                elsif r = '1' then
                        j := ( 0 => '1', others => '0' );
                        i := ( 0 => '1', others => '0' );
                        if k < N then
                                k := k + '1';
                                r := '0';
                                INV_STATE <= InvLambdaRead;
                        else
                                INV_STATE <= InvResRead;
                        end if;
                else
                        j := ( 0 => '1', others => '0' );
                        i := ( 0 => '1', others => '0' );
                        INV_STATE <= InvX1Read;
                end if;
        end if;

when InvX1Read =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '1' & i & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                X1 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvX2Read;
        end if;

when InvX2Read =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '1' & (i + '1') & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                X2 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvX1X2Comp;
        end if;

when InvX1X2Comp =>
        if Ex(conv_integer(i)) = '1' then
```

```
                X1 <= X2;
                X2 <= X1;
                INV_STATE <= InvX1Write;
        else
                INV_STATE <= InvMxX1Calc;
        end if;

when InvX1Write =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_WR;
                DATA_BUF1 <= X1;
                ADRS_BUF1 <= '1' & i & k;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                ALU_ACK <= '0';
                INV_STATE <= InvMxX1Calc;
        end if;

when InvMxX1Calc =>
        if ALU_ACK = '0' then
                ALU_STATE <= FL_MUL;
                DATA_BUF1 <= X1;
                ADRS_BUF1 <= "100000010" & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= CALC_F;
        end if;

        if CALC_DONE = '1' then
                MxX1 <= DATA_BUS;
                ALU_ACK <= '0';
                INV_STATE <= InvEvCalc;
        end if;

when InvEvCalc =>
        ALU_STATE <= FF_ADD;
        DATA_BUF1 <= X2;
        DATA_BUF2 <= ( not MxX1(31) ) & MxX1(30 downto 0);

        if CALC_STATE = SECOND_DATA then
                ALU_STATE <= CALC_R;
                ADRS_BUF1 <= '1' & (i + '1') & k;
        end if;

        if CALC_DONE = '1' then
                if i < (N - '1') then
                        i := i + '1';
                        INV_STATE <= InvX1Read;
                else
                        r := '1';
                        i := N;
                        j := N;
                        RxX <= (others => '0');
                        INV_STATE <= InvXRead;
                end if;
        end if;

when InvResRead =>
        if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                ADRS_BUF1 <= '1' & j & i;
                ALU_ACK <= '1';
        else
                ALU_STATE <= MEM_STOP;
        end if;

        if CALC_DONE = '1' then
                BH <= ( 0 => '1', others => '0' );
                RES <= DATA_BUS;
        end if;

        if OUT_ACK2 = '1' then
```

```
                                BH <= ( others => '0' );
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                elsif j < N then
                                        i := ( 0 => '1', others => '0' );
                                        j := j + '1';
                                else
                                        i := ( 0 => '1', others => '0' );
                                        j := ( 0 => '1', others => '0' );
                                end if;
                        end if;

                when others =>
                        null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DCNT <= '0';
--        DCNT3 <= "0000";
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then
--if DIV_ACK = '1' and DIV_DONE = '0' then
--        DCNT3 <= DCNT3 + '1';
--else
--        DCNT3 <= "0000";
--end if;
--
--if DCNT3 = "1111" then
--        DIV_DONE <= '1';
--else
--        DIV_DONE <= '0';
--end if;

        if DIV_ACK = '1' and DCNT = '0' and DIV_DONE = '0' then
                DCNT <= '1';
        else
                DCNT <= '0';
        end if;
        if DCNT = '1' then
                DIV_DONE <= '1';
        else
                DIV_DONE <= '0';
        end if;
end if;
end process;

------------<< Floating Point Number Divider >>--------------
divider : fpdiv  port map ( CLK => CLK, FA => DIV_A, FB => DIV_B, Q => DIV_Q );

end RTL;
```

I.3.5 ハウスホルダー逆変換

```
-------------------------------------------------------------
-- Householder Inverse Transform (Lower FLEX10k)
-- < hsinv.vhd >
-- 1999/03/15 (Mon)
-- yamaoka@tube.ee.uec.ac.jp
-------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use WORK.MATH.all;

library metamor;
use metamor.attributes.all;

entity hsinv is
```

```
  port (CLK  : in std_logic;
                A    : inout std_logic_vector(15 downto 0);
                BL   : in std_logic_vector(7 downto 0);
                BH   : out std_logic_vector(5 downto 0);
                B_CONF  : in std_logic_vector(1 downto 0);
                CL   : in std_logic_vector(5 downto 0);

                DATA_BUS  : inout std_logic_vector(31 downto 0);
                ADRS_BUS  : out std_logic_vector(16 downto 0);
                CTRL_BUS  : out std_logic_vector(4 downto 0);
                CALC_DONE : in  std_logic;
                OE_ALU    : out std_logic;

                OBF  : in std_logic_vector(1 downto 0);
                ACK  : out std_logic_vector(1 downto 0);
                STB  : out std_logic_vector(1 downto 0);
                IBF  : in std_logic_vector(1 downto 0)
);

attribute pinnum of CLK : signal is  "D22";
attribute pinnum of A   : signal is "BC23,BB24,BC25,BB26,BC27,BB28,BC29,BB30,
BC31,BB32,BC33,BB34,BC35,BB36,BC37,BB38";
attribute pinnum of BL  : signal is  "BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20";
attribute pinnum of BH  : signal is  "BC7,BB8,BC9,BB10,BC11,BB12";
attribute pinnum of B_CONF  : signal is  "BC5,BB6";
attribute pinnum of CL   : signal is  "AU23,AV24,AU25,AU33,AV34,AU35";

attribute pinnum of DATA_BUS  : signal is "A5,B6,A7,B8,A9,B10,A11,B12,A13,B14,
A15,B16,A17,B18,A19,B20,A23,B24,A25,B26,A27,B28,A29,B30,A31,B32,A33,B34,A35,B36,
A37,B38";
attribute pinnum of ADRS_BUS  : signal is "F16,G19,F20,G21,F22,G23,F24,G25,F26,
G29,F30,G31,F32,G33,F34,G35,F36";
attribute pinnum of CTRL_BUS  : signal is  "G11,F12,G13,F14,G15";
attribute pinnum of CALC_DONE : signal is  "F10";
attribute pinnum of OE_ALU    : signal is  "G9";

attribute pinnum of OBF : signal is  "AV18,AV28";
attribute pinnum of ACK : signal is  "AU19,AU29";
attribute pinnum of STB : signal is  "AU21,AU31";
attribute pinnum of IBF : signal is  "AV20,AV30";

end hsinv;
architecture RTL of hsinv is

-------------<< Floating point Number Divider >>-------
component fpdiv is
 port (CLK : in std_logic;
                FA  : in std_logic_vector(31 downto 0);
                FB  : in std_logic_vector(31 downto 0);
                Q   : out std_logic_vector(31 downto 0)
   );
end component;

signal DIV_A : std_logic_vector(31 downto 0);
signal DIV_B : std_logic_vector(31 downto 0);
signal DIV_Q : std_logic_vector(31 downto 0);
signal DIV_ACK   : std_logic;
signal DIV_DONE  : std_logic;
signal DCNT : std_logic;
signal DCNT3 : std_logic_vector(3 downto 0);

signal A_REG : std_logic_vector(15 downto 0);
signal ACK_BUF : std_logic_vector(1 downto 0);
signal STB_BUF : std_logic_vector(1 downto 0);
signal OUT_CNT : std_logic;
signal OUT_ACK : std_logic;
signal OUT_ACK2 : std_logic;

signal N : std_logic_vector(7 downto 0);

--------------------------------
```

```
type CALC_STATE_TYPE is (
                FIRST_DATA, SECOND_DATA
                );

signal CALC_STATE : CALC_STATE_TYPE;

type ALU_STATE_TYPE is (
                R_MEM_WR, R_MEM_RD,
                L_MEM_WR, L_MEM_RD,
                LR_MEM_WR,
                MEM_STOP,
                RR_INPRO, LL_INPRO, LR_INPRO,
                INPRO_F, INPRO_R, INPRO_L, INPRO_LR,
                FF_MUL, FR_MUL, FL_MUL, RR_MUL, LL_MUL, LR_MUL,
                FF_ADD, FR_ADD, FL_ADD, RR_ADD, LL_ADD, LR_ADD,
                CALC_F, CALC_R, CALC_L, CALC_LR,
                aSTOP
                );

signal ALU_STATE : ALU_STATE_TYPE;
signal DATA_BUS_BUF : std_logic_vector(31 downto 0);
signal DATA_BUF1 : std_logic_vector(31 downto 0);
signal DATA_BUF2 : std_logic_vector(31 downto 0);
signal OE_BUF : std_logic;
signal ADRS_BUF1 : std_logic_vector(16 downto 0);
signal ADRS_BUF2 : std_logic_vector(16 downto 0);
signal ALU_ACK : std_logic;

------------------------------

type HSINV_STATE_TYPE is (
                HsInvDimRead,
                HsInvVarIni,
                HsInvUtxXCalc, HsInvUtxXxCCalc, HsInvUtxXxCxUCalc,
                HsInvXmUtxXxCxUCalc,
                HsInvX2Calc, HsInvNormCalc, HsInvInvNormCalc, HsInvXNorm,
                HsInvResRead,
                HsInvStop
                );

signal HSINV_STATE : HSINV_STATE_TYPE;
signal UtxX, UtxXxC, UtxXxCxU, X2, Norm, InvNorm : std_logic_vector(31 downto 0);

signal RES    : std_logic_vector(31 downto 0);

constant FP_ONE : std_logic_vector(31 downto 0)
:= "00111111100000000000000000000000";
constant FP_TWO : std_logic_vector(31 downto 0)
:= "01000000000000000000000000000000";

constant cR_MEM_WR  : std_logic_vector(4 downto 0) := "00001";
constant cR_MEM_RD  : std_logic_vector(4 downto 0) := "00010";
constant cL_MEM_WR  : std_logic_vector(4 downto 0) := "00011";
constant cL_MEM_RD  : std_logic_vector(4 downto 0) := "00100";
constant cLR_MEM_WR : std_logic_vector(4 downto 0) := "00101";
constant cMEM_STOP  : std_logic_vector(4 downto 0) := "00110";
constant cRR_INPRO  : std_logic_vector(4 downto 0) := "00111";
constant cLL_INPRO  : std_logic_vector(4 downto 0) := "01000";
constant cLR_INPRO  : std_logic_vector(4 downto 0) := "01001";
constant cINPRO_F   : std_logic_vector(4 downto 0) := "01010";
constant cINPRO_R   : std_logic_vector(4 downto 0) := "01011";
constant cINPRO_L   : std_logic_vector(4 downto 0) := "01100";
constant cINPRO_LR  : std_logic_vector(4 downto 0) := "01101";
constant cFF_MUL    : std_logic_vector(4 downto 0) := "01110";
constant cFR_MUL    : std_logic_vector(4 downto 0) := "01111";
constant cFL_MUL    : std_logic_vector(4 downto 0) := "10000";
constant cRR_MUL    : std_logic_vector(4 downto 0) := "10001";
constant cLL_MUL    : std_logic_vector(4 downto 0) := "10010";
constant cLR_MUL    : std_logic_vector(4 downto 0) := "10011";
constant cFF_ADD    : std_logic_vector(4 downto 0) := "10100";
constant cFR_ADD    : std_logic_vector(4 downto 0) := "10101";
```

```
constant cFL_ADD    : std_logic_vector(4 downto 0) := "10110";
constant cRR_ADD    : std_logic_vector(4 downto 0) := "10111";
constant cLL_ADD    : std_logic_vector(4 downto 0) := "11000";
constant cLR_ADD    : std_logic_vector(4 downto 0) := "11001";
constant cCALC_F    : std_logic_vector(4 downto 0) := "11010";
constant cCALC_R    : std_logic_vector(4 downto 0) := "11011";
constant cCALC_L    : std_logic_vector(4 downto 0) := "11100";
constant cCALC_LR   : std_logic_vector(4 downto 0) := "11101";

begin

A <= "ZZZZZZZZZZZZZZZZ" when ACK_BUF = "00" else A_REG;

ACK_BUF <= OBF;
ACK <= ACK_BUF;

-----------<< Output Result Data to PC >>------------
process ( BL(0), BL(3), IBF ) begin
if BL(0) = '1' then
        STB <= "11";
        OUT_CNT <= '0';
        A_REG <= ( others => '0' );
elsif rising_edge( BL(3) ) then
        STB <= "00";
        if OUT_CNT = '0' then
                A_REG <= RES(15 downto 0);
                OUT_CNT <= '1';
        else
                A_REG <= RES(31 downto 16);
                OUT_CNT <= '0';
        end if;
end if;
if IBF = "11" then
        STB <= "11";
end if;
end process;

-----------<< Generate Memory Read Signal of Result Data >>------------
process ( BL(0), BL(4), OUT_ACK2 ) begin
if BL(0) = '1' or OUT_ACK2 = '1' then
        OUT_ACK <= '0';
elsif rising_edge( BL(4) ) then
        OUT_ACK <= '1';
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        OUT_ACK2 <= '0';
elsif rising_edge( CLK ) then
        if OUT_ACK = '1' then
                OUT_ACK2 <= '1';
        else
                OUT_ACK2 <= '0';
        end if;
end if;
end process;

-----<< Selecter of ALU Operation >>-----
DATA_BUS <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_BUF = '1' else DATA_BUS_BUF;
OE_ALU <= OE_BUF;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CTRL_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR =>
                        CTRL_BUS <= cR_MEM_WR;
                when R_MEM_RD =>
                        CTRL_BUS <= cR_MEM_RD;
                when L_MEM_WR =>
                        CTRL_BUS <= cL_MEM_WR;
```

```
                when L_MEM_RD =>
                        CTRL_BUS <= cL_MEM_RD;
                when LR_MEM_WR =>
                        CTRL_BUS <= cLR_MEM_WR;
                when MEM_STOP =>
                        CTRL_BUS <= cMEM_STOP;
                when RR_INPRO =>
                        CTRL_BUS <= cRR_INPRO;
                when LL_INPRO =>
                        CTRL_BUS <= cLL_INPRO;
                when LR_INPRO =>
                        CTRL_BUS <= cLR_INPRO;
                when INPRO_F =>
                        CTRL_BUS <= cINPRO_F;
                when INPRO_R =>
                        CTRL_BUS <= cINPRO_R;
                when INPRO_L =>
                        CTRL_BUS <= cINPRO_L;
                when INPRO_LR =>
                        CTRL_BUS <= cINPRO_LR;
                when FF_MUL =>
                        CTRL_BUS <= cFF_MUL;
                when FR_MUL =>
                        CTRL_BUS <= cFR_MUL;
                when FL_MUL =>
                        CTRL_BUS <= cFL_MUL;
                when RR_MUL =>
                        CTRL_BUS <= cRR_MUL;
                when LL_MUL =>
                        CTRL_BUS <= cLL_MUL;
                when LR_MUL =>
                        CTRL_BUS <= cLR_MUL;
                when FF_ADD =>
                        CTRL_BUS <= cFF_ADD;
                when FR_ADD =>
                        CTRL_BUS <= cFR_ADD;
                when FL_ADD =>
                        CTRL_BUS <= cFL_ADD;
                when RR_ADD =>
                        CTRL_BUS <= cRR_ADD;
                when LL_ADD =>
                        CTRL_BUS <= cLL_ADD;
                when LR_ADD =>
                        CTRL_BUS <= cLR_ADD;
                when CALC_F =>
                        CTRL_BUS <= cCALC_F;
                when CALC_R =>
                        CTRL_BUS <= cCALC_R;
                when CALC_L =>
                        CTRL_BUS <= cCALC_L;
                when CALC_LR =>
                        CTRL_BUS <= cCALC_LR;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DATA_BUS_BUF <= ( others => '0' );
        OE_BUF <= '0';
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when R_MEM_WR | L_MEM_WR | LR_MEM_WR | FR_MUL | FL_MUL |
 FR_ADD | FL_ADD =>
                           DATA_BUS_BUF <= DATA_BUF1;
                        OE_BUF <= '0';
                when MEM_STOP | INPRO_F | CALC_F =>
                        DATA_BUS_BUF <= ( others => '0' );
                        OE_BUF <= '1';
                when LR_INPRO | INPRO_LR | LR_MUL | LR_ADD | CALC_LR =>
                        DATA_BUS_BUF(16 downto 0) <= ADRS_BUF2;
                        OE_BUF <= '0';
                when FF_MUL | FF_ADD =>
                        OE_BUF <= '0';
                        if CALC_STATE = FIRST_DATA then
```

```
                                DATA_BUS_BUF <= DATA_BUF1;
                        else
                                DATA_BUS_BUF <= DATA_BUF2;
                        end if;
                when others => null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        CALC_STATE <= FIRST_DATA;
elsif rising_edge( CLK ) then
        case ALU_STATE is
                when FF_MUL | FF_ADD =>
                         CALC_STATE <= SECOND_DATA;
                when others =>
                        if CALC_DONE = '1' then
                                CALC_STATE <= FIRST_DATA;
                        end if;
        end case;
end if;
end process;


process ( BL(0), CLK ) begin
if BL(0) = '1' then
        ADRS_BUS <= ( others => '0' );
elsif falling_edge( CLK ) then
        case ALU_STATE is
                when MEM_STOP | INPRO_F | FF_MUL | FF_ADD | CALC_F =>
                        ADRS_BUS <= ( others => '0' );
                when others =>
                        ADRS_BUS <= ADRS_BUF1;
        end case;
end if;
end process;

------------<< Householder Inverse Transform >>-------------
process( BL(0), CLK )
        variable i, j, k : std_logic_vector(7 downto 0);
begin
if BL(0) = '1' then
        HSINV_STATE <= HsInvStop;
        ALU_STATE <= aStop;
        ALU_ACK <= '0';
        DIV_ACK <= '0';
        DATA_BUF1 <= (others => '0');
        DATA_BUF2 <= (others => '0');
        ADRS_BUF1 <= (others => '0');
        ADRS_BUF2 <= (others => '0');
--        DIV_A <= (others => '0');
--        DIV_B <= (others => '0');
--        Lambda  <= (others => '0');
--        Vice <= (others => '0');
--        Alpha1 <= (others => '0');
--        Alpha2 <= (others => '0');
--        Beta1 <= (others => '0');
--        Beta2 <= (others => '0');
--        Beta3 <= (others => '0');
--        M <= (others => '0');
--        MxBeta2 <= (others => '0');
--        X <= (others => '0');
--        RxX <= (others => '0');
--        XmRxX <= (others => '0');
--        Alpha <= (others => '0');
--        X1 <= (others => '0');
--        X2 <= (others => '0');
--        MxX1 <= (others => '0');
--        Res <= (others => '0');
--        Ex <= (others => '0');
--        N <= (others => '0');
        i := ( others => '0' );
        j := ( others => '0' );
```

```
        k := ( 0 => '1', others => '0' );
        BH <= ( others => '0' );
elsif rising_edge( CLK ) then
        case HSINV_STATE is
                when HsInvStop =>
                        if BL(5) = '1' then
                                HSINV_STATE <= HsInvDimRead;
                        else
                                ALU_STATE <= aSTOP;
                                HSINV_STATE <= HsInvStop;
                        end if;

                when HsInvDimRead =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= L_MEM_RD;
                                ADRS_BUF1 <= "10000000000000000";
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                N <= DATA_BUS(7 downto 0);
                                ALU_ACK <= '0';
                                HSINV_STATE <= HsInvVarIni;
                        end if;

                when HsInvVarIni =>
                        j := N - "00000010";
                        i := j;
                        HSINV_STATE <= HsInvUtxXCalc;

                when HsInvUtxXCalc =>
                        if i < N then
                                ALU_STATE <= LR_INPRO;
                                i := i + '1';
                                ADRS_BUF1 <= '1' & i & k;
                                ADRS_BUF2 <= '0' & i & j;
                        else
                                ALU_STATE <= INPRO_F;
                        end if;

                        if CALC_DONE = '1' then
                                UtxX <= DATA_BUS;
                                i := j + '1';
                                HSINV_STATE <= HsInvUtxXxCCalc;
                        end if;

                when HsInvUtxXxCCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FL_MUL;
                                DATA_BUF1 <= UtxX;
                                ADRS_BUF1 <= "100000000" & j;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_F;
                        end if;

                        if CALC_DONE = '1' then
                                UtxXxC <= DATA_BUS;
                                ALU_ACK <= '0';
                                HSINV_STATE <= HsInvUtxXxCxUCalc;
                        end if;

                when HsInvUtxXxCxUCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FL_MUL;
                                DATA_BUF1 <= UtxXxC;
                                ADRS_BUF1 <= '0' & i & j;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_F;
                        end if;

                        if CALC_DONE = '1' then
                                UtxXxCxU <= DATA_BUS;
```

```
                                ALU_ACK <= '0';
                                HSINV_STATE <= HsInvXmUtxXxCxUCalc;
                        end if;

                when HsInvXmUtxXxCxUCalc =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FR_ADD;
                                DATA_BUF1
<= ( not UtxXxCxU(31) ) & UtxXxCxU(30 downto 0);
                                ADRS_BUF1 <= '1' & i & k;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= CALC_R;
                                ADRS_BUF1 <= '1' & i & k;
                        end if;

                        if CALC_DONE = '1' then
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                        HSINV_STATE <= HsInvUtxXxCxUCalc;
                                elsif j > "00000001" then
                                        j := j - '1';
                                        i := j;
                                        HSINV_STATE <= HsInvUtxXCalc;
                                elsif k < N then
                                        k := k + '1';
                                        HSINV_STATE <= HsInvVarIni;
                                else
                                        i := ( others => '0' );
                                        j := ( 0 => '1', others => '0' );
                                        HSINV_STATE <= HsInvX2Calc;
                                end if;
                        end if;

                when HsInvX2Calc =>
                        if i < N then
                                ALU_STATE <= RR_INPRO;
                                i := i + '1';
                                ADRS_BUF1 <= '1' & i & j;
                        else
                                ALU_STATE <= INPRO_F;
                        end if;

                        if CALC_DONE = '1' then
                                X2 <= DATA_BUS;
                                i := (0 => '1', others => '0');
                                HSINV_STATE <= HsInvNormCalc;
                        end if;

                when HsInvNormCalc =>
                        Norm <= sqrt(X2);
                        HSINV_STATE <= HsInvInvNormCalc;

                when HsInvInvNormCalc =>
                        if DIV_ACK = '0' then
                                DIV_A <= FP_ONE;
                                DIV_B <= Norm;
                                DIV_ACK <= '1';
                        else
                                DIV_A <= ( others => '0' );
                                DIV_B <= ( others => '0' );
                        end if;

                        if DIV_DONE = '1' then
                                InvNorm <= DIV_Q;
                                DIV_ACK <= '0';
                                HSINV_STATE <= HsInvXNorm;
                        end if;

                when HsInvXNorm =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= FR_MUL;
                                DATA_BUF1 <= InvNorm;
                                ADRS_BUF1 <= '1' & i & j;
                                ALU_ACK <= '1';
```

```
                        else
                                ALU_STATE <= CALC_R;
                                ADRS_BUF1 <= '1' & i & j;
                        end if;

                        if CALC_DONE = '1' then
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                elsif j < N then
                                        j := j + '1';
                                        i := (others => '0');
                                        HSINV_STATE <= HsInvX2Calc;
                                else
                                        i := ( 0 => '1', others => '0' );
                                        j := ( 0 => '1', others => '0' );
                                        HSINV_STATE <= HsInvResRead;
                                end if;
                        end if;

                when HsInvResRead =>
                        if ALU_ACK = '0' then
                                ALU_STATE <= R_MEM_RD;
                                ADRS_BUF1 <= '1' & i & j;
                                ALU_ACK <= '1';
                        else
                                ALU_STATE <= MEM_STOP;
                        end if;

                        if CALC_DONE = '1' then
                                BH <= ( 0 => '1', others => '0' );
                                RES <= DATA_BUS;
                        end if;

                        if OUT_ACK2 = '1' then
                                BH <= (others => '0');
                                ALU_ACK <= '0';
                                if i < N then
                                        i := i + '1';
                                elsif j < N then
                                        i := ( 0 => '1', others => '0' );
                                        j := j + '1';
                                else
                                        i := ( 0 => '1', others => '0' );
                                        j := ( 0 => '1', others => '0' );
                                end if;
                        end if;

                when others =>
                        null;
        end case;
end if;
end process;

process ( BL(0), CLK ) begin
if BL(0) = '1' then
        DCNT <= '0';
--        DCNT3 <= "0000";
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then
--if DIV_ACK = '1' and DIV_DONE = '0' then
--        DCNT3 <= DCNT3 + '1';
--else
--        DCNT3 <= "0000";
--end if;
--
--if DCNT3 = "1111" then
--        DIV_DONE <= '1';
--else
--        DIV_DONE <= '0';
--end if;

        if DIV_ACK = '1' and DCNT = '0' and DIV_DONE = '0' then
                DCNT <= '1';
        else
```

```
                DCNT <= '0';
        end if;
        if DCNT = '1' then
                DIV_DONE <= '1';
        else
                DIV_DONE <= '0';
        end if;
end if;
end process;

------------<< Floating Point Number Divider >>--------------
divider : fpdiv  port map ( CLK => CLK, FA => DIV_A, FB => DIV_B, Q => DIV_Q );

end RTL;
```

I.4　ハウスホルダー法実行プログラム

ハウスホルダー法の 4 つのアルゴリズムを順に FPGA へ自動実装し、計算結果を評価ボードからパソコンに受け取るプログラムを C 言語により作成した。

実行にあたって必要となるものは、計算対象となる行列が記述されたファイルと FPGA へのダウンロードプログラム [3] (付録 II.4参照)、そして積和器とハウスホルダー法の 4 つのアルゴリズム (ハウスホルダー変換、二分法、逆反復法、ハウスホルダー逆変換) の FPGA への配置配線ファイル (TTF 形式) である。行列が記述されたファイルはテキスト形式で、数値をスペースまたはタブで区切られた行列形式として与える。

また、計算結果は指定したファイルへ格納することができる。ソースを以下に示す。

```
//Calculation Program for
//Eigen Value and Eigen Vector of Matrix using Householder Method
//   1999/03/15 (Mon)
//   yamaoka@tube.ee.uec.ac.jp

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include <BASE8255.h>

#define LINE 255
#define NAME 25

int main()
{
int dim = 0, i = 0, j = 0, k = 0;
unsigned int *ddata, res[2];
char buf1[LINE], buf2[LINE], fname1[NAME], fname2[NAME], bh;
float buf3;

FILE *fp_data,*fp_res;

printf("Calculation Program for \n");
printf("Eigen Value and Eigen Vector using Householder Method \n");
printf("Input Data File Name >> \n");
scanf("%s",fname1);
printf("Input Result File Name >> \n");
scanf("%s",fname2);
```

```
fp_data = fopen(fname1,"r");   // data file

if(fp_data == NULL ){
        printf("can't open data file %s",fname1);
        exit(1);
}

fp_res = fopen(fname2,"w");    // result file

printf("Download of alu.ttf\n");
system("flex10k2 alu.ttf");

// Householder Transform
printf("\nDownload of hshld10.ttf\n");
system("flex10k1 hshld10.ttf");

//A:handshake  BL:out, C:out
//Control Word = C0 (mode2)
//A:handshake  BH:in, C:out
//Control Word = C2 (mode2)

outp(P_CTRL_L, 0xC0);
outp(P_CTRL_H, 0xC2);

outp(P_BL, 0x01);    // Reset
delay(200);
outp(P_BL, 0x00);

// send data from PC to FPGA
fgets(buf1,LINE,fp_data);
while(1){
        sprintf(buf2,"%s",strtok(buf1," \t\n"));
        sscanf(buf2,"%f",&buf3);
        while(strcmp(buf2,"(null)") != 0){
                ddata = (unsigned int *)&buf3;
                outpw(P_A,*ddata);                // output low 16 bit data
                outpw(P_A,*(ddata + 1));                  // output low 16 bit data
                sprintf(buf2,"%s",strtok(NULL," \t\n"));
                sscanf(buf2,"%f",&buf3);
        }
        fgets(buf1,LINE,fp_data);
//        k++;

//        if(k == 1){
        dim++;
//        }

        if(feof( fp_data ) != 0) break;

//        if(k == 1){
//        k = 0;
        outp(P_BL, 0x02);  // add dimension counter
        outp(P_BL, 0x00);
        //printf("\n");
//        }
}

outp(P_BL, 0x04);  // end of sending data
outp(P_BL, 0x00);

printf("\n");
fclose(fp_data);

//printf("\n\nEnd File Reading\n\n");

// wait for end of calculation
while(1){
bh = inportb(P_BH);
if( (bh & 0x1) == 0x1) break;
}

// Bisection Method
```

```
printf("Download of bisec.ttf\n");
system("flex10k1 bisec.ttf");

outp(P_CTRL_L, 0xC0); // control word
outp(P_CTRL_H, 0xC2);

outp(P_BL, 0x01);    // Reset
delay(200);
outp(P_BL, 0x00);

outp(P_BL, 0x20);    // start of calculation
outp(P_BL, 0x00);

// Read Eigen Values
printf("\nEigen Values >>\n");
fprintf(fp_res,"Eigen Values >>\n");
for(i = 0 ; i < dim ; i++){

        // wait for end of calculation
        while(1){
                bh = inportb(P_BH);
                if( (bh & 0x1) == 0x1) break;
        }

        outp(P_BL, 0x08);     // rising_edge( BL(3) )
        outp(P_BL, 0x00);
        res[0] = inpw(P_A);   // fetch of lower 16bit data
        outp(P_BL, 0x08);
        outp(P_BL, 0x00);     // rising_edge( BL(3) )
        res[1] = inpw(P_A);   // fetch of upper 16bit data
        printf("%f ",*(float *)res);
        fprintf(fp_res,"%9.7f ",*(float *)res);
        outp(P_BL, 0x10);     // rising_edge( BL(4) )
        outp(P_BL, 0x00);

}
printf("\n");
fprintf(fp_res,"\n");

// Inverse Iteration Method
printf("\nDownload of invit.ttf\n");
system("flex10k1 invit.ttf");

outp(P_CTRL_L, 0xC0); // control word
outp(P_CTRL_H, 0xC2);

outp(P_BL, 0x01);    // Reset
delay(200);
outp(P_BL, 0x00);

outp(P_BL, 0x20);    // start of calculation
outp(P_BL, 0x00);

// wait for end of calculation
while(1){
        bh = inportb(P_BH);
        if( (bh & 0x1) == 0x1) break;
}

printf("\nDownload of hsinv.ttf\n");
system("flex10k1 hsinv.ttf");

outp(P_CTRL_L, 0xC0); // control word
outp(P_CTRL_H, 0xC2);

outp(P_BL, 0x01);    // Reset
delay(200);
outp(P_BL, 0x00);

outp(P_BL, 0x20);    // start of calculation
```

```
outp(P_BL, 0x00);

// read eigen vectors
printf("\nEigen Vectors >>\n");
fprintf(fp_res,"Eigen Vectors >>\n");
for(i = 0 ; i < dim ; i++){
        for(j = 0 ; j < dim ; j++){

                // wait for end of calculation
                while(1){
                        bh = inportb(P_BH);
                        if( (bh & 0x1) == 0x1) break;
                }

                outp(P_BL, 0x08);     // rising_edge( BL(3) )
                outp(P_BL, 0x00);
                res[0] = inpw(P_A);   // fetch of lower 16bit data
                outp(P_BL, 0x08);
                outp(P_BL, 0x00);     // rising_edge( BL(3) )
                res[1] = inpw(P_A);   // fetch of upper 16bit data
                printf("%f ",*(float *)res);
                fprintf(fp_res,"%9.7f ",*(float *)res);
                outp(P_BL, 0x10);     // rising_edge( BL(4) )
                outp(P_BL, 0x00);
        }
printf("\n");
fprintf(fp_res,"\n");
}
return 0;
}
```

付録 II

計算システム評価ボード

評価ボードは 1997 年度において松尾竜馬氏 [3] により製作されたものである。以下に評価ボードの詳細を説明する。

II.1　パソコン・評価ボード間のインターフェース

本研究では ALTERA 社により提供される FPGA である FLEX10K シリーズを使用する。これはプログラム素子が SRAM により構成されており、電源を入れた後に必ずコンフィグレーションをする必要がある。また、FPGA 上に実現されたハードウェアに対してパーソナルコンピュータから制御・通信する必要がある。これらを実現するため、パーソナルコンピュータと評価ボード間のインターフェースを用いる。

要求仕様

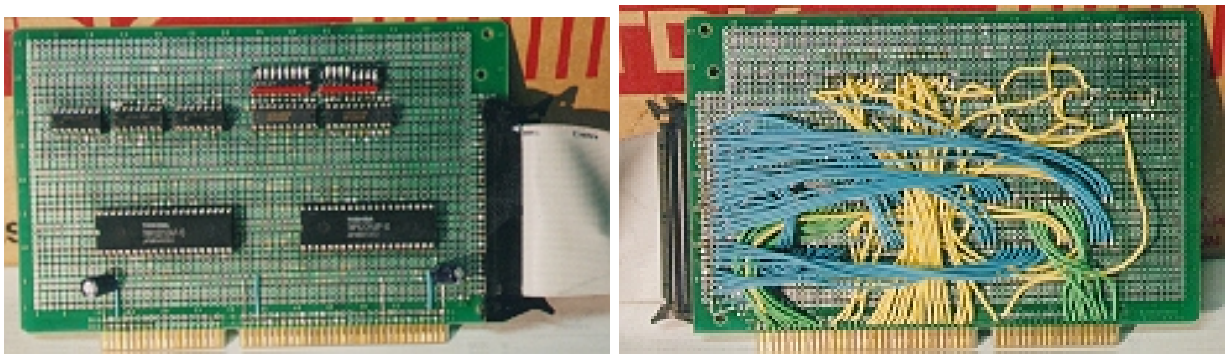
インターフェースは、FPGA のコンフィグレーション、制御・通信用である。コンフィグレーションでは最低入力 2bit、出力 3bit を必要とする。また通信には入出力 16bit は必要であり、入力専用、出力専用の信号もあると良い。入出力の単位は 8bit になるので、コンフィグレーション用に入力 8bit、出力 8bit、制御・通信用に入出力 16bit、入力 8bit、出力 8bit として計 48bit の信号が必要である。また、PC 側において、プログラム言語からの利用が容易である必要がある。

インターフェースの仕様

本設計ではPPI8255(Peripheral Parallel Interface 8255)使用し、また、DOS/Vパソコンで利用できるようにISA BUS用カードとして仕様を満たすよう設計してある。PPI8255はデータ幅が8bitであるが2個使用することにより16bit幅を実現する。また、8bit I/Oとしても使用することができる。PC側からは、プログラム言語によりI/O命令を用いれば制御することができる。インターフェースカードの仕様を表II.1に、外観を図II.1に示す。

表 II.1　インターフェースカードの仕様

名称	ISA BUS 16bit I/O Card
使用LSI	PPI 8255 10MHz版 2個
バス	ISA BUS 16bit
外部端子	Aポート Bポート Cポート それぞれ16bitと VCC GND ※それぞれのポートは入出力を上位8bit 下位8bit 独立に設定可能
コネクタ形状	50pinヘッダ
入出力レベル	TTL コンパチブル
占有アドレス	先頭アドレスから8byte ※先頭アドレスは0000-FFF8の範囲を8byte単位で設定可能



(a) 部品面　　(b) ハンダ面

図 II.1　インターフェースカードの外観 [1]

[1]ファイル名：(a):./fig2/isaif1.eps, (b):./fig2/isaif2.eps

インターフェースカードの回路構成

インターフェースカード

インターフェースカード全体の回路図を図 II.2に示す。

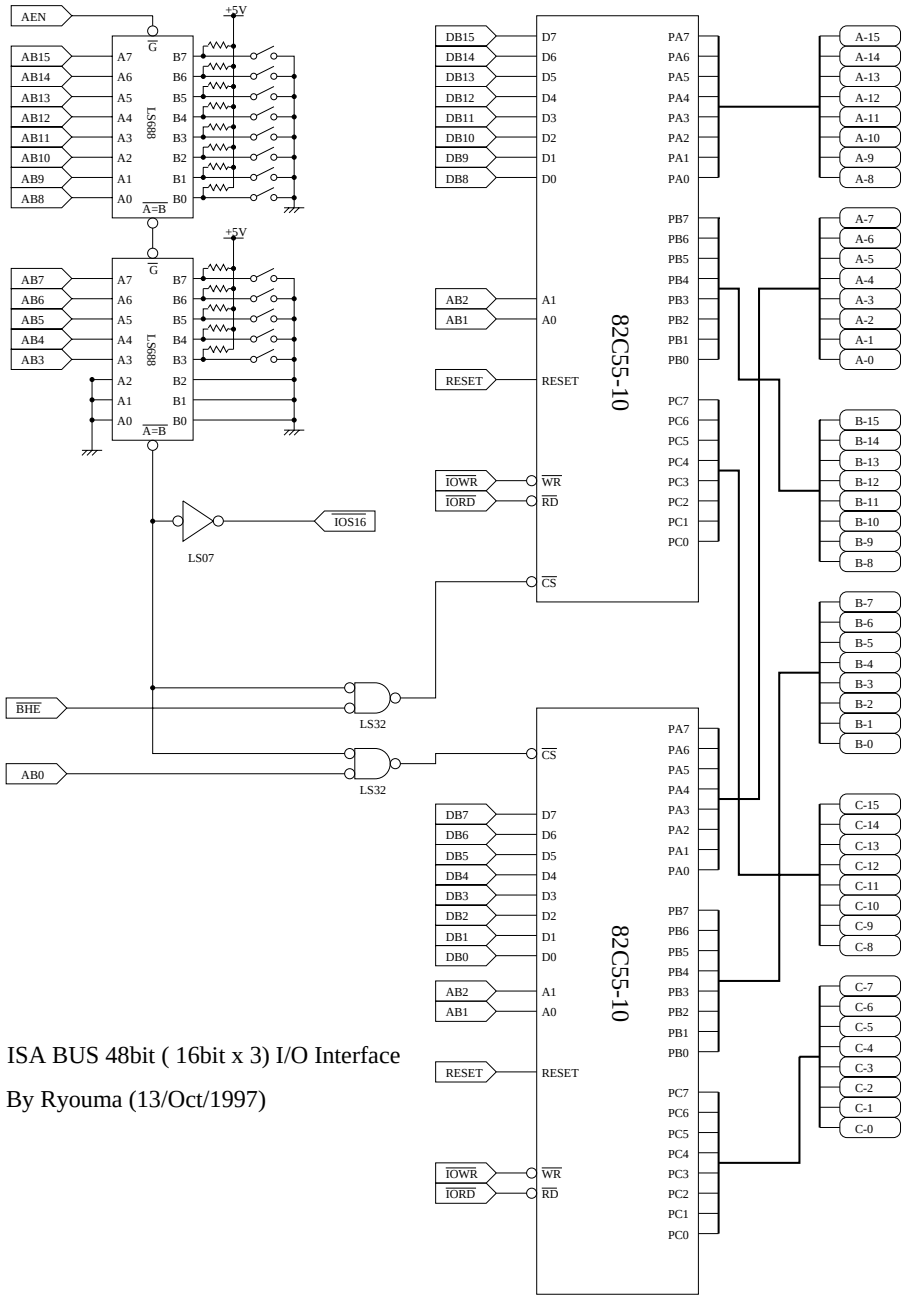


図 II.2　インターフェースカードの回路図 [2]

[2]ファイル名：./fig2/16bit-io.eps

アドレスデコード部

図 II.2におけるアドレスデコード部の回路図を図 II.3に、信号線を表 II.2に示す。

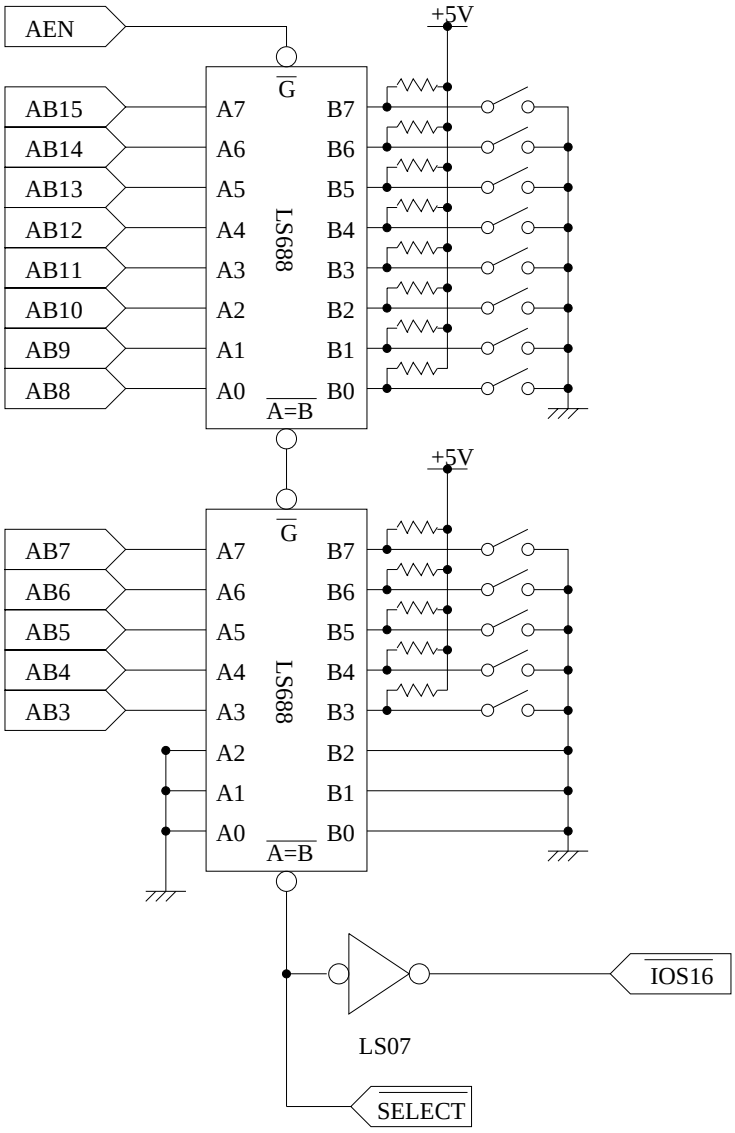


図 II.3　アドレスデコード部の回路図 [3]

[3]ファイル名：./fig2/AddDecode.eps

表 II.2　アドレスデコード部の信号線

信号名	意味	説明
/AEN	Address ENable	アドレス信号が有効なことを示す。
ABxx	Address Bus	アドレス信号
/IOS16	I/O Strobe 16bit	選択されたハードウェアが 16bit であることを知らせる信号
/SELECT	card SELECT	インターフェースが選択されたことを示す Card 内の信号

ISA BUS の信号 AB3 ～ AB15 と AEN により、 Card 上のハードウェアが選択される。上位 bit も全てデコードする。また、カードが 16bit スレーブである事を ISA BUS 側に知らせるために、 /SELECT 信号をオープンコレクタ出力で /IOS16 に入力する。

先頭アドレスを決定するには、 DIP スイッチを用いる。スイッチを ON にすると 0、OFF にすると 1 となる。全て OFF とすると、 FFF8 を指定したことになる。当然空いている所を探して使う。

チップセレクト信号生成部

図 II.2におけるチップセレクト信号生成部の回路図を図 II.4に、信号線を表 II.3に示す。

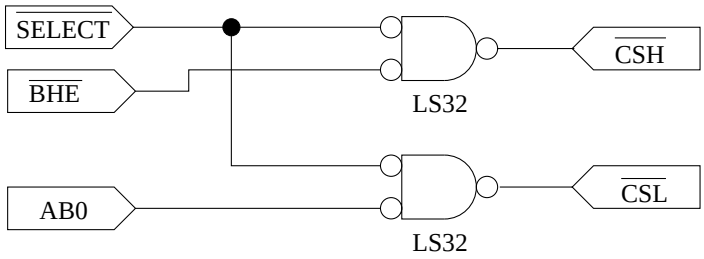


図 II.4　チップセレクト信号生成部の回路図 [4]

[4]ファイル名：./fig2/control.eps

表 II.3　チップセレクト信号生成部の信号線

信号名	意味	説明
/BHE	Byte Half Enable	データが 16bit 分有効かどうかを知らせる信号
AB0	Address Bus 0	/BHE と AB0 で CSH / CSL の 信号を生成する
/SELECT	card SELECT	インターフェースが選択されたことを示す信号
/CSH	Chip Select High byte	上位バイト用 Chip 選択信号
/CSL	Chip Sleect Low byte	下位バイト用 Chip 選択信号

8bit アクセス、16bit アクセスの振り分けを /BHE を用いて行う。16bit アクセスの場合、/CSH /CSL 両方 Enable にする。8bit アクセスの時は、必要な方だけ Enable にする。

PPI8255 周辺

図 II.2における PPI8255 周辺の回路図を図 II.5に、信号線を表 II.4に示す。

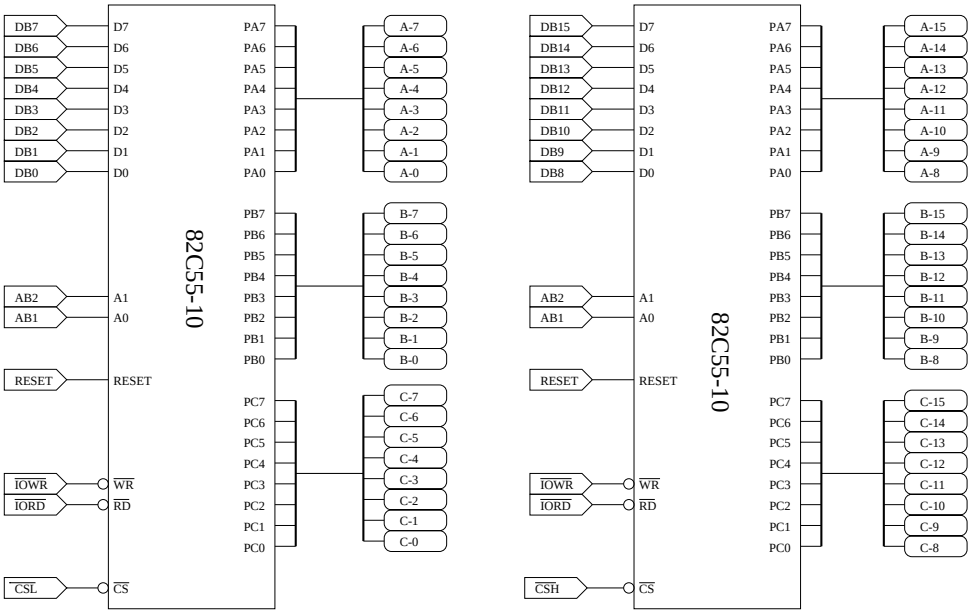


図 II.5　PPI 8255 周辺の回路図 [5]

[5]ファイル名：./fig2/8255.eps

表 II.4　PPI 8255 周辺の信号線

信号名	意味	説明
DB15-8	Data Bus 15-8	データバス 上位 8bit
DB7-0	Data Bus 7-0	データバス 下位 8bit
AB2-1	Address Bus 2-1	LSI 内レジスタの選択をする信号
/IORD	I/O ReaD	I/O リード信号
/IOWR	I/O WRite	I/O ライト信号
RESET	RESET	リセット信号
Axx Bxx Cxx	A B C port	LSI からのパラレル入出力
/CSH	Chip Select High byte	上位バイト用 Chip 選択信号
/CSL	Chip Select Low byte	下位バイト用 Chip 選択信号

外部端子

インターフェースの外部端子には 50pin ヘッダーを用いる。 pin の割り付けは図 II.6を参照。フラットケーブルを用いて他の機器と接続する。

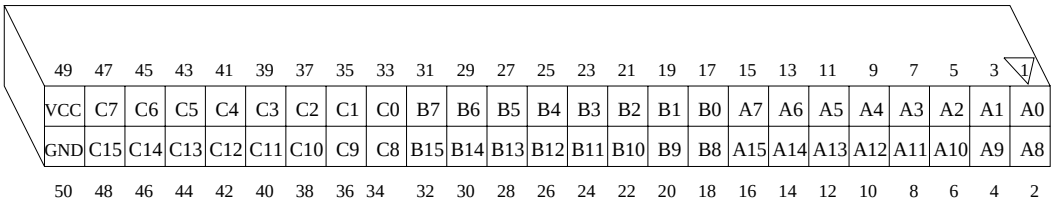


図 II.6　外部端子 pin 割り付け [6]

信号線は TTL レベルであり、プルアップはされていない。必要ならばプルアップする。ちなみに FLEX10K や FLEX8000 は TTL 入力が可能であるのでプルアップは必要無い。 VCC GND は ISA BUS から供給されている。線が細いことを考慮するとあまり容量は取れないので、外部に電源が必要な場合がある。

[6] ファイル名：./fig2/50head.eps

II.2　FPGA 搭載計算システム評価ボード

HDL により設計したプロセッサの機能が実際にハードウェアとして動作するかを検証するため、FPGA を 2 基搭載した評価ボードを用いて実装検証を行う。

評価ボードの仕様

評価ボードの外観を図 II.7に、ブロック図を図 II.8に、仕様を表 II.5に示す。

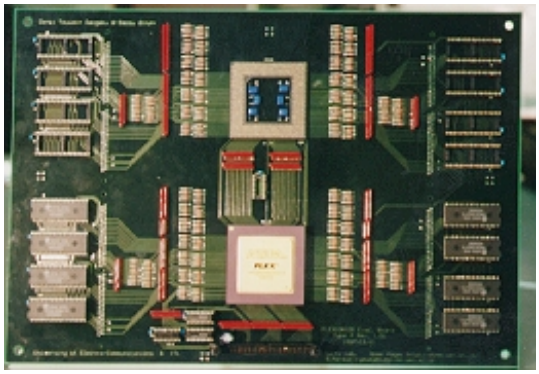


図 II.7　計算システム評価ボード (横 38.5cm ×縦 26.5cm) [7]

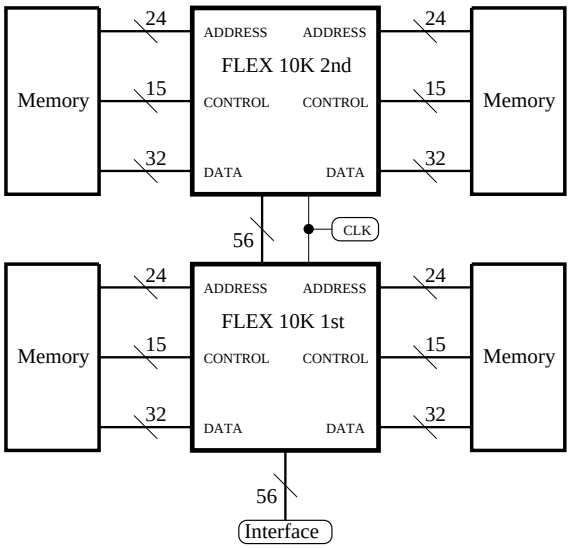


図 II.8　評価ボードのブロック図 [8]

[7]ファイル名：./fig2/eval1.eps

[8]ファイル名：./fig2/ev-block.eps

表 II.5 評価ボードの仕様

名称	FLEX10K100 評価ボード
FPGA	FLEX10K100GC503 x 2
SRAM	1M bit SRAM x 16
DRAM	72PIN SIMM x 4
CLOCK	クロックオシレータより供給可
内部バス	FPGA 間で 56bit 、各 FPGA に 32bit メモリバス が 2 系統
外部端子	A ポート B ポート C ポート それぞれ 16bit と VCC GND
コネクタ形状	50pin ヘッダ

本設計では米 ALTERA 社の開発した SRAM 型 FPGA である FLEX10K シリーズの 1 つである EPF10K100 を使用する。ゲート容量は 10 万ゲート (公称値) であり、本研究で想定している浮動小数点数演算器を単精度で単体であれば十分実装できる回路規模である。ボードには FPGA 2 基を中央の上下に配置し、その左右両側にはそれぞれ 1M ビット SRAM が 4 個づつ合計 16 個、72 ピン SIMM DRAM が 1 個づつ合計 4 個配置配線されており、計算においてはこの 2 種類のメモリを主記憶装置として用いる。 FPGA 同士の通信には FPGA 間に接続されている 56bit のバスを用いる。また、それぞれの FPGA にはクロックオシレータにより共通のクロックが供給されており、FPGA 同士で同期設計をすることが可能である。ボード外部へのインターフェース部分は PPI8255 を通して PC と接続されており、これを通して PC と FPGA 間の通信を行う。

コンフィグレーション

評価ボード上の FPGA を独立にコンフィグレーションできるようにする。そのためにはコンフィグレーション用に 5bit の信号の他に選択用の信号が必要である。選択用の信号は、電源投入時には回路の状態が High か Low になっているので、誤作動しないように 3bit の信号を用いて、それぞれの FPGA をコンフィグレーションできるようにする。

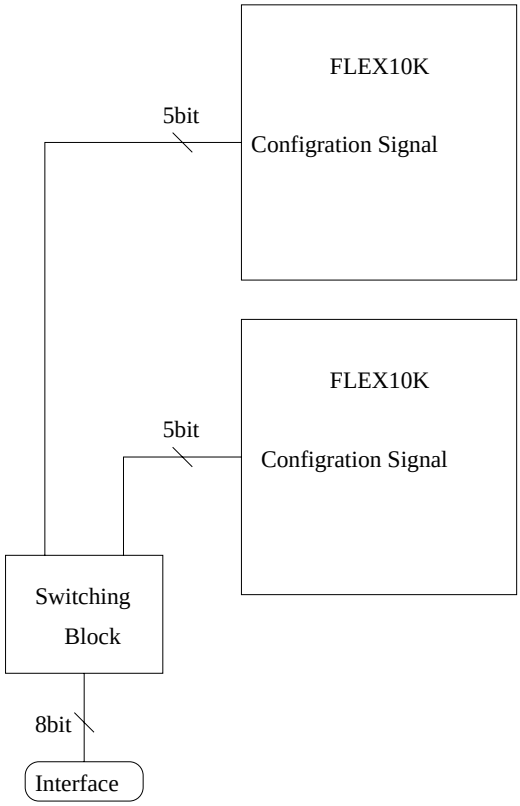


図 II.9　コンフィグレーション回路のブロック図 [9]

評価ボードの回路構成

コンフィグレーション部

コンフィグレーション部の回路図を図 II.10に、信号線を表 II.6に示す。

コンフィグレーションにはインターフェースの信号 C0 C1 C2 C8 C9 C10 B14 B15 を用いる。 FPGA の選択には C8 C9 C10 を用いる。 LS365 を用いて、信号の切替えを行っている。 FPGA が選択されないと、 LS365 は出力をハイインピーダンスにして、回路から切り離す事が出来る。注意するべき点は、 FPGA の通信用信号と共有しているので、 FPGA の回路設計時に、 C0 C1 C2 C8 C9 C10 B14 B15 を入力もしくはハイインピーダンスにしなければならない。そうしないと、再コンフィグレーションができなくなる。

[9] ファイル名：./fig2/fe0.eps

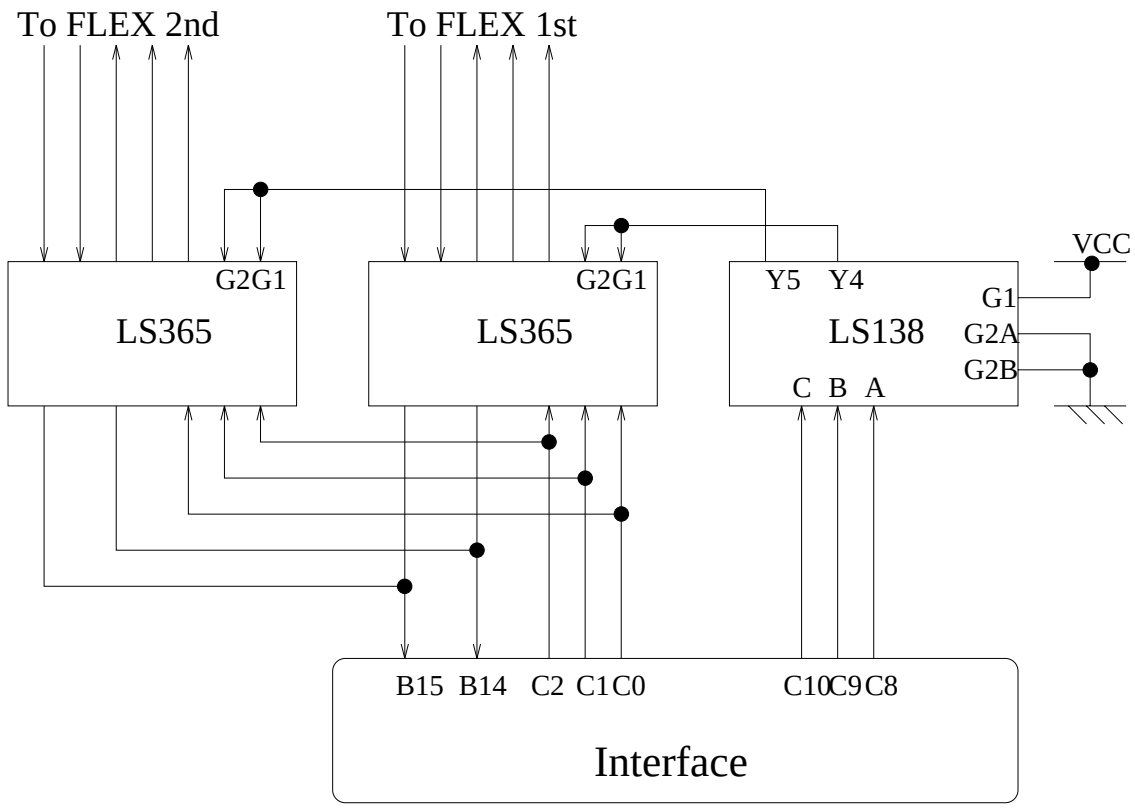


図 II.10　コンフィグレーション部の回路図 [10]

表 II.6　コンフィグレーション部の信号線

C0	DATA0
C1	CLK 選択
C2	nCONFIG
C8	FLEX 選択ピン
C9	C10:C9:C8 = 1:0:0 → FLEX 1st
C10	C10:C9:C8 = 1:0:1 → FLEX 2nd
B14	nSTATUS
B15	CONF_DONE

[10] ファイル名：./fig2/configblock.eps

パソコン・評価ボード間のインターフェース部

パソコン・評価ボード間のインターフェース部の回路図を図 II.11に示す。

全ての信号は $10k\Omega$ 程度の抵抗でプルアップする。通信には主に A ポートと B ポートを用いる。

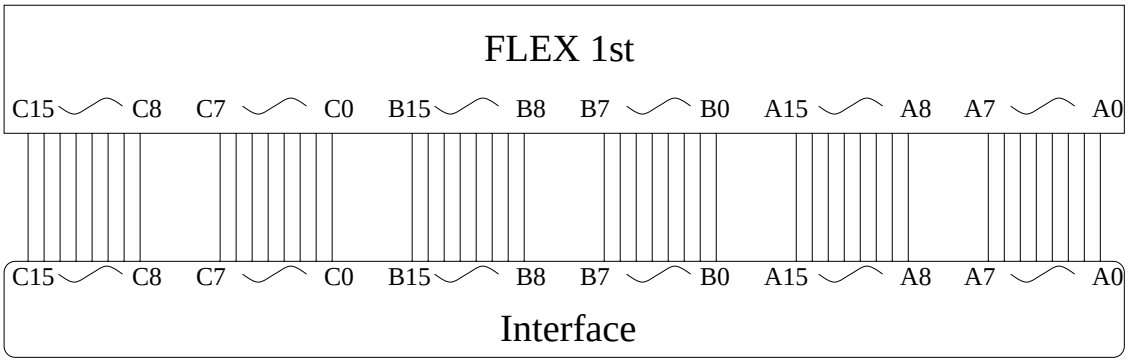


図 II.11　パソコン・評価ボード間のインターフェース部の回路図 [11]

FPGA 間のインターフェース部

FPGA 間のインターフェース部の回路図を図 II.12に示す。

FPGA の端子なので、自由に設計することができる。全ての信号は $10k\Omega$ 程度の抵抗でプルアップする。

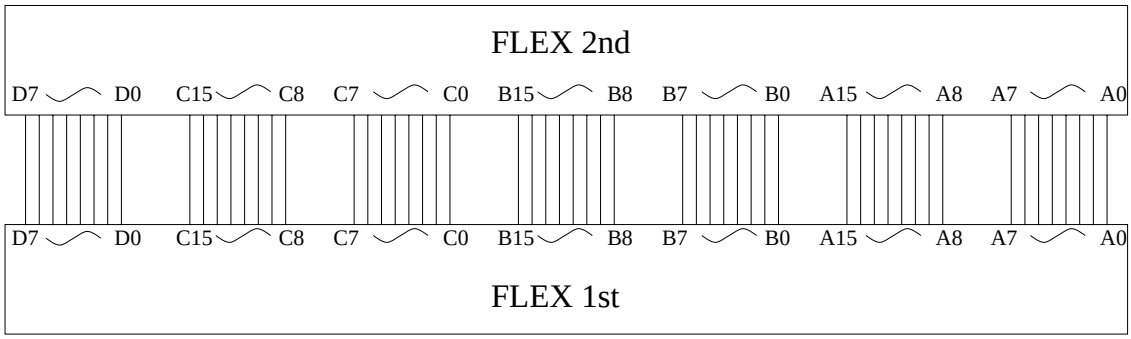


図 II.12　FPGA 間のインターフェース部の回路図 [12]

メモリ部

メモリ部の回路図を図 II.13に示す。

[11] ファイル名：./fig2/comm.eps

[12] ファイル名：./fig2/flexcomm.eps

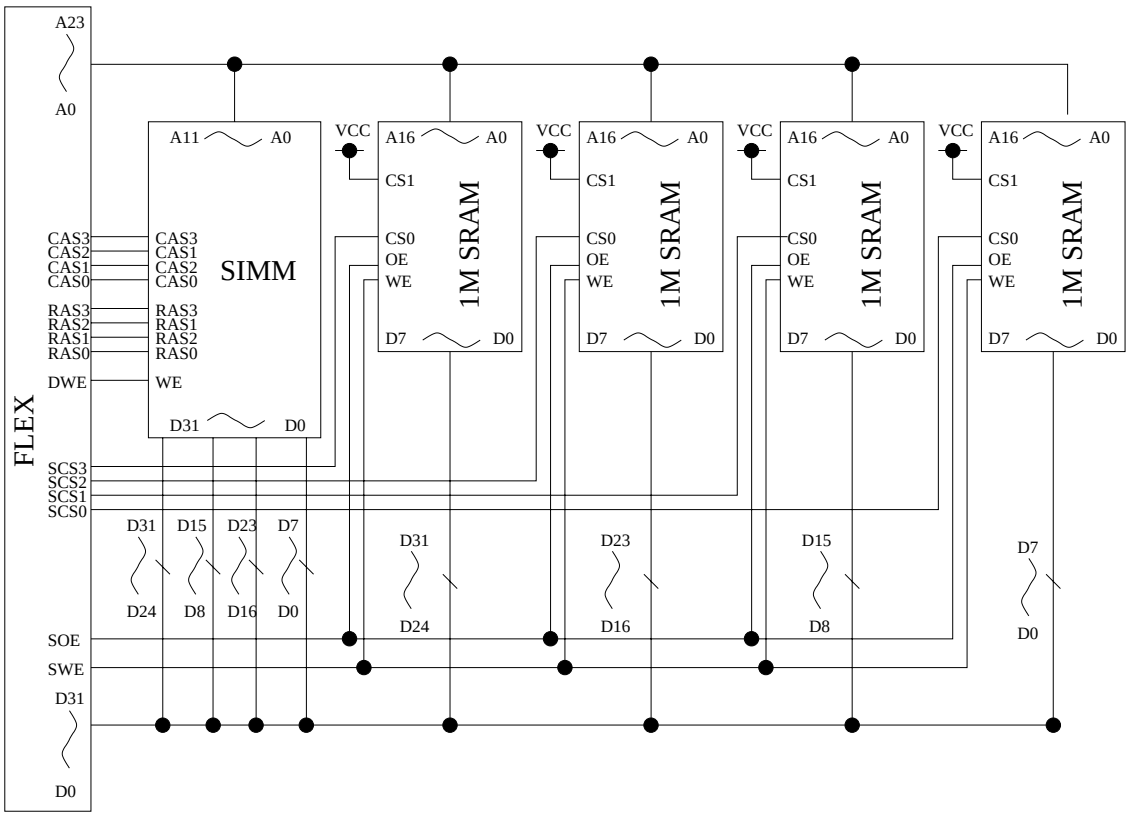


図 II.13　メモリ部の回路図 [13]

図には書いていないが、全ての信号線にダンピング抵抗、プルアップ抵抗を挿入する。FPGA 1 つにつき 2 系統のメモリ (L R) を設ける。メモリとして、72PIN SIMM DRAM と、1Mbit SRAM を混在させる。DRAM のアドレス線は A11 ～ A0 を用いる。DRAM のデータ線は、アートワークが楽になるように、D23 ～ D16 と D15 ～ D8 を入れ換えている。SRAM のアドレス線は A19 ～ A0 を用いる。

クロックオシレータ部

クロックオシレータ部の回路図を図 II.14に示す。

クロックオシレータの出力はダンピング抵抗を介して FLEX 1st と FLEX 2nd に供給する。ダンピンク抵抗は 22 ～ 100Ω 位である。使用するオシレータは 8pin DIP or 14pin DIP 型の物が使える。

[13] ファイル名：./fig2/memory.eps

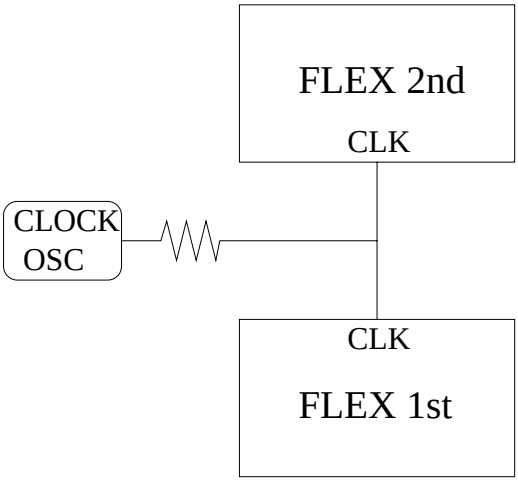


図 II.14 クロックオシレータ部の回路図 [14]

[14]ファイル名：./fig2/clock.eps

FLEX の PIN の割り付け

FLEX10K の PIN の割り付けを以下に示す。

インターフェース部 下側

表 II.7 インターフェース部 下側の PIN の割り付け

ピン番	ピン名	説明	ピン番	ピン名	説明
BB38	A0	ポート A	AU35	C0	ポート C
BC37	A1	〃	AV34	C1	〃
BB36	A2	〃	AU33	C2	〃
BC35	A3	〃	AV32	C3	〃
BB34	A4	〃	AU31	C4	〃
BC33	A5	〃	AV30	C5	〃
BB32	A6	〃	AU29	C6	〃
BC31	A7	〃	AV28	C7	〃
BB30	A8	〃	AU25	C8	〃
BC29	A9	〃	AV24	C9	〃
BB28	A10	〃	AU23	C10	〃
BC27	A11	〃	AV22	C11	〃
BB26	A12	〃	AU21	C12	〃
BC25	A13	〃	AV20	C13	〃
BB24	A14	〃	AU19	C14	〃
BC23	A15	〃	AV18	C15	〃
BB20	B0	ポート B	AU15	D0	ポート D
BC19	B1	〃	AV14	D1	〃
BB18	B2	〃	AU13	D2	〃
BC17	B3	〃	AV12	D3	〃
BB16	B4	〃	AU11	D4	〃
BC15	B5	〃	AV9	D5	〃
BB14	B6	〃	AU8	D6	〃
BC13	B7	〃	AV7	D7	〃
BB12	B8	〃			
BC11	B9	〃			
BB10	B10	〃			
BC9	B11	〃			
BB8	B12	〃			
BC7	B13	〃			
BB6	B14	〃			
BC5	B15	〃			

インターフェース部 上側

表 II.8　インターフェース部 上側の PIN の割り付け

ピン番	ピン名	説明	ピン番	ピン名	説明
B38	I/OA0	ポート A	F36	I/OC0	ポート C
A37	I/OA1	〃	G35	I/OC1	〃
B36	I/OA2	〃	F34	I/OC2	〃
A35	I/OA3	〃	G33	I/OC3	〃
B34	I/OA4	〃	F32	I/OC4	〃
A33	I/OA5	〃	G31	I/OC5	〃
B32	I/OA6	〃	F30	I/OC6	〃
A31	I/OA7	〃	G29	I/OC7	〃
B30	I/OA8	〃	F26	I/OC8	〃
A29	I/OA9	〃	G25	I/OC9	〃
B28	I/OA10	〃	F24	I/OC10	〃
A27	I/OA11	〃	G23	I/OC11	〃
B26	I/OA12	〃	F22	I/OC12	〃
A25	I/OA13	〃	G21	I/OC13	〃
B24	I/OA14	〃	F20	I/OC14	〃
A23	I/OA15	〃	G19	I/OC15	〃
B20	I/OB0	〃	F16	I/OD0	ポート D
A19	I/OB1	〃	G15	I/OD1	〃
B18	I/OB2	〃	F14	I/OD2	〃
A17	I/OB3	〃	G13	I/OD3	〃
B16	I/OB4	〃	F12	I/OD4	〃
A15	I/OB5	〃	G11	I/OD5	〃
B14	I/OB6	〃	F10	I/OD6	〃
A13	I/OB7	〃	G9	I/OD7	〃
B12	I/OB8	〃			
A11	I/OB9	〃			
B10	I/OB10	〃			
A9	I/OB11	〃			
B8	I/OB12	〃			
A7	I/OB13	〃			
B6	I/OB14	〃			

II.3　インターフェースカードの使い方

最初にするべきことは、ベースアドレスの設定である。ISA バスでは、デバイスを識別するための 16bit の I/O アドレス空間がある。デバイスは使用するアドレスを占有する。

インターフェースは、連続した 8 個のアドレスを使用する。その最初のアドレスを DIP スイッチを用いて設定する。ベースアドレスは 0x0000 ～ 0xfff8 まで 8byte を単位として設定できる。例えば図 II.15のように設定すると 0xfff0 ～ 0xfff7 までを使用する (下位 3bit は設定しても無視される)。このとき、 OFF が 1 で ON が 0 である。当然、他のデバイスが使用していないアドレスを使用する。この場合、表 II.9のようにマッピングされる。このような設定をした後、 ISA BUS に装着する。

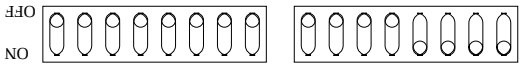


図 II.15　DIP スイッチ [15]

表 II.9　インターフェースカードの I/O マッピング

I/O アドレス	割り当て
Base Address	ポート A 下位
Base Address + 1	ポート A 上位
Base Address + 2	ポート B 下位
Base Address + 3	ポート B 上位
Base Address + 4	ポート C 下位
Base Address + 5	ポート C 上位
Base Address + 6	コントロール ワード 下位
Base Address + 7	コントロール ワード 上位

II.4　評価ボードの使い方

インターフェースの I/O ポートのモード

コンフィグレーションで使用するインターフェースの端子は、 C0 C1 C2 C8 C9 C10 B14 B15 である。このため、インターフェースの I/O ポートのモードは表 II.10のように設定しなければならない。他のポートは自由に設定できる。

[15] ファイル名：./fig2/ifdip.eps

表 II.10　インターフェースの I/O ポートのモード

ポート C 下位	出力	C2:nCONFIG C1:DLCK C0:DATA0
ポート C 上位	出力	C8 C9 C10: FLEX 選択
ポート B 上位	入力	B14:nSTATUS B15:CONF_DONE

コンフィグレーション後のモード変更

PPI8255 はモードを変更すると全ての出力バッファを 0 にクリアする。このため、モードを変更すると、nCONFIG が Low にされてしまい、FPGA がコンフィグレーションモードに入ってしまう事がある。これを防ぐには以下のことを行う必要がある。

1. ポート C 上位は出力にする。
2. コンフィグレーションのあとは、FPGA の選択を 000 に設定する。

コンフィグレーション後に使用可能な端子

コンフィグレーションを行った後は、ポート C 上位に 000 を出力していれば、他のポートは自由に使用できる。

FLEX 2nd のコンフィグレーション

B ポート上位はコンフィグレーション時に FLEX 2nd からドライブされる。もし FLEX 1st で使用している場合は FLEX 1st の端子をハイインピーダンスか入力にしておく必要がある。

FLEX 1st のコンフィグレーションプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

#define BASE_8255 0xfff0

#define P_A    BASE_8255
#define P_B    BASE_8255 + 2
#define P_C    BASE_8255 + 4

#define P_CTRL BASE_8255 + 6
```

```
#define P_AL P_A
#define P_AH P_A + 1
#define P_BL P_B
#define P_BH P_B + 1
#define P_CL P_C
#define P_CH P_C + 1

#define P_CTRL_L P_CTRL
#define P_CTRL_H P_CTRL + 1

#define DATA0 0x1
#define DCLK  0x02
#define nCONFIG 0x04

#define CONF_DONE 0x40
#define nSTATUS 0x80

/*  A:in m0  B:in m0 CH:in CL:out   0x9A */
/*  A:in m0  B:in m0 CH:out CL:out   0x92 */
/*  A: mode2 BL: out BH: in CL3: out CH3: in    0xDB(H)    0xD8(L) */

int main(int argc, char *argv[])
{
long i;

printf("TTF Downloader for FLEX10K100 Eval. Board.\n");

if(argc != 2)
{
        printf("usage : flex10k foo.ttf \n");
        exit(1);
}

FILE *fp;

if ((fp = fopen(argv[1], "rt") == NULL){
fprintf(stderr, "%s を開けません\n",argv[1]);
return 1;
}

outp( P_CTRL , 0xC8);
outp(P_CTRL_H , 0xCA );

outp(P_CL,0xff);
outp(P_CH,0x04);

{
printf("nSTATUS = H  Check...");
unsigned char a;
a = inp(P_BH) & nSTATUS;
if ( a != nSTATUS ){
        printf("nSTATUS が High でない\n");
        exit(1);
}
printf("OK!\n");
}

outp(P_CL , 0 );

printf("nCONFIG PLUS Check...");
while(1)
{
        unsigned char a;
        a = inp(P_BH) & nSTATUS;
        if ( a != nSTATUS ) break;
}

outp(P_CL , nCONFIG );

while(1)
{
        unsigned char a;
```

```
        a = inportb(P_BH) & nSTATUS;
        if ( a == nSTATUS ) break;
}
printf("OK!\n");

for (i = 0l ; i < 8l ; i++)
{
        outp( P_CL , nCONFIG | 1 );
        outp( P_CL , nCONFIG | DCLK | 1);
}

printf("Now Downloading...");

for ( i= 0l ; i < 150000l ; i++){
        unsigned char c;
        int c1;
        if ( EOF == fscanf(fp,"%d,",&c1) ) break;
        int j;
        c = c1;
        for ( j=0l ; j< 8l ; j++){
                outp( P_CL , nCONFIG |        ( c & 1 ) );
                outp( P_CL , nCONFIG | DCLK | ( c & 1 ) );
                c >>= 1;
}

if ( (inp( P_BH ) & 0xC0 ) == 0xC0 ) break;
if ( i == 149132l )
{
        printf("Abnormal end.\n");
        exit(1);
}

}

if ( i != 149131l)
{
        printf("Abnormal end.\n");
        exit(1);
}


for (i = 0l ; i< 8l ; i++)
{
        outp( P_CL , nCONFIG );
        outp( P_CL , nCONFIG | DCLK);
}

printf("Normal end.\n");
outp(P_CH,0x00);

return 0;

}
```

FLEX 2nd のコンフィグレーションプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

#define BASE_8255 0xfff0

#define P_A    BASE_8255
#define P_B    BASE_8255 + 2
#define P_C    BASE_8255 + 4

#define P_CTRL BASE_8255 + 6

#define P_AL P_A
#define P_AH P_A + 1
```

```
#define P_BL P_B
#define P_BH P_B + 1
#define P_CL P_C
#define P_CH P_C + 1

#define P_CTRL_L P_CTRL
#define P_CTRL_H P_CTRL + 1

#define DATA0 0x1
#define DCLK  0x02
#define nCONFIG 0x04

#define CONF_DONE 0x40
#define nSTATUS 0x80

/*  A:in m0  B:in m0 CH:in CL:out   0x9A */
/*  A:in m0  B:in m0 CH:out CL:out   0x92 */
/*  A: mode2 BL: out BH: in CL3: out CH3: in    0xDB(H)    0xD8(L) */

int main(int argc, char *argv[])
{
long i;

printf("TTF Downloader for FLEX10K100 Eval. Board.\n");

if(argc != 2)
{
        printf("usage : flex10k foo.ttf \n");
        exit(1);
}

FILE *fp;

if ((fp = fopen(argv[1], "rt") == NULL){
fprintf(stderr, "%s を開けません\n",argv[1]);
return 1;
}

outp( P_CTRL , 0xC8);
outp(P_CTRL_H , 0xCA );

outp(P_CL,0xff);
outp(P_CH,0x05);

{
printf("nSTATUS = H  Check...");
unsigned char a;
a = inp(P_BH) & nSTATUS;
if ( a != nSTATUS ){
        printf("nSTATUS が High でない\n");
        exit(1);
}
printf("OK!\n");
}

outp(P_CL , 0 );

printf("nCONFIG PLUS Check...");
while(1)
{
        unsigned char a;
        a = inp(P_BH) & nSTATUS;
        if ( a != nSTATUS ) break;
}

outp(P_CL , nCONFIG );

while(1)
{
        unsigned char a;
        a = inportb(P_BH) & nSTATUS;
        if ( a == nSTATUS ) break;
}
```

```
printf("OK!\n");

for (i = 0l ; i < 8l ; i++)
{
        outp( P_CL , nCONFIG | 1 );
        outp( P_CL , nCONFIG | DCLK | 1);
}

printf("Now Downloading...");

for ( i= 0l ; i < 150000l ; i++){
        unsigned char c;
        int c1;
        if ( EOF == fscanf(fp,"%d,",&c1) ) break;
        int j;
        c = c1;
        for ( j=0l ; j< 8l ; j++){
                outp( P_CL , nCONFIG |        ( c & 1 ) );
                outp( P_CL , nCONFIG | DCLK | ( c & 1 ) );
                c >>= 1;
}

if ( (inp( P_BH ) & 0xC0 ) == 0xC0 ) break;
if ( i == 1491321 )
{
        printf("Abnormal end.\n");
        exit(1);
}

}

if ( i != 1491311)
{
        printf("Abnormal end.\n");
        exit(1);
}


for (i = 0l ; i< 8l ; i++)
{
        outp( P_CL , nCONFIG );
        outp( P_CL , nCONFIG | DCLK);
}

printf("Normal end.\n");
outp(P_CH,0x00);

return 0;

}
```

II.5 FLEX10K

本研究で用いるFPGAは、米ALTERA社から提供されるFLEXシリーズである。特徴として、他社の製品に比べ、実現できるゲート数が大きいことがありこの点に注目した。大きな物では10万ゲート相当の回路を実現できる(FLLEX10K100)。プログラム素子はSRAMで構成されているので、電源を投入する度に内容を書き込む必要がある。

コンフィグレーション

FLEX シリーズの記憶素子は、SRAM で構成されている。SRAM 素子は電源が入っていなければ内容が消えてしまう。このため、電源を投入する度に、設計した回路のデータ (configuration data) を、SRAM に送ること (down load) により、デバイスに機能を持たせる。

設計した回路をデバイスに乗るデータにしてダウンロードすることをコンフィグレーションという。FLEX の場合、デバイスを基板上に実装したままの状態でコンフィグレーションする。このデバイスのコンフィグレーション法には幾つかの方法があるが、今回は比較的簡単な Passive Serial 法を用いる。この方法は使用する信号線が少ない利点がある。他に Configration EPROM 法、Passive Synchronous 法、Passive Parallel Asynchronous 法がある。

Passive Serial 法で使用される端子 (FLEX10K)

実際に外部との接続に用いるのは、nSTATUS nCONFIG CONF_DONE DCLK DATA0 の 5 本である。

表 II.11 Passive Serial 法で使用される端子 (FLEX10K)

MSEL0	入力	方法選択 0
MSEL1	入力	方法選択 1
nCE	入力	Low にしないとコンフィグできない
nCE0	出力	カスケード接続用信号
nSTATUS	双方向	コンフィグ status 信号
nCONFIG	入力	コンフィグ制御信号
CONF_DONE	双方向	デバイスがコンフィグされると High
DCLK	入力	データ入力クロック
DATA0	入力	データ線

コンフィグレーション法の選択 (FELX10K)

MSEL0 と MSEL1 を用いて、 Configration 法を選択する。今回は Passive Serial 法を用いるので、 MSEL0 MSEL1 ともに 0 にする。

表 II.12 コンフィグレーション法の選択 (FELX10K)

MSEL0	MSEL1	コンフィグレーション法 (FLEX10K)
0	0	コンフィグレーション EPROM or passive serial
1	0	Passive Parallel Synchronous
1	1	Passive Parallel Asynchronous

コンフィグレーション回路

基本的に 3 本の入力と 2 本の出力合わせて 5 本の信号を接続する。これらの信号を外部の I/O で制御する。

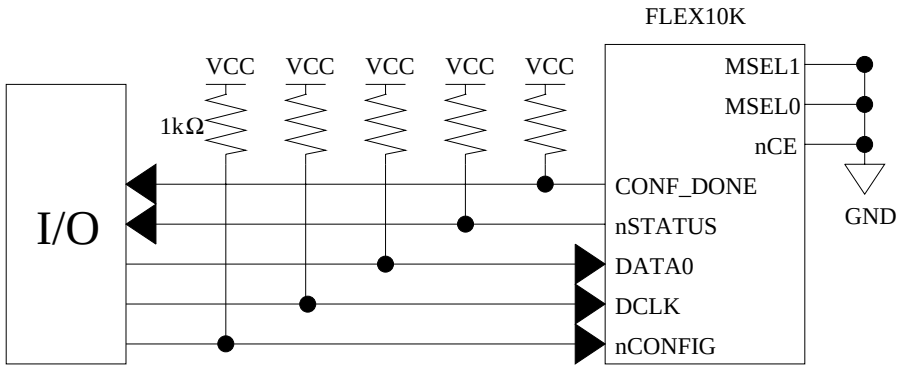


図 II.16 コンフィグレーション回路 [16]

[16] ファイル名：./fig2/config.eps

コンフィグレーションのタイミング波形

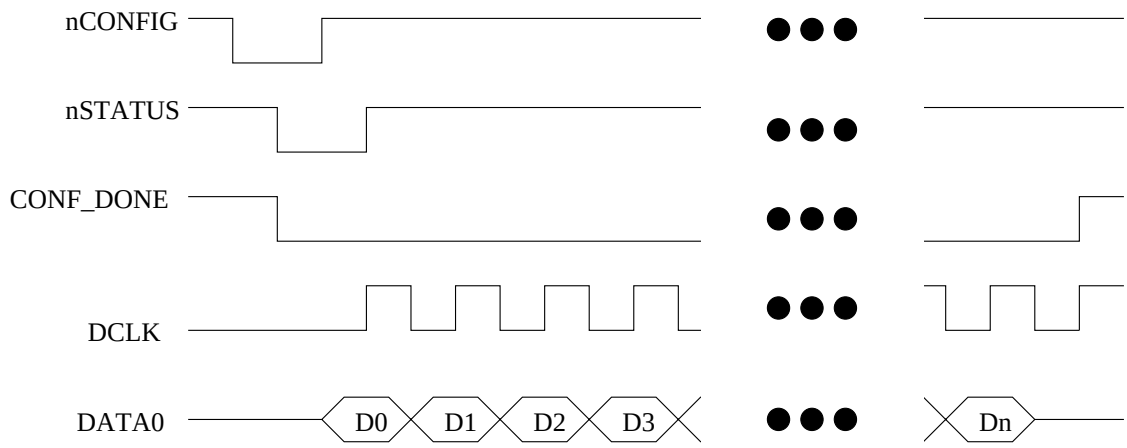


図 II.17 コンフィグレーションのタイミング波形 [17]

表 II.13 コンフィグレーション端子の意味

nCONFIG	パルスを与えることにより、コンフィグの開始の意味になる。 パルスは幅は 2μ 以上。
nSTATUS	最初のパルスはデバイスがコンフィグモードになったことを示す。 コンフィグの途中で Low になるとエラーを示す。
CONF_DONE	デバイスの状態を示す。Low で未コンフィグ状態を示す。
DCLK	立上りで、DATA の読み込みを行う。10Mhz 以下。
DATA0	ここに 1bit づつ、configration data を与える。 TTF のデータは 1byte 単位だが、下位ビットから先に送る。

コンフィグレーションファイル

FLEX のコンフィグレーションデータは、ALTERA 社の配置配線ツール Max+PLUS II で生成できる。このデータファイルをコンフィグレーションファイルという。ファイル形式は幾つかあるが、今回用いるのは、Tabular Text File (.TTF) である。TTF 形式は、コンマ (,) で区切られた ASCII 形式のファイルである。このため、C 等のプロ

[17] ファイル名：./fig2/timewave.eps

グラム言語で、ソースコード中に埋め込んだり、データファイルとして扱うことができる。実際のダウンロード時には先頭に 0xFF を付加しておかないとデバイスが動作しないので注意する必要がある。

II.6 PPI8255

マイコンの周辺 LSI でプログラマブルパラレル I/O として良く使われるデバイスである。8bit の I/O ポートを A B C の 3 つを持っている。全てのポートに出力ラッチがあるので、書き込んだデータは保存されていて随時読み出し可能である。A B C を 3 つのポートとして使う場合と、C ポートを制御用信号として A ポート用 B ポート用の 2 つに分けて使う場合がある。入出力は TTL コンパチブルである。

I/O ポートのモード

PPI8255 の I/O ポートは 3 つのモードを持っている。Mode 0 と Mode 1 と Mode 2 である。Mode 2 は ポート A のみ設定可能であるが、他のモードはポート A ポート B それぞれ別に指定できる。したがって、ポート A を mode 2 ポート B を mode 0 という設定も可能である。

Mode 0

Mode 0 は通常の I/O ポートである。A と B そして C の半分づつの入出力を自由に設定できる。

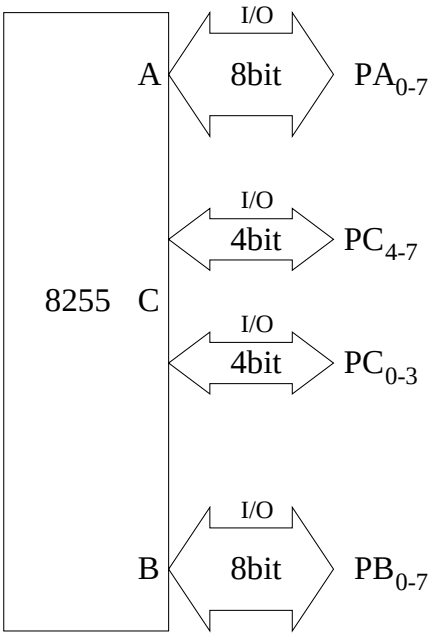


図 II.18 Mode 0 [18]

[18]ファイル名：./fig2/ppi8255-mode0.eps

Mode 1

Mode 1 は、ストローブ付き I/O ポートである。ポート A に関しては C3 C4 C5 C6 C7、ポート B に関しては C0 C1 C2 をハンドシェーク信号として使用する。

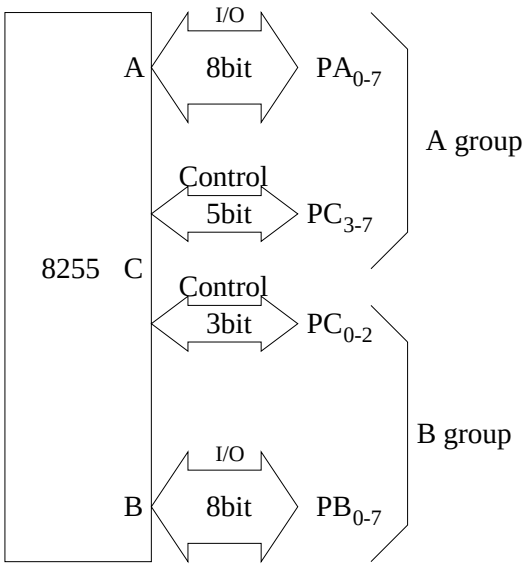


図 II.19 Mode 1 [19]

[19] ファイル名：./fig2/ppi8255-mode1.eps

Mode 2

Mode 2 は、ストローブ付き双方向 I/O ポートである。ポート A のみが設定可能である。ポート A をストローブ付き双方向 I/O ポートとして使用する。 C3 C4 C5 C6 C7 をハンドシェーク用信号として使用する。この時ポート B は Mode 0、 Mode 1 どちらでも構まわない。

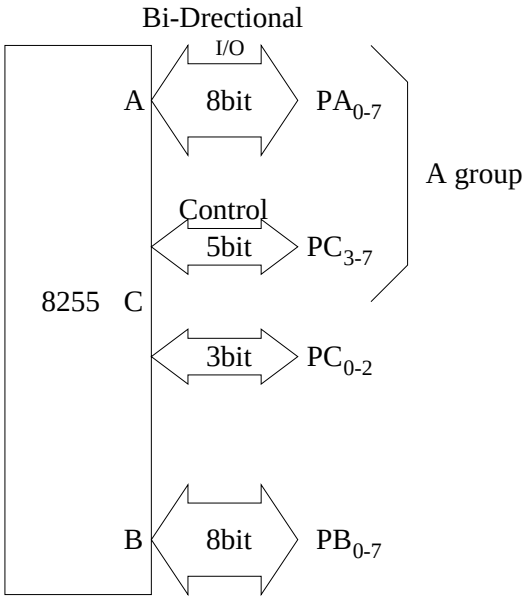


図 II.20 Mode 2 [20]

モードの設定

I/O ポートのモード設定はコントロールワードに書き込むことにより行う。コントロールワードの読み出しはできない。

コントロールワードのビットの内訳は次の通りである。

[20] ファイル名：./fig2/ppi8255-mode2.eps

表 II.14 コントロールワードのビットの内訳

D7	1 → モード設定, 0 → ビット・セット / リセット
D6	ポート A の mode 指定
D5	D6 D5 : 00 → mode 0 , 01 → mode 1 , 1X → mode 2
D4	ポート A の方向 : 0 → 出力, 1 → 入力
D3	ポート C 上位の 方向 : 0 → 出力, 1 → 入力
D2	ポート B の mode 指定 0 → mode 0, 1 → mode 1
D1	ポート B の方向 : 0 → 出力, 1 → 入力
D0	ポート C 下位の方向 : 0 → 出力, 1 → 入力

例として、 10011011 = 0x9B なら A B ともにモード 0 で全てのポートが入力モードに設定される。

ポート C のビット・セット / リセット

コントロールワードに書き込むことにより、指定してポート C の信号の 1 つの状態を強制的にに変化させることができる。

表 II.15 ポート C のビット・セット / リセット

D7	1 → モード設定, 0 → ビット・セット / リセット
D6	X
D5	X
D4	X
D3	D3 D2 D1 の 3bit で セット / リセット するビットを指定する
D2	
D1	
D0	0 → ビットリセット,1 → ビットリセット

例として、 00000001 なら bit0 が 1 にセットされる。