論文94-31A-12-16

Multichip아키텍춰 합성 알고리듬 설계

(The design of a Synthesis Algorithm for Multichip Architectures)

朴 在 煥*, 田 弘 信**, 黃 善 泳***

(Jae Hwan Park, Hong Shin Jun and Sun Young Hwang)

要約

본 논문에서는 multichip 아키텍춰의 상위수준 합성을 위한 휴리스틱 방법을 제시한다. 시스템에서의 칩의 면적, I/O pin의 수, 칩간 연결 수, 칩간 통신 지연시간 그리고 칩 latency등의 제약조건을 만족하는 multichip을 구현하기 위하여, 제안된 시스템은 알고리듬 단계의 행위 기술로부터 제약조건들을 만족하며 파이프라인 구조 multichip 아키텍춰를 자동 합성한다. SFG로부터 효율적으로 multichip 아키텍춰를 합성하기 위해 동시에 partitioning과 스케쥴링을 수행하는 새로운 방법을 제시한다.

벤치마크 회로에 대한 실험 결과 본 시스템이 효율적으로 multichip 하드웨어를 설계함을 보였다.

Abstract

Design of a heuristic algorithm for high level synthesis of multichip architecture is presented in this paper. Considering the design constraints: individual chip area, I/O pin counts, chip-to-chip interconnection counts, interchip communication delay, and chip latency, the proposed system automatically generates pipelined multichip architectures from behavioral descriptions. For efficient mulichip synthesis, a new methodology is proposed, which performs partitioning and scheduling of SFG into multichip architectures simultaneously.

Experimental results for several benchmark programs show that the system can be used for designing multichip hardware efficiently.

Ⅰ. 서 론

상위수준 합성은 설계하고자 하는 하드웨어의 동작을 알고리듬 수준에서 기술하고 이로부터 데이타패스

* 正會員 西江大學校 電子工學科
(Sogang Univ., CAD & Computer Systems Lsb.)
接受日字 : 1994年 1月 10日

와 콘트롤로 구성된 레지스터 전송(Register Transfer) 수준의 설계를 자동적으로 생성하는 과정이며, 그 과정은 스케쥴링과 모듈할당으로 구성된다. 레지스터 전송 수준의 데이타패스는 FU (Functional Units), 레지스터 그리고 그들의 상호 연결을 위한 연결 구조로 구성되며, 콘트롤은 데이타 패스가 요구된 동작을 정확하게 수행하도록 각 모듈을 구동하게 된다. 스케쥴링은 입력된 알고리듬 기술에서 사용한 연산을 하드웨어의 면적, 지연시간, 전력소모 등의 설계 제약조건을 만족하는 최적의 제어 구간에 할당하는 과정이다. 스케쥴링 알고리듬은 합성 방법에 따라 transformation을 기본으로 하는 알고리듬[1]과 모든 연산이 스케쥴될 때까지 한번에 하나씩 제어구간에 할당하며 반복하는 반복개선(iterative/constructive) 방법[2]이 있다. 모듈할당 과정에서는 하드웨어의 효과적인 사용과 latch나 mux와 같은 부가적 하드웨어의 최소화를 위해 최적의 공유가 되도록 연산을 연산자에 할당하고 연결구조를 형성한다.[3][4] 하드웨어는 기본적으로 FU, 메모리 소자와 mux나 버스 등 연결구조가 있다. 모듈할당 과정에서 이들 모두를 동시에 최소화하는 것은 매우 복잡한 문제로, 상위수준 합성 시스템들은 이들 과정을 분리하여 최소화하는 방식을 사용한다.[5]

DSP (Digital Signal Processing)는 speech, audio, image processing, video, radar등 넓은 응용 범위를 갖고 급속한 성장을 하고 있다. DSP는 선형 계산뿐만 아니라 로그나 비트 연산과 같은 비선형 계산을 포함하고 벡터나 행렬과 같은 다차원 신호를 처리하는 경우가 대부분이다.[6] DSP 알고리듬은 샘플 또는 프레임을 단위로 주기적으로 수행되는 것으로 DSP 알고리듬 기술로 부터 데이타패스를 합성하는 DSP 상위수준 합성기는 실시간 처리 개념이 포함되어야 한다. 대부분의 DSP 알고리듬은 제한된 칩 면적 내에서 설계하기엔 너무 커 하드웨어 partitioning이 요구되며, partitioning은 전체 시스템의 성능에 큰 영향을 주게된다. Multichip 설계를 위한 설계 제약조건으로 칩간 통신 지연시간 (한 칩에서 다른 칩으로의 데이타 전송에 필요한 지연시간), 입출력 핀 수, 칩간 연결 수, 각 칩의 면적, 그리고 칩의 latency가 고려되어야 한다.[7] Multichip 아키텍춰의 합성은 주어진 문제를 partitioning하고 스케쥴링하는 과정과 각 칩에 대한 모듈할당 과정으로 구성된다.

최근 multichip 아키텍춰를 지원하는 합성기에 대한 많은 연구가 있었다. 기존 합성기의 대부분은 스케쥴링과 partitioning을 분리하여 수행하며 일부 시스템은 동시에 수행한다. APARTY[8][9]와 Parker의 시스템[10]은 스케쥴링과 partitioning을 분리 수행하는 시스템으로 APARTY는 면적면에서 좋은 결과를 얻지만 multichip에서 중요한 문제인 칩간 통신 지연시간을 고려하지 못했으며, Parker의 시스템은 partition된 문제에 대해 효과적으로 스케쥴링을 수행하나 합성 결과가 partitioning에 많은 영향을 받는 시스템이다. CHOP[11]은 partitioning에 역점을 둔 시스템으로 노드를 partitioning한 후 모든 partition에 대해 다양한 설계를 하지만 이 시스템 역시 스케쥴링으로부터 partitioning을 분리하여 수행하므로 효율적이지 못하다. Gebotys는 partitioning과 스케쥴링, 그리고 모듈할당을 동시에 수행하는 시스템을 발표하였다.[7] Gebotys는 통신 지연시간을 고려하여 integer programming (IP)을 사용 최적의 결과를 얻고 있으나 설계하고자 하는 알고리듬이 크면 비효율적인 시스템이 된다.

본 논문은 fixed DII (Data Initiation Interval)[12]를 갖는 파이프라인 구조를 multichip으로 합성하는 방법을 제안한다. 효과적인 합성을 위해 연산의 스테이지에 분산되는 정도와 각 칩으로 분배되는 정도를 목적함수로 정의하고 목적함수의 미분계수를 우선순위 함수로 사용하여 점진적으로 연산을 각 칩의 스테이지에 균등히 배분되도록 하는 partitioning과 스케쥴링을 동시에 수행하는 방법을 제시한다. 그리고 연결구조로 사용하는 mux 입력의 수와 latch의 수를 비용함수로 multichip 아키텍춰에 대한 모듈할당 알고리듬을 제시한다. 장에서 multichip 아키텍춰에 대해 설명하고, 장에선 multichip 합성을 위한 새로운 알고리듬을 제시하며 장에서 벤치마크 회로에 대한 합성 결과를 기존의 시스템과 비교하고 장에서 결론을 맺는다.

Ⅱ. Multichip 아키텍춰

1. Multichip 아키텍춰

넓은 응용범위를 갖는 DSP는 과거의 아날로그로 처리하던 분야를 디지탈로 처리하게 되면서 그 범위를 날로 확장하고 있다. DSP 알고리듬 구현에는 많은 하드웨어 면적이 필요하여 DSP 알고리듬을 작은 단위로 partitioning을 수행하여 각각의 모듈을 하나의 칩으로 구현하는 접근 방식이 시도되고 있다.

Multichip 아키텍춰는 몇개의 칩과 그들 상호간의 연결 아키텍춰를 갖게되며, 간단한 multichip 아키텍춰는 그림 1와 같다. 각 칩은 연결 구조를 갖으며, 칩 P1에서 다른 칩 P2로 데이타를 전송할 필요가 있을 경우 P1으로부터 P2로 직접적인 통신 연결이 버

스를 통해 이루어 진다. 그림에서 d_{12}는 칩 P1으로부터 P2로 데이타 전송에 소요되는 지연시간이 된다. 합성된 아키텍춰에서 k개의 subtask로 구성된 task를 수행할 때 모든 subtask는 각 칩으로 partition되어 수행되는 구조를 갖는다. 데이타의 전송은 두 subtask S_i와 S_j에 데이타 의존 관계가 있을 경우에 발생하고 remote와 지협적 전송으로 구분된다. Remote 전송은 두 subtask가 서로 다른 칩에 있을 경우의 데이타 전송이고, 이때 소요되는 시간이 칩간 통신 지연시간이다.[13]

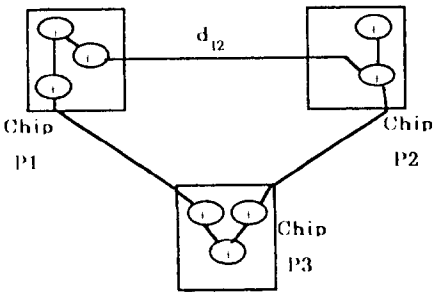


그림 1. 간단한 multichip 아키텍춰
Fig. 1. Simple multichip architecture.

2. 타겟 아키텍춰

그림 2는 multichip 아키텍춰를 나타낸 것이다. 본 연구에선 각 칩의 면적을 균등하게 하며 칩간 연결 모듈로 버스를 사용한다. 각 칩은 병렬적으로 수행될 수 있도록 partitioning을 수행하며, multi-bus 아키텍춰를 취하고 있다. Multi-bus 아키텍춰는 칩간 연결되는 통신 모듈이 하나 이상으로 합성될 경우에 대한 것이다. 그림 2에서와 같이 각 칩은 독립된 지협적 콘트롤러를 가지며 그림에서 COM_1, ... ,COM_k는 칩간 데이타 전송을 위해 사용되는 버스고 pn.1, ... ,p_n.k는 칩 P_n의 첫번째부터 k 번째 port에 연결된 것이다.

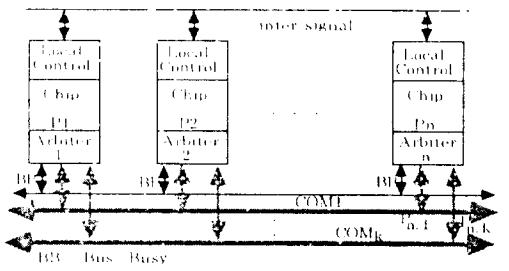


그림 2. Target multichip 아키텍춰
Fig. 2. Target multichip architecture.

Ⅲ. Multichip 아키텍춰 합성 알고리듬

이 장에서는 SODAS-DSP에서 entropy based scheduling (EBS) 알고리듬[15][16]을 이용하여 Multichip 아키텍춰의 합성 알고리듬을 제시한다. 1절에서는 Multichip 아키텍춰 합성을 위한 EBS 알고리듬을 설명하고, 2 절에서는 Multichip 아키텍춰를 고려한 모듈할당에 대해 알고리듬을 제시한다.

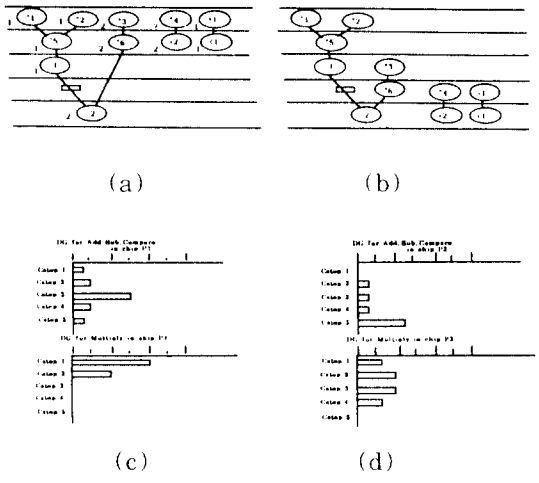
1. Multichip 아키텍춰 합성을 위한 EBS 알고리듬

EBS 스케쥴링 알고리듬[15][16]은 우선순위 함수의 정의에서 SFG상의 연산이 ASAP와 ALAP 스케쥴링 결과로 스케쥴될 수 있는 범위를 가지는 것을 한 상태로 간주하면, 임의의 연산이 특정 스테이지에 스케쥴되는 것을 포함하여 연산의 범위가 변할 때 상태 변화가 일어나는 것으로 한다. 이 개념을 이용하여 EBS 알고리듬을 multichip 아키텍춰 합성하기 위한 알고리듬으로 보안 개선할 수 있다. 위 개념의 확장을 구체화 하면 다음과 같다. 두개의 칩 P1, P2가 있을 경우 P1에 속하는 연산이 P2로 이동되므로 발생하는 노드의 스케쥴될 수 있는 범위의 변화를 또하나의 상태 변화로 간주하여 스케쥴링을 할 수 있다. 1.1절은 multichip을 고려한 ASAP, ALAP 스케쥴링 그리고 DG[17]를 구성하는 방법에 대해 기술하며, 1.2절은 multichip을 고려한 스케쥴링 방식에 대해 설명한다. 1.3절은 간략한 스케쥴링 알고리듬을 기술한다.

1) ASAP와 ALAP 스케쥴링과 DG 구성

SFG를 구성하는 노드와 노드사이에는 네트로 연결되어 있으며, 이 네트들 중 일부는 칩과 칩을 연결하는 네트가 될 수도 있다. 칩과 칩을 연결하는 네트를 통신 네트라 하며, 칩간 연결되는 통신 연산이 된다.[17] ASAP와 ALAP 스케쥴링은 SFG(Signal Flow Graph)를 입력단과 출력단으로부터 탐색하며, 각 스테이지에 연산을 할당하는 방식으로 탐색과정에서 칩간을 연결하는 네트에 도달하면 그 네트를 지연시간이 1인 칩간 통신 연산으로 하여 스케쥴링 과정을 계속 반복하는 것이다. 연산의 ASAP와 ALAP 스케쥴링 값을 변수 asap와 alap로 표현하면 연산이 스케쥴링될 수 있는 범위는 [asap, alap]가 되며, 스케쥴링은 asap의 증가와 alap의 감소에 의한 상태변화뿐 아니라 칩간 이동에 의한 상태변화도 고려한다. 각 노드가 속하는 칩은 변할 수 있으며 이로인해 칩간 연결하는 네트도 바뀔 수 있고, 칩간 연결되는 네트가 많을 수록 이 네트에 대응되는 통신 연산은 증가한다.

ASAP와 ALAP 스케쥴링 결과를 이용해 구성되는 DG는 각 칩에 독립적으로 존재하며 각 칩의 DG가 균등하게 분산되도록 한다. 모든 연산의 asap와 alap에 의한 DG의 계산은 그 연산이 속한 칩의 DG에만 누적이 된다. 그림 3은 간단한 SFG에 대한 ASAP와 ALAP 스케쥴링 결과와 이를 이용 각 칩에서 구한 DG를 예로 보이고 있다. 그림 3의 (a)와 (b)는 SFG의 ASAP, ALAP 스케쥴링 결과이며 연산의 왼쪽 하단의 수는 그 연산이 속하는 칩의 번호가 된다. 합성을 위한 multichip의 수는 2이며 P1, P2로 칩을 표현한다. Single-chip에서는 ASAP와 ALAP 스케쥴링이 4 스테이지에서 수행되며, multichip에서는 그림과 같이 칩간 연결되는 네트가 연산 -1과 -2사이에 존재하므로 ASAP와 ALAP 스케쥴링이 5 스테이지에서 수행된 결과를 보이고 있다. 그림 3의 (c)와 (d)에 각 칩에서의 각 연산자에 대한 DG를 보였다.



(a) (b)

(c) (d)

그림 3. Multichip 생성을 위한 ASAP, ALAP 스케쥴링과 DG
(a) ASAP 스케쥴링 (b) ALAP 스케쥴링
(c) 칩 P1의 DG (d) 칩 P2의 DG

Fig. 3. Example of ASAP and ALAP scheduling and DG for multichip.
(a) ASAP scheduling
(b) ALAP scheduling
(c) DG for chip P1
(d) DG for chip P2

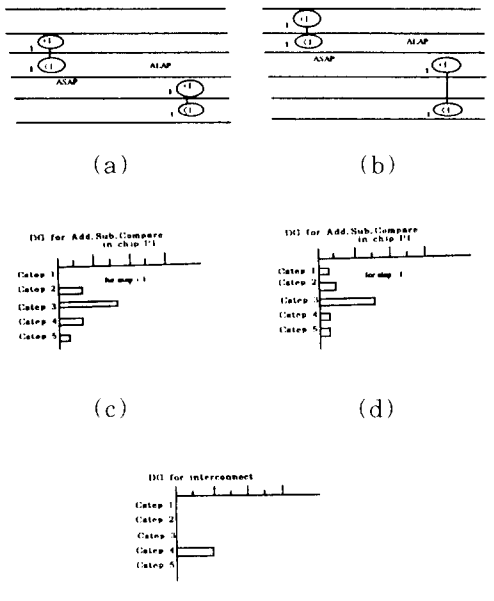
2) Partitioning을 고려한 스케쥴링

EBS 스케쥴링에서 사용하는 목적함수의 특성은 각 요소의 확률적 빈도수가 비슷할 경우 큰 값을 가진다. 이 특성은 스테이지를 요소로 했을 때 뿐만 아니라 칩 partition을 요소로 할 때에도 변하지 않는다. Single-chip 아키텍춰의 경우 목적함수의 확률적 요소는 스테이지와 그에 할당될 수 있는 연산의 수 DG가 되므로 목적함수는 2차원적인 문제가 되나 multichip 아키텍춰에서는 목적함수가 스테이지와 연산의 수 DG에 칩 partition을 요소로 추가한 3차원적인 문제가 된다. Single-chip에서 각 스테이지에 연산을 균등히 분산시키는 문제가 multichip에서는 각 스테이지뿐 아니라 각 칩으로도 균등히 분산시키는 문제로 확장된 것으로 목적함수로 모델링하기 어렵다. 그러므로 본 연구에선 multichip의 3차원적인 목적함수 문제를 스테이지의 확장된 개념으로 칩 partition을 고려하여 단순한 2차원적인 목적함수 문제로 단순화 한다. 이 개념은 각 칩에서 독립적으로 구성된 스테이지에 대한 DG를 스테이지 증가 방향으로 더한다. N을 최대 스테이지로 하는 multichip 아키텍춰의 합성에서 첫번째 칩의 DG가 1에서 N까지, 다음 칩의 DG가 N+1에서 2N까지의 가상적인 스테이지로 구성된 것으로 보며 가상적인 스테이지에 모든 연산을 균등히 분산시킴으로써 partitioning과 스케쥴링이 동시에 고려될 수 있다.

연산의 ASAP와 ALAP 스케쥴링 값이 asap와 alap이라할 때, 연산이 할당될 수 있는 첫째 스테이지는 asap이고 마지막 스테이지는 alap가 된다. 상태변화는 연산의 첫번째 스테이지 asap와 마지막 스테이지 alap를 구성 요소로 하는 스케쥴링 가능 범위의 감소와 연산의 칩간 이동에 의해 발생되며, partitioning과 스케쥴링은 연산의 asap와 alap를 증감시키므로 발생하는 스케쥴링 가능 범위의 감소에 대한 상태변화와 칩간 이동에 의한 상태변화를 목적함수로 모델링한다. 목적함수는 3가지 형태의 상태변화에 대해 구하며, 연산 op의 asap를 증가시킨 결과로 발생된 상태변화를 $S_{op\ asap}$라 하고 alap의 감소로 발생된 상태변화를 $S_{op\ alap}$라 한다. 그리고 연산 op의 칩간 이동으로 발생된 상태변화를 $S_{op\ part}$로 정의한다.

그림 4는 그림 3의 상태에서 연산 +1의 ASAP 스케쥴링 값 asap를 증가시키므로 발생된 상태변화 $S_{+1\ asap}$와 ALAP 스케쥴링 값 alap의 감소로 발생된 상태변화 $S_{+1\ alap}$에 대한 SFG와 그에 따른 DG를 계산한 예이다. 그림 4 (a)와 (b)는 그림 3으로부터 변화된 상태변화 $S_{+1\ asap}$와 $S_{+1\ alap}$에 대해 새로 구성되는 SFG를 연산 +1에 데이타 의존관계가 있는 부분만 그린 것이다. 그림 3에서 초기 ASAP와 ALAP 스케쥴링 결과 연산 +1의 스케쥴링 가능 범위는 [1, 4]

다. 그림 4 (a)에서 연산 +1의 asap 값을 1에서 2로 증가시키면 스케줄링 가능 범위가 [2, 4]로 상태변화 $S_{+1,asap}$가 되고 데이타 의존 관계에 있는 연산 <1도 asap가 3으로 증가된다. 그림 4 (b)에선 연산 +1의 alap를 3으로 감소하며 상태변화 $S_{+1,alap}$가 되나 <1의 alap는 변하지 않는다. 그림 4 (c)는 partition P1에서 $S_{+1,asap}$에 대한 DG로 그림 3 (b)와 비교하면 스테이지 Cstep 1의 DG값이 1/4에서 0으로, Cstep 2에선 1/2에서 2/3으로, 3/2에서 5/3으로, 1/2에서 2/3으로, 그리고 Cstep 5에선 1/4에서 1/3으로 DG가 변한다. 그림 4 (d)는 $S_{+1,alap}$에 대한 DG이며 그림 3 (b)의 DG가 재구성된 것이다. 그림 4 (e)는 칩간 통신 네트에 대한 DG를 나타낸 것으로 새로운 partition으로 DG를 구성할 수 있다.



(a) (b)

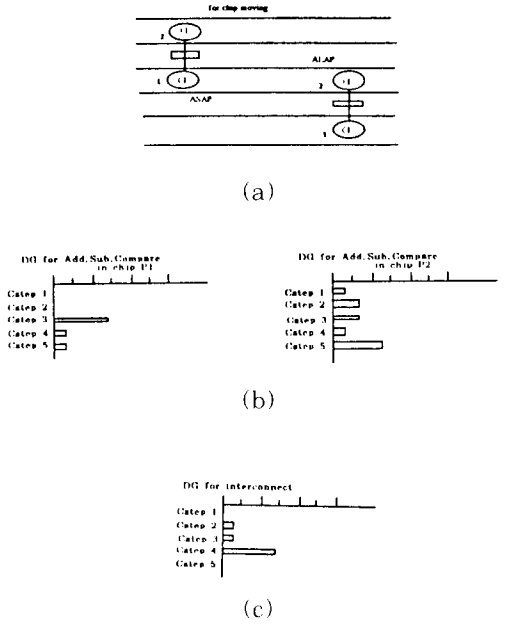
(c) (d)

(e)

그림 4. S+1,asap와 S+1,alap의 DG
(a) S+1,asap에 대한 SFG
(b) S+1,alap에 대한 SFG
(c) S+1,asap에 대한 새로운 DG
(d) S+1,alap에 대한 새로운 DG
(e) 통신 네트에 대한 DG

Fig. 4. DG for S+1,asap and S+1,alap.
(a) SFG for S+1,asap
(b) SFG for S+1,alap
(c) New DG for S+1,asap
(d) New DG for S+1,alap
(e) DG for communication net

그림 5는 그림 3의 상태에서 연산 +1을 다른 partition으로 이동했을 때의 상태변화 S+1,part와 그에 따른 DG를 계산한 예이다. 그림 5 (a)는 연산을 이동한 후의 SFG이고 (b)와 (c)는 새로운 DG 값들이다.



(a)

(b)

(c)

그림 5. S+1,part에 대한 DG,
(a) S+1,part에 대한 SFG
(b) S+1,part에 대한 새 DG 값
(c) 통신 네트에 대한 DG

Fig. 5. DG for S+1,part.
(a) SFG for S+1,part
(b) New DG for S+1,part
(c) DG for commnunication net

목적함수는 EBS 스케줄링에서 사용되는 목적함수를 변형하여 각 칩내의 스테이지에서 연산의 분산정도에 대한 함수와 각 칩으로 분산된 정도에 대한 함수로 표현한다. 식 (1)와 식 (2)은 연산 타입 op가 각 칩내의 스테이지에 분산된 정도를 나타낸 식이다.

식 (1)는 partition c에서 연산 타입 op의 엔트로피를 구하는 식으로 스테이지 엔트로피 H_c로 나타내며, P_{op} (k)는 partition c에서 op 타입 연산이 파이프라인 파티 c 션 k에 할당될 확률값이다. 식 (2)는 각 칩에서 스테이지에 분산된 정도를 나타낸 목적 함수이며 OF_c로 표현한다. 식 (2)의 가중치 $W(op_i)$는 partition c에서 op 타입의 연산 수이다. 통신 네트

에 대한 분산 정도도 식 (2)을 이용하여 모델링하며 통신 네트의 DG는 새로운 partition을 설정하여 구성한다. 식 (2)에서 통신 네트를 위한 partition은 c=0일 때이며, # partiton은 합성하려는 전체 partition 수가 된다. 통신 네트를 위한 통신 모듈의 수와 연산 모듈의 수를 동시에 최적화하는 것은 어려운 문제이며 식 (2)에서 가중치 $W(op_c)$를 다르게 줌으로써 통신 모듈이나 연산 모듈를 최적화시킨다.

$$Stage\ Entropy\ H_s(op_c) = -\sum_{k=0}^{Dif-1} P_{op_c}(k)\ \log P_{op_c}(k) \quad (1)$$

$$Intra\_chip\ \ OF_s = \sum_{c=0}^{\#partition} [\ \Sigma H_s(op_c) * W(op_c)\ ] \quad (2)$$

연산이 각 칩의 스테이지로 균등 분산되며, 칩에 대한 그 연산의 합이 칩으로의 분산 정도를 나타내므로 연산 타입 op에 대해 각 칩으로의 분산정도를 나타내는 함수는 식 (3)과 식 (4)이다. 식 (3)은 가상적인 스테이지에서 연산 타입 op의 엔트로피를 구하는 식이며 칩 엔트로피 H_c로 나타내며, $p_{op,c}(i)$는 칩 partition c에서 op타입의 연산이 스테이지 i에 할당될 확률값이다. 식 (4)는 각 칩으로 연산이 분산된 정도를 나타낸 목적함수로 OF_c로 표현한다. 식에서 가중치 W(op)는 op 타입 연산의 전체 수이다.

$$Chip\ Entropy\ H_c(op) = -\sum_{c=1}^{\#partition} [\ \sum_{i=0}^{max\_stage-1} P_{op,c}(i)\ ] \quad (3)$$

$$Inter\_chip\ \ OF_c = \Sigma H_c(op) * W(op) \quad (4)$$

Partitioning과 스케쥴링 과정에서 사용하는 목적함수는 식 (5)이며 연산의 asap와 alap의 증감과 칩간 이동에 의한 상태변화에 대해 각각의 목적함수를 구한다. 목적함수는 상태변화에 따른 분산 정도를 측정하는 값이며 상태변화로 발생하는 분산 정도에 대한 이득은 식 (6)을 이용한다. 스케쥴링은 상태 변화 중 최대 이득을 주는 것을 선택하며 식에서 상태변화에 대한 이득은 3가지 형태의 상태변화에 대한 목적함수중 최대값과 최소값을 이용하여 직선으로 모델링한다. 각 연산은 스케쥴링 가능 구간이 점진적으로 감소하여 최적의 한 스테이지로 할당이 된다.

$$Objective\ Function = OF_s + OF_c \quad (5)$$

$$Priority\ Function(opn) = (Mx - Mn)/(ALAP_{opn} - ASAP_{opn}) \quad (6)$$

S_{opn} : 연산 opn의 상태변화

$OF(S_{opn}, i)$: i 형태의 상태변화에 대한 목적함수

$Mx = MAX(\ OF(S_{opn,asap}),\ OF(S_{opn,alap}),\ OF(S_{opn,part})\ )$,

$Mn = MIN(\ OF(S_{opn,asap}),\ OF(S_{opn,alap}),\ OF(S_{opn,part})\ )$

Multichip 아키텍춰의 합성에서 발생되는 문제는 칩간 연결되는 통신 네트이다. 통신 네트는 한 칩내의 다른 네트와는 달리 지연시간를 갖는다. 스케쥴링 과정에서 통신 네트는 결정되지 않으며 모든 네트가 스케쥴링 과정에서 통신 네트가 될 수 있다. Mutlichip 아키텍춰에서 통신 네트의 최적화는 중요한 문제로 본 연구에서는 통신 네트의 최적화를 위해 통신 네트도 연산과 같이 DG를 구성 각 스테이지에 균등히 분산시킬 뿐 아니라 통신 네트의 수도 통신 네트의 분산 정도에 대한 함수에 제수로 기여하게함으로 그 수를 최적화한다.

스케쥴링은 각 칩으로 연산이 어느정도 분산된 뒤 수행하면 보다 좋은 결과를 얻을 수 있음을 실험적으로 확인되었다. 이와같은 실험적 결과에 따라 제안된 스케쥴링 과정은 3단계로 수행한다. 첫째 단계에서 각 칩으로 연산이 분산되는 것에 우선순위를 크게 주며, 두번째 단계는 칩간 분산과 스테이지간 분산에 동등한 우선순위를 주는 단계이다. 세번째 단계는 한 칩내에서 각 스테이지에 분산되는 것이 높은 우선순위를 갖게하여 각 연산을 한 스테이지에 할당하게 된다. 세번째 단계는 두번째 단계의 후반기에 자동적으로 형성되는 단계로 특별한 제약이 주어지거나 새로운 제약이 사용되지 않는다. 이는 한 연산의 asap와 alap의 차가 작아지면 partition의 이동에 의한 목적함수보다는 구간의 감소에 의한 목적함수가 크게 되기 때문이다. 알고리듬의 구현시 온도 T의 개념을 도입하였다. 온도 T는 칩간 이동에 대한 목적함수에만 곱하여, 초기 높은 T에서 칩간 이동에 대한 우선순위는 상대적으로 커지게 되며 T가 작아짐에 따라 스케쥴링의 두번째, 세번째 단계를 수행하게 된다. 칩간 이동에 대한 상태변화에 적용되는 목적함수는 식 (5)에 T를 곱한 식 (7)과 같다. 식 (8)은 온도를 나타낸 식으로, N은 SFG의 연산 수를 나타낸다. 식에서 exp 함수는 1 보다 큰 값을 가지며 초기에 높은 T의 값을 갖는 함수로 사용되며, 초기 T는 주어진 SFG의 모든 연산이 최소한 한번은 칩간 이동에 기여한다는 취지에서 연산의 수로 설정한 값이다.

$$Ovjective\ Function\ for\ Move = [OF_s + OF_c] * T \quad (7)$$

$$Temperature\ \ T(n) = \exp(n/N) \quad (8)$$

$$\left\{ \begin{array}{l} N: SFG\text{의 연산 수} \\ n: N, N-1, \ldots, 1 \end{array} \right\}$$

3) Partitioning을 고려한 스케쥴링 알고리듬

Multichip 아키텍춰의 합성에 대한 스케쥴링 알고리듬을 개략적으로 보면 아래와 같다. 칩간 이동에 대한 목적함수를 구한뒤 온도를 곱하는 과정이 삽입되었다.

```
n = number of operations in SFG ;
Repeat until (all operations are scheduled) :
  Calculate temperature T.
  Step 1 : Calculate priority function for each
           operation:
      for each unscheduled node op
          Calculate objective functions for
                    S_op,asap, S_op,alap, and S_op,part.
          Objective function is multiplied by
            temperature T;
          Calculate priority function : end for;
  Step 2 : Schedule or move the operation
                with maximum priority.
  if (n>0) decrease n;
End Repeat ;
```

각 연산의 목적함수를 구하는 단계에서 연산의 asap 증가로 발생된 $S_{op,asap}$와 alap 감소로 발생된 $S_{op,alap}$, 그리고 칩간 이동에 의한 $S_{op,part}$의 세가지 형태의 상태변화에 대한 목적함수를 구하며, 칩간 이동에 대한 목적함수에만 온도 T가 곱해진다. 변수 n은 식 (8)에서 온도 T를 구하는데 사용되는 것으로 초기 연산의 수에서 0까지 감소되며, 0인 상태에서 스케쥴될 수 있는 구간의 감소에 따라 스케쥴링 단계의 두번째와 세번째가 자동으로 형성된다.

2. Multichip 아키텍춰의 모듈할당

SODAS-DSP는 반복적 개선 방식의 모듈할당 알고리듬을 사용하고 있다.[15][16] 모듈할당 과정에서 mux의 입력 수, latch의 수는 연산이 연산 모듈에 할당될 때 결정된다. 데이타 의존 관계가 있는 두 연산이 속하는 스테이지 수 차 만큼의 latch가 두 연산을 수행하는 연산 모듈사이에 사용되고, mux의 입력 수는 연산 모듈에서 수행되는 모든 연산의 연결에 필요한 (출발점,latch의 수)를 원소로 집합을 구성할 때 원소의 개수가 된다. 면적에 대한 비용함수는 식 (9)를 이용하며 α와 β는 각각 mux 입력과 latch에 대한 가중치로 라이브러리에 따라 상대적인 면적을 사용한다.

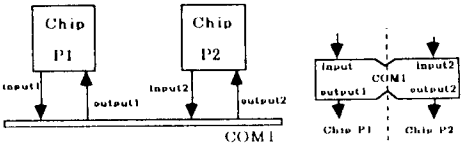
$$\text{비용함수} = (\ \alpha\ *\ \text{mux 입력의 수}\ ) + (\ \beta\ *\ \text{latch의 수}\ ) \qquad (9)$$

모듈할당 알고리듬은 연산자의 수를 스케쥴링의 결과에 따라 결정한 후 같은 타입의 연산에 대해 임의로 초기 할당을 수행한다. 그리고 같은 종류의 연산 모듈에 대하여 각각에 속한 연산 중에서 다른 연산모듈로 이동이 가능한 연산과 서로 교환이 가능한 연산쌍을 찾고 이들 중 교환에 의한 면적의 감소량이 가장 큰 연산쌍을 선택하므로 할당을 개선시킨다.[15][16]

Multichip 모듈할당에선 새로운 제약조건으로 각 칩간 연산의 공유가 이루어질 수 없고, 칩간 결과 전달을 위한 통신 모듈의 할당이 필요하다. 통신 모듈은 칩간 연결되는 버스로 그림 6(a)와 같이 연결되는 두 칩단에 입출력을 위한 두개의 입출력단을 갖는다. 통신 모듈의 할당은 연산의 모듈할당과 동시에 처리하기 위하여 통신 모듈을 그림 6 (b)와 같이 연산 모듈화하여 표현한다.

두 칩 P1, P2에 연결된 통신 모듈 COM1은 칩 P1에 input1과 output1이 연결되고, 칩 P2에 input2와 output2가 연결되어 P1의 결과가 P2로 전달될 경우 input1에서 output2로 연결되는 경로를 구성하게 되며, 반대의 경우 input2에서 output1으로 연결되는 경로를 구성하는 기능을 갖게 된다. 이와같은 통신 모듈의 모델링은 각 칩으로 연결된 입출력단에서 갖게되는 latch와 mux 입력 수를 연산 모듈의 입출력단에서 사용하는 방법으로 계산할 수 있으므로 통신 네트에 대해 연산과 같이 식 (9)의 비용함수를 이용하여 모듈할당을 수행한다. 또한 이러한 모델링은 버스에 대한 제어정보를 유지하는데 유리하다. 즉 버스로의 데이타 이동 패스는 통신 모듈의 입력단에 연결된 mux로 제어된다.

Multichip 모듈할당은 각 칩에서 개별적으로 수행될 수 있으나 제안된 알고리듬에서는 위 제약조건만 삽입하므로 모든 칩의 모듈할당을 동시에 수행한다.



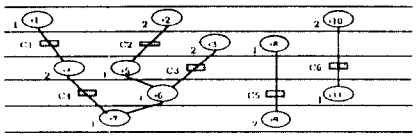
(a) (b)

그림 6. 통신 모듈의 구조
(a) 버스 모듈 (b) 연산 모듈화된 통신 모듈

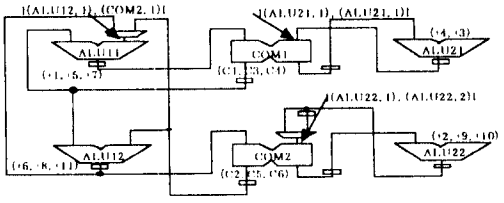
Fig. 6. Structure of communication module.
(a) Bus module
(b) commmunication module nodified to the operation module.

통신 네트는 데이타 전송 지연시간를 갖는 연산으로 스케쥴링 과정에서 각 스테이지에 할당된다. 공유가 가능한 모든 통신 네트는 같은 통신 모듈을 사용한다. 그림 7은 통신 네트와 연산의 공유를 나타낸 예로 (a)는 제어구간에 할당된 연산과 통신 네트를 나타낸 간단한 SFG이다.

그림에서 원은 연산으로 원 왼쪽 아래 수는 연산이 속하는 partition이다. 네모는 통신 네트로 고유번호 Ci를 갖는다. 그림 7 (b)는 스케쥴링 결과 정보를 이용 연산의 공유와 통신 네트의 공유에 대한 예를 나타낸다. Partition P1에서 연산 +1,+5,+7이 다른 스테이지에서 수행되므로 같은 연산자 ALU11를 공유하고 연산 +6,+8,+11은 ALU12에 할당된다. P2에선 +3,+4이 ALU21을 공유하고 +2,+9,+10은 ALU22에 할당되었다. 그리고 통신 네트는 같은 스테이지에서 칩간 통신을 요구하는 것을 제외한 다른 통신 네트와 공유가 되며, 그림에서 같이 C_1, C_3, C_4가 같은 통신 모듈 COM1을 공유하고 C_2, C_5, C_6는 COM2에 할당된다. 그림 7에서 +8과 C5 사이, +10과 C6 사이에선 연산간 스테이지 수 차가 2로 2개의 latch가 사용되며, +2와 C2 사이, 그리고 그외의 연산간의 latch 수는 1개가 사용된다. 그림 7에서 출발점과 latch의 수를 원소로 집합을 만들며 ALU11의 한 입력단에는 { (ALU12,1), (COM2,1) }의 집합을 갖게되어 2-input mux가 사용된다. COM2의 input2 입력단에 { (ALU22,1),(ALU22,2) }의 집합을 갖게되어 2-input mux가 사용되나 COM1의 input2 입력단에는 { (ALU21,1),(ALU21,1) }의 집합으로 mux가 사용되지 않는다. Multichip에서 연산에 대한 공유는 다른 partition에 속하는 연산, 즉 +1과 +3, 사이에선 공유가 될 수 없으며, 각 칩에서의 I/O 포트에 대한 공유는 통신 모듈할당 결과와 같다. 그림 7을 보면 partition P1의 COM1에 대한 I/O 포트는 2,3,4 스테이지에서 사용되며, COM2에 대한 I/O 포트도 같은 스테이지에서 사용되고 있다. 그러므로 partition P1에 대한 I/O 포트 수는 통신 모듈 수와 같다.



(a)



(b)

그림 7. 연산과 통신 모듈 할당
(a) SFG의 스케쥴링 결과
(b) 모듈할당 결과

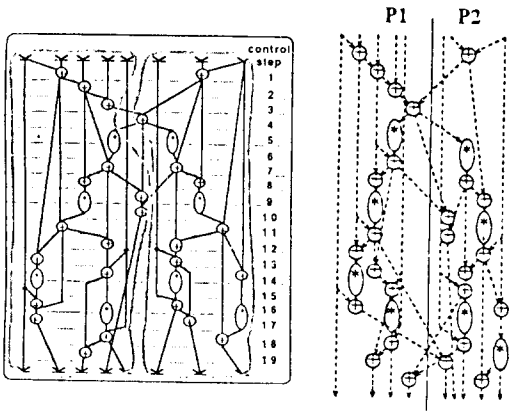
Fig. 7. Allocation for operation and communication module.
(a) Scheduling of SFG
(b) Module allocation

Ⅳ. 실험 결과

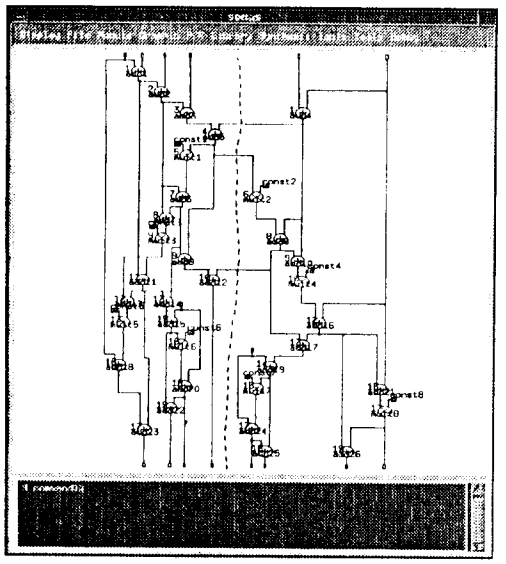
제안된 알고리듬의 구현과 실험은 워크스테이션 SUN SPARK-IPC, UNIX 환경에서 수행하였다.구현된 시스템의 성능 평가를 위해 MCNC benchmark 회로인 5차 엘립틱 웨이브 필터(Fifth Order Elliptical Wave Filter)와 16-point FIR 필터 그리고 AR 필터에 대하여 실험하였다. 본 연구에선 실험을 위해 임계 경로가 최대 스테이지를 넘지 않는 직렬 형태의 초기 파티션이 된 SFG를 입력으로 하였으며, 다른 형태의 초기 파티션에서도 같은 결과를 얻고 있다.

1. 5차 엘립틱 웨이브 필터

5차 엘립틱 웨이브 필터는 덧셈 연산 26개와 곱셈 연산 8개로 구성된다. 합성 과정에서 연산의 지연시간은 상대적인 시간으로 덧셈 연산의 지연시간은 1, 곱셈의 지연시간은 2로 하였다. 그림 8은 APARTY [8]와 Gebotys의 시스템[7]이 최대 스테이지 19, 클럭 주기는 1, 그리고 비파이프라인상에서 두개의 partition으로 합성한 결과이다. 그림 8(a)는 APARTY[8]에서 합성한 결과로 19 스테이지에서 각 칩이 하나의 덧셈기와 하나의 곱셈기를 사용하고 있으나 칩간 통신 지연시간이 계산되지 않았다. 그림 8 (b)는 IP 방식의 Gebotys[7]의 실험 결과로, 칩간 통신 네트 수는 8개고 각 칩에서 한개씩의 덧셈기와 곱셈기를 사용한 최적의 결과이다. 그림 8(c)는 SODAS-DSP 시스템[15][16]의 multichip 아키텍춰 합성기로 실험한 결과이다. 칩간 통신 네트 수는 3개로 Gebotys보다 적은 합성 결과를 보이고 있으며 칩간 통신 모듈의 수는 하나로 최적의 결과와 같다. 그림 8(c)에서 점선은 각 partition을 구분하는 선이다.



(a) (b)



(c)

그림 8. 5차 엘립틱 필터의 스케쥴링 결과
(a) APARTY
(b) Gebotys
(c) SODAS-DSP

Fig. 8. Scheduling results for 5th order elliptic wave filter.
(a) Result by APARTY
(b) Result by Gebotys
(c) Result by SODAS-DSP

표 1은 5차 엘립틱 웨이브 필터의 합성 결과 표로 비파이프라인 구조에서 최대 스테이지 수를 19로 하고 클럭 주기를 1로 한 합성 결과를 Gebotys의 시스템의 합성 결과와 비교한 것이다. 표를 보면 MIPS RC6280에서 실험한 Gebotys의 시스템보다 짧은 합성 시간으로 똑같은 최적의 합성 결과를 보인다.

표 1. 5차 엘립틱 웨이브 필터의 합성 결과 비교
Table 1. Synthesis results for 5th order elliptic wave filter.

(non-pipeline)

	max_stage	+	*	Bus	TIO	CPU Time
Gebotys (P1/P2)	19	2 (1/1)	2 (1/1)	1	9	286 sec
SODAS-DSP (P1/P2)	19	2 (1/1)	2 (1/1)	1	9	52.25 sec

SODAS-DSP multichip 아키텍춰 합성기는 파이프라인 구조를 지원하는 시스템이다. 표 2는 파이프라인 구조로 합성한 데이타로 기존 시스템에서 파이프라인 구조로 합성한 결과 데이타는 없다. 표 2 (a)는 스테이지 구간을 1 clock cycle로 하고 최대 스테이지 수를 19로 한후 DII를 2에서부터 10까지 변화시키며 합성한 결과이다. 표 2 (b)는 스테이지 구간 clock을 2로 하고 최대 스테이지를 10으로 하고 DII를 2에서 10까지 바꾸며 합성한 결과이며, 합성 과정에서의 통신 네트에 대한 가중치로 버스가 덧셈 연산 모듈과 같은 비중을 차지하도록 설정하였다. 표 2 (a)의 결과는 그림 8 (c)의 결과를 DII로 중복시킬 때에 사용되는 연산자의 수보다 작거나 비슷한 수로 거의 최적에 가까운 합성 결과이다. 표 2 (b)에서 DII가 10일 때의 합성 결과는 두 덧셈 연산이 한 제어구간에서 체이닝되기 때문에 각 칩이 덧셈기를 2개 사용한다.

표 2 (a)에서 DII가 4일 때 사용되는 전체 곱셈기의 수가 4이고 DII가 5일 때 6개를 사용하나, 버스는 DII가 4일 때 3개며 DII가 5일 때 2개로 통신 모듈을 연산 모듈로 모델링하여 합성하므로 버스의 감소가 연산자의 수를 증가 시킬 수 있음을 보인다. 표 2를 보면 모든 DII에 대해 각 칩에서 사용하는 연산자의 수는 비슷하다. 이 결과는 각 칩의 면적을 같게하는 제약조건에 부합된다.

그림 9는 최대 스테이지 19, 클럭 주기 1, DII 4의 제약조건으로 합성한 결과다. 칩 P1에는 4개의 덧셈기와 2개의 곱셈기를 사용하며, 칩 P2에서는 3개의 덧셈기와 2개의 곱셈기를 사용하므로 전체 7개의 덧셈기와 4개의 곱셈기를 사용한다. 그림에서 실선은 각 partition을 구분하는 선이다.

표 2. 5차 엘립틱 필터의 합성 결과
(a) 클럭 주기 = 1, 최대 스테이지 = 19
(b) 클럭 주기 = 2, 최대 스테이지 = 10

Table 2. Synthesis results for 5th order elliptic wave filter.
(a) clock period = 1, # of stages = 19
(b) clock period = 2, # of stages = 10

(a)

Util	2	3	4	5	6	7	8	9	10
Adder	15 (7.50)	9 (4.50)	7 (3.50)	7 (3.50)	6 (3.00)	5 (2.50)	4 (2.00)	4 (2.00)	5 (3.00)
Multiplier	5 (5.00)	5 (5.00)	4 (2.00)	6 (3.00)	4 (2.00)	4 (2.00)	3 (2.00)	4 (2.00)	4 (2.00)
Bus	5	4	3	2	1	2	2	1	1
Controller Area (Gate Equivalents)	16.38	11.42	28.53	27.01	23.35	24.12	30.70	38.64	28.86
Min. clock cycle	17.61	11.20	11.63	8.31	5.12	3.95	1.72	1.12	3.68

(b)

DII	2	3	4	5	6	7	8	9	10
Adder	15 (6.70)	10 (3.50)	5 (1.1)	5 (1.10)	6 (3.30)	6 (3.30)	6 (3.30)	6 (3.30)	4 (2.20)
Multiplier	4 (2.20)	4 (2.20)	3 (2.10)	4 (2.20)	4 (2.10)	2 (1.10)	2 (1.10)	2 (1.10)	2 (1.10)
Bus	4	4	3	2	2	2	2	2	2
Controller Area (Gate Equivalents)	8.02	6.52	7.51	12.67	10.05	10.69	13.17	9.95	11.11
Min. clock cycle	22.64	18.30	17.26	14.51	7.55	10.81	6.68	7.32	7.16

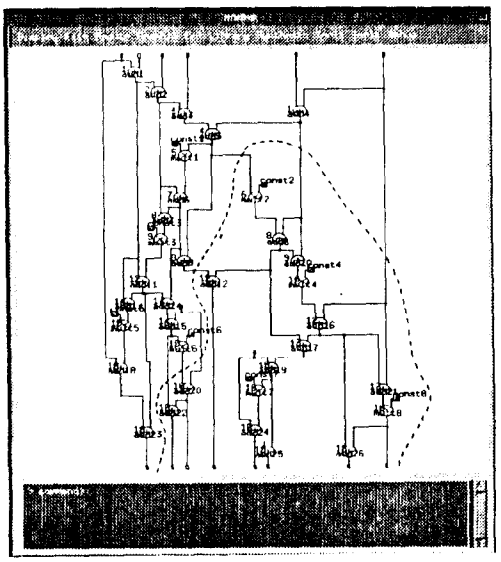


그림 9. 엘립틱 웨이브 필터의 파이프라인 스케쥴링 결과 (최대 스테이지 = 19, DII = 4)

Fig. 9. Pipelined scheduling for 5th order elliptic wave filter.
(# of stages = 19, DII = 4)

2. FIR 필터

FIR 필터는 15개의 덧셈과 8개의 곱셈으로 구성된 회로로, 표 3에서 Gebotys의 시스템과 SODAS-DSP 시스템의 FIR 필터에 대한 합성 결과를 비교하였다. 합성 조건은 클럭 주기가 1, 최대 수테이지 수는 10, 그리고 DII가 10이다. SODAS-DSP의 합성 결과 partition P1에서 2개의 덧셈기와 2개의 곱셈기를 사용하며, P2에서 각 1개씩의 덧셈기와 곱셈기를 사용하였다. 표 3에서 Gebotys의 합성 데이타는 두 칩에서 사용되는 연산자 수를 제시하지 않았으며 두 칩에서 사용되는 수중 큰 수로 테이블을 구성했다.[7] 각 칩에서 사용되는 연산자의 수는 비교할 수 없으며 제안된 시스템이 합성 결과로 각 칩에서 사용하는 연산자 수가 Gebotys의 결과 보다 작거나 같은 수이므로 좋은 결과를 얻고 있음을 알 수 있다. 표 4는 FIR 필터를 Geobotys에서 지원하지 않는 파이프라인 구조로 합성한 결과를 보이고 있다. 기존의 시스템에서 FIR 필터에 대한 파이프라인 구조 합성 결과는 없으며 제안된 시스템의 합성 결과만 나타내고 있다.

표 3. FIR 필터의 합성 결과 비교

Table 3. Synthesis results for FIR filter.

(Non-pipeline)

	max_stage	+	*	Bus	TIO	CPU Time
Gebotys	10	2	2	1	n/a	94 sec
SODAS-DSP (P1/P2)	10	2 (2/1)	2 (2/1)	1	4	6.11 sec

3. AR 필터

AR 필터는 16개의 곱셈과 12개의 덧셈으로 구성된 회로이다. 그림 10은 Parker의 시스템이 입력으로 받아들이는 partition된 AR 필터이며, 표 5 (a)는 클럭 주기를 250 ns, 곱셈 연산의 지연시간을 210 ns, 덧셈 연산의 지연시간을 30 ns, 그리고 DII를 3으로 합성한 결과이다. 표 5 (b)와 그림 11은 SODAS-DSP 시스템이 스테이지와 연산의 지연시간을 상대적 시간으로 설정 클럭 주기는 3, 곱셈 연산은 2, 덧셈 연산은 1로 하고, DII로 3을 갖고 합성한 결과이다. 그림 11에서 partitioning과 스케쥴링을 동시에 수행하는 SODAS-DSP의 합성 결과가 이상적으로 partition된 SFG를 입력으로 합성하는 Parker의 시스템보다 partiton간 통신 네트의 수가 많으나 표 5 (b)와 같이 각 partition에서 덧셈과 곱셈에 대한 연산 모듈을 두개씩 사용한다.

표 4. FIR 필터의 합성 결과 (클럭 주기 = 1, 최대 스테이지 = 10)

Table 4. Synthesis results for FIR filter (clock period=1, # of stages=10).

DII	2	3	4	5	6	7	8	9	10
ALU (+,-,<,=)	8 (3,5)	7 (3,2)	6 (4,2)	4 (3,1)	4 (3,1)	4 (3,1)	4 (3,1)	4 (3,1)	3 (2,1)
Multiplier (*,/,%)	5 (4,1)	5 (4,1)	4 (2,2)	3 (3,0)	3 (3,0)	3 (3,0)	3 (2,1)	3 (2,1)	3 (2,1)
Bus	4	2	2	1	1	1	1	1	1
[illegible]	4.10	4.11	3.64	1.48	6.01	5.68	5.10	1.77	5.95
[illegible]	2.11	1.01	1.12	0.62	1.21	0.69	0.81	0.16	0.11

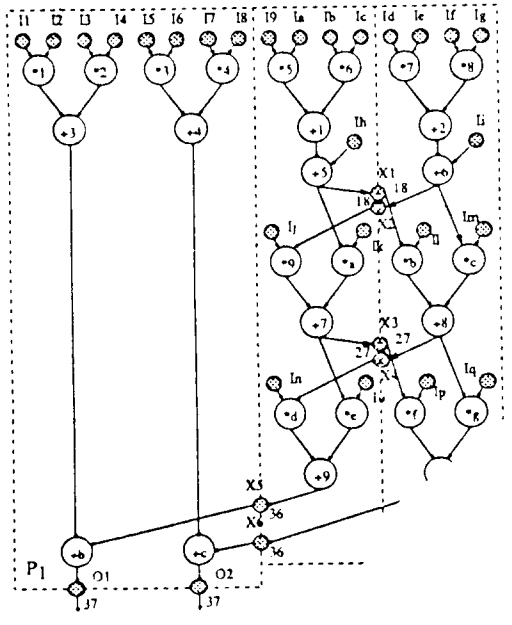


그림 10. Parker의 시스템이 입력으로 사용하는 partition된 AR 필터

Fig. 10. Partitioned AR filter for Parker's system.

표 5. AR 필터의 스케줄링 결과
(a) Parker의 시스템 (b) SODAS-DSP

Table 5. Scheduling results for AR filter.
(a) Result by Parker's system
(b) Result by SODAS-DSP

(a)

Control step	Partition P_1	Partition P_2	Partition P_3
0	I5 I6 I7 I8 *1 *2 *4 *4	Ia Ib Ic *6	Id Ie If Ig *8
1	I2 I3 Ic *2	Ia Ih Ik *3 +1	Id Ie Im *7 +2
2	Ii *1 +3	Ij In Io *a +5	Ip Iq *c +6
3		X1 X3 *9 +7	X2 X3 *b +8
4		X4 *d	X4 *a
5		X3 *e +9	X3 *f +a
7	X5 +b	X5	
8	O1 X6 +c		X6
10	O2		

(b)

Control Step	Partition 1	Partition 2	Partition 3
1	mult1	mult3	mult2
2	add1 mult9	mult4	
3	add3 mult11		
4	add7 mult10	add2 mult12	mult6
5		add4 mult8	
6		add6 mult7	mult5
7			
8	mult13	add8	add5 mult15
9	mult14		add10 mult16
10	add9		add11 add12

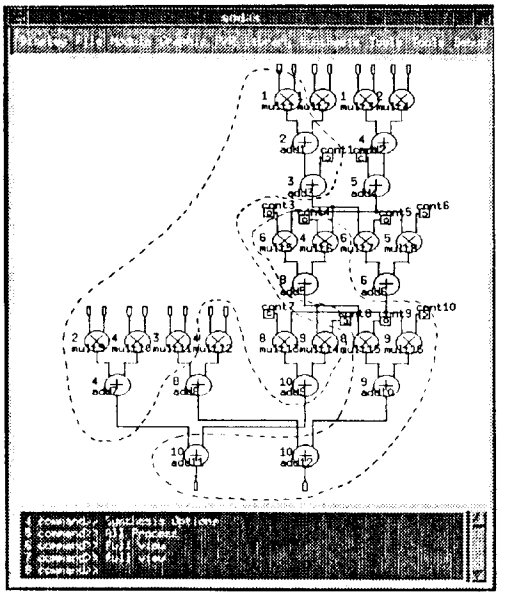


그림 11. SODAS-DSP의 AR 필터 스케줄링 결과

Fig. 11. Scheduling result for AR filter by SODAS-DSP.

V. 결 론

본 논문은 fixed DII (Data Initiation Interval) [12]를 갖는 파이프라인 구조를 multichip으로 합성하는 방법을 제안하며, 효과적인 합성을 위해 연산의 스테이지에 분산되는 정도와 각 칩으로 분배되는 정도를 목적함수로 정의하고 목적함수의 미분계수를 우선순위 함수로 사용하여 점진적으로 연산을 각 칩의 스테이지에 균등히 배분되도록 하는 partitioning과 스케줄링을 동시에 수행하는 방법을 제안하였다. 또한, multichip 아키텍춰에 대해 연결구조로 사용하는 mux 입력의 수와 latch의 수를 비용함수로 반복적 개선 방식의 모듈할당 알고리듬을 사용하며, 각 칩에서 연산의 공유뿐만아니라 칩간 통신 네트 공유도 모든 칩에서 동시에 수행한다.

제안된 알고리듬은 partitioning과 스케줄링을 동시에 수행하는 휴리스틱 알고리듬으로 benchmark

회로에 대한 합성시간이 빠르고 합성된 결과도 기존의 시스템[7][8]과 비교하여 최적의 결과를 얻는다. 또한 제안된 알고리듬은 파이프라인 구조를 지원하고 있으며 각 파이프라인 조건에서 좋은 합성 결과를 얻는다. 합성된 결과에서 각 칩에 사용되는 모듈의 수가 비슷하므로 효율적인 multichip 하드웨어를 생성하며, 시간 제약조건하에서 스테이지 구간·clock, 최대 스테이지 수, DII를 제약조건으로 만족시키며 각 칩과 전체 사용하는 모듈을 최적화한다. 면적 제약조건하에서 각 칩에서 사용하는 모듈의 개수, 칩간 통신 모듈의 개수, 전체 사용가능한 모듈의 개수등을 제약조건으로 합성을 수행하도록 확장되야 하며 최적의 스테이지 수를 구하는 연구가 계속되어야 한다.

SODAS-DSP는 각 칩의 입출력 핀이나 버스의 수를 제약 조건으로 하는 합성은 지원하지 못하며, 같은 우선 순위를 갖는 연산이 존재할 경우 알고리듬상 첫번째 선택된 연산을 스케줄링한다. 추후과제로 각 칩의 입출력 핀과 버스의 수를 제약한 합성에 대한 연구와 같은 우선 순위를 갖는 연산의 선택 알고리듬에 대한 연구가 계속 되어야 한다.

參 考 文 獻

[1] D. D. Gajski, Silicon Compilation, Addison Wesley Pub. : Reading, Mass., 1988.

[2] K. Hwang, A. E. Casavant, "Scheduling and hardware sharing in pipelined data paths," in Proc. ICCAD, pp. 24-27, Nov. 1989.

[3] M. C. McFarland, A. C. Parker, R. Camposano, "The high level synthesis of digital systems," Proceedings of IEEE, Vol. 78, No. 2, pp. 301-318, Feb. 1990.

[4] M. C. McFarland, A. C. Parker, R. Camposano, "Tutorial on high level synthesis," in Proc. 25th DAC, pp. 330-336, June 1988.

[5] D. E. Thomas, E. E. Lagnese, R. A. Walker, Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench, Kluwer Academic Publishers : Boston, Mass., 1990.

[6] J. Allen, F. Catthoor, "Architecture driven synthesis technique for VLSI implementation of DSP algorithms," Proceedings of IEEE, Vol. 78, No. 2, pp. 319-335, Feb. 1990.

[7] C. H. Gebotys, "Optimal synthesis of multichip architectures," in Proc. ICCAD, pp. 238-241, Nov. 1992.

[8] E. D. Lagnese, D. E. Thomas, "Architectural partitioning for system level design," in Proc. 26th DAC, pp. 62-67, June 1989.

[9] E. D. Lagnese, D. E. Thomas, "Architectural partitioning for system level synthesis of integrated circuits," IEEE Trans. CAD, Vol. 10, No. 7, pp. 847-860, July 1991.

[10] Y. H. Hung, A. C. Parker, "High-level synthesis with pin constraints for multiple-chip designs," in Proc. 29th DAC, pp. 231-234, June 1992.

[11] K. Kucukcakar, A. C. Parker, "CHOP: A constraint-driven system-level partitioner," in Proc. 28th DAC, pp. 514-519, June 1991.

[12] P. M. Kogge, The Architecture of Pipelined Computers, McGraw-Hill, 1981.

[13] S. Prakash, A. C. Parker, "Synthesis of application-specific multiprocessor architectures," in Proc. 28th DAC, pp. 8-13, June 1991.

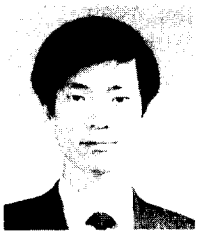
[14] R. Gupta, G. De Micheli, "Partitioning of functional modules of synchronous digital systems," in Proc. ICCAD, pp. 216-219, Nov. 1990.

[15] 전 홍신, 황선영, "디지탈 신호처리를 위한 파이프라인 데이타패스 합성 시스템의 설계", 대한 전자공학회 논문지, 30-A권 6호, pp. 486-494, 1993년 6월

[16] H. S. Jun, S. Y. Hwang, "Design of a pipeline datapath synthesis system for digital signal processing," IEEE Trans. VLSI Systems, Vol. 2, 1994. (to be appeared)

[17] P. G. Paulin, "Force Directed Scheduling for the Bahavioral Synthesis of ASIC's," IEEE Trans. CAD, Vol. 8, No. 6, pp. 661-679, June 1989.

著 者 紹 介



朴 在 煥(正會員)

1992年 2月 서강대학교 전자공학과 졸업. 1994年 2月 서강대학교 전자공학과 석사 졸업. 1994年 2月 ~ 현재 삼성전자 반도체 근무. 주관심 분야는 CAD시스템, VKSI 설계 등임.

黃 善 泳(正會員)

1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국 과학원 전기 및 전자 공학과 공학 석사 취득. 1986年 10月 미국 Stanford 대학 공학 박사 학위 취득. 1976年 ~ 1981年 삼성 반돌페 주식회사 연구원. 1986年 ~ 1989年 Stanford 대학 Center for Integrated Systems 연구소 연구원, Fairchild Semiconductor Palo alto Research Center 기술 자문. 1989年 3月 ~ 현재 서강 대학교 전자 공학과 부교수. 주관심 분야는 CAD 시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임.

田 弘 信(正會員) 第 30卷 1篇 第 6號 參照

1989年 2月 서강대학교 전자공학과 졸업. 1991年 2月 서강대학교 전자공학과 석사. 1991年 3월 ~ 현재 서강대학교 전자공학과 박사과정. 주관심 분야는 CAD 시스템 등임.