1999 年度 修士論文

書き換え可能なゲート素子を持つデバイスを用いた行列計算専用集積回路の設計

電気通信大学 大学院 電気通信学研究科 電子工学専攻

9830046　沼 知典

指導教官 齋藤 理一郎 助教授

木村 忠正 教授

提出日 平成 12 年 2 月 2 日

目次

第 1 章

序論

本章では、最初に本研究の背景と前年度までの研究成果を述べ、次に行列計算の問題点、この研究で使用している VHDL と FPGA やハウスホルダ法、最後に本研究の目的を述べる。

1.1 本研究の背景

量子力学をはじめとする科学計算においては、膨大な計算量を要する。物性計算を例にとると、行列の固有値、固有ベクトルを求める必要がある。その計算量は行列の次数を N とすると $O(N^3)$、つまり次数の 3 乗に比例し、科学計算で使われる 1000 次以上の大規模な計算においては PC では数日以上かかり、コンピュータの使用効率を下げてしまう。

この計算時間短縮の手法として、並列コンピュータを用いた計算の並列化や新しい行列計算アルゴリズムなどがあげられる。しかし、並列化の問題点としては、並列化できない演算部分があり、コンピュータ数を増やしても、その部分は計算時間の短縮はできない。また、コンピュータ間での通信にも時間がかかるため、その部分も短縮はできない。

また、新しい行列計算アルゴリズムとして、$O(N)$ 法などがあげられるが、このようなアルゴリズムでは、計算時間の短縮が期待できるが、厳密解が得られないという問題点がある。

そこで本研究では、計算時間短縮の手法として、ハードウェアを用いた専用計算機を用いる方法と取り上げる。この方法では、計算を構成するアルゴリズムをハードウェア的に動作させるために、デジタル集積回路の機能設計を行なう。そして、その機能をプロセッサに搭載することで、専用計算機として計算を行なう。

この専用計算機のプロセッサを設計し、搭載するところにおいて、計算アルゴリズムの複雑さと計算量の多さにより、機能設計に必要なデジタル回路のゲート数は非常に膨大なものとなる。そのため、この回路の設計手法として、ハードウェア記述言語である HDL(Hardware Description Laguage) を、近年用いるようになった。そして、何度でも書き込み可能なゲート素子（プログラマブルデバイス）である FPGA(Field Programmable Gate Array) を用いることで、集積回路の評価が比較的容易なものとなった。

1.2 前年度までの研究成果

本研究は、当初画像技研（株）との共同研究として、科学計算を高速に行なうために、専用の計算機を開発しようという目的で、 1996 年度から始まった研究である。ここでは、前年度までの本研究の成果について述べる。

まず'96 年度は、本研究室の中島 [1] と八木 [2] が、行列の固有値および固有ベクトルを求めるためのアルゴリズムであるハウスホルダ法をこの専用計算機に搭載するアルゴリズムとして採用した。このアルゴリズムを採用した理由として、本研究室で行なわれている、量子力学における分子軌道計算では、行列計算を多用しており、この計算において、固有値および固有ベクトルを求めるために、多くの時間を要しているため、この計算時間を短縮するための手法として、ハウスホルダ法を採用していたからである。さらに、ハードウェア上での三重対角化から逆反復法までの計算過程のモデルを提案した。

そして'97 年度は、松尾 [3] とグェン [4] が、計算アルゴリズムを実際に動作させるためのハードウェアを作成するための設計方法を決めた。研究室で設計を行なうために、設計の容易さと開発コストを考慮しなければならない。そこで、近年デジタル回路の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 を採用した。そして、この言語により設計した機能をハードウェアとして動作させるために、プログラマブルデバイスである FPGA を採用した。

本研究室では、これらを用いた開発環境を得るために、（株）インターリンクより PeakVHDL を HDL 設計ツールとして購入し、（株）日本アルテラ社のユニバーシティ・プログラムに参加し、 FPGA の配置・配線ツールとして MAX+plusII の無償提供を受けた。そして、同社から FLEX10K シリーズのひとつである EPF10K100GC503-4 という FPGA を 2 個購入した。

このFPGAを使用した専用計算機を構築するためには、FPGAを搭載するための基板が必要である。そこで、松尾はこのFPGA2個、かつSRAM、DRAMといったメモリが搭載可能な基板を設計、製作した。そして、PCとこの基板間でデータの通信が可能なインターフェースボードを製作した。そして、計算アルゴリズムをVHDLによって記述し、シミュレーションによって、この計算アルゴリズムをハードウェアレベルで動作させるためのモデルを作成した。

'98年度は、山岡[5]と私により、先に製作された基板を利用してハウスホルダ法のアルゴリズムを使い、実際に行列の固有値と固有ベクトルの計算をハードウェア上で動作させた。まず、基板とPCとの間でデータの通信を行なうためのVHDLを設計し、PCとFPGA間の通信を行なった。そして、SRAMコントローラを設計し、計算の対象となるデータをSRAM(Static Random Access Memory)に記憶させることができた。

それから、固有値計算を行なうための準備として、積和器の設計を行なった。この積和器は行列の計算を行なう上で非常に重要な要素となっている。最後に、ハウスホルダ法の三重対角化から逆反復法などの４つのアルゴリズムをVHDLで設計し、実際に行列の固有値計算がハードウェア上で動作が可能となった。ただし、この動作にはSRAMを用いているので、メモリー量に制限がある。

1.3 行列計算

先にも述べたが、行列の固有値、固有ベクトルを計算するには膨大な時間を要する。そこで、この行列計算の時間短縮の手法として以下の方法があげられる。

1.3.1 並列化の問題点

行列計算の時間短縮の手法として、まず並列コンピュータによる計算の並列化があげられる。この方法は、コンピュータ、つまりプロセッサの数を複数にして、同時に多くの演算ができるようになっている。例えば、行列の掛け算の場合だと、以下のような方法で並列化が実現できる。

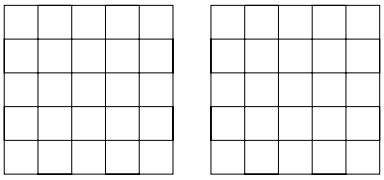


図 1.1: 行列の計算[1]

まず、図 1.1の行列を、図 1.2のように行列 MA を行ベクトルごとに分解し、行列 MB を列ベクトルごとに分解する。

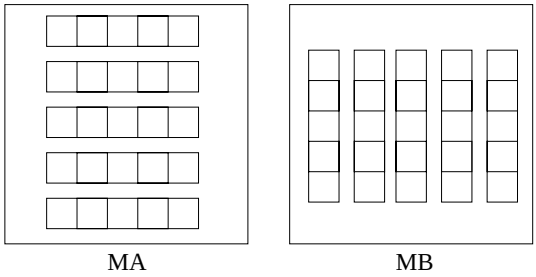


図 1.2: 行列のベクトル分解[2]

そして、図 1.3のように１つの列ベクトルを行ベクトルと同じ数のプロセッサに送り、各行ベクトル要素をそれぞれのプロセッサに送り、それぞれのベクトルの内積を計算させる。計算が終わったら、次の計算をするために、行ベクトルと列ベクトルをプロセッサに送る。

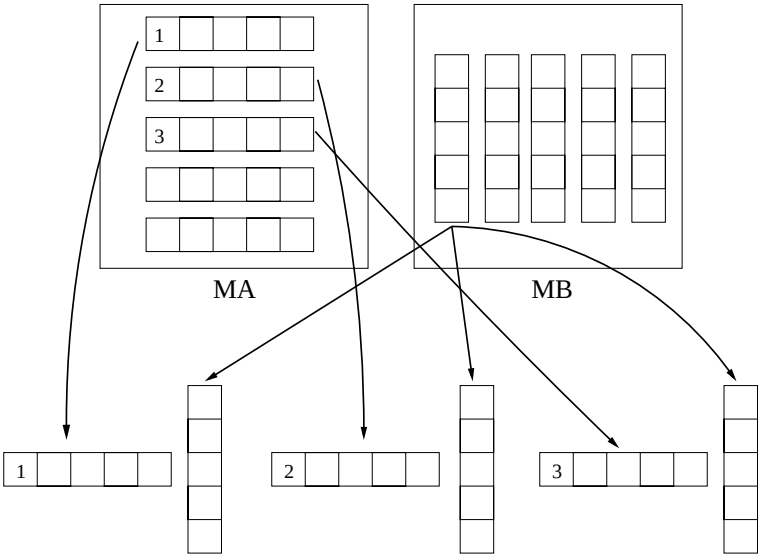


図 1.3: 行列のベクトル分解[3]

この方法だと、この計算に使われるすべての計算データを格納している共有メモリと各プロセッサとの間のデータの通信の速度に問題がある。この通信はプロセッサの演算に比べて低速のため、この通信時間が計算時間短縮のネックとなっている。また、

[1]ファイル名: ./fig/calc1.eps

[2]ファイル名: ./fig/calc2.eps

[3]ファイル名: ./fig/calc3.eps

共有メモリでは、プロセッサの数が多ければ多いほど、データのアクセス量が多くなり、その通信によりプロセッサの待ち時間が増え、計算時間を増やす要因となってしまう。そのため、メモリとの通信を極力減らし、メモリを使わずにプロセッサの中で演算ができるようにし、データを待ち時間なく処理できるようにする必要がある。また、データバスが小さいと、処理できるデータ量が少なくなり、プロセッサの効率を下げてしまう。データバスを大きくとる必要がある。

1.4 VHDL と FPGA

本研究では、行列の固有値および固有ベクトルの計算をハードウェアで実現するために、ハードウェア記述言語である VHDL を設計手段として採用し、プログラマブルデバイスである FPGA を先に設計した機能の実現のために購入した。この VHDL と FPGA について説明する。

1.4.1 VHDL

VHDL(VHSIC Hardware Description Language) はハードウェア記述言語のひとつで、これと論理合成ツール (MaxPlusII+ など) を用いてハイレベル設計手法 (High-level Design Methodology) による論理回路の設計が、現在主流となってきている。ハイレベル設計手法については次章で述べる。

VHDL の特徴は、このデジタル回路の設計、シミュレーション、合成のすべてを実行するための幅広い構文を持っていること、そして、シミュレーションによる動作検証がしやすく、設計の変更に時間がかからないこと、言語記述がそれほど複雑ではないので、習得が容易であることだといえる。

また、 HDL には Verilog-HDL というのがあり、こちらは C 言語に近い構文の記述ができることや、各記述ブロック毎に各信号の入出力の指定が必要という特徴を持っている。したがって、本研究では、言語の習得、動作検証が容易で、各信号の定義が自由な VHDL を採用した。

1.4.2 FPGA

FPGA(Field Programmable Gate Array) とは、 PLD(Programmable Logic Device) の一種で、何度でもデジタル回路機能を書込みできるゲート素子である。この FPGA は数万のゲートと数千のロジック要素、数十 kbits の SRAM から構成されてい

る。図 1.4に FPGA の内部構造を簡単に示す。 FPGA は EAB(Embedded Array Block) とロジックアレイ (Logic Array)、 I/O 要素から成っている。 EAB は入出力レジスタを持つ RAM のブロックでできており、 FPGA 内部のメモリやマルチプライヤなどに使われる。ロジックアレイはロジックを構成するのに使われる。

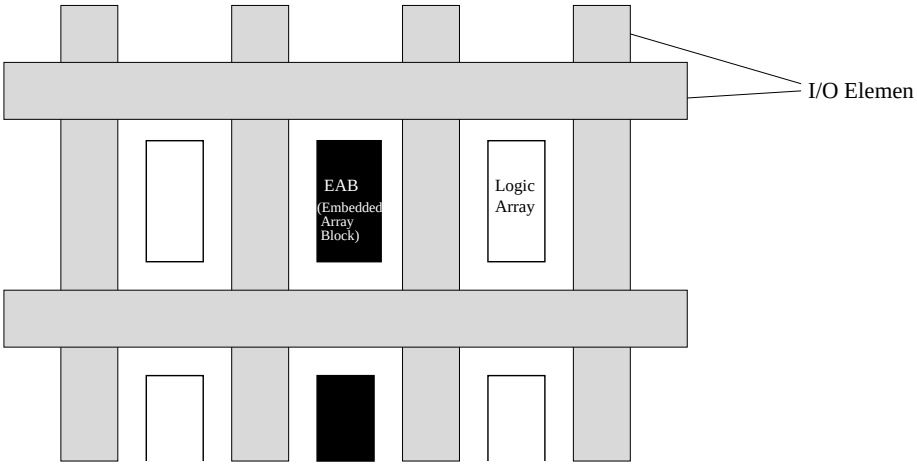


図 1.4: FPGA の内部構造[4]

本研究では、（株）日本アルテラ社から FLEX10k100 という FPGA を 2 個購入した。この FPGA は 10 万ゲート、 4992 個の論理要素、 503 個の外部 pin を持つ。これを用いた基板については、付録 C.2で述べる。

1.5 浮動小数点演算器

昨年度、山岡と共同で浮動小数点演算器を設計した。この演算器は、加算器、乗算器、除算器、平方根計算器からなる。この FPGA を使って、これらの演算器を論理回路として設計した。これらの VHDL での設計方針については、 4 章で説明する。ここではこれらの動作について説明する。また、これに対応する VHDL ソースを付録 B.2に示した。

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なわれる。

[4]ファイル名:./fig/FPGA.eps

1.5.1 浮動小数点加算器

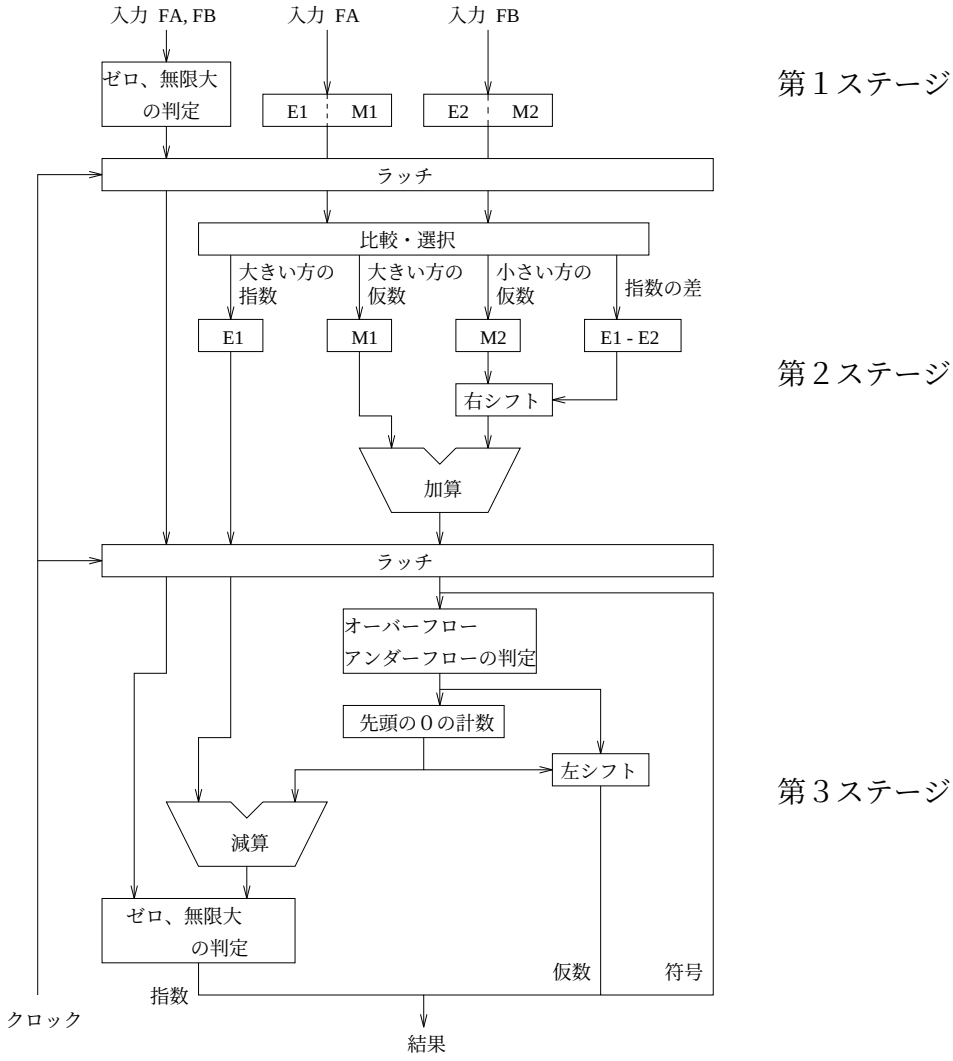


図 1.5: 加算器[5]

この加算器の演算は 3 ステージで構成され、同期パイプライン方式を用いている。このステージ構成を図 1.5に示す。第 1 ステージに入る前に、入力するデータ FA,FB はそれぞれ指数部 E1,E2 と仮数部 M1,M2 に分離され、演算が行なわれる。この図に示してあるラッチによって、各ステージはシステムクロックに同期して、独立して動作するようになっている。

最初に第 1 ステージでは、 FA と FB のゼロおよび無限大の判定を行なう。指数部がすべて'0' ならばゼロ、すべて'1' ならば無限大とみなす。そして、第 2 ステージで

[5]ファイル名: ./fig/add-flow.eps

は、 FA と FB の比較を行ない、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 の加算を行なう。最後に第 3 ステージでは、その加算結果のオーバーフロー、アンダーフローの判定、仮数部の正規化、指数部の調整を行ない、計算結果 Q を出力する。

1.5.2 浮動小数点乗算器

この乗算器も加算器同様に、 3 つのステージから構成されている。このステージ構成を図 1.6に示す。指数部、仮数部などについては、加算器と同じである。

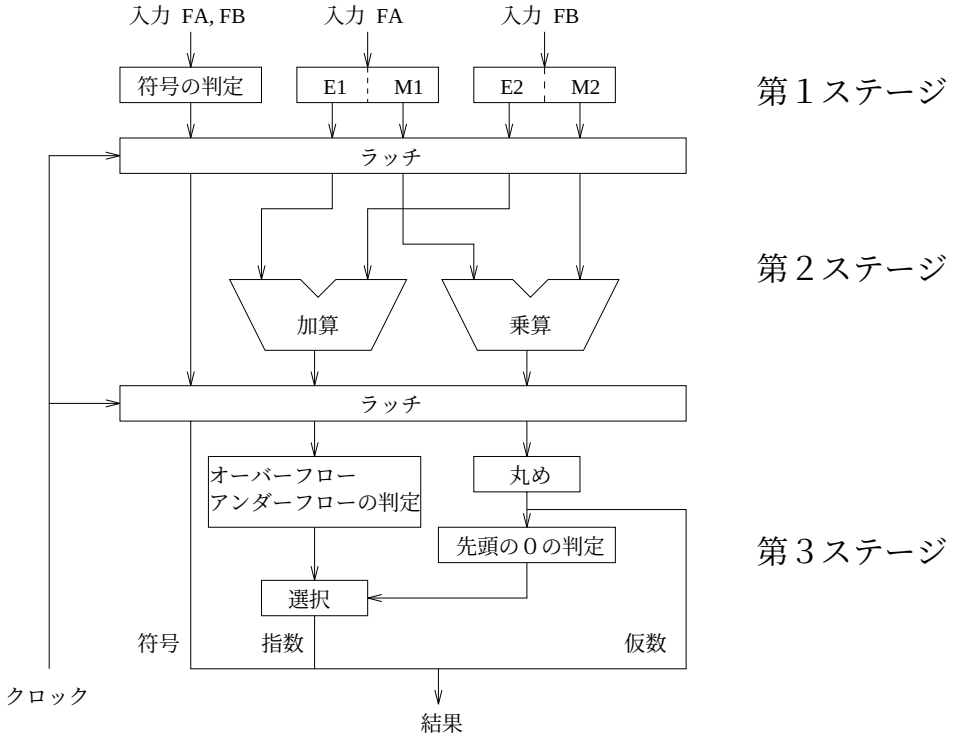


図 1.6: 乗算器[6]

最初に第 1 ステージでは、積の符号の判定をする。そして、第 2 ステージでは、指数部の加算とともに、仮数部の乗算を行なう、そして、次のステージの計算に不要な乗算結果の下位 22bit を切り捨てる。第 3 ステージでは、指数部の結果のオーバーフローとアンダーフローの判定、仮数部の乗算結果の丸めを行ない、最後に正規化して計算結果 Q を出力する。

[6] ファイル名: ./fig/mul-flow.eps

1.5.3 浮動小数点除算器

この除算器も加算器同様に、 3 つのステージから構成されている。このステージ構成を図 1.7に示す。指数部、仮数部などについては、加算器と同じである。

図 1.7に示す。

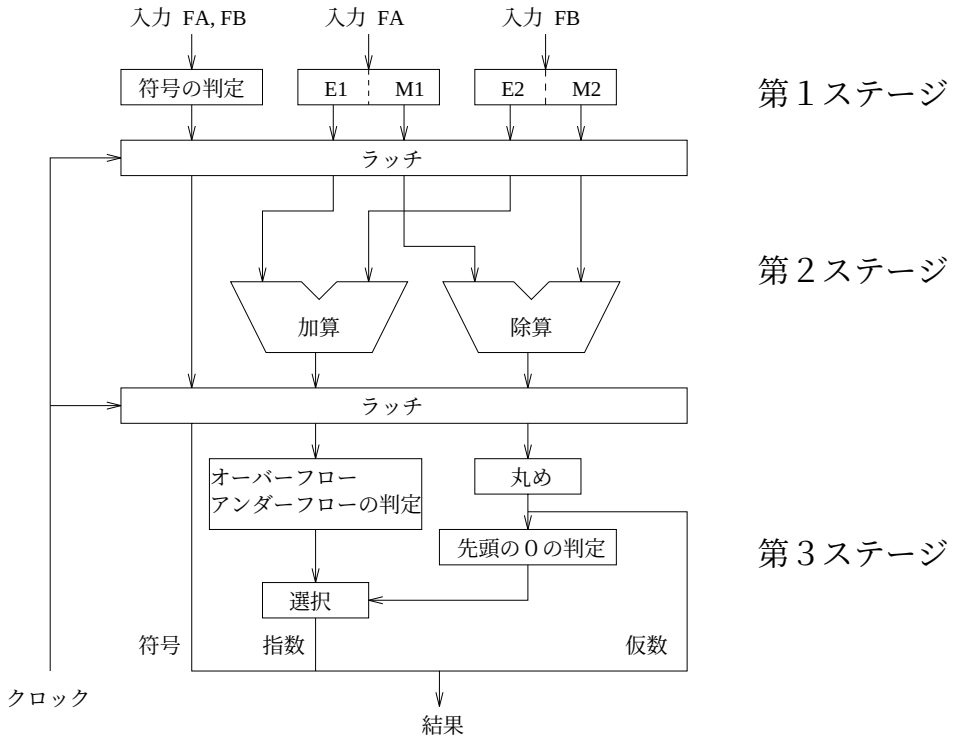


図 1.7: 除算器[7]

最初に第 1 ステージでは、積の符号の判定をする。そして、第 2 ステージでは、指数部の加算は (E1-E2) と差をとり、仮数部は除算を行なう。第 3 ステージでは、乗算器と同様に、指数部の結果のオーバーフローとアンダーフローの判定、仮数部の乗算結果の丸めを行ない、最後に正規化して計算結果 Q を出力する。

[7]ファイル名: ./fig/div-flow.eps

1.5.4 浮動小数点平方根計算器

平方根の計算はハウスホルダ変換のときに使う。この計算は指数部は右 1 ビットシフトによって、仮数部は for 文によって仮数部の開平算を行なうことで計算を行なっている。このステージ構成を図 1.8に示す。

図 1.8: 平方根計算器[8]

1.6 目的

本研究をするにあたり、 FPGA と VHDL および評価用基板については、先の背景に述べた通り、準備が行なわれた。そして、ハウスホルダ法のアルゴリズムの回路設計が行なわれ、小さい次数ではあるが、行列の固有値と固有ベクトルの計算が可能になった。

しかし、次数が 200 次を超えるような大きい次数の場合、その行列データを記憶させるための SRAM(Static Random Access Memory) が容量に限界があり計算できない。また、浮動小数点の計算において、小数の桁数が大きい計算の場合には、計算結果が正しく出なかったり、計算途中で計算が停止してしまう。

これらの問題点を解決する方法として、前者ではメモリを SRAM から DRAM(Dynamic Random Access Memory) にすることで、データを記憶できる容量を増やすことができる。後者はこの計算機に搭載されている、浮動小数点演算器の仮数部の演算部分に問題があるといえる。そこで、この部分を改良し演算を正確に行なえるようにする。

DRAM は SRAM とは違い、メモリの容量に対する制御に必要なピン数が少ない。例えば、記憶する場所を決めるアドレスピンに関しては、アドレスの指定を 2 回行なうことでピン数を SRAM の半分に減らしている。しかし、そのアドレスの指定のための手続きの動作を数回行なっている。したがって、 DRAM の読み書きに必要なクロック数は SRAM よりも多くなり、その分計算時間がかかってしまう。しかし、 DRAM

[8]ファイル名: ./fig/quart-flow.eps

を使うことで、1000 次以上の行列計算がおこなえ、分子軌道計算などの科学計算などでの使用が可能となる。

浮動小数点演算器については、加算器、乗算器、除算器、平方根計算器の 4 つがあるが、これらのうち、乗算器と除算器に関しては、指数部と仮数部に分離して計算を行なっている。この仮数部の演算において、乗算および除算が行なわれており、その計算が演算器で指定した時間内でできないため、計算が途中で停止してしまう。そこで、この乗算および除算の計算時間に余裕を持たせることで、正確な結果が出る計算を行なうことができる。

そこで、本研究はこの行列専用計算機プロセッサにおいて、データの記憶を SRAM から DRAM に変え、行列計算で計算できる次数を増やすことが目的である。そして、浮動小数点演算気器を改良し、より誤動作しない行列計算ができるようにすることが目的である。

第 2 章

ハウスホルダ法

2.1 ハウスホルダ法について

2.1.1 行列の固有値計算

この専用計算機では、行列の固有値および固有ベクトルの計算を行なう。ここではその計算機に用いるアルゴリズムとして八木 [2] と中島 [1] が採用したハウスホルダ法について説明する。このハウスホルダ法を行列専用計算機の対角化計算アルゴリズムとしてハードウェアで計算できるように設計する。

一般的に行列の固有値および固有ベクトルを計算するとき、n 次正方行列 $\boldsymbol{A}$ に対して、

$$\boldsymbol{A}x = \boldsymbol{\lambda}x \tag{2.1.1}$$

この式のベクトル x とスカラー λ が成り立つ場合、λ を $\boldsymbol{A}$ の固有値 (Eigen value)、x を固有ベクトル (Eigen vector) という。この固有値を求めるとき、一般的には、

$$\det|\boldsymbol{A} - \boldsymbol{\lambda}I| = 0 \tag{2.1.2}$$

という n 次多項式を求める必要がある。しかし、この方法で計算機で計算させると $n!$ 個の項を持つ多項式のため、実用には耐えることができない。

2.1.2 行列の三重対角化

上の固有値を求める計算量を減らすために、ハウスホルダ法を用いる。行列の固有値および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法は、三重対角行列

に変換し、その固有値と固有ベクトルを計算する方法である。三重対角行列に変換することで、固有値の計算に使う計算量を減らすことができる。

ここで述べられている三重対角行列とは、対角要素および副対角要素以外はすべて 0 であるような行列で、下式のような行列のことをいう。

$$\begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & 0 & \\ & \beta_2 & \alpha_3 & \beta 3 & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{2.1.3}$$

このアルゴリズムは、ハウスホルダ変換、二分法、逆反復法、逆ハウスホルダ変換の４つの相似変換および計算法から構成される。ハウスホルダ変換で三重対角行列に変換し、二分法でその三重対角行列の固有値を求め、逆反復法でその行列と固有値から固有ベクトルを求める。そして、ハウスホルダ逆変換で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。

2.2 ハウスホルダ法による計算

2.2.1 ハウスホルダ変換

ハウスホルダ変換 (Householder transformartion) は対称行列を三重対角行列 (Tridiagonal matrix) に変換する方法である。

まず、$n \times n$ の行列$\boldsymbol{A}$を相似変換によって三重対角行列 $\tilde{\boldsymbol{A}}$ に変換する。ここで相似変換とは、

$$\tilde{\boldsymbol{A}} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P} \tag{2.2.4}$$

のことで、$\boldsymbol{P}$ は変換行列で正則行列で構成されている。

この変換行列 $\boldsymbol{P}$ を構成する前に適当な n 次元ベクトル $\boldsymbol{\omega}$ を用いて、

$$\boldsymbol{Q} = \boldsymbol{I} - c\boldsymbol{\omega}\boldsymbol{\omega}^T, c = \frac{2}{\boldsymbol{\omega}^T\boldsymbol{\omega}} \tag{2.2.5}$$

となる $n \times n$ 行列 Q を定義する。行列 $\boldsymbol{Q}$ による $\boldsymbol{A}$ のハウスホルダ変換した行列を $\boldsymbol{B}$ として、

$$\boldsymbol{B} = \boldsymbol{Q}^{-1}\boldsymbol{A}\boldsymbol{Q} \tag{2.2.6}$$

この変換によって、まず $\boldsymbol{B}$ の第 1 列目の第 3 行目以下の成分をすべて 0 にすることである。ここで、ハウスホルダ変換を定義するベクトル $\boldsymbol{\omega}$ を

$$\boldsymbol{\omega} = \begin{pmatrix} 0 \\ a_2 + s \\ a_3 \\ \vdots \\ a_n \end{pmatrix}, \quad s^2 = \sum_{j=2}^{n} a_j^2 \tag{2.2.7}$$

と選ぶ。s の符号は a_2 と同じ符号をとる。このとき (2.2.5) の c は以下のようになる。

$$c = \frac{1}{s^2 + |sa_2|} \tag{2.2.8}$$

したがって、$\boldsymbol{B} = \boldsymbol{Q}^{-1}\boldsymbol{A}\boldsymbol{Q}$ の第 1 列のベクトルを $\boldsymbol{b}$ とおくと、

$$(\boldsymbol{I} - c\boldsymbol{\omega}\boldsymbol{\omega}^T)\boldsymbol{a} = \boldsymbol{b} \tag{2.2.9}$$

が成り立つ。よって

$$\boldsymbol{b} = \begin{pmatrix} a_1 \\ -s \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.2.10}$$

となり $\boldsymbol{B}$ が (2.1.3) 第 1 列の形をもつ。つまり、(2.2.6) の変換は以下のようになる。

$$\begin{aligned} \boldsymbol{B} &= (\boldsymbol{I} - c\boldsymbol{\omega}\boldsymbol{\omega}^T)\boldsymbol{A}(\boldsymbol{I} - c\boldsymbol{\omega}\omega^T) && (2.2.11) \\ &= \boldsymbol{A} - (\boldsymbol{\omega}\boldsymbol{q}^T + \boldsymbol{q}\boldsymbol{\omega}^T) && (2.2.12) \end{aligned}$$

ただし、$\boldsymbol{q}$ は

$$\boldsymbol{p} = c\boldsymbol{A}\boldsymbol{\omega} \tag{2.2.13}$$

$$\boldsymbol{q} = \boldsymbol{p} - \frac{c}{2}\boldsymbol{\omega}\boldsymbol{p}^T\boldsymbol{\omega} \tag{2.2.14}$$

そして、行列 $\boldsymbol{B}$ の第 1 行と第 1 列を除いた残りの $(n-1)\times(n-1)$ の部分について同様の変換を行なう。この操作を $n-2$ 回繰り返せば、もとの行列 $\boldsymbol{A}$ は三重対角行列となる。

つまり、第 1 行から第 $k-1$ 行および第 1 列から第 $k-1$ 列まで三重対角化した行列を $\boldsymbol{A}^{(k)}$ とするとき (2.2.7) と同じように選んだベクトル $\boldsymbol{\omega}^{(k)}$ から行列 $\boldsymbol{Q}^{(k)} = \boldsymbol{I} - c_k\boldsymbol{\omega}^{(k)}\boldsymbol{\omega}^{(k)T}$ を構成し、この $\boldsymbol{Q}^{(k)}$ による相似変換

$$\boldsymbol{A}^{(k+1)} = (\boldsymbol{Q}^{(k)})^{-1}\boldsymbol{A}^{(k)}\boldsymbol{Q}^{(k)} \tag{2.2.15}$$

を行なう。これを $n-2$ 回繰り返して

$$\tilde{\boldsymbol{A}} = \boldsymbol{A}^{(n-1)} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P} \tag{2.2.16}$$

$$\boldsymbol{P} = \boldsymbol{Q}^{(n-2)}\boldsymbol{Q}^{(n-1)}\cdots\boldsymbol{Q}^{(1)} \tag{2.2.17}$$

とすればよい。

以上の手順をまとめると、次のようになる。ただし、 $\boldsymbol{A}^{(k)}$ の第 ij 成分を a_{ij}^k と書く。

$\boldsymbol{A}^{(1)} = \boldsymbol{A}$ とおき、 $k = 1, 2, 3, \cdots, n-2$ について以下の手順を繰り返す。

$$s_k = \sqrt{\sum_{j=k+1}^{n}(a_{jk}^{(k)})^2} \tag{2.2.18}$$

ここで、 $s_k = 0$ ならば以下の 5 式を実行せずに次の k へ進む。

$$c = \frac{1}{s_k^2 + |a_{k+1,k}^{(k)}s_k|} \tag{2.2.19}$$

$$\boldsymbol{\omega}^{(k)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1,k}^{(k)} + s_k \\ a_{k+2,k} \\ \vdots \\ a_{n,k}^{(k)} \end{pmatrix} \tag{2.2.20}$$

$$\boldsymbol{p}^{(k)} = c_k\boldsymbol{A}^{(k)}\boldsymbol{\omega}^{(k)} \tag{2.2.21}$$

$$\boldsymbol{q}^{(k)} = \boldsymbol{p}^{(k)} - \frac{c_k}{2}\boldsymbol{\omega}^{(k)}\boldsymbol{p}^{(k)T}\boldsymbol{\omega}^{(k)} \tag{2.2.22}$$

$$\boldsymbol{A}^{(k+1)} = \boldsymbol{A}^{(k)} - (\boldsymbol{\omega}^{(k)}\boldsymbol{q}^{(k)T} + \boldsymbol{q}^{(k)}\boldsymbol{\omega}^{(k)T}) \tag{2.2.23}$$

2.2.2 二分法

次に、行列 $\boldsymbol{A}$ をハウスホルダ変換によって相似変換してできた三重対角行列 $\tilde{\boldsymbol{A}}$ から固有値を計算する。この計算方法としてスツルム (Sturm) の定理に基づく二分法 (bisection method) を使う。

この三重対角行列 $\tilde{\boldsymbol{A}}$ に対応して行列 $\lambda\boldsymbol{I}-\tilde{\boldsymbol{A}}$ を考え、その第 k 主小行列式を $p_k(\lambda)$ とおく。

$$\lambda\boldsymbol{I}-\tilde{\boldsymbol{A}}=\begin{pmatrix}\lambda-\alpha_1 & -\beta_1 & & & & \\ -\beta_1 & \lambda-\alpha_2 & -\beta_2 & & \boldsymbol{0} & \\ & -\beta_2 & \lambda-\alpha_3 & -\beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & \boldsymbol{0} & & -\beta_{n-2} & \lambda-\alpha_{n-1} & -\beta_{n-1} \\ & & & & -\beta_{n-1} & \lambda-\alpha_n\end{pmatrix} \tag{2.2.24}$$

この式の漸化式は、

$$\begin{aligned} p_0(\lambda) &= 1,\ \ p_1(\lambda)=\alpha_1-\lambda, \\ p_k(\lambda) &= (\lambda-\alpha_k)p_{k-1}(\lambda)-\beta_{k-1}^2 p_{k-2}(\lambda) \end{aligned} \tag{2.2.25}$$

$k=n$ のときには、

$$p_n(\lambda)=|\boldsymbol{T}-\lambda\boldsymbol{I}| \tag{2.2.26}$$

となり、この n 次多項式 $p_n(\lambda)$ の根が $\tilde{\boldsymbol{A}}$ の固有値、すなわち $\boldsymbol{A}$ の固有値となる。

上の多項式の列 $p_n(\lambda),p_{n-1}(\lambda),\cdots,p_1(\lambda),p_0(\lambda)$ の符号の変化の回数に関して、スツルムの定理から、隣同士符号の変化の回数を $N(\lambda)$ とすると、$N(\lambda)$ は λ より大きい固有値の個数に等しい。

固有値を大きい方から $\lambda_1,\lambda_2,\cdots,\lambda_n$ と順に並べる。もしも、二つの数 a_{j-1},b_{j-1} に関して、$N(a_{j-1})\ge k, N(b_{j-1})<k$ が成立しているとすると、大きい方から第 k 番目の固有値 λ_k は $a_{j-1}<\lambda_k<b_{j-1}$ の間に存在する。この範囲を十分に狭くすれば、λ_k の近似値を求めることができる。この性質を利用してこの固有値を計算する方法が二分法である。この方法では、a_{j-1},b_{j-1} の中点 $c_j=(a_{j-1}+b_{j-1})/2$ に対して $N(c_j)$ を計算し、$N(c_j)\ge k$ ならば $a_j=c_j$、$b_j=b_{j-1}$ に、$N(c_j)<k$ ならば $a_j=a_{j-1}$、$b_j=c_j$ に、置き換えてさらに中点を求めて計算する。この $|a_j-b_j|$ の範囲が指定の精度の範囲になったところで c_j を λ_k の近似値として採用し、計算を終了する。

ここで、始めの区間 (a_0,b_0) はゲルシュゴーリン (Gerschgorin) の定理より、

$$r=\max_{1\le i\le n}\{|\beta_{i-1}|+|\alpha_i|+|\beta_i|\},\quad \beta_0=\beta_n=0 \tag{2.2.27}$$

として、すべて $(-r,r)$ の範囲にあるのがわかっているので、この区間から始めればよい。

実際に $N(\lambda)$ を計算するときは、(2.2.25) を計算する代わりに、

$$g_k(\lambda) = \frac{p_k(\lambda)}{p_{k-1}(\lambda)} \tag{2.2.28}$$

この式についての符号の変化を調べる。すなわち、

$$\begin{cases} g_1(\lambda) = \lambda - \alpha_1 \\ g_k(\lambda) = \lambda - \alpha_k - \frac{\beta_{k-1}^2}{g_{k-1}(\lambda)} \ , \quad (k = 2,3,\cdots,n) \end{cases} \tag{2.2.29}$$

ただし、$g_{k-1}(\lambda)$ がほぼ 0 のときは $g_k(\lambda) > 0$ とみなしてこの $g_k(\lambda)$ の計算は省略し、次のステップで $g_{k+1}(\lambda) = \lambda - \alpha_{k+1}$ として計算する。このとき列 $g_1(\lambda), g_2(\lambda), \cdots, g_n(\lambda)$ のうちで $g_k(\lambda) \geq 0$ となるものの個数が $N(\lambda)$ となる。

2.2.3 逆反復法

次に二分法で求めた固有値をもとに、逆反復法 (Inverse iteration) により固有ベクトルを求める。

$\boldsymbol{B}$ を $n \times n$ 行列として、その固有値を $\nu_1, \nu_2, \cdots, \nu_n$ とする。そして、これらの固有値は

$$|\nu_1| > |\nu_2| \geq |\nu_3| \geq \cdots \geq |\nu_n| \tag{2.2.30}$$

を満たしているものとする。このとき適当な初期ベクトル $\boldsymbol{x}^{(0)}$ から出発して、

$$\boldsymbol{x}^{(l)} = \boldsymbol{B}\boldsymbol{x}^{(l-1)}, \quad l = 1,2\cdots \tag{2.2.31}$$

を計算していくと、$\boldsymbol{x}^{(l)}$ は次第に $\boldsymbol{B}$ の絶対値最大の固有値 ν_1 に対応する固有ベクトルに近づく。これがべき乗法 (Power method) の原理である。

先に述べた行列 $\tilde{\boldsymbol{A}}$ の k 番目の固有値を λ_k、それに対応する固有ベクトルを $\boldsymbol{u}_k$ とする。いま、μ を λ_k の適当な近似値として、$(\mu\boldsymbol{I} - \tilde{\boldsymbol{A}})^{-1}$ となる行列は

$$(\mu\boldsymbol{I} - \tilde{\boldsymbol{A}})^{-1}\boldsymbol{u}_k = \frac{1}{|\mu - \lambda_k|}\boldsymbol{u}_k \tag{2.2.32}$$

が成り立つ。近似値 μ は λ_k に十分近く、他の固有値 λ_j とは、

$$\frac{1}{|\mu - \lambda_k|} > \frac{1}{|\mu - \lambda_j|}, \quad j \neq k \tag{2.2.33}$$

となる関係をもっているとすると、$1/(\mu - \lambda_k)$ は行列 $(\mu \boldsymbol{I} - \tilde{\boldsymbol{A}})^{(-1)}$ の絶対値最大の固有値になる。したがって、適当な初期値 $\boldsymbol{x}^{(l)}$ から始めて

$$\boldsymbol{x}^{(l)} = (\mu \boldsymbol{I} - \tilde{\boldsymbol{A}})^{(-1)} \boldsymbol{x}^{(l-1)} \tag{2.2.34}$$

を次々と計算していくと、$\boldsymbol{x}^{(l)}$ は次第に固有ベクトル $\boldsymbol{u}_k$ に近づいていくことになる。

実際の計算では、上式の計算の繰り返しは連立一次方程式

$$(\mu \boldsymbol{I} - \tilde{\boldsymbol{A}}) \boldsymbol{x}^{(l)} = \boldsymbol{x}^{(l-1)} \tag{2.2.35}$$

を $\boldsymbol{x}^{(l)}$ について解くことによって求めることができる。

ここで、固有値 λ_k に縮退があると、(2.2.34) の反復によって $\boldsymbol{x}^{(k)}$ は λ_k に対応する複数個の固有ベクトルの 1 次結合に近づく。したがって、縮退がある場合には、λ_k に対応する 2 番目の固有ベクトルを求める反復からは、すでに求めてある固有ベクトルと直交する初期ベクトルをとって反復を開始する必要がある。

上式を解くときには、$\boldsymbol{LU}$ 分解を使うとよい。すなわち、$\mu \boldsymbol{I} - \tilde{\boldsymbol{A}}$ をまず

$$\mu \boldsymbol{I} - \tilde{\boldsymbol{A}} = \boldsymbol{LU} \tag{2.2.36}$$

のように L と U の積に分解する。このとき (2.2.35) は、

$$\boldsymbol{LU} \boldsymbol{x}^{(l)} = \boldsymbol{x}^{(l-1)} \tag{2.2.37}$$

と書くことができる。この方程式は次のように

$$\boldsymbol{Ly} = \boldsymbol{x}^{(l-1)} \tag{2.2.38}$$

$$\boldsymbol{U} \boldsymbol{x}^{(l)} = \boldsymbol{y} \tag{2.2.39}$$

に分けて解く。そして、行列 $\tilde{\boldsymbol{A}}$ として前に得た三重対角行列を考えるとすると、

逆反復法の手順 (2.2.37) において $l = 1$ とおくと

$$\boldsymbol{U} x^{(1)} = \boldsymbol{L}^{-1} \boldsymbol{x}^{(0)} \tag{2.2.40}$$

となるが、最初の反復では (2.2.39) の前進代入の部分 $\boldsymbol{L}^{(-1)} \boldsymbol{x}^{(0)}$ は省略することができる。λ_k の近似値 μ が十分に近いとすると 2 回目の反復 $\boldsymbol{L} U x^{(2)} = \boldsymbol{x}^{(1)}$ で十分精度の高い固有ベクトルを得ることができる。

2.2.4 逆ハウスホルダ変換

ハウスホルダ変換により得られた三重対角行列 $\tilde{\boldsymbol{A}}$ は

$$\tilde{\boldsymbol{A}} = \boldsymbol{P}^{(n-2)} \cdots \boldsymbol{P}^{(1)} \boldsymbol{A} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \tag{2.2.41}$$

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

$$\tilde{\boldsymbol{A}} x = \lambda \boldsymbol{x} \tag{2.2.42}$$

これに (2.2.42) を代入して、

$$\boldsymbol{P}^{(n-2)} \cdots \boldsymbol{P}^{(1)} \boldsymbol{A} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} = \lambda \boldsymbol{x} \tag{2.2.43}$$

両辺に左から $\boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)}$ をかけ、正則行列 P が $\boldsymbol{P}^{-1} = \boldsymbol{P}$ であることをであることを用いると、

$$\boldsymbol{A} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} = \lambda \boldsymbol{x} \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} \tag{2.2.44}$$

すなわち

$$\boldsymbol{y} = \boldsymbol{P}^{(1)} \cdots \boldsymbol{P}^{(n-2)} \boldsymbol{x} \tag{2.2.45}$$

$\boldsymbol{y}$ 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。 $\boldsymbol{x}^{(n-2)} = bix$ として、

$$\boldsymbol{x}^{(r-1)} = \boldsymbol{x}^{(r)} - (c_r \boldsymbol{u}^{(r)T} \boldsymbol{x}^{(r)}) \boldsymbol{u}^{(r)} \tag{2.2.46}$$

これを $r = n-2, n-3, \cdots, 1$ で繰り返し計算し、 $\boldsymbol{x}^{(0)}$ が固有ベクトル $\boldsymbol{y}$ となる。

第 3 章

VHDL による行列計算専用集積回路の設計

3.1 集積回路の設計方針

本研究では、行列専用の計算機を開発するにあたり、その計算機の専用の集積回路を開発が行なわれる。ここで、この集積回路の設計に、デジタル回路の設計手法として一般的になっているハードウェア記述言語 HDL(Hardware Description Language) を用いることにした。本研究ではその HDL の一つである VHDL(VHSIC Hardware Description Language) を使い、この集積回路の機能設計を行なう。そして、 VHDL によって設計されたデジタル回路要素は、何度でも書き込み可能なゲート素子である FPGA(Field Programmable Gate Array) が搭載された評価用基板に実装し、ハードウェアとしての動作を検証する。

3.2 VHDL による回路設計

本研究における VHDL による回路設計の流れ図を図 3.1に示す。実際に設計するときの詳しい手順については、付録 Dで述べる。

まず、 PeakVHDL というソフトウェアで VHDL により回路機能を記述した HDL ファイル (*.vhd) というファイルを作成する。そして、構文解析と機能検証を行なう。もし、 VHDL 構文の記述がおかしければ、コンパイル中にエラーメッセージが出る。その場合は正しい文章に書き直し、再度コンパイルする。そして、この機能検証を実行するために、テストベンチをおこなう。これは、デバイスの入力ポートにデータを入力する動作を VHDL で記述したものをシミュレーションでデバイス内の各信号の動作を検証することである。このシミュレーションで信号の動作が正しくなければ、VHDL ファイルを書き直しシミュレーションに戻り、信号の動作が正しくなるまで書

き直す。

本研究の場合には、 FPGA でのピンの配置も VHDL ソース作成の時点で指定する必要がある。この場合には、 entity の部分で port 文で入出力ピンを指定し、 attribute 文でピン番号を指定する必要がある。詳しい方法については、付録 D.1.1で述べる。

そして、その回路の動作が正しければ、上のソフトウェアで論理合成を行なう。論理合成とは HDL によって記述された回路機能を AND や OR などの論理回路の形に変換することである。これにより HDL ファイルは EDIF(Electronic Design Interchange Format) ファイル (*.edf) に変換される。この EDIF ファイルはデジタル回路をテキストファイルで表したファイルで、回路を実際のデバイスに実装するための情報が含まれている。

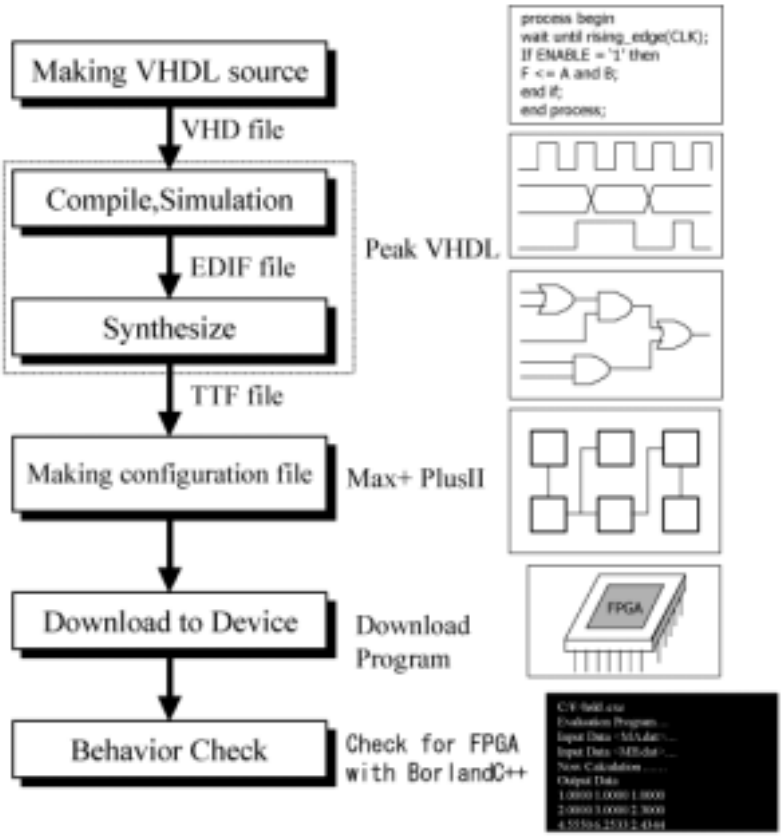


図 3.1: VHDL によるデジタル回路設計の流れ[1]

[1]ファイル名: ./fig/flow1.eps

次に、 MaxPlusII というソフトウェアを用いて、 FPGA に実装できるように配置配線を行なう。配置配線とは、実装する FPGA に最も最適なゲートやアレイなどの回路要素を決めることである。これにより、 EDIF ファイルからコンフィグレーション (配置配線) ファイルである TTF(Tabular Text File) ファイルを生成する。この TTF ファイルは 0 から 255 の数値とカンマで作られており、これを FPGA にダウンロードする。そして、この FPGA で機能の動作検証をする。もしこの動作が正しくなければ、その FPGA にあった動作に変え、実際に正しく動作するまで検証し直す。

3.3 FPGA 評価基板

3.3.1 評価基板の概要

本研究で使用している FPGA の評価基板を図 3.2に示す。この基板は 2 年前に松尾により製作された。この基板を使い、実装検証を行なう。

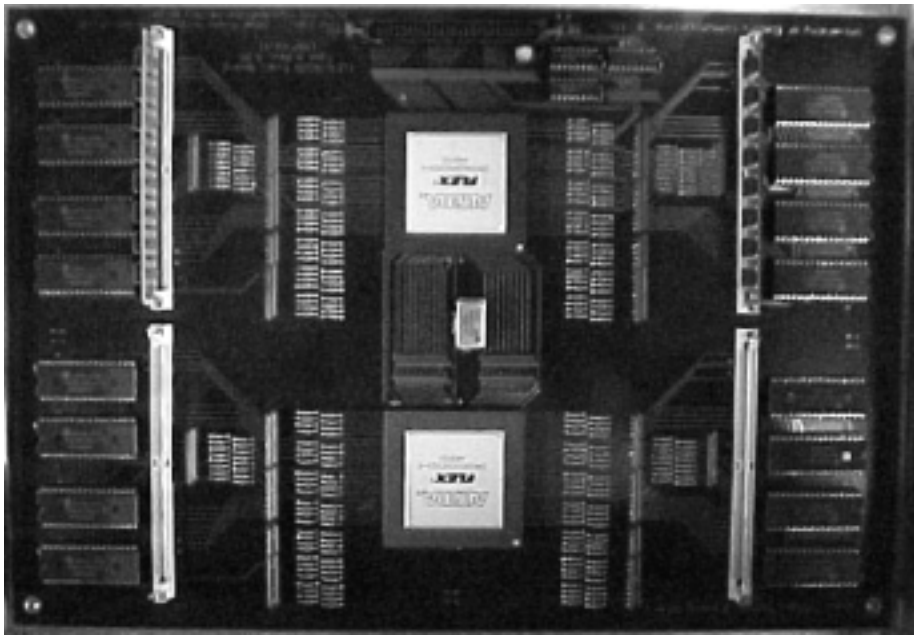


図 3.2: FPGA 評価基板の外観 (松尾: 卒業論文 [3])[2]

この基板には ALTERA 社の FPGA である FLEX10K シリーズの EPF10K100 を 2 個使用している。図の中央にある 2 つの LSI がその FPGA である。この FPGA のゲート容量は 10 万ゲートで、ロジックセル数は 4992 個、ピン数は 503 ピン (ユーザー使

[2]ファイル名: ./fig/board1.eps

用可能 406 ピン) を持つ。この 2 つの FPGA があれば本研究で使用している単精度の浮動小数点演算器やハウスホルダ変換アルゴリズムを実装するのに十分といえる。

そして、それぞれの FPGA の左右両側にはそれぞれ 1MbitSRAM が 4 個、 72 ピン SIMM の DRAM が 1 個、計 16 個の SRAM と 4 個の DRAM が搭載されている。この SRAM、 DRAM を使用することで、片側で 512kByte(SRAM)、 16MByte(DRAM) のデータを記憶することが可能である。

他にも、 2 つの FPGA の通信には FPGA 間に接続されている 56bit のデータバス、FPGA のシステムクロックを決めることができるオシレータ用のソケット (2 つの FPGA は同期)、そして、この基板と外部とのデータの通信を行なうインターフェース部分には、上部にある 50 ピンのコネクタを用いる。このコネクタと PPI8255 インターフェースボードを 50 ピンケーブルで接続することにより、 PC と FPGA 間でデータ通信を行なうことができる。

3.3.2 評価基板を用いた計算システム

先に述べた評価基板のブロック図を図 3.3に示す。図に書かれている数値は、この基板での通信できる bit 数である。

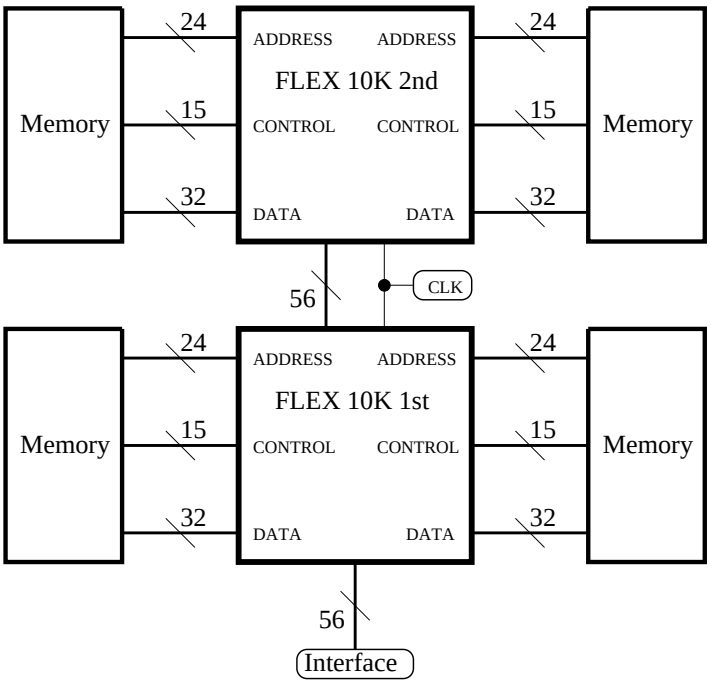


図 3.3: 評価基板のブロック図[3]

[3]ファイル名: ./fig/block.eps

このうち各 FPGA とメモリを結んでいる配線に関しては、以下の表 3.1のように指定されている。そして、この FPGA で配置されているピン番号を表??で指定している。このピン番号は FLEX10K 1st, 2nd ともに共通である。

DATA と ADDRESS ピンは SRAM、 DRAM ともに共通となっている。また、 ADDRESS ピンに関しては、 SRAM は下位 17bit、 DRAM は下位 12bit を使用している。

ブロック	信号名	用途	メモリ	方向	ピン数
DATA		データ通信	SRAM,DRAM	inout	32
ADDRESS		アドレス指定	SRAM,DRAM	out	24
CONTROL	$\overline{SCS}$	チップセレクト	SRAM	out	4
CONTROL	$\overline{SOE}$	メモリ読込命令	SRAM	out	1
CONTROL	$\overline{SWE}$	メモリ書込命令	SRAM	out	1
CONTROL	$\overline{DWE}$	メモリ書込命令	DRAM	out	1
CONTROL	$\overline{DRAS}$	行アドレス指定命令	DRAM	out	4
CONTROL	$\overline{DCAS}$	列アドレス指定命令	DRAM	out	4

表 3.1: FPGA とメモリとのピン配線

信号名	ピン番号 (左)	ピン番号 (右)
DATA	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	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
ADDRESS	T6,W7,Y6,AA7,AB6,AC7,AD6,AE7,AF6, AJ7,AK6,AL7,AM6,AN7,AP6,AR7,AT6	T38,W37,Y38,AA37,AB38,AC37,AD38,AE37,AF38, AJ37,AK38,AL37,AM38,AN37,AP38,AR37,AT38
$\overline{SCS}$	AG5,AH4,AJ5,AM4	AG39,AH40,AJ39,AM40
$\overline{SOE}$	AP4	AP40
$\overline{SWE}$	AN5	AN39
$\overline{DWE}$	AF4	AF40
$\overline{DRAS}$	P4,R5,T4,U5	P40,R39,T40,U39
$\overline{DCAS}$	Y4,AA5,AB4,AC5	Y40,AA39,AB40,AC39

表 3.2: FPGA のピン番号 (メモリ)

PC との通信を行なうための 50 ピンコネクタは表 3.3のようなポートに分かれている。ここでは、 PC と FPGA との制御の連携ができるように、 B ポートの上位を出力、下位を入力に使用する。 C ポートの 10bit および D ポートは PC からのインターフェースボードの制御に使われているため使用できない。さらに、 FPGA でのこれらのポートのピン番号を表に加えておく。このピン番号は 50 ピンコネクタが接続されている FLEX10K 1st にのみ適用される。

このインターフェースボードのデータの通信方法に関しては、付録 C.6に加えておいた。

また、FPGA に供給するクロックのピン番号もこの表に加えておく。ただし、これに関しては、2 つの FPGA で違う番号を使用しているので、注意しなければならない。

信号名	用途	方向	ピン数	ピン番号	
A	データ通信	inout	16	BC23,BB24,BC25,BB26,BC27,BB28,BC29,BB30, BC31,BB32,BC33,BB34,BC35,BB36,BC37,BB38	
BH	PC 制御	out	8	BC5,BB6,BC7,BB8,BC9,BB10,BC11,BB12	
BL	FPGA 制御	in	8	BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20	
CH	インターフェース制御	使用不可	10	AV18,AU19,AV20,AU21,AV22,AV28,AU29,AV30,AU31,AV32	
CL	未使用	in,out	6	AU23,AV24,AU25,AU33,AV34,AU35	
D	インターフェース制御	使用不可	8	AV7,AU8,AV9,AU11,AV12,AU13,AV14,AU15	
OBF	ハンドシェーク制御	in	2	AV18,AV28	
ACK	ハンドシェーク制御	out	2	AU19,AU29	
STB	ハンドシェーク制御	out	2	AU21,AU31	
IBF	ハンドシェーク制御	in	2	AV20,AV30	
CLK	クロック	in	1	D22(FLEX10K 1st)	AY22(FLEX10K 2nd)

表 3.3: FPGA とインターフェースとのピン配線

そして、2 つの FPGA の通信に使われる 56bit のデータバスについては、FPGA のピンによる指定はなく、自由に設計できる。本研究でのデータバスの使用方針については、次章で述べることにする。

第 4 章

集積回路の設計方針と設計モジュール

4.1 集積回路設計でも従来の問題点

本研究を行なう前に、行列の固有値および、固有ベクトルを求める計算機システムの構築が行なわれた。そこではFPGA 間のデータ通信、データ待ちを抑えるような設計思想を持つシステムの構築が行なわれた。

本研究では、行列データの記憶容量を増やすために、記憶装置をSRAM からDRAM に変更するため、データの待ち時間が増える可能性がある。そのため、その待ち時間も抑えるように設計する必要がある。また、行列の次数が増えるため、その分計算が正しくない結果を出したり、途中で停止することがある。したがって、正しい結果が出るように、余裕を持った計算、通信ができるようなシステムを設計する。

4.2 システム改良方針

4.2.1 メモリ

メモリをSRAM からDRAM に変更するときの改良方針を述べる。この評価基板で使われているSRAM とDRAM を比較すると、表4.1のような違いがあげられる。

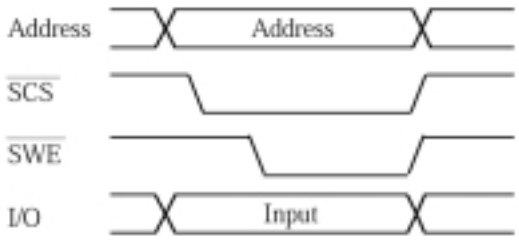
	SRAM	DRAM
アクセス速度	数 ns 以下	60ns
アクセス手続	1,2 回	6 回
データ容量 (32bit 当り)	512kByte	16MByte
記憶できる行列の次数	256 次	2048 次
リフレッシュ	不要	必要

表 4.1: SRAM と DRAM の比較

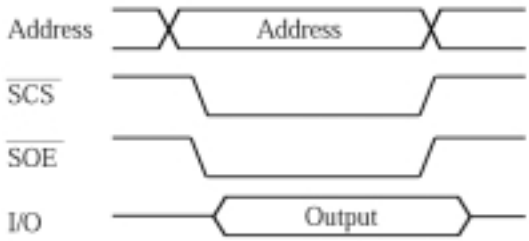
アクセス速度については SRAM の方が高速だが、この FPGA のシステムクロック (4MHz ～ 20MHz) の範囲では問題にはならない。しかし、アクセスの手続きについては、 SRAM はアドレス指定とライトイネーブルもしくはアウトプットイネーブルを設定すれば、メモリの読み書きを始めるのに対し、 DRAM では行列の 2 つのアドレス設定とそれぞれのアドレス指定命令を出力しなければならず、データのアクセスとメモリの読み書きサイクルの終了を含め、最低 6 回の手続きが必要となる。

しかし、アクセスの手続きが複雑な反面、 DRAM では使えるアドレスの空間が広く、少ないピン数でも大容量のデータが記憶できる。そのため、 DRAM のデータ容量は SRAM の 32 倍で、記憶可能な行列の次数は 2048 次と、 SRAM の 8 倍となっている。

DRAM を使用するにあたり、問題点として、リフレッシュを行なう必要がある。このリフレッシュとは DRAM に記憶してあるデータが消去しないように、 15.6μs 以内に再書き込みを行なうことである。そのため、 DRAM コントローラを設計するために、図 4.1のような動作ができるようにする必要がある。

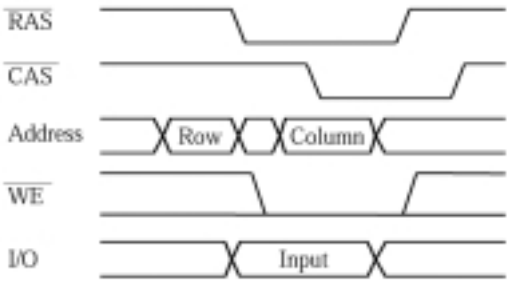


a. 書き込みサイクル (SRAM)

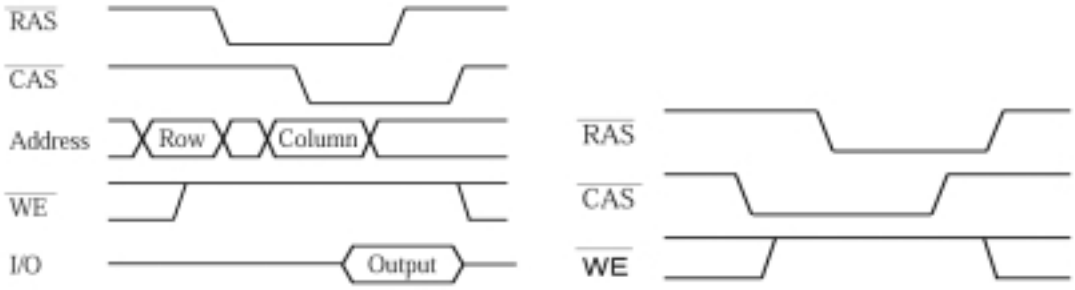


b. 読み込みサイクル (SRAM)

c. 連続読み込みサイクル (SRAM)



d. 書き込みサイクル (DRAM)



e. 読み込みサイクル (DRAM) f. リフレッシュサイクル (DRAM)

図 4.1: 各メモリの動作[1]

次に、この図 4.1のサイクルの動作について述べる。

a. 書き込みサイクル (SRAM)
SRAM にデータを書き込む場合には、アドレスをレジスタに格納してから、$\overline{SCS}$ を立ち下げる。これにより SRAM のアドレスが指定される。そして、しばらくしてから $\overline{SWE}$ を立ち下げる。これにより SRAM にデータが書き込まれる。

b. 読み込みサイクル (SRAM)
SRAM からデータを読み込ませる場合には、書き込みサイクルと同様に、$\overline{SCS}$ を立ち下げてアドレスを指定する。これと同時に $\overline{SOE}$ を立ち下げることで、SRAM からデータを読み込ませることができる。

c. 連続読み込みサイクル (SRAM)
上の読み込みサイクルの時に、アドレスを別の番地に変えると、読み込まれるデータもその指定した番地のものに変わる。このアドレスの部分をカウンタで 1 クロックずつに変えてゆけば、データも連続的に読み込むことができる。

d. 書き込みサイクル (DRAM)
DRAM の場合には、行および列の 2 つのアドレスを指定する必要がある。まず、行および列アドレスをそれぞれのレジスタに格納する。そして、$\overline{RAS}$ を立ち下げて行アドレスを指定する。次に $\overline{CAS}$ を立ち下げて列アドレスを指定する。また、この $\overline{CAS}$ を立ち下げる前に、ライトイネーブル信号である $\overline{WE}$ を立ち下げる必要がある。これにより $\overline{CAS}$ が立ち下がったときに DRAM にデータが書き込まれる。

[1]ファイル名: a.:./fig/SRAM-write.eps, b.:./fig/SRAM-read.eps,
c.:./fig/SRAM-contread.eps, d.:./fig/DRAM-write.eps,
e.:./fig/DRAM-read.eps, f.:./fig/DRAM-refresh.eps

e. 読み込みサイクル (DRAM)
DRAM からデータを読み込ませる場合には、書き込みサイクルと同様に、$\overline{RAS}$、$\overline{CAS}$ を順に立ち下げる。また、データのポートはハイインピーダンス状態にしなければならない。$\overline{CAS}$ が立ち下がってから、15ns 後に DRAM からデータが読み込まれる。

f. リフレッシュサイクル
DRAM のデータの保持に必要なリフレッシュは、$\overline{CAS}$ ビフォア $\overline{RAS}$ リフレッシュで行なわれる。この方法だとリフレッシュアドレスカウンタが DRAM 側にあるので、それを FPGA で設ける必要がない分、ゲート数を減らすことができる。動作としては、$\overline{RAS}$ の前に $\overline{CAS}$ を立ち下げればよい。

4.2.2 積和器

この行列計算において、計算の多くを占めている加算と乗算については、メモリコントローラとともに一つの FPGA に実装されている。そして、行列の固有値および固有ベクトルを求めるのに、積和計算がよく使われる。この計算に使われる積和器でのデータの流れを図 4.2に示す。この積和器は昨年、山岡によって設計されたものを、より正しい計算ができるように改良したものである。

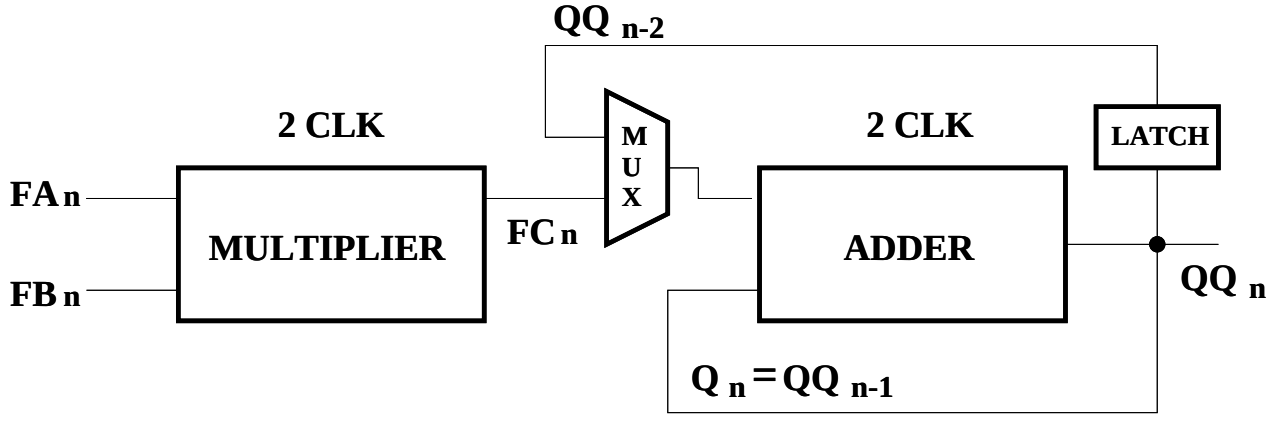


図 4.2: 積和器[2]

[2]ファイル名: ./fig/inpro.eps

この積和器では、SRAM の連続読み込みサイクルを使用して、クロック毎にデータを読み込む、この動作を SRAM の左右両側から同時に読み込み、2 つのデータ (FA,FB) を乗算器に入力する。そして、その結果と今までの計算結果を加算器に入力して、結果 QQ を出力する。

この乗算器と加算器で計算結果を得るには 2 クロックを必要とする。そのため、奇数番目に入力したデータと偶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 に出力される。そして、乗算器の入力が終わると、その 2 クロック後の加算器の入力 FC には何もない状態となる。そのとき、ラッチによりレジスタに格納してあった、前のクロックの加算器の計算結果をマルチプレクサによって選択して、加算器へ入力する。そして、その 2 クロック後の加算器の結果が積和結果となる。

しかし、乗算器の仮数部の乗算の部分で、

```
signal   MA2,MB2 : std_logic_vector(23 downto 0);
variable TMQ1    : std_logic_vector(47 downto 0);
TMQ1 := MA2 * MB2;
```

という VHDL の記述があった。以前はこれを 1 クロックで行なっており、この 24bit の乗算だけでも多くのゲート容量を使う、さらに、この 2 つの 24bit データのうち、'1' が多く含まれているとこの半クロックのうちに計算を終わらせることができず、そこで全体の計算が停止してしまうことがある。

そこで、この仮数部の乗算の計算を、2 つのグループに分けて計算を行なうことにした。下の VHDL より、1 クロック毎に乗算するグループを切り替え、この乗算部分を 2 クロックでできるようにした。

```
signal   MA2,MB2   : std_logic_vector(23 downto 0);
variable TMQ1,TMQ2 : std_logic_vector(47 downto 0);
if DCNT(0) = '1' then
      TMQ1 <= MA2 * MB2;
else
      TMQ2 <= MA2 * MB2;
end if;
```

また、加算器の入力の部分で、同期を取りやすくするため、乗算器全体の所要クロック数を 2 クロックから 4 クロックに増やした。これにより、加算器自体は何の改良も

せずに済む。また、積和器としては、乗算器の入力終了から 2 クロック後にマルチプレクサを切り替えていたところを、 4 クロック後にするだけでいいようにできる。

4.2.3 ハウスホルダ変換

ハウスホルダ法の設計方針として、昨年設計されたシステムを受け継いで、記憶装置であるメモリを SRAM から DRAM 中心に置き換え、積和器を先に述べた、正しい結果が出る積和器に置き換え、設計し直した。

先のシステムでは、このハウスホルダ変換のシステムをすべて、 1 つの FPGA に実装しようとしても、 FPGA の持つロジックセル数の上限を上回るので、 2 つの FPGA に分散させてシステムを構成している。

このシステムの実装構成は、インターフェースに近い FLEX 1st には除算器、平方根計算器、ハウスホルダ法を、もう一方の FLEX 2nd には積和器とメモリコントローラを実装している。また、 FLEX 1st のハウスホルダ法のアルゴリズムを一度にすべて実装することができないので、 4 つのアルゴリズムを 1 つずつ実装し、計算結果は SRAM に記憶しておき、計算を進めていく。このとき、 SRAM に記憶してある行列データは、評価基板の電源を切らない限り、つまり、インターフェースコネクタをはずさない限り、 FPGA 2nd に搭載されているデータは消去しない。

このシステムを改良して、 DRAM に行列データを記憶できるようにし、積和器も正しい結果の出るもの安定動作できるようにした。そのため、その部分におけるシステムの同期や積和器に入力に関する部分について、設計を変更した。

まず、ハウスホルダ変換について、記憶装置を SRAM から DRAM に変えた場合の利用方法について述べる。

DRAM はおもに行列データの記憶に使う。そして、 SRAM は三重対角化のときに行なわれる計算の記憶に使われる。また、行列の行または列ベクトル要素を DRAM から取り出して、一時的に SRAM に格納することで、積和器における計算の同期を取りやすくした。つまり、 SRAM を FPGA の外部キャッシュとして動作させることで、 DRAM を用いての行列計算でも、 SRAM を用いたときに近い性能で計算が可能である。

次に、ハウホルダ変換における DRAM および SRAM の動作を簡単に述べる。

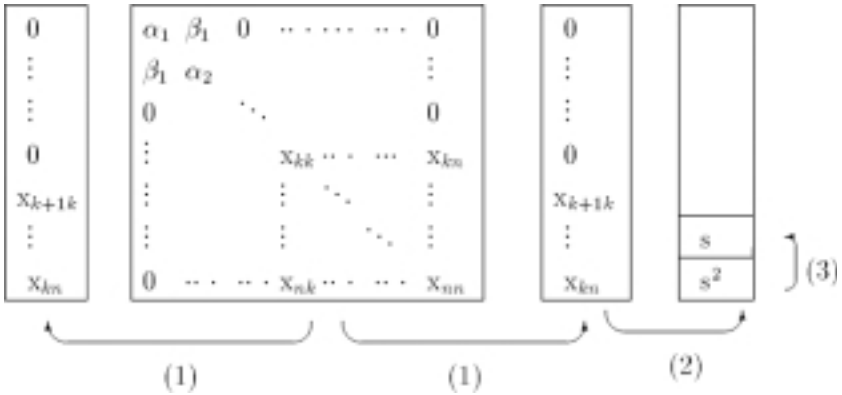


図 4.3: 行列の対角化における DRAM と SRAM の動作 (1)[3]

ここでは例として、ハウスホルダ変換のときの計算をあげる。この図 4.3において、中央の長方形が DRAM、横の縦に長い長方形が SRAM(1 つが 12bit) と見なす。

まず最初に、左右の DRAM に計算の対称となる行列を記憶させる。次に (1.5.18) にある s を求める。このとき、 DRAM からデータを読み込んで計算するのではなく、DRAM から計算の対象となるデータだけを SRAM に移し変えて、そこから FPGA にデータを読み込ませて計算する。そして、計算結果は SRAM の他のアドレスに書き込む。この方法だと、この変換の計算で重要となる ω の計算が高速にできる。

まず (1) のように計算の対象となる列ベクトルを DRAM から読み込み、左右の SRAM へコピーする。そして、 (2) のようにその左右の SRAM から同じアドレスのデータを 1 番地ずつ読み込み、積和器で計算する。その結果が s^2 となる。最後に、平方根計算器で s を計算して、結果を SRAM に格納する。したがって、行列の計算のほとんどが、行ベクトルまたは列ベクトルと、以前に計算した結果が対象となるので、それらを SRAM に格納して読み込ませれば、高速にかつ高次の行列の計算が行なえる。

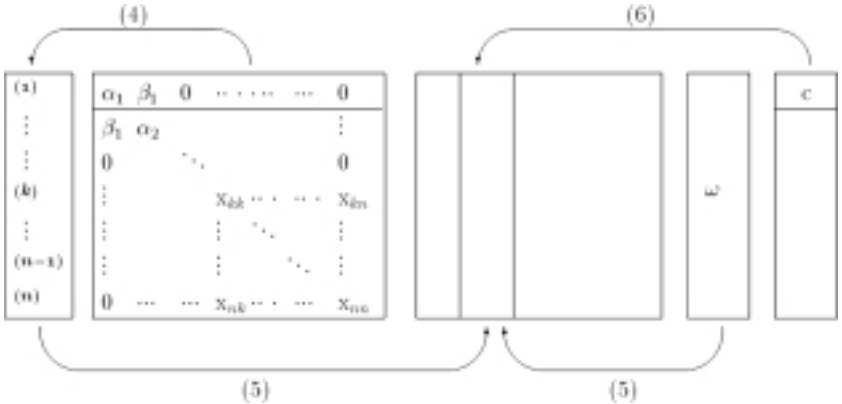


図 4.4: 行列の対角化における DRAM と SRAM の動作 (2)[4]

[3]ファイル名: ./fig/matrix1.eps

[4]ファイル名: ./fig/matrix2.eps

次に、計算対象に行列が含まれている場合の計算を図 4.4に示す。この図では (1.5.13) の行列 $\boldsymbol{A}$ とベクトル ω の掛け算をするところである。これ以外には、逆反復法の $\boldsymbol{L}Ux$ の計算で使っている。

まず、(4) のように、行列の列ベクトルを DRAM から読み込み、行アドレスと同じアドレスへ、左側の SRAM に書き込む。そして、右側の SRAM に格納してあるベクトル ω と、その SRAM のデータを積和器で計算する (5)。その結果を右側の DRAM に一つずつ格納する。そしてこの計算が終わったら、右側の SRAM に格納してあるスカラ量 c と、 DRAM に格納してある計算結果を乗算器に入力し、その計算結果 p は右側の DRAM に格納し直す (6)。これを行列データすべてにすればよい。つまりこの DRAM と SRAM の動作を必要とする計算は、行列とベクトルの掛け算のときである。

以上の設計方針に基づいて、この行列専用計算機の計算モジュールを設計する。

4.3 設計モジュール

ここでは、昨年山岡 [5] が設計したメモリコントローラや浮動小数点演算器について、高次数での動作が可能なように設計し直し、新たにモジュールを追加した。

4.3.1 メモリコントローラ (SRAM,DRAM)

行列計算などにおいて、多くのデータを扱うにはメモリが必要である。昨年は SRAM をメモリコントローラとして、データの記憶が可能になった。そして、本研究ではそのメモリコントローラに DRAM も追加し、さらに大容量のデータを記憶できるようにした。つまり、 SRAM および DRAM を使用するためのメモリコントローラを設計した。

このコントローラの entity の構成を表 4.2に、制御信号の構成を表 4.3に、 VHDL ソースを付録 1 に示す。

信号名	用途	対象	方向	型
CLK	クロック	共通	in	std_logic
RESET	リセット	共通	in	std_logic
ADRS	アドレスバス	共通	out	std_logic_vector(16 downto 0)
SRAM_ADRS_BUF	アドレスレジスタ	SRAM	in	std_logic_vector(16 downto 0)
ROW_ADRS_BUF	行アドレスレジスタ	DRAM	in	std_logic_vector(11 downto 0)
COL_ADRS_BUF	列アドレスレジスタ	DRAM	in	std_logic_vector(11 downto 0)
DATA	データバス	共通	inout	std_logic_vector(31 downto 0)
WR_DATA	書き込みデータ	共通	in	std_logic_vector(31 downto 0)
RD_DATA	読み込みデータ	共通	out	std_logic_vector(31 downto 0)
SCS	チップセレクト	SRAM	out	std_logic_vector(3 downto 0)
SOE	アウトプットイネーブル	SRAM	out	std_logic
SWE	ライトイネーブル	SRAM	out	std_logic
DWE	ライトイネーブル	DRAM	out	std_logic
DRAS	行アドレス指定	DRAM	out	std_logic_vector(3 downto 0)
DCAS	ライトイネーブル	DRAM	out	std_logic_vector(3 downto 0)
MEM_STATE_SEL	動作選択	共通	in	std_logic_vector(2 downto 0)
WR_CYCLE	ライト終了	共通	out	std_logic
RD_CYCLE	リード終了	共通	out	std_logic
RF_CYCLE	リフレッシュ終了	DRAM	out	std_logic

表 4.2: entity の構成

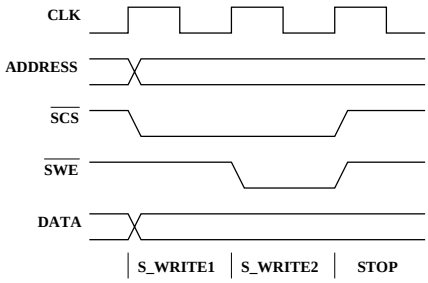
信号名	用途	ステート数
S_WRITE_CYCLE	SRAM 書き込みサイクル	3
S_READ_CYCLE	SRAM 読み込みサイクル	3
CONT_READ_CYCLE	SRAM 連続読み込みサイクル	1
D_WRITE_CYCLE	DRAM 書き込みサイクル	6
D_READ_CYCLE	DRAM 読み込みサイクル	6
D_REF_CYCLE	DRAM リフレッシュサイクル	6
D_RD_WR_CYCLE	キャッシュ書き込みサイクル	10

表 4.3: 制御信号

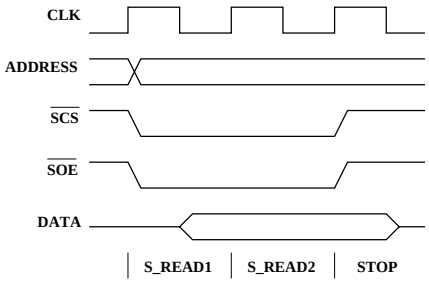
この entity は昨年設計された SRAM コントローラに、 DRAM の入出力に関わる部分を追加した。そして制御信号は DRAM に関するサイクルを 4 つ追加した。 VHDL ソースはステートマシン、ステート制御、アドレス指定の 3 つの process から構成されている。そして、ステートマシンによりシステムクロックに同期して動作するようになっている。

このコントローラのステートは表 4.3より、以上のようなステート数で構成されている。 CONT_READ_CYCLE は 1 クロックごとにアドレスを与え、そのたびに DATA を SRAM から読み込んでいる。そして、キャッシュ書き込みサイクルは DRAM の読み込みサイクルと SRAM の書き込みサイクルを連続して行なっており、 DRAM から読み込んだデータは FPGA のレジスタに一時的に格納し、 SRAM へ書き込まれる。

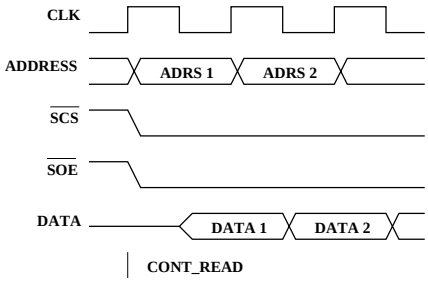
図 4.5に各サイクルのタイミングを示す。



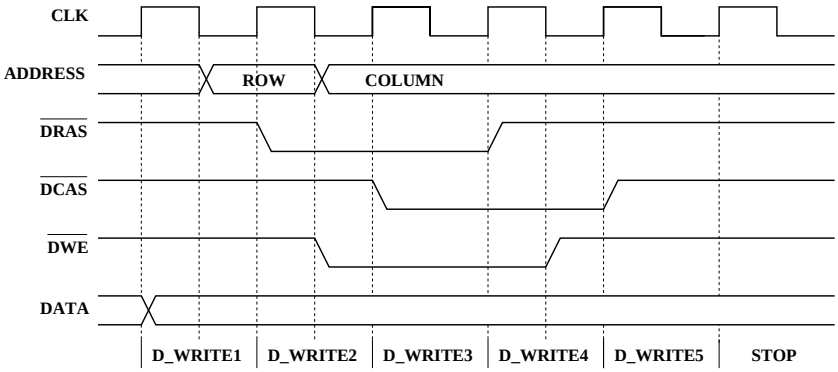
a. 書き込みサイクル (SRAM)



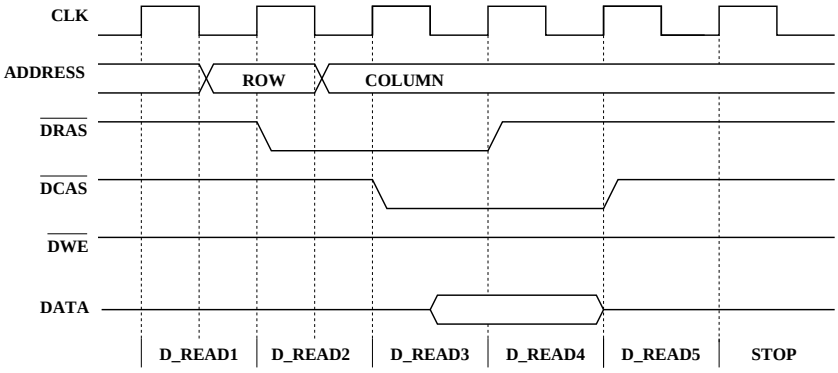
b. 読み込みサイクル (SRAM)



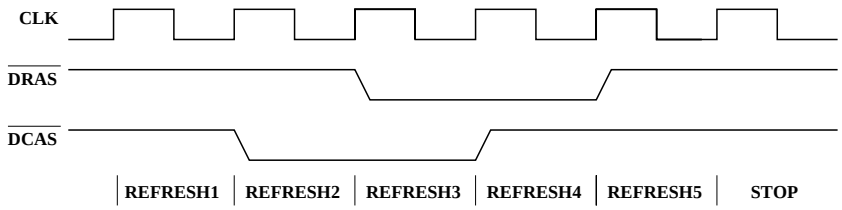
c. 連続読み込みサイクル (SRAM)



d. 書き込みサイクル (DRAM)



e. 読み込みサイクル (DRAM)



f. リフレッシュサイクル (DRAM)

図 4.5: 各サイクルのタイミング (2)[5]

以下に各サイクルのステート毎の動作について説明する。

a. 書き込みサイクル (SRAM)
ステート S_WRITE1 で $\overline{SCS}$ を立ち下げてアドレスを指定し、ステート S_WRITE2 で $\overline{SWE}$ を立ち下げて、 SRAM にデータを書き込む。

b. 読み込みサイクル (SRAM)
ステート S_READ1 で $\overline{SCS}$ を立ち下げてアドレスを指定し、ステート S_READ2 で $\overline{SOE}$ を立ち下げ、 SRAM からデータを読み込む。

c. 連続読み込みサイクル (SRAM)
この場合はステートは 1 つ CONT_READ のみ。アドレスを別の番地に変えることでデータも連続的に読み込む。

d. 書き込みサイクル (DRAM)
ステート D_WRITE1 のクロックの立ち下がりで行アドレスを指定する。そして、ステート D_WRITE2 のクロックの立ち上がりで $\overline{RAS}$ を立ち下げて行アドレスを指定する。このステートの立ち下がりでは列アドレスを指定し、書き込み許可を与えるために $\overline{WE}$ を立ち下げる。次のステート D_WRITE3 では $\overline{CAS}$ を立ち下げる。て列アドレスを指定する。 DRAM へデータを書き込んだ後は、ステート D_WRITE4 で $\overline{RAS}$ を立ち上げ、クロックの立ち下がりで $\overline{WE}$ を立ち上げる。、そして、ステート D_WRITE5 で $\overline{CAS}$ を立ち上げる。

e. 読み込みサイクル (DRAM)
$\overline{RAS}$ および $\overline{CAS}$ のステート動作に関しては、上の D_WRITE を D_READ に

[5]ファイル名：a:./fig/write.eps,b:./fig/read.eps, c:./fig/contread.eps, d:./fig/Dwrite.eps, e:./fig/Dread.eps, f:./fig/Drefresh.eps

置き換えれば良い。$\overline{WE}$ は立ち上げたままにする。データはステート D_READ4 の時に DRAM から FPGA のレジスタへ読み込ませる。

f. リフレッシュサイクル
ステート REFRESH2 で $\overline{CAS}$ を立ち下げ、 REFRESH3 で $\overline{RAS}$ を立ち下げる。これによりリフレッシュで行なわれる。各信号の立ち上げは $\overline{CAS}$ が REFRESH4 で、 $\overline{RAS}$ が REFRESH5 のときに行なう。

また、リフレッシュサイクルについては、$\overline{CAS}$ ビフォア $\overline{RAS}$ リフレッシュを使用している。そのため、 ADDRESS には何もデータを入力する必要がなく、 $\overline{RAS}$ と $\overline{CAS}$ だけを動作させればよい。このリフレッシュサイクルを 15.6μs 毎に DRAM に与えるように設計する。例えば、システムクロックが 4MHz のときは 64 クロック毎に、 2MHz のときは 32 クロック毎、 10MHz のときは 156 クロック毎にリフレッシュできるようにする。

4.3.2 浮動小数点演算器

浮動小数点演算器については、前章で述べたとおり、乗算器が動作しないことがあったので、乗算器の中の仮数部の乗算の部分について改良し、正常に動作できるようにした。乗算器以外については、特に動作に問題がないので、昨年山岡が設計したものをそのまま使用した。

加算器

entity の構成を図 4.4に、 VHDL ソースを付録 B.2.1に示す。

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)

表 4.4: entity の構成

VHDL ソースでは図 1.5の上から順に 3 ステージで構成されており、演算動作は組合わせ回路で記述されており、ステージ間のラッチは process ごとにシステムクロックに同期して行なうように記述されている。

乗算器

entity の構成を図 4.5に、 VHDL ソースを付録 B.2.2に示す。

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)

表 4.5: entity の構成

VHDL ソースでは図 1.6の上から順に 3 ステージで構成されている。第 2 ステージの仮数部の乗算の計算に関して、 2 クロックで動作させるため、 1 クロック目で乗算を行ない、 2 クロック目で不要な下位ビット 22bit を切り捨てる。また、 2 クロックでこの乗算を行なうために、この乗算要素を偶数クロックと奇数クロックの 2 つの論理に分けた。また、指数部の加算に関しては、 1 クロック目で加算を行ない、 2 クロック目は何もせず、ラッチで仮数部の乗算と同期を行なえるようにしてある。この乗算器のクロックによる動作の流れを図 4.6に示す。

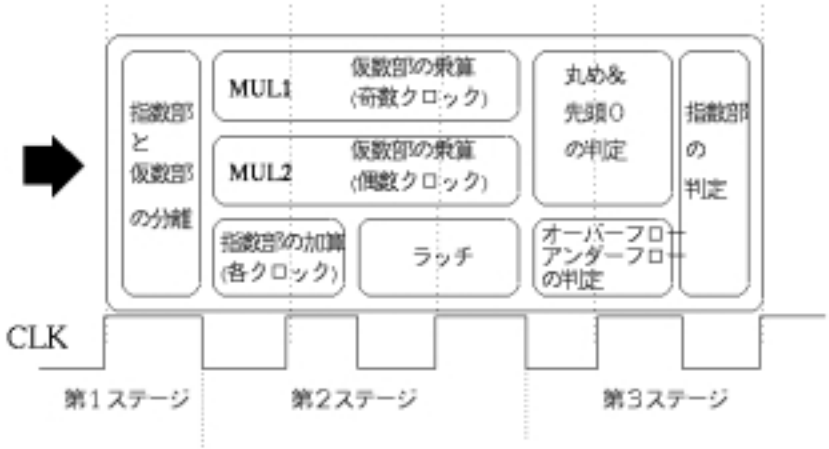


図 4.6: 改良した乗算器のクロックとの同期[6]

[6]ファイル名: ./fig/multiple.eps

除算器

entity の構成を図 4.6に、 VHDL ソースを付録??に示す。

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)

表 4.6: entity の構成

VHDL ソースでは図 1.7の上から順に 3 ステージで構成されている。 VHDL ソースの構成は加算器のものと同じで、ステージ間のラッチは process ごとにシステムクロックに同期して行なうように記述されている。

平方根計算器

entity の構成を図 4.7に、 VHDL ソースを付録 B.2.4に示す。

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)

表 4.7: entity の構成

VHDL ソースは図 1.8より、入力をを FA、出力を Q として、 function 形式で記述した。最初の if 文の部分が指数部の右 1 ビットシフトで、その後の for-loop 文が仮数部の開平算となっている。

この平方根計算器では計算結果は正の数として出力する。現在のシステムでは、虚数に対応していないため、 $\sqrt{-1}$ などは対応していない。負の数の平方根では、その絶対値の平方根として計算する。

4.3.3 積和器

メモリコントローラ、加算器および乗算器を用いて積和器を設計した。この積和器は昨年山岡が、 FLEX2nd に実装するモジュールとして設計したものを、 DRAM コントローラの追加、正常に動作する乗算器のために、その同期が取れるように改良した。 entity の構成を表 4.9に、 VHDL ソースを付録 B.3.1に示す。

この FPGA 間のデータバス (56bit) の配分は、データの通信 (DATA_BUS) に 32bit、アドレスの通信 (ADRS_BUS) に 16bit、 FPGA 間の制御 (CTRL_BUS) に 6bit、 FLEX1st

が命令した計算が終わったことを知らせる信号 (CALC_DONE) とデータバスの方向制御 (OE_ALU) に 1bit ずつ使用する。この場合のデータバスのピン配置を表 4.8に示す。方向は FLEX 2nd からの方向とする。

信号名	用途	方向	ピン数	ピン番号
DATA_BUS	データ通信	inout	32	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
ADRS_BUS	アドレス通信	in	16	AV18,AU19,AV20,AU21,AV22,AU23,AV24,AU25, AV28,AU29,AV30,AU31,AV32,AU33,AV34,AU35
CTRL_BUS	FPGA 間制御	in	6	AV10,AU11,AV12,AU13,AV14,AU15
CALC_DONE	計算終了	out	1	AU9
OE_ALU	データバスの方向制御	in	1	AV8

表 4.8: FLEX 1st と FLEX 2nd との間のデータバスのピン配線

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(15 downto 0)
CTRL_BUS	FPGA 間のコントロールバス	in	std_logic_vector(5 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(15 downto 0)
R_SCS	右メモリのチップセレクト (SRAM)	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 (SRAM)	out	std_logic
R_SWE	右メモリのライトイネーブル (SRAM)	out	std_logic
R_DWE	右メモリのライトイネーブル (SRAM)	out	std_logic
R_DRAS	右メモリの行アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)
R_DCAS	右メモリの列アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)
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
L_DWE	左メモリのライトイネーブル (SRAM)	out	std_logic
L_DRAS	左メモリの行アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)
L_DCAS	左メモリの列アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)

表 4.9: entity の構成

そして、メモリや FLEX1st から送られているデータの演算やメモリの読み書きの動作もこの積和器で行なう。この FLEX1st からの動作はすべて FLEX1st から制御される。 FLEX1st からの制御信号を表 4.10、表 4.11に示す。表 4.11は今回追加した、DRAM に関する制御信号である。

この信号は FLEX1st から FLEX2nd へ FPGA 間の 6bit のコントロールバス (CTRL_BUS) で伝えることにより、積和器が表 4.10およびの表 4.11の動作を行なうことができる。そして、この動作が終了したら、 FLEX2nd から計算終了信号 (CALC_DONE) を伝える。

信号名	用途
R_MEM_WR	右 SRAM の書き込み
R_MEM_RD	右 SRAM の読み込み
L_MEM_WR	左 SRAM の書き込み
L_MEM_RD	左 SRAM の読み込み
LR_MEM_WR	左右 SRAM 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_MEM_R	積和結果を右 SRAM へ転送
INPRO_MEM_L	積和結果を左 SRAM へ転送
INPRO_MEM_LR	積和結果を左右 SRAM へ転送
FF_MUL	FPGA データの乗算
FR_MEM_MUL	FPGA, 右 SRAM データの乗算
FL_MEM_MUL	FPGA, 左 SRAM データの乗算
RR_MEM_MUL	右 SRAM データの乗算
LL_MEM_MUL	左 SRAM データの乗算
LR_MEM_MUL	左右 SRAM データの乗算
FF_ADD	FPGA データの加算
FR_MEM_ADD	FPGA, 右 SRAM データの加算
FL_MEM_ADD	FPGA, 左 SRAM データの加算
RR_MEM_ADD	右 SRAM データの加算
LL_MEM_ADD	左 SRAM データの加算
LR_MEM_ADD	左右 SRAM データの加算
CALC_F	計算結果を FPGA へ転送
CALC_MEM_R	計算結果を右 SRAM へ転送
CALC_MEM_L	計算結果を左 SRAM へ転送
CALC_MEM_LR	計算結果を左右 SRAM へ転送

表 4.10: 制御信号

信号名	用途
LR_DRAM_WR	左右 DRAM の書き込み
L_DRAM_WR	左 DRAM の書き込み
R_DRAM_WR	右 DRAM の書き込み
LR_DRAM_RD	左右 DRAM の読み込み
L_DRAM_RD	左 DRAM の読み込み
R_DRAM_RD	右 DRAM の読み込み
INPRO_DRAM_R	積和結果を右 DRAM へ転送
INPRO_DRAM_L	積和結果を左 DRAM へ転送
FR_DRAM_MUL	FPGA, 右 DRAM データの乗算
FL_DRAM_MUL	FPGA, 左 DRAM データの乗算
LR_DRAM_MUL	左右 DRAM データの乗算
CALC_DRAM_R	計算結果を右 DRAM へ転送
CALC_DRAM_L	計算結果を左 DRAM へ転送
CALC_DRAM_LR	計算結果を左右 DRAM へ転送
RR_DRAM_MEM	右 DRAM から右 SRAM へデータ転送
LL_DRAM_MEM	左 DRAM から左 SRAM へデータ転送
LR_DRAM_MEM	左 DRAM から右 SRAM へデータ転送

表 4.11: 制御信号 (DRAM 追加分)

本研究では、 FPGA1st、 SRAM の制御だけでなく、 DRAM も加わるので、 SRAM を使うものは MEM、 DRAM を使うものは DRAM が信号名の 2 つ目のところに表記してある。

4.3.4 ハウスホルダ法

ハウスホルダ変換

FLEX2nd には積和器を実装するが、 FLEX1st にはハウスホルダ法のアルゴリズム、除算器、平方根計算機を実装する。アルゴリズムのステート構成を表 4.12に、 VHDL ソースを付録 B.3.2に示す。

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
HsS2LoopIni	変数の初期化
HsS2LoopBdy	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	$\boldsymbol{A\omega}$ 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	$\frac{\alpha \times c}{2}\boldsymbol{\omega}$ の計算
HsQCalc	$\boldsymbol{q}$ の計算
HsUxQtCalc	$\boldsymbol{\omega q}^T$ の計算
HsQxUtCalc	$\boldsymbol{q\omega}^T$ の計算
HsUxQtpQxUtCalc	$\boldsymbol{\omega q}^T + \boldsymbol{q\omega}^T$ の計算
HsACalc	$\boldsymbol{A}$ の計算
HsResRead	三重対角行列のメモリ読み込み

表 4.12: ステートの構成

ステートの構成については、ステートの内部 (DRAM が使えるところ) を改良し、構成自体は変更していない。

このハウスホルダ法でのステートの動作を説明する。 HsStop において、行列データを DRAM に書き込む。そして、 HsDimWrite でその行列の次元を SRAM に書き込む。 HsS2LoopIni で変数を初期化し、 HsS2LoopBdy で列ベクトルを DRAM から SRAM に転送し、積和器で s^2 を計算する。そして、 s^2 は FPGA のレジスタに格納する。

次に (2.2.8) の c を計算する。 HsSCalc で平方根計算器で s を計算し、 HsAkRead で DRAM から $a_{k+1,k}$ を読み込み、 HsSNorm でこれと s の符号を合わせ、 HsAkxSCalc で $a_{k+1,k} \times s$ の絶対値を計算する。そして、 HsS2AkxSCalc でその結果に s^2 を加算器で計算し、 HsCCalc で除算器を用いその計算結果の逆数をとり、 c を計算する。 c を SRAM に書き込み、 HsViceWrite で副対角成分を DRAM に書き込む。

次に (2.2.13) の $\boldsymbol{p}$ を計算する。 HsAkpSCalc で加算器を用いて $a_{k+1,k}+s$ を計算し、ω を求める。そしてその結果を SRAM に書き込む。次に HsAxUCalc で積和器を用いて、 $\boldsymbol{A\omega}$ を計算する。そして、 HsPCalc で $\boldsymbol{p} = c\boldsymbol{A\omega}$ を計算する。

そして、 (2.2.14) の $\boldsymbol{q}$ を計算する。まず HsAlphaCalc で α を計算する。 α は

$$\alpha = \boldsymbol{\omega}^T \boldsymbol{p} \tag{4.3.1}$$

そして HsAlphaxCCalc で $\alpha \times c$ を計算し、 HsAlphaxCd2Calc で結果を 2 で割って (結果の指数部を -1)、 HsAlphaxCd2xUCalc で $\boldsymbol{\omega}$ を掛け、 (2.2.14) の右辺第 2 項を求める。そして、 HsQCalc で $\boldsymbol{q}$ を計算する

その次は第 k 行および第 k 列の三重対角成分を計算し、 DRAM に書き込む。 (2.2.12) の右辺第 2 項以降の $\boldsymbol{\omega q}^T + \boldsymbol{q\omega}^T$ を計算する。それには HsUxQtCalc で第 2 項の $\boldsymbol{\omega q}^T$ を計算し、 HsQxUtCalc で第 3 項の $\boldsymbol{q\omega}^T$ を計算する。そして、 HsACalc で (2.2.12) により、 $\boldsymbol{A}$ を計算する。これらを求めるには、最初の 2 つの状態では乗算器、残りは加算器を用いる。

これを第 n 行第 n 列まで繰り返し、三重対角行列を計算し終えたら、 HsResRead で三重対角行列を DRAM からメモリ読み込みを行なう。

entity の構成を表 4.13に示す。 entity はこのハウスホルダ法の 4 つのアルゴリズムすべてに共通している。

信号名	用途	方向	型
CLK	クロック	in	std_logic
A	インターフェースポート A	inout	std_logic_vector(15 downto 0)
BL	インターフェースポート B 下位	in	std_logic_vector(7 downto 0)
BH	インターフェースポート 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	インターフェースポート 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	インターフェースの OBF	in	std_logic_vector(1 downto 0)
ACK	インターフェースの ACK	out	std_logic_vector(1 downto 0)
STB	インターフェースの STB	out	std_logic_vector(1 downto 0)
IBF	インターフェースの IBF	in	std_logic_vector(1 downto 0)

表 4.13: entity の構成

二分法

FLEX1st の実装構成は、除算器、二分法アルゴリズムである。アルゴリズムのステート構成を表 4.14に、 VHDL ソースを付録 B.3.3に示す。

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	$\frac{\beta_{k-1}^2}{g_{k-1}}$ の計算
BisQCalc	g_k の計算
BisQComp	g_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

表 4.14: ステートの構成

先のハウスホルダ変換を計算し終えたあと、三重対角化行列のデータは FLEX 2nd の DRAM に格納される。そして、この二分法のアルゴリズムは FLEX 1st にダウンロードされ、二分法の計算が行なわれる。

この二分法のステートの動作を説明する。 DisDimRead で行列の次元を SRAM から読み込む。そして固有値の存在範囲を決める。そのために、 BisAlphaRead、 BisBetaRead で対角要素 α_k と副対角要素 β_k、 β_{k-1} をメモリから読み込む。そして、 (2.2.27) の $\alpha_k + \beta_k + \beta_{k-1}$ を加算器を用いて計算する。そのために、 BisAlphapBetaCalc で $\alpha_k+\beta_k$ を計算し、 BisMaxCalc でさらに β_{k-1} を加え、 BisMaxComp でゲルシュゴーリンの最大値を比較する。 BisMaxSet で固有値存在範囲を設定する。

次に中点 c を計算する。 BisApBCalc で a+b を計算し、 BisCCalc でそれを 2 で割り、 c を求める。

次に、 g_k の計算を行なう。この計算を行なうためには (2.2.29) を計算する。まず BisAlphamCCalc を加算器を用いて $\alpha_k - c$ を計算する。これに -1 掛けたものが、 (2.2.29) の第 1 式 g_1 となる。

そして、 (2.2.29) の第 2 式を計算する。まずこの式の右辺第 3 項にあたる $\frac{\beta_{k-1}^2}{g_{k-1}}$ を計算する。 BisBeta2Calc ですでにレジスタに格納してある β_{k-1} を使い、 β_{k-1}^2 を乗算器を用いて計算する。そして BisBeta2dQCalc で $\frac{\beta_{k-1}^2}{g_{k-1}}$ を除算器を用いて計算する。そして、 BisQCalc で加算器を用いて $\alpha_k - c$ から $\frac{\beta_{k-1}^2}{g_{k-1}}$ の差をとり、 g_k を計算する。

そして、 BisQComp で g_k の符号を比較し、この符号の結果から、 BisNewRange で新しい固有値存在範囲を決める。これを繰り返して、 k 番目の固有値が求まったら、 BisEvWrite で固有値をメモリに書き込む。そして、すべての固有値が求まったら、 BisResRead で固有値をメモリから読み込む。

逆反復法

FLEX1st の実装構成は、除算器、逆反復法アルゴリズムである。アルゴリズムのステート構成を表 4.15に、 VHDL ソースを付録 B.3.4に示す。

先の二分法を計算し終えたあと、計算結果となる固有値は一度 PC の方に出力され、 FLEX 2nd の SRAM に格納される。そして、この反復法のアルゴリズムは FLEX 1st にダウンロードされ、反復法により固有ベクトルの計算が行なわれる。

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

表 4.15: ステートの構成

この反復法のステートの動作を説明する。 InvDimRead で行列の次元をメモリから

読み込み、InvDimRead で固有値 λ_1 をメモリから読み込む。そして、InvAlphaLambdaCalc で主対角成分 α_k を読み込み、$\alpha_k - \lambda$ を計算する。

そして、InvViceRead、InvViceWrite を使って副対角成分 β_k を右メモリから左メモリへ移す。そして、InvAlpha1Read、InvAlpha2Read、InvBeta2Read、InvBeta3Read により左メモリから α_k、α_{k+1}、β_k、β_{k+1} を InvBeta1Read で右メモリから β_k を読み込む。

InvAbsComp でのピボットの選択では、右メモリの β_k が左メモリの α_k よりも大きい場合には α_k と β_k、α_{k+1} と β_{k+1} をそれぞれ入れ替える。次に InvMCalc で $\mathrm{m} = \frac{\beta_k}{\alpha_k}$ を計算する。そしてその m を左メモリに書き込む。ピボットを行なった場合には、InvAlpha1Write、InvBeta2Write、InvBeta3Write を使い、α_k、β_k、β_{k+1} を左メモリに書き込み、InvMxBeta3Calc で $m \times \beta_{k+1}$ を計算する。ピボットを行なわなかった場合にはこのメモリの書込と計算は飛ばして、InvMxBeta2Calc で $m \times \beta_k$ を計算する。これが終わったら、最後に InvAlpha2mMxBeta2Calc で $\alpha_{k+1} - m \times \beta_k$ を計算する。

次に InvXmRxXCalc で $\boldsymbol{x}$ - $\boldsymbol{Rx}$ に関する計算を行なう。そして、InvAlphaRead で α_k を左メモリから読み込み、InvXCalc で $\boldsymbol{x}$ を計算する。そして、$\boldsymbol{x}$ をメモリに書き込むわけだが、(2.2.31) によるべき乗法により、後退代入を行なって固有ベクトルを求める。そのために、InvRRange に戻り、後退代入の範囲を設定し、InvRxXCalc で新たな $\boldsymbol{Rx}$ を計算し、先に書き込んだ x を InvXRead でメモリから読み込み、さらに InvXmRxXCalc で $\boldsymbol{x}$ - $\boldsymbol{Rx}$ を計算する。

上の動作によって正しい $\boldsymbol{x}$ が求まったら、x_{k+1} の計算を行なう。InvX1Read と InvX2Read で右メモリから x_k と x_{k+1} を読み込む。そして、InvX1X2Comp で x_k と x_{k+1} を交換し、InvX1Write で交換した x_k をメモリに書き込み、InvMxX1Calc で $m \times x_k$ を計算し、その結果を用いて、InvEvCalc で x_{k+1} を計算する。

これらの動作をすべての固有値に行ない、固有ベクトルを求める。そして InvResRead で $\boldsymbol{x}$ をメモリから読み込む。

逆ハウスホルダ変換

FLEX1st の実装構成は、除算器、平方根計算器、逆ハウスホルダ変換アルゴリズムである。アルゴリズムのステート構成を表 4.16に、VHDL ソースを付録 B.3.5に示す。

先の反復法を計算し終えたあと、計算結果となる固有ベクトルは三重対角化されているため、そのままでは出力できない。そのため、この三重対角化を逆ハウスホルダ

変換でもとの行列に対応する固有ベクトルに変換する必要がある。

つまり、反復法で計算した固有ベクトルは FLEX 2nd の SRAM に格納され、逆ハウスホルダ法のアルゴリズムは FLEX 1st にダウンロードされ、固有ベクトルを変換し、もとの行列に対応する固有ベクトルを PC に出力する。

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	$\boldsymbol{\omega}^T\ \boldsymbol{x}$ の計算
HsInvUtxXxCCalc	$c\ \boldsymbol{\omega}^T\ \boldsymbol{x}$ の計算
HsInvUtxXxCxUCalc	c $\boldsymbol{\omega}^T\ \boldsymbol{x}\boldsymbol{\omega}$ の計算
HsInvXmUtxXxCxUCalc	$\boldsymbol{x}$ - c $\boldsymbol{\omega}^T\ \boldsymbol{x}\boldsymbol{\omega}$ 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

表 4.16: ステートの構成

この逆ハウスホルダ法のステートの動作を説明する。まず、 HsInvDimRead で行列の次元データをメモリから読み込む。そして、 HsInvDimRead で変数を初期化する。

次に逆変換の計算式である (2.2.46) を計算する。それには加算器と乗算器を使い、以下の計算を行なう。 HsInvUtxXCalc で $\boldsymbol{\omega}^T\ \boldsymbol{x}$ を計算し、 HsInvUtxXxCCalc で $c\boldsymbol{\omega}^T\ \boldsymbol{x}$ を計算、そして、 HsInvUtxXxCxUCalc で $\boldsymbol{x}$ - c $\boldsymbol{\omega}^T\ \boldsymbol{x}\boldsymbol{\omega}$ を計算し、 (2.2.46) 式の右辺第 2 項が求まる。そして、 HsInvXmUtxXxCxUCalc で (2.2.46) の $\boldsymbol{x}^{(r-1)}$ が求まる。

これを r=n-2 回繰り返し、逆変換の結果である $\boldsymbol{y}$ を求める。まず、 HsInvX2Calc で、積和器を用いて $\sum x_k^2$ を求め、 HsInvNormCalc で、平方根計算器を用いて $\sqrt{\sum x_k^2}$ を求める。そして、 HsInvInvNormCalc で除算器を用いて先の計算結果の逆数である $\frac{1}{\sqrt{\sum x_k^2}}$ を求める。そして HsInvXNorm で $\boldsymbol{y}$ を計算する。

固有ベクトルをすべて逆変換したら、 HsInvResRead で固有ベクトルをメモリから読み込む。

第 5 章

結果、考察

5.1 メモリコントローラ

まず、DRAM コントローラを使い、行列データの記憶を行なった。方法として、PC から行列データを FPGA に入力し、そのデータを DRAM に記憶させる。そして、すべてのデータを記憶したあと、DRAM からデータを読み込み、FPGA から PC に出力する。この DRAM コントローラによって、300 次の行列を DRAM に正しく記憶できることを確認した。

DRAM メモリコントローラを使って、行列の掛け算をおこなうプログラムを作成し、シミュレーションによる動作確認、および FPGA に実装して動作検証を行なった。このプログラムでは、積和器とメモリコントローラと掛け算プログラムで構成されており、FLEX1st に搭載される。entity を表 5.1に、VHDL ソースを付録 Aに示す。この計算で使われている積和器については、図 4.2のように、乗算器の部分は 2 クロックで動作させている。6 次の行列の掛け算の結果を以下に示す。

$$\boldsymbol{A} = \begin{pmatrix} 0.500000 & 0.250000 & 0.500000 & 0.500000 & 0.250000 & 0.125000 \\ 0.500000 & 0.375000 & 0.750000 & 0.250000 & 0.125000 & 0.250000 \\ 0.250000 & 0.375000 & 0.625000 & 0.500000 & 0.250000 & 0.750000 \\ 0.500000 & 0.500000 & 0.250000 & 0.875000 & 0.250000 & 0.125000 \\ 0.500000 & 0.375000 & 0.250000 & 0.250000 & 0.125000 & 0.750000 \\ 0.250000 & 0.750000 & 0.500000 & 0.500000 & 0.250000 & 0.125000 \end{pmatrix}$$

$$\boldsymbol{B} = \begin{pmatrix} 0.250000 & 0.750000 & 0.375000 & 0.125000 & 0.500000 & 0.250000 \\ 0.125000 & 0.750000 & 0.500000 & 0.375000 & 0.125000 & 0.500000 \\ 0.500000 & 0.250000 & 0.125000 & 0.375000 & 0.500000 & 0.750000 \\ 0.250000 & 0.500000 & 0.250000 & 0.125000 & 0.500000 & 0.750000 \\ 0.375000 & 0.250000 & 0.125000 & 0.500000 & 0.750000 & 0.500000 \\ 0.500000 & 0.250000 & 0.375000 & 0.125000 & 0.375000 & 0.500000 \end{pmatrix}$$

$$\boldsymbol{AB} = \begin{pmatrix} 0.687500 & 1.031250 & 0.578125 & 0.546875 & 1.015625 & 1.187500 \\ 0.781250 & 1.062500 & 0.640625 & 0.609375 & 0.984375 & 1.250000 \\ 1.015625 & 1.125000 & 0.796875 & 0.687500 & 1.203125 & 1.593750 \\ 0.687500 & 1.343750 & 0.765625 & 0.593750 & 1.109375 & 1.406250 \\ 0.781250 & 1.062500 & 0.765625 & 0.484375 & 0.921875 & 1.125000 \\ 0.687500 & 1.218750 & 0.734375 & 0.703125 & 0.953125 & 1.375000 \end{pmatrix}$$

信号名	用途	方向	型
CLK	クロック	in	std_logic
A	インターフェースポート A	inout	std_logic_vector(15 downto 0)
BL	インターフェースポート B 下位	in	std_logic_vector(7 downto 0)
BH	インターフェースポート B 上位	out	std_logic_vector(5 downto 0)
OBF	インターフェースの OBF	in	std_logic_vector(1 downto 0)
ACK	インターフェースの ACK	out	std_logic_vector(1 downto 0)
STB	インターフェースの STB	out	std_logic_vector(1 downto 0)
IBF	インターフェースの IBF	in	std_logic_vector(1 downto 0)
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(15 downto 0)
R_SCS	右メモリのチップセレクト (SRAM)	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 (SRAM)	out	std_logic
R_SWE	右メモリのライトイネーブル (SRAM)	out	std_logic
R_DWE	右メモリのライトイネーブル (SRAM)	out	std_logic
R_DRAS	右メモリの行アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)
R_DCAS	右メモリの列アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)
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
L_DWE	左メモリのライトイネーブル (SRAM)	out	std_logic
L_DRAS	左メモリの行アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)
L_DCAS	左メモリの列アドレス指定 (DRAM)	out	std_logic_vector(3 downto 0)

表 5.1: entity の構成

このプログラムにおいては、20 次まで計算が正しく動作し、正しい解答が得られた。しかし、それ以降では、計算が正しく動作せず、計算が途中で停止することがあった。

この行列の掛け算において、計算が停止する理由の一つとして、積和器に入力されるデータがあげられる。浮動小数点の掛け算を行なう場合には、図 1.6の第 2 ステージにおいて、指数部上位 8bit と仮数部下位 24bit に分けて計算を行なうわけだが、仮数部 24bit 同士の乗算部分において計算が止まることがあるといえる。

たとえば、3.5517×1.556 を計算する場合を考える。これを 16 進数の浮動小数点 32bit データに置き換えると、それぞれ”40634f0e”、”3fc72b02” となる。この下位 24bit を乗算すると

$$1\&Ma \times 1\&Mb = Mmultip \tag{5.1.1}$$

より、Mmultip は”b0d8a3eff81c” となる。図 1.6の第 3 ステージの” 選択” の部分で、Multip の最上位 bit が”1” の場合には、この Multip を 1bit 右シフトして指数部の結果に 1 を足す。しかし、先の例では、計算が途中で停止し、結果が PC に出力されない。これはこの指数部に 1 を足す部分で、パイプラインの同期が合わず、オーバーフローが起こり、計算が止まってしまうからといえる。そのため、次の節ではこの乗算器に使われている仮数部の乗算について、2 クロックから 4 クロックにして計算した。

5.2 積和器

乗算器を 4 クロックで動作させたときの積和器の VHDL ソースを付録 A.2に示す。そしてこの積和器を用いた 6 次の行列の掛け算の結果を下に示す。

この結果はコンピュータ上で計算させたときと同じ解となり、小数の桁数が多い演算でも、この積和器で原理的に計算ができる。

$$\boldsymbol{A} = \begin{pmatrix} 0.673625 & 0.096829 & 0.414189 & 0.450488 & 0.861278 & 0.722656 \\ 0.096829 & 0.356556 & 0.419520 & 0.802542 & 0.535471 & 0.305172 \\ 0.414189 & 0.519520 & 0.668377 & 0.047554 & 0.656164 & 0.637889 \\ 0.450488 & 0.802542 & 0.047554 & 0.576995 & 0.738656 & 0.019806 \\ 0.461278 & 0.535471 & 0.656164 & 0.738656 & 0.119806 & 0.353558 \\ 0.722656 & 0.305172 & 0.637889 & 0.019806 & 0.553558 & 0.534221 \end{pmatrix}$$

$$\boldsymbol{B} = \begin{pmatrix} 0.742807 & 0.121131 & 0.082305 & 0.364350 & 0.998257 & 0.070082 \\ 0.121131 & 0.342428 & 0.055522 & 0.448287 & 0.156668 & 0.756265 \\ 0.082305 & 0.055522 & 0.661121 & 0.729354 & 0.587787 & 0.602249 \\ 0.364350 & 0.448287 & 0.729354 & 0.376037 & 0.361563 & 0.573008 \\ 0.998257 & 0.156668 & 0.587787 & 0.361563 & 0.438607 & 0.260220 \\ 0.077082 & 0.756265 & 0.602249 & 0.573008 & 0.260220 & 0.022848 \end{pmatrix}$$

$$\boldsymbol{AB} = \begin{pmatrix} 1.620749 & 1.021152 & 1.604680 & 1.485827 & 1.659768 & 0.868649 \\ 0.997975 & 0.831568 & 1.388989 & 1.171356 & 1.003552 & 1.135268 \\ 1.142654 & 0.871710 & 1.309349 & 1.491930 & 1.358706 & 1.037022 \\ 1.384737 & 0.721383 & 0.980009 & 1.053979 & 1.141141 & 1.190433 \\ 0.875013 & 0.892950 & 1.323592 & 1.410358 & 1.341671 & 1.294969 \\ 1.223510 & 0.727069 & 1.159698 & 1.379058 & 1.533120 & 0.833206 \end{pmatrix}$$

5.3 シミュレーション結果

次に 4.1 で設計した行列の掛け算を行なったときのシミュレーション結果を表 5.1から表 5.4に示す。これらの結果からメモリコントローラの FPGA での動作が正しいことがわかる。

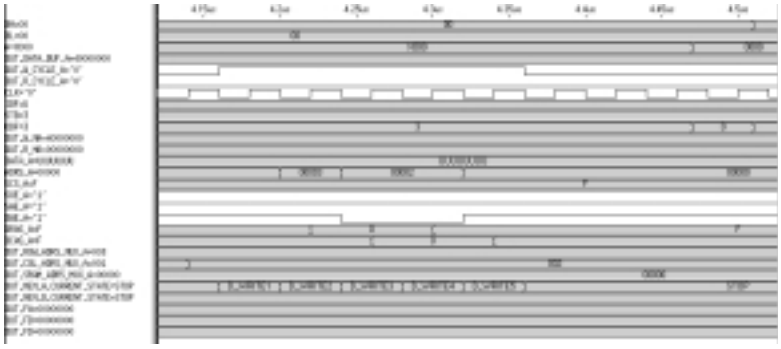


図 5.1: シミュレーション結果 (DRAM 書き込み)[1]

DRAM の書き込みサイクルについて各信号の動作をシミュレーションした。図 4.5.d と比較すると信号の動作が同じとなっていることがわかる。

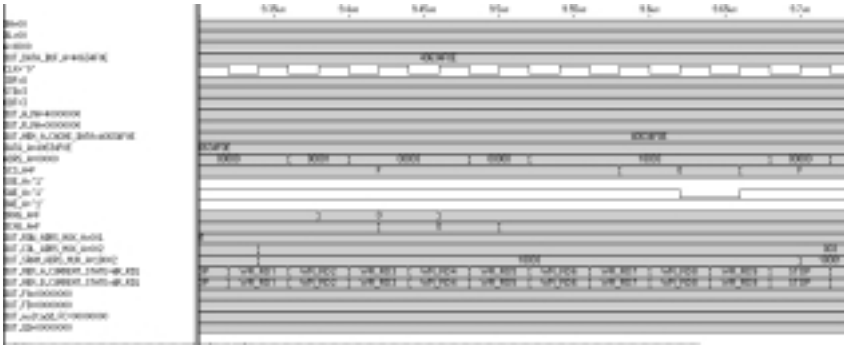


図 5.2: シミュレーション結果 (DRAM 読み込み→ SRAM 書き込み)[2]

DRAM の読み込みサイクルおよび SRAM の書き込みサイクルについて各信号の動作をシミュレーションした。図 4.5.e と図 4.5.a と比較すると信号の動作が同じとなっていることがわかる。

[1] ファイル名:./fig/sim1.eps

[2] ファイル名:./fig/sim2.eps

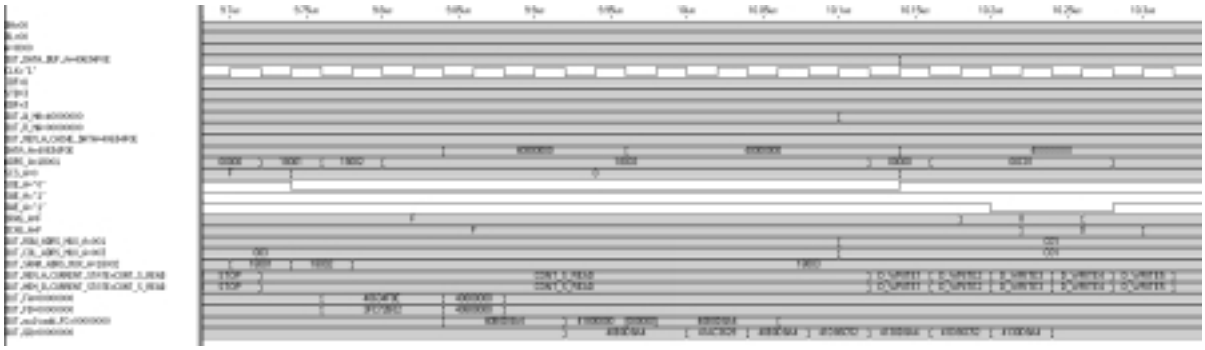


図 5.3: シミュレーション結果 (積和器の計算)[3]

2 度目のクロックの立ち下がりから、 SRAM の連続読み込みが正しく行なわれており、下の FA、 FB、 FC、 QQ より、積和器が正しく動作していることがわかる。そして、最後のところで、 DRAM の書き込みが正しく行なわれているのがわかる。

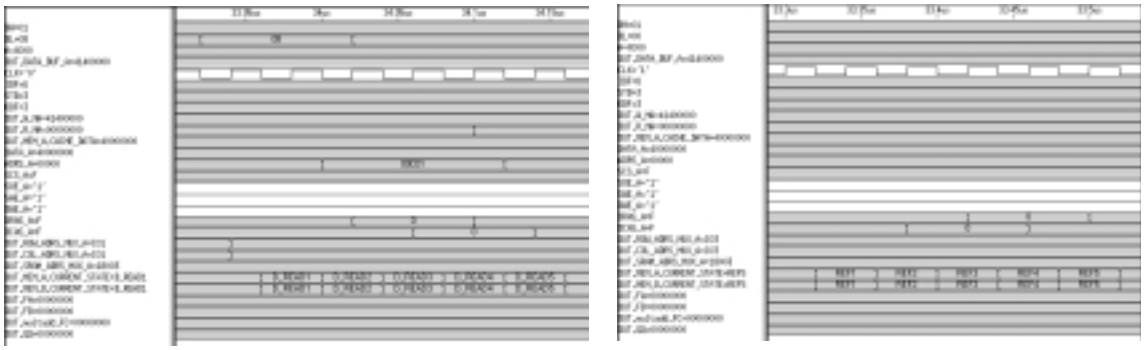


図 5.4: シミュレーション結果 ((左)DRAM 読み込み、 (右) リフレッシュサイクル) [4]

左の結果では図 4.5.e と比較すると、 DRAM の読み込みが正しく行なわれているのがわかる。そして、右の結果では図 4.5.f と比較すると、 DRAM のリフレッシュサイクルが正しく行なわれているのが分かる。

DRAM の 3 サイクルについて、半クロック遅れて動作しているが、これは、ステートマシンの状態変化が、図 4.5での設計方針ではクロックの立ち上がりで行なわれているのに対し、実際の設計ではクロックの立ち下がりで行なわれるようにしているためである。

[3]ファイル名:./fig/sim3.eps

[4]ファイル名:(左)./fig/sim4.eps、 (右). /fig/sim5.eps

5.4 計算性能

ハウスホルダ法を用いた行列の固有値および固有ベクトルを求める行列計算システムをこの評価基板に実装して、そのときの各アルゴリズムおよび通信の所要時間を計測した。その結果を表 5.2に示す。このときのシステムクロックは 4MHz で、計算に使用した行列の次数は 10 次である。

動作	時間 (秒)
積和器のダウンロード (FLEX2nd)	3.46
ハウスホルダ変換のダウンロード	3.41
行列データの入力	0.16
ハウスホルダ変換の計算	0.00
二分法のダウンロード	3.63
二分法の計算	0.00
固有値の出力	0.06
逆反復法のダウンロード	3.62
逆反復法の計算	0.00
逆ハウスホルダ変換のダウンロード	3.63
逆ハウスホルダ変換の計算	0.00
固有ベクトルの出力	0.16

表 5.2: 行列計算の所要時間

次数	時間 (秒)
10 次	0.16
20 次	0.66
30 次	2.14
50 次	6.16
70 次	11.90
100 次	27.41
150 次	51.20
200 次	112.30
250 次	142.10
300 次	198.46

表 5.3: 行列データの入力にかかる通信時間

この計算時間のほとんどが FPGA へのダウンロードで使われている。また、このダウンロードの所要時間はすべて約 3.5 秒となっている。このダウンロードの時間は FPGA に関するものなので、この評価基板において、ダウンロードの時間短縮は望めない。

次に、この行列データの次数による FPGA への入力にかかる通信時間について、表 5.3に示す。この通信時間を計測する方法としては、先に述べた行列データを DRAM にいったん記憶し、その DRAM のデータを PC に返す方法をとった。

10 次から 300 次までの通信時間を比べると、時間は次数の 2 乗に比例して増加していることがわかる。 300 次の場合には、 3 分以上もかかった。この表 5.3から考えると、 1000 次の場合には、約 2500 秒かかると思われるが、これに関しては、この評価基板でするうえでは短縮は望めない。しかし、ここで述べた 2 つの所要時間の問題点に関しては、行列の計算自体の性能の問題ではなく、評価基板の問題なので、計算機の性能を調べる上ではこの 2 つの要素は無視してもよい。

最後に、計算性能を示す 4 つのアルゴリズムの計算に関しては、 10 次の行列計算については 0.00 秒を示し、完全な性能をだすことはできなかった。

第 6 章

結論

本研究では、書き換え可能なゲート素子である FPGA を用いて、 DRAM コントローラの設計および積和器の改良を行なった。この設計は VHDL を用いて行なわれ、 DRAM コントローラをシミュレーションで検証し、 FPGA に実装して実際に動作するのを確認した。これにより、 100 次や 1000 次などの高次数での行列の固有値計算が可能となった。そして、積和器の改良により、浮動小数点の計算において、正しくない結果が出ることが少なくなった。

この DRAM コントローラを用いて行列の対角化を行なおうとしたが、昨年設計された固有値の計算システムにある程度の変更をしなければならなかった。例えば、 2 つの FPGA 間のメモリの動作に関しては、 SRAM だけでなく DRAM も追加されたため、計算方法を一部直したため、 DRAM と行列計算との同期をとるのが難しくなったためである。この同期をうまく取り、対角化ができるようになれば、高次数での使用が可能になったことで、科学計算への利用といった実用的に利用が可能となる。

また、倍精度浮動小数点の演算やハウスホルダ法のアルゴリズムをすべて FPGA に実装したいと思うならば、また新たにゲート数が多い FPGA に置き換える必要がある。そして、現在 PC とのインターフェースはデータについては 16bit しか扱えず、浮動小数点ならば 2 回に分けて通信している。インターフェースについてもバス幅を広くする必要がある。そのため、あらたに評価基板を作り直すこととなるといえる。

第 7 章

謝辞

本研究および論文作成にあたり、懇切なる御指導、を賜わりました指導教官である齋藤理一郎助教授に心より御礼の言葉を申し上げます。

本研究およびセミナー等で御指導を賜りました木村忠正教授、湯郷成美助教授、一色秀夫助手に厚く感謝の意を表します。

また、本研究をするにあたり、さまざまな資産を残して頂いた八木将志様、中島瑞樹様、松尾竜馬様、グエン・ドゥック・ミン様に多大なる感謝をいたします。

本研究を共同で行なってきた山岡寛明氏、ホー・フィ・クーン氏、そして、木村・齋藤研究室の大学院生、卒研生の方々にも感謝の意を表します。

本研究にあたって、 Max+PlusII を無償で提供して頂きましたアルテラ・ユニバーシティプログラムマネージャー浮谷光明様をはじめ、日本アルテラ（株）にも感謝致します。

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” ,199 年度卒業論文

[5] 山岡寛明, “FPGA を用いた行列計算専用プロセッサの設計” ,1998 年度卒業論文

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

[7] 落合 幸弘, “CPLD を使用した DRAM コントローラの設計”, ALTERA PLD WORLD ’96 技術論文集, pp.37-48, 1996.

[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 A

プログラムソース（行列の掛け算）

A.1 メモリコントローラ (DRAM,SRAM 兼用)

```
 ------------------------------------------------------------
-- DRAM Memory Controller  (FLEX10k)
-- < ctdrmsrm3.vhd >
-- 1999/11/09 (Tue)
-- numa@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 ctrl_dramsram is
port (  CLK               : in std_logic;
        RESET             : in std_logic;
        ADRS              : out std_logic_vector(16 downto 0);
        DATA              : in std_logic_vector(31 downto 0);
        DATA_BUF          : out std_logic_vector(31 downto 0);
        SRAM_ADRS_MUX     : in std_logic_vector(16 downto 0);
        ROW_ADRS_MUX      : in std_logic_vector(11 downto 0);
        COL_ADRS_MUX      : in std_logic_vector(11 downto 0);
        WRITE_DATA_REG    : in std_logic_vector(31 downto 0);
        READ_DATA_REG     : out std_logic_vector(31 downto 0);
        SCS               : out std_logic_vector(3 downto 0);
        SOE               : out std_logic;
        SWE               : out std_logic;
        OE                : out std_logic;
        DWE               : out std_logic;
        DRAS              : out std_logic_vector(3 downto 0);
        DCAS              : out std_logic_vector(3 downto 0);
        DOE               : out std_logic;
        MEM_STATE_SEL     : in std_logic_vector(2 downto 0);
        MEM_CYCLE         : in std_logic;
        W_CYCLE           : out std_logic;
        R_CYCLE           : out std_logic;
        REF_CYCLE         : out std_logic
);
end ctrl_dramsram;

architecture RTL of ctrl_dramsram is

type STATE_TYPE is (
  STOP, S_WRITE1, S_WRITE2, S_READ1, S_READ2,
  D_WRITE1, D_WRITE2, D_WRITE3, D_WRITE4, D_WRITE5,
```

```
  D_READ1, D_READ2, D_READ3, D_READ4, D_READ5,
  REF1, REF2, REF3, REF4, REF5, CONT_D_READ1, CONT_D_READ2,
  CONT_D_READ3, CONT_D_READ4, CONT_D_READ5, CONT_D_READ6,
  CONT_D_READ7, CONT_D_READ8, CONT_D_READ9, CONT_S_READ
  );

signal CURRENT_STATE : STATE_TYPE;
signal NEXT_STATE : STATE_TYPE;

signal NEXT_MEM_CYCLE : std_logic;

signal ADRS_COL : std_logic;
signal ADRS_ROW : std_logic;
signal ADRS_SRAM : std_logic;

signal W_CYCLE1 : std_logic;
signal CACHE_DATA : std_logic_vector(31 downto 0);
begin

process ( CLK, RESET, CURRENT_STATE,
 MEM_CYCLE, MEM_STATE_SEL, NEXT_MEM_CYCLE) begin

if RESET = '1' then
        SOE <= '1'; SWE <= '1';
        SCS <= "1111"; OE <= '1';
        DATA_BUF <= "00000000000000000000000000000000";
        READ_DATA_REG <= "00000000000000000000000000000000";
        CACHE_DATA <= "00000000000000000000000000000000";
        NEXT_STATE <= STOP;
        ADRS_COL <= '0'; ADRS_ROW <= '0';
        ADRS_SRAM <= '0';
        DRAS <= "1111"; DCAS <= "1111";
        DOE <= '1';

elsif rising_edge( CLK ) then

case CURRENT_STATE is

    when STOP =>
        SOE <= '1';        SWE <= '1';          SCS <= "1111";
        OE <= '1';
        ADRS_COL <= '0';   ADRS_ROW <= '0';     ADRS_SRAM <= '0';
        DRAS <= "1111";    DCAS <= "1111";
        DOE <= '1';
        NEXT_STATE <= STOP;

-----< SRAM CYCLE >-----
    when S_WRITE1 =>
        SCS <= "0000";
        DATA_BUF <= WRITE_DATA_REG;
        OE <= '0';
        ADRS_SRAM <= '1';
        NEXT_STATE <= S_WRITE2;

    when S_WRITE2 =>
        SWE <= '0';
        NEXT_STATE <= STOP;

    when S_READ1 =>
        SCS <= "0000";    SOE <= '0';          OE <= '1';
        ADRS_SRAM <= '1';
        NEXT_STATE <= S_READ2;

    when S_READ2 =>
        READ_DATA_REG <= DATA;
        NEXT_STATE <= STOP;

-----< DRAM CYCLE >-----

    when D_WRITE1 =>
        SOE <= '1';       SWE <= '1';          SCS <= "1111";
        OE <= '0';
        ADRS_SRAM <= '0'; ADRS_ROW <= '1';
        DATA_BUF <= WRITE_DATA_REG;
        NEXT_STATE <= D_WRITE2;
```

```
when D_WRITE2 =>
    ADRS_COL <= '1';  ADRS_ROW <= '0';
    DRAS <= "0000";   DOE <= '0';
    NEXT_STATE <= D_WRITE3;
when D_WRITE3 =>
    DCAS <= "0000";
    NEXT_STATE <= D_WRITE4;
when D_WRITE4 =>
    ADRS_COL <= '0';
    DRAS <= "1111";   DOE <= '1';
    NEXT_STATE <= D_WRITE5;
when D_WRITE5 =>
    DCAS <= "1111";
    NEXT_STATE <= STOP;

when D_READ1 =>
    ADRS_SRAM <= '0';
    OE <= '1';
    ADRS_ROW <= '1';
    NEXT_STATE <= D_READ2;
when D_READ2 =>
    ADRS_COL <= '1'   ADRS_ROW <= '0';
    DRAS <= "0000";
    NEXT_STATE <= D_READ3;
when D_READ3 =>
    DCAS <= "0000";
    NEXT_STATE <= D_READ4;
when D_READ4 =>
    ADRS_COL <= '0';
    READ_DATA_REG <= DATA;
    DRAS <= "1111";
    NEXT_STATE <= D_READ5;
when D_READ5 =>
    DCAS <= "1111";
    NEXT_STATE <= STOP;

when REF1 =>
    ADRS_SRAM <= '0';
    OE <= '1';
    NEXT_STATE <= REF2;
when REF2 =>
    DCAS <= "0000";
    NEXT_STATE <= REF3;
when REF3 =>
    DRAS <= "0000";
    NEXT_STATE <= REF4;
when REF4 =>
    DCAS <= "1111";
    NEXT_STATE <= REF5;
when REF5 =>
    DRAS <= "1111";
    NEXT_STATE <= STOP;
when CONT_D_READ1 =>
    ADRS_SRAM <= '0';
    OE <= '1';
    ADRS_ROW <= '1';
    NEXT_STATE <= CONT_D_READ2;
when CONT_D_READ2 =>
    ADRS_COL <= '1';  ADRS_ROW <= '0';
    DRAS <= "0000";
    NEXT_STATE <= CONT_D_READ3;
when CONT_D_READ3 =>
    DCAS <= "0000";
    NEXT_STATE <= CONT_D_READ4;
when CONT_D_READ4 =>
    ADRS_COL <= '0';
    CACHE_DATA <= DATA;
    DRAS <= "1111";
    NEXT_STATE <= CONT_D_READ5;
when CONT_D_READ5 =>
    DCAS <= "1111";
    ADRS_SRAM <= '1';
    NEXT_STATE <= CONT_D_READ6;
when CONT_D_READ6 =>
    OE <= '0';
    NEXT_STATE <= CONT_D_READ7;
```

```
    when CONT_D_READ7 =>
        SCS <= "0000";
        DATA_BUF <= CACHE_DATA;
        NEXT_STATE <= CONT_D_READ8;
    when CONT_D_READ8 =>
        SWE <= '0';
        NEXT_STATE <= CONT_D_READ9;
    when CONT_D_READ9 =>
        ADRS_SRAM <= '0';
        SWE <= '1';       SCS <= "1111";
        OE <= '1';
        NEXT_STATE <= STOP;
    when CONT_S_READ =>
        SOE <= '0';       SWE <= '1';       SCS <= "0000";
        OE <= '1';
        ADRS_SRAM <= '0';

    when others =>
        NEXT_STATE <= STOP;

end case;
end if;
end process;

process ( RESET, CURRENT_STATE ) begin
if RESET = '1' then
        W_CYCLE <= '0'; R_CYCLE <= '0'; REF_CYCLE <= '0';
        NEXT_MEM_CYCLE <= '1'; W_CYCLE1 <= '0';

else
        case CURRENT_STATE is
                when S_WRITE1 | S_WRITE2
 | D_WRITE1 | D_WRITE2 | D_WRITE3 | D_WRITE4 | D_WRITE5
            => W_CYCLE <= '1';  W_CYCLE1 <= '1'; NEXT_MEM_CYCLE <= '1';
                when S_READ1  | S_READ2
 | D_READ1 | D_READ2 | D_READ3 | D_READ4 | D_READ5
 | CONT_D_READ1 | CONT_D_READ2 | CONT_D_READ3 | CONT_D_READ4
 | CONT_D_READ5 | CONT_D_READ6 | CONT_D_READ7 | CONT_D_READ8 | CONT_D_READ9
        => R_CYCLE <= '1'; NEXT_MEM_CYCLE <= '1';
                when REF1 | REF2 | REF3 | REF4 | REF5 =>
                        REF_CYCLE <= '1'; NEXT_MEM_CYCLE <= '1';
                when STOP => W_CYCLE <= '0'; R_CYCLE <= '0';
                        W_CYCLE1 <= '0'; REF_CYCLE <= '0';
                        NEXT_MEM_CYCLE <= '0';
                when others => W_CYCLE <= '0'; R_CYCLE <= '0';
                        W_CYCLE1 <= '0'; REF_CYCLE <= '0';
                        NEXT_MEM_CYCLE <= '0';
        end case;
end if;
end process;

process (CLK, RESET, MEM_CYCLE, NEXT_MEM_CYCLE ) begin
-----< RAM Decision Controller >-----
if RESET = '1' then
        CURRENT_STATE <= STOP;

elsif falling_edge( CLK ) then
        if MEM_CYCLE = '1' and NEXT_MEM_CYCLE = '0' then
                case MEM_STATE_SEL is
                        when "000" => CURRENT_STATE <= STOP;
                        when "001" => CURRENT_STATE <= S_READ1;
                        when "011" => CURRENT_STATE <= CONT_S_READ;
                        when "101" => CURRENT_STATE <= D_WRITE1;
                        when "110" => CURRENT_STATE <= D_READ1;
                        when "100" => CURRENT_STATE <= REF1;
                        when "111" => CURRENT_STATE <= CONT_D_READ1;
                        when others => CURRENT_STATE <= STOP;
                end case;
        elsif NEXT_MEM_CYCLE = '1' then
                CURRENT_STATE <= NEXT_STATE;
        end if;
end if;
end process;

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

```
        DWE <= '1';

elsif falling_edge ( CLK ) then
        if ADRS_ROW = '1' then
                ADRS <= "00000" & ROW_ADRS_MUX;
        elsif ADRS_COL = '1' then
                ADRS <= "00000" & COL_ADRS_MUX;
                if W_CYCLE1 = '1' then
                        DWE <= '0';
                else
                        DWE <= '1';
                end if;
        elsif ADRS_SRAM = '1' or MEM_STATE_SEL = "011" then
                ADRS <= SRAM_ADRS_MUX; ---"000000000000";
                DWE <= '1';
        else
                ADRS <= "0000000000000000";
                DWE <= '1';
        end if;
end if;
end process;

end RTL;
```

A.2 積和器 (FLEX 1st)

```
-----------------------------------------------------------
-- Floating Point Number Multiplier and Adder  (FLEX10k)
-- < multadd6.vhd >
-- 1999/12/28 (Tue)
-- numa@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 multadd6 is
  port ( CLK  : in std_logic;
         WQ   : in std_logic;
         FA   : in std_logic_vector(31 downto 0);
         FB   : in std_logic_vector(31 downto 0);
         FD    : out std_logic_vector(31 downto 0);
         Q    : out std_logic_vector(31 downto 0)
       );

end multadd6;
architecture RTL of multadd6 is

----------< fpmult signal >---------
signal MMA, MMB : std_logic_vector(23 downto 0);
signal MMC, MC0 : std_logic_vector(25 downto 0);
signal MMQ : std_logic_vector(22 downto 0);
signal TMC1, TMC2 : std_logic_vector(47 downto 0);
signal MEA, MEB, EC0, EC1, EC02, EC12 : std_logic_vector(9 downto 0);
signal EC0A, EC1A, MEQ : std_logic_vector(7 downto 0);
signal S, SS, SSS, SSSS, MSQ : std_logic;
signal FC : std_logic_vector(31 downto 0);

----------< fpadder signal >----------
signal INF0, INF1, INF2, INF3, ZA, ZB, ZA1, ZB1 : std_logic;
signal SA, SB, SA2, ASQ : std_logic;
signal AEA, EA2, EA3, AEQ, ES, ES2 : std_logic_vector(7 downto 0);
signal AMA, AMB, MC3, AMQ : std_logic_vector(22 downto 0);
signal AMC : std_logic_vector(25 downto 0);
signal V0, V1, V2, V3, V4, V5  : std_logic_vector(24 downto 0);
signal MB2 : std_logic_vector(23 downto 0);
signal VV0, VV1, VV2, VV3, VV4, MC2, MC3A : std_logic_vector(24 downto 0);
```

```
signal QQ : std_logic_vector(31 downto 0);

-----------< others >-------
signal DCNT : std_logic_vector(3 downto 0);
signal QQQ : std_logic_vector(31 downto 0);
--constant DELAY_TIME : std_logic_vector(2 downto 0) := "011";

begin

------------< Multiplier Part >-------
process begin --1st step--
wait until rising_edge( CLK );
    MMA <= '1' & FA(22 downto 0);
    MMB <= '1' & FB(22 downto 0);
    MEA <= "00" & FA(30 downto 23);
    MEB <= "00" & FB(30 downto 23);
    S <= FA(31) xor FB(31);
end process;

process --2nd step--
begin
wait until falling_edge( CLK );
    SS <= S;
    if DCNT(0) = '1' then
        TMC1 <= MMA * MMB;
    else
        TMC2 <= MMA * MMB;
    end if;
    if    ( (MEA = "0011111111") or (MEB = "0011111111") ) then
        EC0 <= "0011111111";
        EC1 <= "0011111111";
    elsif ( (MEA = "0000000000") or (MEB = "0000000000") ) then
        EC0 <= "0000000000";
        EC1 <= "0000000000";
    else
        EC0 <= MEA + MEB - "0001111111";
        EC1 <= MEA + MEB - "0001111110";
    end if;
end process;

process begin --2.5 step--
wait until falling_edge( CLK );
    SSS <= SS;
    EC02 <= EC0; EC12 <= EC1;
    if DCNT(1) = '1' then
        MMC <= TMC1(47 downto 22);
    else
        MMC <= TMC2(47 downto 22);
    end if;
end process;

process begin --3rd step--
wait until falling_edge( CLK );
    SSSS <= SSS;
    if ( MMC(25) = '0' ) then
        MC0 <= MMC + "00000000000000000000000001";
    else
        MC0 <= MMC + "00000000000000000000000010";
    end if;

    case EC02(9 downto 8) is
        when "11" => EC0A <= "00000000";
        when "01" => EC0A <= "11111111";
        when others => EC0A <= EC02(7 downto 0);
    end case;

    case EC12(9 downto 8) is
        when "11" => EC1A <= "00000000";
        when "01" => EC1A <= "11111111";
        when others => EC1A <= EC12(7 downto 0);
    end case;
end process;

process begin --4th step--
```

```
wait until falling_edge( CLK );
    MSQ <= SSSS;
    if ( MC0(25) = '0' ) then
        MMQ <= MC0(23 downto 1);
        MEQ <= EC0A;
    else
        MMQ <= MC0(24 downto 2);
        MEQ <= EC1A;
    end if;
end process;

FC <= QQQ when ( DCNT = "0111" or DCNT = "1000" ) else MSQ & MEQ & MMQ;

---------< Adder Part >--------

INF0 <= '1' when
    ( (FC(30 downto 23 ) = "1111111111")
    or  ( QQ(30 downto 23 ) = "1111111111" )) else '0';
ZA <= '1' when  FC(30 downto 23 ) = "0000000000" else '0';
ZB <= '1' when  QQ(30 downto 23 ) = "0000000000" else '0';

process
    variable VES1 , VES2, BUF_FA, BUF_FB  : std_logic_vector(8 downto 0);
begin
wait until rising_edge( CLK );
INF1 <= INF0;

BUF_FA := '0' & FC( 30 downto 23);
BUF_FB := '0' & QQ( 30 downto 23);

VES1 := BUF_FA - BUF_FB;
VES2 := BUF_FB - BUF_FA;


if ( VES1(8) = '0' ) then
    SA <= FC(31); SB <= QQ(31);  AEA <= FC( 30 downto 23);
    AMA <= FC( 22 downto 0); AMB <= QQ(22 downto 0);
    ES <= VES1( 7 downto 0); ZA1 <= ZA; ZB1 <= ZB;
else
    SA <= QQ(31); SB <= FC(31);  AEA <= QQ( 30 downto 23);
    AMA <= QQ( 22 downto 0); AMB <= FC(22 downto 0);
    ES <= VES2( 7 downto 0); ZB1 <= ZA; ZA1 <= ZB;
end if;
end process;

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

process
    variable VMA , VMB : std_logic_vector(25 downto 0);
begin
wait until falling_edge( CLK );
EA2 <= AEA; INF2 <= INF1;
if ( ZA1 = '1' ) then
    VMA := "00000000000000000000000000";
elsif ( SA = '0' ) then
    VMA := "001" & AMA;
else
    VMA := "00000000000000000000000000" - ("001" & AMA);
end if;

if ( ZB1 = '1') then
    VMB := "00000000000000000000000000";
elsif ( SB = '0' ) then
    VMB := "00" & MB2;
else
    VMB := "00000000000000000000000000" - ("00" & MB2);
end if;
    AMC <= VMA + VMB;
end process;
```

```
process begin
wait until rising_edge( CLK );
SA2 <= AMC(25); EA3 <= EA2; INF3 <= INF2;
if (AMC(25) = '0') then
    MC2 <= AMC(24 downto 0);
else
MC2 <= "000000000000000000000000" - AMC(24 downto 0);
end if;
end process;

MC3A <= MC2 + "000000000000000000000001";

ES2 <=  "00000000" when MC2(24) = '1' else
        "00000001" when MC2(23) = '1' else
        "00000010" when MC2(22) = '1' else
        "00000001" when MC2(21) = '1' else
        "00000010" when MC2(20) = '1' else
        "00000011" when MC2(19) = '1' else
        "00000100" when MC2(18) = '1' else
        "00000101" when MC2(17) = '1' else
        "00000110" when MC2(16) = '1' else
        "00000111" when MC2(15) = '1' else
        "00001000" when MC2(14) = '1' else
        "00001001" when MC2(13) = '1' else
        "00001010" when MC2(12) = '1' else
        "00001011" when MC2(11) = '1' else
        "00001100" when MC2(10) = '1' else
        "00001101" when MC2( 9) = '1' else
        "00001110" when MC2( 8) = '1' else
        "00001111" when MC2( 7) = '1' else
        "00010000" when MC2( 6) = '1' else
        "00010001" when MC2( 5) = '1' else
        "00010010" when MC2( 4) = '1' else
        "00010011" when MC2( 3) = '1' else
        "00010100" when MC2( 2) = '1' else
        "00010101" when MC2( 1) = '1' else
        "00010110" when MC2( 0) = '1' else
        "10000000";

VV0 <= MC2 when ES2(0) = '0' else MC2(23 downto 0) & "0";
VV1 <= VV0 when ES2(1) = '0' else VV0(22 downto 0) & "00";
VV2 <= VV1 when ES2(2) = '0' else VV1(20 downto 0) & "0000";
VV3 <= VV2 when ES2(3) = '0' else VV2(16 downto 0) & "00000000";
VV4 <= VV3 when ES2(4) = '0' else VV3( 8 downto 0) & "0000000000000000";
MC3 <= VV4(23 downto 1) when MC3A(24) = '0' else MC3A(23 downto 1);

process begin
wait until falling_edge( CLK );
ASQ <= SA2; AMQ <= MC3;
if INF3 = '1' then
    AEQ <= "11111111";
elsif ES2(7) = '0' then
    AEQ <= EA3 - ES2 + "00000001";
else
    AEQ <= "00000000";
end if;
end process;

---------------< Feedback >------------
process begin
wait until rising_edge( CLK );
if DCNT = "0110" then
    QQQ <= ASQ & AEQ & AMQ;
elsif DCNT  = "1000" then
    QQQ <= "00000000000000000000000000000000";
end if;
if WQ = '1' or ( "0111" < DCNT and DCNT < "1100" ) then
    DCNT <= DCNT + "0001";
elsif WQ = '0' then
    DCNT <= "0000";
end if;
end process;

QQ <= "00000000000000000000000000000000" when DCNT > "1010" else ASQ & AEQ & AMQ;
```

```
Q <= ASQ & AEQ & AMQ;

end RTL;
```

A.3 行列の掛算計算

```
------------------------------------------------------------
-- Matrix Maltiplier using Memory  (FLEX10k)
-- Data DRAM to DRAM
-- < float8.vhd >
-- 1999/12/28 (Tue)
-- numa@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 float8 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(7 downto 0);
--         BH1  : out std_logic_vector(1 downto 0);
         CL   : in std_logic_vector(5 downto 0);
         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);
------< Right Memories Controller >-------
      DATA_A : inout std_logic_vector(31 downto 0);
      ADRS_A : out std_logic_vector(16 downto 0);
      SCS_A  : out std_logic_vector(3 downto 0);
      SOE_A  : out std_logic;
      SWE_A  : out std_logic;
      DWE_A  : out std_logic;
      DRAS_A : out std_logic_vector(3 downto 0);
      DCAS_A : out std_logic_vector(3 downto 0);
------< Left Memories Controller >-------
      DATA_B : inout std_logic_vector(31 downto 0);
      ADRS_B : out std_logic_vector(16 downto 0);
      SCS_B  : out std_logic_vector(3 downto 0);
      SOE_B  : out std_logic;
      SWE_B  : out std_logic;
      DWE_B  : out std_logic;
      DRAS_B : out std_logic_vector(3 downto 0);
      DCAS_B : out std_logic_vector(3 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  "BC5,BB6,BC7,BB8,BC9,BB10,BC11,BB12";
attribute pinnum of CL  : signal is  "AU23,AV24,AU25,AU33,AV34,AU35";
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";

----- <<SRAM & DRAM (Right)>> -----
attribute pinnum of ADRS_A : signal is "T38,W37,Y38,AA37,AB38,AC37,AD38,AE37,
                                        AF38,AJ37,AK38,AL37,AM38,AN37,AP38,AR37,AT38";
attribute pinnum of DATA_A : 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 SCS_A  : signal is  "AG39,AH40,AJ39,AM40";
attribute pinnum of SOE_A  : signal is  "AP40";
attribute pinnum of SWE_A  : signal is  "AN39";
attribute pinnum of DWE_A  : signal is  "AF40";
attribute pinnum of DRAS_A : signal is  "P40,R39,T40,U39";
attribute pinnum of DCAS_A : signal is  "Y40,AA39,AB40,AC39";

----- <<SRAM & DRAM (Left)>> -----
attribute pinnum of ADRS_B : signal is "T6,W7,Y6,AA7,AB6,AC7,AD6,AE7,AF6,AJ7,AK6,\\
                                        AL7,AM6,AN7,AP6,AR7,AT6";
attribute pinnum of DATA_B : 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 SCS_B  : signal is  "AG5,AH4,AJ5,AM4";
attribute pinnum of SOE_B  : signal is  "AP4";
attribute pinnum of SWE_B  : signal is  "AN5";
attribute pinnum of DWE_B  : signal is  "AF4";
attribute pinnum of DRAS_B : signal is  "P4,R5,T4,U5";
attribute pinnum of DCAS_B : signal is  "Y4,AA5,AB4,AC5";

end float8;
architecture RTL of float8 is

--------------< Memory Controller >------------
component ctrl_dramsram is
port (  CLK           : in std_logic;
        RESET         : in std_logic;
        ADRS          : out std_logic_vector(16 downto 0);
        DATA          : in std_logic_vector(31 downto 0);
        DATA_BUF      : out std_logic_vector(31 downto 0);
        SRAM_ADRS_MUX : in std_logic_vector(16 downto 0);
        ROW_ADRS_MUX : in std_logic_vector(11 downto 0);
        COL_ADRS_MUX : in std_logic_vector(11 downto 0);
        WRITE_DATA_REG : in std_logic_vector(31 downto 0);
        READ_DATA_REG : out std_logic_vector(31 downto 0);
        SCS           : out std_logic_vector(3 downto 0);
        SOE           : out std_logic;
        SWE           : out std_logic;
        OE            : out std_logic;
        DWE           : out std_logic;
        DRAS          : out std_logic_vector(3 downto 0);
        DCAS          : out std_logic_vector(3 downto 0);
        DOE           : out std_logic;
        MEM_STATE_SEL : in std_logic_vector(2 downto 0);
        MEM_CYCLE     : in std_logic;
        W_CYCLE       : out std_logic;
        R_CYCLE       : out std_logic;
        REF_CYCLE      : out std_logic
);
end component;

-------------< 32-bit Multiplier and Adder >-------
component multadd3 is
port ( CLK : in std_logic;
       WQ  : in std_logic;
       FA  : in std_logic_vector(31 downto 0);
       FB  : in std_logic_vector(31 downto 0);
       FD   : out std_logic_vector(31 downto 0);
       Q   : out std_logic_vector(31 downto 0)
      );
end component;

signal FA : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
signal FB : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";
signal FD : std_logic_vector(31 downto 0) ;
signal QQ : std_logic_vector(31 downto 0) := "00000000000000000000000000000000";

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_A : std_logic;
signal IN_CNT_B : std_logic;
signal OUT_CNT : std_logic;
signal OUT_DCNT : std_logic_vector(3 downto 0);
signal Q_CNT : std_logic;
signal CLK_CNT : std_logic_vector(2 downto 0);

signal COL_A : std_logic_vector(9 downto 0);
signal COL_B : std_logic_vector(9 downto 0);
signal COL_Q : std_logic_vector(9 downto 0);
signal COL_A_CNT : std_logic_vector(9 downto 0);
signal COL_B_CNT : std_logic_vector(9 downto 0);
signal COL_Q_CNT : std_logic_vector(9 downto 0);

signal SRAM_CNT : std_logic_vector(9 downto 0);
signal SRAM_CACHE_CNT : std_logic_vector(9 downto 0);
signal SRAM_CNT1 : std_logic_vector(9 downto 0);

signal ROW_A : std_logic_vector(9 downto 0);
signal ROW_B : std_logic_vector(9 downto 0);
signal ROW_Q : std_logic_vector(9 downto 0);
signal ROW_A_CNT : std_logic_vector(9 downto 0);
signal ROW_B_CNT : std_logic_vector(9 downto 0);
signal ROW_Q_CNT : std_logic_vector(9 downto 0);

signal WA : std_logic;
signal WB : std_logic;
signal WQ : std_logic;
signal CACHE_Q : std_logic;

signal CACHE_WAIT : std_logic;

signal START_CALC : std_logic;
signal END_READ : std_logic;
signal END_CALC : std_logic;
signal OUT_READY : std_logic;
signal END_OUT : std_logic;

signal OE_A : std_logic;
signal OE_B : std_logic;
signal DOE_A : std_logic := '0';
signal DOE_B : std_logic := '0';
signal DATA_BUF_A : std_logic_vector(31 downto 0);
signal DATA_BUF_B : std_logic_vector(31 downto 0);

signal MA_L : std_logic_vector(15 downto 0);
signal MA_H : std_logic_vector(15 downto 0);
signal MB_L : std_logic_vector(15 downto 0);
signal MB_H : std_logic_vector(15 downto 0);

signal MEM_CYCLE_A : std_logic;
signal MEM_CYCLE_B : std_logic;
signal MEM_RESET_A : std_logic := '0';
signal MEM_RESET_B : std_logic := '0';
signal NOT_READ_A : std_logic := '0';
signal NOT_READ_B : std_logic := '0';
signal DATA_END_A : std_logic := '0';
signal DATA_END_B : std_logic := '0';
signal MEM_STATE_SEL_A : std_logic_vector(2 downto 0);
signal MEM_STATE_SEL_B : std_logic_vector(2 downto 0);
signal SRAM_ADRS_MUX_A : std_logic_vector(16 downto 0);
signal SRAM_ADRS_MUX_B : std_logic_vector(16 downto 0);
signal ROW_ADRS_MUX_A : std_logic_vector(11 downto 0) ;
signal COL_ADRS_MUX_A : std_logic_vector(11 downto 0) ;
signal ROW_ADRS_MUX_B : std_logic_vector(11 downto 0) := "000000000000";
signal COL_ADRS_MUX_B : std_logic_vector(11 downto 0) := "000000000000";
signal R_CYCLE_A : std_logic;
signal R_CYCLE_B : std_logic;
signal W_CYCLE_A : std_logic;
signal W_CYCLE_B : std_logic;
```

```
signal REF_CYCLE_A : std_logic;
signal REF_CYCLE_B : std_logic;
signal REFRESH_A : std_logic;
signal REFRESH_B : std_logic;
signal REF_CNT_A : integer range 0 to 65;
signal REF_CNT_B : integer range 0 to 65;

signal COUNT : std_logic_vector(20 downto 0);

signal W_MA : std_logic_vector(31 downto 0);
signal W_MB : std_logic_vector(31 downto 0);
signal R_MA : std_logic_vector(31 downto 0);
signal R_MB : std_logic_vector(31 downto 0);

signal WE_MA : std_logic;
signal WE_MB : std_logic;
signal OE_CALCA : std_logic;
signal OE_CALCB : std_logic;
signal OE_MA : std_logic;
signal OE_MB : std_logic;
signal WE_RES : std_logic;
signal OE_RES : std_logic;

signal BH0 : std_logic;
signal BH1 : std_logic;
signal BH2 : std_logic;

constant DELAY_TIME : std_logic_vector(3 downto 0) := "1010"; --"111";

begin

DATA_A <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_A = '1' else DATA_BUF_A;
DATA_B <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" when OE_B = '1' else DATA_BUF_B;

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_A <= '0'; IN_CNT_B <= '0';
    MA_L <= "0000000000000000";     MA_H <= "0000000000000000";
    MB_L <= "0000000000000000";     MB_H <= "0000000000000000";
    BH1 <= '0';
elsif OBF'event and OBF = "11" then
    if WA = '0' then
        if IN_CNT_A = '0' then
            MA_L <= A;    IN_CNT_A <= '1';     BH1 <= '1';
        else
            MA_H <= A;    IN_CNT_A <= '0';     BH1 <= '0';
        end if;
    elsif WA = '1' and WB = '0' then
        if IN_CNT_B = '0' then
            MB_L <= A;    IN_CNT_B <= '1';     BH1 <= '1';
        else
            MB_H <= A;    IN_CNT_B <= '0';     BH1 <= '0';
        end if;
    end if;
end if;
end process;

---------< Matrix A Row Counter >---------
process ( BL(0), BL(1), WA, IN_CNT_A ) begin
if BL(0) = '1' or ( BL(1) = '1' and WA = '0' ) then
    COL_A <= "0000000000";
elsif rising_edge( IN_CNT_A ) then
    COL_A <= COL_A + "0000000001";
end if;
end process;

---------< Matrix B Row Counter >---------
process ( BL(0), BL(1), WA, WB, IN_CNT_B ) begin
```

```
if BL(0) = '1' or ( BL(1) = '1' and WA = '1' and WB = '0' ) then
    COL_B <= "0000000000";
elsif rising_edge( IN_CNT_B ) then
    COL_B <= COL_B + "0000000001";
end if;
end process;

---------< Matrix Column Counter >---------
process ( BL(0), BL(1) ) begin
if BL(0) = '1' then
    ROW_A <= "0000000001"; ROW_B <= "0000000001";
elsif rising_edge( BL(1) ) then
    if WA = '0' then
        ROW_A <= ROW_A + "0000000001";
    elsif WA = '1' and WB = '0' then
        ROW_B <= ROW_B + "0000000001";
    end if;
end if;
end process;

---------< Selecter of Matrix >---------
process ( BL(0), BL(2) ) begin
if BL(0) = '1' then
    WA <= '0'; WB <= '0';
elsif rising_edge( BL(2) ) then
    if WA = '0' then
        WA <= '1';
    elsif WA = '1' and WB = '0' then
        WB <= '1';
    end if;
end if;
end process;

--------------< Memory Controller MA >------------------------

-----< Generate Memory Write Signal ( Matrix A ) >-----
process ( BL(0), W_CYCLE_A, IN_CNT_A ) begin
if BL(0) = '1' or W_CYCLE_A = '1' then
    WE_MA <= '0';
elsif falling_edge( IN_CNT_A ) then
    WE_MA <= '1';
end if;
end process;

-----< Generate Memory Read Signal ( Matrix A ) >-----
process ( BL(0), CLK, END_CALC, END_READ, W_CYCLE_A, CACHE_Q ) begin
if BL(0) = '1' or END_CALC = '1' or END_READ = '1' or W_CYCLE_A = '1' \\
 or CACHE_Q = '1' then
    OE_MA <= '0';
elsif falling_edge( CLK ) then
    if WB = '1' and END_CALC = '0' then
        OE_MA <= '1';
    end if;
end if;
end process;

-----< Generate Memory Cache Signal ( Matrix A ) >-----
process ( BL(0), CLK, END_CALC, END_READ, W_CYCLE_A, R_CYCLE_A ) begin
if BL(0) = '1' or END_CALC = '1' or END_READ = '1' or W_CYCLE_A = '1' then
    OE_CALCA <= '0';
elsif falling_edge( R_CYCLE_A ) then
    if OE_MA = '0' and END_CALC = '0' then
        OE_CALCA <= '1';
    end if;
end if;
end process;

-----< Generate Memory Write Signal ( Result Data ) >-----
process ( BL(0), W_CYCLE_A, OUT_DCNT ) begin
if BL(0) = '1' or W_CYCLE_A = '1' then
    WE_RES <= '0';
elsif OUT_DCNT'event and OUT_DCNT = DELAY_TIME then
    WE_RES <= '1';
end if;
```

```
end process;

-----< Generate Memory Read Signal ( Result Data ) >-----
process ( BL(0), BL(3), R_CYCLE_A ) begin
if BL(0) = '1' then
    OE_RES <= '0';    OUT_CNT <= '0';
elsif rising_edge( BL(3) ) then
    OE_RES <= '1';
    if OUT_CNT = '0' then
        OUT_CNT <= '1';
    else
        OUT_CNT <= '0';
    end if;
end if;
if R_CYCLE_A = '1' then
    OE_RES <= '0';
end if;
end process;

-----< Selecter of Memory Operation ( Matrix A ) >-----
process ( BL(0), CLK ) begin
if BL(0) = '1' then
    MEM_CYCLE_A <= '0';    MEM_STATE_SEL_A <= "000";
    SRAM_ADRS_MUX_A <= "00000000000000000";
    ROW_ADRS_MUX_A <= "000000000000";   COL_ADRS_MUX_A <= "000000000000";
    W_MA <= "00000000000000000000000000000000";
elsif rising_edge( CLK ) then
    if WE_MA = '1' then
        W_MA <= MA_H & MA_L;
        MEM_CYCLE_A <= '1';
        MEM_STATE_SEL_A <= "101";
        ROW_ADRS_MUX_A <= "00" & ROW_A;
        COL_ADRS_MUX_A <= "00" & COL_A;
    end if;
    if OE_MA = '1' then
        MEM_CYCLE_A <= '1';
        SRAM_ADRS_MUX_A <= "1100000" & SRAM_CNT;
        ROW_ADRS_MUX_A <= "00" & ROW_A_CNT;
        COL_ADRS_MUX_A <= "00" & COL_A_CNT;
        MEM_STATE_SEL_A <= "111";
    end if;
    if OE_CALCA = '1' then
        MEM_CYCLE_A <= '1';
        SRAM_ADRS_MUX_A <= "1100000" & SRAM_CACHE_CNT;
        MEM_STATE_SEL_A <= "011";
    end if;
    if WE_RES = '1' then
        W_MA <= QQ;
        MEM_CYCLE_A <= '1';
        MEM_STATE_SEL_A <= "101";
        ROW_ADRS_MUX_A <= "11" & ROW_Q;
        COL_ADRS_MUX_A <= "11" & COL_Q;
    end if;
    if OE_RES = '1' then
        MEM_CYCLE_A <= '1';
        MEM_STATE_SEL_A <= "110";
        ROW_ADRS_MUX_A <= "11" & ROW_Q_CNT;
        COL_ADRS_MUX_A <= "11" & COL_Q_CNT;
    end if;
    if W_CYCLE_A = '1' or R_CYCLE_A = '1' or REF_CYCLE_A = '1' then
        MEM_CYCLE_A <= '0';
        MEM_STATE_SEL_A <= "000";
    end if;
    if REFRESH_A = '1' then
        MEM_CYCLE_A <= '1';
        MEM_STATE_SEL_A <= "100";
    end if;
end if;
end process;

--------------< Memory Controller MB >---------------

-----< Generate Memory Write Signal ( Matrix B ) >-----
process ( BL(0), W_CYCLE_B, IN_CNT_B ) begin
if BL(0) = '1' or W_CYCLE_B = '1' then
```

```
    WE_MB <= '0';
elsif falling_edge( IN_CNT_B ) then
    WE_MB <= '1';
end if;
end process;

-----< Generate Memory Read Signal ( Matrix A ) >-----
process ( BL(0), CLK, END_CALC, END_READ, W_CYCLE_B, CACHE_Q ) begin
if BL(0) = '1' or END_CALC = '1' or END_READ = '1' or W_CYCLE_B = '1' or CACHE_Q = '1' then
    OE_MB <= '0';
elsif falling_edge( CLK ) then
    if WB = '1' and END_CALC = '0' then
        OE_MB <= '1';
    end if;
end if;
end process;

-----< Generate Memory Cache Signal ( Matrix A ) >-----
process ( BL(0), CLK, END_CALC, END_READ, W_CYCLE_B, R_CYCLE_B ) begin
if BL(0) = '1' or END_CALC = '1' or END_READ = '1' or W_CYCLE_B = '1' then
    OE_CALCB <= '0';
elsif falling_edge( R_CYCLE_B ) then
    if OE_MB = '0' and END_CALC = '0' then
        OE_CALCB <= '1';
    end if;
end if;
end process;

-----< Selecter of Memory Operation ( Matrix B ) >-----
process ( BL(0), CLK ) begin
if BL(0) = '1' then
    MEM_CYCLE_B <= '0';
    MEM_STATE_SEL_B <= "000";
    SRAM_ADRS_MUX_B <= "00000000000000000";
    ROW_ADRS_MUX_B <= "000000000000";
    COL_ADRS_MUX_B <= "000000000000";
    W_MB <= "00000000000000000000000000000000";
elsif rising_edge( CLK ) then
    if WE_MB = '1' then
        W_MB <= MB_H & MB_L;
        MEM_CYCLE_B <= '1';
        MEM_STATE_SEL_B <= "101";
        ROW_ADRS_MUX_B <= "00" & ROW_B;
        COL_ADRS_MUX_B <= "00" & COL_B;
    end if;
    if OE_MB = '1' then
        MEM_CYCLE_B <= '1';
        SRAM_ADRS_MUX_B <= "1100000" & SRAM_CNT;
        ROW_ADRS_MUX_B <= "00" & ROW_B_CNT;
        COL_ADRS_MUX_B <= "00" & COL_B_CNT;
        MEM_STATE_SEL_B <= "111";
    end if;
    if OE_CALCB = '1' then
        MEM_CYCLE_B <= '1';
        SRAM_ADRS_MUX_B <= "1100000" & SRAM_CACHE_CNT;
        MEM_STATE_SEL_B <= "011";
    end if;
    if WE_RES = '1' then
        ROW_ADRS_MUX_B <= "000000000000";
        COL_ADRS_MUX_B <= "000000000000";
        MEM_CYCLE_B <= '1';
        MEM_STATE_SEL_B <= "101";
    end if;
    if OE_RES = '1' then
        MEM_CYCLE_B <= '1';
        MEM_STATE_SEL_B <= "110";
    end if;
    if W_CYCLE_B = '1' or R_CYCLE_B = '1' or REF_CYCLE_B = '1' then
        MEM_CYCLE_B <= '0';
        MEM_STATE_SEL_B <= "000";
    end if;
    if REFRESH_B = '1' then
        MEM_CYCLE_B <= '1';
        MEM_STATE_SEL_B <= "100";
    end if;
```

```
end if;
end process;

--------------< Calculation >---------------

-----< Memory Cache Read Counter >-----
process ( BL(0), CLK ) begin
if BL(0) = '1' then
    COL_A_CNT <= "0000000000"; ROW_A_CNT <= "0000000001";
    COL_B_CNT <= "0000000001"; ROW_B_CNT <= "0000000000";
    SRAM_CNT <= "0000000000";
    CACHE_WAIT <= '0';
    END_CALC <= '0';

elsif falling_edge( CLK ) then
    if MEM_STATE_SEL_A = "111" and CACHE_Q = '0' and END_CALC = '0' then
        if COL_A_CNT < COL_A then
            COL_A_CNT <= COL_A_CNT + "0000000001";
            ROW_B_CNT <= ROW_B_CNT + "0000000001";
            SRAM_CNT <= SRAM_CNT + "0000000001";
            if COL_A_CNT = COL_A - "0000000001" then
                CACHE_WAIT <= '1';
            else
                CACHE_WAIT <= '0';
            end if;
        end if;
    elsif WE_RES = '1' then
        CACHE_WAIT <= '0';
        if COL_B_CNT < ROW_A then
            COL_A_CNT <= "0000000000";
            ROW_B_CNT <= "0000000000";
            COL_B_CNT <= COL_B_CNT + "0000000001";
            SRAM_CNT <= "0000000000";
        elsif ROW_A_CNT < ROW_A then
            COL_A_CNT <= "0000000000";
            COL_B_CNT <= "0000000001";
            ROW_B_CNT <= "0000000000";
            ROW_A_CNT <= ROW_A_CNT + "0000000001";
            SRAM_CNT <= "0000000000";
        else
            END_CALC <= '1';
        end if;
    end if;
end if;
end process;

process ( BL(0), CLK, CACHE_WAIT ) begin
if BL(0) = '1' then
    CACHE_Q <= '0';
elsif falling_edge( CLK ) then
    if CACHE_WAIT = '1' then
        CACHE_Q <= '1';
    else
        CACHE_Q <= '0';
    end if;
end if;
end process;

-----< Memory Cache Read Counter >-----
process ( BL(0), CLK ) begin
if BL(0) = '1' then
    SRAM_CACHE_CNT <= "0000000001";
    WQ <= '0'; OUT_DCNT <= "0000";
    START_CALC <= '0';
    END_READ <= '0';
elsif falling_edge( CLK ) then
    if WQ = '1' then
        END_READ <= '1';
        START_CALC <= '0';
        OUT_DCNT <= OUT_DCNT + "0001";
    end if;

    if OUT_DCNT = DELAY_TIME then
        OUT_DCNT <= "0000";
        SRAM_CACHE_CNT <= "0000000001";
```

```
        WQ <= '0';
        --TWQ <= '0';
        END_READ <= '0';
    end if;

    if MEM_STATE_SEL_A = "011" and WQ = '0' and END_CALC = '0' then
        if SRAM_CACHE_CNT < COL_A then
            START_CALC <= '1';
            SRAM_CACHE_CNT <= SRAM_CACHE_CNT + "0000000001";
        else
            WQ <= '1'; --TWQ <= '1';
        end if;
    end if;
end if;
end process;

----< Input Matrix Data to Multiplier and Adder ( component
multadder2 ) >---
process ( BL(0), CLK ) begin
if BL(0) = '1' then
    FA <= "00000000000000000000000000000000";
    FB <= "00000000000000000000000000000000";
elsif falling_edge( CLK ) then
    if START_CALC = '1' then
        FA <= DATA_A;
        FB <= DATA_B;
    else
        FA <= "00000000000000000000000000000000";
        FB <= "00000000000000000000000000000000";
    end if;
end if;
end process;

---------< Result Matrix Row and Column Counter ( WR mode ) >---------
process ( BL(0), W_CYCLE_A, END_CALC ) begin
if BL(0) = '1' then
    COL_Q <= "0000000001";
    ROW_Q <= "0000000001";
    BH0 <= '0';
elsif falling_edge( W_CYCLE_A ) then
    if WB = '1' then
        if COL_Q < COL_B then
            COL_Q <= COL_Q + "0000000001";
        elsif ROW_Q < ROW_A then
            COL_Q <= "0000000001";
            ROW_Q <= ROW_Q + "0000000001";
        else
            BH0 <= '1';
        end if;
    end if;
elsif END_CALC = '1' then
    BH0 <= '1';
end if;
end process;

process ( BL(1), BL(4) ) begin
if BL(1) = '1' then
    OUT_READY <= '0';
elsif BL(4) = '1' then
    OUT_READY <= '1';
end if;
end process;

-----------< Output Result Data to PC >------------
process ( BL(0), R_CYCLE_A ) begin
if BL(0) = '1' then
    Q_CNT <= '0';
    A_REG <= "0000000000000000";
    BH2 <= '0';
elsif falling_edge( R_CYCLE_A ) then
    if OUT_READY = '1' and END_OUT = '0' then
        if OUT_CNT = '1' then
            A_REG <= R_MA(15 downto 0);
            Q_CNT <= '1'; BH2 <= '1';
        else
```

```
            A_REG <= R_MA(31 downto 16);
            Q_CNT <= '0'; BH2 <= '0';
        end if;
    end if;
end if;
end process;

BH <= "00000" & BH2 & BH1 & BH0;

----------< Result Matrix Row and Column Counter ( RD mode ) >---------
process ( BL(0), Q_CNT ) begin
if BL(0) = '1' then
    ROW_Q_CNT <= "0000000001";
    COL_Q_CNT  <= "0000000001";
    END_OUT <= '0';
elsif falling_edge( Q_CNT ) then
    if COL_Q_CNT < COL_B then
        COL_Q_CNT <= COL_Q_CNT + "0000000001";
    elsif ROW_Q_CNT < ROW_A then
        COL_Q_CNT <= "0000000001";
        ROW_Q_CNT <= ROW_Q_CNT + "0000000001";
    else
        END_OUT <= '1';
    end if;
end if;
end process;

---------< Handshake Operation >---------
process ( BL(0), IBF, R_CYCLE_A ) begin
if BL(0) = '1' or IBF = "11" then
    STB <= "11";
elsif falling_edge( R_CYCLE_A ) then
    if OUT_READY = '1' and END_OUT = '0' then
        STB <= "00";
    end if;
end if;
end process;

---------< Refresh Counter >---------
process ( CLK, REF_CNT_A, REF_CYCLE_A, W_CYCLE_A, R_CYCLE_A) begin
if REF_CYCLE_A = '1' or W_CYCLE_A = '1' or R_CYCLE_A = '1' then
    REF_CNT_A <= 0;    REFRESH_A <= '0';
elsif rising_edge( CLK ) then
    REF_CNT_A <= REF_CNT_A + 1;
elsif REF_CNT_A = 62 then
    REF_CNT_A <= 0;    REFRESH_A <= '1';
end if;
end process;

process ( CLK, REF_CNT_B, REF_CYCLE_B, W_CYCLE_B, R_CYCLE_B) begin
if REF_CYCLE_B = '1' or W_CYCLE_B = '1' or R_CYCLE_B = '1' then
    REF_CNT_B <= 0;    REFRESH_B <= '0';
elsif rising_edge( CLK ) then
    REF_CNT_B <= REF_CNT_B + 1;
elsif REF_CNT_B = 62 then
    REF_CNT_B <= 0;    REFRESH_B <= '1';
end if;
end process;

------------< Memory Controller >--------------
MEM_A : ctrl_dramsram port map ( CLK => CLK, RESET =>  BL(0),
        ADRS => ADRS_A, DATA => DATA_A, DATA_BUF => DATA_BUF_A,
        SRAM_ADRS_MUX => SRAM_ADRS_MUX_A, ROW_ADRS_MUX => ROW_ADRS_MUX_A,
        COL_ADRS_MUX => COL_ADRS_MUX_A, WRITE_DATA_REG => W_MA,
        READ_DATA_REG => R_MA, SCS => SCS_A, SOE => SOE_A, SWE => SWE_A,
        OE => OE_A, DWE => DWE_A, DRAS => DRAS_A, DCAS => DCAS_A,
        DOE => DOE_A, MEM_STATE_SEL => MEM_STATE_SEL_A,
        MEM_CYCLE => MEM_CYCLE_A, W_CYCLE => W_CYCLE_A,
        R_CYCLE => R_CYCLE_A, REF_CYCLE => REF_CYCLE_A );

MEM_B : ctrl_dramsram port map ( CLK => CLK, RESET =>  BL(0),
        ADRS => ADRS_B, DATA => DATA_B, DATA_BUF => DATA_BUF_B,
        SRAM_ADRS_MUX => SRAM_ADRS_MUX_B, ROW_ADRS_MUX => ROW_ADRS_MUX_B,
        COL_ADRS_MUX => COL_ADRS_MUX_B, WRITE_DATA_REG => W_MB,
        READ_DATA_REG => R_MB, SCS => SCS_B, SOE => SOE_B, SWE => SWE_B,
```

```
        OE => OE_B, DWE => DWE_B, DRAS => DRAS_B, DCAS => DCAS_B,
        DOE => DOE_B, MEM_STATE_SEL => MEM_STATE_SEL_B,
        MEM_CYCLE => MEM_CYCLE_B, W_CYCLE => W_CYCLE_B,
        R_CYCLE => R_CYCLE_B, REF_CYCLE => REF_CYCLE_B );

------------< Multiplier and Adder >-------------
multadd : multadd3 port map ( CLK => CLK, WQ => WQ, FA => FA, FB => FB, FD => FD, Q => QQ );
end RTL;
```

付録 B

プログラムソース（ハウスホルダ変換）

B.1 メモリコントローラ (SRAM,DRAM 兼用)

```
 ------------------------------------------------------------
-- DRAM&SRAM Memory Controller  (FLEX10k)
-- < ctdrmsrm4.vhd >
-- 2000/01/31 (Mon)
-- numa@tube.ee.uec.ac.jp(DRAM Controller)
-- yamaoka@tube.ee.uec.ac.jp(SRAM Controller)
------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library metamor;
use metamor.attributes.all;

entity memctrld is
port (  CLK           : in std_logic;
        RESET         : in std_logic;
        ADRS          : out std_logic_vector(16 downto 0);
        DATA          : inout std_logic_vector(31 downto 0);
        SRAM_ADRS_BUF : in std_logic_vector(16 downto 0);
        ROW_ADRS_BUF  : in std_logic_vector(11 downto 0);
        COL_ADRS_BUF  : in std_logic_vector(11 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;
        DWE           : out std_logic;
        DRAS          : out std_logic_vector(3 downto 0);
        DCAS          : out std_logic_vector(3 downto 0);
        MEM_STATE_SEL : in std_logic_vector(2 downto 0);
        WR_CYCLE       : out std_logic;
        RD_CYCLE       : out std_logic;
        RF_CYCLE       : out std_logic
);
end memctrld;

architecture RTL of memctrld is

type STATE_TYPE is (
  STOP, S_WRITE1, S_WRITE2, S_READ1, S_READ2,
  D_WRITE1, D_WRITE2, D_WRITE3, D_WRITE4, D_WRITE5,
  D_READ1, D_READ2, D_READ3, D_READ4, D_READ5,
  REF1, REF2, REF3, REF4, REF5, D_RD_WR1, D_RD_WR2,
  D_RD_WR3, D_RD_WR4, D_RD_WR5, D_RD_WR6, D_RD_WR7,
```

```
  D_RD_WR8, D_RD_WR9, CONT_S_READ
  );

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

signal ADRS_COL : std_logic;
signal ADRS_ROW : std_logic;
signal ADRS_SRAM : std_logic;

signal W_CYCLE1 : std_logic;
signal CACHE_DATA : std_logic_vector(31 downto 0);

constant S_WRITE_CYCLE     : std_logic_vector(2 downto 0) := "000";
constant S_READ_CYCLE      : std_logic_vector(2 downto 0) := "001";
constant CONT_READ_CYCLE    : std_logic_vector(2 downto 0) := "010";
constant D_WRITE_CYCLE     : std_logic_vector(2 downto 0) := "100";
constant D_READ_CYCLE      : std_logic_vector(2 downto 0) := "101";
constant D_REF_CYCLE         : std_logic_vector(2 downto 0) := "110";
constant D_RD_WR_CYCLE       : std_logic_vector(2 downto 0) := "111";

begin

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

process ( CLK, RESET, CURRENT_STATE ) begin

if RESET = '1' then
    SOE <= '1'; SWE <= '1';
    SCS <= "1111"; OE <= '1';
    DRAS <= "1111"; DCAS <= "1111";
    DATA_BUF <= ( others => '0');
    RD_DATA <= ( others => '0' );
    CACHE_DATA <= ( others => '0' );
    WR_CYCLE <= '0'; RD_CYCLE <= '0';
    RF_CYCLE <= '0';
    NEXT_MEM_CYCLE <= '0';
    NEXT_STATE <= STOP;
    ADRS_COL <= '0'; ADRS_ROW <= '0';
    ADRS_SRAM <= '0';

-----< SRAM CYCLE >-----
elsif rising_edge( CLK ) then

case CURRENT_STATE is

    when STOP =>
        SOE <= '1'; SWE <= '1';
        SCS <= "1111"; OE <= '1';
        DRAS <= "1111"; DCAS <= "1111";
        WR_CYCLE <= '0'; RD_CYCLE <= '0';
        RF_CYCLE <= '0';
        ADRS_COL <= '0'; ADRS_ROW <= '0';
        ADRS_SRAM <= '0';
        NEXT_STATE <= STOP;
        NEXT_MEM_CYCLE <= '0';

    when S_WRITE1 =>
        SCS <= "0000";
        DATA_BUF <= WR_DATA;
        OE <= '0';
        ADRS_SRAM <= '1';
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= S_WRITE2;

    when S_WRITE2 =>
        SWE <= '0';
        WR_CYCLE <= '1';
        NEXT_STATE <= STOP;

    when S_READ1 =>
```

```
        SCS <= "0000";
        SOE <= '0';
        OE <= '1';
        ADRS_SRAM <= '1';
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= S_READ2;

    when S_READ2 =>
        RD_DATA <= DATA;
        RD_CYCLE <= '1';
        NEXT_STATE <= STOP;

-----< DRAM CYCLE >-----

    when D_WRITE1 =>
        SOE <= '1'; SWE <= '1';
        SCS <= "1111";
        OE <= '0';
        ADRS_SRAM <= '0'; ADRS_ROW <= '1';
        DATA_BUF <= WR_DATA;
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= D_WRITE2;
    when D_WRITE2 =>
        ADRS_COL <= '1'; ADRS_ROW <= '0';
        DRAS <= "0000";
        WR_CYCLE <= '1';
        NEXT_STATE <= D_WRITE3;
    when D_WRITE3 =>
        DCAS <= "0000";
        NEXT_STATE <= D_WRITE4;
    when D_WRITE4 =>
        ADRS_COL <= '0';
        DRAS <= "1111";
        NEXT_STATE <= D_WRITE5;
    when D_WRITE5 =>
        DCAS <= "1111";
        NEXT_STATE <= STOP;
--< READ CYCLE to SRAM >--
    when D_READ1 =>
        ADRS_SRAM <= '0';
        OE <= '1';
        ADRS_ROW <= '1';
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= D_READ2;
    when D_READ2 =>
        ADRS_COL <= '1';
        ADRS_ROW <= '0';
        DRAS <= "0000";
        RD_CYCLE <= '1';
        NEXT_STATE <= D_READ3;
    when D_READ3 =>
        DCAS <= "0000";
        NEXT_STATE <= D_READ4;
    when D_READ4 =>
        ADRS_COL <= '0';
        RD_DATA <= DATA;
        DRAS <= "1111";
        NEXT_STATE <= D_READ5;
    when D_READ5 =>
        DCAS <= "1111";
        NEXT_STATE <= STOP;

--< REFRESH WRITE CYCLE for DRAM >--
    when REF1 =>
        ADRS_SRAM <= '0';
        OE <= '1';
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= REF2;
    when REF2 =>
        DCAS <= "0000";
        RF_CYCLE <= '1';
        NEXT_STATE <= REF3;
    when REF3 =>
        DRAS <= "0000";
        NEXT_STATE <= REF4;
    when REF4 =>
```

```
        DCAS <= "1111";
        NEXT_STATE <= REF5;
    when REF5 =>
        DRAS <= "1111";
        NEXT_STATE <= STOP;
    when D_RD_WR1 =>
        ADRS_SRAM <= '0';
        OE <= '1';
        ADRS_ROW <= '1';
        NEXT_MEM_CYCLE <= '1';
        NEXT_STATE <= D_RD_WR2;
    when D_RD_WR2 =>
        ADRS_COL <= '1';
        ADRS_ROW <= '0';
        DRAS <= "0000";
        RD_CYCLE <= '1';
        NEXT_STATE <= D_RD_WR3;
    when D_RD_WR3 =>
        DCAS <= "0000";
        NEXT_STATE <= D_RD_WR4;
    when D_RD_WR4 =>
        ADRS_COL <= '0';
        CACHE_DATA <= DATA;
        DRAS <= "1111";
        NEXT_STATE <= D_RD_WR5;
    when D_RD_WR5 =>
        DCAS <= "1111";
        ADRS_SRAM <= '1';
        NEXT_STATE <= D_RD_WR6;
    when D_RD_WR6 =>
       OE <= '0';
        NEXT_STATE <= D_RD_WR7;
    when D_RD_WR7 =>
       SCS <= "0000";
        DATA_BUF <= CACHE_DATA;
        NEXT_STATE <= D_RD_WR8;
    when D_RD_WR8 =>
        SWE <= '0';
        NEXT_STATE <= D_RD_WR9;
    when D_RD_WR9 =>
        ADRS_SRAM <= '0';
        SWE <= '1';
        SCS <= "1111";
        OE <= '1';
        NEXT_STATE <= STOP;
    when CONT_S_READ =>
        SOE <= '0'; SWE <= '1';
        SCS <= "0000";
        OE <= '1';
        ADRS_SRAM <= '0';

    when others =>
        NEXT_STATE <= STOP;

end case;
end if;
end process;

process (CLK, RESET, NEXT_MEM_CYCLE ) begin
-----< RAM Decision Controller >-----
if RESET = '1' then
    CURRENT_STATE <= STOP;

elsif falling_edge( CLK ) then
    if NEXT_MEM_CYCLE = '0' then
        case MEM_STATE_SEL is
            when S_WRITE_CYCLE =>
                CURRENT_STATE <= S_WRITE1;
            when S_READ_CYCLE =>
                CURRENT_STATE <= S_READ1;
            when CONT_READ_CYCLE =>
                CURRENT_STATE <= CONT_S_READ;
            when D_WRITE_CYCLE =>
                CURRENT_STATE <= D_WRITE1;
            when D_READ_CYCLE =>
                CURRENT_STATE <= D_READ1;
```

```
            when D_REF_CYCLE =>
                CURRENT_STATE <= REF1;
            when D_RD_WR_CYCLE =>
                CURRENT_STATE <= D_RD_WR1;
            when others =>
                CURRENT_STATE <= STOP;
        end case;
    elsif NEXT_MEM_CYCLE = '1' then
        CURRENT_STATE <= NEXT_STATE;
    end if;
end if;
end process;

process (CLK,RESET ) begin
if RESET = '1' then
    DWE <= '1';

elsif falling_edge ( CLK ) then
    if ADRS_ROW = '1' then
        ADRS <= "00000" & ROW_ADRS_BUF;
    elsif ADRS_COL = '1' then
        ADRS <= "00000" & COL_ADRS_BUF;
        if W_CYCLE1 = '1' then
            DWE <= '0';
        else
            DWE <= '1';
        end if;
    elsif ADRS_SRAM = '1' or MEM_STATE_SEL = CONT_READ_CYCLE then
        ADRS <= SRAM_ADRS_BUF;
        DWE <= '1';
    else
        ADRS <= "0000000000000000";
        DWE <= '1';
    end if;
end if;
end process;

end RTL;
```

B.2 単精度浮動小数点演算器

B.2.1 加算器

このハウスホルダ変換に使われている単精度浮動小数点演算器については、すべて昨年山岡 [5] が設計したものである。

```
------------------------------------------------------------
-- 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 + "0000000000000000000000001";

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;
```

B.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;
```

B.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;
```

B.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;
```

B.3 ハウスホルダー法

B.3.1 積和器

この積和器はDRAMコントローラに対応するために、昨年山岡が設計したものを改良したものである。

```
-------------------------------------------------------------
-- ALU for Householder Method  (upper FLEX10k)
-- < alu101.vhd >
-- 2000/01/31 (Mon)
-- numa@tube.ee.uec.ac.jp(modified for DRAM Controller)
-- 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 alu101 is
  port (CLK    : in std_logic;

        DATA_BUS  : inout std_logic_vector(31 downto 0);
        ADRS_BUS  : in std_logic_vector(15 downto 0);
        CTRL_BUS  : in std_logic_vector(5 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;
        R_DWE  : out std_logic;
        R_DRAS : out std_logic_vector(3 downto 0);
        R_DCAS : out std_logic_vector(3 downto 0);

        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;
        L_DWE  : out std_logic;
        L_DRAS : out std_logic_vector(3 downto 0);
        L_DCAS : out std_logic_vector(3 downto 0)
);

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 "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,AU15";
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 R_DWE  : signal is  "AF40";
attribute pinnum of R_DRAS : signal is  "P40,R39,T40,U39";
attribute pinnum of R_DCAS : signal is  "Y40,AA39,AB40,AC39";

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";
attribute pinnum of L_DWE  : signal is  "AF4";
attribute pinnum of L_DRAS : signal is  "P4,R5,T4,U5";
attribute pinnum of L_DCAS : signal is  "Y4,AA5,AB4,AC5";

end alu101;
architecture RTL of alu101 is

--------------< Memory Controller >------------
component memctrld is
port (  CLK           : in std_logic;
        RESET         : in std_logic;
        ADRS          : out std_logic_vector(16 downto 0);
        SRAM_ADRS_BUF : in std_logic_vector(16 downto 0);
        ROW_ADRS_BUF  : in std_logic_vector(11 downto 0);
        COL_ADRS_BUF  : in std_logic_vector(11 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;
        DRAS          : out std_logic_vector(3 downto 0);
        DCAS          : out std_logic_vector(3 downto 0);
        MEM_STATE_SEL : in std_logic_vector(2 downto 0);
        WR_CYCLE      : out std_logic;
        RD_CYCLE      : out std_logic;
        RF_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_SRAM_ADRS_BUF : std_logic_vector(16 downto 0);
signal R_ROW_ADRS_BUF : std_logic_vector(11 downto 0);
signal R_COL_ADRS_BUF : std_logic_vector(11 downto 0);
signal L_SRAM_ADRS_BUF : std_logic_vector(16 downto 0);
signal L_ROW_ADRS_BUF : std_logic_vector(11 downto 0);
signal L_COL_ADRS_BUF : std_logic_vector(11 downto 0);
signal R_MEM_STATE_SEL : std_logic_vector(2 downto 0);
signal L_MEM_STATE_SEL : std_logic_vector(2 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_RF_CYCLE : std_logic;
signal L_RF_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);
signal R_ADRS_CNT : std_logic;
signal L_ADRS_CNT : std_logic;

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 S_WRITE_CYCLE     : std_logic_vector(2 downto 0) := "000";
constant S_READ_CYCLE      : std_logic_vector(2 downto 0) := "001";
constant CONT_READ_CYCLE    : std_logic_vector(2 downto 0) := "010";
constant D_WRITE_CYCLE     : std_logic_vector(2 downto 0) := "100";
constant D_READ_CYCLE      : std_logic_vector(2 downto 0) := "101";
constant D_REF_CYCLE         : std_logic_vector(2 downto 0) := "110";
constant D_RD_WR_CYCLE       : std_logic_vector(2 downto 0) := "111";
constant STOP_CYCLE      : std_logic_vector(2 downto 0) := "011";

constant cR_MEM_WR     : std_logic_vector(5 downto 0) := "000001";
constant cR_MEM_RD     : std_logic_vector(5 downto 0) := "000010";
constant cL_MEM_WR     : std_logic_vector(5 downto 0) := "000011";
constant cL_MEM_RD     : std_logic_vector(5 downto 0) := "000100";
constant cLR_MEM_WR    : std_logic_vector(5 downto 0) := "000101";
constant cMEM_STOP     : std_logic_vector(5 downto 0) := "000110";
constant cRR_INPRO     : std_logic_vector(5 downto 0) := "000111";
constant cLL_INPRO     : std_logic_vector(5 downto 0) := "001000";
constant cLR_INPRO     : std_logic_vector(5 downto 0) := "001001";
constant cINPRO_F      : std_logic_vector(5 downto 0) := "001010";
constant cINPRO_MEM_R  : std_logic_vector(5 downto 0) := "001011";
constant cINPRO_MEM_L  : std_logic_vector(5 downto 0) := "001100";
constant cINPRO_MEM_LR : std_logic_vector(5 downto 0) := "001101";
constant cFF_MUL       : std_logic_vector(5 downto 0) := "001110";
constant cFR_MEM_MUL   : std_logic_vector(5 downto 0) := "001111";
constant cFL_MEM_MUL   : std_logic_vector(5 downto 0) := "010000";
constant cRR_MEM_MUL   : std_logic_vector(5 downto 0) := "010001";
constant cLL_MEM_MUL   : std_logic_vector(5 downto 0) := "010010";
constant cLR_MEM_MUL   : std_logic_vector(5 downto 0) := "010011";
constant cFF_ADD       : std_logic_vector(5 downto 0) := "010100";
constant cFR_MEM_ADD   : std_logic_vector(5 downto 0) := "010101";
constant cFL_MEM_ADD   : std_logic_vector(5 downto 0) := "010110";
constant cRR_MEM_ADD   : std_logic_vector(5 downto 0) := "010111";
constant cLL_MEM_ADD   : std_logic_vector(5 downto 0) := "011000";
constant cLR_MEM_ADD   : std_logic_vector(5 downto 0) := "011001";
constant cCALC_F       : std_logic_vector(5 downto 0) := "011010";
constant cCALC_MEM_R   : std_logic_vector(5 downto 0) := "011011";
constant cCALC_MEM_L   : std_logic_vector(5 downto 0) := "011100";
constant cCALC_MEM_LR  : std_logic_vector(5 downto 0) := "011101";
-- new command
constant cLR_DRAM_WR   : std_logic_vector(5 downto 0) := "011110";
constant cL_DRAM_WR    : std_logic_vector(5 downto 0) := "011111";
constant cR_DRAM_WR    : std_logic_vector(5 downto 0) := "100000";
constant cL_DRAM_RD    : std_logic_vector(5 downto 0) := "100001";
constant cR_DRAM_RD    : std_logic_vector(5 downto 0) := "100010";
constant cLR_DRAM_WR   : std_logic_vector(5 downto 0) := "100011";
constant cINPRO_DRAM_R : std_logic_vector(5 downto 0) := "100100";
constant cINPOT_DRAM_L : std_logic_vector(5 downto 0) := "100101";
constant cFR_DRAM_MUL  : std_logic_vector(5 downto 0) := "100110";
constant cFL_DRAM_MUL  : std_logic_vector(5 downto 0) := "100111";
constant cLR_DRAM_MU_  : std_logic_vector(5 downto 0) := "101000";
constant cCALC_DRAM_R  : std_logic_vector(5 downto 0) := "101001";
constant cCALC_DRAM_L  : std_logic_vector(5 downto 0) := "101010";

begin

RESET <= '1' when CTRL_BUS = "000000" 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 <= S_WRITE_CYCLE;
        when cR_MEM_RD =>
            R_MEM_STATE_SEL <= S_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 <= S_WRITE_CYCLE;
            else
                R_MEM_STATE_SEL <= STOP_CYCLE;
            end if;
        when cCALC_R | cCALC_LR =>
            if DCNT5 = '1' then
                R_MEM_STATE_SEL <= S_WRITE_CYCLE;
            else
                R_MEM_STATE_SEL <= STOP_CYCLE;
            end if;
        when cLR_DRAM_WR =>
            R_MEM_STATE_SEL <= D_WRITE_CYCLE;
        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 | cLR_DRAM_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_SRAM_ADRS_BUF <= ( others => '0' );
    R_ADRS_CNT <= '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_SRAM_ADRS_BUF <= '0' & ADRS_BUS;
        when cINPRO_R | cINPRO_LR =>
            if DCNT2 = "1000" then
                R_SRAM_ADRS_BUF <= '0' & ADRS_BUS;
            end if;
        when cCALC_R | cCALC_LR =>
            if DCNT5 = '1' then
                R_SRAM_ADRS_BUF <= '0' & ADRS_BUS;
            end if;
        when cLR_DRAM_WR =>
            if R_ADRS_CNT = '1' then
                R_COL_ADRS_BUF <= '0' & ADRS_BUS;
                R_ADRS_CNT <= '0';
            else
                R_ROW_ADRS_BUF <= '0' & ADRS_BUS;
                R_ADRS_CNT <= '1';
            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 <= S_WRITE_CYCLE;
        when cL_MEM_RD =>
            L_MEM_STATE_SEL <= S_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 <= S_WRITE_CYCLE;
            else
                L_MEM_STATE_SEL <= STOP_CYCLE;
            end if;
        when cCALC_L | cCALC_LR =>
            if DCNT5 = '1' then
                L_MEM_STATE_SEL <= S_WRITE_CYCLE;
            else
                L_MEM_STATE_SEL <= STOP_CYCLE;
            end if;
        when cLR_DRAM_WR =>
            L_MEM_STATE_SEL <= D_WRITE_CYCLE;
        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_SRAM_ADRS_BUF <= ( others => '0' );
    L_ADRS_CNT <= '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_SRAM_ADRS_BUF <= '0' & ADRS_BUS;
        when cLR_INPRO | cLR_MUL | cLR_ADD =>
            L_SRAM_ADRS_BUF <= DATA_BUS(16 downto 0);
        when cINPRO_L =>
            if DCNT2 = "1000" then
                L_SRAM_ADRS_BUF <= '0' & ADRS_BUS;
```

```
            end if;
        when cINPRO_LR =>
            if DCNT2 = "1000" then
                L_SRAM_ADRS_BUF <= DATA_BUS(16 downto 0);
            end if;
        when cCALC_L =>
            if DCNT5 = '1' then
                L_SRAM_ADRS_BUF <= '0' & ADRS_BUS;
            end if;
        when cCALC_LR =>
            if DCNT5 = '1' then
                L_SRAM_ADRS_BUF <= DATA_BUS(16 downto 0);
            end if;
        when cLR_DRAM_WR =>
            if L_ADRS_CNT = '1' then
                L_COL_ADRS_BUF <= '0' & ADRS_BUS;
                L_ADRS_CNT <= '0';
            else
                L_ROW_ADRS_BUF <= '0' & ADRS_BUS;
                L_ADRS_CNT <= '1';
            end if;
        when others =>
            null;
    end case;
end if;
end process;
------------< Memory Controller >--------------
R_MEM : memctrld port map (
            CLK => CLK, RESET => RESET,
            ADRS => R_ADRS, SRAM_ADRS_BUF => R_SRAM_ADRS_BUF,
            ROW_ADRS_BUF => R_ROW_ADRS_BUF, COL_ADRS_BUF => R_COL_ADRS_BUF,
            DATA => R_DATA, WR_DATA => R_WR_DATA, RD_DATA => R_RD_DATA,
            SCS => R_SCS, SOE => R_SOE, SWE => R_SWE,
            DRAS => R_DRAS, DCAS => R_DCAS,
            MEM_STATE_SEL => R_MEM_STATE_SEL,
            WR_CYCLE => R_WR_CYCLE, RD_CYCLE => R_RD_CYCLE,
            RF_CYCLE => R_RF_CYCLE
            );
L_MEM : memctrld port map (
            CLK => CLK, RESET => RESET,
            ADRS => L_ADRS, SRAM_ADRS_BUF => L_SRAM_ADRS_BUF,
            ROW_ADRS_BUF => L_ROW_ADRS_BUF, COL_ADRS_BUF => L_COL_ADRS_BUF,
            DATA => L_DATA, WR_DATA => L_WR_DATA, RD_DATA => L_RD_DATA,
            SCS => L_SCS, SOE => L_SOE, SWE => L_SWE,
            DRAS => L_DRAS, DCAS => L_DCAS,
            MEM_STATE_SEL => L_MEM_STATE_SEL,
            WR_CYCLE => L_WR_CYCLE, RD_CYCLE => L_RD_CYCLE,
            RF_CYCLE => L_RF_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;
```

B.3.2 ハウスホルダー変換

このハウスホルダ変換も DRAM コントローラに対応するために、昨年山岡が設計したものを改良したものである。

```
-------------------------------------------------------------
-- Householder Transform  (Lower FLEX10k)
-- < hshld101.vhd >
-- 2000/01/31 (Mon)
-- numa@tube.ee.uec.ac.jp(modified for DRAM Controller)
-- 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 hshld101 is
  port (CLK  : in std_logic;
        A    : inout std_logic_vector(15 downto 0);
        BL   : in std_logic_vector(7 downto 0);
        BH   : in std_logic_vector(7 downto 0);
        CL   : in std_logic_vector(5 downto 0);

        DATA_BUS  : inout std_logic_vector(31 downto 0); -- connection B & A
        ADRS_BUS  : out std_logic_vector(15 downto 0);   -- connection D(0) & C
        CTRL_BUS  : out std_logic_vector(5 downto 0);    -- connection D(5 downto 1)
        CALC_DONE : in  std_logic;                           -- connection D(6)
        OE_ALU    : out std_logic;                           -- connection D(7)

        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  "BC5,BB6,BC7,BB8,BC9,BB10,BC11,BB12";
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 "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,F16";
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 hshld101;
architecture RTL of hshld101 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 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;--_vector(4 downto 0);
signal OUT_ACK1 : std_logic;
signal OUT_ACK2 : std_logic;
```

```
signal DCNT : std_logic;

signal N : std_logic_vector(11 downto 0);
signal ROW : std_logic_vector(11 downto 0);

signal WA : std_logic;

signal MA_L : std_logic_vector(15 downto 0);
signal MA_H : std_logic_vector(15 downto 0);

signal MA_WR : std_logic;
-------------------------------

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,
-- New command
        LR_DRAM_WR, L_DRAM_WR, R_DRAM_WR,
        L_DRAM_RD, R_DRAM_RD, LR_DRAM_RD,
        INPRO_DRAM_R, INPRO_DRAM_L,
        FR_DRAM_MUL, FL_DRAM_MUL, LR_DRAM_MUL,
        CALC_DRAM_R, CALC_DRAM_L, CALC_DRAM_LR,
        RR_DRAM_MEM, LL_DRAM_MEM, LR_DRAM_MEM,
        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 ROW_ADRS_BUF1 : std_logic_vector(15 downto 0);
signal COL_ADRS_BUF1 : std_logic_vector(15 downto 0);
signal SRAM_ADRS_BUF1 : std_logic_vector(15 downto 0);
signal ADRS_BUF2 : std_logic_vector(15 downto 0);
signal ALU_ACK : std_logic;
signal DRAM_ADRS_CNT : std_logic;

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

type HS_STATE_TYPE is (
        HsDimWrite,
        HsS2LoopIni, HsS2LoopBdy,
        HsSCalc, HsAkRead, HsSNorm,
        HsAkxSCalc, HsS2pAkxSCalc, HsCCalc, HsCWrite, HsViceWrite,
        HsAkpSCalc, HsAxUCalc, HsPCalc,
        HsPxUCalc, HsAlphaCalc, HsAlxCCalc, HsAlxCd2Calc,
        HsAlxCd2xUCalc, HsQCalc,
        HsUxQtCalc, HsQxUtCalc, HsUxQtpQxUtCalc, HsACalc, HsResRead,
        hStop );

signal HS_STATE : HS_STATE_TYPE;
signal S2, S, Ak, AkxS, S2pAkxS, C, AxU, AL, ALxC, ALxCd2, ALxCd2xU, 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(5 downto 0) := "000001";
constant cR_MEM_RD      : std_logic_vector(5 downto 0) := "000010";
```

```
constant cL_MEM_WR      : std_logic_vector(5 downto 0) := "000011";
constant cL_MEM_RD      : std_logic_vector(5 downto 0) := "000100";
constant cLR_MEM_WR     : std_logic_vector(5 downto 0) := "000101";
constant cMEM_STOP      : std_logic_vector(5 downto 0) := "000110";
constant cRR_INPRO      : std_logic_vector(5 downto 0) := "000111";
constant cLL_INPRO      : std_logic_vector(5 downto 0) := "001000";
constant cLR_INPRO      : std_logic_vector(5 downto 0) := "001001";
constant cINPRO_F       : std_logic_vector(5 downto 0) := "001010";
constant cINPRO_MEM_R   : std_logic_vector(5 downto 0) := "001011";
constant cINPRO_MEM_L   : std_logic_vector(5 downto 0) := "001100";
constant cINPRO_MEM_LR  : std_logic_vector(5 downto 0) := "001101";
constant cFF_MUL        : std_logic_vector(5 downto 0) := "001110";
constant cFR_MEM_MUL    : std_logic_vector(5 downto 0) := "001111";
constant cFL_MEM_MUL    : std_logic_vector(5 downto 0) := "010000";
constant cRR_MEM_MUL    : std_logic_vector(5 downto 0) := "010001";
constant cLL_MEM_MUL    : std_logic_vector(5 downto 0) := "010010";
constant cLR_MEM_MUL    : std_logic_vector(5 downto 0) := "010011";
constant cFF_ADD        : std_logic_vector(5 downto 0) := "010100";
constant cFR_MEM_ADD    : std_logic_vector(5 downto 0) := "010101";
constant cFL_MEM_ADD    : std_logic_vector(5 downto 0) := "010110";
constant cRR_MEM_ADD    : std_logic_vector(5 downto 0) := "010111";
constant cLL_MEM_ADD    : std_logic_vector(5 downto 0) := "011000";
constant cLR_MEM_ADD    : std_logic_vector(5 downto 0) := "011001";
constant cCALC_F        : std_logic_vector(5 downto 0) := "011010";
constant cCALC_MEM_R    : std_logic_vector(5 downto 0) := "011011";
constant cCALC_MEM_L    : std_logic_vector(5 downto 0) := "011100";
constant cCALC_MEM_LR   : std_logic_vector(5 downto 0) := "011101";
-- new command
constant cLR_DRAM_WR   : std_logic_vector(5 downto 0) := "011110";
constant cL_DRAM_WR    : std_logic_vector(5 downto 0) := "011111";
constant cR_DRAM_WR    : std_logic_vector(5 downto 0) := "100000";
constant cL_DRAM_RD    : std_logic_vector(5 downto 0) := "100001";
constant cR_DRAM_RD    : std_logic_vector(5 downto 0) := "100010";
constant cLR_DRAM_RD   : std_logic_vector(5 downto 0) := "100011";
constant cINPRO_DRAM_R : std_logic_vector(5 downto 0) := "100100";
constant cINPRO_DRAM_L : std_logic_vector(5 downto 0) := "100101";
constant cFR_DRAM_MUL  : std_logic_vector(5 downto 0) := "100110";
constant cFL_DRAM_MUL  : std_logic_vector(5 downto 0) := "100111";
constant cLR_DRAM_MUL  : std_logic_vector(5 downto 0) := "101000";
constant cCALC_DRAM_R  : std_logic_vector(5 downto 0) := "101001";
constant cCALC_DRAM_L  : std_logic_vector(5 downto 0) := "101010";
constant cCALC_DRAM_LR : std_logic_vector(5 downto 0) := "101011";
constant cRR_DRAM_MEM  : std_logic_vector(5 downto 0) := "101100";
constant cLL_DRAM_MEM  : std_logic_vector(5 downto 0) := "101101";
constant cLR_DRAM_MEM  : std_logic_vector(5 downto 0) := "101110";


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_L <= ( others => '0' );
    MA_H <= ( others => '0' );
elsif OBF'event and OBF = "11" then
    if WA = '0' then
        if IN_CNT = '0' then
            MA_L <= A;
            IN_CNT <= '1';
        else
            MA_H <= A;
```

```
            IN_CNT <= '0';
        end if;
    end if;
end if;
end process;

---------< Matrix A 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 Column 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;

---------< Selecter of Matrix >---------
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 ( Matrix A ) >-----
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;


process ( BL(0), CLK ) begin
if BL(0) = '1' then
    OUT_ACK1 <= '0';
    OUT_ACK2 <= '0';
elsif rising_edge( CLK ) then
    if BL(4) = '1' then
        OUT_ACK1 <= '1';
```

```
    else
        OUT_ACK1 <= '0';
    end if;
    if BL(4) = '1' and OUT_ACK1 = '0' 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 LR_DRAM_WR =>              -- New Command
            CTRL_BUS <= cLR_DRAM_WR;
        when R_DRAM_WR =>
            CTRL_BUS <= cR_DRAM_WR;
        when L_DRAM_WR =>
            CTRL_BUS <= cL_DRAM_WR;
        when LR_DRAM_RD =>
            CTRL_BUS <= cLR_DRAM_RD;
        when R_DRAM_RD =>
            CTRL_BUS <= cR_DRAM_RD;
        when L_DRAM_RD =>
            CTRL_BUS <= cL_DRAM_RD;
        when INPRO_DRAM_R =>
            CTRL_BUS <= cINPRO_DRAM_R;
        when INPRO_DRAM_L =>
            CTRL_BUS <= cINPRO_DRAM_L;
        when FR_DRAM_MUL =>
            CTRL_BUS <= cFR_DRAM_MUL;
        when FL_DRAM_MUL =>
            CTRL_BUS <= cFL_DRAM_MUL;
        when LR_DRAM_MUL =>
            CTRL_BUS <= cLR_DRAM_MUL;
        when CALC_DRAM_R =>
            CTRL_BUS <= cCALC_DRAM_R;
        when CALC_DRAM_L =>
            CTRL_BUS <= cCALC_DRAM_L;
        when CALC_DRAM_LR =>
            CTRL_BUS <= cCALC_DRAM_LR;
        when RR_DRAM_MEM =>
            CTRL_BUS <= cRR_DRAM_MEM;
        when LL_DRAM_MEM =>
            CTRL_BUS <= cLL_DRAM_MEM;
        when LR_DRAM_MEM =>
            CTRL_BUS <= cLR_DRAM_MEM;
        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
 | LR_DRAM_WR =>
            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(15 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' );
    DRAM_ADRS_CNT <= '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 LR_DRAM_WR =>
            if DRAM_ADRS_CNT = '1' then
                ADRS_BUS <= COL_ADRS_BUF1;
                DRAM_ADRS_CNT <= '0';
            else
                ADRS_BUS <= ROW_ADRS_BUF1;
                DRAM_ADRS_CNT <= '1';
            end if;
        when others =>
            ADRS_BUS <= SRAM_ADRS_BUF1;
    end case;
end if;
end process;

------------< Householder Transform >-------------

process( BL(0), CLK )
    variable i, j, k : std_logic_vector(11 downto 0);
begin
if BL(0) = '1' then
    HS_STATE <= hStop;
    ALU_STATE <= aStop;
    ALU_ACK <= '0';
    DIV_ACK <= '0';
    DATA_BUF1 <= (others => '0');
    DATA_BUF2 <= (others => '0');
    SRAM_ADRS_BUF1 <= (others => '0');
    ROW_ADRS_BUF1 <= (others => '0');
    COL_ADRS_BUF1 <= (others => '0');
    ADRS_BUF2 <= (others => '0');
    i  := ( others => '0' );
    i2 := ( others => '0' );
    j  := ( others => '0' );
    k  := ( 0 => '1', others => '0' );
elsif rising_edge( CLK ) then
    case HS_STATE is
        when hStop =>
            if WA = '1' then
                HS_STATE <= HsDimWrite;
            elsif MA_WR = '1' then
                DATA_BUF1 <= MA_H & MA_L;
                ROW_ADRS_BUF1 <= "0000" & ROW;
                COL_ADRS_BUF1 <= "0000" & N ;
                --ALU_ACK <= '1';
                ALU_STATE <= LR_DRAM_WR;
            elsif ALU_ACK = '1' then
                ALU_ACK <= '0';
                ALU_STATE <= MEM_STOP;
            else
                ALU_STATE <= aSTOP;
                HS_STATE <= hStop;
            end if;

        when HsDimWrite =>
            if ALU_ACK = '0' then
```

```
                ALU_STATE <= L_MEM_WR;
                DATA_BUF1 <= "00000000000000000000" & N;
                SRAM_ADRS_BUF1 <= "10000000000000000";
                ALU_ACK <= '1';
            else
                ALU_STATE <= MEM_STOP;
            end if;

            if CALC_DONE = '1' then
                ALU_ACK <= '0';
                HS_STATE <= HsS2LoopIni;
            end if;

        when HsS2LoopIni =>
            i := k;
            j := K + '1';
            HS_STATE <= HsS2LoopBdy;

        when HsS2LoopBdy =>
if i2 < N then
ALU_STATE <= RR_MEM_DRAM;
i2 := i2 + '1';
                ROW_ADRS_BUF1 <= "0000" & ROW;
                COL_ADRS_BUF1 <= "0000" & N ;
end if;

            if i < N then
                ALU_STATE <= RR_INPRO;
                i := i + '1';
                SRAM_ADRS_BUF1 <= i(7 downto 0) & k(7 downto 0);
            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;

        when HsAkRead =>
            if ALU_ACK = '0' then
                ALU_STATE <= R_MEM_RD;
                SRAM_ADRS_BUF1 <= k(7 downto 0) + '1'  & k(7 downto 0);
                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;
        SRAM_ADRS_BUF1 <= "0000" & 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 <= LR_MEM_WR;
        DATA_BUF1 <= ( not S(31) ) & S(30 downto 0);
        SRAM_ADRS_BUF1 <= k(7 downto 0) & k(7 downto 0) + '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;
        SRAM_ADRS_BUF1 <=  k(7 downto 0) + '1' & k(7 downto 0);
    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';
            SRAM_ADRS_BUF1 <= j(7 downto 0) & i(7 downto 0);
            ADRS_BUF2 <= i(7 downto 0) & k(7 downto 0);
        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;
            SRAM_ADRS_BUF1 <= j(7 downto 0) & k(7 downto 0);
        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';
            SRAM_ADRS_BUF1 <= i(7 downto 0) & k(7 downto 0);
            ADRS_BUF2 <= i(7 downto 0) & k(7 downto 0);
        else
            ALU_STATE <= INPRO_F;
        end if;

        if CALC_DONE = '1' then
            AL <= DATA_BUS;
            i := k + '1';
            HS_STATE <= HsAlxCCalc;
        end if;

    when HsAlxCCalc =>
        ALU_STATE <= FF_MUL;
        DATA_BUF1 <= AL;
        DATA_BUF2 <= C;

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

        if CALC_DONE = '1' then
            ALxC <= DATA_BUS;
            HS_STATE <= HsAlxCd2Calc;
        end if;

    when HsAlxCd2Calc =>
        if DIV_ACK = '0' then
            DIV_A <= ALxC;
            DIV_B <= FP_TWO;
            DIV_ACK <= '1';
        else
            DIV_A <= ( others => '0' );
            DIV_B <= ( others => '0' );
        end if;

        if DIV_DONE = '1' then
```

```
                ALxCd2 <= DIV_Q;
                DIV_ACK <= '0';
                HS_STATE <= HsAlxCd2xUCalc;
            end if;
-- ok
        when HsAlxCd2xUCalc =>
            if ALU_ACK = '0' then
                ALU_STATE <= FL_MUL;
                DATA_BUF1 <= ALxCd2;
                SRAM_ADRS_BUF1 <= i(7 downto 0) & k(7 downto 0);
                ALU_ACK <= '1';
            else
                ALU_STATE <= CALC_F;
            end if;

            if CALC_DONE = '1' then
                ALxCd2xU <= DATA_BUS;
                ALU_ACK <= '0';
                HS_STATE <= HsQCalc;
            end if;

        when HsQCalc =>
            if ALU_ACK = '0' then
                ALU_STATE <= FR_ADD;
                DATA_BUF1 <= ( not ALxCd2xU(31) ) & ALxCd2xU(30 downto 0);
                SRAM_ADRS_BUF1 <= i(7 downto 0) & k(7 downto 0);
                ALU_ACK <= '1';
            else
                ALU_STATE <= CALC_R;
                SRAM_ADRS_BUF1 <= i(7 downto 0) & k(7 downto 0);
            end if;

            if CALC_DONE = '1' then
                ALU_ACK <= '0';
                if i < N then
                    i := i + '1';
                    HS_STATE <= HsAlxCd2xUCalc;
                else
                    i := k + '1';
                    HS_STATE <= HsUxQtCalc;
                end if;
            end if;

        when HsUxQtCalc =>
            if ALU_ACK = '0' then
                ALU_STATE <= LR_MUL;
                SRAM_ADRS_BUF1 <= i(7 downto 0) & k(7 downto 0);
                ADRS_BUF2 <= j(7 downto 0) & k(7 downto 0);
                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;
                SRAM_ADRS_BUF1 <= j(7 downto 0) & k(7 downto 0);
                ADRS_BUF2 <= i(7 downto 0) & k(7 downto 0);
                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);
                SRAM_ADRS_BUF1 <= j(7 downto 0) & i(7 downto 0) ;
                ALU_ACK <= '1';
            else
                ALU_STATE <= CALC_LR;
                SRAM_ADRS_BUF1 <= j(7 downto 0) & i(7 downto 0);
                ADRS_BUF2 <= j(7 downto 0) & i(7 downto 0);
            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 - "000000000010" ) then
                    k := k + '1';
                    HS_STATE <= HsS2LoopIni;
                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;
                SRAM_ADRS_BUF1 <= j(7 downto 0) & i(7 downto 0);
                ALU_ACK <= '1';
            else
                ALU_STATE <= MEM_STOP;
            end if;

            if CALC_DONE = '1' then
                RES <= DATA_BUS;
            end if;

            if OUT_ACK2 = '1' then
                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';
    DIV_DONE <= '0';
elsif rising_edge( CLK ) then
    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;
```

B.3.3 二分法

二分法、反復法、逆ハウスホルダ変換に関しては、改良までに至らなかったが、今後の参考のために山岡が設計したものをここに示す。

```
-------------------------------------------------------------
-- 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');
        Max <= (others => '0');
        i := ( 0 => '1', 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) = "00000000000000000000000000000" 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';
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then

        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;
```

B.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');
        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';
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then

        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;
```

B.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');
        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';
        DIV_DONE <= '0';
elsif rising_edge( CLK ) then

        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;
```

B.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 C.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 C

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。以下では松尾の卒業論文から、評価ボードについて必要な事項について要約して説明する。

C.1 パソコン・評価ボード間のインターフェース

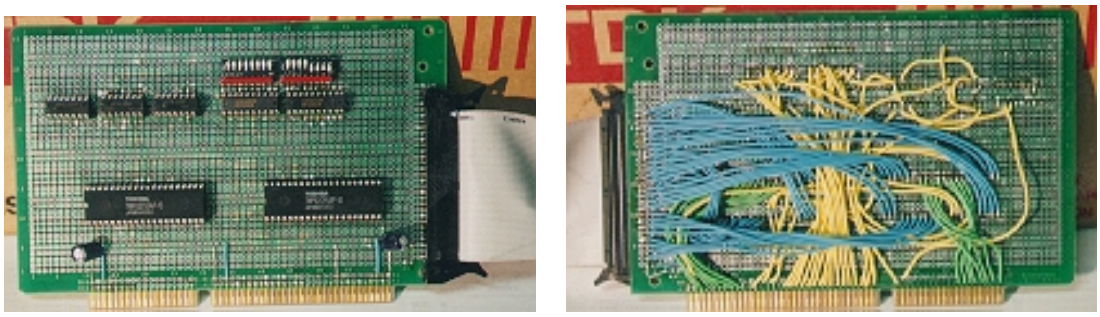
本研究では ALTERA 社により提供される FPGA である FLEX10K シリーズを使用している。これはプログラム素子が SRAM により構成されており、電源を入れた後に必ずコンフィグレーションをする必要がある。また、 FPGA 上に実現されたハードウェアに対して PC から制御・通信する必要がある。これらを実現するため、 PC と評価ボード間のインターフェースを用いている。

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 幅を実現している。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 C.1に、外観を図 C.1に示す。

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

表 C.1: インターフェースカードの仕様（松尾 1997 年度卒業論文 [3] より）



(a) 部品面 (b) ハンダ面

図 C.1: インターフェースカードの外観 [1]

外部端子

インターフェースの外部端子には 50pin ヘッダーを用いている。 pin の割り付けは図 C.2を参照。フラットケーブルを用いて他の機器と接続する。

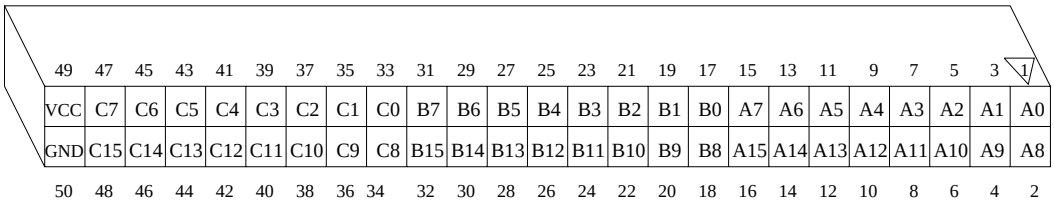


図 C.2: 外部端子 pin 割り付け[2]

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。

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

[2] ファイル名:./appendix/50head.eps

C.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 C.3に、ブロック図を図 C.4に、仕様を表 C.2に示す。

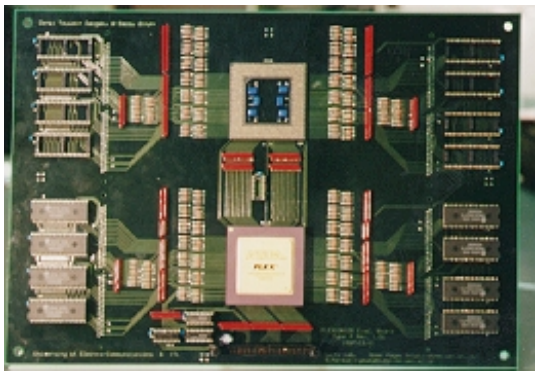


図 C.3: 計算システム評価ボード (横 38.5cm ×縦 26.5cm)[3]

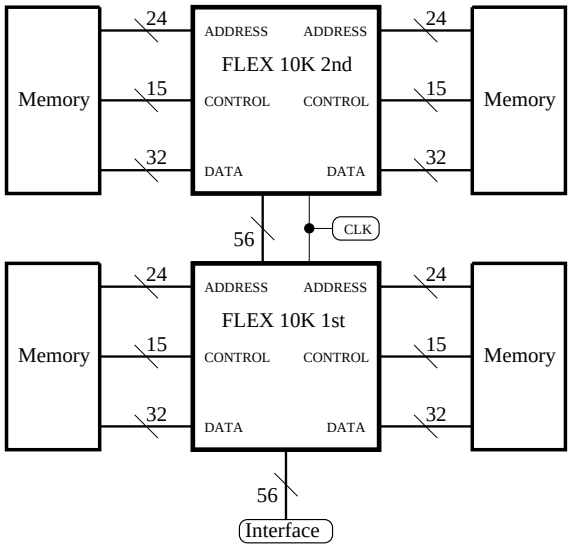


図 C.4: 評価ボードのブロック図[4]

[3]ファイル名:./appendix/eval1.eps

[4]ファイル名:./appendix/ev-block.eps

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

表 C.2: 評価ボードの仕様

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。

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

インターフェース部の下側、つまり FLEX 1st 側の PIN の割り付けを表 C.3に示す。このピン番号を VHDL ファイルの attribute に付加することで、 PC と FPGA とのデータの通信ができるようになる。

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	〃			

表 C.3: インターフェース部 下側の PIN の割り付け

インターフェース部 上側

インターフェース部の上側、つまり FLEX 2ns 側の PIN の割り付けを表 C.4に示す。これに関しては、 FLEX 2nd のコンフィグレーション時には使うが、 VHDL の設計の部分では使うことはない。

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	〃			

表 C.4: インターフェース部 上側の PIN の割り付け

C.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??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 C.5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。

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

表 C.5: インターフェースカードの I/O マッピング

C.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 C.6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。

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

表 C.6: 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

このコンフィグレーションプログラムは、1997 年度に松尾 [3] により作成された、FLEX 1st をコンフィグレーションするときのプログラムの C 言語のソースである。

最初の #define の部分で PPI8255 におけるベースアドレスを指定し、その後各ポートの上下 8bit を割りふり、FPGA のコンフィグレーションに必要なアドレスを指定している。そして、4 行の outp 関数により、ポートのモード設定が行なわれる。あとは、B ポート上位の状態により C ポート下位からコンフィグレーションが行なわれる。

```
#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 == 1491321 )
{
        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

このコンフィグレーションプログラムは、1997 年度に松尾 [3] により作成された、FLEX 1st をコンフィグレーションするときのプログラムの C 言語のソースである。

最初の #define の部分で PPI8255 におけるベースアドレスを指定し、その後、各ポートの上下 8bit を割りふり、FPGA のコンフィグレーションに必要なアドレスを指定している。そして、4 行の outp 関数により、ポートのモード設定が行なわれる。あとは、B ポート上位の状態により C ポート下位からコンフィグレーションが行なわれる。

```
#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;

}
```

C.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げられる。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。

C.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、ポート C を制御用信号としてポート A 用 ポート B 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 A ポートのみ設定可能であるが、他のモードは A ポート B ポートそれぞれ別に指定できる。したがって、A ポートを mode 2、B ポートを 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。

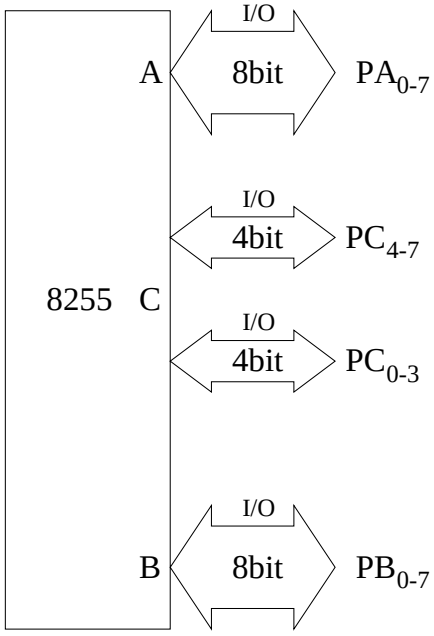


図 C.5: Mode 0[5]

[5]ファイル名:./appendix/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。

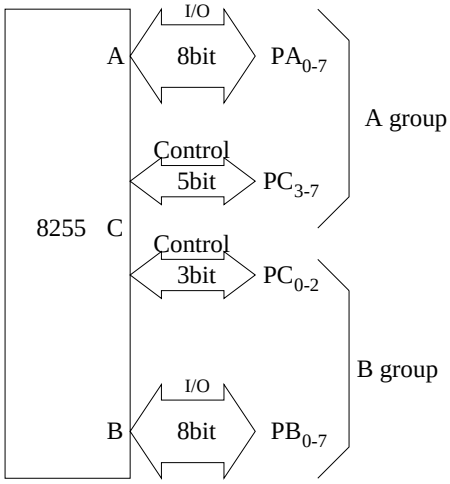


図 C.6: Mode 1[6]

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。

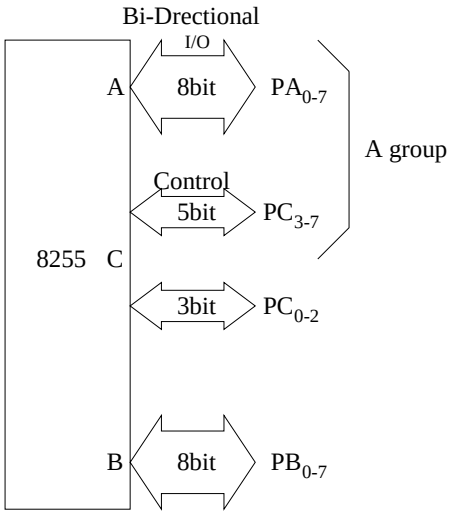


図 C.7: Mode2[7]

[6] ファイル名:./appendix/ppi8255-mode1.eps

[7] ファイル名:./appendix/ppi8255-mode2.eps

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。

D7	1 → モード設定, 0 → ビット・セット / リセット
D6 D5	ポート A の mode 指定 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

表 C.7: コントロールワードのビットの内訳

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。

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 D2 D1	D3 D2 D1 の 3bit で セット / リセット するビットを指定する
D0	0 → ビットリセット,1 → ビットリセット

表 C.8: ポート C のビット・セット / リセット

例として、 00000001 なら bit0 が 1 にセットされる。

PPI8255 の信号動作

Mode 1 および Mode 2 のハンドシェークを利用したときの PPI8255 の信号の動作を図 C.8, 図 C.9示す。 Mode 0 の場合には制御信号を使わないため、 FPGA 側からの制御信号の入力が必要ないので省略する。

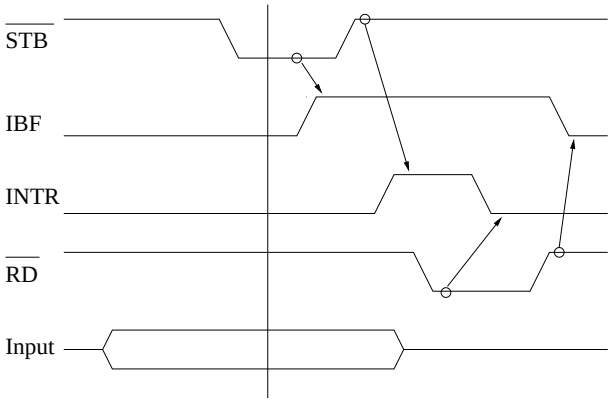


図 C.8: ハンドシェーク入力[8]

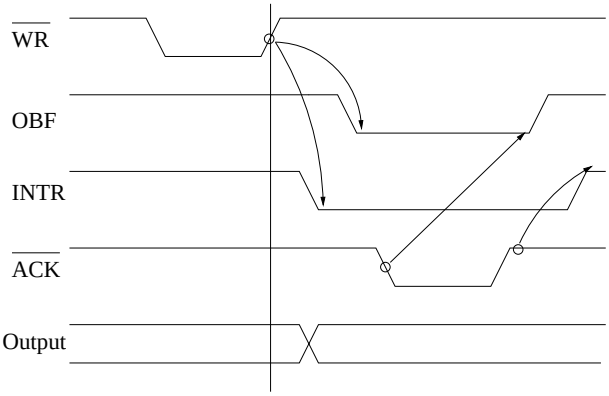


図 C.9: ハンドシェーク出力[9]

entity の構成は図 4.13に書いてあるとおり。 PC から FPGA にデータを入力したいときは、 A ポートの出力をハイインピーダンスにしておいて、 PC から $\overline{OBF}$ を立ち下げる命令を出す。しばらくして FPGA から $\overline{ACK}$ を立ち下げる命令を出す、これによって、データが入力される。

そして、 FPGA から PC に出力したいときは、 FPGA から $\overline{STB}$ を立ち下げる命令を出す。すると IBF が立ち上がりデータが出力される。データが出力されてからしばらくして、 FPGA から $\overline{STB}$ を立ち上げる命令を出し、 PPI で制御が行なわれたあと、 IBF が立ち下がり、出力が終わる。

[8]ファイル名:./appendix/time-in.eps

[9]ファイル名:./appendix/time-out.eps

付録 D

本研究での集積回路の設計方法

ここでは、本研究における集積回路の設計方法として、 Accolade 社の PeakVHDL と Altera 社の Max+PLUSII の使用方法と、 FPGA のダウンロード方法について説明する。

D.1 PeakVHDL

D.1.1 VHDL ファイル作成における注意点

この研究で PeakVHDL を使用する場合、 VHDL ファイルの作成方法について説明する。

まず最初に、

```
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;
```

を記述し、 VHDL のライブラリを呼び出す。また、 metamor のライブラリを呼び出すときには、 PeakVHDL で Synopsys library を呼び出すようにしなければならない。

そして、次の entity という部分で、 FPGA の入出力ポートとピンの設定をする。

```
entity count 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(7 downto 0);
           CL  : in std_logic   );

attribute pinnum of CLK : signal is ‘‘D22’’;
attribute pinnum of A   : signal is ‘‘BC23,BB24,BC25,BB26,BC27,BB28,BC29,BB30, ..
attribute pinnum of BL  : signal is ‘‘BC13,BB14,BC15,BB16,BC17,BB18,BC19,BB20’’;
attribute pinnum of BH  : signal is ‘‘BC5,BB6,BC7,BB8,BC9,BB10,BC11,BB12’’;
attribute pinnum of CL  : signal is ‘‘AU23’’;

end count;
```

まず、 entity 文でモジュール名を指定する。これは FPGA にダウンロードするときのファイル名にもなるので、長い名前はつけない方が良い。そして、 port 文で FPGA のポートを指定する。 FPGA にデータを入力する場合には in、データを出力する場合には out、両方の場合には inout を指定する。

そして attribute 文でピン番号を指定する。各ポートのピン番号は 4,5 章に示す。

```
architecture RTL of count is

signal CLK_CNT : std_logic_vector(7 downto 0);
signal RESET   : std_logic;

begin

end RTL;
```

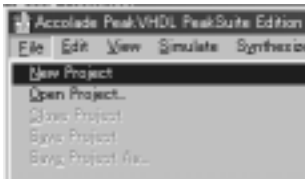
次に architecture 文で回路の設計要素を記述する。 count の部分は entity の名前と同じにする。 begin 文の前で、内部信号 signal を宣言し、 begin 文以降は回路の動作を記述する。

D.1.2 VHDL ファイルの作成

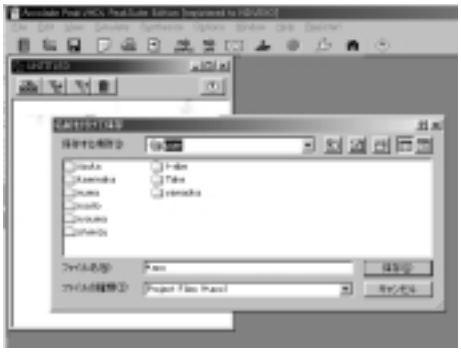
まず、 VHDL ファイルの作成方法について説明する。ここでは、 VHDL のソースをあらかじめエディタなどで作成してあるものとして話をすすめる。

- 最初に PeakVHDL を起動する。

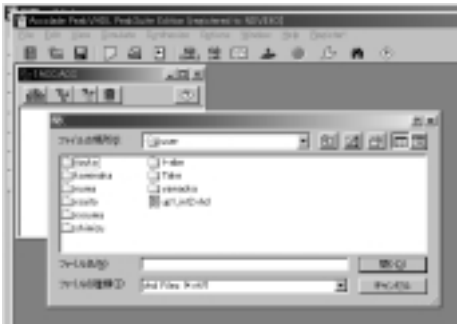
- “File” にある”New Project” を選択。



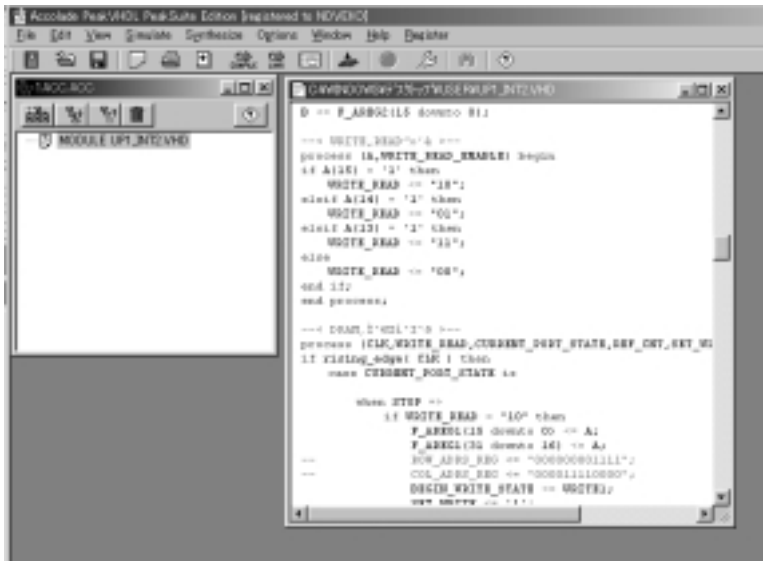
- 下のウインドウが表われたら、右クリックを押して”Add Module” を選択、ダイアログボックスが表われたら、”The Project has not been saved. Save it now?” と出てくるので、”OK” を押す、するとダイアログボックスが表われるので、 VHDL ソースを同じディレクトリにプロジェクトファイル (*.acc) を保存する。



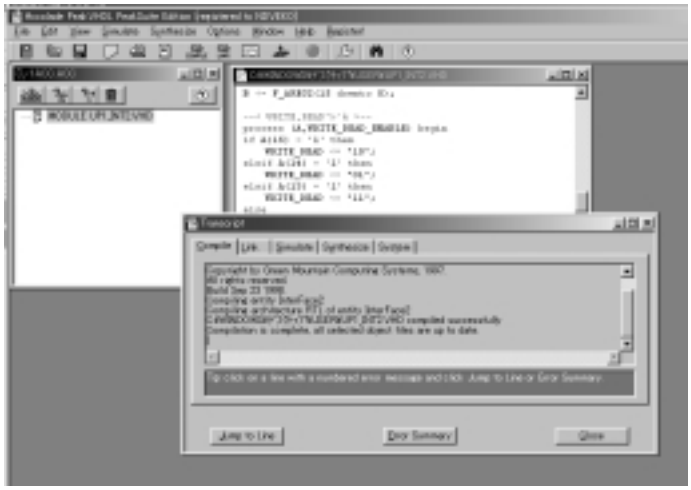
- すると、今度は” 開く” というダイアログボックスが表われるので、作成した VHDL ソースのファイル (*.vhd) を選択して開く。



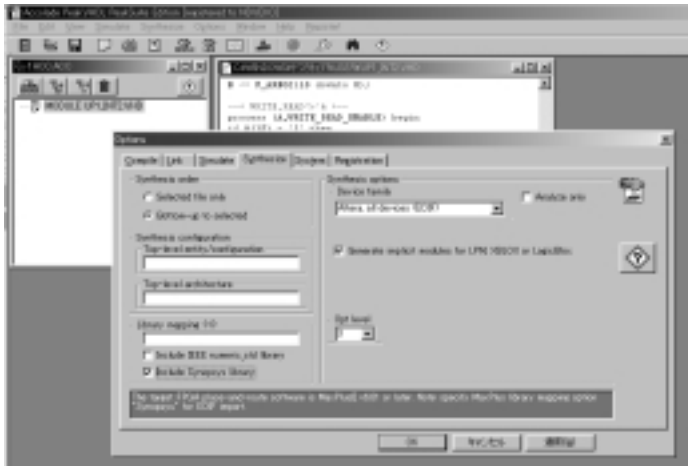
- そして、元のウインドウに戻ると、”MODULE ... .VHD” という文字が表われる。これをダブルクリックすると右側に VHDL ソースが記述してあるウインドウが表われる。さらに他のモジュール (VHDL) を追加したいときは、 2. に戻って”Add Module” を選んで、追加したいファイルを追加する。



- VHDL ファイルを構文解析するには、”COMPILE” というボタンを押す。するとコンパイルが始まり、中央にウインドウが表われる。構文に間違いがあったらここにエラーの原因とその場所が示されるので、先の VHDL が書かれているウインドウから間違いを直す。



- コンパイルが正しければ、次は論理合成を行なう。メニューの”Option” から”Synthesize” を選ぶ。すると、下のウインドウが表われる。ここで、右側にある”Device Family” から”Altera all Devices (EDIF)” を選択する。そして、左下にある”Include Synopsys Library” をチェックする。



- そして、メニューの”Synthesize” を選ぶ。すると、別のウインドウで論理合成が行なわれる。論理合成が正しく行なわれたならば、モジュール名のついた EDF ファイルが作成される。

- このプロジェクトファイルの中で、 VHDL ファイルを 2 つ以上合成して、ひとつのモジュールを作ることができる。このとき、それぞれのファイルは component または function でファイルがリンクできていなければならない。そのリンクができているかを確かめるには、”Rebuild Hiteracy” というボタンを押して、リンクし直し、”Show Hiteracy” というボタンを押して component や function ができているかを確かめる。

D.1.3 PeakVHDL でのシミュレーション方法

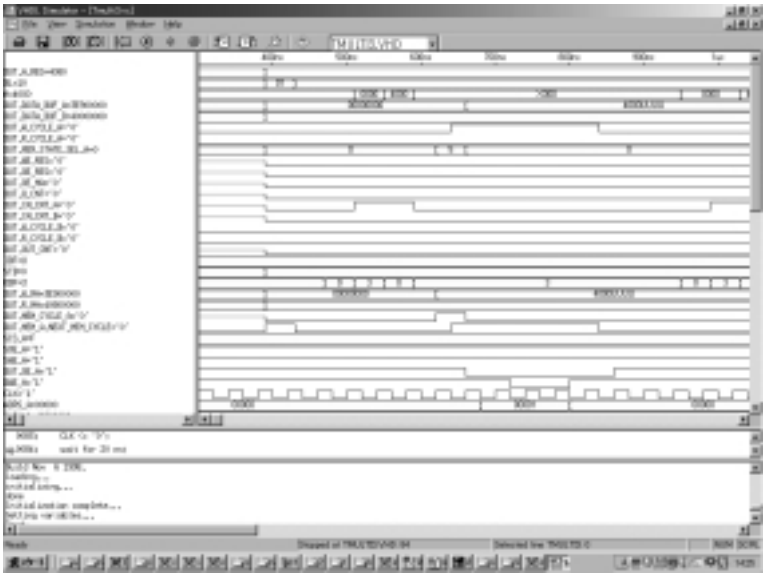
シミュレーションはプロジェクトウインドウにテストベンチ用の VHDL ファイルを挿入することでできる。その VHDL ソースには、テストベンチの対象となるソースと同じ port を宣言し、それと同じ signal も宣言する。そして、その port に使われている信号を並べた DUT 文を begin 文の後に挿入する。 proecss には入力ポートにたいする信号を記述する。例として、シミュレーションの VHDL ソースを以下に示す。

```
------------------------------------------------------------
-- Test Bench for mem13.vhd
-- < tmem15.vhd >
-- 1998/06/16 (Tue)
-- numa@tube.ee.uec.ac.jp
------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity tmem15 is
end tmem15;

architecture TESTBENCH of tmem15 is

component mem15 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(7 downto 0);
         ADRS : out std_logic_vector(16 downto 0);
         DATA : inout std_logic_vector(31 downto 0);
         SCS  : out std_logic_vector(3 downto 0);
         SOE  : out std_logic;
         SWE  : 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);
         CL : in std_logic_vector(5 downto 0)
            );

end component;

signal CLK : std_logic;
signal A : std_logic_vector(15 downto 0);
signal BL : std_logic_vector(7 downto 0);
signal BH : std_logic_vector(7 downto 0);
signal ADRS : std_logic_vector(16 downto 0);
signal DATA : std_logic_vector(31 downto 0);
signal SCS : std_logic_vector(3 downto 0);
signal SOE : std_logic;
signal SWE : std_logic;
signal OBF : std_logic_vector(1 downto 0) := "11";
signal ACK : std_logic_vector(1 downto 0) ;--:= '1';
signal STB : std_logic_vector(1 downto 0) ;--:= '1';
signal IBF : std_logic_vector(1 downto 0) := "00";
signal CL : std_logic_vector(5 downto 0);

begin

DUT : mem15 port map ( CLK, A, BL, BH, ADRS, DATA, SCS, SOE,
SWE, OBF, ACK, STB, IBF, CL);

process begin
```

```
CLK <= '0';
wait for 100 ns;
CLK <= '1';
wait for 100 ns;
end process;

process begin
wait for 250 ns;

wait for 25 ns;
BL <= "00000001";
wait for 200 ns;
BL <= "00000000";
wait for 200 ns;

OBF <= "00";
A <= "0000000000000001";
wait for 800 ns;
OBF <= "11";
wait for 200 ns;

OBF <= "00";
A <= "1010101010101010";
wait for 800 ns;
OBF <= "11";
wait for 200 ns;

-- READ_CYCLE
OBF <= "00";
A <= "0000000000010000";
wait for 800 ns;
OBF <= "11";
wait for 1000 ns;
DATA <= "ZZZZZZZZZZZZZZZZ1111111111111110";
A <= "ZZZZZZZZZZZZZZZZ";

IBF <= "11";

wait;
end process;
end TESTBENCH;
```

シミュレーションの仕方について説明する。まず、先に述べたプロジェクトウインドウにテストベンチの VHDL ソースを追加する。そして、 Rebulid したあとに、テストベンチのファイルを選択して、”Simulation” から”Load Selected” を選ぶ。すると、構文解析と同じウインドウが表われ、コンパイルが始まる。コンパイルが成功すると、下のようなウインドウが表われる。

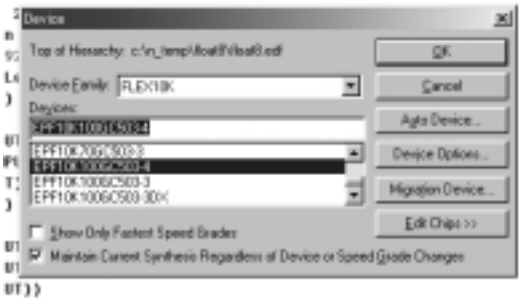
このウインドウが表われたら、シミュレーションしたい内部信号を選択し、”close” を押す。そして、スパナのアイコンを押して、シミュレーションする動作時間を決める。それを決めたら、ボタンのようなアイコンを押して、シミュレーションを開始する。シミュレーションに成功すると、下のような画面となる。



D.2 Max+PLUSII

論理合成が終わったら、次はFPGAに搭載できるファイルに置き換えなければならない。ここではその方法について説明する。まず、Max+PLUSIIを起動する。そして、”Open”,”File”で論理合成したファイルを選択する。

次に”Assign”,”Device”を選択し、ダウンロードするFPGAを選択する。本研究では、EPF10KGC503-4を使用しているので、それを選択する。そして、”Options”を選び、チェックをすべて外す。



次に、メニューの”MAX+plusIIから”Compiler”を選択する。すると次のような画面が出てくるので、”start”ボタンを押す。

コンパイルが終わると、TTFファイルが作成されるので、あとはこれをFPGAにダウンロードすればよい。

付録 E

学外における発表実績

- 学会発表
  - 日本物理学会 秋の分科会 平成 11 年 9 月 (岩手大学)
    書き換え可能なゲート素子を持つデバイスを用いた
    行列計算専用集積回路の設計
    沼　知典, 松尾 竜馬, 山岡 寛明, グエン・ドゥック・ミン,
    齋藤 理一郎, 木村 忠正