人　工　知　能　93−12
（1994．3．9）

HDLによる自律進化的ハードウェア設計システム

溝口潤一　邊見均　下原勝憲
進化システム研究室
ATR人間情報通信研究所
619-02 京都府相楽郡精華町光台2-2

概要

環境に応じて構造を適応的に変化させるハードウェア進化を可能とするためのフレームワークとその計算機構を構築することを目的とし、ハードウェア特にLSIの動作仕様を進化過程を用いて獲得するシステムを提案する。提案するシステムでは、ハードウェアの動作仕様をハードウェア記述言語（HDL）を用いて記述し、この記述を進化的手法を用いて生成する。また、進化的計算および進化的手法を用いることにより、未知で予測できない環境で動作するハードウェア仕様を明示的な設計知識なしに獲得することが可能となる。HDLという文法を持つ言語による記述を進化的手法により生成するために、文法構造に基づく進化的操作を可能とし、HDLのプログラムが発生する過程を操作することを目的とするプロダクション遺伝的アルゴリズムを導入する。最後に、プロダクション遺伝的アルゴリズムに基づく進化過程を通してハードウェア仕様記述がその回路規模を拡張し、機能を増加させる様子が実験結果により示される。

Evolutionary Automated Hardware Design System with HDL

Jun'ichi Mizoguchi　Hitoshi Hemmi　Katsunori Shimohara
Evolutionary Systems Department,
ATR Human Information Processing Research Laboratories,
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

Abstract

This paper proposes a system that obtains hardware (especially LSI) behavior specification with using an evolutionary process, towards the evolutional framework and computational mechanisms of hardware evolution which adaptively changes its structure according to the environment. In the proposed system, the hardware specification is described with a Hardware Description Language (HDL) and the description is obtained through evolutionary methods. Also, evolutionary computation and methods make it possible to design hardwares that act in an unknown and unpredictable environment without explicit design knowledge. In order to evolve HDL description that is regulated by its grammar, Production Genetic Algorithms is introduced which enables evolutionary operations based on the grammar and operates the development process of HDL programs. Finally, experimental results shows that through an evolutionary process based on the PGAs, a hardware specification program expands its circuit scale and as a result its functionality.

1 はじめに

ハードウェアの進化は可能であるのか。また、進化するハードウェアの構築のためにはいかなる機構が必要であるのか。この疑問が、本稿の研究を始めるにあたっての動機である。ここで、ハードウェアの進化とは、ハードウェアが動作する環境に対して適した構造を獲得し、さらに環境の変化に応じて適応的に構造を変化させる能力であると考える。このような適応的に構造を獲得することのできるハードウェアは、未知で予測できない環境での動作が要求される適応システム構築に向けて新しい可能性を開くものとなろう。

近年、進化的計算論の分野において、進化ハードウェア (Evolvable Hardware) の実現に向けての研究が de Garis、樋口等によってなされている [2, 4]。樋口等は、フィールド・プログラマブル・ゲート・アレイ (FPGA) の構造を設定するビット列を進化的学習により生成することに成功している。

進化ハードウェアの実現のためには、進化機構を備えた新しいハードウェア・アーキテクチャやハードウェア・デバイスが必要である。しかし、それと同時に、進化のフレームワークとその計算機構の構築もまた、ハードウェア進化を可能とし、その適用範囲と可能性を探る上で重要である。このようなハードウェア進化機構の構築のためには、ハードウェアの構造を表現する方法を確立する必要があると考える。

本稿では、ハードウェア進化のフレームワークと計算機構を構築するという究極の目的を目指し、ハードウェアの構造を表現する手段として、ハードウェア記述言語 (HDL) のプログラムによる動作仕様記述を用い、そのHDL プログラムを進化過程を通して獲得することを試みる。

また、現在のハードウェア設計は解析的な手法で行なわれており、必要な部品、回路規模といったものを設計時に見積る必要がある。そのため、未知で予測不可能な環境で動作するハードウェアを設計することは困難である。このような通常の方法では設計が困難な問題を解決する一つの方法としても、進化的手法による設計が有効となると考える。

一方、遺伝的アルゴリズム (GAs) は、生物の進化機構を模倣した手法として、さまざまな分野に応用されてきている。生物は膨大な遺伝子の多様性を保ち、環境の変化に応じて自分自身を適応的に変化させることにより、進化を繰り返してきた。このように多様な遺伝子による多様な機能を集団中に維持する機構により、変動する環境に対して適応した機能を発現させ、適応的に構造を変化させる Evolvable Hardware を構築することが可能になると考える。GAs を適応システム構築のための一手法と捉え、目的とするハードウェア進化のフレームワークに利用することとした。

本稿では HDL によりプログラムされたハードウェア仕様を進化させることを目的としたプロダクション遺伝的アルゴリズムを提案し、新しい染色体表現と遺伝子操作を導入する。HDL の文法を書き換えシステムで表すことにより、ハードウェア仕様記述をコード化した進化的操作に適した染色体表現が得られ、HDL プログラムが書き換えシステムの開始記号から生成される過程を進化機構を通じて操作することが可能となることを示す。

2 ハードウェア仕様記述の進化

ハードウェア構造を決定するハードウェア動作仕様を記述するために、ハードウェア記述言語 (HDL) を使用する。HDL はハードウェアの動作仕様を記述するためのプログラミング言語であり、多くの解析的手法に基づく CAD(Computer Aided Design) のツールとして用いられている。

HDL を用いてハードウェアを記述することとしたのは以下のような理由からである。

- 記述されたハードウェア動作仕様がいかなる性能を有するかを動作シミュレータを用いて評価することができる。
- マクロなレベルでの動作記述が可能であり、より高度なハードウェア仕様が獲得できる。
- HDL は文法を持った言語であり、その文法構造に基づいて仕様記述を染色体上にコーディングすることにより、文法的な部分構造が形成され、それが意味的にも機能を持った部分構造となることが期待される。
- 文法構造を操作することで、より多くの問題に対処することが可能となると考えられる。
- 獲得された動作仕様を実際のハードウェアとして実現することができる。

提案するシステムは、進化的過程の中で、HDL プログラムを発生させることにより、多様な機能を維持する遺伝子集団を形成し、環境に適応的に対応した HDL プログラムとしての仕様記述を生成する。

HDL はプログラム言語であり、プログラムの進化の研究として [6] がある。この中で、Koza は遺伝的プログラミング (GP) により、プログラムが進化しうることを示した。Koza は LISP のプログラムそのものを染色体として木構造で表わし、この木に対して交叉などの遺伝子操作を定義している。GP の木構造では、すべてのノー

ドは文法的に等価であり、二個体の任意のノード間での交叉が可能である。すなわち、交叉の結果として生じる個体は、LISP プログラムとして有効なものとなる。これに対して、多くのプログラム言語では、プログラムそのものを染色体として進化的操作を施し、文法的に正しい個体を生成することは不可能である。

進化ハードウェア構築のための統合的フレームワークの中で HDL を用いる時にも同様の問題がある。進化的操作による変異に対して HDL は脆く、文法的に正しい個体を進化的操作を通して生成することは非常に困難である。そのため、この HDL の欠点を補い、かつ、ハードウェア記述性能を損うことのない進化機構を作り出すことが必要となる。

3 プロダクション遺伝的アルゴリズム

上記の HDL の進化的操作に対する脆さを解決し、文法構造に基づいた操作を可能とするために、プロダクション遺伝的アルゴリズムを導入した。

進化的遺伝子操作の結果として、文法的に正しい個体が生成される機構により、文法的に無意味な記述を淘汰することに進化のコストを掛けるのを防ぎ、目的とするハードウェアの仕様を表す HDL 記述を獲得することに選択圧力を集中させることができる。

文法構造に基づく染色体構造と遺伝子操作を取り入れることにより、染色体の各部分に文法により定められた機能構造が現われ、このような機能構造に注目した操作が可能となる。そして、それぞれの機能構造が遺伝子操作を通して機能を変異させ、全体としての機能を向上させることが期待される。

プロダクション遺伝的アルゴリズムを実現し、進化的自動ハードウェア設計システムを構築するために、ハードウェアの動作仕様を記述する HDL として SFL [1] (Structured Function Description Language) を選択した。SFL は制御オートマトンが相互に作用し、データ信号がオートマトンの状態に応じて処理される複雑な制御シーケンスを効率よく記述することができる HDL である。

そして、SFL の文法を書き換えシステムのプロダクション・ルールとして表現した。SFL のプログラムは、このプロダクション・ルールを順次適用することにより生成される。プロダクション・ルールがどのように適用されて SFL プログラムが生成されるかという過程を染色体上にコーディングし、この染色体に文法構造に基づく遺伝子操作を施す。SFL プログラムを生成するために、染色体をプロダクション・ルールに従って解釈し、SFL プログラムへと発達させる段階を設ける。生成された SFL プログラムの性能を SFL 動作シミュレータを用いてシミュレーションすることにより、個々の染色体の性能を評価する。この性能値に基づいて選択が行われる。

[1] SFL は NTT により開発された LSI デザイン・システム、PARTHENON のハードウェア記述言語である。

3.1 実験システム

図 1 に実験システムのブロック図を示す。太線は各個体に関する情報の流れを示し、細線はプロダクション・ルールに関する情報の流れを示している。同じ名前の付いた箱は、その作業が並列して実行されることを示している。まず、染色体の初期集団が生成される。続いて各染色体は解釈され、SFL プログラムへと発達させられる。SFL プログラムは SFL シミュレータにより評価される。その評価値に応じて選択が行われ、交叉、重複等の遺伝子操作が施される。最後に、次の世代の染色体の集団が生成される。新世代の生成、遺伝子操作、及び染色体の管理は同一のワークステーションで行われる。染色体の解釈、SFL プログラムへ発達、シミュレータによるシミュレーションは複数のワークステーションで並列に行われる。

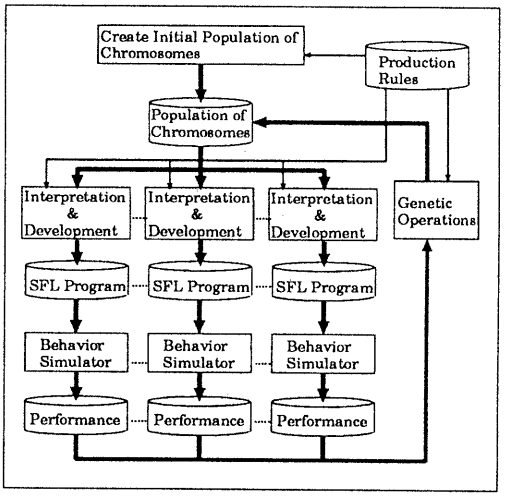

図 1: 実験システム ブロック図

3.2 書き換えシステム

SFL 文法を書き換えシステムのプロダクション・ルールにより表現する。書き換えシステムは開始記号、非終端記号、終端記号、及びプロダクション・ルールよりなる。プロダクション・ルールは、開始記号が SFL プログラムへと発達していく過程を定めるものである。まず、

1個のプロダクション・ルールが開始記号に適用され、開始記号は新しい記号列に置き換えられる。続いて、これらの新しい記号に対してプロダクション・ルールが適用され、また別の記号列に置き換わる。この過程がすべての記号が終端記号になるまで続けられる。この置き換えの結果としてSFLプログラムが形作られる。プロダクション・ルールの構成により、生成されるSFLプログラムは全く異なるものとなる。

SFLプログラムの発生過程にプロダクション・ルールを適用することにより、複雑で階層構造を持ったSFLプログラムを生成することが可能となる。さらに、プロダクション・ルールの適用を続けることにより、任意の規模のSFLプログラムを獲得することが可能となる。実際のシステムでは、SFLプログラムが無限に成長することを防ぐために、それぞれのプロダクション・ルールの適用回数に制限を設けている。

図2はプロダクション・ルールの一例の一部を示している。このプロダクション・ルールの総数はおよそ350個である。それぞれのルールには、カテゴリ番号とサブ番号からなる番号の組が付いている。この番号の組をプロダクション番号と呼ぶ。プロダクション・ルールの中で、同じカテゴリ番号を持つものは、文法的に等価なルールとなり、文法的には入れ替えが可能となる。

3.3 染色体表現

染色体は木構造を持ち、そのノード上に、書き換えシステムの開始記号から始まり、SFLプログラムが生成されるまでにプロダクション・ルールが適用される過程が表現される。染色体表現の例を図3に示す。木のそれぞれのノードには書き換え過程で生じる記号を置き換えるプロダクション番号が付けられる。木のルートには開始記号を置き換えるプロダクション・ルールの番号が付く。子供のノードには、その親ノードのプロダクション・ルールの適用によって生成された記号列に対して適用されるルールのプロダクション番号が入る。

プロダクション・ルールに基づいた遺伝子オペレータを導入することにより、文法的に正しいSFLプログラムを遺伝子操作を通して生成することができるようになり、また機能的構造体が染色体中に構成される。

3.4 遺伝子オペレータ

以下に示す遺伝子操作が用意されている。

選択 ルーレット・ホイール選択 [3] が使われる。各個体の遺伝子はそれぞれの性能に比例して次の世代に受け継がれる。加えて、エリート戦略 [3] が使われ、集団中の最良の個体の幾つかが、必ず次の世代に残される。エリート戦略を使用することで、確率的な遺伝的浮動により最良の個体がすべて消滅してしまうことを防いでいる。

交叉 交叉は2個の染色体間での部分木の交換であり、遺伝的プログラミング [6] で用いられている交叉と類似しているが、以下の点で大きく異なっている。すなわち、交叉の結果として得られる染色体から生成されるSFLプログラムが、文法的に正しいものであるようにするために、部分木の交換はプロダクション・ルールのカテゴリ番号が同じノード間でのみ行われる。交叉が任意のノード間でランダムに行われるのではなく、文法的に等価なノード間で行われるために、それぞれの機能を有する部分プログラムが染色体間で交換されうる。このように、交叉を施すことにより、有効な部分プログラムが集められ、目的の動作をするSFLプログラムが形成されることが期待される。

突然変異 突然変異は突然変異率に応じて、各ノードを新しいノードに変化させる。文法を維持するために、変化後の新しいノードには、同じカテゴリ番号を持ったプロダクション番号が選ばれる。突然変異したノードは新しいノードと置き換えられ、元のノードの下の部分木は削除される。そして、新しいノードの下には、プロダクション・ルールに従って新しい部分木が生成される。突然変異は交叉や次に述べる重複で獲得された部分プログラムにわずかな変更を加える局所探索を行うとともに、全く新しい部分木を生成する。

重複 重複は生物の遺伝子重複 [1, 7] を模倣した操作であり、単純な構造の染色体から複雑な構造の染色体へと進化させることを可能とする。重複により機能の複雑さを増加させ回路規模を増大させることを目的とする。

重複は同一染色体中で機能ブロックの複製を挿入する操作である。機能ブロックとは同じカテゴリのノードがリスト構造となって連続して現れる部分のことであり、重複の結果もまた文法的に正しいものとなる。重複が施された直後では、多くの場合、挿入された機能ブロックは固体全体としての動作に影響を与えることはなく中立的な存在である。そのため、重複は個体としての機能に変化を与えるものではない。その後、挿入された機能ブロックは突然変異等により変異を受け、新しい機能が出現する。染色体の他の部分も変異を受けることにより、挿入されたブロックが個体全体としての動作に組み込まれる。このように、重複に

```
(r0.0)  module        -> K_MOD name list_comp list_pin list_action
(r1.0)  name          -> K_NAME
(r2.0)  list_comp     -> comp
(r2.1)  list_comp     -> list_comp comp
(r3.0)  list_pin      -> empty
(r3.1)  list_pin      -> list_pin pin
(r4.0)  list_action   -> action
(r4.1)  list_action   -> par_action
(r4.2)  list_action   -> cond_action
                      :
(r7.0)  action        -> action1
(r7.1)  action        -> action action1
(r7.2)  action        -> action action2
(r8.0)  action1       -> register
(r9.0)  action2       -> memory
                      :
(r20.0) comp          -> K_INPUT input_name
(r20.1) comp          -> K_OUTPUT output_name
(r20.2) comp          -> K_BIDIRECT bus_name
(r20.3) comp          -> K_INSTRIN inst_name
                      :
```

図 2: プロダクション・ルールの一例

左端の括弧内の'r' に続く番号がプロダクション番号である。例えば、r4.1 はカテゴリ番号が 4 で、サブ番号が 1 のルールとなる。

より個体として新しい機能が追加されることが期待される。

挿入 挿入は他の染色体から機能ブロックが挿入されるということを除いて、重複と同様の操作である。

欠損 欠損は染色体から機能ブロックを削除する操作であり、不要な遺伝子が取り除かれることが期待される。その結果、機能的には等価であるが、より簡潔な仕様記述が得られる。

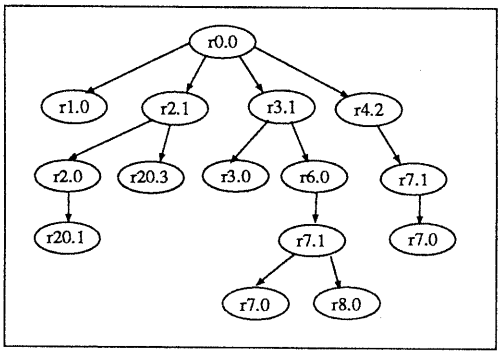

図 3: 染色体表現

木の各ノードには、カテゴリ番号とサブ番号が入る。染色体全体は非常に大きいので、一部分だけ図示する。

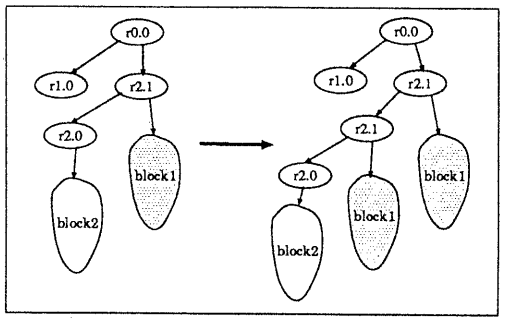

図 4: 重複

block1 という機能ブロックの複製が block1 と block2 の間に入る。

4 実験および結果

提案する進化的設計機構を人工蟻問題に適用した。人工蟻問題とは、人工蟻がトロイド状の白い格子上において、曲がり途切れた黒いセルよりなる軌跡をいかに辿るかを学習する問題である。人工蟻は目前のセルに軌跡があるかないかの 1 個のセンサ入力を得ることができる。この 1 入力をもとに、人工蟻は直進、右折、左折の中から次の行動を決定する。

Jefferson 等 [5] は John Muir Trail(図 5) と呼ばれる軌跡で、有限状態オートマトンとニューラル・ネットワークを用いて人工蟻の学習に成功している。Koza[6] は遺伝的プログラミングを用いて Santa Fe Trail と呼ばれる

軌跡上での学習を行っている。

ここでは、設計システムを用いてJohn Muir Trail上で人工蟻を制御するハードウェア仕様記述を生成することを試みた。以下に述べる3種類のケースについて実験を行った。いづれの場合も生成しようとするハードウェアは、一入力、二出力である。入力は人工蟻のセンサ入力であり、二個の出力により人工蟻の次の動作が決定される。

ケース1 回路規模は指定せず、人工蟻の制御にとって不十分であると思われる程度の規模から設計システムの動作を始める。遺伝子重複などの操作により、回路の規模を拡大し、機能を向上させ、目的の機能を有する回路を獲得する。

ケース2 同一のハードウェア内で独立に動作する二個のハードウェア・モジュールを異なる集団の中で別々に進化させる。入力は同じものが二つのモジュールに与えられ、それぞれのモジュールが一個づつの出力を出し、全体として二出力となる。それぞれのモジュールは異なる種として別の集団の中で進化していき、それが一個の個体であるハードウェアとして動作する共生のモデルである。

ケース3 SFLの文法の中に含まれる演算子の数を減らして、ケース1と同様の実験を行なう。SFLは多くの演算子を有しており、その中からいくつかの演算子が削除されるようにプロダクション・ルールに変更を加えた。

いづれの場合もその他の実験条件は同じであり、集団数は200個体、交叉率は50%、突然変異率は0.5%、重複率は2%、欠損率は1% として実験を行った。この実験では挿入は使用していない。すべてのケースで性能値は以下の式で与えられる。

$$performance = score + (time\_limit - time\_steps)$$

scoreは、限られた時間(time_limit = 350)の中で通過した軌跡上のセルの数である。time_stepは、軌跡を辿るのにどれだけの時間を費やしたかである。このように性能値を定義することにより、より短い時間で軌跡を辿るハードウェアを生成するように選択圧が働くことになる。ケース1、2、3の進化の過程をそれぞれ図6、7、8に示す。ケース1では、100世代まではほとんどのハードウェア仕様の制御状態の数は3か4である。251世代において、軌跡全体を通過するハードウェア仕様が得られた。この仕様は6個の制御状態よりなり、332ステップで全軌跡を通過する。重複等の遺伝子操作 により、回路規模が拡大し機能を増加させ、230ステップで全軌跡を通過するハードウェア仕様が849世代目で得られた。その制御状態数は8であった。ケース2では、144世代目で全軌跡を通過する個体が得られており、これには314ステップを要している。270世代目で266ステップで全軌跡を通過する個体が得られ、2個のハードウェア・モジュールはそれぞれ4個の制御状態より構成されている。ケース3では、103世代目において、300ステップで全軌跡を通過する個体が得られ、381世代目で、282ステップで通過する個体が得られている。その制御状態数は6であった。図9にケース3で獲得されたハードウェア仕様の回路図を示す。

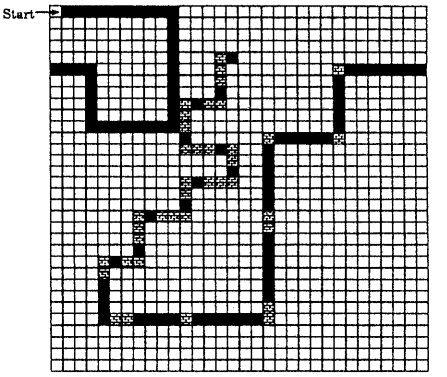

図 5: John Muir Trail

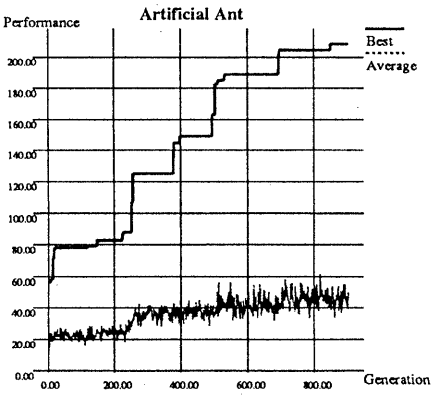

図 6: Artificial Ant: Case 1

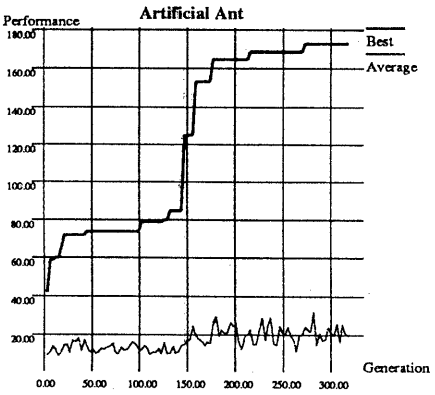

図 7: Artificial Ant: Case 2

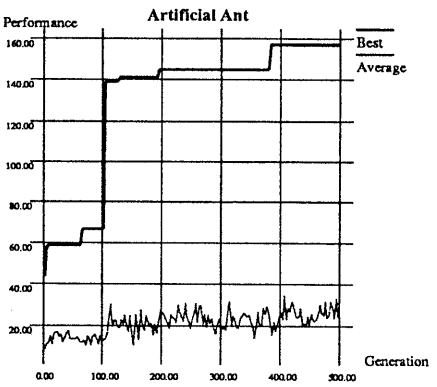

図 8: Artificial Ant: Case 3

5 結論

本稿では、ハードウェアの進化のフレームワークを構築することを目的としてプロダクション遺伝的アルゴリズムを提案し、この進化機構を用いてハードウェアを自動的に設計するシステムについて述べた。この設計システムは通常の解析的手法では設計することが困難なハードウェアの設計を可能にする一つのアプローチでもある。HDL のプログラムがハードウェアの動作仕様を記述する設計図となる。HDL として SFL を選択したが、他の HDL にもこのシステムを適用することが可能である。

プロダクション遺伝的アルゴリズムは HDL のプログラムを発生させる過程を定式化し制御し、また、遺伝子操作の結果として文法的に正しいプログラムを生成することを可能とする。文法構造に基づく染色体構造および遺伝子操作により、染色体の各部分にそれぞれの機能を有する機能構造が形成され、重複等の遺伝子操作により機能の拡張等がなされ、多様な相互に関連する機能を有したハードウェア仕様記述を獲得することができる。

この設計システムで獲得されるハードウェアは必ずしも最適なものではない。むしろ、無駄な部分、冗長な部分を多く含んだ仕様記述が生成される。しかし進化機構においては、このような無駄および冗長さが更なる進化をもたらし、環境に適応的に対応するために必要であると考える。

謝辞

PARTHENON システムを使用する機会を与えて下さいました NTT の小栗氏に感謝致します。

参考文献

[1] 土井洋文. 老化. 岩波書店. 1993

[2] H. de Garis. Evolvable Hardware : Genetic Programming of Darwin Machines, Int. Conf. on Neural Networks and Genetic Algorithms, Lecture Notes in Computer Science, Springer Verlag, 1993.

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

[4] T. Higuchi et al. Evolvable Hardware-Genetic Based Generation of Electric Circuitry at Gate and Hardware Description Language (HDL) Levels, ETL Tech. Report, 93-4, 1993.

[5] D. Jefferson. Evolution as a Theme in Artificial Life: The Genesys/Tracker System, Artificial Life II, edited by Christopher Langton.

[6] J.R. Koza. Genetic Programming On the Programming of Computers by Means of Natural Selection, The MIT Press.

[7] 和田健之介、田中真一. GA は生き残れるか？. 計測と制御. 1993,Vol.32,1. p17-23.

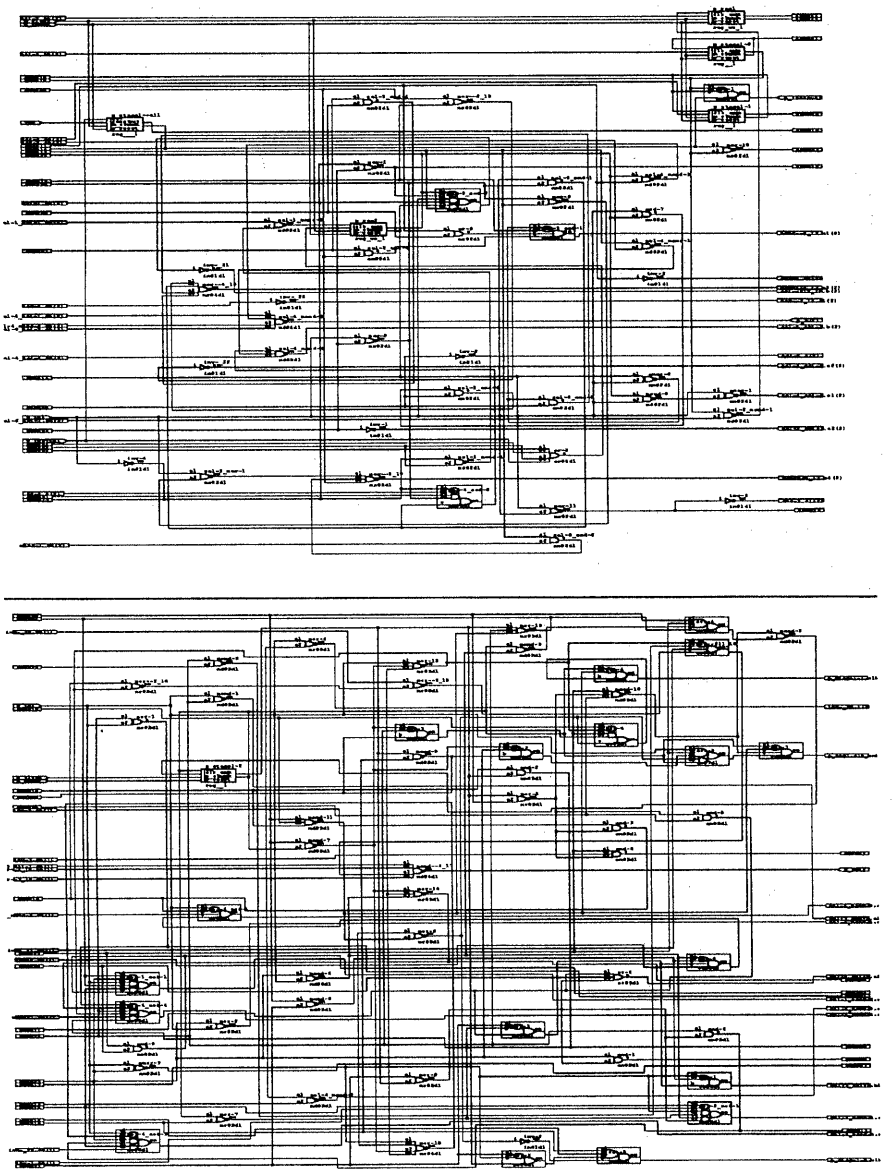

図 9: ケース 3 で獲得されたハードウェアの回路図