한국 퍼지 및 지능시스템 학회 2002, Vol. 12, No. 5, pp. 462-466

진화하드웨어를 위한 유전자 알고리즘 프로세서(GAP) 설계

Design of Genetic Algorithm Processor(GAP) for Evolvable Hardware

심귀보 · 김태훈
Kwee-Bo Sim and Tae-Hoon Kim

중앙대학교 전자전기공학부

요 약

GA(Genetic Algorithm)는 자연계 진화를 모방한 계산 알고리즘으로서 단순하고 응용이 쉽기 때문에 여러 분야에 전역적 최적해 탐색에 많이 사용되고 있다. 최근에는 하드웨어를 구성하는 방법의 하나로서 사용되어 진화하드웨어라는 분야를 탄생시켰다. 이와 함께 GA의 연산자체를 하드웨어로 구현하는 GA processor(GAP)의 필요성도 증가하고 있다. 특히 진화하드웨어를 소프트웨어에서 진화 시키는 것이 아닌 GAP에 의해 진화 시키는 것은 독립된 구조의 진정한 EHW 설계에 필수적이 될 것이다. 본 논문에서는 진화하드웨어의 빠른 재구성을 위한 유전자 알고리즘 프로세서를 설계한다.

Abstract

Genetic Algorithm (GA) which imitates the process of nature evolution is applied to various fields because it is simple to theory and easy to application. Recently applying GA to hardware, it is to proceed the research of Evolvable Hardware(EHW) developing the structure of hardware and reconstructing it. And it is growing a necessity of GAP that embodies the computation of GA to the hardware. Evolving by GA don't act in the software but in the hardware(GAP) will be necessary for the design of independent EHW. This paper shows the design GAP for fast reconfiguration of EHW.

Key Words : 유전알고리즘(GA), 유전자 알고리즘 프로세서(GAP) 설계, FPGA, 진화하드웨어(EHW)

1. 서 론

자연계의 각종 생물은 진화의 과정에서 환경에 적응하도록 그 형태나 구조를 진화시킨다. 진화연산은 자연계의 진화과정을 모방하여, 그 구조를 나타내는 코드나 구조를 변경하여 새로운 기능을 가지도록 하는 것이다. 이것은 소프트웨어의 진화만이 아니라, 하드웨어를 진화시키는 것도 자율성, 창조성이 풍부한 정보처리시스템의 창출에 있어서 중요한 연구과제이다. 소프트웨어 진화와 유사하게 표현하면, 하드웨어 진화는 환경의 변화나 오차를 이용하여 전자회로인 하드웨어가 그 구조를 자율적으로 바꾸어 그 구조는 물론 기능까지 복잡화하고 다양화하는 것이다[1]. 따라서 그 표현형이 프로그램이 아니라 전자회로인 점을 제외하면, 방법론적으로는 소프트웨어 진화와 기본적으로 차이는 없다. 단 회로소자 그 자체는 진화하지 않지만, 예를 들어 CPLD(Complex Programmable Logic Device)나 FPGA (Field Programmable Gate Array) 등 소자의 결선이나 조합이 재구성 가능한 구조를 가진 하드웨어 디바이스를 전제로 한다.

이와 관련된 연구 중에는 재구성 가능한 디바이스로서 셀룰라 오토마타 머신(CAM: Cellular Automata Machine)을 이용하여 뇌 신경계의 기본구조 모델인 신경망을 하드웨어로서 발생, 성장, 진화시키고자 하는 연구와 미지의 혹은 끊임없이 변동하는 환경에 자율적으로 적응하여 목적의 기능을 획득하는 유연한 하드웨어의 구축을 목표로 하드웨어기술언어(HDL: Hardware Description Language)를 이용한 하드웨어 진화시스템의 연구도 있다[2].

하드웨어를 재구성하는 방법에는 Extrinsic method와 intrinsic method가 있다. Extrinsic method는 컴퓨터(소프트웨어로)에서 하드웨어를 진화 최종진화 후에 하드웨어에 download하는 것이고, intrinsic method는 하드웨어를 구성하여 적합도를 평가하고 다시 재구성하는 방법이다. 이러한 방법은 하드웨어를 재구성하기 위해 부피가 큰 데스크탑 컴퓨터가 필요하다. 본 논문에서는 재구성에 필요한 GAP와 EHW를 하나의 칩으로 설계함으로써 개체 생성에 필요한 환경만 설정하면 적합도가 높은 개체를 생성하는 하드웨어 설계방법을 제안한다[3].

접수일자 : 2002년 6월 10일
완료일자 : 2002년 9월 19일
이 논문은 2002년도 중앙대학교 학술연구비 지원에 의한 것입니다. 연구비 지원에 감사드립니다.

2. 유전자 알고리즘

유전자 알고리즘은 어떤 세대(generation)를 형성하고 있

는 개체(individual)의 집합 가운데 환경에 대한 적응도(fitness)가 높은 개체가 높은 확률로 살아남도록 재생되도록 하고 교차(crossover)나 돌연변이(mutation)에 의해서 다음 세대의 개체군이 형성되어 가도록 하는 것이다[4].

이와 같은 유전자 알고리즘이 지금까지의 고전적인 탐색법과 크게 다른 점은 Goldberg에 의하면 다음과 같은 네 가지의 특징으로 요약할 수 있다[3].

(1) 파라미터를 코딩한 것을 직접 이용한다.
(2) 한 점 탐색이 아니라 다 점 탐색이다.
(3) 샘플링에 의한 탐색으로 블라인더 탐색이다.
(4) 결정론적인 규칙이 없고 확률적 연산자를 이용하는 탐색이다.

기본적인 유전 알고리즘 및 pseudo code는 다음과 같다.

```
procedure GA()
    initialize(Population);
    evaluate(Population);
    while not (terminal condition satisfied) do
        MatingPool = reproduce(Population);
        MutationPool = crossover(MatingPool);
        Population = mutation(MutationPool);
        evaluate(Population);
    end while
end procedure
```

하지만 FPGA로 구현하기 위해서는 위와 같은 순차적인 code가 아니라 클럭에 의한 concurrent한 동작을 확인해야 한다.

이 알고리즘을 하드웨어에 적용하기 위해서는 생성된 개체의 bit string에 따라 변형될 수 있는 하드웨어가 필요하다. 진화하드웨어 연구에 주로 사용되는 것은 Xinlinx의 XC6200 시리즈이다[4]. 이 칩의 구조는 그림 2와 같이cell이 상하좌우로 연결되어 있다[5].

3. FPGA의 특성

FPGA에 회로를 설계하기 위해서 스키메틱 편집기(schematic editor)나 HDL(Hardware Description Language)를 이용하는데 스키메틱은 각 제품이나 툴마다 다른 포맷 형식을 가지기 때문에 제품이 바뀌면 완전히 새로 회로를 구성해야만 한다. 하지만 HDL (Hardware Description Language)은 언어이기 때문에 모든 제품에 사용될 수 있어 호환성이 높다.

FPGA는 사용하는 것은 이처럼 하드웨어 구조를 소프트웨어적으로 변경할 수 있는 유연성(flexibility)이 있어 ASIC에 많이 사용한다.. FPGA에 프로그래밍 하는 것은 HDL로 프로그래밍을 한 다음, 이것을 실제 논리회로로 구성하는데 이 과정을 synthesis라고 하다. 이렇게 synthesis 된 회로로 FPGA를 구성하기 위해서는 FPGA의 구성물에 맞도록 다시 변경해야 하는데 이 과정을 Place & Route라 한다. 이 과정을 통하여 각 회로들은 위치를 잡게 되는데, 이 위치가 어디냐에 따라 속도에 현저한 차이가 날 수도 있다. 이 문제는 프로그래밍 하는 툴에서 최적화시켜 주기 때문에 큰 문제가 없다. 하지만 진화 알고리즘을 사용할 경우에는 진화를 통하여 회로의 위치나 구조 등을 최적화한다.

이 논문에서 사용한 FPGA는 VertexE로서 내부는 CLB와 IOB, switch block으로 이루어져있다. 그림 1은 CLB의 구조를 보여준다. LUT(Look-up Table)과 FF (Flip-Flop)를 사용하여 논리회로를 구성한다.

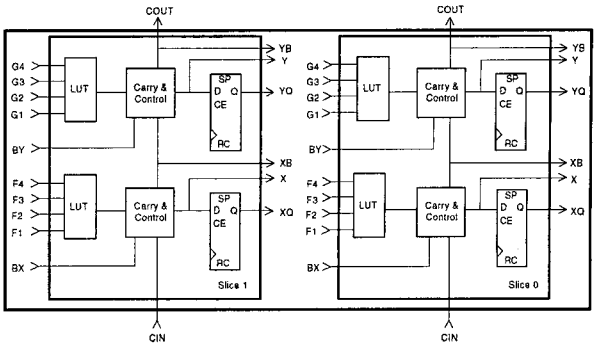


그림 1. CLB 구조
Fig. 1. Structure of CLB

4. 진화하드웨어(EHW)의 구조

일반적으로 조합회로를 설계할 때는 순차적으로 진행되지만 state-machine은 피드백이 필요하기 때문에 단순히 순차적 구조로는 구성할 수 없기 때문에 피드백이 가능한 구조를 만들어야 한다.

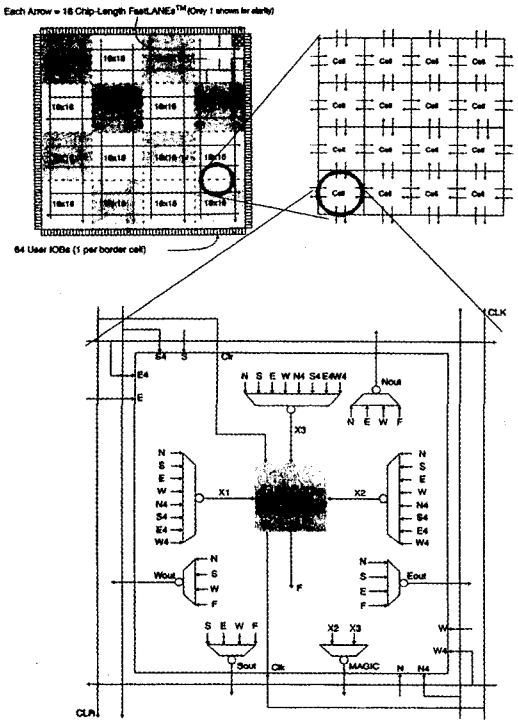


그림 2. XC6200시리즈의 구조
Fig. 2. Structure of XC6200 series

그림 2에서 보는 바와 같이 XC6200의 구조는 셀 단위로 형성되어 각 셀은 상하좌우로 연결되어 있다. 이렇게 연결된

경우 연결된 셀이 다시 다른 셀에 연결이 되어 다른 셀의 출력이 자신의 입력이 된다. 즉 출력이 다른 셀을 통하여 출력셀의 입력이 되어 피드백이 된다.

Intrinsic method를 사용한 진화하드웨어연구는 하드웨어의 구조를 결정하는 코드를 소프트웨어로 만든 후 하드웨어에 적용한다. 이러한 방식으로 설계가 가능한 FPGA가 XC6200시리즈이다. XC6200시리즈는 intrinsic method를 사용하기 때문에 외부에서 재구성하는 장치가 필요하다.

FPGA가 발달함에 따라 FPGA에 수용할 수 있는 논리회로의 수는 증가하고 이를 configuration할 수 있는 bit수는 증가하게 되었고 소프트웨어로 진화시키기에는 한계가 있다. 또한 하드웨어를 진화시키기 위해 다른 장치가 필요하기 때문에 본 논문에서는 GAP를 이용하여 하드웨어가 직접 코드를 생성하고 평가하여 진화를 시킨다. 이 논문에서 사용한 FPGA는 VertexE(모델명: XCV2000E)이다. 이 FPGA의 cell은 CLB (Configurable Logic Block)으로 구성되어 있다. 이는 XC6200과는 달리 LUT(Look Up Table)과 플립플롭으로 구성되어 있기 때문에 논리 게이트의 구성방법이 다르고 각cell을 게이트 단위의 조작이 힘들다.

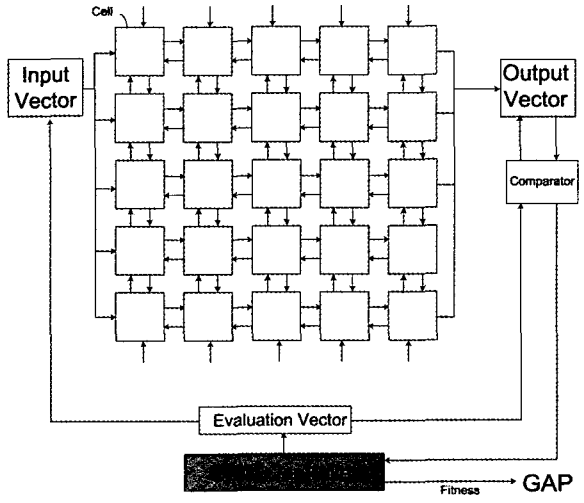


그림 3. EHW 구조
Fig. 3. Structure of EHW

XC6200시리즈와 비슷한 기능을 하기 위해서 XC6200시리즈의 cell과 같은 기능을 하는 component를 작성한다. function level의 cell을 HDL을 사용하여 연결한다. 그림 3은 이와 같은 구조를 나타낸 것이다. 각 셀들은 상하좌우의 다른 셀들과 연결되어 있고, 그 셀들은 다시 다른 셀들과 연결이 되어 있어 각 셀의 기능에 따라 논리회로를 구성하게 된다.

유전자 알고리즘을 사용할 때는 적합도 평가가 중요하다. 하드웨어가 진화하기 위해서는 적합도를 평가하는 것이 하드웨어(GAP)든지 소프트웨어든지 적합도를 유전자 알고리즘을 수행하는 곳에 되돌려 줘야 한다. 이를 위해서 진화하드웨어 내에서 적합도를 구할 수 있도록 다른 모듈을 덧붙였다. 적합도를 구하기 위한 평가벡터를 메모리에 저장하고, 각 개체가 발생 혹은 재생될 때마다 이를 진화하드웨어에 입력하고 출력을 비교하여 적합도를 구한다. 이를 위해서는 GAP와 EHW사이의 데이터전송을 위한 동기화가 필요하므로 EHW의 동작, 평가벡터 저장, 하드웨어재구성, 적합도 평가는 GAP의 컨트롤에 의해 이루어진다.

EHW의 component로 구성한 각 cell은 그림 2와 같이 네 개의 입력을 네 개의 방향으로 각각 다른 회로를 가진 출력을 구성한다. 이 논리회로의 역할은 AND, OR, NOT, BUFFER이다. 이 기능을 결정하기 위해 2bit씩 사용되는데, 상하좌우에 2bit, 한 cell에 8bit가 필요하다. 전체에 필요한 bit는 cell의 개수에 따라 달라진다. 이 논문에서는 회로를 구성할 수 있는 cell의 개수는 36개이기 때문에 총 288bit가 필요하다. 회로가 복잡해지면 더욱 많은 게이트가 필요하기 때문에 더욱 많은 셀이 필요하다.

5. 유전자 알고리즘 프로세서(GAP) 설계

5.1 GAP의 구조

유전자 알고리즘을 하드웨어로 구현하기 위해서 필요한 것이 생성된 bit string을 발생, 선택, 교차, 돌연변이 등을 시키는 프로세서가 필요하다.

그림 3의 진화하드웨어를 재구성하는데 필요한 bit는 288bit이다. 그림 4의 각 버스를 288bit로 사용하는 것은 많은 리소스를 낭비하고 또한 이런 방법으로는 HDL을 synthesis 할 수 없을 만큼 무리가 있다.. 그래서 각 버스는 32bit로 하고 이를 9번 반복하여 수행함으로써 288bit를 생성하고, 교차 돌연변이 시킨다. 이와 같은 구조는 다른 하드웨어가 적용되더라도 bit수를 변경시킬 수 있기 때문에 진화하드웨어가 아니라 다른 하드웨어에 적용시키기 좋은 구조이다.

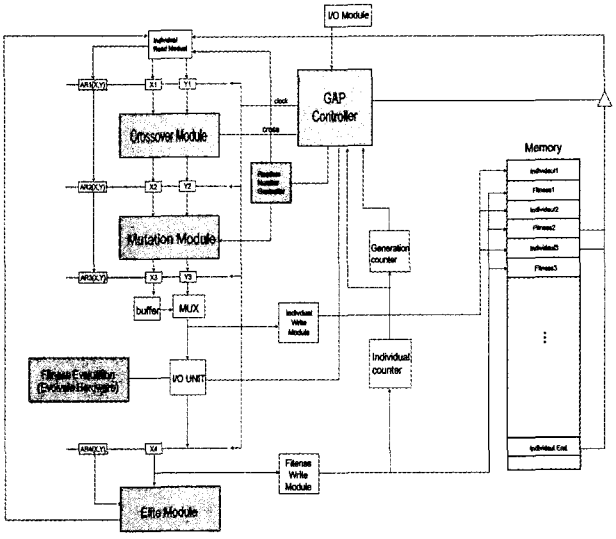


그림 4. GAP 구조
Fig. 4. Structure of GAP

교차는 32bit일 경우 2점 교차를 행하고 32bit 이상일 때는 32bit 단위로 교차를 진행할 수 있어서 여러 종류의 다점 교차를 수행한다. 돌연변이의 경우 각 bit에 대하여 돌연변이율을 적용할 경우 전체 평가에 있어 문제가 발생하므로 난수를 통하여 돌연변이의 위치를 결정한 후 돌연변이를 적용한다. 이 경우 일반적인 유전 알고리즘보다 돌연변이가 적게 나타나게 때문에 돌연 변이 율을 높여야 한다.

적합도 평가(Fitness evaluation)부분은 앞에서 설명한 진

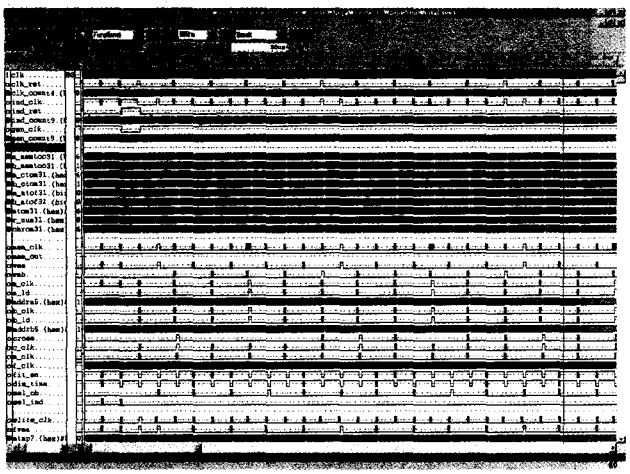


그림 5. GAP 동작 파형
Fig. 5 GAP performance waveform

화하드웨어가 GAP의 한 부분으로 들어가서 적합도를 평가한다. EHW의 bit수와 마찬가지로 적합도 평가에 걸리는 시간은 어떤 적합도 함수이냐에 따라 평가하는데 걸리는 시간이 다르므로 적합도 평가를 하는 동안은 프로세서의 동작을 멈추고 평가가 끝나도록 대기하다가 적합도 평가가 끝나면 다시 다른 과정을 수행한다. 이를 위해 I/O 모듈이 필요하다.

엘리트 모듈은 이렇게 발생한 개체의 재생에 있어 영향을 준다. 적합도가 높은 개체의 재생확률을 높임으로써 수렴을 빨리 할 수 있도록 했다. 개체가 288bit이므로 이를 저장하기 위해서는 많은 메모리가 필요하므로 메모리 주소와 적합도만을 저장하여 메모리의 크기를 줄였다.

그림 5는 이와 같은 과정이 EHW에서 동작하는 것을 logic simulator로 확인한 것이다. 같은 세대에서는 일련의 동작을 반복하며 재생시킨다.

GAP를 사용할 때 항상 필요한 RNG (Random Number Generator)는 random number, Ri+1을 발생시키는 과정은 다음 (1)을 이용했다.

$$R_{i+1} = (A \times R_i + B)\%M \qquad (1)$$

위의 (1)식에서 A, B, M은 임의의 정수를 대입하는데 특정한 법칙은 없다. 초기 R값에 따라 그 수는 달라지므로 초기값(seed)을 정하는 것이 중요한데 이것은 clock을 count하여 정하였다. 나머지(remainder)연산은 수행함에 있어서 매우 복잡한 연산이므로 곱셈 후 64bit를 32bit로 줄임으로써 232로 나머지 연산을 하도록 하였다.

개체의 저장 역시 32bit 버스에서 사용하기 위해서는9개의 clock이 필요하므로 EHW에 configuration 하는 동시에 메모리에 저장하여 저장을 위한 clock을 줄인다.

개체와 적합도를 저장하기 위한 메모리는 VertexE에서 제공하는 Block Memory를 사용하였다. 적합도 평가를 위한 input data는 유전자 알고리즘을 사용하기 위한 최소한의 환경과 적합도를 평가하기 위한 평가벡터를 입력한다. 이후의 진행과정은 GAP Controller를 사용하여 일괄적으로 처리하도록 했고, 종료조건은 세대 혹은 적합도에 따라 유전자 알고리즘을 수행한다.

5.2 GAP를 이용한 State machine

State machine이란 각각의 State들을 어떤 조건에 따라 연결해 놓은 것이다. 이 State machine 중에서도 유한한 개수의 상태를 가진 것을 FSM(Finite State Machine)이라고 한다. FSM은 하나의 입력(Input)을 받고 그에 의거해서 현재 상태(Current State)로부터 다음 상태(Next State)로 전이(transition)하는 식으로 동작한다. 외부 입력과 시스템 클럭에 의해 state(상태)가 천이 되고, State에 의존되어 출력값이 결정된다.

State machine을 설계하기 위해서는 다음과 같은 과정을 거친다.

State machine을 설계하기 위해서는 그림 6과 같은 과정을 거친다[7].

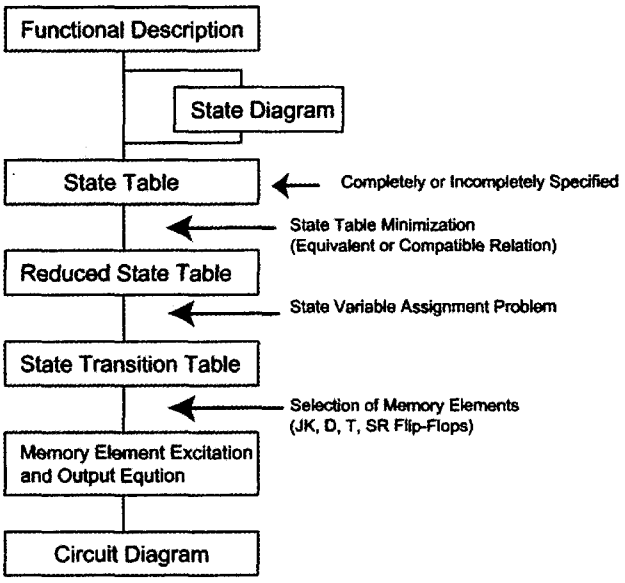


그림 6. State machine의 설계 과정
Fig. 6. Design Process of State machine

우선 state machine이 어떤 동작을 하는지 기술해야 한다. EHW의 평가 벡터도 이 동작에 따라 순서에 맞는 연속적인 데이터로 작성한다. 그 후 state diagram이나 state table을 작성하여 state를 특성화시키고 최소화 화여 state에 사용되는 code를 결정하고, 같은 동작을 하는 state를 확인하여 최소화한다. 그 후 state transition table를 작성하여 state에 코드를 할당하여 Race problem와 같이 이 후 발생하는 문제들을 해결한다. 이렇게 할당된 state는 각각의 bit가 카르노 프맵을 통하여 Circuit diagram으로 변환하여 회로를 구성하게 된다. 이때 주의 할 점은 state machine는 state에 따라 출력이 달라지므로 피드백을 설정하여 회로를 구성해야 한다.

6. 시뮬레이션 결과

본 논문에서 GAP를 이용하여 수행한 실험은 선형방정식의 해를 찾는 것이다. 결과 값은 엘리트 개체를 통하여 얼마나 진화하는가를 나타내었다. 교차율은 0.8, 돌연변이 율은 0.3이나 한 비트에만 적용이 되기 때문에 실제 돌연변이 율은 0.01로 적용된다. 개체 수는 50이고, 클럭 주파수는 50Mhz이지만 클럭싸이클이 적기 때문에 70세대를 수행하는데 1초가 걸린다.

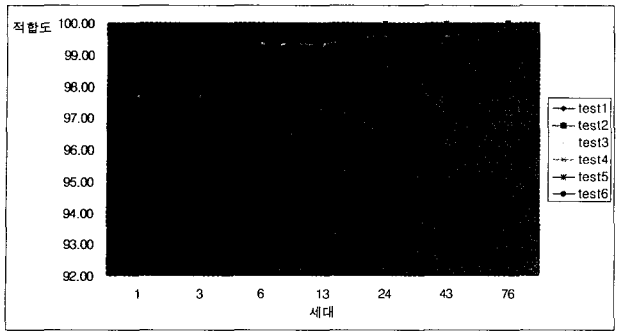


그림 7. 각 테스트의 엘리트의 평균 적합도
Fig. 7. Elite Average fitness of each test

그림 7은 엘리트가 얼마나 진화하는 가를 100으로 정규화 시켜서 표현하였다. 간단한 선형방정식의 경우 초기에 높은 적합도를 나타내어 20세대에서 해가 수렴하는 모습을 보여준다. 이후 세대에서는 엘리트의 적합도는 크게 변화가 없지만 평균 적합도는 올라가고 있다.

위의 데이터를 하드웨어에 적용하였을 경우 데이터를 확인하기 위해서는 각 핀의 신호를 샘플링 하여 데이터를 확인한다. 이 경우 하드웨어에서 동작을 확인 할 수 있지만 진화가 진행되는 과정을 확인하기 어렵다. 진화의 과정을 정확히 확인하기 위해서는 인터페이스를 사용하여 확인하여야 한다.

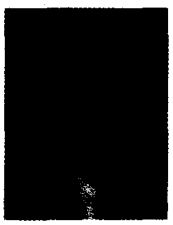
7. 결 론

현재의 인간은 많은 기계를 사용하고 있다. 미래사회에서 구현하기 어려운 문제에 봉착했을 때 이를 기계 스스로가 진화하여 문제를 해결한다면 사람 역시 많은 발전을 할 수 있을 것이다. 본 논문에서 주변 환경에 맞춰 진화하는 하드웨어를 구현함으로써 이러한 하드웨어진화의 가능성을 보였다. 특히 이전 연구에서는 EHW는 컴퓨터에서 configuration bit를 download하였으나 이를 GAP와 함께 설계하여, 하나의 칩으로 구현함으로써 부가적인 장비 없이 독자적으로 진화하는 하드웨어를 구현하였다. 특히 피드백회로인 state machine을 설계하여 이후 여러 분야에 적용할 수 있을 것으로 기대된다. 본 논문에서는 인터페이스를 제작하지 못하여, VertexE칩의 input/output를 확인하기 위해서는 컴퓨터를 사용하였으나 이후 연구에서는 인터페이스를 제작하여 독립적인 모듈로 동작할 수 있도록 할 것이다.

참 고 문 헌

[1] Tetsuya Higuchi, Hitoshi Iba, Bernard Manderick, Evolvable Hardware, Massively Parallel Artifical Intelligence, pp. 398-421, MIT Press, 1994.
[2] 반창봉, 곽상영, 이동욱, 심귀보, "FPGA를 이용한 진화형 하드웨어 설계 및 구현에 관한 연구," 한국퍼지 및 지능시스템학회 논문지, Vol. 11, No. 5, pp. 426-431, 2001. 10.
[3] David E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, INC., 1989.
[4] Gordon Hollinworth, Steve Smith, Andy Tyrrell, "The Intrinsic Evolution of Virtex Devices," Evolvable Systems: From biology to Hardware, ICES2000, pp. 72-79, Springer, 2000
[5] Adrian Thompson, Hardware Evolution, Springer, 1998.
[6] Jin Jung Kim, Daek Jin Chung, "Implementation of genetic algorithm based on hardware optimization," TENCON 99, Proceedings of the IEEE Region 10 Conference, Vol. 2, pp. 1490-1493, 1999.
[7] Ziv Kohavi, Switching and Finete Automata Theory, McGraw-Hill Inc., 1978.

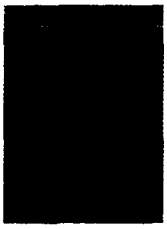
저 자 소 개



심귀보(Kwee-Bo Sim)
1984년 : 중앙대학교 전자공학과 공학사
1986년 : 동 대학원 전자공학과 공학석사
1990년 : The University of Tokyo 전자공학과 공학박사
2000년~현재 한국퍼지 및 지능시스템학회 편집이사 및 논문지 편집위원장
2001년~현재 대한전기학회 제어 및 시스템부문회 편집위원 및 학술이사
2002년~현재 제어·자동화·시스템공학회 이사
1991년~현재 중앙대학교 전자전기공학부 교수

관심분야 : 인공생명, 진화연산, 지능로봇시스템, 뉴로-퍼지 및 소프트 컴퓨팅, 자율분산시스템, 로봇비전, 진화하드웨어, 인공면역계 등

Phone : +82-2-820-5319
Fax : +82-2-817-0553
E-mail : kbsim@cau.ac.kr



김태훈(Tae-Hoon Kim)
2002년 : 중앙대학교 전자전기공학부 공학사
2002년~현재 중앙대학교 전자전기공학부 석사과정

관심분야 : 인공생명, 진화연산, 다개체 시스템, 진화 하드웨어 등

Phone : +82-2-820-5319
Fax : +82-2-817-0553
E-mail : Jonathan@jupiter.cie.cau.ac.kr