東京電機大学

博　士　論　文

進化型ハードウェアのアーキテクチャに関する研究

A Study on Architecture for Evolvable Hardware

2017 年 3 月

塚原　彰彦

A Study on Architecture for Evolvable Hardware

DISSERTATION

Akihiko Tsukahara

Tokyo Denki University

March, 2017

要　旨

近年，集積回路の発展は目覚ましく，それに伴い電子機器の高性能化が進んでいる．家電は高性能化し，小型で高性能な情報端末などが身近なものになる等，その恩恵を受けている．その一方，高機能な製品を短いサイクルで開発することが要求され，回路設計の厳しさは一段と増している．しかしながら，そのような回路設計には様々な知識・経験が求められる．さらに，医療・福祉機器分野では，筋電などの個人差がある生体信号を扱う回路は仕様が特定しづらいといった，回路で扱う対象によって生じる難しさもある．そこで，品質良く，生体や多様な環境に対応できる回路を自動的に設計する手法として，進化型ハードウェア [1]が提案されている．

進化型ハードウェアは，目標とする入出力特性を与えるとハードウェア自身が学習を繰り返し所望の動作を行う回路に進化する仕組みを持つ．そのため，回路設計が不要なことや，使用環境に応じた回路定数の自動調整ができるなどの特徴があり，生体信号を扱う回路や画像処理など幅広い分野で応用が期待されている．例えば，筋電で制御する義手 [2]や，ノイズ除去を行う画像フィルタ [3]などである．

進化型ハードウェアは，回路構成や回路定数を決定する進化アルゴリズムを専用プロセッサとして組み込むことが多く，これまで様々な構成の専用プロセッサが提案されている．これまで提案されている構成の多くは，データ形式は整数であり，解を評価する評価関数計算を専用回路によって実行する構成である．この構成では，高速な実行速度が期待できるが，対象とする問題を変更する際には，評価関数回路を再設計する必要がある．また，実数を扱う構成や，評価関数計算をソフトマクロ汎用 CPU で実行する構成も提案されている．これらの構成では，扱える精度の向上や対象とする問題の変更に対して柔軟に対応できるなどの高い汎用性が期待できるが，回路規模の増大や実行時間の低下が問題となる．さらに，ハードウェア実装において扱える未知パラメータ数は，実装する対象の回路資源に依存するため，多くの未知パラメータが扱える専用プロセッサはあまり見受けられない．

進化型ハードウェアでは，ハードウェアの学習に進化アルゴリズムが用いられ，それを専用プロセッサとして組み込む場合が多い．これまで提案されている進化アルゴリズム専用プロセッサの多くは，扱うデータ形式は整数であり，解の候補の良し悪しを判定する評価関数計算を専用回路によって実行する構成である．この構成では，高速な実行速度が期待できるが，対象とする問題を変更する際には，評価関数回路を再設計する必要がある．また，実数を扱う構成や，評価関数計算をソフトマクロ汎用 CPU で実行す

る構成も提案されている．これらの構成では，扱える精度の向上や対象とする問題の変更に対して柔軟に対応できるなどの高い汎用性が期待できるが，実行時間や回路規模の増大が問題となる．さらに，ハードウェア実装において扱える未知パラメータ数は，実装する対象の回路資源に依存するため，多くの未知パラメータが扱える専用プロセッサはあまり見受けられない．

そこで本研究では，汎用的な進化型ハードウェアのアーキテクチャの確立のため，これまでの進化アルゴリズム専用プロセッサの短所を改善した，実数値遺伝的アルゴリズム（Genetic Algorithm；以下 GA）専用プロセッサの研究・開発を行っている．さらに，提案する実数値 GA 専用プロセッサの一応用として，ディジタルフィルタの設計へ適用し，その有効性を確認している．

本論文は全 5 章から構成されている．

第 1 章では，本研究の背景として，進化型ハードウェアの特徴と解決すべき課題を示し，本研究を行った目的について述べ，本論文の構成と各章の概要についてまとめている．

第 2 章では，進化型ハードウェアの基本的な構成と応用分野を示し，回路構成や回路定数を決定する進化アルゴリズムとその専用プロセッサの概要を述べている．専用プロセッサの一例として，リソースシェアリングを適用した遺伝的アルゴリズム専用プロセッサについて述べている．

第 3 章では，本研究で提案する実数値 GA 専用プロセッサの構成について詳細に述べている．提案する実数値 GA 専用プロセッサは，従来と同様に汎用性を高めるため，実数値をそのまま扱うことができる実数値 GA に基づいて設計され，評価計算をソフトマクロ汎用 CPU で行う構成をとっている．したがって，精度高く，対象とする問題が変更されてもプログラムの変更で柔軟に対応できる．これまでの課題であった実行時間の増大に対して，ソフトマクロ汎用 CPU を多数組み込み，各解候補の評価関数計算を並列に実行することで実行時間の短縮を試みている．さらに，ソフトマクロ汎用 CPU を採用する構成や，多くの未知パラメータに対応する際に問題となる回路規模の増大に対しては，リソースシェアリングを適用することにより使用する演算器の増加を抑えている．すなわち，動作中に回路を共有することで，回路を共有しない場合に比べて 1/3 から 1/4 の演算器で実数値 GA 特有の処理を実現している．これにより，より多くの未知パラメータを扱うことが期待できる．これらの結果，1 チップの FPGA に実装可能で，対象とする問題の変更に対して柔軟に対応でき，多くの未知パラメータ数が扱える構成となっている．また，提案する実数値 GA 専用プロセッサを FPGA に実装し，3 つのベ

ンチマーク問題に適用して性能評価を行った結果について述べている．その結果，実行時間において，汎用 PC と比較して，最高で約 3 倍の高速化を確認した．

第 4 章では，提案する実数値 GA 専用プロセッサの応用として，ディジタルフィルタの設計へ適用した 2 つの事例について述べている．FIR（Finite Impulse Response）フィルタの係数を実数値 GA 専用プロセッサによって決定することで，所望の周波数特性のフィルタが設計できる．実数値 GA 専用プロセッサと FIR フィルタを 1 チップの FPGA に実装し，実験を行っている．1 つ目の事例として，26 次，58 次，122 次のバンドストップフィルタを設計した場合，他の進化アルゴリズムで設計した場合と同等の特性のフィルタが，数百 ms から 2s 程度で設計できることを確認した．さらに 2 つ目の事例として，ディジタル補聴器のフィッティングを想定したシミュレーションを行った結果について述べている．提案する実数値 GA 専用プロセッサを用いて，6 つの難聴パターンのオージオグラムに対してフィッティングを行った結果，すべてフィッティング可能であることを確認した．

最後に第 5 章では，本研究の総括と今後の展望について述べている．

目次

図目次

表目次

第1章　序　論

1.1　本研究の背景

近年，集積回路の発展は目覚ましく，それに伴い電子機器の高性能化が進んでいる．スマートフォンやタブレットなどの小型で高性能な情報端末などが身近なものになり，家電なども高性能化し，その恩恵を受けている．

しかしながら，高機能な製品を短いサイクルで開発することが要求され，回路設計の厳しさは増している．高機能な回路設計には様々な知識・経験が求められる．さらに，医療・福祉機器分野などでは，筋電などの個人差がある生体信号を扱うため，回路の仕様が特定しづらいなどの難しさがある．また昨今，様々な物がインターネットにつながり情報交換を行う，IoT（Internet of Things；モノのインターネット） [4]が急速に普及しつつある．今後，多種多様な IoT システムが想定され，それらのシステムの末端デバイスは様々な環境で使用されることが予想される．

そのような，短い開発サイクル，生体や多様な環境に対応できる回路設計自動化の一手法として，1992 年に樋口らによって提案された進化型ハードウェアがある [1, 5]．これは，進化アルゴリズムとハードウェアを融合させたハードウェアである．進化アルゴリズムは，メタヒューリスティックな最適化アルゴリズムであり，遺伝的アルゴリズム(Genetic Algorithm, GA) [6, 7, 8]や遺伝的プログラミング(Genetic Programming, GP) [9]などがある．

進化型ハードウェアは，目標とする入出力特性を与えると，進化アルゴリズムによってハードウェアの構成を自律的に決定する．すなわち，ハードウェア自身が所望の動作を行う回路に進化するとみなすことができる．したがって，回路を設計する必要がない，使用環境に応じた回路定数の自動調整が可能などの特長がある．

進化型ハードウェアは，進化アルゴリズムが専用プロセッサとして組み込まれることが多い．その際，データ表現にはビットストリングがよく用いられている．しかし，アナログ回路の定数やフィルタの係数を決める場合，ビットストリングに基づく通常の GA は，実数値の扱いに優れているとはいえない．

また，進化アルゴリズムは解の候補に対して，良し悪しを判定するための評価処理を行う．ここで，評価関数は進化させる対象に固有のものとなるため，対象とする問題を変更する度，新たな回路を設計する必要がある．複雑な評価関数であれば，回路設計・検証に長時間を要する．これまで提案されている進化型ハードウェアの多くは，回路の

一部を再設計する必要がある．さらに，決定すべきパラメータが数十程度の高次の問題を扱える進化型ハードウェアはまだ少ない．

上記のように，汎用性が高いアーキテクチャはまだ見受けられない．

1.2　本研究の目的

1.1 節で述べたように，これまで提案されている進化型ハードウェアは，汎用的なものはまだ見受けられない．そこで，本研究では汎用性の高い進化型ハードウェアのアーキテクチャの確立を目的とする．

本論文では，実数値 GA 専用プロセッサによって回路構成や回路定数を決める，実数に基づく進化型ハードウェアを提案する．実数値 GA は，実数値をそのまま扱うことができ，遺伝子型をビットストリングで扱う従来の GA より解探索能力に優れ，決定すべきパラメータが多い高次元の問題に対しても有効である．実数値 GA 専用のハードウェアが実現できれば，従来の GA では扱うことが困難であった複雑な問題や高次元の問題に適用することができる．

さらに提案する実数値 GA 専用プロセッサでは，これまでの進化アルゴリズム専用プロセッサの以下の短所を改善している．

従来，対象とする問題を変更する際は評価回路の再設計が必要であった．しかし提案する実数値 GA 専用プロセッサでは，ソフトマクロ汎用 CPU によって評価関数の計算を行う．これにより，プログラムの書き換えだけでさまざまな問題に対応することができる．

さらに，動作中に回路を共有することで少ない演算器で実数値 GA 特有の処理を実現し，演算器を削減する．すなわち，回路資源を有効に共有することにより，少ない回路規模で実装できる．これにより，共有しない場合に比べ，1/3 から 1/4 の演算器数で実装可能である．その結果，提案する実数値 GA 専用プロセッサは，1 チップの FPGA に実装することができる．

そして，実数値 GA 専用プロセッサを用いた一応用として，ディジタルフィルタの設計に適用した．一例として，バンドストップフィルタの設計に適用し，さらにディジタル補聴器のフィッティングを想定したシミュレーションを行い，提案する実数値 GA 専用プロセッサの有効性を示す．

本研究は特に回路実装の面で，実数に基づく進化型ハードウェアの応用に貢献しようとするものである．

1.3　本論文の構成

本論文の構成を図 1.1 に示す．本論文は 5 章から構成されている．以下，各章ごとにその概要を述べる．

第 1 章「序論」

本研究の背景として，進化型ハードウェアの特徴と解決すべき課題を示し，目的について述べ，さらに本論文の構成と各章の概要についてまとめている．

第 2 章「進化型ハードウェアの概要」

進化型ハードウェアの基本的な構成を示し，回路構成や回路定数を決定する進化アルゴリズムとそれらの専用ハードウェアの概要を述べる．進化アルゴリズムの専用ハードウェアの一例として，リソースシェアリングを適用した遺伝的アルゴリズム専用プロセッサについて述べる．また，提案する実数値 GA 専用プロセッサで用いているリソースシェアリングの概要も述べる．

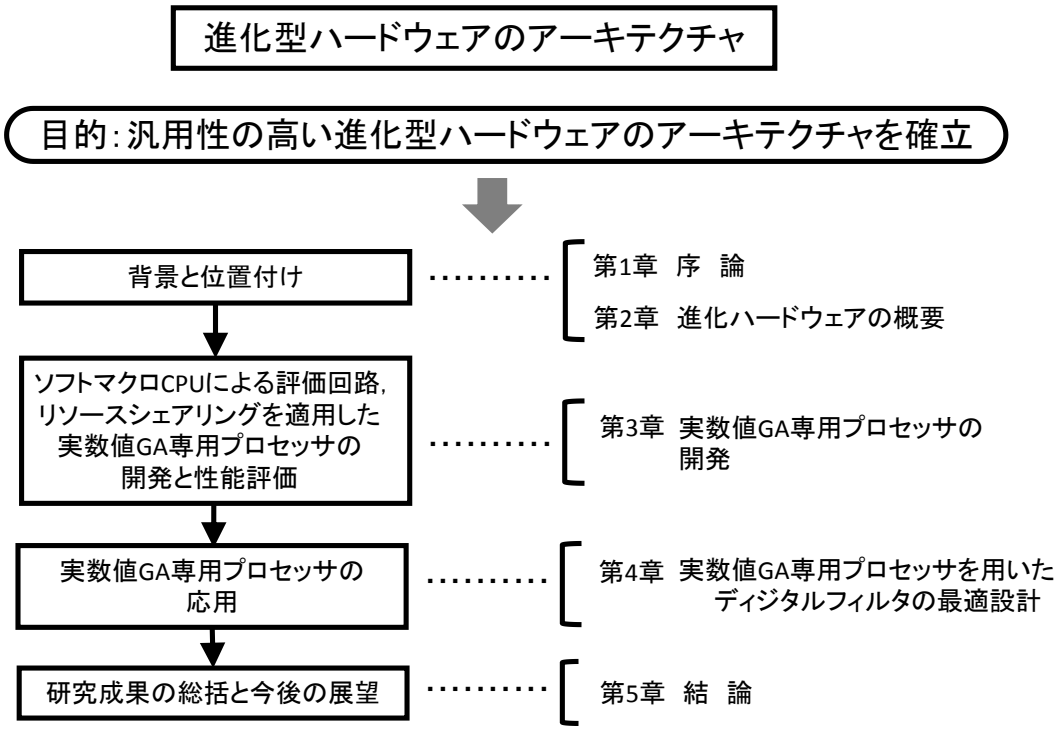


図 1.1. 本論文の構成

第 3 章「実数値 GA 専用プロセッサの開発」

汎用的な進化型ハードウェアのアーキテクチャの確立を目的として開発した, 実数値 GA 専用プロセッサの構成について詳細に述べる．実数値 GA は，実数値をそのまま扱うことができ，通常の GA よりも探索能力が優れる．提案する実数値 GA 専用プロセッサでは, これまで対象とする問題の変更によって再設計が必要であった評価計算部分をソフトマクロ汎用 CPU によって実行する．これにより，回路の再設計が不要になりプログラムの変更によって対応可能となる．また，リソースシェアリングを適用することによって，必要な演算器を削減して回路規模を削減している．これにより，最適化すべきパラメータ数が増加しても，少ない回路規模で対応できる．

さらに，提案する実数値 GA 専用プロセッサを FPGA に実装し，ベンチマーク問題により性能評価を行った結果について述べる．その結果，実行時間において，汎用 PC と比較して，最高で約 2.9 倍の高速化が確認できた．

第 4 章「実数値 GA 専用プロセッサを用いたディジタルフィルタの最適設計」

提案する実数値 GA 専用プロセッサの応用として, ディジタルフィルタへ適用した結果について述べる．一例として，バンドストップフィルタを設計した結果を述べ，さらにディジタル補聴器のフィッティングを想定したシミュレーションを行った結果を示す.

第 5 章「結論」

本章では，本研究の総括と今後の展開を述べる.

第2章　　進化型ハードウェアの概要

2.1　序

進化型ハードウェアには，進化アルゴリズムを実行する回路が組み込まれ，それにより回路構成や回路定数を決定する．本章では，進化型ハードウェアの構成，進化アルゴリズムとその専用プロセッサの概要について述べる．進化アルゴリズム専用プロセッサの一例として，リソースシェアリングを適用した遺伝的アルゴリズム専用プロセッサについても述べる．

2.2 進化型ハードウェアの構成

進化型ハードウェアは，進化アルゴリズムによって回路構成や回路定数を決定する．その特徴として，回路を設計する必要がないことや，使用環境に応じた回路定数の自動調整ができることなどがある．

進化型ハードウェアの構成の一例を図 2.1 に示す．まず，進化アルゴリズムにより，解の初期集団を生成する．各個体は進化させる対象の回路構成情報や回路定数に対応する．各個体に対して，その進化アルゴリズム特有の処理を行い，子を生成する．生成された子の回路情報に従って，進化させる回路を構成する．そして，子によって構成された回路に入力信号を与え，回路の出力信号を得る．その出力信号と目標とする回路の出力信号である目標信号の差分を計算する．その計算値が回路構成情報の適合度となる．この適合度を基に，その個体が生き残るかを決める．上記の処理を繰り返すことにより，出力信号が目標信号に近づくように回路の構成が進化する．

使用される進化アルゴリズムは，遺伝的アルゴリズム（Genetic Algorithm; GA），遺伝的プログラミング（Genetic Programing; GP），粒子群最適化（Particle Swarm Optimization; PSO） [10]など様々である．

また，進化する回路は，ディジタル回路，アナログ回路のどちらにも適用できる．ディジタル回路の場合には，再構成可能な LSI である FPGA（Field-Programmable Gate Array）などが用いられ，本研究においてもこれを用いる．アナログ回路の場合には，再構成可能なアナログ IC である FPAA（Field-Programmable Analog Array）などが用いられる．

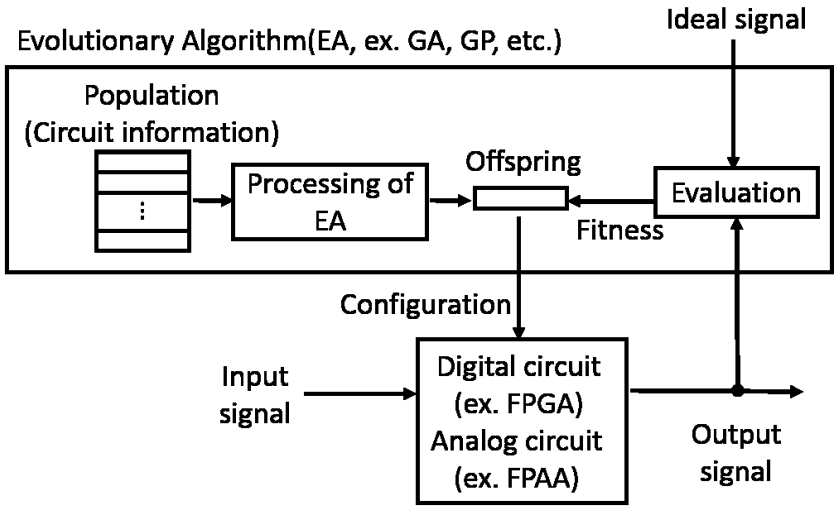


図 2.1. 進化型ハードウェアの構成の一例

2.3 進化型ハードウェアの応用例

進化型ハードウェアはアナログ回路，ディジタル回路，システムなど幅広く適用できる．これまで様々な構成の進化型ハードウェアが提案されている [11, 12].

アナログ回路を対象としている事例では，遺伝的アルゴリズム（GA）によってコンデンサの容量を調整するアナログ回路 [13]，マイクロ波回路への応用 [14]，再構成可能なアナログ回路である FPAA を進化させて筋電を測定している事例 [15]，GA を用いたアナログ回路設計 [16]，蟻コロニー最適化手法（Ant colony optimization; ACO）を用いたアナログフィルタ設計 [17]など数多く存在する．

また，ディジタル回路を対象としている事例では，ハードウェアには FPGA がよく用いられる．ハードウェアの高速性と並列性を活かした画像処理の高速化 [18]や，通信方式の変更が著しい通信の分野にしばしば用いられる．使用される進化アルゴリズムは，カルテシアン遺伝的プログラミング（Cartesian genetic programming; CGP） [19]に基づく手法，GA，差分進化（Differential evolution; DE） [20]などが用いられる．順序回路の自動設計 [21]だけでなく，文献 [22, 23, 24, 25]では，各種演算を動的に切り替えられるプロセッシングエレメント（PE）を多数配置し，その接続や演算などの構成を GA プロセッサで設定し，画像フィルタを実現する回路が提案されている．これらの手法はノイズ除去などに有効である [24, 3]．また，PE を用意し，GPU（Graphics Processing Unit）を用いて構成を決定する手法も提案されている [26]．文献 [27, 28]ではセルラーニューラルネットワーク [29]を構成し，各ニューロンを 1 画素に割当て，ニューロンの結合係数を進化アルゴリズムで決定している．さらに，FPGA に提供されている機能である部分動的再構成機能を使用してパターン認識を行う事例 [30]も報告されている．部分動的再構成とは，回路の動作中に FPGA の一部の領域を再構成する技術である．

また，個人によって異なる生体信号を扱う事例にも進化型ハードウェアは有用である．GA を用いて GAL (Generic Array Logic)を再構成することで適応させる筋電制御義手 [2]，CGP（Cartesian Genetic Programming）を用いた筋電のパターン分類 [31]，パターン認識 [32]などの事例がある．

さらに，産業への応用も多く存在し [33]，LSI 製造後のクロック調整 [34]， NASA がアンテナ設計 [35]に用いた事例などがある．さらに，リカレントニューラルネットワークを並列 GA で進化させるロボット [36]や，無人航空機のための経路探索を FPGA に実装 [37]した事例など幅広く応用されている．

2.4 進化アルゴリズムの概要

進化型ハードウェアでは，進化アルゴリズムにより回路構成や回路定数を決定する．多くの工学的な問題において，解法が知られていなかったり，莫大な計算コストを必要とするものが少なくない．このような困難な問題に対して，ヒューリスティックなアプローチが有効である．すなわち，実用上十分な精度を持つ近似解を経験的な方法により，現実的に許容できる時間内で求める．その一手法として，進化アルゴリズムが知られている［38, 39, 40］．この手法は，組み合わせ最適化問題や NP 困難な問題など様々な分野の最適化問題に適用できる．

これまで，様々な進化アルゴリズムが提案されている．1975 年ジョン・H・ホランド氏によって遺伝的アルゴリズム（GA）［6, 7, 8］が提案された．GA は，生物の進化に着想を得た近似解探索手法の一つであり，組み合わせ最適化問題や NP 困難な問題など，様々な問題に適用することができる．また，数式やプログラムを遺伝的操作によって生成する遺伝的プログラミング（Genetic Programming; GP）や，GP をディジタル回路設計に適用したカルテシアン遺伝的プログラミング(Cartesian GP; CGP) [19]などがある．

また，実数を扱えるアルゴリズムとして，GA の実数値版である実数値 GA［41, 42］が提案され，注目されている．その他，実数のベクトルで解を表し、探索を行うと同時に自己変異用のパラメータも更新する進化戦略（Evolution Strategy; ES）［43］，さらに発展させた共分散行列適応進化戦略（Covariance Matrix Adaptation Evolution Strategy; CMA-ES）［44］などが提案されている．また，群知能と呼ばれる集合の振る舞いに基づくアルゴリズムも提案されている．1995 年には鳥や魚の群れの振る舞いを解探索に応用した PSO（Particle Swarm Optimization）［10］，蟻の採餌行動の振る舞いに着想を得た ACO（Ant Colony Optimization）［45］などが提案され，様々な分野に用いられる．これらのアルゴリズムは，提案されてから今もなお様々な改良が研究されている［46］.

2.5 進化アルゴリズム専用プロセッサ

一般に，ソフトウェアによる処理において長い計算時間を必要とする場合，専用ハードウェアを用意することが有効である．また，ハードウェア化することにより，処理の高速化だけでなく，高度なアルゴリズムでも組み込み分野に適用することができ，応用範囲も広がる．これまで進化アルゴリズム専用プロセッサは数多く提案されている．

PSO では文献 [47, 48, 49, 50, 51]などで提案されている．文献 [48]では PSO と GA を組み合わせて経路探索を行った事例が報告されている．また，文献 [49]では浮動小数点数ベースの専用ハードウェア，さらに文献 [50, 51] では，Altera 社のソフトマクロ CPU NiosII を組み込んだプロセッサなどが提案されている．また，ACO の専用ハードウェア [52]やファジイコントローラに適用した事例 [53]などが報告されている．その他，ABC (Artificial Bee Colony)アルゴリズム専用のハードウェア [54]，DE (Differential Evolution)専用のハードウェア [55]などが提案されている．

GA では文献 [56, 57, 58, 59, 60, 61, 62, 63, 64, 65]など数多く報告されている．ビットストリング形式の GA において，交叉，突然変異等の GA 特有の処理はハードウェア化しやすい．しかし，評価計算部分は問題によって異なるため，問題ごとにハードウェアを設計する必要がある．そのため，GA 特有の処理をハードウェア化し，外部で GA のパラメータ設定と評価部分を実行することにより，汎用性を高めたプロセッサが文献 [56, 57, 58]で提案されている．文献 [56]では，評価を分離した GA 特有の処理を IP コア(Intellectual Property core, LSI を構成するための部分的な回路情報)として設計した回路が提案されている．染色体長として 16 ビット扱える回路を基本のコアとし，それらを並列に使用することにより拡張可能となっている．文献 [56, 57]では，1 チップの FPGA 上に，GA 特有の処理を行うハードウェアと，評価計算や GA パラメータの設定などを実行する CPU を実装した回路が提案されている．これにより，扱う問題が変更されても，GA パラメータと評価関数のプログラムの変更だけで対応可能となっている．

また，GA を並列化した並列 GA [66] が種々提案されている．並列化し，個体の交換を行うことにより，局所解への収束が緩和され，探索能力および速度が向上する利点がある．並列化の粒度によって種類があるが，文献 [60] では，種類の異なる並列 GA に対応できるフレームワークを提案している．さらに，並列 GA ハードウェアを適用した事例として，文献 [61] ではロボットの経路探索，文献 [62] では巡回セールスマン問題に適用した事例が報告されている．

ハードウェア化に向いた GA として，集団を 1 つの確率ベクトルとして扱う Compact GA(cGA) [67] が提案されている．集団を表す確率ベクトルから個体を生成するため，全個体を保存する必要がないのでメモリが節約できる．文献 [63]では，cGA に突然変異とエリート戦略を付加した回路を提案している．文献 [64] では，ブロック構造ニューラルネットワーク [68]の構造と重みを，並列化した cGA を用いて決定する回路を提案している．

これらの進化アルゴリズム専用プロセッサは, 扱う対象が変わると評価回路に変更が必要なものが多く，また実数値を扱うものは少ない.

動的再構成技術を応用した GA 専用プロセッサ

一般に，回路の高速化のためには，回路規模を縮小し，並列度を増やすことが効果的である．回路面積の削減に有効な手法として，動的再構成技術 [69]が知られている．これまでに，動的再構成を適用した GA 専用プロセッサ [70, 71, 72, 73, 74]を提案した.

動的再構成

動的再構成技術とは，回路内部の構成要素の機能や接続を，動的に変更させて面積効率の改善を図る技術である．再構成可能デバイスである FPGA や CPLD を使用したものや, その他様々な動的再構成技術を使ったシステムが研究・開発されている [75]. 動的再構成の主な実現方法は，命令/構成データ配送方式とマルチコンテキスト方式の 2 つがある．命令/配送データ方式は，主に粗粒度構成のデバイスで採用される．回路の構成データを，チップ内のメモリに分散させて持たせておき，これらを専用のバスにより各プロセッサや接続スイッチに配送することにより，動的再構成させる方式である，マルチコンテキスト方式とは，構成データを記憶するメモリを複数持ち，これらをマルチプレクサで切り替えることにより回路構成を変える方式である. 近年の汎用的な動的再構成のプロセッサとして MuCCRA-4 [76]，STP エンジン [77]などがある．また，動的再構成を用いた特定用途向け専用回路が文献 [78, 79]で提案されている.

個体数と精度を動的に変更可能な GA 専用プロセッサ

一般に，個体数を増加させると解候補の数や多様性が増加するため，広く解空間を探索することができ，探索能力が向上することが知られている．しかし，次世代を個体数だけ確保するまで遺伝的操作(選択，交叉，突然変異)を繰り返す GA において，個体数を増加させると実行時間も増加してしまう.

通常の GA において，世代数，解候補の精度(染色体のビット数)，個体数は常に一定であり，実行中に変化することはない．GA における探索の過程において，探索の初期では，広範囲に解を探索し(図 2.2 (a))，世代数を重ねるに従って，その最適解の近傍を詳細に探索する(図 2.2 (b))と考えられる.

したがって，探索の前半では，解空間全体を広範囲に探索する必要があるため個体数が多いほうが望ましい. また, 大まかな探索でよいため, 解の精度はあまり重要でない. 一方，探索の後半では，最適解を求められる十分な解の精度が必要である．前半におけ

る探索で，解の範囲が狭められていると仮定すると，少ない個体数でも探索効率を低下させることなく探索できると考えられる．

この考えを基に，個体数と解の精度を動的に変更する手法をこれまで提案した［70, 71, 72, 73, 74］．探索の前半では個体数が重要であるため，解候補のビット数を半分にして精度を落とし，適当な個体群の個体数を 2 倍にして探索を行う(図 2.3 (a))．そして，ある世代数になると，個体群の個体数を半分にして，解候補のビット数を 2 倍して，解の精度を上げて探索を行わせる(図 2.3 (b))．

図 2.2. GA の探索の様子

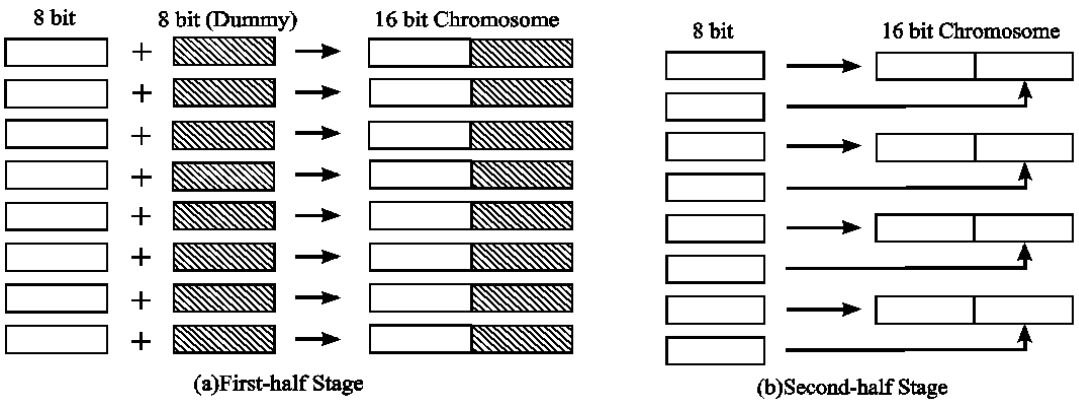


図 2.3. 動的再構成メモリの概要図

GA 処理を実行中に，解候補の精度(染色体の長さ）と個体数をこのように変更させれば，効率的な探索を行うことができると考えられる．また，所望の個体数の半分で，探索能力を低下させずに探索を行うことができる．この個体数の削減は，染色体と適合度を保存するメモリ容量の削減に貢献し，処理時間の短縮にもつながる．

提案する手法における染色体を保存するメモリの概念図を図 2.3 に示す. 図 2.3 は，例として 8 ビットの染色体を 8 つ保存できるものを示している．つまり，解の精度は 8 ビット，個体数は 8 つである．探索の前半では，8 ビットのダミーデータを付加して，擬似的に 16 ビットの染色体を 8 つ作り出す(図 2.3 (a))．後半では，8 ビットデータの 2 つのメモリ要素を連結させ，16 ビットの染色体を 4 つ作り出す(図 2.3 (b))．染色体を保存するメモリがこのように振舞えば，前述の方法を実現できる．

上記の動的再構成メモリを持つ GA 専用プロセッサを，ハードウェア記述言語の VHDL を用いて設計し，FPGA に実装して実装実験を行った．設計したプロセッサのブロックダイアグラムを図 2.4 に示す. 組み合わせ最適化問題の一例として，ナップザック問題に適用した. 結果として，通常の GA より探索能力が改善することを確認した.

上記のように，動的再構成技術は進化アルゴリズム専用プロセッサにおいても有用であり，回路規模の縮小が期待できる．

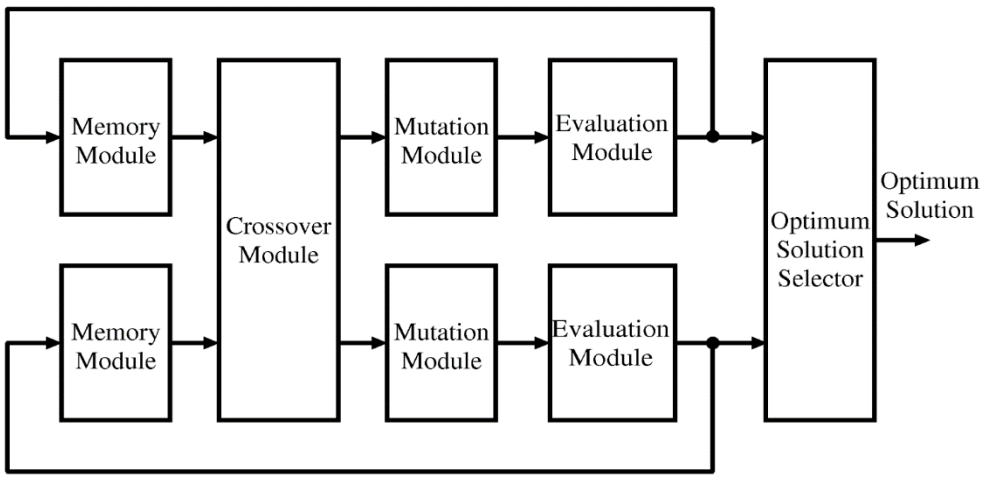


図 2.4. 動的再構成メモリを持つ GA プロセッサのブロックダイアグラム

2.6 結言

本章では，進化型ハードウェアの概要と応用例について述べ，回路構成や回路定数を決定する進化アルゴリズムとその専用プロセッサの構成について述べた．これまで様々な進化アルゴリズム専用プロセッサが提案されているが，多くのプロセッサでは対象とする問題を変更する際には，評価回路の再設計が必要である．また，実数を扱えるものもまだ多くない．

また，進化アルゴリズム専用プロセッサの一例として，動的再構成技術を適用したGA専用プロセッサについて述べた．動的再構成を適用することにより，進化アルゴリズム専用プロセッサにおいても，回路規模の縮小が期待できることを示した．

第3章　実数値 GA 専用プロセッサの開発

3.1　序

本章では，汎用性の高いアーキテクチャの確立を目的として開発した，実数値 GA 専用プロセッサ [80, 81, 82]のアーキテクチャについて，詳細に述べる．

従来，進化型ハードウェアの構成の決定にはビットストリング形式や整数のアルゴリズムが用いられることが多かった．しかし，実数に基づくアルゴリズムを採用することにより，さらに汎用性が高まると考えられる．

開発した実数値 GA 専用プロセッサのアーキテクチャの特長は，まず，評価関数をソフトマクロ汎用 CPU で計算するため，進化させる対象が変わってもプログラムの変更だけで対応できることである．さらに，ソフトマクロ汎用 CPU を複数個実装し，並列に評価を行うことで浮動小数点の評価計算時間を短縮し，扱える問題の規模も増やしている．そして，リソースシェアリングを適用し，演算器などの回路資源を有効に共有することで回路規模を少なく実装できることである．共有しない場合に比べ，1/3 から 1/4 の演算器数で実装可能である．さらに 1 チップの FPGA で実装することができる．

提案する実数値 GA 専用プロセッサを FPGA に実装し，3 つのベンチマーク問題に適用した結果を報告し，提案する実数値 GA 専用プロセッサの有効性を示す

3.2 実数値 GA 専用プロセッサを用いた進化型ハードウェアの概要

進化型ハードウェアにおいて，進化計算を行う部分は重要である．2.5 節で示した多くの進化アルゴリズム専用プロセッサは，バイナリ形式や整数を扱う構成であり，実数を扱う構成は少ない．

実数による最適化が望ましいような応用，即ち，アナログ回路の回路定数の調整，ニューラルネットワークの重み調整，ディジタルフィルタの係数設定などでは，実数を直接扱う進化アルゴリズムを用いることが有効と考えられる．したがって，実数を直接扱うことができ，通常の GA よりも探索能力の高い実数値 GA によって，回路構成や回路定数を決めることが有効であると考えられる．

そこで，本研究では実数値 GA 専用プロセッサを開発する．

3.3 実数値 GA 専用プロセッサの特長

提案する進化型ハードウェアは，実数値 GA 専用プロセッサによって回路構成や回路定数を決める．実数値 GA は，実数値をそのまま扱うことができ，遺伝子型をビットストリングで扱う従来の GA より解探索能力に優れ，未知パラメータの多い高次元の問題に対しても有効である．

さらに，提案する実数値 GA 専用プロセッサには，これまでの進化アルゴリズム専用プロセッサの課題であった以下の問題点を改善している．

従来，対象とする問題を変更する際は評価回路の再設計が必要であった．しかし提案する実数値 GA 専用プロセッサでは，ソフトマクロ汎用 CPU により，評価関数の計算を行う．これにより，プログラムの書き換えだけでさまざまな問題に対応させることができる．

さらに，回路を実行中に動的に共有することで少ない演算器で実数値 GA 特有の処理を実現し，演算器を削減する．すなわち，回路資源を有効に共有することで回路規模を少なく実装できる．これにより，共有しない場合に比べ，1/3 から 1/4 の演算器数で実装可能である．また，提案する実数値 GA 専用プロセッサは，1 チップの FPGA に実装することができる．

3.3.1 ソフトマクロ汎用 CPU を用いた評価回路

進化アルゴリズム専用プロセッサにおいて，評価回路は動作する頻度が多く，その性能は全体の実行時間に大きく影響する．そのため一般的に，評価回路は専用回路で実現されることが多い．ハードウェア記述言語を用いて設計する場合，未知パラメータ数が数個と少なく，加減乗算の組み合わせ，且つ計算回数が少ない評価関数の回路を設計する場合には，回路の設計・検証は容易で工数少なく，回路規模・実行速度・品質のどれも優れた専用回路が実現できると考えられる．

しかしながら，未知パラメータが数十から数百個の場合や，評価関数に除算や三角関数などの計算が必要な場合，回路設計は難しくなる．未知パラメータ数が増加すると，回路規模は増加し，それにより動作周波数の低下が予想される．そのため，計算の並列化やパイプライン化などのハードウェア実装上の工夫が必要となる．また，除算や三角関数は，IP コア（Intellectual Property core, LSI を構成するための部分的な回路情報）などを用いても，複数クロックを使用して計算するため，タイミング制御が複雑になる．さらに，回路規模の面においても評価関数の複雑な場合には，回路規模が増加し，実際の FPGA などのハードウェアに実装することが不可能な場合も生じる．その他，回路の外部から設定・変更するパラメータ等がある場合には，それを作り込む必要がある．また，検証においても確認する事項が多く複雑であれば，検証作業にも時間を要する．さらに実装において，回路に少しの変更を加えた場合でも配置配線は変わるため，回路合成はやり直しとなり，回路規模が大きければ合成時間は数時間以上かかることもある．

そこで，本研究では評価はソフトマクロ汎用 CPU で実行する．これにより，回路設計が不要となり，プログラムの変更のみで様々な問題に対応可能となる．さらに，未知パラメータ数の変更や，外部からのパラメータ設定などの変更も容易に行え，回路の再合成は不要となる．また，回路規模も使用するソフトマクロ CPU の回路規模で決まり，評価関数の複雑さに左右されない．その他，ベンダが提供するソフトマクロ CPU を使用する場合は，開発環境が充実しているため，C 言語などの高級言語によってプログラム開発が可能であり，デバッグ作業も容易に行える．

しかしながら，ソフトマクロ汎用 CPU の性能は，汎用 PC に搭載されている CPU と比べると性能は劣る．また，未知パラメータ数の増加に伴い，計算時間は長くなる．そのため，ソフトマクロ汎用 CPU を多数搭載し，並列に計算することにより，その性能差を緩和している．除算や三角関数など実行時間を要する関数も，実行時間が許す範囲で使用できる．さらに，動作中にプログラムによって評価関数を適応的に変更することなども可能と考えられる．

したがって，本研究で設計したプロセッサは柔軟性を持ちつつ，並列性を高めた構成と考えられる．図 3.1 に実数値 GA 専用プロセッサの構成の概要図を示す．

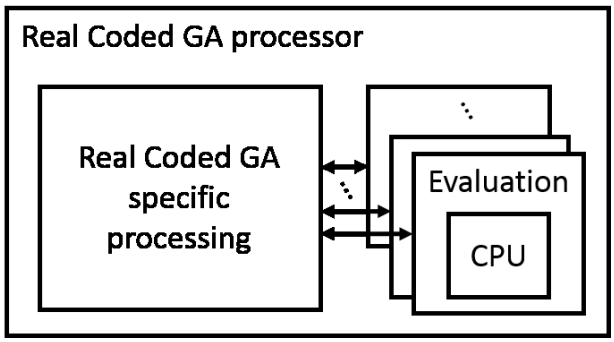


図 3.1. 実数値 GA 専用プロセッサの構成の概要図

3.3.2 リソースシェアリングを用いた浮動小数点演算器の削減

最適化問題において，問題の次元数（最適化するべきパラメータ数）を上げると実行時間が増加する．進化アルゴリズム専用プロセッサの高速化のためには，並列度を上げる必要があるが，その分回路規模も増大する．しかしながら，常に全ての回路が同時に動作しているわけではない．多くの進化アルゴリズムにおいて，解の良し悪しを判断する評価計算が必要である．複雑な問題であれば，評価回路も複雑になり，実行時間も多くかかる．その評価結果によって，次世代に残る個体が決まる．すなわち，すべての子個体の評価計算が終了しなければ次世代の処理を行うことができないため，評価計算の時間が長ければ，待ち時間が発生してしまう．そのため，評価計算の実行時間を考慮し，演算器を共有することで，回路の削減ができると考えられる．

そのような事例には，リソースシェアリングが有効である．これまで，動的再構成可能な融合型積和演算回路 [83, 84, 85]を提案した．実数値 GA における交叉の計算は，主に積和演算である．限られた個数の浮動小数点演算器を用いて，その接続を変更することにより，少ない演算器で処理を実現する．演算器が削減された結果，回路規模は小さくなり，更なる並列化につながることが期待できる．図 3.2 にリソースシェアリングを適用した回路の概念図を示す．

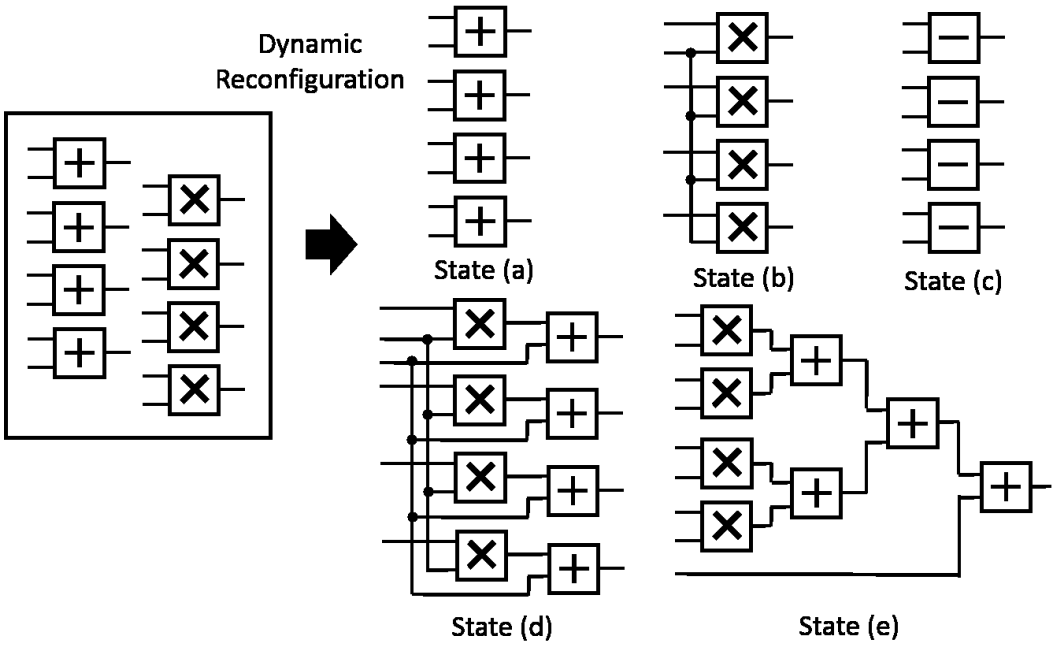


図 3.2. リソースシェアリングを適用した回路の概念図

3.4 実数値 GA

近年では，遺伝子型に実数値を扱う実数値 GA の研究が盛んである [42]．これは遺伝子型をビットストリングで扱う従来の GA より解探索能力に優れ，未知数の多い高次元の問題に対しても有効である．

また，実数値をそのまま扱えるため，適用分野も広い．例えば，レンズ系設計に適用した事例 [86]や，ファジィ推論と組み合わせて，入院患者のベッド上における転倒危険度のモデル化に適用した事例 [87]などが報告されている．その他，実数値 GA は実数をそのまま扱えるため，制御関係のパラメータ調整などに用いられる．実数値 GA とニューラルネットワークを組み合わせて倒立振子の制御 [88]を行った事例や，ロボットの制御 [89]などで用いられた事例がある．

従来の GA では，遺伝子型にビットストリングを使用している．しかし，変数ベクトルをビットストリングに変換して交叉・突然変異などの遺伝的操作を施すと，変数間依存性の情報が失われ，親の特徴が破壊されるという問題点があった．これに対して実数値 GA は，実数値ベクトルを直接扱うため，親個体群の近傍に親の特徴を壊すことなく子を生成することができる．

実数値 GA の主な処理は，個体の複製選択および生存選択を行う世代交代モデルと交叉からなる．現世代集団から複製選択により親を選び，交叉によって子個体群を生成して，生存選択により集団に子を戻して次世代集団を生成する．この繰り返しにより最適解を求める．実数値 GA のフローチャート図 3.3 に示す．

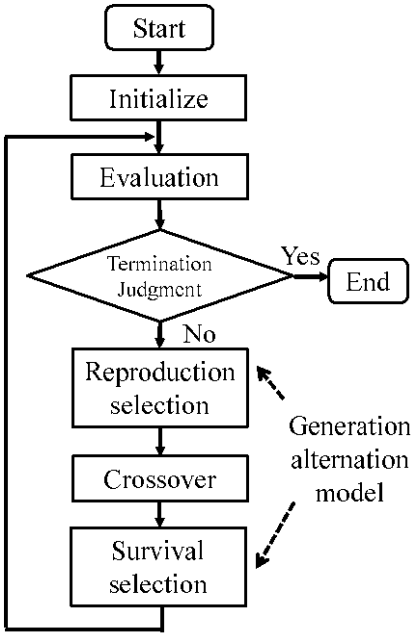


図 3.3. 実数値 GA のフローチャート

3.4.1 世代交代モデル JGG

実数値 GA における世代交代モデルには，MGG（Minimal Generation Gap） [90] や JGG（Just Generation Gap） [91] などが提案されている．MGG では複製選択と生存選択に用いる親は 2 つであるが，JGG では 2 つ以上の親を用いる．また，MGG よりも JGG を用いた方が良い結果が得られるという報告がある [92]．そこで，提案する実数値 GA 専用プロセッサでは，世代交代モデルに JGG を採用する．

世代交代モデル JGG では，以下の 3 つの処理を行う．

(1) 複製選択：集団から N_p 個の個体をランダムに非復元抽出して，交叉に用いる親個体群とする
(2) 子の生成：親個体群に対し交叉を繰り返し適用して，N_{os} 個の子個体を生成する
(3) 生存選択：子個体群から評価値が上位 N_p 個の個体を選び，複製選択で抽出した親個体と置き換える

図 3.4 に JGG の概念図を示す.

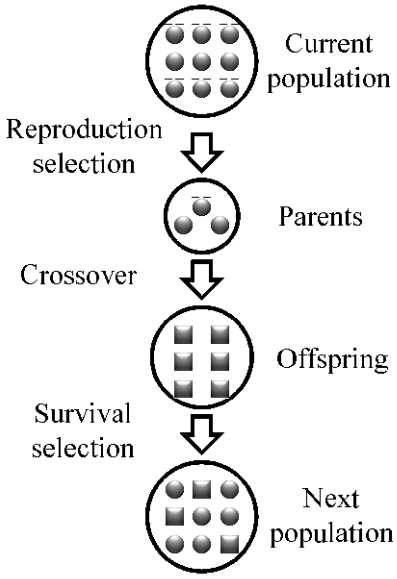


図 3.4. JGG の概念図

3.4.2 多親交叉 REX

実数値 GA における交叉には，BLX-α [93]，単峰性正規分布交叉（Unimodal Normal Distribution Crossover; UNDX） [94]，シンプレクス交叉（Simplex Crossover; SPX） [95]，多親交叉 REX（Real coded Ensemble Crossover） [42] などが提案されている．BLX-α は初期に提案されたものであり，性能は他に比べて高くない．また，ハードウェア化する場合には UNDX と SPX は計算コストが高い．REX は上記 2 つよりも計算コストが少なくかつ性能が良い．提案する実数値 GA 専用プロセッサでは，REX を採用する．

交叉 REX は複製選択された親個体群から式(3.1)より，子個体群を生成する [42].

$$x^c = x^g + \sum_{i=1}^{N_p} \xi^i (x^i - x^g), \qquad \xi^i \sim \varphi(0, \sigma_\xi^2) \tag{3.1}$$

$$\sigma_\xi^2 = \frac{1}{N+k} \tag{3.2}$$

ここで，次元数 Nの実数値ベクトルを $\boldsymbol{x}$，親個体群を $\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^{N+k}$，重心を $\boldsymbol{x}^g$，子を $\boldsymbol{x}^c$とする．k は，$1 \leqq k \leqq P\text{-}N$ (Pは集団数) の範囲で任意に設定する数である．$\varphi(0, \sigma_\xi^2)$は，平均が 0，分散が σ_ξ^2 の任意の対称な確率分布を表し，ξ_iは確率分布 $\varphi(0, \sigma_\xi^2)$に従うパラメータである．σ_ξ^2 は式(3.2)で設定する．φ に$[-\alpha, \alpha]$ の区間の一様乱数を用いる場合には，$\alpha = \sqrt{3/(N+k)}$ のように設定する．まず，親個体群における重心を計算し，重心に各親の重心からの偏差に乱数を掛けた値を加えることで新しい子個体を生成する．図 3.5 に 2 次元における REX の概念を示す. 子は図 3.5 において点線内の領域に生成される．

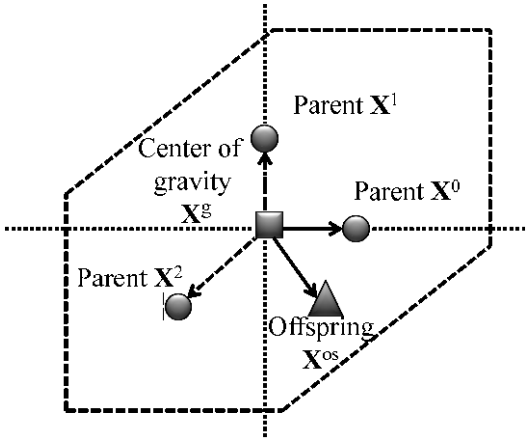


図 3.5. REX の概念図

3.4.3 実数値 GA の改良

近年，実数値 GA は様々な改良がされており，REX をさらに改良した AREX [96]が提案されている．AREX では稜構造関数や多峰性関数の最適化において発生する初期収束を回避するためのメカニズムが備わっている．さらに，AREX と JGG を基に大域的多峰性関数のために最適化された Big-valley Explorer(BE) [97]が提案されている．いくつかのベンチマーク関数において，BE が MS-CMA-ES (探索空間全体に集団を初期化する CMA-ES)よりも良い性能であることが報告されている．

3.5 実数値 GA 専用プロセッサの構成

提案する実数値 GA 専用プロセッサは，世代交代モデル JGG に従いランダムに親を選択し，交叉 REX によって子を生成する．1 つ目の子が生成されるとソフトマクロ汎用 CPU にて評価が開始される．それ以降，子の生成と評価は並行して行われる．

ソフトマクロ汎用 CPU を複数個搭載し，複数の子を並行して評価する．CPU による評価計算が終了次第，次世代の集団に残す子の選択を行い，新しい子で集団を更新する．この繰り返しにより最適解を探索する．

実数値 GA において設定するパラメータは，次元数 N，親個体数 N_p，子個体数 N_{os}，個体数 Pの 4 つである．次元数以外のパラメータは次元数を基準に決定される．N_pは N+k，N_{os}と Pは Nの整数倍に設定される．提案する実数値 GA 専用プロセッサでは，N_pを N+2 かつ 2 の冪乗の数として，N_pを基に並列化する．すなわち，演算回路や評価用 CPU など主要な回路は N_p個ずつ存在する．N_{os}は N_pの整数倍で設定する．この整数を N_{osr}とする．N_p個存在する回路要素を使用して，N_{osr}回繰り返し行うことにより，N_{os}個の子を生成する．表 3.1 に提案する実数値 GA 専用プロセッサのパラメータ設定を示す．

提案する実数値 GA 専用プロセッサのブロックダイアグラムを図 3.6 に示す．各次元のデータは IEEE754 に準拠した単精度浮動小数点数として扱う．プロセッサは以下の(a)～(h) のブロックより構成される．

(a) メモリ（集団，親，子，最適解）
(b) 乱数発生回路
(c) ランダム選択回路
(d) REX 回路
(e) 評価回路
(f) ソート回路
(g) 初期化回路
(h) 更新回路

各ブロックの概要を以下に示す．

表 3.1 実数値 GA のパラメータ

Parameter	General RCGA	Proposed processor
Dimensions N	N	$Np-2$ ($k=2$)
Number of parent N_p	N+k	N_p
Number of offspring N_{os}	αN	$N_{osr}\times N_p$
Number of individual P	βN	βN

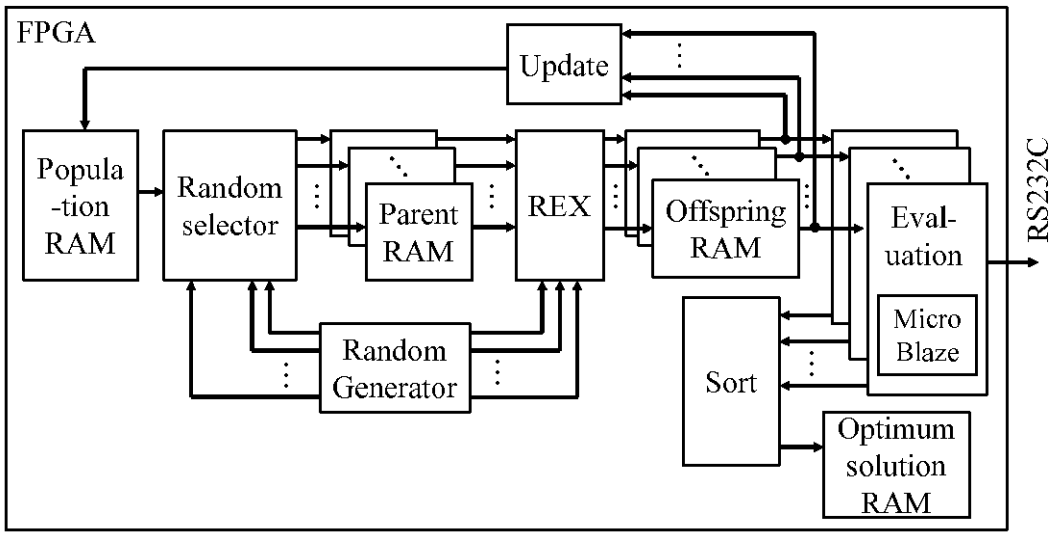


図 3.6. 実数値 GA 専用プロセッサのブロックダイアグラム

3.6 メモリの内部形式

集団メモリは次元数 N個の要素を持つ P組の個体を保存する．親メモリおよび子メモリは，N_p組用意する．最適解メモリは，最も適合度の高い個体 1 組を保存する．各親メモリは 1 個体あたり N個の単精度浮動小数点数を保存する．最適解メモリも同様である．また，各子メモリは N_{osr}組の個体を保存する．集団メモリと子メモリのアドレスは，上位ビットが個体の番号，下位ビットが各要素の番号を表す．メモリの内部形式を図 3.7 に示す．

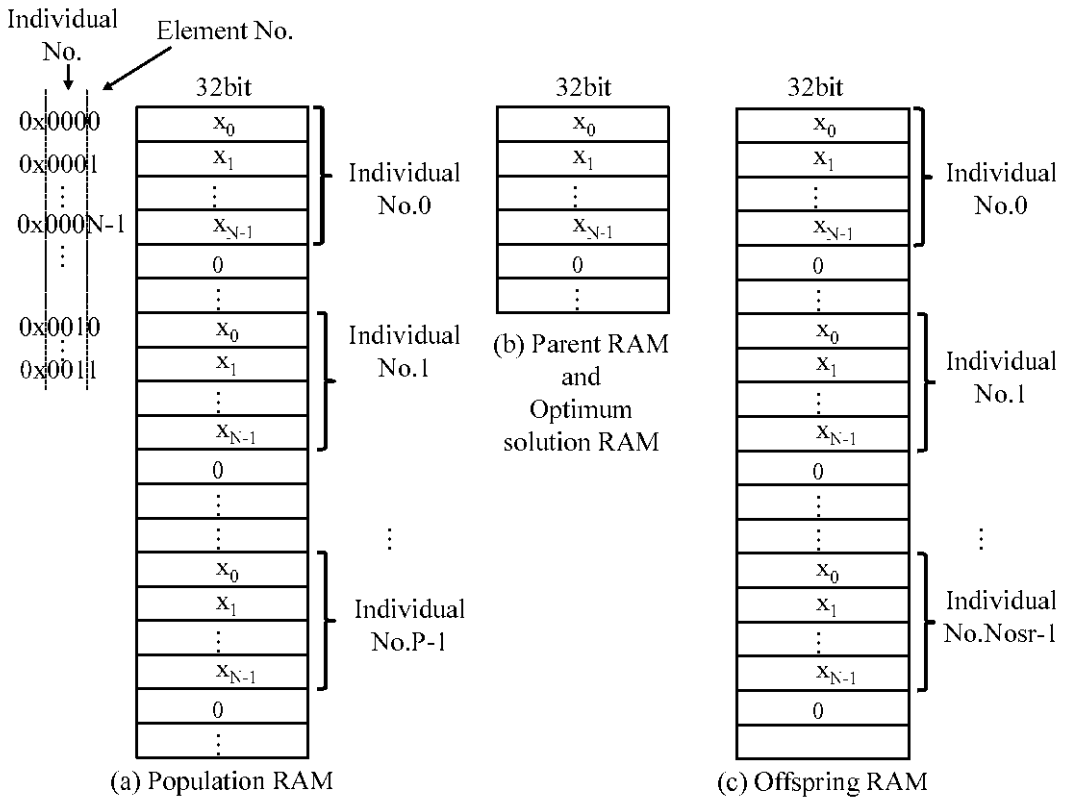


図 3.7. メモリの内部形式

3.7 世代交代モデル JGG の実装

世代交代モデル JGG の実現のために以下の回路設計を行った．複製選択のために乱数発生回路およびランダム選択回路，生存選択のためにソート回路と更新回路である．

3.7.1 複製選択の実装

乱数発生回路

この回路は M 系列乱数発生回路の N_p 個の出力を IEEE754 単精度浮動小数点形式に変換して出力する．まず，23 ビットの乱数を N_p 個発生させる．ビット数は単精度浮動小数点の仮数部 23 ビットと合わせている．出力された乱数を，整数部なし，小数部 23 ビットの[0, 1.0)の固定小数点として扱い，N_p 個の浮動小数点変換回路で同時に変換して出力する．また，M 系列乱数回路の出力の下位 $\log_2 P$ ビット(P_L)は，ランダム選択において親として選択する番号としても使用する．

ランダム選択回路

乱数発生回路からの乱数を N_p 個取得し，その値が P 以下ならば集団メモリから親として選択する個体番号とする．N_p カウンタの出力($\log_2 N_p$ ビット, N_{pL}) に従って，この番号を選択する．選択した番号の下位ビットに N カウンタの出力($\log_2 N$ ビット, N_L) を付加してアドレスとして集団メモリに与え，集団メモリから順次データを読み出して親メモリに保存する．1 個体読み出しが完了したら親個体数をカウントして，選択する個体番号と保存する親メモリを切り替える．この親の読み出しと並行して，REX 回路では各次元の総和を計算する．親を N_p 組読み出したら終了する．

3.7.2 生存選択の実装

ソート回路

生存選択において，次世代に保存する個体を N_p 個選ぶ必要がある．この回路は各個体の適合度に基づいて，昇順に整列しながら上位 N_p 個の子の番号を保存する．主な回路構成として，N_p 個の浮動小数点比較器と，上位 N_p 個の適合度を保存するレジスタと対応する子番号のレジスタからなる．評価回路による評価が完了すると，この回路に評価回路で計算された適合度が個体番号と共に順次入力される．そのとき，入力された適合度と現在の上位 N_p 個の適合度を同時に比較する．また同時に，1 つ上の順位の比較結果との排他的論理和により，新しい個体の挿入する位置を割り出す．その後，挿入位置より下位の適合度レジスタと子番号レジスタを同時に 1 つ下位のレジスタにずらして保存する．このとき，最下位の個体の適合度と個体番号は消去される．そして適合度と個体番号を挿入位置に挿入して保存する．上記の処理を行うことで，すべての子の適合度の入力が完了すると，レジスタの配列は整列済みであり上位 N_p 個の個体が求まる．また，レジスタの配列の 0 番目の個体番号の個体が，その世代における最良解となる．

更新回路

この回路では，ソート回路によって求まった上位 N_p 個の子の番号に従い，子メモリから各個体のデータを読み出し，親として選択された集団メモリの個体に上書きを行う．上書きされる親は，ランダム選択回路に保存されている親の個体番号のレジスタに従う．さらに，ソート回路の 0 番目の個体の適合度と現在の最適解の適合度が比較され，新しい個体の適合度が高い場合には，最適解メモリをこの個体で更新する．この最適解メモリは 1 つのソフトマクロ汎用 CPU (Xilinx 社 MicroBlaze)に接続されており，実数値 GA の処理が終了次第，実行結果としてこのメモリの内容を読みだしてシリアル通信で PC に送信する．

3.8 リソースシェアリングを適用した多親交叉 REX 回路の実装

REX 回路では，ランダム選択回路により選択された親を用いて式(3.1)の計算を行い，子を N_{os} 個生成する．この回路では，式(3.1)の計算を以下の 5 つの手順に分割して実行する．

(1) 総和計算：ランダム選択時に並行して親の各次元の総和を求める
(2) 重心 x^g の計算：親の各次元の総和と親個体数の逆数の乗算により重心 x^g を求める
(3) 偏差($x^i - x^g$)の計算 : 重心と各親の偏差を求め，その結果で親メモリを上書きする
(4) 乱数(ξ) の生成：[-α, α] の N_p 個の乱数を生成し，レジスタに格納する
(5) 子生成：偏差を読み出し，乱数と偏差を乗算して総和をとり，子を生成する

REX 回路のブロックダイアグラムを図 3.8 に示す．この回路は主に，N_p 個の単精度浮動小数点乗算器と単精度浮動小数点加算器，総和を保存するレジスタ，重心を保存するレジスタ, および乱数を保存するレジスタからなる. また, 浮動小数点演算器は Xilinx 社の IP コアを使用しており，パイプライン段数の設定により，動作周波数が大きく変わる．提案する実数値 GA 専用プロセッサは，動作周波数 100MHz 以上での実装可能なパイプライン段数として 5 を設定した．

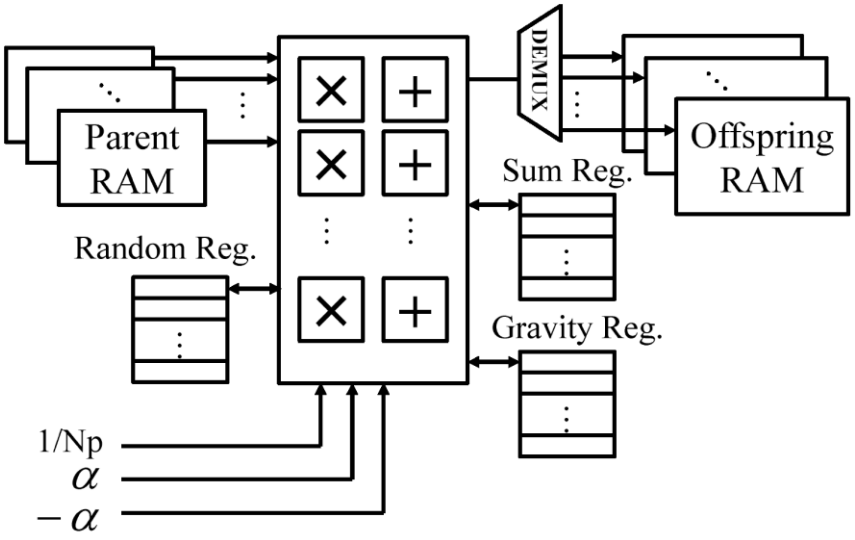


図 3.8. REX 回路のブロックダイアグラム

この回路の特徴は，各手順において必要な数だけの乗算器と加算器を用いて，その接続を動的に変更することで上記の計算を共に N_p 個の演算器だけで実現することである．これにより，全ての手順において必要な個数だけ演算器を用意する場合よりも少ない回路で実装できる．全ての手順において必要な個数だけ演算器を用意する場合には，乗算器が $3N_p$ 個，加算器が $4N_p$ 個必要になる．すなわち，回路を共有することによって乗算器の個数が 1/3，加算器の個数が 1/4 に削減できる．演算器を共有することにより，各手順は逐次的な実行に制限され，計算時間は増加する．しかし，これらの計算は評価と並列に実行される．REX の計算時間が増加しても，評価関数の計算時間より短く，全体の計算時間は評価の時間により決まるため問題はないと考えられる．

N_p =4 の場合の各手順での演算器間の結線を図 3.9, 図 3.10, 図 3.11 に示し，各状態での動作を以下に述べる．

(1) 総和計算時には，図 3.9 (a) に示すように，全ての加算器を並列に使用する．集団メモリからデータが読み出されるごとに，各次元の累積を保存するレジスタに加算して結果を保存する．

(2) 重心計算時には，図 3.9 (b) に示すように，全ての乗算器を並列に使用する．各次元の累積レジスタに親個体数の逆数を並列に乗算することにより，重心を求める．親個体数の逆数は，評価用 CPU の 1 つを使用して設定する．

(3) 偏差計算時には，図 3.10(c) に示すように，全ての加算器を減算器として並列に使用する．N_p 個の親メモリから同時にデータを読み出し，減算器で重心の各次元との偏差を計算して親メモリのデータを上書きする．手順(1) から(3) は子を生成する前処理として，1 世代に 1 回だけ実行される．

(4) 乱数生成時には，図 3.10 (d) に示すように，乗算器と加算器を接続し並列に使用する．[0, 1.0) の乱数 R_1 に式(3.3)の計算を行い，[-α, α]の乱数 R_2 を生成する．

$$R_2 = (\max - \min)R_1 + \min = 2\alpha R_1 - \alpha \qquad (3.3)$$

ここで，max は乱数の上限 であり，min は下限である．並列に N_p 個生成された乱数をレジスタに格納する．

(5) 子生成時には，図 3.11 に示すように，乗算器と加算器を結合して使用する．手順(3)で計算した結果を N_p 個の親メモリから 1 次元ごと同時に読み出す．それらの値と，手順(4)で生成した N_p 個の乱数レジスタの値を同時に乗算して，その後に接続された加算器で総和を計算する．最後の加算器でその次元の重心に加算し

て，子の 1 次元のデータが計算される．計算結果は子メモリに保存される．上記の処理を次元数分の N回行い，1 つの子が生成される．

手順(4)と(5)を $N_{osr} \times N_p$回繰り返し実行することで N_{os}個の子を生成する．

また，初期化として，初期個体を乱数発生回路と REX 回路(4)の状態で生成し，子メモリに格納する．子メモリの内容を集団メモリに順次コピーする．全ての個体のコピーが完了すると初期化終了となり，実数値 GA の処理が開始する．

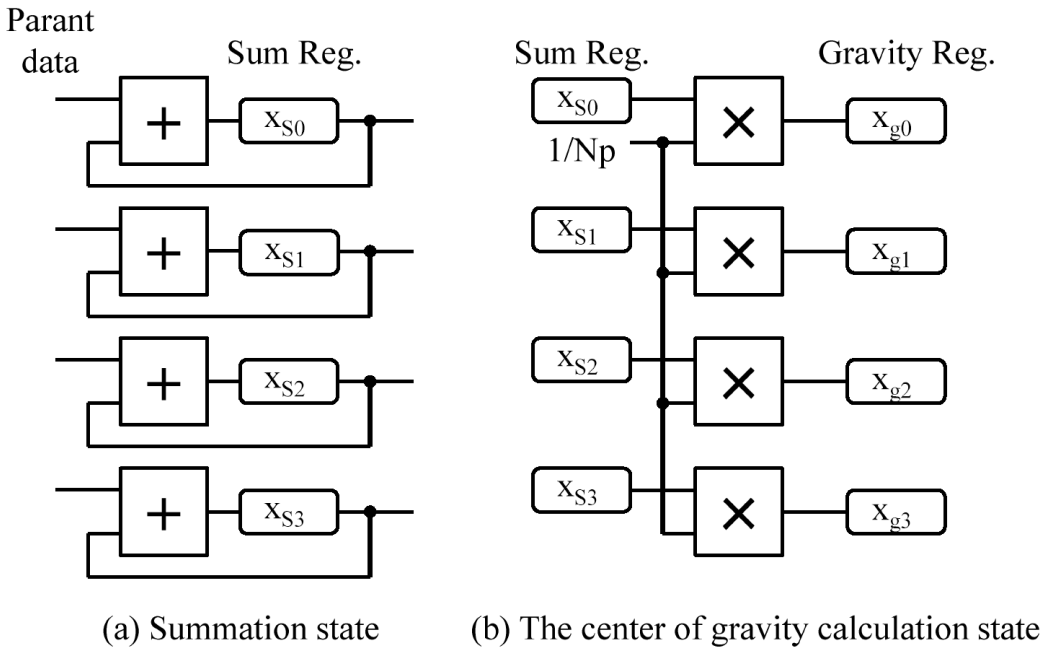
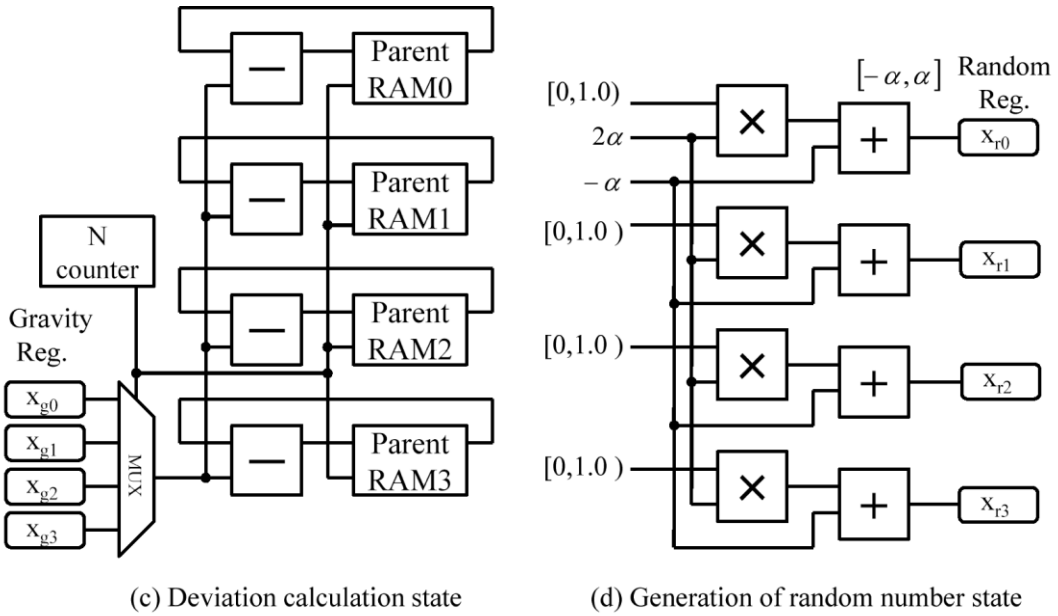
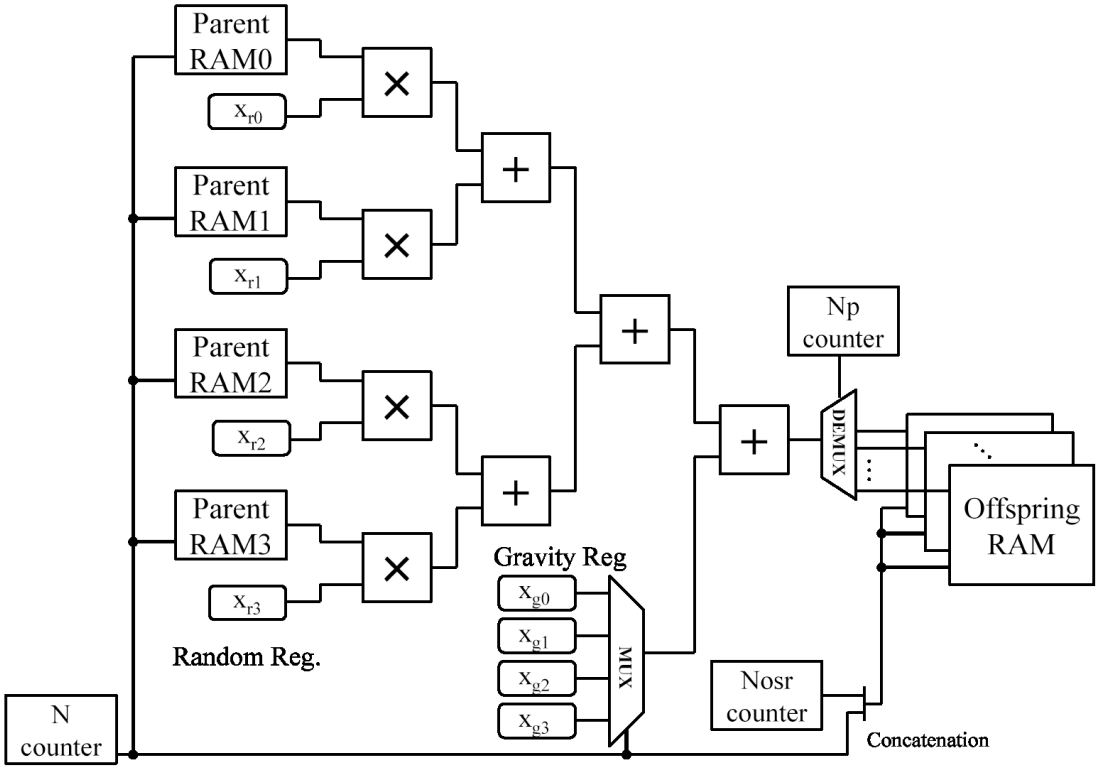


図 3.9. REX 回路の回路構成（a, b）



(c) Deviation calculation state (d) Generation of random number state

図 3.10. REX 回路の回路構成（c, d）



(e) Generation of offspring state

図 3.11. REX 回路の回路構成（e）

3.9 ソフトマクロ汎用 CPU (MicroBlaze)を用いた評価回路の実装

REX によって生成された子個体の適合度を評価関数から計算する．評価関数は扱う問題によって異なるため，評価関数を回路で実現しようとすると問題ごとに評価回路を新たに設計しなければならない．そこでソフトマクロ汎用 CPU で適合度の計算を行うことにより，評価関数のプログラムの書き換えだけで対応させる．CPU には Xilinx 社のソフトマクロ汎用 CPU の MicroBlaze [98]を N_p 個使用する．MicroBlaze は RISC アーキテクチャ，5 段パイプラインである．図 3.12 に MicroBlaze のブロックダイアグラムを示す．

3.9.1 MicroBlaze 周辺回路の構成

MicroBlaze 周辺回路として，AXI (Advanced eXtensible Interface)バスを介して，評価開始・終了のフラグレジスタ，子メモリ，子に対応する適合度レジスタが接続されている．1 つの CPU だけに，実数値 GA のパラメータ（乱数の範囲，目標適合度など）を設定するレジスタと，最終的な最適解の表示のためのシリアル通信モジュールが接続されている．MicroBlaze および周辺回路を図 3.13 に示す．

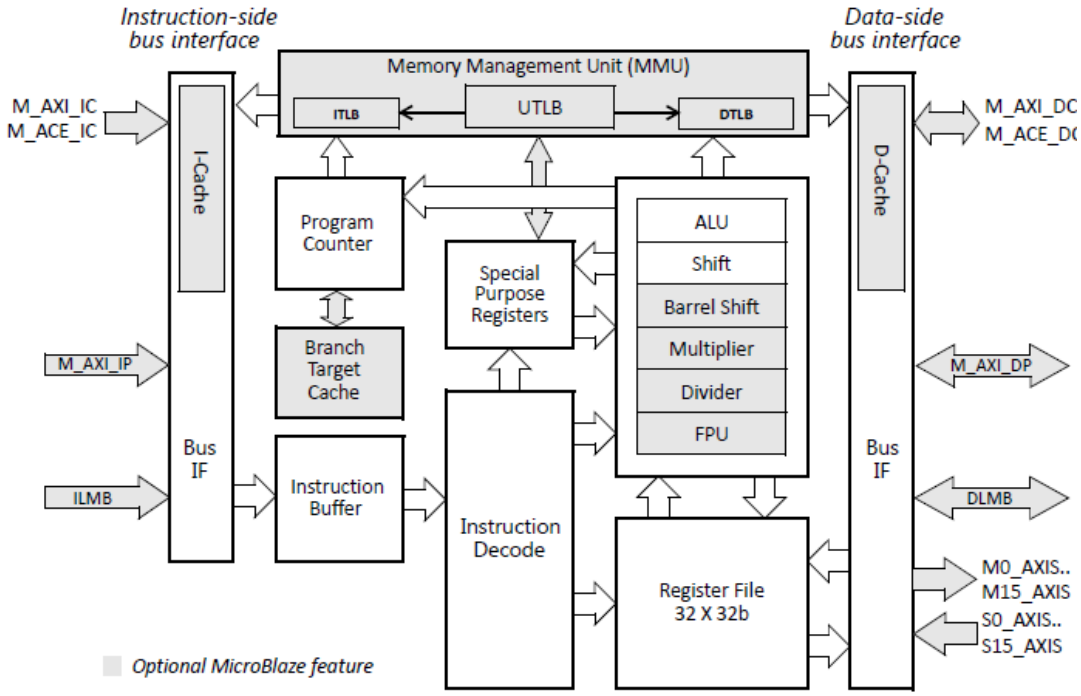


図 3.12. MicroBlaze のブロックダイアグラム [98]

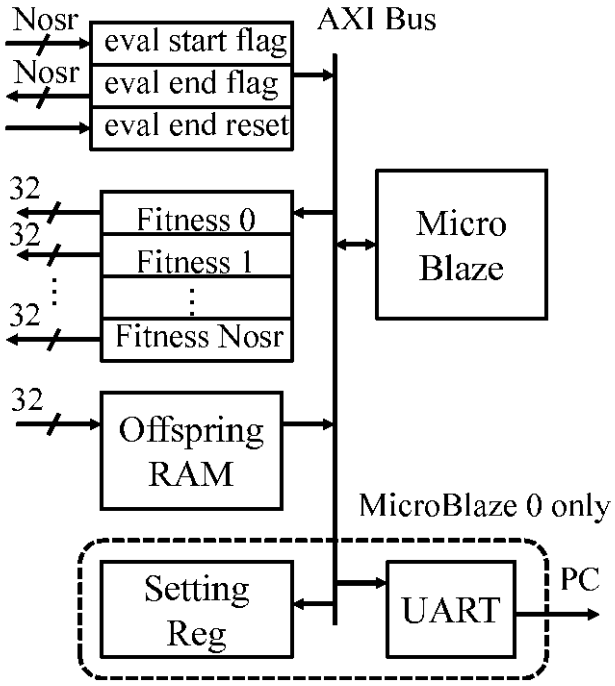


図 3.13. 周辺回路の構成

3.9.2 MicroBlaze による評価関数の計算フロー

MicroBlaze のプログラムのフローチャートを図 3.14 に示す．この動作を以下に述べる．REX によって 1 つ目の個体が生成されると，評価開始フラグが立ち，評価計算を開始する．計算が終了次第，適合度をレジスタに保存して評価完了フラグを立てる．REX で子が生成されるたびに評価開始フラグが立ち，子の生成と並行して順次評価される．この回路を N_p 個用意して，並列に N_p 個の個体を評価することで，GA においてボトルネックとなる評価の実行時間を軽減している．また，REX 回路による子の生成と並行して評価を行うことにより，全体の実行時間の短縮を図っている．

また，1 つの MicroBlaze から実数値 GA のパラメータが設定可能である．設定できるパラメータは，終了条件となる目標適合度，処理を打ち切る評価回数，親個体数，乱数の範囲などである．実数値 GA の処理の終了判定は，現在の最適解の適合度が目標適合度以下になった場合，もしくは処理を打ち切る評価回数に達した場合とする．

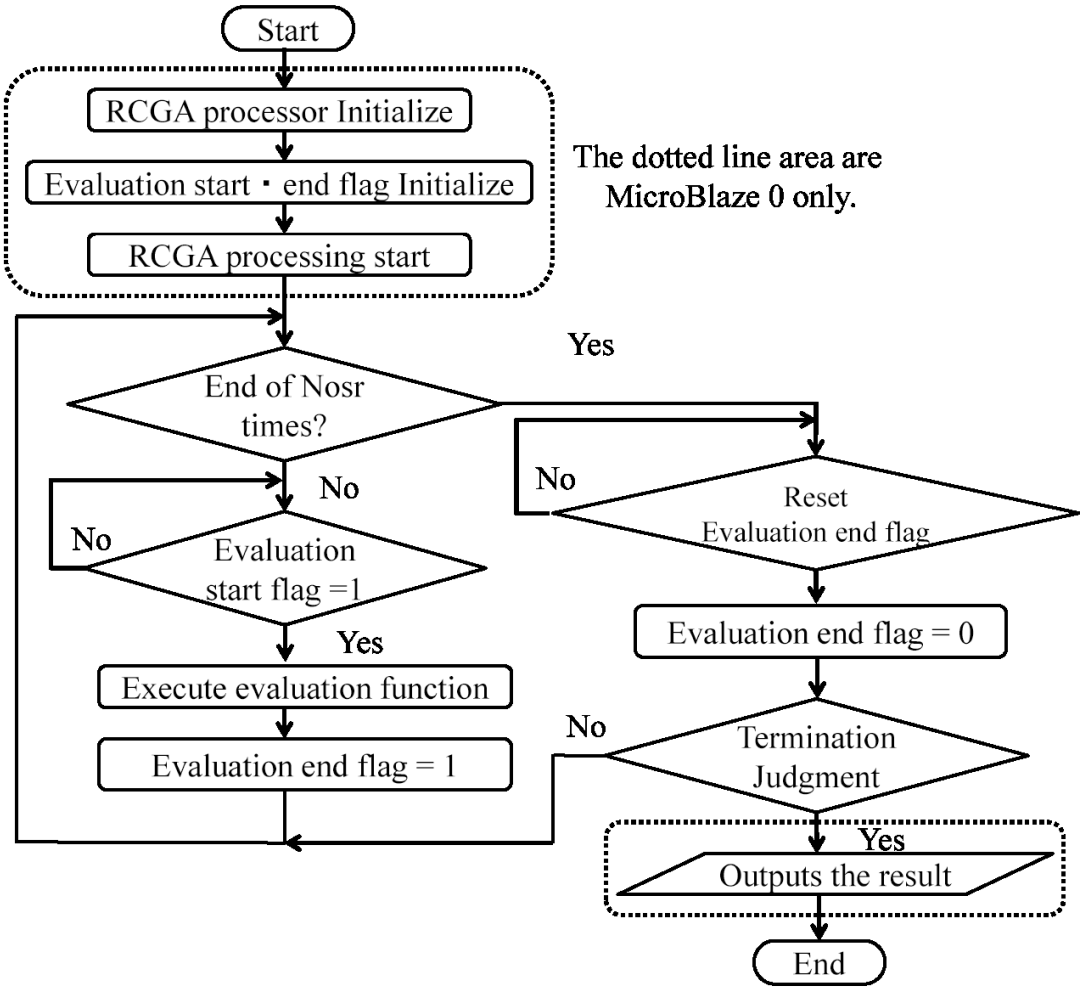


図 3.14. MicroBlaze による評価関数の計算フロー

3.10 実装実験環境

FPGA ボードとして，Xilinx 社の Virtex-7(XC7VX485T-2FFG1761C)が搭載されている VC707 評価ボードを使用した．使用した FPGA ボードを図 3.15 に，搭載されている FPGA の回路規模を表 3.2 に示す．

表 3.2 Virtex-7 (XC7VX485T) の回路規模

FPGA resource	Virtex-7 (XC7VX485T)
Slice Register	607,200
Slice LUT	303,600
Block RAM	1,030
DSP	2,800

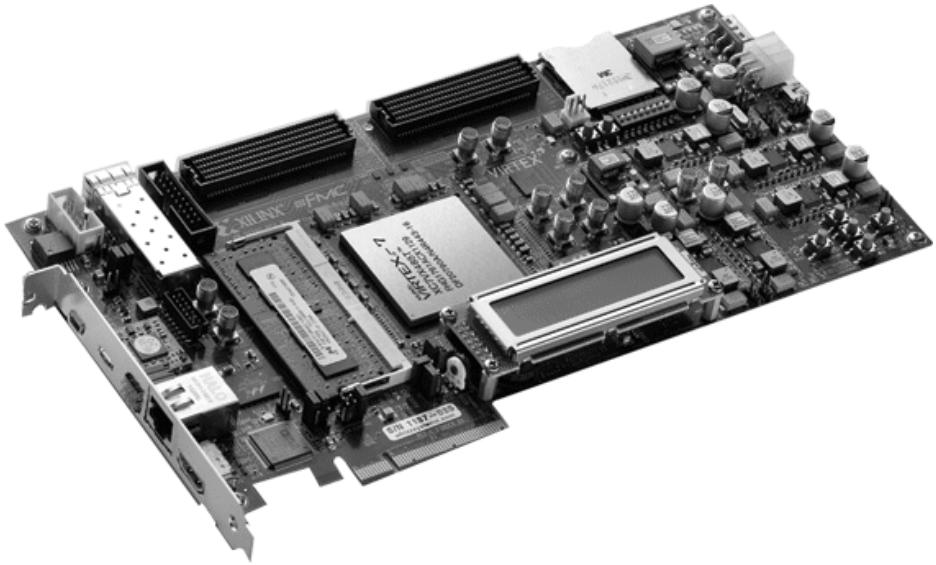


図 3.15. VC707 FPGA 評価ボード

3.11 リソースシェアリングを適用した多親交叉 REX 回路の有効性

提案する実数値GA専用プロセッサをハードウェア記述言語VHDLにより設計した．論理合成と回路シミュレーションにはXilinx社の設計ツールVivado(2016.2)を使用した．ターゲットデバイスをVirtex-7 (XC7VX485T)として論理合成を行った．

N_p=16, 32, 64の3つの場合で提案する実数値GA専用プロセッサを実装したときの全体の回路規模と，その実数値GA処理部とMicroBlaze及び周辺回路のそれぞれの回路規模を表 3.3, 表 3.4, 表 3.5に示す．

提案する実数値GA専用プロセッサは，演算回路やCPUはN_p個ずつ実装される．そのため，MicroBlazeとその周辺回路が，全体の回路の6割以上を占めている．更なる並列化を行う場合には，MicroBlaze ではなく最低限の機能に絞った回路規模の小さい専用のCPUを実装することが望ましい．

一方，実数値GA処理部は1割から3割程度の回路規模と小さい．これは浮動小数点演算器を共有しているためと考えられる．浮動小数点演算器を共有は，DSP ブロックの使用率に影響する．使用している浮動小数点演算器のIPコアは，FPGAに組み込まれているDSPブロックを使用することができる．使用しているFPGAのDSPブロックは，可変長ビットの乗算器と加算器が組み合わせたシンプルな構成であり，使用したVirtex-7には2,800個組み込まれている．使用した浮動小数点演算器のDSP使用数は，浮動小数点乗算器が1個につき3個，浮動小数点加減算器が1個につき2個である．回路の共有をしない場合には，乗算器が 3 N_p個，加減算器が 4 N_p個必要になる．図3.16および表 3.6にリソースシェアリング適用後のDSPの使用数と示す．N_p =64の場合には，DSPの使用数は乗算器で576個，加減算器で512個となり，総数では1088個になると考えられる．実数値GA処理部のDSP使用数は320個であるため，回路を共有することで7割程度のDSPが削減できていると考えられる．

表 3.3 実数値 GA 専用プロセッサの回路規模（N_p=16）

FPGA resource	Virtex-7 (XC7VX485T)	N_p=16		
		All circuit	RCGA processing	MicroBlaze and peripherals
Slice Register	607,200	79,258 (13%)	18542 (3%)	60,709 (10%)
Slice LUT	303,600	70,252 (23%)	20,900 (7%)	49,385 (16%)
Block RAM	1,030	118 (11%)	8 (1%)	110 (11%)
DSP	2,800	160 (6%)	80 (3%)	80 (3%)

表 3.4 実数値 GA 専用プロセッサの回路規模（N_p=32）

FPGA resource	Virtex-7 (XC7VX485T)	N_p=32		
		All circuit	RCGA processing	MicroBlaze and peripherals
Slice Register	607,200	156,799 (26%)	36,875 (6%)	119,912 (20%)
Slice LUT	303,600	138,145 (46%)	40,466 (13%)	97,729 (32%)
Block RAM	1,030	254 (25%)	48 (5%)	206 (20%)
DSP	2,800	320 (11%)	160 (6%)	160 (6%)

表 3.5 実数値 GA 専用プロセッサの回路規模（N_p=64）

FPGA resource	Virtex-7 (XC7VX485T)	N_p=64		
		All circuit	RCGA processing	MicroBlaze and peripherals
Slice Register	607,200	311,515 (51%)	73,244 (12%)	238,248 (39%)
Slice LUT	303,600	278,915 (92%)	84,667 (28%)	194,585 (64%)
Block RAM	1,030	558 (54%)	160 (16%)	398 (39%)
DSP	2,800	640 (23%)	320 (11%)	320 (11%)

表 3.6 リソースシェアリング適用後の DSP の使用数

N_p	16		32		64	
	必要数	提案手法	必要数	提案手法	必要数	提案手法
乗算の DSP 使用数	144	48	288	96	576	192
加算の DSP 使用数	128	32	256	64	512	128
合計	272	80	544	160	1088	320

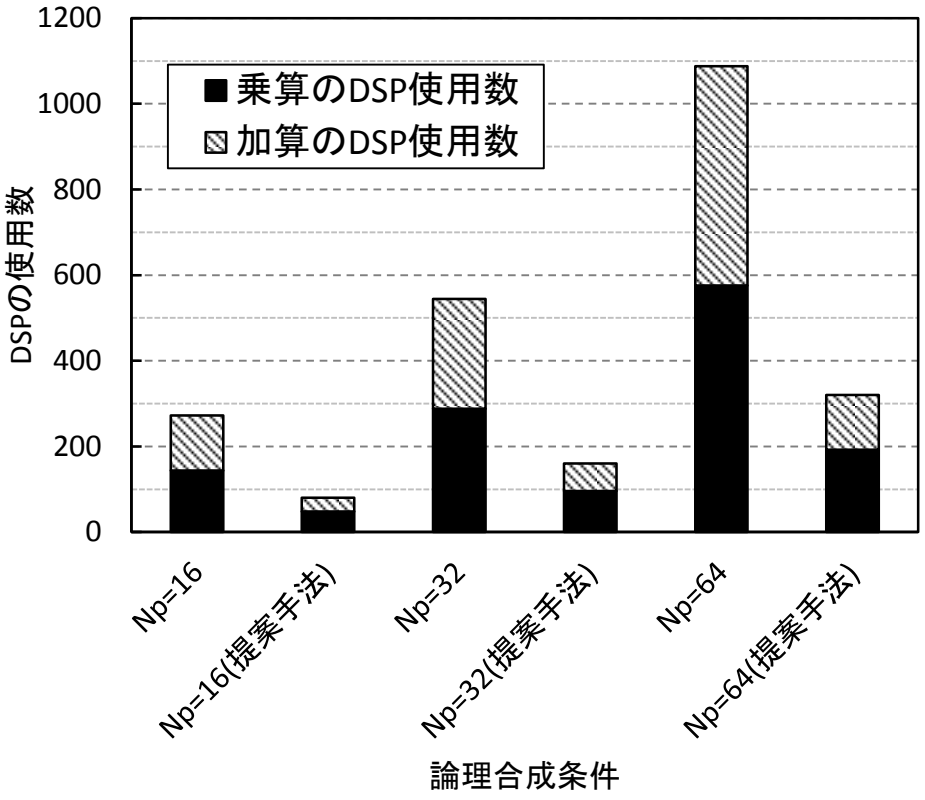


図 3.16. リソースシェアリング適用後の DSP 使用数

3.12 ベンチマーク問題を用いた実行時間の評価

3.12.1 適用したベンチマーク問題の概要

提案する実数値 GA 専用プロセッサの性能を確認するため，ベンチマークに使用される評価関数に適用した．使用した評価関数は，Sphere 関数(式(3.4))，Ellipsoid 関数(式(3.5))，および Rosenbrock 関数(式(3.6), star 型) である．これらのベンチマーク関数は性能評価によく用いられる [99]．Sphere 関数と Ellipsoid 関数は，全ての変数が 0 のときが最適解である(関数値 0)．また，Rosenbrock 関数は全ての変数が 1 のときが最適解である(関数値 0)．Ellipsoid 関数は目的関数に対する感度が変数によって異なる悪スケール性を弱く持つ．また，Rosenbrock 関数は変数 x_1 と他の変数間に強い変数間依存性がある．

$$f_1(x) = \sum_{i=1}^{N} x_i^2 \qquad (-5.12 \le x_i \le 5.12) \tag{3.4}$$

$$f_2(x) = \sum_{i=1}^{N} (1000^{\frac{i-1}{N-1}} x_i)^2 \qquad (-5.12 \le x_i \le 5.12) \tag{3.5}$$

$$f_3(x) = \sum_{i=2}^{N-1} (100(x_1 - x_i^2)^2 + (1 - x_i)^2) \qquad (-2.048 \le x_i \le 2.048) \tag{3.6}$$

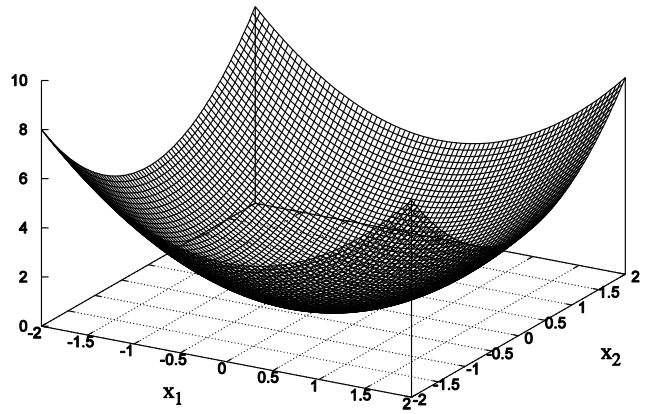


図 3.17. Sphere 関数のプロット（*N*=2）

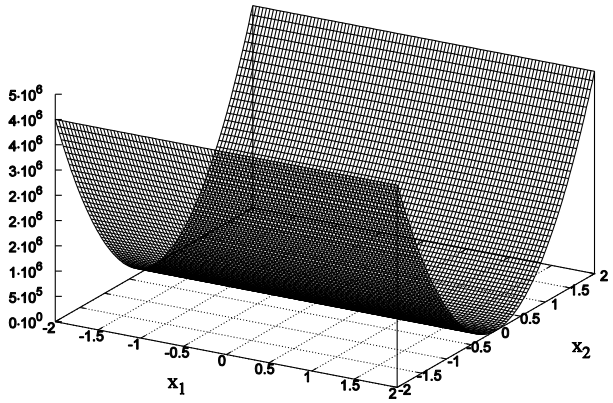


図 3.18. Ellipsoid 関数のプロット（*N*=2）

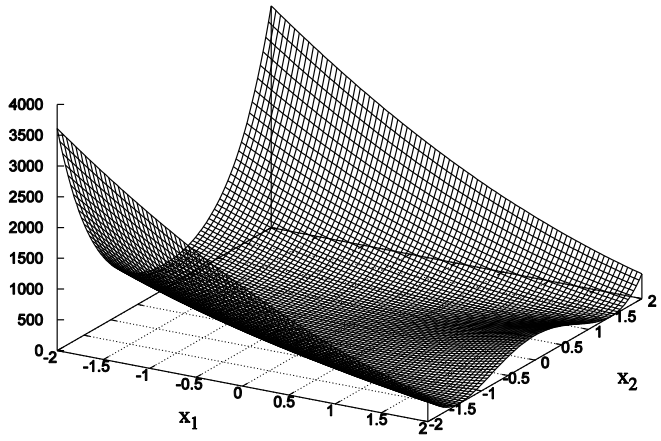


図 3.19. Rosenbrock 関数のプロット（*N*=2）

上記 3 つの関数を用いて, FPGA ボードへの実装実験を行った. 実験の様子を図 3.20 に示す．N_p =16 で実装した場合には次元数 N=14，N_p =32 の場合には N=30，N_p =64 の場合には N=62 として実行した．比較として，実数値 GA の C 言語プログラムを作成し，PC（Intel(R) Xeon(R) CPU E5-1630, 3.7GHz 搭載）で実行した．コンパイラは GCC-4.8.2 を使用した．プログラム中の実数値は全て単精度浮動小数点数（float 型）として扱っている．プログラムの並列化はしておらず，シングルコアとして動作する．いずれも関数値が 10^{-7} 以下に到達したら最適解とみなしている．各関数で設定した実数値 GA パラメータを表 3.7 に示す．提案する実数値 GA 専用プロセッサでは，N_{os} は N_p の整数倍で設定するため，N の整数倍と近い値となるよう設定した.

表 3.7 各ベンチマーク関数で設定した実数値 GA パラメータ

Parameter	Sphere	Ellipsoid	Rosenbrock
N	14, 30, 62		
N_p	16, 32, 64		
N_{os}	$7N$(≈ $6N_p$)	$8N$(≈ $7N_p$)	$10N$(≈ $9N_p$)
P	$7N$	$8N$	$16N$

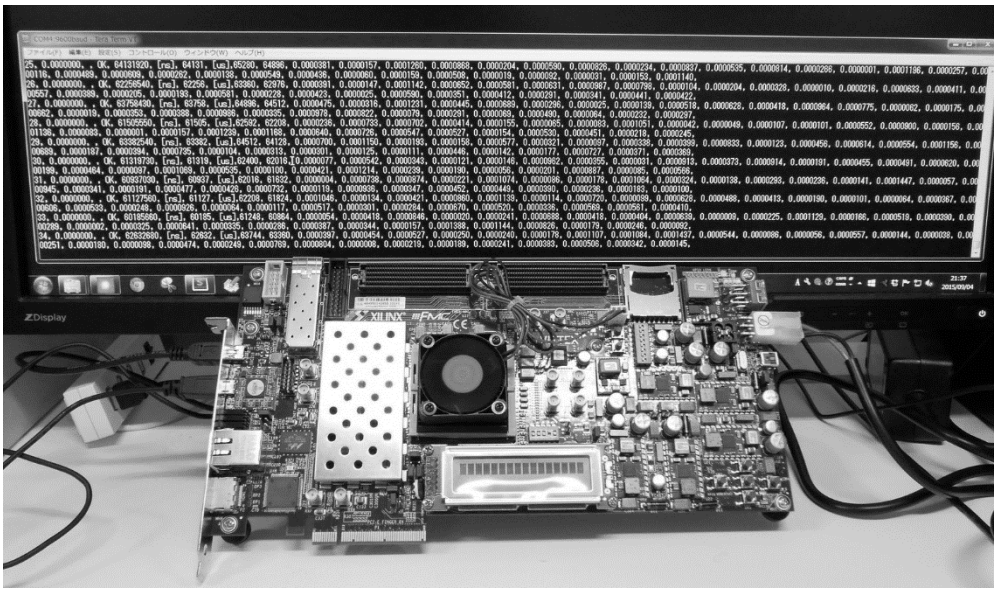


図 3.20. 実験の様子

提案する実数値 GA 専用プロセッサの FPGA ボード上での実行結果と PC の実行結果を図 3.21, 図 3.22, 図 3.23 に示す．図は各関数(N=14)での評価回数に対する最適解の適合度の遷移を示しており，各環境で 30 回実行したときの平均値をプロットしている．Sphere 関数において，最適解に到達する評価回数は PC と提案する実数値 GA 専用プロセッサ共に 19,000 回程度であり，同等な評価回数で最適解が求まっている．Ellipsoid 関数，Rosenbrock 関数においても同様に，PC と同程度の評価回数で解が求まることを確認した．

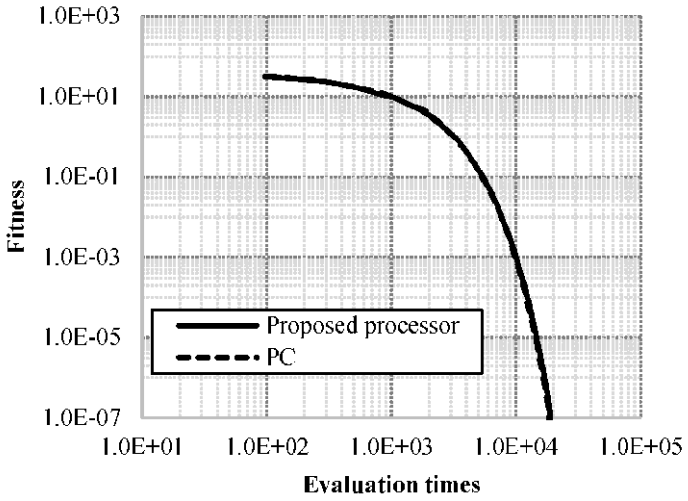


図 3.21. Sphere 関数の結果

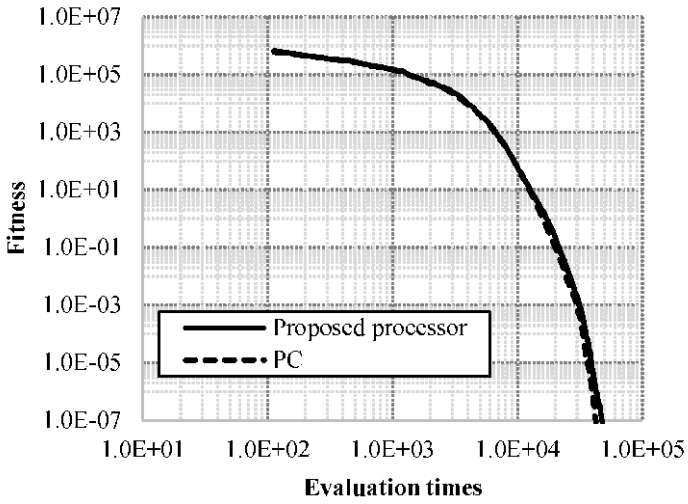


図 3.22. Ellipsoid 関数の結果

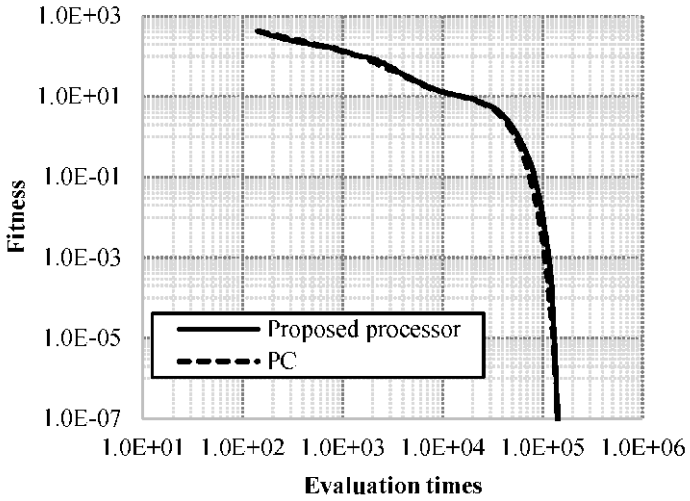


図 3.23. Rosenbrock 関数の結果

3.12.2 PC との実行時間の比較

提案する実数値 GA 専用プロセッサの実行時間と，PC で実行した C 言語プログラムの実行時間を表 3.8, 図 3.24, 図 3.25, 図 3.26 に示す．提案する実数値 GA 専用プロセッサは，N_p =16,32 のときは周波数 140MHz で動作させた結果，N_p =64 のときは 100MHz で動作させたときの結果である．いずれの結果も各環境で 30 回実行した平均値をプロットしている．N=14, 30 の場合には，提案する実数値 GA 専用プロセッサは PC とほぼ同等の実行時間であるが，N=62 の場合には，全ての評価関数において 2 倍以上の実行時間の改善が確認できた．最も実行時間が改善した Rosenbrock 関数の N=62 の場合には，PC の実行時間は 4632ms に対して，提案する実数値 GA 専用プロセッサの実行時間は 1607ms であり 2.9 倍の改善を確認した．

これらの実行時間の改善は，N_p が 2 倍になる毎に評価を行う CPU の数が 2 倍になり，並列に評価する子の個体数が 2 倍に増加するためであると考えられる．そのため，並列数をさらに増やすことによって，更なる実行時間の改善が期待できる．しかしながら，現在使用している FPGA では，N_p=64 のとき最大 92%の回路資源を消費しているため，実装できる限界である．そのため，更なる並列化のためには，回路資源を 64%消費している CPU の機能を絞り回路規模を縮小させることが有効であると考えられる．

表 3.8 PC と FPGA の実行時間比較

Function	N	PC (Intel Xeon CPU E5-1630, 3.7GHz),GCC4.8.2		FPGA (Virtex-7)			Speed up
		Execution Time [ms]	Evaluation Time	Execution Time [ms]	Evaluation Time	Operation frequency [MHz]	
Sphere	14	9.4	18761	10.4	19267	140	0.90
	30	82.2	64828	51.0	63360	140	1.61
	62	816.9	202274	356.0	195635	100	2.29
Ellipsoid	14	18.7	35710	19.4	37285	140	0.96
	30	152.4	119705	96.3	124798	140	1.58
	62	1574.0	370645	699	397033	100	2.25
Rosenbrock	14	60.3	115146	65.6	128376	140	0.92
	30	490.9	338851	272.5	358694	140	1.80
	62	4632.2	974414	1606.5	974842	100	2.88

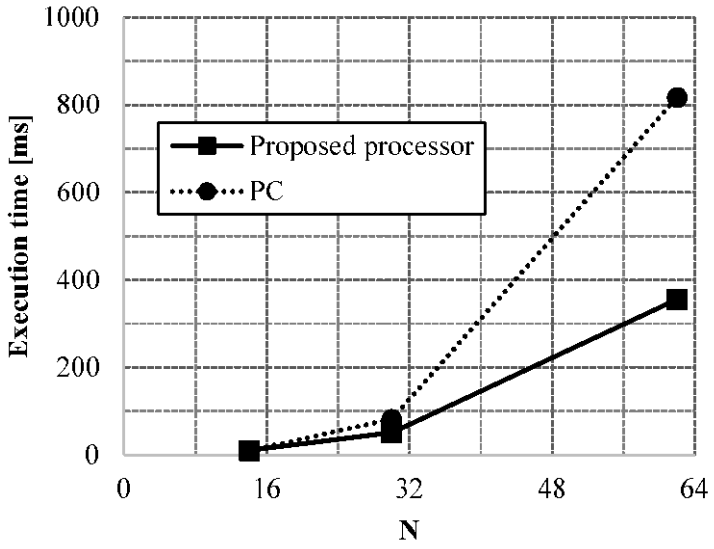


図 3.24. Sphere 関数の実行時間

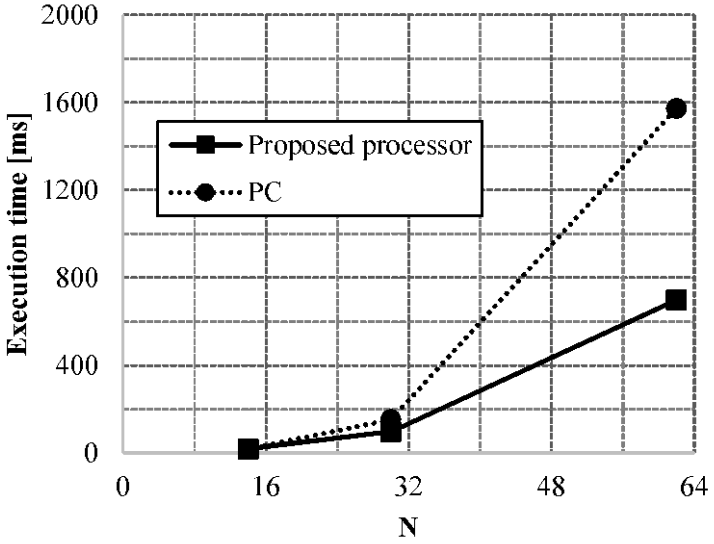


図 3.25. Ellipsoid 関数の実行時間

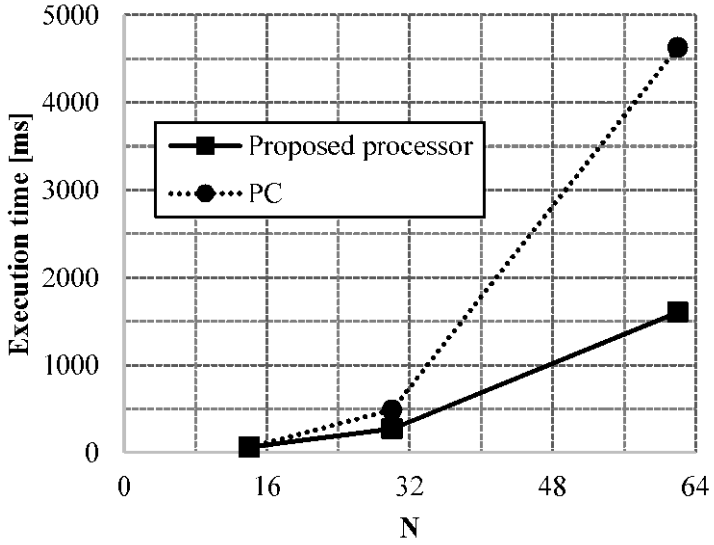


図 3.26. Rosenbrock 関数の実行時間

3.13 実行時間についての考察

設計した実数値 GA 専用プロセッサは，N_p 個の CPU を組み込み，並列に N_p 個の評価計算を行うことで高速化を試みている．その構造的に，N_p が 2 倍になれば実行時間の改善は約 2 倍になると考えられる．Rosenbrock 関数の場合，N_p=16 のとき動作周波数 140MHz で PC と比べて実行時間の改善は約 0.9 倍であった．N_p =32 のとき同じく 140MHz で動作させることができ，実行時間の改善は約 1.8 倍であった．この結果は，N_p が 2 倍になったことにより実行時間の改善は倍になり，狙い通りの結果である．し

かしながら，N_p=64 のとき実装可能な動作周波数は 100MHz であった．これは，使用している FPGA の 92%の回路規模を使用しているため，配線遅延の影響が大きくなり動作周波数が低下したと考えられる．N_p=64 においても 140MHz で動作した場合，実行時間の換算値は 1148ms となり，PC と比べて約 4 倍の実行時間の改善が見込める．この場合，1.8 倍の倍の 3.6 倍よりも実行時間の改善は大きくなる．これは，未知パラメータ増加に伴う PC 側の性能低下によるものと考えられる．

しかし，Sphere 関数と Ellipsoid 関数では，N_p =16 のとき実行時間の改善は約 0.9 倍であり，N_p =32 のとき約 1.6 倍であった．Rosenbrock 関数と同様に N_p=64 においても 140MHz で動作した場合，換算すると約 3.2 倍の実行時間の改善が見込める．したがって，N_p =32 から N_p =64 の変化に対して，実行時間は倍の改善が期待できるが，N_p =16 から N_p =32 の変化では，実行時間の倍の改善が確認できなかった．これは，評価計算の時間が短く，交叉 REX の時間が影響しているものと考えられる．設計したプロセッサは，交叉 REX による子の生成とソフトマクロ汎用 CPU での評価を並行して行っており，子が生成され次第，評価が開始する．したがって，評価関数の計算時間が，交叉 REX による子の生成時間より短ければ，子の生成時間時間が全体の実行時間に影響する．これら 2 つの関数の計算量は Rosenbrock 関数よりも少ないため，実行時間が短くなったものだと考えられる．

したがって，実行時間の改善と換算値より，N_p の増加により期待通りの実行時間の改善が得られることを確認した．評価関数の計算時間が短い場合，子の生成時間が影響することを確認したが，実際の応用では，より複雑な評価関数であることが予想され，特に問題はないと考えられる．

3.14 先行研究との比較

近年提案されている進化アルゴリズム専用プロセッサと設計した実数値 GA 専用プロセッサとの比較を表 3.9 に示す．表中に記載しているプロセッサは，GA [56, 57]，並列化した GA [60, 65]，PSO [47, 50]などのプロセッサのデータ形式と評価している次元数および比較対象との高速化率を示している．

評価関数をソフトマクロ汎用 CPU で評価する構成のプロセッサでは，ソフトマクロ汎用 CPU 上で実行させたソフトウェアの実行結果と比較して，数倍から百倍程度の高速化が報告されている．しかしながら，PC との比較を行っている事例はあまり見受けられない．

文献 [60]では，評価回路が専用回路として組み込まれており，CPU と比較して 21 から 34 倍の高速化，GPU と比較して 6.8 倍の高速化が報告されている．マルチコア CPU や GPU よりも高速化のためには専用回路を設計することが有効と考えられるが，評価

関数回路が複雑な場合には，回路設計は複雑になり，設計・検証に時間がかかる．それ故か，多くの文献では評価に用いている未知パラメータ数は N=2 など少ないパラメータ数で評価を行っている事例が多く見受けられる．

また進化アルゴリズムは，マルチコア CPU や GPU を用いた実装も高速化に有効である．文献 [100]では，実数値 GA に GPU を適用した場合，50 次元や 100 次元の Ellipsoid 関数，Rosenbrock 関数において 7 から 46 倍の高速化が報告されている．文献 [100] の実数値 GA は，本研究で採用した JGG と REX とは異なる方式（トーナメント選択，Simulated Binary Crossover [101], 突然変異あり）であるが，同様に速度向上が期待できると考えられる．

しかしながら，組込み機器では使用可能な電力は限られており，動作周波数が高く消費電力が大きいマルチコア CPU や GPU などを使用することが難しい．例えば，自動車などの電力が限られた空間の中で複雑な画像処理を行う場合などは, 動作周波数当たりの計算性能が高い FPGA は有効な選択肢となる [102].

本研究では，未知パラメータ数 N=62 のとき，PC（シングルコアとして動作）と比較して最大で約 2.9 倍の実行時間の改善であった．マルチコア CPU や GPGPU を用いた実装においては，数倍から数十倍の高速化が期待できるため，組込み機器でもそれらと同等以上の高速化を達成するためには, 以下の点において更なる改善が必要と考えられる．

まず，現在使用しているソフトマクロ CPU は汎用であるが故に，回路規模が大きく実行時間においてもボトルネックとなっている点である．これに対して，汎用 CPU ではなく, 評価関数に良く使用される命令を精査して機能を絞り, データ形式も整数命令, 固定小数点命令のみを扱うなど評価関数計算に特化した CPU を作成し，それを使用するなど改良が必要と考える．

また，実数値 GA 専用プロセッサで使用している演算器は，単精度浮動小数点形式である．これらを整数や固定小数点形式にすることで，全体的な動作周波数向上と実行時間の短縮が見込める．一般的に単精度浮動小数点形式は，同じビット数でも，整数や固定小数点形式より広い範囲の数が扱えるが，計算処理が整数・固定少数より複雑になるため，それらより計算には時間がかかる．応用によっては，広いデータ範囲を扱う必要がないこともあると考えられるため，対象分野を絞り，このような変更も有効と考えられる．

これらの改善が，並列数の向上と実行時間の改善に有効であると考えられ，今後の課題としたいと考えている．

表 3.9 先行研究との比較

発表年	文献番号	FPGA	アルゴリズム	データ形式	評価関数の実装方法	評価に用いている未知パラメータ数 (N)	動作周波数[MHz]	比較対象	高速化率	備考
2010	50	Altera Cyclone II EP2C70	PSO	整数，固定少数 (8bit)	ソフトマクロ 汎用CPU (NiosII)	2, 32	50	NiosII	20倍	N=2の関数の場合，NiosIIと比較
								PC (Windows XP, Matlab)	PC測定不可. (FPGAで163から1659秒)	N=32の場合PCと比較. 専用回路にすると，ソフトマクロ汎用CPUの時に比べて12倍改善
2010	56	Xilinx Virtex2Pro (XC2VP30)	GA	整数，固定少数 (16bitごと拡張可)	専用回路	2 (拡張化)	50	FPGA内のハードマクロCPU (Power PC)	約5倍	
2013	57	Altera Stratix II (EP2S60)	GA	整数，固定少数 (16bit)	ソフトマクロ 汎用CPU (NiosII)	2 (最大16)	50	NiosII	37から102倍	GAのライブラリGalibを実行. (37倍はNiosIIのオプションあり，102倍はデフォルト設定)
2014	47	Xilinx Virtex-5 (XC5VFX70T)	PSO	浮動小数点数	ソフトマクロ 汎用CPU (MicroBlaze)	2	50	MicroBlaze	20倍	
2014	60	Xilinx Virtex-6 (XC6VSX475T)	並列GA	整数，固定少数，浮動小数点数	専用回路	2	160	PC (Dual Intel Xeon X5650, 2.67GHz)	21から34倍	
								GPU (nVidia Tesla C1060)	6.8倍	
2016	65	4 FPGA Virtex-6 SX475T	並列GA	整数，固定少数	専用回路	4	150	PC (Dual Intel Xeon X5650, 2.67GHz, 8thread)	27から34倍	
2017	提案手法	Xilinx Virtex-7 (XC7VX485T)	実数値GA	浮動小数点数	ソフトマクロ 汎用CPU (MicroBlaze)	14, 30, 62	140 (N=14, 30), 100 (N=62)	PC (Intel Xeon E5-1360 CPU, 3.7GHz)	1.9から2.8倍	

3.15 結言

本章では，実数値 GA 専用プロセッサの一方式を提案した．提案する実数値 GA 専用プロセッサは，世代交代モデルに JGG，交叉には REX を採用した．演算器などの回路資源を有効に共有することで回路規模少なく実装できる．これにより，必要な数だけ演算器を用意した場合に比べ，1/3 から 1/4 の演算器数で実装可能である．また，扱う問題によって異なる評価関数をソフトマクロ汎用 CPU を用いて計算するため，汎用性も備えている．

提案する実数値 GA 専用プロセッサは，N=62 の場合でも 1 チップの FPGA に実装可能である．さらに，関数評価を行う CPU とその周辺回路をさらに小型にし，並列度を上げることにより，更なる高速化が期待できる．

ベンチマーク関数を用いて実装実験を行った結果，悪スケール性や変数間依存性を持つ関数においても，PC の場合と同等の評価回数で最適解が得られることを確認した．また，回路の並列数を示す N_p を増やすことにより，PC と比べて実行時間が改善することを確認し，実装できる範囲で回路を並列化することにより，多くの次元を扱う問題において提案する実数値 GA 専用プロセッサが有効であることを示した．

第4章　実数値 GA 専用プロセッサを用いたディジタルフィルタの最適設計

4.1　序

本章では，実数値 GA 専用プロセッサの一応用として，ディジタルフィルタ設計に適用した事例［103, 104]を示す．FIR フィルタの特性を決めるフィルタ係数を実数値 GA で求めることにより，バンドストップフィルタを設計する．さらにディジタル補聴器のフィッティングを想定したシミュレーションを行った結果を示す．

4.2　進化アルゴリズムを用いたディジタルフィルタの設計

進化型ハードウェアの応用としてフィルタの設計がある. フィルタは音声や画像など幅広く応用される．音声に用いられるディジタルフィルタでは，FIR (Finite Impulse Response)フィルタ，IIR (Infinite Impulse Response)フィルタなどがある．これらのフィルタは，フィルタ係数によって特性が決まる．このフィルタ係数を進化アルゴリズムによって求めることによって, 所望の周波数特性を持つフィルタを設計することができる．単純 GA を改良した方法や DE と PSO などを組み合わせた手法を用いて，バンドストップフィルタを設計した事例が報告されている [103, 104, 105, 106, 107]. そこで, 実数値 GA 専用プロセッサの一応用例として，これに適用した．図 4.1 に実数値 GA 専用プロセッサを用いた FIR フィルタ設計の概念図を示す.

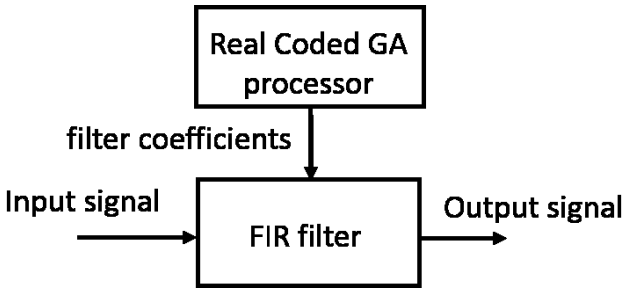


図 4.1. 実数値 GA 専用プロセッサを用いた FIR フィルタ設計の概念図

4.3　FIR フィルタの概要

FIR フィルタは有限長のインパルス応答をもつディジタルフィルタである. その特徴として, 安定で歪みのない位相特性を持つフィルタが比較的容易に実現できる点などがある．FIR フィルタのブロックダイアグラムを図 4.2 に示す．この回路は乗算器，加算器および遅延器から構成される．入力信号をサンプリング周波数で N 個の遅延器で遅延させ，各遅延器の出力信号とフィルタ係数（$h(n)$）との乗算結果の総和が出力信号となる．これを伝達関数は式(4.1)のように表わされる．FIR フィルタは，適切なフィルタ係数 $h(n)$を設定することにより，様々なフィルタを構成できる.

$$H(z) = \sum_{n=0}^{N} h(n) z^{-n} \qquad (4.1)$$

さらに，Nが偶数であり，$h(n)$が偶対称のときは，線形位相 FIR フィルタと呼ばれ，求める係数は半分の $N/2$ 個となる．線形位相 FIR フィルタのブロックダイアグラムを図 4.3 に示す.

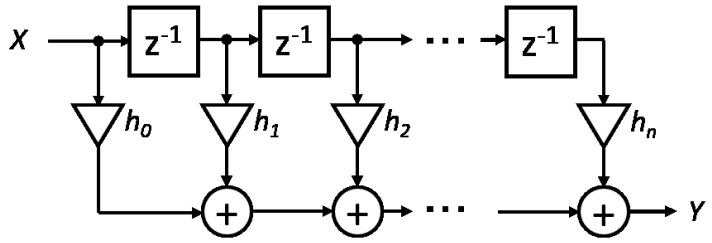


図 4.2. FIR フィルタのブロックダイアグラム

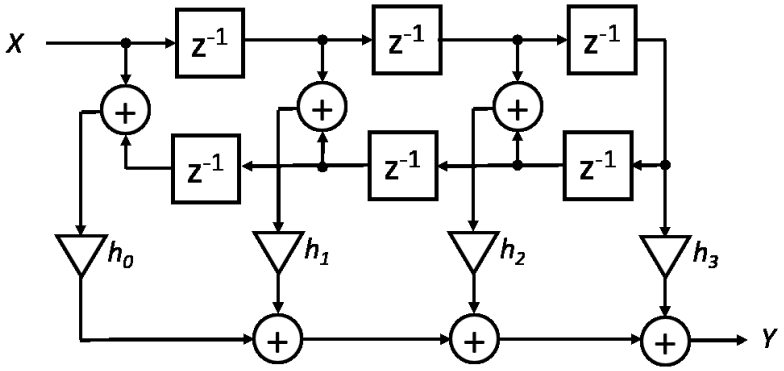


図 4.3. 線形位相 FIR フィルタのブロックダイアグラム

4.4　バンドストップフィルタの設計

バンドストップフィルタとは，ある周波数成分を遮断し，その他の周波数成分は通過させるフィルタである．バンドストップフィルタの周波数特性を図 4.4 に示す．その伝達関数を式(4.2)に示す.

$$H(\omega)=\begin{cases}0 & for \quad \omega_{ls}\le\omega\le\omega_{us}\\ 1 & otherwise\end{cases} \tag{4.2}$$

ここで，ω は正規化周波数，ω_{ls}は遮断周波数の下限，ω_{us}は遮断周波数の上限である．

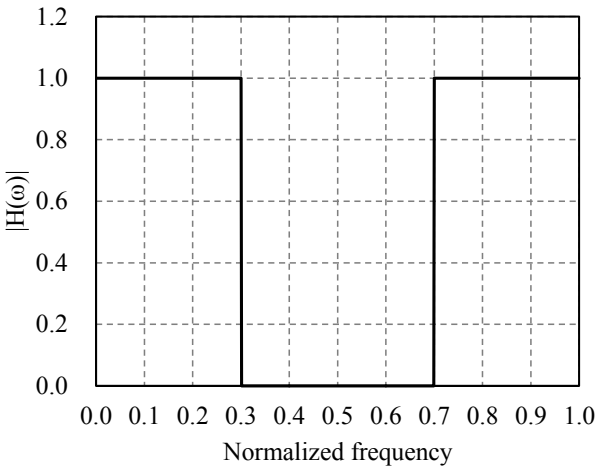


図 4.4. 理想的なバンドストップフィルタの周波数特性

実数値 GA で使用する評価関数

実数値 GA でこの問題を解くためには，適切な評価関数を設定する必要がある．

FIR フィルタの周波数応答は式(4.3)のように計算できる．

$$H(\omega_k) = \sum_{n=0}^{N} h(n) e^{-j\omega_k n} \tag{4.3}$$

ここで，線形位相 FIR フィルタの場合には，式(4.3)は式(4.4)のように書き換えられる．

$$\left|H(\omega_k)\right| = h(M) + 2\sum_{n=0}^{M-1} h(n)\cos\left((M-n)\omega\right) \tag{4.4}$$

$$M = \frac{N-1}{2} \tag{4.5}$$

実数値 GA においては，子個体の実数ベクトルをフィルタ係数として扱う．そのフィルタ係数から周波数特性を式(4.4)により計算し，理想的なバンドストップフィルタの周波数応答との絶対誤差の総和を計算し，これを適合度とする．子個体の評価に使用する評価関数を式(4.6)に示す．

$$fitness = \sum abs\left[abs\left(\left|H(\omega)\right| - 1\right) - \delta_p\right] + \sum abs\left(\left|H(\omega)\right| - \delta_s\right) \tag{4.6}$$

ここで，δ_pは通過帯域のリプル，δ_sは遮断帯域のリプルである．リプルはギブス現象により発生する．これは不連続点を持つ周期関数をフーリエ級数で表した場合に生じる現象である．

誤差を最小にする評価関数では，理論的には絶対 2 乗誤差を用いた方が優れたフィルタが設計できると考えられるが，実行時間においては計算量の少ない差分による絶対誤差が優れると考えられる．絶対誤差による評価関数を用いた先行研究［105, 106]においても，設計条件通りのフィルタが設計されているため問題ないと思われる．

実行時間の削減

式(4.4)においてコサイン関数が存在し，個体の評価計算の度に頻繁に計算される．実数の三角関数の計算は，実行時間に大きく影響する．そこで，実行時間の短縮のために，MicroBlaze のプログラムではコサインの関数値を表（定数配列）として持たせる．サンプル数とフィルタ次数が既知であるならば，事前に計算が可能である．

4.5　バンドストップフィルタ設計の実装実験

提案する実数値 GA 専用プロセッサを用いて実装実験を行った．使用した環境は第 3 章で示したものと同じである．また比較として，PSO と DE でも実行した．

音声信号のサンプル数は 128 とし，周波数はナイキスト周波数をπとした[0, 1]の正規化周波数として扱う．FIR フィルタの設計条件を表 4.1 に示す．また，実験に用いた実数値 GA のパラメータを表 4.2，DE のパラメータを表 4.3, PSO のパラメータを表 4.4 に示す．DE および PSO のパラメータは文献［105, 106]を参考に設定した．また，

表 4.1 FIR フィルタの設計条件

Parameter	Value
Sample	128
Sampling frequency (f_s)	1
Pass band ripple (δ_p)	0.03
Stop band ripple (δ_s)	0.003
Lower pass band (normalized) cut-off frequency (ω_{pl})	0.3
Lower stop band (normalized) cut-off frequency (ω_{sl})	0.35
Upper pass band (normalized) cut-off frequency (ω_{ph})	0.75
Upper stop band (normalized) cut-off frequency (ω_{sh})	0.7

事前実験により個体数の調整だけを行い，評価回数が実数値 GA と同一になるように反復回数を調整している．また，すべての結果は 30 回実行したときの平均値である．

表 4.2 実数値 GA のパラメータ

Parameter		Filter order (2(N-1))		
		26 (N=14)	58 (N=30)	122 (N=62)
N_p		16	32	64
P		100	150	300
N_{os}	PC	100	150	300
	FPGA	$4N_p$ (= 96)	$5N_p$ (= 160)	$5N_p$ (= 320)
Evaluation times	PC	40,000	60,000	120,000
	FPGA	38,400	64,000	128,000

表 4.3 DE のパラメータ

Parameter	Filter order (2(N-1))		
	26 (N=14)	58 (N=30)	122 (N=62)
Population size	50	50	100
Evaluation times	40,000	60,000	120,000
Crossover rate	0.5		
Scaling factor F	0.4		

表 4.4 PSO のパラメータ

Parameter	Filter order (2(N-1))		
	26 (N=14)	58 (N=30)	122 (N=62)
Population size	100	150	300
Evaluation times	40,000	60,000	120,000
C_1, C_2	2.05		
V_{min} / V_{max}	-1.0 / 1.0		
W_{min} / W_{max}	0.4 / 1.0		

26 次 FIR フィルタ（N = 14）の結果

各アルゴリズムにおける適合度の遷移を図 4.5 に, 最終的な適合度を表 4.5 に示す. すべてのアルゴリズムにおいて，10,000 回の評価内にほぼ収束している．また，実数値 GA の結果は，FPGA と PC でほぼ同等な結果であり，その他のアルゴリズムとも同程度である．設計された FIR の周波数応答を図 4.6, 図 4.7 に示し，フィルタの性能を表 4.6 に示す. 設計されたフィルタにおいても, すべてのアルゴリズムで同程度であった.

表 4.5 各アルゴリズムの適合度（N=14）

Environment	Algorithm	Fitness		
		Minimum	Average	Maximum
FPGA	RCGA	3.30	3.30	3.32
PC	RCGA	3.30	3.30	3.31
	DE	3.30	3.32	3.36
	PSO	3.32	3.37	3.46

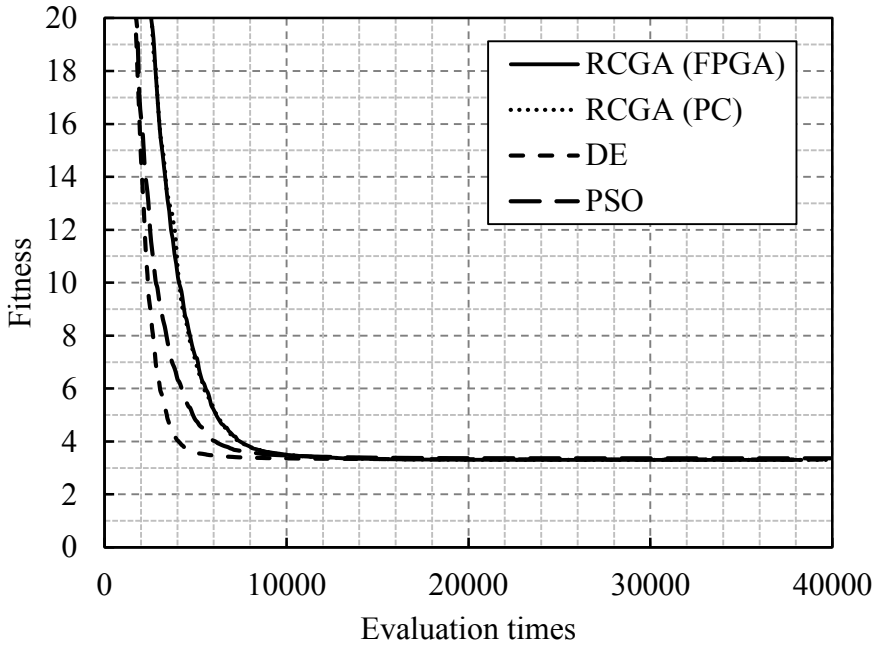
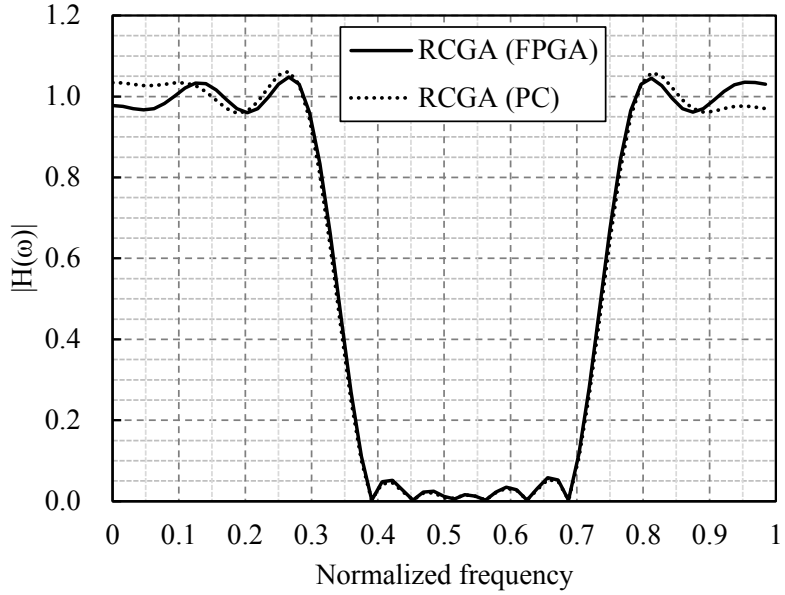
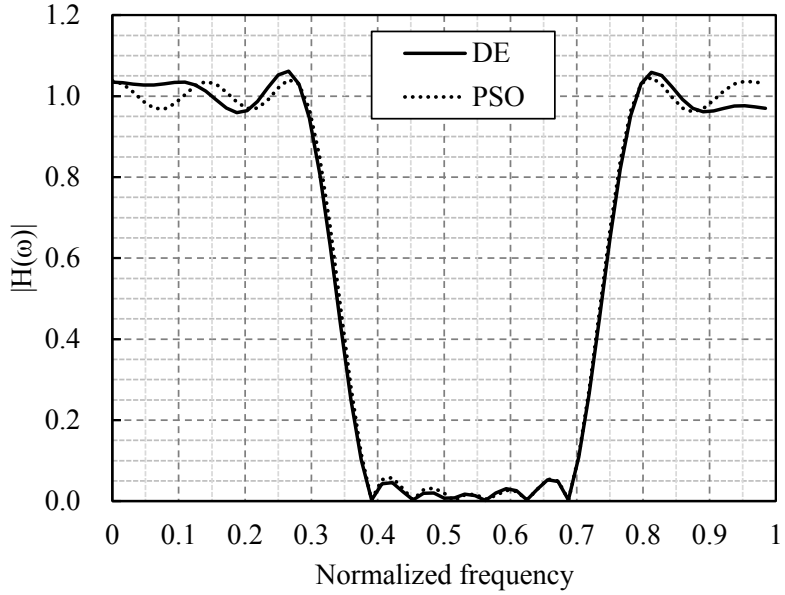


図 4.5. 各アルゴリズムの適合度の遷移（N=14）



(a) 実数値 GA の結果

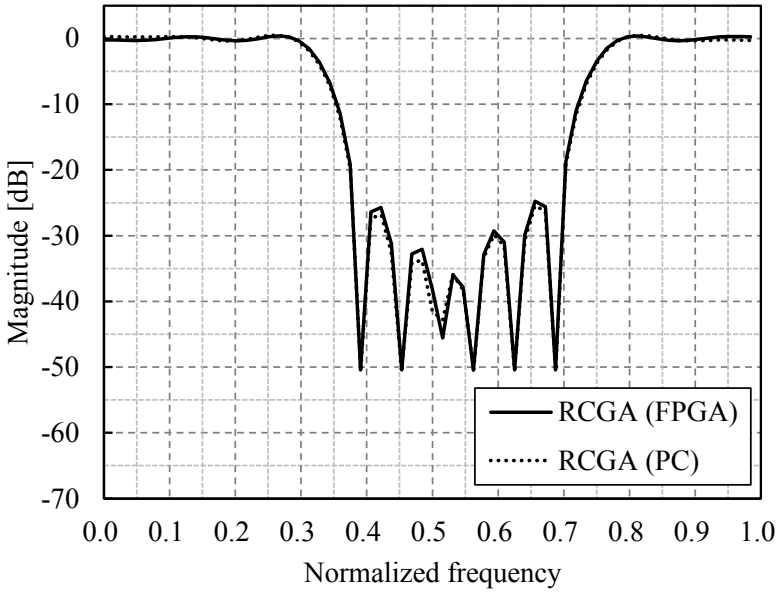


(b) DE, PSO の結果

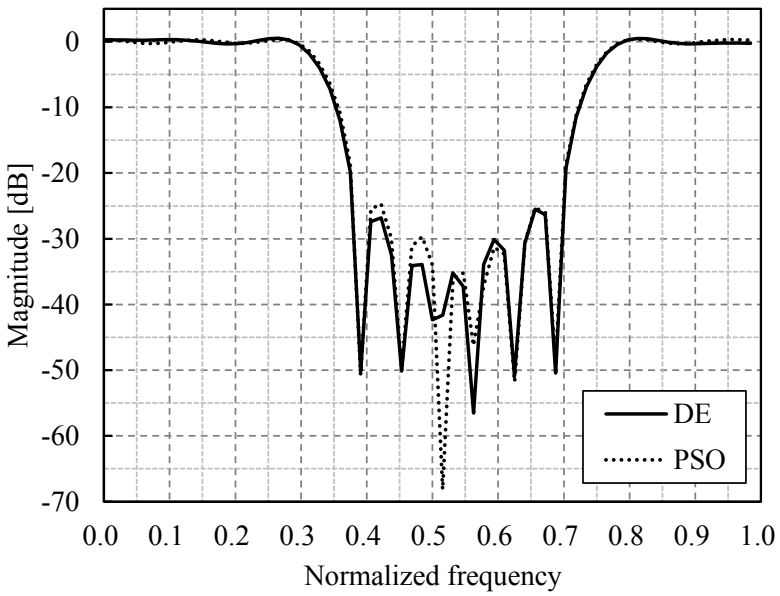
図 4.6. バンドストップフィルタ(26 次)の周波数応答（絶対値）

表 4.6 設計されたフィルタの性能（N=14）

Environment	Algorithm	Stop band attenuation (dB)		
		Minimum	Average	Maximum
FPGA	RCGA	11.38	34.64	50.46
PC	RCGA	11.99	35.20	50.47
	DE	11.99	35.42	56.51
	PSO	10.90	35.18	68.16



(a) 実数値 GA の結果



(b) DE, PSO の結果

図 4.7. バンドストップフィルタ(26 次)の周波数応答（dB）

58 次 FIR フィルタ（N = 30）の結果

各アルゴリズムにおける適合度の遷移を図 4.8 に，最終的な適合度を表 4.7 に示す．実数値 GA，DE において，30,000 回の評価内にほぼ収束している．しかし，PSO においては．適合度は 7 程度に留まっている．設計された FIR の周波数応答を図 4.9，図 4.10 に示し，フィルタの性能を表 4.8 に示す．設計されたフィルタは実数値 GA と DE がほぼ同程度であり，PSO よりも優れる結果となった．

表 4.7 各アルゴリズムの適合度（N=30）

Environment	Algorithm	Fitness		
		Minimum	Average	Maximum
FPGA	RCGA	1.75	1.84	2.25
PC	RCGA	1.72	1.78	1.88
	DE	1.76	1.89	3.03
	PSO	3.54	7.51	23.72

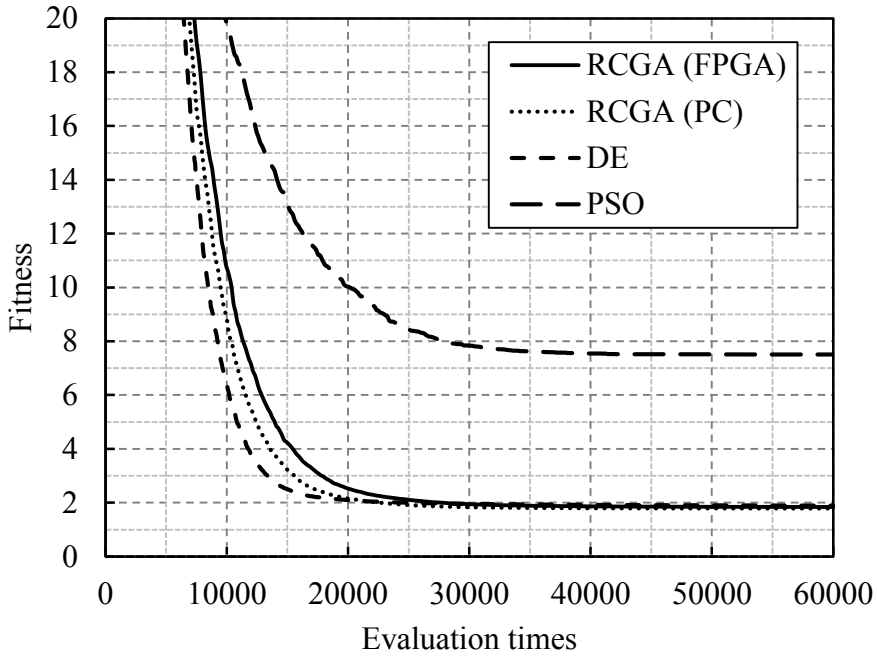
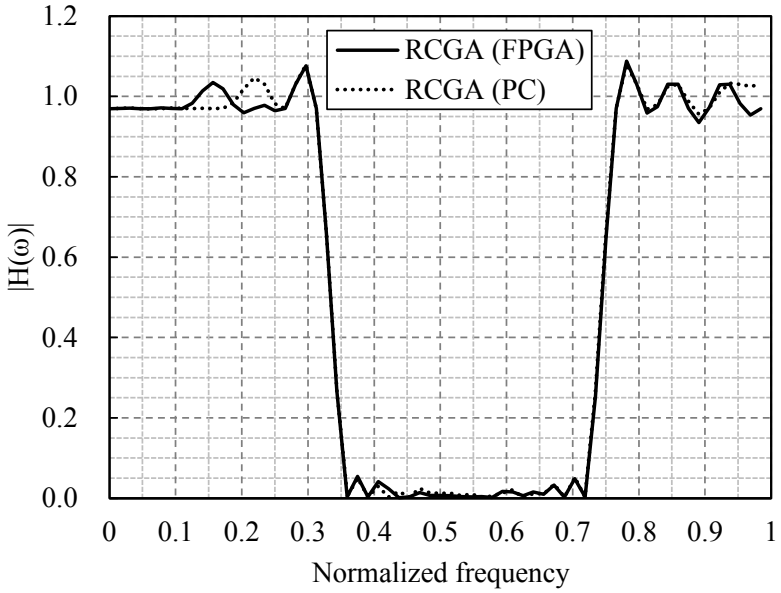
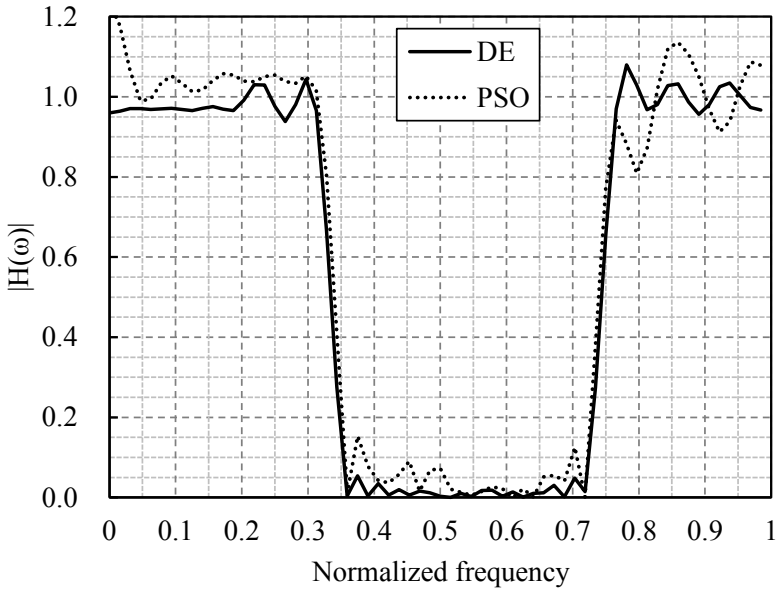


図 4.8. 各アルゴリズムの適合度の遷移（N=30）



(a) 実数値 GA の結果

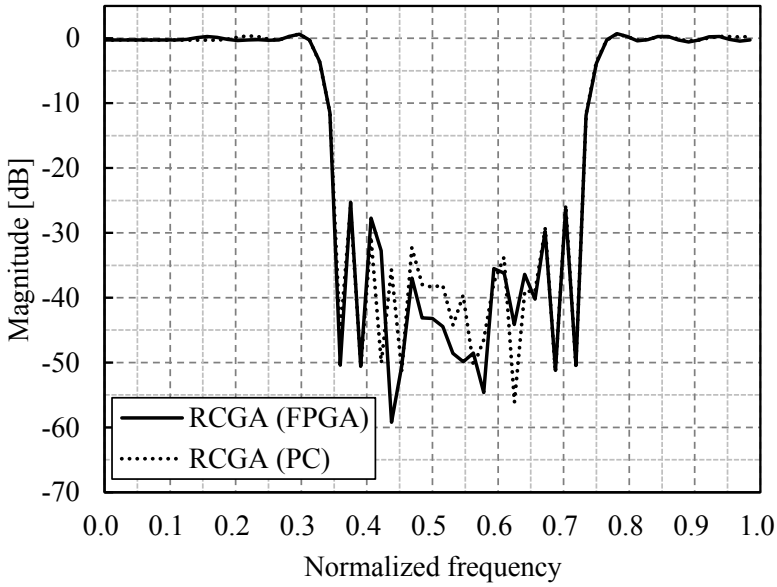


(b) DE, PSO の結果

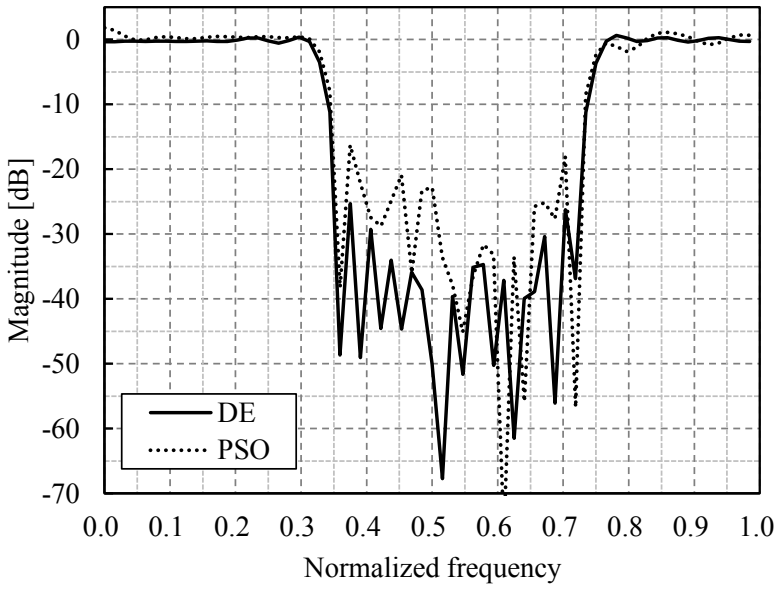
図 4.9. バンドストップフィルタ(58 次)の周波数応答（絶対値）

表 4.8 設計されたフィルタの性能（N=30）

Environment	Algorithm	Stop band attenuation (dB)		
		Minimum	Average	Maximum
FPGA	RCGA	25.30	41.60	59.20
PC	RCGA	26.11	40.18	56.11
	DE	25.30	42.87	67.74
	PSO	16.28	32.84	75.94



(a) 実数値 GA の結果



(b) DE, PSO の結果

図 4.10. バンドストップフィルタ(58 次)の周波数応答（dB)

122 次 FIR フィルタ（N = 62）の結果

各アルゴリズムにおける適合度の遷移を図 4.11 に, 最終的な適合度を表 4.9 に示す. 実数値 GA の収束が最も早い．対して，DE は収束までに 100,000 回程度の評価が必要となる．設計された FIR の周波数応答を図 4.12, 図 4.13 に示し，フィルタの性能を表 4.10 に示す. また, PSO はフィルタの設計に失敗している. 設計されたフィルタは, 実数値 GA で設計したフィルタの平均値が DE よりも優れている.

表 4.9 各アルゴリズムの適合度（N=62）

Environment	Algorithm	Fitness		
		Minimum	Average	Maximum
FPGA	RCGA	0.44	1.19	2.64
PC	RCGA	0.28	0.98	2.23
	DE	0.86	2.75	11.97
	PSO	31.68	51.66	88.21

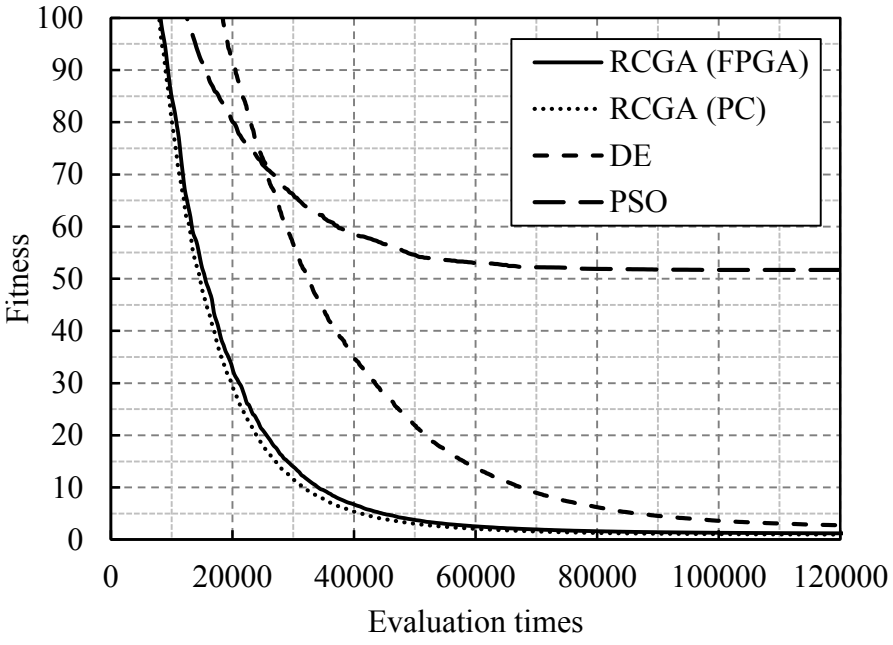
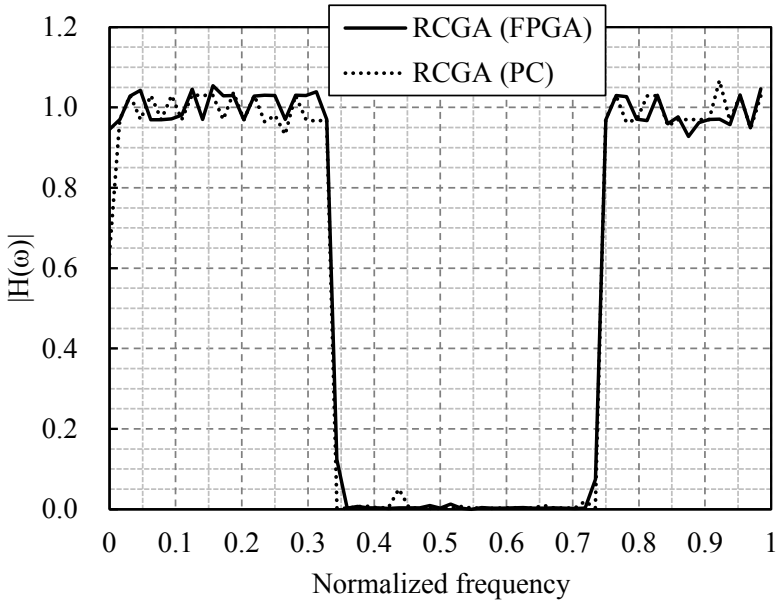
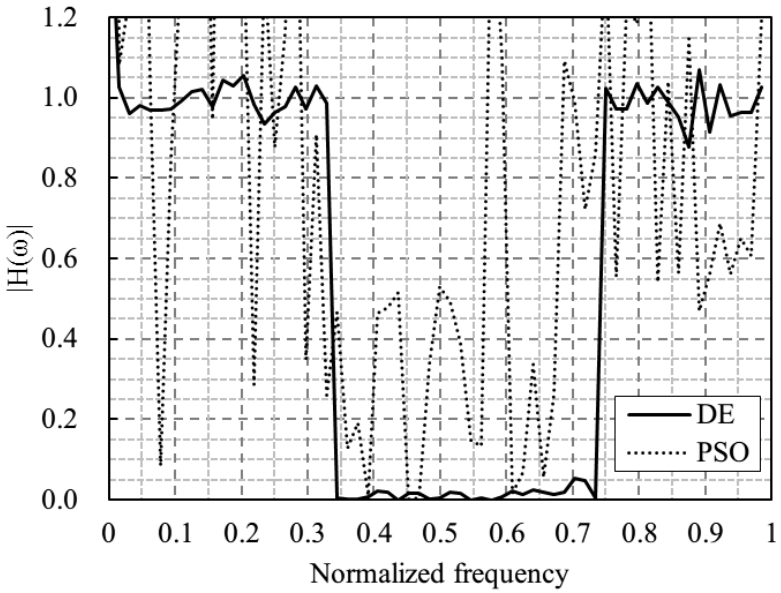


図 4.11. 各アルゴリズムの適合度の遷移（N=62）



(a) 実数値 GA の結果

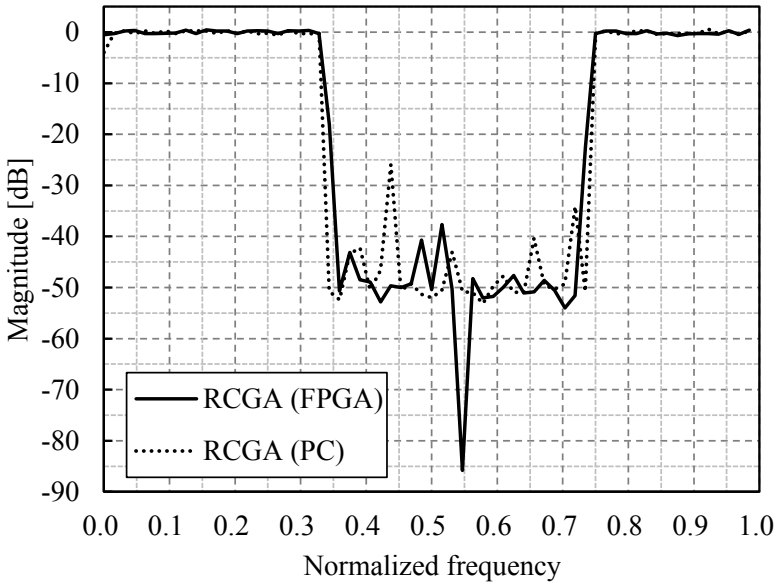


(b) DE, PSO の結果

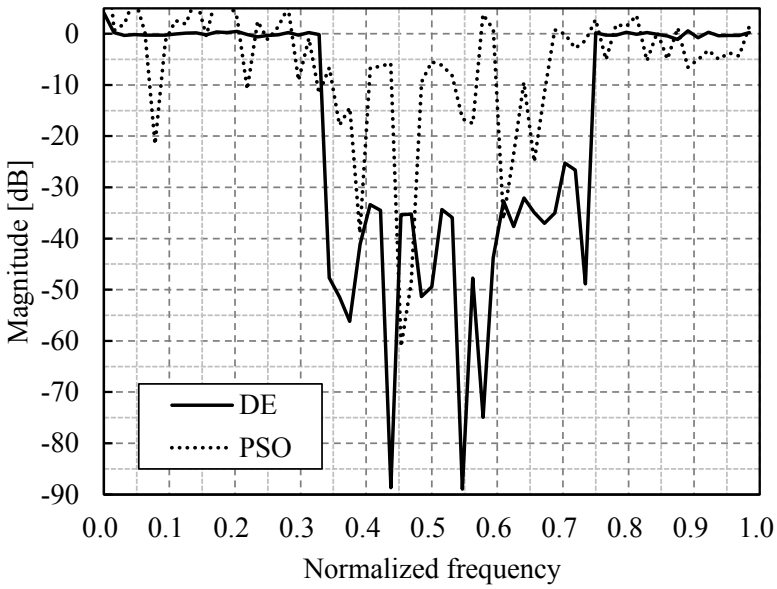
図 4.12. バンドストップフィルタ(122 次)の周波数応答 (絶対値)

表 4.10 設計されたフィルタの性能（N=62）

Environment	Algorithm	Stop band attenuation (dB)		
		Minimum	Average	Maximum
FPGA	RCGA	37.68	50.39	85.83
PC	RCGA	25.94	47.83	53.11
	DE	25.24	45.99	88.94
	PSO	3.80	16.52	61.13



(a) 実数値 GA の結果



(b) DE, PSO の結果

図 4.13. バンドストップフィルタ(122 次)の周波数応答（dB)

実行時間の比較

各環境および各アルゴリズムにおける実行時間を表 4.11 に示す．提案する実数値 GA 専用プロセッサは，PC と比較して実行時間は遅くなるものの，26 次の場合には 104ms 程度，58 次の場合には 284ms 程度，122 次の場合には 1697ms 程度で設計できる．したがって，次数があまり高くない場合には，リアルタイムにフィルタを切り替える必要のある装置への応用が期待できる．

表 4.11 各環境および各アルゴリズムにおける実行時間

Filter Order (N)	Evaluation times	Execution times [ms] (Best fitness)			
		FPGA (Virtex-7)	PC (Intel Xeon CPU E5-1630, 3.7GHz),GCC4.8.2		
		RCGA	RCGA	DE	PSO
26 (14)	20,000	104 (3.30)	17.2 (3.30)	14.6 (3.30)	14.6 (3.38)
	40,000	207 (3.30)	34.3 (3.30)	23.9(3.30)	28.6 (3.32)
58 (30)	30,000	284 (1.70)	73.8 (1.75)	47.8 (1.82)	52.5 (13.7)
	60,000	570 (1.74)	145 (1.72)	98.8 (1.76)	99.8 (3.54)
122 (62)	60,000	847 (1.14)	381 (1.01)	215 (9.11)	240 (62.5)
	120,000	1697(0.44)	801 (0.28)	429 (0.86)	463 (31.7)

実行時間についての考察

実数値 GA 専用プロセッサをベンチマーク関数に適用した場合には，PC と比べて実行時間の改善は約 2.9 倍であったが，バンドストップフィルタの設計に適用した場合，PC と比較して 1/6 から 1/2 に性能は低下した．

この原因として，ソフトマクロ汎用 CPU の MicroBlaze が多くの時間を消費していると考えられる．バンドストップフィルタ設計に使用した評価関数は，信号のサンプル数の増加にしたがって，計算回数が増えるため，実行時間が増加する．そこで，N=62(122 次)のときの信号のサンプル数を 64, 32, 16 と減らして実行時間を確認した．その結果を表 4.12 に示す．表 4.12 の結果より，サンプル数 32 で PC と同等の実行時間となり，サンプル数 16 にすると FPGA の方が短い実行時間で実行が完了する．また，図 4.14 にサンプル数に対する実行時間のプロットを示す．線形近似を行ったところ，サンプル数が増加すると PC で約 3 倍実行時間は増加し，FPGA で約 12 倍実行時間は増加する．したがって，サンプル数が増加すると PC と FPGA の実行時間の差は約 4 倍ずつ広がると考えられる．

これらの結果より，より機能を絞り，高速に処理をするソフトマクロ CPU に置き換えるかであると考えられる．それでも実行時間が要求を満たさなければ，専用回路化することが望ましいと考えられる．

表 4.12 信号のサンプル数と実行時間（N=62，122 次）

Sample	Execution times [ms]		Speed up
	PC	FPGA	
128	801	1697	0.47
64	616	930	0.66
32	522	542	0.96
16	486	353	1.38

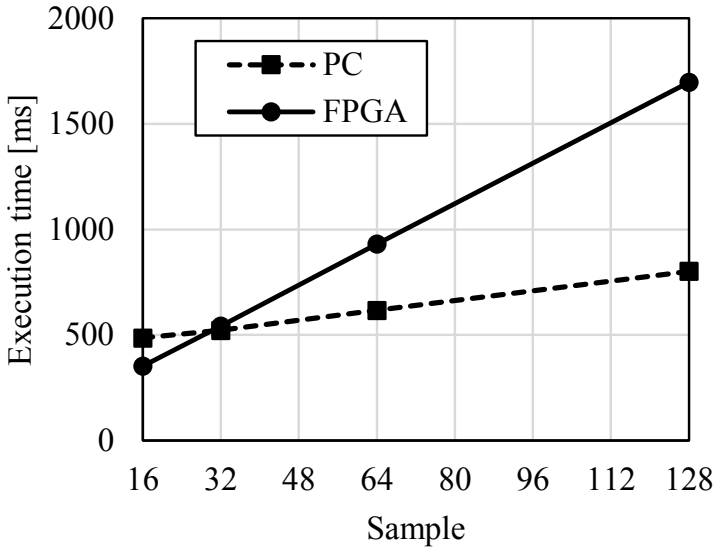


図 4.14. 信号のサンプル数増加に伴う実行時間の増加

4.6　ディジタル補聴器への適用

前節のディジタルフィルタ設計の応用として，ディジタル補聴器について検討する．人間は加齢に伴って高い周波数が聴こえづらくなる傾向がある．また，生活環境や様々な要因により難聴になることがある．そのため，難聴者ごとに必要となる補聴器の特性は異なる．聴き取りづらい音域を聴き取りやすい音量に合わせることをフィッティングという．

従来の補聴器では，帯域幅を細かく区切った多くのゲイン調整用のフィルタを持たせていることが多い．これに対し，計算量の削減と低消費電力化のため，3 つの可変フィルタを組み合わせてフィッティングを行う方法が提案されている［108, 109］．また本研究と同様に FIR フィルタを用いてフィッティングを行った事例もある［110］．

4.6.1 オージオグラム

オージオグラム（聴力図）とは，聴力検査の結果の図である．入力された各周波数の音声信号に対する可聴な音圧レベルを示している．その一例を図 4.15 に示す．図の○は右耳で取得した結果であり，X は左耳で取得した結果である．図の点線で囲まれた領域は，人間が会話する音声の周波数帯域と音圧レベルがおおよそ収まる領域であり，スピーチバナナと呼ばれる．また，聴力と難聴の程度を表 4.13 に示す．このオージオグラムに基づいて，補聴器のフィッティングが行われる．

表 4.13 聴力と難聴の程度［113］

程度	測定値［dB］	実際の聞こえ具合
正常	0～25	聞こえに問題はない
軽度	25～40	小声だとやや聞き取り難い
中度	40～70	普通の会話の聞き取りが困難
高度	70～90	耳元の大声なら聞こえる
聾	90～	殆ど何も聞こえない

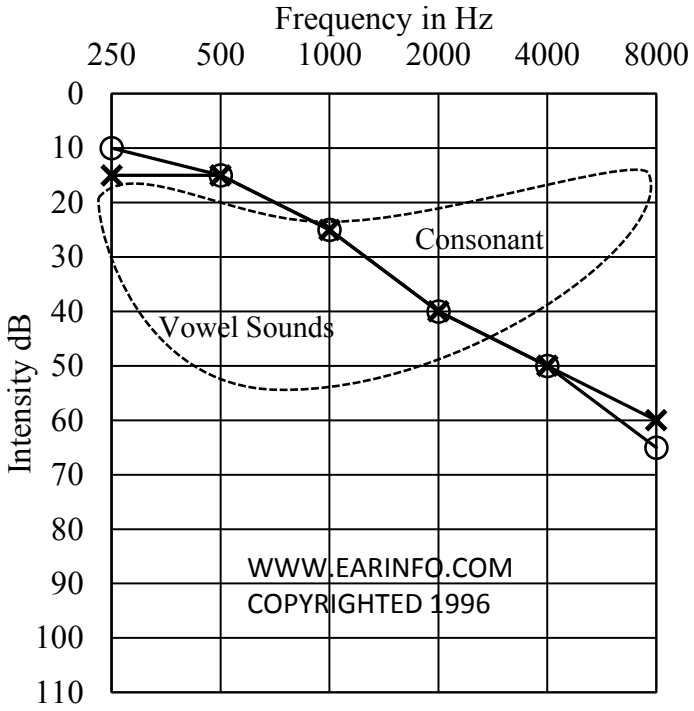


図 4.15. オージオグラムの例

4.6.2 実数値 GA 専用プロセッサを用いたフィッティング

以下の 6 つのオージオグラムの例を用いて, 実数値 GA 専用プロセッサによりフィッティングを行った．すべて右耳の結果に対してフィッティングを行った.

1. 高齢者の典型的なオージオグラム
2. 長い間騒音の多い環境下で生活してきた難聴者に多く見られるパターン
3. 軽度の難聴
4. 軽度の難聴（低い周波数が聴こえづらい）
5. オージオグラム 1 よりさらにひどくなったもの(補聴器がなければ会話も厳しい)
6. オージオグラム 1 から 5 よりもさらに難聴が進んだもの

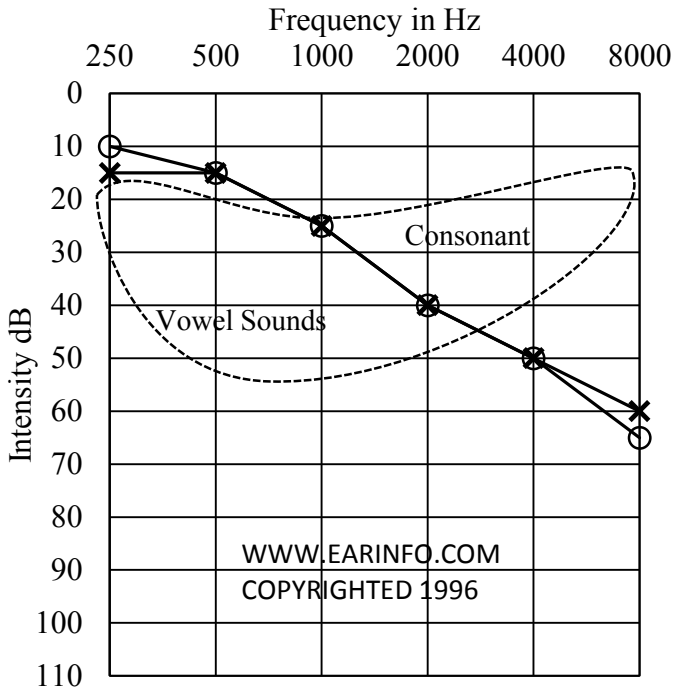
また，実数値 GA に用いた評価は，FIR フィルタの振幅特性（$H(\omega)$）とオージオグラムの理想特性（$D(\omega)$）との絶対誤差を最小化することにより行う．その評価関数を式(4.7)に示す.

$$fitness = \sum abs\left(|H(\omega)| - D(\omega)\right) \qquad (4.7)$$

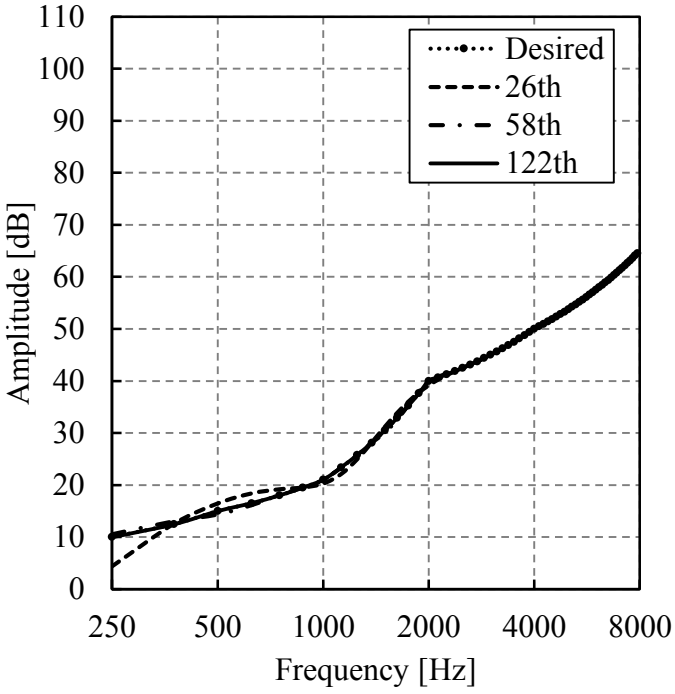
各オージオグラムに対して，フィッティングを行った結果を図 4.16～図 4.21 に示す．なお，実験に用いた実数値 GA のパラメータを表 4.14 に示す．26 次と 58 次の場合には，表 4.2 で示した条件と同じであり，122 次の場合には子個体数と評価回数を変更している．

表 4.14 実数値 GA のパラメータ

Parameter	Filter order (2(N-1))		
	26 (N=14)	58 (N=30)	122 (N=62)
N_p	16	32	64
P	100	150	400
N_{os}	$4N_p$ (= 96)	$5N_p$ (= 160)	$6N_p$ (= 384)
Evaluation times	38,400	64,000	192,000

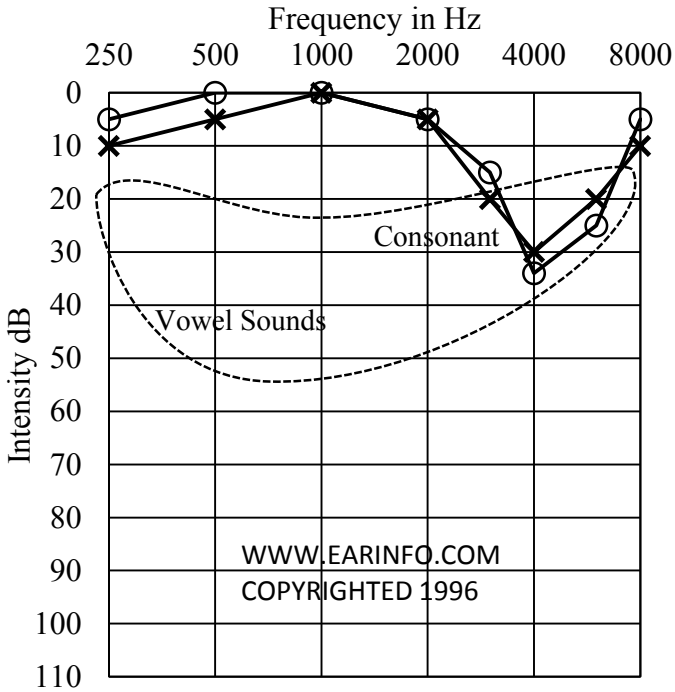


(a) オージオグラム 1

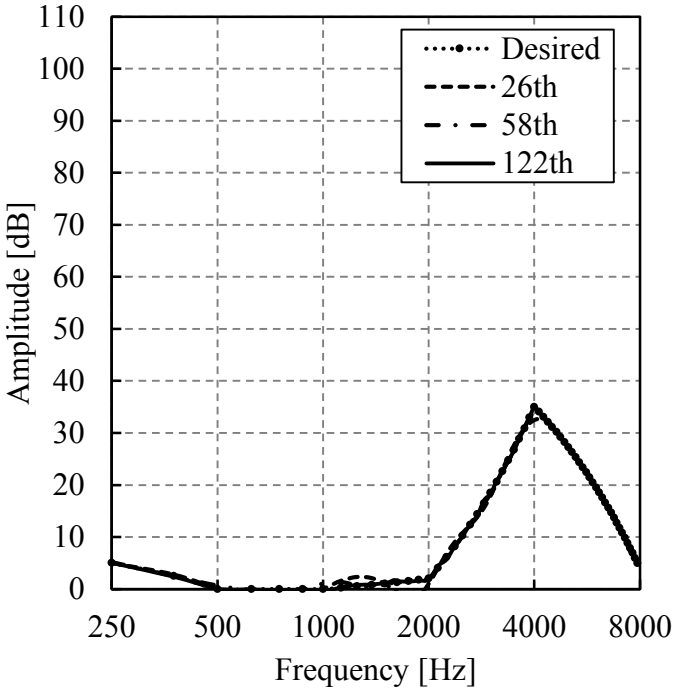


(b) 実数値 GA プロセッサでの結果

図 4.16. オージオグラム 1 のフィッティング結果

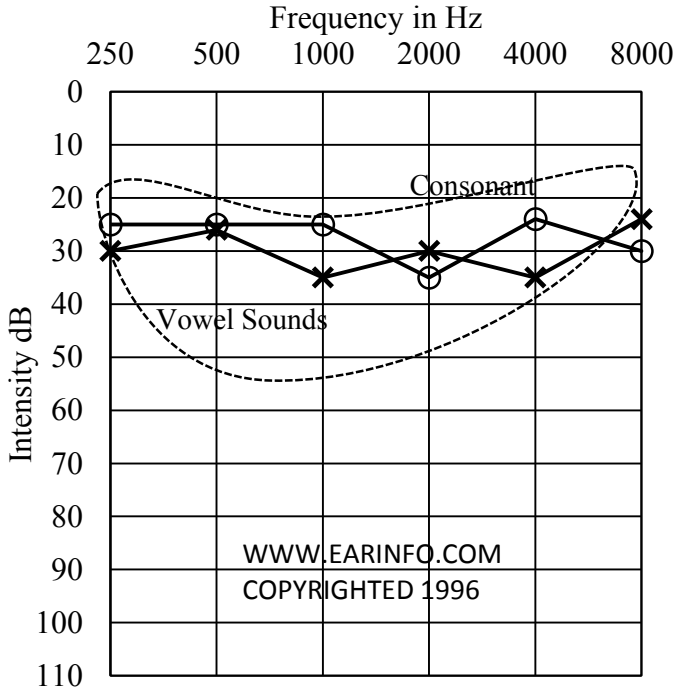


(a) オージオグラム 2

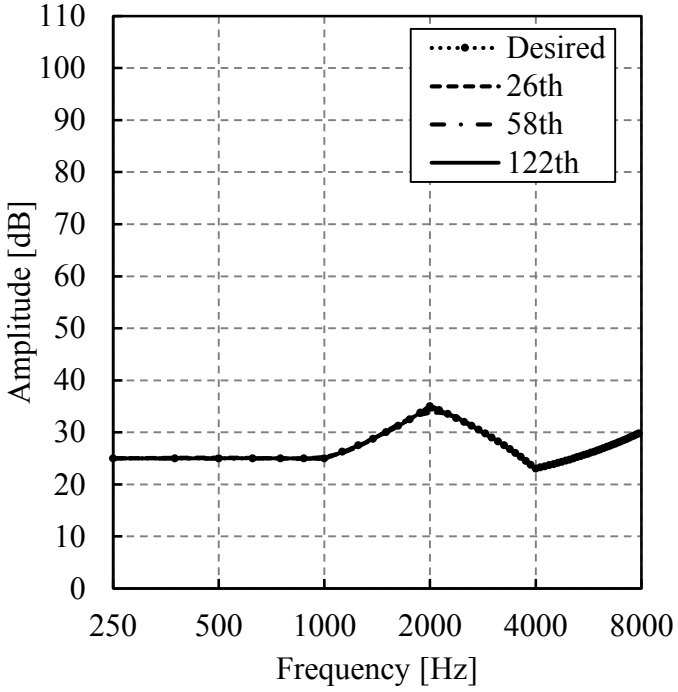


(b) 実数値 GA プロセッサでの結果

図 4.17. オージオグラム 2 のフィッティング結果

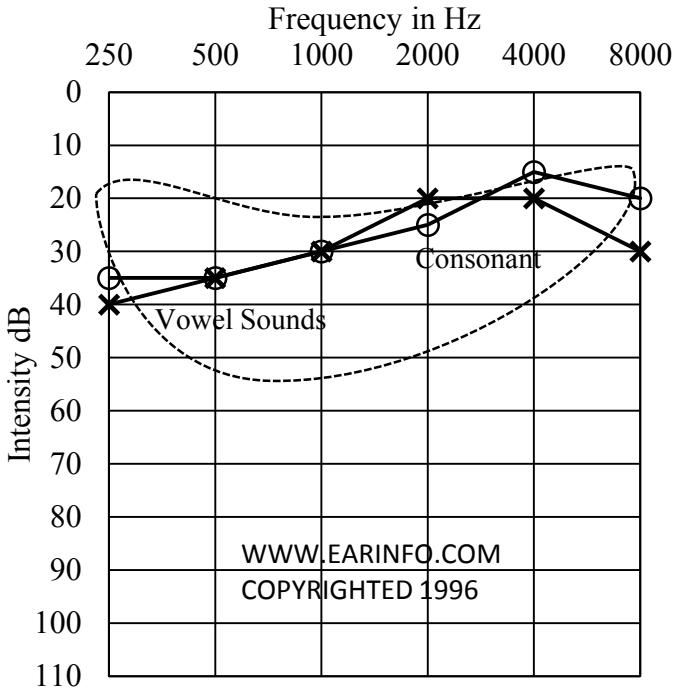


(a) オージオグラム 3

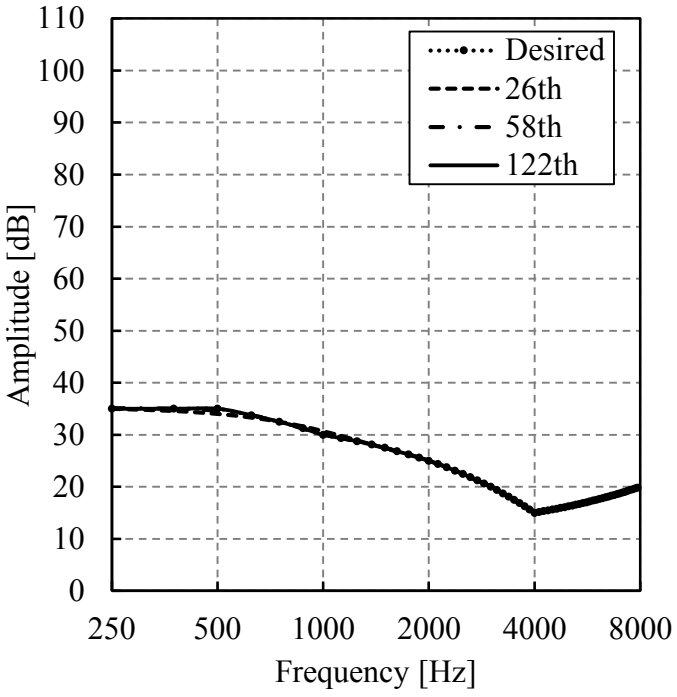


(b) 実数値 GA プロセッサでの結果

図 4.18. オージオグラム 3 のフィッティング結果

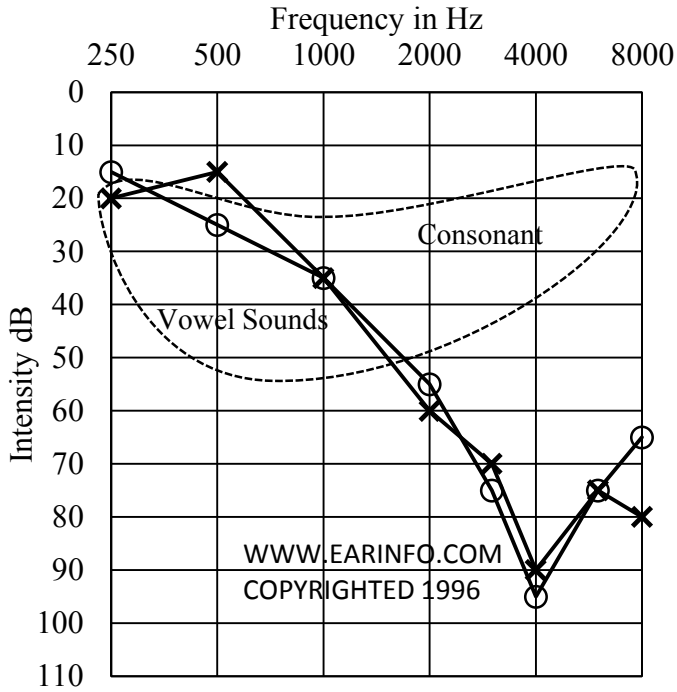


(a) オージオグラム 4

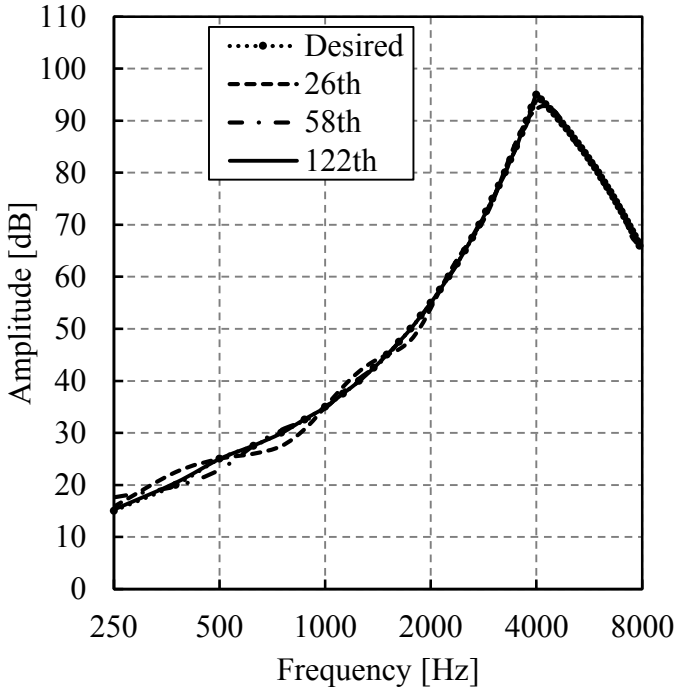


(b) 実数値 GA プロセッサでの結果

図 4.19. オージオグラム 4 のフィッティング結果

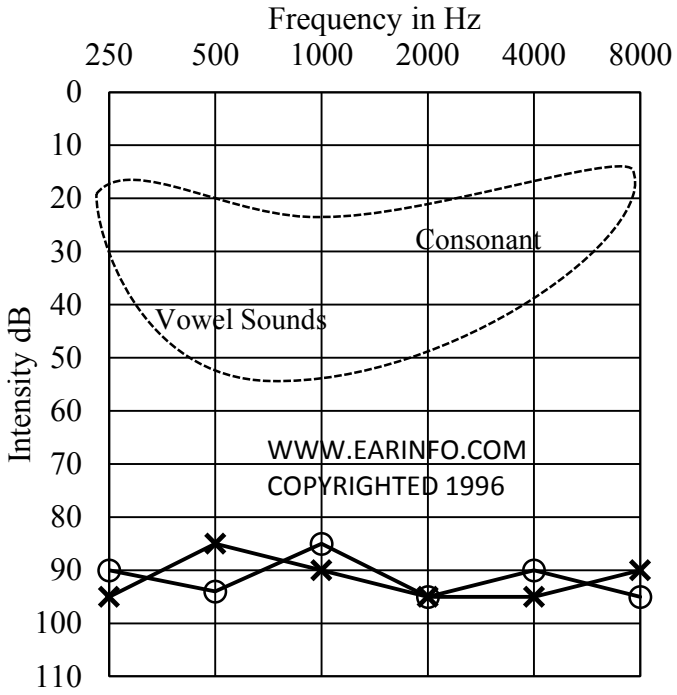


(a) オージオグラム 5

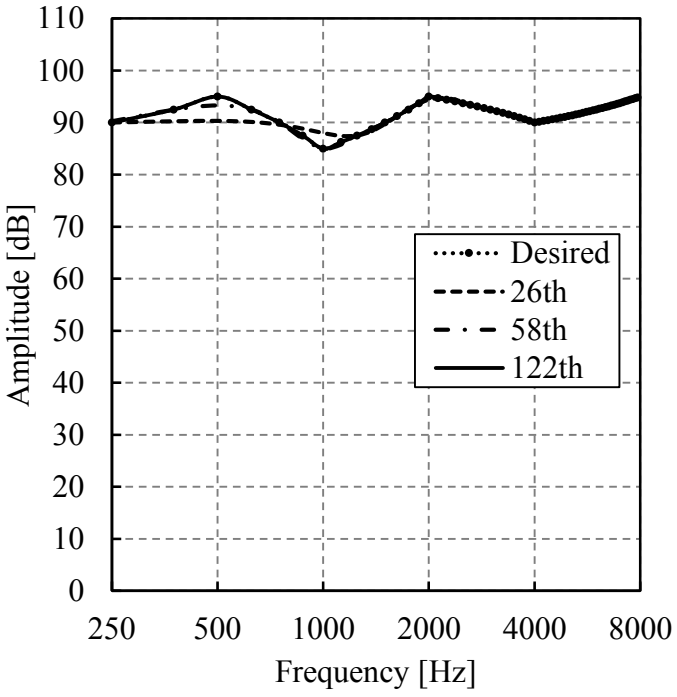


(b) 実数値 GA プロセッサでの結果

図 4.20. オージオグラム 5 のフィッティング結果



(a) オージオグラム 6



(b) 実数値 GA プロセッサでの結果

図 4.21. オージオグラム 6 のフィッティング結果

全てのオージオグラムにおいてフィッティング可能であることを確認した．26 次, 58 次の FIR フィルタでも，良好にフィッティングできていると考えられる．さらに，122 次の場合には，必要な評価回数は増加したが，より周波数帯域の細かいフィッティングも可能と考えられる．

本節で示したように，任意の周波数応答のフィッティングが可能であった．58 次の場合には，表 4.2 で示した条件と同じであるため，表 4.11 で示した実行時間で設計することができる．即ち 570ms 程度でフィッティングが可能である．

したがって，オージオグラムを取得できる仕組みを組み込めば，対話的に補聴の調整を行うことができる他，調整後の微調整を自由に行うことができると考えられる．さらに，環境による騒音などを検知する仕組みを組み込めば，環境による変化に適応的に対応するようなディジタル補聴器などが考えられる．環境によるノイズ除去は，ディジタル補聴器だけでなく，音楽機器，通信分野など様々な分野に適用可能であると考えられる．

4.7　補聴器フィッティングの先行研究との比較

補聴器フィッティングのシミュレーション結果を文献 [108, 111]と比較を行った．文献 [108]は，3 種類の可変フィルタを組み合わせてフィッティングを行う方法を提案している．文献 [108]では，18 個のパラメータ（通過帯域周波数端など）をネルダーミード法 [112]で最適化を行っている．文献 [111]は，文献 [108]と同様な構成で GA により最適化を行っている．表 4.15 に各オージオグラムにおける，本研究と 2 つの文献の絶対フィッティング誤差を示す．文献 [111]は本研究で評価に用いたオージオグラムの内，オージオグラム 3 での結果が記載されていたためその結果を記す．実数値 GA 専用プロセッサを用いた場合の 58 次のフィルタの結果は，文献 [108]の結果と同等以上であり，122 次の場合には全てのオージオグラムにおいて，絶対フィッティング誤差は小さい．また，文献 [111] のオージオグラム 3 の結果よりも，実数値 GA 専用プロセッサの 58 次と 122 次共に，絶対フィッティング誤差は小さいことが確認できた．

一般的に，FIR フィルタより IIR フィルタの方が少ない次数で実現できる．2 つの文献で最適化しているパラメータ数は 18 個であり，実数値 GA 専用プロセッサの 58 次では最適化しているパラメータ数は 30 個，122 次では 62 個である．

したがって，2 つの文献と同様な構成もしくは，IIR フィルタに対応することも今後の課題と考えられる．

表 4.15 先行研究との比較（絶対フィッティング誤差）

Audiogram	Maximum fitting error [dB]				
	Proposed RCGA processor			Reference [108] (3ch variable filter bank (VFB), 2009)	Reference [111] (3ch VFB(GA), 2011)
	26th	58th	122th		
1	5.64	0.68	0.07	1.88	-
2	3.32	1.10	0.60	2.92	-
3	1.25	0.63	0.17	1.77	0.42
4	0.99	0.38	0.28	0.94	-
5	2.75	2.61	0.79	2.35	-
6	4.69	1.69	0.07	2.49	-

また，実際の補聴器に組み込まれている DSP チップなどは，非常に小型化されており，現在使用しているチップサイズの大きい FPGA をそのまま実装することは難しいと思われる．また一般に，FPGA は，マルチコア CPU や GPU よりも消費電力は小さいものの，マイクロコントローラや補聴器に用いられている DSP チップなどより消費電力は高い．そのため，電力の面においても実装は厳しいと考えられる．しかし，本研究のようなシミュレーションは，補聴器に組み込まれるチップの要素技術として，専用回路の設計の参考や指針になると思われる．

組み込み機器やモバイル機器は基本的には，バッテリ駆動である．消費電力は動作周波数と使用している回路規模なども大きく影響する．次数を上げるとフィルタの性能は上がるが，多くの回路規模を消費するため，多くの電力を消費すると考えられる．そこで，必要な性能を満たしつつ，小規模な回路で実現するような工夫を入れることも今後の課題である．

4.8　進化型ハードウェアのアーキテクチャの検討

4.6.2 項では，実数値 GA 専用プロセッサが，任意の周波数応答に対してフィッティング可能であることを示した．そこで，更なる発展として，これまで示してきた実数値 GA 専用プロセッサと FIR フィルタを評価回路として扱うアーキテクチャなどが考えられる．つまり，環境に応じた理想的な周波数応答を最適化する進化型ハードウェアである．その例を図 4.22 に示す．その実現のためには，ノイズと音声の分離する仕組みなどを組み込まねばならないため，今後の研究課題と考えている．

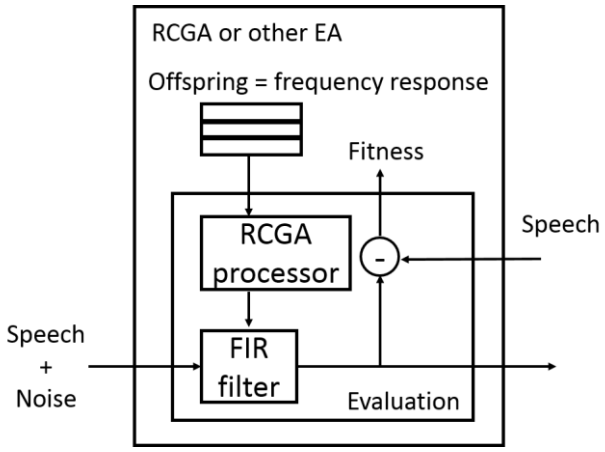


図 4.22. 進化型ハードウェアのアーキテクチャの例

4.9　結言

本章では実数値 GA 専用プロセッサを用いた一応用として，ディジタルフィルタの設計に適用した事例の結果を示した．26 次，58 次，122 次の線形位相 FIR フィルタにおいてバンドストップフィルタを設計する例を示した．26 次の場合には 104ms 程度，58 次の場合には 284ms 程度，122 次の場合には 1697ms 程度で設計できる．

進化アルゴリズムによるディジタルフィルタの設計は，補聴器などへの適用が考えられる．そこで，6 つのオージオグラムに対してフィッティングを行い，すべてフィッティング可能であることを確認した．

第5章　結　論

5.1　本研究の成果

本研究は，汎用的な進化型ハードウェアのアーキテクチャの確立のため，実数をそのまま扱うことができる実数値遺伝的アルゴリズム（GA）に着目し，実数値 GA 専用プロセッサを開発した．提案したプロセッサは，評価計算をソフトマクロ汎用 CPU で実行するため，対象とする問題が変更になってもプログラムの変更だけで対応できる．さらに，未知パラメータの多い高次元の問題を扱う際に課題となる回路規模の増加に対して，リソースシェアリングを適用することにより緩和している．そして，実数値 GA 専用プロセッサによって回路構成や回路定数を決める進化型ハードウェアを提案した．

本論文の第 1 章では，本研究の背景として，進化型ハードウェアの特徴と解決すべき課題を示し，本研究を行った目的について述べ，本論文の構成と各章の概要についてまとめた．

また第 2 章では，進化型ハードウェアの基本的な構成を示し，回路構成や回路定数を決定する進化アルゴリズムとそれらの専用プロセッサの概要を述べた．進化アルゴリズム専用プロセッサの一例として，リソースシェアリングを適用した遺伝的アルゴリズム専用プロセッサについて述べた．

第 3 章では，汎用的な進化型ハードウェアのアーキテクチャの確立を目的として開発した，実数値 GA 専用プロセッサの構成について詳細に述べた．実数値 GA は，実数値をそのまま扱うことができ，通常の GA よりも探索能力が優れる．提案する実数値 GA 専用プロセッサでは，これまで対象とする問題の変更によって再設計が必要であった評価計算部分をソフトマクロ汎用 CPU によって実行する．これにより，回路の再設計が不要になりプログラムの変更によって可能となる．また，リソースシェアリングを適用することによって，必要な演算器を削減することによって回路規模を削減している．これにより，最適化すべきパラメータ数が増加しても，少ない回路規模で対応できる．

さらに，提案する実数値 GA 専用プロセッサを FPGA に実装し，ベンチマーク問題により性能評価を行った結果について述べた．その結果，実行時間において，汎用 PC と比較して，最高で約 2.9 倍の高速化が確認できた．

第 4 章では，提案する実数値 GA 専用プロセッサを用いた応用の一つとして，ディジタルフィルタへ適用した結果について述べた．一例として，バンドストップフィルタを

設計した結果を述べ，さらにディジタル補聴器のフィッティングを想定したシミュレーションを行った結果を示した．

第 5 章では，本研究の総括と今後の展望について述べた．

以上，本研究は汎用的な進化型ハードウェアのアーキテクチャに関する一連の研究をまとめたものであり，これらの成果は進化型ハードウェアの応用に貢献できるものであると期待される．

5.2　今後の展望と課題

進化型ハードウェアは柔軟性に優れたハードウェアであり，アナログ回路・ディジタル回路問わず適用できるため，幅広い分野で用いることが可能である．

今後の課題は，実行時間の更なる改善と様々な応用を行うことである．実行時間の改善では，更なる並列化が有効であると考えられる．しかし，回路規模において，MicroBlaze とその周辺回路が大きく占めている．同等以上の性能を持ち，より省面積で済む CPU に変更することで，更なる並列化によって実行時間の改善が可能と考えられる．

さらに発展として，実数値 GA 専用プロセッサとそれによって構成を決める FIR フィルタを評価回路として扱う進化型ハードウェアなどが考えられる．具体的には，補聴器・オーディオ機器への応用が期待できる．環境のノイズに適応したフィルタをリアルタイムに構成することにより，環境に応じたノイズ除去を行うことが期待できる．補聴器やポータブルなオーディオ機器は基本的に電池駆動であり，実用十分な効果を低消費電力に実現することが重要であると考えられる．そのため，実数値 GA 専用プロセッサのさらなる低消費電力化も今後の課題と考えている．

謝辞

本研究の全過程を通じて，懇切なる御指導と御鞭撻を賜りました東京電機大学先端科学技術研究科電気電子システム工学専攻 金杉昭徳教授に心から感謝の意を評します．

また，本論文をまとめるにあたり貴重な御助言を頂きました東京電機大学 田所貴志教授，西川正教授，和田成夫教授，小松聡教授に厚く御礼申し上げます．

また，回路設計や実験にご協力頂いた集積情報システム研究室卒業生の西島賢悟氏，大学院生の金子博昭氏および松井悠真氏に深く感謝いたします．さらに，集積情報システム研究室諸氏にも感謝いたします．

参考文献

[1] 樋口哲也, “特集やわらかいハードウェア：進化ハードウェア,” 情報処理, vol. 40, no. 8, pp. 795-800, 1999.

[2] 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.

[3] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo , L. Sekanina, “Self-reconfigurable evolvable hardware system for adaptive image processing,” IEEE Transactions on Computers, vol. 62, no. 8, pp1481, 2013.

[4] J. Gubbi, R. Buyya, S. Marusic , M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” Future Generation Computer Systems, vol. 29, no. 7,, pp. 1645-1660, 2013.

[5] T. Higuchi, T. Niwa, T. Tanaka, I. Iba, H. d. Garis , T. Furuya, “Evolving hardware with genetic learning: A first step towards building a Darwin machine,” In Proc. of 2nd Int. Conf. on the Simulation of Adaptive Behaviour (SAB92), MIT Press,, pp. 398-421, 1992.

[6] J. H. Holland, “Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence,” U Michigan Press, 1975.

[7] L. Davis, Handbook of Genetic Algorithms, vol. a, 1991, p. a.

[8] B. R.K. , M. Vose, Foundations of Genetic Algorithms 4, 1997.

[9] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection, MIT press, vol. 1, 1992.

[10] J. Kennedy , R. C. Eberhalt, “Particle swarm optimization,” Proc. Int. Conf. on Neural Netw., pp. 1942-1948, 1995.

[11] F. Cancare, S. Bhandari, D. B. Bartolini, M. Carminati , M. D. Santambrogio, “A bird's eye view of FPGA-based Evolvable Hardware,” Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on. IEEE, pp. 169-175, 2011.

[12] M. A. Trefzer , A. M., “Evolvable hardware: from practice to application,” Springer, 2015.

[13] 安藤晋, 石塚満 , 伊庭斉志, “可変長遺伝子を用いた進化型アナログ回路,” 人工知能学会誌, vol. 15, no. 5, pp. 844-853, 2000.

[14] 河西勇二, 坂無英徳, 高橋栄一, 村川正宏 , 樋口哲也, “進化型ハードウェア技術のマイクロ波回路への適用,” 情報処理学会論文誌数理モデル化と応用 (TOM), 43(SIG10 (TOM7)), pp. 176-182, 2002.

[15] M. Andrzej , F. Piotr, “Analog Reconfigurable Circuits,” Int. Journal of Electronics and Telecommunications, vol. 60, no. 1, pp. 8-19, 2014.

[16] H. Q. Xu, Y. S. Ding, H. Liu , X. Li, “An Immune Genetic Algorithm with Orthogonal Initialization for Analog Circuit Design,” Int. Conf. on Intelligent Computing. Springer Berlin Heidelberg, 2012.

[17] W. G. Wang, Y. B. Ling, J. Zhang , Y. Wang, “Ant colony optimization algorithm for design of analog filters,” 2012 IEEE Congress on Evolutionary Computation IEEE, pp.1-6, 2012.

[18] Y. Matsui, A. Tsukahara , A. Kanasugi, “Design of a ZNCC template matching processor based on FSBMA,” Proc. of the 21th International Symposium on Artificial Life and Robotics (AROB 2016), pp. 809-802, 2016.

[19] J. F. Miller, P. Thomson , T. C. Fogarty, “Designing Electronic Circuits Using Evolutionary Algorithms: Arithmetic Circuits: A Case Study.,” 1998.

[20] R. Storn , K. Price, “Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces,” Journal of global optimization, vol. 11, no.4, pp. 341-359.

[21] Y. Tao, J. Cao, Y. Zhang, J. Lin , M. Li, “Using module-level evolvable hardware approach in design of sequential logic circuits,” 2012 IEEE Congress on Evolutionary Computation, IEEE, pp. 1-8, 2012.

[22] L. Sekanina, “Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware,” Int. Conf. on Evolvable System, pp. 186-197, 2003.

[23] A. Gallego, J. Mora, A. Otero, E. de la Torre , T. Riesgo, “A Scalable Evolvable Hardware Processing Array,” 2013 Int. Conf. on Reconfigurable Computing and FPGAs (ReConFig). IEEE, pp. 1-4, 2013.

[24] Z. Bao , T. Watanabe, “Evolutionary design for image filter using GA,” TENCON 2009 - 2009 IEEE Region 10 Conf., pp. 1-6, 2009.

[25] V. Z. , L. Sekanina, “Evaluation of a New Platform For Image Filter Evolution, Adaptive Hardware and Systems,” Proc. of the Second NASA/ESA Conf. on Adaptive Hardware and Systems, pp. 577-584, 2007.

[26] S. Harding, “Evolution of image filters on graphics processor units using cartesian genetic programming,” 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) IEEE., pp. 1921-1928, 2008.

[27] 大塚純二, 矢田紀子 , 長尾智晴, “進化型ニューラルネットワークのセル結合モデルによる画像変換,” 電気学会論文誌 C, vol. 132, no. 3, pp. 430-438, 2012.

[28] B. Alper , E. Günay, “Efficient edge detection in digital images using a cellular neural network optimized by differential evolution algorithm,” Expert Systems with Applications, vol. 36, no. 2, pp. 2645-2650, 2009.

[29] L. O. Chua , L. Yang, “Cellular neural networks: Applications,” IEEE Transactions on circuits and systems, vol. 35, no. 10, pp. 1273-1290, 1988.

[30] 川合浩之, 山口佳樹 , 安永守利, “FPGA の動的部分再構成を利用した進化型高速パターン認識ハードウェア,” 電子情報通信学会論文誌 D, vol. 93, no. 11, pp. 2354-2367, 2010.

[31] K. Paul, G. Kyrre, G. Thiemo, P. Marco, T. Jim , S. Bernhard, “Classification of Electromyographic Signals: Comparing Evolvable Hardware to Conventional Classifiers,” IEEE Trans. on Evolutionary Computation, vol. 17, no. 1, pp. 46-63, 2013.

[32] 岩田昌也, 梶谷勇, 村川正宏, 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 D, vol. 81, no. 10, pp. 2411-2420, 1998.

[33] 坂無英徳, 岩田昌也, デイデイエケミュレン, 村川正宏, 梶谷勇, 田中雅晴 , 樋口哲也, “進化型ハードウェアと産業応用,” 電子情報通信学会技術研究報告. AI, 人工知能と知識処理, vol. 99, no. 95, pp. 17-24, 1999.

[34] 諏佐達也, 村川正宏, 高橋栄一, 樋口哲也, 古谷立美, 古市愼治 , 和田淳, “動作マージンを確保可能なディジタル LSI の製造後クロック調整手法の提案,” 情報処理学会研究報告数理モデル化と問題解決 (MPS), vol. 85, no. 71, pp. 97-90, 2008.

[35] A. Mesquita, “Introduction to evolvable hardware: A practical guide for designing self-adaptive systems,” Genetic Programming and Evolvable Machines, vol. 9, no. 3, pp. 275-277, 2008.

[36] G. Capi , D. Kenji, “Evolution of recurrent neural controllers using an extended parallel genetic algorithm,” Robotics and Autonomous Systems, vol. 52, no. 2, pp. 148-159, 2005.

[37] J. Kok, L. F. Gonzalez , N. Kelson, “FPGA Implementation of an Evolutionary Algorithm for Autonomous Unmanned Aerial Vehicle On-Board Path Planning,” IEEE trans. on evolutionary computation, vol. 17, no. 2, pp. 272-281, 2013.

[38] (社) 電気学会, 進化技術応用調査専門委員会編, 進化技術ハンドブック 第 I 巻 基礎編, 近代科学社, 2010.

[39] (社) 電気学会, 進化技術応用調査専門委員会編, 進化技術ハンドブック 第 II 巻 応用編: 情報・通信システム, 近代科学社, 2011.

[40] (社) 電気学会, 進化技術応用調査専門委員会編, 進化技術ハンドブック 第 III 巻 応用編: 生産・物流システム, 近代科学社, 2012.

[41] D. E. Goldberg, “Realcoded Genetic Algorithms Virtual Alphabets and Blocking,” Urbana 51.61801: 16., 1990.

[42] 小林重信, “実数値 GA のフロンティア,” 人工知能学会論文誌, vol. 24, no. 1, pp. 147-162, 2009.

[43] T. Back, “Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms,” Oxford university press, 1996.

[44] N. Hansen, “The CMA evolution strategy: a comparing review, Towards a new evolutionary computation,” Springer Berlin, Heidelberg, pp.75-102, 2006.

[45] M. Dorigo, “Optimization, learning and natural algorithms,” Ph. D. Thesis, Politecnico di Milano, Italy, 1992.

[46] 石渕久生 , 半田久志, “進化計算の内外の研究動向,” 進化計算学会論文誌, vol. 1, no. 1, pp. 15-22, 2010.

[47] N. Nedjah , L. d. M. Mourelle, “A Reconfigurable Hardware for Particle Swarm Optimization,” Hardware for Soft Computing and Soft Computing for Hardware, Springer International Publishing, pp. 29-42, 2014.

[48] H. C. Huang, “FPGA-based hybrid GA-PSO algorithm and its application to global path planning for mobile robots,” Przeglad elektrotechniczny, vol. 88, no, 7B, pp. 281-284, 2012.

[49] D. M. Munoz, C. H. Llanos, L. d. S. Coelho , M. Ayala-Rincon, “Hardware Architecture for Particle Swarm Optimization Using Floating-Point Arithmetic,” 2009 Ninth International Conference on Intelligent Systems Design and Applications. IEEE, pp. 243-248, 2009.

[50] S. A. Li, C. C. Hsu, C. C. Wong , C. J. Yu, “Hardware/software co-design for particle swarm optimization algorithm,” Information Sciences, vol. 181, no. 20, pp. 4582-4596, 2011.

[51] A. Farmahini-Farahani, S. Vakili, S. M. Fakhraie, S. Safari , C. Lucas, “Parallel scalable hardware implementation of asynchronous discrete particle swarm,” Engineering Applications of Artificial Intelligence, vol. 23, no. 2, pp. 177-187, 2010.

[52] M. Yoshikawa, “Hardware-oriented ant colony optimization considering intensification and diversification,” INTECH Open Access Publisher, 2008.

[53] C. F. Juang, C. M. Lu, C. Lo , C. Y. Wang, “Ant colony optimization algorithm for fuzzy controller design and its FPGA implementation,” IEEE Trans. on Industrial Electronics, vol. 55, no. 3, pp. 1453-1462, 2008.

[54] D. M. Muñoz, C. H. Llanos, L. D. S. Coelho , M. Ayala-Rincón, “Accelerating the artificial bee colony algorithm by hardware parallel implementations,” In Circuits and Systems (LASCAS), 2012 IEEE Third Latin American Symposium on, pp. 1-4, 2012.

[55] Y. Jewajinda, “Parallel hardware architecture and FPGA implementation of a differential evolution algorithm,” TENCON 2014-2014 IEEE Region 10 Conference. IEEE, pp. 1-4, 2014.

[56] P. R. Fernaldo, S. Katkoori, D. Keymeulen, R. Zebulum , A. Stoica, “Customizable FPGA IPcore implementation of a general-purpose genetic algorithm engine,” IEEE Trans. on Evolutionary Computation, vol. 14, no. 1, pp. 133-149, 2010.

[57] V. P. Nambiar, S. Balakrishnan, M. Khalil-Hani , M. N. Marsono, “HW/SW codesign of reconfigurable hardware-based genetic algorithm in

FPGAs applicable to a variety of problems," Computing, vol. 95, no. 9, pp. 863-896, 2013.

[58] K. Nishijima, A. Kanasugi , K. Ando, "Accuracy improvement of genetic algorithm for obtaining floating-point solution," Artificial Life and Robotics, vol. 19, no. 4, pp. 328-332, 2014.

[59] K. Nishijima, A. Kanasugi , K. Ando, "Accuracy Improvement of Genetic Algorithm for Obtaining Floating-Point Solution," Proc. of the 19th International Symposium on Artificial Life and Robotics (AROB 2014), pp. 850-853, 2014.

[60] L. Guo, B. Thomas D., C. Guo , W. Luk, "Automated Framework for FPGA-Based Parallel Genetic Algorithms," 24th Int. Conf. on Field Programmable, pp. 1-7, 2014.

[61] C. C. Tsai, H. C. Huang , C. Chan, "Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation," IEEE Trans. on Industrial Electronics, vol. 58, no. 10, pp. 4813-4821, 2011.

[62] P. V. dos Santos, J. C. Alves , C. Ferreira J., "A scalable array for cellular genetic algorithms: TSP as case study," Int. Conf. on. Reconfigurable Computing and FPGAs, IEEE, pp. 1-6, 2012.

[63] M. A. Moreno-Armendriz, N. Cruz-Cortés, C. A. Duchanoy, A. Len-Javier , R. Quintero, "Hardware implementation of the elitist compact Genetic Algorithm using Cellular Automata pseudo-random number generator," Computers and Electrical Engineering, vol. 39, no. 4, pp. 1367-1379, 2013.

[64] Y. Jewajinda , P. Chongstitvatana, "A parallel genetic algorithm for adaptive hardware and its application to ECG signal classification," Neural Computing and Applications, vol. 22, no. 7-8, pp. 1609-1626, 2012.

[65] L. Guo, A. I. Funie, D. B. Thomas, H. Fu , W. Luk, "Parallel Genetic Algorithms on Multiple FPGAs," ACM SIGARCH Computer Architecture News, vol. 43, no. 4, pp. 86-93, 2016.

[66] G. Luque , E. Alba, Parallel Genetic Algorithms Theory and RealWorld Applications, Berlin: Springer-Verlag, 2011.

[67] G. R. Harik, F. G. Lobo , D. E. Goldberg, "The compact genetic algorithm," IEEE Trans. on Evolutionary Computation, vol. 3, no. 4, pp. 287-297, 1999.

[68] S. W. Moon , S. G. Kong, “Block-based neural networks,” IEEE Trans. on Neural Networks, vol. 12, no. 2, pp. 307-317, 2001.

[69] 末吉敏則 , 天野英晴(編), リコンフィギャラブルシステム, オーム社, 2005.

[70] A. Kanasugi, A. Tsukahara , K. Ando, “Hardware implementation of evolutionary algorithms using dynamic reconfiguration technology,” Natural Computing, vol. 14, no. 4, pp. 593-601, 2015.

[71] A. Tsukahara , A. Kanasugi, “Genetic Algorithm that can Dynamically Change Number of Individuals and Accuracy,” Proceeding of 2007 International Conference on Frontiers in the Convergence of Bioscience and Information Technologies (FBIT 2007), pp. 785-789, 2007.

[72] A. Kanasugi , A. Tsukahara, “A Processor for Genetic Algorithm using Dynamically Reconfigurable Memory,” Proceeding of 2006 International Conference on Hybrid Information Technology (ICHIT 2006), pp. 310-313, 2006.

[73] 塚原彰彦 , 金杉昭徳, “個体数と精度を動的に変更可能な遺伝的アルゴリズム専用プロセッサ,” 電子情報通信学会技術研究報告, vol. 106, no. 425, ICD2006-157, pp. 79-84, 2006.

[74] 塚原彰彦 , 金杉昭徳, “動的再構成メモリを用いた遺伝的アルゴリズム専用プロセッサ,” 電子情報通信学会技術研究報告, vol. 106, no. 254, VLD2006-39, pp. 1-6, 2006.

[75] 弘中哲夫, “粗粒度リコンフィギュラブルプロセッサの動向,” 電子情報通信学会技術研究報告. CAS, 回路とシステム, vol. 105, no. 502, pp. 19-23, 2006.

[76] 片桐徹 , 天野英晴, “動的再構成プロセッサ MuCCRA-4 の実装,” 電子情報通信学会技術研究報告（リコンフィギャラブルシステム), vol. 113, no. 418, pp. 119-124, 2014.

[77] ルネサステクノロジ（株）, “STP エンジンカタログ,” https://www.renesas.com/ja-jp/doc/DocumentServer/012/r06cp0001jj0100_stpengine.pdf.

[78] K. Hasegawa, A. Kanasugi , K. Ando, “Dynamically Reconfigurable Circuit for Correlation Calculation,” Proc. of the 19th International Symposium on Artificial Life and Robotics (AROB 2014), pp. 892-895, 2014.

[79] T. Sega, A. Kanasugi , K. Ando, “Generator of Dynamically Reconfigurable Processor,” Proc. of the 19th International Symposium on Artificial Life and Robotics (AROB 2014),, pp. 901-904, 2014.

[80] 塚原彰彦 , 金杉昭徳, “実数値 GA 専用プロセッサの一方式,” 電気学会論文誌 C, vol. 136, no. 11, pp. 1586-1595, 2016.

[81] A. Tsukahara , A. Kanasugi, “Design of a Real Coded GA Processor,” Proceedings of the 7th International Joint Conference on Computational Intelligence (IJCCI 2015), pp. 334-339, 2015 年.

[82] 塚原彰彦 , 金杉昭徳, “実数値 GA 専用プロセッサの一設計,” 電子情報通信学会技術研究報告, vol. 115, no. 84, pp. 51-58, 2015.

[83] A. Tsukahara , A. Kanasugi, “A novel architecture of dynamically reconfigurable fused multiply-adder for digital signal processing,” Journal of Artificial Life and Robotics, vol. 19, no. 3, pp. 233-238, 2014.

[84] A. Tsukahara , A. Kanasugi, “A novel architecture of dynamically reconfigurable fused multiply-adder for digital signal processing,” Proceedings of the 19th International Symposium on Artificial Life and Robotics(AROB 2014), pp. 841-844, 2014.

[85] 塚原彰彦 , 金杉昭徳, “動的再構成可能なマイクロプロセッサの研究,” 東京電機大学総合研究所 総合研究所年報 2013, no. 33, pp. 93-96, 2014.

[86] 田中雅晴, 秋本洋平, 佐久間淳, 小野功 , 小林重信, “２段階 GA “SolidEMO” によるレンズ系設計,” 人工知能学会論文誌, vol. 23, no. 3, pp. 193-204, 2009.

[87] 櫛田大輔, 安藤泰正, 北村章 , 深田美香, “RCGA とファジィ推論を用いた看護師の判断モデルの構築:――入院患者のベッド上における転倒危険度の推定――,” 電気学会論文誌 C, vol. 135, no. 5, pp. 498-504, 2015.

[88] 井口圭一, 木村元 , 小林重信, “GA による並列二重倒立振子の振り上げ安定化制御,” 計測自動制御学会第 13 回自律分散システムシンポジウム, pp. 277-282, 2001.

[89] 宮前惇, 佐久間淳, 小野功 , 小林重信, “インスタンスベース政策最適化のための実数値 GA と非ホロノミック系制御への適用,” 人工知能学会論文誌, vol. 24, no. 1, pp. 104-115, 2009.

[90] 佐藤浩, 小野功 , 小林重信, “遺伝的アルゴリズムにおける世代交代モデルの提案と評価,” 人工知能学会論文誌, vol. 12, no. 5, pp. 734-744, 1997.

[91] 秋本洋平, 羽佐田理恵, 佐久間淳, 小野功 , 小林重信, “多親を用いた実数値 GA のための世代交代モデル Just Generation Gap (JGG) の提案と評価,” 第 19 回自律分散システムシンポジウム資料, p. 341-346, 2007.

[92] 秋本洋平, 永田裕一, 佐久間淳, 小野功 , 小林重信, “実数値 GA における生存選択モデルとしての MGG と JGG の挙動解析,” 人工知能学会論文誌, vol. 25, no. 2, pp. 281-289, 2010.

[93] J. D. Schaffer , L. Eshelman, “Real Coded Genetic Algorithms and Interval-Schemata,” Foundations of Genetic Algorithms 2, 1993, pp. 187-202.

[94] 小野功, 佐藤浩 , 小林重信, “単峰性正規分布交叉 UNDX を用いた実数値 GA による関数最適化,” 人工知能学会論文誌, vol. 14, no. 6, pp. 1146-1155, 1999.

[95] 樋口隆英, 筒井茂義 , 山村雅幸, “実数値 GA におけるシンプレクス交叉の提案,” 人工知能学会論文誌, vol. 16, no. 1, pp. 147-155, 2011.

[96] 秋山洋平, 永田祐一, 佐久間淳, 小野功 , 小林重信, “適応的実数値交叉 AREX の提案と評価,” 人工知能学会誌, vol. 24, no. 6, pp. 446-458, 2009.

[97] 上村健人, 木下峻一, 永田裕一, 小林重信 , 小野功, “大域的多峰性関数最適化のための実数値 GA の枠組み Big-valley Explorer の提案,” 人工知能学会誌, vol. 4, no. 1, pp. 1-12, 2013.

[98] Xilinx, MicroBlaze Processor Reference Guide, v9.6, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug984-vivado-microblaze-ref.pdf, 2016.

[99] J. J. Liang, B. Y. Qu, P. N. Suganthan , Q. Chen, “Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective opti-mization,” Nanyang Technological University, Singapore, 2014.

[100] R. Arora, R. Tulshyan , K. Deb, “Parallelization of Binary and Real-Coded Genetic Algorithms on GPU using CUDA,” Int. Cong. on Evolutionary Computation (CEC), pp. 1-8, 2010.

[101] D. K. , B. A. R., “Simulated binary crossover for continuous search space,” Complex Systems, vol. 9, p. 1–34, 1994.

[102] J. Choi , R. A. Rutenbar, “Video-rate stereo matching using Markov random field TRW-S inference on a hybrid CPU+ FPGA computing platform,” IEEE Trans. on Circuits and Systems for Video Technology, vol. 26, no.2 , pp. 385-398, 2016.

[103] 塚原彰彦 , 金杉昭徳, “実数値 GA 専用プロセッサを用いた FIR フィルタの最適設計,” 電子情報通信学会技術研究報告, vol. 116, no. 415, CPSY 2016-107, pp. 7-12, 2017.

[104] A. Tsukahara , A. Kanasugi, “Design of Linear Phase FIR Filter using Real Coded Genetic Algorithm Processor based on FPGA,” 2017 RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP'17). 信号処理学会, pp. 105-108, 2017.

[105] 阿部正英, 対馬尚之 , 川又政征, “進化論的ディジタルフィルタの ASIC 上の並列実現,” 電子情報通信学会論文誌. A, 基礎・境界, vol. 86, no. 4, pp. 383-392, 2003.

[106] D. Mandal, K. Rajib , S. P. Ghoshal, “Digital FIR filter design using fitness based hybrid adaptive differential evolution with particle swarm optimization,” Natural Computing, vol. 13, no. 1, pp. 55-64, 2014.

[107] J. Dash, S. Rajkishore , D. Bivas, “Design of Linear Phase Band Stop Filter Using Fusion Based DEPSO Algorithm,” Computational Intelligence in Data Mining–Volume 1, Springer India, pp. 273-281, 2016.

[108] R. Kar, D. Mandal, S. Mondal , S. P. Ghoshal, “Craziness based particle swarm optimization algorithm for FIR band stop filter design,” Swarm and Evolutionary Computation, pp. 58-64, 2012.

[109] S. Sharma, L. D. Arya , S. Katiyal, “Design of linear-phase digital FIR filter using differential evolution optimization with ripple constraint,” Computing for Sustainable Global Development (INDIACom), 2014 International Conference on. IEEE, pp. 474-480, 2014.

[110] 伊藤登, “3 チャンネル可変フィルタバンクの最適設計とディジタル補聴器への応用,” 電気通信普及財団研究調査報告書, no.24, pp. 315-323, 2009.

[111] T. B. Deng, “Three-channel variable filter-bank for digital hearing aids,” IET signal processing, vol. 4, no. 2, pp. 181-196, 2010.

[112] Y. Lian , Y. Wei, “A computationally efficient nonuniform FIR digital filter bank for hearing aids,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 52, no. 12, pp. 2754-2762, 2005.

[113] P. Srisangngam, S. Chivapreecha , K. Dejhan, “A design of IIR based digital hearing aids using genetic algorithm,” Int. Conf. Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, pp. 967-970, 2011.

[114] J. A. Nelder , R. Mead, “A simplex method for function minimization,” The computer journal, vol. 7, no. 4, pp. 308-313, 1965.

[115] “中度難聴（感音性難聴）WebSite 【 静かの森 】,” [オンライン]. Available: http://home.att.ne.jp/grape/take3/nanchou/002.html. [アクセス日: 30 10 2016].

本研究に関する発表文献

本研究に関する学術雑誌論文

[1] 塚原 彰彦, 金杉 昭徳, “実数値 GA 専用プロセッサの一方式”, 電気学会論文誌 C, vol. 136, no. 11, pp. 1586-1595, 2016 年 11 月

[2] A. Kanasugi, A. Tsukahara and K. Ando, "Hardware implementation of evolutionary algorithms using dynamic reconfiguration technology", Natural Computing, vol. 14, Issue. 4, pp. 593-601, 2015 年 12 月

[3] A. Tsukahara and A. Kanasugi, “A novel architecture of dynamically reconfigurable fused multiply-adder for digital signal processing”, Journal of Artificial Life and Robotics, vol. 19, Issue 3, pp. 233-238, 2014 年 11 月

[4] A. Tsukahara and A. Kanasugi, “Genetic Algorithm with Dynamic Variable Number of Individuals and Accuracy”, International Journal of Control, Automation, and Systems, vol. 7, no.1, pp.1-6, 2009 年 2 月

[5] A. Kanasugi and A. Tsukahara, “A Processor for Genetic Algorithm using Dynamically Reconfigurable Memory”, Journal of Convergence Information Technology, vol. 2, no.1, pp.4-15, 2007 年 3 月

本研究に関する国際会議論文

[1] A. Tsukahara and A. Kanasugi, “Design of Linear Phase FIR Filter using Real Coded Genetic Algorithm Processor based on FPGA”, 2017 RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP'17). 信号処理学会, pp. 105-108, 2017 年 3 月

[2] A. Tsukahara and A. Kanasugi, “Design of a Real Coded GA Processor”, Proceedings of the 7th International Joint Conference on Computational Intelligence (IJCCI 2015), pp.334-339, 2015 年 11 月

[3] A. Tsukahara and A. Kanasugi, “A novel architecture of dynamically reconfigurable fused multiply-adder for digital signal processing”, Proceedings of the 19th International Symposium on Artificial Life and Robotics (AROB 2014), pp. 841-844, 2014 年 1 月

[4] A. Tsukahara and A. Kanasugi, “Genetic Algorithm that can Dynamically Change Number of Individuals and Accuracy”, Proceeding of 2007 International Conference on Frontiers in the Convergence of Bioscience and Information Technologies (FBIT 2007), pp. 785-789, 2007 年 10 月

[5] A. Kanasugi and A. Tsukahara, “A Processor for Genetic Algorithm using Dynamically Reconfigurable Memory”, Proceeding of 2006 International Conference on Hybrid Information Technology (ICHIT 2006), pp.310-313, 2006 年 11 月

本研究に関する学会研究会資料等

[1] 塚原 彰彦, 金杉 昭徳, “実数値 GA 専用プロセッサを用いた FIR フィルタの最適設計”, 電子情報通信学会技術研究報告, vol. 116, no. 415, CPSY 2016-107, pp.7-12, 2017 年 1 月

[2] 塚原 彰彦, 金杉 昭徳, “実数値 GA 専用プロセッサの一設計”, 電子情報通信学会技術研究報告, vol.115, no.84, COMP2015-8, pp.51-58, 2015 年 6 月

[3] 塚原 彰彦, 金杉 昭徳, “動的再構成可能なマイクロプロセッサの研究”, 東京電機大学総合研究所 総合研究所年報 2013, no. 33, pp. 93-96, 2014 年 7 月

[4] 塚原 彰彦, 金杉 昭徳, “個体数と精度を動的に変更可能な遺伝的アルゴリズム専用プロセッサ”, 電子情報通信学会技術研究報告, vol. 106, no. 425, ICD2006-157, pp. 79-84, 2006 年 12 月

[5] 塚原 彰彦, 金杉 昭徳, “動的再構成メモリを用いた遺伝的アルゴリズム専用プロセッサ”, 電子情報通信学会技術研究報告, vol. 106, no. 254, VLD2006-39, pp. 1-6, 2006 年 9 月