Special Thema

FPGA로 살펴보는 진화 하드웨어

심귀보 교수
(중앙대 전자전기공학부)

'매트릭스' 란 영화를 보면 기계(컴퓨터)가 지배하는 세상이 나온다. 이 세상에서 태양은 이미 사라졌고 기계는 자신의 에너지를 공급받기 위해 인간을 '재배' 한다. 여기서 기계들은 어떻게 사람의 힘없이 자기 스스로 그러한 시스템을 만들고 인간을 지배하는 힘을 가졌을까? 아마도 기계 환경의 변화에 스스로를 적응 • 진화함으로써 그와 같은 시스템이 구축되지 않았을까 생각된다. 기계에서의 진화는 자연계생물체에서 일어났던 것과 같이 그리 긴 시간을 요구하지 않았을 것이다.

1. 진화 하드웨어에 숨겨진 비밀

진화라고 하는 것은 기본적으로 유전적인 변화를 수반하는 자기복제의 과정이다. 따라서 소프트웨어 진화는 기계 언어로 된 자기복제 프로그램을 돌연변이를 만들어내는 기구를 가진 컴퓨터내의 가상세계에서 실행함으로써 이뤄지는 것이다. 그렇다면 하드웨어를 진화시키기 위해서는 어떠한 요건이 갖춰져야 할까? 소프트웨어 진화에서와 같이 자신을 복제하고 유전적 변이를 가할 수 있는 구조의 하드웨어가 필요할 것이다. 이러한 하드웨어로 재구성 가능한 하드웨어(Reconfigurable Hardware)가 있다. 재구성 가능한 하드웨어는 사용자가 소프트웨어적으로 구조를 변경할 수 있는 반도체 집적회로로, 반도체 제조공정을 거치지 않고 다양한 구조의 반도체를 구현할 수 있기 때문에 많은 분야에서 응용되고 있다.

또한 이것은 환경의 변화에 적응하고 결함에도 견고한 하드웨어 시스템을 구축할 수 있는 길을 열어주고 있다. 재구성 가능한 하드웨어의 대표적인 예는 FPGA(Field Programmable Gate Array)로, 이는 하드웨어 내부의 구성을 결정하는 비트스트링을 다운로드함으로써 임의의 하드웨어 기능을 구현할 수 있다. 환경이 변함에 따라 하드웨어를 재구성하고 적응적으로 수행할 수 있는 기술로 현재 가장 많은 사람들로부터 주목받고 있는 것 중 한 가지가 진화 알고리즘(Evolutionary Algorithm)이다. 진화 알고리즘에 의해 그 구조가 자동으로 변하는 하드웨어를 '진화 하드웨어' 라고 한다. 하드웨어 구조를 결정하는 비트스트링이 있고 이것을 바꿔 씀으로써 여러 논리회로를 구현할 수 있는데, 이것은 유전자 알고리즘에서 해의

후보를 이진 비트스트링으로 나타내고 이것을 탐색해 최적의 해를 발견하는 방식을 적용할 수 있게 만든다. 적응적 설계 방법은 FPGA의 구조 결정 비트스트링을 유전자 알고리즘에 있어 염색체로 생각하고 환경에 가장 적합한 비트스트링을 찾는 방법이다.

한편 자연계의 각종 생물은 진화 과정을 통해 환경에 적응하도록 그 형태나 구조를 변화시킨다. 어떤 프로그램을 변경해 새로운 기능을 갖도록 하는 소프트웨어의 진화만이 아니라 형태나 구조 즉, 하드웨어를 진화시키는 것도 자율성 • 창조성이 풍부한 정보처리시스템의 창출에 있어 중요한 연구 과제다. 소프트웨어 진화와 유사하게 표현하면 하드웨어 진화는 환경의 변화나 오차를 이용해 전자회로인 하드웨어가 그 구조를 자율적으로 바꿔, 구조는 물론 기능까지 복잡화하고 다양화하는 것이다. 따라서 그 표현형이 프로그램이 아니라 전자회로인 점을 제외하면, 방법론적으로는 소프트웨어 진화와 기본적으로 차이는 없다. 단 회로소자 그 자체는 진화하지 않지만 예를 들어 셀룰러 오토마타(CA: Cellular Automata, 우리말로 번역하자면 세포 자동차쯤으로 번역되나 일반적으로 약어로 많이 표현되므로 본 글에서는 발음 그대로 셀룰러 오토마타라 표현한다)나 FPGA 등 소자의 결선이나 조합이 재구성 가능한 구조를 가진 하드웨어 디바이스를 전제로 한다.

2. 진화 하드웨어의 특징

궁극적으로 하드웨어 진화는 변화하는 정보에 의존해 하드웨어의 구조를 자율적으로 바꾸는 것을 말한다. 정보를 종자로, 특별한 하드웨어 소자 기반을 밭에 각각 비유하면 종자에 대한 하드웨어 구조(전자회로 등)를 밭에서 만들어내고, 목적에 맞도록 종자에서 회로를 성장시키는 것을 반복하면서 서서히 품종을 개량해 나가는 것에 해당한다(그림1). 따라서 다음과 같은 특징을 갖고 있다.

(1) 종자는 같아도 밭이 다르면 만들어지는 하드웨어 구조가 다르다.

(2) 밭은 다소의 차이(디바이스의 구조결함이나 에러)가 있어도 좋다.

(3) 구조결함, 동적인 잡음, 에러 등을 정보의 변화로서 흡수할 수 있다.

앞의 글만으로 과연 하드웨어 진화가 어떻게 가능할 것인지에 대한 의문은 풀리지 않을 것이다. 이제 그 가능성을 조금씩 높여가 보자.

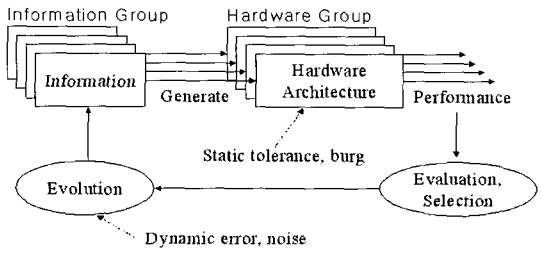


그림 1. 하드웨어진화의 기본적인 개념.

3. 재구성이 가능한 반도체 소자 "FPGA"

컴퓨터 개발과 함께 컴퓨터상에서 소프트웨어(프로그램)를 진화시키는 방법이 연구돼 왔다. 그 대표적인 것이 유전자 알고리즘이라 할 수 있다. 소프트웨어가 아닌 하드웨어 진화의 개념을 생각하지 못한 것은 아니지만 최근 하드웨어를 진화할 수 있는 기술적 기반이 마련되면서 진화 하드웨어에 대한 연구가 본격적으로 시작됐다. 진화 하드웨어의 연구를 가능하게 한 것은 바로 재구성 가능한 반도체 소자인 FPGA의 개발 덕택이다. 진화 하드웨어에 대한 연구는 1992년부터 스위스와 일본에서 톰슨(Thompson)과 히구찌(Higuchi) 등의 연구자에 의해 독립적으로 시작됐다. 시간이 지남에 따라 이에 대한 관심이 증대되면서 1995년 첫 번째 국제 워크샵이 스위스 로잔에서 개최됐다. 그 이후로 미국 • 일본 • 영국 등의 연구자들에 의해 활발하게 연구되고 있다.

초기에 진화 하드웨어는 진화된 회로의 크기가 작다는 공통적인 문제들을 갖고 있었다. 하드웨어

진화는 AND 게이트와 OR 게이트와 같은 기초적인 게이트를 기본으로 하고 있었다. 이런 진화 상태를 게이트 레벨(Gate-level) 진화라고 불렀는데, 이 진화는 산업적인 응용에서는 크게 효용이 없었다. 그러다 비약적인 발전을 할 수 있는 계기가 있었는데, 바로 기능이 향상된 FPGA의 등장이었다. 이로서 게이트 수준에서 벗어나 기능 레벨(Function-level)의 새로운 형태의 하드웨어 진화를 연구할 수 있게 됐고, 그 응용 범위 또한 넓힐 수 있었다.

3.1 진화 하드웨어의 핵심 "FPGA"

진화 하드웨어를 구현하는 데 있어 하드웨어의 유연성(Flexibility)은 매우 중요하다. 이것을 적용하기 위해서는 하드웨어가 하나의 셀(Cell) 단위로 이뤄져서 협조 행동이 이뤄지도록 유도해야 한다. 현재 진행되고 있는 진화 하드웨어의 적용은 대부분 FPGA로 이뤄지고 있다. 그러므로 진화 하드웨어의 구현을 이해하기 위해서는 FPGA의 구조가 어떻게 이뤄져 있는가를 알아야 한다.

FPGA는 PLD(Programmable Logic Device)의 일종으로, 사용자가 원하는 기능을 수행하기 위해 회로를 프로그램에 의해서 구성할 수 있도록 하는 것이다. PLD는 60년대 중반 해리스라는 사람에 의해 처음으로 만들어졌다. 초기 모델은 다이오드와 퓨즈(Fuse)를 사용해 만들어졌고, 이후 IBM, 제너럴 일레트릭, National, Intersil, Signetics에 의해 개발됐으나 1978년 MMI(Monolithic Memories)에 의해 처음으로 상용화됐다. PLD는 다음과 같은 세 가지로 분류할 수 있다.

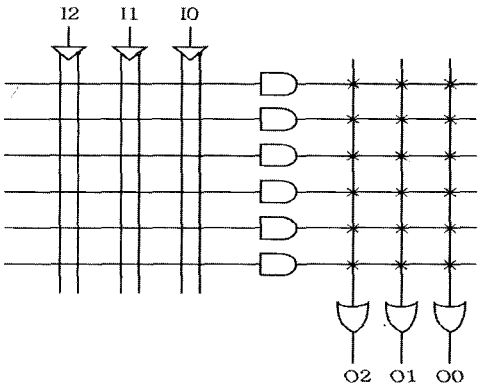


그림 2. PROM의 구조.

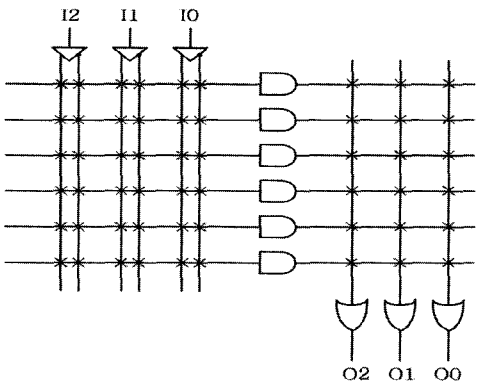


그림 3. PAL의 구조.

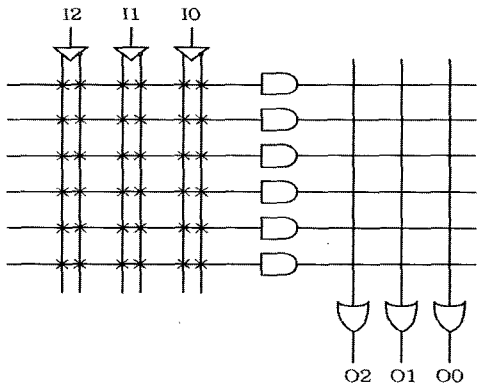


그림 4. PLA의 구조.

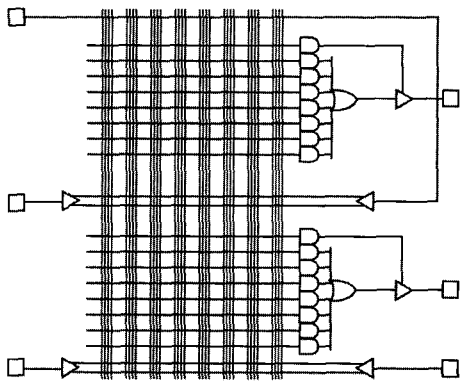


그림 5. 출력단에 MUX를 단 형태.

그림2처럼 OR 게이트가 연결돼 있는 출력 쪽을 프로그래밍하면 PROM, AND 게이트가 연결돼 있는 입력쪽을 프로그래밍하면 PAL(그림3), 둘 다 프로그래밍하면 PLA(그림4)가 된다. 유연성 면에서 PAL이 PLA보다 많이 사용됐다. 그리고 이를 더욱 개선하기 위해 출력 쪽에 멀티플렉서를 사용해 2차 구조를 만들고(그림5) 순차회로의 구현을 쉽게 하기 위해 플립플롭을 덧붙였다(그림6).

이로써 여러 논리 회로를 만들 수 있다. 이것은 5~15개의 표준 TTL 회로를 대체할 수 있다. 하지만 이 회로는 다층의 논리회로는 구성하지 못하고, 이미 구성된 회로를 다시 사용할 수 없었다. 게다가 초기에는 퓨즈 방식으로 프로그래밍 했기 때문에 한번 프로그래밍하면 고칠 수 없어 많은 비용이 들었다. 그 이후 많은 개선을 통해 CPLD와 FPGA가 만들어지게 됐다. 이 회로는 그림6과 같은 블럭이 여러 개

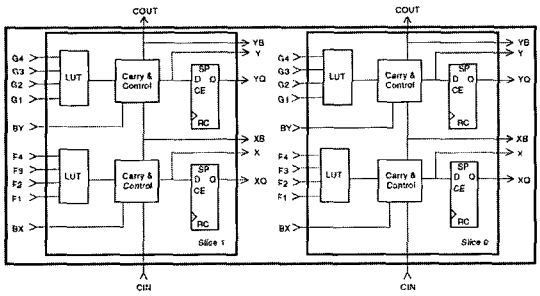


그림 6. CLB의 구조.

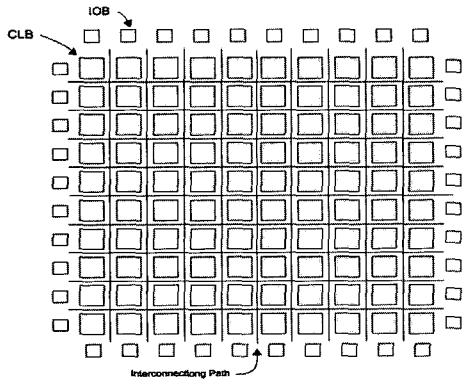


그림 7. FPGA 전체 구성도.

놓여져 있는 형태이다. 또한 플래시메모리나 SRAM을 사용해 만들었기 때문에 새로 구성할 수 있다.

마크로 셀(Macro Cell)로 만든 것이 CPLD이고, CLB(Configurable Logic Block)로 구성한 것이 FPGA다. 마크로 셀을 구성하는 로직의 규모가 FPGA의 로직 유닛에 비해 매우 큰 반면 하나의 마크로 셀을 거치는 데 소요되는 로직 딜레이가 일정한 특징을 갖기 때문에 타이밍에 대한 예측이 쉽지만, 연결해서 프로그래밍하는 데 한계가 있어 게이트 수가 증가하면 FPGA를 사용하게 된다. 현재 우리나라에서는 알테라와 자일링스 칩을 가장 많이 사용하고 있다. 그 구조는 제품에 따라 약간씩 차이가 나지만 비슷하다. 여기서는 자일링스의 Vertex-E의 구조를 살펴보자. 하나의 CLB는 그림7과 같이 구성돼 있는데 CLB는 두 개의 슬라이스로 구성돼 있고, 슬라이스는 또 다시 두 개의 LUT(Look-Up Table)로 구성돼 있다. LUT는 조합 회로를 작성하는 부분으로 Shifter도 LUT로 구성할 수 있다.

LUT는 4 입력과 1 출력을 갖고, CLB는 전체적으로 16 입력 1 출력을 갖게 된다. 이런 CLB가 여러 개로 나열돼 있다. 그림8과 같이 각각의 CLB가 나열돼 있고 CLB는 인터커넥션 패스에 연결돼 있다. 그리고 그 패스들은 다시 IOB(Input/Output Block)에 연결돼 있다. 그림8과 같이 CLB들은 출력이 각 라인에 연결돼 있고, 이 라인들은 다시 PSM(Programmable Switch Matrices)에 연결돼 있는데, PSM이 각 CLB들을 복잡한 논리 회로로 구성할 수 있도록 Path를 연결한다. 그리고 주변부에 CLB들은 IOB와 연결돼 있어 외부 회로와 연결할 수 있도록 돕는다. 이로써 FPGA는 몇 만 게이트가 되는 매우 복잡한 논리회로를 구성할 수 있는 것이다.

또한 그림 9에서 보듯이 8bit Adder를 구성하더라

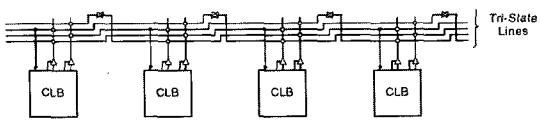


그림 8. 인터커넥션 path와 CLB의 연결.

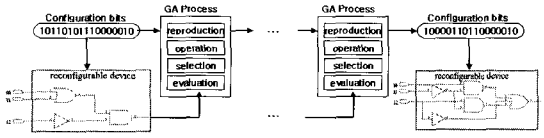


그림 9. 게이트레벨 하드웨어 진화의 개념도.

도 전체 FPGA칩의 일부분을 차지할 뿐이다(8bit Adder는 하나의 CLB로 만들 수 있다).

진화 하드웨어에 FPGA를 사용하는 것은 이처럼 하드웨어를 프로그래밍으로 구성해 하드웨어의 구성 요소를 비휘발성 플래시 메모리 셀(Nonvolatile Flash Memory Cell)을 사용해 전원이 공급되지 않아도 프로그래밍된 상태를 유지한다. 그러나 SRAM으로 구성돼 있는 FPGA는 전원이 차단된 경우 새로 프로그래밍 해야만 한다. 그러나 실제 시스템에 장착된 FPGA의 경우 전원이 공급되면 자동으로 정보를 다시 저장하도록 외부에 ROM이나 플래시 메모리를 다시 설치한다. FPGA에 회로를 설계하기 위해 스키마틱 편집기(Schematic Editor)를 이용한다. 이것은 각 제품이나 툴마다 다른 포맷을 갖기 때문에 제품이 바뀌면 완전히 새로 회로를 구성해야만 한다. 하지만 HDL(Hardward Description Language)을 이용해 논리 회로를 프로그램처럼 기술하는 것이 가능하게 됐다. 물론 HDL에도 VHDL • HDL • Erilog • BEL 등 여러 가지가 있지만 우리나라에서는 VHDL과 Verilog를 사용한다. FPGA의 기능과 구조는 더 많고 더 복잡하지만, 이 글에서는 이정도로 마치도록 한다. FPGA에 대해 더욱 자세한 자료는 알테라(www.altera.com)나 자일링스(www.xilinx.com)를 참고하기 바란다.

3.2 Place and Route 과정

진화 하드웨어에 FPGA를 사용하는 것은 이처럼 하드웨어 구조를 소프트웨어적으로 변경할 수 있는 유연성(flexibility)이 있기 때문이다. FPGA에 프로그래밍하는 것은 HDL로 프로그래밍 한 다음, 이것을 논리 회로로 구성해 다시 각 CLB와 인터커넥팅 라인에 구성할 수 있도록 Place & Route(이하 P&R)라는 과정을 거친다. 이 과정을 통해 각 회로들은 위치를 잡게 되는데, 이 위치가 어디냐에 따라 속도에 현저한 차이가 날 수도 있다. 이 문제는 프로그래밍하는 툴에서 최적화시켜주기 때문에 큰 문제가 없다(그림9). 하지만 진화 알고리즘을 사용할 경우에는 진화를 통해 회로의 위치나 구조 등을 최적화한다. 간단한 예로 설명하면 그림9에 나오는 많은 CLB에 진화 알고리즘을 사용해 위치를 정하는 방법을 통해 CLB의 개수나 가장 짧은 거리의 위치를 설계하지 않고 찾아낼 수 있다.

3.3 진화하드웨어 구현방법

3.3.1 게이트레벨 진화하드웨어

FPGA는 비트스트링을 가해줌으로써 내부의 회로 구조가 결정된다. 즉, 가해진 트스트링이 달라지면 내부 구조 또한 달라지는 것이다. 게이트 레벨 진화에서는 FPGA의 구조결정 비트스트링을 유전자 알고리즘의 염색체로 이용한다. 이 염색체는 FPGA의 회로 구조를 결정하고 환경으로부터 평가를 받는다. 이 평가 결과를 토대로 선택과 유전자 연산을 수행해 다음 세대의 개체군을 형성한다. 세대를 반복하면서 환경에 대해 최적의 결과를 내는 하드웨어 구조가 생성된다. 게이트 레벨 진화 하드웨어는 명시적인 설계 없이 이론적으로 어떠한 회로도 구성할 수 있다는 장점이 있는 반면 회로가 커질수록 구조결정 비트스트링이 커지기 때문에 진화 시간이 오래 걸린다는 단점을 갖고 있다. 따라서 '게이트 레벨' 진화 하드웨어는 작은 규모의 새로운 회로를 설계하는데 적용된다. 또한 다음에 설명하는 '기능 레벨' 진화 하드웨어의 기능(Function) 모듈을 설계하는 데에도 적용된다.

3.3.2 기능레벨 진화하드웨어

그림10은 기능 레벨 진화를 구현하기 위한 모델을 나타낸다. 이것은 게이트 레벨 진화 방식에 비해 더 크고 복잡한 회로를 진화할 수 있는 방법이다. 일단 기능 모듈 설계나 게이트 레벨 진화를 통해 구현한다. 그림10의 예에서는 PFU(Programmable Floating Processing Unit)가 기능 모듈이 된다. 그 다음 진화 알고리즘을 통해 이 기능 모듈의 연결 구조를 결정한다. 즉, 진화알고리즘으로 기능 모듈 사이의 결선이나 배치 등을 결정함으로써 시스템을 구성하는 방

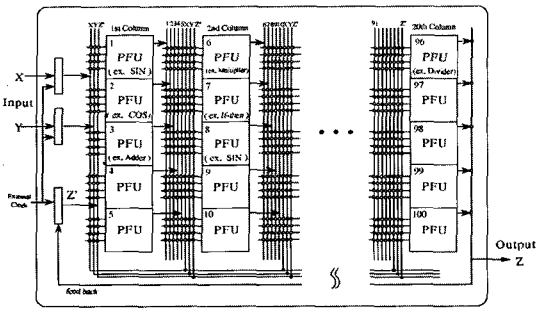


그림 10. 기능레벨 진화하드웨어.

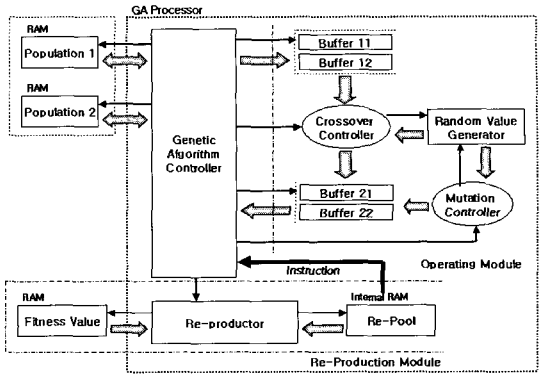


그림 11. 유전자 알고리즘 프로세서의 블록 다이어그램.

식이다. 이 방식은 한 구조에서 동일한 기능을 가진 모듈이 많이 사용될 때 그리고 복잡한 구조의 회로를 설계할 때 사용하는 방식이다. 또한 기능 레벨 진화 하드웨어는 20개의 열로 구성돼 있고, 각 열에는 5개의 PFU를 포함하고 있다. 특히 각각의 PFU는 Adder, Subtracter, If-then, Sine Generator, Cosine Generator, Multiplier, Divider의 7가지 기능 중에 한 가지를 수행한다. PFU의 기능 선택은 진화 과정에서 만들어진 염색체에 의해 선택되고, 각 열들은 크로스바 스위치(Crossbar Switch)에 의해 연결된다. 크로스바는 PFU의 입력을 결정한다.

3.3.3 언어레벨 진화 하드웨어

언어 레벨(Language-level) 진화 하드웨어는 코자에 의해 수행된 방식으로 하드웨어를 설계하는 언어 즉, HDL(또는 SPICE)을 이용해 하드웨어를 진화시킨다. 이 방식은 HDL을 염색체로 하고 여기서 생성된 비트스트링으로 하드웨어의 구조를 결정한다. 이 방식은 범용 컴퓨터와 설계 언어(HDL 또는 SPICE)를 사용하고 적합도 평가 과정이 종종 소프트웨어상에서 이뤄지기 때문에 소프트웨어 진화와 유사하지만 결과물이 하드웨어라는 점 때문에 진화 하드웨어로 구분된다. 이러한 진화 방식을 외부적(Extrinsic) 진화라고 한다.

3.4 GAP(유전자 알고리즘 프로세스)

진화 하드웨어와 관련된 중요한 분야 중 하나인 GAP(Genetic Algorithm Processor)는 말 그대로 유전자 알고리즘을 범용 컴퓨터가 아닌 하드웨어(칩)

Op-Code	Parent 1 ID	Parent 2 ID	Direction1	Direction2

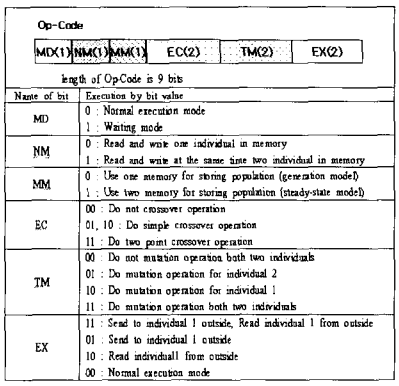
그림 12. 명령어 포맷.

에서 수행하기 위해 개발된 프로세서다. GAP는 진화 연산을 하드웨어상에서 수행함으로써 연산 시간을 줄이며, 범용 컴퓨터에 의한 하드웨어 진화가 아닌 하드웨어 자체에 의한 하드웨어 진화를 가능하게 한다. 그림11의 GAP 내에서는 유전자 알고리즘의 기본 연산자인 교차와 돌연변이 기능을 하는 연산 모듈(Operating Module)과 개체군의 재생산을 위한 재생산 모듈(Re-production Module), 개체군을 저장하는 두 개의 외부 메모리, 적합도 값을 저장하는 한 개의 외부 메모리, 그리고 전체 기능을 제어하는 제어기 (Controller)로 이뤄져 있다. 재생산 모듈(Re-Production Module)에서는 적합도 값을 저장하고 있는 메모리에서 적합도 값을 참조해, 룰렛 선택을 통해 재생될 개체들을 재생 공간(Re-Pool)에 저장한다.

이때 그림12와 같이 교차 연산을 하게 될 두 부모 개체의 ID(메모리 내에 저장된 위치를 가리키는 값)에 연산 코드(Op-Code)와 교차 후와 돌연변이 후 자손들이 저장될 위치를 지정해 주는 방향 코드(Direction-Code)를 붙여 하나의 명령어(Instruction)를 만들어 저장한다. 연산 코드는 연산 수행의 유무,

데이터 전송의 유무를 가리키는 두 개의 비트로 구성돼 있고, ID와 방향 코드는 개체가 메모리 내에 저장돼 있거나 저장될 상대적 위치 값을 표시한다. 이런 방식으로 다음 세대에 재생산될 개체들을 명령어 형태로 만들어 저장해 놓으면, 제어기(Controller)에서 이 명령어를 읽어 들여 실행하게 된다. 두 개의 개체 ID를 참조해 메모리에서 두 개체를 읽어 들이고, 연산 코드를 참조해 두 개체의 교차와 돌연변이 연산을 수행한다. 그리고 연산후 생성된 자손은 방향 코드(Direction1, 2)를 참조해 다른 메모리에 차례로 저장하게 된다. 그림13은 그림12의 연산 코드의 명령어를 상세하게 보여주고 있다.

연산 모듈(Operating Module)에서는 유전자알고리즘의 기본 연산인 교차와 돌연변이를 수행하게 된다. 그림14는 연산 모듈의 내부 구조를 나타낸다. 그



Name of bit	Execution by bit value
MD	0 : Normal execution mode 1 : Waiting mode
NM	0 : Read and write one individual in memory 1 : Read and write at the same time two individual in memory
MM	0 : Use one memory for storing population (generation model) 1 : Use two memory for storing population (steady-state model)
EC	00 : Do not crossover operation 01, 10 : Do simple crossover operation 11 : Do two point crossover operation
TM	00 : Do not mutation operation both two individuals 01 : Do mutation operation for individual 2 10 : Do mutation operation for individual 1 11 : Do mutation operation both two individuals
EX	11 : Send to individual 1 outside, Read individual 1 from outside 01 : Send to individual 1 outside 10 : Read individual1 from outside 00 : Normal execution mode

그림 13. Op-code 포맷.

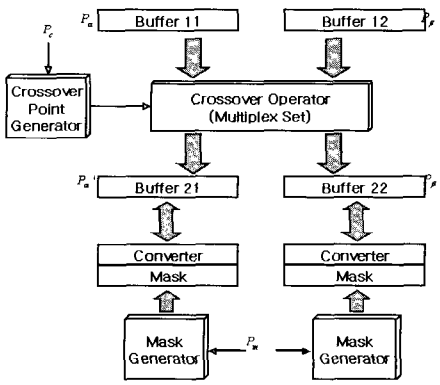


그림 14. 오퍼레이팅 모듈의 내부구조.

리고 선택된 두 부모 개체가 교차 직후 돌연변이를 수행해 빠른 처리 속도를 가질 수 있다. 교차 연산은 단순 교차 연산을 한다. 먼저 교차 연산을 수행하게 될 두 부모 개체를 메모리에서 읽어 들어 버퍼 11과 버퍼 12에 저장한다. 동시에 교차점 생성기(Crossover Point Gene Rator)에서는 0과 1사이의 난수를 발생시켜 그 수가 미리 정해진 교차확률보다 크면 교차점을 발생하고, 그렇지 않으면 교차점을 0으로 설정한다. 교차점은 1과 염색체의 길이보다 작으며, 1보다 작은 사이의 정수 값을 난수로 발생시킨다. 교차 연산기(Crossover Operator)에서는 두 부모 개체와 교차점을 참조해 교차연산을 수행하고, 연산 후 생산된 자손을 버퍼 21과 버퍼 22에 저장한다. 돌연변이 연산은 점 돌연변이 연산을 한다. 교차 연산 후 자손이 버퍼 21, 버퍼 22에 저장됨과 동시에, 마스크 발생기(Mask Generator)에서 0과 1사이의 난수를 발생시켜 그 수가 돌연변이 확률보다 크면 마스크 비트(Mask Bit)를 1로 설정하고, 그렇지 않으면 0으로 설정한다. 이 때 염색체의 길이만큼 마스크 비트를 설정해 마스크를 생성한다. 변환기(Converter)에서는 마스크를 참조해 염색체의 해당 위치의 마스크 비트가 1이면 변환하고, 0이면 그대로 둬 돌연변이 연산을 수행한다. 돌연변이연산 후 자손은 다시 버퍼 21, 버퍼 22에 저장된다.

4. 진화하드웨어 적용사례

4.1 진화형 우주시스템

진화 하드웨어를 이용하면 우주선의 생명력을 늘리기 위한 두 가지 이점을 얻을 수 있다. 첫 번째 이점은 갖고 있던 기능들의 오동작, 온도 문제, 생존 기간의 대처 동작에 대한 유지 보수가 가능하다는 것이다. 두 번째는 새로운 임무들에 대해 필요한 새 기능들을 만들어 낼 수 있다는 것이다. 진화 하드웨어는 잘못으로 인해 망가진 기능을 피해 새로운 회로를 재구성함으로써 회복할 수 있다. 회로가 작동하는 한계값을 정한 후에 하드웨어의 수행값이 한계 수행값보다 작으면, 진화 하드웨어는 전의 회로 구성을 기억하면서 새로운 회로 구성을 찾아 진화한

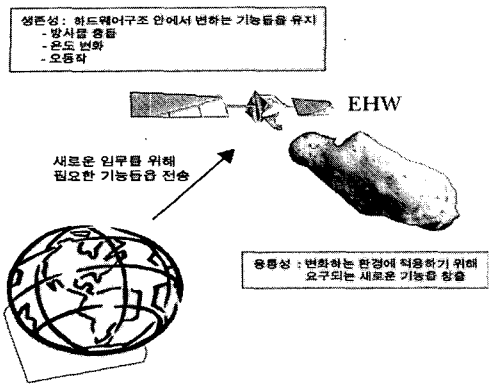


그림 15. 진화우주시스템.

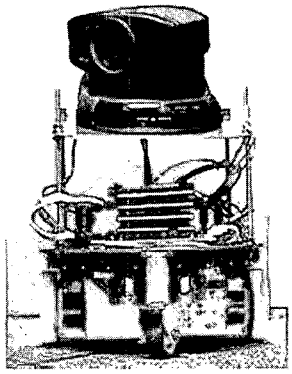


그림 16. 진화 로봇시스템.

다. 스토이카는 이런 개념을 이용해 결함을 허용하는 진화형 우주시스템을 연구하고 있다. 우주 전자공학의 진화는 진화형 우주 시스템(Evolvable Space System)으로 가는 첫 단계다. 진화 하드웨어는 보드 센서, 안테나, 재구성 메커니즘, 최적화 시스템 등을 포함하는 SOC(System on a Chip)로 발달될 수 있고 그것은 진화 우주 시스템의 발달을 더욱 가속화할 것이다(그림15).

4.2 진화적인 로봇 네비게이션 시스템

이동 로봇의 임무는 장애물을 피해 최소한의 동작으로 목표 지점에 도착하는 것인데 원하는 동작을 이끌어 내기 위해 진화 하드웨어를 이용해 로봇의 동작을 제어하게 될 Boolean Function을 만들어 낸다. Boolean Function은 진화 하드웨어에 의해 수행된 Function Form이나 RAM에 의해 만들어진 Truth Table Form에 의해 표현된다. 이 두 Form에 의해 각각 수행된 모바일 로봇의 이동 경로를 비교한 결과 진화 하드웨어를 이용한 네비게이션 시스템의 수행 속도가 RAM을 이용한 것보다 약 두 배 정도 빠르다는 것을 알 수 있었다(그림16).

4.3 디지털 통신채널의 적응 "Channel Equalization"

디지털 통신에서 중요한 문제 중 하나가 바로 Channel Equalization이다. Channel Equalization은 수신 쪽에서 받는 과정에서 망가진 신호를 재생하기 위한 방법이다. Channel Equalization은 두 가지 이유로 인해 상당히 어려운 게 사실이다. 첫 번째로는

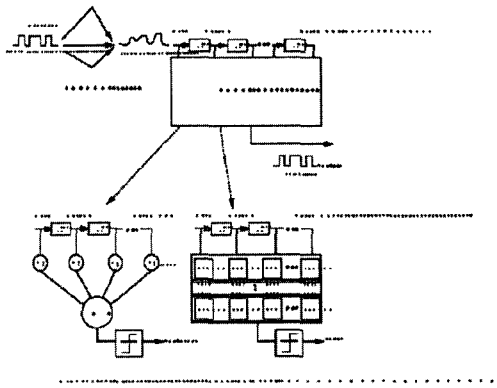


그림 17. Adaptive Equalization.

다양한 방법으로 전해지는 신호의 비선형적인 파괴이고, 두 번째는 보내지는 환경이 계속해서 변한다는 것이다. 이런 문제점을 해결하기 위한 좋은 방법이 진화 하드웨어를 이용하는 것이다. 진화 하드웨어는 환경의 변화에 따라 재구성할 수 있는 특성을 갖고 있기 때문이다(그림17).

4.4 의수 컨트롤을 위한 진화하드웨어칩

의수(Myoelectric Prosthetic Hands)는 근육이 움직일 때 발생하는 신호에 의해서 작동한다. 각 사람마다 발생되는 근육 신호가 다르기 때문에 각각의 신호에 따라 적응하기 위해서는 약 한달간의 시간이 걸린다. 이런 문제를 해결하기 위해 진화 하드웨어 칩을 설계했다. 진화 하드웨어 칩은 7개의 기능적인 부분들로 구성돼 있다.

의수의 동작은 6가지이다. Open(A), Grasp(B),

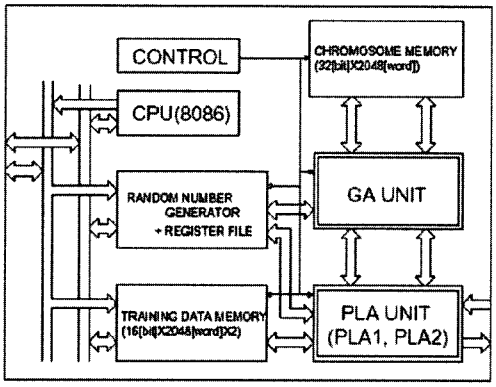


그림 18. 진화하드웨어 칩 블럭 다이어그램.

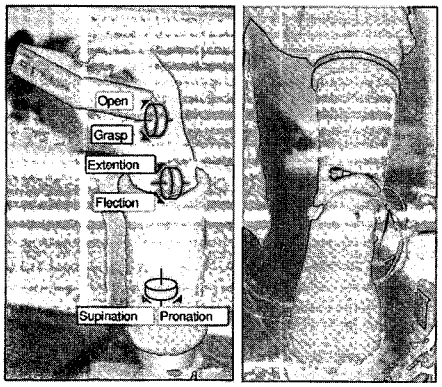


그림 19. 의수의 동작 6가지와 Micro Electronic 신호를 위한 센서.

Suspination(C), Pronation(D), Flection(E), Extension(F) 각각의 동작들은 세 부분으로 묶을 수 있고, 이 세 부분의 동작들은 각각 독립적으로 작동한다. 이제 동작을 하는 데 있어 진화 과정을 살펴보면 다음과 같다.

(1) Hand를 조정할 회로를 위한 입력 패턴을 만든다.
(2) 재구성할 조정회로를 위한 데이터를 만든다.
(3) 만들어진 데이터들로 회로의 최적화를 평가한다.
(4) 회로를 재구성한다.
(5) 진화된 회로를 최종적으로 최적화를 평가한다.

4.5 신경회로망의 구현

신경회로망은 동물의 뇌를 모델링한 계산 구조로서 예측, 데이터분류, 복원, 인식 등 여러 분야에 쓰

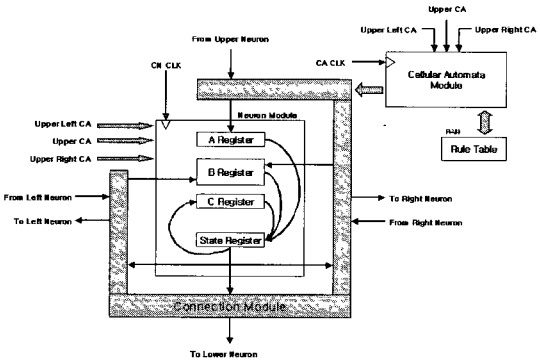


그림 20. 뉴런의 구조.

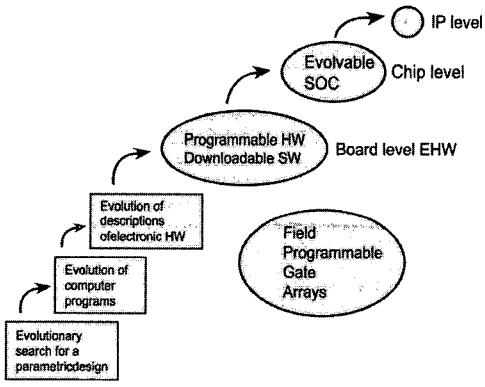


그림 21. 진화하드웨어를 위한 진화경로.

이고 있는 모델이다. 신경회로망을 하드웨어로 구현하기 위해 카오스 뉴런 모델을 설계 툴로 설계했다(그림20). 이것은 기능 레벨 진화 하드웨어에서 사용될 기능 모듈에 해당된다. 여기에서 사용된 신경회로망은 뉴런의 연결이 상하 좌우로만 돼 있는 단순구조의 신경망이다. 신경망구조의 단순성에 비해 뉴런 모델은 카오스 신경 모델을 사용해 복잡한 문제를 해결할 수 있도록 했다. 이 연구에서는 신경회로망의 구조를 하드웨어 상에서 진화시켜 XOR 함수 매핑과 시계열 예측 문제에 적용했다.

5. 진화하드웨어의 전망

진화 방식에 의한 하드웨어의 설계는 기존의 회

로 설계 방법으로는 설계하기 힘들거나 설계할 수 없는 시스템을 설계할 수 있는 가능성을 제시하고 있다. 진화 방식에 의한 하드웨어 설계가 기존의 회로 설계를 대치하는 수단으로 사용됐을 때, 사람에 의한 기존의 설계 방식에 비해 다음과 같은 장점을 지니고 있다.

(1) 진화적 설계는 사람이 설계할 때보다 더욱 많은 가능성을 고려하고 탐색할 수 있다. 특히 원하는 하드웨어 구조를 표현하기 어려울 때도 적용할 수 있다.
(2) 진화적 설계는 문제 영역에 특수한 기반 지식이 없어도 적용 가능하다. 특히 배경 지식을 획득하는 데 경비가 많이 들 경우 아주 유용하며 전문가에 의존하지 않아도 된다.
(3) 진화적 설계 기법은 다양한 형태의 제약 조건과 특수한 요구사항을 고려하는 것이 비교적 용이하며 이것은 보통 염색체의 표현이나 적합도 함수를 통해 이뤄진다.

따라서 진화 하드웨어의 응용 분야는 명시적으로 설계하기 어려운 회로의 자동설계, 결함 허용 시스템의 구현 등 그 적용분야가 무척 넓다. 결함 허용 시스템도 진화적 방식에 의해 결함이 발생한 회로를 재 진화시키게 되는데 이것은 발생한 결함의 종류나 위치, 지속 시간 등에 대한 정보나 분석 없이도 적합성의 판단만으로 회로를 재구성할 수 있다는 장점이 있다. 현재 진화 하드웨어에서 가장 많이 사용하는 플랫폼(Platform)은 보드(Board)다.

즉, 이것은 소프트웨어상에서 진화 알고리즘에 의해 만들어진 하드웨어를 결정하는 비트의 컨트롤 아래에서 프로그래밍이 가능한 하드웨어의 구조가 재구성된다는 것을 의미한다. 한 단계 진화된 진화 하드웨어의 모습은 진화 하드웨어 칩들이 단순한 보드 수준에서 벗어나 진화적 메커니즘을 갖고 있는 소프트웨어와 재구성이 가능한 하드웨어가 하나의 칩에 있는 SOC으로 발달할 것이다. 앞서 소개한 GAP에 관한 연구도 진화 프로세서를 하드웨어로 구현함으로써 진화 대상과 주체가 같은 칩에 존재하는 SOC의 형태를 구현하기 위한 방법이다. 한편 여기서 더욱 진화된 모습은 어느 하나의 특수한 목적을 갖고 있는 칩으로서의 의미가 아니라 진화적인 기능을 필요로 하는 모든 곳에서 사용될 수 있는 지능적인 특성(IP: Intellectual Property)을 갖는 레벨로 진화 하드웨어는 발전할 것이다(그림21).

지금까지 진화 하드웨어를 구현하기 위한 방법들을 알아봤다. 현재까지 구현된 진화 하드웨어의 문제점은 크기가 작고 구현이 FPGA라는 매체에 한정된다는 점과 실시간으로 변화하는 환경에 적응하기 위해서는 빠른 처리속도가 필요한데 현재의 연구로는 많은 실행시간이 걸린다는 점이다. 여러분들은 진화 하드웨어라는 말을 들었을 때 스스로 진화 적응하는 하드웨어를 생각했을 것이다.

아직까지는 여러분이 생각한 것만큼의 기능이 구현되지 않고 있음에 실망할지도 모른다. 그러나 진화하드웨어는 그 역사가 길지 않은 태동 학문으로서 반도체와 컴퓨터 기술의 발전에 따라 그 발전 가능성이 무궁한 분야라고 생각된다. 진화 하드웨어에 대한 자세한 설명을 모두 하진 못했지만 새로운 것을 소개한다는 기분으로 이 글을 썼다. 이 소개로 말미암아 새로운 지식에 대한 목마름을 조금이나마 채웠으면 한다.

참고 문헌

[1] Adrian Stoica " Evolvable Hardware: from on-chip circuit synthesis to EvolvableSpace System," Multiple-Valued Logic, 2000.(ISMVL 2000) Proceedings. 30th IEEE International Symposium on; p.161, 2000.
[2] Paul Layzell1 "Reducing Hardware Evolution' s Dependency on FPGAS," Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspried Systems, MicroNeuro '99 , p.171, 1999.
[3] 반창봉 "A syudy on the Design and Embodiment of Evolvable Hardware," 중앙대학교 석사 학위 논문, 2001.
[4] W. Liu, M. Murakawa, T. Higuchi, "ATM Cell Scheduling by Function Level Evolvable Hardware", in Lecture Note in Computer Science,

Evolvable System: From Biology to Hardware, Springer, No. 1259, 1997.

[5] F.H. Benett III, J.R. Koza, D. Andre, M.A.Keane," Evolution of a 60 Decibel Op Amp Using Genetic Programming" in Lecture Note in Computer Science, Evolvable Systems: From Biology to Hardware, Springer, No. 1259, 1997.

[6] H. de Garis," CAM-BRAIN : The Evolutionary of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speeds inside a Celluar Automata Machine(CAM)," in Lecture Notes in Computer Science, Towards Evolvable Hardware: The Evolutionary Engineering Approach, Springer, No.1062, p.76, 1996.

[7] A. Thompson," An Evolvable Circuit, Intrinsic in Silicon Entwined with Physics," in Lecture Notes in Computer Science, Evolvable Systems: From Biology to Hardware, Springer, No. 1259, 1997.

[8] M. Murakawa, S. Yoshizawa, T. Higuchi, "Adaptive Equalization of Digital Communication Channels Using Evolvable Hardware", Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, Springer-Verlag, No.1259, p.470, 1997.

[9] I. Kajitani, M.Murakawa, D.Nishikawa, H.Yokoi, N.Kajihara, M.Iwata, D.Keymeulen, H.Saksnashi, T.Higuchi ," An Evolvable Hardware Chip for Prosthetic Hand Controller", Proc. of the Seventh International Conference on Microelectrionics for Neural, Fuzzy and Bio-Inspired System(Micron euro 99), p.176, 1999.

[10] 김태훈 "Design of Evolvable Hardware Based on Genetic Algorithm Processor", 중앙대학교 석사학위 논문, 2004.

저|자|약|력

성 명 : 심귀보

◈ 학 력

- 1984년 중앙대 전자공학과 공학사
- 1986년 중앙대 대학원 전자공학과 공학석사
- 1990년 The Univ. of Tokyo 전자공학과 공학박사

◈ 경 력

- 1991년 – 현 재 중앙대 전자전기공학부 교수
- 2002년 – 현 재 중앙대 산학연컨소시엄센터 센터장
- 2004년 – 현 재 중앙대 기술이전센터 소장
- 2005년 – 현 재 (사)제어자동화시스템공학회 (ICASE) Fellow 회원
- 2006년 – 현 재 (사)한국퍼지 및 지능시스템학회 (KFIS) 회장