The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 3, pp.137-144, Jun. 30, 2016. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2016.16.3.137
JIIBC 2016-3-19

새로운 멀티프로세서 디자인을 위한 상위수준합성 시스템의 회로 복잡도 최적화 ILP 알고리즘

A Circuit Complexity Optimization ILP Algorithm of High-level Synthesis System for New Multiprocessor Design

장정욱*, 인치호**

Jeong-Uk Chang*, Chi-Ho Lin**

요 약 본 논문에서는 새로운 멀티프로세서 디자인을 위한 상위 수준 합성 시스템의 회로 복잡도 최적화 ILP 알고리즘을 제안하였다. 상위수준 합성에서 가장 중요한 연산자의 특성과 데이터패스의 구조를 분석하고, 멀티사이클 연산의 스케줄링 시 가상연산자 개념을 도입함으로써, 멀티사이클 연산을 구현하는 연산자의 유형에 관계없이 공통으로 적용시킬 수 있는 ILP 알고리즘을 이용하여 증명하였다. 기술된 알고리즘의 스케줄링 성능을 평가하기 위하여, 표준벤치마크 모델인 5차 디지털 웨이브필터에 대한 스케줄링을 행한 결과, 기존의 데이터패스 스케줄링 결과와 정확하게 일치함으로서, 제시된 모든 ILP 수식이 정확하게 기술되었음을 알 수 있었다.

Abstract In this paper, we have proposed a circuit complexity optimization ILP algorithm of high-level synthesis system for new multiprocessor design. We have analyzed to the operator characteristics and structure of datapath in the most important high-level synthesis. We also introduced the concept of virtual operator for the scheduling of multi-cycle operations. Thus, we demonstrated the complexity to implement a multi-cycle operation of the operator, regardless of the type of operation that can be applied for commonly use in the ILP algorithm. We have achieved is that standard benchmark model for the scheduling of the 5th digital wave filter, it was exactly the same due to the existing datapath scheduling results.

Key Words : ILP Algorithm, Circuit Complexity, High-level synthesis, Multiprocessor

Ⅰ. 서 론

컴퓨터를 이용하여 디지털 시스템을 설계하고자 하는 자동설계 기술은 대단히 복잡하고도 방대한 설계과정을 거쳐야 하므로 전 설계과정을 다음과 같은 3단계의 합성과정으로 구분지어, 보다 체계적으로 설계과정의 효율을 극대화 시킨다. 즉, 전 설계과정은 동작영역 합성과정과 논리영역 합성과정 및 하위영역 합성 과정으로 구분되어, 각 합성과정의 입력정보로서 한 단계 높은 상위영역의 합성과정으로부터 얻어지는 출력정보를 취한다.[1-3]

상위수준합성의 각 단계 중, 가장 핵심적인 것은, 시스템의 동작알고리즘으로부터 비교적 구체적인 데이터패스의 구조적인 형태를 추출해 내는 데이터패스 합성단계이다. 따라서 상위수준합성에 관한 대부분의 연구는 데이터패스 합성단계의 스케줄링과 하드웨어할당에 초점을 두고, 이의 효율성과 성능을 개선하고자 노력하였다.

*준회원, 세명대학교 컴퓨터학부
**정회원, 세명대학교 컴퓨터학부
접수일자 : 2016년 5월 20일, 수정완료 : 2016년 6월 8일
게재확정일자 : 2016년 6월 10일

Received: 20 May, 2016 / Revised: 8 June, 2016 / Accepted: 10 June, 2016
**Corresponding Author: ich410@semyung.ac.kr
School of Computer, Semyung University, Korea

가장 기본적인 스케줄링 기법은 하드웨어 자원과 동작속도의 제약이 가해지지 않은 상태에서 행할 수 있는 ASAP(As Soon As Possible) 스케줄링[4]과 ALAP(As Late As Possible) 스케줄링[5] 기법과 자원의 제약이 가해진 상태에서는 변형된 ASAP 스케줄링 기법[6]이나, 리스트(list) 스케줄링 기법[7]을 사용한다.

본 논문에서는 상위영역합성에서의 여러 제한조건을 개별적인 변수로 간주한 후, 이들 변수 사이의 관계에 따라 스케줄링 문제를 다원 일차방정식으로 전개시켜, 원하는 목적함수를 최소/최대화 시키는 ILP(Integer Linear Programming)를 이용하여 새로운 멀티프로세서 디자인을 위한 회로의 복잡도를 최적화하기 위한 스케줄링 알고리즘을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제안한 스케줄링 방법의 설계 및 구현에 대하여 기술한다. 3장에서는 구현된 스케줄링 알고리즘을 적용하여 얻어진 결과에 대하여 기술한다. 끝으로 4장에서는 결론을 맺는다.

II. 유형에 따른 ILP 스케줄링 알고리즘

1. 스케줄링의 절차

데이터패스 합성과정 중, 가장 중요한 것은 스케줄링 과정과 하드웨어 할당 과정이다. 스케줄링 과정에서는 DFG(data flow graph) 상의 연산을 적당한 제어스텝에 할당시키고, 하드웨어 할당과정에서는 연산자, 레지스터 및 데이터 이동로를 데이터패스에 할당시킨다. 이러한 스케줄링과 하드웨어 할당과정은 데이터 의존성, 제어스텝의 시간간격, 면적 비용, 성능이 만족될 수 있도록 수행되어야만 한다.

그림 1은 임의의 DFG의 한 부분을 나타낸 것으로서, 연산 '+1'은 제어스텝-1 또는 제어스텝-2 중, 어느 위치에 있어도 이들 각 연산 간의 데이터 의존성은 그대로 유지된다. 따라서 연산 '+1'의 할당을 제어스텝-1로 할 것인가, 또는 제어스텝-2로 할 것인가를 결정하는 것이 바로 스케줄링 문제이다. 이러한 구조는 이들 연산과 래치를 그림 2의 프로세서와 래치에 각각 대비시키면 기본적으로 그림 2와 같은 파이프라인 구조와 동일하다는 것을 알 수 있다.

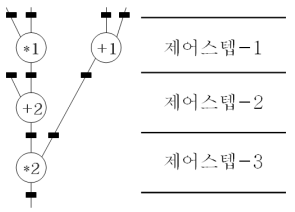


그림 1. DFG 모델
Fig. 1. The model of DFG

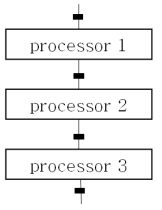


그림 2. 파이프라인 모델
Fig. 2. The model of pipeline

2. 스케줄링을 위한 기본형 ILP 관계식

DFG 상의 연산 O_i를 제어스텝 C_j에 할당시키는 과정은, 연산 O_i를 제어스텝 C_j에 사상(mapping)시키는 과정을 의미한다. 이를 위해서 연산 O_i와 제어스텝 C_j는 일정한 관계를 유지하여야만 한다. 즉, 연산 간의 선후관계와 연산이 할당될 수 있는 제어스텝의 범위로 표현되는 연산 O_i의 속성은, 일정한 관계식 R로서, 임의의 시간간격인 제어스텝 C_j의 속성에 결합될 수 있어야만 한다. 관계식 R은 연산 O_i와 이들의 할당에 소요되는 제어스텝의 개수 및 임의의 제어스텝에 할당될 수 있는 연산의 개수와의 관계를 의미한다. 이러한 조건을 ILP 관계식으로 표현하기 위한 기호를 아래와 같이 정의한다.

n_t : 연산유형 t인 연산의 개수

t_T : 사이클 타임-제어스텝의 시간간격

d_i : 연산 O_i의 지연시간

D_i : 사이클타임 수로 표시되는 연산 O_i의 지연시간 ($= \lceil d_i/t_T \rceil$)

S_j, L_i : ASAP/ALAP 스케줄링에 의해 결정되는 연산 O_i의 상/하한 제어스텝

$O_i(k)$: 연산 O_i의 k번째 요소

P_{ik} : $O_i(k)$가 할당되는 제어스텝

$X(i,k)$: $O_i(k)$가 제어스텝-P_{ik} 할당 되었을 때, 값이 1이 되는 정수변수

(1) 선후관계식

연산 O_i의 최 근접 후행연산을 O_j라 할 때, 연산 O_i와 연산 O_j 사이에는 다음의 식(1)이 성립한다.

$$\sum_{k=1}^{r_i}\{P_{ik}\times X(i,k)\}-\sum_{h=1}^{r_j}\{P_{jh}\times X(j,h)\}\le -D_i \quad \cdots(1)$$

관계식(1)은 연산 O_j의 각 요소가 할당될 수 있는 제어스텝은, 연산 O_i가 할당된 제어스텝 보다도 D_i-사이클타임 이상 이격된 제어스텝이어야만 한다.

(2) 범위관계식

연산 O_i는 상한 제어스텝 S_i와 하한 제어스텝 L_i의 범위에 걸쳐, 오직 한번만 할당되어야하기 때문에, 연산의 각 요소 사이에는 다음의 식(2)이 성립한다.

$$\sum_{k=1}^{ri} X(i,k)=1,(i=1,2,...,n_t) \quad \cdots\cdots\cdots(2)$$

(3) 시간관계식

연산 O_i의 각 요소가 할당되는 제어스텝-P_{ik}는 제어스텝의 최대치 $T_{\max}$을 초과해서는 안 된다. 따라서 후행연산이 존재치 않는 모든 연산 O_i의 요소 $O_i(k)$와 제어스텝의 최대치 $T_{\max}$ 사이에 다음의 식(3)이 성립할 경우, DFG 상의 연산 O_i의 각 요소가 할당되는 제어스텝은 제어스텝의 최대치 $T_{\max}$을 결코 초과하지 않는다.

$$\sum_{k=1}^{ri}\{P_{ik}\times X(i,k)\}\le T_{\max}-Di+1,(i=1,2,...n_t) \quad \cdots(3)$$

(4) 자원관계식

특정 제어스텝-P에 할당될 수 있는 연산유형 t인 연산의 총 개수는 식(4)과 같이 사용가능한 자원의 개수 N_t을 초과해서는 안 된다.

$$\sum_{i=1}^{n_t} X(i,P-Si+1)\le N_t,(p=1,2,...,T_{\max}-D_i+1) \quad \cdots(4)$$

3. 스케줄링을 위한 확장형 ILP 관계식

스케줄링은 스케줄링의 목적과 설계상의 제한조건에 따라, 성능제약 스케줄링과 비용제약 스케줄링, 그리고 연산자의 연산유형 및 데이터패스의 여러 조건에 따라, 크게 파이프라인 스케줄링과 비파이프라인 스케줄링으로 구분된다. 따라서 이들 여러 경우에 각기, 또는 일괄적으로 적용시킬 수 있고, 또한 연산의 지연시간, 사용하고자 하는 연산자의 연산유형, 구성하고자 하는 데이터패스의 형태에 따라, 각기 적용시킬 수 있는 ILP 알고리즘이 요구된다.

(1) 멀티사이클 연산의 스케줄링

멀티사이클 연산은 비파이프라인형 연산자나 또는 파이프라인형 연산자에 의해 수행될 수 있다. 즉, 비파이프라인형 연산자는 일단 연산의 수행이 시작되면, 해당 연산을 마칠 때 까지 다른 연산의 수행에 할당될 수 없는 반면, 파이프라인형 연산자는 연산의 수행 도중에도 다른 연산의 수행에 할당될 수 있기 때문에, 보다 많은 자원 공유를 이룰 수 있다. 그림 3과 같이 지연시간이 D_i인 연산 A가 제어스텝-P에 할당되는 경우를 가정한다.

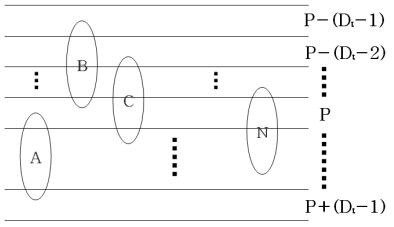


그림 3. 제어스텝-P에서의 멀티사이클 연산의 중복
Fig. 3. The redundancy of a multi-cycle operation in the control step-P

연산 A가 제어스텝-P에 할당되기 위해서는 제어스텝-P에 존재하는 동일유형의 연산의 개수가 사용가능한 자원의 개수를 초과해서는 안 된다. 즉, 제어스텝 $P-(D_i-1)$에 할당된 연산 B, 제어스텝 $P-(D_i-2)$에 할당된 연산 C, ... , 제어스텝-$(P-1)$에 할당된 연산 N 및 제어스텝-P에 할당된 연산 A의 총 합은 사용가능한 자원의 개수 N_t보다 적어야만 한다. 따라서 연산유형 t인 연산 O_i의 요소 $O_i(k)$가 제어스텝-P에 할당되기 위해서는 다음의 조건식(5)이 만족되어야만 한다.

$$\sum_{i=1}^{n_t} X(i,(P-S_i+1)-(D_i-1))+\sum_{i=1}^{n_t} X(i,(P-S_i+1)-(D_i-2))+$$
$$\cdots+\sum_{i=1}^{n_t} X(i,(P-S_i+1)-1)+\sum_{i=1}^{n_t} X(i,(P-S_i+1))\le N_t \quad \cdots(5)$$

이를 정리하면, 다음과 같은 식(6)으로 요약된다.

$$\sum_{j=0}^{D_i-1}\sum_{i=1}^{n_t} X(i,(P-S_i+1)-j)\le N_t \quad \cdots\cdots(6)$$

파이프라인형 연산자는, 연산자 내에서 이전 데이터의 연산이 수행되고 있는 동안에도 새로운 데이터가 동일 연산자 내에 유입될 수 있는 기능을 보유하고 있다. 이미 연산이 진행되고 있는 입력 데이터와 새로이 유입되는 데이터 간의 시차인 회전지연시간이 l-사이클타임이고, 지연시간이 D_i인 연산자는, $l \le D_i$의 조건 하에서 매 l-사이클타임마다 새로운 연산을 수행할 수 있다. 그림 4와 같이, 지연시간이 D_i인 연산을 회전지연시간이 2-사이클타임이고 지연시간이 3-사이클타임인 비파이프라인형 연산자로 구현시키는 경우를 가정한다.

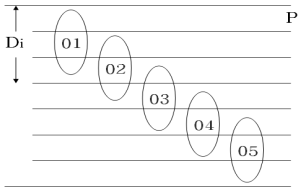


그림 4. 파이프라인형 연산자로 수행되는 멀티사이클 연산의 자원공유

Fig. 4. The multi-cycle operations on the shared resource is performed in pipelined operator

제어스텝-P의 연산 O_1의 수행에 할당된 연산자는 2-사이클타임 이후에 새로운 연산의 수행에 할당될 수 있으므로, 해당 연산자에는 연산 O_3, O_4, O_5 중 임의의 연산이 할당되어도 무방하고, 연산 O_2을 수행하는 연산자에는 2-사이클타임 이후의 제어스텝에 존재하는 연산 O_4, O_5가 할당될 수 있다. 결과적으로, 파이프라인형 연산자는 제어스텝-$P(1 \le P \le l)$를 기준으로 제어스텝-$(P+1)$ 이후의 모든 제어스텝에 존재하는 연산이 공유할 수 있기 때문에, 연산유형 t인 연산의 수행에 필요한 파이프라인형 연산자의 개수는 연산의 공유가 가능한 것끼리 묶어, 다음 식(7)과 같이 구분 지을 수 있다.

$$N_t = N_{t1} + N_{t2} + \cdots + N_t l \qquad \cdots\cdots\cdots(7)$$

$N_{tp}(p = 1, 2, \cdots, l)$ 는 제어스텝-P와 제어스텝-$(P+1)$ 이후의 제어스텝에 존재하는 연산유형 t인 연산이 공유할 수 있는 파이프라인형 연산자의 최대치를 나타낸다. 따라서 파이프라인형 연산자에 의한 멀티사이클 연산의 스케줄링에서는 자원관계식이 다음의 식(8a), 식(8b)과 같이 표현된다.

$$\sum_{i=1}^{n_t} X(i, p - S_i + 1) \le N_{tp}, (p = 1, 2, \ldots, \ell) \qquad \cdots(8a)$$

$$\sum_{i=1}^{n_t} X(i, q - S_i + 1) \le N_{np}, (q = p + \ell, p + \ell + 1, \ldots, T_{\max} - D_i + 1) \qquad \cdots(8b)$$

(2) 가상연산자

멀티사이클연산의 효율적인 스케줄링을 위하여, 가상연산자를 아래와 같이 정리하여 사용한다.

(정 리 1)

지연시간이 D_i이고 제어스텝-P에서 연산의 수행을 시작하는 임의의 멀티사이클연산 O_i를, 비파이프라인형 연산자 F_i로 구현하는 경우, 연산자 F_i의 n차 가상연산자는 제어스텝-$(P+n)$에서 연산의 수행을 시작하는 지연 시간이 1-사이클타임인 연산자이다.

(정 리 2)

지연시간이 D_i-사이클타임이고, 제어스텝-P에서 연산의 수행을 시작하는 임의의 멀티사이클 연산 O_i를, 회전지연시간이 l-사이클타임인 파이프라인형 연산자 F_i로 구현하고자 하는 경우, 연산자 F_i의 n차 가상연산자는 제어스텝-$(P+nl)$에서 연산의 수행을 시작하는 지연시간이 l-사이클타임인 비파이프라인형 연산자이다.

(증 명)

연산자 F_i에 입력된 i-번째 데이터의 연산수행이 시작된 후, $(n \times l)$-사이클타임 뒤에는, $(i+n)$-번째의 데이터가 새로이 연산자 F_i에 입력된다. $D_i > (n+1)$인 경우, 연산자 F_i는 i-번째부터 $(i+n)$-번째 가지의 모든 입력데이터를 동시에 처리하여야만 한다. 그러므로 $(i+1)$-번째 입력되는 데이터를 처리하는 가상연산자를 1차 가상연산자, $(i+n)$-번째 입력되는 데이터를 처리하는 가상연산자를 n차 가상연산자라고 명명할 때, 제어스텝-P에서 연산의 수행을 시작하는 파이프라인형 연산자의 n차 가상연산자는 제어스텝-$(P+nl)$에서 연산의 수행을 시작하는 비파이프라인형 연산자이다. 따라서 가상연산자 개념에 따른 자원 관계식을 식(9)과 같이 표현할 수 있다.

$$\sum_{j=0}^{A-1} \sum_{i=1}^{n_t} X(i, (P - S_i + 1) - j) \le N_t \qquad \cdots\cdots(9)$$

(3) 기능적 파이프라이닝

기능적 파이프라이닝에서는 데이터패스로 하여금 파이프라인 기능을 보유하도록 하는 방식이다. 따라서 회

전지연시간이 l-사이클타임이고, 지연시간이 D_i인 하나의 파이프라인형 연산자와 동일하게 동작한다. 기능적 파이프라인 구성을 위한 제어스텝-P의 자원관계식은 아래의 식(10)과 같이 표현된다.

$$\sum_{j=0}^{qi}\sum_{i=1}^{n_t} X(i,(P-S_i+1)+j\ell) \le N_t \quad \cdots\cdots(10)$$

(정 리 3)

파이프라인형 데이터패스 스케줄링에서, 멀티사이클 연산을 비파이프라인 연산자로 수행시키고자 하는 경우, 최소한 한 개 이상의 자원공유를 이루기 위해서는 연산의 지연시간 D_i와 데이터패스의 회전지연시간 l과의 사이에는 다음 식(11)과 같은 조건이 만족되어야만 한다.

$$2\times D_i \le l \quad \cdots\cdots(11)$$

(증 명)

구간 $i[\mathrm{mod}-l]$에서 지연시간이 D_i인 연산의 수행을 시작한 비파이프라인 연산자는 구간 $(i+D_i-1)[\mathrm{mod}-l]$에서 연산의 수행을 종료하기 때문에, 최소한의 자원공유를 이루기 위해서는 구간 $(i+D_i)[\mathrm{mod}-l]$에서 또 다른 연산의 수행에 할당될 수 있어야만 한다. 또한 구간 $(i+D_i)[\mathrm{mod}-l]$에서 연산의 수행을 시작한 비파이프라인 연산자는 구간 $(i+2\times D_i-1)[\mathrm{mod}-\ell]$에서 연산의 수행을 종료하기 때문에, 해당 연산자가 다른 연산의 수행에 할당되어서는 안 된다. 따라서 구간 $(i+2\times D_i-1)[\mathrm{mod}-\ell]$이 구간 $i[\mathrm{mod}-l]$과 중복되거나, 또는 이를 초과하는 경우는 자원의 충돌이 발생하게 되므로 자원의 충돌을 방지하기 위해서는 다음의 조건이 만족되어야만 한다.

$$i[\mathrm{mod}-l] > (i+2\times D_i-1)[\mathrm{mod}-\ell]$$

(정 리 4)

연산의 지연시간 D_i와 파이프라인형 데이터패스의 회전지연시간 l과의 관계가 $2\times D_i \le l$ 인 경우는, 멀티사이클 연산의 개수 n과 이의 수행에 필요한 비파이프라인 연산자의 최소치 $N_{\min}$과의 사이에는 다음 식(12)의 관계식이 성립한다.

$$N_{\min} = \lceil D_i/l \rceil \times n \quad \cdots\cdots(12)$$

(증 명)

$2\times D_i \le l$ 인 조건하에서, n개의 멀티사이클 연산의 수행에 필요한 비파이프라인 연산자의 최소치 $N_{\min}$은 l값의 범위에 따라 다음과 같이 구분되고, l 값의 범위가

$2\times D_i > l \ge D_i$ 인 경우는 $N_{\min} = n$,
$D_i > l \ge D_i/2$ 인 경우는 $N_{\min} = 2n$,
$D_i/2 > l \ge D_i/3$ 인 경우는 $N_{\min} = 3n$,

로 각각 표시할 수 있으므로, 일반적으로 l의 값이 $D_i/(k-1) > l \ge D_i/k$인 범위에서는 $N_{\min} = k\times n$이 된다. 이러한 l값을 만족하는 k값의 범위는 $D_i/l > k \ge (D_i/l)+1$로 표시된다. k는 언제나 양의 정수 값을 취하므로, 이러한 k값의 범위는 $k = \lceil D_i/l \rceil$ 로 압축하여 표현할 수 있다. 따라서 n개의 멀티사이클 연산의 수행에 필요한 비파이프라인 연산자의 최소치 $N_{\min}$은 $N_{\min} = k\times n = \lceil D_i/l \rceil \times n$으로 표현된다.

(4) 조건부 자원공유

동일 연산자로 하여금 여러 연산을 수행도록 하는 자원공유는, 서로 다른 제어스텝에 할당되는 연산들 사이에서 이루어지는 무조건 자원공유와 조건 분기의 상호배타적 연산들 사이에서 이루어지는 조건부 자원공유로 분류된다.

$$\sum_{i=1}^{w} X(mi,(p-S_i+1)) \le w \quad \cdots\cdots(13a)$$

$$\sum_{i=1}^{q} X(\ni,(p-S_i+1)) + X(mj,(p-S_j+1)) \le N_t\ (j=1,2,\ldots,w) \quad \cdots(13b)$$

관계식(13a)는 연산집합 O_m의 모든 여산 O_{mi}는 동일 제어스텝에 할당 될 수 있다는 것을 의미하며, 관계식(13b)는 연산 O_{mi}의 수행에는 오직 한 개의 연산자가 필요하다는 것을 의미한다.

그림 5와 같이 조건부 분기연산이 포함된 DFG 모델에 대한 자원 관계식을 기술한다. 모델의 조건부 분기연산 중, 상호배타적인 연산의 집합을 구하기 위한 방법으로서는 Node Coloring 알고리즘[8]을 이용하였다.

그림의 각 연산 우측에 표시된 코드가 Node Color로서, 이로부터 추출되는 각 제어스텝에서의 상호배타적 연산의 집합 O_m과 O_m의 여집합 O_n은 다음과 같다.

제어스텝-2 : $O_m = \{-2, -3\}$

$O_n = \{+2, -1, +3, +4, -4\}$

제어스텝-3 : $O_m = \{+3, +5, +6\}$

$O_n = \{+2, -1, -2, +4, -4, -5\}$

제어스텝-4 : $O_m = \{-5, -6\}$

$O_n = \{+4, -4, +5\}$

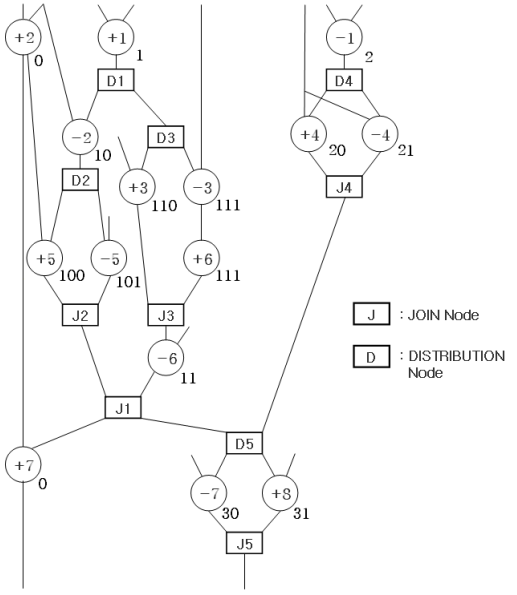


그림 5. 조건부 분기연산이 포함된 DFG 모델
Fig. 5. The DFG model that includes a conditional branch operation

따라서 식(13)로부터 구해지는, 조건부 자원공유를 위한 자원관계식은 표 1과 같이 기술된다.

4. ILP 기반의 데이터패스 스케줄링

제안한 데이터패스 스케줄링은 확장형 ILP 방법 중 하나로, 비용제약 스케줄링 기법을 적용하여 자원의 수가 제한된 조건 하에서 주어진 제한조건을 만족하는 최대성능의 스케줄링을 말한다.

사용가능한 자원의 수가 제한된 조건에서, 동일유형의 연산을 전 제어스텝에 걸쳐 균일하게 분포시키기 위해서는, 연산이 균일하게 분포될 수 있는 제어스텝 수의 상/하한을 결정한 후, 제어스텝의 수를 최소화 시키는 방향으로 스케줄링을 행한다.

표 1. 조건부 자원공유를 위한 자원관계식
Table 1. The resource relationship for the conditional resource sharing

1 : X(+1,1) + X(+2,1) ≤ N_+
X(-1,1) ≤ N_-
2 : X(+2,2) + X(+3,1) + X(+4,1) ≤ N_+
X(-2,1) + X(-3,1) ≤ 2
X(-1,2) + X(-4,1) + X(-2,1) ≤ N_-
X(-1,2) + X(-4,1) + X(-3,1) ≤ N_-
3 : X(+2,3) + X(+4,2) + X(+3,2) ≤ N_+
X(+2,3) + X(+4,2) + X(+3,1) ≤ N_+
X(+2,3) + X(+4,2) + X(+6,1) ≤ N_+
X(+3,2) + X(+5,1) + X(+6,1) ≤ 3
X(-1,3) + X(-2,2) + X(-4,2) + X(-5,1) ≤ N_-
4 : X(+4,3) + X(+5,2) ≤ N_+
X(-4,3) + X(-5,2) ≤ N_-
X(-4,3) + X(-6,1) ≤ N_-
X(-5,2) + X(-6,1) ≤ 2
5 : X(+7,1) + X(+8,1) ≤ N_+
X(-7,1) ≤ N_-

(1) 가용자원과 제어스텝의 확장

성능제약 스케줄링은 그림 6과 같이 연산이 할당될 수 있는 제어스텝의 폭이 한정된 상태 하에서, 최대의 자원공유를 꾀하는 스케줄링인 반면, 비용제약 스케줄링은 그림 7과 같이 사용가능한 자원의 수가 한정된 상태 하에서, 가장 빠른 시간 내에 동작 알고리즘의 수행을 마치기 위한 스케줄링을 말한다.

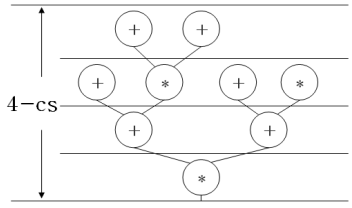


그림 6. 성능제약 스케줄링
Fig. 6. The performance constraint scheduling

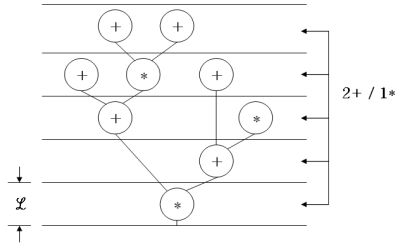
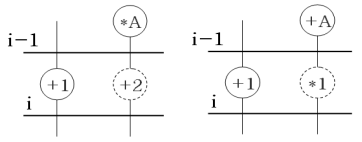


그림 7. 비용제약 스케줄링
Fig. 7. The cost constraint scheduling

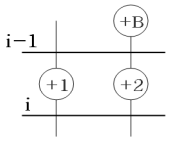
따라서 성능제약 스케줄링 시에 요구되는 자원의 개수보다도 적은 개수의 자원으로서 스케줄링을 행하는 비용제약 스케줄링에서는 그림에서 볼 수 있듯이, 제어스텝이 l만큼 필연적으로 증가하게 된다.

(2) 연산의 이동유형

제한된 자원으로 스케줄링을 행하는 경우, 특정 제어스텝 내에서는 자원의 충돌이 필연적으로 발생하게 된다. 이는 초과된 만큼의 연산을 뒤쪽 제어스텝에 할당시킴으로서 해당 제어스텝의 자원의 충돌을 방지할 수 있다.



(a) D-유형 연산이동 (b) I-유형 연산이동



(c) E-유형 연산이동

그림 8. 연산의 이동 유형
Fig. 8. The move type of operation

그림 8은 제어스텝-$(i-1)$내의 연산이 제어스텝-i로 이동됨에 따라, 제어스텝-i에 할당되는 연산의 변화를 나타낸 것이다. 그림 8(a)는 제어스텝-(i-1)내의 '*1' 연산이 뒤쪽 제어스텝으로 이동됨에 따라, 제어스텝-i의 '+' 연산의 개수가 감소하는 경우를 나타낸 것이고, 그림 8(b)는 '+1'연산의 이동에 의해 오히려 뒤쪽 제어스텝의 특정유형의 연산이 증가하는 경우를 나타낸 것이다. 그림 8(c)는 '+1' 연산의 이동에도 불구하고 제어스텝-i의 '+'연산의 개수에는 아무런 변화가 없는 경우를 나타낸 것이다.

Ⅲ. 실험 및 결과

본 논문에서 제안한 스케줄링 알고리즘의 타당성을 입증하기 위하여, 다음과 같은 벤치마크 모델에 적용시켰다. 실험의 대상인 벤치마크 모델로서는 High-level Synthesis Workshop에서 표준 벤치마크 모델로 채택된 5차 디지털 웨이브필터[9]를 택하였으며, 실험결과는 공개된 스케줄링 성능이 우수한 것으로 판정된, ALPS 시스템의 스케줄링[10] 결과와 비교하였다. 5차 디지털 웨이브필터를 실험모델로 선택한 대부분의 논문에서와 마찬가지로, '+' 연산의 지연시간은 $40ns$, '*' 연산의 지연시간은 $80ns$, 제어스텝의 시간간격은 $50ns$으로 설정하였다. 실험결과는 표 2와 같다. 표의 'Y_i'는 본 논문에서 제안한 ILP 기반의 스케줄링 결과를 나타낸다.

표 2. 파이프라인 데이터패스 스케줄링 결과
Table 2. The result of Pipelined datapath scheduling

회전지연 시간		1	2	3	4	5	6	7	8	9	13	16
소 요 자 원		26+ 16*	13+ 8*	9+ 8*	7+ 4*	6+ 4*	5+ 3*	4+ 3*	4+ 2*	3+ 2*	2+ 2*	2+ 1*
지연	ALPS	20	20	20	20	21	23	22	23	24	26	24

표 3. 파이프라인형 데이터패스 스케줄링 결과
Table 3. Pipelined Datapath Scheduling Results

(단위:싸이클타임)

회전지연시간		1	2	3	4	5	6	7	8	9	10
소요 자원 (지연 시간 =9)	ALPS	26+ 8*	13+ 4*	9+ 4*	7+ 3*	8+ 4*	5+ 2*	6+ 2*	6+ 2*	4+ 2*	
	Yi	26+ 8*	13+ 4*	9+ 4*	7+ 3*	8+ 4*	5+ 2*	6+ 2*	6+ 2*	4+ 2*	
소요 자원 (지연 시간 =10)	ALPS	N/A	13+ 4*	10+ 4*	7+ 2*	6+ 3*	5+ 2*	5+ 2*	6+ 2*	4+ 2*	4+ 2*
	Yi	26+ 8*	13+ 4*	9+ 3*	7+ 2*	6+ 2*	5+ 2*	4+ 2*	5+ 2*	3+ 2*	3+ 2*

표 2는 DFG 모델의 '*' 연산의 지연시간을 3-싸이클타임으로 가정한 경우, 이를 회전지연시간이 2-싸이클타임이고 지연시간이 3-싸이클타임인 파이프라인형 승산기로 수행시켰을 때의 스케줄링 결과이다.

표 3은 '+' 연산의 체이닝에 의한 스케줄링 결과로서, 이 경우, 제어스텝의 시간간격을 100ns로 설정하였다. 임의의 회전지연시간으로 동작하는 데이터패스의 구성에 필요한 자원의 수를, ALPS 시스템의 결과와 비교하였다.

데이터패스의 지연시간이 10인 경우를 비교해볼 때, 기존의 연구 결과에서는 데이터패스의 회전지연 시간이 3-싸이클타임인 경우는 한 개의 '+' 연산자와 한 개의 '*' 연산자가 2개의 가산기(adder)와 2개의 감산기(subtracter)를 필요로 하나, 본 논문의 결과에서는 3번의 조건부 자원공유를 행하고 있음을 알 수 있다. 따라서 DFG 내에 조건부 분기연산이 다수 존재하는 경우는, 본 논문의 스케줄링 방식이 보다 폭 넓은 조건부 자원공유

를 이룰 수 있어, 보다 효율적인 자원의 절감을 꾀할 수 있다.

Ⅳ. 결 론

본 논문은 새로운 멀티프로세서 디자인을 위한 상위 수준합성 시스템의 회로 복잡도 최적화 ILP 알고리즘을 제안하였다. 상위수준 합성에서 가장 중요한 연산자의 특성과 데이터패스의 구조를 분석하고, 멀티사이클 연산의 스케줄링 시 가상연산자 개념을 도입하여, 멀티사이클 연산을 구현하는 연산자의 유형에 관계없이 공통으로 적용시킬 수 있는 확장형 ILP 알고리즘을 이용하여 증명하였다. 제안된 ILP 알고리즘의 수식에는, 데이터패스 합성 시 유용하게 적용될 수 있도록, 연산의 체이닝과 멀티사이클 연산의 구조적 파이프라이닝과 기능적 파이프라이닝 스케줄링을 위한 ILP 수식과, 조건부 분기연산의 스케줄링을 위한 ILP 수식이 포함되었다.

본 논문에서 제안한 알고리즘의 성능을 평가하기 위하여, 표준벤치마크 모델인 5차 디지털 웨이브필터에 대한 스케줄링을 행한 결과, 기존의 데이터패스 스케줄링 결과와 일치함으로서, 제시된 모든 ILP 수식이 정확하게 기술되었음을 알 수 있었다.

References

[1] C-H. Lin, “A New Register Transfer Level Synthesis Methodology for Efficient SOC Design”, The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), VOL.11 No.2, April 2011.

[2] Gebotys, Catherine H., and Mohamed I. Elmasry. “Optimal VLSI architectural synthesis: area, performance and testability.” Vol. 158. Springer Science & Business Media, 2012.

[3] Gajski, Daniel D., et al. “High-Level Synthesis: Introduction to Chip and System Design.” Springer Science & Business Media, 2012.

[4] C.Tseng, D.P.Siewiorek, “Automated Synthesis of Datapath in Digital System”, IEEE Tr.CAD, Vol.CAD-5, pp.379-385, July. 1986.

[5] S.Y.Kung, H.J.Whitehouse, T.Kailath, “VLSI and Modern Signal Processign”, Englewood Cliffs, NJ : Prentice Hall, pp.258-264, 1985.

[6] H.Tricky, “Flamel : A High-Level Hardware Compiler”, IEEE Tr. on CAD, pp.259-269, March. 1987.

[7] P.G.Paulin, et al, “HAL : A Multi-Paradigm Approach to Automatic Data Path Synthesis”, Proc.of 23rd DAC, pp.263-270, June. 1986.

[8] Nemhauser, George L. "Column generation for linear and integer programming." Optimization Stories 20:64, 2012.

[9] Lars Wanhammar, Ya Jun Yu, “Digital Filter Structures and Their Implementation”, Academic Press Library in Signal Processing, Vol.1, pp.245 -338, 2014.

[10] Jothi Komal, Akkary Haitham, “Tuning the continual flow pipeline architecture.” In: Proceedings of the 27th international ACM conference on International conference on supercomputing. ACM, pp.243-252. 2013.

저자 소개

장 정 욱(준회원)

- 2005년 : 세명대학교 컴퓨터학과 이학사
- 2007년 : 세명대학교 일반대학원 이학석사(전산정보학 전공)
- 2007년~현재 : 세명대학교 일반대학원 박사과정(전산정보학 전공)

<주관심분야 : CAD, SoC, ASIC, Embedded System, RTOS>

인 치 호(정회원)

- 1985년 : 한양대학교 공과대학 전자공학과 공학사
- 1987년 : 한양대학교 대학원 공학석사(CAD 전공)
- 1996년 : 한양대학교 대학원 공학박사(CAD 전공)
- 1992년~현재 : 세명대학교 컴퓨터학부 교수

<주관심분야 : CAD, SoC, ASIC 설계, CAD 알고리즘, RTOS 및 내장형 시스템>