진화형 하드웨어를 위한 하드웨어 최적화된 유전자 알고리즘 프로세서의 구현

論 文
49D-3-7

Implementation of Genetic Algorithm Processor based on Hardware Optimization for Evolvable Hardware

金 鎭 貞* · 鄭 德 鎭**
(Jin-Jung Kim · Duck-Jin Chung)

Abstract - Genetic Algorithm(GA) has been known as a method of solving large-scaled optimization problems with complex constraints in various applications. Since a major drawback of the GA is that it needs a long computation time, the hardware implementations of Genetic Algorithm Processors(GAP) are focused on in recent studies. In this paper, a hardware-oriented GA was proposed in order to save the hardware resources and to reduce the execution time of GAP. Based on steady-state model among continuos generation model, the proposed GA used modified tournament selection, as well as special survival condition, with replaced whenever the offspring's fitness is better than worse-fit parent's. The proposed algorithm shows more than 30% in convergence speed over the conventional algorithm in simulation. Finally, by employing the efficient pipeline parallelization and handshaking protocol in proposed GAP, above 30% of the computation speed-up can be achieved over survival-based GA which runs one million crossovers per second (1MHz), when device speed and size of application are taken into account on prototype. It would be used for high speed processing such of central processor of evolvable hardware, robot control and many optimization problems.

Key Words : 유전자 알고리즘, 진화형 하드웨어, GAP, modified tournament selection, steady-state model

1. 서 론

최근 공학의 여러 가지 응용 분야 중 복잡한 제약성을 가진 대규모의 최적화 문제들은 일반적인 수학적인 프로그램이나 최적화 알고리즘의 조합을 이용하여 짧은 연산 시간 안에 최적의 해를 구하기가 매우 어렵다. 그러므로 이러한 문제들을 적당한 연산 시간 안에 풀 수 있는 주요한 메커니즘으로써 진화 알고리즘(Evolutionary Algorithm)의 하나인 유전자 알고리즘(Genetic Algorithm, GA)이 사용된다. 그러나 유전자 알고리즘은 반복적인 진화과정과 적응과정을 거치게 되므로 많은 연산시간이 필요하기 때문에 별도의 프로세서, 즉 유전자 알고리즘 프로세서(Genetic Algorithm Processor, GAP)라고 불리는 하드웨어의 연구의 필요성이 요구된다.

기존의 유전자 알고리즘은 소프트웨어 기반의 응용 분야에 한정되어 적용되었기 때문에 고속의 연산 처리를 요구하는 시스템 및 실시간 처리가 요구되는 최적화 문제들, stand-alone이 요구되어지는 시스템들의 응용분야의 적용에 제약을 가지고 있었다. 본 연구의 유전자 알고리즘 프로세서는 VLSI 구현을 위한 하드웨어 지향의 유전자 알고리즘을 기반으로 연산 속도가 크게 향상되어 진화형 하드웨어(Evolvable Hardware, EHW)와 같은 고속 처리가 요구되어지는 다양한 응용 분야에 적용 가능하다.

본 연구에서는 하드웨어 구현 및 성능 면에서 기존의 소프트웨어 및 하드웨어 기반의 알고리즘의 장·단점을 분석함으로써 유전자 알고리즘의 하드웨어를 구현하였다. 기존의 알고리즘 분석을 통해, 하드웨어 구현에 효율적인 수정된 개체 생존 결정 방법과 수정된 토너먼트 선택 방법의 장점을 접목한 개선된 정상상태(steady-state) 모델을 선택하였다. 이 알고리즘의 타당성을 검증하기 위하여 MATLAB을 이용한 시뮬레이션으로 알고리즘의 우수성을 입증하였다. 또한 제안된 알고리즘을 VHDL을 이용하여 설계하여 FPGA가 내장된 PCIGEN10K board 상에서 유전자 알고리즘 프로세서를 제작한 후 논리 분석기 등의 장비를 이용하여 시뮬레이션 상의 동작과 비교 검증하였다. 제안된 하드웨어의 성능 평가를 위해 Dejong 함수와 같은 수학적 최적화 문제 및 set covering problem을 적합도 함수로 적용, 하드웨어로 구현하여 평가하였다.

다음 2장에서는 유전자 알고리즘의 동작, 특성 및 구조를 설명하고, 제안된 유전자 알고리즘을 설명한다. 3장에서는 진화형 하드웨어, 기존의 유전자 알고리즘의 하드웨어 구현 및 제안된 유전자 알고리즘 프로세서의 병렬처리 및 pipeline 등의 구조와 VLSI 구현에 관해 설명한다. 그리고 4장에서는 적용된 두 가지의 적합도 함수 및 구현된 prototype의 결과 및 분석에 대해 설명하고 마지막으로 결론을 맺고자 한다.

* 準 會 員 : 仁荷大 電子材料工學科 碩士課程
** 正 會 員 : 仁荷大 電子材料工學科 敎授 · 工博
接受日字 : 1999年 10月 22日
最終完了 : 2000年 2月 23日

2. 유전자 알고리즘(Genetic Algorithm)

2.1 유전자 알고리즘의 동작 및 구조

자연계에 있는 생물의 진화과정에 있어서, 어떤 세대를 형성하는 개체들의 집합, 즉 개체군 중에서 환경에 대한 적합도가 높은 개체가 높은 확률로 살아남아 재생할 수 있게 되며, 이때 교차 및 돌연변이로서 다음 세대의 개체군을 형성하게 된다. 이처럼 유전자 알고리즘은 선택과 교차, 돌연변이 등의 연산자를 이용하여 문제에 대한 후보해 또는 유기체를 표현하는 개체군을 새로운 개체군으로 반복적으로 변형한다. 각각의 개체는 보통 한 개의 염색체로 구성되는데 이는 다시 이진수 값을 취하는 유전자들의 스트링으로 이루어진다. 개체들은 그 적합도에 따라서 선택되어 자식(offspring)을 복제(reproduction)하는데 적합도가 높은 개체가 적합도가 낮은 개체보다 평균적으로 더 많은 자손을 재생산한다. 교차는 임의로 선택된 두 부모 염색체의 스트링의 일부분이 끊어져서 상호 교환됨으로써 정보를 교환하는 연산자이다. 이것은 두 개의 단일 염색체 유기체 사이의 이성간의 재결합을 모사하는 것이다. 돌연변이는 염색체 내의 어떤 유전자의 값을 임의적으로 바꾼다. 반전 연산자는 염색체의 연속된 부분의 순서를 거꾸로 함으로써 유전자들을 재정렬한다.[2,3]

유전자 알고리즘의 동작원리를 설명하면 문제에 대한 후보해를 염색체의 형태로 표현하고, 무작위로 생성된 염색체들로 개체군을 초기화한다. 임의의 값으로 초기화된 개체는 상대적인 문제해결 능력에 따라 그 적합도가 평가되며 적합도에 따라 다음 세대에 부모의 유전자가 복제되는 정도를 달리 함으로써 우성형질을 지닌 개체들은 열성 형질을 지닌 개체들에 비하여 더욱 많은 자식을 생성할 수 있도록 유도된다. 이러한 선택(selection)은 제공된 요소들 중에 임의로 다음 세대로 진화시킬 대상을 선택하게 되고, 이 선택 복제된 개체들은 선택된 두 인자들이 합쳐서 새로운 인자를 생성하게 되는 교차, 돌연변이 등의 여러 가지 유전 연산자들에 의하여 재결합(recombination)되어 다음 세대의 개체군을 생성한다. 마지막으로 개체군의 통계(population statistics) 과정에서는 그 결과를 평가해 기록하고 최적화 된 결과를 판단하게 된다. 이와 같은 세대 교체는 원하는 수준의 해가 개체군 내에 존재하거나 또는 다른 종료 조건이 만족할 때까지 반복된다.[1]

유전자 알고리즘의 진화 형태를 나타내는 세대 모델은 크게 두 가지로 구분이 된다. 일반적으로 간단한 유전자 알고리즘에서는 모든 개체가 일제히 자손을 만들고, 다음 세대의 집합을 만드는 이산세대 모델(discrete generation model)을 사용하고 있는데, 그것과는 반대로 각 세대에서 몇 개의 개체만 교체되는 연속세대 모델(continuous generation model)에 기반을 둔 것도 있다. De jong은 현 세대 중에서 몇 퍼센트가 교체될 것인가를 표현하는 매개변수인 세대차이(generation gap)를 도입하여 연속세대 모델을 처음 제시하였다.[5] 연속 세대 모델의 한 형태인 정상상태 모델(steady-state model)[4]에서는 다음 세대로 넘어갈 때 두 개체만을 선택하고 두 개의 자손의 개체를 만들어 적합도가 낮은 개체를 두 개 제거한다. Crowding model[5]은 세대 차이에 의해서 설정된 개체 수를 교체시키기 위해서, 제거되는 개체를 전체집합 중에서 crowding factor에 의해서 지정된 부분집합으로부터 선택한다. 이런 연속세대 유전자 알고리즘의 중요한 특징의 하나는 전체의 개체가 비동기적으로 선택 교차를 수행한다는 것이다.

2.2 제안된 하드웨어 지향의 유전자 알고리즘

다양한 문제에 대한 실제 적용에 있어, 유전자 알고리즘 프로세서의 하드웨어적 요소 및 실행시간을 줄이기 위해서 하드웨어 지향의 알고리즘이 요구된다. 이를 위해 수정된 생존 결정방법 및 수정된 토너먼트 선택 방법을 지닌 개선된

표 1 제안된 유전자 알고리즘
Table 1 The proposed genetic algorithm

◆ *Initialize Parent β and best chromosome*
Parent β, Best_chrom = all bits is initialized by zero
While optimal value (Best_chrom) **do**
◆ *Select two chromosomes from population using two random numbers.*
Chrom α = Population{Chrom α_adrs =Random($N_{p\alpha}$)}
Chrom β = Population{Chrom β_adrs =Random($N_{p\beta}$)}
◆ *Copy the previous parent for the other parent*
Parent β = *previous* Parent α
◆ *Decide one parent from two chromosomes using fitness value*
Parent α = **Max. of Fitness**(Chrom α, Chrom β)
Worse_adrs
= **Min. of Fitness**(Chrom α_adrs, Chrom β_adrs)
Worse_fit
= **Min. of Fitness**(Chrom $\alpha_{fitness}$, Chrom $\beta_{fitness}$)
◆ *Do Mutation and Crossover using a few methods*
• **Crossover**(α, β): [1-point, 2 point, uniform]
$Children_{data}$ (α, β) = **Crossover**(Parent α, Parent β)
• **Mutation** (α, β) : [single-point, multi-point](P_m)
$Children_{data}$ (α, β)
= **Mutation**(children α_{data}, children β_{data})
◆ *Evaluate a fitness on a specific problem*
$Children_{fitness}$ (α, β)
= **Fitness**(children α_{data}, children β_{data})
◆ *Update a population using new offspring compared a worse chromosome*
New_chrom
= **Max. of Fitness**(Children α_{fit}, Children β_{fit})
IF New_chrom$_{fitness}$ > Worse_fit **then**
Population(Worse_adrs) = New_chrom
END IF
IF New_chrom$_{fitness}$ > Best_chrom$_{fitness}$ **then**
Best_chrom = New_chrom
END IF
◆ *Adapt a Mutation & Crossover using a relation of best chromosome and worse chromosome*
End While

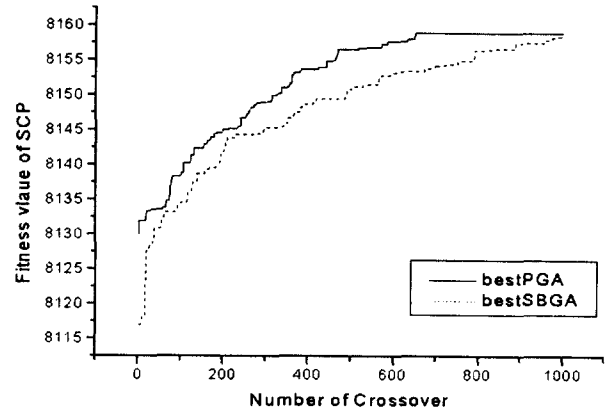
정상상태 모델이 채택되었다.

표 1은 제안된 유전자 알고리즘의 의사코드를 보여주고 있다. 표 1에 보여지는 것처럼 제안된 유전자 알고리즘은 의사 난수 발생기를 이용하여 개체군의 데이터(population)로부터 두 부모 개체를 무작위 선택한 후에 두 개체 중에서 우성 형질을 지닌 개체를 parentα로 정의하고, 앞선 세대의 반복과정에서의 parentα를 parentβ로 정의한다. 그 다음 parentα와 parentβ를 이용하여 새로운 두 개의 자손 개체를 생성하기 위하여 돌연변이와 교차를 수행 후에 적합도 평가를 통해 두 자손 중 우성 형질을 지닌 자손 개체를 결정한다. 이 생존된 자손 개체와 열성 형질의 부모 개체의 대체를 통해 개체군을 갱신함으로써 원하는 개체 형질을 얻도록 한다. 또한 마지막으로 더 빠른 수렴속도를 얻기 위해, 유전자 알고리즘의 파라미터 및 방법의 경험적인 변화가 선택적으로 수행 가능하다.

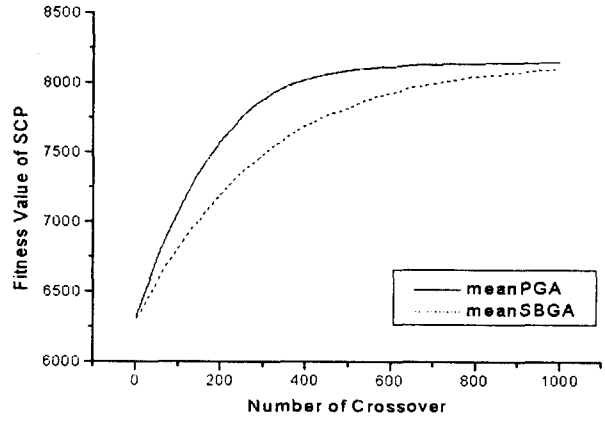
제안된 알고리즘은 이산세대 모델에 비해 개체군의 저장을 위해 사용되는 메모리를 반으로 줄일 수 있는 개선된 정상상태 모델이 적용되었다. 기존의 정상상태 모델의 경우, 두 개의 자손을 생성하여 두 개체를 부모 개체와 대체하지만 토너먼트 선택 방법과 함께 쓰일 경우 기존의 탐색 결과를 최대한 이용하려는 경향, 즉 exploitation이 상대적으로 강조되어 premature convergence가 발생할 확률이 상대적으로 높아지게 된다. 이를 해결하기 위하여 두 생성된 자손의 개체 중 하나의 우수한 자손만이 부모 개체를 대체할 수 있도록 하여 신속한 해의 확산을 막고, 최적의 해에 수렴할 수 있도록 하였다. 즉 개체의 보존 방법 면에서 보면, 두 부모 개체 중 열성형질을 지닌 부모 개체의 적합도보다 자손의 적합도가 나올 때 자손 개체의 생존이 결정되는 생존 방법을 사용하였다.

제안된 유전자 알고리즘은 기존의 속도 지향의 survival-based 알고리즘의 무작위 선택과 비교할 때, 정상상태 모델과 함께 쓰일 때 가장 우수한 성능을 나타내는 토너먼트 선택 방법을 통해 비교되는 개체의 증가시킴으로써 선택 압력을 높여 확률적으로 최적의 해에 도달할 가능성을 높였다. 또 사용된 개체의 선택 방법인 토너먼트 선택 방법은 확률에 기반한 모델에 비해 개체들의 적합도 총합이나 적합도에 따른 개체의 분류를 위한 추가적인 정보를 요구하지 않기 때문에 VLSI의 면적과 속도의 향상을 얻을 수 있을 뿐 만 아니라 정상상태 모델과 동시에 사용되어 하드웨어적인 면에서 그 성능을 더 향상시킨다.

유전자 알고리즘의 성능 평가의 기준은 각 세대의 최적인 개체의 적합도나 개체군의 평균 적합도를 사용한다. 그림 1은 19 rows와 63 columns로 구성된 non-unicost set covering problem에 대한 survival-based 알고리즘과 제안된 알고리즘의 각 개체군의 적합도 평균과 각 개체의 적합도의 최대값의 대비를 통해 성능을 평가하는 나타낸 것이다. 그림에서 보는 바와 같이 각 개체의 적합도의 최대값 및 그 평균값에서 큰 차이가 나타내며 모두 제안된 알고리즘이 우수함을 알 수 있다. 즉, 개체군의 크기 200, 스트링에 대한 돌연변이 확률 0.04, 두점 교차 확률 1의 방법을 이용한 시뮬레이션을 통해 그림과 같이 survival-based 알고리즘은 평균적으로 760번의 교차 후에 최적의 해를 찾는데 반해 제안된 알고리즘은 평균적으로 단지 512번의 교차 수행 후에 같은 해를 찾을 수 있



(a) Comparison of best value in each generation



(b) Comparison of mean value in each generation

그림 1 Non-unicost set covering problem에서의 survival-based 알고리즘(SBGA)과 제안된 알고리즘(PGA)의 비교

Fig. 1 Comparison of survival-based GA(SBGA) and proposed GA(PGA) in non-unicost set covering problem

다. 특히 평균값에서의 두 알고리즘의 값의 차이는 선택 압력의 차이에 따른 것으로써 확률적으로 최적의 해에 수렴할 가능성의 차이를 나타낸 것이며, 이는 기존 알고리즘과 차별화 된 알고리즘의 우수성을 나타낸다.

3. 유전자 알고리즘 프로세서

3.1 진화형 하드웨어(Evolvable Hardware)

EHW는 그 주변의 환경과 상호 작용하여 동적이면서도 자율적으로 구조와 반응을 변하게 할 수 있는 하드웨어로서 언급되어 왔으며 90년대 초기에 FPGA와 같은 쉽게 재구성 할 수 있는 하드웨어의 발달로 많은 관심을 끌어 왔다[6,7]. 또 EHW는 패턴 인식으로부터 적응 제어와 같은 넓은 범위의 일을 수행할 수 있다는 것이 기존의 연구 결과이다.[6] EHW의 중요한 motivation의 하나는 외부의 힘 없이도 인간과 같이 진화를 하기 때문에 자연으로부터 배운다는 것이다.

현재의 거의 모든 EHW는 주요 적응 메커니즘으로서 진화 알고리즘을 사용한다.[6] 본 연구에서는 EHW의 구현을 위해

표 2 과거 및 개발된 유전자 알고리즘의 하드웨어 구현
Table 2 Previous & developed work for hardware implementation of GA

	S.D. Scott (HGA)[8] *Washington Univ. 1994*	Paul Graham[9] *Brigham young Univ. 1995*	M. Salami (GAP)[10] *Victoria Univ. 1996*
Generation Steady-state	Steady-state	Generation	Generation
Selection	Roulette	Roulette	Roulette
R.N.G.	CA	RNG	CA
Crossover	1-point	1-point	1-point
Pop. Size / Chrom. L	16 / 3	64, 128, 256 / 24	64 / 32
Note	BORG board	Splash2	NeoCAD FPGA
	Tommoska & Vuori[11] *Helsinki Univ.,1996*	B. Shakleford (S-GA)[12] *Mitsubishi E.C., 1997*	T.Higuchi, I.Kajitani[13] *Tsukuba,1998*
Generation Steady-state	Steady-state	Steady-state	Steady-state
Selection	Round-robin fashion	Survival	Elitist Recombination
R.N.G.	LSHR	CA	CA
Crossover	1-point	1-point	Uniform
Pop. Size / Chrom. L	32 / 32	256 / 94	32 / 2048
Note	Altera FPGA	Aptix AXB-MP3 FPGA	EHW chip
	N.Yoshida[14] (GAP) *Kyushu Univ.,1997*	S. Wakabayashi [15] (GAA) *Hiroshima Univ.,1998*	Our developed GAP 1999
Generation Steady-state	Steady-state	Generation	Steady-state
Selection	Simplified tournament	Roulette & elitist strategy	Tournament
R.N.G.	CA	CA	CA
Crossover	1-point	2-point, Uniform, adaptive	1-point,, 2-point, Uniform
Pop. Size / Chrom. L	64 /32	64, 128 / 64	128Min. (64K Max.) / 64 Max.
Note	SFL(HDL)	0.5㎛ CMOS technology	PCIGEN10K (Altera's FPGA)

진화 알고리즘 중에서 유전자 알고리즘을 사용하였다. EHW는 보통 FPGA와 같은 Programmable Logic Devices(PLD) 상에서 구현되었으며 이는 PLD의 구조와 기능이 변할 수 있는 구조적 bit들의 집합으로 되어있기 때문이다. EHW는 재구성 하드웨어의 구성에 필요한 구조적 bit을 유전자 알고리즘의 개체들로 간주하며, 원하는 결과가 나오지 않거나 외부 환경의 변화가 발생하면 유전자 알고리즘을 수행함으로써 하드웨어의 구조가 적합하게 재구성되는 시스템이다. 즉 EHW는 재구성 하드웨어와 진화 알고리즘과의 결합체라고 할 수 있다.

3.2 기존의 유전자 알고리즘 하드웨어 연구

진화 연산에 있어서의 심각한 문제인 연산 시간을 줄이기 위해서 유전자 알고리즘의 병렬 분산 처리 및 하드웨어 구현 등의 방법이 시도되어 왔다.[8] 최근의 연구 방향은 하드웨어 구현에 병렬 분산 처리 기법을 적용하는 것이며, 이는 하나의 효율적인 프로세서를 구현하여 확장 적용하여 병렬 분산 처리하는 것으로, 바탕이 되는 확장 가능한 하나의 효율적인 프로세서 구현이 매우 중요하다.

기존의 유전자 알고리즘 하드웨어의 구현은 Scott[8] 등에 의해 처음 시도되었으며, 이는 간단한 표준적인 유전자 알고리즘(standard simple generation algorithm)에 기초한 Hardware-based Genetic Algorithm(HGA)이다. Scott의 연구 및 그를 뒤이은 유전자 알고리즘의 하드웨어 구현 및 현재 개발된 하드웨어의 구현은 표 2와 같다. Yoshida[14] 등은 Genetic Algorithm Processor(GAP)라고 불리는 하드웨어 지향의 유전자 알고리즘을 제안하였으며 이는 실질적으로 세대라는 개념을 가지지 않는 정상상태 모델을 기반으로 하였다. Shakleford[12] 등은 정상상태 모델과 어느 정도 비슷한 Survival-based 유전자 알고리즘을 제안하였으며 Kajitani [13] 등은 LSI chip 상에서 진화형 하드웨어(EHW)를 위한 유전자 알고리즘 하드웨어를 제시하였다. 또 Wakabayashi [15] 등은 범용 유전자 알고리즘 하드웨어로서 GA Accelerator(GAA) chip이라 불리는 유전자 알고리즘을 VLSI 구현하였다.

그러나 이들 연구는 FPGA 상에서 간단히 합성한 유전자 알고리즘의 유전 연산자와 소프트웨어를 비교한 속도의 향상에 의해 평가되어 진행되어 왔다.

3.3 유전자 알고리즘 프로세서의 구현

3.3.1 병렬처리와 Pipeline 구조의 적용

유전자 알고리즘 프로세서의 효율적인 구현은 하드웨어의 복잡성과 성능을 동시에 고려한 구조여야 한다. 이런 관점에서 볼 때 유전자 알고리즘의 병렬 처리 방법은 개체군과 개체의 처리 방법에 따라 크게 세 가지로 구분된다.[16] 제안된 유전자 알고리즘 프로세서는 병렬 처리 방법에 있어 *serially on population and parallel on chromosome*을 사용하여 개체군에서의 개체는 직렬로, 대립 유전자는 병렬로 처리한다. 이는 정상상태 모델의 구현의 용이성, 개체의 길이, 적용된 문제의 난이도 및 유전 연산자의 처리속도를 기반으로 하여 채택된 것이며 하드웨어의 복잡성과 성능 모두를 고려할 때 매우 효율적인 선택이다.[14]

그림 2는 제안된 유전자 알고리즘 프로세서의 블록 다이어그램을 나타내며, 알고리즘과 함수의 각각의 블록별로 정의하면 8개의 부분으로 구성되어 handshaking 신호 및 기능적인 세부모듈에 따라 동작한다. 각 블록들은 적용되는 알고리즘에 따라 연산하는데 소요되는 지연시간이 다르므로 전체 프로세서의 동시 실행성(concurrency)에 중요한 영향을 미치는 요소가 된다. 연산의 효율성을 위해서는 이러한 동시성과 블

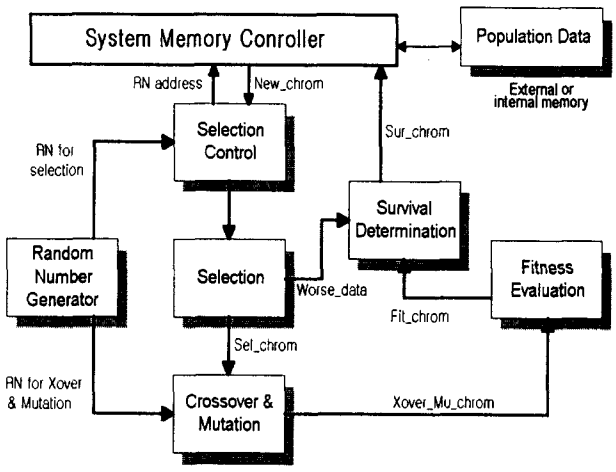


그림 2 유전자 알고리즘 프로세서의 블록 다이어그램
Fig. 2 Block diagram of genetic algorithm processor

제안된 하드웨어의 각 블록의 함수와 알고리즘에 따른 모든 가능한 상태에 대해 상태도로 나타내었다. 이를 기준으로 전체 신호의 의존성을 알기 위해 signal-dependency diagram을 작성하면 그림 3과 같다. 그림 3에서 보면 (S_0, S_1), (S_2, S_3), (S_5, S_6), (S_7, S_8)의 블록이 각각 독립적인 병렬 그룹을 형성하고 있으며, 이러한 그룹을 형성하는 블록들은 그룹내의 다른 블록과 독립적으로 연산이 가능하게 된다. 따라서 이러한 신호의존성을 고려하여 같은 병렬 그룹내의 블록들은 동일한 지연시간을 갖도록 설계되었다. 즉 각 블록의 효율성과 동시 실행성의 증대를 위해 동일한 작업 처리량을 유지하는 범위 내에서 병렬 처리를 실행하도록 설계하여 서로 다른 지연시

표 3 각 모듈의 블록 별 상태도
Table 3 State diagram of each module

S_0	R/W Register(*Population*)
S_1	*Chrom* α, *Chrom* β ←Load Bufffer($RN\_N_{p\alpha}$, $RN\_N_{p\beta}$)
S_2	$RN\_N_{p\alpha}$, $RN\_N_{p\beta}$ ←Random Number for selection
S_3	*Chrom* α, *Chrom* β, $Chrom_{adrs}$, $Chrom_{adrs}$ ←Selection Controller
S_4	*Parent* α, *Parent* β, $Less\text{-}fit_{data}$ ←Selection(*chrom* α, *chrom* β)
S_5	RN_p_c, RN_p_m ←Random Number for crossover & mutation
S_6	*Child* α, *Child* β ←Crossover_Mutation (*Parent* α, *Parent* β, RN_p_c, RN_p_m)
S_7	$Child_{fits\,\alpha}$ ←Fitness Evaluation(*Child* α)
S_8	$Child_{fits\,\beta}$ ←Fitness Evaluation(*Child* β)
S_9	*New_chrom*(Write Buffer) ←Survival Determination ($Child_{fits}$, $Child_{fits}$, $Less\text{-}fit_{data}$)

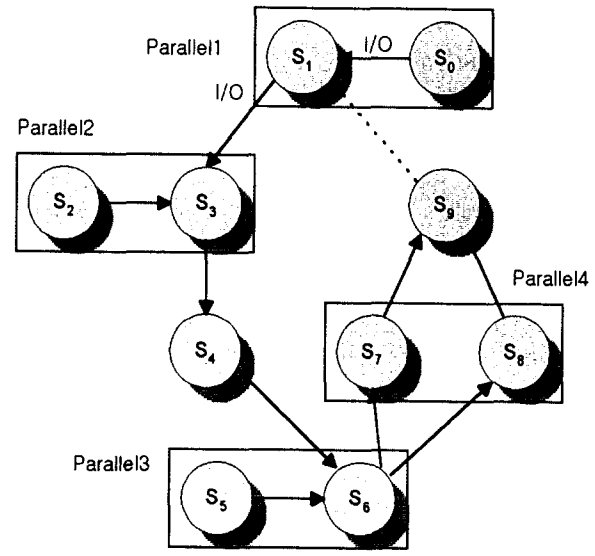


그림 3 Signal dependency diagram
Fig. 3 Signal dependency diagram

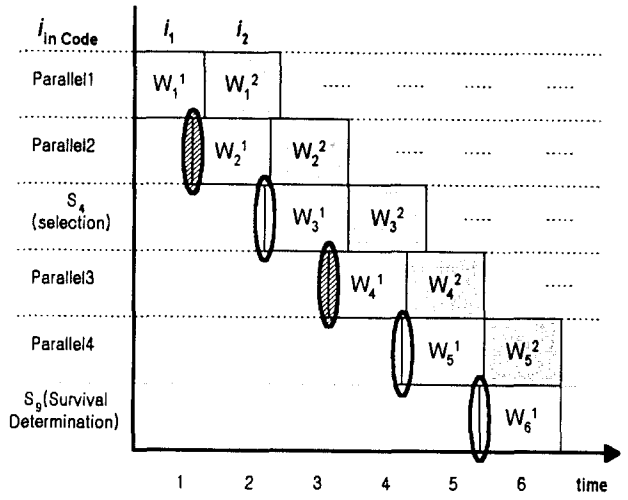


그림 4 pipeline 처리를 위한 space diagram
Fig. 4 Space diagram for pipeline processing

간으로 인한 별도의 지연시간을 없애도록 하였다.

또한 각 병렬그룹내의 지연시간이 모두 다르게 되므로 연속적인 개체의 update를 위해 pipeline 구조를 사용하였다. 이

표 4 각 병렬 그룹 내에서의 clock 수
Table 4 Number of clock in each parallel group

Parallel Group	Number of Clock
Parallel 1	7 (Read)
Parallel 2	5
S4(selection)	6
Parallel 3	8
Parallel 4	6
S9(Survival Determination)	6

러한 pipeline 구조는 개체의 길이 및 하드웨어의 지연 시간에 따라 다른 스테이지를 가지게 되므로 pipeline을 적용하기 위해 우선적으로 각 병렬그룹의 지연시간을 프로세서의 동작에 사용되는 clock을 이용하여 나타내면 표 4과 같다. 여기서 각 병렬 그룹의 지연시간은 표현의 용이성을 위해, 정수로 표현 가능한 최소값을 이용하여 나타내었으며 자세한 지연 시간의 분석은 뒤에서 설명될 것이다. Pipeline이 이러한 병렬 그룹의 지연시간을 활용하여 space diagram을 이용해 전체 데이터의 플로우를 표시하면 그림 4와 같다.

각 모듈 상에서 pipeline에 영향을 미치는 파라미터 결정을 통해 유전자 알고리즘 프로세서의 성능을 분석하였으며, 이에 사용된 파라미터는 다음과 같다.

1. S_i (actual service time) : 각 pipeline의 stage i 가 메시지의 입력을 받아서 처리하고, 다음 stage에 출력을 보내는데 걸리는 시간
2. F_i (flow rate) : 전체 실행동안 stage i 에 도착하는 메시지들의 수
3. S_{norm_i} (normalized service time) : 다음 식(1)과 같이 정의되며,

$$S_{norm_i} = \frac{S_i \cdot F_i}{F_{out}} \tag{1}$$

F_{out}은 normalizing factor로써 pipeline의 flow rate를 나타내며, 이는 정상상태 모델에서는 세대수 그 자체를 나타내므로 $F_{out} = g$로 나타낼 수 있다.

모든 각 stage에 대해 S_{norm_i}를 결정하고, 이에 따라 S_{norm_i}들을 비교하여 가장 큰 부분을 찾음으로써 pipeline의 병목지점을 찾는다. 제안된 pipeline 구조와 Kenyon 등의 모델[18]과의 가장 큰 차이점은 "모든 모듈의 출력이 유용한 것은 아니다" 라는 가정 하에 연산의 결과가 pipeline의 나머지 모듈에 계속적으로 사용되는 것과 같은 오직 유용한 연산만을 고려하였다. 모델에 사용된 모든 작업 처리 시간은 상수이며, technology에 의존하지 않고 분석하기 위하여 모든 식은 clock cycle을 이용하였다. 또 프로세서의 무작위 성질은 service time 및 flow rate의 정확한 예측을 막기 때문에, 이를 위해 무작위 변수(random variable)는 기대값 측정에 독립(independence)과 균일 분포(uniform distribution)를 가정하였다.

각 모듈의 파라미터를 분석하면, 다음 아래 부분에 열거한 것과 같다. 특히 내부 메모리 부분의 지연은 system memory controller와 survival determination module에 영향을 미치므로 이의 동작을 고려하여 계산하였다.

a. *Internal Memory Module*(IMM): SMC와 SDM와의 읽기 · 쓰기 동작을 위한 handshaking 처리에 2 cycle이 요구되며, 내부 메모리 모듈의 읽기 · 쓰기 동작(r, w)의 지연시간은 technology에 의존하는 파라미터로써 읽기 동작은 두 개체의 데이터를 연속적으로 읽기 때문에 쓰기 동작에 비해 그 연산 시간이 더욱 크다. 쓰기 동작이 발생할 확률을 p_w로 정의하면, $S_0^{avg} = 2 + r + (2+w)p_w$로 표현 가능하다. 이에 따라 $S_0^{\min} = 2 + r$, $S_0^{\max} = 4 + r + w$로 나타낼 수 있으며, $F_0 = g$ 이므로 표준화된 평균 작업 처리 시간은 다음 식(2)과 같이 나타낼 수 있다.

$$S_{norm_0}^{avg} = \frac{S_0 \cdot F_0}{F_{out}} = 2 + r + p_w(2+w) \tag{2}$$

b. *System Memory Controller*(SMC): 내부의 IMM, SM 및 RNG와의 handshaking 처리에 4 cycle이 요구되며, 데이터의 연산에 1 cycle이 요구된다. 단, SDM이 개체군, 즉 메모리에 쓰기 작업을 하고 있을 경우, 최소 1 cycle에서 최대 2+ w cycle 동안 대기 상태를 유지해야만 한다. 그러므로 $S_1^{\min} = 5$, $S_1^{\max} = 7 + w$ 로 나타낼 수 있으며, 평균 작업 처리 시간은 식(3)과 같다.

$$S_1^{avg} = 5 + \sum_{i=0}^{i=2+w} (i \cdot p_i) \cdot p_w \approx 5 + \frac{(2+w)p_w}{2} \tag{3}$$

위 식(3)에서의 p_i 는 쓰기 동작에 따른 지연 시간이 발생할 확률로써 $\sum_{i=1}^{2+w} p_i = 1$ 이어야만 한다. 앞서 가정한 바와 같이 p_i를 균일 분포로 가정하면 최대값과 최소값을 이용하여 식(3)과 같이 표현 가능하다. 따라서, flow rate는 $F_1 = F_0 = g$ 와 같이 나타낼 수 있으므로 $S_1 = S_{norm_1}^{avg}$된다. 따라서 표준화된 연산 시간의 평균은 식(3)과 동일하게 표현된다.

c. *Selection Module*(SM): SMC와 MCM과의 handshaking 처리 및 데이터의 처리에 6 cycle이 요구되므로, $S_2 = 6$ cycle로 나타낼 수 있으며, flow rate는 $F_1 = F_2 = g$ 와 같이 표현되므로 $S_{norm_2} = 6$ 으로 나타낼 수 있다.

d. *Mutation & Crossover Module*(MCM): SM과 마찬가지로 간단히 구할 수 있다. 즉, SM와 FEM과의 handshaking 처리 및 데이터의 처리에 8 cycle이 요구되므로 $S_3 = 8$ cycle, flow rate는 $F_3 = g$와 표현되며, $S_{norm_3} = 8$로 나타낼 수 있다.

e. *Fitness Evaluation Module*(FEM): 적합도를 평가하기 위해서는 개체마다 연산시간, t_{eval}이 요구되며, 주변 모듈(MCM, SDM)과 handshaking을 위해 3 cycle이 요구된다. 그러므로 $S_4 = t_{eval} + 3$ cycle로 나타낼 수 있으며, flow rate는 $F_4 = g$와 같이 표현되므로 $S_{norm_4} = t_{eval} + 3$로 나타낼 수 있다.

f. *Survival Determination Module*(SDM): IMM와 FEM과의 handshaking 처리 및 데이터의 처리에 6 cycle이 요구된다. 더불어, SMC이 개체군, 즉 메모리에 읽기 작업을 하고 있을 경우, 최대 1+r cycle 동안 대기 상태를 유지해야만 한다. 그러나 동시에 읽기와 쓰기 작업을 위한 신호가 요구되었을 경우에는 쓰기 동작에 그 우선 순위가 있다. 그러므로

표 5 각 모듈의 표준화된 평균 작업 처리 시간
Table 5 Normalized service time of each module

	Minimum	Maximum	Approx. Average
IMM	$S^{\min}_{norm_0}=2+r$	$S^{\max}_{norm_0}=4+r+w$	$S^{avg}_{norm_0}=2+r+p_w(2+w)$
SMC	$S^{\min}_{norm_1}=5$	$S^{\max}_{norm_1}=7+w$	$S^{avg}_{norm_1}\approx 5+\frac{(2+w)p_w}{2}$
SM	$S^{\min}_{norm_2}=6$	$S^{\max}_{norm_2}=6$	$S^{avg}_{norm_2}=6$
MCM	$S^{\min}_{norm_3}=8$	$S^{\max}_{norm_3}=8$	$S^{avg}_{norm_3}=8$
FEM	$S^{\min}_{norm_4}=t_{eval}+3$	$S^{\max}_{norm_4}=t_{eval}+3$	$S^{avg}_{norm_4}=t_{eval}+3$
SDM	$S^{\min}_{norm_5}=6$	$S^{\max}_{norm_5}=7+r$	$S^{avg}_{norm_5}\approx 6+\frac{(1+r)p_w}{2}$

$S_5^{\min}=6$, $S_5^{\max}=7+r$ cycle로 나타낼 수 있고, 역시 p_i를 균일 분포로 가정하면 $\sum_{i=0}^{1+r} p_i=1$ 이어야만 하며, 평균은 SMC의 표현과 비슷하게 최대값과 최소값을 이용하여 식(4)로 표현될 수 있다.

$$S_5^{avg}=6+\sum_{i=0}^{i=1+r}(i\cdot p_i)\cdot p_w\approx 6+\frac{(1+r)p_w}{2} \tag{4}$$

그러므로 $S^{\min}_{norm_5}=6$, $S^{\max}_{norm_5}=7+r$ cycle로 나타낼 수 있으며, 평균은 식(4)와 동일하게 표현된다.

모든 모듈에서 flow rate가 같기 때문에 작업 처리 시간과 표준화된 값과 같으며, 이에 따라 각 모듈의 값을 정리하면 표 5과 같다.

표 8의 수식을 비교하면, $S^{avg}_{norm_2}<S^{avg}_{norm_3}$ 이므로 SM은 병목지점이 아니다. 더불어 앞서 설명하였듯이 읽기 동작은 두 번 연속으로 발생하고 한번의 r과 w에 걸리는 clock cycle이 같다고 가정하면 $r=2w-1$로 표현 가능하다. 이에 따라 $S^{avg}_{norm_1}<S^{avg}_{norm_5}$ 이므로 SMC도 역시 병목지점이 아니다. SDM이 병목지점인지를 알기 위해서 식(5)과 같이 분석하면, 쓰기 동작은 해당되는 메모리의 주소 및 메모리 쓰기가 요구되므로 최소한 $w\geq 2$ 이고 일반적으로 $p_w\geq 0.95$이므로 식(8)과 같이 SDM에 요구되는 cloock cycle가 IMM에 덜 필요하다는 것을 통해 SDM도 역시 병목지점이 아님을 알 수 있다.

$$\begin{aligned} S^{avg}_{norm_0}-S^{avg}_{norm_5} &\approx \{2w+1+(2+w)\cdot p_w\}-\{6+w\cdot p_w\} \\ &= w+2\cdot p_w-3\geq 0 \end{aligned} \tag{5}$$

읽기 동작은 두 개의 데이터와 한번의 데이터 처리 신호를 필요로 하므로 최소 3 clock, 쓰기 동작도 역시 최소 2 clock가 필요하므로 이를 $S^{avg}_{norm_0}$에 대입하면 $S^{avg}_{norm_0}\geq 5+4\cdot p_w$이 되고 앞서 설명하였듯이 $p_w\geq 0.9$이므로, $S^{avg}_{norm_0}>S^{avg}_{norm_3}$ 임을 통해 MCM도 역시 병목지점이 아님을 알 수 있다.

그러므로 병목지점은 IMM과 FEM이 될 수 있음을 알 수 있으며 적합도 함수를 평가하는데 걸리는 시간, $t_{eval}\geq 5$일 경우 FEM이 병목지점이 되고 그 반대일 경우는 IMM이 병목지점이 될 수 있음을 알 수 있다.

이를 기반으로 유전자 알고리즘 프로세서 내부에서의 효율적인 연산을 위해 병목 부분의 병렬 처리와 빠른 연산 처리를 위해 pipeline를 사용하였으며 전체적인 데이터의 입출력은 handshaking 프로토콜 및 병렬 구조를 기반으로 구성되었다. 즉 문제에 따라 두 개 이상의 적합도 평가모듈을 갖는 병렬 구조로 설계되었으며, 메모리 인터페이스 블록도 두 개의 개체군 데이터를 한 번에 가져올 수 있는 pipeline을 지닌 병렬 구조로 설계되었다.

3.3.2 각 모듈의 VLSI 구현

제안된 유전자 알고리즘 프로세서는 앞서 설명한 효율적인 연산을 위한 병렬 처리와 pipeline 적용 및 handshaking 프로토콜 사용과 더불어 개체군의 크기, 교차 확률, 돌연변이 위치 및 확률, 최대 세대수 및 난수발생기의 초기 값 등의 파라미터 값을 쉽게 변경할 수 있도록 설계하였다. 또 개체의 교차 및 돌연변이 등의 유전 연산자도 적용된 문제 및 진화의 정도에 따라 다양하게 수행될 수 있도록 설계되었다.

각 모듈에서의 handshaking 프로토콜의 사용은, 기존의 pipeline만 적용된 구조보다 더 빠른 clock 속도를 필요로 하고 더 많은 하드웨어 요소를 필요로 한다. 그러나 제안된 구조는 돌연변이와 교차가 부분적으로 동작하지 않을 때, 메모리 쓰기를 수행하지 않을 때, 적합도 평가 모듈이 긴 연산시간을 필요로 할 때, 각 모듈사이에 연산시간이 많은 차이를 보일 때 혹은 내부나 외부 메모리 모듈이 다른 모듈과 인터페이스를 필요로 할 때 효과적이며 전체적인 성능에서는 기존 방법보다 handshaking이 사용된 pipeline 구조가 더 효율적이다.

각 모듈의 구성 및 구현 방법에 대해서는 다음부터 설명된다.

a. *Memory Module*: 유전자 알고리즘 프로세서는 큰 크기의 개체군를 저장하기 위하여 많은 메모리들을 필요로 하기 때문에, 외부 SRAM등의 메모리를 이용한 많은 용량을 필요로 한다. 그러나 40 비트보다 작은 크기의 개체 저장을 위한 개체군의 데이터는 FPGA의 내부 메모리 블록(Embedded Array Logic)으로 구현될 수 있다. 초기 메모리 모듈은 개체에 해당하는 적합도 값과 무작위로 생성된 개체를 저장하며 이들 개체와 적합도 값은 유전자 알고리즘 프로세서가 정상 상태 모델에 기반한 제안된 알고리즘을 실행함으로써 반복적으로 갱신되게 된다.

b. *System Memory Controller*: 시스템의 내부의 연산 모듈과 외부(혹은 내부) 메모리 모듈 사이의 인터페이스 및 데이터의 입출력을 관리한다. 특히, 제안된 메모리 제어 부분은 빠른 데이터의 조작을 위해 한번에 두 개의 개체군 데이터를 가져오도록 설계되었다.

c. *Selection Module*: 선택된 개체들 중에 다음 세대로 진

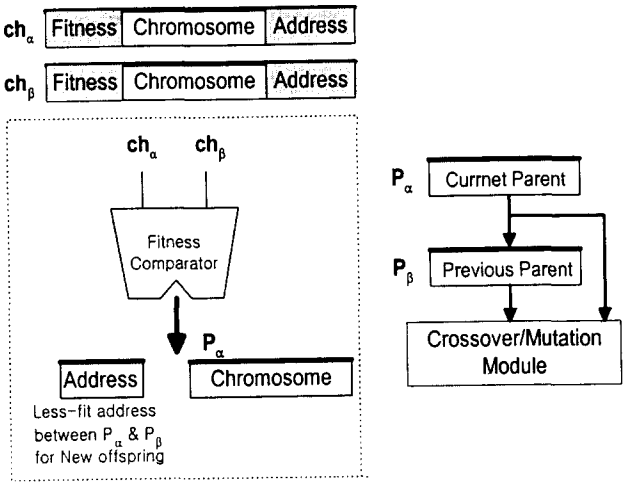


그림 5 두 부모 개체의 선택 메커니즘

Fig. 5 Selection method of two parents

화시킬 대상을 임의로 선택하는 부분으로써 수정된 간단한 토너먼트 선택 방법을 이용하여 기존의 확률선택 모델에 비해 하드웨어 구현시 면적의 감소 및 속도의 향상을 얻을 수 있다. 가장 직접적인 방법인 roulette-wheel 선택 방법의 경우에는 모든 개체에 대해 적합도 값의 덧셈 및 현재 세대에서 모든 개체에 대해 다시 적합도의 덧셈 및 재분류가 필요하며 이것은 추가적인 하드웨어 구성이 필요하며 면적과 속도 두 부분 모두에서 불이익이다.

또한 토너먼트 선택 방법은 개체 선택 방법에 있어 무작위 선택을 하는 survival-based 유전자 알고리즘에 비해 선택 압력을 더 높임으로서 가능한 더 빠르게 최적의 해에 수렴이 가능하도록 설계되었다. 이 방법은 세대 모델에 기반한 알고리즘에서는 성능 향상이 뚜렷하지 않지만, 정상상태 모델에서는 매우 효율적이다. 그림 5는 앞서 설명한 선택 메커니즘을 도식적으로 나타낸 것이다.

d. *Mutation & Crossover Module*: 여러 가지 문제에 대

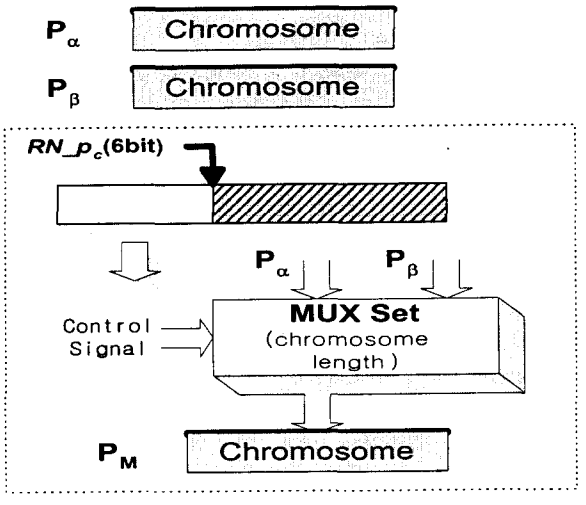


그림 6 단순 교차(한점 교차)

Fig. 6 Simple crossover(1-point crossover)

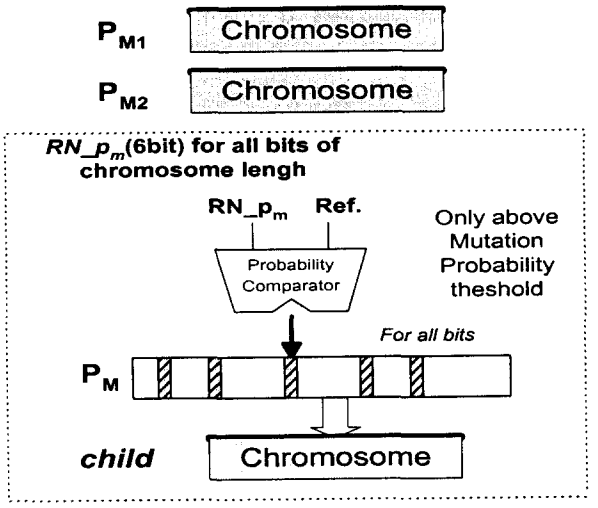


그림 7 스트링에 대한 돌연변이

Fig. 7 Mutation of string (Multi-point mutation)

한 일반성을 위해, 개체의 교차 방법에 있어서는 그림 6에 나타낸 단순 교차(1-point crossover) 및 두 점 교차(2-point crossover)와 균일 교차(uniform crossover) 등의 방법이 모두 수행될 수 있도록 설계되었다. 이를 바탕으로 개체의 생존 결정 모듈에서 결정된 일정한 기간동안에 진화의 정도에 따른 변화를 도입하여, 다양한 해의 생성을 위해 전반부는 균일 교차를 통해 전체 해 공간을 탐색하고, 좋은 해의 유지를 위해 후반부의 두 점 교차를 이용하여 좋은 schema를 보존하는 동적인 적응 교차 조작도 문제의 특성에 따라 사용되었다.

뿐만 아니라, 돌연변이 방법에 있어서도 단순 돌연변이(single-point mutation) 및 그림 7과 같은 스트링에 대한 돌연변이(multi-point mutation)가 발생하도록 설계되었으며 쉬운 문제에 대해 면적은 작고 빠르게 진화할 수 있는 교차와 돌연변이가 결합된 형태의 유전 연산자도 함께 설계되었다.

e. *Fitness Evaluation Module*: 교차와 돌연변이와 같은 유전 연산자에 의해 생성된 새로운 개체에 대해 해의 적합도를 평가하는 부분으로서 주어진 문제에 대한 사상 함수와 평가 함수에 절대적으로 의존적이며, 더 복잡하고 어려운 문제에 대해 적합도 평가 과정의 연산시간이 전제 성능에 지배적인 영향을 미친다. 또한 그 연산시간으로 인하여 병목 현상이 발생하는 부분으로서 효율적인 병렬처리를 위해 적합도 함수에 따라 두 개 이상의 모듈로 구성되어 유전자 알고리즘 프로세서에 사용되었다.

f. *Random Number Generator(RNG)*: 난수 발생기는 개체군의 배열로부터 개체의 선택, 교차의 발생 가능성 및 교차와 돌연변이의 발생 위치를 결정하기 위해 사용되었다. 난수의 평균이 피연산자의 반이 되지 않아서 정확한 계산을 기대하기 어려운 일반적인 LFSR(linear feedback shift register) 방법의 단점을 보완한 cellular automata(CA) 방법을 이용하여 설계되었다.

그림 8에 나타낸 바와 같이 주기 및 초기 값의 문제를 개선하기 위해 4 bit counter를 이용하여 CA의 경계조건을 변화시켜 초기 값에 관계없이 난수를 발생하도록 설계되었으며 maximum length cycle를 증가시켰다. 난수발생기에 사용된

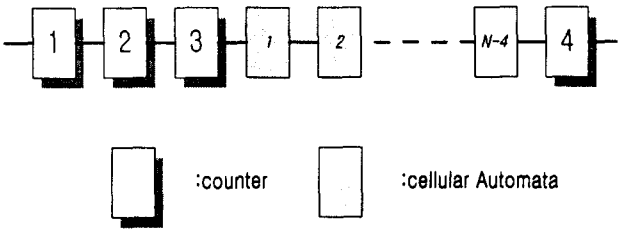


그림 8 난수 발생기의 구조

Fig. 8 Architecture of pseudo-random number generator

CA는 시뮬레이션의 성능에서 rule 150(식(7))과 거의 비슷한 rule 90(식(6))[17]에 의해 설계되었다.

$$a_i(t+1)=a_{i-1}(t)\oplus a_{i+1}(t) \tag{6}$$

$$a_i(t+1)=a_{i-1}(t)\oplus a_i(t)\oplus a_{i+1}(t) \tag{7}$$

g. *Survival Determination Module*: 현재의 개체보다 더 적합한 자손만이 생존하도록 결정하는 부분으로서 모든 개체 중 가장 적합한 개체와 현재의 두 부모 개체 중 적합도가 나쁜 개체 사이의 차이에 따른 개체 생존 조건 변화 및 방법의 변화를 꾀할 수 있도록 설계되었으며 survival-based 유전자 알고리즘에 비해 비교하는 개체의 수를 늘임으로서 빠르게 최적의 값에 도달하도록 구성되었다.

4. 하드웨어 성능 평가

4.1 적합도 함수(Fitness Function)

적합도 함수의 회로는 각 문제에 따라 그 구성이 달라지므로 일반적으로 FPGA와 같이 재구성이 가능한 하드웨어 상에서 구현이 된다. 그러나 제안된 하드웨어는 빠른 검증을 위해 FPGA내의 내부 모듈로서 구현되었다.

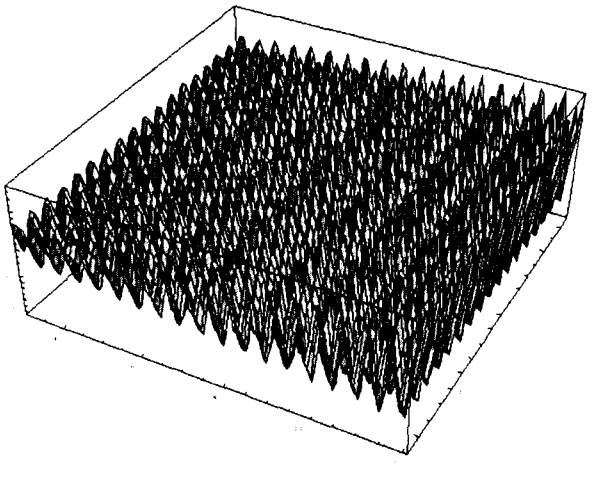


그림 9 유전자 알고리즘 프로세서를 위한 수학적 최적화 적합도 함수

Fig. 9 A mathematical optimization, fitness function for GAP

본 연구에서는 제안된 하드웨어의 파라미터 추출과 동작 특성의 파악 및 성능을 평가하기 위하여 두 가지 적합도 함수 - 수학적 최적화 문제 및 set covering problem를 사용하여 하드웨어로 구현하였다.

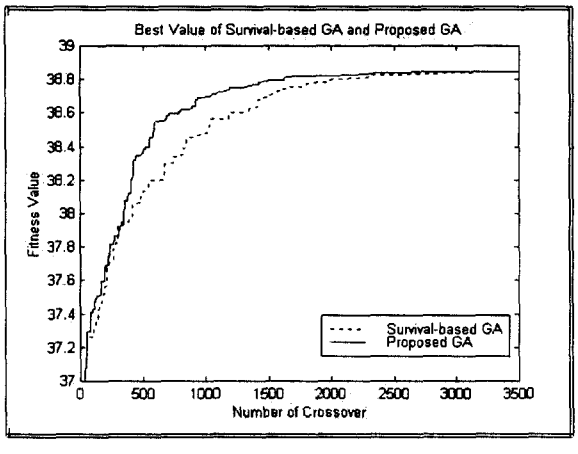
4.1.1 수학적 함수 최적화 문제

첫 번째 적합도 함수는 다음 식 (8)과 같은 최적화 함수에 대해서 $0\le x\le 12.1$와 $4.1\le y\le 5.8$의 범위에서 성능을 검증하였다.

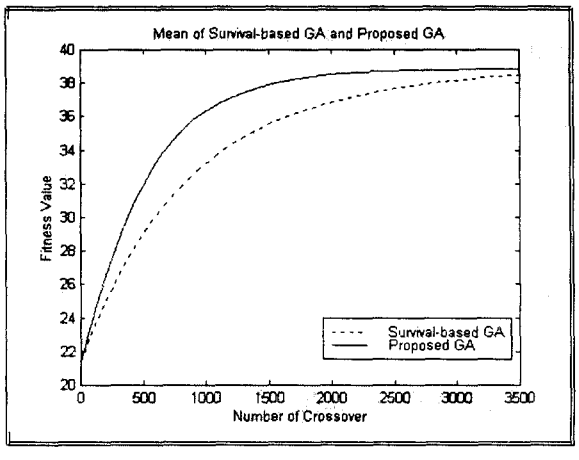
$$f(x,y)=21.5+x\sin(4\pi x)+y\sin(20\pi y) \tag{8}$$

그림 9에서 보여지는 바와 같이, 많은 국부적 수렴지점을 지닌 매우 다루기 힘든 특징을 가지고 있기 때문에 최적의 해를 찾아내기 어렵다.

그림과 같은 적합도 함수에 대해, 유전자 알고리즘의 방법



(a) Best value between GAs



(b) Mean value between GAs

그림 10 수학적 최적화 문제를 이용한 survival-based 알고리즘과 제안된 유전자 알고리즘의 성능비교

Fig. 10 Performance comparison between survival- based GA and proposed GA in mathematical fitness function

간의 성능 비교를 하기 위해 개체군의 크기는 300, 돌연변이 확률 p_m은 0.04의 파라미터와 스트링에 대한 돌연변이 및 두 점 교차의 방법을 이용하여 반복적인 시뮬레이션을 수행하였다. 그 결과로써 그림 10의 (a)와 (b)는 최적의 해를 찾기 위해서, 제안된 알고리즘과 survival-based 알고리즘의 각 세대에서의 최적값과 평균값을 평균화하여 나타낸 것이다. 앞서 2.3절에서 비교한 바와 같이 최적값과 평균 값 모두에서 제안된 알고리즘이 보다 빠르고 정확하게 최적의 해에 근접해감을 알 수 있다. 즉, 주어진 문제에 대해 survival-based 알고리즘은 최적의 해를 찾는데 평균적으로 약 2363번의 교차를 수행하는데 반해, 제안된 방법은 같은 해를 찾는데 약 1603번의 교차만을 필요로 하기 때문에 그 수렴 속도가 기존의 방법에 비해 매우 향상되었음을 알 수 있다.

4.1.2 Set Covering Problem(SCP)

두 번째 적합도 함수는 잘 알려진 조합의 최적화문제로서 일반적으로 문제 푸는 방법이 존재하지 않는 NP 문제의 일종인 non-unicost set covering problem이다. Set covering problem은 m-row, n-column을 가진 zero-one 행렬의 rows를 최소의 비용의 columns를 이용하여 covering 하는 문제이다.[19] 이는 주로 승무원 스케줄링, 비상 설비의 위치 결정, 조립 라인 평형 및 boolean 표현 간략화 등과 같은 많은 실제적인 응용문제에 적용된다. 본 연구에서는 19 rows와 63 columns로 구성된 non-unicost set covering problem이 적용되었고 이는 모든 63 columns를 만족시킬 수 있는 rows의 요소들을 최소 크기의 집합을 찾는 것으로써 적용된 문제의 개념적인 구성의 형태는 그림 11에 나타낸 것과 같다.

VHDL을 이용한 non-unicost set covering problem의 시뮬레이션 결과는 그림 13의 (a)에 보여지고 있다. 그림의 상위 부분은 입출력 신호를 나타내며, 하위 부분은 시뮬레이션

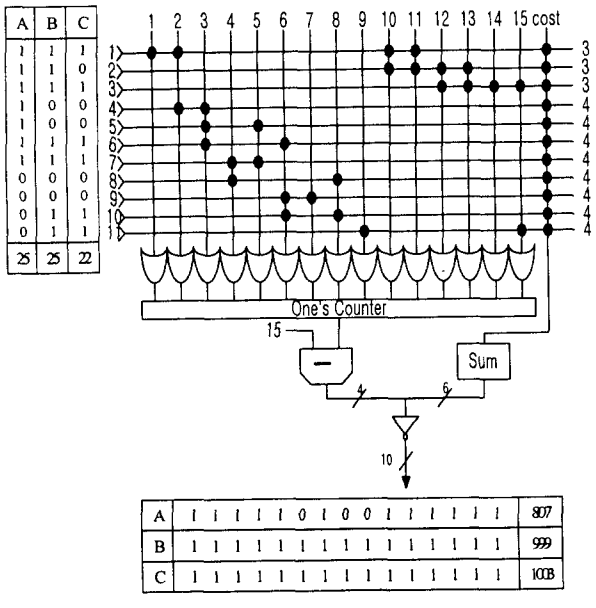


그림 11 Non-unicost set covering problem
Fig. 11 Non-unicost set covering problem

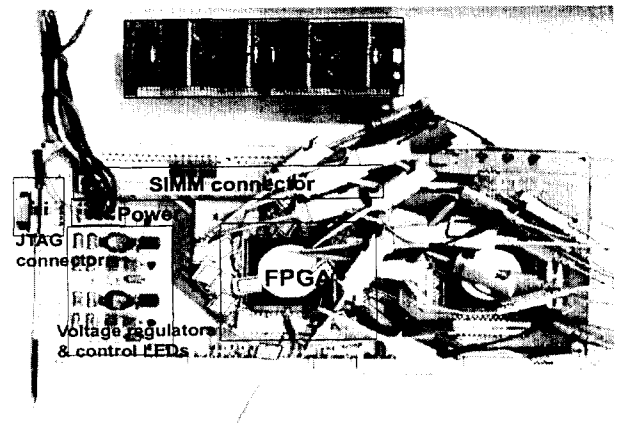


그림 12 제작된 유전자 알고리즘 프로세서의 원형
Fig. 12 The prototype of GAP

된 각 모듈의 상태를 나타낸다. 사용된 개체군의 크기는 128이고, 스트링에 대한 돌연변이에서 각 비트가 변할 돌연변이 확률은 0.008이며 균일 교차는 항상 발생한다. 이들 파라미터를 이용하여 최적의 해를 찾을 때, 반복적인 하드웨어 상에서의 실험 결과, 다른 돌연변이 및 교차의 확률 등의 파라미터와 다른 유전 연산자 방법을 이용할 때의 약 1022번의 교차보다 적은, 평균적으로 약 850번의 교차가 요구된다.

4.2 실험 및 결과

4.2.1 Prototype

제안된 유전자 알고리즘 프로세서의 prototype은 그림 13에서 보여지는 것처럼 하나의 EPF10K100A FPGA와 SRAM 모듈을 가진 PCIGEN10K field programmable circuit 보드 상에 구현되었다. 사용된 EPF10K100A FPGA는 약 100,000개의 게이트, 정확히 말하면 4992개의 로직 요소와 RAM과 ROM을 위한 2,048 bits를 가지는 12개의 내부 메모리 블록(Embedded Array Blocks)으로 구성되어 있다. 또 SRAM 모듈은 40 bits 이상의 개체의 길이를 가지는 population을 위한 외부 메모리로써 128k word×32 bit RAM으로 구성되었다.

제안된 프로세서는 사용된 FPGA의 4992개의 로직 요소 중 72%을 이용하여 구현되었으며 26.7 MHz의 클럭 속도에서 동작한다. 또한 그림 2에서 보여진 각 모듈은 매 8 클럭마다 동작이 수행된다. 이러한 prototype 상에서의 동작은 논리 분석기(HP 1660A)를 이용하여 측정되었다.

4.2.2 결과 및 분석

앞서 언급되었던 바와 같이 기존의 연구는 하드웨어와 소프트웨어 사이의 성능 비교만이 행해져 왔다. 그러나 본 연구에서는 표 6과 같이 제안된 하드웨어와 survival-based 알고리즘을 이용한 하드웨어 사이의 상대적인 비교를 수행하였다. 그러나 두 유전자 알고리즘 프로세서의 비교를 위해서는 적용된 문제와 FPGA의 차이에 따른 성능의 차이가 있기 때

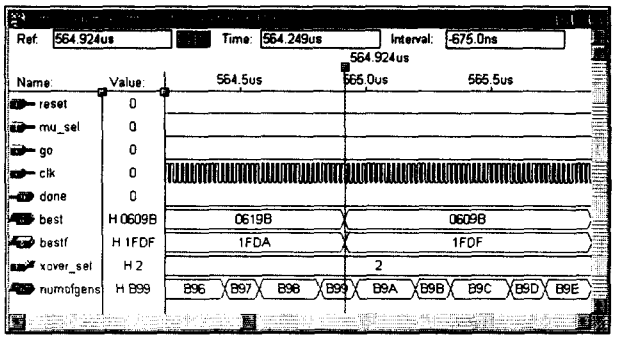
표 6 Prototype 및 시뮬레이션에서의 제안된 방법 및 survival-based 유전자 알고리즘 사이의 연산 시간의 상대적 비교

Table 6 The relative comparison of computation time between survival-based GA and proposed GA on prototype & simulation

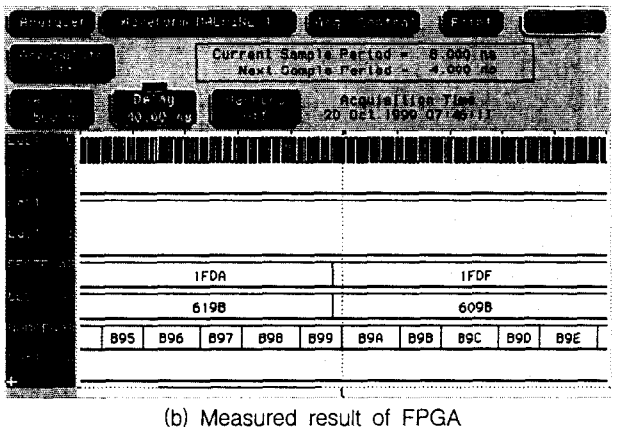
		Survival-based GA[15]		Proposed GA	
Simulation		number of generation	relative ratio	number of generation	relative ratio
Fitness	Set covering problem	760.6	1	512.3	0.674
	Mathematical problem	2363.2	1	1603.9	0.679
Total			1		0.676
Hardware		factor	relative ratio	factor	relative ratio
computation	by application	unicost SCP (94 / 521)	1	non-unicost SCP (19 / 63)	1.42
	by device	EPF81188A	1	EPF10K100A	1.576
computation	(clock rate)	(1MHz)		(26.67MHz)	
	number of crossover per sec.	1 million	1	3.33 million	0.3
Total			1		0.671

문에 정확한 상대적인 성능 비교를 하기 위해서는 위의 차이를 보상하기 위한 두개의 추가적인 인자가 요구된다. 두 prototype은 모두 set covering problem에 적용되었지만, 제안된 프로세서에 적용된 문제는 survival-based 알고리즘에 적용된 문제에 비해 어려운, 선택된 값에 따라 각 row가 cost를 가지는 non-unicost set covering problem이다. Set covering problem는 set의 크기의 증가함에 따라 개체의 길이의 증가, 각 모듈의 연산시간의 증가, set value의 추가적인 증가가 뒤따른다. 하지만 이는 앞서 3.3절에서 설명한 바와 같이 "serially on population and parallel on chromosome"의 하드웨어 구조는 모듈의 pipeline 및 병렬구조를 이용하여 쉽게 확장이 가능하며, set 크기의 증가에 따른 지연 시간의 증가는 주로 적합도 평가 모듈의 연산시간에 영향을 미치게 된다. 따라서 표 6의 비교를 위한 상대적 수치는 적합도 평가 모듈의 구현을 통한 지연시간 및 이를 확장하여 얻어진 값이다. 두 번째로 FPGA의 성능 차이를 보상하기 위해서 프로세서 구현에 사용된 각 모듈을 각각 EPF81188A와 EPF10K100A에 적용하여 실험적으로 얻은 상대적인 지연시간인 1.576으로 속도 차이를 보상하였다.

위의 두 가지 비교인자를 바탕으로 프로세서의 속도 차이를 비교하기 위해서, 적용된 문제의 연산 시간을 초당 수행하는 교차의 수로 나타내었다. 표에 나타난 clock rate의 차이는 각 모듈의 임계경로를 줄임으로써 얻을 수 있는 clock 속도의 차이를 나타내었으며, 두 프로세서를 비교하는데는 영향을 미치지 않도록 하였다. 또 표 6에 나타난 바와 같이, 두



(a) Simulation result of VHDL coding



(b) Measured result of FPGA

그림 13 시뮬레이션 결과와 측정 결과의 비교

Fig. 13 Comparison of simulation result and measured result

가지 적합도 함수에 따른 각 알고리즘에서 최적의 해를 찾는데 걸리는 세대 수의 차이를 전체 연산시간으로 표현하였을 때 시뮬레이션 상에서 제안된 알고리즘이 30%이상의 연산시간의 향상이 있었다.

이에 따라 보고된 survival-based 유전자 알고리즘과 제안된 유전자 알고리즘의 시뮬레이션 및 이에 기반한 하드웨어의 상대적 비교를 하면, 제안된 알고리즘과 프로세서가 FPGA에 따른 속도와 적용된 문제의 차이를 고려하더라도 survival-based 알고리즘과 프로세서와 비교하여 30%이상의 연산시간의 향상이 있었다.

5. 결 론

다양한 문제의 실제 적용을 위해, 새로운 하드웨어 지향의 알고리즘이 유전자 알고리즘 프로세서의 하드웨어적 요소 및 실행시간을 줄일 수 방법으로써 제안되었다. 제안된 알고리즘은 새로 생성된 자손의 적합도가 열성 형질을 지닌 부모 개체의 적합도보다 향상되었을 때 개체군의 갱신이 결정되는 개체 생존 방법과 수정된 토너먼트 선택 방법에 기초한 정상상태 모델이 채택되었다. 즉, 이산세대 모델에 비해 개체군의 저장을 위해 사용되는 메모리를 반으로 줄일 수 있는 개선된 정상상태 모델이 적용되었고, 개체의 생존 방법은 개체의 선택 방법 및 보존 방법을 변화 시켰다. VHDL 설계에 의한 제안된 알고리즘의 FPGA 구현은 연산 시간에 있어 기존의 가장 우수한 survival-based 유전자 알고리즘의 초당 백만 번

적으로 30% 이상의 속도 향상을 얻을 수 있었다. 덧붙여, 여러 가지 문제에 대한 보편성을 위해 1-point, 2-point 및 균일 교차와 다양한 돌연변이를 통해 다양한 해의 생성, 유지 및 빠른 해의 도달을 하도록 구현되었다.

연구된 유전자 알고리즘 프로세서는 빠른 연산 시간을 바탕으로 차후의 진화형 하드웨어의 중앙연산처리 장치 및 실시간 처리가 요구되는 최적화 문제, Robot Control, 음성 및 문자인식, 컴퓨터 비전 분야 등의 다양한 문제의 최적의 해를 얻기 위한 도구로써 수많은 응용 분야를 갖는다.

참 고 문 헌

[1] 장병탁, "유전 알고리즘 이론 및 응용", 전자공학회지, 제22권 제11호, pp. 60-69, 1995. 11.

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

[3] G. Winter et al., Genetic Algorithms in Engineering and Computer Science, John wiley & Sons, 1996.

[4] G. Sysweda, "Uniform Crossover in Genetic Algorithm", Proc. of ICGA-89, 1989.

[5] K. Dejong, An analysis of the behavior of a class of genetic adaptive system, Ph.D Thesis, University of Michigan, 1975.

[6] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware", IEEE Transactions on Systems, Man & Cybernetics Part C: Applications & Reviews 1997., V.29 pp. 87-97 N.1, 1999

[7] T. Higuchi et al., "Evolvable hardware and its applications to pattern recognition and fault-tolerant systems," in Towards Evolvable Hardware: The Evolutionary Engineering Approach, Lecture Notes in Computer Science, Vol. 1062, pp. 118-135., 1996.

[8] S. D. Scott, A. Samal and S. Seth, "HGA : A hardware-based genetic algorithm", Proc. ACM/SIMDA 3rd International Symposium on FPGA, pp. 53-59., 1995.

[9] P. Graham, B. Nelson, "A hardware genetic algorithm for the traveling salesman problem on Splash2", 5th International Workshop on Field-Programmable Logic and its Applications, pp. 352-361., August 1995.

[10] M. Salami, "Multiple genetic algorithm processor for hardware optimization", Proc. First International Conference, ICES96 Evolvable System: From Biology to Hardware, pp. 249-259., October 1996.

[11] M. Tommiska, J. Vuori, "Implementation of genetic algorithms with programmable logic devices", Proceeding of the 2NWGA, August 1996.

[12] Barry Shackleford et al., "A High-Performance Hardware Implementation of a Survival-Based Genetic Algorithm", ICONIP'97 pp. 686-691., Nov. 1997.

[13] I. Kajitani, et al., "A gate-level EHW chip: Implementing GA operations and reconfigurable hardware on a single LSI", Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science 1478, pp. 1-12., Springer Verlag, 1998.

[14] N. Yoshida, T. Moriki and T. Yasuoka, "GAP: Genetic VLSI processor for genetic algorithm", Second International ICSC Symp. on Soft Computing, pp. 341-345., 1997.

[15] Shin'ichi Wakabayashi et al., "GAA: A VLSI genetic algorithm accelerator with on-the-fly adaptation of crossover operators", ISCAS 98, 1998.

[16] Dan Mihaila, Florin Fagarasan, Mircea Gh. Negoita, "Architectural Implications of Genetic Algorithms Complexity in Evolvable Hardware Implementation" EUFIT'96, Vol. 1, pp. 400-404 September 1996.

[17] Peter D. hortensius et al., "Parallel Random Number Generation for VLSI System Using Celluar Automata", IEEE Trans. on Computers. Vol. 38. No. 10 pp. 1466-1473 October 1989.

[18] P. Kenyon er al., "Programming pipelined CAD applications on message architectures", concurrency practice and Experience, vol. 7, no. 4, pp. 315-337, June 1995.

[19] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem", EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, pp. 392-404, Vol. 94, No. 2, October 1996.

저 자 소 개

김 진 정 (金 鎭 貞)

1998년 인하대 전자재료공학과 졸업. 2000년 동 대학원 전자재료공학과 석사과정. 현재 LG종합기술원 연구원.

Tel : 032-874-1663, FAX : 032-875-5882

E-mail : ulsi98@unitel.co.kr

정 덕 진 (鄭 德 鎭)

1948년 2월 8일 생. 1970년 서울대 전기공학과 졸업. 1984년 미국 Utah State University (석사) (Electrical Eng.) 1988년 미국 University of Utah (공박) (Electrical Eng.). 1989년~인하대 전자재료공학과 교수

Tel : 032-860-7435, FAX: 032-875-5882

E-mail : djchung@dragon.inha.ac.kr