모듈 회로 진화를 통한 효과적인 진화 하드웨어

황금성° 조성배
연세대학교 컴퓨터과학과
yellowg@candy.yonsei.ac.kr, sbcho@csai.yonsei.ac.kr

An Effective Evolvable Hardware Through Modular Circuit Evolution

Keum-Sung Hwang° Sung-Bae Cho
Dept. of Computer Science, Yonsei University

요 약

진화 하드웨어(Evolvable Hardware: EHW)는 환경에 적응하여 스스로 하드웨어 구성을 변경할 수 있는 하드웨어로서 최근에 많은 관심과 함께 연구가 이뤄지고 있다. 하지만, 하드웨어의 복잡도가 증가할수록 진화를 위해 탐색해야 하는 해공간의 크기가 기하급수적으로 증가하기 때문에 아직까지 복잡한 하드웨어에 대해서는 좋은 활용방안을 찾지 못하고 있다. 이 논문에서는 이런 복잡한 하드웨어를 모듈별로 나눠서 진화시키는 방법을 제시하여 좀더 효율적인 진화의 가능성을 보인다. 기존에 주로 사용되던 회로 진화 디자인과 이를 모듈별로 나눠서 진화하는 방식을 실험을 통해 비교하고, 효과적으로 진화시간을 단축할 수 있음을 보인다.

1. 서론

1990년대 FPGA(Field Programmable Gate Array)라는 프로그래밍 가능한 하드웨어가 개발된 이래로 진화 하드웨어(Evolvable Hardware)라는 분야가 관심을 모으기 시작했다. 진화 하드웨어는 변형 가능한 하드웨어에 진화 알고리즘을 적용하여, 필요에 따라 언제든지 하드웨어 구성을 스스로 진화하여 변경할 수 있도록 한 하드웨어이다. 따라서 변화하는 환경에 따라 실시간으로 하드웨어의 상태를 변경하여 최적의 상태를 유지하는 우수한 환경 적응성을 가진다.

이러한 진화 하드웨어의 능력을 실용화하기 위해 많은 연구가 이루어져 왔다. 진화 회로 디자인을 위한 다양한 방법이 제시되었으며[1], 효율적인 디지털 회로의 진화적인 결합을 위한 연구가 있었다[2]. 이러한 연구들은 진화하드웨어를 실용화하기 위한 기본적인 방법을 제시하고는 있으나 한계를 가지고 있다. 바로 진화되는 하드웨어 크기의 한계이다. 하드웨어가 복잡해질수록 염색체의 길이가 길어지고, 이에 따라 진화에 요구되는 시간이 기하급수적으로 늘어나기 때문이다. 이러한 문제를 해결하기 위해서 게이트 회로 대신 높은 수준의 기능을 진화의 단위로 사용하는 기능 수준(function level)의 진화 하드웨어 디자인[3], 전체 하드웨어를 부분 기능으로 분할하여 진화하는 분할정복(divide-and-conquer) 하드웨어 디자인[4], 그리고 적합도 공간(Fitness Landscape)과 확장성(Scalability) 연구를 통해 회로를 일정 개수만큼 묶어서 진화시키려는 시도가 있었다[5].

본 논문에서는 이러한 연구들과 같이 복잡한 하드웨어의 진화를 위한 방법으로, 하드웨어의 출력별로 모듈화한 진화 방법을 제시하고자 한다. 즉, 하드웨어의 출력 별로 회로를 진화한 다음 이를 결합하여 목표했던 하드웨어를 얻는 것이다. 기존의 모듈화된 진화 방법은 하드웨어의 특성에 맞춰서 하드웨어의 부분을 나누어서 진화한 후 결합해야 했으나, 이 방법은 진화될 하드웨어의 부분을 나누기 위한 특별한 지식의 도움 없이 단지 출력의 수만큼 나누어서 진화하는 방식이다. 따라서, 게이트 수준의 회로 디자인에 효과적으로 응용될 수 있을 것으로 기대된다.

2. 디지털 회로의 진화

디지털 회로를 진화시키기 위해서는 보통 유전자 알고리즘(Genetic Algorithm)을 사용한다[6]. 각각의 하드웨어의 구조를 나타내는 염색체 집단을 임의의 구조로 초기화시키고, 원하는 하드웨어 기능에 유사한 정도를 적합도로 사용하여, 만족스러운 염색체가 발견될 때까지 유전 연산에 의한 진화를 하는 방법이다. 본 논문에서 사용한 유전자 알고리즘은 교차 방법으로 일정교차(uniform crossover)를 사용하였고, 적합도에 비례하여 선택하는 룰렛 선택 방식, 하드웨어 구조를 나타내는 유전자가 일정 확률로 변경되도록 하는 bit-flip 방법과 유사한 돌연변이를 적용하였다. 그리고 엘리트 유지 전략을 적용하여 최상의 해가 유지되도록 하였으며, 적합도 평가는 가능한 모든 입력에 대한 출력 패턴이 원하는 결과에 일치하는 정도를 퍼센트로 나타낸 값을 사용했다.

실험에서 염색체는 그림1과 같이 외부에서 들어오는 입력과 바둑판처럼 엮어져 있는 하드웨어 기본 셀들, 그리고 출력을 하나의 배열로 표현하도록 하여 사용하였다. 각각의 셀들은 2개의 입력과 1개의 출력, 그리고 특정 논리 기능을 가진다.

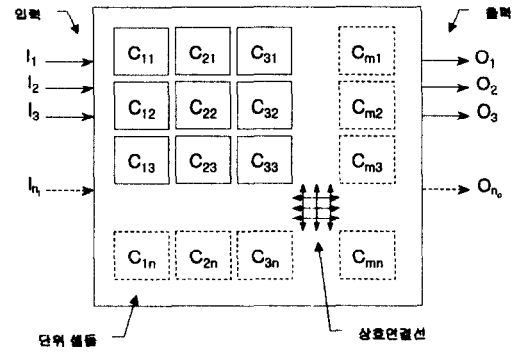

그림 1. n×m 개의 셀이 회로에 배열되어 있는 모습

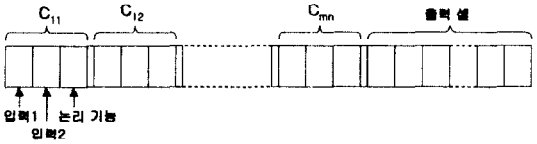

그림 2. 염색체의 구조

각 셀에 들어오는 입력값의 표시는 약속된 번호로 표현하였다. 외부입력0에서부터 차례로 번호를 매겨서 모든 셀에 번호를 매겨서 사용하였다. 그리고 각 셀은 이전 열에 있는 셀의 출력만을 사용할 수 있다. 표 1은 실제로 진화시켜서 얻은 염

색체의 한 예를 보여준다.

표 1. 진화시켜서 얻은 2×2 게이트 배열의 1비트 전덧셈기 염색체의 예. 입력이 2개이므로 셀이 2번부터 시작한다.

셀번호	셀의 유전자
입력	0, 1
2	0, 1, 8
3	0, 1, 4
4	2, 3, 9
5	3, 3, 1
출력	4, 5

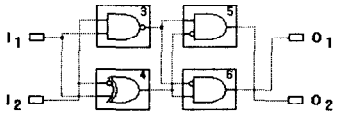

그림 3. 2×2 게이트 1비트 전덧셈기. 표 1에 나타나 있는 1비트 전덧셈기의 회로 구조이다. O_1은 Carry, O_2는 Sum을 나타낸다.

각 셀의 세 번째 수는 논리 기능을 나타낸다. 실험에서 사용된 논리 기능은 표 2와 같이 할당하여 사용했다. 이때, 입력값을 통과시키거나 NOT을 적용하여 통과시키는 기능은 따로 부여하지 않았다. 중복된 입력의 게이트로 같은 효과를 나타낼 수 있기 때문이다.

표 2. 셀에서 선택 가능한 10종류의 논리 기능. ¬는 NOT, ∧는 AND, ∨는 OR, ⊕는 XOR을 나타낸다.

번호	기능	번호	기능
0	x∧y	5	x∨(¬y)
1	x∨y	6	(¬x)∨y
2	x∧(¬y)	7	(¬x)⊕y
3	(¬x)∧y	8	¬(x∧y)
4	x⊕y	9	¬(x∨y)

3. 효율적인 하드웨어 디자인 방법

기존의 진화는 전체 하드웨어를 한꺼번에 비교 탐색하는 것에 많은 시간을 소모한다. 따라서 본 논문에서는 출력 부위별로 일부분씩 진화시키는 방법을 제시한다. 각각의 출력값 패턴에 대해 부분 하드웨어들을 진화시킨 뒤 결합하는 방법이다. 이렇게 하면 복잡한 하드웨어를 대신 간단하고 작은 하드웨어 여러개 진화시키므로 진화에 필요한 시간을 단축시킬 수 있다.

표 3은 기존의 직접 진화 방법과 모듈별 진화 방법을 실험하여 비교한 표이다. 직접 진화 방법으로 진화한 경우 평균 세대수가 상당히 크게 나왔다. 특히, 2비트 덧셈기와 곱셈기에서는 직접 진화로는 250만 세대 이내에 수렴되지도 않았다. 단지 7×4 게이트를 사용한 경우 한 번의 진화가 성공했다. 그에 비해 모듈별 진화는 사용되는 게이트의 규모가 작기 때문에 비교적 쉽게 진화가 이뤄짐을 알 수 있다. 실험에서 1비트 덧셈기에서는 약 37배 정도 빠른 세대에, 그리고 2비트 덧셈기와 2비트 곱셈기에서는 400배, 6000배 이상의 빠른 세대에 수렴되어 효율적인 진화 성능을 보이고 있다. 이는 하드웨어가 복잡해질수록 진화되기 어려운 하드웨어의 특성에 기인한다.

표 3에서 볼 수 있듯이 하드웨어의 진화는 구성에 사용되는 게이트의 수와 큰 연관성을 가지고 있다. 그것은 게이트의 수가 바로 해공간의 크기에 관련되기 때문이다. 게이트의 수에 따른 해공간의 크기(표현가능한 염색체의 수)는 다음의 수식 1과 같이 계산할 수 있다. 이때 n_i는 입력의 수, n_f는 셀의 기능의 수, n_r과 n_c는 게이트의 행과 열의 수이다.

$$n_I^{2n_r} \times n_f^{n_c n_r} \times n_r^{2n_r(n_c-1)} \quad (1)$$

위 수식의 의미를 좀더 자세히 알기 위해 직접 값을 넣어 수치를 살펴보도록 한다. f_n은 본 논문에서 10이고, 입력수와 출력수를 적당히 2로 준 뒤, c_n과 r_n에 대한 해공간의 크기를 살펴보면 표 4와 같다. 해공간의 크기가 기하급수적으로 증가함을 알 수 있다. 따라서 복잡한 하드웨어를 간단하게 바꾸어서 진화시키는 방법은 상당히 유용한 것이 될 수 있는 것이다.

표 3. 진화 방법별 하드웨어 진화 비교. 각 방법으로 10회의 진화 결과를 평균하였다. 실험 한계 세대인 250만 세대에 수렴하지 않은 경우 '>>'로 표시하였다. '>'는 일부만 수렴되었을 경우이다.

하드웨어	게이트수, 진화방법	평균 세대수
1비트 덧셈기	2×2, 직접진화	36.7
	2×1, 직접진화	6.7
	2×1, 모듈별 진화	0
	(1×1, 모듈1 진화)	0
	(1×1, 모듈2 진화)	0
2비트 덧셈기	7×4, 직접진화	>417,454
	4×3, 직접진화	>>2,500,000
	4×3, 모듈별 진화	6,108.1
	(3×3, 모듈1 진화)	5,982.6
	(2×2, 모듈2 진화)	125.1
	(1×1, 모듈3 진화)	0.4
2비트 곱셈기	7×3, 직접진화	>>2,500,000
	6×2, 직접진화	>>2,500,000
	6×2, 모듈별 진화	371.1
	(2×2, 모듈1 진화)	23.8
	(2×2, 모듈2 진화)	180.4
	(2×2, 모듈3 진화)	166.6
	(1×1, 모듈4 진화)	0.3

표 4. 게이트 배열의 행과 열 수에 따른 해공간의 크기

게이트 수	해공간 크기
1×1	4.0×10^{1}
2×2	2.6×10^{6}
3×3	3.4×10^{16}
4×4	7.2×10^{32}
5×5	9.3×10^{55}

실험에서 나타난 진화의 과정을 살펴본다. 그림4는 모듈 진화방법과 직접 진화 방법의 최고 적합도를 세대별로 나타내어 비교한 그래프이다. 하드웨어의 크기가 작은 모듈일수록 빠른 세대에 수렴을 하고 있다. 그리고 수렴하는 세대의 값이 크기에 따라 상당히 늦춰지고 있음을 알 수 있다.

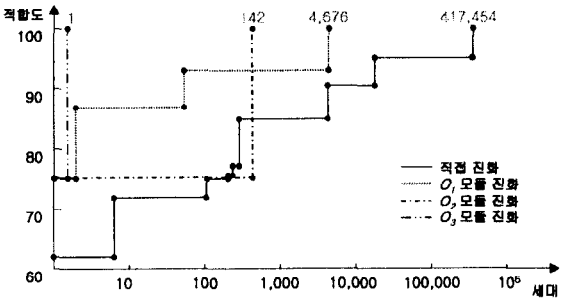

그림 4. 2비트 덧셈기의 직접 진화, 모듈별 진화시 최고 적합도 변화 그래프. 각각 10번씩 진화시킨 뒤에 평균에 가까운 그래프를 선택하였고, 직접 진화의 경우는 수렴에 성공한 그래프를 표현하였다.

그림 5는 앞에서 얻은 모듈별 부분 하드웨어들을 결합하여 나타낸 회로이다. 열과 행을 적절히 늘려서 단순히 결합하였으며, 출력 부분까지의 전달은 AND 게이트에 같은 입력을 통과

하도록 하였다.

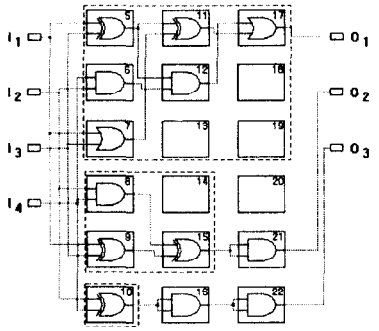

그림 5. 모듈별로 진화시킨 뒤 결합하여 얻은 6×3 게이트 배열의 2비트 덧셈기. 점선으로 표시된 부분이 모듈이다.

그림에서 모듈별로 진화시켜 얻은 회로는 6×3의 게이트만 사용하고 있다. 직접 진화로 얻은 회로의 경우 7×4의 게이트를 사용하고 있는데 이보다 더 간단한 구조를 가지고 있다. 속도도 빠르고 구조도 간단한 것이다. 물론 직접 진화로 계속 시도하면 셀의 기능이 중복사용 되어서 더 효율적인 회로가 나올 수 있다. 하지만 진화 속도의 측면에서 볼 때 시도할 가치가 적다. 게다가, 모듈별 진화된 회로는 중복된 게이트를 재사용하도록 처리해 주면 더욱 효율적인 회로를 얻을 수 있다. 그림 5를 살펴보면, 게이트 5번과 게이트 9번, 그리고 게이트 6번과 게이트 8번이 같은 기능을 가지고 있다. 이를 재사용하도록 처리하면 그림6과 같이 된다.

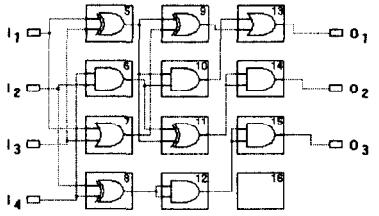

그림 6. 모듈별로 진화시킨 뒤 결합하고, 재사용 처리하여 얻은 4×3 게이트 2비트 덧셈기.

재사용 처리 결과 상당히 간단한 회로가 완성되었다. 이와 같이 모듈별로 진화하여 얻은 하드웨어에서 재사용 가능한 셀을 고려하여 결합을 하면 짧은 세대에 좋은 성능을 지닌 하드웨어를 얻을 수 있다. 하지만, 이와 같은 효과를 얻기 위해서는 부분적으로 얻은 회로들에 재사용 가능한 부분이 존재해야 한다. 제시하는 방법대로 모듈별로 진화했다고 해서 항상 재사용 가능한 부분이 존재하는 것은 아니기 때문이다.

이같은 점을 해결하기 위해서는 부분 하드웨어를 여러 개 얻을 필요가 있다. 즉, 모듈별로 충분히 다양한 해를 얻은 후 가장 효율적인 조합을 찾아서 결합하는 것이다. 이는 전체 하드웨어의 진화하는 시간에 비하면 상당히 간단한 과정이다. 부분 하드웨어를 여러 개 얻기 위해서는 진화를 여러 번 하는 방법도 있겠지만, 종분화를 이용한 다양한 하드웨어의 진화 방법이 유용하다[7]. 종분화 알고리즘을 사용하면 한 번의 진화로도 다양한 개체를 동시에 얻을 수 있다. 이렇게 얻어진 개체들을 서로 결합하여 효율적인 하드웨어를 얻을 수 있을 것이다. 그림 7의 회로들은 실제로 종분화 알고리즘을 사용하여 478세대에 얻은 1비트 덧셈기이다. 이처럼 한 번에 얻어진 여러 모듈을 결합에 이용하면 효율적인 하드웨어를 얻을 수 있다.

4. 결론

진화 하드웨어는 환경에 따라 변화하고 적응하는 능력이 뛰어나기 때문에 상당히 기대가 큰 분야임에도 불구하고 여러 실용상의 문제들로 인해 아직도 실생활에 활용되지 못하고 있다. 그러한 문제 가운데 하나가 바로 진화에 걸리는 시간의 문제이다. 실제로 본 논문에서 얻은, 효율적인 크기의 2비트 덧셈기와 같은 크기인 4×3 게이트를 가지고, 직접 진화 방법을 여러 번 시도하여 보았지만, 250만 세대 내에 진화되는 결과를 얻을 수가 없었다. 다른 논문의 결과를 보면 3입력 멀티플렉서를 사용하고도 3비트 곱셈기를 진화시키는 데에 거의 100만 세대 정도가 소요되고 있다[5].

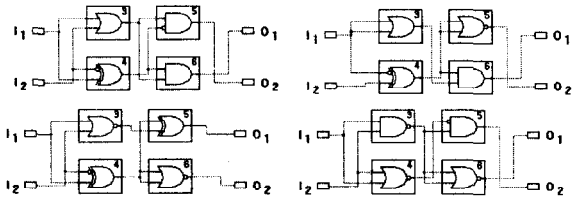

그림 7. 종분화를 이용하여 한 번에 얻은 4개의 2×2 게이트 1비트 전덧셈기.

본 논문에서는 이런 문제를 해결하기 위한 방안의 하나로 하드웨어를 출력 부위를 기준으로 나눠 모듈별 진화한 후 결합하는 방법을 제시하였고, 상당히 효율적인 진화가 가능함을 보였다. 실험에서는 약 100배 이상의 세대가 단축되었으며, 발견된 하드웨어도 직접 진화 방법에 의한 하드웨어보다 훨씬 최적화된 하드웨어였다. 따라서 하드웨어의 크기를 줄이는 방법으로 유용함을 알 수 있었다. 하지만, 여전히 문제는 남아 있다. 우선, 출력별로 모듈을 나누는 것은 출력이 여러 개 존재하는 경우에만 가능하다. 출력의 수가 적은 복잡한 하드웨어의 경우에는 의미가 없다. 또한 모듈별로 나눈다고 하더라도 기본적인 하드웨어 복잡도가 크게 되면 모듈별 진화도 잘 되지 않는다. 실험에서 결과를 제시하지는 않았지만 3비트 덧셈기나 3비트 곱셈기의 경우에는 모듈로 나눈 하드웨어조차도 250만 세대 내에 수렴되지 않았다. 앞으로 좀더 효과적인 하드웨어 분할정복(divide-and-conquer) 방법에 대한 연구가 계속해서 필요하다.

5. 참고문헌

[1] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Transactions on Systems, Man, and Cybernetics*, Part C, Vol.29, pp.87-97, February 1999.

[2] A. Thompson, *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. London: Springer-Verlag.

[3] W. Liu, et al., "ATM cell scheduling by function level evolvable hardware," *Proceedings of the First International Conference Evolvable Systems: From Biology to Hardware*, pages 180-192, Tsukuba, Japan, 7.-8. October 1996.

[4] J. Torresen, "Increased complexity evolution applied to evolvable hardware," *Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining, and Complex Systems*, Proc. of ANNIE'99. ASME Press, November 1999.

[5] V. K. Vassilev, "Scalability Problems of Digital Circuit Evolution: Evolvability and Efficient Designs," *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 55-64, IEEE Computer Society, Palo Alto, California, July 13 - 15, 2000.

[6] J. F. Miller, et al., "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study," In D. Quagliarella, et al., editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pp. 105-131, 1997.

[7] 황금성, 조성배, "종분화를 이용한 다품종 하드웨어의 진화," 정보과학회 봄 학술발표 논문집(B) 16:1 229-232, 2001.