論文 92-29A-4-11

파이프라인형 데이타패스 합성을 위한 스케쥴링 기법

(A Scheduling Technique for Pipelined Datapath Synthesis)

李 近 萬*, 林 寅 七*

(Keun Man Yi and In Chil Lim)

要 約

본 논문에서는 상위수준 합성(High-level systhesis)의 가장 중요한 과제인 스케쥴링 문제를 다루었다. 스케쥴링 문제에 대한 접근방식으로서, integer linear programming을 이용한 방식을 택하였다. 디지탈 시스템 합성시 효과적으로 적용될 수 있도록, 파이프라인형 데이타패스 스케쥴링을 위한 새로운 ILP 모델을 제시하였다. 본 논문에서 제시한 ILP 모델의 성능을 비교하기 위하여, 표준 benchmark 모델인 fifth-order digital wave filter에 대한 스케쥴링을 행 하였다. 그 결과, 기존 방식에 비해 보다 우수한 결과를 얻을 수 있었다.

Abstract

This paper deals with the scheduling problems, which are the most important subtask in High-level Synthesis. ILP(integer linear programming) formulations are used as a scheduling problem approach. For practical application to digital system design, we have concentrated our attentions on pipelined datapath scheduling. For experiment results, we choose the 5-th order digital wave filter as a benchmark and do the schedule. Finally, we can obtain better and near-optimal scheduling results.

I. 서 론

디지탈 시스템의 동작기술문(動作記述文)으로 부터 레지스터-전송 수준(Register-transfer level) 의 구조적인 정보를 얻기 위한 상위수준합성(High-level synthesis) 과정 중, 가장 중요한 것이 데이타패스합성(Datapath synthesis) 과정이다. 이러한 데이타패스합성 과정은 크게 나누어, 연산의 스케쥴링(operation scheduling)과 하드웨어할당(hardware allocation) 단계로 구분된다. 스케쥴링 단계에서는 동작 알고리듬의 모든 연산을 적절한 제어스텝(control step)에 할당시키고, 하드웨어할당 단계에서는 연산자(functional unit), 레지스터 및 상호연결선(interconnect)을 데이타패스에 할당시킨다.

가장 기본적인 스케쥴링 기법은, 사용 가능한 하드웨어자원(hardware resource)의 갯수와 동작속도의 제약이 가해지지 않은 상태에서 행해지는, ASAP 스케쥴링(as soon as possible scheduling) 기법[1], [2]과 ALAP 스케쥴링(as last as possible scheduling) 기법[3]을 들 수 있다. 자원의 제약이 가해진 상태에서는 변형된 ASAP 스케쥴링 기법[4],[5]이나, list 스케쥴링 기법[6]-[8]이 사용된다. 또한, 동작속도의 제약이 가해진 상태에서는 freedom-based 스케쥴링 기법[6], [7],[9]과 force-directed 스케쥴링 기법[10]이 사용된다.

이상에서 언급한 스케쥴링 기법들은 각각의 연산을 한번에 하나씩 제어스텝에 할당하기 때문에, 그

*正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)

接受日字: 1991年 2月 1日

스케쥴링 결과는 할당되는 연산의 순서에 크게 의존하게 된다. 즉, 임의의 연산이 해당 제어스텝에 할당되고 나면, 이후에 스케쥴링 되는 모든 연산은 이미 스케쥴링된 모든 연산의 스케쥴링 결과에 전적으로 의존하므로, 그 스케쥴링 결과는 결코 최적의 것이라고 단정지을 수 없다. 또한, 모든 스케쥴링 문제는 NP-hard(NP-complete)[11]이므로, 최상의 해를 구할 수 있는 스케쥴링은 결코 기대할 수 없는 것이기 때문에, 주어진 조건하에서 최적의 해를 구하는 것이 보다 실제적이고 바람직한 방법이다. 따라서, 스케쥴링 시 발생되는 모든 제약조건을 동시적이고 일괄적으로 처리함으로서, 이미 할당된 연산이 할당시키고자 하는 연산에 미치는 영향을 배제시킬 수 있는 스케쥴링 기법이 요구된다.

이러한 문제를 해결할 수 있는 방법으로는, 연산과 제어스텝 간의 관계에 따라 스케쥴링 문제를 다원 일차방정식으로 전개시켜, 원하는 목적함수를 최소 또는 최대화 시키는 ILP(integer linear programming) 기법이 이용된다. 이러한 ILP 기법을 사용한 기존의 연구로서는, 레지스터-전송 수준에서의 논리합성을 꾀한 integer programming 모델[12]과, multiprocessor DSP 시스템용 합성시스템인 CATHEDRAL-II[13]의 마이크로코드의 스케쥴링을 위한 integer programming 접근방식을 들 수 있다. 그러나, 이러한 기존의 연구는 수학적인 표현방식이 너무 복잡하고, 해를 구하기 위한 CPU 시간이 너무 길어 비 실용적인 단점을 갖고 있다.

따라서, 이러한 단점을 보완하고, 실제적으로 데이타패스 스케쥴링에 유용한 ILP 기법[14]-[18]이 제안되었다. 이들 논문에서는 스케쥴링의 목적에 따라, 각각, 성능제약 스케쥴링(performance-constrained scheduling)을 위한 ILP 모델[15]-[18]과 비용제약 스케쥴링(cost-constrained scheduling)을 위한 ILP모델[14]-[16]이 제시되었다. 또한, 이들은 사용하는 연산자의 유형에 따라, 복수기능의 연산자(multifunctional unit)를 사용하는 유형[17][18]과 단수기능의 연산자(unifunctional unit)를 사용하는 유형[14]-[16]으로 구분된다.

그러나, 이러한 기존의 방식들도 다음과 같은 단점을 내포하고 있다. 즉,

1) ILP 수식의 형태가 비구조적(unstructural) 이기 때문에, 모델의 싸이즈가 큰 경우는 수식을 생성, 분석하거나 재구성하는 데에 상당한 시간이 요구된다[15][18]

2) 조건부 자원공유(conditional resource sharing)를 전혀 고려치 않았다[14][16]-[18]

3) ILP 모델을 비선형적인 형태로 기술한 후, 이를 선형적인 형태로 변환하였기 때문에, 수식이 난해함은 물론, 수식이 생성과정이 복잡하다[15]

본 논문에서는 이러한 단점을 보완한 ILP 모델의 스케쥴링 기법에 관해 논하였다. 본 논문에서 제안한 ILP 모델의 특징은 첫째, 파이프라인 스케쥴링 시, 할당시키고자 하는 연산자의 형태-비파이프라인형과 파이프라인형을 구별할 필요가 없으며, 둘째, 모든 ILP 모델이 선형적인 형태로 표현되었기 때문에, 비선형적인 모델을 선형적인 형태로 변환하기 위한 별도의 변환과정이 필요치 않으며, 셋째, 분기연산(branch operation) 간의 자원공유가 최대로 이루어지도록 조건부 자원공유를 위한 ILP 모델이 효과적으로 기술되었다는 점이다. 2장에서는 데이타패스 스케쥴링을 위한 기본적인 ILP 모델에 대해 논하고, 제3장에서는 개선된 형태의 ILP 모델에 대해 언급하였다. 실험 및 결과는 제4장에서 논하였다.

Ⅱ. 데이타패스 스케쥴링을 위한 ILP 모델

DFG상의 연산 O_i를 제어스텝 C_j에 할당시키는 과정은, 연산 Oi를 제어스텝 Cj에 mapping시키는 과정을 의미한다. 이를 위해서 연산 Oi와 제어스텝 Cj는 일정한 관계를 유지하여야만 한다. 즉, 연산간의 선후관계와 연산이 할당될 수 있는 제어스텝의 범위로 표현되는 연산 O_i의 속성(attribute)은, 일정한 관계식 **R**로서, 임의의 시간간격인 제어스텝 C_j의 속성에 결합될 수 있어야만 한다. 관계식 **R**은 연산 O_i와 이들의 할당에 소요되는 제어스텝의 갯수 및 임의의 제어스텝에 할당될 수 있는 연산의 갯수와의 관계를 의미한다.

따라서, 연산간의 선후관계를 나타내는 선후관계(precedence relation) 식과, 연산과 스케쥴링 범위와의 관계를 나타내는 범위관계(range relation)식, 연산과 제어스텝의 갯수와의 관계를 나타내는 시간관계(time relation)식 및 연산과 특정 제어스텝에 할당될 수 있는 연산의 갯수와의 관계를 나타내는 자원관계(resource relation)식을 효과적으로 기술함으로서, 연산 O_i와 제어스텝 C_j간의 결합을 이룰 수 있다.

이러한 4개의 관계식을 ILP 모델로 표현하기 위하여 먼저, 모델의 표현에 사용되는 기호를 아래와 같이 정의한다.

n: 연산유형 t인 연산 O_i의 갯수

t_T: 싸이클타임(cycle time)-제어스텝의 시간간격(time solt)

d_t: 연산유형 t인 연산의 지연시간

D_t:제어스텝의 싸이클타임 수로 표시되는 연산유형 t인 연산의 지연시간($=\lceil d_t/t_T \rceil$)

S_i, L_i:ASAP 스케쥴링과 ALAP 스케쥴링에 의해 결정되는 연산 O_i의 상하한 제어스텝

$O_i(k)$:연산 O_i의 k번째 요소(element)

$[O_i=\{O_i(1), O_i(2), \cdots, O_i(r_i)\}]$, $(r_i=L_i-S_i+1)$

P_{ik}:연산 $O_i(k)$가 할당되는 제어스텝 $(=S_i+k-1)$

$X(i, k)$:연산 O_i의 k-번째 요소가 제어스텝-P_{ik}에 할당되었을 때, 그 값이 1이 되는 0-1 정수변수(0-1 integer variable)

앞에서 설명한 4개의 관계(relation)를 정의된 기호들로서 표현하면 다음과 같다.

1) 선후관계식

연산 O_i의 최근접 후행연산(nearest successor)를 O_j라 할 때, 연산 O_i와 연산 O_j 사이에는 다음의 관계식(1)이 성립한다.

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

2) 범위관계식

연산O_i는 상한제어스텝 S_i와 하한제어스텝 L_i의 범위에 걸쳐 오직 한번만 할당되어야 하기 때문에, 연산의 각 요소 사이에는 다음의 관계식(2)가 성립한다.

$$\sum_{k=1}^{r_i} X(i,k) = 1, \ (i=1, 2, \cdots, n) \quad (2)$$

3) 시간관계식

연산 O_i의 각 요소가 할당되는 제어스텝-P_{ik}는 제어스텝의 최대치 T_{max}를 초과해서는 안된다. 따라서, 후행연산이 존재치 않는 모든 연산 O_i의 요소 $O_i(k)$와 제어스텝의 최대치 T_{max}와는 다음의 관계식(3)이 성립한다.

$$\sum_{k=1}^{r_i} (P_{ik}\times X(i,k)) \leq T_{max} - D_t + 1, \ (i=1, 2, \cdots, n) \quad (3)$$

-체인드연산(chained operation)의 자원관계식

연산 O_j의 최근접 후행연산을 O_s, 연산 O_j의 최근접 선행연산을 O_i라 할 때, $(d_j+d_s) \leq t_T$인 조건과 $(d_i+d_j+d_s) > t_T$인 조건이 동시에 만족되는 경우, O_j와 연산 O_s의 체이닝은 다음의 관계식(4)로 표현된다.

$$\sum_{h=1}^{r_j} (P_{jh}\times X(j,h)) - \sum_{m=1}^{r_s} (P_{sm}\times X(s,m)) \leq 0 \quad (4a)$$

$$\sum_{k=1}^{r_i} (P_{ik}\times X(i,k)) - \sum_{m=1}^{r_s} (P_{sm}\times X(s,m)) \leq -1 \quad (4b)$$

4) 자원관계식

특정 제어스텝-P에 할당될 수 있는 연산유형 t인 연산의 총 갯수는 사용가능한 연산자의 갯수 N_t를 초과해서는 안된다.

$$\sum_{i=1}^{n} X(i, P-S_i+1) - N_t \leq 0, (P=1, 2, \cdots, T_{max}-D_t+1) \quad (5)$$

-멀티싸이클 연산(multicycle operation)의 자원관계식

멀티사이클 연산은 비파이프라인형 연산자(non-pipelined functional unit)나 또는 파이프라인형 연산자(pipelined functional unit)에 의해 수행될 수 있다. 어느 경우나 멀티사이클 연산을 수행할 수 있다는 점은 동일하나, 자원공유의 관점에서 볼 때, 그 방법상 차이가 있다. 즉, 비파이프라인형 연산자는 일단 연산의 수행이 시작되면, 해당 연산을 마칠 때까지 다른 연산의 수행에 할당될 수 없는 반면, 파이프라인형 연산자는 연산의 수행 도중에도 다른 연산의 수행에 할당될 수 있기 때문에, 보다 많은 자원공유를 이룰 수 있다. 비파이프라인형 연산자는 일단 임의의 연산을 수행하게 되면, 해당 연산이 종료되기 이전에는 다른 연산의 수행에 할당될 수 없다. 그림 1과 같이 지연시간이 D_t인 연산 A가 제어스텝-P에 할당되는 경우를 가정한다. 연산 A가 제어스텝-P에 할당되기 위해서는 제어스텝-P에 존재하는 동일 유형의 연산의 갯수가 사용 가능한 자원의 갯수를 초과해서는 안된다. 즉, 제어스텝 $-(P-\{D_t-1\})$에 할당된 연산B, 제어스텝-$\{P-(D_t-2)\}$에 할당된 연산 C, …, 제어스텝 $-(P-1)$에 할당된 연산 N 및 제어스텝-P에 할당된 연산 A의 총 합은 사용가능한 연산자의 갯수 N_t보다 적어야만 한다.

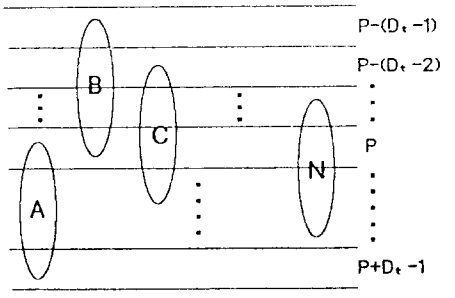

그림 1. 비파이프라인형 연산자로 수행되는 멀티사이클 연산의 스케쥴링

Fig. 1. Multicycle operation scheduling with nonpipelined implementation.

따라서, 연산유형 t인 연산 O_i의 요소 $O_i(k)$가 제어스텝-P_i에 할당되기 위해서는 다음의 조건식(6)이 만족되어야만 한다.

$$\sum_{j=1}^{D_t-1}\sum_{i=1}^{n} X(i,\ (P-S_i+1)-j) \leqq N_t \qquad (6)$$

파이프라인형 연산자로 수행되는 멀티싸이클연산의 스케쥴링은 제3장에서 다루기로 한다. 파이프라인형 연산자로 수행되는 멀티싸이클연산의 스케쥴링 이론은 문헌[15]에서 상세히 다루기는 하였으나, ILP 수식을 비선형적인 형식으로 기술한 후, 이를 선형적인 형태로 변환시켰기 때문에, 수식이 난해하고 복잡함은 물론, DFG의 기하학적 구조와 수식의 형태상 의미가 잘 부합되지 않는다. 따라서, 제3장에서는 수식의 전개와 표현방법이 선형적인 ILP 수식을 유도한다.

Ⅲ. 파이프라이닝 (pipelining)과 스케쥴링

1. 파이프라인형 연산자

파이프라인형 연산자는, 연산자 내에서 이전 데이타의 연산이 수행되고 있는 동안에도 새로운 데이타가 동일 연산자 내에 유입될 수 있는 기능을 보유하고 있다. 이미 연산이 진행되고 있는 입력 데이타와 새로이 유입되는 데이타 간의 시차인 latency가 ℓ-싸이클 타임이고, 연산의 수행시간이 D_t인 연산자는 , $\ell \leqq D_t$인 조건하에서 매ℓ-싸이클 타임마다 새로운 연산을 수행할 수 있다.

그러므로, 파이프라인형 연산자로 수행되는 멀티사이클 연산의 스케쥴링에서는 파이프라인형 연산자에 유입된 데이타가 ℓ-싸이클타임 만큼 처리된 후에는, 어느 때라도 새로이 유입된 데이타가 동일 파이프라인형 연산자를 사용할 수 있으므로, 파이프라인형 연산자가 연산의 수행을 시작한 제어스텝 보다도 ℓ-싸이클타임 이상 이격된 임의의 제어스텝 내에 존재하는 동일유형의 연산은 해당 파이프라인형 연산자를 공유할 수 있게 된다.

따라서, 제어스텝-P ($1 \leqq P \leqq \ell$)에서 연산의 수행이 시작된 파이프라인형 연산자는 제어스텝 ($P+\ell$) 이후의 모든 제어스텝에 존재하는 연산이 공유할 수 있기 때문에, 필요한 연산유형 t인 파이프라인형 연산자의 갯수는 연산의 공유가 가능한 것끼리 묶어, 다음 식(7)과 같이 구분지을 수 있다.

$$N_t = N_{t1} + N_{t2} + \cdots + N_{tP} \quad (P=1, 2, \cdots, \ell) \qquad (7)$$

여기서, N_{tP}는 제어스텝-P와 제어스텝-($P+\ell$) 이후의 제어스텝에 존재하는 연산유형 t인 연산이 공유할 수 있는 파이프라인형 연산자의 최대치를 나타낸다.

따라서, 파이프라인형 연산자에 의한 멀티사이클 연산의 스케쥴링에서는 자원관계식이 다음 식(8)과 같이 표현된다.

$$\sum_{i=1}^{n_t} X(i, P-S_i+1) \leqq N_{tP}, \quad (P=1, 2, \cdots, \ell) \qquad (8a)$$

$$\sum_{i=1}^{n_t} X(i, q-S_i+1) \leqq N_{tP}, (q=P+\ell, P+\ell+1; \cdots, T_{max}-D_t+1) \qquad (8b)$$

2. 가상연산자

멀티싸이클연산의 효과적인 스케쥴링을 위하여, 가상연산자(image operator)를 아래와 같이 정의하여 사용한다.

〈정의 1〉

임의의 연산자 F_i의 가상연산자는, 연산자 F_i와 동일한 데이타 처리기능을 보유할 수 있는 연산자 집합을 말한다.

〈정리 1〉

지연시간이 D이고 제어스텝-P에서 연산의 수행을 시작하는 임의의 멀티싸이클 연산 O_i를, latency가 ℓ인 파이프라인형 연산자 F_i로 구현하고자 하는 경우, 연산자 F_i의 n차 가상연산자는 제어스텝-($P+n\ell$) 에서 연산의 수행을 시작하는 복수개의 비파이프라인형 연산자이다 ($n=1, \cdots, \lceil d/\ell \rceil$).

〈증　　명〉

연산 O_i에 입력된 i-번째 데이타가 연산의 수행을 시작한 후, ($n*\ell$)-싸이클타임 뒤에는, (i+n)-번째의 데이타가 새로이 연산 O_i에 입력된다. 이 경우, 연산 O_i는 i-번째 부터 (i+n)번째까지의 모든 입력데이타를 동시에 처리하여야만 한다. 따라서, n개의 비파이프라인형 연산자를 시간간격이 ℓ-싸이클 타임이 되도록 계단형태로 나열시켜 입력된 데이타를 순차적으로 처리시키면, 기능적으로 지연시간이 D인 파이프라인형 연산자와 동일한 기능을 보유할 수 있다. 그러므로, (i+1)-번째 입력되는 데이타를 처리하는 가상연산자를 1차 가상연산자, (i+1)-번째 입력되는 데이타를 처리하는 가상연산자를 n차 가상연산자라고 명명할 때, 제어스텝-P에서 연산의 수행을 시작하는 파이프라인형 연산자의 n차 가상연산자는 제어스텝-($P+n\ell$)에서 연산의 수행을 시작하는 비파이프라인형 연산자이다.

그림2는 파이프라인형 연산자의 가상연산자를 도시해 놓은 것이다. 임의의 파이프라인형 연산자는 하나의 비파이프라인형 연산자와 n개의 가상연산자

로 구성되어, 비파이프라인형 연산자 F_i가 i번째 데이타를 처리하는 도중에 유입되는 (i+n)-번째의 데이타는 n차-가상연산자에 의해 처리됨을 알 수 있다. 임의의 파이프라인형 연산자는 계단형태의 복수개 비파이프라인형 연산자와 기능적으로 동일한 것으로 간주할 수 있고 또한, 가상연산자는 실제로 존재하지 않는 연산자이므로, 자원공유의 관점에서 볼 때, latency가 ℓ인 파이프라인형 연산자는 지연시간이 ℓ인 비파이프라인형 연산자로 간주하여도 무방하다. 따라서 멀티싸이클 연산을 파이프라인형 연산자로 구현하고자 하는 경우의 스케쥴링 방법은 비파이프라인형 연산자를 사용한 스케쥴링 방법과 동일하다. 즉, 데이타 입력 latency가 ℓ인 파이프라인형 연산자를 사용하는 경우는 지연시간이 ℓ-싸이클타임인 비파이프라인형 연산자를 사용하는 것과 동일한 방법으로 스케쥴링을 행할 수 있다.

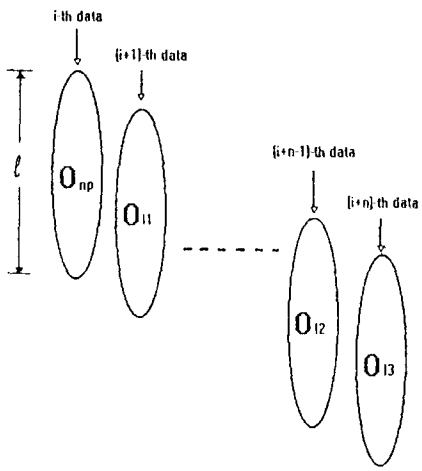

그림 2. 파이프라인형 연산자의 가상연산
Fig. 2. Image operators for a pipelined operator.

따라서, 가상연산자 개념에 따른 멀티싸이클연산의 스케쥴링에서는 자원 관계식을 식(9)와 같이 일률적으로 표현할 수 있다.

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

(A=D_t:비파이프라인형 연산자 사용시
A=ℓ:latency가 ℓ인 파이프라인 연산자 사용 시)

3. 기능적 파이프라이닝

파이프라인형 연산자에 의해 파이프라인닝을 실현하는 구조적 파이프라이닝과는 달리, 기능적 파이프라이닝에서는 데이타패스로 하여금 파이프라인 기능을 보유토록 하는 방식이다. 따라서, 기능적 파이프라이닝의 데이타패스-파이프라인형 데이타패스-는 입력 latency가 ℓ-싸이클타임이고, 지연시간이 D_t인 하나의 파이프라인형 연산자와 동일하게 동작한다. 기능적 파이프라인에 의한 데이타패스를 구현할 경우, 구간i-제어스텝 i와 제어스텝 (i+nℓ)[n=0, 1, ⋯, N]-에 존재하는 모든 연산은 동일한 시각에 동시에 수행되므로, 이들간에는 자원의 공유가 허용되지 않는다. 정수 k_i가 $k_i=P-S_i+1$이고, 정수 q_i가 $q_i=\lfloor(r_i-k_i)/\ell\rfloor$로 표시될 때, 기능적 파이프라인 구성을 위한 제어스텝-P의 자원관계식은 아래의 식(10)과 같이 표현된다.

$$\sum_{j=0}^{q_i}\sum_{i=1}^{n} X(i,\ (P-S_i+1)+j\ell)-N_t \leqq 0 \qquad (10)$$
$$(P=1, 2, \cdots, \ell)$$

이러한 관계식은, 멀티싸이클 연산을 비파이프라인 연산자로 수행시킬 경우, 연산의 지연시간 D_t와 데이타패스의 layency ℓ과의 관계에 따라, 다음과 같은 제약조건이 가해진다.

〈정리 2〉

파이프라인형 데이타패스 스케쥴링에서, 멀티싸이클 연산을 비파이프라인 연산자로 수행시키고자 하는 경우, 최소한 한개 이상의 자원공유를 이루기 위해서는 연산의 지연시간 D_t와 데이타패스의 latency ℓ과의 사이에는 다음과 같은 조건이 만족되어야만 한다.

$$2\times D_t \leqq \ell$$

〈증 명〉

구간 i[mod-ℓ]에서 지연시간이 D_t인 연산의 수행을 시작한 비파이프라인 연산자는 구간 (i+D_t-1)[mod-ℓ]에서 연산의 수행을 종료하기 때문에, 최소한의 자원공유를 이루기 위해서는 구간 (i+D_t)[mod-ℓ]에서 또 다른 연산의 수행에 할당될 수 있어야만 한다. 이 경우 구간 (i+D_t)[mod-ℓ]에서 연산의 수행을 시작한 비파이프라인 연산자는 구간 (i+2×D_t-1)[mod-ℓ]에서 연산의 수행을 종료하기 때문에, 이 사이에서는 해당 연산자가 다른 연산의 수행에 할당되어서는 안된다. 따라서, 구간 (i+2×D_t-1)[mod-ℓ]이 구간 i[mod-ℓ]과 중복되거나, 또는 이를 초과하는 경우는 자원의 충돌이 발생하게 된다. 그러므로, 자원의 충돌을 방지하기 위해서는 다음의 조건이 만족되어야만 한다.

$i[mod-\ell] > (i+2\times D_t-\ell)[mod-\ell]$ $(i=0, 1, \cdots \ell-1)$

i는 ℓ보다 작거나 같은 값이므로 $i[mod-\ell]=i$이다.

또한, $2\times D_t \leqq \ell$인 경우,

$$\begin{aligned}&(i+2\times D_t-1)[mod-\ell]\\ &\leqq (i+\ell-1)\ [mod-\ell]\\ &=(i-1)\ [mod-\ell]\\ &=i-1\end{aligned}$$

이므로 위의 조건식은 언제나 만족된다. 따라서, $2\times D_t \leqq \ell$인 조건이 만족되는 경우는 최소의 자원공유가 가능하다.

〈정리 3〉

연산의 지연시간 D_t와 파이프라인형 데이타패스의 latency ℓ과의 관계가 $2\times D_t > \ell$인 경우는, 멀티싸이클 연산의 갯수 n과 이의 수행에 필요한 비파이프라인 연산자의 최소치 N_{min}과의 사이에는 다음의 관계식이 성립한다.

$$N_{min}=\lceil D_f/l \rceil \times n$$

〈증 명〉

$2\times D_t > \ell$인 조건하에서, n개의 멀티싸이클 연산의 수행에 필요한 비파이프라인 연산자의 최소치 N_{min}은 ℓ값의 범위에 따라 다음과 같이 구분된다. 즉, ℓ값의 범위가

$2\times D_t > \ell \geqq D_t$인 경우는 $N_{min}=n$,

$D_t > \ell \geqq D_f/2$인 경우는 $N_{min}=2n$,

$D_f/2 > \ell \geqq D_f/3$인 경우는 $N_{min}=3n$

　　⋮　　⋮

로 각각 표시할 수 있으므로, 일반적으로

ℓ값이 $D_f/(k-1) > \ell \geqq D_f/k$인 범위에서는 $N_{min}=k\times n$이 된다. 이러한 ℓ값을 만족하는 k값의 범위는,

$D_f/\ell \leqq k < (D_f/\ell)+1$이 된다.

k는 언제나 양의 정수값을 취하므로, 이러한 k값의 범위는,

$$k=\lceil D_f/\ell \rceil$$

로 압축하여 표현할 수 있다.

따라서, n개의 멀티싸이클 연산의 수행에 필요한 비파이프라인 연산자의 최소치 N_{min}은

$$N_{min}=k\times n=\lceil D_f/\ell \rceil \times n$$

로 표현된다.

4. 조건부 자원공유(conditional resource sharing)

동일 연산자로 하여금 여러 연산을 수행하도록 하는 자원공유는, 서로 다른 제어스텝(disjoint control steps, non-overapped control steps)에 할당되는 연산들 사이에서 이루어지는 무조건 자원공유(unconditional resource sharing)와 조건분기(conditional baanch)의 상호배타적(mutually exclusive) 연산들 사이에서 이루어지는 조건부 자원공유로 분류된다.

상호배타적 연산은 조건에 따라 분기하는 분기연산(branch operation) 중, 임의의 조건하에서도 결코 동일 데이타를 처리하지 않는 연산의 집합(operation set)을 말한다. 구조적 파이프라인 스케쥴링의 경우, 이러한 상호배타적 연산들 사이에서 이루어지는 조건부 자원공유는 언제나 가능하다. 그러나, 기능적 파이프라인 스케쥴링의 경우는 twisted pair에 기인하는 deadlock 현상 때문에 조건부 자원공유를 완전히 보장하지는 못한다.[19] 기능적 파이프라인 스케쥴링에서의 deadlock 현상을 방지하기 위해서는, 동일 제어스텝에 할당되는 상호배타적 연산들 사이에서만 조건부 자원공유를 행하면 된다. 또한 합성의 초기단계에서 파이프라인 시스템의 입력 latency가 고정되는 경우는, twisted pair의 각 연산을 각기 다른 구간(partition)에 할당 시킴으로서, deadlock의 방지는 물론, 최대의 자원공유를 이룰 수 있다.

임의의 제어스텝-P 또는 구간-P에 할당되는 연산집합 중, 상호배타적인 연산의 집합을 $O_m=\{O_{m1}, O_{m2}, \cdots O_{mw}\}$, 그 여집합을 $O_n=\{O_{n1}, O_{n2}, \cdots, O_{nq}\}$ 이라 할 때, 이들 사이에는 아래의 관계식(11)이 성립한다.

$$\sum_{i=1}^{w} X(m_i, (P-S_i+1)) - w \leqq 0 \qquad (11a)$$

$$\sum_{i=1}^{q} X(n_i, (P-S_i+1)) + X(m_j, (P-S_j+1)) \leqq N_t,\ (j=1, 2, \cdots, w) \qquad (11b)$$

관계식(11a)는 연산집합 O_m의 모든 연산 O_{mi}는 동일 제어스텝에 할당될 수 있다는 것을 의미하며, 관계식(11b)는 연산 O_{mi}의 수행에는 오직 한개의 연산자가 필요하다는 것을 의미한다.

5. 복잡도 해석(Complexity Analysis)

제 II 장에서 정의한 기호와 아래의 기호들로서 각 모델에 대한 복잡도는 해석을 행한다.

e:DFG내의 모든 절선(edge)의 갯수

e_c:critical path의 연산과 이들의 선행연산을 연결하는 절선의 갯수

e_f:$e-e_c$

r:DFG상에서 후행연산이 존재하지 않는 연산의 갯수

s:제어스텝의 갯수

m:연산자 유형의 갯수

본 논문에서 기술된 4개의 기본관계식을 기술하는데 필요한 변수는 제어스텝의 갯수 s와 연산의 갯수 n과의 곱으로 표현된다. 또한, 관계식 (1), (2), (3) 및

(4)를 기술하는데 필요한 방정식의 갯수는 각각, e_f, n, r 및 s×m으로 표현된다. 따라서, 기본관계식의 기술에 필요한 방정식은 $O(s\times m+n+r+e_f)$의 복잡도를 갖는다. 멀티사이클 연산의 스케쥴링에서는 관계식(6) 및 (9)에서 알 수 있는 바와 같이, 기본관계식 이외에 더 이상의 수식이 추가되지 않는다. 따라서, 방정식의 복잡도는 더 이상 증가되지 않는다.

관계식(8)에 의해 ILP 모델을 기술하는 경우, 식(8a)에 의해 $\ell\times m$개의 방정식이 생성되며, 또한, $\{(\ell\times(s-D_t+1)-\ell^2-\sum_{i=0}^{t-1} i)\}\times m$개의 방정식이 관계식(8b)의 기술에 필요하다. 따라서, 모델에 의해 생성되는 방정식의 복잡도는 $O[\{\ell\times(s-D_t+1)-\ell^2-\sum_{i=0}^{t-1} i)\}\times m+n+r+e_f]$로 표현된다. 파이프라인형 연산자가 $D_t=2$이고 $\ell=1$ 경우, 방정식의 복잡도는 $O(s\times m+n+r+e_f)$로 되어, 기본관계식의 복잡도와 동일한 값을 갖는다.

조건부 자원공유의 경우, 임의의 제어스텝에 할당되는 상호배타적 연산의 최대갯수가 p일때, 관계식(11a)의 기술에는 m개의 방정식이 필요하며, 관계식(11b)의 기술에는 m×p개의 방정식이 요구된다. 따라서, 방정식의 복잡도는 $O\{s\times(p+1)\times m+n+r+e_f\}$에 따라 증가하게 된다. 그러나, twisted pair에 기인하는 deadlock 현상을 방지하기 위하여, 동일 제어스텝에 할당되는 상호배타적 연산들 사이에서만 조건부 자원공유를 행하게 되므로, 임의의 제어스텝 내에서 자원공유를 이룰 수 있는 상호배타적 연산의 갯수는 대폭 축소된다. 따라서, 대부분의 경우, 자원관계식의 복잡도는 이보다 훨씬 큰 값으로 줄어들게 된다.

Ⅳ. 실험 및 결과

본 논문에서 제안한 스케쥴링 기법의 타당성을 입증하기 위하여, benchmark 모델에 대한 스케쥴링을 행하였다. 실험의 대상인 benchmark 모델로서는 1988 High-Level Synthesis Workshop에서 표준 benchmark 모델로 채택된 fifth-order digital wave filter[3]을 택하였으며, 실험결과는 공개된 자료가 가장 풍부하고 스케쥴링 성능이 우수한 것으로 판정된 ALPS 시스템[15]의 스케쥴링 결과와 비교하였다. 모든 ILP 수식은 SUN SPARC-II의 SunOS 하에서 LINGO[23] 패케지를 이용하여 해를 구하였다.

표1은 곱셈연산을 수행하기 위한 연산자로서 비파이프라인형 승산기를 사용하였을 때의 스케쥴링 결과로서, 참고문헌[15]의 결과와 완전하게 일치하고 있음을 알 수 있다. 따라서, 본 논문에서 제시한 ILP 수식이 정확하게 記述되었음을 알 수 있다.

Latency가 3인 경우, 본 연구에서는 8개의 승산기가 필요하였다. 그 이유는 정리(3)에서 증명한 바와 같이, $2\times D_t>\ell$인 조건에서는 8개의 곱셈연산의 수행에는 $\lceil 2/3\rceil\times 8=8$개의 승산기가 필요하기 때문이다. ALPS 시스템에서는 latency가 3인 경우, 6개의 승산기가 필요한 것으로 명시되어 있으나,[15] 이는 표기상의 에러이거나 또는 결과에 대한 검증을 거치지 않은 부정확한 자료일 것으로 생각된다(표의 '*'표시는 본 논문에서 제시한 ILP 모델에 대한 스케쥴링 결과를 나타내는 행이다).

표2는 DFG 모델의 '*' 연산의 지연시간을 3-싸이클타임으로 가정한 경우, 파이프라인형 승산기를 사용한 스케쥴링 결과로서, latency=2이고 지연시간이 3-싸이클타임인 연산자를 사용하였다. 각 latency에서 소요되는 자원의 수가, 지연시간이 2- 싸이클타임인 비파이프라인형 연산자를 사용한 경우[15]와 동일함을 알 수 있다.

표3은 '+'연산의 체이닝에 의한 스케쥴링 결과로서 이 경우, 제어스텝의 시간간격을 100nS로 설정하였다. 임의의 latency로 동작하는 데이타패스의 구성에 필요한 자원의 수를, ALPS 시스템의 결과 및 Hwang[22]의 결과와 비교하였다. 본 연구 역시, ALPS 시스템과 마찬가지로 최적해를 보장하고 있음을 알 수 있으며, 또한, 데이타패스의 지연시간이 10인 경우는 Hwang의 결과와 비교해 볼 때, latency=3에서 한개의 '+'연산자와 한개의 '*'연산자가, latency=5에서 한개의 '*'연산자가, latency=7, 8, 9, 10 에서는

표 1. 파이프라인형 데이타패스(비파이프라인형 승산기 사용 시)
Table 1. Pipelined datapath with nonpipelined multiplier.

	latency	1	2	3		4	5	6	7	8	9	13	16	26
	소요 자원	26+ 16*	13+ 8*	9+ 6*	9+ 8*	7+ 4*	6+ 4*	5+ 3*	4+ 3*	4+ 2*	3+ 2*	2+ 2*	2+ 1*	1+ 1*
deley	ALPS	17	17	17		18	19	19	18	20	21	23	21	33
	*	17	17		17	18	19	19	18	20	21	23	21	33

표 2. 파이프라인형 데이타패스(파이프라인형 승산기 사용 시)
Table 2. Pipelined datapath with pipelined multiplier.

	latency	1	2	3	4	5	6	7	8	9	13	16
	소요 자원	26+ 16*	13+ 8*	9+ 8*	6+ 4*	5+ 4*	5+ 3*	4+ 3*	4+ 2*	3+ 2*	2+ 2*	2+ 1*
delay	*	20	20	20	20	21	23	22	23	24	26	24

표 3. 파이프라인형 데이타패스(연산의 체이닝)
Table 3. Pipelined datapath with operation chaining.

	latency	1	2	3	4	5	6	7	8	9	10
소요자원 (delay=9)	ALPS	26+ 8*	13+ 4*	9+ 4*	7+ 3*	8+ 4*	5+ 2*	6+ 2*	6+ 2*	4+ 2*	
	*	26+ 8*	13+ 4*	9+ 4*	7+ 3*	8+ 4*	5+ 2*	6+ 2*	6+ 2*	4+ 2*	
소요자원 (delay=10)	Hwang	N/A	13+ 4*	10+ 4*	7+ 2*	6+ 3*	5+ 2*	5+ 2*	6+ 2*	4+ 2*	4+ 2*
	*	26+ 8*	13+ 4*	9+ 3*	7+ 2*	6+ 2*	5+ 2*	4+ 2*	5+ 2*	3+ 2*	3+ 2*

한개의 '+'연산자가 감소되었음을 알 수 있다. 조건부분기 연산의 스케쥴링을 위한 benchmark 모델로서는 자료가 공개된 MAHA 모델[7]과 SEHWA 모델[21]을 택하였다.

표4는 이들의 스케쥴링 결과와 본 연구의 스케쥴링 결과를 비교한 것으로서 모두 연산의 체이닝에 의한 스케쥴링을 행하였다. 표의 내용 중, ( )안의 연산은 자원공유가 이루어진 상호배타적 연산집합을 나타낸다. MAHA 및 SEHWA 시스템의 결과와 본 연구의 결과는 모두, 2개의 가산기(adder)와 2개의 감산기(subtracter)를 필요로 하나, MAHA와 SEHWA 시스템은 2번의 조건부 자원공유를 행한 반면, 본 연구는 3번의 조건부 자원공유를 행하고 있음을 알 수 있다. 따라서, DFG내에 조건부 분기연산이 다수 존재하는 경우는 본 연구의 스케쥴링 방식이 MAHA나 SEHWA시스템의 스케쥴링 방식에 비해 보다 폭 넓은 조건부 자원공유를 이룰 수 있어, 보다 효율적인 자원의 절감을 꾀할 수 있다.

표 4. 조건부 분기연산의 자원공유
Table 4. Resource sharing for conditional branch operations.

	MAHA 모델				SEHWA 모델					
제어스텝	1	2	3	4	제어스텝	1	2	3	4	5
MAHA 시스템	+1 +4 −1 (−2, −3)	+2 +5 (−6, −5)	+6 +3 −7 −4	+7 +8 −8	SEHWA 시스템	−1 +2 −1 −6	+4 +7 (−3, −2) −4	(+3, +6, +5) +8 −5 −7		
*	+1 +3 −1 (−2, −3)	+5 (+2, +4 (−5, −6)	+6 −4 −7	+7 +8 −8	*	+1 +2 −1 (−5, −6)	+7 +8 (−2, −3) −7	+4 (+3, +5, +6) −4		

Ⅵ. 결 론

본 논문에서는 데이타패스 스케쥴링을 위한 새로운 ILP 모델을 제안하였다. 제안된 ILP 모델에는, 파이프라인형 데이타패스 합성 시 효과적으로 적용될 수 있도록 구조적 파이프라이닝과 기능적 파이프라이닝 스케쥴링을 위한 모델과 조건부 분기연산의 스케쥴링을 위한 모델이 포함되었다. 또한, 가상연산자 개념을 도입하여 멀티싸이클 연산의 스케쥴링을 효율적으로 행할 수 있는 방법을 제시하였다.

본 논문에서 제시한 ILP 모델의 변수는 $O(s \times n)$인 형태로 증가하고 생성되는 방정식은 $O(s \times m + n + r + e_f)$로 증가하기 때문에, 복잡도는 기존의 ILP 모델[15][18]과 거의 동일하다.

본 논문에서 제시한 ILP 모델의 스케쥴링 성능을 평가하기 위하여 표준 benchmark 모델인 fifth-order

digital wave filter에 대한 스케쥴링을 행한 결과, 기존의 ILP 모델에 비해 효율성과 정확성이 보다 뛰어남을 알 수 있었다.

參 考 文 獻

[1] C. Tseng, D.P. Siewiorek, "Automated synthesis of data path in digital system," *IEEE Tr.* CAD, Vol. CAD-5, pp. 379-365, July 1986.

[2] C.H. Gebotys, M.I. Elmasry, "A VLSI Methodology with testability constraints," *in Proc. 1987 Canadian Conf.*, pp. 271-277, July 1986.

[3] S.Y. Kung, H.J. Whitehouse, T.Kailath, *VLSI and Modern Signal Processing.* Englewood Cliffs, NJ:Prentice Hall, pp. 258-264, 1985.

[4] P. Mardewel, "A new synthesis algorithm for the MIMOLA software system," *in proc.* 23rd DAC, pp. 271-277, July 1986.

[5] H. Trickey, "Compiling pascal programs into silicon," Ph. D Dissertation, *Dept. Computer Science*, Stanford Univ., July 1985.

[6] E.F. Girczyc, "An ADA to standard cell hardware compiler based on graph grammers and scheduling," Proc. of ICCD-84, pp. 726-731, Oct. 1984.

[7] A.C. Parker et al, "MAHA:A program for datapath synthesis,"*Proc.of 23rd DAC.*, pp.461-466, 1986.

[8] P.G. Paulin et al, "HAL:A multi-paradigm approach to automatic data path synthesis," *Proc. of 23rd DAC.*, pp. 263-270, 1986.

[9] B.M. Pangrle, D.D. Gajski, "State synthesis and connectivity binding for microarchitecure compilation," *Proc. of ICCAD-86,* pp. 210-213, Nov. 1986.

[10] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Tr. CAD*, vol. 8, pp. 661-679, June 1989.

[11] M. Gary and D. Johnson, Computer and Intractability: A Guide to the NP-Com plete-ness, Freeman, 1979.

[12] L. Hafer, A.C. Parker, "A formal method for the specification, analysis, and design of register-transfer level digital logic,"*IEEE Tr. CAD*, vol. CAD-2, pp. 4-18, Jan. 1983.

[13] H. DeMan, J. Rabaey, P. Six, and L. Clasen, "Cathedral-II:A silicon compiler for digital signal processing," *IEEE Design & Test,* pp. 13-15, Dec. 1986.

[14] M. Balakrishnan, P. Marwedel, "Integrated scheduling and binding: A synthesis approach for design space exploration," *Proc. of 26th DAC.*, pp. 68-74, 1989.

[15] C.T. Hwang, et al. "A formal approach to the scheduling problem in high-level synthesis," *IEEE Tr. DAD,* vol. 10, no. 4, pp. 464-475, 1991.

[16] C.H. Gebotys, M.I. Elmasry, "A global optimization approach for architectural synthesis, " *Proc. of ICCAD-90,* pp. 258-261, 1990.

[17] C.A. Papachristou, H. Konuk,, "A linear program driven scheduling and allocation method followed by an interconnect optimization algorithm," *Proc. of 27th DAC.*, pp. 77-83, 1990.

[18] Y.S. Baek, Y.H. Bae, et al., "A new optimal scheduling algorithm," *Proc. of ICVC-91,* pp. 58-61, 1991.

[19] N. Park, Synhesis of high-speed digital systems, Ph. D thesis, University of Southern California, Oct. 1985.

[20] P.G. Paulin. "Scheduling and binding algorithm for High-Level synthesis," *Proc. of 26th DAC,* pp. 1-6, 1989.

[21] N. Park, A.C. Parker, "SEHWA: A program for synthesis of pipelines," *Proc. of 23rd DAC.*, pp. 454-460, 1986.

[22] K.S. Hwang, et al., "Scheduling and hardware sharing in pipelined data paths," *Proc. of ICCAD-89*, pp. 24-27, 1989.

[23] "LINGO: Optimization modeling language," *LINDO Systems Inc.*, 1991.

著 者 紹 介

李 近 萬 (正會員) 第29卷 A編 第 4 號 參照
현재 청주대학교 전자공학과 부교수

林 寅 七 (正會員) 第28卷 A編 第 2 號 參照
현재 한양대학교 전자공학과 교수