1997 年度 卒業論文

行列計算専用大規模集積回路の開発

9410171

木村・齋藤研　松尾 竜馬

電気通信大学 電子工学科 電子デバイス工学講座

指導教官 齋藤 理一郎 助教授

提出日 平成 10 年 2 月 4 日

概要

物性の分子軌道法による解析は多大な計算量を必要する。計算時間の大部分は、永年方程式を解く為の行列の固有値・固有ベクトルを求める計算である。

計算時間を短縮するためにどの様な方法を取るべきか? そのためにまず考えられる方法は、計算の並列化である。しかし、汎用の並列コンピュータを用いた並列化では実際の計算時間の短縮があまりされない。なぜなら並列化率が低いからである。

計算の高速化の問題で、汎用コンピュータでは工夫が困難な計算を、専用計算機・専用プロセッサを用いる手法が一般的になってきた。また、半導体デバイスやそれに付随する技術の発展が目覚しく、 HDL(Hardware Description Language: ハードウェア記述言語) を用いての LSI(Large Scale Integrated circuit: 大規模集積回路) の設計が一般的になって来た。そのおかげで、半導体デバイスに関する高度な専門技術をもったエンジニアでなくとも、 LSI の機能を設計することが可能になったことは重要である。

まず、現在の並列計算機の問題点を指摘して、専用プロセッサが解決すべき点を示す。次に行列演算に特化してこの問題を解決した計算システムを提案する。次に、そのようなシステム上で実現できるハウスホルダ変換とその逆変換の計算手順を示した。

以上の事を実現するプロセッサを設計したいのだが、 HDL の記述とシミュレーションの環境だけでは、プロセッサの設計をする事は難しい。論理が正しくても、論理合成が可能でないかも知れない。また、プロセッサ設計や HDL についての経験が不足している。そこで、 FPGA(Field Programmable Gate Array) を用いて、HDL で記述したプロセッサを試してみることを考えた。

FPGA の書き込み装置として DOS/V パソコンから利用できる、 ISA BUS 用 パラレル I/O カードを設計・製作した。そして FPGA 2 つ と メモリ 4 系統 を搭載した、評価用プリント基板を設計・製作した。

目次

1 序論

1.1 背景

物性に関する研究には分子軌道法による数値解析が用いられる。この計算は永年方程式を解く部分が計算時間の大部分を占めている。永年方程式を解くことは行列の固有値・固有ベクトルを求めることに帰着されるが、この計算は行列の次元を N とすると、計算量は N の 3 乗に比例すると言われている。つまり、N が 10 倍になると計算量は 1000 倍になる。おそらく計算時間もそれだけ時間がかかるだろう。このことが、複雑な分子の分子軌道法による解析を困難なものにしている。それゆえ、できる限り計算時間を短縮することは重要な課題である。

計算時間を短縮する方法として、演算プロセッサの数を複数にして、同時に多くの演算をできるようにすることが考えられる。しかし多くの場合プロセッサを増やしてもプロセッサ間の通信量か増大し、計算時間の短縮は頭打ちになることが多い。ある時点でプロセッサに必要なデータを他のプロセッサも必要としている場合が多く、メモリアクセスの競合が生じ、それぞれのプロセッサに時間をずらして送ることになる。このため、アルゴリズム上並列化が可能でも、実際のシステムでは十分に分散処理が出来ないでいる。並列処理が可能なスーパーコンピュータでも、このメモリから各プロセッサにデータを送る時のオーバーヘッドが大きな問題になっている。この問題はプロセッサ数が多くなる程、顕著に現れてくる。この問題は並列コンピュータのメモリとプロセッサ間の通信方法に負うところが大きい。

このような問題を解決する為に、汎用コンピュータでは工夫が困難な計算を、専用計算機・専用プロセッサを用いて高速化する手法が一般的になってきた。

近年、半導体デバイスやそれに付随する技術の発展が目覚しく、HDL(Hardware Description Language: ハードウェア記述言語) を用いての LSI(Large Scale Integrated circuit: 大規模集積回路) の設計が一般的になって来た。そのおかげで、半導体デバイスに関する高度な専門技術をもったエンジニアでなくとも、LSI の機能を設計することが可能になったことは重要である。

1.2 本研究の目的

分子軌道法による解析の時間を短縮するために、行列の固有値・固有ベクトルを求めるのに適した計算システムを考案する。そのシステム用の専用のプロセッサ LSI を HDL を用いて設計する。

1.3 本論文の構成

まず、行列の固有値・固有ベクトルを計算するアルゴリズムとして、ハウスホルダー法を説明する。

次に、ハウスホルダー法の計算をを並列化したとき、どのような計算システムなら、効率良く計算できるかを示す。

また、考案したシステムのコンセプトを検証する為、ハードウェア記述言語で表された演算プロセッサの動作環境として FPGA を用いた評価用プリント基板の設計・製作した。それにそれに付いて説明する。

2　行列の固有値・固有ベクトル問題とハウホルダ法

物性に関する研究には、分子軌道法による解析が行なわれる。この理論計算は永年方程式を解く必要があり、行列の固有値・固有ベクトル問題に置き換えられる。まず、固有値・固有ベクトルについて簡単に説明し、その解法としてハウスホルダ法を説明する。

2.1　行列の固有値問題について

2.1.1　固有値・固有ベクトル

n 次正方行列　に対して、

$$= \lambda \qquad (\neq \emptyset) \tag{1}$$

を満たすベクトル　と、スカラー λ が存在する時、λ を　の固有値、　を固有値 λ に対する　の固有ベクトルという。

また、x の多項式 $f_A(x) = | - x |$ を　の固有多項式、方程式 $f_A(x) = 0$ を　の固有方程式という。

2.1.2　相似変換

　を正則行列とする。

’$= ^{-1}$　　を相似変換といい、’の固有値は　と一致する。

　と’の固有ベクトルは一般には異なるが、それぞれの固有ベクトルを　、とすると、

$$= ' \tag{2}$$

の関係がある。

2.1.3　3 重対角行列

3 重対角行列とは、対角成分と、それに隣接する副対角成分以外は全て 0 であるような行列である。

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

適当な正則行列 での相似変換により、実対称行列を 3 重対角化できる。相似変換なので、固有値はもとの実対称行列と一致する。

2.2　ハウスホルダ法

行列の固有値・固有ベクトルを求めるアルゴリズムの一つにハウスホルダ法がある。このアルゴリズムは多少複雑であるが、行列の次元が大きい場合他の方法よりも計算量が少なくて済む利点がある。簡単に説明すると、ハウスホルダ法の手順は以下の 4 つからなる。

2.2.1　ハウスホルダ変換による 3 重対角化

まず、固有値と固有ベクトルを求めるべき対称行列　をハウスホルダ変換を用いて 3 重対角行列 ’ に変換する。 3 重対角化をする事により、後の固有値や、固有ベクトルを求める計算量を少なくする事が出来る。

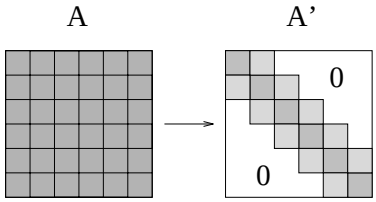


図 2.1　3 重対角化 [1]

2.2.2　二分法

固有値は固有方程式を解く事により求められる。固有方程式は行列の次元を N として N 次方程式になるが、行列が 3 重対角化されているため、実際に方程式を立てなくても 2 分法を用いて数値解を求めることできる。

2.2.3　逆反復法

固有ベクトルは行列と固有値から N 次連立方程式がたてられ、それを解く事により求められる。ここで、直接元の実対称行列の固有ベクトルを求めるより、 3 重対角行列の固有ベクトルを求めてから元の行列の固有ベクトルに変換するほうが計算量や計算のアルゴリズムの面で有利である。計算方法として、逆反復法を用いて 3 重対角行列の固有ベクトルを求める。

[1]図 2.1 = u97ryou/hause.eps

2.2.4　逆変換

こうして求めた 3 重対角行列の固有ベクトルをに対して、ハウスホルダ変換のときと逆の変換を施して、元の行列の固有ベクトルを得る。

2.3　ハウスホルダ法の計算手順

2.3.1　ハウスホルダ変換の計算手順

実対称行列 を 3 重対角化をする変換行列 を求めるのだが、

$$' = \quad ^{-1}$$

まず、

$$= \ -c \quad ^{T}, \qquad c = \frac{2}{^{T}}, \qquad \text{は適当な}N\text{次の列ベクトル},$$

で表される、行列 を考える。この行列をハウスホルダ行列という。

この行列は、 の第 ik 成分を a_i として

$$= \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1} + s \\ a_{k+2} \\ \vdots \\ a_N \end{pmatrix}, \qquad s = \sqrt{\sum_{j=k+1}^{N} (a_j)^2}, \tag{4}$$

と決めると、 $^{()}$ による相似変換は、

$$= \ ^{(k)-1} \qquad ^{(k)}$$

実対称行列 を k 行と k 列について 3 重対角化する。

これを用いれば、 $= \ ^{(1)(2)} \ldots ^{(N-3)(N-2)}$ と表せる。つまり、一段ずつ 3 重対角化していくことができる。

また、ハウスホルダ変換は、直接行列 を作り行列同士の積を計算しなくてもよい。次のように ベクトル を用いて、ベクトルの計算に置き換えることが出来る。

$$= c, \qquad = \ -\frac{c}{2} \ ^{T}, \tag{5}$$

$$= \ -(\ ^{T} + \ ^{T}). \tag{6}$$

ハウスホルダ変換による実対称行列の 3 重対角化の計算手順をまとめると、次のようになる。ただし、$^{(k)}$ の第 ij 成分を $a_{ij}^{(k)}$ と書く。

$^{(1)} =$ とおき、$k = 1, 2, 3, \cdots, N-2$ について次の手順をくり返す。

$$s_k = \sqrt{\sum_{j=k+1}^{n} \left(a_{jk}^{(k)}\right)^2}, \tag{7}$$

もしも $a_{k+1,k}^{(k)} < 0$ ならば s_k の符号を負にする。

もしも $s_k = 0$ ならば (8) から (10) を実行せずに次の k へ進む。

と c_k を計算する。

$$c_k = \frac{1}{s_k^2 + a_{k+1,k}^{(k)} \times s_k}, \qquad ^{(k)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1,k}^{(k)} + s_k \\ a_{k+2,k}^{(k)} \\ \vdots \\ a_{N,k}^{(k)} \end{pmatrix}, \tag{8}$$

$^{(k)}$ を作り、$^{-}$　　を計算するのだが、これは以下の計算と同値である。

行列同士の積の計算が無くすことができ、ベクトルの計算だけになる事は重要である。ここがハウスホルダ変換を用いる時の利点になる。

$$^{(k)} = c_k \, ^{(k)} \, ^{(k)}, \qquad ^{(k)} = ^{(k)} - \frac{c_k}{2} \, ^{(k)} \, ^{(k)T} \, ^{(k)}, \tag{9}$$

$$^{(k+1)} = ^{(k)} - \left(^{(k)} \, ^{(k)T} + ^{(k)} \, ^{(k)T} \right), \tag{10}$$

最終的に、$' = ^{(N-2)}$ となる。

2.3.2　2 分法の計算手順

ハウスホルダ変換で 3 重対角化された行列を ' として、行列 $\lambda -$ ' を考えその、第 k 主小行列式を $p_k(\lambda)$ とおく。

$$p_k(\lambda) = \begin{vmatrix} \lambda-\alpha_1 & -\beta_1 & & & \\ -\beta_1 & \lambda-\alpha_2 & \ddots & & \\ & \ddots & \ddots & \ddots & 0 \\ & & \ddots & \lambda-\alpha_{k-1} & -\beta_{k-1} \\ 0 & & & -\beta_{k-1} & \lambda-\alpha_k \end{vmatrix} \tag{11}$$

これを最後の行について展開すると、

$$p_k(\lambda) = (\lambda-\alpha_k)p_{k-1}(\lambda) - {\beta_{k-1}}^2 p_{k-2}(\lambda), \tag{12}$$

λ の多項式列 $\{p_k(\lambda)\}$ に関する漸化式が得られる。

$2 \leq k \leq N$ なので、便宜的に $p_0(\lambda)$ を 1 とおけば、漸化式は成立する。

$p_N(\lambda)$ の根が行列 ' の固有値であるが、スツリムの定理によると、次の多項式の列

$$p_N(\lambda), p_{N-1}(\lambda), \cdots, p_1(\lambda), p_0(\lambda), \tag{13}$$

の左から見た時の符号の変化の回数が、 λ よりも大きい固有値の個数に等しい。

このことを用いれば、 $N(c)$ を $p_k(c)$ の符号変化回数と定義する。大きい方からの k 番目の固有値 λ_k が区間 (a,b) $(a<b)$ に存在するとして、 $c = \frac{a+b}{2}$ とおく。

すると、 $N(c) \geq k$ ならば 区間 (c,b) に、 $N(c) < k$ ならば区間 (a,c) に存在する事がわかる。範囲を $1/2$ に狭める事を繰り返す事により、値を数値的に求めることが出来る。この方法が 2 分法である。

また、全ての固有値の範囲は、ゲルシュゴーリンの定理により、 $(-r,r)$ と定められる。

$$r = \max_{1\leq i\leq N}\{|\beta_{i-1}| + |\alpha_i| + |\beta_i|\}, \qquad \beta_0 = \beta_N = 0. \tag{14}$$

2 分法の計算をまとめる。

$N(\lambda)$ の計算は、

$$g_k(\lambda) = \frac{p_k(\lambda)}{p_{k-1}(\lambda)}, \tag{15}$$

について、負または 0 の個数を数えればよく、 $g_k(\lambda)$ は以下の漸化式で計算できる。

$$g_1(\lambda) = \lambda - \alpha_1$$

$$g_k(\lambda) = \lambda - \alpha_k - \frac{{\beta_{k-1}}^2}{g_{k-1}(\lambda)}, \qquad k = 2, 3, \cdots, N, \tag{16}$$

ただし、 $g_{k-1}(\lambda) \simeq 0$ のときは、 $g_k(\lambda) > 0$ 、次の計算では $g_{k+1}(\lambda) = \lambda - \alpha_{k+1}$ とする。

出発の区間 (a_0, b_0) を先のゲルシュゴーリンの定理を用いて決めておく。

- $c_j = \frac{a_{j-1}+b_{j-1}}{2}$ とおき、
- $N(c_j) \geq k$ ならば $a_j = c_j, b_j = b_{j-1}$ とする。

  $N(c_j) < k$ ならば $a_j = a_{j-1}, b_j = c_j$ とする。
- 上の計算を繰り返し、 $|a_j - b_j|$ が十分小さくなったならば、 λ_k の近似値として c_j とし、終了。

2.3.3　逆反復法の計算手順

n 次正方行列　の絶対値最大の固有値 λ_m に縮退がない場合、これに対応する固有ベクトルを $_m$ とすると、

$$_m = \lim_{n\to\infty} {}^n \qquad (\ :\text{零でない適当なベクトル})$$

となる。これを絶対値最大でない固有値・固有ベクトルに対応させるには、固有値・固有ベクトルを $\lambda_i, _i$ として、

$$(\mu\ -\ )^{-1}{}_i = \frac{1}{\mu-\lambda}{}_i \tag{17}$$

が成り立つ事を利用する。$\frac{1}{\mu-\lambda_1}$ が十分大きければ $(\mu \simeq \lambda_i)$、$_i$ が対応する固有値が絶対値最大の固有ベクトルになる。

$$_{k+1} = (\mu\ -\ )^{-1}{}_k \tag{18}$$

を繰り返せば は固有ベクトルに近づく。

実際の計算では、n 次 連立方程式 $(\mu\ -\ )_{k+1} = {}_k$ を解くことにして、逆行列 $(\mu\ -\ )^{-1}$ の計算をしないようにする。この連立方程式はをガウスの消去法を用いて解ける。$(\mu\ -\ )$ を LU 分解して説明する。

$$(\mu -) = \qquad = \begin{pmatrix} 1 & & & & 0 \\ m_1 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & 1 & \\ 0 & & & m_{N-1} & 1 \end{pmatrix} \begin{pmatrix} a_1 & -\beta_1 & & & 0 \\ & a_2 & \ddots & & \\ & & \ddots & \ddots & \\ & & & a_{N-1} & -\beta_{N-1} \\ 0 & & & & a_N \end{pmatrix} \tag{19}$$

$$a_1 = \mu - \alpha_1, \qquad m_i = \frac{-\beta_i}{a_i}, \qquad a_{i+1} = \mu - \alpha_{i+1} - m_i\beta_i$$

と下三角行列と上三角行列の積に分解できる。

これより、　　$= {}_i$,　　$_{i+1} =$ について解けばよい事がわかる。要素の少ない 3 角行列なので比較的簡単に解ける。

この計算をまとめると、ベクトル $_{i,i+1,}$ の i 番目の要素を それぞれ、x_i, X_i, y_i として、

$$y_i = x_i - x_{i-1} m_{i-1} \qquad (i = 2 \to n)$$

$X_N = \dfrac{y_N}{a_N}$ とおき、

$$X_i = \frac{y_i - \beta_i X_{i+1}}{a_i} \qquad (i = (N-1) \to 1)$$

という漸化式で表す事が出来る。この計算を数回繰り返せば、固有値 μ に対応する、行列 ' の 固有ベクトルをもとめることが出来る。

2.3.4　逆変換の計算手順

’の固有ベクトルの一つを ′ とし、それに対応する の固有ベクトルを　とする。

$$= \prime \quad = {}^{(1)\,(2)} \cdots {}^{(N-3)\,(N-2)} \prime \ ,$$

という関係がある。ここで、

$$\prime \quad = (\ - c \quad {}^{T}) \prime \ = \prime - c_k (\prime \cdot \ )$$

である事を考えると、

逆変換の計算手順は以下のようになる。

${}^{\prime(N-2)} = \prime$とおき、$k = N-2, N-1, \cdots, 2, 1$ について次の計算を繰り返す。

$$ {}^{\prime(k-1)} = {}^{\prime(k)} - c_k \big({}^{\prime(k)} \cdot {}^{(k)} \big) {}^{(k)} \tag{20}$$

最終的に、 $= {}^{\prime(1)}$ である。

3　ハウスホルダ法を並列処理する方法

ハウスホルダ法を並列化するには、まず互いに独立な計算式に分解をしなければならないが、このこと自体は比較的容易である。しかしながら、独立な計算式に分解できても、実際に並列処理出来るとは限らない。まずハウスホルダ法を分解してみて、この分解法は通常の並列計算機には向かない事を示し、どの様な点が問題なのかを述べる。

3.1　ハウスホルダ法の分解

まずハウスホルダ法は独立な計算式に分解出来る事を述べる。

3.1.1　ハウスホルダ変換

ハウスホルダ変換の計算は基本的にベクトルの内積、行列・ベクトルの内積、行列の引き算などの、ベクトル・行列演算で表される。これらの行列演算は、互いに独立なベクトルの演算に分割することができる。

3.1.2　二分法

行列の固有値は、3 重対角行列の主対角成分と副対角成分の約 2N 個のデータから求めることができる。そして、それぞれの固有値は全く独立に求めることができる。このことを利用して、それぞれの固有値を同時に計算できる。

3.1.3　逆反復法

二分法と同様に、3 重対角行列の主対角成分と副対角成分の約 2N 個のデータと固有値から $\tilde{A}$ の固有ベクトルを求めることができる。それぞれの固有値に対応する固有ベクトルは、それぞれ独立に求める事ができる。つまり、それぞれの固有ベクトルを同時に計算できる。

3.1.4　逆変換

それぞれの 3 重対角行列の固有ベクトルに対して、ハウスホルダー変換の逆変換を行なう。ハウスホルダー行列と固有ベクトルの積を繰り返すだけなので、ハウス

ホルダ変換と同様に独立な計算に分解できる。

3.2　並列コンピュータでの問題点

3.2.1　並列化率

並列計算の効率を示すものとして、並列化率がある。計算の中で、並列化可能な部分とそうでない部分があるが、並列化可能な部分の比を並列化率という。

プロセッサの数を n とし、並列化率を X とし、計算時間の短縮率を P とすると、

$$P = \frac{1}{(1-X) + \frac{X}{n}} \tag{21}$$

当然 X が大きければ並列化の効果は高くなるが、アルゴリズムの上では X が大きくなるはずでも、実際の計算機システムでは、 X はそれほど大きくならない。これは、計算機システムにおいてプロセッサやメモリ間のデータのやり取り (通信) に、有限の時間が必要であることが理由である。それゆえ通信の効率を高めることが重要な課題になる。

3.2.2　メモリとプロセッサの通信

並列処理が可能なコンピュータでは、プロセッサ間やメモリの通信をルータを介して行われている。このため、メモリやその他のリソースを共有することができる。

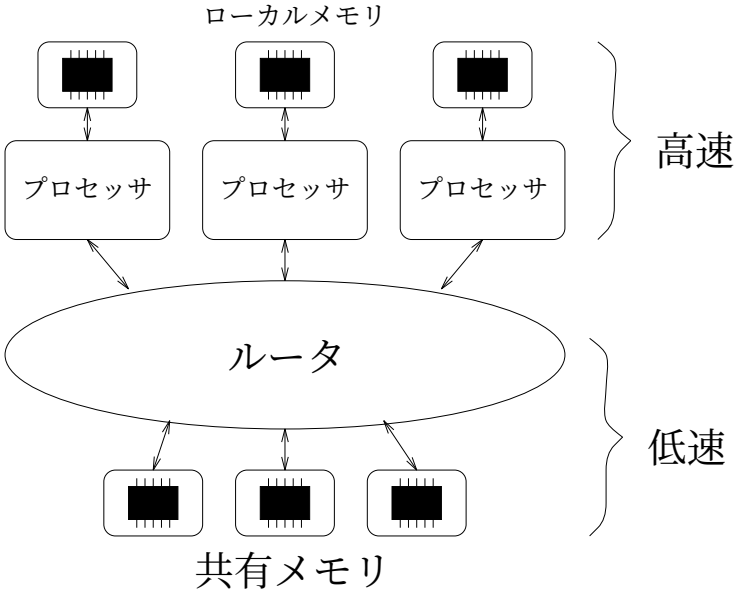


図 3.2 並列コンピュータの通信 [2]

[2]図 3.2 = u97ryou/promemj.eps

しかし、並列コンピュータでのプログラミングはできるかぎり共有メモリやプロセッサ間の通信が少なくなるようなコーディングをする。つまり、出来るだけ高速なローカルメモリにデータをおいたまま計算を続けられるようにする。何故ならルータを介した通信は、プロセッサの演算処理に比べて低速であるからである。それぞれのプロセッサは非常に高速であるが、データのやり取りが多くなると、ルータの部分がボトルネックになる。

3.2.3　行列演算の並列化の場合

行列演算の場合を考えてみよう。行列と列ベクトルの内積を例にする。

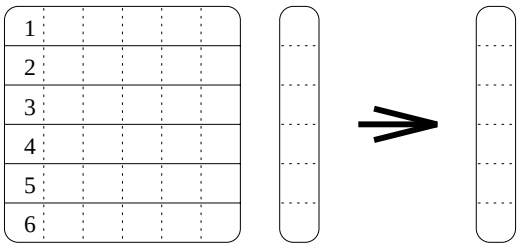


図 3.3 行列と列ベクトルの内積 [3]

行列と列ベクトルの内積の計算は、行列を行ベクトルに分けて、行ベクトルと列ベクトルの内積に分けられる。

一つの内積の計算を 1 つのプロセッサが受け持つとする。それぞれの内積は別々に計算しても構わないので、行列と列ベクトルの内積はそれぞれ同時に計算出来る可能性がある。

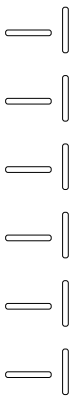


図 3.4 内積の分解 [4]

[3]図 3.3 = u97ryou/matdiv.eps
[4]図 3.4 = u97ryou/matdiv2.eps

ルータを通じてそれぞれのプロセッサにデータを配ることになるが、プロセッサはデータを受け取るまでの間処理を待たされる。行列を分けた行ベクトルがそれぞれのプロセッサのメモリに記憶されているとしても、列ベクトルを送るにはプロセッサの数だけ繰り返さなくてはならない。この場合プロセッサ数を増して計算が速くなるのは、データの伝送時間が演算時間よりもずっと短い場合である。

今考えている行列の演算の並列化の場合は、一つのプロセッサに対しての通信量と演算量が同程度であるため、データの伝送時間は無視できない。このため、並列化しても並列化率は低く、計算時間の短縮が望めないことになる。

3.2.4　データ伝送の工夫

受け取り終ったプロセッサから処理をしていき、最初のプロセッサが処理を終る時に、データを配り終っていれば、あまり時間の無駄にはならないので問題はないと考えるかも知れない。しかし、それは、演算量が通信量よりも大きい場合で、そうでない場合つまり行列演算の場合などでは、先に終了したプロセッサへのデータの転送は、他のプロセッサへの転送が終るまで待たされることになる。

この待ち時間を減らすには、単純に共通のデータを全てのプロセッサへ同時に転送できるようにすれば良い。しかし、通常の並列コンピュータは、そのような密な結合は現実的ではないが、専用の計算システムを考えるならば十分可能である。

3.2.5　通信と演算の同時処理

通常、並列コンピュータのプロセッサは、処理すべきデータをまとめて受け取ってから処理をおこなう。なぜなら、プロセッサ側の高速なメモリから演算器に入力することができるからである。この場合、演算量がデータ量が多い場合に処理速度が速くなる。しかし行列演算の場合はデータ量と演算量が同程度なので、データ入力の時間と計算時間は余り違わない。この場合は、ルータからのデータの入力と同期して演算処理を行うようにする方が効率が良くなる可能性がある。

3.3　プロセッサとメモリの結合を考える

行列演算に関して、プロセッサ間の通信を特化したシステム考える。

まず、通信量を減らす事を考える。また、プロセッサへのスムーズなデータ入力の為には、データの記憶の仕方や、メモリとプロセッサの接続の仕方が重要である。

3.3.1　データの記憶

演算プロセッサに滞りなくデータを送り、各プロセッサが待たされることの無いようにしなければならない。では、どのように記憶すればよいのか?

ハウスホルダ変換や逆変換では、行列とベクトルの積や 2 つのベクトルから作られる行列と行列の加減算が主な処理である。これは、行もしくは列ベクトルの演算に分けることが出来る。このことからベクトルの形で分割してメモリに連続して記憶するのが有利である。

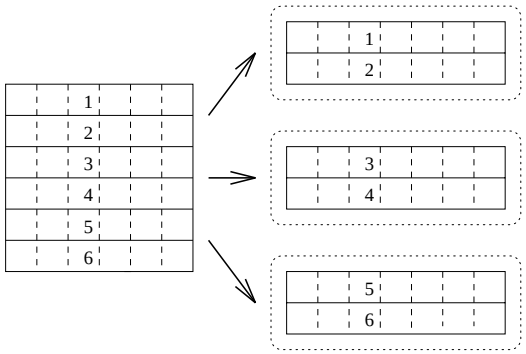


図 3.1 データの分割記憶 [5]

[5]図 3.1 = u97ryou/memdiv.eps

3.3.2　メモリとプロセッサの接続

プロセッサが演算処理をするということは、概念上、2つの入力、1つの出力で表すことができる場合が多い。

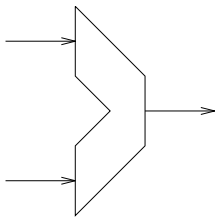


図 3.2 演算器の例 [6]

このことから、滞り無くプロセッサにデータを送るには、送られるべきデータは、それぞれ別のメモリに有ったほうがよさそうである。なぜなら同じメモリに 有るならば、2つめのデータが得られるまでプロセッサが処理を待たされるからである。

ところで、行列演算の場合は 2つのデータのうち、1つは 他のプロセッサと共通になることが多い。それならば、プロセッサ群の片方の入力を共通にすることを考えてもよいだろう。

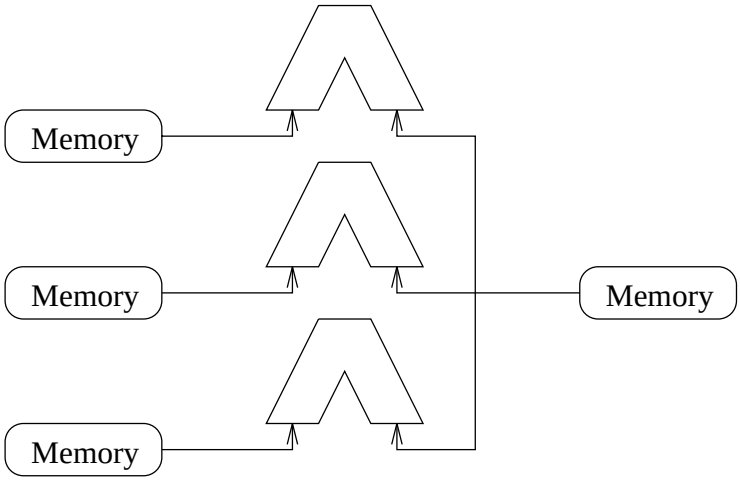


図 3.3 演算器とメモリの配置 [7]

[6]図 3.2 = u97ryou/operator.eps
[7]図 3.3 = u97ryou/opemem.eps

3.3.3　プロセッサの配置について

以上のことを考慮すると、次のような構成が良さそうである。

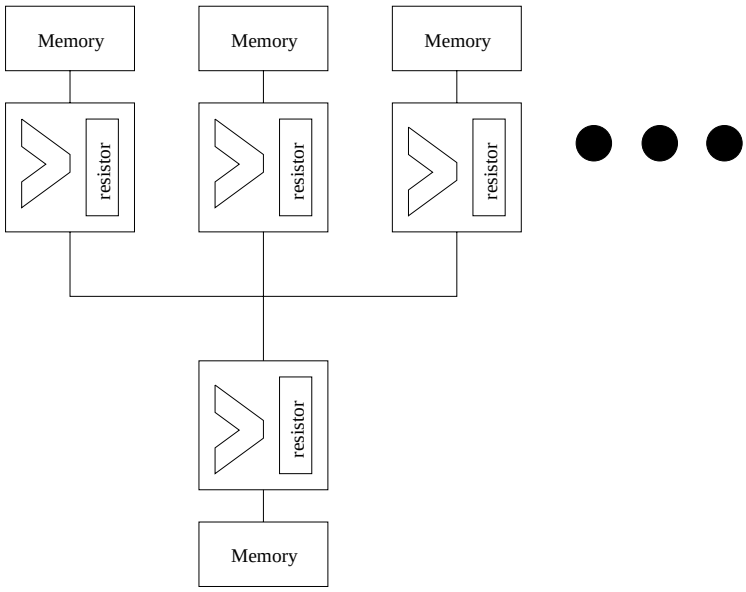


図 3.4 プロセッサの配置 [8]

この配置のコンセプトは、並列に配置してある 2 次プロセッサに計算される行列を分割して記憶していて、1 次プロセッサから送られるデータと同期して計算させようというものである。

このようにすることで、プロセッサ数を n とすると、1 次プロセッサから送るべきデータ量をを $1/n$ にしかつ 2 次プロセッサが出来るだけ休まずに演算を実行できるようにして、並列化率を高めようとする。

[8]図 3.4 = u97ryou/layout.eps

4　システムの制限を考慮したハウスホルダ法の手順

前の章の最後で示したようなプロセッサとメモリの配置で、プロセッサ間の通信がプロセッサの演算を妨げずに、ハウスホルダ変換や逆変換をすることが出来ることをこの章で示す。

4.1　ハウスホルダー変換

例として、 9 x 9 の 対称行列を考える。重要なのは、データの通り方である。データの流れが把握出来るように説明する。

4.1.1　各メモリーに行列のデータを送る

列ベクトル (行ベクトルでも同じ) に分解して、各メモリーに分配する。分け方は、図のようにする。

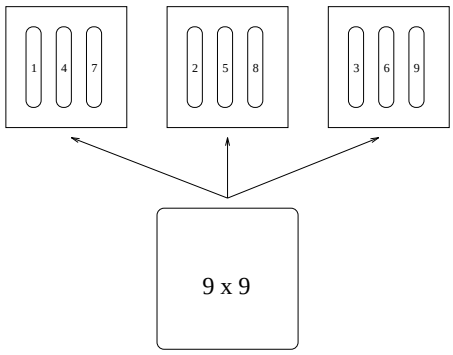


図 4.1 行列の分割 [9]

[9]図 4.1 = u97ryou/hause1.eps

4.1.2　$\omega^{(1)}\ \alpha_1\ \beta_1\ c_1$ を求める。

第 1 列ベクトル $\boldsymbol{a}$ を Primary プロセッサに送る。

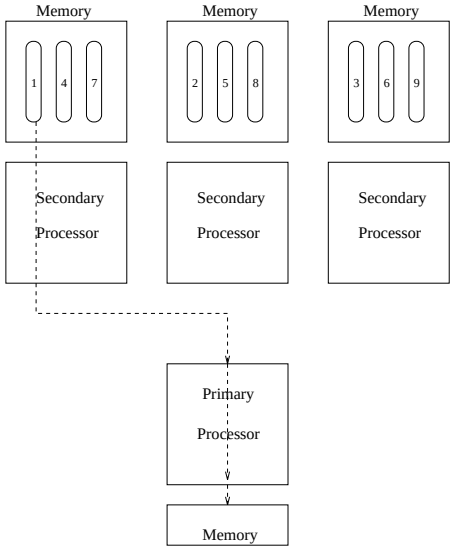


図 4.2　第 1 列ベクトルを演算処理する [10]

この課程で、

$$\boldsymbol{w^{(1)}} = \begin{pmatrix} 0 \\ a_2 + s \\ a_3 \\ \vdots \\ a_9 \end{pmatrix}, \qquad s^2 = \sum_{j=2}^{9} a_j^2 \tag{22}$$

$$c_1 = \frac{1}{s^2 + |sa_2|} \tag{23}$$

を計算する。

そして Primary のメモリへ $\boldsymbol{w^{(1)}}$ を記憶する。

主対角成分 $\alpha_1 = a_1$ と副対角成分 $\beta_1 = -s$ と c_1 を Primary のメモリへ送る。

$\boldsymbol{a^{(1)}}$ の $a_2^{(1)}$ があった Secodary メモリに $a_2 + s$ を送る。

4.1.3　p_1 を求める

$$P^{(1)} = c_1 A^{(1)} w_{(1)} \tag{24}$$

[10] 図 4.2 = u97ryou/hause2.eps

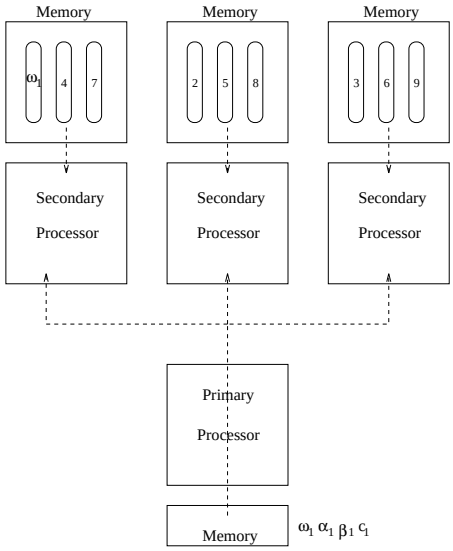


図 4.3　$\boldsymbol{w}$ と行列全体の内積をとる [11]

全ての列ベクトル と $\boldsymbol{\omega}^{(1)}$ との内積を取る。取られた内積は c_1 を乗じて $\boldsymbol{p}^{(1)}$ として Primary のメモリに記憶する。

[11]図 4.3 = u97ryou/hause3.eps

4.1.4　$q^{(1)}$ を求める

$$q^{(1)} = P_{(1)} - \frac{c_1}{2} w^{(1)} P^{(1)T} w^{(1)} \tag{25}$$

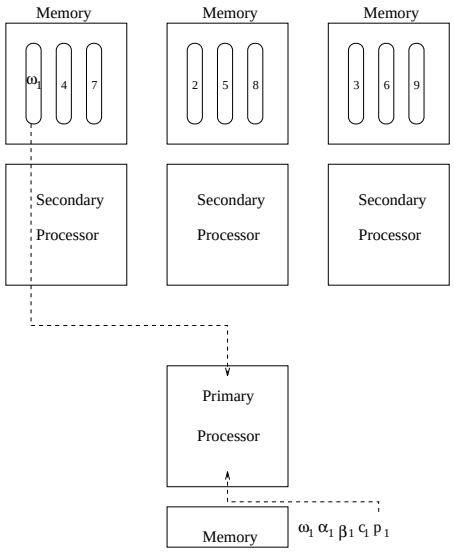


図 4.4　TMP を求める [12]

まず、$\omega^{(1)}$ と $\boldsymbol{p^{(1)}}$ との内積をとる。結果に $c_1/2$ を掛け TMP とする。

[12]図 4.4 = u97ryou/hause4.eps

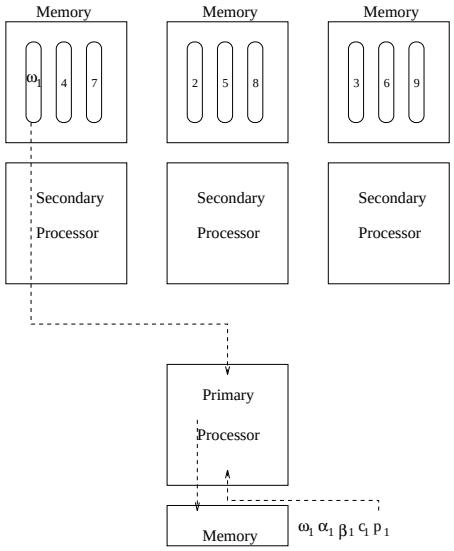


図 4.5　q を求める [13]

$\boldsymbol{\omega}^{(1)}$ の要素に TMP を掛けて $\boldsymbol{p}^{(1)}$ の要素から引く。結果を $\boldsymbol{q}^{(1)}$ の要素として Primary のメモリに記録する。

[13]図 4.5 = u97ryou/hause5.eps

4.1.5　$A^{(2)}$ を求める。

$$A^{(2)} = A^1 - (w^{(1)}q^{(1)T} + q^{(1)}w^{(1)T}) \tag{26}$$

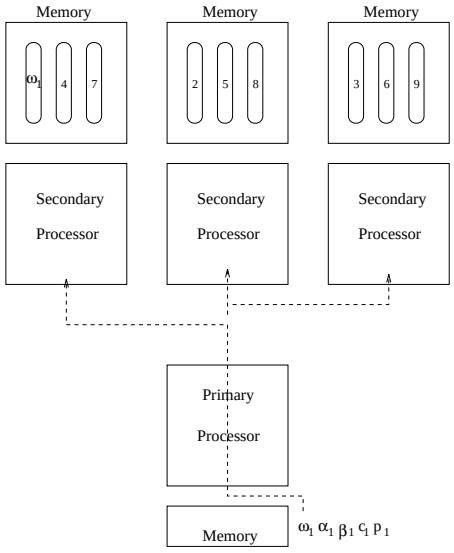


図 4.6　ω のコピー [14]

Secondary プロセッサに $\boldsymbol{\omega^{(1)}}$ の要素 $\omega_x^{(1)}$ を Secondary プロセッサに 一つずつ配る。

[14]図 4.6 = u97ryou/hause6.eps

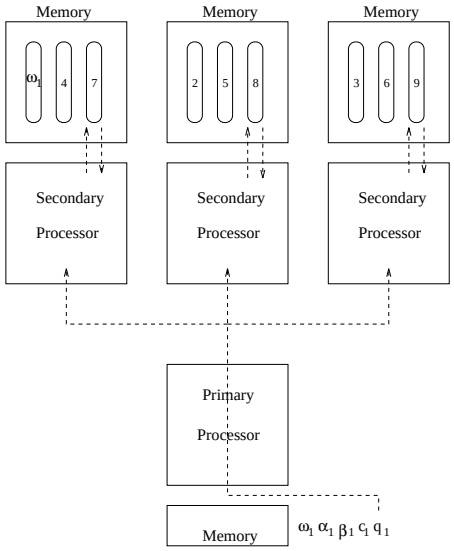


図 4.7　$A^{(2)}$ を求める [15]

それぞれの Secondary プロセッサにおいて、配った 要素 $\omega_x^{(1)}$ に 対応する $\boldsymbol{a}^{()}$ から $\omega^{(1)}$ を乗じた $\boldsymbol{q}^{(\mathbf{1})}$ を引く。 全ての $\boldsymbol{\omega}^{(\mathbf{1})}$ に対して実行する。また、 $\boldsymbol{\omega}^{(\mathbf{1})}$ と $\boldsymbol{q}^{(\mathbf{1})}$ を入れ換えて同じ事をする。

[15]図 4.7 = u97ryou/hause7.eps

4.1.6　$A^{(2)}$ の次は...

$A^{(2)}$ 求めたら 次は 同様に $A^{(3)}$ .... と 2 行 2 列の行列 $A^{(2)}$ になるまで実行する。

最終的にメモリ上のデータの配置は、次のようになる。

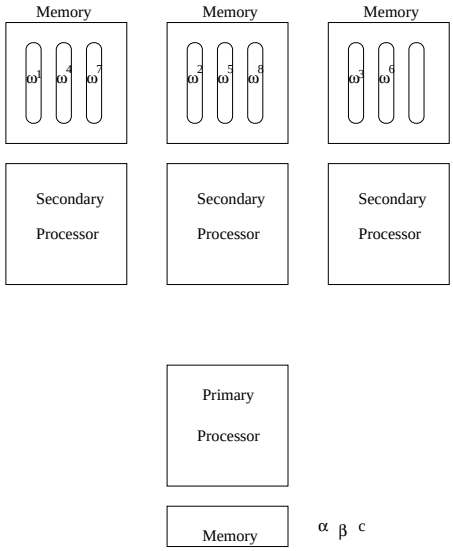


図 4.8 最終のデータ配置 [16]

[16]図 4.8 = u97ryou/hause8.eps

4.2　逆変換

4.2.1　$\omega^{(8)}$ を Primary へコピー

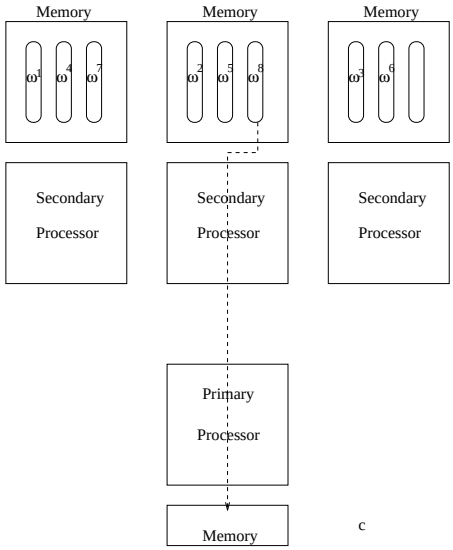


図 4.1　$\omega^{(8)}$ のコピー [17]

4.2.2　$\omega^{(8)}$ と A' の固有ベクトルとの内積 TMP をとる

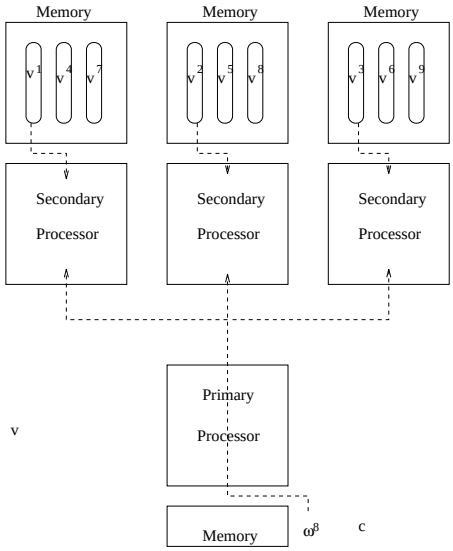


図 4.2 $\omega^{(8)}$ と A' の固有ベクトルとの内積 [18]

[17]図 4.1 = u97ryou/gyaku1.eps
[18]図 4.2 = u97ryou/gyaku2.eps

4.2.3　$\omega^{(8)}$ に C^8 と 内積 TMP を掛けて A' の固有ベクトルから引く

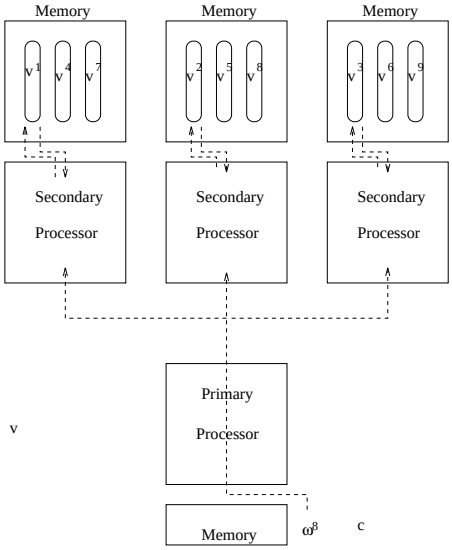


図 4.3　$\omega^{(8)}$ に C^8 と 内積 TMP を掛けて A' の固有ベクトルから引く [19]

これらを $\omega^{(1)}$ まで繰り返す事により A の固有ベクトルが求まる。

それを 全ての固有ベクトル が求まるまで繰り返す。

[19]図 4.3 = u97ryou/gyaku3.eps

5　HDL を用いた 演算プロセッサの設計

演算プロセッサを Hardware Description Language (HDL) を用いて設計しよう考えている。プロセッサを HDL で記述しシミュレーションをする事ができても、実際に動作する記述をする事は難しい。こそで演算プロセッサ設計と併せて HDL の学習を必要としている。

5.1　LSI 設計 の流れ

まず、 LSI の設計の流れを簡単に示す。

仕様設計　まず仕様設計をする。この段階では、どのようなハードウェアやソフトウェアとして実現するかを具体的には考慮せずに、アルゴリズムやシステムに要求される特性や機能について検討する。

アーキテクチャ設計　アルゴリズムを構成する各部分をどのように分解してシステムを構築するかを検討する。

機能設計　LSI の内部の機能を HDL で 詳細に記述する。記述した HDL は論理シミュレーションを行なう事ができる。

論理合成　回路の性能やテクノロジーなど設計制約条件のもとに、ゲート回路を合成する事が出来る。合成した回路に対してゲートレベルのシミュレーションが出来き、要求されるタイミングなどを検証できる。

レイアウト設計　ゲート回路のネットリストより、マスク作製のため実際のチップ上の配置や配線を設計する。こうして実配線のゲートレベルのタイミング検証を行なう事が出来る。

製造工程　チップ製造のためのレイアウトマスクを作製し、製造工程にはいる。

5.2　演算プロセッサの機能記述

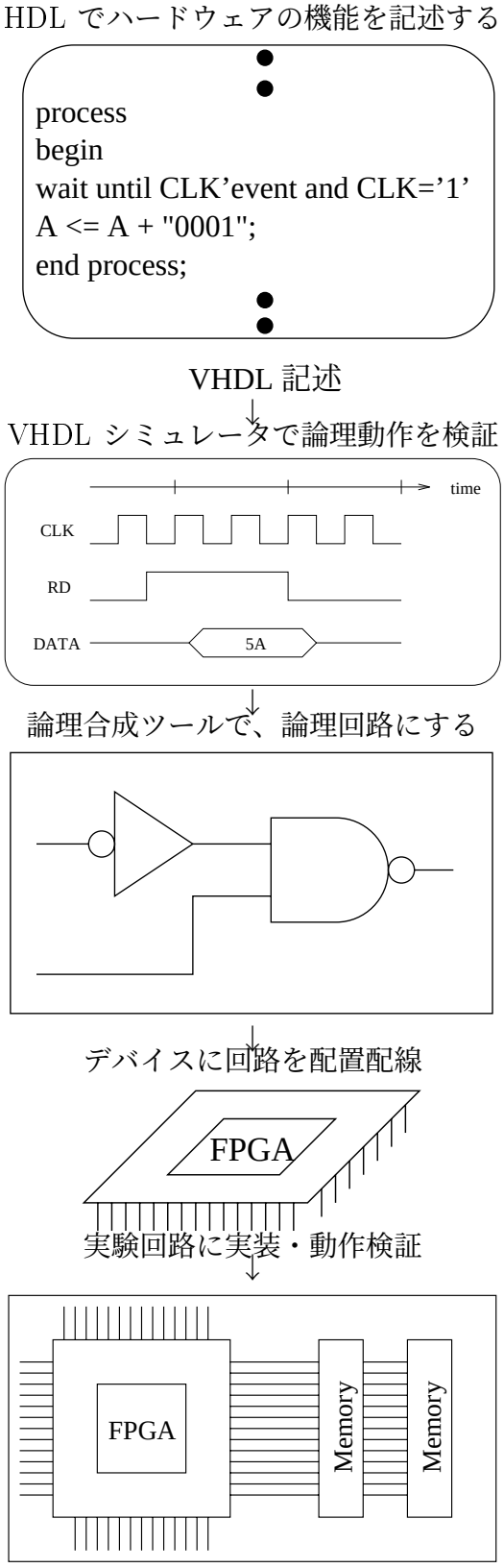
今回、仕様設計・アーキテクチャ設計として、行列の固有値・固有ベクトルをもとめるアルゴリズムと計算手順を示した。この設計を基に、 HDL の一つである VHDL で演算プロセッサを記述しようと考えている。 VHDL で演算プロセッサを記述し、シミュレーションで検証をおこなう。

しかしながら、 HDL の経験が乏しく、そのまま実際に動作する計算システムを記述するのは難しい。そこで、演算プロセッサの設計の為の VHDL 学習をするに何をすればよい考えた。

5.3　FPGA を用いたデバイス設計

PLD(Programmable Logic Device) の一種である、 FPGA(Field Programmable Gate Array) がある。これは、利用する為のソフトゥエアと論理回路のデータをデバイスに落とすためのハードウェアがあれば、自分の手元で次に示すような LSI 設計の流れと同じように LSI を作製できる。

5.4　FPGA を用いた設計のながれ



5.5　FPGA を用いた HDL 評価基盤

演算プロセッサ LSI の HDL 記述を FPGA に落として、メモリ等を実装してある実験基盤上で実際に動作検証をして、HDL 記述の善し悪しを試すことが可能である。なんどもやり直しをして、HDL 記述と実際のハードウェアの動作との対応を学ぶ事が出来る。

そのような目的のため、FPGA を用いた HDL 評価基盤を製作する事にした。製作する物は、演算プロセッサの設計に十分なゲート容量とメモリをもち、パソコン等から簡単に制御できる物にする。

6　Configuration 用インターフェースの設計・製作

6.1　目的

今回用いる アルテラ社の FPGA の FLEX シリーズ は プログラム素子が SRAM により構成されている。だから、電源を入れた後に必ず Configuration をする必要がある。また、 FPGA 上に実現されたハードウェアに対してパーソナルコンピュータから制御・通信する必要がある。これらを実現する為、パーソナルコンピュータ と FPGA 間のインターフェースを製作する。

6.2　製作したインターフェースの写真

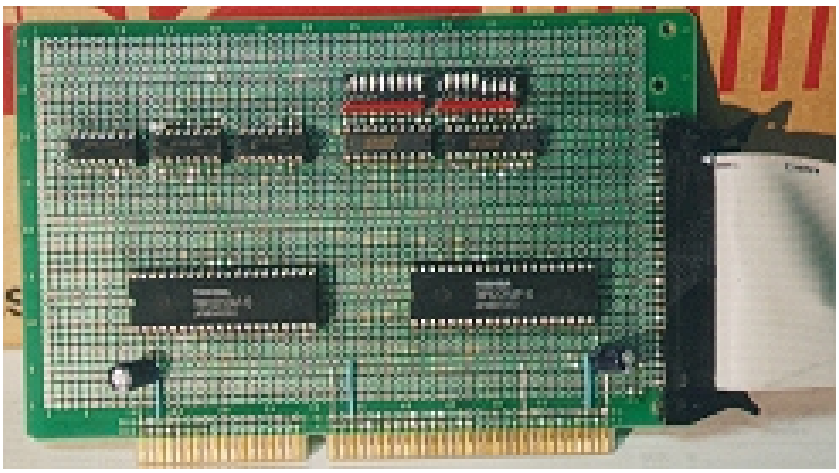


図 6.1 インターフェースカード・部品面 [20]

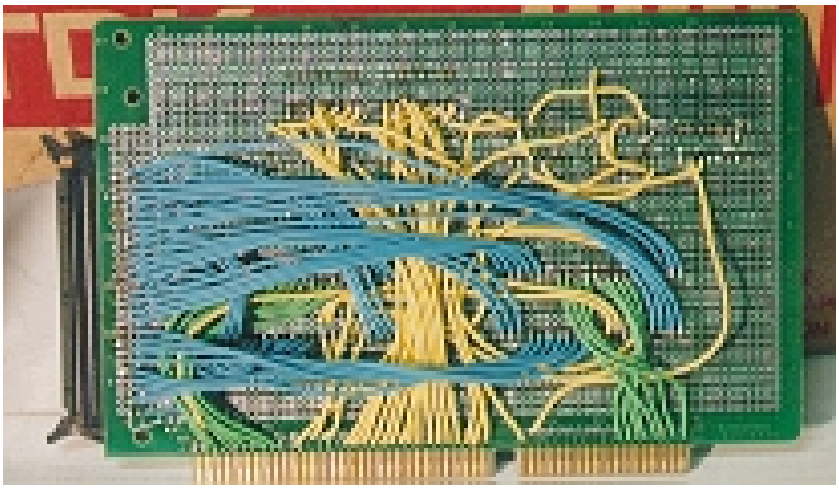


図 6.2 インターフェースカード・ハンダ面 [21]

[20]図 6.1 = u97ryou/isaif1.eps
[21]図 6.2 = u97ryou/isaif2.eps

6.3 インターフェースの設計

6.3.1 要求される仕様

今回製作するインターフェースは、 FLEX のコンフィグレーションとそれとの通信用である。コンフィグレーションでは、最低 入力 2bit 出力 3bit 必要である。また通信には、入出力 16bit は必要である。また、入力専用、出力専用の信号も欲しい。入出力の単位は 8bit になるので、コンフィグレーション用に、入力 8bit、出力 8bit、通信用に、入出力 16bit、入力 8bit、出力 8bit として、計 48bit 分の 信号が必要である。また、プログラム言語からの利用が容易である必要がある。

6.3.2 インターフェースの概要

要求を満たす為に PPI 8255 使用して、 DOS/V パソコンで利用できるよう、 ISA BUS 用カードにする。プログラム言語からも IO 命令を用いればコントロールできる。

6.3.3 インターフェースカードの諸元

表 6.1 インターフェースカードの諸元

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

今回製作するインターフェースは、要するにパラレルでディジタルの入出力をする物である。今回は専用 LSI を用いて実現する。用いたのは Peripheral Parallel Interface 8255 である。この LSI はデータ幅は 8bit であるが 2 個使用することにより 16bit 幅を実現する。また、 8bit I/O としても使用することができる。

6.4 インターフェースの回路

回路図をもとに、インターフェース回路の説明をする。

6.4.1 全体の回路図

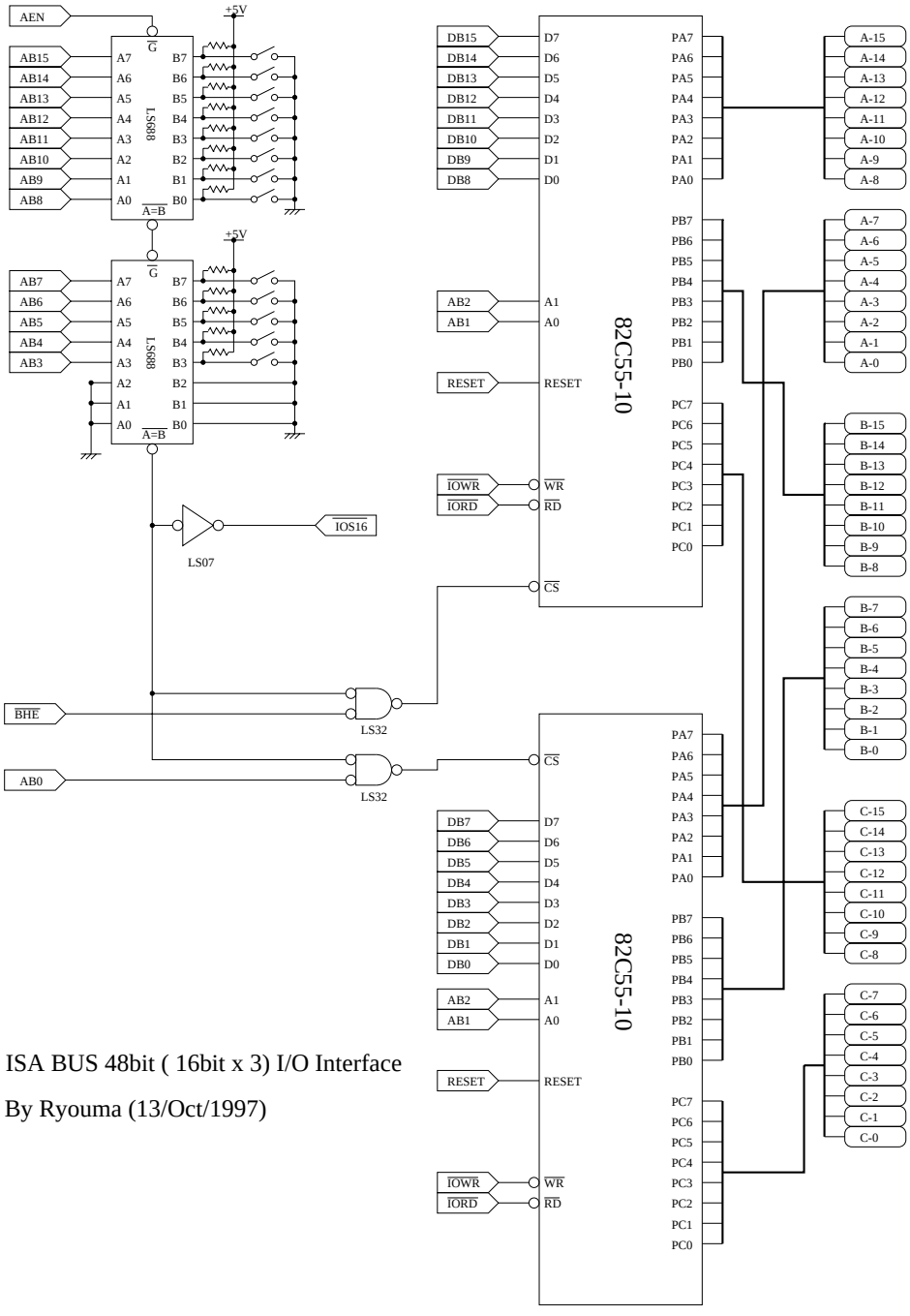


図 6.3 インターフェースカードの回路図 [22]

[22]図 6.3 = u97ryou/16bit-io.eps

6.4.2　アドレスデコード部

回路図は次の通りである。

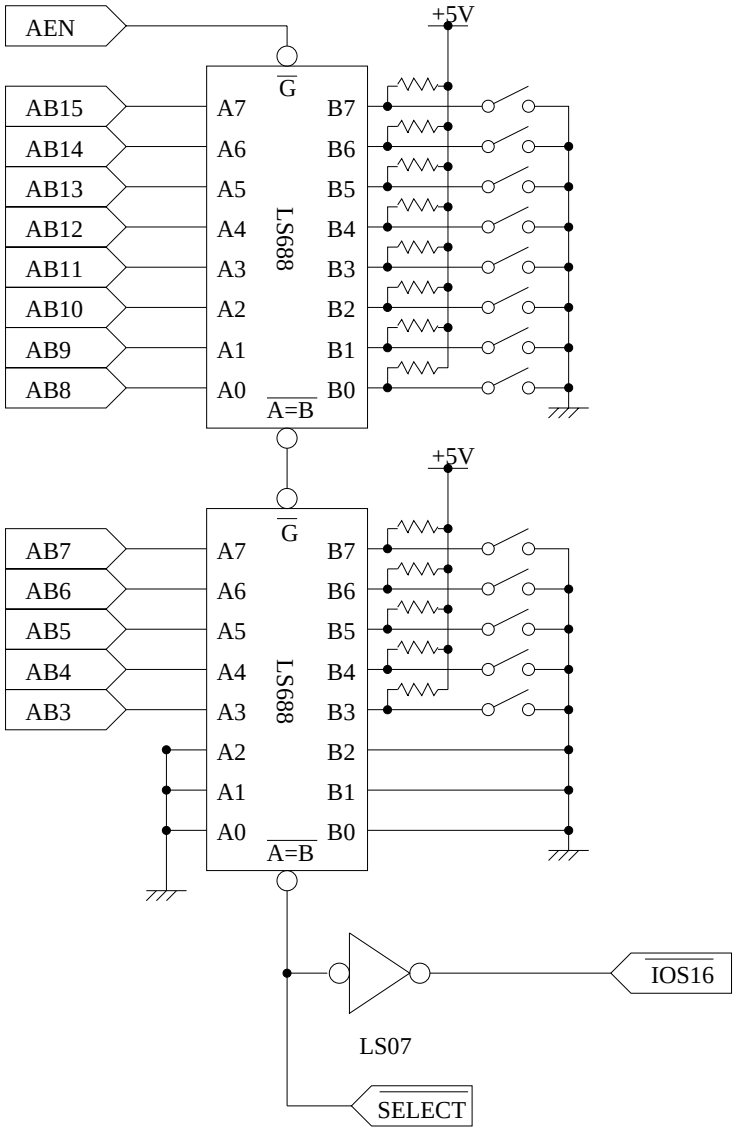


図 6.4 アドレスデコード部 の回路図 [23]

表 6.2 アドレスデコード部の信号線

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

[23]図 6.4 = u97ryou/AddDecode.eps

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

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

6.4.3 チップセレクト信号生成部

回路図は次の通りである。

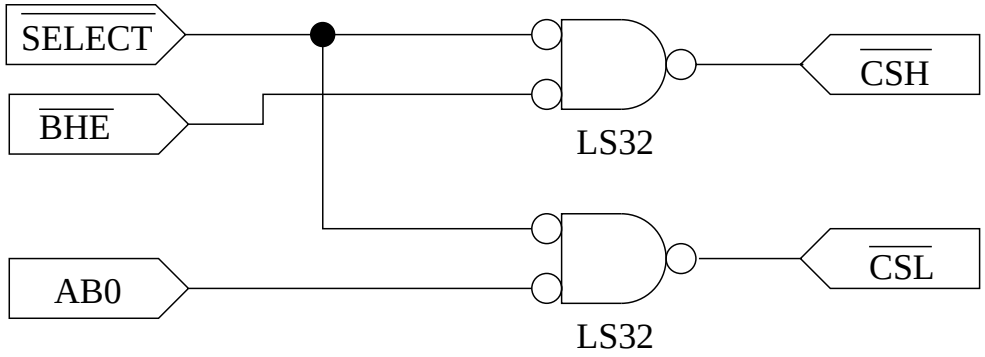


図 6.5 チップセレクト信号生成部の回路図 [24]

表 6.3 チップセレクト信号生成部の信号線

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

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

[24]図 6.5 = u97ryou/control.eps

6.4.4 PPI 8255 周辺

回路図は次の通りである。

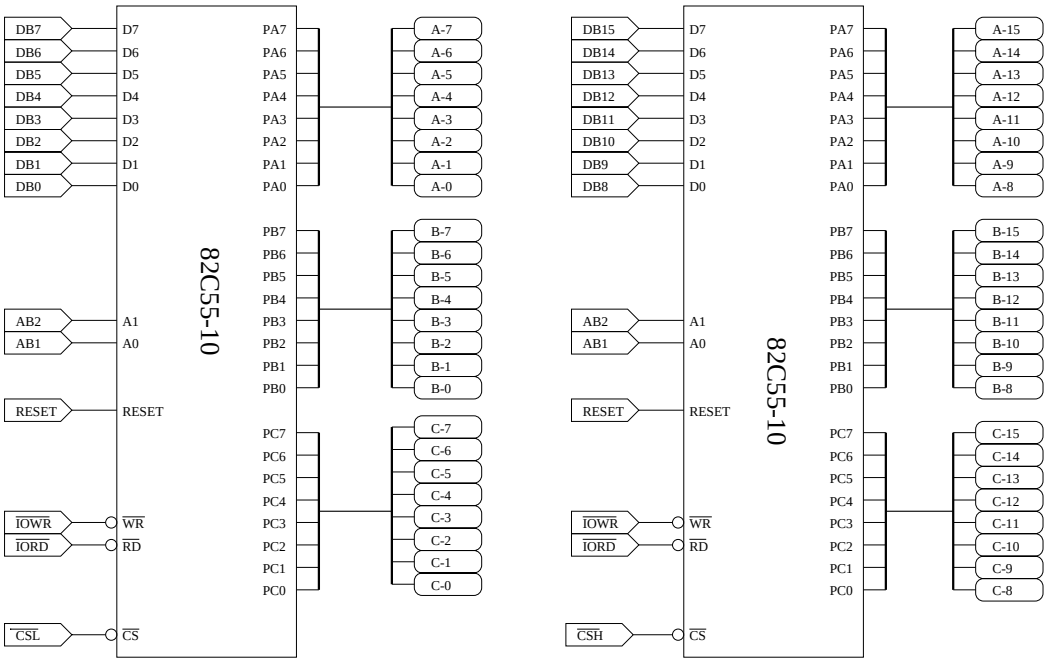


図 6.6 PPI 8255 周辺 の回路図 [25]

表 6.4 PPI 8255 周辺の信号線

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

[25]図 6.6 = u97ryou/8255.eps

6.4.5　外部端子

50pin ヘッダー。 PIN の割り付けは 図を参照。フラットケーブルを用いて他の機器と接続をする。

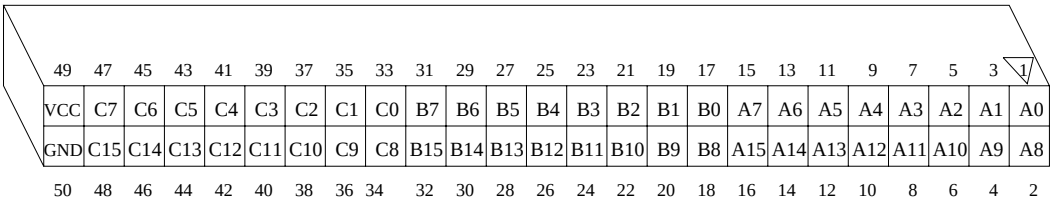


図 6.7 外部端子 pin 割り付け [26]

信号線は TTL レベルであり、プルアップはされていない。必要ならばプルアップする。ちなみに FLEX10K や FLEX8000 は TTL 入力が可能であるのでプルアップは必要無い。

VCC GND は ISA BUS から供給されている。線が細いことを考慮するとあまり容量は取れないので、外部に電源が必要な場合がある。

[26]図 6.7 = u97ryou/50head.eps

7　FPGA を用いた評価用プリント基板の製作

7.1　目的

演算プロセッサの HDL 記述がが妥当であるか、実際にハードウェア上で実現して、考察するのが目的である。

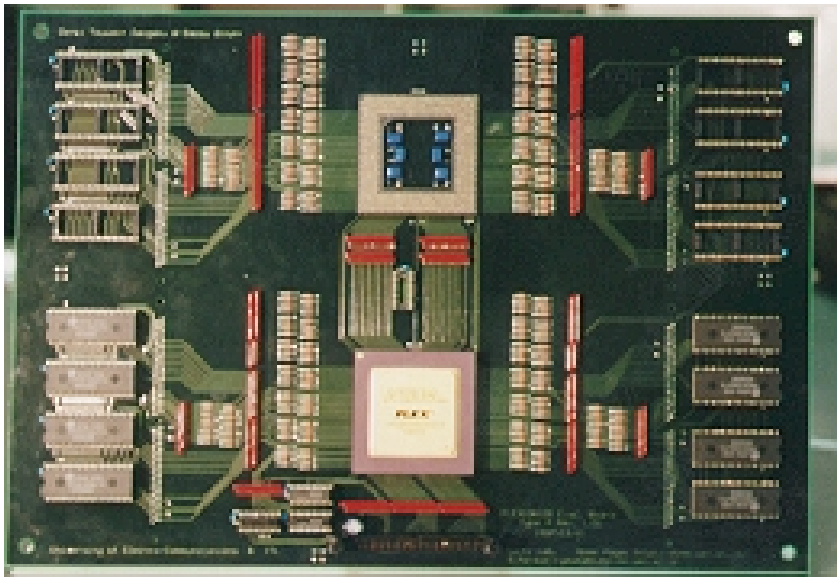


図 7.1 評価用プリント基盤 (横 38.5cm ×縦 26.5cm) [27]

7.2　プリント基盤の設計

評価ボードに要求されるのは、演算プロセッサの HDL を検証することができることである。一番重要なのが データパスの実現である。次に重要なのが、演算を実行できることである。いきなり大規模な物を作るのは無理があるので、小規模な物で実績を積みたいと思う。

[27]図 7.1 = u97ryou/eval1.eps

7.2.1　評価ボードのブロック図

今回は FLEX10K503 が 2 つ使用できるので、 FLEX 1 個につき、 32bit メモリバスを 2 系統設け、また FLEX 同士をつなぎ合わせることにした。また、回路の設計は同期設計であるので、クロック信号が必要である。

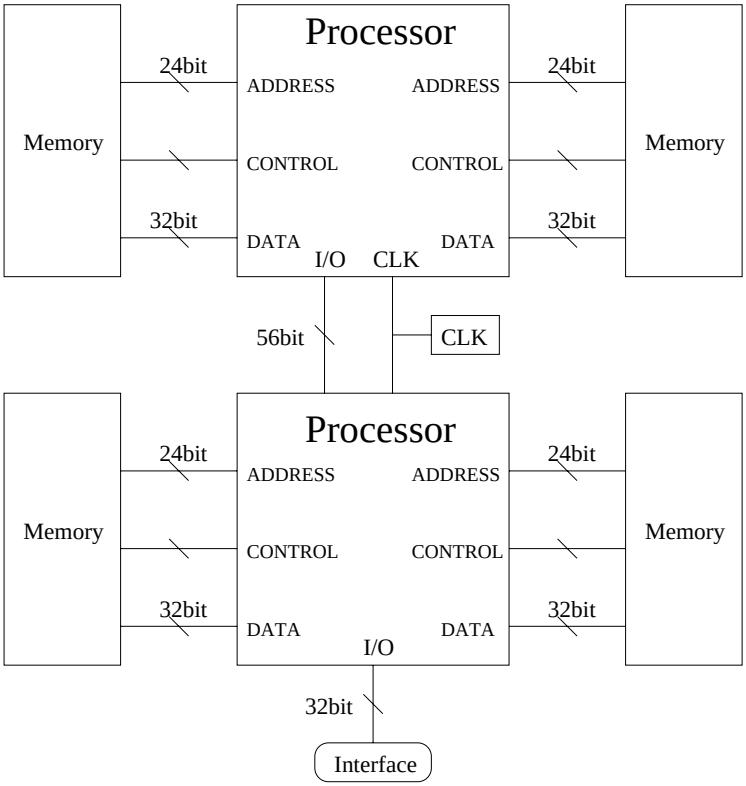


図 7.2 評価基盤のブロック図 [28]

[28]図 7.2 = u97ryou/fe1.eps

7.2.2　評価ボードのコンフィグレーション時のブロック図

評価ボード上の FLEX を独立にコンフィグレーションできるようにする。そのためには コンフィグレーション用に 5bit の信号の他に選択用の信号が必要である。選択用の信号であるが、電源投入時は回路の状態が High か Low になっているので、誤作動しないように 3bit の信号を用いて、 010 011 の時にそれぞれの FLEX をコンフィグレーションできるようにする。

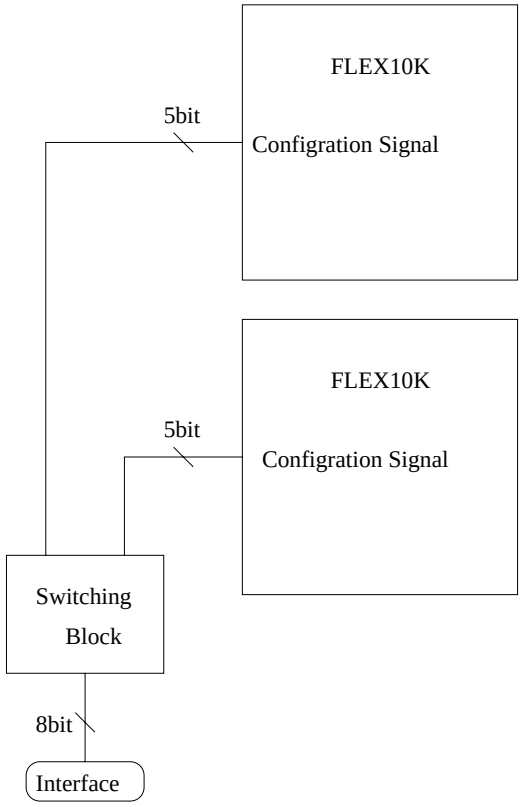


図 7.3 評価ボードのコンフィグレーション時のブロック図 [29]

[29]図 7.3 = u97ryou/fe0.eps

7.3　評価用プリント基板の諸元

表 7.1 評価用プリント基板の諸元

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

7.4　評価用プリント基板の回路の詳細

ブロック毎に詳細な説明をする。

7.4.1　コンフィグレーション部

インターフェース の信号 C0 C1 C2 C8 C9 C10 B14 B15 を用いる。

C8 C9 C10 で FLEX chip の選択をする。 LS365 を用いて、信号の切替えを行っている。 FLEX が選択されないと、 LS365 は 出力を ハイインピーダンスにして、回路から切り離す事が出来る。

注意するべき点は、 FLEX の通信用信号と共有しているので、 FLEX の回路設計時に、 C0 C1 C2 C8 C9 C10 B14 B15 を入力 もしくは ハイインピーダンス にしなければならない。そうしないと、再コンフィグレーションが出来なくなる。

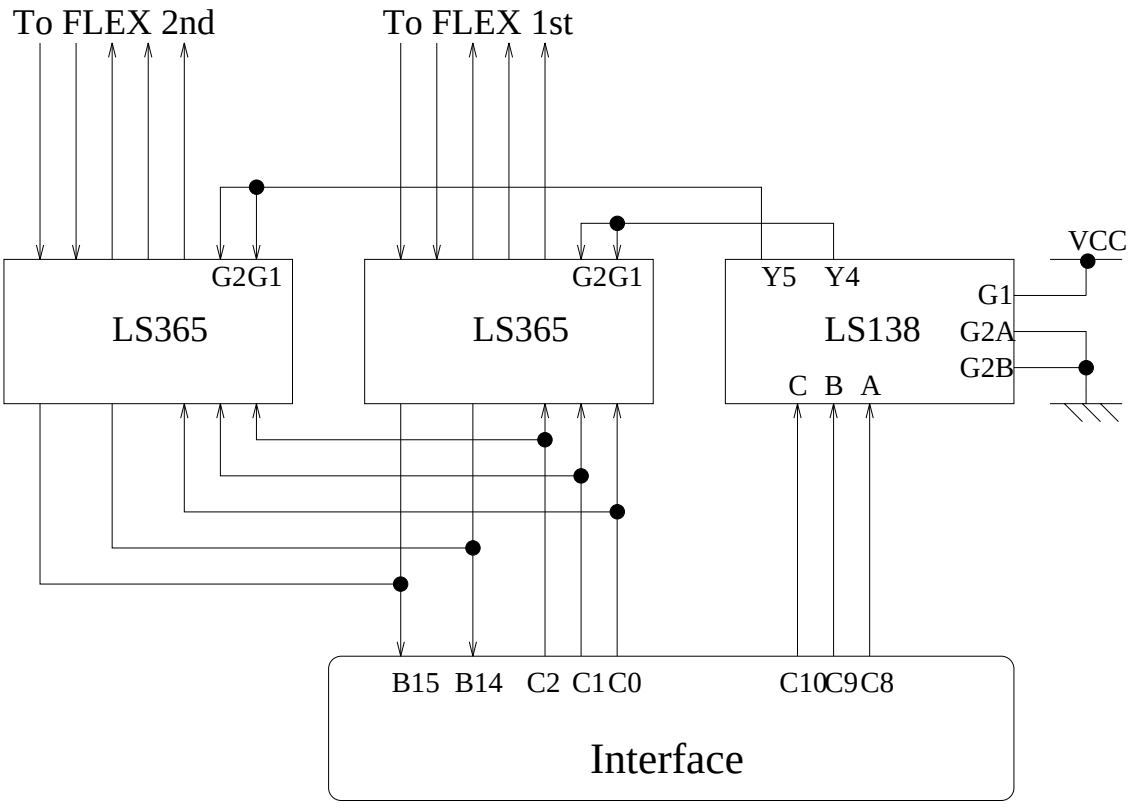


図 7.4 コンフィグレーション部の回路図 [30]

表 7.2　コンフィグレーション部の信号線

[30]図 7.4 = u97ryou/configblock.eps

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

7.4.2　パソコンとのインターフェース 部分

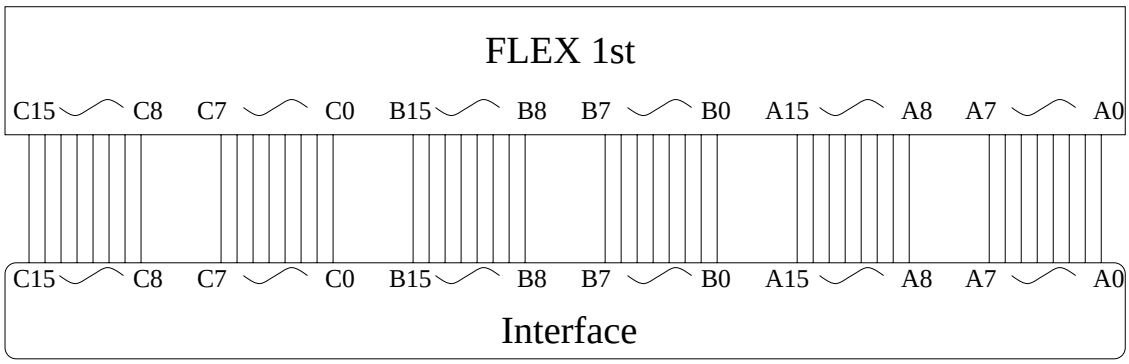
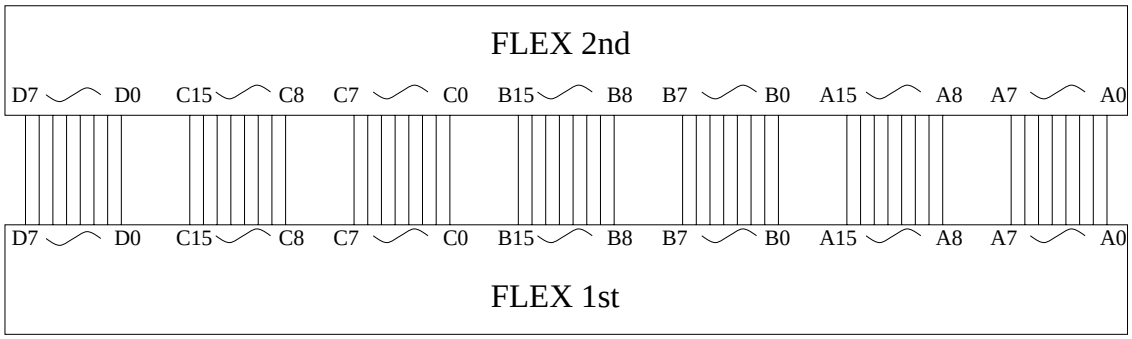


図 7.5 パソコンとのインターフェース 部分の回路図 [31]

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

7.4.3　FLEX 同士のインターフェース部分

FLEX の端子なので、自由に設計出来る。



[31]図 7.5 = u97ryou/comm.eps

図 7.6　FLEX 同士のインターフェース部分の回路図 [32]

全ての信号は 10kΩ 程度の抵抗でプルアップする。

7.4.4　メモリ部

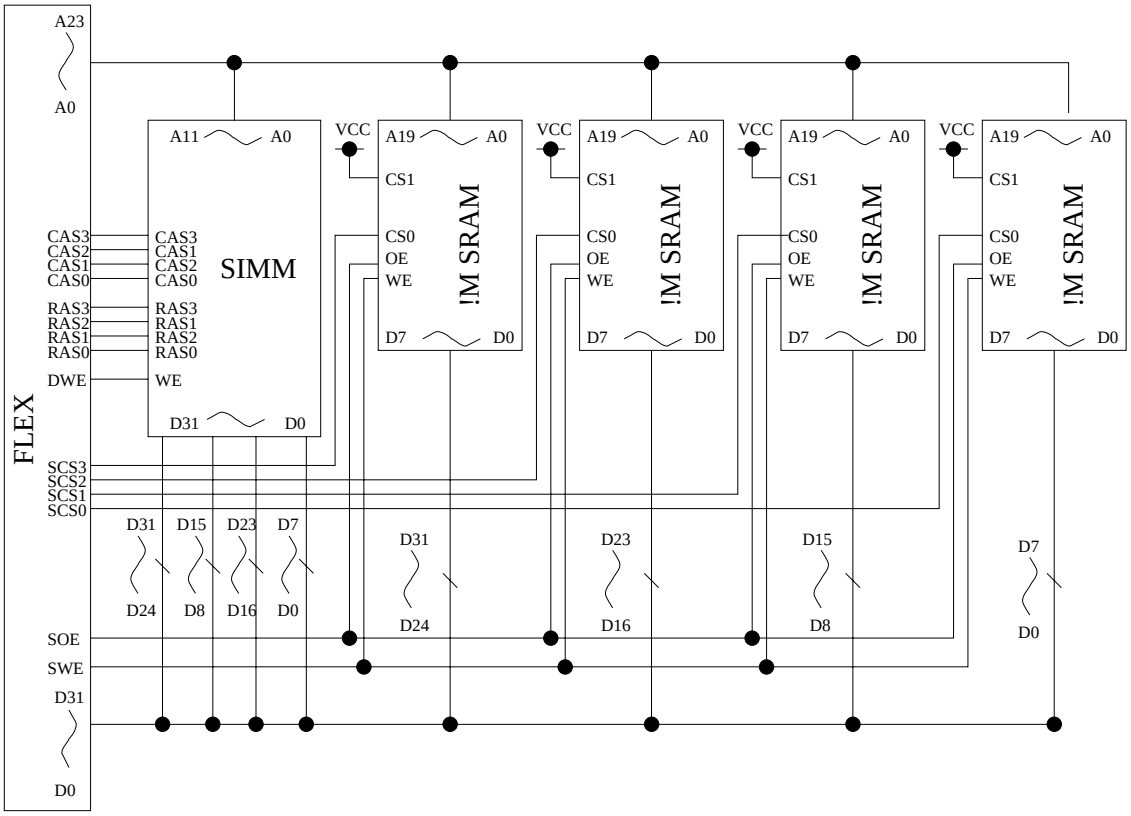


図 7.7 メモリ部の回路図 [33]

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

[32]図 7.6 = u97ryou/flexcomm.eps
[33]図 7.7 = u97ryou/memory.eps

7.4.5　クロックオシレータ部

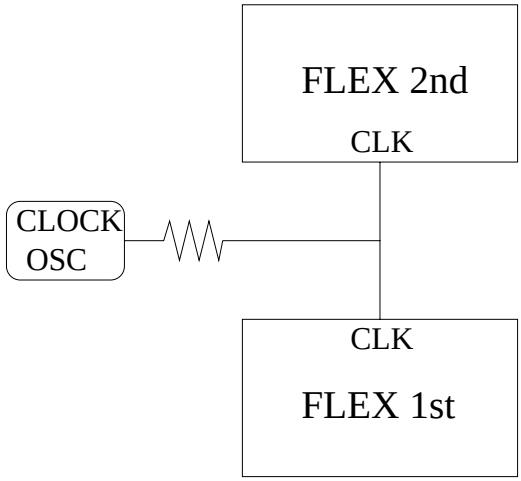


図 7.8 クロックオシレータ部の回路図 [34]

クロックオシレータの出力は、ダンピング抵抗を介して、 FLEX 1st と FLEX 2nd に供給する。ダンピンク抵抗は 22〜100Ω 位。使用するオシレータは、 10MHz で、 8pin DIP or 14pin DIP 型の物が使える。

[34]図 7.8 = u97ryou/clock.eps

7.5　FLEX の PIN の割り付け

7.5.1　インターフェース部 下側

表 7.3 インターフェース部 下側の pin の割り付け

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

7.5.2　インターフェース部 上側

表 7.4 インターフェース部上側 のピンの割り付け

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

8　インターフェースカードの使い方

8.1　ベースアドレスの設定

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為の 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。

今回製作した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できる。

ベースアドレスは 0x0000 から 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。(なお、下位 3bit は 設定しても無視される)

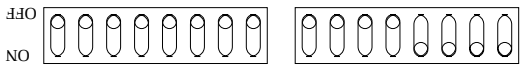


図 8.1 DIP スイッチ [35]

OFF が 1 で ON が 0 である事に注意する事。

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と。

次の様にマッピングされる。

表 8.1 インターフェースカード の I/O マッピング

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

8.2　ISA BUS に 装着

設定したら ISA BUS に装着する。そして、フラットケーブルを接続する。

[35]図 8.1 = u97ryou/ifdip.eps

8.3　プログラムからの使い方

8.3.1　I/O 関数

C 言語 (Borland C++ 等) では、I/O 関数 が使える。

int inport()

unsigned char inportb()

void outport()

void outportb()

int inp()

unsigned inpw()

int outp()

unsigned outpw()

などの関数がある。

これらの関数については、言語のマニュアルを参照すること。

8.3.2　関数の使いかた例

ベースアドレスを 0xfff0 として、ポート A の下位 を読み込むならば、

unsigned char a;

a = inportb(0xfff0)

とする。

また、 ポート A を 16bit で読む場合は、

unsigned int a;

a = inpw(0xfff0);

とする。

8.3.3　インターフェースの初期化

I/O として使うには初期化が必要である。初期化には 8255 の コントロールワードに初期化データを書き込むことにより行う。コントロールワードへの書き込みは、他のポートと同じように I/O 関数で行う。ただし、読みだしは出来ないので注意すること。初期化する事により、ポートの入出力方向やモードを決定する。

ベースアドレスを 0xfff0 とすると、

コントロールワードは 下位バイトは 0xfff6 上位バイトは 0xfff7 に書き込めばよい。

全てのポートを 入力としたい場合は、

outp(0xfff6,0x9b); outp(0xfff7,0x9b);

または、

outpw(0xfff6 , 0x9b9b);

とする。

9　評価ボードの使い方

9.1　コンフィグレーション

ここでは、 FLEX の コンフィグレーションデータのダウンロード について説明する。

9.1.1　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 9.1 インターフェースの I/O ポートのモード

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

と設定しなければならない。他の ポートは自由に設定出来る。

9.1.2　コンフィグレーション後のモード変更の注意

PPI8255 は モードを変更すると 全ての出力バッファを 0 にクリアする。このため、モードを変更すると、 nCONFIG が Low にされてしまい、 FLEX が コンフィグレーションモードに入ってしまう事がある。

これを防ぐには、

1. ポート C 上位は必ず出力にする。
2. コンフィグレーションのあとは、 FLEX の選択を 000 に 設定しおくこと。

この 2 点を守れば OK。

9.1.3　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。

9.1.4　FLEX 2nd のコンフィグレーション時の注意

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 入力にしておかなければならない。

9.2　コンフィグレーション用プログラム例

```
#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()
{
long oo,i;

printf("FLEX10K down-loader\n by ryouma since 1997\n");
/* data の読み込みから */


FILE *fp;

if ((fp = fopen("test.ttf", "rt"))
   == NULL)
   {
  fprintf(stderr, ".ttf を開けません \n");
  getch();
  return 1;
   }
```

```
outpw( P_CTRL , 0xCAC8);
outp(P_CL,0xff);
outp(P_CH,0x04);

printf("%02x\n", inp(P_BH) & 0xC0);

printf("start.\n");
getch();


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

outp(P_CL , 0 );

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;
}



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

}



for ( i= 0 ; i < 150000l ; i++)
{
  unsigned char c;
  int c1;
  if ( EOF == fscanf(fp,"%d,",&c1) ) break;
  int j;
//  printf("%ld ",i);
  c = c1;

 for ( j=0 ; j<8 ; j++)
  {

  outp( P_CL , nCONFIG |        ( c & 1 ) );

  inp(P_CL);
  inp(P_CL);

  outp( P_CL , nCONFIG | DCLK | ( c & 1 ) );
  c >>= 1;
```

```
  }

if ( (inp( P_BH ) & 0xC0 ) == 0xC0 ) break;

  }

  printf("%ld",i);

for (i = 0 ; i< 11 ; i++)
{
 outp( P_CL , nCONFIG );
 outp( P_CL , nCONFIG | DCLK);

}


if ( ( inp(P_BH) & 0xC0 ) != 0xC0 ) { printf("Abnormal end.\n"); getch();return 1;}

printf("Normal end\n");
printf("%02X\n",inp(P_BH) & 0xC0);

getch();
return 0;

}
```

10　結論

物性の研究には、分子軌道法による計算が用いられている。この計算は永年方程式を解く計算が計算時間の大部分を閉める。この永年方程式は、行列の固有値・固有ベクトルを解く問題に帰着される。行列の次元を N とすると、固有値・固有ベクトルを求めるのに N^3 に比例する計算がかかるといわれている。複雑な分子構造の計算の場合 N が大きくなるが、その分計算時間も膨大な物となる。計算時間の大部分を固有値・固有ベクトルを求める計算に費されているので、この部分を特に高速化することにメリットがある。

計算の高速化の方法として多数の演算プロセッサをもつ並列コンピュータを用いて計算を高速化しようという試みが多くなされている。しかしながら、固有値・固有ベクトルを求める計算の並列化と高速化は単純には結び付かない。これは一般の並列コンピュータにいえる問題なのだが、アルゴリズム上必要な演算を分解して別々に計算することは可能な計算でも、通信の量が多くなるような演算の分解の仕方になるなら、計算時間の短縮は果たせない。これは、プロセッサ間の通信には有限の時間が必要ある事から、いくら演算の速度が速いプロセッサでもデータを演算プロセッサに配っている時間が多くなるなら並列化率が低くなってしまい、結局計算時間の短縮にはならない。

固有値・固有ベクトルを求める計算のアルゴリズムの多くは、この並列コンピュータの問題の演算プロセッサ間の通信が多くなって並列化率が低くなる。これは計算のなかで、分解される計算の一つ一つに注目すると、分解された計算について、計算されるべきデータの量と演算の回数を比べると同程度になるからである。つまり計算を分解してもデータを配る時間も同じだけ増加してしまう。そうなる理由として、固有値・固有ベクトルを求める計算アルゴリズムは本質的に行列の線形演算の繰り返しと同じであるからであろう。

計算時間の短縮つまり並列化率を高めるには、演算プロセッサ間の通信を減らす事が必要である。固有値・固有ベクトル問題の計算の場合、ある共通のベクトルデータと、別々のベクトルデータの計算の群にになり、通信のほとんどが共通のデータをプロセッサに配る事に費されている。この部分を工夫すれば並列化率が向上する

と考えた。それが 行列のデータを分割して記憶し、並列プロセッサのデータに入出力はそれぞれの持つメモリからとプロセッサ共通のデータバスからおこない、共通のデータバスは一つだけある親プロセッサが入出力をおこなうようにする計算システムである。

ではこのようなメモリとプロセッサの配置で、この研究で行なおうとしている、ハウスホルダ変換による 3 重対角化や 2 分法、逆反復法 逆変換が行なえるのかといった問題がある。この論文では、ハウスホルダ変換と逆変換についてアルゴリズムの上では、滞りなくプロセッサに計算すべきデータを送り、計算を完了できることが示された。

最終的な目的は、専用プロセッサの設計であるが、プロセッサの設計は、ハードウェア記述言語 (HDL:Hardware Description Language) の一つである VHDL を用いようと考えている。今回、仕様設計・アーキテクチャ設計として、行列の固有値・固有ベクトルをもとめるアルゴリズムと計算手順を示した。この設計を基に、 HDL の一つである VHDL で演算プロセッサを記述し、シミュレーションで検証をおこなう。しかしながら、 HDL の経験が乏しく、そのまま実際に動作する計算システムを記述するのは難しい。そこで、演算プロセッサの設計の為の VHDL 学習をするため、 FPGA(Field Programmable Gate Arry) 評価ボードを製作した。この 評価ボードは、 FPGA を用いており、 HDL 等の設計をすぐに回路として FPGA にダウンロードして試すことが出来る。このボードはパソコンから 16 bit 幅の IO ボードからコントロールされる。この評価ボードは、メモリを一個の FPGA につき 2 系統あり、 FPGA は 2 つあるので、並列動作を評価することも可能である. また、片方の FPGA を動作させたまま片方の動作を変更することも可能である。動的に動作の変更をすることもできる。

この評価ボードを用いて、 VHDL の動作の検証を行うことが出来るようになった。次にするべきことは VHDL によるプロセッサの記述である。

謝辞

本研究及び論文作成に当たり、懇切なる御指導を賜わりました指導教官の齋藤理一郎助教授に厚く御礼の言葉を申しあげます。

本研究を進めるにあたり、研究室セミナー等にてさまざまな御指導を賜わりました、木村忠正教授、湯郷成美助教授、一色秀夫助手に感謝致します。

また、研究活動をともにし、多くの助力をいただいたグェン・ドゥック・ミンさんに感謝いたします。

そして、数々の御援助、御助言をしていただいた竹谷隆夫さんをはじめ、八木 将志さん、木村・齋藤研究室の大学院生、卒研生の皆様に感謝の意を表したいと思います。

共同研究のパートナーとして、ハードウェアに関して御教授を賜わりました高田亮氏をはじめ、プリント基板の設計に御助力を頂きました、(株) 画像技研の皆様に感謝します。

参考文献

[1] 数値計算の手順と実際　高田勝 春海佳三朗　コロナ

[2] FORTRAN77 数値計算プログラミング　森正武　岩波書店

[3] IBM PC と ISA バスの活用法　ドランジスタ技術編集部編　CQ 出版社

[4] ALTERA Data Book 1996　ALTERA Corp. ALTERA Corp.

A　付録 FLEX10K

今回用いる FPGA は、 ALTERA 社の製造する FPGA の FLEX シリーズを用いる事にした。特徴として、他の会社の製品に比べ、実現できるゲート数が大きいことがありこの点に注目した。とくに大きな物では 10 万ゲート相当の回路を実現できる (FLLEX10K100)。プログラム素子は SRAM で構成されているので、電源を投入する度に内容を書き込む必要がある。書き込みに関してこの FPGA は 基板上に実装した状態で書き込みををする。

A.1　デバイスのコンフィグレーション

アルテラの FLEX シリーズ の記憶素子は、 SRAM で構成されている。 SRAM 素子は電源が入っていなければ内容が消えてしまう。このため、電源を投入する度に、設計した回路のデータ (configuration data) を、 SRAM に送ること (down load) により、デバイスに機能を持たせる。

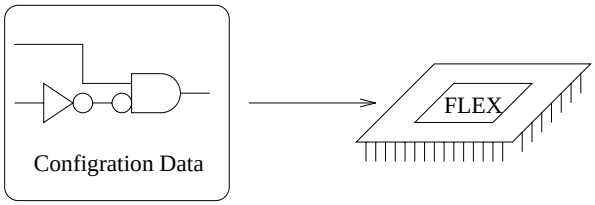


図 1.1　回路データのダウンロード [36]

設計した回路をデバイスに乗るデータにしてダウンロードすることをコンフィグレーションという。 FLEX の場合、デバイスを基板上に実装したままの状態でコンフィグレーションする。このデバイスのコンフィグレーション法には幾つかの方法があるが、今回は比較的簡単な、 Passive Serial 法を用いる。この方法は使用する信号線が少ない利点がある。他に Configration EPROM 法、 Passive synchronous 法、 Passive parallel asynchronous 法がある。

[36]図 1.1 = u97ryou/flexconf.eps

A.1.1　Passive serial 法で使用される端子 (FLEX10K)

表 1.1 Passive serial 法で使用される端子 (FLEX10K)

MSEL0	入力	方法選択 0
MSEL1	入力	方法選択 1
nCE	入力	Low にしないとコンフィグできない
nCE0	出力	カスケード接続用信号
nSTATUS	双方向	コンフィグ status 信号
nCONFIG	入力	コンフィグ制御信号
CONF_DONE	双方向	デバイスがコンフィグされると High
DCLK	入力	データ入力クロック
DATA0	入力	データ線

実際に外部との接続に用いるのは、 nSTATUS nCONFIG CONF_DONE DCLK DATA0 の 5 本である。

A.1.2　コンフィグレーション法の選択 (FELX10K)

表 1.2 コンフィグレーション法の選択 (FELX10K)

MSEL0	MSEL1	コンフィグレーション法 (FLEX10K)
0	0	コンフィグレーション EPROM or passive serial
1	0	Passive parallel synchronous
1	1	Passive parallel asynchronous

MSEL0 と MSEL1 を用いて、 Configration 法を選択する。今回は Passive Serial 法 を用いるので、 MSEL0 MSEL1 ともに 0 にする。

A.1.3　コンフィグレーション回路

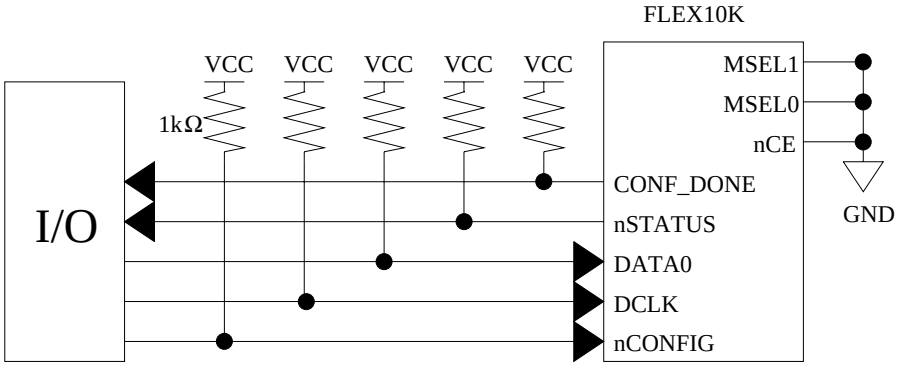


図 1.2 コンフィグレーション回路 [37]

[37]図 1.2 = u97ryou/config.eps

基本的に 3 本の入力と 2 本の出力 合わせて 5 本 の信号を 接続する。これらの 信号を外部の I/O で制御する。

A.1.4　コンフィグレーションのタイミング波形

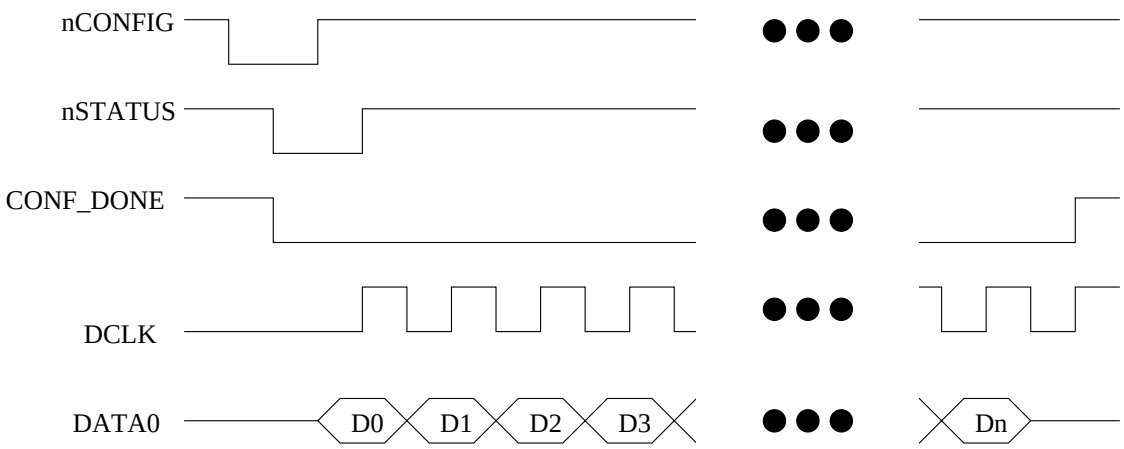


図 1.3 コンフィグレーションのタイミング波形 [38]

表 1.3 コンフィグレーション端子の意味

nCONFIG	パルスを与えることにより、コンフィグの開始の意味になる。 パルスは幅は 2μ 以上。
nSTATUS	最初のパルスはデバイスがコンフィグモードになったことを示す。 コンフィグの途中で Low になるとエラーを示す。
CONF_DONE	デバイスの状態を示す。 Low で未コンフィグ状態を示す。
DCLK	立上りで、 DATA の読み込みを行う。 10Mhz 以下。
DATA0	ここに 1bit づつ、 configration data を与える。 TTF のデータは 1byte 単位だが、下位ビットから先に送る。

A.1.5　device configuration file

FLEX の コンフィグレーションデータ は、 ALTERA の配置配線ツールの Max+PLUS II で生成できる。このデータファイル を device configuration file 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 file である。このた

[38]図 1.3 = u97ryou/timewave.eps

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容易である。

他に注意すべきこととして、実際のダウンロード時に 先頭に 0xFF を付加しておかないとデバイスが動作しない。データシート等にもその記載が無いようだが、気をつけるべきである。

B 付録 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、随時読みだしが可能である。A B C を 3 つのポート として使う場合と、C ポートを制御用信号として A ポート用 B ポート用の2つに分けて使う場合がある。入出力は TTL コンパチブルである。

B.1 I/O ポートの 3 つのモード

PPI8255 のI/O ポートは3つのモードを持っている。 Mode 0 と Mode 1 と Mode 2 である。 Mode 2 は ポート A のみ設定可能であるが、他のモードは ポート A ポート B それぞれ別に指定できる。つまり、ポート A を mode 2 ポート B を mode 0 という設定も可能である。

B.1.1 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。

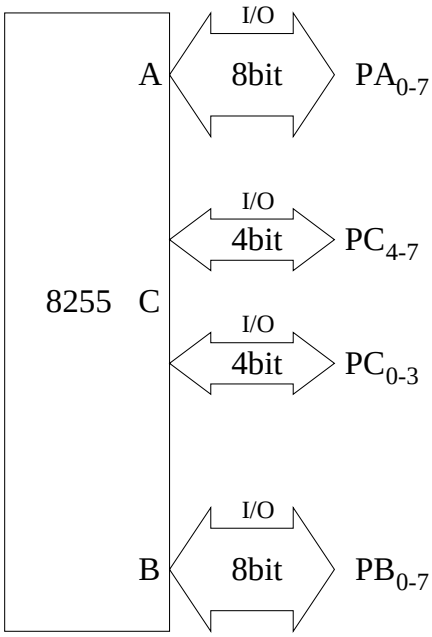


図 2.1 Mode 0 [39]

[39]図 2.1 = u97ryou/ppi8255-mode0.eps

B.1.2 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。

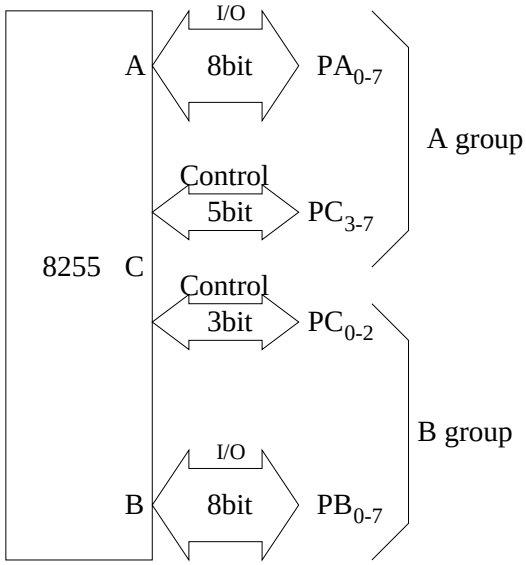


図 2.2 Mode 1 [40]

[40]図 2.2 = u97ryou/ppi8255-mode1.eps

B.1.3 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 モード 0 モード 1 どちらでも構まわない。

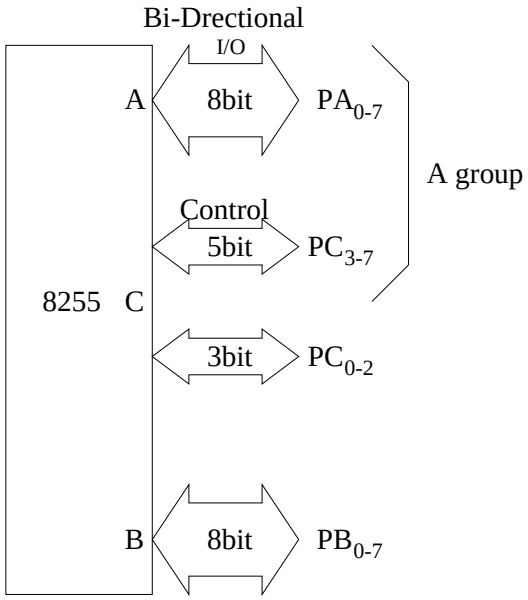


図 2.3 Mode 2 [41]

[41]図 2.3 = u97ryou/ppi8255-mode2.eps

B.2 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出来ない。

コントロールワードの ビットの内訳は次の通りである。

表 2.1 コントロールワードの ビットの内訳

D7	1 → モード設定, 0 → ビット・セット / リセット
D6 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

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。

B.3 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出来る。

表 2.2 ポート C の ビット・セット / リセット モード

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

例として、 00000001 なら bit 0 が 1 にセットされる。

C　工作についての Tips.

C.1　拡張 Card 用 基板

基板は、サンハヤト の ハーフサイズ ISA ボードを用いる。あらかじめ電源ラインが櫛状に配線されているものを使う。

C.2　配線用コード

配線には、電子工作用の耐熱コードを 用いる。ϕ は 22 か 24 くらいが良い。

C.3　IC ソケット

全ての IC には 丸ピン IC ソケット用いる。半田付けの不良を無くすために必要。IC 本体よりも値段が高い。

C.4　バイパスコンデンサー

通称 パスコン、電源ラインの安定化のため必要。基板上の電源ラインはロジックデバイスのスイッチングノイズなどで、電圧レベルが変動している。これを吸収して、安定させるのがパスコンである。

C.4.1　IC の 電源端子

0.1 μF の積層セラミックコンデンサが良い。FLEX は大きいので 1.0μF のものを使う。とりあえず全ての IC に つけておけば安心。

C.4.2　回路全体の電源

電源に 200 μF 程度の 電解コンデンサを入れる。場合によってはもっと大きいものを使う。

C.5　半田コテ

ハンダゴテは 20W から 30W のものを使う。小手先は細い方が作業しやすい。半田付けは、下図の用になるのが望ましい。

C.6　半田

半田は、ヤニ入りでスズ 60% の物を用いる。ペーストは接触不良の原因になるので使わない。ちょどよい太さの物を使う。