

# ASIC설계를 위한 하드웨어 할당 및 바인딩 알고리즘

최 지 영<sup>†</sup> · 인 치 호<sup>\*\*</sup> · 김 희 석<sup>\*\*\*</sup>

## 요 약

본 논문에서는 ASIC 설계를 위한 하드웨어 할당 및 바인딩을 동시에 수행하는 새로운 하드웨어 할당 및 바인딩 알고리즘을 제안한다.

제안된 알고리즘은 스케줄링의 결과를 입력으로 받아들이고, 각 기능 연산자에 연결된 레지스터 및 연결 구조가 최대한 공유하도록 게이스탬마디 연산과 기억 소자의 상호 연결 관계를 고려하여 기능 연산자, 연결 구조 및 레지스터를 동시에 할당 및 바인딩을 한다. 특히 레지스터 할당은 그래프 킬리딩을 이용하여 최적의 할당을 수행한다.

제안된 알고리즘은 비교 실험을 통하여 기존의 기능 연산자와 레지스터의 수를 미리 정했거나, 분리하여 수행한 방식들과 비교함으로써 본 논문의 효율성을 보인다.

## A Hardware Allocation and Binding Algorithm for ASIC Design

Ji-Young Choi<sup>†</sup> · Chi-Ho Lin<sup>\*\*</sup> · Hi-Seok Kim<sup>\*\*\*</sup>

## ABSTRACT

This paper proposes a hardware allocation and binding algorithm for ASIC Design.

The proposed algorithm works on scheduled input graph and simultaneously allocates and binds functional units, interconnections and registers by considering interdependency between operations and storage elements in each control step. In order to share registers and interconnections connected to functional units, as much as possible. Especially, the register allocation is executed the allocation optimal using graph coloring.

This paper shows the effectiveness of the algorithm by comparing experiments to determine number of functional unit and register in advance or to separate executing allocation and binding of existing system.

### 1. 서 론

특수 목적 및 소량 생산이 요구되는 ASIC (Application Specific Integrated Circuit)이 산업 전반에 걸쳐 제품의 소형화와 고급화의 필수적인 요건이 되고 있기 때문에 집적회로 설계에서 제작에 이르는 회송(turn-around)시간의 단축과 비용의 감소를 위해 CAD 기술

은 필수적이다[1]. 또한 설계할 IC칩의 동작 기술로부터 칩 제조를 위한 마스크(mask) 도면을 자동으로 생산하는 설계자동화(Design Automation)는 CAD (Computer-Aided Design)기술의 최종 목표이다[2-4].

현재 RT레벨의 회로 기술로부터 게이트 레벨의 회로를 설계하는 과정의 논리 설계나[5], 테크놀로지 라이브러리로부터 매핑된 각 회로소자를 배치(placement), 배선(routing)하여 칩 제조를 위한 마스크 도면을 생성하는 단계의 레이아웃 설계 자동화의 많은 연구가 진행되어 상용화되고 있으나[6], VLSI의 집적도와 복잡도의 증가에 따라 짧은 설계 시간, 설계 초기 단계에서 칩의

※ 본 연구는 과학기술부·한국과학재단지정 청주대학교 정보통신 연구센터의 지원에 의한 것입니다.

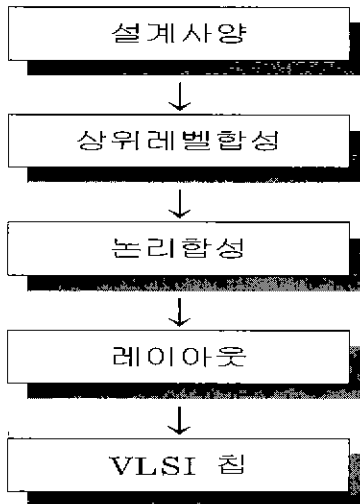
† 정희원 : 청주대학교 대학원 전자공학과

\*\* 정희원 : 세명대학교 컴퓨터학과 교수

\*\*\* 정희원 : 청주대학교 전자공학과 교수

논문접수 : 1999년 11월 11일, 심사완료 : 2000년 3월 24일

성능 평가에 따른 다양한 설계, 자동화에 의한 디버깅 시간 단축 및 적은 오류, 설계 피정에 대한 다큐멘테이션등의 특징을 가진 상위 레벨 합성에 대한 연구가 미비한 실정이다[7,8]. (그림 1)은 CAD 기술에 의한 VLSI (Very Large-Scale Integration)설계 과정을 나타낸다.



(그림 1) VLSI 설계 과정

상위 레벨 합성의 궁극적인 목적은 원하는 기능의 동작 기술(description)을 주어진 비용 함수 값이 최소가 되도록 주어진 하드웨어 소자로 구현하는 것이다. 그래서 상위 레벨 합성을 정의하자면 설계하고자하는 시스템의 동작 기술로부터 주어진 제한 조건과 목적 함수를 만족하는 레지스터 전송(register-transfer)레벨의 구조를 생성하는 단계로서 스케줄링, 할당, 바인딩으로 구성된다. 스케줄링(scheduling)은 동작기술에서 연산(operation)들을 특정한 제어스텝에 할당하는 과정이다[9]. 그러나 응용 분야에 따라 스케줄링 과정에서 고려해야 할 항목들(조건 분기, 파이프라인, 루프등)이 많기 때문에 제한적인 응용 분야에 대한 해를 구하는 알고리즘들이 개발되어 왔다 대표적인 접근 방법은 가장 간단한 방법으로 연산을 가장 빨리 스케줄될 수 있는 제어구간에 할당하는 방식인 ASAP(As Soon As Possible), ASAP 방법과 유사하지만 데이터 흐름에 따라 각 연산이 동작할 수 있는 제일 늦은 시간에 각 연산을 할당하는 방법인 ALAP (As Late As Possible)이 있고, 또한 시간 제한에 가장 일반적으로 사용되는

스케줄링 기법으로 force-directed scheduling, 마지막으로 우선 순위 함수에 의해 순차적으로 연산의 동작 시간을 결정하는 방법으로 list scheduling방법 등이 있다. 할당(allocation)은 구현되는 하드웨어 면적이 최소가 되도록 연산을 기능 연산자(functional unit)에, 변수(variable)를 레지스터에 지정하고 레지스터와 연산자 사이의 연결구조(interconnection)로 버스(bus)나 멀티플렉서(multiplexer)를 할당하는 과정이다[10] 대표적인 방법은 순차적으로 각 시간 구간에서 동작하는 변수와 연산에 대한 하드웨어 자원을 할당하는 greedy allocation[11], 변수들의 생성 시간이 증가하는 순서로 정렬된다. 정렬된 변수들에 대해 라이프타임(lifetime)을 계산한 후 정렬된 순서로 차례차례 검색하면서 변수 수명이 증첩되지 않는 변수들을 레지스터에 할당하는 left-edge algorithm[12]. 그래프 이론을 이용한 접근 방법으로 연산자의 메모리의 할당에 적용되는 clique partition[13]방법 등이 있다.

기존의 할당과 바인딩 방법으로 HAL[14] 시스템은 각 기능 연산자의 형(type)을 한 개로 가정한 상태에서, 기능 연산자의 할당과 스케줄링을 함께 수행하고, clique partition을 사용하여 레지스터와 연결 구조를 할당 및 바인딩 하였다. Splicer[15] 시스템은 기능 연산자의 수를 기능 연산자의 수를 미리 정한 상태에서 스케줄링을 수행하고, 상태 경계를 지나가는 최대 데이터 선의 수를 레지스터의 수로 결정하며 branch and bound방법을 사용하여 연결 구조를 최소화하였다. 그러나 splicer 시스템은 기능 연산자와 레지스터의 수를 미리 정한 상태에서 할당과 바인딩을 하였으므로, 최적의 설계를 얻을 수 없다 REAL[16]은 Left Edge 알고리즘을 이용하여 할당하는데 상호 배타적이지 않은 경우 최소의 레지스터가 할당된다 그러나 연결구조에 끼치는 영향은 고려하지 않았고, 기능 연산자와 연결 구조 할당은 다룰 수가 없었다. 기존의 방법은 첫째 레지스터와 기능 연산자의 수와 형을 미리 정했거나, 둘째 할당과 바인딩을 분리하여 수행하였다. 그러므로 기존의 접근 방식들은 최적의 방법이라고 할 수 없다.

따라서 본 논문에서는 ASIC 설계를 위한 새로운 하드웨어 할당과 바인딩을 동시에 수행하고, 각 기능 연산자에 연결된 레지스터(register) 및 연결 구조(interconnection)가 최대한 공유하도록 제어 스텝마다 연산

과 기억 소자(storage element)의 상호 연결 관계를 고려하여 할당 바인딩 함으로써 기존의 문제점을 해결함과 동시에 진척 비용을 줄인다. 또한 레지스터 할당에서 그래프 컬러링을 이용하여 최적의 레지스터 할당을 한다

본 논문의 구성은 서론 다음으로 2장에서는 본 논문에서 제안한 하드웨어 할당 및 바인딩 알고리즘을 다루고, 3장은 비교 실험을 통하여 본 알고리즘의 효율성을 보이고, 마지막 4장은 본 논문의 결론으로 구성하였다.

## 2. ASIC 설계를 위한 하드웨어 할당 및 바인딩 알고리즘

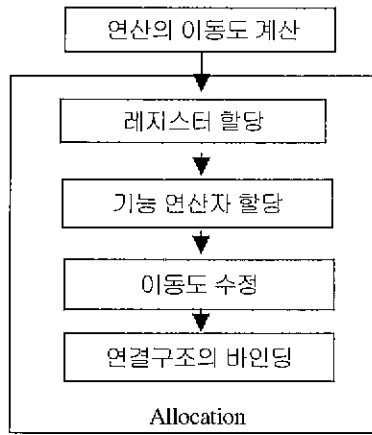
본 논문에서 제안한 하드웨어 할당 및 바인딩에 대한 알고리즘은 (그림 2)와 같다. 입력은 스케줄링 결과를 받으며 전 처리 과정으로 기능 연산자가 할당 및 바인딩 될 모든 연산의 이동도를 계산한다. 연산의 이동도는 데이터의 의존도를 조사하여 구한다. 전 처리 과정에서 각 연산의 이동도를 계산한 후 첫 번째 제어 스텝부터 단계별로 레지스터, 기능 연산자, 연결 구조를 할당 바인딩 한다. 이때 한 제어 스텝에 대해서 기능 연산자의 할당 및 바인딩이 끝나면 다음 제어 스텝들에서 존재하는 연산의 이동도들을 수정한다. 전체 제어 스텝에 대해서 하드웨어 할당 및 바인딩을 수행한 후, 연결 구조 병합을 행한다

```

Input_mobility( ), /* 모든 연산의 최대 구간 */
while(control step) {
    Register_allocate( ); /* 레지스터 할당 바인딩 */
    Function_allocate( ); /* 기능 연산자 할당 바인딩 */
    Mobility_modify( ), /* 이동도 수정 */
    Inter_allocate_merge( ), /* 연결 구조 할당 및 바인딩 및 병합 */
}
    
```

(그림 2) 하드웨어 할당 및 바인딩 알고리즘

(그림 3)은 본 연구의 전반적인 할당과정의 흐름도이다. (그림 2)의 설명한 바와 같이 레지스터 할당, 기능 연산자 할당, 연결구조 바인딩을 순차적으로 처리한 과정을 도식화하였다.



(그림 3) 전체적인 할당과정의 흐름도

### 2.1 레지스터의 할당과 바인딩

레지스터의 할당 및 바인딩은 단계별 제어 스텝마다 다음과 같이 수행한다. 첫 번째 레지스터가 할당될 형을 분류한다. 즉, 변수 또는 상수 이전의 제어 스텝에서 행해진 기능연산자의 출력인지 분류한다. 두 번째, 분류된 형에 따라 레지스터에 할당한다. 변수인 경우, 첫 번째 제어 스텝이면 새로운 레지스터에 할당하고, 그 다음 제어 스텝이면 이전 제어 스텝에서 사용한 레지스터를 그대로 할당한다. 즉, 다른 제어스텝이었던 중복되는 레지스터 할당에 대해서는 그래프 컬러링을 이용한 최적의 레지스터 할당을 수행한다. 각 변수의 입력 별로 생존 주기를 구성한 후 간섭 그래프를 생성한다. 만약 간섭 그래프에서 가용한 레지스터를 k개라고 가정했을 때  $degree(n) < k$  (n: 노드, k: 가용한 레지스터의 수)를 가진 노드가 없으면 대피시키는 대신 스택에 위치할 노드를 삽입한다. 컬러링은 스택에서 색을 이용 가능하다고 가정할 후에 노드가 스택에서 판될 때 색이 이용가능하지 않으면 노드는 채색하지 않고 놓아두고 최적의 컬러링을 수행한다. (그림 5)는 컬러링 알고리즘의 예이다. 반면 상수인 경우 레지스터 할당에서 제외된다. 이전 제어 스텝에서 행해진 기능 연산자의 출력인 경우, 먼저 다른 연산의 입력인지 조사한다. 다른 연산의 입력인 경우, 기능 연산자의 형과 입력을 받는 연산의 형을 고려하여 레지스터에 할당한다. 다른 연산의 입력인 경우가 아니면 기능 연산

자의 형만을 고려하여 레지스터에 할당한다. 레지스터의 할당과 바인딩 알고리즘은 (그림 4)와 같다.

```

Register_allocate( )
{
    Input_type_search( ); /* 입력 값의 형 분류 */
    if(control_step > 1) then {
        Reg_chart_search( ),
        /* 이전 제어 스텝에서 할당된 레지스터 파악 */
        Variable_allocate( );
        /* 변수에 레지스터 할당하기 */
        In_fun_allocate( );
        /* 기능 연산자의 출력에 레지스터 할당 */
        Register_chart( );
        /* 레지스터 정보를 지닌 표 작성 */
    }
}
    
```

(그림 4) 레지스터 할당 및 바인딩 알고리즘

```

if(node) {
    color_stack_pop( ); /* 스택의 역순으로 pop */
    if(dgree(n) > k) {
        Non_coloring( ); /* 컬러링 하지 않음 */
        Spill_code( ); /* 대피 코드 삽입 */
    }
    else
        Coloring( ); /* 컬러링 함 */
}
    
```

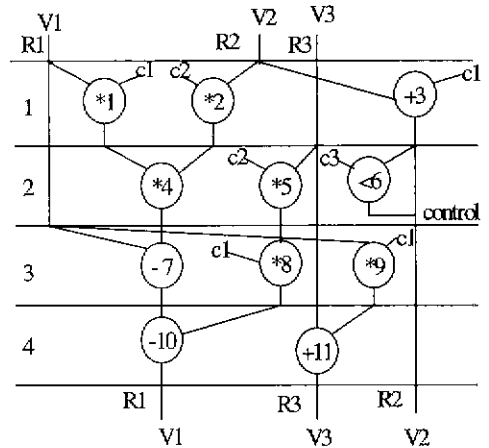
(그림 5) 컬러링 알고리즘

<표 1>은 제어 스텝 1과 제어 스텝 2에 해당되는 레지스터 할당을 보여준다.

<표 1> Register\_chart

CS	레지스터				
1	R1	R2	R3		
2	R1	R2	R3	R4	R5

루프가 존재하는 경우 한 루프의 처음과 끝에 사용되는 레지스터를 동일한 것으로 (그림 6)와 같이 할당한다. 다시 말하면, 각각의 변수 V1, V2, V3는 첫 번째 제어스텝에 따라 레지스터 R1, R2, R3로 할당됨과 동시에 루프가 존재하는 즉, 사이클인 경우는 마지막 제어스텝에 동일한 레지스터 각 R1, R3, R2로 할당된다.

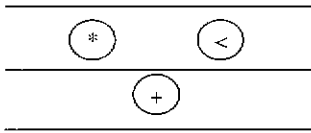


(그림 6) 루프가 포함된 레지스터 할당 및 바인딩

### 2.2 기능 연산자의 할당 및 바인딩

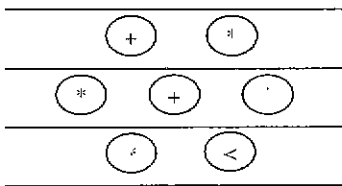
레지스터의 할당과 바인딩을 수행한 후, 현재 제어 스텝에 존재하는 각 연산에 대해서 기능 연산자를 할당 및 바인딩을 수행한다. 즉 각 연산의 계산된 수행 시간을 만족하면서 최대한 작은 면적을 지닌 기능 연산자를 셀 라이브러리에서 선택하기 위해서 기능 연산자를 할당 및 바인딩할 연산의 다섯 변수 즉, 자체 분포수, 상대 분포수, 자체 고정수, 상대 고정수 및 자체 이동도를 조사한다. 먼저 자체 분포수 (self distributed number)는 기능 연산자를 할당하고자 하는 연산과 같은 제어 스텝에 있고 같은 형을 지닌 연산의 개수를 나타내고, 상대 분포수 (relative distributed number)는 기능 연산자를 할당하고자 하는 연산과 다른 제어 스텝에 있고 같은 형을 지닌 연산의 최대 개수를 나타낸다. 또한 자체 고정수 (self fixed number)는 기능 연산자를 할당하고자 하는 연산의 이동도와 같은 제어 스텝에 존재하고 같은 형을 지닌 연산의 이동도를 조사했을 때 이동도가 영인 개수이고, 상대 고정수 (relative fixed number)는 기능 연산자를 할당하고자 하는 연산과 다른 제어 스텝에 있고 같은 형을 지닌 연산의 이동도를 조사했을 때 한 제어 스텝 당 이동도가 영인 최대의 개수이다. 자체 이동도 (self mobility)는 기능연산자를 할당하고자 하는 연산의 이동도이다. (그림 7)은 할당 및 바인딩할 때 사용되는 자체 분

포수를 적용한 것이다 첫 번째 제어스텝에 존재하는 곱셈 연산의 자체 분포수는 1이다.



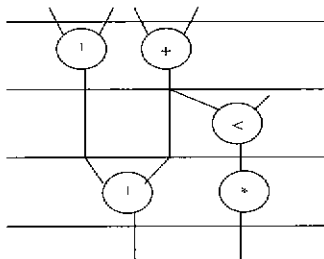
(그림 7) 자체 분포수의 예

(그림 8)은 상대 분포수의 예이다 첫 번째 제어스텝에 존재하는 곱셈 연산의 상대 분포수는 아래와 같이 결정된다. 두 번째 제어스텝에 존재하는 곱셈 연산의 개수가 2, 세 번째 제어스텝에 존재하는 곱셈 연산의 개수가 1인데 최대 값을 취하므로 곱셈 연산의 상대 분포수는 2이다



(그림 8) 상대 분포수의 예

(그림 9)은 자체 고정수의 대한 예이다 첫 번째 제어스텝에 존재하는 곱셈 연산은 두 번째 제어스텝에서도 수행 가능하므로 이동도는 1이다. 따라서 자체 고정수는 0이다.



(그림 9) 자체 고정수의 예

기능 연산자를 할당 바인딩하는 예로, (그림 6)의 비교기(comparator)에 기능 연산자를 할당하는 과정을 살펴본다. 비교기의 초기 입력은 두 번째 제어스텝에 스케줄링이 되었다. 그러나 비교기의 변수들을 조사한 결과 비교기에 두 번째 제어스텝과 세 번째 제어스

텝 즉 두 개의 제어스텝을 차지하는 기능 연산자를 할당하는 것이 가능하다. 다시 말하면, 하나의 연산이 복수개의 제어스텝에 걸쳐 있는 멀티사이클링(multi-cycling)이다. 그러므로 셀 라이브러리 상에서 지연시간이 고정시간과 같고 가능한 작은 면적을 지닌 기능 연산자를 선택하여 비교기에 할당한다. 고정시간이란 두 개의 제어스텝의 시간에서 연결구조 지연시간과 레지스터 지연시간을 제외한 시간을 말한다. 기능 연산자의 할당과 바인딩 알고리즘은 (그림 10)과 같다.

```
Function_allocate (
{
  if (control step ==1) then 네 변수 조사( );
    /* 자체 분포수, 자체 고정수, 상대 분포수,
      상대 고정수 조사 */
  if(같은 연산 && 같은 이동도) {
    Existed_function_allocate ( ),
    /* 기존에 존재된 기능 연산자를 연산에 할당 */
    Function_unit_allocate ( ),
    /* 기능 연산자 할당 */
    Function_register_chart( );
    /* 기능 연산자와 레지스터 정보를 지닌 표 작성 */
  }
}
```

(그림 10) 기능 연산자의 할당 및 바인딩 알고리즘

전체적인 기능 연산자와 레지스터의 테이블 (Function\_register\_chart)은 <표 2>와 같다. 예를 들어 첫 번째 제어스텝에서 기능 연산자 FU1(\*)은 인덱스가 1이고, 각각 R1과 c1을 입력으로 받는다.

<표 2> Function\_register\_chart

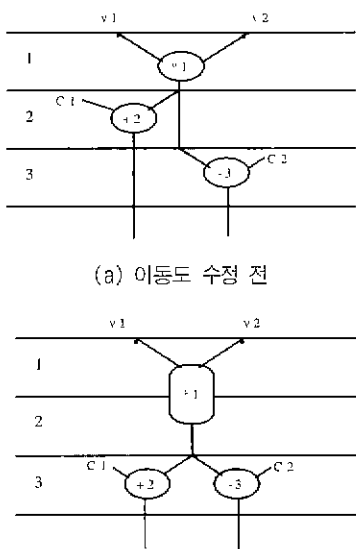
	FU1(*)			FU2()			FU3(+)			FU4(<)			FU5(-)		
	op	a	b	op	e	b	op	a	b	op	a	b	op	a	b
S1	1	R1	c1	2	c2	R2	3	R3	c1						
S2	4	R4	R5	5	c2	R3				6	R2	c3			
S3	8	c1	R5	9	R1	c1							7	R1	P1
S4							11	R3	R5				10	R1	R4

첫 번째 제어스텝인 경우는 자체 분포수, 자체 고정수, 자체 이동도, 상대 분포수를 고려하여 기능 연산자를 할당한다. 그 다음 제어스텝부터는 먼저 존재한 연산에 기능 연산자를 조사한다. 만약, 기능 연산자를 할당할 연산에 적합한 기능 연산자가 존재하면 그대로 할당하고, 존재하지 않으면 위의 다섯 변수를 조사하여 기능 연산자를 할당한다.

3.3 이동도 수정 단계

연산에 기능 연산자를 할당 시 복수개의 제어스텝을

이용하는 멀티사이클링(multi-cycling)을 이용한 경우, 그 연산에 매달린 모든 연산에 대해서 이동도를 조정한다. 복수개의 제어스텝은 라이브러리 상에서 릴레이 교정시간과 같다. 예로, (그림 11) (a)에서 \*1은 자체 분포수가 1, 자체 이동도가 2이고 자체 고정수, 상대 분포수, 상대 고정수가 모두 0이므로 복수개의 제어스텝을 이용하여 기능 연산자를 할당 및 바인딩한다. 따라서 \*1의 출력을 받는 +2는 이동도가 1에서 0으로 바뀐다 (그림 11)의 (b)는 할당 및 바인딩된 결과이다.

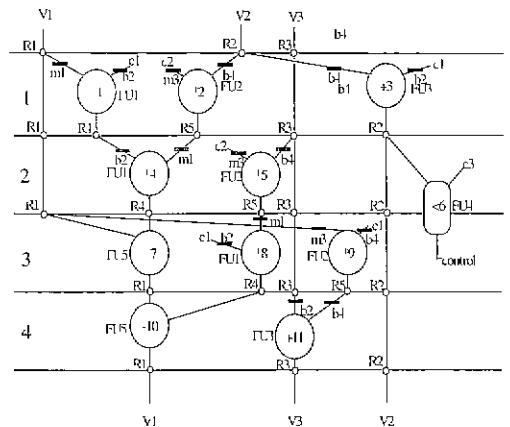


(a) 이동도 수정 전  
(b) 이동도 수정 후  
(그림 11) 이동도 수정의 예

3.4 연결 구조의 할당 및 바인딩

기능 연산자의 할당 및 바인딩을 수행한 후, 연결 구조의 할당 및 바인딩을 수행한다. 첫 번째, 제어스텝인 경우 각 연산에 대해 새로운 멀티플렉서를 할당한다 두 번째, 제어스텝부터는 기능 연산자의 행과 입력 형을 조사하여 가능한 같은 형을 찾아 멀티플렉서를 할당한다. 세 번째, 멀티플렉서의 입력 수를 조사한다. 한 개인 경우 멀티플렉서를 생략하고, 흰 개가 아닌 경우 각 멀티플렉서의 제어스텝을 조사한다 제어스텝이 서로 중복되지 않거나, 중복이 되어도 입력 값이 동일하면 병합한다. 이때 병합한 멀티플렉서는 비스로 대체한다 실제적인 예로 (그림 12)를 보면, 두 번째 제어스텝에 존재하는 5와 첫 번째 제어스텝에 존재하는 \*2는 같은 기능 연산자(FU2)에 할당 및 바인

딩되었다. 그러므로 \*5의 연결구조는 \*2의 연결구조와 같은 것으로 할당 및 바인딩한다. 이 때 두 개의 연결 구조에 대한 구별된 할당 및 바인딩은 형을 고려하여 수행한다 (그림 12)는 전체적인 레지스터, 기능 연산자, 연결구조의 할당 바인딩의 결과이다.



(그림 12) 전체적인 할당 바인딩의 결과

4. 실험 결과

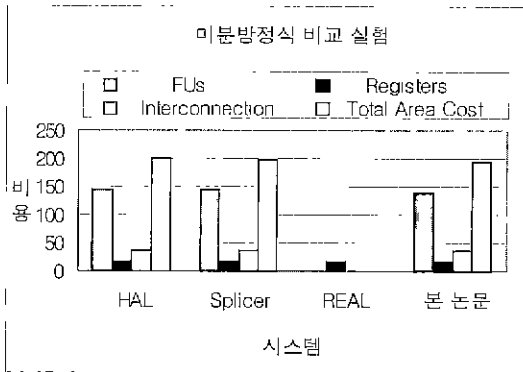
본 논문의 하드웨어 할당 및 바인딩 알고리즘 결과를 HAL, Splicer, REAL 시스템의 결과와 비교하였으며 정확한 비교를 위하여 HAL 시스템에서 사용하는 force directed scheduling을 적용하여 나온 결과를 입력으로 받는다

4.1 미분 방정식(Differential equation)을 적용한 실험

<표 3> Differential equation에 대한 면적 비용(area cost)를 HAL, Splicer, REAL 시스템과 비교한 결과이다 본 알고리즘을 적용시킨 결과 상위 레벨 합성 중 할당과 바인딩을 분리하여 수행한 Splicer, REAL, HAL 시스템은 동일하나 본 논문은 기존의 시스템보다 기능 연산자의 면적 비용 4.2% 줄어서 전체 면적 비용(total area cost)이 줄어드는 효과를 얻었다

<표 3> Differential equation의 비교 실험

	HAL	Splicer	REAL	본 논문
FUs	145	145	-	139
Registers	18	17	18	18
Interconnection	37	35	-	37
Total Area Cost	300	197	-	194

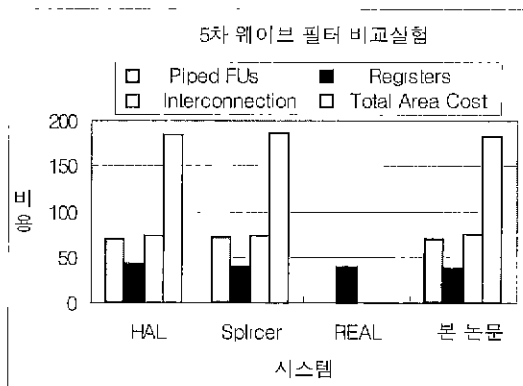


4.2 5차 웨이브 필터(Fifth-order elliptic wave filter)를 적용한 실험

<표 4>는 벤치마크 모델로서 1988년 High Level synthesis Workshop에서 표준 벤치마크 모델로 채택된 Fifth-order elliptic wave filter에 대한 면적 비용(area cost)을 HAL 시스템과 비교한 결과이다. 기능 연산자의 비용은 동일하고 연결 구조의 비용은 다소 증가하였으나, IC 설계시 레이아웃 설계에서의 배선 영역 연결구조는 상위 레벨에서는 고려하지 않으므로 큰 문제는 되지 않는다. 또한 레지스터의 비용이 HAL 시스템과 비교하여 9.6% 줄어들어서, 결과적으로 전체 면적 비용이 줄어들었다.

<표 4> Fifth-order elliptic wave filter의 비교 실험

	HAL	Splicer	REAL	본 논문
Piped FUs	70	72	-	70
Registers	42	40	40	38
Interconnection	73	74	-	75
Total Area Cost	185	186	-	183



5. 결론

근 논문에서는 ASIC 설계를 위한 하드웨어 할당 및 바인딩 알고리즘을 제안하였다.

본 알고리즘은 AVION SYS에서 구현하였으며 다음과 같은 특징을 가지고 수행한다 첫 번째 게이트 스태부터 단계별로 레지스터, 기능인산자, 연결구조를 할당 및 바인딩을 수행한 후 연결 구조 병합을 행한다. 종속 관계에 있는 할당과 바인딩을 동시에 수행함으로써, 성능 평가를 위한 비교 실험결과에서 볼 수 있듯이 미분 방정식을 적용한 실험은 HAL 시스템과 비교하여 기능 연산자의 면적 비용이 42% 줄어들어 전체 면적 비용의 감소를 보였고, 표준 벤치마크 모델인 5차 웨이브 필터에 적용한 HAL 시스템에서는 연결구조의 비용은 1% 증가하였으나, 반민 레지스터의 면적비용이 9.6% 감소하였으므로 결과적으로 기존의 연산자의 수를 미리 정했거나 할당과 바인딩을 분리하여 수행한 방식보다 작은 전체 칩 면적비용을 얻을 수가 있었다 반면에 Splicer는 HAL 과 유사한 면적 비용을 보이지만 REAL은 기능연산자와 연결구조의 비용을 고려하지 못하였다 결과적으로 본 논문의 실험 결과가 상위 레벨 합성 기술의 궁극적 목표인 주어진 하드웨어 비용 함수 값이 최소가 되도록 하는 효율성을 보였다.

또한 향후 연구 과제로는 ASIC 설계를 위한 하드웨어 할당 및 바인딩 알고리즘을 바탕으로 다양한 성능 평가와 합성 결과에 대한 예측과 평가에 대한 연구가 선행되어야 하겠다

참 고 문 헌

- [1] M. A. Breuer, Digital System Design Automation, Computer Science Press, Inc, 1975.
- [2] S. M Rubin, Computer Aided for VLSI Design, Addison-Wesley.
- [3] S. G. Shiva. "Automatic Hardware Synthesis," Proceedings of the IEEE, Vol 71, No1, pp.76-87, Jan.1983
- [4] Daniel D Gajski, Silicon Compilation, Addison-Wesley. 1988
- [5] R. K. Brayton, A L Sangiovanni-Vincentelli, and G. D. hatchtel, "Multi-level Logic Synthesis," Proceedings of the IEEE, Vol.78, No.2, pp.264-300, Feb. 1990.

[6] E. S. Kuh and T. Ohtuska, "Recent Advance in VLSI Layout". Proceedings of the IEEE, Vol 78, No 2, pp.237-263, Feb 1990

[7] M. C. McFarland, A. C. Parker, and R. Camposano, "The High-Level Synthesis of the Digital System," Proceedings of the IEEE, Vol 78, No.2, pp.301-318, Feb. 1990.

[8] C. Y. Hitchcock, D. E. Thomas, "A Method for Automatic DataPath synthesis," Proc. for the 20th Design Automatic Conference(DAC), pp.484-489. 1983.

[9] R. Camposano, "From Behavior to Structure : High-Level Synthesis," IEEE Design & Test of Computer, pp.8-19. Oct.1990.

[10] James R. Armstrong, F. Gail Gray. "Structured Logic Design With VHDL," 1993

[11] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu. "High -Level Synthesis : introduction to chip and system design." pp.272-277, 1992.

[12] James R. Armstrong, F. Gail Gray. "Structured Logic Design With VHDL," pp 391-393, 1993.

[13] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu. "High-Level Synthesis : introduction to chip and system design," pp.277-283. 1992

[14] P. Paulin, J. Knight and E. Girczyc, "HAL : A Multi-Paradigm Approach to Automatic Data Path Synthesis," Proc. of 23rd DAC, pp.263-270, 1986

[15] B. Pangric, "Splicer - A Heuristic Approach to Connectivity Binding." Proc. of the 25th Design Automation Conf.. pp.536-541. 1988

[16] Kurdahi. F. J. and A. C. Parker, "REAL : A Program for register allocation," in Proc. of the 24th Design Automation Conf . pp 210-215, 1987.



**최 지 영**

e-mail : m7515103@kebi.com  
 1997년 세명대학교 전자계산학과 졸업(이학사)  
 1999년 세명대학교 전산정보학과 졸업(이학석사)  
 2000년~현재 청주대학교 전자공학과 박사과정  
 관심분야 : CAD 알고리즘, 상위레벨합성, 저전력 설계



**인 치 호**

e-mail : ich410@venus.semyung.ac.kr  
 1985년 한양대학교 전자공학과 졸업 (공학사)  
 1987년 한양대학교 전자공학과 졸업 (공학석사)  
 1996년 한양대학교 전자공학과 졸업 (공학박사)  
 1992년~현재 세명대학교 컴퓨터과학과 부교수  
 관심분야 : VLSI CAD, ASIC 설계, CAD 알고리즘 등



**김 희 석**

e-mail : khs8391@chongju.ac.kr  
 1977년 한양대학교 전자공학과 졸업 (공학사)  
 1980년 한양대학교 전자공학과 졸업 (공학석사)  
 1985년 한양대학교 전자공학과 졸업 (공학박사)

1987년~1988년 미국 University of Colorado at Boulder  
 객원 교수  
 1996년~1997년 미국 University of California at Irvine  
 객원 교수  
 1981년~현재 청주대학교 전자공학과 교수  
 관심 분야 : CAD, 컴퓨터 구조, 컴퓨터 알고리즘