교육용 논리회로 시뮬레이터를 위한 계층적 컴포넌트

양승조*, 김은주*, 류승필*
*세명대학교 컴퓨터학부
e-mail:nysjttl@nate.com

Hierarchical Component for Educational Logic Circuit Simulator

Seung-Ju Yang*, Eun-Ju Kim*, Sung-Pil Lyu*
*Dept of Computer Science, Semyung University

요 약

본 논문은 조립 개념을 탑재한 계층적 컴포넌트를 소개하며 이를 교육용 논리회로 시뮬레이터에 적용하여, 그 장점을 평가하고자 한다. 본 논문에서 제시하는 방법에 의한 컴포넌트의 설계과정은 계층 구조적이므로 설계 변경과 유지보수가 편리하고 컴포넌트의 재사용성을 상승시킨다. 이를 교육용 논리회로 시뮬레이터에 적용하면 다음과 같은 장점이 있다. 첫째, 불필요한 부분을 캡슐화하기 때문에 복잡성을 줄인다. 둘째, 그 캡슐화된 부분의 내부 회로도와 동작 상태를 확인할 수 있기 때문에 수준별 · 자율(자기 주도적) 학습에 효과적이다. 셋째, 병렬 개발이 가능해서 조별 협동 학습 수행이 가능하다.

1. 서론

소프트웨어 개발 역사를 40년 정도로 볼 때 생산성이나 개발 방법에 있어 뚜렷한 전환점을 찾아볼 수 없는 소프트웨어 위기를 맞고 있고 그 원인으로는 크게 소프트웨어 생산성이 사용자들의 서비스 요구를 따라가지 못한다는 점과 소프트웨어 품질이 향상되지 못하고 유지 보수가 힘들다는 점을 들 수 있다[1]. 이러한 정보기술 환경과 소프트웨어 위기를 극복할 수 있는 한 가지 해결책으로써 '컴포넌트 소프트웨어'가 제시되고 있다.

컴포넌트의 개념이 가장 활발히 발전되고 있는 프로그래밍 언어나 소프트웨어 공학 분야에서도 '조립개념'의 결여로 설계가 난해하고, 설계사항의 변경이 어려우며, 가독성과 재활용성이 저하되어 실제로 CBD(Component Based Development) 프로젝트를 수행하는 사람들은 방법론과 프로세서스의 불일치 그리고 여러 기술들의 혼란 상태에서 헤어 나오지 못하고 있다[2]. 이에 발전된 컴포넌트 개념인 '계층적 컴포넌트'를 소개하며 이를 교육용 논리회로 시뮬레이터에 적용하여, 내부 논리를 확인 할 수 있게 함으로서 컴포넌트 단위 테스트를 용이하게 하며 가독성을 높이고, 병렬 개발이 가능하게 된다. 이러한 장점을 효과적으로 나타낼 수 있는 계층적 컴포넌트를 교육용 논리회로 시뮬레이터에 적용하여 응용프로그램에서의 계층적 컴포넌트의 실효성을 확인하고 나아가 프로그래밍 언어 및 소프트웨어 공학 분야에서 활용 가능성과 비주얼 언어로의 발전 가능성을 확인해 보고자 한다.

2. 관련 연구

2.1 컴포넌트 기반 개발(CBD)의 정의

CBD란 재사용 가능한 소프트웨어 모듈 컴포넌트를 생성 및 조립 생산, 선택, 평가 및 통합으로 구성하여 더 큰 컴포넌트를 생성하거나 완성된 어플리케이션 소프트웨어를 구축하는 개발 기법이다.

프로그래밍 위주의 전통적인 정보공학 개발방법론과는 달리 이미 만들어진 테스트 완료된 컴포넌트를 기본 아키텍처와 설계도에 따라 조립하는 방식의 새로운 개발형태로서, 컴포넌트의 오퍼레이션 및 상호 오퍼레이션을 정의하는 명세의 개발, 객체나 컴포넌트들로부터 컴포넌트의 구축, 컴포넌트들을 이용해 어플리케이션을 조립 등 이 세 가지 활동이 필요로 한다. 소프트웨어 컴포넌트를 조립해 새로운 어플리케이션을 만들 수 있으므로, 개발기간을 단축할 수 있고, 기존의 컴포넌트를 재사용함으로서 생산성과 정체성을 높일 수 있다. 컴포넌트로 시스템을 만들 때의 장점은 개발하려는 시스템의 전체를 컴포넌트화 할 수 있는 여러 개의 단위로 분리해 개발하므로 복잡성을 단순화시킨 수 있고 초기 개발을 실질적으로 수행할 수 있도록 해준다. 컴포넌트는 캡슐화 되어 있어 에러의 범위로 컴포넌트로 한정할 수 있어 단위 테스트가 가능하며, 유지보수가 용이하다. 또한 설계가 끝나면 병렬 개발이 가능하므로 프로젝트의 시간 관리가 용이해지며 프로젝트 시간을 줄일 수 있다.[3]

2.2 선행된 교육용 논리회로 시뮬레이터들의 연구

기존의 교육용 논리회로 시뮬레이터들은 논리회로의

편집과 수행을 용이하게 할 수 있도록 여러 가지 기능을 가지고 있는데, Burch[5]는 이러한 교육용 논리회로 시뮬레이터의 기본적인 기능을 표 1과 같이 비교하였다[6].

<표 1> Comparison of graphical logic circuit simulators

	LogIc SIm 3.0b	Logic Works 2.0	xLogic Circuits	EasySim 2.04	Logic Sim 2.0β	Multimedia Logic 1.2c	WinLog Lab
custom wire paths	yes	yes	yes	yes	yes	yes	yes
wires indicate values	yes	no	yes	yes	yes	no	yes
variable-input gates	no	no	no	yes	no	no	no
clock component	yes	yes	yes	yes	yes	yes	yes
flip-flops	yes	yes	yes	yes	yes	yes	yes
custom components	yes	yes	no	no	yes	no	no
save circuit	yes	yes	yes	yes	yes	yes	yes
print circuit	no	yes	no	yes	yes	yes	yes

표 1을 보면 Logicsim 3.0β, Logic works 2.0, Logicsim 2.0β를 제외한 나머지는 컴포넌트를 정의할 수도 없었으며 계층적 컴포넌트 또한 지원하지 않는다.

다음 표는 표 2에서 컴포넌트의 사용이 가능했던 것들의 최선 버전인 것들이다.

<표 2>최신 교육용 시뮬레이터들의 컴포넌트 지원 현황

	LogicSim 3.0β	Logic Works 5.0.2	LogicSim 2007
인터페이스	개별인터페이스	회로작성 인터페이스사용	개별 인터페이스
계층구조	No	Partial	No
내부 논리 확인	No	Partial	No
컴포넌트 재사용성	Partial	Partial	Partial

컴포넌트의 계층적인 구성은 Logic Works 5.0.2을 제외하고 불가능 하였으며 Logic Works 5.0.2도 2단계로 제한되어 있어 사용이 제한적이다.

3. 제안된 계층적 컴포넌트의 표현

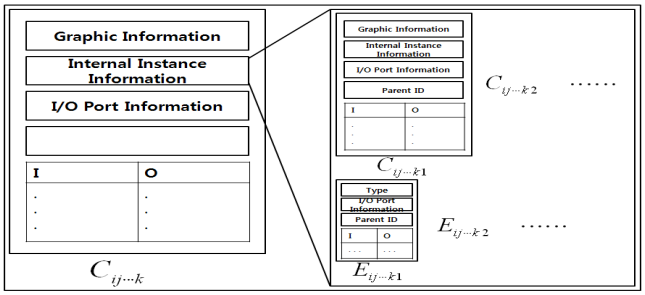
CBD(Component Based Development) 방법론은 개발 진행 도중 발생하는 요구사항 및 분석/설계의 변경 과정을 점진적 개발 주기를 반복적으로 수행함으로써 실제적인 개발 주기 계획에 반영하는 것이다. 현재 CBD 방법론은 프로그래밍 언어에만 국한되어 사용되어 사용되고 있다. 하지만 응용프로그램에서의 컴포넌트 활용되면 응용프로그램의 활용도가 높아지며, 재활용성으로 작업 시간이 대폭 감소할 것이다. 따라서 이러한 '컴포넌트 소프트웨어'의 필요성이 높아지고 있다. 또한 현재 CBD 방법론에도 문제점이 있는데 바로 조립 개념의 결여이다. 이로 인해 설계, 설계 변경, 유지 보수, 컴포넌트 단위 테스트 등이 용이하지 않고, 전문가의 도움이 꼭 필요하게 된다. 하지만 사람이 하드웨어를 조립하듯 컴포넌트를 만들고 여러 컴포넌트를 조립하여 하나의 응용프로그램을 만드는 조립 개념이 첨가되면 위의 문제점이 한결 감소될 것으로 예상된다. 이렇게 조립 개념을 주로 이용한 것이 계층적 컴포넌트이다.[3]

교육용 논리회로 시뮬레이터에서는 기본적인 요소(Element)인 And, Or, Xor, Nand등을 필요로 한다. 이러한 기본 요소를 이용하여 컴포넌트를 구성하게 되는데 이 컴포넌트는 더 상위 개념의 컴포넌트의 구성요소로 이용된다. 이러한 방법으로 점진적, 반복적으로 컴포넌트 개발 과정을 수행함으로써 여러 컴포넌트가 자연히 생성되며 최종적으로는 하나의 응용프로그램까지 생성할 수 있게 된다. 여러 단계에서 생성된 컴포넌트는 각자의 역할을 수행 할 수 있으며 그로 인해 컴포넌트 단위별 테스트가 가능해지며, 생성라인이 다른 컴포넌트를 생성하는 경우 병렬개발이 가능하다. 또한 이 컴포넌트가 다른 프로젝트나 컴포넌트를 생성하는 경우 병렬개발이 가능하다. 또한 이 컴포넌트가 다른 프로젝트나 컴포넌트에 재사용됨으로써 재활용성이 극대화된다.

식 1은 제안된 계층적 컴포넌트의 표현이다. C는 컴포넌트(Component)를 나타내며 E는 기본 요소(Element)를 나타낸다. 일반화 된 식과 그림에서 설명의 편의성을 위해 C는 컴포넌트 또는 기본 요소도 될 수도 있다.

$$C_{ij....k} = C_{ij....k1}, C_{ij....k2}, ... C_{ij....kn} \quad \text{식1}$$
$$i,j,k = 1,2,3.....n$$

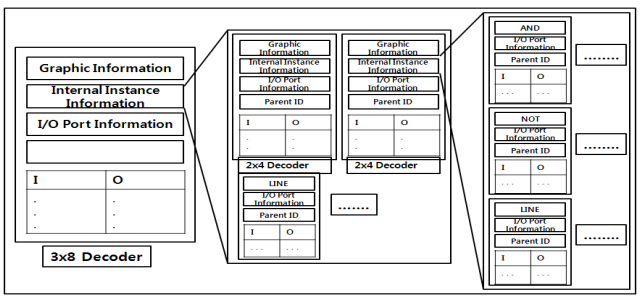
하나의 컴포넌트 $C_{ij...k(name)}$ 은 내부에 여러 컴포넌트 $C_{ij...k1(name)}$, $C_{ij...k2(name)}$, ...$C_{ij...kn(name)}$ 까지의 컴포넌트를 포함할 수 있다. 먼저 $C_{ij..k}$는 $C_{ij..k1}$, $C_{ij..k2}$,... ,$C_{ij..kn}$ 까지를 포함하며, 다시 $C_{ij..k1}$는 $C_{ij..k11}$, $C_{ij..k12}$...$C_{ij..k1n}$까지를 포함한다. 마지막으로 $C_{ij..k12}$는 $C_{ij..k121}$, $C_{ij..k122}$...$C_{ij..k12n}$까지를 포함한다.



(그림 3) 계층적 컴포넌트의 데이터 구성 표현

그림 3은 컴포넌트가 갖는 정보를 표시하는 그림이다.

하나의 컴포넌트는 그래픽 정보, 내부 객체 정보, 입출력 포트 정보, 부모 컴포넌트의 ID 그리고 연결 정보를 가지고 있다. 여기서 최상위의 컴포넌트는 부모 컴포넌트의 ID를 가지지 않고 입출력 연결 정보는 사용자가 연결하면서 생성된다. 또 다시 내부객체 정보에는 내부의 컴포넌트와 기본 요소 즉, 각종 게이트들을 포함할 수 있다. 내부 객체 역시 내부 객체들 간의 연결정보를 가지고 있어서 데이터를 보내는 데에 도움을 준다.



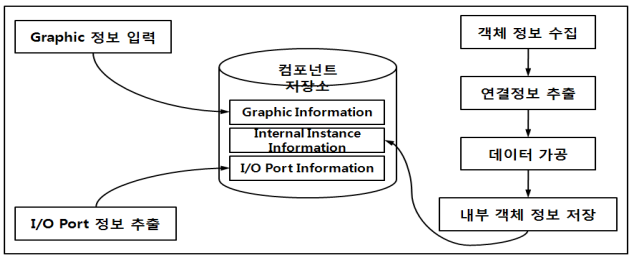
(그림 4) 계층적 컴포넌트의 데이터 구성 예

그림 4는 2×4 디코더와 3×8 디코더의 예를 그림으로 표현한 것이다. 이런 방식으로 여러 컴포넌트가 제작되며 각 단계에서 생성된 컴포넌트는 여러 용도로 사용될 수 있어서 재활용성이 증가된다. 또한 각 단계마다 테스트를 거치게 되면서 컴포넌트의 무결성은 높아지고 컴포넌트 단위로 테스트가 되고 컴포넌트의 내부 논리가 확인되므로 문제 해결과 유지 보수가 더욱 쉬워지게 된다. 설계 사항이 변경되었을 경우에도 컴포넌트를 제작하여 해당 컴포넌트만 교체가 가능해서 복잡한 과정을 생략할 수 있다.

4. 교육용 논리회로 시뮬레이터를 위한 계층적 컴포넌트 설계 및 구현

4.1 컴포넌트 저장 프로세스

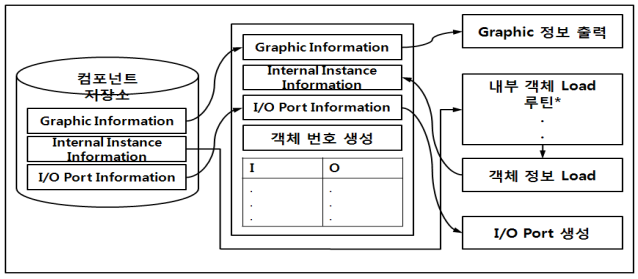
컴포넌트를 사용하기 위해서는 컴포넌트를 저장, 로드하는 중요한 과정이 필요하다. 컴포넌트를 저장하기 위해서는 선택된 객체의 정보를 수집하여 연결정보를 추출한 후 데이터 가공을 통하여 컴포넌트 저장소에 내부 객체 정보 부분에 저장하게 된다. 이 때 연결정보를 추출하면서 컴포넌트의 I/O 포트 정보도 추출하여 저장소에 저장한다. Graphic 정보는 사용자에게 입력받아 저장하게 된다.



(그림 5) 컴포넌트 저장 프로세스

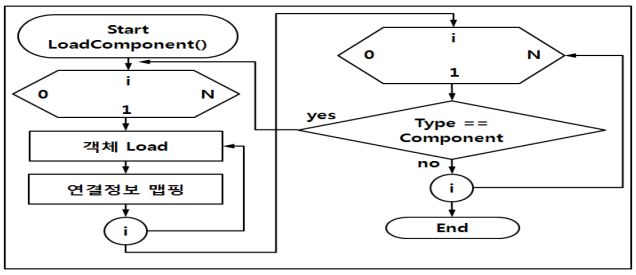
4.2 컴포넌트 로드 프로세스

그림 6은 컴포넌트의 로드 프로세스이다. 컴포넌트 저장소에서 그래픽 정보와 입출력 포트 정보를 불러오며, 내부 객체 정보는 내부 객체 로드 루틴을 통하여 객체정보를 불러온다.



(그림 6) 컴포넌트 로드 프로세스

그림 7은 컴포넌트의 내부 객체 로드 프로세스이다. 컴포넌트 내부의 객체 수만큼 반복하여 객체정보를 불러오고 그 불러온 내부 객체들 간의 연결 정보를 매핑하는 선행 과정을 수행함으로서 가장 상위의 컴포넌트가 포함하고 있는 내부 객체를 모두 불러온 후 다시 그 내부 객체 중에서 컴포넌트가 있는지 없는지 확인한 후 컴포넌트가 있을 경우 그 컴포넌트에 대해 다시 내부 객체 정보를 불러오는 과정을 거쳐 한 컴포넌트의 내부에 계층적으로 구성되어 있는 컴포넌트를 불러온다.



(그림 7) 컴포넌트 내의 객체 로드 루틴

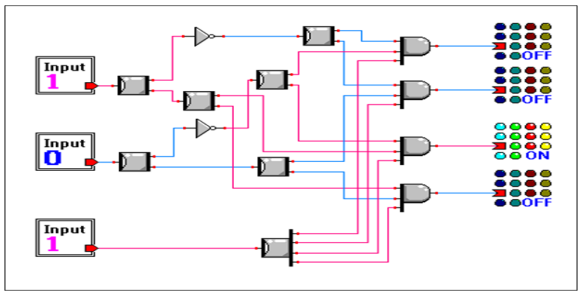
4.3 수행과정

Component로 저장될 때 객체식별번호와 불러올 때의 객체식별번호는 차이가 있다. 현재 프로젝트에서 이미 불러온 객체 및 컴포넌트가 존재할 수 있기 때문이다. 따라서 이미 불러온 객체의 수를 고려해 불러오는 컴포넌트 및 그것의 내부객체나 내부컴포넌트 모두 객체 식별번호가 변경되어야 한다. 객체 간의 연결정보 또한 변경된 객체 식별 번호에 맞게 변경되어야 한다. 이러한 과정이 컴포넌트 로드 프로세스 중에 연결정보 맵핑의 과정이다.

Input/Output으로 구분하여 저장하고, 자신과 연결되어 있는 객체 식별 번호를 저장한다. 현 프로젝트에서 Line 객체와 Component는 Port를 통해 연결되며 연결되는 순간 Port 정보에는 연결되는 Line객체의 식별 번호를 저장하게 되며, I/O 연결정보에도 저장되어 추후 메시지를 전달할 때 이용된다. 예를 들면 메시지 전달은 Return이나 Parameter로 볼 수 있고 전달 할 곳의 주소를 I/O 연결정보라고 보면 된다. 이러한 일련의 과정이 끝나면 실행을 하여 동작 상태를 확인한다. 이러한 수행과정은 메시지 전달과 연산과정으로 나눌 수 있다. 한 객체가 데이터를 받아 자신의 연산을 한 후 자신이 이전에 갖고 있던 데이터에서 변경이 필요하면 자신의 데이터를 변경한 후 자신의 Output 포트와 연결되어 있는 객체의 메시지 전달 프로세스를 실행시키고 변경된 데이터를 전달한다. 이렇게 객체간의 Communication이 연계되어 유동적으로 움직이게 된다. 컴포넌트가 계층적으로 구성되어 있어도 내부의 객체들도 메시지를 받고 변경된 메시지를 다음 객체에 전달하기 때문에 별도의 과정은 필요하지 않다. 이러한 방법으로 객체 모두 각자가 데이터를 받아 연산하고 전달하는 과정을 통해 시스템 전체가 움직인다. Component 내부 상태 확인도 선택된 Component의 내부객체들도 모두 Load되어있는 상태이므로 선택된 Component의 식별번호를 부모 식별면호(Parent ID)로 가지고 있는 객체들을 내부 상태 확인 창에 출력하고 그 객체의 데이터에 맞게 상태를 출력하면 동작상태 확인이 가능하게 된다.

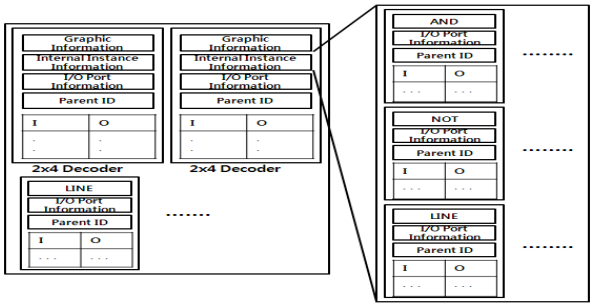
4.4 실행화면

그림 8는 2×4 디코더를 구현하기 위한 내부 회로도 이다. 이 회로도를 토대로 컴포넌트화하여 2×4 디코더를 구성한다.



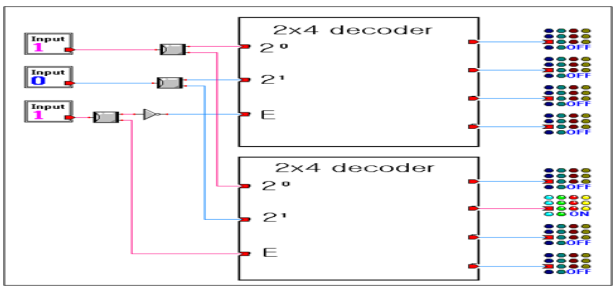
(그림 8) 2×4 디코더를 구성하기 위한 회로도

이 회로도를 통하여 2×4 디코더를 구성할 때의 계층적 컴포넌트 데이터 구성도를 표현하면 그림 9와 같다. 2×4 디코더 자신의 그래픽 정보, I/O 포트정보, 입출력 연결 정보, 내부 객체정보를 가지고 내부 객체정보에는 LINE 객체 여러 개와 AND, NOT 게이트 등의 정보를 내부객체 정보에 저장하게 된다. 이렇게 구성된 2×4 디코더는 필요한 부분에서 언제나 불러와 사용할 수 있다.



(그림 9) 2×4 디코더의 내부 데이터 구성도

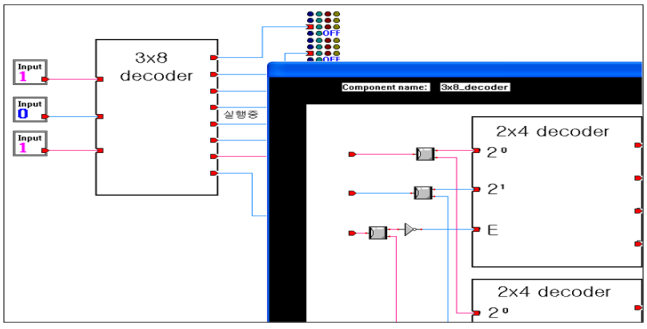
이렇게 구성된 2×4 디코더를 이용하여 3×8 디코더를 구현하려고 하면 다음과 같이 회로도를 구현하여 테스트한 후 3×8 디코더를 생성하여 추후에 사용이 가능하다.



(그림 10) 2×4 디코더를 이용하여 3×8 디코더를 구성하기 위한 회로도

3×8 디코더는 자신의 그래픽 정보와 입출력 포트 정보, 입출력 연결 정보, 내부 객체 정보를 가지고 내부 객체 정보에는 위의 회로도와 같이 2×4 디코더 두 개와 기본 요소가 되는 게이트 4개와 라인 객체 18개로 구성된다.

그림 9의 회로도를 통하여 다시 3×8 디코더를 구현하여 실행시킨 화면과 그것의 내부 회로도를 볼 수 있는 그림은 다음과 같다.



(그림 12) 3×8 디코더의 실행 화면과 내부 논리 확인

3×8 디코더의 구성 과정을 위의 그림과 같이 거쳤다. 이렇게 점진적으로 하나의 컴포넌트를 만들어가는 과정이 조립 개념과 유사하고, 또한 회로도의 일부를 컴포넌트화하여 사용할 수 있다. 또 개별적으로 3×8 디코더를 실행하여 테스트가 가능하여 단위 테스트가 용이하다.

5. 결 론

본 논문에서 제시하는 방법에 의한 컴포넌트의 설계 과정은 계층 구조적이므로 설계 변경과 유지보수가 편리하고 컴포넌트의 재사용성을 상승시킨다. 이를 교육용 논리회로 시뮬레이터에 적용하면 다음과 같은 장점이 있다. 첫째, 불필요한 부분을 캡슐화하기 때문에 복잡성을 줄인다. 둘째, 그 캡슐화된 부분의 내부 회로도와 동작 상태를 확인할 수 있기 때문에 수준별 ·자율(자기 주도적) 학습에 효과적이다. 셋째, 병렬 개발이 가능해서 조별 협동 학습 수행이 가능하다.

참고문헌

[1] 박병형, 블록놀이와 CBD, 태영, 2002.

[2] John Cheesman, UML Components : 컴포넌트 기반 소프트웨어 명세를 위한 실용적인 프로세서, 서울:인터비전, 2001.

[3] 우경창, “ CBD 방법론 적용 과정 및 효과에 관한 연구”, 서강대학교 정보통신 대학원, 2003.

[4] 최성, “소프트웨어 개발성 향상을 위한 컴포넌트 기반 기술 연구”, NSU Collection of paper Special Edition in Celebration of the 10th Anniversary, pp87-108, 2004

[5] Carl. Burch, "Logisim: A Graphical System for Logic Circuit Design and Simulation" Journal of Educational and Resources in Computing, Vol. 2, NO. 1, March 2002.

[6] 김은주, 류승필, “ 교육용 디지털 논리회로 시뮬레이터 설계 및 구현”, The Journal of Korean Association of Computer Education, Vol. 11, No. 2, March 2008.

[7] Ursula S초듀두 “Hierarchical composition of industrial components", Science of Computer Programming 56, pp117-139, 2005.