論文99-36C-3-1

파이프라인 시스템의 최적화를 위한 설계 변환

(Design Transformation for the Optimization of Pipelined Systems)

權 成 勳 *, 金 衷 希 *, 申 鉉 哲 *

(Sunghoon Kwon, Chunghee Kim, and Hyunchul Shin)

요 약

본 연구에서는 파이프라인 구조를 갖는 시스템의 효율적인 설계를 위하여 변환을 이용한 설계 최적화 기술을 개발하였다. 변환 최적화 기술은 파이프라인 구조로의 변환과 retiming을 이용한 변환을 포함한다. 새로운 변환 방법은 다음의 세 가지 특징을 갖는다. 첫째, 여러 개의 파이프라인 블록을 동시에 고려하여 retiming 등의 변환을 수행함으로써, 파이프라인 구조 시스템의 전체 성능을 최적화한다. 둘째, 시스템의 면적과 수행시간 간의 trade-off를 가능하도록 하여, 회로 설계자가 다양한 설계의 대안을 찾고자 할 때 실용적인 도움을 준다. 셋째, 본 방법은 새로운 변환 및 알고리즘 개발 등의 문제로 쉽게 확장 가능하고, 메모리 또는 버스 등을 고려한 최적화 문제에도 사용될 수 있다. DSP 예제들에 대하여 실험한 결과, 평균적으로 면적은 21%, 성능은 17% 개선되었다. 특히, 본 기술은 여러 설계 대안의 효율적인 탐색에 유용하다.

Abstract

In this research, transformation-based optimization techniques for pipelined designs have been developed. The transformation-based optimization techniques include pipelined architecture transformations and retiming transformations. The new transformation method has the following three features. First, the overall performance of a pipelined system is optimized owing to various transformations including retiming of multiple pipelined blocks. Second, these techniques can be used to search a large solution space by allowing efficient exploration of trade-offs between area and performance. Third, these techniques can be easily extended to a new transformation or algorithm and can be used to optimize memory or bus architectures. Experimental results illustrate that these transformation-based optimization techniques improve area by 21% and performance by 17% on the average for a set of pipelined designs. Especially, the techniques are useful to efficiently explore a large design space.

I. 서 론

디지털 신호 처리 시스템의 수요 증가에 따라, 여러 제약 조건하에서의 효율적인 설계를 위한 상위 단계의 합성 툴에 대한 필요성이 점차 증가하고 있다. 상위 단계에서의 합성은 여러 과정을 거치고 많은 제약 조건들로 복잡도가 크게 증가하기 때문에 최적화하기가 쉽지 않다[1]. 따라서, 합성되는 회로의 성능 개선을 위해서 설계 변환에 의한 최적화 방법이 자주 사용된다.

현재까지 상위 단계에서 몇 가지 변환에 의한 설계 최적화 방법들이 제안되었다. 먼저, CDFG에서 하드웨어 연산기의 사용율을 최대로 하여 회로의 면적을 최

* 正會員, 漢陽大學校 電子工學科

(Dept. of Electronics Engineering, Hanyang University)

※ 본 연구는 서울대학교 반도체공동연구소의 교육부 반도체분야 학술연구조성비(과제번호 : ISRC 97-E-2040)에 의해 수행되었음.

接受日字:1998年8月20日, 수정완료일:1999年2月18日

소화하는 방법들이 제안되었다[2, 3, 4]. 이러한 방법들은 주어진 시간에 대한 제약조건하에서 CDFG 형태로 변환하여 스케쥴한 후에 각 리소스의 사용율을 증가시켰다. [5, 6, 7] 등에서는 여러 가지 변환 기법들을 사용하여 소프트웨어나 상위 단계에서의 동작기술에서 메모리 최적화 및 코드 최적화 등을 수행하는 방법을 제안하였다. [8] 에서는 리소스의 사용율을 최대로 하는 변환 방법을 이용하여, 소모전력의 최적화도 고려하였다. [9] 에서는 특히 컨트롤 부분이 많은 부분을 차지하는 상위 단계의 동작기술에서 throughput과 소모전력을 최적화하기 위한 변환 기법들을 제안하였다. [10] 방법에서는 순차적으로 수행되는 시스템의 설계 최적화를 위한 변환 알고리즘을 제안하였다. 교환, 결합, 배분 법칙을 포함하는 변환과 모듈의 변환, 스케줄된 결과 자체의 변환 방법 등을 사용하였다.

본 연구에서는 실시간 파이프라인 구조를 가지는 시스템의 효율적인 설계를 위한 상위 단계에서의 설계 최적화 기술을 개발하였다. 파이프라인 구조를 가지는 시스템을 CDFG로 표현한 후, 계층적인 스케줄 및 retiming 등의 변환이 가능하도록 하였다. 본 연구에서는 여러 개의 파이프라인 블록내에서 동시에 최적화를 고려한 변환을 수행하는 점이 특징적이다. 또한, 설계하고자 하는 시스템의 면적과 수행시간 간의 trade-off가 가능하기 때문에, 회로 설계자는 여러 가지의 가능한 설계 대안 중에서 원하는 설계 결과를 얻을 수 있다. 제안한 변환 방법은 면적과 지연시간 이외에도 다중 포트 메모리 등을 고려한 최적화 문제에도 쉽게 확장 가능하다.

본 논문은 다음과 같이 구성되어 있다. Ⅱ 장에서는 파이프라인 블록 구조를 가지는 시스템의 CDFG 표현 방법과 시스템의 스케줄 방법을 기술하였고, Ⅲ 장에서는 새로운 변환 방법과 이들을 조합한 최적화 방법에 대하여 기술하였다. 그리고, Ⅳ 장에서는 실험 결과를 기술하였고, Ⅴ 장에서 결론을 맺는다.

Ⅱ. 파이프라인 블록 구조로의 표현

대부분의 상위 단계에서의 합성 시스템들은 C나 VHDL, Silage 등의 동작기술 (behavioral description)을 입력받아 계층적인 CDFG로 표현한 후에 이를 이용하여 합성한다. CDFG는 상위 단계의 동작기술을 방향성이 있는 그래프 (V, E)로 표현한 것이다. 여기서, V는 오퍼레이션 노드의 집합이고, E는 노드들을 연결하는 컨트롤 및 데이터 의존성을 나타내기 위한 에지의 집합이다. 계층적인 CDFG는 여러 개의 노드로 구성된 부분 그래프 (sub-graph)들의 집합으로 이루어진다. 예를 들면, 하나의 함수는 다른 여러 개의 부분함수를 부를 수 있으며, 조건문이나 루프 등을 기술하는 경우에도 여러 개의 부분 함수로 표현하는 것이 일반적이다.

본 논문에서는 파이프라인 구조를 가지는 시스템을 표현하기 위하여 기존의 CDFG 표현 방법을 개선하였다. 먼저, 파이프라인 시스템에서 사용되는 기본적인 용어들을 정리한다.

- 파이프라인 유닛 (unit) : 여러 개의 파이프라인 단계 (stage) 들로 구성된 회로 전체 블럭
- 파이프라인 단계 또는 블록 (block) : 전체 계산 중에 일부분의 계산을 수행하는 블록
- 파이프라인 initiation interval (latency) : 연속적인 파이프라인 유닛 사이의 초기 시간 간격
- 계산시간 (computation time) 또는 지연시간 (turn around time/clock cycle) : 입력된 데이터의 계산 및 처리를 완료하는데 걸리는 시간

본 논문에서 제안한 변환 방법은 회로 전체에서 파이프라인 블록 구조로의 변환과 retiming 등을 이용한 파이프라인 변환으로 이루어진다. 각 파이프라인 블록 구조는 파이프라인 단계를 구분하는 레지스터의 출력 및 외부 입력에서부터 시작하여, 레지스터의 입력 및 외부 출력까지를 의미한다. 본 논문에서는 여러 개의 파이프라인 단계를 동시에 스케줄하여, 서로 다른 파이프라인 단계간의 하드웨어 공유가 용이하도록 하였다.

순차적인 회로를 파이프라인 블록 구조의 시스템으로 변환하는 방법에 대하여 설명한다. 순차적인 구조에서 입력된 데이터가 처리되어 출력되기까지 l 개의 컨트롤 스텝 수 (지연시간) 를 가진다고 가정하자. 이를 단계 수가 p인 파이프라인 구조로 변환하면 하나의 파이프라인 단계는 $\lceil l/p \rceil$ 개의 컨트롤 스텝 수를 가지게 된다. 그러므로, $\lceil l/p \rceil$ 개의 컨트롤 스텝 수를 주기로 데이터가 입력되고 출력된다. 본 파이프라인 구조로의 변환에서 파이프라인 단계 수는 사용자가 원하는 대로 분할하도록 하였다. 그러므로, 사용자는 여러 가지의 파이프라인 단계에 대하여 실험하여 가장

좋은 파이프라인 단계를 선택할 수 있다. 파이프라인 구조로의 변환 후에 retiming 등의 여러 가지 변환을 수행하면 더욱 최적화 된 설계 결과를 얻을 수 있다.

Ⅲ. 설계 변환과 Trade-off 방법

설계 최적화를 위해서는 주어진 설계를 정확히 평가하기 위한 비용함수의 선택이 중요하다. 본 연구에서 사용한 비용함수에 대하여 알아보고, 기본적인 변환 및 retiming 에 의한 변환 방법에 대하여 기술한다.

1. 비용 함수

상위 단계의 설계 최적화 문제는 수행시간의 제약조건 하에서 하드웨어 면적을 최소화하거나, 하드웨어의 제약조건하에서 수행시간을 최소화하는 것으로 나눌 수 있다. 수행시간은 사용된 최대 컨트롤 스텝 수 (지연시간) 로 나타낼 수 있으며, 면적은 사용된 각각의 리소스 (resource)들의 면적의 합으로 나타낼 수 있다. 정확한 수행시간과 면적은 합성이 끝난 후에 알 수 있으므로, 대부분 이와 같이 추정하여 계산한다.

본 방법에서 사용한 비용함수의 계산식은 다음의 식 (1) 및 (2)와 같다.

$$cost_1 = \begin{cases} c\_step + w_1 \times \#regs + \\ w_2 \times \#readports + w_3 \times \#writeports, & \text{if for } each\ resource\ s \in S,\ n_s \le max\_num_s \\ \infty, & otherwise \end{cases} \tag{1}$$

$$cost_2 = \begin{cases} \sum_{s \in S} size_s n_s + size_{reg} \#regs, & \text{if } c\_step \le max\_step \\ \infty, & otherwise \end{cases} \tag{2}$$

식 (1)은 주어진 리소스를 이용하여 컨트롤 스텝 수를 최소화하기 위한 비용식이고, 식 (2)는 주어진 수행시간의 제약조건 하에서 리소스의 면적을 최소화하기 위한 비용식이다. 식에서 S는 리소스 형태들의 집합이고, $size_s$, n_s, max_num_s는 각각 형태 s∈S인 리소스의 면적, 사용된 리소스의 수, 최대 리소스의 수이다. 변수들을 저장하는 데에 사용되는 레지스터의 수와 면적은 각각 #regs, $size_{reg}$로 표시되었다. c_step은 컨트롤 스텝 수이며, max_step은 제약 조건으로 주어진 컨트롤 스텝 수이다. 각 컨트롤 스텝에서의 변수들은 레지스터 외에도 다중포트 메모리에 저장 가능하다. #readports(#writeports)는 다중포트 메모리에서 동시에 읽을 수 있는 포트의 수로 전체 컨트롤 스텝에서의 입력(출력) 에지수의 최대값을 나타낸다.

메모리는 전체 하드웨어의 상당한 부분을 차지할 수 있기 때문에, 메모리 비용을 고려하여 최적화할 필요가 있다. 각 컨트롤 스텝에서 출력되는 변수들을 저장하기 위해서는 독립된 레지스터들 외에도 다수의 포트에서 데이터를 엑세스할 수 있는 다중포트 메모리가 사용될 수 있다. 레지스터 그룹들로 구성될 수 있는 다중포트 메모리를 사용하면 포트들의 공유가 가능하기 때문에 연결선(버스)이나 멀티플렉서 등의 회로소자 수가 감소된다. 다중포트 메모리의 입·출력 포트 수를 줄이면 다중포트 메모리의 크기도 줄어들 수 있다. [11] 등에서의 메모리의 실제 레이아웃 크기를 참고로하여, 다중포트 메모리의 크기 (비용)를 실험적으로 다음과 같이 구하였다.

$$memory\_cost = \sum_{i=1}^{K} (regs_i)^{\eta}(1 + \omega(ports_i - 1)) \tag{3}$$

여기서, K는 다중포트 메모리 모듈의 수를, $ports_i$는 i번째 메모리 모듈의 포트 수를, $regs_i$는 i번째 다중포트 메모리 모듈 내에 있는 레지스터들의 수를 나타낸다. η는 0.68이고 ω는 0.48이며, 이들 값은 실험적으로 결정하였다.

식 (3)은 메모리의 포트 수가 증가하면 메모리의 크기가 커짐을 보여 준다. 본 논문에서는 포트 수를 최소화하기 위하여, 식 (1)에 #readports(#writeports)로 계산되는 항을 사용하였다. 다중포트 메모리를 사용할 경우에는, 식 (1)에서의 가중치 w_1 은 0을 (레지스터를 사용하지 않기 때문), w_2, w_3 은 일정 상수값 (현재는 10)을 사용한다.

본 논문에서 제안하는 변환은 개선하고자하는 목적에 따라 비용함수를 적절하게 조절함으로써, 메모리와 저전력 뿐만 아니라 최적화를 위한 여러 응용 분야에도 적용될 수 있다.

2. 기본적인 변환 방법

파이프라인 구조 시스템의 최적화를 위한 변환 기술은 [10] 에서 사용한 오퍼레이션 (operation)의 변환, 모듈 (module)의 변환 그리고 스케줄의 변환 방법 등을 확장하여 구현하였으므로 이를 간단히 기술한다.

오퍼레이션의 변환은 연산들을 수학적으로 등가인 다른 형태로 변환하는 것이다. 즉, 오퍼레이션의 변환에 의하여 오퍼레이션들의 순서, 사용되는 오퍼레이션의 종류, 또는 사용되는 오퍼레이션의 수가 변경될 수 있기 때문에, 많은 개선이 가능하다.

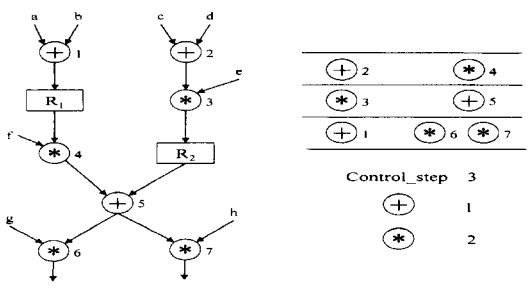
모듈의 변환은 각 오퍼레이션 노드 또는 여러 개의 오퍼레이션 노드를 하나 이상의 하드웨어 모듈을 사용하여 구현하도록 변환하는 것이다. 그러므로, 다양한 하드웨어 모듈을 탐색하여 최적의 모듈을 선택하는 것이 중요하다. 변환에서는 하나의 모듈 또는 여러 개의 모듈들을 같은 기능을 수행하는 다른 하나의 모듈이나 모듈들로 변환할 수 있는지를 점검하고, 변환하는 것이 유용한 경우에는 변환한다.

스케줄의 변환은 각 오퍼레이션이 수행되는 컨트롤 스텝을 임의로 변경시켜서 설계의 최적화를 수행하는 것이다. 컨트롤 스텝을 변경시키면 필요한 리소스의 수를 감소시켜 면적을 줄일 수 있고, 에지의 수가 최대인 컨트롤 스텝에서 에지의 수를 감소시킬 수가 있어서 메모리의 비용을 줄이는데 효과적일 수 있다.

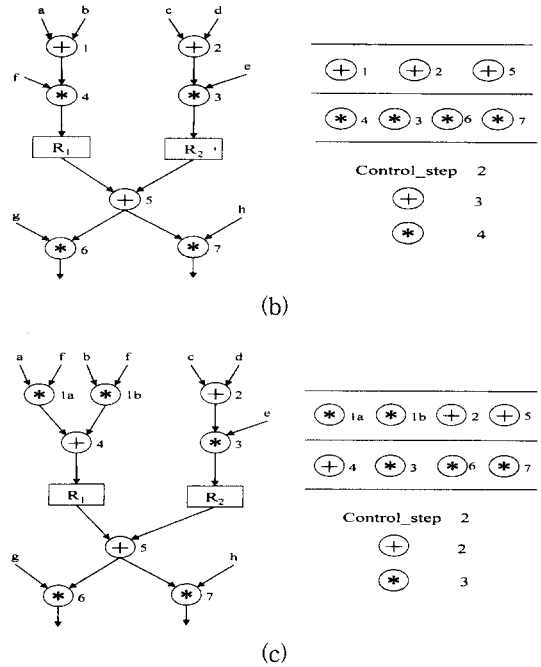
3. Retiming을 이용한 변환

Retiming에 의한 파이프라인 단계의 변화는 오퍼레이터의 공유에 의한 면적의 최소화 및 컨트롤 스텝의 감소에 의한 성능 향상에 유용할 수 있다. 또한, retiming을 수행하면 다른 변환을 적용시킬 수 있는 경우가 발생하므로, 다른 변환 방법들과 병행하여 수행하면 더 많은 최적화를 수행할 수 있다는 장점이 있다. 본 retiming을 이용한 변환에서는 각각의 파이프라인을 구분하는 레지스터에 대하여 retiming을 수행하였다. Retiming을 수행한 후에 스케줄을 다시 수행하고 결과가 개선되는 경우에만 적용한 retiming을 선택하였다.

그림 1은 본 retiming에 의한 변환의 유용성을 보여준다. 그림 1의 (a)는 변환이 수행되기 전의 CDFG로 2개의 파이프라인 단계를 가진다. 서로 다른 컨트롤 스텝에서는 다른 파이프라인 단계의 오퍼레이터들의 공유가 가능하다. 스케줄 결과 하나의 덧셈기와 두 개의 곱셈기를 사용하여 3개의 컨트롤 스텝 내에서 수행 가능하였다.



(a)



(b)

(c)

그림 1. Retiming과 오퍼레이션 변환의 수행 예
(a) Retiming을 수행하기 전의 CDFG와 스케줄 결과 (b) Retiming을 수행한 후의 CDFG와 스케줄 결과 (c) 오퍼레이션 변환을 수행한 후의 CDFG와 스케줄 결과

Fig. 1. An Example of retiming and operation transformation.
(a) A CDFG and schedule result before retiming (b) A CDFG and schedule result after retiming (c) A CDFG and schedule result after operation transformation

그림 1의 (b)는 컨트롤 스텝 수를 줄이기 위하여 retiming을 수행한 결과이다. 파이프라인 레지스터 R1이 그림에서와 같이 아래쪽으로 이동하였다.

Retiming을 수행한 후의 스케줄 결과를 보면 컨트롤 스텝의 수는 2로 감소하였지만, 3개의 덧셈기와 4개의 곱셈기가 필요하여 retiming을 수행하기 전보다 하드웨어의 면적이 증가한 것을 알 수 있다. 그림 1의 (c)는 retiming 후의 결과에서 오퍼레이션의 변환까지 수행한 결과이다. CDFG에서 (a+b)*f 부분이 (a*f)+(b*f)로 변환되었다. 변환 후에도 역시 2개의 컨트롤 스텝이 필요하지만, 2개의 덧셈기와 3개의 곱셈기가 필요하여 각각 하나의 덧셈기와 곱셈기가 감소한 것을 알 수 있다. 여기에서 주목해야 할 점은 곱셈의 수를 줄이기 위하여 (a+b)*f 의 표현을 일반적으로 선호하지만, 경우에 따라서는 (a*f)+(b*f)의 표현

을 이용하면 더욱 최적의 결과를 얻는다는 점이다. 이와 같이 retiming과 여러 가지 다른 변환을 함께 사용하면 설계자는 편리하게 많은 설계 대안을 자동으로 탐색하여 설계를 최적화할 수 있다.

4. 변환에 의한 최적화 알고리즘

본 논문에서는 오퍼레이션, 모듈, 스케줄 자체의 변환 등의 기존의 변환 방법들과 새로 개발한 파이프라인 블록 구조로의 변환 방법 및 retiming 변환 방법을 병행하여 설계를 최적화 하였다. 본 알고리즘에서 파이프라인 블록 구조로의 변환은 사용자가 원하는 경우에 주어진 단계의 파이프라인 블록 구조로 자동 변환한다. 본 논문에서 제안하는 변환에 의한 최적화 알고리즘은 다음과 같다.

Algorithm : Optimization by transformation

```
{
    input hierarchical CDFG;
    schedule;
    if(Pipeline_Transform==TRUE)      /* User option */
        make scheduled CDFG for pipelined structure;
    while(TRUE) {
        for(each register) {              /* Retiming */
            retiming; undo if the cost is increased;
        }
        for(each replaceable module) { /* Module trans. */
            module transform; undo if the cost is
            increased;
        }
        for(each transformable set of operations) { /*
        Operation transformation */
            operation transform; undo if the cost is
            increased;
        }
        for(each operation) {   /* Schedule transf. */
            change control step of the operation;
            reschedule affected nodes; undo if the cost is
            increased;
        }
        if(the cost has not been reduced in this iteration)
        break;
    }
}
```

설계 최적화는 스케줄된 CDFG를 입력받아서 주어진 리소스들을 이용하여 수행시간이 최소화되도록 수행하거나, 주어진 수행시간 조건하에서 리소스들의 면적이 최소화되도록 수행한다. 수행시간 최소화를 위한 변환은 주어진 리소스를 이용하여 컨트롤 스텝 수가 최소가 되도록 한다. 즉, 비용이 감소하지 않을 때까지 retiming, 오퍼레이션의 변환, 모듈의 변환, 스케줄 자체의 변환을 반복적으로 수행한다.

주어진 수행시간 제약조건 하에서 리소스의 면적을 최소화할 경우에는 (2)식의 비용함수 값을 최소화해야 한다. 이를 위하여, (1)식의 비용 최소화를 위한 변환 알고리즘을 리소스를 줄이면서 반복적으로 수행하였다.

5. 면적과 성능간의 trade-off 방법

본 연구에서는 성능과 면적간의 다양한 trade-off를 수행하여 여러 가지 해의 집합을 구할 수 있도록 하였다. 먼저 주어진 리소스를 이용하여 최소의 컨트롤 스텝 수를 가지도록 스케줄한 후에 사용율이 가장 큰 리소스부터 하나씩 증가시키면서 최소의 컨트롤 스텝 수를 가지도록 변환하는 것을 반복하여 수행하였다. 만약 사용율이 가장 큰 리소스를 증가시킨 경우에 컨트롤 스텝수가 감소되지 않은 경우에는, 그 다음으로 사용율이 큰 리소스를 증가시키고 반복하여 수행한다. 더 이상 증가시킬 리소스가 존재하지 않는 경우까지 이와 같은 과정을 반복하여 면적과 컨트롤 스텝간의 다양한 trade-off 결과를 얻는다. 설계자가 직접 여러 변수를 변경하면서 설계를 해 보는 것은 많은 시간을 필요로 하며 지루한 일이다. 따라서 이러한 다양한 설계 탐색의 자동화는 유용하다.

Ⅳ. 실험 결과

본 연구에서 제안한 변환 알고리즘은 DEC3000 워크스테이션에서 "C" 언어로 구현되었으며, 변환 실험을 위하여, HYPER 시스템 [8, 12] 에서 사용한 DSP 예제들을 이용하였다. HYPER 는 Silage 언어로 표현된 회로를 CDFG로 바꾸고, 여러 단계에 걸쳐 최적화 과정을 수행한다. 본 논문에서는 HYPER에서 출력된 CDFG를 읽어서 파이프라인 최적화를 위한 변환에 사용하였지만, 라이브러리에 대한 정보 등의 부족으로 HYPER 시스템과의 직접적인 비교는 가능하지 않았다.

표 1은 retiming을 포함한 변환 알고리즘을 이용하여 면적 최적화를 수행한 실험 결과이다.

표 1에는 각 예제에 대하여 CDFG의 노드 수, 변환 전과 후의 면적에 대한 비용, 컨트롤 스텝 수, 다중포트 메모리의 포트수 그리고 수행시간이 기술되어 있다. 면적은 사용하는 라이브러리에 따라서 차이가 있으므로, 본 실험에서는 덧셈기 및 뺄셈기는 10, 곱셈기는 80 등으로 가정하여 사용하였다. 최적화 변환 후의 실험 결과가 평균적으로 21% 정도 면적이 감소된 것을 알 수 있다. 또한, 평균적으로 17% 정도의 컨트롤 스텝 수가 줄어들었는데, 이는 면적이 더 이상

감소하지 않는 경우에는 컨트롤 스텝 최적화를 수행하였기 때문이다. 특히 retiming을 이용하면 파이프라인 구조가 바뀌어 컨트롤 스텝 최적화에 효과적이다. 다중포트 메모리의 포트 수 최소화를 고려한 실험에서는 읽기 포트와 쓰기 포트의 수가 각각 3 개씩 줄어들었다. 결과적으로, 최적화를 고려하지 않은 실험 결과에 비하여 면적, 컨트롤 스텝 및 메모리에서의 포트수가 모두 개선되었음을 알 수 있다.

표 1. 면적 최적화 실험 결과
Table 1. Area optimization results.

Examples	#Oper. Nodes	Area cost		#Control steps		Multiport memory considerations (#readports, #writeports)		CPU Time (sec)
		Before opt.	After opt.	Before opt.	After opt.	Before opt.	After opt.	
cascade	52	218	209	18	16	(7, 4)	(7, 4)	1
cf	62	213	213	25	23	(4, 3)	(4, 3)	1
cordic	40	162	151	14	11	(6, 4)	(5, 3)	5
dct	75	261	237	27	18	(6, 5)	(6, 5)	4
decby4	84	229	159	26	25	(9, 5)	(8, 4)	2
df	58	389	384	13	11	(9, 8)	(8, 7)	1
filter	70	431	236	21	20	(7, 4)	(7, 4)	5
fir	101	307	163	26	22	(3, 2)	(3, 2)	26
gm	76	238	147	53	40	(4, 2)	(4, 2)	2
iir	43	209	133	15	10	(8, 4)	(8, 4)	1
lattice	34	295	214	17	11	(6, 4)	(6, 4)	1
nc	108	623	514	19	18	(15, 9)	(15, 9)	7
parallel	62	315	302	17	16	(8, 6)	(8, 6)	1
volterra	50	313	230	14	11	(7, 6)	(7, 6)	1
wavelet	70	335	329	18	15	(7, 6)	(7, 6)	2
wf	71	245	147	35	31	(5, 3)	(5, 3)	1
Total (%)		4783 (100%)	3768 (78.8%)	358 (100%)	298 (83.2%)	(111, 75)	(108, 72)	

표 2는 DSP 예제를 사용한 면적과 컨트롤 스텝간의 trade-off 결과를 나타낸다. 표에는 초기 스케줄 후의 컨트롤 스텝수 및 면적과, 반복적으로 면적을 증가시키면서 최적화를 수행하였을 경우의 면적과 컨트롤 스텝수를 보여준다. 대부분의 경우에 어느 정도까지는 면적을 증가시키면 컨트롤 스텝 수가 줄어든다. 각 예제에 대하여 더 이상 컨트롤 스텝이 증가하지 않는 경우에는 면적 최적화를 수행하였는데, 몇 개의 예제에서 면적이 줄어든 것을 알 수 있다. 표 2의 실험 결과는 본 설계 변환 기술을 이용하면 다양한 설계 대안을 얻을 수 있다는 것을 보여 준다.

표 2. 면적과 컨트롤스텝 간의 trade-off 실험 결과
Table 2. Area and control step trade-off results.

Examples		Initial	After optimization (The number of control steps is minimized for the given area)							
cascade	control step	30	26	17	14					
	area	123	127	207	215					
cf	control step	41	37	21						
	area	141	141	213						
dct	control step	39	33	19						
	area	163	161	235						
decby4	control step	38	36	26	20	19	18			
	area	143	143	157	239	243	245			
df	control step	36	32	16						
	area	129	131	211						
filter	control step	41	38	20						
	area	156	156	234						
fir	control step	37	33	26	21	15	12	11		
	area	186	188	189	267	276	342	326		
gm	control step	55	45	39	36	32	31	29	27	20
	area	143	231	239	242	334	417	495	497	584
iir	control step	15	10							
	area	129	131							
lattice	control step	22	18	10						
	area	132	132	212						
znc	control step	68	64	33						
	area	185	203	275						
parallel	control step	40	36	18						
	area	135	139	223						
wavelet	control step	32	28	19	14					
	area	139	163	239	251					
wf	control step	39	34	32	30					
	area	145	147	227	237					

V. 결 론

본 연구에서는 파이프라인 구조를 가지며 실시간으로 동작하는 시스템의 설계 최적화를 위한 효율적인 변환 기술과 설계된 회로의 성능 trade-off를 위한 기술을 개발하였다. 변환 최적화를 위하여 오퍼레이션, 모듈 및 스케줄 변환 등의 기존의 변환 방법들과, 새로 제안한 파이프라인 구조로의 변환 및 retiming을 이용한 변환 알고리즘을 조합하여 사용하였다. 특히, 본 논문에서는 여러 개의 파이프라인 블록을 동시에 고려하여 retiming 등의 변환을 수행하였고, 시스템의

면적과 수행시간 간의 trade-off를 가능하도록 하였으며, 메모리 등을 고려하여 최적화 문제로 확장 가능하게 한 점이 특징이다. 실제의 DSP 예제 등을 이용하여 변환 실험한 결과, 수행시간과 면적이 평균적으로 각각 21% 및 17% 감소하였으며 메모리의 크기도 줄어들었다. 또한, 본 변환 방법은 여러 가지 다양한 설계를 고려하는 등의 설계 최적화에 유용하게 사용될 수 있다.

참 고 문 헌

[1] C. Mandal, P. Chakrabarti and S. Ghose, "Design Space Exploration for Data Path Synthesis," Proc. of VLSI Design'97, pp. 166-171, 1997.

[2] M. Potkonjak and J. Rabaey, "Optimizing Resource Utilization Using Transformations," IEEE Transactions on CAD of ICAS, vol. 13, no 3, pp. 277-292, Mar. 1994.

[3] S. H. Huang and J. M. Rabaey, "TAO: A Transformation Framework for DSP Algorithm Optimization," Tech. Report UCB/ERL M95/89, Aug. 1995.

[4] Inki Hong, Darko Kirovski, and Miodrag Potkonjak, "Potential-Driven Statistical Ordering of Transformations", Proc. of Design Automation Conference, pp. 347-352, 1997.

[5] W. K. Cheng and Y. L. Lin, "A Transformation-Based Approach for Storage Optimization", Proc. of ACM/IEEE Design Automation Conference, pp. 158-163, 1995.

[6] J. Li and R. K. Gupta, "HDL Optimization Using Timed Decision Tables", Proc. of ACM/IEEE Design Automation Conference, pp. 51-54, 1996.

[7] D. J. Kolson, A. Nicolau and N. Dutt, "Elimination of Redundant Memory Traffic in High-Level Synthesis," IEEE Transactions on CAD of ICAS, vol. 15, no 11, pp. 1354-1364, Nov. 1996.

[8] A. P. Chandrakasan, M. Potkonjak, J. Rabaey and R. W. Brodersen, "HYPER-LP: A System for Power Minimization Using Architectural Transformations," Proc. of the International Conference on CAD, pp. 300-303, 1992.

[9] G. Lakshminarayana and N. K. Jha, "FACT: A Framework for the Application of Throughput and Power Optimizing Transformations to Control-flow Intensive Behavioral Descriptions," Proc. of ACM/IEEE Design Automation Conference, Section 6.1, 1998.

[10] 김 충희, 신 현철, "변환을 이용한 상위 단계에서의 VLSI 설계 최적화 기술," 한국정보과학회 논문지(A), vol. 24, no. 12, pp. 1375-1382, 1997

[11] K. Nii, H. Maeno, T. Osawa, S. Iwade, S. Kayano, and H. Shibata, "A Novel Memory Cell for Multiport RAM on 0.5 *um* CMOS Sea-of-Gats," IEEE Journal of Solid-State Circuits, vol. 30, no. 3, pp. 316-320, March 1995.

[12] J. M. Rabaey, C. Chu, P. Hoang and M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures," IEEE Design and Test of Computers, vol. 7, no. 5, pp. 40-51, Oct. 1990.

저 자 소 개

權 成 勳(正會員) 第 34卷 C編 第 11號 參照
1998년 9월 ~ 현재 한양대학교 공학기술연구소에서 박사후 과정

申 鉉 哲(正會員) 第 34卷 C編 第 11號 參照
현재 한양대학교 전자공학과 교수

金 泰 希(正會員) 第 24卷 A編 第 6號 參照
1998년 9월 ~ 현재 U. C. Berkeley에서 박사후 과정