特輯

실리콘 컴파일러에서의 논리 합성

黃善泳, 李在亨, 金泰善
西江大學校 電子工學科

I. 서 론

반도체 제조 공정의 발달은 VLSI의 회로에서 ULSI로의 전환을 이루고 있고 칩의 집적 능력은 점차 발전하고 있으나, 인간의 회로 설계 능력만으로 대규모 회로의 구성을 단기간내에 설계하여 동작 속도 또는 면적의 제한 조건을 만족하는 회로로 구성하기는 매우 어려운 일이다. 이러한 제약을 극복하기 위하여 자동화된 회로 설계 방식에 대한 연구가 집중적으로 이루어져 현재는 상업적으로 이용되기에 이르렀다[28],[31].

LSI 이상의 대규모 회로는 분할된 기능 모듈의 집적을 통하여 구현이 가능하며, 이러한 회로를 설계하는 경우 분할된 기능 모듈을 각기 설계하여 집적하는 방식과 전체 시스템에 대하여 필요한 기능의 회로를 정의하고 정의된 기능 모듈의 세분화된 회로 설계를 통하여 전체 회로를 구성하는 두가지의 방식이 사용된다. 전자를 bottom-up 방식, 후자를 top-down 방식이라 하며, VLSI의 자동화된 회로 설계 방식의 도입을 위해서는 회로의 개념 설계에서 차츰 세분화된 회로의 구성을 추출하는 top-down 방식이 적합하다. VLSI 설계시 top-down 방식의 적용을 위하여 동작 수준, 레지스터-트랜스퍼 수준, 논리 수준, 레이아웃 수준의 네가지 설계 계층을 정의할 수 있으며, 설계 계층에 따라 동일한 회로의 각기 다른 표현 기능을 가진다. 동작 수준, 레지스터 트랜스퍼 수준, 논리 수준의 상위 계층에서 요구되는 회로의 동작 기능의 정의에 대하여 레이아웃 수준의 하위 계층에서는 구체적인 내부 구조의 정의를 필요로 한다.

상위 단계의 설계 계층에서는 설계에 사용되는 기본 모듈을 하위 수준에 비하여 상대적으로 복잡한 기능과 다양한 표현 방식으로 정의하고 포괄적인 기능만을 제시하므로, 설계 계층을 상위 단계로 확장하여 설계 시간 및 설계 비용을 상대적으로 감소시켜 대규모 회로의 설계를 용이하게 할 수 있다. 그림 1은 설계 계층에 따른 회로 소자의 적정 사용 한계와 사용되는 기본 소자의 단위를 보인다. 설계 계층이 높아질 수록 설계에 사용되는 기본 모듈 기능의 복잡도는 높아지며 사용 가능한 게이트의 수는 많아진다[26]. 레이아웃 수준에서의 설계는 사용되는 기본 모듈을 트랜지스터 단위로 하여 게이트 100개 정도의 적정 설계 한계를 가지는 반면, RTL(register transfer level)에서는 flip-flop, MUX 등을 기본 소자로 사용하여 회로를 설계하게 되어 약 100K의 게이트를 포함하는 설계가 가능하다. 일반적으로 회로의 설계에서 개개의 게이트를 직접 기술하지 않고 데이타의 흐름을 기술하여 이로부터 RT 수준(혹은 논리 회로 수준)의 기술을 생성해 내는 단계를 상위 수준의 합성이라 한다[24].

상위 수준 설계에서는 회로 기능의 정의를 통하여, 소자간의 연결 구조(structure)나 소자의 내부 구조 및 구현 기술에 무관한 설계가 이루어지므로 제조 기술의 변화에 따른 적절한 설계의 변화가 가능하다. 그러나 상위수준에서의 동작 기술(behavioral description)로부터 자동화된 회로 설계를 얻기 위한 노력은 회로 기능의 표현이 설계자에 따라 다양한 패턴을 가지게 되어 이를 지원하기 위한 software의 개발에 대한 많은 연구가 진행되고 있다[39].

회로 설계의 효율화, 최적화를 위한 CAD 방식은 초기에 레이아웃 디자인을 위한 하위 수준의 설계 자동화에서 도입되어, 점차 논리 설계 및 상위 수준의 설계를 위한 연구로 발전하였으며, 최근에는 이들 각 분야를 통합하여 상위 수준의 설계로부터 레이아웃의 회로 구현이 가능한 silicon compiler가 개발되고 있다. Silicon

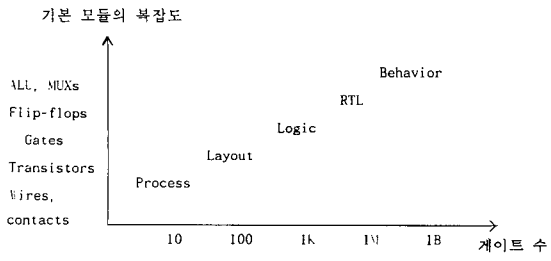


그림 1. 설계 계층에 따른 적정 설계 규모

compiler는 종래의 software 개발의 module화 및 계층화를 위한 compiler 개념을 hardware 설계에 적용하여 HDL(hardware description language)을 사용한 top-down 회로 설계 방식으로 효율적인 자동 설계 및 simulation이 가능하다[3],[15],[23],[24],[34],[38]. HDL로 설계된 회로는 기술된 HDL 자체에 내포된 documentation 기능을 가지므로, 회로에 대한 명확한 이해가 가능하여 회로의 재설계에 유리하다. Silicon compiler는 설계 계층의 각 단계를 하위 단계로 전환하여 주는 시스템과 simulator의 통합으로 구성되며 그 예를 그림 2에 보였다.

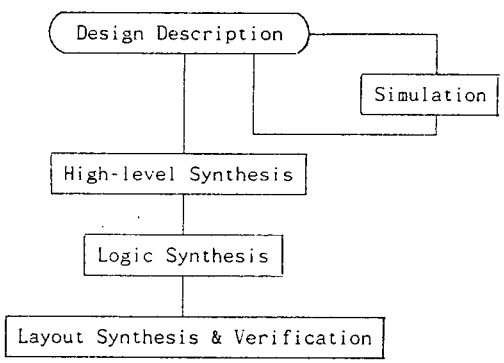


그림 2. Silicon compiler의 흐름도

상위 수준의 HDL을 이용하여 설계된 회로는 simulation을 통하여 설계자가 구상한 회로 동작과 실제 회로의 동작간의 차이를 확인함으로써, 설계 초기의 회로 오류를 제거 또는 수정할 수 있다. 상위 수준의 합성과 simulation을 위한 컴파일러는 입력 HDL의 각 구문에 대한 정보를 포함하는 자료 구조인 AST(abstract syntax tree)를 구성하고, AST로부터 synthesis와 simulation을 위한 자료 구조를 가지는 중간 형태를 생성한다. 대표적으로 사용되는 CDFG(control/data flow graph)의 control 및 data flow를 따라 event-driven simulation의 수행이 가능하며, 생성된 CDFG에 합성을 위한 정보를 포함시킴으로써 합성에 필요한 정보를 얻을 수 있다.

상위 수준 합성 단계에서는 CDFG의 각 operation을 실제의 하드웨어 operator로 할당하여, 할당된 하드웨어들의 연결 관계를 정의하여 논리 합성을 위한 출력을 제공한다. 이 단계에서는 CDFG에서 공유 가능한 operation을 찾고 각 operation이 수행되는 시간을 결정하는 scheduling및 각 operation에 대한 module 할당 및 MUX, register, BUS등의 할당을 결정하는 allocation으로 구성된다. 상위 수준 합성의 overhead를 줄이기 위한 중간 형태 언어의 사용으로 설계의 효율성을 높이기 위한 방안이 제안되어 이를 이용한 설계방식의 도입도 가능하다[3],[39].

Scheduling에 의하여 정의된 control 신호는 state의 최적화된 bit coding을 위한 state assignment를 통하여 2-level 논리식을 생성한다. 이 식은 논리 합성 과정을 거쳐 다단 논리로 구현이 가능하다. 다단 논리 합성은 decomposition, extraction, collapsing, phase assignment등의 algorithmic approach 방식과 사용자가 정의한 룰의 복합적 적용으로 최적화를 이루는 rule-base 방식이 사용된다. 다단 논리 합성의 결과는 technology-dependent logic optimization 과정을 통하여, 동작 속도 및 면적 제약 조건을 고려한 최적화와 각 논리 소자를 라이브러리에서 제공하는 게이트로 변환할 수 있다[13],[35].

Data-path의 module 중 라이브러리에서 제공되지 않는 module은 module generation 과정을 거쳐 회로의 layout을 생성한다. 생성된 각 layout의 수정 또는 기능 향상을 위한 layout editor 및 layout으로부터 동작기술의 추출을 위한 extraction을 수행하여 verification을 수행한다. Module의 layout이 생성된 후 module간의 연결 관계를 입력으로하여 placement 및 routing 과정을 수행하여 최종적인 layout을 완성하여 반도체 칩을 제작할 수 있다.

지금까지 상용화된 silicon compiler는 상위 수준 합성 시스템부터 배치 및 배선에 이르는 모든 시스템을 한 회사가 개발하는 형식이라기 보다는, 개발된 여러가지 tool들 중에서 성능이 뛰어난 tool을 각 level에서 하나씩 선택하여 각 시스템들의 각기 다른 입출력 형식을 사용자가 신경쓰지 않고 사용 가능하도록 맞추어 주고 통합된 그래픽 설계 환경 하에서 마치 하나의 시스템의 흐름으로 느껴지도록 framework을 구성해 주는 형태를 가진다. 그러나 상위수준 합성 시스템은 DSP와 같이

개발하려는 대상의 특성에 따라 그 성능이 좌우되므로 상용화에 어려움이 따르며 일반적으로 논리 합성 단계에서 최종 layout을 생성하는 시스템이 대상이 된다. 논리 합성 단계에서 상용화에 성공한 시스템으로는 Synopsys사에서 개발된 logic synthesis tool이 대표적이며[30],[31], 설계하려는 시스템의 구성을 하드웨어 기술 언어의 일종인 VHDL[33]을 이용하여 기술하여 주면 면적과 동작 시간 측면에서 최적화된 네트리스트를 생성한다.

Silicon compiler의 상위 수준 합성 단계는 참고문헌 [38]에서 중점적으로 논하였고, 본고에서는 상위 수준 합성 근간을 이루며 상업적으로 상용화 단계에 이른 논리 합성 분야에 대하여 중점적으로 논하기로 한다. "논리 합성"의 "합성"이라는 어휘는 생성(generation) 및 가공(manipulation)의 두가지 의미를 가지며, 이에 따라 RTL의 표면으로부터 논리 네트리스트의 생성과 생성된 네트리스트의 최적화 및 레이아웃의 생성 과정을 논리 합성이라 한다. 과거의 논리 합성은 주로 후자의 개념으로서 PLD 및 gate array의 레이아웃의 생성 또는 논리의 다단화를 통한 회로의 최소화를 지칭하였다. 최근의 설계 자동화 툴의 진보는 단순한 네트리스트의 최적화 또는 회로 레이아웃의 생성에 국한되지 않고, RTL의 회로 동작 기술로 부터 회로 구조를 추출하는 단계에 이르고 있으므로 논리 합성의 이러한 개념을 포함하여 광역의 의미로 지칭되고 있다[27].

논리 합성은 그림 3과 같이 회로의 기능이 표현된 입력으로부터 칩으로 구현 가능한 회로 패턴을 제한된 면적과 동작 속도 및 기타의 제약 조건이 만족되도록 생성하는 과정이며, 회로의 구현 과정에서 칩의 제조 비용을 결정하는 중요한 분야가 된다. RTL 표현에서 생성된 논리의 최적화는 많은 연구가 진행되었다[4],[5],[6],[10],[18]. 특히 논리 구현에 있어서 회로의 다단화는 여러 논리식에서 공통되는 논리식을 추출하여 대치함으로써 PLA 형태의 이단 논리에 비하여 면적을 크게 줄일 수 있으며, 동작속도의 향상이 가능한 여러 형태의 논리식으로 표현이 가능한 장점이 있다.

회로의 다단화는 논리 합성의 주된 연구 분야가 되고 있으며, 공통되는 논리식의 추출방법이 그 핵심이 된다. 기술 매핑은 논리 합성의 back-end 과정으로서 다단 논리 합성으로 생성된 논리식을 주어진 라이브러리의 게이트로 대치하여 레이아웃의 생성이 가능한 논리 회로를 얻는 과정이다. 사용되는 라이브러리는 논리 회로에서 자주 사용되는 AND, OR, NAND, NOR, 인버터, 복합 게이트 등의 논리 소자를 미리 레이아웃으로 디자인하여 이들에 대한 면적, 동작속도 및 구동능력등의 정보를 기술하여 구성한다. 다단 논리 합성으로 생성된 논리는 구성된 라이브러리에서 제공하는 게이트 형태로 변환하며, 이때 회로 제약 조건을 고려하여 회로 동작이 동일한 게이트 집합으로 대치함으로써 최적화된 레이아웃의 구현이 가능하도록 한다. 본고의 II장에서는 다단 논리 최적화를, III장에서는 기술 매핑의 각 과정에서 사용되는 방법과 그 알고리듬에 대하여 기술하고, IV장에서는 결론을 맺는다.

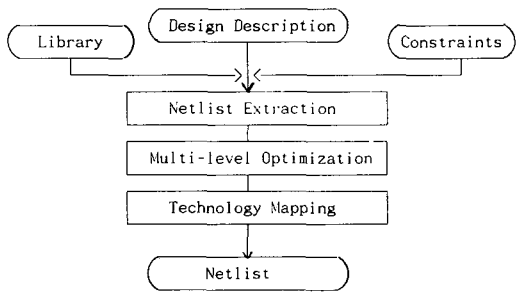


그림 3. 논리 합성의 흐름도

II. 다단 논리 합성

논리 회로는 이단 및 다단 논리로 표현이 가능하다. PLA를 이용한 이단 논리 회로의 면적은 PLA의 행과 열의 수에 의하여, random logic을 이용한 다단 논리 회로의 면적은 트랜지스터의 수에 해당하는 변수의 수로 평가될 수 있다[4],[7]. 논리의 최적화는 함수를 보다 간단하며 동일한 기능을 수행하는 형태로 변환하는 과정으로 칩의 구현시 실리콘 면적을 최소화한다. 이단 논리의 최소화는 PLA에서 행에 해당하는 product 항의 수를 최소화하는 과정이며, 이 논리 최적화 문제는 NP-complete 문제로 중간 규모 이상의 논리 회로 최적화를 위하여 MINI[20], ESSPRESO[4] 등의 휴리스틱 알고리듬이 개발되었다. 그러나 요구되는 시스템의 규모가 커짐에 따라 면적과 지연 시간이 커지게 되어 PLA의 회로 구현 방식은 고속 동작 회로 설계의 제약 조건을 만족시키기 어려운 단점을 가지므로, 이를 해결하기 위하여 다단 논리의 회로 구현 방식을 채택하게 되었다. 다단 논리 최적화를 위한 연구로 BOLD[19], MIS[7] 등의 시스템이 발표되었으며, 논리 최적화를 위한 효과적인 휴리스틱 알고리듬 개발이 계속 연구되고 있다[40].

다단 논리의 최적화는 RTL 기술로부터 추출된 초기 형태 네트리스트를 재구성하여 회로의 최적화를 이룬

다. RTL 기술에서 추출된 논리식은 일반적으로 다단 논리 형태를 이루고 있으나 부분적으로 불필요한 항을 가질 수 있어 collapsing 또는 flattening 과정을 통하여 이단 논리로 구성한 후, 논리 최소화를 수행한다. 이단 논리 최소화는 ESSPRESO에서 효과적인 최소화 알고리듬이 구현되었으며, 논리식에서 회로의 기능에 변화를 주지 않는 부분을 찾아 제거하며 최소한의 변수로서 논리식을 구성한다. 논리 최소화의 결과 이단 논리로 구성된 네트리스트를 다단 논리로 구성할 때 최적화 과정을 거친다. 다단 논리의 최적화 방식은 크게 Boolean 방식과 algebraic 방식의 두가지로 분류된다. 이들 방식은 공히 decomposition, extraction, factoring, substitution의 기본적인 방법으로 다단 논리를 합성하며 그 예를 그림 4에 보였다.

```
F = abc + abd + a'c'd'          F = XY + X'Y'
    + b'c'd'          =>        X = ab
                                Y = c + d
```

(a) Decomposition

```
F = (a + b)cd + e               F = XY + e      X = a + b
G = (a + b)e'          =>       G = Xe'         Y = cd
H = cde                         H = Ye
```

(b) Extraction

```
F = ac + ad + bc + bd      =>      F = (a + b)(c + d)
```

(c) Factoring

```
F = a + bc                      F = G(a + c)
  = (a + b)(a + c)      =>
G = a + b                       G = a + b
```

(d) Substitution

그림 4. 다단 논리 합성의 방법

논리의 다단화를 위한 decomposition은 하나의 식으로 표현된 함수식을 새로이 정의되는 중간 변수(intermediate variable)을 유도하여 함수식을 재구성하는 것이며, extraction은 decomposition과 유사한 개념을 갖지만 2개 이상의 함수식에 대하여 중간 변수를 유도하는 점에서 다르다. Factoring은 factored form으로 함수식을 변환하여 변수의 수를 최적화함으로써 논리 구현시 면적이 최소화되도록 한다[8]. Substitution은 특정한 함수식이 다른 함수식의 일부를 이루는 경우 그 식을 대표하는 변수로서 다른 식을 재구성하는 것을 말하며 이러한 이유로 resubstitution이라고도 불린다. 이상의 방법들은 나눗셈과 비슷한 과정이며 사실상 나눗셈은 다단 논리 합성의 기본을 이룬다. 나눗셈의 연산자 선택을 위하여 각 함수식에서 겹쳐지는 부분을 찾아야 하며, 선택된 연산자로 함수식을 나누어 나머지가 생기는 연산자를 factor라 하고 그렇지 않은 경우 divisor라고 한다.

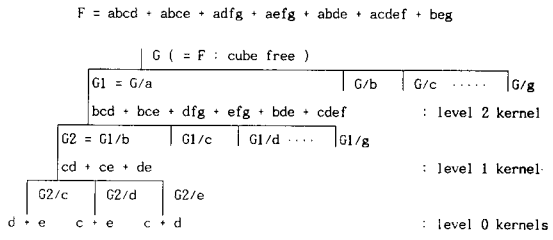
1. Divisor의 추출

Divisor의 추출을 위하여 커널을 사용하는 방법이 알려져 있다[9]. 커널은 함수식 F를 큐브 c로 나눈 몫 F/c가 cube free인 경우 F/c를 커널이라 한다. 큐브는 함수식을 구성하는 한 변수 또는 이 변수들의 곱의 표현을 말하며, cube free는 어느 큐브로 나누어도 "나누어도 떨어지지 않는" 함수식을 말한다. 이때 커널을 추출하기 위하여 사용된 큐브 c를 co-kernel이라 한다. 함수식은 추출된 커널과 co-kernel의 곱, 그리고 나머지로 표현될 수 있으므로, 함수식에서 추출된 커널 또는 커널의 일부로 함수식에 대하여 나눗셈의 수행이 가능하다.

```
Procedure Kernel_extract( F ) {
    c = largest cube factor of f ;
    K = Kernel( 0, F/c ) ;
    if( F is cube free )
        return( F ∪ K ) ;
    return K ;
    }

Procedure Kernel(j, G) {
    R = G ;
    n = number of literals in G ;
    for( i = j ; i <= n ; i ++ ) {
        c = largest cube dividing G/li evenly ;
        if( lk ∉ c for all k < i )
            R = R ∪ Kernel( i + 1, G/(li ∩ c) );
        }
    return R ;
    }
```

(a) 커널 추출 알고리듬



(b) 커널 추출 알고리듬 적용의 예

그림 5. 커널의 추출

커널의 레벨에 대한 정의는 반복적인 적용으로 이루어지며, 먼저 자신이외의 다른 어떠한 커널도 포함하지 않는 커널을 레벨 0 커널이라 하고 레벨 N-1 커널을 포함하는 커널은 레벨 N 커널이라 한다[7]. 커널의 레벨은 높을 수록 복잡한 식으로 표현되며, 높은 레벨의 커널을 이용할 수록 더 많은 공통적인 부분을 가지는 divisor의 선택이 가능하다. 그러나 높은 레벨의 커널을 사용하는 경우 time complexity 때문에 대부분의 경우 레벨 0 커널을 이용하여 연산자를 선택한다. 그림 5에 커널 추출 알고리듬과 이 알고리듬을 이용한 커널 추출의 예를 보였다.

2. Algebraic Division

추출된 커널에서 선택된 divisor로 함수식을 나누어 논리의 다단화가 가능하다. 주어진 함수식을 divisor로 나누었을 경우 몫을 이루는 각 변수는 divisor의 각 변수와 서로 중첩되어 사용되지 않는 분리성(disjoint)의 연산을 algebraic 연산이라 하며, 그 특성상 나눗셈의 연산이 단순하여 알고리듬의 구성이 용이하다. 나눗셈을 수행하는 algebraic division 알고리듬을 그림 6에 보였다. 함수식 F=abcd+abce+adfg+aefg+abde+acdef+beg의 다단화를 위하여 이 함수식에서 추출된 커널 중 d+e를 divisor로 선택하여 algebraic division을 수행하는 경우, 먼저 U와 V를 정의하면

$$U=d+e+d+e+de+de+e$$
$$V=abc+abc+afg+afg+ab+acf+bg$$

로 된다. 몫 H는

$$h1=\{v_i : u_i=d\}=abc+afg$$
$$h2=\{v_i : u_i=e\}=abc+afg+bg$$
$$H=h_1 \cap h_2=abc+afg$$

가 되며, divisor를 함수식으로 하는 새로운 변수를 K라 하여 decomposition하면 다음과 같은 다단화가 이루어진다.

$$F=K(abc+afg)+abde+acdef+beg$$
$$K=d+e$$

Divisor로써 커널을 사용하지 않고, 기존의 함수식을 이용하여 algebraic division을 수행하는 경우 resubstitution이 가능하다. 한편 factoring을 수행하는 그림 7의 알고리듬은 함수식의 커널을 divisor로 하여 algebraic division을 수행하며, 이때 몫과 나머지에 대하여 반복적인 적용을 통하여 주어진 함수식을 factored form으로 구성한다.

```
Procedure Algebraic_division( F, K ) {
    L = restriction of f to the literal in K :
    V = restriction of f to the literal not in K :
    hi = { vj ∈ V | uj == ki } :
    H = ∩ hi :
    return( H, F - KH ) :
    }
```

그림 6. Algebraic division 알고리듬

```
Procedure Algebraic_factor( F ) {
    if( | F | == 1 ) return F :
    K = kernel_extract( F ) :
    (H, R) = Algebraic_division( F, K ) :
    return( Algebraic_factor(K) * Algebraic_factor(H),
            Algebraic_factor(R) ) :
```

그림 7. Factoring 알고리듬

Decomposition 또는 factoring은 한 함수식을 분해하는 과정이므로 커널의 직접적인 사용이 가능하다. Extraction은 여러 함수식에서 공통되는 연산자를 선택하여 사용하여야 하므로 커널의 공통되는 부분(kernel intersection)을 divisor로 선택한다. 커널의 intersection은 co-kernel을 사용하여 얻는다. Co-kernel은 커널을 유도할 때 사용한 큐브를 말하며, 그림 5(b)의 예제에서 커널 (d+e)의 co-kernel은 abc이고 레벨 1 커널인 (cd+ce+de)의 co-kernel은 ab이다. Co-kernel을 이용한 kernel intersection의 추출은 다음의 단계를 따라 이루어진다.

단계 1) 주어진 함수식들의 커널을 추출하여 각 커널에 존재하는 개개의 큐브를 새로운 변수로 선언한다. 선언된 변수들의 곱의 형태로 각 커널을 구성하고 이들의 합의 표현으로 새로운 함수식을 정의한다. 예를 들어 추출된 커널의 집합 K={(abc+de+fg), (abc+de+fh), (abc+fh+gh)}가 주어진 경우 각 튜브를 $t_1=abc$, $t_2=de$, $t_3=fg$, $t_4=fh$, $t_5=gh$로 선언하면, 새로운 함수식은 $K'=t_1t_2t_3+t_1t_2t_4+t_1t_4t_5$가 된다.

단계 2) 새로이 정의된 함수식으로부터 커널을 추출

한다. 이때의 co-kernel이 커널의 intersection이다. 즉 그림 6의 알고리듬을 K'의 식에 적용하는 경우 커널 t_3+t_4, t_2+t_5, $t_2t_3+t_2t_4+t_4t_5$에 대하여 각각 t_1t_2, t_1t_4, t_1의 co-kernel을 얻을 수 있다. 구해진 co-kernel을 다시 정리하면 abc+de, abc+fg, abc가 되며 이들이 커널 intersection이다.

단계 3) 추출된 커널 intersection 중에서 하나를 divisor로 선택하여 extraction을 수행한다.

3. Boolean Division

Algebraic division 알고리듬을 이용한 다단 논리의 합성 방법외에 Boolean division 알고리듬을 사용하여 다단 논리를 이루는 방식이 있다. Algebraic division 알고리듬은 그 몫을 이루는 각 변수가 divisor의 각 변수와 서로 중첩되어 사용되지 않으나 Boolean 연산은 사용한 나눗셈에서는 몫과 divisor간의 중첩되는 변수가 존재할 수 있으므로, 이러한 연산을 위하여 이단 회로 최소화에서 사용된 알고리듬을 이용한다. 이단 회로 최소화에서 사용하는 알고리듬은 함수식에서 함수식의 선언상에 함유된 DC(don't care)집합을 이용함으로써 식의 최소화가 가능하다[1,2,14].

함수식의 선언상에 함유된 DC 집합은 SDC(satisfiability don't care)와 ODC(observability don't care)로 이루어진다. SDC는 함수식 G를 대표하는 새로운 변수 g를 정의함으로써 생기는 DC를 말하며 SDC= g'G+gG'로 정의된다. 두개의 함수식 F=abX+a'cX, X=a'+b에서 중간 변수 X와 함수 (a'+b)는 함수식 F에 대하여 다음과 같은 SDC를 제공한다.

$$SDC_F = X'(a'+b)+X(a'+b)' = a'X'+bX'+ab'X$$

SDC_F는 함수식 F에 대한 DC가 되므로, 이들을 더하여 F=F+SDC_F로 다시 정의할 수 있다. 즉 F는

$$F = abX+a'cX+a'X'+bX'+ab'X = aX+a'c$$

로 단순화가 가능하다.

한편 ODC는 선언된 중간 변수가 어떻게 사용되는가에 따라서 발생하는 OCD이며,중간 변수의 값이 최종 출력에 영향을 미치지 않는 경우를 말한다. 함수식 G를 대표하는 중간변수 g가 n개의 함수식 Z={ Z_1,Z_2, ...Zn }에서 사용되는 경우, G에 대한 ODC는

$$ODC_G = Z_{i(g=0)}Z_{i(g=1)}+Z_{i(g=0)}Z_{i(g=1)}$$

로 정의된다.

Boolean division에서는 DC 집합을 사용하여 algebraic division에서 불가능한 연산의 수행이 가능하다.

그림 8의 Boolean division 알고리듬은 연산자 G를 새로운 변수 g로 할당하여 생기는 SDC를 DC 집합에 포함하여 최소의 off-set R과 함수식 F를 구성한다. 이후에 R에 포함되지 않는 부분에 대하여 F를 확장하고 불필요한 부분을 제거하여 최소화된 몫과 나머지를 추출한다.

```
Boolean_division( F, G, DC ) {
 /* R : OFF-set
  * F : ON-set
  */
   let g is a new variable defining function G :
   DC = DC + (g'G + gG') :
   R = (F + DC)' :
   F = (R + DC)' :
   F = Remove( F, g' ) :
   F = Min_literal( F, R ) :
   F = Expand( F, R ) :
   F = Irredundant( F, DC ) :
   H = Cofactor( F, g ) :
   E = F - H * G :      /* remainder */
   return( H, E ) :
   }
```

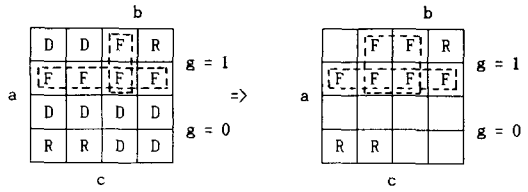
그림 8. Boolean division 알고리듬

예를 들어 함수식 F=a+bc를 연산자 a+b로 Boolan division을 수행하면, DC와 R 및 F는 다음과 같이 계산된다.

$$DC = g'(a+b)+a'b'g$$
$$R = (F+DC)' = a'b'g'+a'bc'g$$
$$F = (R+DC)' = ag+ bcg$$

이후 F를 그림 9와 같이 expend하여 F=ag+cg로 할 수 있다. Expand는 off-set R이 제외된 영역에서 함수내의 각 큐브가 확장 가능한 최대의 큐브로 변환하는 기능을 가진다[4].

Expand된 결과의 함수식 F에서 g가 포함된 큐브를 취하여 몫으로 하고, 나머지를 결정하면 다음과 같다.



(a)Expand 대상 (b) Expand 후

그림 9. Karnaugh map에서의 expansion

H=a+c
E=0

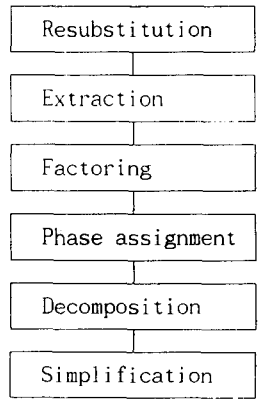


그림 10. 다단 논리 합성 과정

4. 다단 논리 최적화

다단 논리 합성 방법에는 나눗셈의 연산이 수반되는 decomposition, factoring extraction, substitution 외에, 나눗셈의 연산을 사용하지 않는 phase assignment, simplification의 방법이 포함된다. Phase assignment는 함수식의 최종 출력의 phase를 변화함으로써 회로의 최소화를 이루는 방법을 말한다. Simplification은 이단 논리 최적화를 통하여 한 함수식에서 사용되는 변수의 수를 최소화 하는 방법이며, SDC와 ODC 및 외부에서 주어지는 DC 집합을 모두 사용할 수 있다. 이상의 방식은 회로를 집적 다단 논리화하지는 않으나, 논리 다단화에 따르는 부수적인 회로 면적의 증가등에 대한 trade-off를 취할 수 있도록 한다.

다단 논리 최적화의 각 방법이 적용되는 순서 및 조합에 따라 합성된 논리 회로의 결과는 각기 달라질 수 있다. 그림 10의 다단 논리 합성 과정의 흐름도에서는 다단 논리 합성의 각 방법이 적용되는 순서의 예를 보인다. 실제의 다단 논리 합성에서는 decomposition에 의하여 논리식이 유도되어 새로운 함수식을 구성하는 경우, 그 함수식을 이용한 resubstitution의 반복적 적용으로 회로의 최적화를 이룰 수 있는 가능성이 있으므로 그림 10의 흐름도의 전체를 반복하거나 일부의 과정을 반복하여 적용함으로써 회로 결과를 얻기 위한 방식을 선택한다.

Ⅲ. 기술 매핑

다단 논리 합성의 결과는 실제 사용되는 구현 기술과는 무관하게 설계된 회로일 수 있으므로, 라이브러리에서 제공되는 게이트의 집합으로 회로를 재구성함으로써 회로의 레이아웃 생성이 가능한 기술 매핑 과정을 거치도록 한다[11],[16],[35],[36]. 이때 주어진 시간 제약 조건을 만족시키며 회로의 면적을 줄이는 논리 설계가 필요하다[25]. 기술 매핑 시스템을 이용하여 기술의 발달과 함께 새롭게 부각되고 있는 기술 전환(technology translation)을 이룰 수도 있다. 기술 전환은 기존의 구현 기술에서 새로운 구현 기술로 회로를 재구성하는 방식을 말하며, 새로운 구현 기술로 구성된 라이브러리를 사용한 기술 매핑을 통하여 가능하다[30].

기술 매핑을 위하여 미리 설계된 각 논리 소자의 레이아웃에 대한 동작 속도 및 면적등의 정보를 가지는 라이브러리의 구성이 필요하다[32]. 라이브러리에 포함되는 정보의 예를 그림 11에 보였다. 지연 시간 및 fanout 제약 조건의 기술등의 정보가 상세하게 작성될 수 있도록 기술 매핑시 제약 조건에 대한 회로의 상태를 정확하게 판정하여 구현된 회로의 오동작을 방지할 수 있다.

기술 매핑은 초기의 local transformation에서 rule-based에 의한 방식으로 연구가 이루어졌고 최근에는 graph covering에 의한 방법이 사용되고 있다[12],[17],[21],[29]. Local transformation은 rule-based의 초기 형태의 방정식이며 국소 범위에서 회로 변환을 통하여 기술 매핑을 수행하는 방식이다[12]. Rule-based 방식은 SOCRATES[17]에서 사용되었고 국소 범위 최적화의 한계를 제거하기 위한 meta-rule의 설정등으로 많은 관심을 끌

```
Library( example ) {
  default_max_fanout : 3 ;
  default_fanout_load : 1 ;
      cell( A\D4 ) {
          area : 2 ;
          pin( A, B, C, D ) {
              direction : input ;
              capacitance : 10 pF ;
              fanout_load : 1 ;
              }
          pin( out ) {
              direction : output ;
              time() {
                  intrinsic_rise_time : 2560 ps ;
                  intrinsic_fall_time : 2240 ps ;
                  rise_resistance : 12 ;
                  fall_resistance : 13 ;
                  related_pin : "A B" ;
                  }
              function : "A∩B∩C∩D" ;
              max_fanout : 4 ;
              }
          }
  }
```

그림 11. 라이브러리의 예

었으나, 방만한 rule의 설정, 기술 변화에 따른 룰 재설정의 overhead 등의 문제점을 가지고 있다. 그래프 커버링을 이용한 기술 매핑은 초기에 DAGON[21], MIS-II technology mapper[13] 등에서 제안되어 많은 연구를 거쳐 최근에는 fanout을 고려한 기술 매핑에 이르렀다[29].

1. 그래프 커버링

그래프 커버링은 주어진 회로를 대상 그래프(subject graph)로 구성하여 시스템 라이브러리에서 제공되는 게이트의 패턴 그래프중 대상 그래프의 일부와 매칭되는 게이트 집합을 구성하고, 이 게이트 집합중에서 대상 그래프 전체를 포함하며 최소의 비용을 가지는 게이트 집합을 선택함으로써 기술 매핑을 이루는 방식이다. 회로의 대상 그래프는 DAG(directed acyclic graph)를 이용할 수 있으나 패턴 매칭 과정은 NP problem이므로 트리 형태의 분할된 그래프가 사용된다. 트리 커버링은 DAG 커버링의 유사 모델링이며 분할된 각 트리내에서 최적화된 기술 매핑을 이룸으로써 전체 회로의 기술 매핑을 최적화 한다. 트리 커버링은 국소 범위의 최적화와 트리간의 경계면에서 발생하는 비효율적인 기술 매핑의 문제점을 가지며, 또한 트리의 구성시 구성되는 트리의 형태에 따라 각기 다른 기술 매핑의 결과를 보일 수도 있다[22].

DAGON 시스템에서 처음 제안된 트리 커버링 방식은 fanout 노드를 boundary로 설정하여 DAG의 형태를 가지는 초기 회로를 입력 회로의 primary output 노드들과 각 fanout 노드들을 root 노드로 하는 트리들로 재구성하여 각 트리들을 최적화시켰다. 이 boundary 설정에서 발생되는 면적 손실의 문제점을 해결하기 위하여 MIS-II technology mapper는 트리의 boundary를 fanout 노드에 국한시키지 않고 한 primary output을 root 노드로 하는 transitive fanin을 하나의 cone로 하여 각각의 cone을 최적화시키는 방법을 이용하였다. 이 방법 역시 cone의 매핑 순서에 따라 기술 매핑 결과의 차이를 가져오기 때문에 linear ordering에 의한 cone 순서 설정 과정이 필요하다.

패턴 그래프와 대상 그래프의 각 버텍스는 기본 함수(base function)로 정의된 타입을 사용하여 구성하며, 초기에는 가능한 여러가지 타입의 기본 함수를 사용하여 구성하였으나 2-input NAND 게이트와 인버터만으로 버텍스를 구성하는 방식이 효율적임이 알려졌다[6]. 그림 12는 4-input AND 게이트의 그래프들을 나타낸다. 라이브러이에서 제공하는 각 게이트들은 그림 12와 같은 패턴 그래프를 구성하여 패턴 라이브러리를 형성한다.

회로를 구성하는 대상 그래프내의 4-input AND 게이트도 그림 12와 동일한 그래프 구성이 가능하다. 그러나 대상 그래프에서는 구성된 그래프의 각 버텍스가 패턴 라이브러리의 패턴과 매칭됨에 따라 각기 다른 게이트의 일부로 구성될 수 있으므로, 어느 형태로 결정되는 가에 따라 회로의 동작 시간이 다른 기술 매핑의 결과를 보일 수 있다. 따라서 대상 그래프는 지연 시간 제약 조건을 만족할 수 있는 형태로 구조를 변환하도록 한다. Restructuring은 그림 12의 (a)와 같은 불균형 트리의 기술 매핑시 회로의 지연 시간이 시간 제약 조건을 만족하지 못하는 경우를 제거하기 위해 균형 트리로의 변환 또는 다른 형태의 불균형 트리로 변환하여 시간 제약을 만족시키는 기능을 가진다.

결정된 구조의 대상 그래프의 각 버텍스를 루트로 하는 트리에 대하여 패턴 라이브러리와의 패턴 매칭을 수행하여 각 버텍스에 매칭된 모든 패턴 그래프 집합을 구성하며, 패턴 매칭은 최종 출력단에서 입력단의 방향으로 수행한다. 매칭된 패턴 집합에서 그림 13의 알고리듬을 적용하여 최적의 커버를 형성하는 게이트를 선택하여 제약조건을 만족시키는 기술 매핑을 이룬다. 이때 비용 함수의 결정에 따라, 지연 시간 및 면적 제약 조건을 고려한 기술 매핑이 가능하다. 비용함수에서 loading effect에 의한 지연 시간을 고려하기 위해서는 매칭된 패턴 그래프의 종단에 해당하는 버텍스에 해당 게이트의 입력 용량에 대한 정보를 기록해야 한다. 그림 14는 그래프의 예를 보인다.

2. Fanout 제약하의 논리 회로 합성

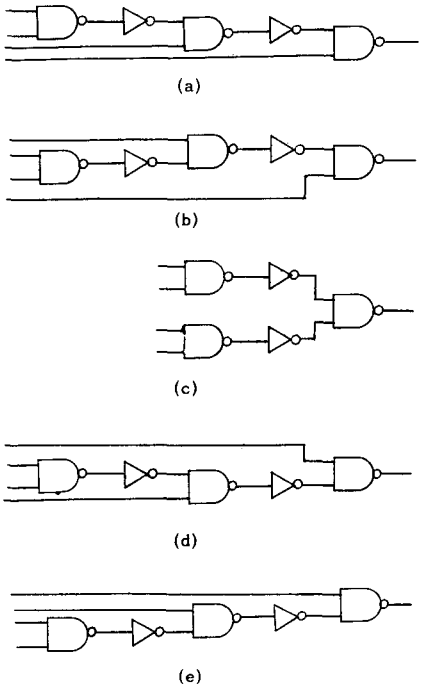


그림 12. 4-input AND 게이트의 패턴 그래프

```
Procedure Graph_cover( V, P ) {
  * V : vertex set of subject graph
  * P : set of pattern graphs constructed from library
  *
    for( each children vertex v of V )
        Graph_cover(v, P) ;
    cost(V) = - ∞ ;
    M = Matched (V, P) ;
    for( all m in M ) {
        cost(m) = Cost_function( V, m ) ;
        if( cost(V) > cost(m) ) {
            cost(V) = cost(m) ;
            cover(V) = m ;
            }
        }
  ;
```

그림 13. 그래프 커버링 알고리듬

기술 매핑된 논리 회로내의 게이트가 구동 능력을 초과하는 게이트를 구동하는 경우 회로의 오동작을 유발할 수 있으므로, 그림 13의 알고리듬의 비용 함수는 매칭된 게이트의 구동 능력과 해당 버텍스에서 구동되는 게이트의 수를 비교하여 구동 능력이 그보다 작은 게이트는 선택되지 않도록 정의한다. 충분한 구동 능력의

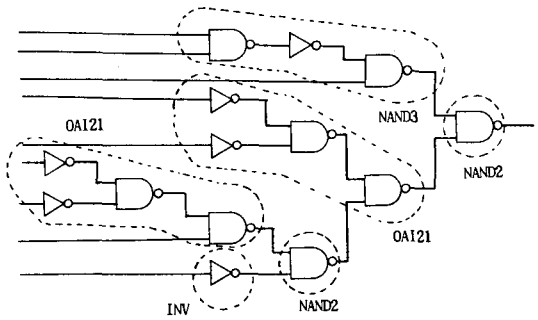


그림 14. 그래프 커버링의 예

게이트 매칭이 이루어지지 않은 버텍스는 fanout 제약 조건을 고려하기 위한 일반적인 방법으로 버퍼를 삽입한다. 버퍼의 삽입은 지연 시간을 고려하여 최적의 구조를 가지는 트리 형태로 삽입되는 것이 가장 효과적인 방법이다[37]. 그러나 트리 형태의 버퍼 구조 설정은 트리의 depth에 따라 NP problem이 되므로, 대개의 경우 depth를 2 이하로 제한하여 사용한다. 그림 15는 버퍼 트리의 구성 예를 보인다.

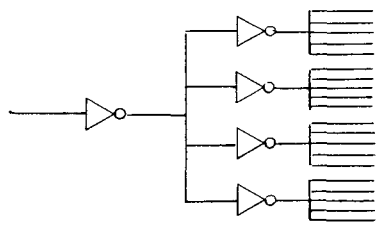


그림 15. 버퍼 트리의 구성

버퍼의 삽입은 지연 시간을 증가시키는 효과를 가져올 수 있으므로, 동작 시간 제약을 만족하지 못하는 임계 경로상에서는 비효율적이다. 이러한 버텍스에서는 해당 버텍스에 매칭된 게이트를 복사(replication)하여 사용하는 방법을 생각할 수 있다. 게이트의 복사는 입력단에서 선택된 게이트의 구동능력을 침해하는 경우가 있을 수 있으며, 간단한 해결 방식으로 입력단에서 선택된 게이트의 구동 능력을 침해하지 않는 정도의 게이트를 복사하여 복사된 게이트 중 비임계 경로의 게이트에 대하여 버퍼를 삽입하는 방법을 사용한다. 버퍼의 삽입 또는 게이트의 복사의 경우는 삽입되는 게이트에 해당하는 버텍스를 대상 그래프에 삽입하여 표현한다.

Ⅳ. 결 론

설계자의 최소의 노력으로 최적의 회로를 구현하기 위한 논리 합성은 설계 자동화의 한 분야로써 꾸준한 연구가 추진되고 있다. 본고에서는 설계 자동화를 위한 실리콘 컴파일러 시스템에 대하여 설명하였고, 특히 논리 합성을 위하여 다단 논리 합성과 기술 매핑의 각 알고리듬에 대하여 기술하였다. 기술된 알고리듬은 구체적인 구현 방식에 따라 프로그램의 수행 성능상의 차이를 보일 수 있으나, 논리 합성 시스템의 구현을 위하여 기본적으로 사용되는 알고리듬이다. 최근에는 논리 합성의 성능 향상을 위하여 레이아웃 과정에서 수반되는 문제점을 합성단계에서 고려하기 위한 연구가 진행되고 있으며, 상위 수준의 논리 합성 시스템의 완성을 위한 전 단계로써 시스템의 특성에 적합한 논리 합성 방식이 연구되고 있다. 국내에서도 여러 연구 기관에서 논리 합성을 위한 알고리듬의 개발 및 구현을 위한 노력이 진행 중이며, 조만간 실용화되어 회로 논리 설계의 자동화가 이루어 질 것으로 전망되고 있다.

參 考 文 獻

[1] K.A. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A.R. Wang, A. Sangiovanni-Vincentelli, "Multiple-level logic minimization using implicit don't cares", *IEEE Trans. Computer-Aided Design*, vol. 7, no. 6, pp.723-740, June 1988.

[2] D. Brand, "Redundancy and don't cares in logic synthesis", *IEEE Trans. Computers*,vol. C-32, no. 10, pp.947-952, Oct. 1983.

[3] R.K. Brayton, R. Camposano, G. De Micheli, R. Otten and J. von Eijndhoven, "The Yorktown Silicon Compiler", in *Silicon Compilation*, D. Gajski (ed.), Addison Wesley Pub. Reading, Mass., 1987.

[4] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, Mass.,1985.

[5] R.K. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, A. Sangiovanni-Vincentelli, "Multiple-Level Logic Optimization System", in Proc. I Int'l Conf. on CAD, pp.356-359, Nov. 1986.

[6] R.K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli,"Multilevel logic synthesis", *IEEE Proceedings*, vol. 78, no. 2, pp.264-300, Feb. 1990.

[7] R.K. Brayton and R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS : A multiple-level logic optimization system", *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 6, pp. 1062-1081, Nov. 1987.

[8] R.K. Brayton, "Factoring logic function", *IBM J. Res. Develop.*, vol. 31, no. 2, pp.187-198, Mar. 1987.

[9] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "Multi-Level Logic Optimization and the Rectangular Covering Problem", in Proc. IEEE Int. Conf. on CAD, pp.66-69, Nov. 1987.

[10] M.A.Breuer, *Design Automation of Digital System (Vol. 1) Theory and Techniques*, Prentice-Hall, Englewood Cliffs, NJ., 1972.

[11] J.A. Darringer, D. Brand, J. Gerbi, W.H. Joyner Jr., and Trevillyan, "Logic synthesis through local transformation", *IBM J. Res. Develop.*, vol. 25, no. 4, pp.272-280, July 1981.

[12] J.A. Darringer, D. Brand, J.V. Gerbi, W.H. Joyner Jr., and L. Trevillyan, "LSS : A system for production logic synthesis", *IBM J. Res. Develop.*, vol. 28, no. 5, pp.537-544, Sept. 1981.

[13] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology Mapping in MIS", in Proc. IEEE Int. Conf. on CAD, pp.116-119, Nov. 1987.

[14] S. Devadas, A. Wang, R.K. Brayton and A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multi-Level Optimization", in Proc. IEEE Int. Conf. on CAD, pp.290-293, Nov. 1988.

[15] D. Gajski, "Introduction to Silicon Compiler", in *Silicon Compilation*, Addison Wesley Pub., Reading, Mass., 1987.

[16] D. Gregory, K. Bartlett, A. de Geus, and G.

Hachtel, "SOCRATES : A System for Automatically Synthesizing and Optimizing Combinational Logic", in Proc. 23th ACM/IEEE Design Automation Conf., pp. 79-85, June 1986.

[17] A.J. De Geus and D.J. Gregory, "The SOCRATES Logic Synthesis and Optimization System", in *Design Systems for VLSI Circuits*, G. De Micheli(Ed.), Martinus Nijhoff Pub., Dordrecht, Netherlands, pp.473-498, 1987.

[18] G. Hachtel, R. Jacoby, K. Keutzer, and C. Morrison, "On Properties of Algebraic Transformations and the Multifault Testability of Multilevel Logic", in Proc. IEEE Int. Conf. Computer-Aided Design, pp.422-425, Nov. 1989.

[19] G. Hachtel, R. Jacoby, P. Moceyunas and C. Morrison. "Performance Enhancements in BOLD Using Implications", in Proc. IEEE Int. Conf. CAD, pp.94-97, Nov. 1988.

[20] S.J. Hong, R.G. Cain, and D.L. Ostapko, "MINI : A Heuristic Approach for Logic Minimization", IBM J. Res. Develop., vol. 18, pp.443-458, Sept. 1974.

[21] K. Keutzer, "DAGON : Technology Binding and Local Optimization by DAG Matching", in Proc. 24th ACM/ IEEE Design Automation Conference, pp.341-347, June 1987.

[22] K. Keutzer and M. Vancura, "Timing Optimization in a Logic Synthesis System", in *Logic and Architecture Synthesis for Silicon Compilers*, G. Saucier and P.M. McLellan(ed.), North-Holland, 1989.

[23] T.J. Kowalski, "The VLSI Design Automation Assistant : An Architecture Compiler", in *Silicon Compilation*, D. Gajski(ed.), Addison Wesley Pub., Reading, Mass., 1987.

[24] M.C. McFarland, A.C. Parker, R. Camposano, "Tutorial on High-Level Synthesis", in Proc. 25th ACM/ IEEE Design Automation Conference, pp.330-336, June 1988.

[25] 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.

[26] A.R. Newton, and A.L. Sangiovanni-Vincentelli, "CAD tools for ASIC design", *Proceedings of IEEE*, vol. 75, no. 6, pp.765-776, June 1987.

[27] W. Rosenstiel, D. Schmid, "Logic Synthesis", in *Logic Design and Simulation*, E. Horbst(ed.), North-Holland, 1986.

[28] D.E. Thomas, P. Moorby, *The Verilog Hardware Description Language*, Kluwer Academic Publishers, Boston, Mass., 1991.

[29] H.J. Touati, C.W. Moon and R.K. Brayton, "Performance-Oriented Technology Mapping", in Proc. of Sixth MIT Conf., MIT Press, pp.79-97, 1990.

[30] Defining Synthesis, Synopsys Inc., Aug. 1990.

[31] Design Compiler Reference Manual Version 2.0, Synopsys Inc., March 1991.

[32] Library Compiler Reference Manual Version 1.2, Synopsys Inc., Nov. 1989.

[33] IEEE Standard VHDL Laguage Reference Manual, IEEE, Mar. 1988.

[34] 전홍신,이해동,황선영, "서강 실리콘 컴파일러의 High Level Synthesis 시스템 설계", 전자공학회논문지, 제 28-A권 제 6호, pp.82-93, 1991년 6월.

[35] 김태선, 황선영, "논리 회로의 기술 매핑 시스템 설계", 전자공학회논문지, 제 29-A권 제 2호, pp. 147-158, 1992년 2월.

[36] 이재형,황선영, "성능 구동 자동 논리 회로 설계 시스템", 전자공학회논문지, 제 28권-A권 제 1호, pp.74-84, 1991년 1월.

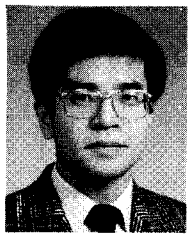
[37] 이재형,황선영, "Fanout 제약 조건하의 논리 회로 합성", 전자공학회논문지, 제 28-A권 제 5호, pp. 387-397, 1991년 5월.

[38] 황선영, "Silicon Compilation", 대한전자공학회 씨에이디연구회지, 제 1권 제 1호, pp.4-12, 1992년 1월.

[39] 이봉선,황선영, "행위 단계 VHDL 합성 시스템을 위한 중간 언어의 설계", 한국정보과학회 논문지 제출.

[40] 임춘석,황선영, "다단 논리 최적화 시스템의 설계", 전자공학회논문지 제 29-A권 제 4호, 1992년 4월

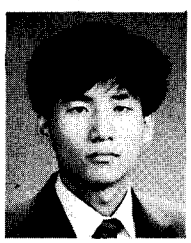
筆 者 紹 介



黃 善 泳
1954年 5月 20日生
1976年 2月 서울대학교 공대 전자공학과(학사)
1978年 2月 한국과학원 전기 및 전자공학과(석사)
1986年 9月 Stanford대학 전기전자공학과(박사)

1976年 3月 ~ 1981年 7月 삼성반도체(주) 연구원, 팀장
1986年 7月 ~ 1989年 1月 Stanford대학 CIS연구소 연구원
1986年 12月 ~ 1988年 2月 Fairchild Semiconductor PARC 기술자문
1989年 3月 ~ 현재 서강대학교 전자공학과 교수

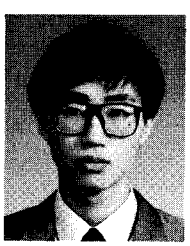
주관심분야 : CAD, Computer Architecture 및 System Design, VLSI 설계



李 在 亨
1968年 4月 6日生
1990年 2月 서강대학교 전자공학과(학사)
1992年 2月 서강대학교 대학원 전자공학과(석사)

1992年 2月~현재 선경정보통신연구소

주관심분야 : CAD 시스템, Computer Architecture 및 VLSI 설계, 이동통신 등.



金 泰 善
1968年 1月 18日生
1990年 8月 서강대학교 공대 전자공학과(학사)

주관심분야 : CAD 시스템, Computer Architecture 및 VLSI 설계 등.