한국ITS학회논문지
제6권, 제3호 (2007년 12월)
pp.165~173

최적의 설계 자동화를 위한 최소자원 할당 알고리듬

A Minimum Resources Allocation Algorithm for Optimal Design Automation

김영숙* (Young-Suk Kim)　　인치호** (Chi-Ho Lin)

요 약

본 논문에서는 최적의 설계 자동화를 위한 최소자원 할당 알고리듬을 제안한다. 제안된 할당 알고리듬은 연산자 간을 연결하는 신호선이 반복적으로 이용되어 연결 신호선 수가 최소가 될 수 있도록 기능 연산자를 할당한다. 레지스터 할당 시 연결구조에 따라 가중치를 갖는 구간 그래프를 구성한다. 최소의 클러스터 파티션 알고리듬을 이용하여 생성된 최대 크기의 클러스터들에 연결구조가 고려된 레지스터들을 할당한다. 또한 연결구조에 대한 멀티플렉서의 중복 입력을 제거하고 연산자에 연결된 멀티플렉서 간의 입력을 교환하는 입력 정렬 과정으로 연결 구조를 최소화한다. 또한, 벤치마크 실험을 통하여 제안된 알고리즘의 효용성을 보인다.

Abstract

In this paper, we propose a new minimum resources allocation algorithm for optimal design automation. In the proposed algorithm, the operation are allocated to functional units so that the number of interconnection wires between functional units can be minimized. The registers are allocated to the maximal clusters generated by the minimal cluster partitioning algorithm. Finally, the interconnection is minimized by removing the duplicated inputs of multiplexers and exchanging the inputs across multiplexers. The efficiency of the proposed allocation algorithm is shown by experiments using benchmark examples.

Key Words : Allocation, automation, functional unit, interconnection, register

Ⅰ. 서　론

다품종 소량 생산이 요구되는 ASIC (Application Specific IC)이 산업 전반에 걸쳐 제품의 소형화와 고급화의 필수적인 부품이 되고 있기 때문에 집적 회로의 설계에서 제작에 이르는 회송 (turn-around) 시간의 단축 및 비용의 감소를 위해 CAD 기술은 필수적이다. 또한 설계할 IC 칩의 동작 기술로부터 칩 제조를 위한 마스크(mask) 도면을 자동으로 생산하는 설계 자동화 시스템에 대한 연구가 활발히 진행되고 있다 [1-4].

상위 레벨 합성은 설계 하고자 하는 시스템의 동작을 알고리듬 레벨의 특정 언어 (Hardware Description Language, HDL)로 기술하는 동작 기

* 주저자 : 세명대학교 대학원 박사과정
** 공저자 : 세명대학교 컴퓨터학과 교수
논문접수일 : 2007년 10월 23일

술, 하드웨어 기술 언어를 분석하는 HDL 분석기 (analyzer), 연산 (operation)을 특정한 제어 스텝에 할당하는 스케쥴링 (scheduling), 그리고 상위 레벨 합성의 마지막 단계인 하드웨어 할당 (allocation)으로 구성된다 [5 ,6].

하드웨어 할당은 구현되는 하드웨어의 면적이 최소가 되도록 연산 (operation)을 기능 연산자 (functional unit)에, 변수(variable)를 메모리에 지정하고 메모리와 연산자 사이의 연결구조 (interconnection)로 버스 (bus)나 멀티플렉서 (multiplexer)를 할당하는 것이다. 1980년대 중반에 발표된 FACET [2], REAL [7]등에서 제시된 할당 알고리듬은 연산자 사용의 최소화 또는 메모리 사용의 최소화만을 목표로 하여 연결구조에 대한 고려가 없었다. 그러나 연결구조의 비용(cost) 증가가 전체 회로의 단가에 미치는 영향이 커짐에 따라 할당시 연결 구조의 최소화에 대한 반영이 요구되고 있다. 이에 따라 메모리 사용의 최소화보다는 연결구조의 최소화를 고려한 할당 알고리듬들이 제안되었다 [7, 8].

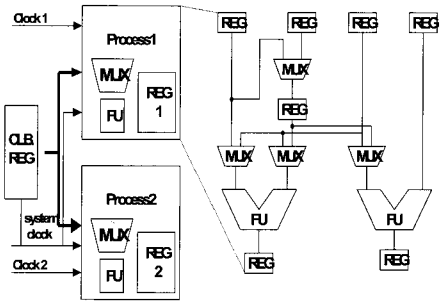
본 논문의 최소 자원 할당 알고리듬은 하드웨어 할당이 전체적인 하드웨어 비용에 큰 영향을 미치기 때문에 이들 자원을 최대한 공유하는 할당 방법, 즉 레지스터와 연결구조의 최소화가 될 수 있는 할당 방법을 제안하였다. 본 논문에서는 스케쥴링의 결과를 입력으로 하여 시간 복잡도와 전체적인 하드웨어 비용을 최소화하기 위해 기능 연산자 및 메모리 할당 과정과 입력 정렬 과정으로 분리해서 수행하고 각 할당 과정에서 각 하드웨어 자원간의 연결구조를 고려한다. 스케쥴링 결과를 입력으로 연산자간을 연결하는 신호선이 반복적으로 이용되어 연결 신호선 수가 최소가 될 수 있도록 연산들을 연산자에 할당한다. 또한 연산자와 레지스터간의 연결구조가 최소가 될 수 있도록 레지스터를 할당한다. 또한 기능 연산자 할당과 레지스터 할당에 의해 결정된 연결 구조에 멀티플렉서의 중복되는 입력을 제거하고 하나의 연산자에 연결된 멀티플렉서간의 입력을 교환함으로서 멀티플렉서 입력을 최소로 하는 입력 정렬 과정으로 연결구조를 최소화한다.

Ⅱ. 최적의 설계 자동화를 위한 최소자원 할당 알고리듬

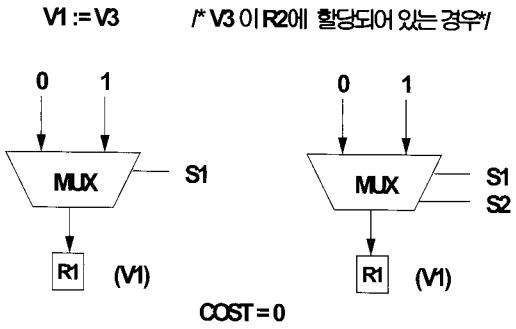
본 논문에서는 연산자에 연결되는 멀티플렉서의 크기를 최소로 하는 것이 목적이 아니고, 고정된 입력의 멀티플렉서를 최소한으로 사용하는 것을 목적으로 한다. 목표 아키텍춰는 <그림 1>과 같이 다단으로 멀티플렉서를 이용하여 연결 구조를 구현하였다. 본 논문의 할당 방법은 고정된 입력의 연결 구조를 갖는 FPGA나 ASIC 라이브러리를 지원하기 때문에 연결 구조 수의 최소화를 목적 함수로 한다. 예를 들면 <그림 2(a)>와 같이 레지스터 R1의 입력 수가 2이고 입력 선택을 위해 4:1 멀티플렉서가 할당되어 있다고 가정하자. 변수 V1이 R1에 새로이 할당될 경우 V1의 입력이 되는 V3가 레지스터 R2에 할당되어 있다면 <그림 2(b)>와 같이 입력 수가 3으로 증가한다. 입력 수가 증가하더라도 입력 연결 구조를 위해 멀티플렉서의 증가는 이루어지지 않는다. 따라서 변수 V1이 레지스터 R1에 할당될 경우 연결 신호선 수는 입력 신호와 멀티플렉서 제어 신호의 증가로 기존에 할당된 신호선에 비해 2만큼의 증가분이 발생하지만 멀티플렉서의 수가 증가하지 않으므로 비용(cost)는 0이 된다.

1. 기능 연산자 할당

기능연산자 할당 과정은 연산을 기능연산자에 할



<그림 1> 목표 아키텍춰
<Fig. 1> Target architecture



(a) 할당 전 (b) 할당 후

(a) Before memory allocation

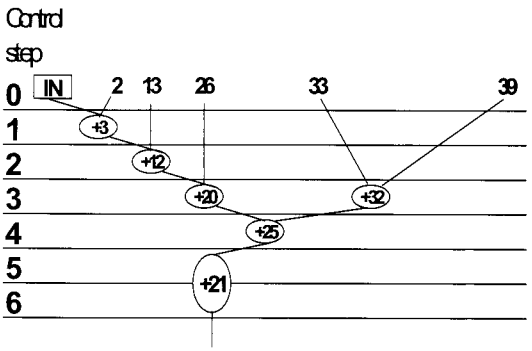
(b) After memory allocation

<그림 2> 비용 계산 예

<Fig. 2> An example of the cost calculation

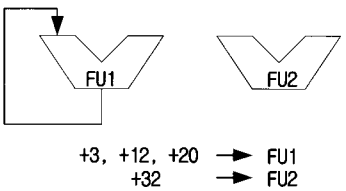
당하는 과정으로서 기능연산자의 수는 스케쥴링 단계에서 결정된 것으로 한다. 할당에 사용되는 연산자는 다기능 연산자가 아닌 단일 기능을 갖는 연산형 별(예, +, *, 카운터 등)로 할당을 수행한다. 연산을 기능 연산자에 할당하는 방법에 따라 연결 구조가 변하므로 연결 구조를 최소화할 수 있도록 연산을 기능 연산자에 할당하여야 한다. 본 논문에서 제안한 기능 연산자 할당은 두개의 연산 종류에 대해 따로 수행한다. 즉, 1개의 연산만이 존재하는 경우와 2개 이상의 연산이 존재하는 경우이다.

특정 형(type)의 연산은 <그림 3(a)>에서 제어 스텝 5에 있는 +21과 같이 하나의 제어 스텝에 올 수 있는 같은 형의 연산의 수가 1개인 연산으로 이 연산을 수행하는 기능연산자를 우선적으로 할당한다. 남은 형의 연산은 <그림 3(a)>에서 제어 스텝 3에 있는 +20과 같이 하나의 제어 스텝에 올 수 있는 같은 형의 연산의 최대수가 2이상인 연산으로 이와 같은 연산이 할당 될 수 있는 기능연산자 수가 N이고 하나의 제어 스텝에 오는 이 연산의 수가 M인 경우에 연산을 기능연산자에 할당하는 방법은 N 순열 M의 가지 수가 있고 할당하는 방법에 따라 연결 구조가 변하게 된다. <그림 3(b)>와 <그림 3(c)>는 기능 연산자의 할당 형태에 따라 연결구조가 변



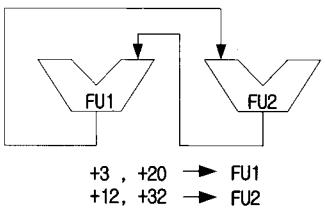
(a) CDFG의 예

(a) An example of CDFG



(b) 기능 연산자의 할당

(b) Functional unit allocation



(c) 연결 구조를 고려한 기능 연산자의 할당

(c) Functional unit allocation for the interconnection

<그림 3> 기능 연산자 할당의 예

<Fig. 3> An example of Functional unit allocation.

하는 예를 보여준다.

<그림 3(b)>는 연산 +3과 +20을 기능연산자 1에, 연산 +12와 +32를 기능연산자 2에 할당한 경우이고 <그림 3(c)>는 +3, +12 와 +20을 기능연산자 1에 +32를 기능연산자 2 에 할당한 경우이다. <그림 3(c)>와 같이 할당할 때 연결 구조가 최소가 될 수 있다. <그림 3(c)>의 할당은 <그림 3(a)>에서 보

면 한 연산이 할당된 기능 연산자와 그 연산의 입력과 출력이 되는 연산이 할당된 기능연산자가 같은 경우로서 기능 연산자 할당과정에서 연결 구조를 최소로 하기 위해 비용을 식(1)같이 설정하고 하나의 제어 스텝에서 연산이 기능 연산자에 할당될 수 있는 경우에 대하여 각각의 비용을 계산한 후 비용이 가장 큰 경우로 연산을 할당한다. 스케쥴링된 결과의 첫 번째 제어 스텝에서 마지막 제어 스텝까지 식(1)을 이용하여 모든 연산을 기능 연산자에 할당한다.

$$\cos t = \sum_{x=1}^{N}\sum_{y=1}^{M}((\text{같은입력}FU\text{수} + \text{같은출력}FU\text{수}) - (\text{다른입력}FU\text{수} + \text{다른출력}FU\text{수})) \quad (1)$$

(N = 같은 type의 연산이 할당되는 기능연산자의 수)

(M = 각 기능연산자에 할당된 연산의 수)

2. 레지스터 할당

본 알고리듬에서는 메모리 소자로서 레지스터를 이용한다. 따라서 메모리 할당 과정은 레지스터의 수를 결정하고 연산의 출력인 변수를 레지스터에 할당하는 과정으로서 변수가 가지는 일반적인 특성과 제약 조건에 의해 크게 좌우된다.

변수의 일반적인 특성은 하나의 입력이 되는 연산을 가지고 하나 이상의 출력이 되는 연산을 가지며 life time을 가진다. Life time은 변수의 입력 연산과 출력 연산 사이의 제어 스텝 구간이며 변수가 레지스터에 할당되는 제어스텝 구간이다. 그러나 VHDL 기술문에서 보면, 메모리에 저장될 변수는 크게 전역(global) 변수와 내부(local) 변수로 구분할 수 있다. 전역 변수는 VHDL 기술문에서 2개 이상의 프로세스(process) 문에서 읽기/쓰기를 하는 변수이고 내부 변수는 하나의 프로세스문 또는 블럭(block)문 내에서만 사용되는 변수이다. 전역 변수의 경우 시스템 레지스터로 사용되는 경우가 많고 또한 프로세스별 또는 블럭별로 합성을 수행하기 때문에 각 프로세스에서 전역 변수의 읽기/쓰기를 예측하기가 어렵다. 따라서 전역 변수는 변수간의 메모리 공유를 하지 않고 별도의 메모리를 할당한다. 본 논문에서 제안하는 메모리 할당 알고리듬은 내부 변수에 대해서만 수행한다.

본 논문에서는 같은 변수 명을 갖는 변수들을 같은 레지스터에 할당한다. 그러나 조건 분기등에 의해 같은 변수명을 갖는 변수들이 다른 조건에 나타나기 때문에 처음부터 하나의 변수로 처리할 경우, 레지스터를 공유할 변수를 찾기가 쉽지 않다. 따라서 같은 변수명을 갖더라도 다른 그래프 경로에 나타나는 변수들은 새로운 변수로 분리한다. 스케쥴링된 그래프의 변수의 life time을 구성한다. 그림 4와 같이 변수의 life time이 구해지면 1) 같은 변수명, 2)life time의 중첩, 및 3) 동작 조건등을 검색 하여 변수명이 같거나, 동작 조건이 중첩되지 않거나, 또는 변수의 life time 이 중복되는 기간 중 동작 조건이 다른 변수들은 레지스터 공유가 가능한 변수라 하고 각 변수간의 레지스터 공유 가능 여부를 나타내는 구간 그래프를 구성한다.

구성된 구간 그래프를 최소의 클러스터 파티션 알고리듬을 적용하여 최소의 클러스터 분할 수를 구성하는 최대 크기의 클러스터들을 구하며 알고리듬은 <그림 4>와 같다.

```
변수들을 register 입력 별로 분리.
각 변수의 life time을 구성.
Life time의 중첩 관계에 의해 구간 그래프 생성.
for(각 변수의 구간 그래프에 대하여)
    최소의 cluster partitioning 알고리듬.
for(각 minimal cluster partitions 에 대하여)
    각 register를 할당.
```

<그림 4> 레지스터 할당 알고리듬
<Fig. 4> Register allocation algorithm

변수의 구간 그래프에 대하여 최소수의 최대의 클러스터들을 구하면 <그림 5>와 같이 4 개의 클러

스터들이 생성되며, 각 클러스터의 변수들에 대해 각각의 레지스터 결정하고 분리된 변수들을 각 결정된 수만큼의 레지스터에 할당한다. 따라서 하나의 레지스터에 할당되는 모든 변수들은 같은 기능 연산자의 입력을 갖게 된다. 이는 기능 연산자의 출력과 레지스터의 입력 사이에 멀티플렉서를 지원하지 않고 레지스터의 출력과 기능 연산자의 입력 사이에만 멀티플렉서를 지원함으로서 멀티플렉서의 입력이 공유되는 경우를 증가시키게 되고 따라서 연결 신호선 수가 줄어든다.

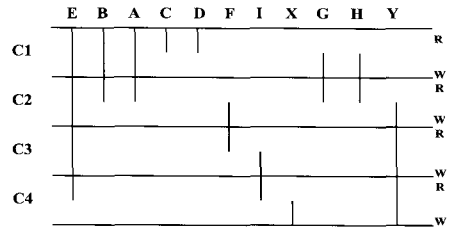
3. 연결구조 바인딩

연결 구조 바인딩 은 연결구조의 공유를 최대화하여 연결 구조의 비용을 최소화하는 단계로서 기능 연산자 할당과 레지스터 할당에 의해 각 기능 연산자에 연결될 신호선이 결정된다. Mux-based ASIC에는 입력수가 고정된 멀티플레서에 의해 연결구조를 구성한다.

따라서 본 논문에서는 연결구조 바인딩은 고정된 입력의 멀티플렉서를 최소함을 목적으로 하며, 설정된 타킷 아키텍춰는 멀티플렉서의 입력을 줄이기 위하여 멀티플렉서의 입력을 정렬(alignment)을 수행한다. 멀티플렉서의 입력 정렬은 덧셈기나 곱셈기와 같이 입력의 위치에 관계없이 연산을 수행할 수 있는 기능 연산자에 대해 두 입력의 위치를 교환함으로서 멀티플렉서의 입력 수를 최소화시키며 동시에 연결구조 수를 최소화시킨다.

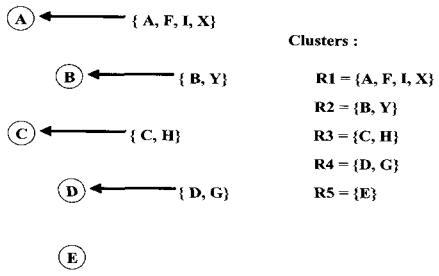
<그림 6(a)>는 레지스터 할당후의 멀티플렉서 입력의 예로서 제어 스텝 1에서 한쪽의 멀티플렉서 입력은 레지스터 1의 출력이고 다른 한쪽의 멀티플렉서 입력은 레지스터 2의 출력이다. 교환법칙의 성질을 이용하여 제어 스텝 1의 멀티플렉서 입력을 바꾸어 주면 <그림 6(b)>와 같이 1개의 2입력 멀티플렉서의 수 가 감소된 회로를 얻을 수 있다. 따라서 입력 정렬 과정은 연결 구조에 대한 하드웨어 비용을 줄이기 위해 필요하다.

이와 같이 멀티플렉서의 수 및 입력을 최소화하기 위한 입력 정렬에 대한 설명을 하기 위해 <그림 7>과 같이 덧셈기에 연결된 입력이 있다고 가정하자. <그림 7>은 각 레지스터의 연결상태를 나타내



(a) 각 변수의 life time 예

(a) Life time example of each variable

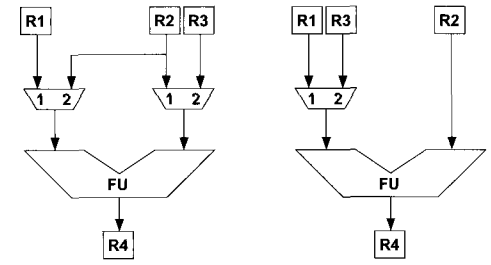


(b) 최종 cluster 예

(b) Final cluster example

<그림 5> 최소의 레지스터 cluster 생성

<Fig. 5> Minimal register cluster generation



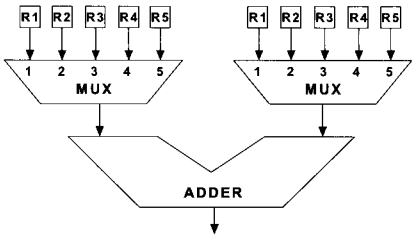
(a) 입력의 정렬 전 (b) 입력의 정렬 후

(a) Before input alignment

(b) After input alignment

<그림 6> 멀티플렉서 입력 정렬

<Fig. 6> The input alignment of multiplexer



<그림 7> 멀티플렉서 입력의 정렬 예
<Fig. 7> An alignment example of multiplexer

어 한 개에 5 입력 멀티플렉서에 의해 연결 구조를 구현할 수 있다고 볼 수 있지만 실제 각 레지스터가 연결되는 시간은 <그림 8(a)>와 같다. <그림 8(a)>와 같이 연결된 입력에 대해 <그림 8(b)>와 같이 레지스터가 덧셈기의 좌, 우 입력에 나타나는 레지스터의 빈도수를 계산한 후 레지스터의 수가 1인 레지스터를 제외하고 레지스터의 총 빈도수가 적은 쪽에서 큰 쪽으로 정렬하여 총 빈도수가 가장 작은 레지스터를 선택한다. 선택된 레지스터는 <그림 6>에서와 같이 위치를 입력수를 줄이기 위해 위치를 이동하게 된다.

<그림 8>에서 총 빈도수가 가장 적은 레지스터 R2를 선택하여 입력의 위치를 교환한 결과는 <그림 9와 같다. <그림 9>에서 보면 레지스터 R2가 좌

제어스텝	입력	
	좌측	우측
1	R1	R3
2	R1	R5
3	R5	R1
4	R3	R2
5	R3	R4
6	R2	R3
7	R3	R4
8	R4	R1
9	R4	R5
10	R4	R3

(a) 제어스텝별 입력
(a) The input per control step

	총빈도수	좌측	우측
R1	4	2	2
R2	2	1	1
R3	6	3	3
R4	5	3	2
R5	3	1	2

(b) 레지스터별 연결
(b) The connection for each register.

<그림. 8> 그림 7. 의 입력 연결 상태
<Fig. 8>The input connection status of Fig. 7.

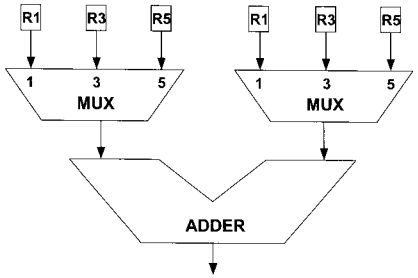
	총빈도수	좌측	우측
R1	4	2	2
R2	2	2	0
R3	6	2	4
R4	5	3	2
R5	3	1	2

(a) 제어 스텝별 입력 연결
(a) The input per control step.

제어스텝	입력	
	좌측	우측
1	R1	R3
2	R1	R5
3	R5	R1
4	R2	R3
5	R3	R4
6	R2	R3
7	R3	R4
8	R4	R1
9	R4	R5
10	R4	R3

(b) 레지스터별 연결
(b) The connection for each register.

<그림 9> 레지스터 R2의 위치 교환
<Fig. 9> The position exchange of register R2.



<그림 10> 멀티플렉서 입력 정렬후의 결과
<Fig. 10> Result after multiplexer input alignment

측 입력에서만 나타나므로 우측 입력의 수가 1만큼 감소하였다. <그림 8> 에서 위치를 이동할 레지스터를 선택하면 총 빈도수가 적은 레지스터 R5가 선택된다. 선택된 레지스터가 이동하기 위해서는 같은 제어스텝에서 연결된 반대쪽 입력 레지스터와 위치 교환을 하여야 한다. 따라서 선택된 레지스터의 연결빈도가 많은 쪽 입력에서 적은 쪽 입력으로 이동하게 되면 상대적으로 많은 레지스터들의 교환이 발생하게 된다. 따라서 연결빈도가 적은 쪽의 입력을 이동하게 된다. <그림 9>에서 선택된 레지스터 R5는 좌측 입력에서 우측 입력으로 이동한다. 이와 같이 멀티플렉서의 수 및 입력을 최소화 과정을 반복하면 <그림 10>과 같이 연결구조인 멀티플렉서의 크기가 5입력에서 3입력으로 감소하였음을 알 수 있다.

이와 같이 멀티플렉서의 수 및 입력을 최소화하기 위한 연결구조 정렬 알고리즘은 그림 11과 같다.

```
for(교환이 되는 할당된 각 functional unit에 대하여)각 register가 multiplexer의 입력과 연결되는 수 계산.
register가 나타나는 수가 증가하는 순으로 정렬.
for(정렬된 register에 대하여)
현재의 multiplexer 입력 수보다 작도록 교환.
```

<그림 11> 연결 구조 바인딩 알고리즘
<Fig. 11> Interconnection binding algorithm.

Ⅳ. 실험 결과

본 논문에서 최적의 하드웨어 할당 알고리즘은 Sparc 시스템에서 C++로 구현하였으며 벤치마크 회로에 대해 다음과 같이 실험을 하였다.

<표 1>와 <표 2>는 fifth-order elliptic filter 와 sixth-order bandpass filter에 대한 실험 결과를 보여준다. 하드웨어 할당에서는 제어 스텝의 수 와 연산자의 수는 기존의 다른 시스템과 같으나, 연결구조의 비용을 고려한 멀티플렉서 입력 수의 측면에서 <표 2>의 HAL [2]과 비교할 때 약 30%만 을 필요로 하며, <표 1>의 MAP[4]과 비교 할 때 약 67%만을 필요로 하는 결과를 보여 주고 있다. 그러나 본 하드웨어 할당이 주로 멀티플렉서를 연결 구조로 사용했기 때문에 버스 측면에서는 유리한 결과를 얻지 못했다.

그러나 본 논문의 하드웨어 할당 결과는 작은 수의 논리 블록에 의해 연결 구조가 구현됨을 보여주며, 하드웨어 할당의 경우 칩 면적을 감소시킬 수 있다.

<표 1> Fifth-order elliptic filter에 대한 실험 결과
<Table 1> Experimental results for the fifth-order elliptic filter.

	본 논문	Map[4]	ADPS[2]	PUBSS[7]
C_steps	11	11	11	11
#ALUs	2	2	2	2
#Multip -liers	1	1	1	1
#MUX_ inputs	8 / (4xMUX21)	12	27	10
#Buses	5	5	N/A	5
#Registers	11	11	14	11

<표 2> Sixth-order bandpass filter에 대한 실험 결과
<Table 2> Experimental results for the sixth-order bandpass filter.

	본 논문	HAL[2]	Map[4]	SPAID[5]
C_steps	19	19	19	19
#ALUs	2	2	2	2
#Multip -liers	1	1	1	1
#MUX_ inputs	9 / (4×MUX21)	26	10	17
#Buses	5	6	5	N/A
#Registers	12	12	14	19

N/A : Not Available.

V. 결 론

본 논문에서는 IC 설계시 전체 면적의 40% 이상이 되는 배선 영역내의 연결구조의 최소화를 고려하여 기능 연산자 및 메모리를 할당하였다. 하드웨어 최적화를 위한 본 알고리듬은 최소의 하드웨어 모듈 사용 및 연결구조 비용의 최소화를 목적으로 한 연산자 및 레지스터 할당 과 기능 연산자 와 메모리 모듈 간의 연결구조의 최소 비용을 위한 바인딩의 과정으로 수행하였다.

제안된 하드웨어 할당 알고리듬은 연결구조의 최소화와 연결구조의 비용을 고려한 레지스터 및 멀티플렉서 입력 수를 최소화 할 수 있었다. 또한 실제 하드웨어를 타킷 칩으로 하여 비용 함수를 설정함으로서 단순히 연결구조의 입력 수 보다는 실제 구현되는 하드웨어의 단가를 줄이고자 하였다.

앞으로의 연구 과제로는 계층 설계에 대한 지원과 RAM이나 FIFO 같은 메모리 소자에 대한 합성 방법의 개발이다. 특히 논리 합성 툴 들이 사용하고 있는 하드웨어 모델링 기법을 적용함으로서 사용자가 의도한 회로를 쉽게 설계할 수 있도록 하는 연구가 더 진행되어야 한다.

참고문헌

[1] A. R. Newton, "Computer-aided design of VLSI circuits," *Proc. IEEE*, vol. 69, no. 10, pp. 1189-1199, Oct. 1981.

[2] A. R. Newton and A. L. S. Vincentelli, "Computer-aided design for VLSI circuits," *IEEE Computer*, pp. 38-63, April 1986.

[3] F. J. Kurdahi and A. C. Parker, "REAL: A program for register allocation," *Proc. 24th ACM/IEEE Design Automation Conf.*, pp. 234-341, June 1987.

[4] C. J. Tseng and D. Siewiorek, "Facet: A procedure for the automated synthesis of digital system," *Proc. 20th DAC*, pp. 490-496, June 1983.

[5] S. G. Shiva, "Automatic hardware synthesis," *Proc. IEEE*, vol. 71, no. 1, pp. 76-87, Jan. 1983.

[6] D. Gajski, *Silicon Compilation*, Addison-Wesley, 1988.

[7] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, vol. 78, no. 2, pp.301-318, Feb. 1990.

[8] R. Camposano, "From behavior to structure: high-level synthesis," *IEEE Design & Test of Computers*, pp. 8-19, Oct. 1990.

저자소개

김영숙(Kim, Young-suk)
1977년 : 고려대학교 이학사
2002년 : 충주대학교 공학석사(전자계산학 전공)
2002년 ~ 현재 : 세명대학교 대학원
전산정보학과 박사과정
수료(CAD 전공)
2006년 ~ 현재 : 극동대학교 컴퓨터정보표준학부 겸임교수
<주관심분야 : SOC CAD, CAD 알고리즘, Web 코스웨어 개발>

인치호(Lin, Chi-Ho)
1985년 : 한양대학교 공과대학 전자공학과 공학사
1987년 : 한양대학교 대학원 공학석사(CAD 전공)
1996년 : 한양대학교 대학원 공학박사(CAD 전공)
1992년 ~ 현재 : 세명대학교 컴퓨터학과 교수
<주관심분야 : SOC CAD, ASIC 설계, CAD 알고리즘, SOC 설계, RTOS 및 내장형 시스템>