進化型ハードウェアの設計と応用に関する研究

1999年3月

梶谷　勇

進化型ハードウェアの設計と応用に関する研究

１９９９年　３月

梶谷　勇

梗概

近年のEDA（Electronic Design Automation）技術の進歩により，複雑な回路を簡単に合成する事が可能となってきた．例えば，シノプシス社などの論理合成ツールを使う事で，Veirlog等のHDL（Hardware Description Language）記述から回路を合成する事ができる．ただし，このような回路合成が可能なのは，入出力仕様を合成時点で完全に決定できる場合に限られている．

しかしながら，入出力仕様が完全に決定できない場合にも，回路を合成する必要性がパターン認識等の応用において指摘され始めている．例えば，手書き文字の認識を高速に行う回路の場合，手書き文字パターンにはノイズがのりやすいために，入出力仕様を正確に決めることができない．さらに，入力パターンの数が多すぎるため，全入力パターンに対する正しい出力を決定できない．このように入力パターンにノイズが入ったり，あるいは一部の入力パターンに対する正しい出力しか決定できない場合には，従来のEDAツールを適用することはできない．

このような回路の入出力仕様を決定できない場合の合成方式として，遺伝的アルゴリズム（Genetic Algorithm,GA，以下GAと略す）に代表される進化的探索手法を用いる方式が提案されている．この方式は，GAなどで回路を合成するので，進化型ハードウェア（Evolvable Hardware；以下EHWと略す）と呼ばれており，１９９２年の提案以降，活発に研究されてる．

本研究では，このEHWの従来の研究における二つの問題点，すなわち，システムの大きさと，回路合成の遅さの問題を解決するために，自律的に回路構成を動作環境に適応可能なEHWチップを設計しVLSIに実装した．このEHWチップは，主に，再構成可能ハードウェアとGA操作用の回路からなり，外部のCPUを介さずに，再構成可能ハードウェアの回路を，自律的に合成，再構成可能である．

このEHWチップの設計にあたり，まず，その中心となるGA操作用の回路をより小さく実装するために，最もハードウェアへの実装に適したGA操作の組合せを提案した．そして，そのGA操作の組合せを高速に実行する回路をFPGA (Field Programmable Gate Array) に実装して基本動作を確認した．

次に，1997年～1998年にかけてEHWチップversion 1（EHWv 1）を設計し，現在，EHWチップversion 2（EHWv 2）を設計中である．このEHWv 2は，１９９９年２月完成予定である．

このEHWv 1の設計では，GAにより再構成可能ハードウェアの回路構成を動作環境に適応させる処理を高速化するために，最小項学習法と呼ぶ方式を提案し，その効果を確認した．EHWv 2の設計では，より小さく実装するために，GRGA (Gene Replacement Genetic Algorithm) による回路合成方式を提案した．つまり，最小項学習法では，GAで回路を合成する再構成可能ハードウェアの他に，最小項学習用の再構成可能ハードウェアが必要であったのに対し，GRGAでは，最小項学習用の再構成可能ハードウェアは必要なく，より小さく実装可能である．

本研究では，EHWチップの応用の一つとして，筋電義手と呼ばれる義手の制御回路の一部に応用可能であることを示した．筋電義手とは，筋肉を動かしたときに体表面で観測される数μV～数mV程度の電位（表面筋電位；Surface Electromyography, 以下ではEMGと略す）を使って操作可能な義手の総称である．

筋電義手は，従来の義手と比べて自然に操作が可能であるという特徴があるが，EMGの個人差のために，障害者が自由に扱えるようになるには，約１ヶ月間のリハビリトレーニングが必要であった．つまり，障害者が，義手に対して適応する必要があった．

これとは逆に，義手を障害者に対して適応させようとする研究が行われている．これらの研究では，ニューラルネットワークによるEMGの特徴ベ

クトルのパターン認識により義手の動作を決定する方式が用いられている．つまり，ニューラルネットワークを，障害者，個人個人のEMGの特徴に対して学習させることで，それぞれのEMGの特徴ベクトルに最適なパターン認識を行うネットワークを実現可能し，その認識結果により義手の動作を決定する．

しかしながら，EMGは，個人個人で異なっているだけでなく，切断後の筋肉の縮退や，センサーの位置などによっても変化し，そのたび毎にニューラルネットワークを学習し直さなければならないので，高速に学習する必要がある．一般的に，ニューラルネットワークは計算量が多いため，高速に実行，学習させるには，ハイスペックのCPUやDSP，あるいは，ニューロチップが必要であるが，これらを用いたシステムは大きいため，義手の内部に実装することが困難である．そのため，現在までに，ニューラルネットワークを用いた筋電義手は実現されていない．

そこで本研究では，EMGの特徴ベクトルのパターン認識に，EHWチップを用いる．EHWチップは，コンパクトに実装可能であるため義手の内部に実装しやすく，さらに，回路構成を高速に適応させることができるので，短時間で，障害者，個人個人のEMGの特徴ベクトルのパターン認識を行う回路を実現可能である．

本論文では，EMGの特徴ベクトルのパターン認識回路を，次に示す二つの方式で合成した．すなわち，回路合成に必要なトレーニングパターンの作成と回路合成が完全に別れている方式（オフライン方式）と，回路を合成しながらトレーニングパターンを徐々に追加する方式（オンライン方式）である．これらの回路合成の結果，EHWチップによりEMGの特徴ベクトルのパターン認識をおこなうことで，筋電義手の動作を決定可能であることが実証できた．

目次

図一覧

表一覧

第 1 章

序論

近年のEDA（Electronic Design Automation）技術の進歩により，複雑な回路を簡単に合成する事が可能となってきた．例えば，シノプシス社[1]などの論理合成ツールを使う事で，Verilog[2]等のHDL（Hardware Description Language）記述から回路を合成する事ができる．ただし，このような回路合成が可能なのは，入出力仕様を合成時点で完全に決定できる場合に限られている．

しかしながら，入出力仕様が完全に決定できない場合にも，回路を合成する必要性がパターン認識等の応用において指摘され始めている．例えば，手書き文字の認識を高速に行う回路[3][4]の場合，手書き文字パターンにはノイズがのりやすいために，入出力仕様を正確に決めることができない．さらに，入力パターンの数が多すぎるため，全入力パターンに対する正しい出力を決定できない．このように入力パターンにノイズが入ったり，あるいは一部の入力パターンに対する正しい出力しか決定できない場合には，従来のEDAツールを適用することはできない．

このような入出力仕様を決定できない場合の回路合成方式として，遺伝的アルゴリズム（Genetic Algorithm,GA，以下GAと略す）[5]に代表される進化的探索手法を用いる方式が提案されている[6]．この方式は，GAなどで回路を合成するので，進化型ハードウェア（Evolvable Hardware；以下EHWと略す）と呼ばれている．

EHW に関する研究は，回路構成単位の違いにより，三つのグループに分類可能である（表 1.1）．すなわち，回路構成単位をゲート (e.g. AND, OR) のレベルに置く場合 (ゲートレベル EHW) と，ハードウェア関数 (e.g. 乗算器) のレベルに置く場合 (関数レベル) と，アナログ素子 (e.g. 抵抗，コンデンサ) のレベルに置く場合 (アナログレベル) である．

表 1.1: EHW の研究例

ゲートレベル EHW	関数レベル EHW	アナログレベル
マルチプレクサ [6]	適応等化器 [7][8][9]	周波数フィルタ [10]
カウンター [11]	ATM [12][13]	周波数認識 [14]
有限状態オートマトン [11]	データ圧縮 [15][16]	
V 溝トレーサ [17][18][19]		
自律走行ロボット [20][21][22][23][24]		
パターン認識システム [3][4]		
筋電操作型義手 [25][26]		
電子印刷機 [27]		

ATM : Asynchronous Transfer Mode

その中でもゲートレベル EHW は，EHW の提案初期段階から研究されており，現在でも，最も活発に研究されている．このゲートレベル EHW では，PLA (Programmable Logic Array) に代表される，回路を AND ゲート，OR ゲート単位で変更可能なハードウェアを用い，PLA の回路を指定するビット列を GA の染色体とみなして進化させることで，回路を合成，再構成する．

本研究もこのゲートレベル EHW の研究であるので，本論文では，ゲートレベル EHW のことを，単に EHW と呼ぶことにする．

従来の EHW の研究では，まず，基本的な組合せ回路や順序回路の合成シミュレーションによる基礎研究 [6][11][17][18] が行われ，GA により再構成可能ハードウェアの回路を合成，再構成可能であることが示されている．さらに，これらの研究では，回路によっては，GA による合成に時間がかかることが指摘されている．手書き文字パターンの認識回路 [3][4] への応用では，ニューラルネットワークの特徴の一つである汎化能力を，組合せ回路によって実現可能であることが実証されている．自律走行ロボットの制御回路 [20][21][22][23][24] への応用では，センサーが故障した場合に，残りのセンサーで希望の動作を実行するように制御回路を動的に書き換えることが可能であることを実証している．電子印刷機 [27] への応用では，印刷するデータを一時的に保存するハードディスクの容量を小さくするために，データに応じて最適な圧縮を行う回路を合成，再構成可能であることを示している．

これらの研究では，システムの大きさと，回路合成の遅さが問題であった [18][25][19]．つまり，従来の EHW の研究では，GA のプログラムを実行する PC (Personal Computer) や WS (Work Station) が必要であるためにシステムが大きくなり，さらに，ソフトウェアによる GA の各操作の実行と評価値計算の遅さのために，回路合成に時間がかかっていた．そのため，小さく実装する必要のある場合や，高速に回路を合成，再構成する必要のある場合には，EHW を応用することが困難であった．

本研究では，これらの問題を解決するために，PC や WS を用いず，かつ，高速に回路構成を適応可能な LSI （EHW チップ）を設計する [25]．つまり，図 1.1 のように，従来 PC や WS で実行していた GA の各処理を実行するハードウェアと，PLA などの再構成可能ハードウェアを一つの LSI に実装する．そのため，この EHW チップは自律的に回路構成を動作環境に適応でき，システムを小さく作ることができる．さらに，GA の各操作と評価値計算をハードウェアで高速に実行するので，適応時間を短くすることができる．

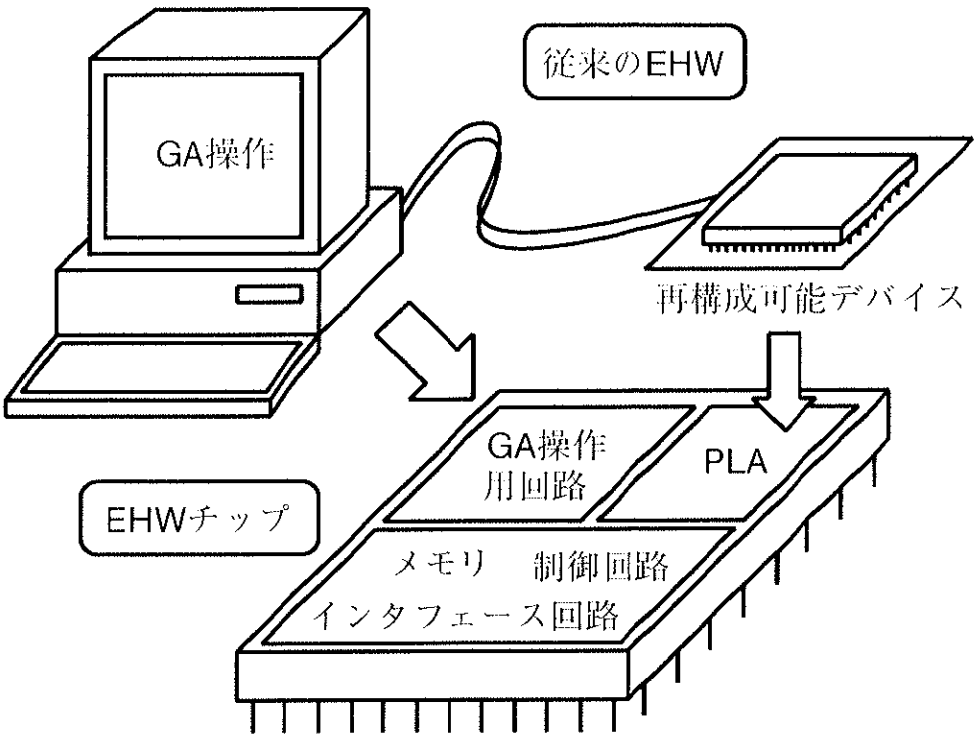

図 1.1: EHW チップの概念図

このEHWチップの設計にあたり，本研究では，まず，GA操作をハードウェアに実装するときの問題点を考察し，最もハードウェア向きのGA操作の組合せを提案した[25]．さらに，提案したGA操作を高速に実行する回路をFPGAに実装し，その基本動作を確認した[28]．

そして，1997年～1998年にかけてEHWチップversion 1（以下では，EHWv 1と略す）を設計し[25]，現在，EHWチップversion 2（以下では，EHWv 2と略す）を設計中である[26]．EHWv 2は，１９９９年2月完成予定である．

本研究では，EHWチップの応用の一つとして，筋電義手と呼ばれる義手の制御回路の一部に応用可能であることを示した．筋電義手とは，筋肉を動かしたときに体表面で観測される数μV～数mV程度の電位（表面筋電位；Surface Electromyography, 以下ではEMGと略す）[29]を使って操作可能な義手の総称である．

筋電義手は，従来の義手と比べて自然に操作が可能であるという特徴があるが，EMGの個人差のために，障害者が自由に扱えるようになるには，約１ヶ月間のリハビリトレーニングが必要であった[30][31]．つまり，障害者が，義手に対して適応する必要があった．

これとは逆に，義手を障害者に対して適応させようとする研究が行われている[30][32][31][33][34][35][36]．これらの研究では，ニューラルネットワークによるEMGの特徴ベクトルのパターン認識により義手の動作を決定している．つまり，ニューラルネットワークを，障害者，個人個人のEMGの特徴に対して学習させることで，それぞれのEMGの特徴ベクトルに最適なパターン認識を行うネットワークを実現し，その認識結果により義手の動作を決定する．

しかしながら，EMGは，個人個人で異なっている[29][34]だけでなく，切断後の筋肉の縮退や，センサーの位置などによっても変化し，そのたび毎にニューラルネットワークを学習し直さなければならないので

[37][38][39]，高速に学習する必要がある．一般的に，ニューラルネットワークは計算量が多いため，高速に実行，学習させるには，ハイスペックなCPUやDSP，あるいは，ニューロチップが必要であるが，これらを用いたシステムは大きいため，義手の内部に実装することが困難である．そのため，現在までに，ニューラルネットワークを用いた筋電義手は実現されていない．

そこで本研究では，EMGの特徴ベクトルのパターン認識に，EHWチップを用いる．EHWチップは，コンパクトに実装可能であるため義手の内部に実装しやすく，さらに，回路構成を高速に適応させることができるので，短時間で，障害者，個人個人のEMGの特徴ベクトルのパターン認識を行う回路を実現可能である．

本論文の構成は，2章で本研究の背景として，まず，EHWの仕組みと従来の研究の問題点を述べる．本研究では，この問題点の解決のために，EHWチップを設計，実装した．さらに，EHWチップの応用の一つである筋電義手について，なぜEHWチップが必要であるかを説明する．3章では，EHWチップに実装するGA操作用回路のついて説明し，4章でEHWv 1，5章でEHWv 2について述べる．EHWv 1の設計では，GAによる回路合成を高速化する方式として最小項学習法を提案し，実装した．EHWv 2の設計では，最小項学習法を改良したGRGA (Gene Replacement Genetic Algorithm) を提案することで，GAによる回路合成をより高速化した．6章では，EHWチップによるEMGの特徴ベクトルのパターン認識により，筋電義手の動作を決定可能であることを示す．7章で本論文のまとめと今後の研究課題を示す．

第 2 章

研究の背景

2.1 進化型ハードウェア (EHW) と EHW チップ

本章では，まず，EHW の仕組みを説明し，つぎに，従来の EHW の研究の問題点を述べる．本研究では，この問題点を解決するために，EHW チップを設計した．

2.1.1 進化型ハードウェア (EHW)

従来の回路合成手法は，入出力仕様が決定できない場合に対しては有効な解決策を持たない．例えば，実環境の問題では入力にノイズがはいることがあり，この場合は，真理値表に矛盾が生じるため，従来手法では取り扱えない．また，入力パターンが多すぎるために一部の入力パターンに対する出力しか設定できない場合は，未知の入力に対する出力を全てドント・ケア（0 でも 1 でも良い）とする不完全記述関数 [10] として合成できるものの，未知の入力に対する出力の正しさは保証されない．これに対して EHW では，未知の入力に対しても正しい出力を生成する可能性の高い回路を合成できることが，実証されている [3]．

EHW の基本アイデアは，GA （Genetic Algorithm）に代表される進化的探索手法と，PLA に代表される，回路構成をビット列で指定可能なハードウェア (再構成可能ハードウェア) の組合せである．

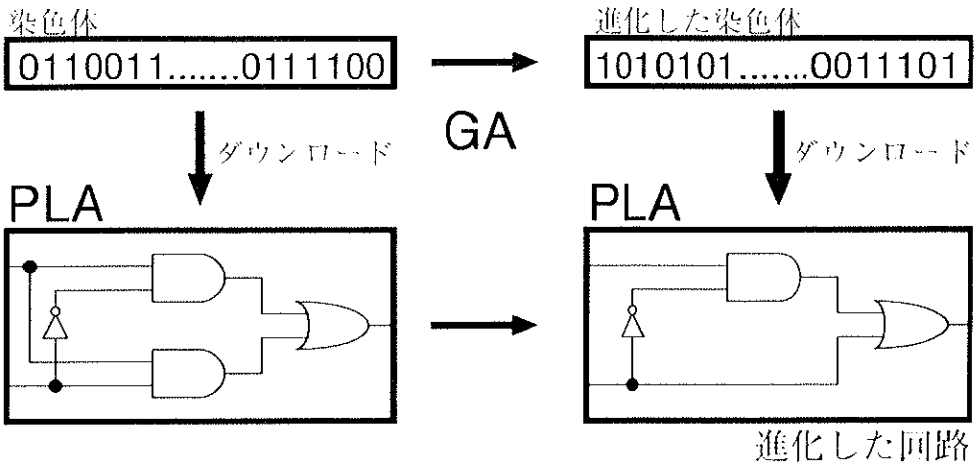

図 2.1: EHW の基本アイデア

GA は，探索対象のパラメータなどを二進ビット列に符号化し，そのビット列に対して交叉や突然変異と呼ばれる操作を施して進化させることによって，探索，最適化する．つまり，図 2.1 に示すように，再構成可能ハードウェアの回路を指定するビット列を GA の染色体と見なして進化させることによって，その動作環境に最適な回路を合成，再構成することができる．

以下では，まず，PLA と GA について説明し，次に，GA により PLA の回路を合成する方法を述べる．

2.1.1.1 Programmable Logic Array (PLA)

一般に PLA は，任意の論理回路を図 2.2 のように AND アレイと OR アレイの二段構成で実現する．すなわち，初段の AND アレイで入力信号の論理積を生成し，次に OR アレイにおいてそれらの論理和を出力する．

図 2.2 に具体例をあげる AND アレイでは，黒丸と白丸のスイッチの ON（黒丸），OFF（白丸）を指定することで信号線（$P0 \sim P3$）に接続する入力信号（$X0, \overline{X0} \sim X3, \overline{X3}$）を決定しそれらの論理積（積項）を

出力する．例えば図 2.2 のスイッチの設定では，信号線 $P0$ には，$X0$ と $X3$ が接続しているので，それらの論理積である積項 $X0X3$ が出力される．以下では，これらの信号線 （$P0$ ～ $P3$）のことを積項線と呼ぶことにする．

各積項線は，接続する入力信号に応じていくつかの入力パターンに対する出力信号が "1" となる．例えば積項線 $P0$ は，入力信号 $X0$ と $X3$ の否定が接続しているので，$X0$ が "1"，$X3$ が "0" である全ての入力パターン ("1000"、"1010"、"1100"、"1110") に対して "1" を出力する．

この $P0$ の出力は OR アレイへ行き，この場合，Y1 への接続がスイッチにより設定されている．同様に $P2$ の出力も Y1 に接続しているので，信号線 Y1 はこれら 2 つの論理和である $X0X3+X0X1X2X3$ を出力する．つまり信号線 Y1 は，入力パターン "1000"、"1010"、"1100"、"1110" と，"1011" に対してのみ "1" を出力し，残りの入力パターンに対しては "0" を出力する．

PLA の回路合成では，回路の動作仕様によって "1" を出力する入力パターンが決定され，それらの入力パターンに対して "1" を出力するために必要な積項の全てを PLA に合成しなければならない．各積項は，AND アレイの積項線と OR アレイのスイッチの設定を指定することで合成することができる．AND アレイの積項線と OR アレイのスイッチの設定はビット列とみることができ，このビット列のことをアーキテクチャビットと呼ぶ．例えば図 2.2 の PLA には，40 個のスイッチがあり，スイッチのオン，オフを 1，0 と見なすことで，40 ビットのアーキテクチャビットとなる．つまり，アーキテクチャビットが異なればスイッチの設定も異なるので，PLA には別のハードウェア回路が実現される．

そこで，EHW では，図 2.1 のようにアーキテクチャビットを GA の染色体と見なし，GA でアーキテクチャビット，すなわち回路構成を進化させることによって，目的の性能を満たすために必要な積項の組合せを探索

し，回路を合成する．

2.1.1.2 Genetic Algorithm (GA)

GA は，1960 年代後半から 1970 年代初めにかけて J.H.Holland らが提案した探索，最適化手法である．

その基本的な動作の仕組みは，生物の進化の過程に発想を得ており，生物の個体に見立てた複数の探索点を，並列に探索する手法である．つまり，複数の探索点をそれぞれ，染色体と呼ぶ複数のビット列に符号化し，それらに対して交叉や突然変異と呼ぶ操作を施して進化させる．こうすることによって，探索，最適化を行う．

ここでは，まず，GA による探索，最適化の手順を述べた後，GA の各操作について説明する．

GA による探索，最適化の手順 GA では，次に示す 1 ～ 5 の手順にしたがって探索，最適化を行う．

1. 初期染色体生成：初期染色体として，ランダムなビット列をつくる．
2. 評価：各染色体が，どれだけ探索目的にあっているか，あるいは，最適解に近いかによって，評価値を計算する．
3. 選択：評価値の大きい染色体を残し，小さい染色体を削除する．
4. 交叉，突然変異：選択した染色体に対して，交叉，突然変異と呼ぶ操作を施し，新しい染色体 (探索点) を作る．
5. 2 ～ 4 を終了条件を満たすまで繰り返す．

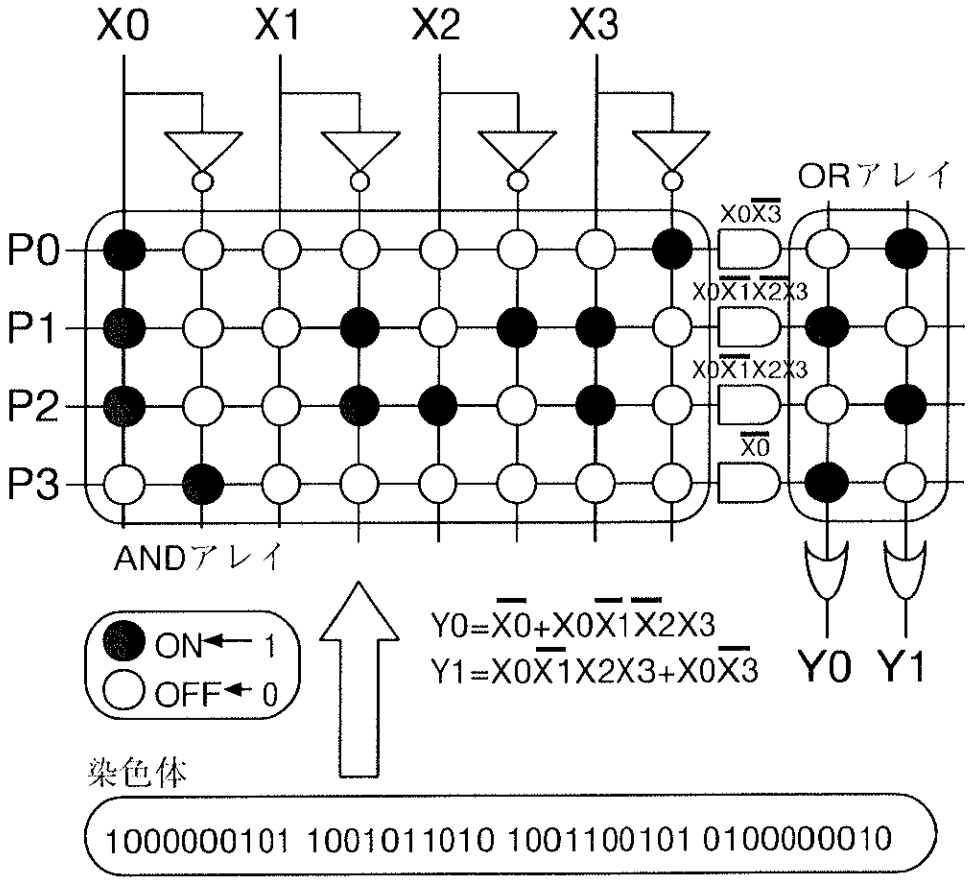

図 2.2: PLA の構造と回路構成の指定方法

選択　生物の進化の過程では，生存環境により適した生物が生き残り，適さない個体は絶滅することで，よりその生存環境に適したものへと進化してきた．

GA による探索でも同様に，探索目的に適した染色体が高い評価値を得ることができる．そこで，評価値が大きい染色体を優先的に選択し，次に述べる交叉処理により，選択した (評価値の大きい) 染色体間の情報を交換することで，より評価値の大きい染色体を作る．こうすることによって，より探索目的に適する染色体を探索する．

この選択処理には，世代モデルと定常状態モデルがある [11][12].

世代モデルでは，選択処理を行う前に，全ての染色体の評価値を計算し，それらの評価値に応じて，元の染色体と同じ数の染色体を選択する．染色体の選択方法には，評価値に比例した確率で選択する方式などがあり，評価値の大きい染色体は複数回選択される．

定常状態モデルは，いくつかの染色体の評価値を計算し，それらの間だけで選択処理を施す操作である．例えば，二つの染色体の評価値を計算し，評価値の大きい方の染色体のコピーを，評価値が小さい方の染色体と置き換える．

交叉　交叉処理は，ある決められた確率 (交叉率) で，二つの染色体の情報を交換する操作である．

GA による探索では，交叉処理により，評価値の大きい染色体同士で情報を交換することによって，より評価値の大きい染色体を作る．つまり，評価値が大きい染色体には，評価値が大きくなる原因となるビット列 (例：図 2.3 の網かけ部分) が存在しており，それらを交叉処理によって交換することによって，より評価値の大きい染色体を作ることができる (図 2.3).

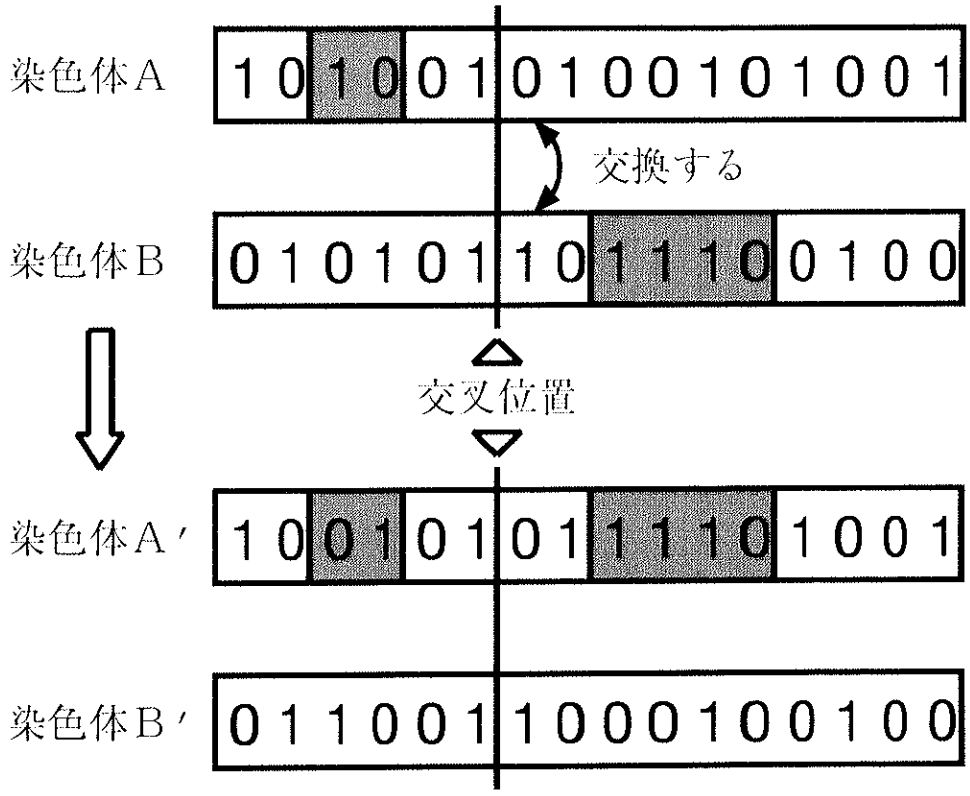
染色体A
1 0 1 0 0 1 0 1 0 0 1 0 1 0 0 1
交換する
染色体B
0 1 0 1 0 1 1 0 1 1 1 0 0 1 0 0
交叉位置
染色体A′
1 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1
染色体B′
0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 0

図 2.3: 交叉

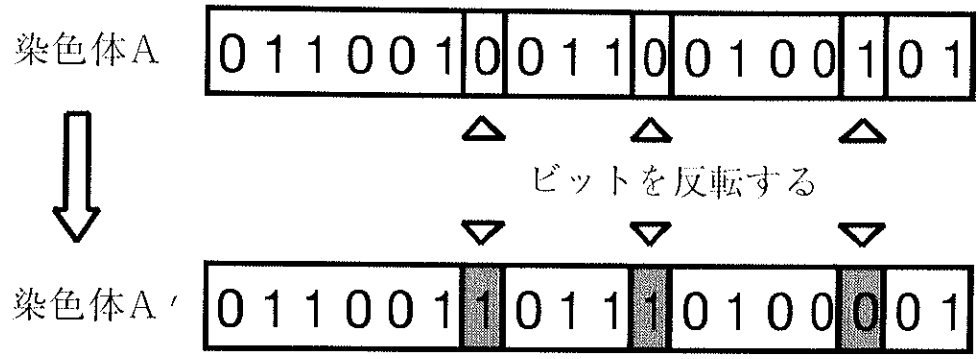

図 2.4: 突然変異

突然変異 突然変異処理は，図 2.4 のように，ある決められた確率で染色体の情報を変化させる操作である．ただし，交叉では染色体毎に交叉処理を施すかどうか決定していたが，突然変異では各ビット毎に突然変異処理を施すかどうか決定する．

突然変異は，交叉処理では作ることのできない新しい染色体を作る効果がある．例えば図 2.5 に示す 5 個の染色体の場合，最も左のビットで 1 が立っている染色体はなく，交叉処理だけでは，このビットに 1 を立てることはできない．このとき，最も左のビットに突然変異を施すことで，このビットに 1 を立てることができる．

2.1.1.3 GA による PLA の回路の合成方法

GA により PLA の回路を合成，最構成するには，PLA の回路がどの程度，探索目的にあっているかを評価しなければならない．

回路の評価方法としては，回路を組み込んだシステムの動作を評価したり [27]，表 2.1 に示すようなトレーニングパターンを用いる方式がある [6]．

EHW に関する従来の研究では，後者のトレーニングパターンを用いる方式により回路を評価することが多く，本研究でもこの方式を採用する．つまり，あらかじめ用意したトレーニングパターンに対する PLA の回路

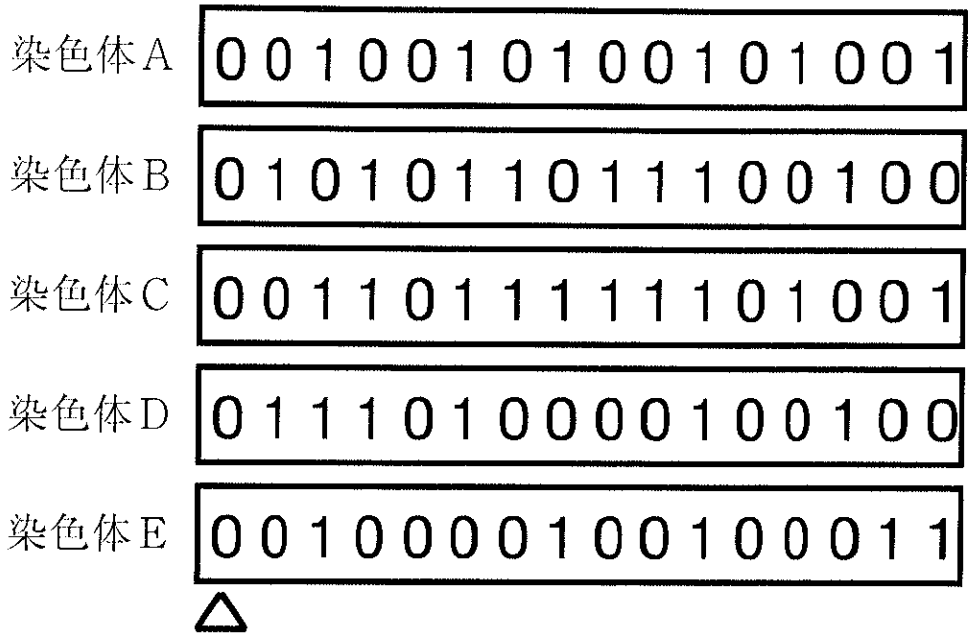

図 2.5: 突然変異の効果；交叉だけではつくることのできない染色体をつくる

の出力の正解率を，PLA の回路の評価値として用いる．ここで，一つのトレーニングパターンは，表 2.1 に示すように，入力パターンとそれらに対する正しい出力パターンからなり，ある決められた数のトレーニングパターンに対する PLA の回路の出力の正解率を評価値として用いる．例えば表 1 の場合，トレーニングパターンは 4 ビットの入力パターンと，それに対する正しい出力 (2 ビット) からなる．そして，10 個のトレーニングパターンに対するある回路 A の出力は，10 個のうち 8 個が正解なので，評価値は 0.8 である．この評価値を用いて染色体を進化させることによって，回路を動作環境に適応させることができる．

ただし，一般的には，表 2.1 のようなトレーニングパターン，つまり，いくつかの入力パターンとそれらに対する出力パターンが決定できる場合は

Espresso[13] などの論理圧縮手法によって，高速に PLA の回路を合成可能である．

しかしながら，EHW では実環境における問題を扱うため，ノイズなどのためにトレーニングパターンに矛盾したものが存在したり [19]，トレーニングパターンが動的に変化することがある [26]．そのため，従来の論理圧縮手法は適用できない．

また，入力のビット幅が小さく，かつ，動作速度が遅い場合は，全入力パターンに対する出力を RAM に持たせ，マイクロプロセッサなどによる表引きでも実現可能である．しかしながら，この方式では，入力のビット幅が大きい場合には，全入力パターンに対する正しい出力を短時間で決定することが困難である．それに対して EHW では，入力のビット幅が大きい問題にも応用可能である．例えば [25] では 21 ビット入力，[1] では 61 ビット入力の回路を扱っている．

表 2.1: トレーニングパターンと評価値の例

トレーニングパターン		
入力パターン $X0X1X2X3$	正しい出力パターン	回路 A の出力 $Y0Y1$
0000	00	01
0001	01	01
0010	01	01
0011	01	01
0100	01	01
0101	01	01
1001	01	01
1010	00	10
1100	10	10
1110	10	10
	評価値（＝正解率）	0.8

2.1.2 EHW の問題点と EHW チップの提案

従来の EHW の研究では，システムの大きさと，回路合成の遅さが問題であった [18][25]．まず，システムの大きさの原因は，GA のプログラムを実行する PC や WS が必要なことである．次に，適応の遅さは，GA の各操作の実行の遅さと，評価値計算の遅さが原因である．

そこで，これらの問題を解決するために，PC や WS を用いず，かつ，高速に回路構成を適応可能な LSI （EHW チップ）を設計する．つまり，図 1.1 のように，従来 PC や WS で実行していた GA の各処理を実行する専用のハードウェアと，PLA などの再構成可能ハードウェアを一つの LSI に実装する [25]．そのため，この LSI は自律的に回路構成を動作環境に適応でき，システムを小さく作ることができる．さらに，GA の各操作と評価値計算をハードウェアで高速に実行するので，適応時間を短くすることができる．

2.1.3 EHW チップの設計方法

EHW チップの設計にあたっては，Verilog-HDL (Verilog Hardware Description Language) [2] と呼ばれる C 言語に似た高級言語により回路を記述した．この方式は，従来の回路図により入力する方式と比べて，設計者にとって直感的にわかりやすく，シミュレーション環境も充実しているため，設計初期段階での回路の問題の発見が容易である．

Verilog-HDL で記述した回路の例を以下に示す．以下の例は BCD カウンタで，1 ビットのレジスタで 0 から 9 までを数えることができる．

この回路は，ENABLE 信号が 1 のときに，クロック （CLK） に同期して，1 ビットのレジスタ (Q) の値をインクリメントし，レジスタ Q の値が 9 (1001) の時は，CARRY を 1 にセットし，レジスタ Q の値を 0 に戻す．

```
//
module BCD_COUNTER (CLK, RESET, ENABLE, LOAD, DATA, Q, CARRY);
  input CLK, RESET, ENABLE, LOAD;
  input [3:0] DATA;
  output [3:0] Q;
  output CARRY;
  reg [3:0] Q;
  always @ (posedge CLK or negedge RESET) begin
    if (!RESET)
      Q <= 0;
    else if (LOAD) begin
      if (DATA > 9) begin
        Q <= 0;
      end else begin
        Q <= DATA;
      end
    end else if (ENABLE==1) begin
      if (Q==9) begin
        Q <= 0;
      end else begin
        Q <= Q+1;
      end
    end
  end
  assign CARRY = (Q==9);
endmodule
//
```

Verilog-HDL で記述した回路は，シノプシス社の回路合成ツール（design compiler[1]）でコンパイルすることで，バッファや論理ゲートなどに変換することができる．例えば，上の例に示した BCD カウンタを合成すると図 2.6 の回路になる．さらに，この論理合成ツールは，回路の遅延制約などを与えることができ，その制約に応じて回路を最適化する．この操作では，実装するデバイスによってライブラリが提供されており，EHW チップの場合は，NEC, 0.35μm CMOS セルベース IC，CBC9 ファミリ [14] に実装したので，このライブラリを用いた．

2.2 EHW チップの応用

本研究では，EHW チップの応用の一つとして，筋電義手と呼ばれる義手の制御回路の一部に応用可能であることを示す．

ここでは，一般的な義手の説明に続いて，現在市販されている筋電義手の問題点を述べ，次に，なぜ筋電義手の制御回路の一部に EHW チップが必要であるかを説明する．

2.2.1 義手

義手とは，前腕，あるいは上腕で手を失った，あるいは，先天的に障害がある人が手の代わりにつけるもので，その機能から，大きく分けて 4 つに分類可能である．つまり，「装飾用義手」，「作業用義手」，「能動義手」，そして「筋電義手」の 4 つである [15].

2.2.1.1 装飾用義手

装飾用の義手は，その名の通り，手があるかのように見せるための義手である．

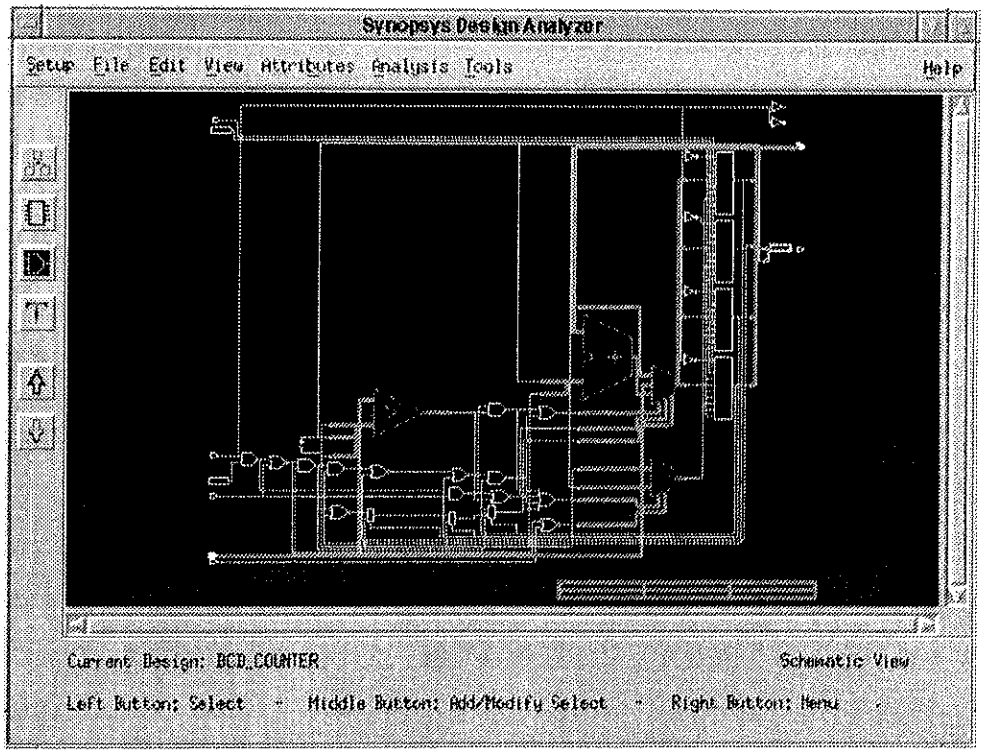

図 2.6: design_compiler での合成結果（BCD カウンタ）

装飾用義手の歴史は古く，エジプトのピラミッドから，義手をつけたミイラが発見されている [15].

その後，外見だけでなく，簡単な機能を持つ義手（機能的装飾義手）が開発されている．例えば，手先の部分を他動的に動かせるようになっており，その部分に，剣や盾を持たせることができるものがある [15].

2.2.1.2 作業用義手

作業用義手は，装飾用義手とは全く逆の立場で，見た目はあまり考えず，ある特定の作業を効率良く行うために開発された義手である．例えば，手先に草刈り鎌が付いていたり，野球のグローブを固定できるようになっているものがある [15].

2.2.1.3 能動義手

能動義手は，例えば，背中や肩を動かすことで，手先や肘を動かすことのできる義手である．つまり，手先のフックから背中にワイヤーが延びており，背中を丸めると，このワイヤーが引っ張られ，その張力によって手先のフックが開くようになっている．そして，手先のフックに輪ゴムが付いており，背中をもとに戻すと，輪ゴムの力でフックが閉じるようになっている．

しかしながら，能動義手の場合，義手を動かすために背中や肩を動かすという，不自然な動作が必要である．それに対して，以下で説明する筋電義手は，切断部に残った筋肉の動きで操作できるため，自然に操作できる．

2.2.2 筋電義手

筋電義手とは，筋肉を動かしたときに体表面で観測される数 μV ～数 mV 程度の電位（表面筋電位；Surface Electromyography, 以下では EMG と

略す）を使って操作可能な義手の総称である．

以下では，まず，EMG の特徴を述べ，次に，従来の筋電義手とその問題点を述べる．

2.2.2.1 EMG (Electromyography)

筋肉は，たくさんの筋繊維の束からなり，それらの収縮によって筋肉全体が収縮し，各筋肉の収縮の組合せによって，様々な動作を行う [29]．この筋肉の収縮は，筋肉の動作指令信号を伝える運動ニューロンと，その運動ニューロンによって動作が支配される筋繊維の束 (運動単位) を単位としておこる．つまり，ある運動単位に動作指令信号が届くと，その運動単位の筋繊維が収縮する．ただし，全運動単位が同時に収縮するのではなく，いくつかの運動単位が交替で収縮する．

EMG は，各運動単位の収縮によって起こるが，体表面まで伝わる間に，他の筋繊維や脂肪などの影響で，減衰したり周波数特性が変化する．つまり，観測される EMG の特徴は，収縮した運動単位から体表面までの距離や，筋肉や脂肪のつきかたによって決まる．この性質を利用することで，観測された EMG から，どの筋肉が動いたかを推定することができ，これを義手の動作決定に応用したものが，筋電義手である [46]．

2.2.2.2 筋電義手の問題点と EHW チップの必要性

現在，日本で入手可能な前腕筋電義手には，今仙技術研究所のワイムハンドや，オットボック社の筋電義手がある．これらの筋電義手は，ともに 1 自由度で，摘む動作と，離す動作が可能である．つまり，前腕の屈筋と伸筋の近くの体表面にセンサーを付けて EMG を観測し．屈筋側の EMG がある閾値を越えると摘む動作をし，伸筋側の EMG が閾値を越えると離す動作をする [47]．

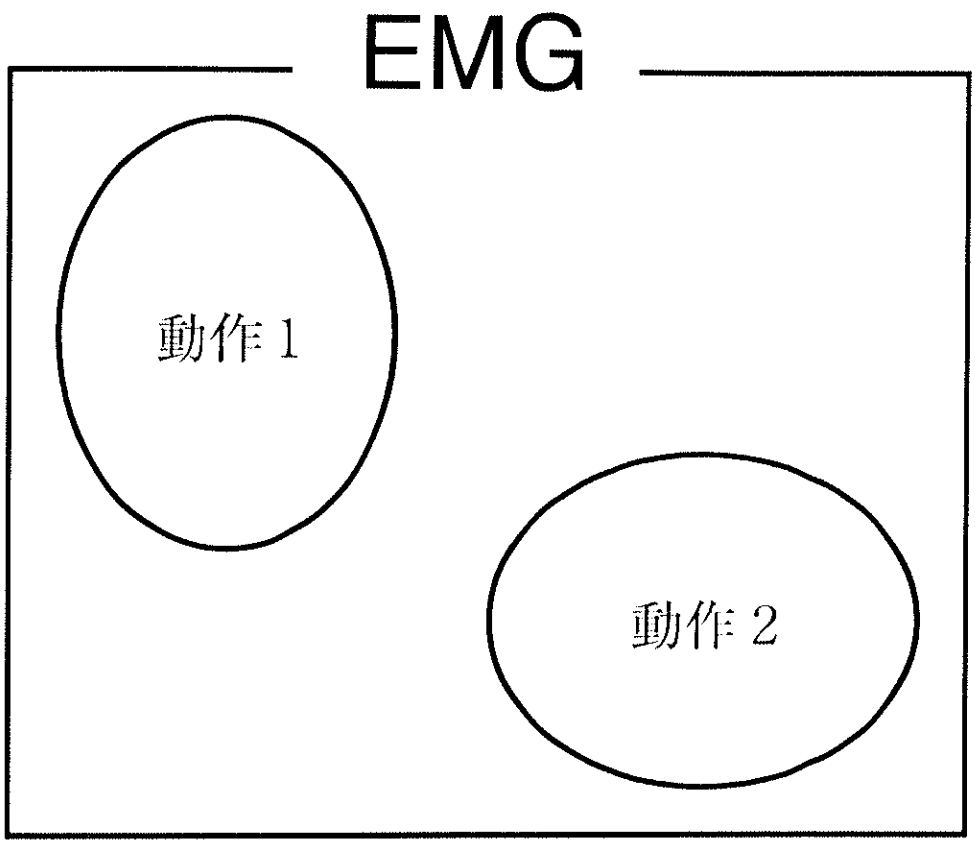

図 2.7: EMG から 2 動作の識別

筋電義手の問題点としては，1992 年にアメリカの NIH (National Institutes of Health) と NCMRR (National Center for Medical Rehabilitation Research) で調査が行われ，その結果が，1996 年に報告されている [48]．その報告の中では，自由度の少なさを問題点としてあげている人が最も多かった．1 自由度の義手の場合，図 2.7 に示すように，EMG 全体の集合から二つのグループを識別するだけでよい．それに対して多自由度の筋電義手の場合は，図 2.8 のように，観測された EMG を，義手の動作数と同じ数のグループに分類する必要がある（図 2.8 は 3 自由度，6 動作の例）．

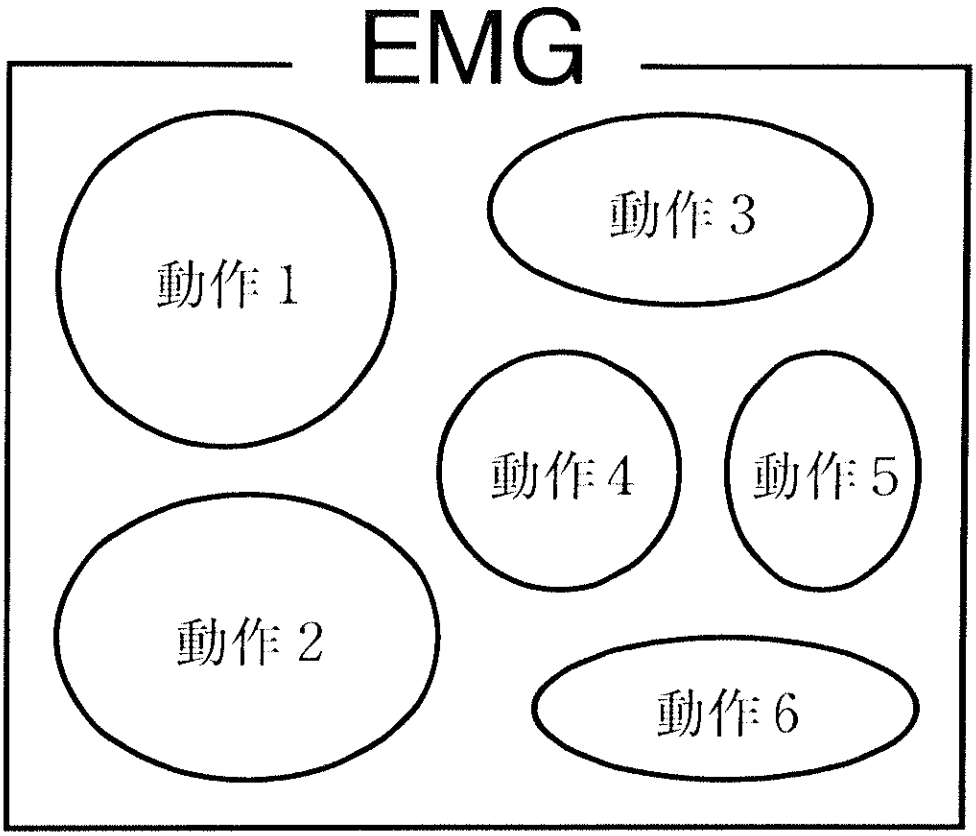

図 2.8: EMG から複数動作の識別

従来の筋電義手では，EMG と対応する義手の動作の関係（図2.8 の囲い）が製造段階で決まっており，個人の EMG の特徴にあわせて変更することはできない．そのため．従来の筋電義手では，障害者が，行おうとする動作に対応する EMG を発生させるために，約 1 ヶ月間のリハビリ訓練を受ける必要があった [30][31]．つまり，従来の筋電義手では，障害者が，義手に対して適応する必要があった．

この問題を解決するために，従来の義手とは逆に，義手の方に適応能力を持たせようとする研究が行われている．これらの研究では，従来は，統計的な手法を用いる方式 [19][50][51][52][53][54] や AR （Autore-

gressive) モデルや ARMA (Autoregressive Moving Average) モデルを用いる方式 [55][56][57] により EMG から義手の動作を決定していたが，近年では，ニューラルネットワークによる EMG の特徴ベクトルのパターンマッチングにより義手の動作を決定する研究が行われている [58][37][30][31][33][31].

しかしながら，EMG は人によって異なっているだけでなく，切断後の筋肉の縮退や，観測位置の違いによっても変化するため，頻繁にネットワークを学習しなおさなければならない [37][38][39]．そのためには，ハイスペックな CPU や DSP，あるいは，ニューラルネットワーク専用の LSI を用いてネットワークの結合強度を高速に学習する必要がある．ところが，一般的に，これらを用いたシステムは大きいため，義手のように大きさと重さの制限[1]のある応用には適さず，現在までに，ニューラルネットワークを用いた筋電義手は開発されていない．

そこで本研究では，EMG の特徴ベクトルのパターンマッチングに EHW チップを応用する．EHW チップは，コンパクトに実装可能であるので，義手の内部に実装しやすく，かつ，高速に回路構成を適応させることができるので，短時間で，障害者の EMG の特徴に最適なパターン認識回路を合成可能である．

[1] 人の手と同じ大きさ，重さ＝約７００グラム

第 3 章

GA 操作用回路

本章では，GA 操作用の回路を設計し，FPGA (Field Programmable Gate Array) に実装することで，その基本動作を確認する．この設計にあたっては，GA 操作用の回路の設計上の問題点を考察し、最もハードウェア向きの GA 操作の組合せを提案する。

ここで設計した GA 操作用回路は，EHW チップへの応用に留まらず、自律的に学習可能なニューラルネットワーク LSI[59][9][60] の内部に実装することで，ネットワークの結合係数の学習だけでなく，ネットワークの構造の学習 [61][62][63][64] を自律的に行うことのできる LSI を実現可能である [28].

3.1 ハードウェア向きの GA 操作

従来の GA のハードウェア化の研究 (表 3.1) は，GA による探索を高速化するために，GA 操作の実行を高速化することを目的としていた．

しかしながら，GA 操作に要する時間よりも評価に要する時間がはるかに大きいことが多く [65]，そのような応用の場合は，GA 操作を高速化しても，全体の探索はそれほど速くならない．ただし，[66] や [67] では，ソフトウェアによる評価に時間がかかる問題において，評価用の回路も同

じ LSI[66] や FPGA[67] に実装することで，全体の探索時間を短縮している．

EHW チップは，高速，かつ，コンパクトな適応を目的としているので，ここでは，GA 操作用回路をよりコンパクトに実装する方式を提案する．以下では，まず，GA 操作をハードウェアに実装するときの問題点を考察し，次に，最もハードウェア向きの GA 操作の組合せを提案する．

3.1.1　GA のハードウェア化における問題点

GA の各処理をハードウェア化するときの問題点として，まず，各個体の染色体を保存するためのメモリ（染色体メモリ）の大きさについて述べ，次に，選択処理と交叉処理，それぞれについて考察する．

3.1.1.1　染色体メモリの大きさ

GA の探索は，一般に，個体数が大きいほど効率良く探索できるが，そのために個体数を大きくすると，染色体を保存するために大きなメモリが必要になる．つまり，より回路を小さくするためには，個体数は小さいほうがよい．しかしながら，そのために探索能力をおとすことはできないので，よりコンパクトに実装するには，小さい個体数でも効率良く探索可能な方式が必要である．

3.1.1.2　選択処理

GA では，評価値の大きい個体を選択し，それらの染色体に交叉と突然変異処理を施し，より評価値の大きい個体を生成することで探索を行う．個体を選択する方式としては，『世代モデル』と『定常状態モデル』[11] の，二つの方式がある．

表 3.1: GA のハードウェア化の研究例

	世代交代	乱数発生	交叉	突然変異	評価
[65][68][69][70]	世代 (ルーレット)	*Systolic*	一様	有	なし
[66]	格子 (パラレルGA)	RNG	*Order-Based* 交叉[71]	有	ビット スタビィリティ
[67]	トーナメント	混合合同法	一点	有	トポロジック グラフ分割 *IPD*
[72][73]	定常状態 (ルーレット)	*CA*	1点	有	関数最適化
[74]	世代 (ルーレット)	*RNG*	1点	有	TSP
[75]	世代 (ルーレット)	*CA*	1点	有	PID コントローラ
[76]	世代 (ルーレット)	*CA*	1点	有	関数最適化
[77]	定常状態 (単純トーナメント)	*CA*	1点	有	関数最適化
[78]	格子	*LFSR*	一様	無	次元セルラ オートマトン の遷移規則
[79]	定常状態 (最悪個体の置換)	*CA*	1点	有	論理関数の最小化

RNG: Pseudo Random number generator
Systolic: Systolic array.
CA: Cellular Automaton [80][81][82][83].
LFSR: Linear Feedback Shift Register.
IPD: Iterated Prisoner's Dilemma

世代モデルでは，全個体（N 個体）の評価値を計算した後，評価値に応じて（重複を許し）N 個の個体を選択する．そのため，選択した N 個体の染色体を一時的に保存するためのメモリが必要である．つまり，世代モデルでは，選択前の N 個体と選択後の N 個体，合計 $2N$ 個体の染色体を保存するためのメモリが必要である [65][79].

それに対して『定常状態モデル』では，一部の個体の評価値を計算し，評価値を計算した個体の間だけで選択処理を施す．例えば，4 個体ずつ選択処理を行う場合は，まず，4 つの個体の評価値を計算し，次に，それら 4 個体の中で評価値の大きい個体を選択する．そのため，選択した個体の染色体を小さなメモリやレジスタに保存できる．例えば上に示した例では，4 つの個体を保存するレジスタと，選択前の N 個体の染色体を保存するメモリがあればよい．つまり，定常状態モデルを用いると，N 個体の染色体を保存するメモリと小さなレジスタだけが必要であるので，より小さく実装可能である．

3.1.1.3　交叉

交叉処理は，二つの染色体の情報を組み換える処理で，任意の一点（一点交叉），あるいは複数点（多点交叉）で染色体を切断し，それらの前後で情報を組み換える．具体的には，染色体長を最大値とする乱数に応じて組み換える点（交叉点）を決定し，それらの前後で染色体を組み換える．そのため，交叉処理をハードウェアに実装するには，染色体長を最大値とする乱数を発生する回路が必要である．つまり，染色体長は問題に応じて任意の値をとるので，任意の最大値を持つ乱数を発生する回路が必要である．

しかしながら，ハードウェアに実装した乱数発生器はランダムなビット列しか作れないため，その最大値は 2^N (N は染色体長) に固定される．そのため，任意の最大値の乱数を作るには，正規化のための回路が必要であ

る．つまり，一点交叉や多点交叉では正規化用の回路が必要であるので，交叉処理用の回路を小さく設計するには，任意の最大値の乱数を必要としない交叉方法が適している．

3.1.2　ハードウェア向きの GA 操作

以上の考察から，ハードウェア向きの GA 操作は，次の三つの条件を満たす必要がある．

① 小さい個体数に対しても効率の良い探索手法

② 定常状態モデル

③ 任意の最大値の乱数を必要としない交叉方法

本論文では，上の三つの条件を満たす遺伝操作の組合せとして，ER (Elitist Recombination) [84][85][86] の探索能力を改善した HSP-ER (High Selection Pressure - ER) と UC (Uniform Crossover, 一様交叉) [87][88][89][90] の組合せを提案する [28].

3.1.2.1　ER, HSP-ER

ER は，GA の各処理を簡単化し，かつ，小さい個体数でも効率良く探索可能な方式として 1994 年に提案された [84][85][86]．ER は，定常状態モデルで，図 3.1 に示す手順のように，任意に選択した二つの個体（親 1，親 2）と，それらから作った二つの個体（子 1，子 2）の間だけで選択処理を施す．つまり，個体の交換が親と子の間だけで行われるので，小さい個体数でも多様性を維持できるために効率良く探索できるという特徴がある．

さらに，一般的な GA では，交叉処理を施すかどうかを決めるための適切な交叉率を試行錯誤的に決定しなければならないが，ER では，交叉率を 1.0 とすることができる [84]．一般的な GA では，子供の染色体が，常に

親の染色体と交換されるので，親の染色体中の高い評価値の原因となるビット列（図 2.3 の網かけ部分）が交叉によって破壊され，子供の評価値が親の評価値よりも小さい場合でも，親の染色体は，子供の染色体に置き換えられる．交叉率は，ある一定の確率で交叉処理を施さず，親の染色体をそのままコピーすることで，親の染色体が持つ高い評価値の原因となるビット列が保存されるように設定する．そのため，交叉率を適切に設定できない場合は，親の染色体が持つ高い評価値の原因となるビット列が破壊されやすくなり，効率良く探索できなくなる．これにたいして ER の場合，交叉処理を施した後の子供の染色体が，親の染色体よりも評価値が小さい場合は，親の染色体がそのまま保存される．そのため，交叉率を適切に設定する必要はなく，交叉率を 1.0 として，常に交叉処理を行うことができる．

しかしながら，ER は，親の二個体の選択に評価値を考慮しないため，評価値の大きい個体の染色体情報が個体集団中に広まりにくく，学習に時間がかかることがある．

そこで本論文では，より高速に学習させるために，HSP-ER を提案する．ER では任意に親の個体を選択していたが，HSP-ER では，それまでで最も評価値の大きい個体を一つの親として選択し，もう一つの親はランダムに選択する．こうすることによって，評価値の高い個体の染色体情報を個体集団中に広めることができ，高速に探索することができる．

HSP-ER は，ER と同様に，親と子の間だけで選択処理を施すので，少ない個体数でも多様性を維持でき，かつ，定常状態モデルであるので，先に述べた三つの制約のうち①② の制約を満たしている．

3.1.2.2　UC (Uniform Crossover)

次に UC は，染色体の各遺伝子座毎に 50 %の確率で遺伝子の情報を交換する方式である．

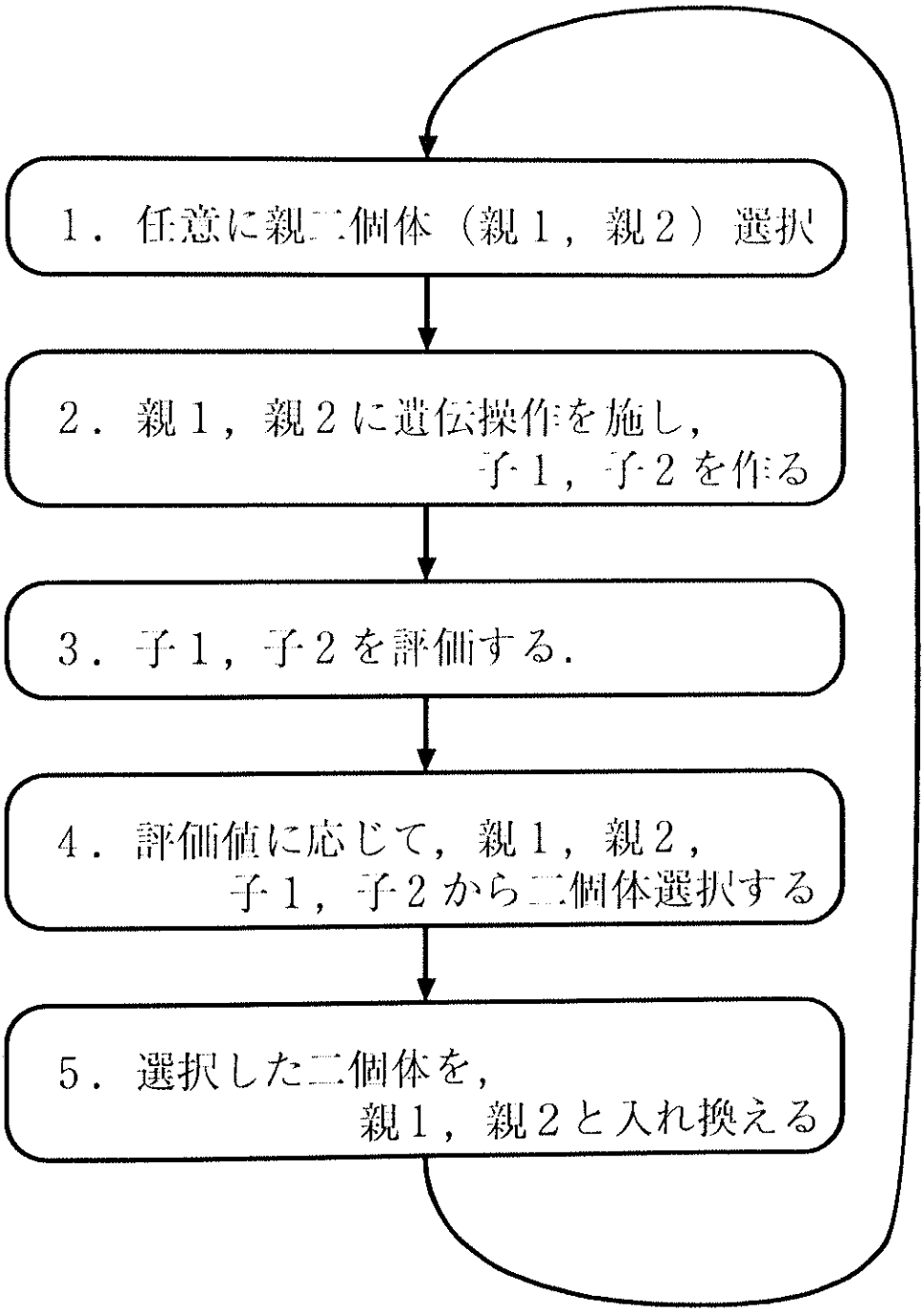

図 3.1: ER (Elitist Recombination)

この方式では，交叉位置を決定する必要がなく，図?に示すように，染色体と同じ長さのランダムなビット列を用意し，1がたっているビットで染色体の情報を交換すればいいため，交叉位置を決めるための乱数は必要なく，条件③を満たす．ただし，[65][68][69][70][78] の GA 用ハードウェアでも一様交叉を採用しているが，その理由は示されていない．

以上のように，HSP-ER と UC を組み合わせることで，ハードウェア向きの GA の 3 つの条件を満たすことができ，EHW チップに実装する GA 操作用回路には，この組合せを採用する．

3.1.2.3　HSP-ER+UC の探索能力の検証

ここでは，ER や HSP-ER の特徴の一つ『少ない個体数でも効率良く探索可能』である能力を検証する．この検証にあたっては，HSP-ER+UC，ER+UC と比較するために，フリーで入手可能な GA のソフトウェアパッケージで，様々な研究で用いられている GAucsd（一点交叉，ルーレット選択）[91] を用いることにする．各 GA の諸設定を表3.2 に示す．

表 3.2: GA の諸設定

	ER+UC,HSP-ER+UC	GAucsdL4
交叉率	1.0	0.8
突然変異率	0.07812	0.07812

探索能力の検証に用いる問題としては，個体数による影響だけを検証するために，GA による探索を妨げる『だまし』と呼ばれる要素のない単純な問題を用いる．具体的には，染色体中の 1 が立っているビットの数をそのまま評価値とする問題を用いることにする．この問題は，単純に 1 が立っている遺伝子を交叉処理で組換えることで容易に探索可能である．具体的

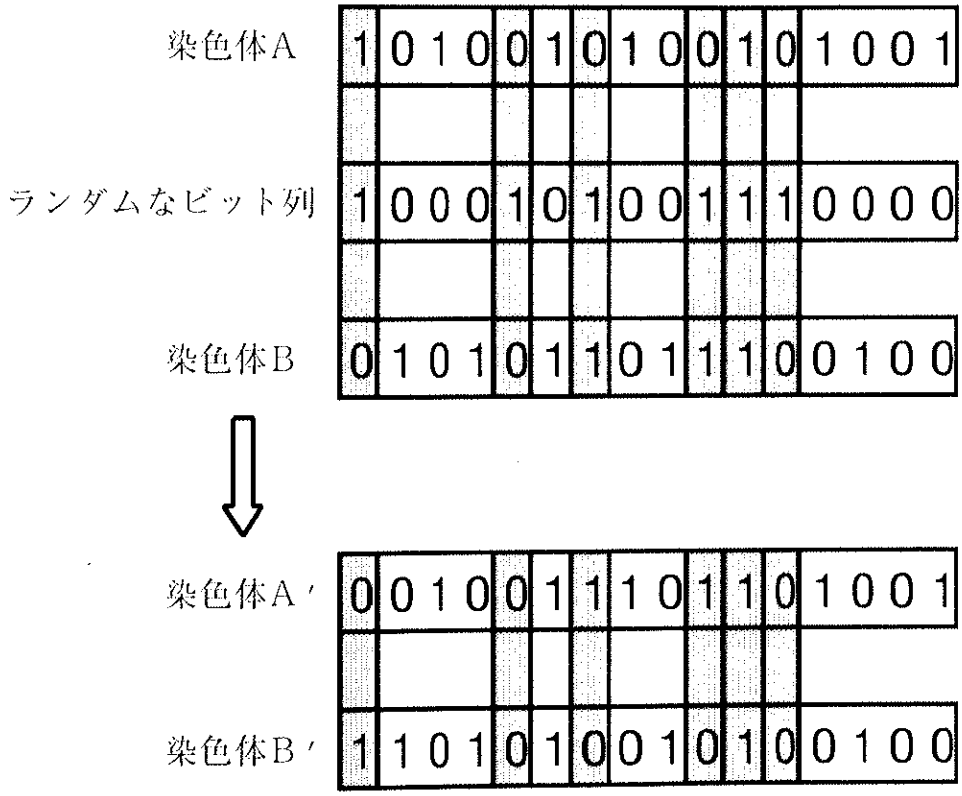

図 3.2: UC (Uniform Crossover)

には，染色体長を 1000 ビットとしたので，1000 ビット全てで 1 がたっているときに評価値が最大値 (1000) となる．

実験では，個体数を 32,64,128,256,1021 と変化させ，乱数の初期値を変えてそれぞれ 10 回探索し，評価回数 10000 回までの最高評価値の平均値を図 3.3 に示す．個体数による影響を調べるために，それらの分散を求めると，HSP-ER+UC の場合が 44.2，ER+UC が 13.1，GAucsd が 381.5 であった．この結果から，ER+UC や HSP-ER+UC が，GAucsd と比べて個体数の影響を受けにくく，小さい個体数でも探索能力が落ちないことがわかる．

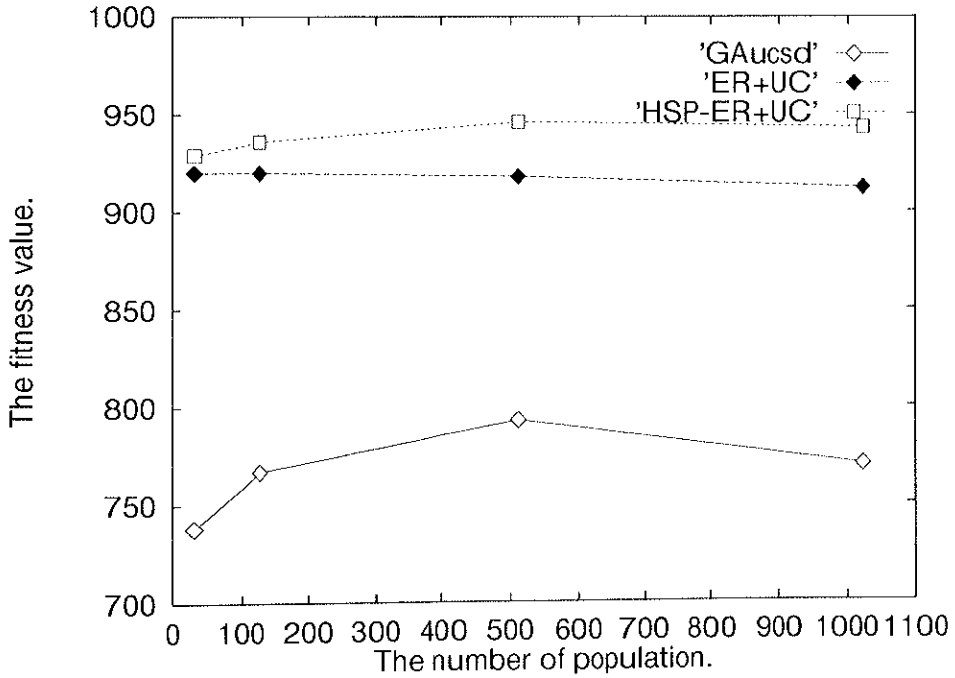

図 3.3: 個体数の影響

3.2　FPGA への実装による基本動作の確認

ここでは，HSP-ER+UC を高速に実行可能な回路を FPGA に実装し，その回路規模と動作速度を評価する．

実装には，図 3.1 の基板を用いた．この基板は，1 個の FPGA（Xilinx 1025，25000 ゲート相当）がのっており，最大 10 万ゲート相当の回路を実装可能である．ただし，本論文で提案した HSP-ER+UC を高速に実行する回路は，小さく実装することができるので，FPGA を 一つ用いるだけで済む．

3.2.1　全体の回路構成

FPGA に実装した回路は，図 3.5 に示したように，親の個体の染色体用のレジスタ（32 ビット） 二つ (親 1, 親 2) と，子供の個体の染色体用のレジスタ（32 ビット）二つ (子 1, 子 2)，突然変異率用のレジスタ 一つ，そして，遺伝操作用回路（Figure1）からなる．親の染色体用レジスタと突然変異率用レジスタは，PC の ISA バス経由で書き込むことができ，子供の染色体用のレジスタは，ISA バス経由で PC から読みだし可能である．そして，親の染色体用のレジスタに 32 ビット単位で染色体を書き込めば，それらに対して遺伝操作が施され，子供の染色体（32 ビット）ができる．

3.2.2　遺伝操作用回路

遺伝操作用回路は，図 3.6 に示すように，交叉用回路 (図 3.7)，突然変異用回路 (図 3.8)，突然変異用マスク生成回路 (図 3.9)，そして乱数発生器 (図 3.10) の 4 つからなる．

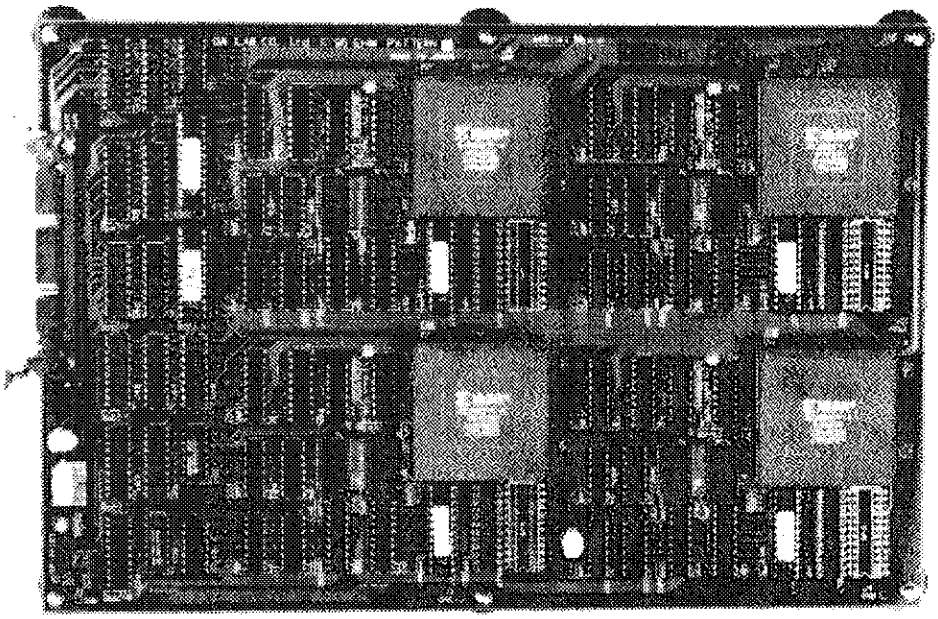

図 3.4: 実装に用いた EHW ボード

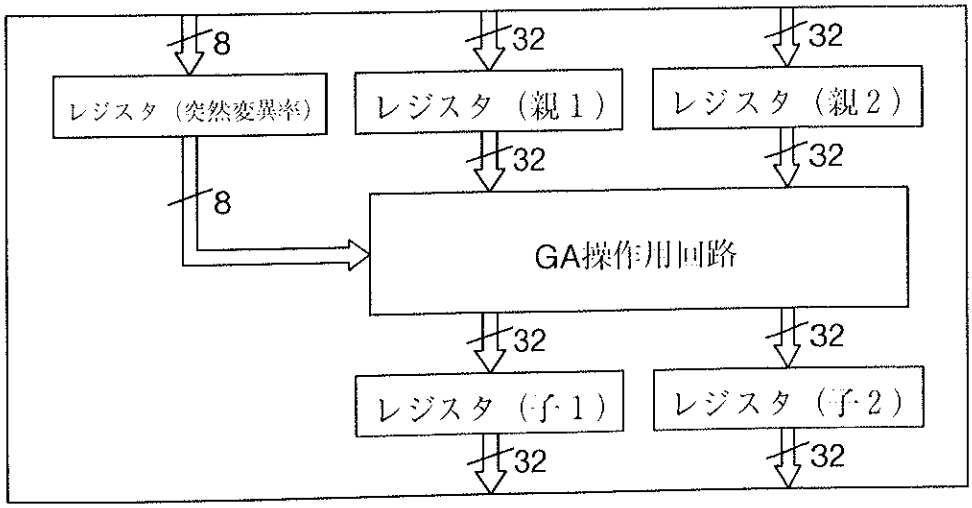

図 3.5: FPGA に実装した回路のブロック図

3.2.2.1 交叉用回路

交叉用回路は，乱数発生器から 32 ビットのランダムなビット列を受け取り，1 が立っているビットで親の染色体の情報を交換し，二つのビット列 (C1,C2) を作る (図3.7)．そしてビット列 (C1,C2) は，突然変異用回路 (図 3.8) で突然変異を施される．

3.2.2.2 突然変異用回路

突然変異用回路 (図3.8) は，図 3.9 の突然変異用マスク生成回路から 32 ビットのビット列 (突然変異用マスク) 二つ (M1,M2) を受け取り，それぞれ，交叉用回路で作ったビット列 (C1,C2) とビット毎の XOR をとる．つまり，M1,M2 のうち 1 が立っているビットで突然変異を施す．

3.2.2.3 突然変異用マスク生成回路

突然変異用マスク生成回路 (図 3.9) は，乱数と，突然変異率を比較することで，各ビット毎に，突然変異を施すかどうか決定し，突然変異用のマスク (M1, M2) を作る．具体的には，突然変異率 (8 ビット) と乱数 (8 ビット) を各ビット毎に比較する必要があるので，8 ビットの比較回路が必要である．しかし，8 ビットの比較回路は大きいため，突然変異率の性質を考慮して，次に示す方法を用いて回路を小さくする．

突然変異率は，その値が大きいと探索がランダムサーチに近くなってしまうため，あまり大きい値を採用することはない．つまり，突然変異率の上位ビットは常に 0 であることが多い．そこで，それらのビットを 0 に固定し，下位の 2 ビットだけを有効とすることにする．具体的には，図 3.9 のように，乱数の下位 2 ビットだけを比較し，上位 6 ビットに 1 がある場合は，突然変異率より大きいと判断する．この方式は，8 ビットの比較回路は必要なく，2 ビットの比較回路と 6 ビットの論理和だけで実現可能であるので，回路を小さくできる．

3.2.2.4 乱数発生器

乱数発生器は，従来の GA のハードウェア化の研究で最も多く用いられている 1 次元セルラオートマトンによる乱数発生器 [80][81][82][83] を用いる．

1 次元セルラオートマトンは，ある機能を持ったセルを 1 次元に並べた物である．i 番目のセルは，ある離散的な時間 (t) における状態 $S_i(t)$ と，次の時間 $(t+1)$ における状態 $S_i(t+1)$ を決定する規則からなる．$S_i(t+1)$ は，そのセルの状態 $S_i(t)$ や，そのとなりのセルの状態 $S_{i-1}(t)$，$S_{i+1}(t)$ などによってきまる．例えば，S.Wolfram が rule90 と名付けたものは，
規則
$S_i(t+1) = XOR(S_{i-1}(t), S_{i+1}(t))$ (XOR：排他的論理和)
にしたがって次の状態が決まり，rule150 の場合は，規則
$S_i(t+1) = XOR(S_{i-1}(t), S_i(t), S_{i+1}(t))$
に従う．

セルラオートマトンによる乱数発生器は，上の例に示した rule90 と rule150 を組み合わせることで，乱数を発生させる．例えば，１６ビットのランダムなビット列を発生させる場合，rule90 と rule150 を図 3.10 のように並べることで実現可能である[1]．

3.2.2.5 FPGA への実装

実装した回路 (図 3.11) は，XC4025 (25000 ゲート相当) の CLB (Configurable Logic Block) の 65 % (最大 1024 個中 668 個) しか使っていない．使用 CLB の数から正確なゲート数を計算することはできないが，16250 ゲート (25000 ゲートの 65 %) ～25000 ゲート程度の大きさである．ま

[1] この１６ビットの乱数発生器の周期は 65535 である [80]

た，親 1，親 2 のレジスタからから，子 1，子 2 のレジスタまでの遅延は 46.3[ns] であった．

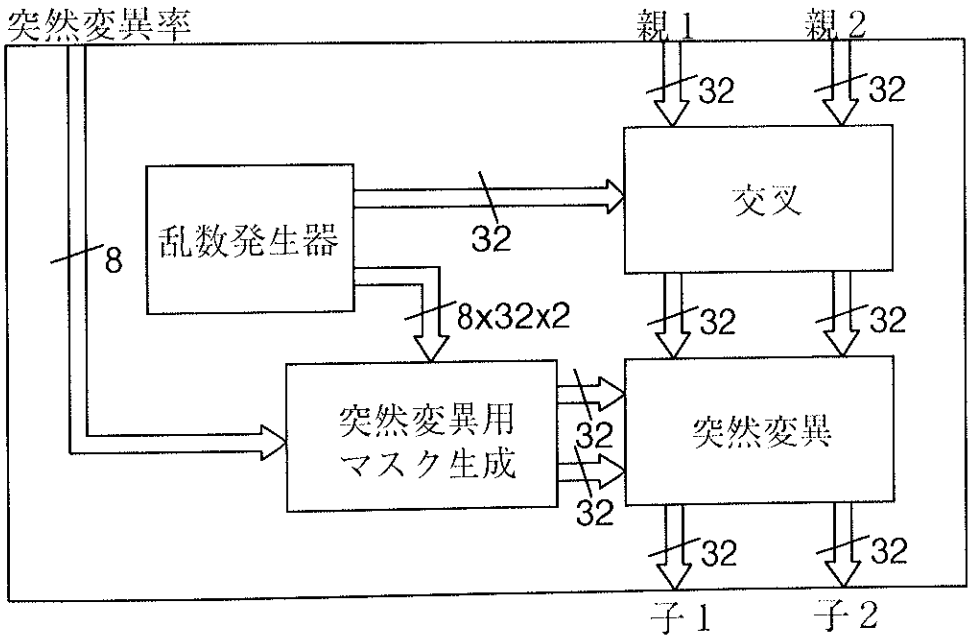

図 3.6: GA 操作用回路のブロック図

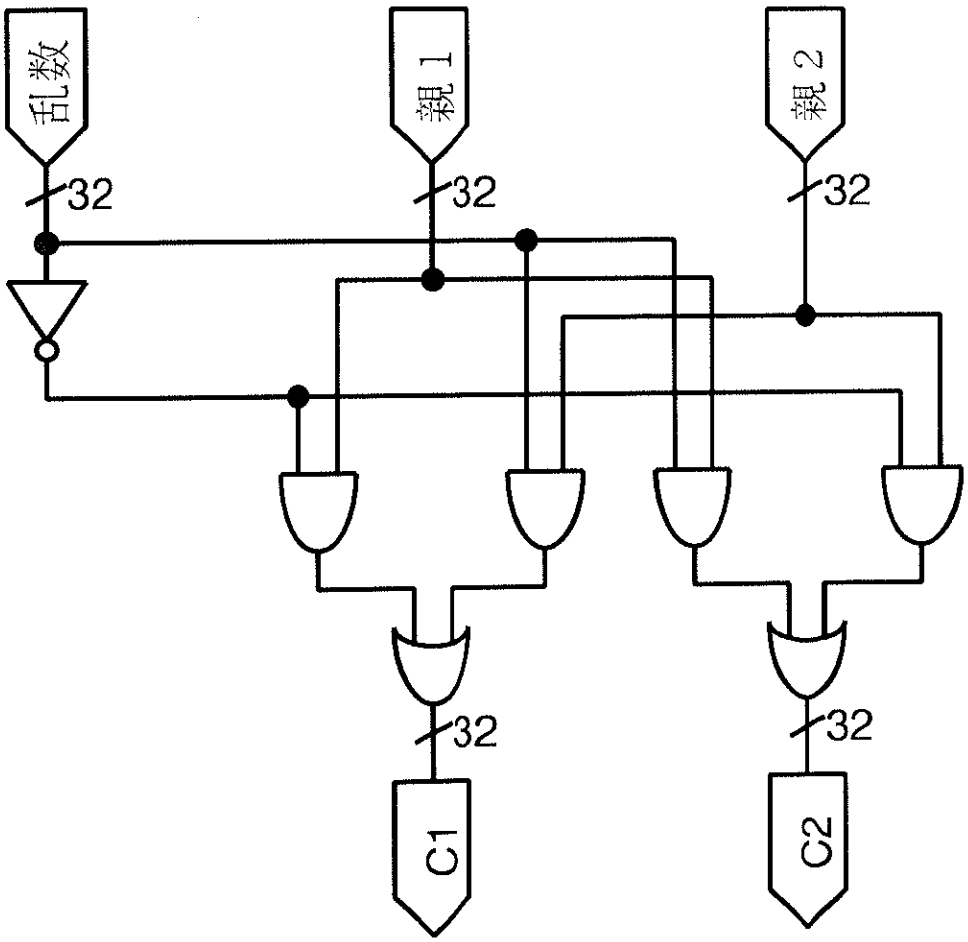

図 3.7: 交叉用回路

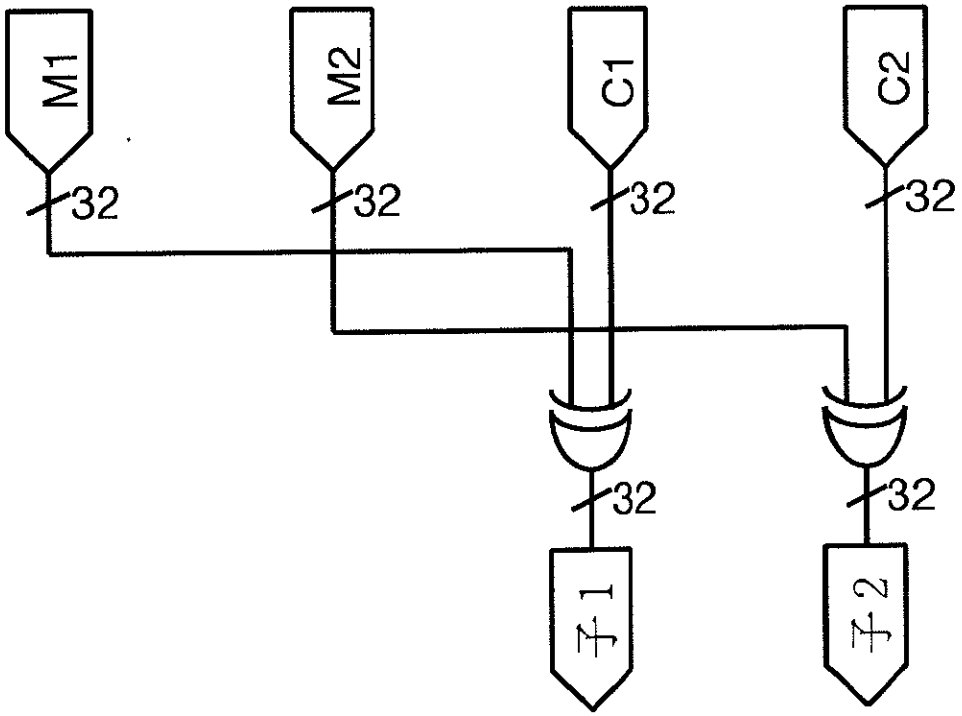

図 3.8: 突然変異用回路

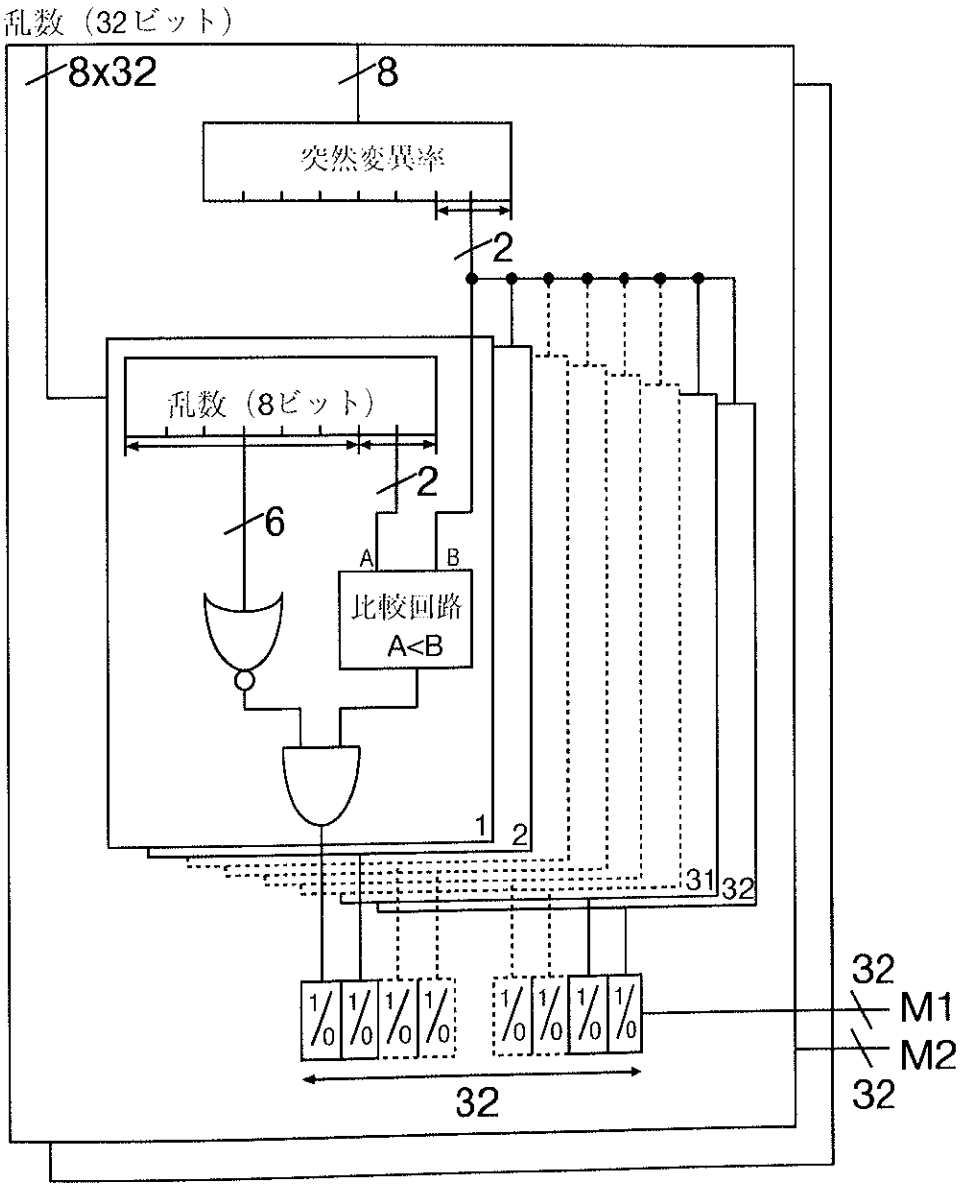

図 3.9: 交叉用マスク生成回路

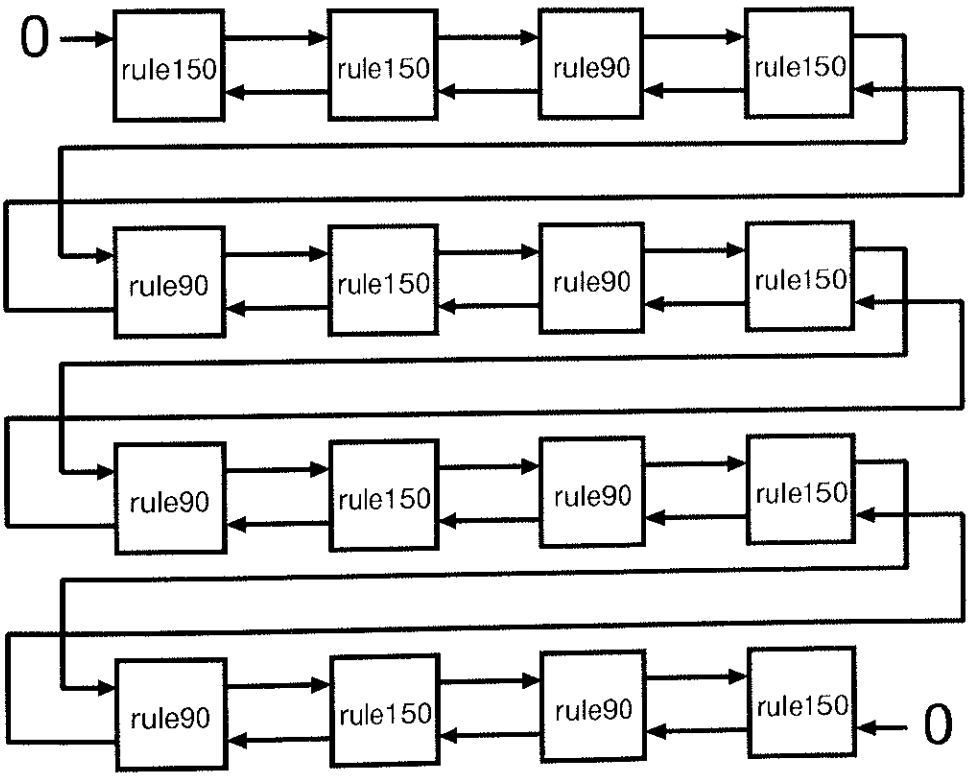

図 3.10: 1次元セルラオートマトンによる乱数発生（16ビット）

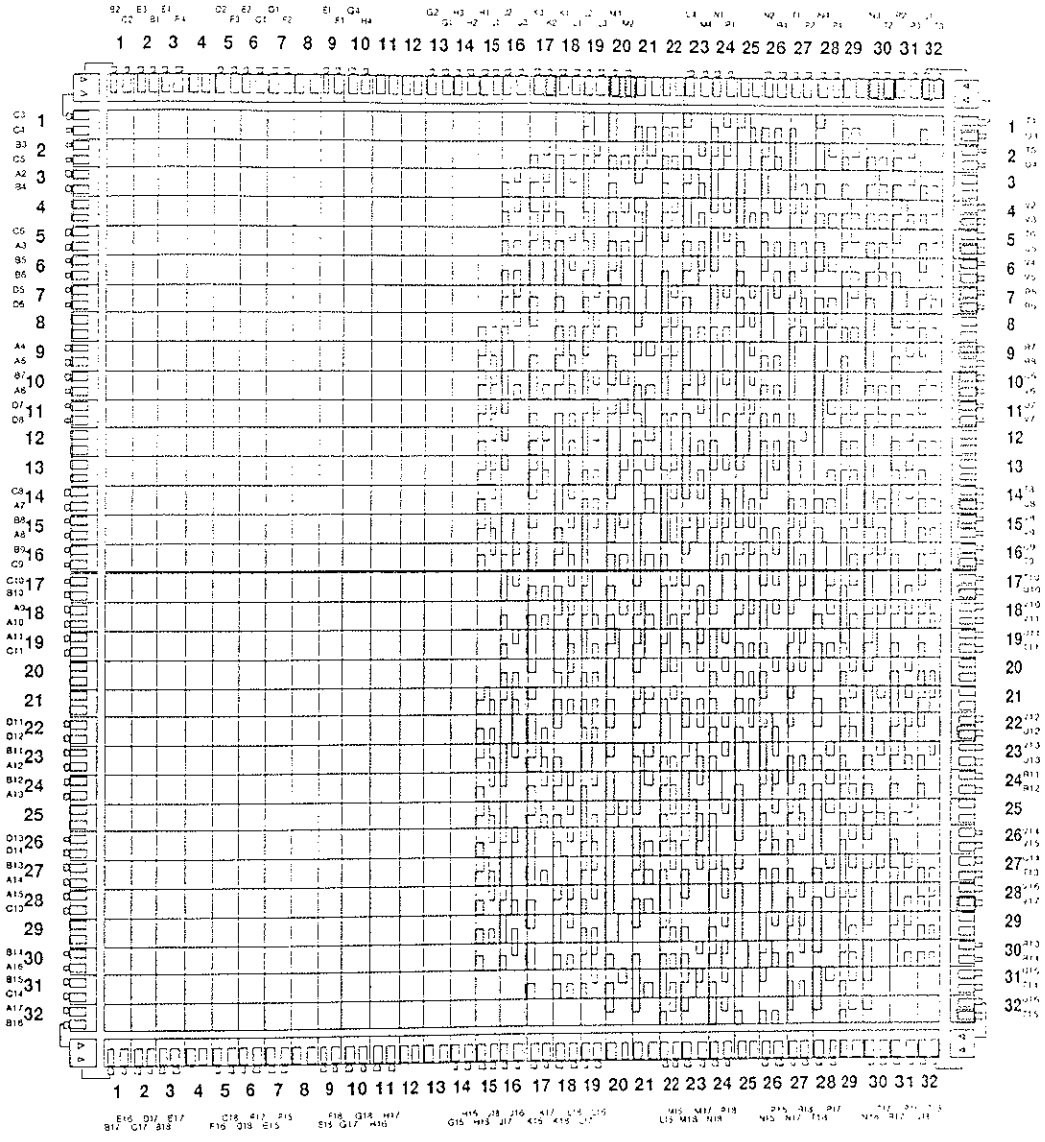

図 3.11: 実装した回路のフロアプラン

第 4 章

EHW チップ version 1 (EHWv1)

4.1 システム構成

EHW チップ *version 1* (EHWv1) は，図 4.1 に示すように，6 つの機能ブロックからなる．つまり，PLA ブロック，GA 操作用回路，染色体を保存するためのメモリ（染色体メモリ），トレーニングパターンを保存するためのメモリ（トレーニングパターンメモリ），そして，LSI 外部とのインタフェース用の CPU とレジスタファイルである．

4.1.1 各機能ブロックの説明

4.1.1.1 PLA ブロック

PLA ブロックは，二つの染色体を同時に評価するための二つの PLA（評価用 PLA）と，以下で説明する最小項学習用の PLA からなる（図 4.2）．

二つの評価用 PLA の出力は，セレクタで片方だけが選ばれる．この選択は，前回の PLA の回路の評価で，評価値の大きかったほうの PLA の出力を選択する．なぜなら，HSP-ER では，片親として常にエリートを選択するので，前回の評価で評価値の大きかったほうの PLA には，エリートの染色体情報をたくさん受け継いでいるビット列が来ているので，この

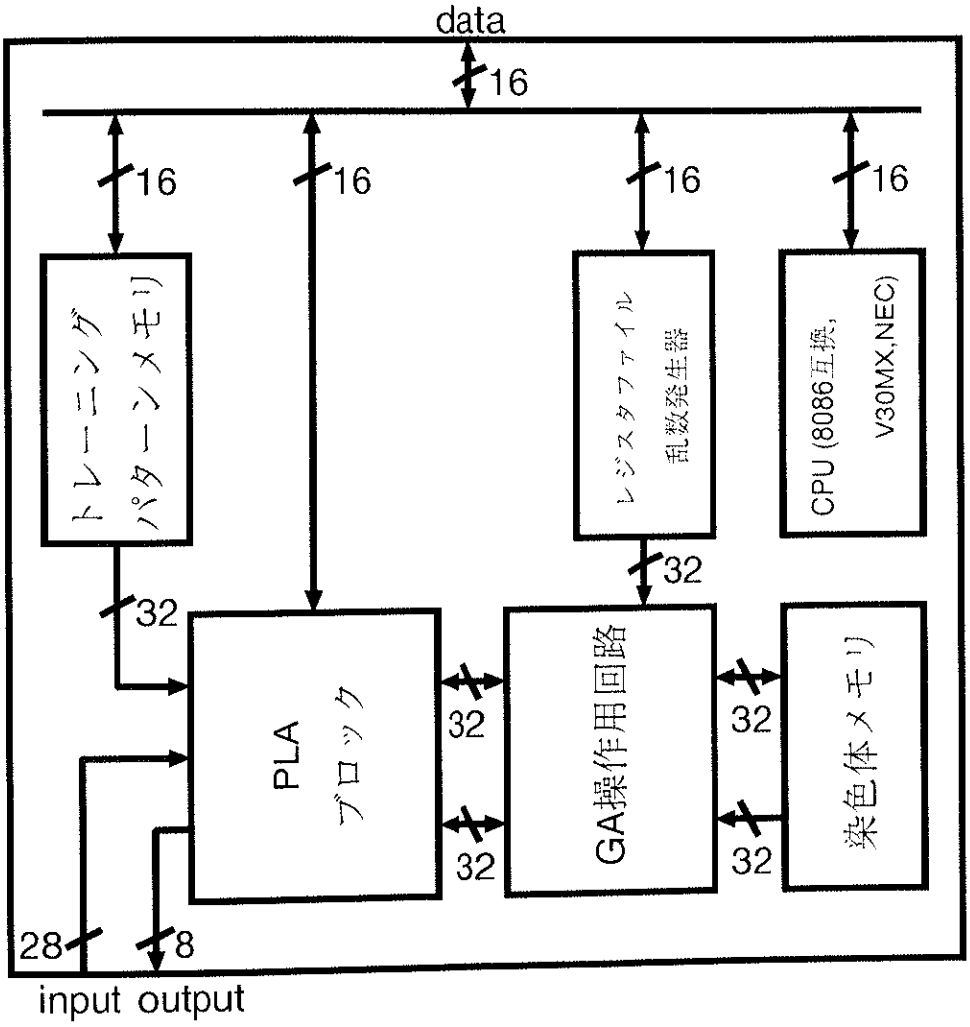

図 4.1: EHWv 1 のブロック図

PLA の出力パターンの方が，より目的にあっていると期待できるからである．

この PLA ブロックの出力は，セレクタの出力と，最小項学習用 PLA の出力の XOR （排他的論理和）である．

各 PLA （図 4.2 に評価用 PLA のブロック図を示す）は，それぞれ，入力のビット幅の最大値が２８ビット，出力のビット幅の最大値が 8 ビットで，評価用の PLA の積項線数はそれぞれ３２本で，最小項学習用の PLA の積項線数は６４本である．入力信号は，チップ外部からの入力パターンと，トレーニングパターンとを選択可能である．GA により回路を合成しているときは，トレーニングパターンを選択する．このトレーニングパターンのビット幅の最大値は２４ビットであるので，残りの 4 ビットには "0000" を入力する．入力信号（２８ビット）のうち 4 ビットは，出力のフィードバックを選択可能である．この PLA の回路を指定するスイッチは２０４８個あるので，これらを指定するビット列（染色体）の長さは２０４８ビットである．

4.1.1.2 GA 操作用回路

GA 操作用回路は，染色体メモリから二つの染色体を３２ビット単位で読み，それらに対して交叉と突然変異を施す．次に，それらを，PLA ブロックの評価用 PLA に書き，二つの回路を実現する．

4.1.1.3 染色体メモリ

これは，染色体を保存するためのメモリである．染色体長の最大値は２０４８で，個体数の最大値は３２であるので，このメモリの大きさは，３２ビット×２０４８ワードである．このメモリに保存された染色体は，３２ビット単位で，GA 操作用回路から読むことができる．

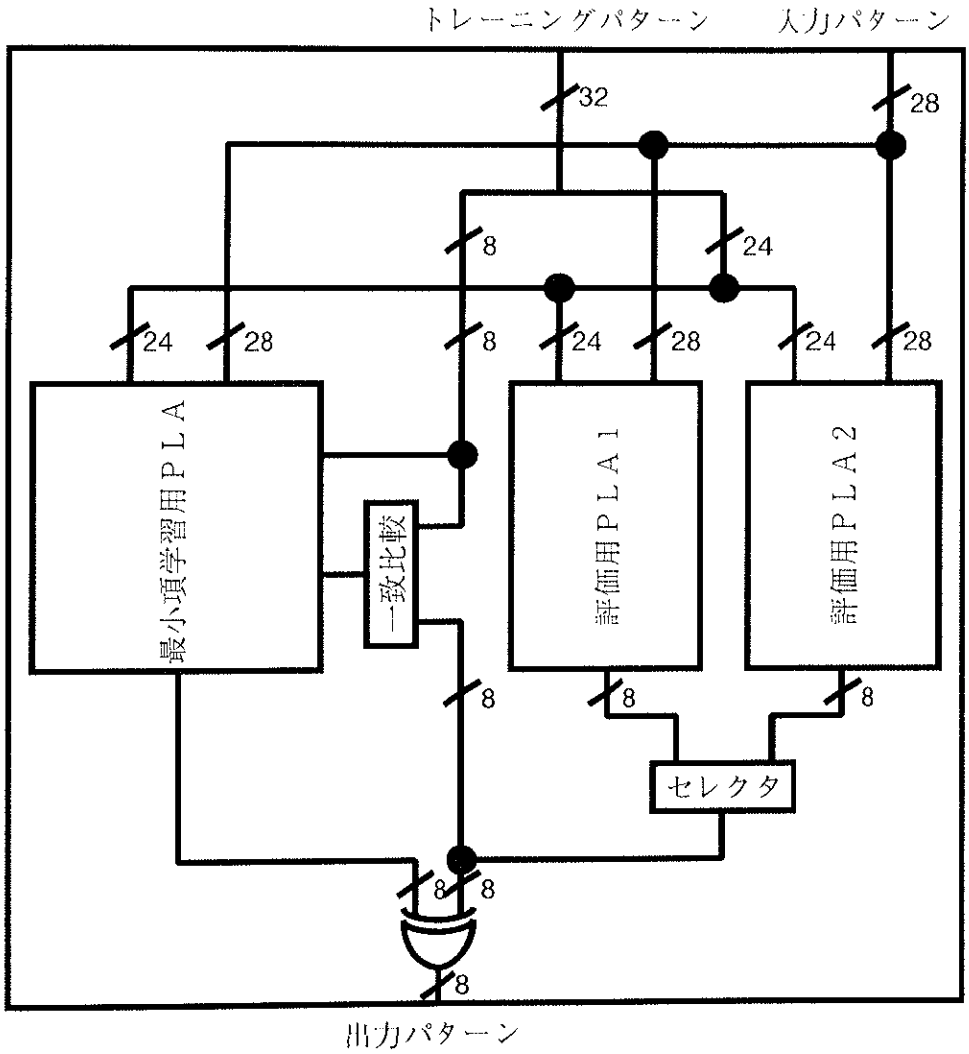

図 4.2: PLA ブロック（PLA への入出力信号線のみ）

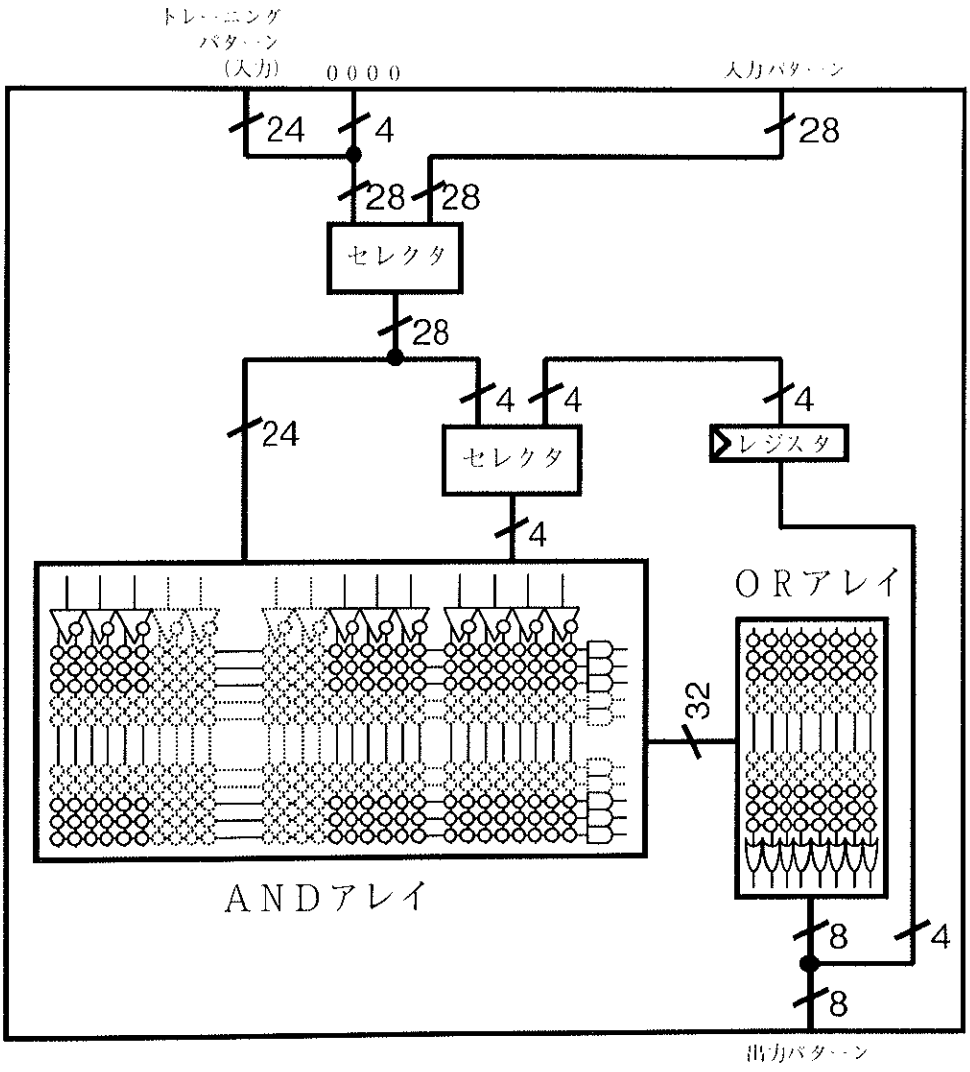

図 4.3: EHWv1 の評価用 PLA の構造

4.1.1.4 トレーニングパターンメモリ

PLA の回路を評価するために必要なトレーニングパターンを保存するためのメモリ．このメモリは，１６ビット×２０４８ワードのメモリ二つからなり，１６ビットデータバスを使って，CPU から読み書き可能である．

トレーニングパターンは，入力パターン２４ビットと出力パターン８ビット（合計３２ビット）からなり，１６ビットずつ二つに分けて二つのメモリに保存する．PLA の回路を評価するときは，二つのメモリから同時に１６ビットずつに分けられたトレーニングパターンを読み，これを評価用 PLA に入力する．

4.1.1.5 16bit CPU (8086 compatible,V30MX(NEC))

この CPU は，基本的にはチップの内部と外部のインタフェースとして用いる．さらに，PLA の回路の評価にトレーニングパターンを用いることができない場合は，この CPU を使って評価可能である．

4.1.1.6 レジスタファイル

このレジスタファイルも，上記 CPU とともに，チップ内部と外部のインタフェースとして用いる．さらに，このレジスタファイルに，PLA の入出力のビット幅や，最大評価回数などを設定可能である．

このレジスタファイルの一部のレジスタは，乱数を発生させるための CA としても動作可能である．

4.1.2 EHWv1 の動作フロー

EHWv1 は，次のように PLA の回路を合成する．

1. 乱数発生器初期化

   レジスタファイルの一部は乱数発生器としても動作する．そこで，レジスタファイルの一部に，CPU を使って乱数を書き込むことで，乱数発生器を初期化する．

2. 染色体メモリ初期化

   乱数発生器を動作させ，ランダムなビット列を，染色体メモリに書き込む．

3. トレーニングパターン書き込み

   染色体メモリの初期化と同時に，CPU を使ってトレーニングパターンをメモリに書き込む．

4. GA 動作開始

   2，3が終了した時点で，GA を動作させる．染色体メモリから二つの染色体を選択し，32ビット単位でGA ブロックに読み込む．それらに対して，交叉，突然変異を施し，PLA に書き込む．

5. PLA の回路の評価

   PLA ブロックには，二つのPLA があり，GA 操作を施された二つの染色体に対応する二つの回路を，同時に評価することができる．評価は，トレーニングデータメモリから順番にトレーニングデータを読み，それらに対するPLA の回路を求める．そして，トレーニングデータの出力と，PLA の出力を比較し，それらの一致率を評価値として用いる．

6. 選択

   二つの回路（染色体）の評価値が決まったら，それらと GA 操作を施す前の染色体の評価値を比較する．GA 操作後の染色体の評価値が大きい場合は，その染色体を GA 操作前の染色体と置き換える．

4.2 最小項学習による回路合成の高速化

ここでは，GA による PLA の回路合成を高速化するために提案した最小項学習について述べる．最小項学習は，合成に時間のかかる回路の特徴を考察し，それらの回路を高速に合成するために提案した．そこで，以下ではまず，合成に時間のかかる回路の特徴を調べる．

4.2.1 合成に時間のかかる回路の解析

4.2.1.1 解析方法

合成に時間のかかる積項の特徴を考察するために，ここでは，以下に示す三つの基本的な組合せ回路を合成するシミュレーションを行う．

- マルチプレクサ：6 ビット入力，1 ビット出力
- コンパレータ（大小比較）：6 ビット入力，1 ビット出力
- 加算器：6 ビット入力，3 ビット出力

マルチプレクサ（表 4.1）は，6 ビット入力のうち 4 ビットがデータ入力，2 ビットがアドレス入力で，アドレス入力で指定されたデータビットを出力する．例えば，アドレス入力が 0 0 である場合は，0 ビット目のデータ入力を出力する．

コンパレータ（表 4.2）は，二つの 3 ビット入力（入力 A，入力 B）を持ち，入力 A が入力 B より大きいとき 1 を出力する．

加算器（表 4.3）は，二つの 3 ビット入力（入力 A，入力 B）を持ち，それらの和（3 ビット，桁上がりは省略）を出力する．

シミュレーションでは，各回路の真理値表をトレーニングパターンとし，それらに対する正解率を評価値として用いた．回路合成の諸設定について

表 4.1: マルチプレクサの真理値表

入力		入力		入力		入力	
アドレス　データ	出力	アドレス　データ	出力	アドレス　データ	出力	アドレス　データ	出力
00_0000	0	01_0000	0	10_0000	0	11_0000	0
00_0001	1	01_0001	0	10_0001	0	11_0001	0
00_0010	0	01_0010	1	10_0010	0	11_0010	0
00_0011	1	01_0011	1	10_0011	0	11_0011	0
00_0100	0	01_0100	0	10_0100	1	11_0100	0
00_0101	1	01_0101	0	10_0101	1	11_0101	0
00_0110	0	01_0110	1	10_0110	1	11_0110	0
00_0111	1	01_0111	1	10_0111	1	11_0111	0
00_1000	0	01_1000	0	10_1000	0	11_1000	1
00_1001	1	01_1001	0	10_1001	0	11_1001	1
00_1010	0	01_1010	1	10_1010	0	11_1010	1
00_1011	1	01_1011	1	10_1011	0	11_1011	1
00_1100	0	01_1100	0	10_1100	1	11_1100	1
00_1101	1	01_1101	0	10_1101	1	11_1101	1
00_1110	0	01_1110	1	10_1110	1	11_1110	1
00_1111	1	01_1111	1	10_1111	1	11_1111	1

表 4.2: コンパレータの真理値表

入力 入力A_入力B	出力	入力 入力A_入力B	出力	入力 入力A_入力B	出力	入力 入力A_入力B	出力
000_000	0	010_000	1	100_000	1	110_000	1
000_001	0	010_001	1	100_001	1	110_001	1
000_010	0	010_010	0	100_010	1	110_010	1
000_011	0	010_011	0	100_011	1	110_011	1
000_100	0	010_100	0	100_100	0	110_100	1
000_101	0	010_101	0	100_101	0	110_101	1
000_110	0	010_110	0	100_110	0	110_110	0
000_111	0	010_111	0	100_111	0	110_111	0
001_000	1	011_000	1	101_000	1	111_000	1
001_001	0	011_001	1	101_001	1	111_001	1
001_010	0	011_010	1	101_010	1	111_010	1
001_011	0	011_011	0	101_011	1	111_011	1
001_100	0	011_100	0	101_100	1	111_100	1
001_101	0	011_101	0	101_101	0	111_101	1
001_110	0	011_110	0	101_110	0	111_110	1
001_111	0	011_111	0	101_111	0	111_111	0

表 4.3: 加算器の真理値表

入力		入力		入力		入力	
入力A 入力B	出力	入力A 入力B	出力	入力A 入力B	出力	入力A 入力B	出力
000_000	000	010_000	010	100_000	100	110_000	110
000_001	001	010_001	011	100_001	101	110_001	111
000_010	010	010_010	100	100_010	110	110_010	000
000_011	011	010_011	101	100_011	111	110_011	001
000_100	100	010_100	110	100_100	000	110_100	010
000_101	101	010_101	111	100_101	001	110_101	011
000_110	110	010_110	000	100_110	010	110_110	100
000_111	111	010_111	001	100_111	011	110_111	101
001_000	001	011_000	011	101_000	101	111_000	111
001_001	010	011_001	100	101_001	110	111_001	000
001_010	011	011_010	101	101_010	111	111_010	001
001_011	100	011_011	110	101_011	000	111_011	010
001_100	101	011_100	111	101_100	001	111_100	011
001_101	110	011_101	000	101_101	010	111_101	100
001_110	111	011_110	001	101_110	011	111_110	101
001_111	000	011_111	010	101_111	100	111_111	110

は，表 4.1 に示した設定を用い，シミュレーションは乱数の初期値を変えて 50 回行った．

表 4.1: シミュレーションの諸設定

	マルチプレクサ	コンパレータ	加算器
入力数 / 出力数	6/1	6/1	6/3
PLA の積項線数	16	16	32
GA 操作の種類	ルーレット選択，一点交叉		
個体数	50 個体		
交叉率 (GA)	0.5/ 染色体		
突然変異率 (GA)	0.01/ ビット		
終了条件 (GA)	正解率 100 %，または 10000 世代		

シミュレーションの結果，マルチプレクサは 50 回平均 181.8 世代で合成できたのに対して，コンパレータの合成には 50 回平均 1785.2 世代を要し，加算器は 10000 世代までには合成できなかった．そこで，コンパレータの合成について，合成途中の回路の特徴を調べることにする．

まず，64 個 (6 ビット入力だから $2^6 = 64$ 個) の入力パターンのうち 60 個に対して正しい信号を出力する回路 (評価値 = 正解率 =60/64=0.9375) が合成できた時点で，それらの回路の出力が正しくなかった入力パターンを調べた．そして，各入力パターンについて，出力が正しくなかった回数を図 4.4 のグラフに示した．グラフの横軸は 64 個の入力パターンを 10 進数で表現したもので，縦軸はその入力パターンに対する出力が正しくなかった回数である．

グラフから明らかなように，4 つの入力パターン 001000(8)，011010(26)，101100(44)，111110(62) に対する出力が間違っていた回数が，他の入力パターンの場合よりはるかに大きい．そして，これら 4 つの入力パターンに対するコンパレータの出力は 1 であるのに対し，合成途中の回路は 0 を

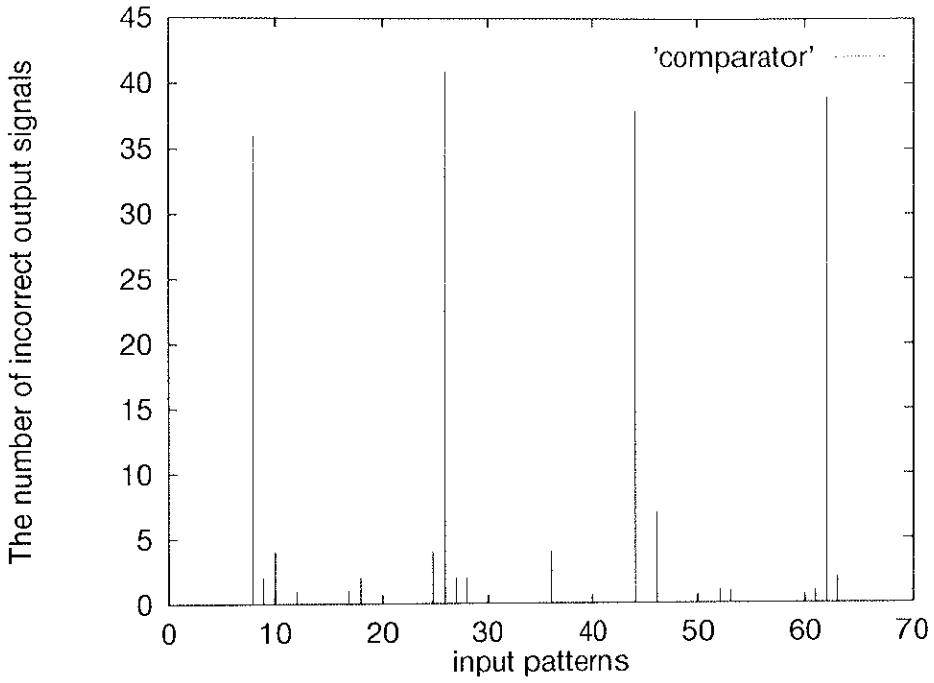

図 4.4: 各入力パターンについて出力が正しくなかった回数

出力している．つまり，これらの 4 つの入力パターンに対して **1** を出力する積項 （表 4.5） が合成できてないために，合成途中の回路は正しい出力をすることができないのである．

また，表の各積項はいずれも 4 ～ 6 個の入力信号の論理積であり，1 ～ 3 個の入力信号の論理積はない．このことから，積項を構成する入力信号の数と，GA による合成の難しさには何らかの関係があると考え，その関係を次のように調べる．

1. 出力が **1** の入力パターンについて，それらに対して **1** を出力する積項を全て求める．

   例 入力パターン 111110 の場合，表の第一列に示した積項．

2. それらの積項のうち，構成する入力信号が最も少ないものを求める．

表 4.5: 出力が正しくなかった 4 つの入力パターンに対して 1 を出力する積項 ($X0 \sim X5$ は入力信号)

111110	101100	011010	001000
$X0X1X2X5$	$X0X2X4\bar{X}5$	$X1X2X3X5$	$X2X3X4X5$
$X0X1X2X3X5$	$X0X1X2X4X5$	$X0X1X2X3X5$	$X0X2X3X4X5$
$X0X1X2X4X5$	$X0X2X3X4X5$	$X1X2X3X4X5$	$X1X2X3X4X5$
$X0X1X2X3X4X5$	$X0X1X2X3X4X5$	$X0X1X2X3X4X5$	$X0X1X2X3X4X5$

例 入力パターン 111110 の場合，表に示した積項のうち，積項 $X0X1X2X5$ が構成する入力信号の数 (4 個) が最も少ない．

3. 上の二つの操作 (1),(2) を，出力が **1** の入力パターン全てについて行う．

4. 各入力パターンを，(2) で求めた積項を構成する入力信号の数によって，6 つのグループに分ける．(積項を構成する入力信号が 1 個の場合をグループ 1 とし，6 個の場合をグループ 6 とする．)

   例 表の 4 つの入力パターンの場合は，いずれもグループ 4 に分類される．

5. 各入力パターンに対する図 4.4 のグラフの値を，(4) で分類した各グループ毎に平均する．

   例 グループ 4 には表の 4 つの入力パターンが分類されているから，それらに対する出力が正しくなかった回数を図のグラフから求め，それらの平均を求める．

これらの操作を，コンパレータ，加算器，マルチプレクサの合成のシミュレーションについて行い，その結果を表 4.6 に示した．ただし，加算器の

場合は出力が 3 ビットであるので，MSB(Most Significant Bit) だけについて調べた．

4.2.1.2　解析結果の考察

コンパレータの場合は，各入力パターンはグループ 2 ～ 1 に分類され，グループ 4 に分類された入力パターンに対する出力が間違っていた回数の平均が最も大きい．加算器の場合はグループ 5 と 6 に分類され，コンパレータと同じように構成する入力パターンの数が大きいグループのほうが出力が正しくなかった回数の平均が大きい．つまり，積項を構成する入力信号の数が多いほど，GA により合成しにくいと考えられる．マルチプレクサの場合は，全ての入力パターンがグループ 3 に分類されるので，構成する入力信号の数が大きい積項は必要がなく，そのため高速に回路を合成できたと考えることができる．

以上の結果から，構成する入力信号の数が大きい積項の合成には時間がかかり，そのような積項を必要とする回路の合成は遅いことがわかる．

構成する入力信号の数が大きい積項は，コンパレータや加算器などの算術演算回路に必要な積項である．例えば，コンパレータの場合，二つの入力をそれぞれ X 軸，Y 軸として二次元のグラフにあらわすと，図 4.5 になる．コンパレータは，図の右下の網かけ部分の入力パターンに対して 1 を出力する．表 4.5 に示した 4 つの入力パターンは，図の黒丸に対応し，これらの入力パターンは二つの入力値の差が 1 しかないので，これらに対する出力が正しくない場合は二つの入力値の比較の精度が悪くなる．

残りの入力パターン (図の白丸に対応) については，例えば，図の右上の四角で囲った 4 つの入力パターン ("110100", "110101", "111100", "111101") は，一つの積項 $X0X1X3X4$ が 1 を出力し，左下の四角で囲った 4 つの入力パターン ("010000", "100001", "011000", "011001") は，一つの積項 $X0X1\overline{X3}X4$ が 1 を出力し，右下の大きな四角で囲った 16

個の入力パターン ("100000"、"100001"、"100010"、"100011"、"101000"、"101001"、"101010"、"101011"、"110000"、"110001"、"110010"、"110011"、"111000"、"111001"、"111010"、"111011") は，一つの積項 $X0X3$ が 1 を出力する．つまり，これらの入力パターンに対して 1 を出力するには，積項 $X0X1X3X4$，$X0X1X3X4$，$X0X3$ が合成できていればよく，最小項のような構成する入力信号の数が多い積項は必要ない．

しかしながら，構成する入力信号の数が少ないの積項は"1"を出力する入力パターンが少ないので，その積項が存在しない場合でも，残りの多くの入力パターンに対する出力パターンには影響しない．例えば，表 4.5 の一番下の積項のように全入力信号から構成される積項は最小項と呼ばれ，一つの入力パターンに対してのみ"1"を出力する．そのため，この最小項が合成できてなくても，一つの入力パターンに対する出力パターンが異なるだけであるので，GA の評価値（出力パターンの正解率）にはほとんど影響せず，GA による回路合成では合成されにくい．

そこで本論文では，回路合成に要する時間を短縮するために，回路合成に時間のかかる積項，つまり最小項に代表される構成する入力信号の数が多い積項を，その他の積項とは別に合成する方法（以下では，最小項学習法と呼ぶ）を提案する．

表 4.6: 各グループ毎の出力が正しくなかった回数の平均

グループ番号	6	5	4	3	2	1
コンパレータ	-	-	38.5	1.15	0.01	-
加算器	21.3	7.79	-	-	-	-
マルチプレクサ	-	-	-	3.34	-	-

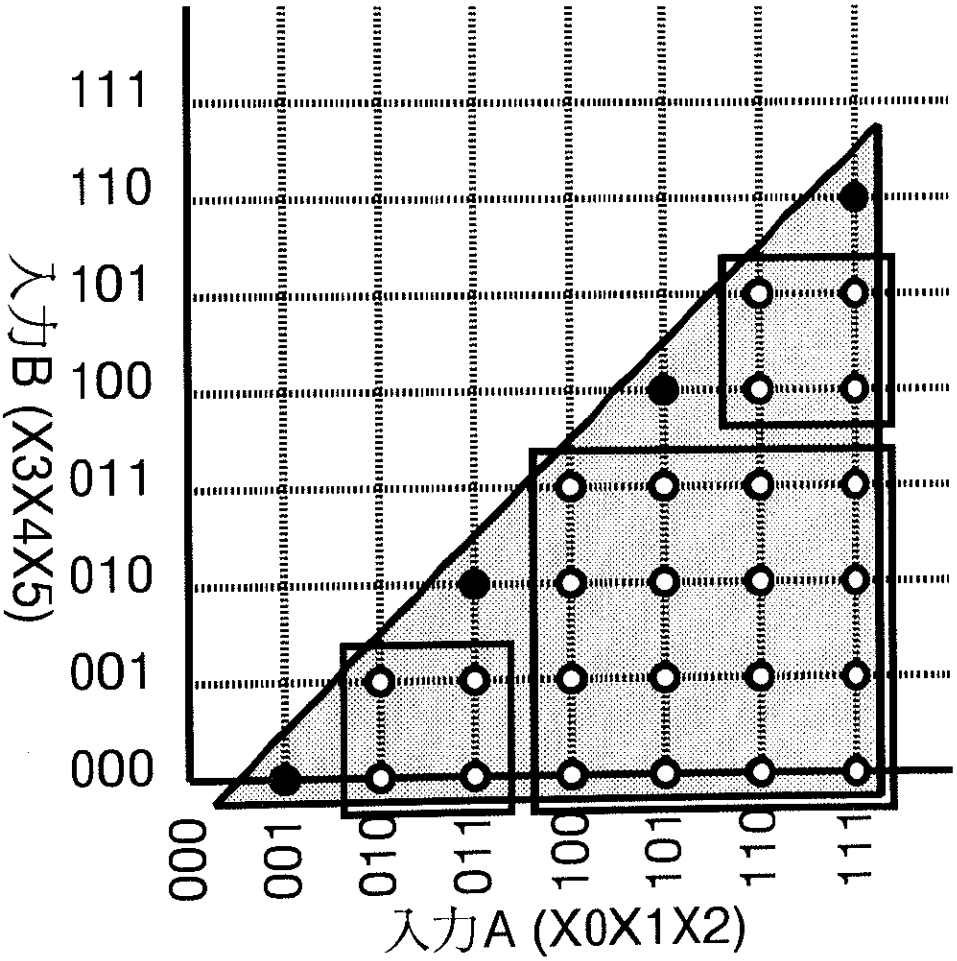

図 4.5: コンパレータの入出力パターン；各入力値を X 軸，Y 軸とし，二次元のグラフあらわしたもの．右下の網かけ部分の入力パターンに対して 1 を出力する．

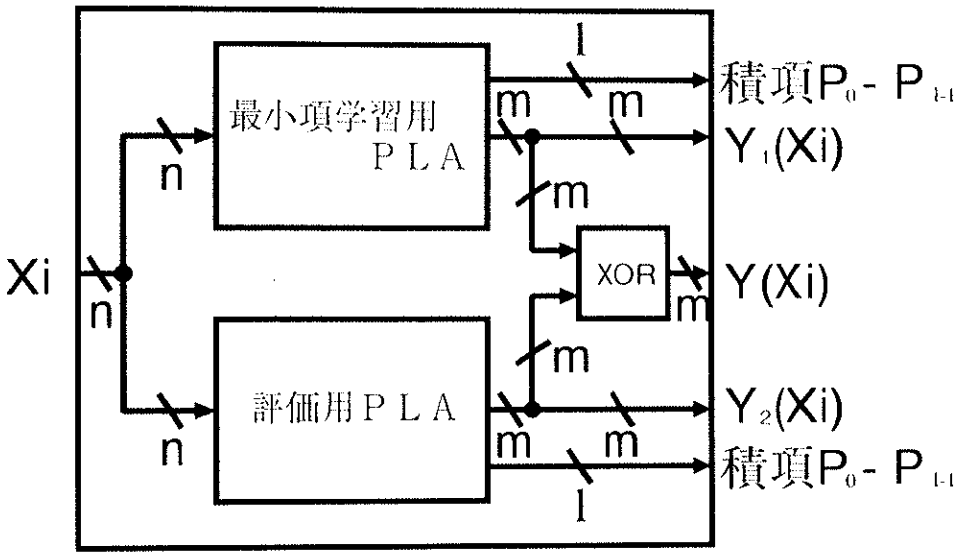

図 4.6: 簡単化した PLA ブロック；評価用 PLA と最小項学習用 PLA だけからなる場合

4.2.2 最小項学習による回路合成

最小項学習法の基本アイデアは，合成に時間のかかる積項（最小項などの構成する入力信号の多い積項）と，それら以外の積項を別々に合成することである．つまり，EHWv1 の PLA ブロックは，図 4.2 に示すように，評価用の PLA が 2 つ（評価用 PLA 1，評価用 PLA 2）と，最小項学習用 PLA からなり，最小項学習用の PLA に GA による合成に時間のかかる積項を合成し，評価用の PLA にはそれら以外の積項を GA により合成する．

ここでは，説明を簡単にするために，図 4.6 に示すような評価用の PLA と最小項学習用 PLA だけからなる場合について述べる．まず，評価用の PLA の回路は従来のように GA により合成し，最小項学習用の回路は，評価用 PLA の合成がうまくいかないときに図 4.7 に示したフローチャートに従って合成する．つまり，ある入力パターン X に対する評価用 PLA の出力 $Y_1(X)$ が正しくなかったとき ($\overline{Y_1(X)}$ が正しい出力)，最小項学習

用 PLA にはその入力パターンに対して 1 を出力する積項を合成する．このとき，出力 $Y(X)$ は評価用 PLA の出力 $Y_1(X)$ と最小項学習用 PLA の出力 $Y_2(X)$ の排他的論理和 (XOR) であり，排他的論理和は一方の入力が 1 のときに他方の入力の否定を出力する．そのため，$Y_2(X) = 1$ の時は $Y(X) = XOR(Y_1(X), Y_2(X)) = \overline{Y_1(X)}$ となり，入力パターン X に対する出力 $Y(X)$ は，正しい出力信号となる．

ただし，最小項学習用 PLA の積項線の数には限りがあるので，評価用 PLA の出力が正しくない入力パターンが多い場合は，それらに対して 1 を出力する積項の全てを最小項学習用 PLA に合成することはできない．

ところが，評価用 PLA の合成が進むと，最小項学習用 PLA の積項には不要なものができる．つまり，ある入力パターンに対する評価用 PLA の出力が正しいにも関わらず，最小項学習用 PLA が **1** を出力することがある．この場合は，その入力パターンに対して 1 を出力する最小項学習用 PLA の積項を削除する．こうすることによって，最小項学習用 PLA には別の積項を合成することができる．

この方法で各 PLA を合成すると，GA により合成するのに時間のかかる積項が，結果的に最小項学習用 PLA に合成される．

以下では，入力パターン X_i に対する正しい出力を $R(X_i)$ とし，図のフローチャートに従って最小項学習用 PLA の回路の合成方法を説明する．なお，出力 Y, Y_1, Y_2 はそれぞれ 1 ビットとするが，出力が多ビットの場合でも同じ方法で合成可能である．

処理 1 入力パターン X_i に対する出力 $Y(X_i)$ が，正しい出力 $R(X_i)$ と等しい場合は処理 5 へ，正しくない場合は処理 2 へ．

処理 2 入力パターン X_i に対する最小項学習用 PLA の出力 $Y_2(X_i)$ が 1 の時は処理 4 へ，0 の時は処理 3 へ．

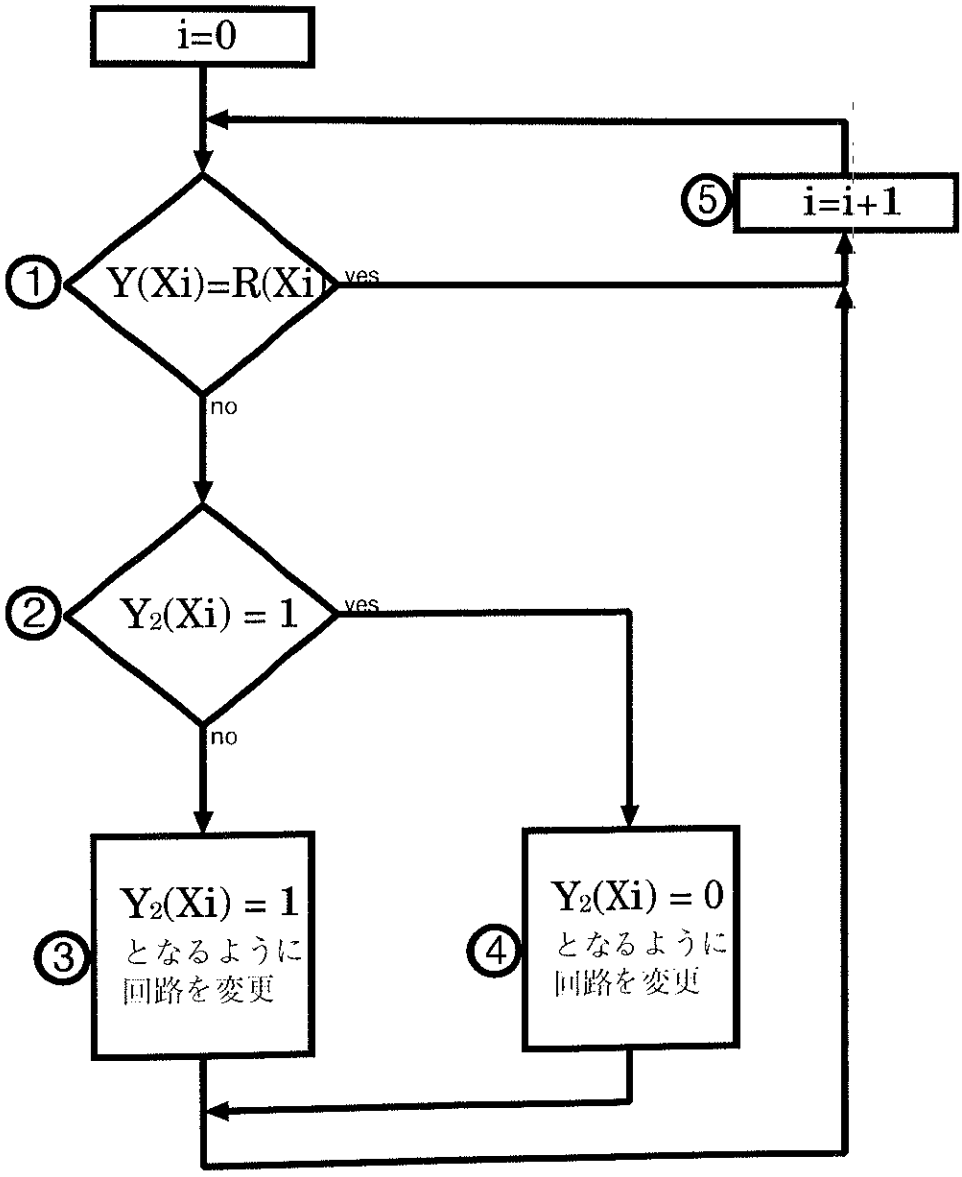

図 4.7: 最小項学習用 PLA の回路合成の流れ

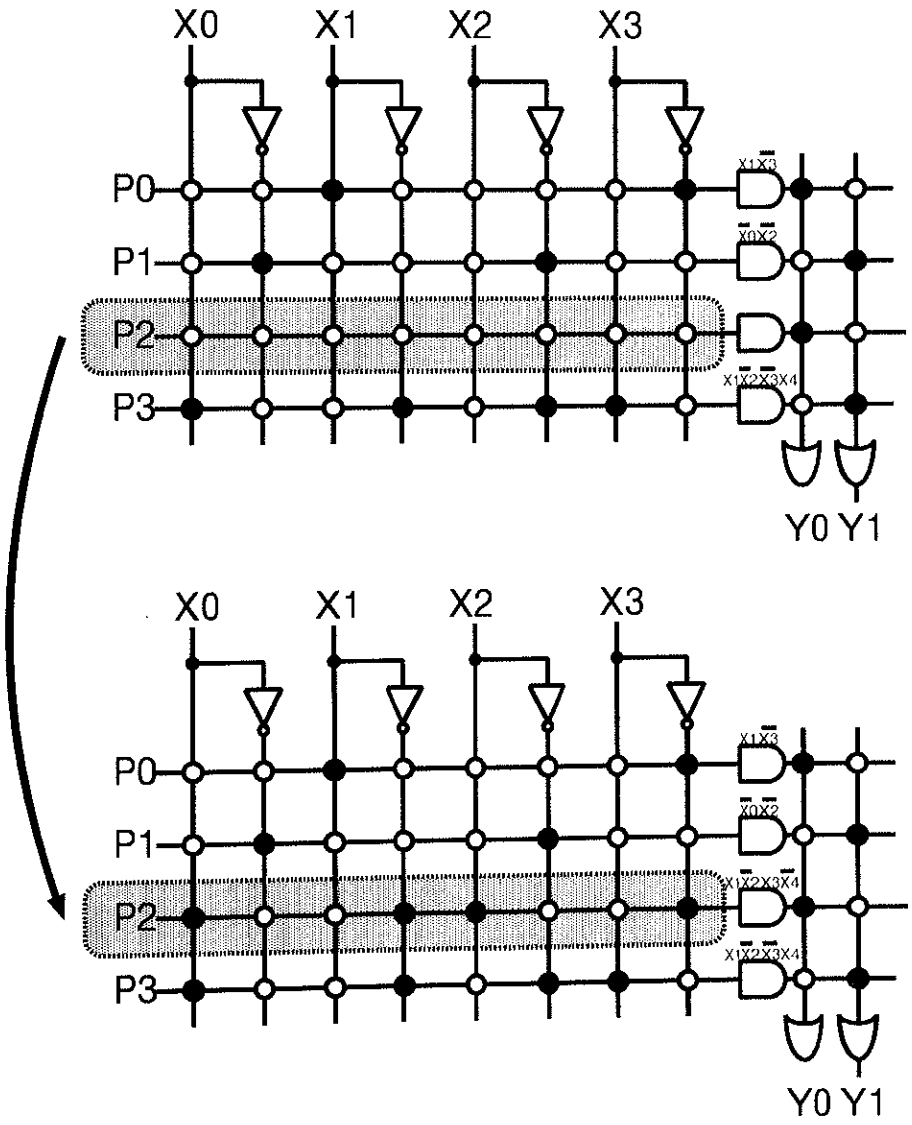

図 4.8: 最小項学習用 PLA の積項の合成方法

処理 3 評価用 PLA の出力が正しくないので，次に示す方法で $Y_2(X_i) = 1$ となるように最小項学習用 PLA の回路を変更し，処理 5 へ

- 入力パターン X_i に対してのみ 1 を出力する積項を最小項学習用 PLA に合成する
  - 最小項学習用 PLA の横方向の信号線のうち，図 4.8 左の PLA の信号線 $P2$ のように全てのスイッチが OFF 状態のものがあれば，そのスイッチの設定を図 4.8 右のように変更する
    - 例：図 (左) の PLA の信号線 $P2$ に，入力パターン 1010 に対してのみ 1 を出力する積項を合成する場合は，スイッチの設定を以下のルールに従って図 (右) の PLA のように変更する．
    - 信号が 1 の入力 ($X0, X2$) については，肯定入力に接続するスイッチを ON にする．
    - 信号が 0 の入力 ($X1, X3$) については，否定入力に接続するスイッチを ON にする．

処理 4 評価用 PLA の出力が正しいのに最小項学習用 PLA が 1 を出力しているので，$Y_2(X_i) = 0$ となるように最小項学習用 PLA の回路を変更し，処理 5 へ

- 最小項学習用 PLA の積項 ($P_0 \sim P_{k-1}$) のうち 1 を出力している積項を探す→例：P_k=1 の場合
  - 最小項学習用 PLA の k 番目の積項を削除する．
    - スイッチを全て OFF 状態にする．
    - 別の積項を合成できるようになる．

処理 5 $i = i + 1$，処理 1 へ

4.2.3 最小項学習法による回路合成シミュレーション

4.2.3.1 シミュレーション方法

トレーニングパターンは，XX のシミュレーションと同じく 3 つの回路それぞれの真理値表を用いる。そして，最小項学習法による回路合成のシミュレーションを乱数の初期値を変えて 50 回行い，合成に要した世代数の平均を，GA だけで PLA の回路を合成する方式と比較する。GA の各パラメータは表 3 の設定を用いる。

シミュレーションで用いた PLA は，入力 6 ビット，出力は，マルチプレクサとコンパレータの場合が 1 ビット，加算器の場合が 3 ビットである。積項線数は，マルチプレクサとコンパレータの場合が 8，加算器は他の 2 つの回路よりも多くの積項を必要とするので 16 とした。また，回路合成に要する世代数の比較が目的であるので，回路合成は最大 10000 世代まで行い，その時点で回路が合成できてなくても，強制的にシミュレーションを終了した。

4.2.3.2 シミュレーション結果と考察

表 6 に示したように，最小項学習法を用いたほうが GA だけで合成する方式と比べて，マルチプレクサの合成では約 20 倍，コンパレータの合成では約 614 倍，高速に回路を合成できた。さらに加算器の合成では，GA だけで合成する方式では 10000 世代までに 2 回しか回路を合成できなかったのに対し，最小項学習法では 50 回とも回路合成が終了した。

マルチプレクサの場合は，3 章で述べたように合成に時間のかかる積項，つまり構成する入力信号の数が少ない積項を必要としないので，最小項学習法を用いても 20 倍しか合成に要する時間が速くならなかった。それに対して，コンパレータや加算器のように，合成に時間のかかる積項を必要

とする回路の場合，最小項学習法を用いて回路を合成することで，最大で約 614 倍，高速に回路を合成できた。

表 4.7: シミュレーション結果

	GA + 最小項学習		GA のみ	
	世代数の平均	成功回数	世代数の平均	成功回数
6 マルチプレクサ	17.4	50	356.76	50
3 ビットコンパレータ	8.1	50	4975.57	37
3 ビット加算器	829.0	50	7287.0	2

4.3 VLSI への実装

本研究では，図 4.1 のブロック図に示す回路を，VLSI (NEC, 0.35μm CMOS セルベース IC, CBC9 ファミリ [44], QFP 304 ピンパッケージ) に実装した (図 4.9) ．実装した回路の総ゲート数は 853011 ゲートである．

この回路の GA 操作，評価，選択に要する時間を表 4.8 に示す．

GA 操作は，染色体メモリから３２ビット単位で染色体を読み，それらに GA 操作用回路で交叉と突然変異を施して，評価用の PLA に書く操作である．染色体長の最大値は２０４８であるから，１回の GA 操作では，これらの操作を６４回繰り返す必要がある．３２ビット単位の染色体の移動には，１回辺り１クロックサイクル必要で，EHWv1 は３３ MHz 動作であるから，１回の GA 操作に要する時間は３０ [ns] $\times 2 \times 64 = 3840$ [ns] $= 3.84$ [μs] である．

評価は，トレーニングパターンをメモリから読み，PLA の回路を評価する操作である．評価時間は，トレーニングパターンの数に依存し，例えば，６章の XXX 実験では，１２００個のトレーニングパターンを用いて

回路を合成する．この場合，評価に要する時間は，３０ [ns] ×２×１２００＝７２０００ [ns] ＝７２．０ [μs] である．また，トレーニングパターンの最大値は２０４８であるので，この場合は，３０ [ns] ×２×２０４８＝１２２８８０ [ns] ＝１２２．８８ [μs] である．

選択は，PLA の回路の評価値が，GA 操作を施す前の染色体の評価値より大きい場合に，染色体を PLA からメモリに戻す操作である．この操作は，GA 操作と逆に染色体を移動するだけであるので，実行時間は，GA 操作と同じである．つまり，１回の選択操作には３．８４ [μs] 必要である．

表 4.8: 各操作の実行時間

GA 操作	３．８４ [μs]
評価 （トレーニングパターン１２００個）	７２．００ [μs]
評価 （トレーニングパターン２０４８個）	１２２．８８ [μs]
選択	３．８４ [μs]

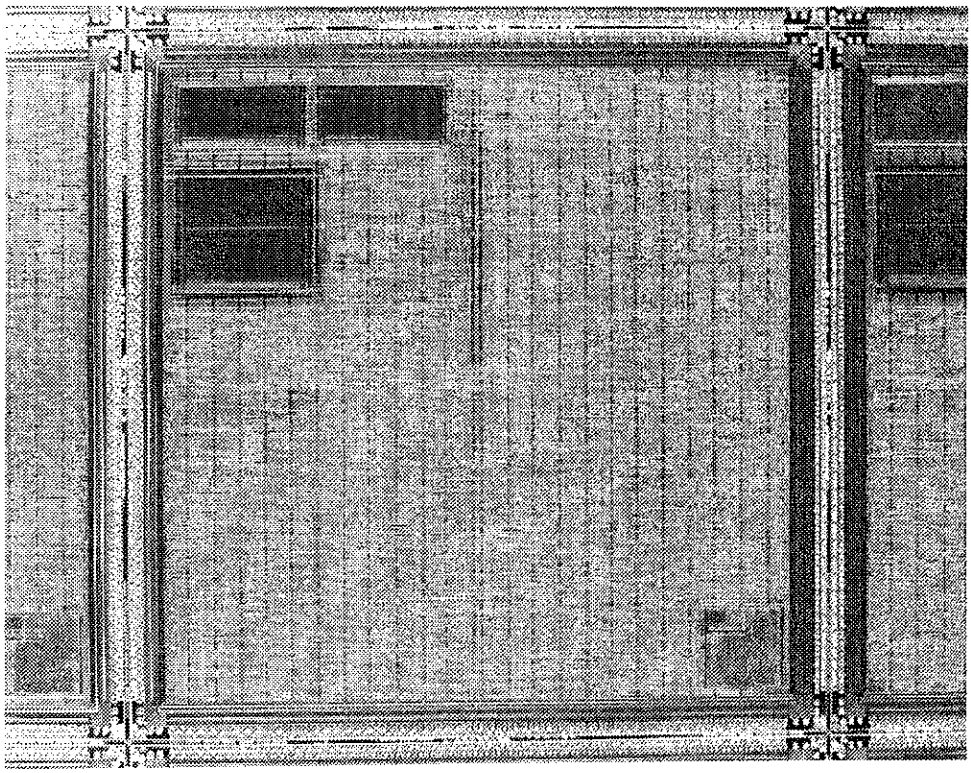

図 4.9: EHW チップ version 1 (EHWv1)

第 5 章

EHW チップ version 2 (EHWv2)

ここでは，１９９９年２月完成予定の EHW チップ version2 (EHWv2) について説明する．以下では，まず，EHWv1 からの改良点を述べ，次に，VLSI へ実装する回路について述べる．

5.1 EHWv1 からの改良点

EHWv2 の EHWv1 からの主な改良点は，

- 回路の小型化
- 適応の高速化
- トレーニングパターンのオンライン書き換え

の３点である．これら３点を実現するために，本研究では，

1. 遺伝子置換型 GA (Gene-Replacement GA; GRGA) による回路合成方式の提案
2. PLA の構造の変更
3. オンライン・トレーニングパターン書換えモードの追加

を行った．

5.1.1 遺伝子置換型 GA (Gene-Replacement GA; GRGA) による回路合成

5.1.1.1 最小項学習法による回路合成の問題点

EHWv1 では，最小項学習法により，回路合成を高速化した．

最小項学習法では，GA による合成に時間のかかる積項，つまり，構成する入力信号の数が大きい積項（例：最小項）を別に合成することで，回路合成を高速化した．

しかしながら最小項学習法では，最小項学習用の PLA が必要であるため，回路が大きくなるという問題がある．さらに，この方式では，最小項学習用 PLA の積項線の数だけしか，正しい出力をする入力パターンの数は増加しない．つまり，EHWv1 では，最小項学習用 PLA の積項線数が６４であるから，最小項学習を使っても，６４個の入力パターンに対する出力が正しくなるだけである．例えば，６章の実験 XX では，１２００個のトレーニングパターンを使って回路を合成しているので，６４個の出力パターンが正しくなっても，５．３％しか認識率が良くならない．

5.1.1.2 GRGA による回路合成の高速化

そこで，EHWv2 では，最小項を合成するための PLA が必要ない方式として，GRGA (Gene Replacement GA) により回路を合成する方式を提案し，実装する．

GRGA による回路合成では，表 1 のようなトレーニングパターンから直接，得ることのできる染色体の情報をつかうことで，回路合成を高速化する．つまり，トレーニングパターンから染色体の一部の候補となるビット列を得ることができ，そのビット列を染色体の一部と置換することで，回路合成を高速化する．この方式のことを，遺伝子置換型 GA (Gene-Replacement GA,GRGA) と呼ぶことにする．

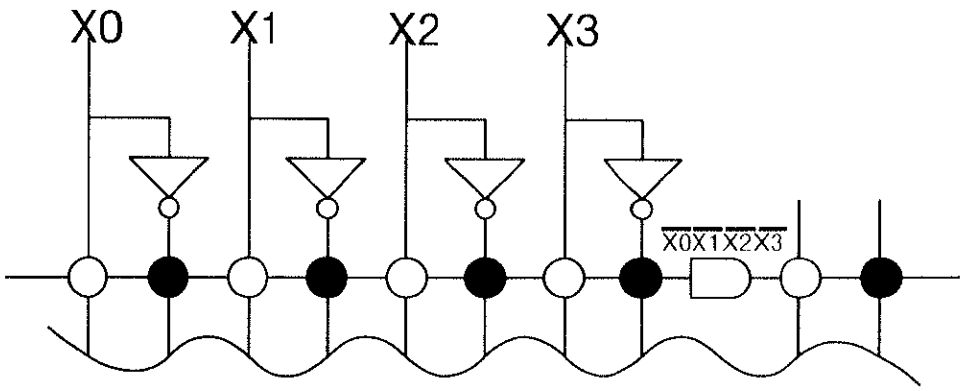

図 5.1: 入力パターン 0000 に対してパターン 01 を出力する PLA のスイッチ設定

以下では，GRGA のしくみを，表 1 の最初のトレーニングパターン（入力パターン 0000 に対して 01 を出力）を例に説明する．まず，入力パターン 0000 に対してのみ，1 を出力する積項（最小項）は $\overline{X0}\overline{X1}\overline{X2}\overline{X3}$ である．そして，PLA の AND アレイと OR アレイのスイッチが図 5.1 のように設定されていれば，その回路は入力パターン 0000 に対して 01 をする．つまり，図のスイッチを指定するビット列（0101010101）を染色体中に持っていれば，PLA の回路は入力パターン 0000 に対して 01 を出力できる．本論文では，このビット列のことを，染色体候補と呼ぶことにする．EHWv 2 では，PLA の入力のビット幅の最大値を 12 ビット，出力のビット幅の最大値を 8 ビットとしたので，PLA の一つの積項線のスイッチを指定するビット列，つまり染色体候補のビット幅は 32 ビット（一つの入力に対して二つのスイッチがあるので，$12 \times 2 + 8 = 32$）である．

GRGA では，この染色体候補を染色体の一部と置換することで，回路合成を高速化する．つまり，あるトレーニングパターンに対する PLA の回路の出力が正しくない場合に，そのトレーニングパターンから得られる染色体候補を，染色体の一部分と置換する．具体的には，図 5.2 にブロック

図を示すハードウェアにより，あるトレーニングデータの入力パターンに対する出力パターンが正しくない場合，そのトレーニングデータから作った染色体候補をレジスタに取り込む．このレジスタは遺伝操作用の回路から参照でき，交叉と突然変異を施した染色体の一部と染色体候補を置換する（図5.3）．

図5.3にブロック図を示すハードウェアでは，３２ビット単位で染色体メモリから読み出した染色体の一部（親１，親２）に対してGA操作を施し，二つの３２ビットのビット列をつくる．そして，これらのビット列と染色体候補をセレクタで選び，子１，子２とする．このセレクタの動作は，一つ前の評価処理で評価値の小さかった方のPLA側のセレクタで，染色体の一部の代わりに染色体候補を選択する．なぜなら，HSP-ERでは，片親として常にエリートを選択するので，前回の評価で評価値の大きかったほうのPLA側には，エリートの染色体情報をたくさん受け継いでいるビット列が来ている可能性が大きいからである．

GRGAは，この染色体候補の置換と，交叉や突然変異などの操作を組み合わせることで，高速に探索する．

5.1.1.3　GRGAによる回路合成シミュレーション

ここでは，GRGAを用いた回路合成の効果を確認するために，基本的な組合せ回路を合成するシミュレーションと，６．２の実験で用いるトレーニングパターンを使った回路合成のシミュレーションを行う．

まず，基本的な組合せ回路の合成シミュレーションでは，GAによる合成に時間がかかることが示されているコンパレータ回路（6ビット入力，1ビット出力）を用いることにする[19]．

この回路合成では，コンパレータ回路の真理値表（表4.2）をトレーニングパターンとして用い，乱数の初期値をかえて１０回シミュレーションを行い，回路合成に要した評価回数の平均をGAだけで合成した場合と，

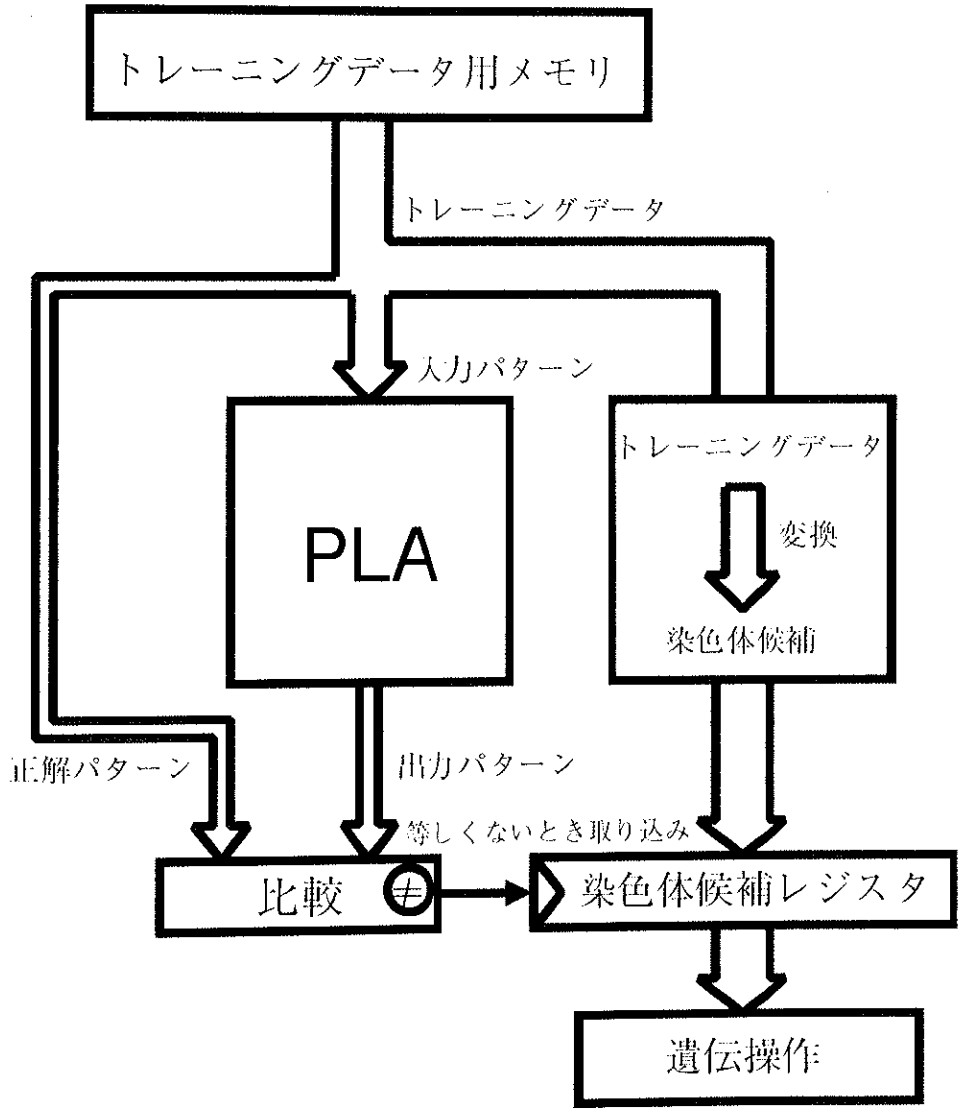

図 5.2: 染色体候補を作るハードウェアのブロック図

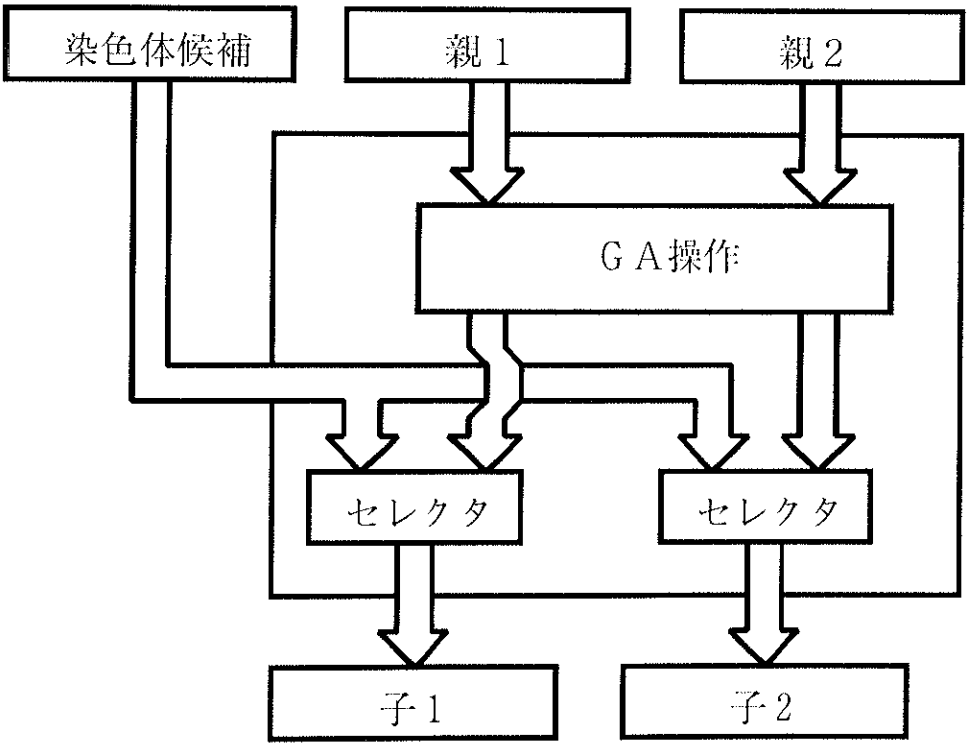

図 5.3: 染色体候補と染色体の一部を置換するハードウェアのブロック図

GRGA で合成した場合で比較する．回路合成に用いる GA としては，3 章で提案した HSP-ER と UC の組合せ (HSP-ER+UC) を用い，GRGA では，染色体候補の置換と HSP-ER+UC を組み合わせる．GA の諸設定は表 5.2 に示したものを用いる．

シミュレーションの結果，表 5.2 に示すように，GA だけで合成した場合は 61873.6 回（10 回平均）の評価が必要であったが，GRGA では，9710 回（10 回平均）の評価だけでコンパレータ回路を合成できた．

表 5.1: シミュレーションの諸設定

入力数 / 出力数	6/1
PLA の積項線数	16
交叉率 (GA)	1.0/ 染色体
突然変異率 (GA)	0.01/ ビット
終了条件 (GA)	正解率 100 %

表 5.2: 実験結果

合成方法	評価回数の平均
GRGA	9710.0
GA のみ	61873.6

つぎに，６．３．２の実験で用いるトレーニングパターンを使った回路合成のシミュレーションでは，EMG 信号の特徴ベクトル（８ビット）を入力パターンとし，その特徴ベクトルのパターン認識回路を合成する．このパターン認識回路の出力は，６章で述べる筋電義手の６つの動作の識別パターン（６ビット）である．トレーニングパターンには，義手の６つの動作それぞれ１０パターン，合計６０パターンを用い，それらに対する PLA の回路の出力パターンの正解率を，その PLA の回路の評価値とする．トレーニングパターンの詳細は６．３．２で述べる．

このパターン認識回路の合成では，各シミュレーション毎にトレーニングパターンを作り直し，回路合成のシミュレーションを１０回行った．そして，GA だけで合成した場合と，GRGA で合成した場合で，回路合成をはじめてから５分間の評価値を，１分毎に比較した．

シミュレーションの結果，図5.4（１０回のシミュレーションの平均）に示すように，回路合成開始から３分までは，GA だけで合成したほうが評価値が良いが，５分後には，GRGA で合成したほうが評価値が良くなっている．この原因は，回路合成の初期段階では，染色体候補の置換のために，交叉による染色体の組み換えが有効に働かず，GRGA の方が探索が遅くなっている．しかしながら，GA による探索が収束しはじめると，GA だけで合成する場合は，４章で述べた最小項などの積項の合成に時間がかかるが，GRGA の場合は，染色体候補の置換の効果で高速に探索できている．

5.1.2　PLA の構造の変更

EHWv 1 では，図4.3 に示すような，入力信号の論理積（積項）の論理和を出力する AND-OR 型の PLA を用いた．しかしながら，この PLA では，積項線が多いため，回路の総ゲート数の約 66.7 % を PLA が占めていた[25]．そこで，EHWv 2 では，PLA の積項線を少なくするために，図5.5 に示すような，入力信号の論理積（積項）の論理和の排他的論理和（XOR）を出力する AND-OR-XOR 型の PLA を用いる．この AND-OR-XOR 型の PLA は，AND-OR 型の PLA よりも少ない積項線で回路を実現可能であることが示されている [92]．[92] では，AND-OR ２段型，AND-XOR ２段型，AND-OR-XOR 3 段型について，1228158 個の 5 入力関数に必要な積項数を求め AND-OR-XOR 型が最も積項数を少なくできることを示している．

この PLA は，入力のビット幅の最大値が１２ビット，出力のビット幅の

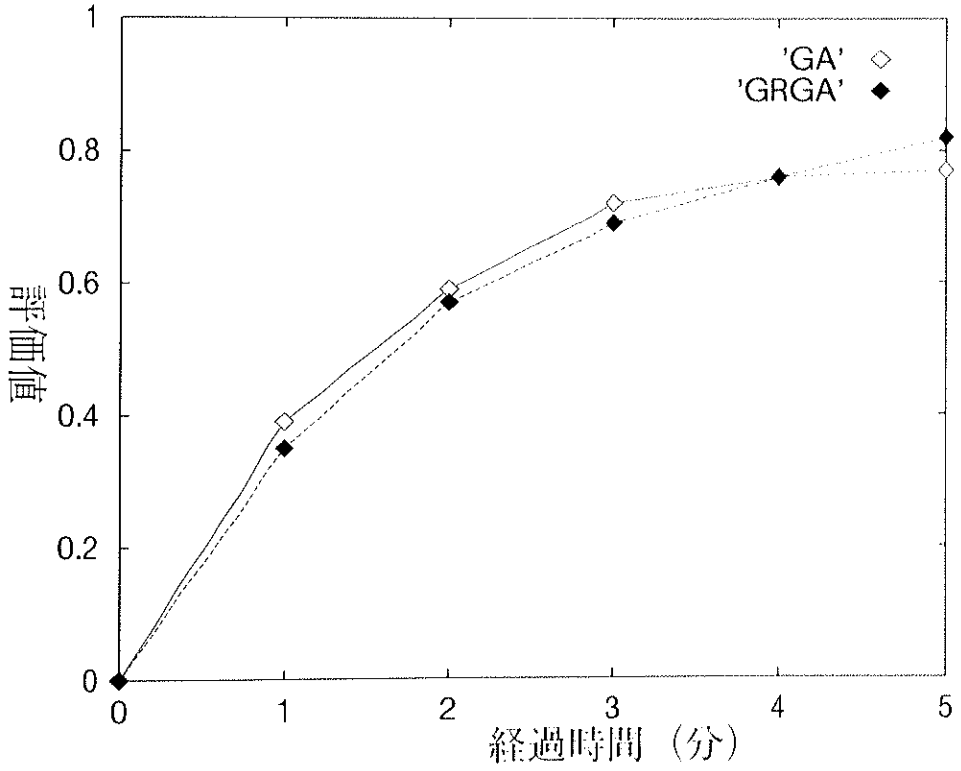

図 5.4: ６．３．２で用いるトレーニングパターンを使った回路合成のシミュレーション結果

最大値が８ビットで，入力のうち４ビットは，出力パターンのフィードバックを選択可能である．積項線の数は全体で３２本で，１６本ずつORをとり，それらのXORを出力する．

5.1.3 オンライン・トレーニングパターン書換えモード

トレーニングパターンは，８ビットの入力パターンと，それに対する８ビットの出力パターンからなので，トレーニングパターンを保存するメモリは，１６ビット×２０４８ワードのメモリを用いる．

６．３．２で述べる方式では，PLAの回路を合成しながらトレーニングパターンを追加する必要があるため，EHWv 2では，PLAの回路を合成中に，トレーニングパターンを書き換えるモードを追加した．具体的には，トレーニングパターンを追加するときは，CPUからオンライントレーニングパターン書換えモード用のレジスタに１をセットする．すると，GA操作を一時停止し，待機状態になる．そこで，CPUを使ってトレーニングパターンを書き換える．トレーニングパターンの書き換えが終了したら，オンライントレーニングパターン書換えモード用のレジスタに０を書き込むことで，GA操作が再開する．

5.2 VLSIへの実装

本研究では，図5.6のブロック図に示す回路を，VLSI (NEC, 0.35μm CMOSセルベースIC, CBC9ファミリ[44], QFP 208ピンパッケージ)に実装する．実装する回路の総ゲート (MOSトランジスタ) 数は231135ゲートである．

この回路のGA操作，評価，選択に要する時間を表5.3に示す．

GA操作は，染色体メモリから３２ビット単位で染色体を読み，それらにGA操作用回路で交叉と突然変異を施し，評価用のPLAに書く操作であ

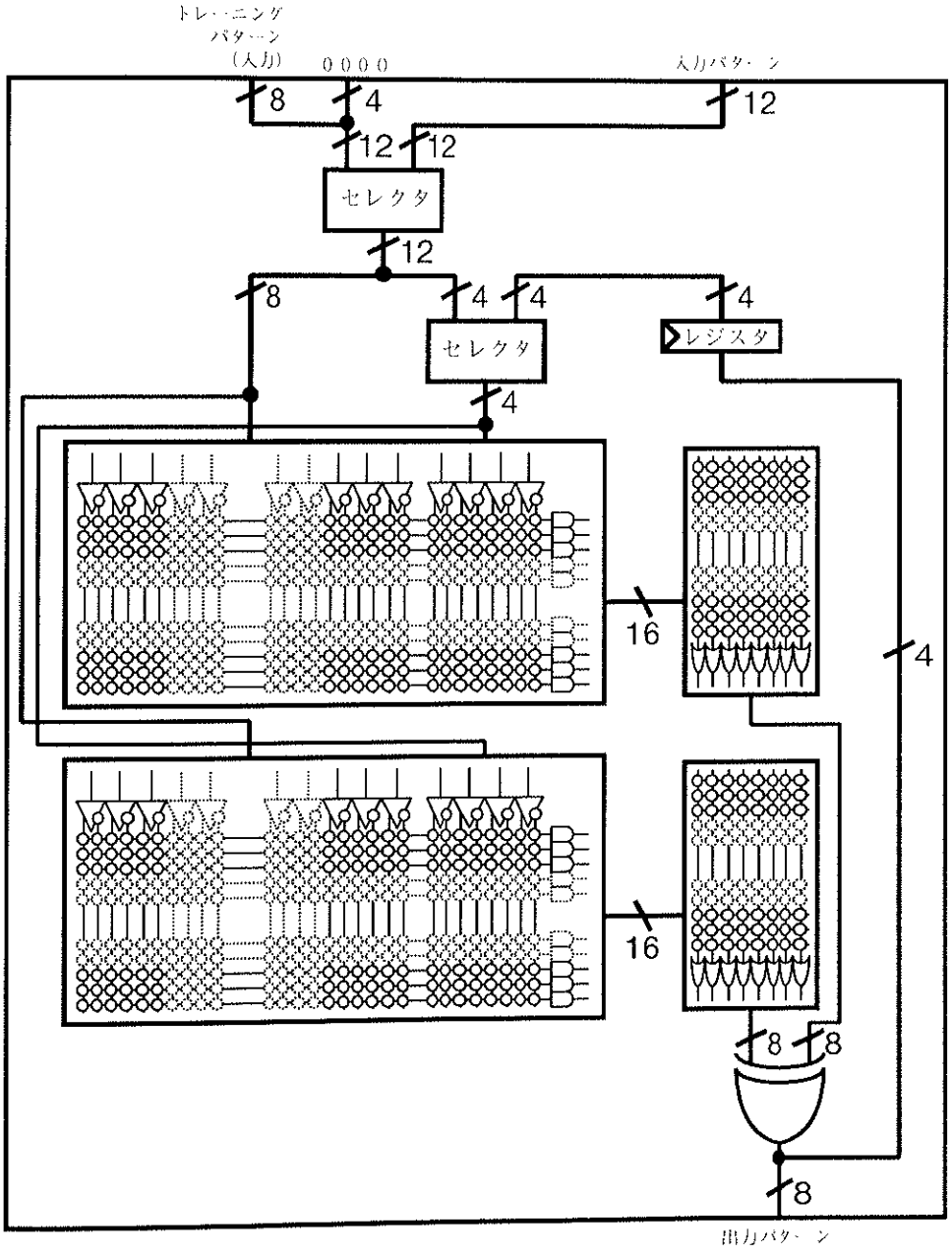

図 5.5: EHWv 2 で採用した AND-OR-XOR 型の PLA

る．染色体長の最大値は１０２４であるから，１回の GA 操作では，これらの操作を３２回繰り返す必要がある．３２ビット単位の染色体の移動には，１回辺り１クロックサイクル必要で，EHWv2 は３３ MHz 動作であるから，１回の GA 操作に要する時間は３０ [ns] ×２×３２＝１９２０ [ns] ＝１．９２ [μs] である．

評価は，トレーニングパターンをメモリから読み，PLA の回路を評価する操作である．評価時間は，トレーニングパターンの数に依存し，例えば，６章の XXX 実験では，２５６個のトレーニングパターンを用いて回路を合成する．この場合，評価に要する時間は，３０ [ns] ×２×２５６＝１５３６０ [ns] ＝１５．３６ [μs] である．

選択は，PLA の回路の評価値が，GA 操作を施す前の染色体の評価値より大きい場合に，染色体を PLA からメモリに戻す操作である．この操作は，GA 操作と逆に染色体を移動するだけであるので，実行時間は，GA 操作と同じである．つまり，１回の選択操作には１．９２ [μs] 必要である．

表 5.3: 各操作の実行時間

GA 操作	１．９２ [μs]
評価 （トレーニングパターン２５６個）	１５．３６ [μs]
選択	１．９２ [μs]

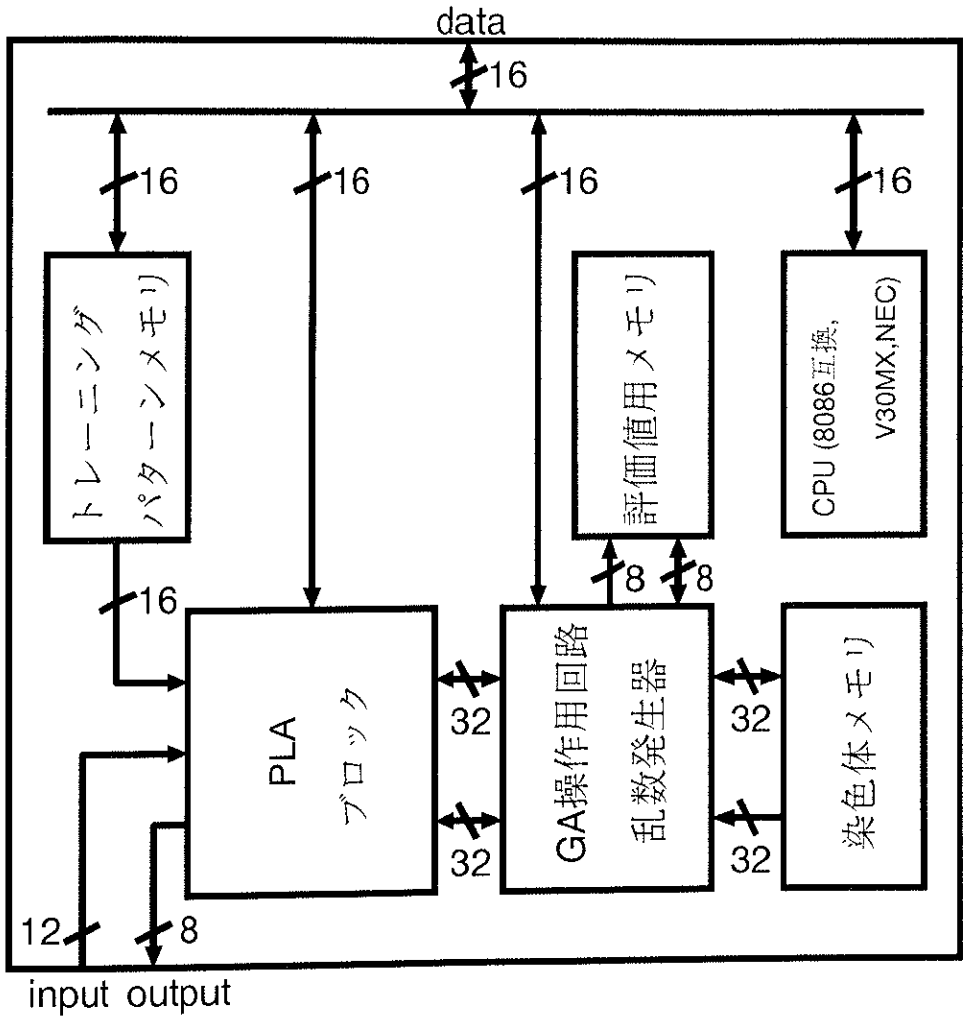

図 5.6: EHW チップ version 2 (EHWv2)

第 6 章

EHW チップの応用 (筋電義手)

本章では，EHW チップの応用の一つとして，筋電義手の制御回路の一部に応用可能であることを示す．具体的には，EMG の特徴ベクトルのパターン認識に EHW チップを用いる．この EMG のパターン認識を行う回路は，筋電義手への応用に留まらず，EMG をインタフェースとする様々な応用に適用可能である．

以下では，まず，EHW チップによる筋電義手の動作決定方法について述べ，次に，EMG の特徴ベクトルのパターン認識を行う回路の合成方法を説明する．そして最後に，EHWv 1，EHWv 2 により，筋電義手の動作を決定するシミュレーションについて述べる．

6.1 EHW チップによる筋電義手の動作決定

6.1.1 EMG の特徴ベクトルのパターン認識による筋電義手の動作決定

多自由度の筋電義手の動作を決定するには，図 6.1 に示すように，センサーで測定された EMG 信号から特徴ベクトルを抽出し，その特徴ベクトルのパターン認識により義手の動作を決定する．本研究では，この特徴ベクトルのパターン認識に EHW チップを応用する．

筋電義手としては，図 6.2 に示す今仙技術研究所の 3 自由度義手（ワイムハンド）を用いる．この義手は，”回内”，”回外”，”屈曲”，”伸展”，”握る”，”開く”の 6 動作が可能である．

以下では，EMG の測定方法と，EMG の特徴ベクトルの抽出方法を説明する．

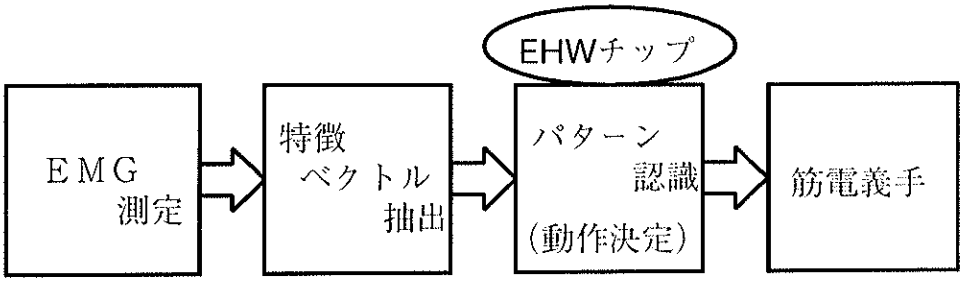

図 6.1: EMG の特徴ベクトルのパターン認識による義手の動作決定

6.1.2 EMG の測定方法

図 6.3 に示すように，肘から手先方向に約 3 cm の内側と外側に付けた二つの EMG センサー（delsys 製 [93]）で 2 チャンネルの EMG を測定する．

EMG の測定においては，一対の表面電極を用いて双極誘導し，差動増幅することで，ノイズの影響を少なくする．

例えば，図 6.4 の例では，二つの電極で観測される EMG 信号（X+n, Y+n) には，共通のノイズ成分 (n) がのっており，これらを差動増幅することで，ノイズ成分を除去する [93]．

測定した EMG は，生体アンプ（delsys 製）を使って 10000 倍に増幅し，A/D 変換ボード（KYOUWA 製）を使ってデジタル信号（12 ビット）に変換してパソコンにとりこむ．A/D 変換ボードのサンプリングレートは 1600Hz である．

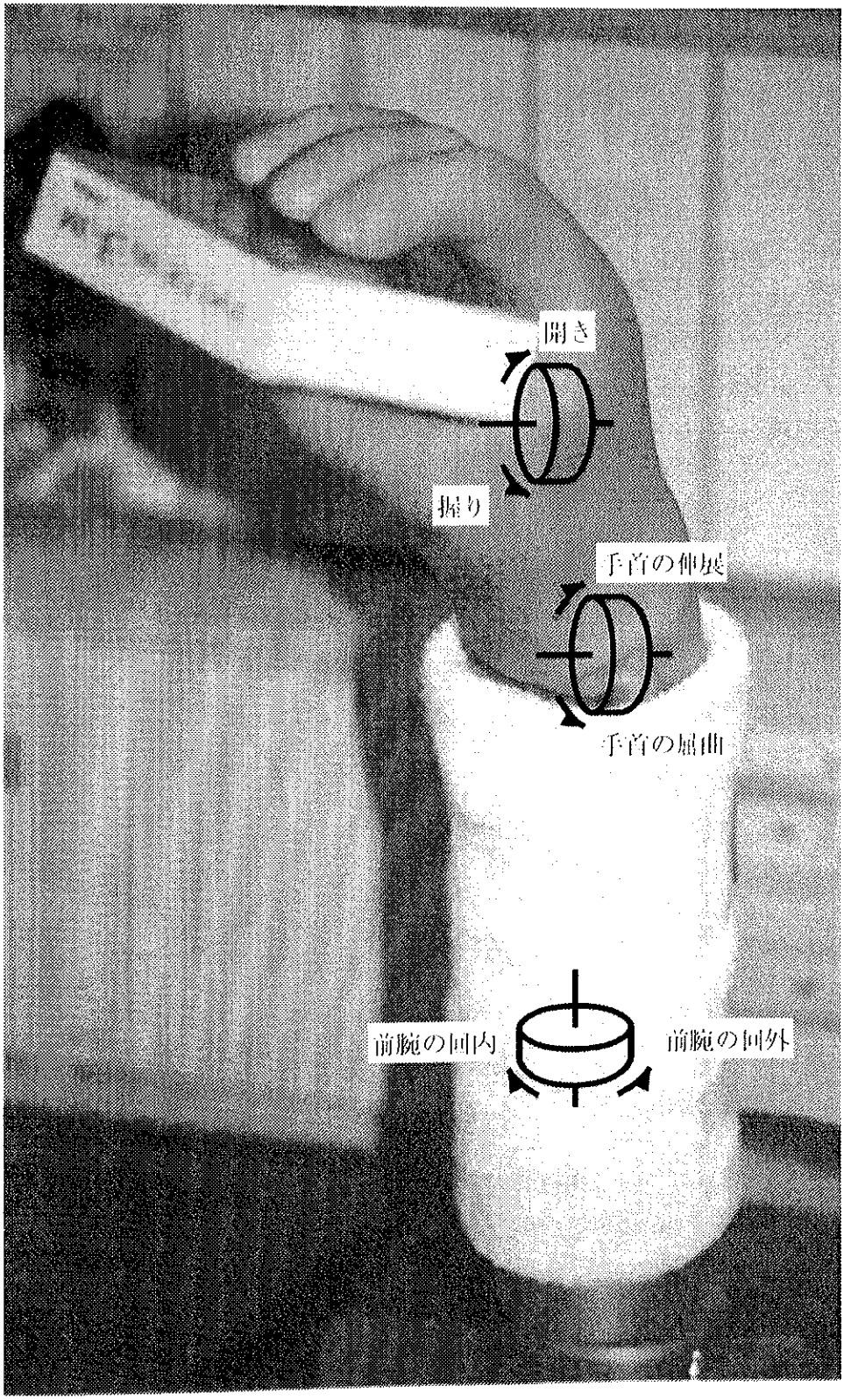

図 6.2: 本研究で用いた筋電義手　(今仙技術研究所製)

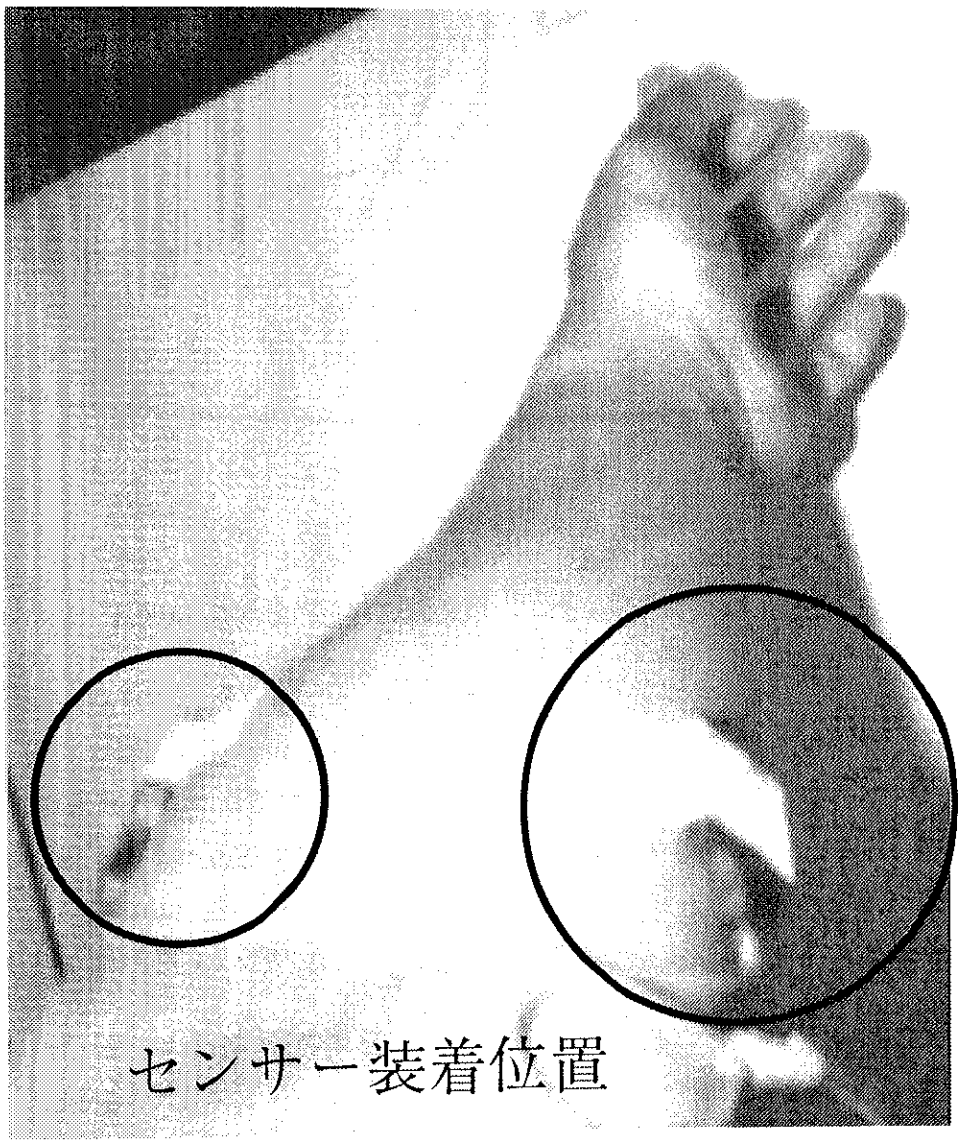

図 6.3: センサーの装着位置

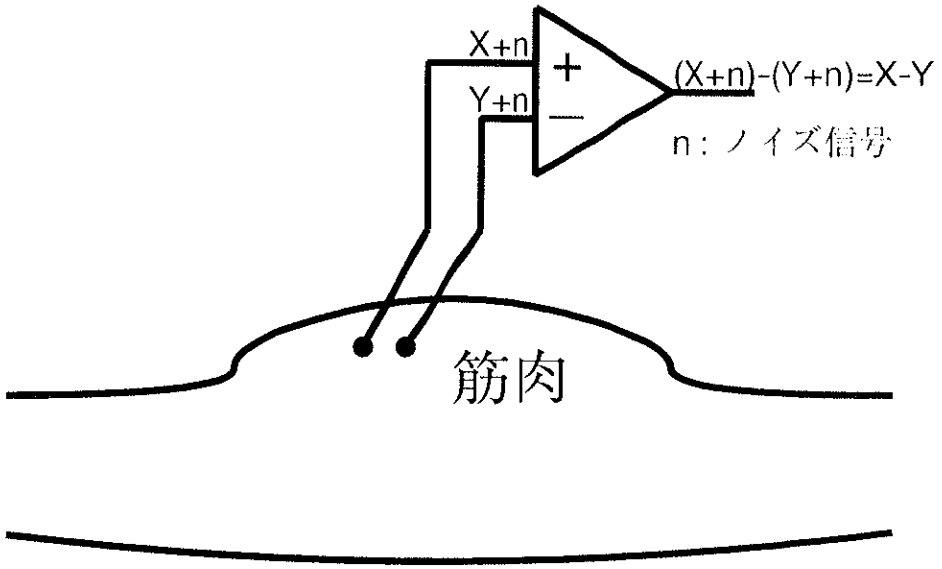

図 6.4: EMG センサーの仕組み（差動増幅）

6.1.3　EMG の特徴ベクトル抽出

EMG の特徴ベクトルとしては，EMG の周波数成分を用いる方式と，EMG の振幅成分を用いる方式の二つがある [34]．ここでは，EMG の周波数成分により義手の動作を決定する方式 [94][39][25] と，EMG の振幅成分により義手の動作を決定する方式 [26] を説明する．

6.1.3.1　EMG の周波数成分の分類による動作決定

筋肉の動きは運動単位の収縮によって起こるが，運動単位は，全てが同時に収縮するのではなく，いくつか運動単位が交代で収縮する．そのため，筋繊維の収縮によって起こる活動電位は，ある周波数を持つ．この活動電位は，体表面まで伝わる間に他の筋肉や脂肪の影響で周波数特性が変化するので，体表面で観測される EMG は，動作した筋肉に応じて周波数特性が異なる [29]．

ここでは，この周波数特性の違いを，義手の動作決定に用いる方式を説明する．

センサーで測定したEMGは，前節で述べたように，1600Hzでパソコンに取り込まれる．そして，その信号にFFT (Fast Fourier Transform) をかけて，周波数成分に分解する．

FFTでは，6.25Hz間隔で256個の周波数スペクトルを求め，各チャンネル毎に3つの特徴的な周波数スペクトルを選ぶ[39]．EMGは0 ～約500Hzが有効な周波数であるので[93]，6.3.1のオフライン実験では，表6.1に示す周波数を用いた．

表 6.1: オフライン実験で採用した周波数

チャンネル 1	87.50, 168.75, 250.00 (Hz)
チャンネル 2	100.00, 181.25, 262.00 (Hz)

そして，これら6つの周波数スペクトルをそれぞれ0 ～ 15の整数に正規化して，4ビットの二進ビット列に符号化する．この（1ビット）×（3周波数成分）×（2チャンネル）＝24ビットの二進ビット列をEHWチップの入力パターンとし，それらのパターン認識により，義手の動作を決定する．

6.1.3.2 EMGの振幅成分の分類による動作決定

ここでは，EMGの振幅成分の特徴により，義手の動作を決定する方式を説明する．

EMG信号をある時間幅で積分すると，そのときの筋肉の収縮に応じて，ほぼ一定の値となる[95]．

そこで，パソコンにとりこんだ2チャンネルのEMGをそれぞれ640ms毎に積分する．次に，積分した値を0 ～ 15までの整数に正規化し，それ

ぞれ 4 ビットの二進ビット列に符号化する．この正規化処理では，値の小さい信号の解像度を高くするために，対数変換を行った [26]．例えば，図 6.5 の例では，回内，回外，握る，開くに対応する EMG の積分値が近いため，図 6.6 のように対数変換する．こうすることによって，各動作の認識が容易になる．

この（4 ビット）×（2 チャンネル）＝8 ビットの二進ビット列を EHW チップの入力パターンとし，それらのパターン認識により義手の動作を決定する．

この EMG の周波数成分を用いる方式では，周波数成分を求めるための FFT などの計算が必要なく，より小さく実装できるという利点がある．

6.2 パターン認識回路の合成方法

6.2.1 トレーニングパターン

ここでは，EHW チップの回路を合成するために必要なトレーニングパターンの作成方法を説明する．トレーニングパターンとしては，義手の 6 動作それぞれに対応する EMG の特徴ベクトル（周波数成分，または，振幅成分）を用いる．

従来のニューラルネットワークによる方式では，ネットワークの学習を始める前に，あらかじめトレーニングパターンを作成し，その後，ネットワークの学習を行うオフライン方式が用いられてきた [57][96][36][94][97]．本研究でも，まず，このオフライン方式により EHW チップの回路を合成する (実験 1 ; 6.3.1)．

しかしながら，この方式では，センサーの位置のずれなど，トレーニングパターン作成後の EMG の特徴の変化により，合成した回路が希望どおりに動作しないことがあり，この場合は，トレーニングパターンを作りなおし，EHW チップの回路を最初から合成しなければならない．同様の問題

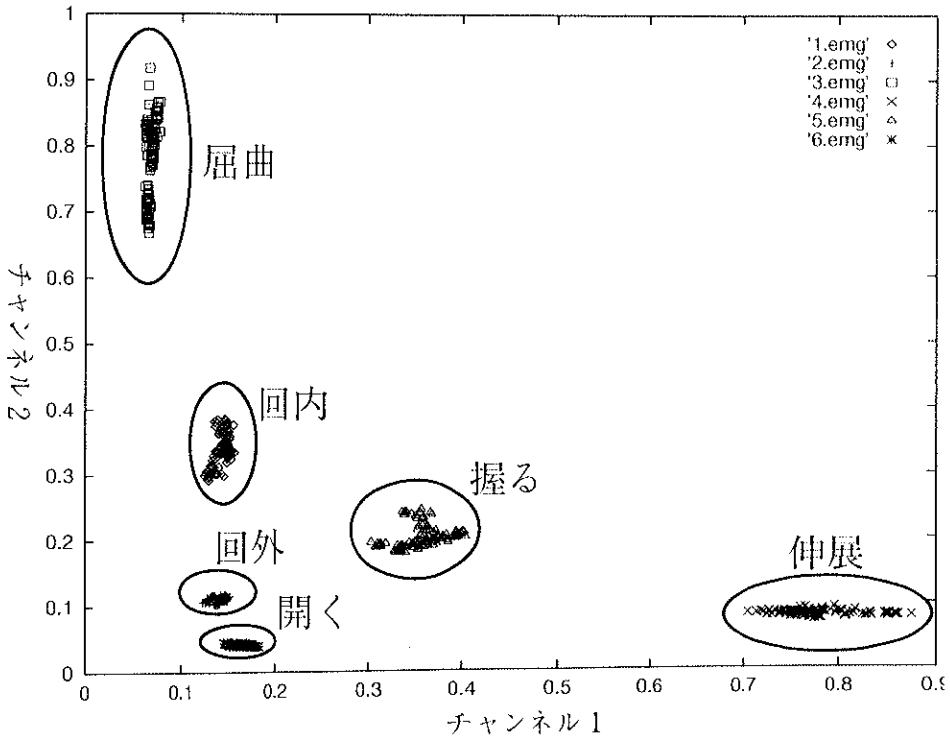

図 6.5: EMG の積分値；チャンネル 1 を X 軸，チャンネル 2 を Y 軸としてプロットしたもの

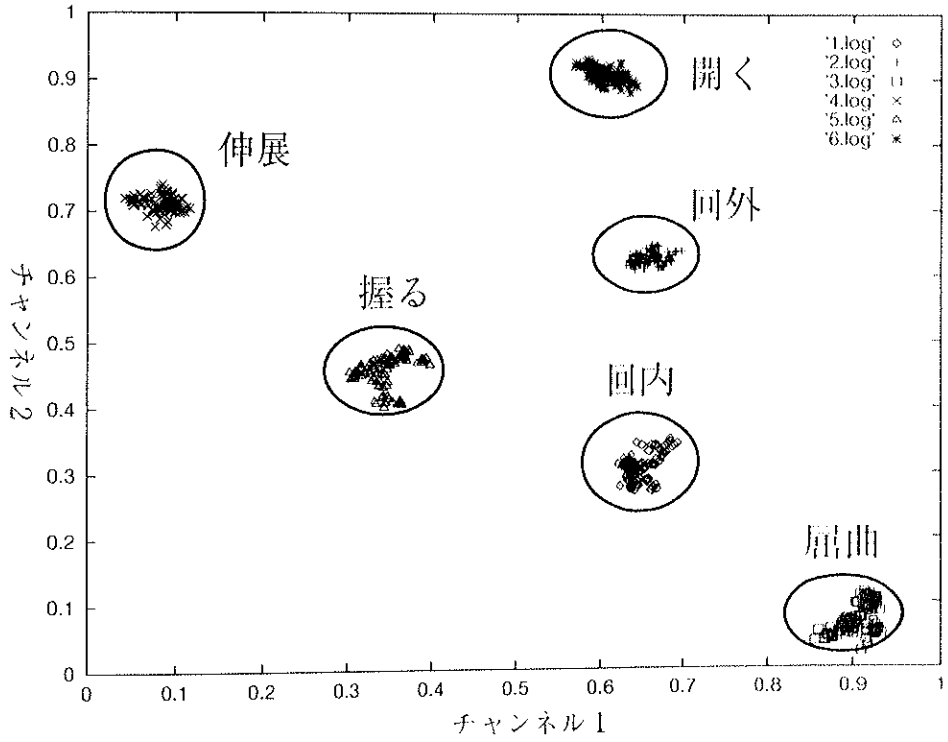

図 6.6: 対数変換した EMG の積分値；チャンネル 1 を X 軸，チャンネル 2 を Y 軸としてプロットしたもの

がニューラルネットワークによる方式でも指摘されており，その解決方法として，義手を動作させ，その動きを見ながらトレーニングパターンを作る方式が提案されている [37][38][39][97]．この方式のことを，オンライン方式と呼ぶことにする．6.3.2 の実験では，このオンライン方式によりトレーニングパターンを作成し，回路を合成する．

以下では，オフライン方式，オンライン方式それぞれについて，トレーニングデータの作成方法を説明する．ただし，オフライン方式では，EMG の周波数成分のパターン認識により義手の動作を決定し，オンライン方式では，より小さく実装するために，EMG の振幅成分のパターン認識により義手の動作を決定する．

6.2.2 オフライン方式

6.2.2.1 実験装置

オフライン方式では，図 6.7 に示すように，障害のない手にデータグローブと筋電センサをつけることで，トレーニングパターンを作成する [94][97][25]．つまり，データグローブで手の動作を測定し，間接の角速度に応じて 6 つの動作を識別する．この方式では，同じ人であれば，左右の手の EMG の特徴が似ていることを利用している．

このオフライン方式の利点は，データグローブを用いるため，手の動作途中（非定常状態，例：握る途中など）の EMG をトレーニングパターンとして使うことができる．定常状態（例：手を握った状態）をトレーニングパターンとする場合（6.3.2 のオンライン実験），筋肉が定常状態になってから，義手の動作が開始される．例えば，握る動作の場合，筋肉が握る動作に対応する収縮状態になってから，義手が動作する．そのため，義手の使用者が義手を動かそうと思ってから，実際に義手が動き出すまでに，遅れが生じてしまう．

それに対して非定常状態の EMG で合成したパターン認識回路は，例えば，握る途中の筋肉の動きから生じる EMG を認識するので，ある義手の動作を意図して筋肉を動かしてから，実際に義手が動き出すまでの遅れが小さい．

6.2.2.2 トレーニングパターン作成

オフライン方式では，次に示す手順でトレーニングパターンを作成する．

1. 障害のない手にデータグローブを装着する．
2. 間接の角速度に応じて，その時の動作を識別する（6 動作）．
3. ある動作に認識されている間の入力パターンは，全てその動作に対するトレーニングパターンとして保存する．
4. どの動作とも識別されなかった入力パターンを，NO ACTION に対するトレーニングデータとして保存する．

このオフライン方式による以下の実験 (6.3.1) では，EMG の特徴ベクトルには周波数成分を用いる．具体的には表 6.1 の示した周波数を用いる．

6.2.3 オンライン方式

6.2.3.1 実験装置

オフライン方式では，トレーニングパターンの作成にデータグローブが必要であるため，両手に障害のある人には使うことができない．さらに，切断後の筋肉の縮退のために左右の EMG の特徴が異なってしまうことがあり，この場合も，オフライン方式は使えない．

それに対してオンライン方式では，オフライン方式のようにデータグローブは必要なく，その代わりに，キーボード入力，または，音声認識によ

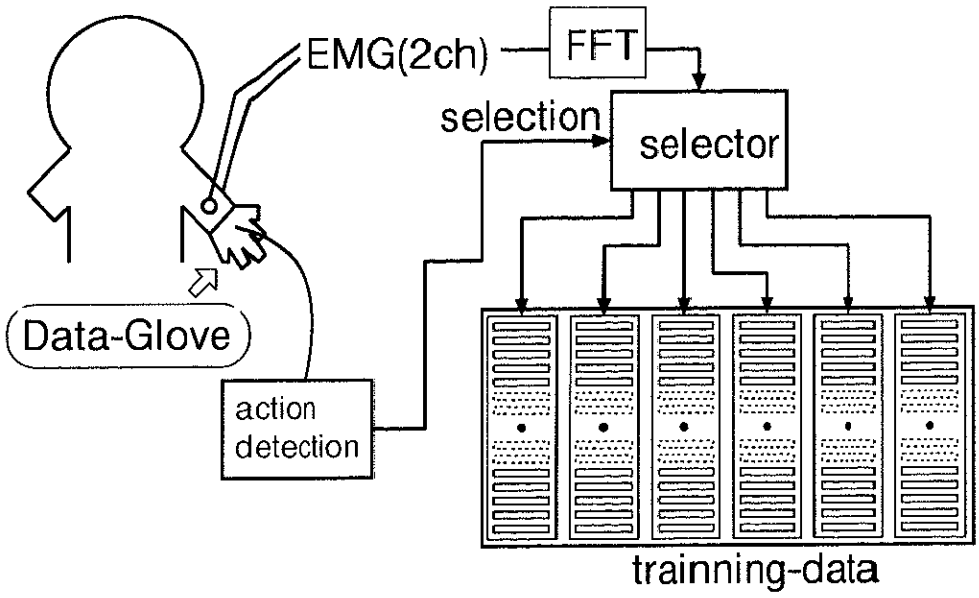

図 6.7: トレーニングパターンの作り方（オフライン）

り，トレーニングデータを作成する．そのため，両手に障害のある人や，左右の EMG の特徴が異なる人でも使うことができる．音声認識は，両手に障害のある人のようにキーボード操作が困難な人でも扱えるようにするために採用した．この音声認識処理には，パソコン用の音声認識ボード（スカラベ製）を用いた．

6.2.3.2 トレーニングパターン作成

トレーニングパターンは，切断前の手の動きを思い出しながら筋肉を動かすことで作成する．例えば，「握る」動作に対するトレーニングパターンを作る場合，「握る」動作を思い出しながら筋肉を動かし，そのとき観測される EMG を「握る」動作に対するトレーニングパターンとする．具体的には，このとき，キーボードの「握る」動作に対応するキーを押すか，音声認識の場合は，「握る」と発声することで，そのとき観測された EMG が，「握る」動作に対するトレーニングパターンとして保存され

る.

このオンライン方式では, EMG の特徴ベクトルには振幅成分を用いる. さらに, 合成した回路の動作が希望どおりでないときは, トレーニングパターンを追加して回路を再構成することができる. EHWv 1 には, このトレーニングパターンを追加する機能はないが, EHWv 2 を用いることでこの機能を実現可能である.

6.3 実験

6.3.1 オフライン実験

ここでは, 6.2.1 のオフライン方式により作成したトレーニングパターンを用いて EHW チップの PLA の回路を合成する方式 [25] について説明する.

6.3.1.1 実験方法

オフライン実験では, 義手の３つの自由度それぞれについて, 別のパターン認識回路を合成する. つまり, {" 回内", " 回外"}, {" 屈曲", " 伸展"}, {" 握る", " 開く"} の組みそれぞれについて, パターン認識回路を合成する.

1. 6.2.1 で作成したトレーニングパターンのうち, 各動作毎２００パターンをトレーニングパターンとする.
2. NO ACTION に対する EMG パターン８００個と, 各動作の組に対するトレーニングデータ (２００個×2 動作) を使って, 回路を合成する (評価回数１００００回まで).
3. ３つの自由度に対して, 繰り返す.

1. 6.2.1 で作成したトレーニングパターンから，各動作毎に９８００パターンをテストパターンとして選び （９８００パターン×６動作＝５８８００パターン），それらに対する，合成したパターン認識回路の認識率を求める．

ここで求めた，テストパターンに対する合成結果のパターン認識回路の認識率を，同じテストパターンで学習したニュラルネットワークの認識率と比較した．

6.3.1.2 実験結果

オフライン方式で作成したトレーニングパターンで合成したパターン認識回路と，同じトレーニングパターンで学習したニューラルネットワークの認識率を，図 6.8 に示す．

平均認識率は，合成したパターン認識回路が８５．０％で，学習結果のニューラルネットワークが８０．０％であった．[1]この結果から，EHW チップによるパターン認識によっても，ニューラルネットワークと同程度の認識率を実現可能であることが確認できた．

6.3.2 オンライン実験

ここでは，6.2.2 のオンライン方式によりトレーニングパターンを作成し，EHW チップの PLA の回路を合成する方式 [26] について説明する．

6.3.2.1 実験方法

オフライン実験では，各自由度ごとに別々のパターン認識回路を合成したが，オンライン実験では，一つのパターン認識回路だけで 3 自由度全ての

[1][97] でも同様の方式でニューラルネットワークを学習しているが，この学習結果のニューラルネットワークの平均認識率は８１．２％であった．

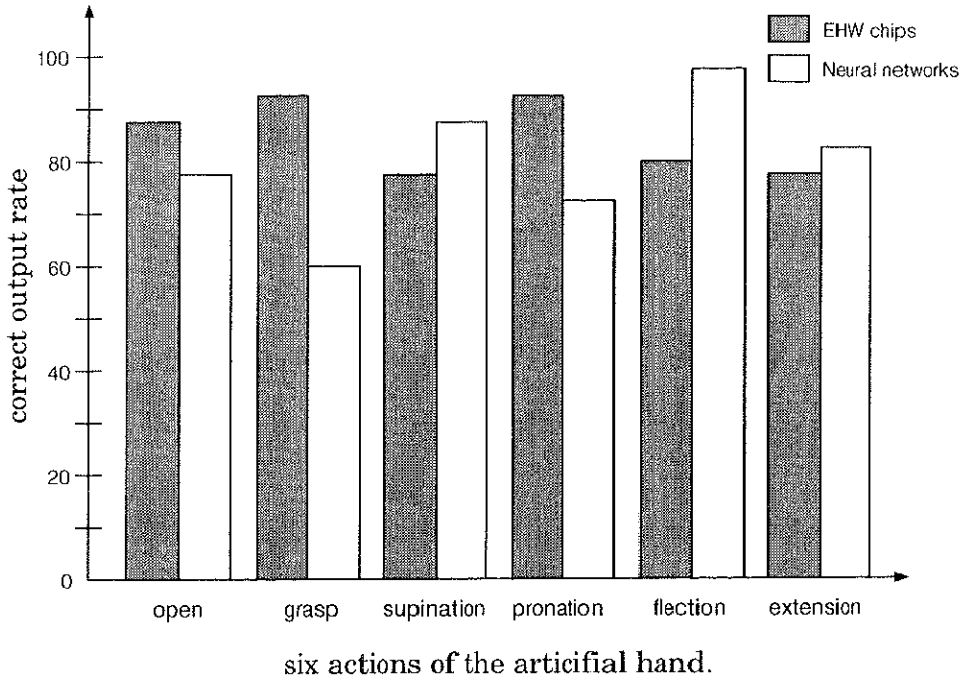

図 6.8: 合成結果の回路と学習後のニューラルネットワークによるテストパターンの認識率

動作を決定する．以下では，3 人の被験者（健常者）の表面筋電位について，義手の 6 つの動作を分類するパターン認識回路を，EHW チップで実現可能であることをシミュレーションで実証する．

そこで，まず，各動作毎に 10 個ずつ（合計 60 個）のトレーニングパターンを作る．つまり，トレーニングパターンを作りたい動作をしている状態で，その動作に対応する単語を発声するか，対応するキーを押す．例えば，握る動作に対するトレーニングパターンを作る場合，『握る』と発声するか，『握る』動作に対応するキーを押す．こうすることによって，トレーニングパターンが 5 個ずつとりこまれる．

そして，このトレーニングパターンを使って適応させ，5 分後にパターンの認識率を求める．パターンの認識率としては，各動作毎に 10 秒間その動作を維持し，そのときの EMG の特徴ベクトルの認識率を用いる．例え

ば『握る』動作の場合，EMG センサーを付けた手を握った状態で 10 秒間維持し，その間の表面筋電位の認識率を計算する．

次に，認識率が悪かった 3 つの動作について，さらに 10 個ずつトレーニングデータを追加して回路を再構成する．そして，トレーニングパターンを追加して 5 分後（実験開始 10 分後）に，もう一度認識率を求め，トレーニングパターンを追加した効果を確認する．

この実験に用いた EHW シミュレータの PLA は，8 ビット入力，6 ビット出力で，積項線数を 32 とした．回路を適応させるための GA には，本稿で提案した GRGA を用い，その諸設定は，個体数 32，交叉率 1.0，突然変異率 0.01 とした．

6.3.2.2 実験結果

表 6.2 に，各被験者に対する，適応開始 5 分後の表面筋電位のパターンの認識率と，トレーニングパターンを追加し再適応させた後（10 分後）の認識率を示す．

被験者 1 の場合，5 分後の平均認識率が 55 % で，『回内』と『握る』と『開く』の認識率が悪い（表の下線を付けた数字）．そこで，それらの動作について，それぞれ 10 個ずつトレーニングパターンを追加して，さらに適応させる．そして，さらに 5 分後（実験開始から 10 分後）の認識率を求めると，平均認識率が 88 % まで良くなっており，トレーニングパターンを追加した 3 つの動作それぞれの認識率も良くなっているのがわかる．被験者 2 と 3 の場合も，トレーニングパターンを追加することで平均認識率が良くなっているのがわかる．

[97] でも，同様の方式でトレーニングパターンを作成し，それらを使ってニューラルネットワークを学習させる実験が行われている．その実験では，オンライン方式により徐々にトレーニングパターンを追加し，３０分

後に各動作ごとの３秒間のパターン認識率を求めている．その結果，平均パターン認識率は８１．５％であった．

このように，オンライン方式によりパターン認識回路を合成しても，ニューラルネットワークと同程度の認識率を実現可能である．

表 6.2: 表面筋電位の認識率

		回内 (%)	回外 (%)	屈曲 (%)	伸展 (%)	握る (%)	開く (%)	平均 (%)
被験者 1	5 分後	0	74	100	68	53	35	55
	10 分後	44	85	100	100	96	100	88
被験者 2	5 分後	99	74	100	85	62	24	74
	10 分後	84	100	100	85	57	89	86
被験者 3	5 分後	100	0	100	100	0	50	58
	10 分後	94	32	63	100	72	63	71

第 7 章

結論

7.1 本論文のまとめ

本論文では，進化型ハードウェアの設計と応用に関する研究と題し，まず，進化型ハードウェアの中心となるGA操作用の回路を設計した．この設計にあたっては，ハードウェアへの実装という観点からGAの各操作を考察し，最もハードウェアへの実装に適したGA操作の組合せとして，HSP-ERと一様交叉の組合せを提案し，これらを高速に実行するハードウェアをFPGAに実装することでその基本動作を確認した．

次に，このGA操作用の回路と再構成可能なハードウェアなどを1チップ化したEHWチップを設計し，VLSIに実装した．このEHWチップには，1997年から1998年にかけて設計したEHWv 1と，現在設計中のEHWv 2がある．

EHWv 1の設計では，チップ上の再構成可能ハードウェアの回路合成を高速化するために，回路合成に時間のかかる解析結果から，最小項学習法を提案した．この最小項学習法では，最小項に代表されるGAによる合成に時間のかかる積項を別に合成することで，回路合成を高速化した．

EHWv 2の設計では，より小さく実装するために，最小項学習法の代わりに，GRGAにより回路を合成する方式を提案した．最小項学習法では，最小項を合成するためのPLAが必要であったが，このGRGAによ

る回路合成ではその PLA が必要ないため，より小さく実装可能である．さらに，EHWv 1 では AND-OR 型の PLA を採用していたが，EHWv 2 では同じ積項線数でより大きな回路を実装可能な AND-OR-XOR 型の PLA を採用した．

これらの EHW チップの応用の一つとして，本論文では，筋電義手の制御回路の一部に応用可能であることを示した．つまり，EMG の特徴ベクトルのパターン認識回路に EHW チップを応用することで，障害者個人個人の EMG の特徴に最適なパターン認識回路を，適応的に合成可能である．

この EMG のパターン認識回路の合成では，回路合成に必要なトレーニングパターンの作成と回路合成を別々に行うオフライン方式と，回路を合成しながらトレーニングパターンを作成するオンライン方式により，被験者の EMG の特徴に最適なパターン認識回路を合成した．合成した回路のパターン認識能力は，オフライン方式，オンライン方式ともに，同様の方式でトレーニングパターンを作成し，それらを使って学習したニューラルネットワークと同程度のパターン認識が可能であった．

この EHW チップによる EMG のパターン認識は，筋電義手への応用に留まらず，EMG をインタフェースとする様々な問題への応用が可能である．

7.2 今後の課題

7.2.1 EHW チップ

EHW チップに関する今後の課題は，次の 2 点である．

1. 染色体メモリの大きさ
2. 再構成可能なハードウェア単位
3. トレーニングパターンを使わない回路合成

7.2.1.1 染色体メモリの大きさ

本研究で設計したGA操作用回路では，個体数が小さくても効率良く探索可能なERとその改良版のHSP-ERを用いることで，小さいメモリに染色体を保存した．しかしながら，ERやHSP-ERを用いても染色体メモリは大きく，例えばEHWV 2では，回路の約３５．４％を染色体メモリが占めている．

この染色体メモリを小さくする，あるいは，なくすために，PLAの各積項を指定するビット列を一つの染色体と見なす方式や，GA以外の探索手法の適用を検討中である．

7.2.1.2 再構成可能なハードウェア単位

本研究では，再構成可能ハードウェアとして，ANDゲートやORゲート単位で回路を変更可能なPLAを用いた．EHWに関する研究は，このゲートレベルEHWの他に，算術関数単位で変更可能な関数レベルEHWや，アナログ素子レベルで変更可能なアナログレベルEHWがある．

再構成可能なハードウェア単位としては，これらの他にも，FPGAで採用されているALUやルックアップテーブルからなる基本的な機能ブロック単位で変更するものなど，様々なものが考えられる．EHWチップを応用した筋電義手の動作認識率をあげるために，よりパターン認識向きのハードウェア単位で回路を変更可能なEHWを検討中である．

7.2.1.3 トレーニングパターンを使わない回路合成

本研究では，PLAの回路がどれだけ目的にあっているかを，トレーニングデータを使って評価した．

回路の評価方法としては，この他に，EHWを組み込んだシステムの動作を評価する場合がある．例えば，電子印刷器（参考文献）の例では，電子

印刷器のデータ圧縮用のハードウェアの一部に EHW を応用しており，このハードウェアのデータ圧縮能力を，EHW の回路の評価に使っている．

本研究では，トレーニングパターンを使った場合の高速回路合成方式を提案したが，トレーニングパターンを使わない場合の高速回路合成方式についても検討中である．

7.2.2 筋電義手

7.2.2.1 認識率を挙げるための課題

6．3．2 の方式では，EMG の振幅成分のパターン認識により義手の動作を決定するので，トレーニングパターンを作ったときと異なる力で筋肉を収縮させると，希望した動作とは異なる動作として認識されることがある．つまり，この方式では，トレーニングパターンを作ったときと同じくらいの力で筋肉を収縮させる必要がある．

健常者の場合は，自分の手の動きを見ることで筋肉の動きを知ることができるので，トレーニングパターンを作ったときと同じ筋肉の収縮を行うことができる．

それに対して，上肢を切断した障害者の場合は手の動きを見ることができないので，切断前の記憶を頼りに筋肉を動かすことで，トレーニングパターンを作らなければならない．そのため，切断前の記憶が曖昧なために，自分が意図した通りに筋肉を動かすことができないことがある．例えば，「握る」という動作をしているつもりが，実際には，「屈曲」の動作に対する筋肉の動きをしてしまうことがある．例えば，ある障害者の方の場合，オットボック社の 1 自由度 (握る，開く) 筋電義手を使っており，「屈曲」と「伸展」の筋肉の動きを使って，義手の「握る」，「開く」の動作を行っていた．そのため，切断前の「握る」「開く」の動作に対する筋肉の動きを忘れてしまっており，これらの動作に対するトレーニングパターンを作ることが困難であった．

この問題の解決のために，EMGの変化(筋肉の動き)を視覚的に知ることのできるインタフェースを開発中である．このインタフェースは，測定した2チャンネルのEMGの積分値をそれぞれ2次元のグラフ上の点としてコンピューターのディスプレーに表示するので，この点の位置と動きから，筋肉の収縮具合を知ることができる．つまり，切断後の速い時期に，このインタフェースを使うことで，切断前の筋肉の動かし方を忘れないようにすることが可能である．

さらに，先天的に手に障害のある人にこのインタフェースを使ってもらうことによって，多自由度の義手を使えるようにするための筋肉の動かし方の訓練が可能である．

7.2.2.2 実装に関する課題

本論文では，EMGの特徴ベクトルのパターン認識に，コンパクトに実装可能なEHWチップを応用した．しかしながら筋電義手には，EMGの特徴ベクトルのパターン認識の他に，A/D，D/A変換器や，EMGの特徴ベクトルを抽出するハードウェアが必要である．

今後は，A/D変換器やEMGの特徴ベクトル抽出用ハードウェアなどもEHWチップ内部に実装することで，より，筋電義手内部に実装しやすいEHWチップの開発を目指す．

第 8 章

謝辞

星野力教授には，大学4年の研究室配属時よりたくさんのご指導をいただき，さらに，電子技術総合研究所において実習生として研究する機会を与えてくださったことに感謝します．

樋口哲也博士には，電子技術総合研究所において，進化型ハードウェアというたいへん興味深いテーマを研究する機会を与えてくださり，さらに，研究者としての基礎を指導していただいたことに感謝します．

本研究の一部は通産省リアルワールドコンピューティングプロジェクトにおいて行われており，島田潤一ＴＲＣ研究所長、大津展之ＲＷＩセンター長に感謝します．

ＮＥＣ　Ｃ＆Ｃメディア研究所の梶原信樹さんには，EHWv 1，EHWv 2の実装でお世話になりました．

電子技術総合研究所の岩田昌也博士には，EHWv 2の設計でお世話になっております．

電子技術総合研究所の坂無英徳博士には，本研究についてたくさんの議論をさせていただいただけでなく，さまざまなアドバイスをいただきました．

筑波大学構造工学系の丸山勉助教授には，たくさんのアドバイスをいただきました．

東京大学大学院の村川正宏君には，日頃のさまざまな議論だけでなく，計算機環境の整備などお世話になりました．

北海道大学大学院の西川大亮君には，筋電義手のデモシステムの作成でお世話になり，たくさんの議論をしていただきました．

中野薫さん（オットボック社製，前腕筋電義手ユーザー）は，大阪から私費で筑波まで来ていただき，筋電義手について貴重なアドバイスをいただきました．

長野県立情報技術試験場の武久泰夫さんには，回路の FPGA への実装を手伝っていただき，さらに，スイスでの学会参加時にはたいへんお世話になりました．

株式会社 OA 研究所に皆さんには，試験用のボードの設計だけでなく，ハードウェアに関するたくさんのアドバイスをいただきました．

筑波大学大学院の山口佳樹君には，音声認識ボード関係のプログラム作成でお世話になりました．

筑波大学星野研究室の皆さんには，ゼミやコンパや温泉旅行でお世話になりました．

最後に，両親には，現在までの金銭的な補助だけでなく，たくさんの励ましをいただいたことに感謝します．

参考文献

[1] Synthesis datasheet. http:// www.synopsys.com/ products/logic/ synthesis_ds.html.

[2] D.E. トーマス, P.R. モアビー共著, 飯塚哲哉, 浅田邦博共訳. 設計言語 verilog-hdl 入門. 倍風館, 1985.

[3] M. Iwata, I. Kajitani, H. Yamada, H. Iba, and T. Higuchi. Parallel problem solving from nature - ppsn iv. In *Lecture Notes in Computer Science 1141, Springer-Verlag*, pp. 761–770, 1996.

[4] 岩田昌也, 梶谷勇, 村川正広, 平尾友二, 伊庭斉志, 樋口哲也. 進化するハードウェアを用いたパターン認識システム. 電子情報通信学会論文誌 Vol.J81-D-II No.10, pp. 2411–2420, 1998, 10 月.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[6] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolvable hardware with genetic learning: A first step towards building a darwin machine. In *Proceedings of 2nd International Conference on the Simulation of Adaptive Behavior*, pp. 417–424. MIT Press, 1993.

[7] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi. Hardware evolution at function level. In *Proceedings of the Parallel Problem Solving from Nature IV*, pp. 62–71. Springer Verlag, 1996.

[8] M. Murakawa, S. Yoshizawa, and T. Higuchi. Adaptive equalization of digital communication channels using evolvable hardware. In *Evolvable Systems: From Biology to Hardware*. Springer, 1996.

[9] M. Murakawa, S. Yoshizawa, I. Kajitani, and T. Higuchi. On-line adaptation of neural networks with evolvable hardware. In *Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

[10] M. Murakawa, S. Yoshizawa, T. Adachi, S. Suzuki, K. Takasuka, M. Iwata, and T. Higuchi. Analogue ehw chip for intermediate frequency filters. In *Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science 1478, Springer Verlag*, pp. 134–143, 1998.

[11] T. Higuchi, H. Iba, and B. Manderick. Evolvable hardware. *Massively Parallel Artificial Intelligence*, 1994.

[12] W. Liu, M. Murakawa, and T. Higuchi. Atm cell scheduling by function level evolvable hardware. In *Evolvable Systems: From Biology to Hardware*. Springer, 1996.

[13] W. Liu, M. Murakawa, and T. Higuchi. Evolvable hardware for on-line adaptive traffic control in atm networks. In *Proceedings of the Genetic Programming 1997*, pp. 504–509. Morgan Kaufmann, 1997.

[14] Adrian Thompson. An evolved circuit,intrinsic in silicon,entwined with physics. In *Evolvable Systems: From Biology to Hardware*. Springer, 1996.

[15] M. Salami, M. Murakawa, and T. Higuchi. Data compression based on evolvable hardware. In *Evolvable Systems: From Biology to Hardware*. Springer, 1996.

[16] M. Salami, M. Murakawa, and T. Higuchi. Lossy image compression by evolvable hardware. In *Proceedings of the Workshop on Evolvable Systems in IJCAI-97*, pp. 53-59, 1997.

[17] T. Higuchi, M. Iawata, I. Kajitani, H. Iba, Y. Hirao, T. Furuya, and B. Manderick. Evolvable hardware and its applications to pattern recognition and fault-tolerant systems. In Eduaro Sanchez and Marco Tomassini, editors, *Towards Evolvable Hardware*. Springer, 1995.

[18] I. Kajitani, T. Hoshino, M. Iwata, and T. Higuchi. Variable length chromosome ga for evolvable hardware. In *Proceedings of ICEC96*, 1996.

[19] 梶谷勇, 星野力, 平尾雄二, 岩田昌也, 樋口哲也. 遺伝的アルゴリズムによる回路合成方式の高速化手法. 電気学会論文誌 投稿中, 1998.

[20] D. Keymeulen, M. Durantez, K. Konaka, Y. Kuniyoshi, and T. Higuchi. An evolutionary robot navigation system using a gate-level evolvable hardware. In *Evolvable Systems: From Biology to Hardware*. Springer, 1996.

[21] D. Keymeulen, K. Konaka, M. Iwata, Y. Kuniyoshi, and T. Higuchi. Evolvable reactive execution system using reconfigurable hardware:

a robot navigation system case study. In *Working notes of AAAI Fall Symposium on Model-directed Autonomous Systems*, pp. 23–31, 1997.

[22] D. Keymeulen, K. Konaka, M. Iwata, Y. Kuniyoshi, , and T. Higuchi. Robot learning using gate-level evolvable hardware. In *Sixth European Workshop on Learning Robots (EWLR-6)*, 1998.

[23] D. Keymeulen, M. Iwata, K. Konaka, R. Suzuki, Y. Kuniyoshi, and T. Higuchi. Off-line model-free and on-line model-based evolution for tracking navigation using evolvable hardware. In *First European Workshop on Evolutionary Robotics*, 1998.

[24] Adrian Thompson. Evolving electronic robot controllers that exploit hardware resources. In *Proceedings of ECAL95*, 1995.

[25] I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajiwara, D. Keymeulen, M. Iwata, and T. Higuchi. A gate-level ehw chip; implementing ga operations and reconfigurable hard ware on a single lsi. In *Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science 1478, Springer Verlag*, pp. 1–12, 1998.

[26] 梶谷勇, 星野力, 西川大亮, 横井浩史, 梶原信樹, 樋口哲也. 障害者に適応可能な筋電操作型義手の開発. 信学技報 PRMU 98-85, pp. 9–16, 1998.

[27] H. Sakanashi, M. Salami, M. Iwata, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, and T.Higuchi. Evolvable hardware chip for high precision printer image compression. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), AAAI press /MIT press, Cambridge, Massachusetts*, pp. 486–491, 1998.

[28] 梶谷勇, 星野力, 村川正宏, 樋口哲也. Gaによるニューラルネットワークの構造学習用回路の実 現. 日本神経回路学会誌, Vol. 5, No. 4, pp. 145–153, 1998.

[29] C. J. De luca. Physiology and mathematics of myoelectric signals. *IEEE Transactions on Biomedical Engineering*, Vol. BME-26, No. 6, pp. 313–325, 1979.

[30] 平岩明, 内田典佳, 下原勝憲. 筋電操作ハンドの制御のための皮膚表面筋電のニューラルネットによる認識. 電子情報通信学会 MBE 91-114, pp. 49–56, 1992.

[31] M. Uchida, Ide H, and S. P. Ninomija. Control of a robot arm by myoelectric potential. In *Journal of Robotics and Mechatronics vol.5 no.3*, pp. 259–265, 1993.

[32] B. Hudgins, P. Parker, and R. N. Scott. A new strategy for multifunction myoelectric control. *IEEE Transaction on Biomedical Engineering*, Vol. 40, No. 1, pp. 82–94, 1993.

[33] O. Fukuda, T. Tsuji, and M. Kaneko. An emg controlled robotic manipulator using neural networks. In *IEEE RO-MAN'97*, 1997.

[34] Asela Gunawardana. Function classification for the myoelectric control of a prosthetic hand. In *IEEE Region II Student Paper Contest at University of Pennsylvania*, 1995.

[35] K. A. Farry. Myoelectric teleoperation of a complex robotic hand. *IEEE Transaction on Robotics and Automation*, Vol. 12, No. 5, pp. 775–788, 1996.

[36] 福田修, 辻敏夫, 金子真. ニューラルネットによる連続動作emgパターンの識別. 電学論C, 117巻19号, 平成9年, pp. 1490–1497, 1997.

[37] K. Ito, T. Tsuji, A. Kato, and M. Ito. Limb-function discrimination using emg signals by neural network and application to prosthetic forearm control. In *Proceedings of the IJCNN91*, pp. 1214–1219, 1991.

[38] 辻敏夫, 市延弘行, 伊藤宏司, 長町三生. エントロピーを用いた誤差逆伝搬型ニューラルネットワークによる emg からの前腕動作の識別. 計測自動制御学会論文集, Vol.29, No.19, pp. 1213–1220, 1993.

[39] 西川大亮, 兪文偉, 樋口哲也, 横井浩史, 嘉数侑昇. 多自由度義手のための on-line 学習装置の設計. ロボット学会学術講演会予稿集, 1998.

[40] 笹尾勤. 論理設計 スイッチング回路理論. 近代科学社, 1995.

[41] Darrel Whitley. Genetic algorithm tutorial. In *Statistics and Computing Volume 4*, pp. 65–85, 1994.

[42] C.H.M. and van Kemenade. Comparison of selection schemes for evolutionary constrained optimization. In *proc. of the Dutch Conference on AI*, pp. 245–254, 1996.

[43] R. K. Brayton, G. D. Hachel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[44] Nec cb-c9 ファミリ ユーザーズ・マニュアル, 1997.

[45] LiebhardLoffler 著, 平澤泰介訳. 義手 – その起源と発達 –, パシフィックサプライ株式会社, 1986.

[46] G. W. Horn. Electromyographic signal produced by muscle movement controls grasp of prosthetic fingers. *electronics*, pp. 34–36, 1963.

[47] マイオトレーナーの使用方法. http:// www.p-supply.co.jp/ ronbun/maio/maio2.html.

[48] D. J. Atkins, D. C.Y. Heard, and W. H. Donovan. Epidemiologic overview of individuals with upper-limb loss and their reported research priorities. In *Journal of Prosthetic and Orthotics, Volume 8, Number 1*, 1996.

[49] E. Kwatny, D. H. Thomas, and H. G. Kwatny. An application of signal processing techniques to the study of myoelectric signals. *IEEE Transactions on Bio-Medical Engineering*, Vol. BME-17, No. 4, pp. 303–313, 1970.

[50] N. Hogan and R. W. Mann. Myoelectric signal processing: Optimal estimation applied to electromyography – part1 : Derivation of the optimal myoprocessor. *IEEE Transactions on Biomedical Engineering*, Vol. BME-27, No. 7, pp. 382–395, 1980.

[51] N. Hogan and R. W. Mann. Myoelectric signal processing: Optimal estimation applied to electromyography - part1 : Optimal myoprocessor performance. *IEEE Transactions on Biomedical Engineering*, Vol. BME-27, No. 7, pp. 396–410, 1980.

[52] G. N. Saridis and T. P. Gootee. Emg pattern analysis and classification for a prosthetic arm. *IEEE Transaction on Biomedical Engineering*, Vol. BME-29, No. 6, pp. 403–412, 1982.

[53] S. C. Jacobsen, D. F. Knutti, R. T. Johnson, and H. H. Sears. Development of the utah artificial arm. *IEEE Transaction on Biomedical Engineering*, Vol. BME-29, No. 4, pp. 249–269, 1982.

[54] S. Lee and G. N. Saridis. The control of a prosthetic arm by emg pattern recognition. *IEEE Transaction on Automatic Control*, Vol. AC-29, No. 4, pp. 290–302, 1984.

[55] D. Graupe and W. K. Cline. Functional separation of emg signals via arma identification methods for prosthesis control purposes. *IEEE Transactions on Systems, Man, And Cybarnetics*, Vol. SMC-5, No. 2, pp. 252–259, 1975.

[56] D. Graupe, J. Magnussen, and A. A. Beex. A microprocessor system for multifunctional control of upper-limb prostheses via myoelectric signal identification. *IEEE Transactions on Automatic Control*, Vol. AC-23, No. 4, pp. 538–544, 1978.

[57] 辻敏夫, 伊藤宏司, 長町三生. 義手制御を目的とした多チャンネル emg 動作識別法. 電子情報通信学会論文誌, Vol. J70-D, No. 1, pp. 207–215, 1987.

[58] M. F. Kelly, P. A. Parker, and R. N. Scott. The application of neural networks to myoelectric signal analysis: Preliminary study. *IEEE Transaction on Biomedical Engineering*, Vol. 37, No. 3, pp. 221–230, 1990.

[59] Brain maker professional cnaps system. http:// www.sxst.it/ cls_cnp.htm.

[60] M. Murakawa, S. Yoshizawa, I. Kajitani, and T. Higuchi. Evolvable hardware for generalized neural networks. In *Fifteenth International Joint Conference on Artificial Intelligence*, pp. 1146–1151. Morgan Kaufmann, 1997.

[61] G. F. Miller and P. M. Todd. Designing neural networks using genetic algorithms. In *Proceedings of the International conference on Genetic Algorithms.*, pp. 379–384, 1989.

[62] Xin Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, Vol. 8, No. 4, pp. 539–567, 1993.

[63] E. Fiesler. Comparative bibliography of ontogenic neural networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 1994.

[64] Darrel Whitley. Genetic algorithms and neural networks. In *Genetic Algorithms in Engineering and Computer Science.*, pp. 203 216, 1995.

[65] I.M. Bland and G.M. Megson. Implementing a generic systolic array for genetic algorithms. In *Proceedings of the First On-Line Workshop on Soft Computing*, pp. 268–273, 1996.

[66] B.C.H. Turton and T. Arslan. A parallel genetic vlsi architecture for combinatorial real-time applications - disc scheduling. In *Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 493–500, 1995.

[67] T. Maruyama, T. Funatsu, and T. Hoshino. A field-programmable gate-array system for evolutionary computation. In *Lecture Notes in Computer Science 1482 Field-Programmable Logic and Applications (FPL)*, pp. 356–365, 1998.

[68] G.M. Megson and I.M. Bland. A generic systolic array for genetic algorithms. In *Technical report, Dept. of Computer Science, University of Reading, U.K.*, 1996.

[69] I.M. Bland and G.M. Megson. Systolic array library for hardware genetic algorithms. In *Technical report, Dept. of Computer Science, University of Reading, U.K.*, 1998.

[70] I.M. Bland and G.M. Megson. Synthesis of a systolic array genetic algorithms. In *Technical report, Dept. of Computer Science, University of Reading, U.K.*, 1998.

[71] Lawrence Davis. Order-based genetic algorithms and graph coloring problem. In *Handbook of Genetic Algorithms*, pp. 72–90, 1991.

[72] S. D. Scott, A. Samal, and S. Seth. Hga: A hardware-based genetic algorithm. In *Proceedings of the 1995 ACM/SIGDA Third Int. Symposium on Field-Programmable Gate Arrays*, pp. 53–59, 1995.

[73] S. D. Scott, S. Seth, and A. Samal. A hardware ebgine for genetic algorithms. In *Technical Report UNL-CSE-97-001, University of Nebraska-Lincoln*, 1997.

[74] P. Graham and B. Nelson. Genetic algorithms in software and in hardware – a performance analysis of warkstation and custom computing machine implementation. In *Proceedings of the IEEE symposium on FPGAs for Custom Computing Machines*, pp. 216–225, 1996.

[75] M. Salami. Multiple genetic algorithm processor for hardware optimization. In *Evolvable Systems : From Biology to Hardware*, pp. 249–259, 1996.

[76] N. Yoshida, T. Moriki, and T. Yasuoka. Gap: Generic vlsi processor for genetic algorithms. In *Second Int'l ICSC Symp. on Soft Computing*, pp. 341–345, 1997.

[77] N. Yoshida, T. Yasuoka, and T. Moriki. Vlsi architecture for steady-state genetic algorithms (in japanese). Technical report. Research Reports on Information Science and Electrical Engineering of Kyushu University Vol.3 No.1, 1998.

[78] M. Geoke, M. Sipper, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini. Online autonomous evolware. In *Evolvable Systems: From Biology to Hardware*. Springer, 1996.

[79] B. Shackleford, E. Okushi, M. Yasuda, H. Koizumi, K. Seo, and T. Iwamoto. Hardware framework for accelerating the execution speed of a genetic algorithm. *IEICE Transaction on Electron*, Vol. E80-C, No. 7, pp. 962-969, 1997.

[80] P. D. Hortensius, R. D. Mcleod, and H. C. Card. Parallel random number generation for vlsi systems using cellular automata. *IEEE Transaction on Computers*, Vol. 38, No. 10, pp. 1466–1473, 1989.

[81] P. D. Hortensius, H. C. Card, R. D. Mcleod, and W. Pries. Importance sampling for ising computers using one-dimensional cellular automata. *IEEE Transaction on Computers*, Vol. 38, No. 6, pp. 769-774, 1989.

[82] P. D. Hortensius, R. D. Mcleod, and H. C. Card. Cellular automata-based signature analysis for built-in self-test. *IEEE Transaction on Computers*, Vol. 39, No. 10, pp. 1273–1283, 1990.

[83] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller. The analysis of one-dimensional linear cellular automata and their aliasing properties. *IEEE Transaction on Computer-Aided Design*, Vol. 9, No. 7, pp. 767–778, 1990.

[84] D. Thierens and D.E. Goldberg. Elitist recombination: an integrated selection recombination ga. In *Proceedings of First IEEE conference on Evolutionary Computation*, pp. 508–512, 1994.

[85] D. Thierens. Selection schemes, elitist recombination and selection intensity. In *ICGA97*, pp. 152–159, 1997.

[86] C.H.M. and van Kemenade. Cross-competition between building blocks - propagating information to subsequent generations -. In *ICGA97*, pp. 2–9, 1997.

[87] G. Syswerda. Uniform crossover in genetic algorithms. In *ICGA89*, pp. 2–9, 1989.

[88] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer. Biases in the crossover landscape. In *ICGA89*, pp. 10–19, 1989.

[89] Lashon B. Booker. Recombination distributions for genetic algorithms. In *Foundations of Genetic Algorithms - 2*, pp. 29–44, 1992.

[90] D. Thierens and D.E Goldberg. Mixing in genetic algorithms. In *Fifth International Conference on Genetic algorithms*, pp. 38–45, 1993.

[91] A user's guide to gaucsd 1.4. ftp:// ftp.cs.ucsd.edu/ pub/GAucsd/ GAucsd14.ps.Z.

[92] D. Debnath and T. Sasao. Minimization of and-or-exor three-level networks with and gate sharing. *IEICE Transaction on Information and Systems*, Vol. E80-D, No. 10, pp. 1001–1008, 1997.

[93] Surface electromyography: Detection and recording. http://www.delsys.com /emg_articles /EMG.shtml.

[94] W. Yu, H. Yokoi, Y. Kakazu, and D. Nishikawa. Electromyographic (emg) pattern recognition by reinforcement learning method for prosthetic arm control. In *Intelligent Engineering Systems through Artificial Neural Networks, Volume 7*, pp. 503–508, 1997.

[95] 吉田正樹, 赤沢賢造, 藤井克彦. 整流積分筋電位の精度改善方法. 医用電子と生体工学 第26巻 第1号, pp. 25–31, 1988 3月.

[96] 辻敏夫, 森大一郎, 伊藤宏司. 統計的構造を組み込んだニューラルネットワークによるemg動作識別法. 電学論C, 112巻8号, 平成4年, pp. 465–473, 1992.

[97] S. Fujii, D. Nishikawa, and H. Yokoi. Development of prosthetic hand using adaptable control method for human characteristics. In *IAS-5*, pp. 360–367, 1998.