논문 2008-45IE-4-5

진화하드웨어 구현을 위한 유전알고리즘 설계

(Hardware Implementation of Genetic Algorithm for Evolvable Hardware)

동 성 수*, 이 종 호**

(Sung Soo Dong and Chong Ho Lee)

요 약

본 논문은 진화 하드웨어 시스템에 적용하기 위해서 유전알고리즘을 하드웨어 기술언어를 사용하여 구현하였다. 진화 하드웨어는 응용에 따라 동작되어지는 환경에 적응하여 동적이면서 자동적으로 자기의 구조를 바꿀 수 있는 능력을 가진 하드웨어를 의미한다. 따라서 정확한 하드웨어 사양이 주어지지 않는 응용에 있어서도 동작을 수행할 수 있다. 진화 하드웨어는 재구성 가능한 하드웨어 부분과 유전알고리즘과 같은 진화 연산을 하는 부분으로 구성 되어 있다. 유전알고리즘을 소프트웨어로 구현하는 것 보다 실시간 응용 부분 등에 있어서 하드웨어로 유전알고리즘을 구현하는 것이 유리하다. 하드웨어로 처리하는 것이 병렬성, 파이프라인 처리, 그리고 함수 사용 부분 등에 있어 소프트웨어의 단점을 보완하여 속도 면에서 이득이 있기 때문이다. 논문에서는 진화 하드웨어를 임베디드 시스템으로 구현하기 위하여 유전알고리즘을 하드웨어로 구현하였고, 몇 가지 예제에 대하여 검증을 수행하였다.

Abstract

This paper presents the implementation of simple genetic algorithm using hardware description language for evolvable hardware embedded system. Evolvable hardware refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment. So, it is especially suited to applications where no hardware specifications can be given in advance. Evolvable hardware is based on the idea of combining reconfigurable hardware device with evolutionary computation, such as genetic algorithm. Because of parallel, no function call overhead and pipelining, a hardware genetic algorithm give speedup over a software genetic algorithm. This paper suggests the hardware genetic algorithm for evolvable embedded system chip. That includes simulation results for several fitness functions.

Keywords: evolvable hardware, genetic algorithm, embedded system, reconfigurable hardware

I. 서 론

기존의 하드웨어, 반도체 칩은 사용 목적에 따라 설계된 후에는 그 구조가 고정적인 상태에서 동작을 수행한다. 그러나 이와는 대조적으로 진화(적응)하드웨어(evolvable hardware; EHW)는 환경에 따라 자체 구조를 동적이고 자율적으로 변경시킬 수 있는 능력을 가지고 있다. 따라서 동작환경이 바뀌더라도 환경에 적응하여 자체 하드웨어를 변경시켜 동작을 수행할 수 있는 유용한 하드웨어이다[1].

EHW는 재구성이 가능한 하드웨어 장치와 자율적으로 재구성을 시킬 수 있는 알고리즘, 주로 유전알고리즘(genetic algorithm; GA)으로 이루어져 있다. 재구성 하드웨어 장치는 소프트웨어로 구성된 이진수 비트열(bit string)을 장치내로 다운로드 받아서 내부구성을 변경 시킬 수 있는 구조로 이다. GA는 집단 내에서 유전적 진화적응을 근간으로 하는 탐색 알고리즘으로 광범위한 탐색 공간에서 해답을 찾는데 효율적이다[2]. EHW를 좀 더 복잡한 응용에 사용하면 GA 구조상 집단의 크기와 세대수가 증가한다. 소프트웨어로 구현된 GA는 그에 따른 시간의 지연이 매우 커서 효율성이 떨어진다. 따라서 GA 연산을 병렬처리 및 파이프라인처

* 정회원, 용인송담대학 정보미디어학부
(School of Information and Media, YonginSongdam College)
** 정회원, 인하대학교 정보통신공학부
(Dept. of Information Technology & Telecommunication, Inha University)
접수일자: 2008년8월29일, 수정완료일: 2008년12월1일

리를 하여야 효율성을 증가시킬 수 있는데 이는 하드웨어로 구현하여 해결 가능하다.

본 논문은 EHW를 임베디드 시스템으로 구현하기 위하여, 먼저 GA를 하드웨어로 구현하였고 처리속도에 이득이 있음을 보였다.

Ⅱ. 본 론

1. 진화하드웨어

하드웨어 설계에 있어서 초기에는 하나의 반도체 칩에 집적할 수 있는 회로의 용량에 제약을 많이 받았다. 그러나 반도체 기술의 발전으로 인하여 한 칩의 하드웨어 디바이스의 용량은 크게 증가하였다. 따라서 최근의 회로설계는 빠르게 변하는 설계환경에서의 설계능력에 대한 제약이 더 커졌다. 이에 대한 해결책으로 자동 설계 방법들이 나오고 있는데, 그 중 하나가 EHW 이다[3]. 회로를 일일이 설계하는 대신에 입력과 출력들의 관계를 설정하고, 자연 진화에 근거한 적응 알고리즘을 사용하여 회로를 자동으로 설계할 수 있다. 이를 그림 1. 에 표현하였다.

그림 1.에서 초기에 무작위로 회로들의 집합인 집단(population)을 생성한다. 이 회로의 집합은 FPGA의 구성 비트(configuration bits)처럼 회로를 구성할 수 있는 내용을 표현한 것이다. 각 회로의 동작은 평가되어지고 최적의 회로들이 결합되어 더 나은 회로들로 생성된다. 평가 기준이 되는 설계 동작은 초기에 사용자에 의해서 정의되어 진다. 이와 같은 동작이 여러 번 되풀이된 후, 초기에 정의된 설계 동작에 가장 적합한 회로가 구해진다.

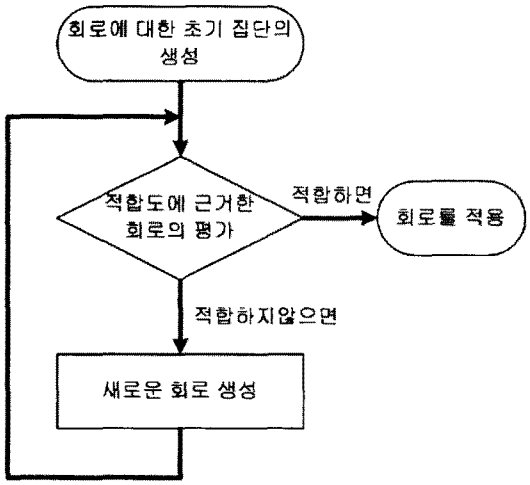

그림 1. 진화적응 회로의 알고리즘
Fig. 1. The algorithm for evolvable circuits.

가. EHW 분류

다음의 EHW 분류[4] 기준으로 GA를 설계하였다.

(1) 진화알고리즘

- 유전알고리즘 (Genetic Algorithm : GA)
- 유전프로그래밍 (Genetic Programming : GP)
- 진화프로그래밍 (Evolutionary Programming : EP)

세 개의 알고리즘 중, 본 논문은 현재 많이 쓰이는 GA를 먼저 대상으로 하여 설계하였다.

(2) 구현하려는 목표 EHW 기술

디지털 (Digital)과 아날로그 (Analog) 기술 중, 회로로 쉽게 구현 및 변경이 용이한 디지털 기술을 대상으로 하였다.

(3) 빌딩 블록

진화 대상이 되는 하드웨어 회로의 기본 단위 레벨인 아날로그 부품 레벨 (Analog component level), 게이트 레벨 (Gate level), 함수 레벨 (Function level)에서, 염색체 길이에 있어서 장점을 가지는[5] 함수 레벨의 진화 대상을 고려하였다.

(4) 대상 하드웨어

진화가 이루어지는 하드웨어 자체를 의미하는데 상용 디바이스 (Commercially available devices)인 FPGA와 주문형반도체 (Application Specific Integrated Circuit: ASIC) 중, 비용 및 적용에 유리한 FPGA를 대상으로 설계 하였다.

본 논문은 EHW 플랫폼을 임베디스 시스템으로 한 칩에 구현하기 위하여, 먼저 진화연산을 특정 하드웨어로 구현한 것이다.

나. EHW 문제점

(1) 염색체 길이

GA는 염색체의 길이가 길어질수록 세대 수는 증가한다. 이는 탐색 영역의 확장과 더불어 탐색시간의 증가를 가져온다. 따라서 적합도 함수계산을 병렬로 하거나, 집단을 세분화하고 병렬로 계산하여 속도를 빠르게 하고 있다[6]. 또 다른 방법 중 하나는 일정 기능을 가진 함수 레벨의 진화 (function level evolution) 이다[5].

(2) 적합도 계산

온라인으로 적합도를 계산하기 위해서는 집단 안에서 서로 다른 구성들 간에 빠른 교체가 이루어져야 한다. FPGA에서 재구성이 가능하기는 하지만, 이는 디바이스 안에서 어느 한 순간에 하나의 구성만이 가능하다. 진화를 위해 새로운 구성으로 재구성하기위해서는 많은 시간이 걸린다. 이를 해결하기 위한 방법 중 하나가 FPGA 내부에 여러 개의 서로 다른 구성이 가능한 레지스터들을 준비하고, 한 클럭 시간 안에 교체시켜 재구성이 가능하게 할 수 있다[7].

2. 하드웨어 엔진 구현

가. 전체블록의 개요

EHW는 프로세서 및 메모리와 진화대상 하드웨어 그리고, GA 엔진으로 구성되어 임베디드 시스템 (embedded system)으로 구현된다. 전체 블록을 그림 2. 에 나타내었다.

그림 2.에서 구현된 GA 엔진에 대한 동작은 다음과 같다. 프로세서 (processor : μP)를 통해 사용자가 정의한 모든 변수들을 메모리 (memory : MEM) 에 저장한다. 그 후, 프로세서는 메모리 제어 모듈 (memory control module : MCM) 에 시작 신호를 보낸다. 시작 신호를 받은 MCM은 각 모듈에 GA 연산을 시작한다는 내용을 알린다. 각 모듈들은 MCM을 통해 필요로 하는 변수들을 MEM으로부터 가져와 초기화를 한다. 염색체 추출모듈 (chromosome extract module : CEM) 은 MCM을 통해 MEM으로부터 염색체 하나를 가져와서 선택 모듈 (selection module : SM) 로 전달하는 작업을 연속적으로 한다. SM은 CEM로부터 새로운 염색체를 받을 때마다 선택할 것인지 아닌지 결정을 한다. 결정 후, 선택된 염색체 A와 B를 교차 및 돌연변이연산

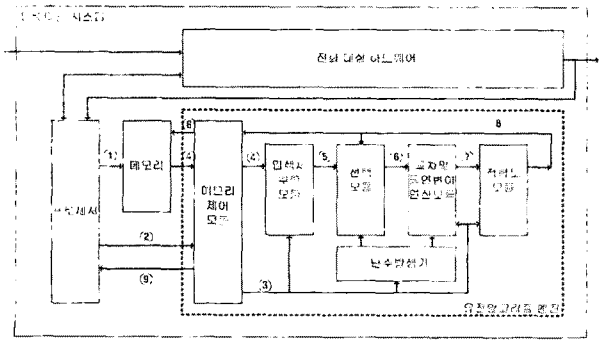

그림 2. 진화적응 하드웨어 플랫폼
Fig. 2. The evolvable hardware platform.

모듈 (crossover and mutation module : CMM)로 보낸다. CMM은 SM으로부터 한 쌍의 염색체 A와 B를 받으면, 난수발생기 (pseudo-random number generator : PRNG) 로부터의 확률 값을 기초로 교차연산 및 돌연변이연산을 수행할 지를 결정한다. 연산을 수행하면 새로운 염색체 A'와 B'가 만들어지고, 이들을 적합도 모듈 (fitness module : FM) 로 보낸다. FM은 CMM에서 받은 A'와 B'를 평가한다. 평가 후에 새로운 염색체들을 MCM을 통해 MEM으로 저장한다. 그리고 현재 상태와 관련된 기록들을 유지한다. 이 기록들은 SM이 새로운 염색체를 선택하는데 필요한 내용과 전체 GA 동작이 끝났는지를 결정하는데 필요한 내용들이다. FM이 GA 동작이 끝났는지 결정할 때까지 위의 과정들이 순환된다. GA동작이 끝나면 MCM에 알리고, MCM는 모든 모듈을 정지시키고 μP 에게 종료 신호를 보내어 종료 시킨다.

나. 세부 블록

(1) 메모리 제어 모듈

메모리제어 모듈은 유전알고리즘 시스템 전체를 관장하는 역할을 하며, 메모리와 나머지 모듈들 사이에 매개 역할을 한다. 프로세서에서 들어오는 시작 신호에 의해 다른 모듈들을 시작 상태로 만들고, 필요한 동작 요청을 받을 상태가 된다. 먼저 적합도 판정이 끝났는지 아닌지를 검사하여 끝났으면 다른 모듈들을 종료시키고, 프로세서에 유전알고리즘 동작이 종료되었음을 알린다. 그렇지 않으면 각 모듈들로부터 요청 받아 필요한 매개변수를 공급시켜주는 동작을 수행한다.

(2) 염색체 추출 모듈

염색체 추출 모듈은 현재 집단 전체를 순환하면서 염색체들을 선택 모듈로 전달하는 역할을 한다. 전체 집단의 크기에 대한 정보를 먼저 가져온 후에, 메모리제어 모듈로 집단의 염색체에 대한 주소를 지표형태로 전달한다. 그리고 염색체를 가져와서 선택 모듈로 전달한다. 염색체를 가져온 후에는 주소를 하나 증가시켜서 다음 염색체를 가져올 준비를 한다. 각 모듈의 동작이 완료되기 전까지는 이 동작을 계속 반복한다.

(3) 선택 모듈

선택 모듈에서 사용된 방법은 SGA에서 사용된 룰렛 휠(roulette wheel) 방법과 유사하게 사용되었다.

단계 1. 랜덤 실수 $r \in [0,1]$ 값을 사용하여, 현시점 집단의 적합도들의 합을 축소시킨 Sscale $= r \cdot S_{fit}$ 을 구한다.

단계 2. 집단의 첫 번째 염색체부터 시작하여, 각 염색체들을 순서대로 검사한다.

단계 3. 새로운 염색체를 검사할 때마다 그 염색체의 적합도들의 합을 누적한 S_R 을 구한다. 그 때 $S_R \geq$ Sscale 이면, 검사된 그 염색체를 선택한다. 그렇지 않으면 그 다음 염색체로 넘어가서 단계 2부터 반복한다.

새로운 염색체가 선택되어질 때마다 위의 과정이 반복된다.

(4) 교배 및 돌연변이 모듈

교배확률과 돌변변이 확률의 값을 메모리로부터 가져와 세팅한 후에 선택모듈에서 요청이 있을 때까지 기다린다. 선택모듈이 새로운 염색체를 선택하면 요청이 오고, 교배 및 돌연변이 모듈에서 그 요청을 받으면 한 쌍의 염색체를 가져온다. 그 후에, 난수발생기에서 생성된 랜덤 값이 메모리에 정의해 둔 교배확률보다 작으면 교배연산자가 수행되고, 교배지점도 난수발생기에 의해서 결정된다. 교배 후, 난수발생기에서 생성된 랜덤 값이 메모리에 정의해 둔 돌연변이확률 보다 작으면 이때는 돌연변이 연산자가 수행되어 돌연변이 지점의 비트값(염색체)을 반전 시킨 값으로 저장한다. 새로 만들어진 염색체는 적합도 모듈로 보내진다.

(5) 적합도 모듈

적합도 모듈이 시작되면, 집단의 크기 정보와 초기 집단의 적합도들의 합, 그리고 수행하고자 하는 세대의 수를 저장한다. 초기화가 끝난 후에, 새로운 세대를 생성하기 위한 계산을 시작한다. 먼저, 전체 동작이 종료되었는지 확인해서 그렇지 않으면 집단의 크기를 초기화하고 새로운 세대를 위한 적합도 누적 합을 초기화한다. 그리고 두 개의 집단 중 어느 것을 사용할 것인지 메모리제어모듈에게 알려주고 선택모듈을 동작 시킨다. 그 뒤 교배 및 돌연변이 연산모듈로부터 새로운 염색체 쌍을 받기위한 상태로 간다. 새로운 염색체 쌍을 받아서 적합도 함수에 대한 계산을 수행한다. 다음 세대를 위한 염색체와 그의 적합도 값을 MCM을 통해 메모리로 저장하고, 현 세대의 모든 염색체들의 적합도 값은 누적한다.

(6) 난수 발생기

난수발생기의 출력은 두 모듈에 적용된다. 하나는 선택연산모듈이고, 다른 하나는 교배/돌연변이 연산 모듈이다. 선택모듈에는 집단으로부터 염색체 쌍을 선택할 때 사용되는 적합도들의 합을 비례축소 (scale down)시키는데 사용되는 값이 적용 된다. 교배/돌연변이 모듈의 입력으로 사용되는 값은, 교배와 돌연변이 연산을 수행할 지 결정하는데 쓰이는 신호 2개와, 교배연산 시 교차지점과 돌연변이 연산 시 돌연변이 지점을 나타내는 값이 사용된다. 난수발생기 모듈은 초기 값을 받은 후에 선형 셀룰러 오토마타 (Cellular automata ; CA)를 사용하여 연속적인 난수 비트열을 생성한다. 여기서 사용된 셀룰러오토마타는 16개의 셀들로 이루어져 각 상태가 규칙 90과 150으로 변환된다[8].

규칙 90: $s_i^+ = s_{i-1} \oplus s_{i+1}$ (1)

규칙 150: $s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1}$ (2)

여기서

s_i : i 번째 셀의 현재 상태

s_i^+ : s_i 의 다음 상태

$\oplus$: XOR 연산자

사용된 셀룰러 오토마타의 규칙의 순서는 150-150-90-150-...90-150 으로 변경되면서 모두 0인 비트열을 제외한 비트열들을 생성한다. 이것은 선형 되먹임 이동 레지스터 (linear feedback shift register ; LFSR)에 의해 발생되는 비트열들 보다 무작위성 (randomness)면에서 좋은 결과를 보인다[9].

III. 실 험

진화하드웨어 플랫폼을 구현하기위하여 먼저 GA 하드웨어 엔진을 VHDL로 구현하였다. 하드웨어 구현의 용이성을 위하여 우선 간단한 유전알고리즘(simple genetic algorithm)을 대상으로 하였다. HDL로의 구현은 응용에 따른 목적함수 변경과, 염색체 길이, 집단의 개수 등 의 변경 및 적용하려는 타겟 디바이스의 다양성에 매우 유용하다. 기존의 GA 엔진[10]과 달리 종료시점을 고정된 세대의 진행 횟수 뿐 만이 아니라 종료조건도 병행할 수 있도록 하였다. 설계된 GA 엔진의 동작을 확인하기 위하여 몇 가지 예제의 목적함수에 대해서 FPGA로 구현하였다.

표 1. 최대동작주파수와 디바이스 이용률
Table 1. Maximum frequency and device utilization.

	Max. Freq. (MHz)	Num. of Slices	Num. of Slice Flip Flops	Num. of 4 input LUTs	Num. of MULT 18X18SIOs
$f=2x$	71.782	9 %	4 %	8 %	5 %
$f=x^2$	71.782	9 %	4 %	8 %	15 %
$f=\|x-10\|$	71.968	9 %	4 %	8 %	5 %
$f=10-\|x-10\|$	71.968	9 %	4 %	8 %	5 %
$f=127-\|x-127\|$	71.362	10 %	4 %	9 %	5 %

표 1 은 타겟 디바이스를 Xilinx의 3s500efg320-4로 하여 다섯 가지 예제에 대해 메모리를 제외한 디바이스 이용률과 최대 동작 주파수를 나타낸 것이다.

표 1에서 상위 네 개의 예제는 염색체를 4비트, 적합도합을 8비트로 하여 구현한 것이고, 다섯 번째 예제는 염색체 7비트, 적합도합을 8비트로 하여 구현한 것이다. 표 1의 결과, 하나의 디바이스에 소프트 IP(intellectual property)의 프로세서와 진화대상 하드웨어를 GA 엔진과 함께 구현하여 EHW 플랫폼을 구현할 수 있음을 보인다.

구현된 GA 엔진에 대한 검증은 모델심(ModelSim)을 이용하였다. 집단은 염색체 32개만으로 구성하였고 세대는 10번으로 제한하여 실험하였다.

그림 3에 다섯 가지 예제에 대하여 GA 연산을 수행한 결과를 나타내었다. 구현된 GA 엔진에 대한 검증 차원에서 실험을 수행하였고, 교배율은 0.8, 돌연변이율은 0.1로 검증 결과 목적함수 값에 수렴하여 GA 연산에 이상이 없음을 보였다.

세대를 일정하게 하여 실험한 것은 하드웨어로 구현된 GA 동작 속도를 확인하기 위한 것이다. 각 예제에

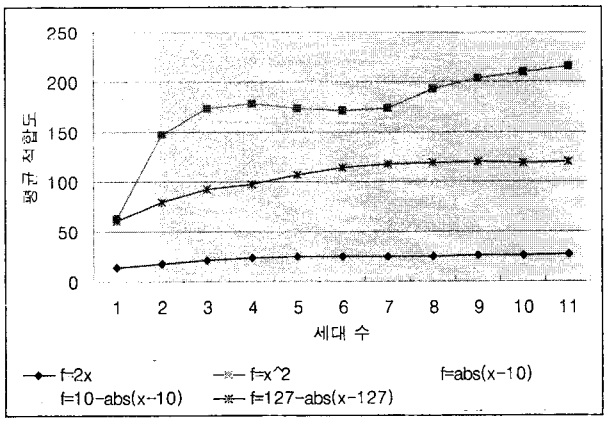

그림 3. 세대 진행에 따른 평균 적합도
Fig. 3. Average fitness (simulation results).

표 2. GA 연산 속도
Table 2. GA operation speed.

	Num. of clock (cycle)	Max. Freq. (MHz)	speed (μs)
$f=2x$	22119	71.782	308.1
$f=x^2$	20763	71.782	289.2
$f=\|x-10\|$	21963	71.968	305.2
$f=10-\|x-10\|$	20919	71.968	290.7
$f=127-\|x-127\|$	20727	71.362	290.4

따라 10 세대에 대한 연산에 걸린 속도를 표 2에 나타내었다.

표 2는 세대 수를 열 번으로 동일하게 하고 연산을 수행하여 나온 결과이다. 타겟 디바이스의 속도 정도(speed grade)는 4로 다소 저속용이다. 고속용으로 바꿀 경우 다른 변경 없이 속도를 올릴 수 있다.

IV. 결 론

동작환경이 바뀌더라도 환경에 적응하여 자체 하드웨어를 변경시켜 동작을 수행할 수 있는 EHW를 구현하기위해 먼저 GA 엔진을 구현하였다. 특정 응용에 전용으로 사용하기 쉽도록 HDL로 설계하여, 구현하고 검증하여 가능성을 보였다. 그러나 실제 활용 가능한 EHW를 구현하려면 구현된 GA의 각 세부블록에 대한 연구, 다른 구조의 GA, 또는 GA 외의 진화알고리즘 등 많은 연구가 필요하다. 따라서 본 논문을 기초로 하여 EHW 구현에 대한 연구를 계속 진행할 것이다.

참 고 문 헌

[1] T. Higuchi, N. Kajihara, "Evolvable Hardware Chips for Industrial Applications", *Communications of the ACM*, vol. 42, no.4, pp.60-66, 1999.

[2] Goldberg, D.E. "*Genetic Algorithm in Search, Optimization, and Machine Learning.*" Addison-Wesley, 1989.

[3] T. Higuchi et al. "Evolvable hardware A first step towards building a Darwin machine." *In Proc. of the 2nd International Conference on simulated Behaviour*, pp.417-424. MIT Press, 1993.

[4] J. Torresen, "An Evolvable Hardware Tutorial", *in Proc. FPL*, pp.821-830, 2004.

[5] M. Murakawa, S Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi. "Hardware

evolution at function level." *In Proc. of Parallel Problem Solving from Nature IV (PPSN IV), volume 1141 of Lecture Notes in Computer Science*, pp.62-71. Springer-Verlag, September 1996.

[6] E. Cantu-Paz. "A survey of parallel genetic algorithms." *Calculateurs Parallels, Reseaux et Systems Repartis*, 10(2): pp.141-171, 1998.

[7] J. Torresen K.A. Vinger. "High performance computing by context switching reconfigurable logic." *In Proc. of the 16th European Simulation Multiconference (ESM2002)*, pp.207-210. SCS Europe, June 2002.

[8] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller. "The analysis of one-dimensional linear cellular automata and their aliasing properties." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp.767-778. July 1990.

[9] P. D. Hortensius, H. C. Card, and R. D. McLeod. "Parallel random number generation for VLSI using cellular automata." *IEEE Transactions on Computers*, v.38, pp.1466-1473. October 1989.

[10] Scott, S and Seth. "HGA: A Hardware-Based Genetic Algorithm." *Proc. of the ACM/SIGDA Third Int. Symp. on Field-Programmable Gate Arrays*, pp.53-59, 1995.

저 자 소 개

동 성 수(정회원)
대한전자공학회 논문지
제40권 CI편 제6호 참조
현재 용인송담대학 디지털전자전공 교수

이 종 호(정회원)
대한전자공학회 논문지
제40권 CI편 제6호 참조
현재 인하대학교 정보통신공학부 교수