論文 97-22-3-6

효율적인 하드웨어 공유를 위한 단어길이 최적화 알고리듬

正會員 崔 正 一***, 田 弘 信**, 李 廷 周*, 金 汶 秀*, 黃 善 泳*

A Bitwidth Optimization Algorithm for Efficient Hardware Sharing

Jeong-Il Choi***, Hong-Shin Jun**, Jung-Joo Lee*, Moon-Soo Kim*,
Sun-Young Hwang* *Regular Members*

요 약

본 논문은 고정소수점 하드웨어 합성 과정에서 효율적인 하드웨어 공유를 위하여 연구된 단어길이 최적화 알고리듬에 대해 기술하고 실험 결과를 제시한다. 제안된 알고리듬은 가능한 작은 칩면적으로 하드웨어를 구현하기 위하여 단어길이 최적화 과정을 통해 DSP 알고리듬의 내부 신호를 최적화된 고정소수점 형식으로 표현하고, 각 연산의 입력신호 단어길이를 결정한다. 전체 연산기의 면적을 줄이기 위하여 상위 수준 합성 전에 각 연산들을 특정 입력신호 단어길이를 가지는 연산기를 공유 가능한 연산들의 그룹들로 분할하며, 상위 수준 합성을 통하여 레지스터 전송 수준의 고정소수점 데이타패스를 생성한다.

제안된 알고리듬은 DSP 하드웨어 설계 자동화 시스템 SODAS-DSP에서 구현되었다. SODAS-DSP는 스케메틱으로 모델링된 DSP 알고리듬으로부터 원하는 시스템의 동작 속도등의 설계 제약을 만족하면서 최적화된 면적의 고정소수점 데이타패스와 컨트롤러를 자동 생성한다. 실험 결과, 내부 신호를 고정소수점으로 전환하고 동일한 입력신호 단어길이를 공유하는 연산들로 분할하여 합성함으로써 원하는 시스템의 성능을 만족하면서 최적화된 면적으로 고정소수점 하드웨어를 생성한다.

ABSTRACT

This paper presents a bitwidth optimization algorithm for efficient hardware sharing in digital signal processing system. The proposed algorithm determines the fixed-point representation for each signal through bitwidth optimization to generate the hardware requiring less area. To reduce the operator area, the algorithm partitions the ab-

*서강대학교 전자공학과
**삼성전자 System LSI 본부
***삼성전관(주)
論文番號:96036-0201
接受日字:1996年 2月 1日

stract operations in the design description into several groups, such that the operations in the same group can share an operator. The partitioning results are fed to a high-level synthesis system to generate the pipelined fixed-point datapaths.

The proposed algorithm has been implemented in SODAS-DSP an automatic synthesis system for fixed-point DSP hardware. Accepting the models of DSP algorithms in schematics, the system automatically generates the fixed-point datapath and controller satisfying the design constraints in area, speed, and SNR(Signal-to-Noise Ratio). Experimental results show that the efficiency of the proposed algorithm by generates the area-efficient DSP hardwares satisfying performance constraints.

Ⅰ. 서 론

오늘날 DSP(Digital Signal Processing) 응용 범위가 확대되고 그 수요가 증대됨에 따라 다양한 DSP 하드웨어를 높은 수준에서 효율적으로 설계하기 위한 설계 자동화에 대한 연구가 활발이 진행되고 있다[1]. DSP 하드웨어 설계는 가능한 작은 칩면적에 고속의 동작 속도를 갖는 시스템을 구현하여야 하며 이 목적을 달성하기 위하여 내부 신호를 최적화된 고정소수점으로 표현하는 것이 요구된다. 범용 DSP 칩들은 내부 신호를 부동소수점으로 사용하여 표현하기 때문에 각 신호들이 비교적 정확한 값을 가질 수 있다는 장점을 가지고 있으나, 큰 면적의 하드웨어가 요구되고 계산 시간이 커지는 단점을 가지고 있다. 반면, 입력신호 처리시 내부 신호 값들의 변화가 크지 않은 특정 응용 영역에서는 고정소수점으로 표현이 가능하다. 시스템의 성능이 설계 사양을 만족하는 범위에서 고정소수점으로 구현하는 것이 부동소수점으로 구현하는 것보다 짧은 단어길이로 표현이 가능하여 하드웨어 낭비를 줄일 수 있는 장점을 지니고 있다. 따라서, 특정 응용 범위에 사용되는 DSP 칩의 경우에는 내부 신호들을 최적화된 고정소수점으로 전환하여 허용 오차내의 값을 유지하면서 하드웨어 비용을 줄이는 과정이 필요하다.

고정소수점으로 내부 신호를 최적화하여 이로부터 각 연산들의 입력신호 단어길이를 결정할 수 있으며, 내부 신호들의 처리에 있어 각기 다른 단어길이를 가지는 연산들이 많아질 수가 있다. 이 경우 상위 수준 합성시 연산기의 공유가 불가피한 경우에 각기 다른 입력신호 단어길이를 필요로 하는 연산들을 하나의 연산기에 할당 시켜야 한다. 상위 수준 합성 과정에서는 공유 가능한 연산들은 스케쥴링 과정에서 결정되고 실제 연산기에 대한 할당은 모듈할당 과정에서 수행되나 명목상의 지연시간과 하드웨어 비용을 가정한 가상 컴포넌트를 사용하여 수행하므로 각 연산의 입력신호 단어길이를 고려할 수 없기 때문에, 입력신호에 관계없이 불필요하게 큰 입력신호 단어길이를 가지는 연산기를 할당하게 되는 경우가 있다. 따라서, 상위 수준 합성 수행전에 가능한 작은 입력신호 단어길이를 가지는 연산기로 구현할 수 있도록 각 연산들을 동일한 입력신호 단어길이의 연산기를 공유할 수 있는 연산들의 집합들로 분할하여 주는 과정이 필요하다.

DSP 하드웨어 합성 시스템으로 Sehwa[2], HAL[3], SPAID[4], CATHEDRAL[5], SPW[6]등이 연구 발표되었다. CATHEDRAL 시스템은 아키텍춰 구동 합성방식을 가진 전형적인 시스템으로, CATHEDRAL-I과 CATHEDRAL-II는 각각 bit-serial, bit-parallel 구조를 지원하며[5], CATHEDRAL-III는 계층적인 제어를 갖으며 불규칙적이고 recursive 형태의 고속 알고리듬 합성에 적합하고, CATHEDRAL-IV는 규칙적인 어레이 구조를 타겟으로 하고 있다. 파이프라인 구조를 지원하는 CATHEDRAL 시스템은 ILP(Integer Linear Programming)[7]기법으로 스케쥴링을 수행하여 최적의 해를 보장하나 긴 계산 시간을 필요로 하여 큰 시스템을 설계하거나 상위 수준 합성기가 시스템 설계자에게 설계공간의 탐색으로 아이디어를 제공한다는 측면에서 보면 문제가 있으며, 특정 응용 영역에서 효율적인 고정소수점 하드웨어를 합성해내지 못하는 단점이 있다. SPW는 하드웨어 면적을 최소화하기 위해 VHDL을 이용한 고정소수점 변환 알고리듬을 사용하여 시스템 내부 신호들을 최적화된

단어길이로 전환하였으나, 전체 연산기의 면적을 최소화하기 위한 알고리듬을 구현하지 못하고 있다.

본 논문에서는 연산기의 입력신호 단어길이를 고려하여 효율적으로 연산기를 공유함으로써 연산기의 면적을 최소화하기 위한 단어길이 최적화 알고리듬을 제안한다. 제안된 알고리듬은 구현하고자하는 시스템의 내부신호 단어길이를 최적화된 고정소수점으로 전환하고, 각 연산의 입력신호 단어길이에 대한 정보를 이용하여 동일한 입력신호 단어길이의 연산기를 공유 가능한 연산들로 분할하여 상위 수준 합성을 통해 레지스터 전송수준의 데이타패스를 자동 생성한다.

본 논문의 구성은 다음과 같다. 2장에서는 SODAS-DSP 시스템의 개관을 소개하고, 3, 4장에서는 각각 본 시스템에서 구현한 단어길이 최적화, 연산의 분할 알고리듬을 설명한다. 5장에서는 4차 LMS 필터를 이용한 noise canceller를 이용한 실험 결과를 제시하고, 6장에서는 결론 및 추후 과제에 대해 기술한다.

Ⅱ. SODAS-DSP 시스템 개관

SODAS-DSP는 시스템 수준에서 DSP 하드웨어를 자동설계하기 위하여 개발되었다. 시스템의 전체 구성도는 그림 1에 보였다. 다양한 설계 영역의 탐색을 위하여 SODAS-DSP는 시스템 수준에서 주어진 알고리듬을 모델링할 수 있도록 스케메틱 에디터와 심볼 에디터를 제공한다. 스케메틱 에디터상에서 구현된 시스템은 VHDL 또는 C 시뮬레이션을 통하여 그 동작을 검증하고, 하드웨어 비용을 줄이기 위하여 내부의 각 신호들을 최적화된 고정소수점으로 전환하며, 각 연산의 입력신호 단어길이를 결정한다. 이 정보를 이용하여 상위 수준 합성 전에 가능한 적은 하드웨어 비용으로 구현할 수 있도록 각 연산들을 특정 입력신호 단어길이의 연산기를 공유할 수 있는 그룹들로 분할한다. 분할 결과로부터 상위 수준 합성기의 입력기술로 사용되는 VHDL과 SFGDL(Signal Flow Graph Description Language)[8]을 추출한다.

상위 수준 합성 과정은 각 연산의 수행시간을 결정하게 되는 스케쥴링 과정과 실제적인 레지스터 전송수준의 데이타패스를 생성하는 연산기 할당 과정으로 이루어진다. SODAS-DSP 시스템에서는 스케쥴링 과정에 HAL 시스템의 force-directed 스케쥴링 알고리듬[9]의 단점을 개선한 entropy-based 스케쥴링 알고리듬[8]과 variable DII 파이프라인 구조[10]를 지원하고, 연산기 할당 과정에서는 각 연산기의 면적, 속도 등을 고려한 목적 함수 similarity measure[11]를 정의하여 연산기의 공유를 수행함으로써 효율적인 할당과 전체적인 데이타패스의 연결 구조 비용을 최소화하는 방향으로 연산기 할당 과정을 수행한다.

SODAS-DSP 시스템은 자동 설계 과정의 제반 과정에서 라이브러리를 참조하여 다양한 설계 사양을 제공할 수 있도록 하였다. 라이브러리는 알고리듬을 구현하기 위한 레지스터 트랜스퍼 수준, 논리 수준의 컴포넌트들과 이들의 기술독립적인 어트리뷰트 정보를 ASCII중간 형태로 데이타베이스상에서 저장하고 있다[12]. 상위 수준 합성과정에서는 라이브러리와의

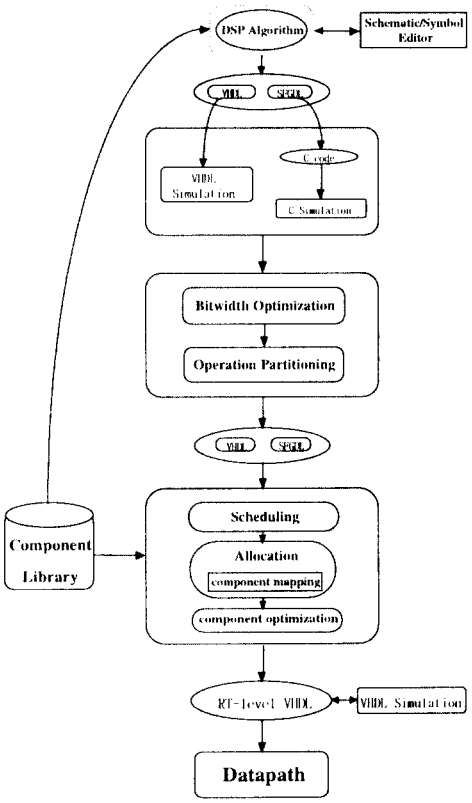


그림 1. SODAS-DSP의 합성 과정
Fig. 1 Synthesis flow in the SODAS-DSP.

인터페이스를 통한 컴포넌트 매핑과 최적화 과정에서 전체적인 합성결과를 유지하면서 최적의 비용으로 하드웨어를 구현할 수 있도록 지원하고 있다[12].

Ⅲ. 단어길이 최적화

SODAS-DSP 시스템은 하드웨어의 절감을 위해 VHDL을 이용한 고정소수점 변환 알고리듬을 보완하여 각 신호들을 특정 입력신호들에 대하여 시뮬레이션을 수행하면서 최적화된 단어길이를 결정한다. 내부 신호들의 단어길이는 시뮬레이션에 앞서 동일한 단어길이를 사용할 수 있는 내부 신호들로 구성된 몇 개의 그룹으로 나누는 신호의 분할 과정과 각 그룹별 정수부와 소수부의 단어길이를 결정하는 과정을 거쳐 최적화된다[6].

칩의 집적도가 커질수록 신호의 양은 많아지게 되고, 모든 신호에 대해 단어길이를 최적화 하는 것은 복잡한 계산을 요구한다. 따라서, 동일한 단어길이를 가질 수 있는 신호들을 하나의 그룹으로 묶어 그룹단위로 최적화 시킨다면 탐색 영역을 효과적으로 줄일 수 있는 장점이 있다[6]. 신호의 분할 과정에서는 입출력 신호가 동일한 단어길이를 가질 수 있는 컴포넌트의 입출력 신호들은 동일한 그룹으로, 다른 단어길이를 가지는 신호들은 다른 그룹으로 배치하는 방식을 사용한다. 예를 들어, 덧셈기나 뺄셈기는 입출력 신호의 단어길이가 동일하므로 같은 그룹으로 배치되나 단어길이가 다른 곱셈기에서의 입출력 신호는 다른 그룹으로 배치된다. 그러나 시스템에 궤환 경로가 존재할 경우 위와 같은 규칙에 위배될 수 있다. 가령, 입출력 신호의 단어길이가 다른 컴포넌트를 경유하여 궤환 경로가 생성되어 해당 컴포넌트의 출력값이 다시 그 컴포넌트의 입력이 될 경우, 해당 컴포넌트의 출력과 입력 사이의 경로에 단어길이를 변화시키는 컴포넌트가 존재하지 않는다면 입출력 신호는 동일한 그룹으로 배치될 수 있다.

단어길이의 결정은 정수부와 소수부 단어길이를 결정하는 두 과정으로 나뉘고, 정수부 단어길이는 각 신호의 절대값의 평균과 절대값의 표준편차를 이용하여 식 (1)과 같이 결정된다[6]. 여기서 M(|Sx|)는 신호들의 절대값들의 평균, $\sigma(|Sx|)$는 신호들의 절대값의 표준편차, k는 설계자가 지정한 계수이다. 이들로부터 얻어진 값 Rx는 신호 Sx의 추정 범위이며, 이 범위로 부터 구해진 Wx를 신호 x의 정수부 단어길이로 결정한다. 평균과 표준편차는 스케메틱에서 추출되어 부동소수점 형태의 내부신호로 기술되는 VHDL 코드나 C 코드에 대한 시뮬레이션을 수행하면서 각 신호의 값을 관찰함으로써 얻어진다.

$$Rx = M(|Sx|) + k * \sigma(|Sx|)$$
$$Wx = \lceil \log_2(Rx) \rceil \quad (1)$$

소수부 단어길이는 먼저 각 신호의 그룹별로 최소 단어길이를 결정하고 전체적인 최적화 과정을 통해 결정한다. 각 그룹의 최소 단어길이는 다른 그룹들을 부동소수점으로 하고 해당 그룹만을 고정소수점으로 두어, 소수부 단어길이를 하나씩 증가시켜 나가면서 시뮬레이션을 수행할 때 원하는 시스템의 성능을 만족하는 가장 작은 단어길이를 각 그룹의 소수부 최소 단어길이로 결정한다. 전체적인 최적화 과정으로는 각 그룹별 단어길이 증가시 늘어나는 하드웨어 증가량을 계산한 후, 그 증가량이 작은 순으로 그룹들의 소수부 단어길이를 증가시키며 시뮬레이션을 수행하여, 원하는 시스템 성능을 만족시키는 가장 작은 단어길이를 최종적인 내부 신호들의 고정소수점 단어길이로 결정한다. 고정소수점 시뮬레이션은 부동소수점으로 표현된 내부 신호 값을 정해진 고정소수점 단어길이로 표현 가능한 값으로 전환하여 수행한다[6].

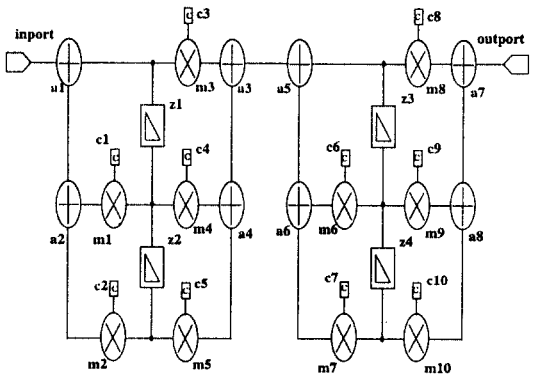


그림 2. 4차 IIR 필터
Fig. 2 A 4th order IIR filter.

표 1. 4차 IIR 필터의 단어길이 최적화 결과
Table 1. Results of bitwidth optimization for the 4th order IIR filter.

그룹	신호	고정소수점 단어길이 〈정수부, 소수부〉
P1	inport, a1out, a2out, m1out, m2out, z1out, z2out, c1out, c2out, c3out, c4out, c5out	〈5, 11〉
P2	a3out, a4out, a5out, a6out, m3out, m4out, m5out, m6out, z3out, z4out, c6out, c7out, c8out, c9out, c10out	〈7, 10〉
P3	a8out, m8out, m9out, m10out, outport	〈9, 9〉

성능 분석은 시불변 시스템의 경우 신호 대 잡음 비(SNR:Signal-to-Noise Ratio)로 판단하고, 시변 시스템의 경우는 오차를 최소화 하도록 빨리 수렴시키는 것이 목적이므로 수렴 속도, 또는 수렴 후 오차의 파워 등으로 정하는 것이 적당하다[6]. 그림 2는 4차 IIR 필터 모델을 스케메틱으로 표현한 것이고, 표 1은 70db의 SNR을 목표로 4차 IIR 필터의 내부 신호를 최적화한 결과이다.

그림 2에서 각 신호의 이름은 해당 신호를 출력하는 연산의 이름 뒤에 out을 붙여 지정하였다. 표 1에서 보듯이 전체 신호들은 동일한 단어길이를 사용하는 3개의 그룹으로 나뉘고, 이들은 각각 〈5, 11〉, 〈7, 10〉, 〈9, 9〉의 단어길이로 최적화되었다. 입력신호는 크기 -10~10 사이의 샘플링된 음성신호 4,664개를 사용하였으며, 이때 시스템 성능은 70.49 db가 되었다. 결정된 단어길이는 상위 수준 합성에서 공유 가능한 연산들로 분할하는 연산의 분할과정에서 각 연산의 입력신호 단어길이에 대한 정보로 사용된다. 표 2는 최적화된 내부신호 단어길이에 의해 결정된 각 연산의 입력신호 단어길이를 보여준다.

표 2. 4차 IIR 필터에서 최적화된 단어길이에 의한 각 연산의 입력신호 단어길이
Table 2. Bitwidths of input signals for each operator module after bitwidth optimization.

곱셈기	단어길이 (비트)	덧셈기	단어길이 (비트)
m1, m2, m3, m4, m5	16	a1, a2	16
m6, m7, m8, m9, m10	17	a3, a4, a5, a6	17
		a7, a8	18

Ⅳ. 연산의 분할

1. 문제의 정의 및 접근 방법

일반적으로 상위 수준 합성의 스케쥴링 과정에서 다른 컨트롤 스텝에 할당된 동일한 타입의 연산들은 모두 동일한 연산기를 공유 가능하다[13]. 연산기 할당 과정을 통해 연산기를 공유하는 연산들이 결정된 후, 그 연산기의 실제 면적은 공유하는 연산들의 입력신호 단어길이 중 최대값에 의해 결정된다. 따라서, 공유 가능한 연산들에 제약을 가해 실제 구현되는 연산기들의 입력신호 단어길이를 작게하여 전체 연산기의 면적을 줄여야 한다. 일반적으로 시스템에서 사용하는 연산기의 면적은 전적으로 상위 수준 합성 결과에 의존하게 되나, 대부분의 합성 시스템은 각 연산 타입별로 동일한 지연시간과 면적을 가정하여 수행하게 되므로, 고정소수점으로 표현된 내부 신호를 통해 얻어진 각 연산의 입력신호 단어길이에 의한 각 연산의 면적을 고려하지 못하게 된다. 최악의 경우 모든 연산기가 내부 신호 단어길이 중 최대값을 입력신호 단어길이로 설정하여 구현될 수도 있다. 따라서, 상위 수준 합성 전에 각 연산의 입력신호 단어길이 정보를 이용하여 구현될 전체 연산기의 면적을 줄일 수 있도록 특정 입력신호 단어길이를 가지는 연산기를 공유할 연산들을 지정할 필요가 있다.

그림 3에 상위 수준 합성 과정에서 연산기 할당 결과의 한 예를 가상으로 보인다. 이 예는 본 알고리듬의 목표를 개념적으로 단순화 시켜 제시한 것이다. 각각 5 비트(그룹 1)와 10 비트(그룹 2)의 입력신호 단어길이를 가지는 곱셈이 2개씩 총 4개가 있고, 이 연산들이 2개의 컨트롤 스텝에 걸쳐 수행되는 시스템의 설계에서 일반적인 상위 수준 합성기는 각 컨트롤 스텝에 2개씩을 시스템의 합성 알고리듬에 맞게 할당할 것이다. 이 때 그림 3(a)와 같이 각 컨트롤 스텝에 동일한 입력신호 단어길이를 가지는 연산들이 할당될 경우 연산기 할당 과정에서 반드시 10 비트와 5 비트의 연산을 하나씩 동일한 연산기에 할당하여야 하므

로, 결과적으로 10 비트짜리 곱셈기 2개가 필요하게 된다. 입력신호 단어길이를 이용하여 미리 2개의 그룹으로 나누고 연산기의 공유는 동일한 그룹내의 연산들끼리만 할 수 있도록 제약을 두고 합성을 수행할 때, 그림 3(b)의 결과와 같이 스케쥴링되고, 연산기 할당 역시 동일한 그룹내의 연산끼리만 공유 가능하므로 5 비트, 10 비트 곱셈기를 각각 1개씩만 사용하도록 할당된다. 결과적으로 전체 연산기를 구현함에 있어 그 면적을 줄일 수 있는 장점을 가진다. 또한 전체 연산의 면적은 사용할 연산기의 개수에 의존하므로,

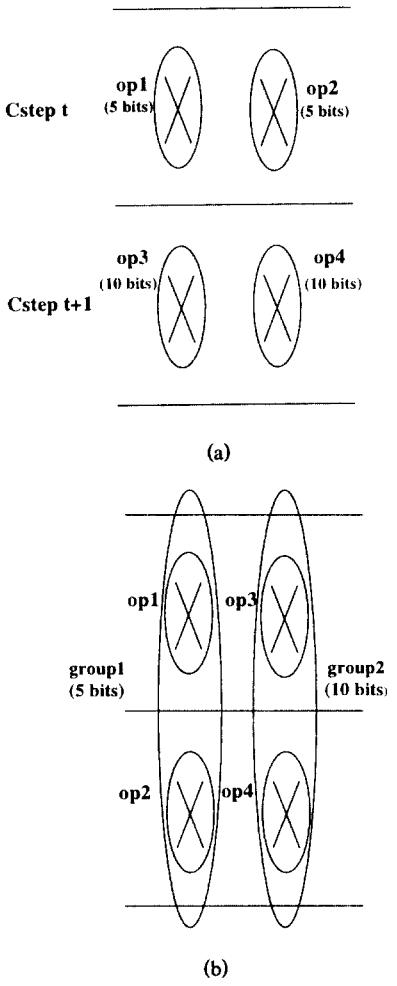


그림 3. 연산의 공유 (a)분할 전 (b)분할 후
Fig. 3 Sharing of FUs. (a) before partitioning (b) after partitioning

사용될 연산기의 하한치[14]를 계산하여 연산기 개수를 최적화하는 방향으로 분할함으로써 분할에 의하여 동일한 타입의 연산기 개수가 증가하는 것을 방지하여 효과적으로 면적을 줄일 수 있다.

SODAS-DSP 시스템에서는 시간 제약 조건과 각 연산별 분할하고자 하는 최대 분할수(MaxNp)를 입력으로 받아들여 사용될 연산기의 하한치를 이용하여 분할하지 않는 경우(분할수가 1일때)에서부터 최대 분할수까지 각각의 분할수 별로 최적화된 분할 결과를 찾아내고, 이 결과들 중에서 전체 연산기의 면적이 작은 결과를 최종적인 결과로 선택한다. 여기서 최대 분할수(MaxNp)는 분할하지 않을 때 사용될 연산기의 하한치로 결정한다. 개략적인 알고리듬은 다음과 같다.

```
OP_Partition {
   for(Npart = 1; Nparts MaxNp; Npart ++) {
      Initial_Partition();  /* 초기 분할수행 */
      Move_Single_OP();
            /* 단일 연산 이동에 의한 최적화 수행 */
      Move_Multiple_OPs();
            /* 다중 연산 이동에 의한 최적화 수행 */
      Cost[Npart] = Area_Npartition();
            /* Npart개로 최적화시 전체 연산기의 면적 */
   }
   Select_Minimal_Cost_Solution();
            /* 비용이 가장 작은 결과를 선택 */
}
```

MaxNp개까지 분할수를 하나씩 증가시키면서 각 분할수 별로 최적화된 분할 결과를 얻는 과정은 초기 분할 과정과 연산의 이동에 의한 최적화 과정으로 구성된다. 초기 분할 과정의 목적은 주어진 분할수대로 각 연산이 균등히 나뉘어 질 수 있도록 하는 것이다. 연산의 이동 과정의 목적은 초기 분할 결과로부터 하드웨어의 양이 줄어드는 방향으로 각 연산을 다른 그룹으로 이동 시킴으로써 하드웨어 비용을 최적화하는 것으로, 단일 연산 이동에 의한 최적화와 다중 연산 이동에 의한 최적화 과정으로 구성된다. 단일 연산 이동은 한번에 하나의 연산이 다른 그룹으로 이동하며 대부분의 최적화가 이 과정에서 이루어 진다.

그러나, 각 그룹에서 사용되는 연산기의 면적은 각 그룹에서 사용할 연산기의 하한치와 입력신호 단어 길이에 의존하기 때문에, 둘 이상의 연산을 이동시키는 다중 연산 이동을 통해 더욱 최적화된 결과을 얻을 수 있다.

2. 사용될 연산기의 하한치 결정

타입 T의 연산을 수행하기 위해 필요한 연산기의 하한치는 ASAP, ALAP 스케쥴링[1] 결과로부터 생성된 각 연산의 생존 구간으로부터 결정된다. 연산 타입 T로 사용되는 연산기의 하한치를 LB_T라고 표기하고, LB_T를 결정하는 컨트롤 구간을 하한 컨트롤 구간, 이 구간의 컨트롤 스텝수를 C_T라고 정의하고, 이 구간 내에 존재하는 연산 타입 T의 연산 개수를 N_T라고 하면 LB_T는 식 (2)로 계산된다[14].

$$LB_T = \lceil N_T / C_T \rceil \quad (2)$$

분할 과정에서 그룹 P_i에서 사용되는 연산 타입 T의 하한치를 $LB_{Pi,T}$라고 정의하면 이 값은 그룹 P_i 내에 속해있는 타입 T인 연산들 만을 고려하여 식 (2)에서와 같이 구할 수 있다. 이 하한치는 분할 과정 전과정에서 전체 연산기의 면적을 계산할 때와 다중 연산 이동 과정에서 각 그룹별 사용되는 연산기의 개수를 줄이는 방향으로 연산들을 이동할 때 사용된다. 그림 4는 총 수행시간 6 스텝으로 할때 4차 IIR 필터내의 곱셈 연산들의 생존구간을 나타낸다. 그림에서 하한 컨트롤 구간은 Cstep1에서 Cstep3까지이므로 C_{MUL}은 3이고 이 구간내에서 N_{MUL}은 7, LB_{MUL}은 3이 된다.

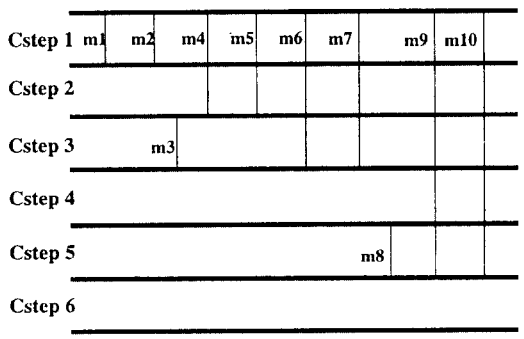


그림 4. 4차 IIR 필터 곱셈들의 생존 구간
Fig. 4 Lifetimes of the multiplications in the 4th order IIR filter.

3. 초기 분할

먼저 간단한 그래프 알고리듬으로 주어진 분할수가 될 때까지 각 연산들의 입력신호 단어길이의 차이가 작은 것끼리 병합시켜 나가는 방식으로 분할한다. 전체 연산이 공유 가능할 때(분할수가 1일때), 연산기의 하한치를 결정하는 컨트롤 구간에 존재하는 연산을 우선적으로 분할하고, 그 외의 연산을 후에 분할한다. 각 분할 과정에서는 연산이 집중적으로 모여있는 컨트롤 구간내의 연산들을 우선 분할하도록 하였다. 제안된 알고리듬에서는 분할하지 않았을때의 하한치의 70%를 반올림한 값을 기준으로 하여, 이 값보다 크거나 같은 연산기 수를 필요로하는 컨트롤 구간내의 연산들을 먼저 분할하도록 하였으며, 이러한 구간이 여럿 존재할 때는 큰 구간이 작은 구간을 포함할 수 있으므로 작은 구간부터 먼저 분할하도록 하였다. 이는 가능한 각 그룹에서 사용될 연산기의 개수를 균등하게 해주기 위해서이다. 또한 전체 분할 과정에서 주어진 분할수가 될 때까지 입력신호의 단어길이간의 차이가 작은 연산끼리 먼저 병합시킴으로써 비슷한 입력신호 단어길이를 가지는 연산들을 공유토록 한다.

초기 분할이 완료되면 각 그룹의 연산기가 사용하는 입력신호 단어길이는 해당 그룹에 속한 연산들의 입력신호 단어길이 중 최대값으로 설정하며, 그룹별 사용될 연산기의 하한치 $LB_{Pi,T}$도 결정된다. 표 3은 4차 IIR 필터를 2개의 그룹으로 초기 분할한 결과이다. 그림 4에서 곱셈들의 생존 구간을 보면 하한 컨트롤 구간은 Cstep1~Cstep3이고, 이 구간내에서 하한치의 70% 이상의 연산기가 필요하도록 연산이 몰려있는 구간은 Cstep1, Cstep1~Cstep2이며 구간의 크기에따라 분할 순서는 Cstep1, Cstep1~Cstep2, Cstep1~Cstep3순으로 된다. 따라서 Cstep1의 분할에 의해 {m1}과 {m2}가 분할되고, Cstep1~Cstep2 구간의 분할에 의해 {m1, m4}, {m2, m5}로, Cstep1~Cstep3 구간의 분할에 의해 {m1, m4, m3}, {m2, m5, m6, m7}로 분할되고, 최종적으로 그룹 P_1은 {m1, m4, m3, m8}, 그룹 P_2는 {m2, m5, m6, m7, m9, m10}으로 분할된다. 이때 P_1은

표 3. 4차 IIR 필터의 초기 분할 결과
Table 3. Results of initial partitioning for the 4th order IIR filter.

순서	연산 (입력길이(비트))	P_1 (입력길이(비트))	P_4 (입력길이(비트))
1	m1(16), m2(16)	m1(16)	m2(16)
2	m4(16), m5(16)	m1, m4(16)	m2, m5(16)
3	m3(16), m6(17), m7(17)	m1, m4, m3(16)	m2, m5, m6, m7(17)
4	m8(17), m9(17), m10(17)	m1, m4, m3, m8(17)	m2, m5, m6, m7, m9, m10(17)
하한치		1	2

1, P_2는 2개의 하한치로 연산자를 사용하게 된다.

4. 단일 연산 이동에 의한 최적화

초기 분할로부터 가능한 비슷한 단어길이를 가지는 연산들을 하나의 그룹으로 병합하고 각 그룹별로 사용될 연산기의 하한치가 균등하도록 원하는 개수의 분할 결과를 얻은 후, 이로부터 더욱 최적화된 결과를 얻기 위해 각 연산들을 다른 그룹으로 이동시키는 과정이 필요하다. 이 과정은 단일 연산 이동과 다중 연산 이동에 의한 최적화 과정으로 나뉘며, 전자는 한번에 하나의 연산만을 이동시키고 후자는 한번에 동일한 그룹내의 여러 연산들을 다른 그룹으로 분산 이동 시킴으로써 최적화하는 과정이다.

단일 연산 이동에 의한 최적화 과정에서는 각 연산들이 다른 그룹으로 이동할 때 얻어지는 하드웨어 이득을 계산하여 이 중 가장 큰 이득을 주는 연산을 이동시킨다. 각 연산의 이득은 초기 분할된 결과로부터 계산된다. 모든 연산이 한번씩 이동할 때까지 단일 연산 이동을 수행하고, 이동된 리스트를 기록하여 최종적으로는 이득의 합이 가장 큰 이동까지만 실제로 수행한다. 이는 Kernighan-Lin Paritioning 알고리듬[15]을 변형하여 적용시킨 것으로, 이득은 연산의 이동 전에 연산기를 구현하는데 필요한 로직 게이트의 수와 이동 후에 필요한 게이트 수의 차로써 게이트 수의 감소량으로 정의한다. 연산기를 구현하는데 필요한 게이트의 수는 각 연산기가 정해진 입력신호 단어길이로 동작할 때 필요한 게이트의 수와 사용될 해당 타입의 연산기의 하한치의 곱으로 계산된다.

각 연산의 이동 시 얻어지는 이득 G는 해당 연산이 속해있던 그룹에서 제거될 때 얻어지는 해당 그룹에서의 이득과, 다른 그룹으로 이동 시 그 그룹에서 얻어지는 이득중 최대값의 합으로 계산된다. 그룹 P_i에서 연산 op를 다른 그룹으로 이동 시 얻어지는 전체 이득 G(op)는 식 (3)으로 계산된다.

$$G(op) = InCost(op) + \min_j \{OutCost(op, Pj)\} \quad for \ op \in Pi \tag{3}$$

G(op) : 연산 op가 다른 그룹으로 이동 시 얻는 최대 이득

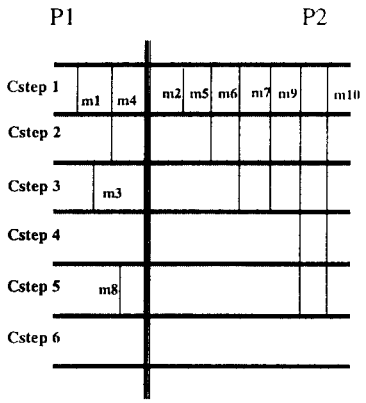
InCost(op) : 연산 op가 그룹 Pi에서 제거될 때 얻어지는 이득(≥ 0)

OutCost(op, P_j) : 연산 op가 그룹 P_j로 이동 시 얻어지는 이득 (≤ 0)

한번 이동된 연산은 lock되어 다시 선택되지 않도록 하고, 각 이동에 따른 이득을 계산할 때 그 이전의 이동들을 실제로 이동한 것처럼 고려하여 주어야 한다. 이는 이전의 이동에 의해 각 그룹의 사용될 연산기의 하한치와 입력신호 단어길이가 변하기 때문이다. 이러한 과정을 모든 연산들이 이동할 때까지 수행하고 이 중 이득의 합이 가장 큰 이동까지만 실제로 수행한다. 그림 5는 초기 분할 결과로부터 단일 이동에의한 최적화 과정을 수행한 결과이다. 그림 5(a)의 초기 분할 결과로부터 각 연산의 이득을 계산하면 연산 m8을 그룹 P_1에서 P_2로 이동시 P_1의 입력신호 단어길이가 17비트에서 16비트로 감소하여 이득이 768(게이트 수가 768개 줄어듬)로 가장 높으므로 먼저 m8을 이동시킨다. 그 이후는 모든 연산의 이득이 음수를 가지게 되므로 실제 이동을 수행할 때는 m8의 이동까지만 수행하게 된다.

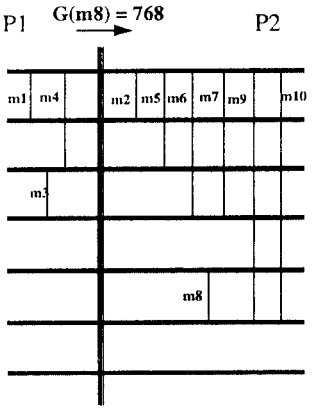
5. 다중 연산 이동에 의한 최적화

각 그룹별 연산기의 면적은 입력신호 단어길이와 사용될 연산기의 개수에 의존하며 그룹들의 면적들의 합이 전체 시스템의 연산기의 면적이 된다. 각 그룹별로 면적을 줄이기 위해서는 입력신호 단어길이를 줄이거나 연산기의 개수를 줄여야 한다. 최대 입력신호 단어길이를 가지는 연산의 수가 여러 개이거나 여러 연산들을 제거해야만 연산기의 수가 줄어드는 경우가 대부분이므로, 다중 연산 이동에 의해 이



	1	2
하한치		
단어길이	17	17
게이트수	7153	14306

(a)



	1	2
	16	17
	6385	14306

(b)

그림 5. 4차 IIR 필터에서 단일 연산 이동에의한 최적화 결과
(a)최적화 전 (b)최적화 후

Fig. 5 Optimization results by single operation movement.
(a)before optimization (b)after optimization

러한 연산들을 동시에 다른 그룹으로 분산 이동함으로써 더욱 최적화된 면적을 구할 수 있다. 이 과정은 한번에 이동될 연산들의 집합을 OpSet이라 할때, 분산 이동에 의해 가장 큰 이득을 주는 OpSet을 선택하여 이동함으로 수행된다. 이 과정을 더 이상 분산 이동할 OpSet이 존재하지 않을 때까지 계속 수행하면서 그 이동 리스트를 작성하고, 최종적으로는 가장 큰 이득의 합을 가지는 이동까지만 실제로 이동한다. 집합 OpSet의 모든 원소들은 동일한 그룹에 속해있는 연산들이다.

그룹 P_i의 연산 집합 OpSet이 다른 그룹으로 분산 이동될 때 얻어지는 전체 이득 G(OpSet)은 식 (4)와 같이, P_i에서 OpSet을 제거시 P_i에서 얻어지는 이득과 OpSet의 각 원소들을 각각 다른 그룹 P_j로 이동 시 P_j에서 얻어지는 이득 중 가장 큰(게이트의 증가량이 가장 적은) 것들의 총합과의 합으로 결정된다.

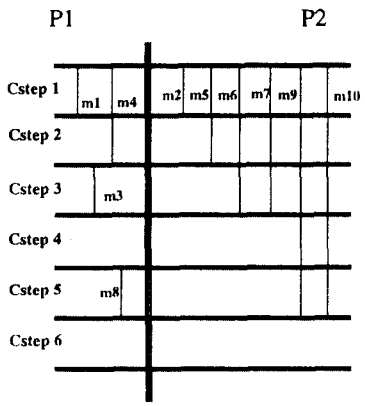
$$G(OpSet) = InCost(OpSet) + \Sigma \max_j\{OutCost(op, P_j)\} \quad \text{for } op \in OpSet \subset P_i \tag{4}$$

G(OpSet): 그룹 P_i내의 연산들 OpSet을 다른 그룹들로 분산 이동시킬 때 얻어지는 최대 이득

InCost(OpSet): P_i에서 OpSet을 제거시 얻어지는 이득 (≥ 0)

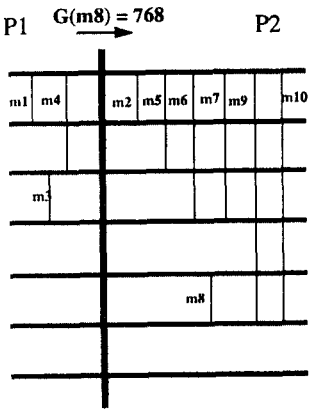
OutCost(op, P_j): OpSet의 원소 op가 P_j로 이동 시 P_j에서 얻어지는 이득 (≤ 0)

집합 OpSet의 원소들을 선택함에 있어 다음 사항을 고려하여 준다. 먼저 각 그룹에서 사용되는 연산기의 하한치를 줄일 수 있도록 연산들을 선택한다. 이를 위하여 각 그룹에서 사용하는 연산기의 하한치를 결정하는 하한 컨트롤 구간을 결정하고, 이 구간내에 존재하는 연산들 중에서 분산시킴으로써 해당 그룹의 연산기의 수를 줄이면서 가장 이득이 큰 OpSet을 추출해낸다. 이때 OpSet의 원소개수는 제거함으로써 연산기의 개수를 줄일 수 있는 최소 연산의 수로 결정하며, 그 값은 그룹 P_i에서 사용되는 연산기의 하한치를 $LB_{P_i,T}$, 하한치를 결정하는 컨트롤 구간의 컨트롤 스텝수를 $C_{P_i,T}$, 그 구간내에 존재하는 연산들의 수를 $N_{P_i,T}$라고 할 때, 식 (5)로 구해지고 이 값은 각 그룹에 할당된 연산기의 수를 줄일 수 있는 가장 적은 연산의 개수이며 이 값을 한번에 이동시키는 연산의 수로 설정하고 M_{P_i}로 표기한다. 최대 컨트롤 구간에서 M_{P_i} 수만큼 OpSet을 설정함에 있어 연산 선택의 탐색 순서는 가장 큰 생존 구간을 가지는 연산들 순으로



	P1	P2
하한치	1	2
단어길이	17	17
게이트수	7153	14306

(a)



	1	2
	16	17
	6385	14306

(b)

그림 6. 4차 IIR 필터에서 하한치를 고려한 다중 연산 이동에의한 최적화 결과
(a)최적화 전 (b)최적화 후

Fig. 6 Optimization results by multiple operation movement considering lower bound.
(a)before optimization (b)after optimization

하였다. 이는 큰 생존 구간의 연산을 다른 그룹으로 이동할 경우 연산을 받아들일 그룹에서 연산기의 개수가 늘어날 가능성이 적어지기 때문이다.

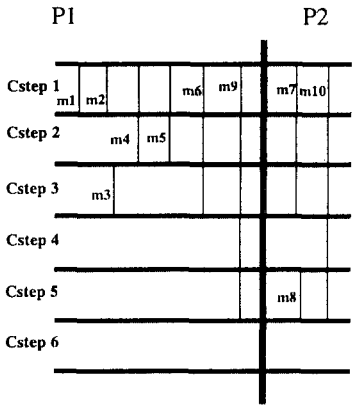
$$M_{Pi} = N_{Pi,T} - C_{Pi,T} * (LB_{Pi,T} - 1) \quad (5)$$

4차 IIR 필터의 단일 연산 이동에의한 결과로부터 하한치를 고려한 다중 연산 이동의 M_{Pi}를 계산해보면 다음과 같다. 그룹 P_2에서 하한치를 결정하는 구간은 1−5, $C_{P2,MUL}$는 5, $N_{P2,MUL}$은 7, $LB_{P2,MUL}$은 2 이므로 M_{P2}는 $7-5*(2-1)=2$가 되어 한번에 2개의 연산을 분산 이동시킨다. 이 M_{P2} 수만큼 연산을 택하여 OpSet을 설정하고 다중 연산 이동에의한 최적화를 수행한 결과는 그림 6과 같다.

다음으로 각 그룹에서 사용될 연산기의 입력신호 단어길이를 줄일 수 있도록 연산들을 선택한다. 각 그룹에서 이미 이동되지 않은 연산 중 가장 큰 입력 신호 단어길이를 가지는 연산들을 하나의 OpSet으로 설정한다. 이전의 이동에 의해 이미 해당 그룹으로 이동된 연산들 중 설정된 OpSet의 단어길이보다 더 큰 길이를 가지는 연산들이 있을 경우 이 연산들도 함께 해당 그룹에서 제거하여야 하므로 이 연산들을 설정된 OpSet의 원소로 첨가한다. 이 중 가장 큰 이득을 가지는 OpSet을 최종적으로 선택하되, 이 과정을 각 그룹에서 가장 작은 단어길이를 가지는 연산만이 남을 때까지 수행한다. 4차 IIR 필터의 예로는 앞과정까지 최적화된 결과를 얻을 수 있었으므로 이 과정에 대한 예는 4차 IIR 필터의 실험 결과로는 보일 수가 없다. 만일 단일 연산 이동에의한 결과가 그림 7(a)라고 가정하면 이 과정에의한 최적화 결과는 그림 7(b)와 같이 P_1에서의 m6, m9가 OpSet이 되어 P_2로 이동하여 결과적으로 16비트, 17비트의 곱셈기 각각 2개, 1개를 하한치로 사용하도록 분할될 것이다.

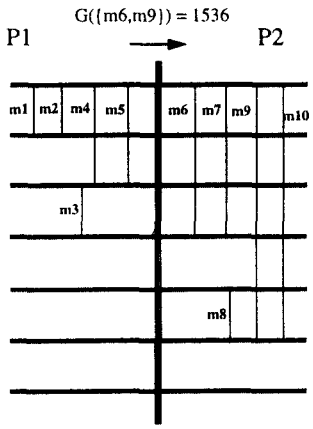
SODAS-DSP 시스템에서는 평균적으로 이득이 큰 연산기 사용 개수를 줄이는 경우를 먼저 수행하고 단어길이를 줄이는 경우를 이후에 수행한다. 이렇게 초기분할과 연산 이동에 의한 최적화 과정을 각 연산 타입별로 제약조건으로 주어진 최대 분할수까지 수행하고, 이 중 가장 적은 하드웨어 비용을 가지는 분할을 선택한다.

표 4는 단어길이 최적화된 4차 IIR 필터를 이용하



하한치	2	1
단어길이	17	17
게이트수	14306	7153

(a)



2	1
16	17
12770	7153

(b)

그림 7. 4차 IIR 필터에서 입력신호 단어길이를 고려한 다중 연산 이동에의한 최적화 결과
(a)최적화 전 (b)최적화 후

Fig. 7 Optimization results by multiple operation movement considering bitwidths of input signals.
(a)before optimization (b)after optimization

여 연산의 분할을 수행한 결과를 보인다. 전체 수행 시간을 6 컨트롤 스텝으로 하고, 체이닝과 멀티사이클링을 고려하여 곱셈기와 덧셈기의 최대 분할수는 각각 3, 3으로 하여 수행하였다. 표에서 보듯이 곱셈기는 16 비트의 입력신호 단어길이를 가지는 것 2개와 17 비트 1개, 그리고 덧셈기는 17 비트, 18 비트 각각 1개씩을 사용하도록 분할하였다. 하한치는 각 연산기별 사용될 하한치를 계산한 것이고, 면적은 하한치 개수대로 각 연산기를 구현할 때 필요한 로직 게이트의 개수를 계산한 것이다.

표 4의 연산 분할 결과를 입력으로 상위 수준 합성한 결과를 표 5에 보인다. 합성 결과 16 비트와 17비트 곱셈기가 각각 2, 1개를 사용되어 연산의 분할에서 예상한 하한치와 동일하고, 덧셈기는 17비트 1개, 18비트 2개로 하한치보다 18비트 덧셈기 하나가 더 사용되도록 합성되었다. 동일한 합성기로 연산들을 분할하지 않고 합성할 경우, 곱셈기 3개, 덧셈기 3개를 사용하도록 스케쥴링된다. 이 결과로 연산을 분할함으로써 곱셈기와 덧셈기의 면적을 줄일 수 있음을 확인할 수 있다.

V. 실험 결과

구현된 시스템의 성능을 평가하기 위해 4차 LMS 필터를 이용한 noise canceller를 설계하는 과정을 보인다. 실험은 SPARC10 기종의 work station상에서 C 언어로 구현하여 수행하였다. 시스템 성능은 40 db를 목표로 내부 신호들을 고정소수점으로 최적화시키고, 이를 6 컨트롤 스텝의 총 수행 시간으로 체이닝과 멀티사이클링을 고려하여 합성하였다. 그림 8은 noise canceller를 모델링한 것이다.

표 6은 단어길이를 최적화한 결과이다. 4개의 그룹으로 분할되었으며 각각 7비트, 12비트, 10비트, 9비트의 고정소수점으로 표현되며 이때 시스템의 성능은 40.65 db이고, 수행 CPU 시간은 25.9 sec이다.

표 7은 최적화된 단어길이로 부터 결정된 각 연산의 입력신호 단어길이이다. 1개의 덧셈이 9비트, 4개의 곱셈과 4개의 덧셈이 10비트, 5개의 곱셈과 2개의 덧셈 그리고 1개의 뺄셈이 12비트의 입력신호 단어길이를 가진다.

이때 연산의 분할 과정은 곱셈기, 덧셈기, 뺄셈기의

표 4. 연산의 분할 결과
Table 4. Results of operation partitioning.

	곱셈기				덧셈기			
그룹	연산	단어길이	하한치	게이트수	연산	단어길이	하한치	게이트수
P1	m1, m2, m3, m4, m5	16	2	12,770	a1, a3, a4	17	1	1,051
P2	m6, m7, m8, m9, m10	17	1	7,153	a2, a5, a6, a7, a8	18	1	1,124

표 5. 4차 IIR 필터에 대한 상위 수준 합성 결과
Table 5. Results of high level synthesis for the 4th order IIR filter.

	곱셈기		덧셈기	
컨트롤 스텝	M1 (16 비트)	M2 (17 비트)	A1 (17 비트)	A2 (18 비트)
1	m1, m2	m6		
2	m4, m5	m7	a1	a2
3	m3	m10	a4	
4		m9	a3	a5, a6
5		m8		a8
6				a7

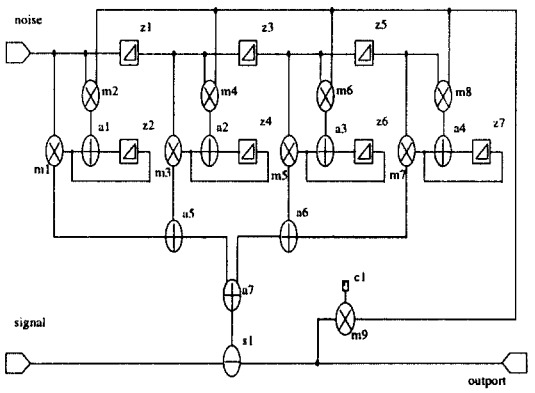


그림 8. 4차 LMS 필터를 이용한 noise canceller
Fig. 8 A noise canceller using the 4th order LMS filter.

표 6. 신호의 단어길이 최적화 결과
Table 6. Results of bitwidth optimization.

그룹	신호	단어 길이 〈정수부, 소수부〉
P1	noise	〈1, 6〉
P2	signal, c1out, a5out, a7out, s1out, m1out, m9out	〈6, 6〉
P3	z1out, z2out, z3out, z4out, z5out, z6out, z7out, a1out, a2out, a3out, a4out, m2out, m4out, m6out, m8out	〈4, 6〉
P4	a6out, m3out, m5out, m7out	〈3, 6〉

표 7. 신호의 단어길이 최적화에 따른 각 연산의 입력신호 단어길이
Table 7. Bitwidths of input signals for each operation determined by bitwidth optimization.

단어길이	곱셈기	덧셈기	뺄셈기
9 bits		a6	
10 bits	m1, m3, m5, m7	a1, a2, a3, a4	
12 bits	m2, m4, m6, m8, m9	a5, a7	s1

최대 분할수를 각각 4, 4, 1로 지정하고 실행하였다. 그 결과를 이용하여 상위수준합성을 한 결과를 표 8에 보인다. 표에서 보이듯이 곱셈은 4개의12 비트 곱셈기, 덧셈은 1개의 12 비트 덧셈기, 3개의 10 비트 덧셈기, 그리고 1개의 12 비트 뺄셈기를 하한치로 사용하여 noise canceller가 구현되었다. 분할을 수행하지 않을 경우 4개의 12비트 곱셈기, 4개의 12비트 덧셈기, 1개의 12비트 뺄셈기를 사용하도록 합성된다. 따라서 4개의 12비트 덧셈기를 사용할 것을 1개의 12비트 덧셈기와 3개의 10비트 덧셈기를 사용하도록 합성

함으로써 면적을 줄일 수 있었다. 분할을 수행하는데 걸린 CPU 시간은 16.1sec이다.

표 8. 연산의 분할 후 상위수준합성 결과
Table 8. Results of high-level synthesis after operation partitioning.

연산 타입	그룹	연산	단어길이 (비트)	하한치	면적 (게이트수)
곱셈	P1	m1, m2, m3, m4, m5, m6, m7, m8, m9	12	4	14,848
덧셈	P1	a1, a5, a7	12	1	710
	P2	a2, a3, a4, a6	10	3	1,626
뺄셈	P1	s1	12	1	722

분할 결과를 이용하여 상위 수준 합성을 한 결과는 표 9와 같이 연산의 분할에서 예상한 하한치로 합성됨을 알 수 있다. 면적은 각 입력신호 단어길이를 수행하는 연산기를 구현하는데 사용되는 로직 게이트의 수와 합성결과 사용될 하한치와의 곱으로 계산되었다.

표 9. noise canceller에 대한 상위 수준 합성 결과
Table 9. Results of high level synthesis for the noise canceller.

컨트롤 스텝	곱셈	덧셈		뺄셈
		p1	p2	
1	m1, m3, m5, m7			
2		a5	a6	
3		a7		s1
4	m9			
5	m2, m4, m6, m8			
6		a1	a2, a3, a4	

Ⅵ. 결론 및 추후 과제

본 논문은 DSP 하드웨어 설계 자동화 시스템인 SODAS-DSP에서 최적화된 하드웨어 비용으로 고정소수점 하드웨어를 합성하기 위해 구현한 고정소수점 전환 알고리듬, 연산의 분할 알고리듬과 그 구현 결과에 대하여 기술하였다. SODAS-DSP는 설계자가 시스템 수준에서 모델링한 DSP 알고리듬으로부터 고정소수점 내부신호를 가지는 하드웨어를 자동 합성해주는 시스템으로, 내부신호를 최적화된 고정소수점으로 전환하고 각 연산들을 특정 입력신호 단어길이를 가지는 연산기를 공유할 수 있는 그룹들로 나누어 상위 수준 합성을 통해 고정소수점 데이타패스와 컨트롤러를 합성해낸다.

결과로서 제시한 4차 IIR 필터와 4차 LMS 필터를 합성한 결과에서 SODAS-DSP 시스템은 특정 응용범위에서 사용되는 입력신호들을 이용하여 내부신호를 시스템의 성능을 만족하는 범위에서 고정소수점으로 최적화를 수행하였고, 이 결과로부터 상위 수준 합성 전에 각 연산들을 특정 입력신호 단어길이를 사용할 수 있는 여러개의 그룹들로 분할함으로써 사용될 연산기의 면적을 최적화가 가능하다. 최적화된 알고리듬은 상위 수준 합성 과정을 통해 원하는 시스템의 수행 속도에 맞도록 데이타패스를 생성할 수 있다.

SODAS-DSP는 각 연산의 입력신호 단어길이를 이용하여 연산들을 분할하는 과정에서 연산기의 면적만을 고려하여 수행하였으나, 추후 연구로 연산기의 면적뿐만 아니라 멀티플렉서등의 연결 구조 비용을 포함한 전체 하드웨어 면적을 고려하여 합성해주는 알고리듬과, 고정소수점 연산에의한 각 연산기의 처리 속도 변화에 따른 다양한 합성 방법에 대한 연구가 필요하다. 또한 연산의 분할 과정을 기존의 module selection의 일환으로써 전체 시스템의 면적과 처리 속도를 고려하여 다양한 연산 모듈중 적절한 모듈을 선택하는 과정으로 통합시켜야 할 것이다.

Acknowledgement

본 논문은 통상산업부, 정보통신부, 과학기술처에서 시행한 선도기술 개발사업인 ASIC 기반 기술개발의 연구 개발의 결과임.

참 고 문 헌

1. M. C. McFarland, "The High Level Synthesis of Digital Systemss", *IEEE Proceedings*, Vol. 78, No. 2, pp. 302-318, Feb. 1990.
2. N. Park, A. C. Parker, "SEHWA: A Software Package for synthesis of pipelines from behavioral

specification," *IEEE Trans. CAD*, Vol. 7, No. 3, pp. 356-370, March 1988.
3. D. Gajski, *Silicon Compilation*, Addison Wesley Pub.: Reading, MA, 1988.
4. B. S. Haroun, M. I. Elmasry, "Architectural synthesis for DSP silicon compilers," *IEEE Trans. CAD*, Vol. 8, No. 4, pp. 431-447, April 1989.
5. F. Catthoor, H. J. De Man, "Application specific architectural methodologies for high throughput digital signal and image processing," *IEEE Trans. ASSP*, Vol. 38, No. 2, pp. 339-349, Feb. 1990.
6. K. Kum, W. Sung, "VHDL Based Fixed-point Digital Signal Processing Algorithm Development Software", in Proc. International Conference on VLSI and CAD, pp. 257-260, Seoul, Korea, Nov. 1993.
7. C. T. Hwang, J. H. Lee, "A Formal Approach to the Scheduling Problem in High Level Synthesis," *IEEE Trans. CAD*, vol. 10, no. 4, pp. 464-475, April 1991.
8. H. S. Jun, S. Y. Hwang, "Design of a Pipelined Datapath Synthesis System for Digtal Signal Processing," *IEEE Trans. VLSI Systems*, Vol. 2, No. 3, pp. 292-303, Sept. 1993.
9. P. G. Paulin, "Force Directed Scheduling for the Behavioral Synthesis of ASIC's", *IEEE Trans. CAD*, Vol. 8, No. 6, pp. 661-679, June 1989.
10. H. S. Jun, S. Y. Hwang, "Automatic Synthesis of Pipeline Structures with Variable Data Initiation Intervals", in *Proc. 31st DAC*, pp. 537-541, June 1994.
11. H. D. Lee, H. S. Jun, S. Y. Hwang, "Datapath Synthesis in Sogang Silicon Compiler", in *Proc. JTC-CSCC*, pp. 301-318, Dec. 1990.
12. 임완수, 박재환, 황선영, "상위수준 합성을 위한 컴포넌트 라이브러리에 관한 연구", 한국정보과학회 논문지, 제 20권 12호, pp. 1867-1878, 1993년 12월.
13. D. Gajski, N. Dutt, A. Wu, S. Lin, *High-level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers: Norwell, MA, 1992.
14. 엄성용, 전주식, "하한 비용 추정에 바탕을 둔 최적 스케줄링 기법", 대한전자공학회 논문지, 제 28권, 12호, A편, pp. 73-87, 1991년 12월.
15. W. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, Vol. 49, pp. 291-307, 1970.

崔 正 一(Jeong-Il Choi) 정회원
1994년 2월 : 서강대학교 전자공학과 졸업
1996년 2월 : 서강대학교 전자공학과 석사
1996년 3월~현재 : 삼성전관(주) 재직중
※주관심분야 : DSP 시스템 설계, VLSI 설계 등임

田 弘 信(Hong-Shin Jun) 정회원
1967년 8월 22일생
1989년 2월 : 서강대학교 전자공학과 학사
1991년 2월 : 서강대학교 전자공학과 석사
1995년 2월 : 서강대학교 전자공학과 박사
1991년 4월~1996년 3월 : 서강대학교 산업기술연구소 연구원
1995년 3월~1996년 3월 : 삼성전자 반도체총괄 기술자문
1996년 4월~현재 : 삼성전자 System LSI 본부 선임연구원
※주관심분야 : Design Automation, High Level Synthesis, Design for Testability 등

李 廷 周(Jung-Joo Lee) 정회원
1996년 2월 : 서강대학교 전자공학과 학사
1997년 3월~현재 : 서강대학교 전자공학과 석사 과정 재학중
※주관심분야 : Design Automation, High Level Synthesis, Digital Signal Processing 등

金 汶 秀(Moon-Soo Kim)정회원

1996년 2월 : 서강대학교 전자공학과 학사

1997년 3월~현재 : 서강대학교 전자공학과 석사과정 재학중

※주관심분야 : Design Automation, High Level Synthesis, Digital Signal Processing 등

黃 善 泳(Sun-Young Hwang)정회원

1976년 2월 : 서울대학교 전자공학과 학사

1978년 2월 : 한국과학원 전기 및 전자공학과 공학석사 취득

1986년 10월 : 미국 Stanford 대학 공학박사 학위 취득

1976년~1981년 : 삼성 반도체 주식회사 연구원

1986년~1989년 : Stanford 대학 Center for Integrated Systems 연구소 책임연구원, Fairchild Semiconductor Palo Alto Research Center 기술자문

1989년~1992년 : 삼성전자(주) 반도체 기술자문

1989년 3월~현재 : 서강대학교 전자공학과 부교수

※주관심분야 : CAD 시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임