論文 91-28A - 1 - 10

성능 구동 논리 회로 자동 설계 시스템

(Performance-driven Automatic Logic Synthesis System)

李 在 亨*, 黃 善 泳**

(Jae Hyeong Lee and Sun Young Hwang)

요 약

논리 회로 설계에 있어 기술 의존적인 최적화와 테크놀로지 매핑을 위한 알고리듬을 제안하고, 이에 바탕을 두어 구현한 성능 구동 자동 논리 설계 시스템인 SiLOS에 대하여 기술한다. SiLOS는 시스템 라이브러리에서 제공되는 게이트 타입으로 회로를 재구성하고 가능한 적은 면적에서 동작 속도 사양을 만족하는 회로를 자동적으로 생성한다. 실험 결과 구현된 시스템이 고성능의 논리 회로 설계를 위해 설계자에게 유용한 툴이 될 수 있음을 보였다.

Abstract

This paper presents an algorithm for technology-dependent logic optimization and technology mapping, and describes a performance-driven logic synthesis system, SiLOS, implemented based on the proposed algorithm. The system analyzes circuits and resynthesizes the critical sections such that generated circuit operates within time constraints, using only gate types supported by library for direct implementation.

Experimental results show that the system can be a viable tool in synthesizing high-performance logic circuits.

I. 서 론

최근 VLSI 칩의 집적도가 크게 향상되어 넓은 응용 분야에서 요구하는 기능의 회로를 하나의 칩으로 구현 가능하게 되었으며, 단일 칩에 구현하려고 하는 회로의 기능도 더욱 복잡해지는 추세를 보이고있다. 한편 단기간에 방대한 양의 정보 처리를 위해서는 회로의 동작 성능 측면에서 또한 고속의 처리 속도가 요구되고 있으므로 논리 회로의 설계시 주어진 사양의 동작 속도로 동작하며 가능한한 최소의 면적으로 구현하기 위하여 알고리듬과 시스템에 대한 연구가 계속되고 있다.[1][3][4][6]

최소화된 면적상에서 회로를 구현하는 것보다 더욱 중요한 문제는 회로의 동작 속도를 주어진 시간 제약 조건에 맞도록 설계하는 일이다. 특히 동기 회로에서 기억 소자간의 지연 시간이 긴 조합 회로가 존재할 경우 동기 회로의 동작 속도는 그 조합 회로에 의하여 결정되므로, 설계자는 주어진 시간 제약 조건을 만족하는 회로를 구현하기 위한 노력을 경주한다. 이 경우 주어진 시간 제약을 만족하는 회로를 사람이 설계하기 위해서는 많은 설계시간과 trial-and-error 과정이 요구되며, 사용되는 논리 라이브러리에 따른 최적화된 회로를 구현하기는 더욱 어려운

*準會員, **正會員, 西江大學校 電子工學科 CAD및 컴퓨터 시스템 硏究室

(CAD & Computer Systems Lab., Dept. of Elec. Eng., Sogang Univ.)

接受日字 : 1990年 7月 19日

일이다. 따라서 시간 제약내에 동작 가능한 최소 면적의 회로를 설계하는 자동화된 시스템의 개발이 요구된다.

SiLOS(sogang intelligent logic optimization system)[10][11]는 이러한 목적으로 개발된 성능 구동 논리 최적화 시스템으로, 그 구성도를 그림 1에 보였다. SiLOS는 VHDL의 구조 기술 혹은 중간 언어인 SLIF를 입력으로 받아들여 회로 정보를 추출한 후 사용자의 요구에 따라 (1) 주어진 라이브러리로 구현 가능하고 (2) 시간 제약 조건을 만족하며 (3) 가능한 최소의 면적으로 회로를 재설계하는 기능을 가진다. 회로의 재설계 방식은 기술 독립적인 최소화 과정과 기술 의존적인 최적화 과정으로 구성된다.[10][11] 기술 독립적인 최소화 과정은 회로 동작에 영향을 미치지 않는 게이트를 제거함으로써 불필요하게 낭비되는 면적을 줄이는 과정이며, 이 과정을 통하여 생성된 결과를 기술 의존적인 최적화 과정에서 사용하여 회로의 최적화 정도를 높일 수 있다. 기술 의존적인 최적화 과정은 회로에서 시간 제약 조건을 만족하지 않는 부분에 대하여 회로 변환을 이용한 재설계를 함으로써 회로의 동작이 시간 제약내에 이루어지도록 하며, 이 과정에서 라이브러리가 제공하는 게이트 타입만으로 회로를 재구성함으로써 결과로 얻어진 회로가 직접 구현 가능하도록 할 수 있다. 회로의 변환은 동작 속도를 향상시킬 수 있으며, 보다 작은 면적에서 구현이 가능한 방향으로 적용된다.

본 논문에서는 SiLOS의 기술 의존적인 최적화 과정에 대하여 설명하도록 한다. Ⅱ절에서는 회로의 각 경로가 시간 제약 조건을 만족하는 정도를 계산하는 과정을 설명하고, Ⅲ절에서는 기술 의존 최적화 과정에서 적용하는 회로 변환 방법에 대하여 기술한다. Ⅳ절에서는 회로 변환 대상 게이트에 대한 적용 방법의 선택과 적용 방식, Ⅴ절에서는 이 과정에서 사용되는 라이브러리의 관리 방식을 설명하고 Ⅵ절에 실험결과를 보였다.

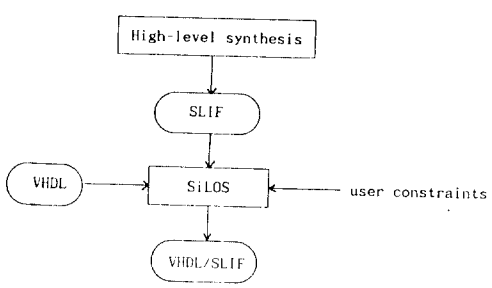

그림 1. SiLOS 구성도
Fig. 1. SiLOS overview.

Ⅱ. 회로 제약 조건

구현하고자 하는 회로는 설계자가 디자인하거나 논리 합성기를 통하여 생성된 것 또한 이들의 결과가 기술 독립적인 최소화 과정을 거친 결과일 수 있다. 이러한 회로는 모두 실제로 구현 가능한 기술과는 무관하게 설계 혹은 재설계되어진 것이므로 기술 의존적인 재설계를 통하여 구현 가능한 회로로 변환되어야 한다. 이때 변환 과정에서 회로 제약 조건을 만족하도록 회로를 설계하여야 한다.

회로에 대한 제약 조건은 전체 회로의 동작 시간과 면적 측면에서 고려할 수 있으며 이들은 서로 trade-offs의 관계를 갖고 있어 회로에서 동작 시간과 밀접하게 관련된 부분은 면적을 늘려서라도 동작 속도를 높이도록 하고, 그렇지 않은 부분은 전체 동작 속도에 영향을 미치지 않는 한도 내에서 면적을 줄이는 방법을 취한다.

전체 회로의 동작 속도에 관한 제약의 만족 여부는 회로 각 단자에서의 slack 값으로 판정되며, 이 slack 값은 그림 2의 알고리듬으로 계산된다.[5] Slack 값이 가지는 의미는 어느 단자를 구동하는 게이트 또는 primary input에서 그 단자까지의 경로를 구성하는 게이트에 대해 시간 제약을 침해하지 않으면서 더해질 수 있는 지연시간의 양을 나타낸다. Slack값이 음의 값을 가지는 경우는 시간 제약내에 그 단자의 출력값이 형성되지 않음을 나타내며, 출력값이 형성되는 시간은 그 단자의 신호 대기시간(data ready time)으로 표현한다. 임계 경로(critical path)는 이 slack 값이 음인 단자들을 지나는 경로를 말하며 이 임계 경로가 회로의 동작 속도를 결정한다. 그림2에 slack 계산을 위한 알고리듬을 기술하였다.

신호 대기 신호의 계산 루틴에서 Topological-order ( )는 회로의 각 단자에 대해 신호 대기 시간을 계산할 순서를 정하기 위하여 회로를 levelize하는 루틴이다. 한 단자의 신호 대기 시간의 계산을 위해서는 그 단자를 구동하는 게이트의 입력단의 신호 대기 시간이 미리 계산되어 있어야 하며 신호 대기 시간의 계산 순서는 primary input에서 primary output으로 진행하는 순서를 가진다. Slack 값의 계산순서는 신호 대기 시간 계산의 역순이 되므로 신호 대기 시간의 계산이 끝난 단자는 스택에 저장하여 slack의 계산 루틴에서 사용하도록 한다.

단자 j의 신호 대기 시간은 j단자를 구동하는 게이

```
*
*G=(V, A):circuit graph
*Topological-order( )makes a node sequence in topological order
*gate j has output node j
*R(j):data ready time of node j
*D(j):delay of gate j
*K:fanins of gate j
*/
  procedure    READY-TIME(G)
    begin
      N=Topological_order(G);
      Stack=φ;
      while ‖N‖ 〈〉 0  do /*‖N‖;size of set N */
        begin
          j=First node in N;
          if j is primary input
            then  R(j)=0;
          else R(j)=D(j)+Max [R(j)];
                        k∈k
          ‖N‖=‖N‖-1;
          PUSH(Stack, j);
        end ;
    return(Stack);
  end;
/*
*G=(V, A):circuit graph
*J={j | (Ni, Nj)=A}
*  gates that node Ni drives
*K={N | fanins to gate j including Ni}
*S(Ni)=slack value of node Ni
*/
  procedure  Slack(G)
    begin
      Stack=READY_TIME(G);
      while Stack is not empty do
        begin
          Ni=POP(Stack);
          S(Ni)=Min [S(Nj)+Max[R(Nk)]-R(Ni)]
                j∈J          k∈k
        end;
    end;
```

그림 2. Slack의 계산 알고리듬
Fig. 2. Algorithm for slack calculation.

트 j의 입력단 k(k∈K:fanins to gate j)가 가지는 신호 대기 시간 R(k)중 최대값에 게이트 j의 지연 시간 D(j)를 더하여 계산한다. 단자 Ni의 slack 값은 Ni가 구동하는 게이트 j(j∈J:Ni가 구동하는 게이트 집합)의 입력단 Nk(Nk∈K:fanins to gate j)들이 가지는 신호 대기 시간 R(Nk)중 최대값에서 단자 Ni의 신호 대기 시간 R(Ni)를 감하고 여기에 게이트 j의 출력단 Nj의 slack 값 S(Nj)를 더한 값을 구하여, 이 값의 최소값으로 결정한다.

Ⅲ. 기술 의존 최적화 방법

구현 가능한 회로를 얻기 위해서는 회로상의 게이트를 라이브러리에서 제공하는 게이트 형태만으로 변환하여야 하며 라이브러리에서 지원되지 않는 게이트들은 cell generator를 통하여 생성하거나 혹은 회로내에서의 기능은 같으면서 다른 형태를 갖는 일련의 게이트 조합으로 대치할 수 있다. 이 절에서는 회로의 최적화가 고려된 technology mapping 방법에 대해 기술한다.

이후의 설명을 용이하게 하기 위하여 회로내의 단순 게이트 i, j에서 게이트 i가 게이트 j의 입력단을 직접 구동하는 관계에 있을때 게이트 i를 'tail gate', 게이트 j를 'head gate'라고 정의한다. Tail gate의 출력은 head gate를 구동하며 하나의 head gate는 입력의 수만큼의 tail gate 집합을 가질 수 있다. Head gate와 tail gate들이 이루는 회로를 하나의 게이트로 구현할 경우 이를 compound gate라 하며, 단순 게이트를 하나의 head gate와 몇 개의 tail gate로 구성하는 과정을 gate decomposition 이라 한다.

1. Gate Compounding

설계된 논리 회로의 구현에 있어 각 logic family마다 독특한 기법으로 복잡한 회로를 하나의 게이트로 구현하여 제공하는 compound gate가 존재한다. 이러한 compound gate들은 단순 게이트만으로 구성된 논리 회로가 차지하는 면적 및 지연 시간보다 작은 값을 갖고 동작하므로 시간과 면적상에서 효과적이다. 예를들어 LSI Logic사의 HCMOS family는 2-input NAND 게이트를 head gate로하고 2개의 2-input OR 게이트를 tail gate로 하여 하나의 compound gate (A04)로 제공하는데 fanout이 1인 경우 지연 시간은 4nS이며 8개의 트랜지스터로 구성된다. 그러나 각각의 단순 게이트로 구현하는 경우에는 모두 12개의 트랜지스터와 5.8nS의 지연 시간이 소요된다.[8]

따라서 회로의 일부가 compound gate로 대치 가능하다면 이들을 사용하여 임계 경로를 제거하거나 면적을 줄일 수 있으며, 두가지 방법의 gate compounding이 가능하다. 먼저 대상 게이트의 출력 단자가 구동하는 게이트를 head gate로 하며 대상게이트를 포함하여 head gate의 입력단을 구동하는 게이트 집합을 tail gate들로 하는 compound gate의 대치 방법

(forward compounding)과 대상 게이트를 head gate로 하고 그 입력단을 구동하는 게이트 집합을 tail gate들로 하는 방법이다(backward compounding) 그림 3에 gate compounding의 예를 보였다.

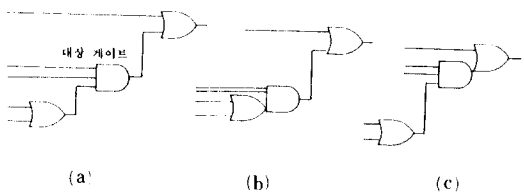

그림 3. 게이트 compounding (a) 대상 회로
(b) Backward compounding 수행 결과 회로
(c) Forward compounding 수행 결과 회로

Fig. 3. Gate compounding. (a) given circuit,
(b) circuit after backward compounding,
(c) circuit after forward compounding.

2. Gate Decomposition

입력단이 많은 게이트는 구현 기술에 따라 구현이 불가능하거나, 자체의 큰 지연 시간에 의해 임계 경로부를 형성할 수 있다. 여러 입력단 중에서 어느 특정한 입력단을 구동하는 경로가 임계 경로일 경우를 가정하자. 그림 4의 (a)에서 A의 입력과 B의 입력의 slack 값이 다른 입력단에 비하여 작게 나타나 있다. 이 경우 그림 4의 (b)와 같이 A, B 입력을 분리하여 2개의 게이트 조합으로 회로를 구현할 경우 임계 경로의 지연 시간을 줄여 최종 출력단 Z의 slack 값이 커지게 되므로 임계 경로부를 제거할 수 있다.

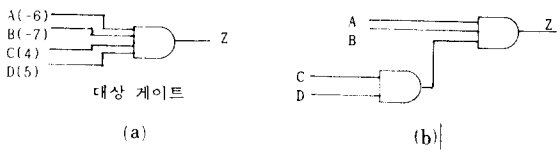

그림 4. 게이트 decomposition (a) 초기의 회로
(b) 두개의 게이트로 decomposition된 결과

Fig. 4. Gate decomposition. (a) initial circuit,
(b) circuit decomposed into two gates.

Decomposition의 다른 목적은 forward compounding의 적용 범위를 넓혀주는데 있다. 예를 들어 라이브러리에서 하나의 2-input NOR gate와 하나의 3-input AND gate를 각각 head gate와 tail gate로 하는 compound gate (A0131 of Synopsis library[9])를 제공하는 경우 그림 4(a)의 출력단 Z가 2-input NOR gate 를 구동한다면 decomposition에 의해서 생겨난 그림 4(b)의 3-input AND gate와 2-input NOR gate가 조합되어 그림 3의 (c)와 같은 형태를 가지는 gate compounding이 가능하다. 하지만 라이브러리에서 3-input AND gate를 제공하지 않을 경우 A0I31과 같은 회로를 형성하는 것은 고려되지 않을수도 있는데, 이는 decomposition의 가능 탐색 범위를 제한하기 때문이다.

3. Gate Inversion

Gate inversion은 두가지 목적을 갖는다. 하나는 대상 게이트의 극성(gate polarity)을 바꿈으로써 라이브러리가 한가지의 게이트 극성만을 제공하는 경우 (NAND-NOR, AND-OR)구현 가능하도록 하며, 라이브러리가 양쪽 모두의 극성을 제공하는 경우에 대상 게이트 앞단의 불필요한 인버터를 제거하거나 forward compounding 또는 backward compounding의 범위를 확대하기 위해서이다.

Gate decomposition과 gate compounding 방법은 주어진 게이트 타입으로 국한하여 라이브러리에 존재하는 게이트 형태로 맞추어가는 과정이므로 적용 범위가 제한되는 단점을 갖는다. 따라서 이 방법의 적용 범위를 보다 넓히기 위하여 대상 게이트의 극성을 바꾸어 보는 방법을 사용하며 다음과 같은 4가지의 변화가 가능하다.

NAND → AND+인버터
AND → NAND+인버터
NOR → OR+인버터
OR → NOR+인버터

4. Gate Conversion

Gate inversion과 마찬가지의 목적을 위해 게이트의 형태를 바꾸는 방법으로 다음과 같은 게이트의 conversion을 생각할 수 있으며 AND를 OR대신 NOR로 conversion한 이유는 게이트를 inversion한 이후에 conversion한 경우 또는 그 역의 방법을 사용하려 하는 경우에 출력단에서 인버터의 첨가와 제거가 반복적으로 일어나는 부담을 없애기 위해서이다.

NAND → OR+입력단에 첨가되는 인버터
AND → NOR+입력단에 첨가되는 인버터
NOR → AND+입력단에 첨가되는 인버터
OR → NAND+입력단에 첨가되는 인버터

Ⅳ. 기술 의존 최적화 방법의 적용

회로의 기술 의존 최적화는 전체 대상 범위 중 특정 게이트를 선택하여 앞에서 제시한 여러 변환 방법 중의 하나 또는 일련의 방법들을 적용하여 수행한다. 이 경우 어떤 방법을 적용할 것인가 하는 문제가 발생하며, 선택된 방법의 적용에 의해 나타나는 결과는 각기 다를뿐만 아니라, 그 이후의 적용 결과에도 영향을 준다. 예를들어 한 게이트를 conversion 시킨 결과가 그 게이트에 대해서는 최적의 결과를 가져올 수는 있지만, 다음 대상의 게이트에서는 compounding이 불가능하게 되는 결과가 될 수 있다.

제시된 방법의 선택 및 적용의 문제점을 해결하기 위해 한 게이트와 그 전후의 게이트들의 관계를 살펴 최적의 방법을 선택하는 방식이 요구된다. Rule-based 방식[2]과 simulated annealing[7]방식은 모두 국소 범위의 최적화로 인한 단점을 해결하기 위하여 제안 되었지만 simulated annealing 방식의 경우 전체 범위의 최적화를 얻기 위한 계산 시간이 너무 긴 단점을 가지며, rule-based 방식은 rule의 적용 범위를 너무 깊게 할 경우 기술에 따라 방대한 양의 rule 적용을 요구하고 pattern matching에 많은 시간을 소비하게 된다.

따라서 SiLOS에서는 제시된 선택 및 적용을 위해 한 게이트의 전후 관계만을 대상으로 하여 가능한 많은 경우에 대해 고려하고, 이때 동작 시간과 면적

```
/*
 * G=(V, A):circuit graph
 */
  procedure  Technology_Mapping(G)
    begin
      for each gate g in G do
        if type of gate g is inverter then
            mark it;
      while unmarked gates do
        begin
          Select a target gate;
          Preprocessing;
          Construct the search tree;
          Determine the rule application sequence;
          Transform the circuit;
      end;
    end;
```

그림 5. 기술 의존 최적화 알고리듬
Fig. 5. Algorithm for technology-dependent optimization.

중 특히 동작 시간에 중점을 두어 최대의 효과를 얻는 방법을 선정하는 방식을 통하여 빠른 시간 내에 적용되어질 일련의 방법을 선정하는 것에 목적을 두었다. 이 방식은 여러단의 논리 회로에 대해 제시된 방법의 적용의 결과를 미리 살펴봄으로써 보다 최적화된 회로를 얻을 수 있다. 그림 5에 최적화가 고려된 technology mapping 과정의 전체적인 flow를 보였다.

그림 5의 기술 의존 최적화 알고리듬은 먼저 인버터를 제외한 모든 게이트가 기술 구현이 되지 않은 것으로 가정하여 인버터만을 marking 한다. 이후에 mark되지 않은 게이트들에 대해 변환 대상 게이트의 선정, 전처리(preprocessing), 탐색 트리의 구성, 적용 방법의 결정 및 회로 변환 과정을 통하여 기술 구현과 최적화를 이루도록 한다. 다음에 여기서 사용된 각 루틴을 설명하였다.

1. 변환 대상 게이트의 선정

회로의 최적화와 기술 구현은 선택되는 게이트의 순서에 관련되어 그 결과가 달라질 수 있다. 따라서 한 게이트에 대해 적용 가능한 방법들 중에서 실제 적용할 방법의 결정 이외에 적용 대상 게이트의 선택도 결과에 많은 영향을 미치므로 주의깊은 고려가 필요하다. 게이트의 선택 방법은 임의로 선택하는 방법과 회로의 변환이 효과적으로 이루어질 수 있는 부분을 선택하고 나중에 구현되지 않은 게이트를 선택하는 방법이 있으나, 전자는 적용된 방법에 의한 최적화가 효과적으로 이루어지지 않을 수도 있으며, 후자는 변환을 위한 게이트 선택에 많은 계산 시간이 요구되어 비효율적이다.

따라서, 임계 경로부를 우선으로 하여 출력단에서 입력단으로 전파하며 게이트를 선택하는 방법을 취한다. 이 방법은 일단 선택된 게이트의 출력단 측은 기술 구현이 된 상태를 가정할 수 있어; 선택된 게이트에 대한 기술 구현만을 고려하여 최적화를 이루는 방법을 찾으므로 수행상의 부담을 줄일 수 있고 대상 게이트의 선정이 간단하다. 이 방식에서 탐색 영역이 제한되는 단점은 적용 방법의 탐색 영역을 대상 게이트의 입력단에 대해 가능한 모든 경우를 고려함으로써 해결할 수 있다.

적용된 방법에 의한 최적화가 효과적으로 이루어지지 않을수도 있으며, 후자는 변환을 위한 게이트 선택에 많은 계산 시간이 요구되어 비효율적이다.

대상이 되는 게이트 집합은 AND, NAND, OR, NOR 형태만으로 한정하며 인버터를 포함한 그 밖의 게이

트는 포함하지 않는다. 인버터는 게이트의 전처리 과정 및 후처리 과정에서 최적화될 수 있으며 그 자체로서 적용 가능한 방법을 고려하지 않았다. 라이브러리는 항상 인버터를 제공한다고 가정하여 인버터는 이미 기술 구현이 된 것으로 본다. 선정된 변환 대상 게이트는 가장 큰 효과가 기대되는 일력의 방법을 적용한 후, 이 게이트에 대해 marking을 하여 반복적으로 변환 대상이 되지 않도록 한다.

2. 대상 게이트의 전처리 과정

대상 회로에 효과적인 방법의 적용을 위하여 전처리 과정을 거치게 되며, 이 과정에서 대상 게이트의 입력, 출력단에 존재하는 인버터의 제거와 게이트 병합(merging)을 수행하게 된다. 인버터들은 원래의 논리 회로에 존재하거나 혹은 다른 게이트가 변환된 결과로 생성되어, 다른 게이트에 대한 변환이 원활하게 되지 못하도록 방해하는 경우가 있기 때문이다. 주어진 논리 회로는 그림 6의 (a) (b) (c)와 같이 인버터가 있을 수 있으나, 이들은 모두 전처리 과정을 거쳐 그림 6의 (d)와 같은 형태로 바뀌거나 상쇄되어 제거된다. 다음에 설명할 inspector routine은 모두 그림 6의 (d)와 같은 형태로 인버터가 존재하는 것을 가정하게 된다. 한가지 형태의 인버터 상태는 inspector가 복잡하게 존재하는 인버터에 대한 고려없이 제시된 방법이 실제 적용되었을 경우의 상태를 쉽게 예측할 수 있으므로 효율적이다.

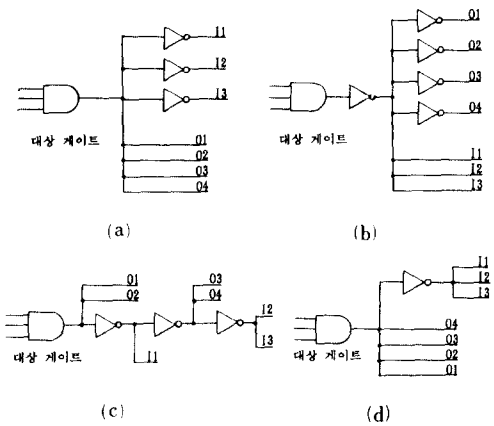

그림 6. 인버터에 대한 전처리 과정
(a) (b) (c) 초기의 인버터 분포 상태
(d) 전처리 결과

Fig. 6. Preprocessing for inverters.
(a) (b) (c) initial inverter distribution,
(d) circuit after preprocessing.

그림 7과 같은 회로에 대해서는 앞에서 제시한 방법만으로는 효율적인 재설계가 불가능하다. 작은 fanin 갯수를 갖는 게이트에 대해서는 gate decomposition이 적용되기 어렵고 단지 inversion과 conversion에 의한 기술 구현만이 가능하며 최적화는 gate compounding을 고려할 수는 있으나 효과적으로 적용되지 못할 수도 있다. 그러므로 게이트에 적용 가능한 탐색 영역을 넓히기 위해 게이트 병합 방법을 사용한다. 게이트 병합은 해당 게이트의 입력단에 게이트 타입과 같은 타입의 게이트가 존재할 때 이를 하나의 게이트로 만들어준다. 게이트 병합의 과정은 그림 8에 나타나 있으며 gate decomposition과 반대의 기능을 갖는 이유로 기술 구현및 최적화 방법에 포함하지 않았다. 만약 게이트 병합을 최적화 방법에 포함한다면 gate decomposition과의 충돌로 불필요한 반복적 변환 작업이 생겨날 수 있다.

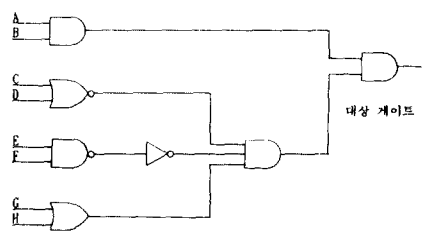

그림 7. 게이트 병합 이전의 회로

Fig. 7. Circuit before gate merging.

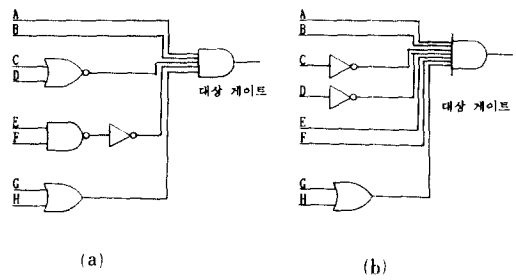

그림 8. 게이트 병합 과정
(a) 그림 7의 AND 게이트를 병합한 회로
(b) (a) 회로를 다시 병합한 결과

Fig. 8. Gate merging process.
(a) circuit obtained by merging AND gates of fig. 7,
(b) after gate merging from circuit in (a).

위와 같은 과정이 끝나면 회로의 각 단자의 slack 값을 재계산하여 다음의 변환이 원활하게 되도록 한다.

3. 탐색 트리의 구성

회로 최소화 과정을 거친 회로로부터 주어진 라이브러리의 logic family로 재구성된 최적화 회로를 얻기 위하여 회로내 각각의 게이트를 대상 게이트로 하여 하나 또는 여러개의 방법을 적용한다. 이때 적용되는 방법의 순서를 '적용 순서'라고 한다. 적용 순서에 따라 나타나는 회로의 최적화 정도는 서로 다를 수 있으므로 보다 최적화된 결과를 얻기 위해서는 대상 게이트에 적용될 모든 적용 순서들의 결과를 미리 살펴본 다음 가장 최적화된 결과를 얻을 수 있다고 예상되는 적용 순서를 선정하여 적용하여야 한다.

적용 순서를 이루는 방법들은 한 대상 게이트에 대하여 다시 적용되지 않는다. 즉, 어느 순간 적용 가능한 방법의 집합을 Rp라 할 때, 방법 r(r∈Rp)의 적용 후 다음번에 적용 가능한 방법의 집합 Rs는

$$Rs = Rp - r - Rn(r)$$

의 관계를 갖는다. 집합 Rn(r)은 방법 r이 적용되으로써 적용 불가능해지는 방법의 집합을 나타내며 각 방법에 대해 다음과 같이 결정된다.

$$Rn(\text{inversion}) = \{\phi\}$$
$$Rn(\text{conversion}) = \{\phi\}$$
$$Rn(\text{decomposition}) = \{\text{inversion, conversion}\}$$
$$Rn(\text{compounding}) = \{\text{inversion, conversion, decomposition}\}$$

초기의 Rp를 Rp={inversion, conversion, decomposition, compounding}으로 하며 각 방법을 노드로 하여 표현할 경우, 대상 게이트에 가능한 적용 순서는 그림 9의 트리 구조를 이루게 된다.

최적의 적용 순서를 선정하기 위해서는 이 트리에서 모든 노드에 이르는 방법이 적용되었을 경우에 대해, 동작 속도의 향상치와 면적의 감소분을 미리 살펴 보아야 한다. 이러한 작업은 개개의 방법이 가지는 inspector routine들을 통하여 수행된다.

Inspector routine은 그와 관련된 방법의 적용 가능성을 살펴보기 위해 대상 게이트 주변의 회로상태를 검색하며, 만일 한 방법의 적용이 가능할 경우 어떠한 형태로 게이트 변환이 가능한가를 판단하여 각 게이트 변환에 대한 정보와 그 방법이 적용되어 변환되었을 때, 예상되는 동작 속도의 향상치와 면적

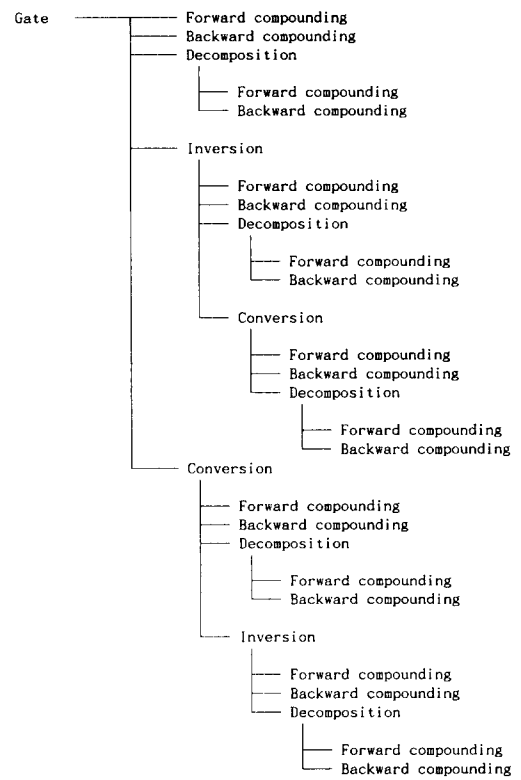

그림 9. 적용 순서의 트리 형태
Fig. 9. Tree form of rule sequence.

의 감소분을 특정한 자료구조에 저장한다. 이 과정에서 주변 회로에 대한 정보를 자료 구조에 함께 저장함으로써 실제로 그 방법이 적용 순서로 선정 되었을 때 회로의 변환을 위하여 주변회로를 검색하는 부담을 줄일 수 있다. 또한 inspector routine은 Rs (다음에 적용 가능한 방법의 집합)와 관련된 inspector routine들을 호출하여 이들이 넘겨주는 정보를 첨가하여 자료 구조를 완성하며, 이 자료 구조를 가리키는 포인트를 넘겨주게 된다.

대상 게이트에 대하여 inspector routine들이 이루는 전체 자료 구조는 그림 9와 비슷한 형태의 트리 구조를 갖게되며 4절에서 설명할 적용 순서의 결정에 필요한 탐색 영역을 이룬다.

4. 적용 순서의 결정과 회로의 변환

적용 순서의 선택은 탐색 트리의 각 노드에서 제공하는 동작 속도의 향상치와 면적의 감소분을 참조하여 이 두가지를 변수로 하는 비용 함수에 의한 가치값을 계산한 다음, 루트에서 내부 노드에 이르는 경로의 가치값의 합이 최대인 노드를 찾아 그 경로

상에 있는 모든 노드의 플래그를 setting하여 이루어진다. 루트로부터 플래그가 set된 노드를 따라 선정된 방법을 적용하여 대상 게이트에 대한 기술 구현과 최적화를 마치며, 새로이 생성된 게이트와 제거되는 게이트에 관한 정보를 테이블에 저장하여 design mode에서 설계자에게 제공할 수 있도록 한다.

V. 라이브러리

기술 구현에 사용되는 라이브러리는 다음과 같은 양식으로 제공한다.

simple gate: $ [gate type] [gate typename] [delay] [area]

compound gates: # [head-gate] [tail-gate1 pin-pos1 tail-gate2 pin-pos2…] [gate typename] [delay] [area]

example : $ and3 AND3 5 10
$ or4 OR4 8 16
# or2 and2 0 and2 2 and2 4
AOI31 15 20

여기서 gate type은 and3, nor5 또는 nand2와 같은 형태로 기능과 fanin의 수를 명시하며, gate typename은 해당 module의 고유 이름을 나타낸다. Compound gate는 구성되는 각각의 단순 게이트를 명시하고 입력 단자의 위치를 지정하여 compound gate의 논리 구성에 오류가 없도록 한다.

단순 게이트는 TYPE-SIMPLE 테이블에 저장되며 각 게이트 타입에 대하여 하나의 bucket을 할당하였다. 각 bucket에는 특정 게이트의 면적과 지연 시간및 고유 이름을 갖는 자료 구조를, fanin의 갯수가 작은 것에서 큰 순서로 sorting하여 liked list로 저장함으로써 적용 순서의 설정및 적용 과정에서 용이하게 사용하도록 하였다. Compound gate는 TYPE-COMPOUND 테이블에 head gate의 게이트 타입에 따른 bucket을 설정하고 그 아래 linked list로 형성된 자료 구조에 head gate의 fanin 갯수와 면적, 지연 시간, 고유 이름 및 compound gate의 fanin 수를 기록하며, compound gate가 요구하는 tail gate의 종류를 listed list로 저장한다. 이 자료 구조는 gate compounding의 pattern matching과 그에 관련된 적용 순서 설정 및 선정된 방법의 적용에 사용하도록 한다.

사용되는 라이브러리에 따라 TYPE-SIMPLE 테이블의 일부가 비어 있을 수도 있다. 이 경우 적용방법의 선택 및 적용 과정에서 요구되는 게이트의 지연 시간과 면적에 관한 정보는 같은 fanin을 가지는 다른 게이트 타입의 지연 시간과 면적을 고려하여 정하거나, fanin의 변화에 따르는 지연 시간과 면적의 변화율을 고려하여 정하도록 한다. 지연 시간과 면적의 정보는 적용 순서의 선택에 있어 매우 중요한 요소가 되므로 가급적 실제의 경우와 같도록 하여야한다.

VI. 실험 및 고찰

LSI Logic 사에서 제공하는 라이브러리의 논리소자를 사용하여 실험을 수행하였다. 라이브러리의 논리 소자중 실험을 위해 채택된 각 논리 소자들의 지연 시간과 면적을 그림10에 보였다. 지연 시간의 단위는 0.1nS이며 면적은 논리 소자를 구성하는 트랜지스터들의 갯수를 기준으로 하였다. 사용된 compound gate의 형태는 그림11에 보였다.

게이트 타입	면적(# transistors)	지연시간(0.1nS)
NOT	2	20
NAND2	4	24
NAND3	6	31
NOR2	4	34
O2Nd2	6	38
A2Nr2	6	49
AO4	8	40
EON1	10	38

그림10. LSI Logic사에서 제공하는 게이트들 중에서 라이브러리로 채택한 게이트들의 특성

Fig. 10. A subset of LSI Logic library.

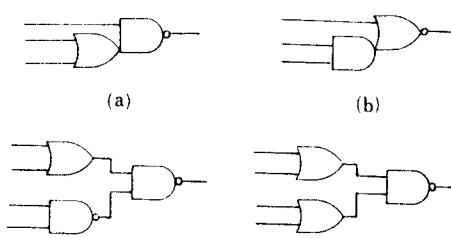

그림11. 사용된 compound gate의 형태
(a) O2Nd2 (b) A2Nr2
(c) EON1 (d) AO4

Fig. 11. Topology of used compound gates.
(a) O2Nd2, (b) A2Nr2,
(c) EON1, (d) AO4.

라이브러리에서 제공되는 게이트 타입만으로 회로가 재구성되는 과정과 라이브러리에 따른 회로의 최적화 정도를 살펴보기 위하여, 그림10에서 제시한 게이트 타입의 일부를 제한하거나 첨가하는 방법을 취하였다. 27개의 게이트로 이루어진 그림12의 '태스트 회로 I'을 대상 회로로 하여, 그림10의 게이트 타입 중 EON1과 AO4를 제외하고 기술 의존 최적화를 이룬 회로를 그림13에 보였으며, 그림10의 모든 게이트 타입을 사용하여 그림14의 회로를 얻었다. 그림10에서 EON1과 A04를 제외하고 AND와 OR의 게이트 타입을 추가하여 최적화된 결과를 그림15에, 그림10의 모든 게이트와 AND, OR게이트 타입을 사용한 경우의 최적화를 그림16에 보였다. 회로의 각 단자에 나타난 숫자는 slack 값을 의미하고 강조된

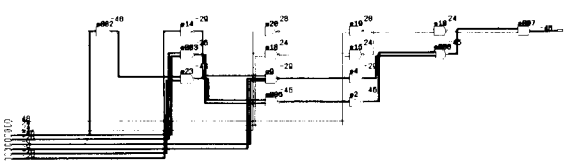

그림12. '대상 회로 I'
Fig. 12. 'Test circuit I.'

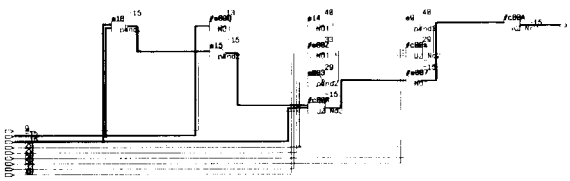

그림13. 그림10의 라이브러리중 AO4와 EON1을 제외하여 구현한 회로
Fig. 13. Circuit implemented using library of fig. 10 except for AO4 and EON1.

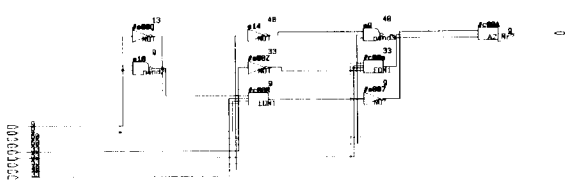

그림14. 그림10의 라이브러리를 사용하여 구현한 회로
Fig. 14. Circuit implemented using libray of fig. 10.

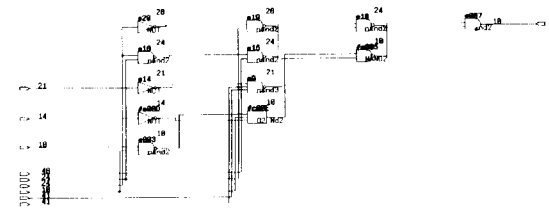

그림15. AND, OR 게이트 타입과 그림10의 라이브러리중 AO4와 EON1을 제외한 게이트 타입으로 구현한 회로
Fig. 15. Circuit implemented using AND, OR and the gate types in library of fig. 10 except for AO4 and EON1.

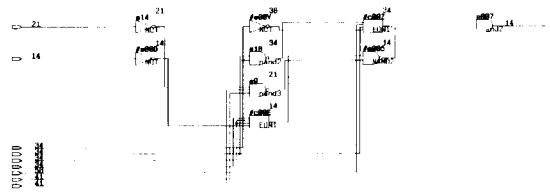

그림16. AND, OR 게이트 타입과 그림10의 라이브러리에서 제공하는 게이트 타입으로 구현된 회로
Fig. 16. Circuit implemented using AND, OR and the all gate types in library of fig. 10.

선은 임계 경로를 뜻한다. Slack 값은 primary input에서 primary output까지의 시간 제약 조건을 14 nS로 하여 0.1 nS 단위로 계산된 값이며, 임계 경로상의 slack 값은 음의 값을 가지고 있음을 알 수 있다. 그림12의 대상 회로는 primary output 단자의 slack 값이 −46(4.6 nS)이며 사용된 트랜지스터 수는 54개이다.

주어진 게이트 타입 중 EON1과 A04의 사용을 제한한 경우 기술 구현된 그림13의 회로에서 primary output의 slack 값은 −15로 향상되며, 44개의 트랜지스터만으로 구현 가능하다. 이 회로의 기술 구현 과정은 먼저 대상 회로의 AND 게이트 e007이 라이브러리에서 제한하는 게이트 타입인 NOR로 변형된 후 OR 게이트 e008이 inversion 및 conversion된 다음 NOR 게이트와 forward compounding에 의해 A2Nr2의 compound gate로 된다. 그 후에 선택된 게이트 e19가 conversion되어 게이트 e10과 compounding되고, 게이트 e23도 마찬가지로 conversion되어 게이트 e006과 compound gate 02Nd2를 형성하여 최적화

를 마친다.

그림10의 모든 게이트를 사용한 경우의 결과는 그림14에 보였다. 이 회로의 primary output slack값은 9로 향상되었고, 면적은 트랜지스터 수 48로 나타났다. 그림13의 회로에서 보다 성능이 향상된 이유는 그림12의 게이트 e10, e15, e19가 compound gate EON1을 형성하며 또한 게이트 e006, e003, e23 도 EON1으로 compounding 되었기 때문이다. 결과에서 각 단자의 slack 값을 살펴보면 대상 회로의 임계 경로를 이루지 않던 경로의 slack 값이 더 작아짐을 볼 수 있다. 이는 회로의 변환 과정에서 gate compounding으로 인하여 임계 경로의 지연 시간이 감소하는데 비해 비임계 경로의 지연 시간이 늘어나기 때문이다.

LSI Logic family는 AND와 OR의 게이트 타입을 제공하지 않으므로, AND와 OR가 제공되는 경우의 회로 최적화 정도를 알아보기 위하여 AND와 OR 의 게이트 타입이 NAND+인버터, NOR+인버터로 구성되는 것을 가정하여 그림15와 그림16의 회로를 얻었다. 그림15는 EON1과 A04의 게이트 타입을 제한하여 얻은 것이다. 라이브러리에서 제공되는 게이트의 다양성에 따라 회로 최적화의 결과는 각기 다름을 그림17의 실험 결과에서 알 수 있다. 그림17의 괄호안에 표기된 수치는 원래 회로의 면적과 수치를 나타낸다. '테스트 회로 I'에서의 최적화는 주로 gate inversion 및 gate conversion 이후 gate compounding에 의해 이루어졌으며, 그 이유는 fanin이 작은 게이트들로 회로가 구성되었기 때문이다.

그 밖의 '테스트 회로 II'와 '테스트 회로 III'에 대해 기술 의존적인 최적화 과정을 수행하여 얻어진 결과를 그림18에 보였다. 테스트 회로 II'는 fanin이 큰 게이트들로 이루어진 회로이며, '테스트 회로III'은 fanin의 크기가 서로 다른 게이트들로 이루어진 회로이다. '테스트 회로 II'의 경우 gate decomposition에 따른 gate compounding에 의해 최적화가 이루어졌다. Gate decomposition은 slack 값이 크게 차이

Library	면적(54)	slack(−46)
except EON1, A04	44	−15
All gate types in LSI Logic	48	10
including AND, OR except EON1, A04	44	9
including AND, OR with all gate types	48	16

그림17. 라이브러리의 변화에 따른 '대상 회로 I'의 최적화 결과
Fig. 17. Optimization result of 'Test circuit I.'

Library	테스트 회로II		테스트 회로III	
	면적 (98)	slack (−62)	면적 (70)	slack (−86)
except EON1, A04	72	−40	64	−75
All gate types in LSI Logic	88	−16	58	−60
including AND, OR except EON1, A04	76	−9	62	−73
including AND, OR with all gate types	82	−4	56	−58

그림18. 테스트 회로에 대한 최적화 결과
Fig. 18. Opimization results for the test circuits.

가 나는 입력단이 많은 게이트가 존재할 경우, 또는 게이트의 병합에 의해 그러한 게이트가 생성되었을 경우에 효과적인 동작 속도의 향상을 얻을 수 있다.

테스트 회로 I, II, III에 대한 실험 결과에서 라이브러리가 제공하는 게이트 타입의 종류가 다양할 경우, 특히 적은 지연 시간을 가지며 복잡한 기능을 수행하는 여러 종류의 compound gate가 제공될 경우 동작 속도의 향상이 효과적으로 이루어지며, 면적의 감소도 어느 정도 원활하게 이루어짐을 알 수 있다.

Ⅵ. 결 론

본 논문에서는 주어진 제약 조건을 만족하는 논리 회로를 자동적으로 생성해 주는 SiLOS의 기술 의존 최적화 방법에 대해 기술하였다. SiLOS는 SUN 3/60에서 C언어로 구현되었으며 VHDL로 기술된 회로 정보로부터 시간 제약 조건을 만족하지 못하는 임계 경로를 추출하고 이에 대한 정보를 제공하는 consulting mode와 재설계를 통하여 임계 경로가 제거된 논리회로를 생성하는 design mode를 가지며 효율적인 user interface를 위하여 SunView를 이용한 graphic 출력 기능을 갖도록 구현하였다.

최적화된 회로의 설계는 기술 독립적인 최소화 과정과 기술 의존적인 최적화 과정을 거친다. 기술 독립적인 최소화 과정[11]은 rule-based 방식으로 구현되었으며, 기술 의존적인 최적화 과정은 gate compounding, gate decomposition, gate inversion, gate conversion을 통한 변환 과정을 이용하여 구현하였다.

자동화된 회로의 최적화는 실험 결과에서 살펴 본 바와 같이 논리 회로를 작은 면적상에서 구현이 가능하게 하며, 이로 인한 회로의 성능 향상은 응용분야에서 설계의 질을 높이고 개발 시간의 감소를 가져오므로 이 분야에 대한 많은 연구가 필요하다. 본 논문에서 구현한 최적화 방법은 기본적으로 적용 방법 선정의 복잡성과 계산 시간의 지나친 증가를 방

지하기 위하여, 국소 범위에 국한된 적용 방식을 이루었으므로 항상 최적화된 결과를 얻을 수는 없다. 따라서 효과적인 최적화 시스템을 구현하기 위한 방법의 계속적인 연구가 필요하며, 향후 본 논문에서 제안한 기술 의존적인 최적화 과정의 보다 효율적인 적용을 위해 기술 의존적인 최적화 과정에서 변환 방법의 선정과 관계되는 가치값의 계산 방법에 관한 연구를 더욱 진행함으로써 회로 최적화의 정도를 높이기 위한 노력이 계속될 것이다.

參 考 文 獻

[1] J. Allen, "Performance-Directed Synthesis of VLSI Systems," *Proc. IEEE*, vol. 78, no. 2, pp. 336-355, Feb. 1990.

[2] K. Bartlertt, W. Cohen, A. De Geus, and G. Hachtel, "Synthesis and Optimization of Multilevel Logic under Timing Constraints," *Proc. IEEE Trans. Computer Aided-Design,* vol. CAD-5, no. 4, pp. 582-596, Oct. 1986.

[3] J.A. Darringer, D. Brand et al., "LSS: A System for Production Logic Synthesis," *IBM Journal of Research and Development,* vol. 28, no. 5, pp. 537-544, Sep. 1984.

[4] J.A. Darringer, W.H. Joyner, C.L. Berman, and L. Trevillian, "Logic Synthesis through Local Transformations," *IBM Journal of Research and Development,* vol. 25, no. 4, pp. 272-280, July 1981.

[5] G. De Micheli, "Performance-Oriented Synthesis of Large Scale Domino CMOS Circuits," *IEEE Trans. Computer-Aided Design,* vol. CAD-6, no. 5, pp. 751-765, Sept. 1987.

[6] K. Keutzer and M. Vancura, "Timing Optimization in a Logic Synthesis System," *in Proc. Int. Workshop on Logic and Architectural Synthesis,* May 1987.

[7] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science,* vol. 220, no. 4598, pp. 671-680, 1983.

[8] "Databook and Design Manual: HCMOS Macrocells, Macrofunctions," *LSI Logic,* Oct. 1986.

[9] "Synopsys Library Compiler Reference Manual," *Synopsys Inc.*, Oct. 1989.

[10] 이재형, 김태선, 황선영, "성능 구동 자동 논리설계 시스템," 한국정보 과학회 학술발표 논문집 제17권, pp. 423-426, 1990년 4월.

[11] 황선영, "성능 구동 논리회로 자동설계 방식 개발," 문교부 학술 진흥 연구 결과 보고서, ISRC 90-E-DE-D003, 서울대학교 반도체 공동연구소, 1990년 7월.

著 者 紹 介

黃 善 泳(正會員)

1954年 5月 20日生. 1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국과학원 전기및 전자공학과 공학석사 취득. 1986年 10月 미국 Stanford 대학 공학박사 학위 취득. 삼성 반도체 주식회사 연구원, Stanford 대학 CIS 연구소 연구원, Fairchild Semiconductor 기술 자문. 1989年 3月～현재 서강대학교 전자공학과 교수. 주관심분야는 CAD 시스템, Computer Architecture 및 System Design, VLSI설계 등임.

李 在 亨(準會員)

1968年 4月 6日生. 1990年 2月 서강대학교 전자공학과 졸업. 1990年 2月～현재 서강대학교 전자공학과 대학원 석사과정. 주관심분야는 CAD 시스템, Computer Architecture 등임.