정수선형계획법의 반도체 설계자동화에의 응용 (Application of Integer Linear Programming on VLSI Design Automation)

백 영 석, 이 현 찬°
한국전자통신연구소, 홍익대학교

요 약

본 논문에서는 정수선형계획법을 반도체 설계 자동화 과정에 이용한 예를 보인다. 반도체 설계 자동화 과정은 매우 여러 단계를 거치게 되는데, 본 논문에서는 상위수준 합성중 스케쥴링(*scheduling*) 문제에 정수선형계획법을 응용하였다. 여기서 스케쥴링 문제는 설계 자동화의 초기단계에서 알고리듬으로 주어진 입력을 하드웨어 요소들로 표현하는 과정에서 매 제어단계(*control step*)에서 수행하여야 할 연산내용을 결정하는 문제이다. 스케쥴링의 목적함수는 주어진 제어단계 갯수내에서 하드웨어 비용의 최소화이다. 이를 위해 우선 *ASAP*(*As Soon As Possible*)과 *ALAP*(*As Late As Possible*) 방법을 이용하여 매 연산의 수행 시작이 가능한 가장 빠른 시간과 가장 늦은 시간을 구한다. 이 두 시간 사이가 각 연산의 *time frame* 이 되며 이를 이용하여 스케쥴링 문제를 정수 선형 계획법으로 공식화하여 풀었다. 이 공식화는 *chaining*, *multicycle* 연산, *pipeline data path*, *pipeline* 기능 유닛등에도 일반화하여 적용가능함을 보인다. 실험을 통해 본 공식화 방법이 기존 알고리듬에 의한 해보다 우수한 해를 제공함을 보인다. 비교를위해 잘 알려진 *benchmark* 회로인 *bandpass filter*를 이용하였는데 이 회로는 8개의 덧셈, 7개의 뺄셈 및 12개의 곱셈연산을 포함하고 있다. 제시된 알고리듬은 이 회로를 8개의 제어단계내에 총비용 675 (연산별 하드웨어 비용은 라이브러리로 주어짐)로 스케쥴링하였는데 이는 기존의 최상의 결과인 685보다 우수한 결과이다.

I. 서 론

최근 논리 합성 부분이 상용화되어 널리 상용되고 있는 가운데, 논리 합성의 상위 단계인 상위레벨 합성에대한 연구는 매우 활발하게 진행되고 있다. 상위레벨 합성에는 크게 제어 합성과 데이타 패스 합성의 두가지로 나눌 수 있다. 여기서, 데이타 패스 합성은 연산을 제어단계에 할당하는 스케쥴링과 하드웨어 자원을 각 연산에 실제적으로 할당하는 얼로케이션 (*allocation*)의 두가지로 이루어 진다. 스케쥴링은 설계에 있어서 비용과 속도의 절충점 (*trade-off*)을 결정한다. 한번 스케쥴링이 결정되면, 사용되는 기능 유닛의 갯수와 변수의 라이프 타임, 시간 제약조건이 결정된다. 따라서, 자동화 데이타 패스 합성에서 좋은 스케쥴러는 매우 중요하다.

본 연구에서는 주어진 전체 제어 단계하에서 다기능의 유닛의 라이브러리를 가지고 최적의 스케쥴링을 하는 알고리듬을 제안한다. 본 연구에서는 주어진 스케쥴링 문제를 *integer linear programming* (*ILP*) 식으로 모델링하여 상용의 *ILP* 툴을 사용하여 최적의 스케쥴링 결과를 산출한다. 이 스케쥴링 알고리듬에서는 서로 비용과 기능이 다른 다기능의 유닛을 고려하였다. 또한, 체이닝 연산, 멀티 사이클 연산, 파이프라인 기능 유닛, 파이프라인 데이타 패스에대한 *ILP* 식을 모델링 하였을뿐아니라, 부분 설계 방식에 대하여도 고려를 하였다.

II. 기존의 스케쥴링 알고리듬

스케쥴링에서 가장 간단한 형태는 *as soon as possible* (*ASAP*) 스케쥴링[1]으로 *data flow graph* (*DFG*)에 있는 연산들을 처음 제어 단계부터 차례로 할당하는 알고리듬이다. 여기서 각 연산들은 자신의 앞에 있는 연산(*predecessor*)들이 모두 스케쥴링되어 있어야만 다음 제어 단계에 스케쥴링된다.

리스트 스케쥴링 (*list scheduling*) [2, 3]은 기존의 많은 시스템에서 채택하고 있는 하드웨어 자원 제약하에서의 스케쥴링 방법으로 휴리스틱 우선순위 함수에 의하여 할당가능한 연산을 순서화된 리스트로 만든다. 이 리스트에 따라서 주어진 하드웨어 자원을 만족하는 범위 안에서 각 제어 단계에서 순서화된 리스트 순으로 연산을 스케쥴한다.

Force-directed 스케쥴링[4]은 후크의 힘의 법칙을 응용한 것으로 스케쥴이 가능한 모든 제어 단계에 대하여 모든 연산에대한 힘(*force*)의 값을 계산하고, 가장 끄는 힘이 큰 연산과 제어단계의 쌍을 할당한다.

Force-directed list 스케쥴링[5]은 리스트 스케쥴링의 하나로 우선순위 함수로 힘의 값을 사용하는 방법이다. 이 방법은 주어진 하드웨어 자원하에서 스케쥴링하는 방법이다.

상위의 스케쥴링 방법은 한번에 하나의 연산을 제어 단계에 할당하는 방법으로 그 할당 순서에따라서 스케쥴링 결과가 크게 차이가 나므로, 상위의 스케쥴링 알고리듬은 최적의 스케쥴링 알고리듬이 아니다. 따라서, 전체적으로 최적의 해를 얻으려는 연구가 진행되고 있으며, 이의 한 방법으로 *ALPS* 시스템[6]에서는 주어진 스케쥴링 문제의 제약 조건을 *ILP* 식으로 모델링하여 *ILP* 툴을 이용하여 최적의 해를 얻는다. 그러나, *ALPS* 시스템은 각 연산에 사용하는 기능 유닛이 설계자에의하여 이미 결정되어야

하며, 기능 유닛의 기능이 서로 배타적이어서 다기능의 기능 유닛에대한 스케줄링은 할 수 없는 단점이 있다. *ILP* 기법을 사용하여 다기능의 유닛을 스케줄링 방법으로 *ADPS* 시스템[7]이 있으나, 이 방법은 스케줄링 문제를 전체적으로 모델링하지 않고, 각 제어 단계별로 *ILP* 식으로 모델링하여 스케줄링하므로, 최적의 결과를 얻을 수 없는 단점이 있다.

따라서, 본 연구에서는 *ALPS* 시스템을 기본으로하여 다기능 유닛에대하여 스케줄링을 행하는 새로운 스케줄링 알고리듬을 발표한다. 본 스케줄링은 먼저 *ASAP*, *ALAP* 스케줄링으로 각 연산이 할당가능한 범위(*time frame*)을 결정하고, 이를 가지고 *ILP* 식으로 모델링하여 최적의 해를 구한다.

III. 스케줄링 알고리듬

본 연구에서는 주어진 전체 제어 단계하에서 다기능 유닛의 라이브러리를 가지고 최적의 스케줄링을 하는 알고리듬을 제안한다. 본 연구에서는 주어진 스케줄링 문제를 *ILP* 식으로 모델링하여 상용의 *ILP* 툴을 사용하여 최적의 스케줄링 결과를 산출한다. 이 스케줄링 알고리듬에서는 서로 비용과 기능이 다른 다기능의 유닛을 고려하였다. 또한, 체이닝 연산, 멀티 사이클 연산, 파이프라인 기능 유닛, 파이프라인 데이타 패스에대한 *ILP* 식을 모델링 하였을뿐아니라, 부분 설계 방식에 대하여도 고려를 하였다.

본 연구는 시간 제약 조건하에서 각 연산들을 스케줄링하는 방법으로 *ALPS* 시스템[6]을 기본으로하여 확장한 스케줄링 알고리듬이다. *ALPS* 시스템은 주어진 제약 조건들을 *ILP* 식으로 모델링하여 최적의 해를 얻고 있으나, 이 방법은 기능 라이브러리의 각 기능들이 서로 배타적, 즉, 하나의 기능이 어느 하나의 기능 유닛에만 존재하는 단기능 유닛 (*single functional unit*)이어야 하므로 실용적인 측면에서 문제점이 있다. 따라서, 본 연구에서는 *ALPS* 시스템을 확장하여 각 기능들이 다수의 기능 유닛에서 행할 수 있는 다기능 (*multifunctional unit*)의 경우에 대하여 스케줄링이 가능하도록 하였다.

1. 용어 정의

본 연구에서 사용하는 용어를 정의하면 다음과 같다.

1) M_k : 기능 유닛의 종류 k의 필요한 수 를 나타내는 정수형 변수.
2) c_k : 최적화 시키고자하는 라이브러리의 기능 유닛 k의 비용(cost).
3) o_i : 연산을 구분하기 위하여 i로 인덱스된 연산.
4) x_{ijk} : 연산 o_i와 연관된 0-1 정수형 변수로, o_i가 형태 k의 기능 유닛으로 제어단계 j에 할당되면 $x_{ijk}=1$이고 그렇지 않으면 $x_{ijk}=0$.
5) S_i : 연산 o_i가 *ASAP* 스케줄링으로 할당된 제어단계.
6) L_i : 연산 o_i가 *ALAP* 스케줄링으로 할당된 제어단계.
7) C_{jk} : 기능 유닛 k를 가지고 제어단계 j에서 실현할 수 있는 연산들의 집합.
8) Q_i : 연산 o_i를 실현할 수 있는 기능유닛들의 집합.

2. *ILP* 식으로 모델링된 스케줄링 알고리듬

본 연구에서 행한 스케줄링 문제를 *ILP* 식으로 모델링하면 다음과 같다.

$$Minimize \quad \sum_{k=1}^{m} c_k * M_k \tag{1}$$

subject to

$$\sum_{j=1}^{s}\sum_{k=1}^{m}\Big[\sum_{o_i \epsilon C_{jk}} x_{ijk} \le M_k\Big] \tag{2}$$

$$\sum_{i=1}^{n}\Big[\sum_{j=S_i}^{L_i}\sum_{k \epsilon Q_i} x_{ijk} = 1\Big] \tag{3}$$

$$\sum_{j=S_i}^{L_i}\sum_{k \epsilon Q_i} j*x_{ijk} - \sum_{j=S_r}^{L_r}\sum_{k \epsilon Q_r} j*x_{rjk} \le -1, \quad \text{for } o_i \rightarrow o_r \tag{4}$$

식 (1)은 최적화 시키는 목적 함수로 기능유닛의 전체 비용의 최적화를 나타낸다. 이 식은 기능유닛들이 완전히 사용되고, 공유될때 최적화가 된다. 즉, 같은 형태의 연산들이 모든 제어단계에 골고루 분포되어야 한다. 식 (2)는 모든 제어단계에서 기능유닛 k의 사용 갯수는 M_k 이내에서 사용되어야 하는 제약 조건을 나타낸다. 식 (3)은 연산 o_i가 S_i와 L_i 사이에서 오직 하나의 제어단계에 할당되어야 하는 제약 조건을 나타낸다. 식 (4)는 *DFG*에서의 데이타 의존성 (*data dependency*)을 만족하는 제약 조건을 나타낸다. 여기서, $o_i \rightarrow o_r$은 연산 o_i가 연산 o_r의 직접 상위 연산 (*immediate predecessor*)임을 나타낸다.

3. *ALPS* 시스템과 본 연구의 스케줄링 알고리듬과의 차이점

본 연구와 *ALPS* 시스템과의 큰 차이점은 다음과 같다. *ALPS* 시스템에서는 각 연산에 최적의 제어단계를 할당하지만 이 연산이 사용할 기능 유닛의 종류가 이미결정된 상태이고, 본 연구에서 개발된 스케줄링 알고리듬은 각 연산에 최적의 제어단계와 최적의 기능 유닛을 할당한다는 것이다. 따라서, *ALPS* 시스템에서 사용하는 변수는 연산, 제어단계의 2차원이지만, 본 연구에서 사용하는 변수의 차원은 연산, 제어단계, 기능 유닛의 3차원이다.

ALPS 시스템과 본 연구의 스케줄링 알고리듬상의 *ILP* 식의 차이점은 이러한 기능 유닛을 고려한 차이이다. 상위의 식(1)의 목적 함수는 *ALPS* 시스템과 서로 동일하다. 그러나, 식(2)에서 *ALPS* 시스템과의 차이점은 상술한 것과 같이 *ALPS* 시스템에서는 각 연산이 행할 기능 유닛이 이미 결정되어 있으므로 기능 유닛에 대하여 고려하지 않아도 되지만, 본 연구의 스케줄링 알고리듬은 각 연산이 행하는 최적의 기능 유닛을 결정하므로 이에 대하여 모델링 (k)을 하였다. 식(3)에서 *ALPS* 시스템과의 차이점 역시, *ALPS* 시스템은 각 연산이 행할 기능 유닛이 이미 결정되어 있으므로 각 연산이 타임 프레임 범위내의 하나의 제어단계에 할당되면 되지만, 본 연구의 스케줄링 알고리듬은 기능 유닛의 할당을 위하여, 각 연산이 타임 프레임내의 제어단계에 연산의 기능을 행할 수 있는 기능 유닛의 집합가운데 하나의 기능 유닛으로 할당되어야 함을

나타낸다. 식 (4)에서 *ALPS* 시스템과의 차이점 역시 기능 유닛의 고려에 있다. *DFG*상에서 데이타 의존성이 있는 두 연산에 대하여 *ALPS* 시스템은 각 연산을 행할 기능 유닛이 결정되어 있으므로 기능 유닛의 고려가 없고, 본 연구의 스케쥴링 알고리듬은 기능 유닛의 할당을 고려하여 데이타 의존성이 있는 상위 연산이 하위 연산보다 최소한 1 제어단계 이상 전에 실행되어야 함을 나타낸다.

4. 스케쥴링의 결과 예

그림 1은 *HAL* 논문의 2차 미분 방정식에대한 스케쥴링 결과를 나타낸다. 이에대한 *DFG*가 전체 4의 제어단계로 스케쥴링하였으며, 이 *DFG*의 *ASAP* 과 *ALAP* 스케쥴링은 그림 1의 (a)와 그림 1의 (b)에 보여준다. 이 *ASAP* 과 *ALAP* 스케쥴링에의한 타임 프레임은 그림 1의 (c)에 보여준다. 본 *DFG*의 기능 유닛으로 다음과 같은 다기능 유닛을 입력으로 고려하였다.

$F_1 = \{+\}$, $F_2 = \{-\}$, $F_3 = \{<\}$,
$F_4 = \{*\}$, $F_5 = \{+, -\}$, $F_6 = \{+, *\}$,
$F_7 = \{-, *\}$, $F_8 = \{+, -, <\}$, $F_9 = \{+, -, *\}$

여기서, 각 기능 유닛들의 비용을 각각 50, 60, 55, 250, 75, 275, 280, 120, 305 라고 하였다. 상위의 조건을 *ILP* 식으로 모델링하고, *ILP* 툴로 해를 얻으면, $x_{1,1,4}$, $x_{2,1,6}$, $x_{3,2,1}$, $x_{4,3,6}$, $x_{5,1,8}$, $x_{6,2,6}$, $x_{7,3,4}$, $x_{8,4,6}$, $x_{9,2,8}$, $x_{10,3,8}$, $x_{11,4,8}$, M_4, M_6, 과 M_8이 1의 값을 가지고 나머지는 모두 0의 값을 갖는다. 상위의 목적함수의 비용은 645의 값을 갖는다. 상위의 스케쥴의 결과는 그림 1의 (d)에 보여주며, 이 결과는 최적이다.

5. 스케쥴링 알고리듬의 복잡도

본 연구는 *ILP* 툴을 이용하므로 스케쥴링 알고리듬의 복잡도를 변수의 수와 제약조건식의 수에대하여 분석하기로 한다. 식에서 0-1 정수형 변수 $x_{i,j,k}$의 수는 $\sum_{i=1}^{n}(L_i - S_i + 1)\cdot K(i)$ 이며, 여기서 $K(i)$는 연산 o_i를 행할 수 있는 기능 유닛들의 갯수, 즉 $|Q(i)|$ 이다. 이 값들은 $(n \cdot s \cdot m)$의 값 범위안에 있다. 여기서, n 은 전체 연산들의 갯수이고, s 는 주어진 제어단계의 수 이고, m 은 전체 기능유닛의 가지수 이다. 제약조건식의 수에서는 식 (2)에서의 제약조건의가지수는 $(s \cdot m)$이고, 식 (3)에서의 제약조건의 가지수는 n이고, 식 (4)에서의 제약조건의 가지수는 e이다, 여기서 e는 *DFG*의 에지(*edge*)의 수를 나타낸다.

따라서, 본 연구의 식에서 사용되는 변수에대한 복잡도는 $O(n \cdot s \cdot m)$이고, 제약조건의 식에대한 복잡도는 $O(s \cdot m + n + e)$이다.

IV. 일반화

실용적인 사용을 위하여, *ILP* 식을 다음에 대하여 일반화 시킨다.

- o 체이닝 연산
- o 멀티 사이클 연산

o 파이프라인 데이타 패스
o 파이프라인 기능 유닛
o 부분 설계 방법

1. 체이닝 연산

어느 경우에는 여러개의 연산의 전체 실행 시간이 하나의 주기 시간보다 적어서 그 연산들이 하나의 주기안에서 행해질 수 있다. 이러한 연산들을 체이닝 연산이라고 하며, 이러한 스케줄링을 체이닝 연산을 가진 스케줄링이라고 한다. 같은 기능 유닛으로 실현 가능한 체이닝 연산들은 같은 제어단계에서 같은 기능 유닛을 공유할 수 있다. 예로 연산 o_i와 연산 o_r이 기능 유닛 k로 제어단계 j에서 스케줄된다면, 이에대한 기능 유닛의 갯수를 2개보다는 1개로 계산하여야 한다. 이러한 제약조건을 0-1 변수인 y_{jk}를 새로이 사용함으로 고려한다. 즉, $x_{ijk}=x_{rjk}=0$ 이면 y_{jk}는 0의 값을 가지며, 그렇지 않으면 1의 값을 가지도록 한다. 이것은 식 (2)에서 "$x_{ijk}+x_{rjk}$"가 나오는 곳에 이것대신 "y_{jk}"로 대치를 하고, 다음과 같은 제약 조건을 추가함으로 고려된다. 여기서, o_i to_r는 연산 o_i가 연산 o_r의 직접 상위 연산이며, 두 연산이 체이닝이 가능한 것을 나타낸다.

$$x_{ijk}+x_{rjk}-2y_{jk}\le 0,\quad \text{for } o_i\ t o_r$$

또한 식 (4)는 다음과같이 수정된다.

$$\sum_{j=S_i}^{L_i}\sum_{k\in Q_i} j^*x_{ijk}-\sum_{j=S_r}^{L_r}\sum_{k\in Q_r} j^*x_{rjk}\le 0,\quad \text{for } o_i\ t o_r$$

$$\sum_{j=S_i}^{L_i}\sum_{k\in Q_i} j^*x_{ijk}-\sum_{j=S_r}^{L_r}\sum_{k\in Q_r} j^*x_{rjk}\le 1,\quad \text{for } o_i\rightarrow o_r.$$

2. 멀티 사이클 연산

어느 경우에는 연산의 지연시간이 주어진 주기시간보다 커서 하나의 제어단계에서 실행할 수 없고 여러개의 주기 시간이 필요한 경우가 있다. 이러한 연산을 멀티 사이클 연산이라고 하며, 이러한 연산의 스케줄링을 멀티 사이클의 스케줄링이라고 한다.

예로 연산 o_i 가 멀티 사이클 지연 시간 d_i를 가지고 있다고 하자. 멀티 사이클 연산 o_i 를 실행하는데는 d_i 의 제어단계가 소요되므로, 연산 o_i의 기능 유닛은 d_i 제어 단계 동안은 다른 연산들과 공유할 수 없다. 따라서, 각 기능 유닛에대한 각 제어단계에서 필요한 기능 유닛의 수는 d_i 제어단계 동안의 기능 유닛의 수의 합이다. 따라서, 식 (2)는 다음과 같이 수정 된다.

$$\sum_{j=1}^{s}\sum_{k=1}^{m}\Big[\sum_{o_i\in C_{jk}}\sum_{p=0}^{d_i-1} x_{ij-pk}\le M_k\Big].$$

또한, 멀티 사이클 연산 o_i는 d_i의 제어단계가 소요되므로 데이타 의존성에의하여, 연산 o_i의 직접 하위 연산 (*immediate successor*)은 연산 o_i가 실행된후 최소한 d_i 제어단계후에 실행되어야 한다. 따라서, 식 (4)는 다음과 같이 수정된다.

$$\sum_{j=S_i}^{L_i} \sum_{k \in Q_i} j^* x_{i\,j\,k} - \sum_{j=S_r}^{L_r} \sum_{k \in Q_r} j^* x_{r\,j\,k} \leq -d_i, \quad \text{for } o_i \rightarrow o_r.$$

3. 파이프라인 데이타 패스

파이프라인 데이타 패스는 동시에 여러개의 일을 수행하는 것을 허용한다. 두개의 연속적인 일은 임의의 간격을 가지고 다시 수행될 수 있는데, 이 간격을 파이프라인 데이타 패스의 대기 시간 (*laytency*) 이라고 한다.

대기 시간이 l로 주어진 경우, $j+pl$ (p = 0,1,2,...) 제어단계에 있는 연산들은 동시에 실행이 되고, 이 연산들은 같은 기능 유닛을 공유할 수 없다. 따라서, 식 (2)는 다음과 같이 수정 된다.

$$\sum_{p=0}^{\lfloor \frac{s-j}{l} \rfloor} \sum_{o_i \in C_{j+pl\,k}} x_{i\,j+pl\,k} \leq M_k.$$

4. 파이프라인 기능 유닛

기능 유닛가운데 어느 유닛은 기능 유닛의 실행이 다 끝나지 않았으나 일정한 대기 시간후에 다시 입력을 받을 수 있으며, 이러한 기능 유닛을 파이프라인 기능 유닛이라고 한다. 즉, 지연 시간이 d이고, 대기 시간이 l인 파이프라인 기능 유닛은 매 l 주기마다 새로운 연산을 실행할 수 있다($l \leq d$). 파이프라인 기능 유닛은 두개의 제어단계 s_i, s_j의 연산에의하여 공유될 수 있다. 여기서, $|s_i - s_j|$ 은 l의 정수 배이다. 따라서, 식 (2)는 다음과 같은 식으로 수정된다.

$$\sum_{p=0}^{l_k-1} \left[\sum_{q=0}^{\lfloor (d_k-p-1)/l_k \rfloor} \sum_{o_i \in C_{j-ql_k-p\,k}} x_{i\,j-ql_k-p\,k} \leq f_p \right]$$

$$\sum_{p=0}^{l_k-1} f_p \leq M_k.$$

다기능 유닛을 고려한 경우, 상위의 두 식만으로는 부족하다. 상위의 식은 파이프라인 기능유닛의 입력 부분에서의 필요한 기능 유닛의 갯수이지만, 다기능에서는 각각의 기능이 서로 다른 지연시간을 가지므로 입력에서는 공유할 수 있으나 출력

단자를 공유하지 못하는 경우가 발생한다. 즉, 같은 제어단계에서 연산이 같은 파이프라인 기능 유닛을 가지고 끝나는 경우에는 주어진 대기 시간을 만족하더라도 파이프라인 기능 유닛을 공유할 수는 없다. 따라서, 파이프라인 기능 유닛인 M_k에 대하여 다음의 식이 추가되어야 한다.

$$\sum_{j=0}^{} [\sum_{o_i \epsilon C_{j-d_i+1,k}} x_{i,j-d_i+1,k} \leq M_k]$$

5. 부분 설계 방법

데이타 패스 부를 설계시, 데이타 패스부의 일부 연산을 특정한 기능 유닛으로 고정 시키거나, 임의의 제어단계로 고정 시키거나, 특정한 기능 유닛과 제어 단계를 고정 시킬 필요가 있다. 이러한 부분을 고려하여 설계하는 것을 부분 설계라 한다. 부분 설계를 고려한 스케줄링 방법에는 다음의 세가지 방법이 있다.

1) 기능 유닛을 고려한 부분 설계

임의의 연산 o_i를 특정한 기능 유닛으로 고정 시킨다는 것은 연산 o_i를 실현할 기능 유닛의 종류가 F_k 하나뿐 이라는 것과 같다. 따라서 용어 정의의 Q_i의 값을 다음과 같이 수정하여 고려 할 수 있다.

$$Q_i = k$$

2) 제어 단계를 고려한 부분 설계

임의의 연산 o_i를 특정한 제어 단계 j로 고정 시킨다는 것은 연산 o_i는 반드시 주어진 제어 단계 j에서 실행 되어야 한다는 것과 같다. 따라서, 이것은 연산 o_i의 타임 프레임이 주어진 제어단계와 같은 결과이므로, 연산 o_i의 타임 프레임을 다음과 같이 수정 하여 고려 한다.

$$S_i = j$$
$$L_i = j$$

3) 기능 유닛과 제어 단계를 고려한 부분 설계

임의의 연산 o_i를 특정한 기능 유닛 F_k로 주어진 제어 단계 j로 고정 하여 설계하는 방법은 1)과 2)의 경우를 같이 고려한 것이므로 다음과 같이 고려할 수 있다.

$$Q_i = k$$
$$S_i = j$$
$$L_i = j$$

V. 실 험

본 연구의 알고리듬은 C 언어로 SUN4/330에서 실현되었다. 본 스케줄링 알고리듬을 테스트하기위하여 *ALPS* 시스템의 예를 가지고 체이닝, 멀티 사이클, 파이프라인 데이타 패스, 파이프라인 기능 유닛에 대하여 스케줄링을 행하였다. 이 결과는 *ALPS* 시스템의 결과와 같은 최적의 스케줄링 결과를 산출하였다. 다기능 유닛에 대한 테스트를

위하여, 다기능 유닛에대하여 고려한 *ADPS* 시스템과 비교를 행하였다. 사용한 벤치마크 데이타로는 *ADPS*시스템의 실험예를 사용하였으며, band pass filter, elliptic wave filter, facet 예에대한 결과가 표 1에 있다. Band pass filter의제어 단계가 8로 주어진 경우를 살펴보면, 본 연구에서 사용한 다기능 유닛을 고려한 스케쥴링 알고리듬이 *ADPS* 시스템보다 뛰어남을 알수 있다. 이 스케쥴링의 결과는 그림 2에 보여준다.

VI. 결 론

본 연구에서는 시간 제약 조간하에서의 다기능 유닛을 고려한 새로운 스케쥴링 알고리듬을 제안 하였다. 본 스케쥴링 알고리듬은 *ASAP*, *ALAP* 과 *ILP* 스케쥴링으로 이루어져 있다. 본 연구에서 모델링한 *ILP* 식들은 다기능 유닛에 있어서 매우 효과적이며, 이 식의 복잡도는 변수의 경우에 $O(n \cdot s \cdot m)$이다. 여기서, n, s, m은 각각 연산의 수, 주어진 제어 단계의 수 그리고 라이브러리의 기능 유닛의 가지수를 나타낸다. 본 연구에서는 또한, 실용적인 면을 위하여, 체이닝, 멀티 사이클, 파이프라인 데이타 패스 그리고, 파이프라인 기능 유닛에대하여 *ILP* 식으로 모델링을 하였으며, 부분 설계에대하여도 모델링을 하였다.

참 고 문 헌

[1] C.Y. Hitchcock and D.E. Thomas, " A Method of Automatic Data Path Synthesis," in *Proc. 20th Design Automation Conf.*, pp. 484-489, 1983.

[2] B.M. Pangrle and D.D. Gajski, "State Synthesis and Connectivity Binding for Microarchitecture Compilation," in *Proc. ICCAD*−86, pp. 210-213, 1986.

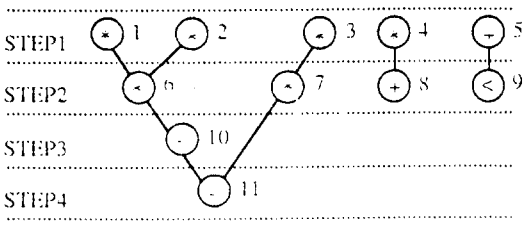
[3] N. Park and A.C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications," *IEEE Trans. on Computer−Aided Design*, pp. 356-370, 1988.

[4] P.G. Paulin and J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Trans. on Computer−Aided Design*, pp. 660-679, 1989.

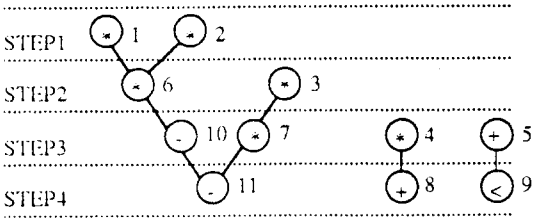
[5] P.G. Paulin and J.P. Knight, "Scheduling and Binding Algorithms for High-Level Synthesis," in *Proc. 26th Design Automation Conf.*, pp. 1-6, 1989.

[6] J.H. Lee, Y.C. Hsu, and Y.L. Lin, "A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis," in *Proc. ICCAD*−89, pp. 20-23, 1989.

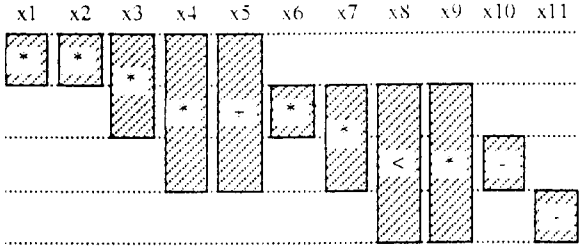
[7] C.A. Papachristou and H. Konuk, "A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnect Optimization Algorithm," in *Proc. 27th Design Automation Conf.*, pp. 77-83, 1990.



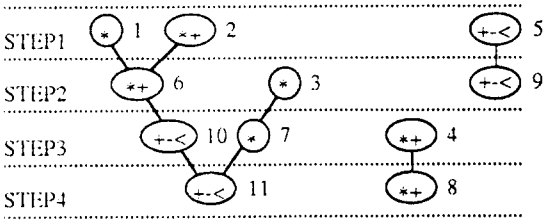
(a) ASAP 스케줄링



(b) ALAP 스케줄링



(c) 타임 프레임



(d) 스케줄링 결과

그림1. HAL 예의 스케줄링

Example	# of control steps	ADPS	FDS	FDLS	OURs	Facet
Band pass filter	8	685			675	
	9	650			650	
Elliptic filter	17	850	600	900	850	
	18	600	650	600	600	
Facet example	4	595			595	667

표1. 스케줄링결과의 비교

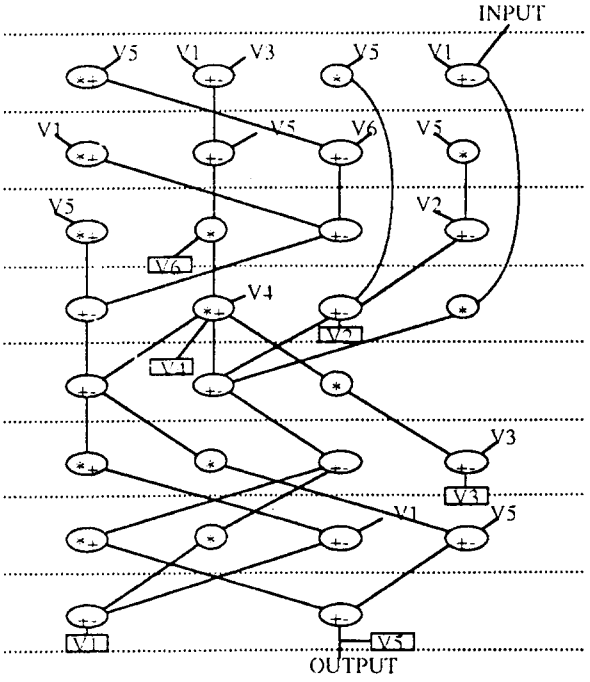


그림 2. 제어단계 8에서의 band pass filter의 스케줄링 결과