상위레벨 회로합성을 위한 자원제한 스케줄링 알고리즘

A Resource-Constrained Scheduling Algorithm for High Level Synthesis

황 인 재*
In-Jae Hwang*

요 약

스케줄링은 CDFG 내의 각 연산에 우선순위 관계를 유지하면서 연산이 수행될 제어스텝을 할당하는 과정으로 합성된 하드웨어의 성능에 직접적인 영향을 미치는 중요한 단계이다. 본 논문에서는 자원제한 스케줄링 알고리즘을 제안한다. 제안된 알고리즘은 주어진 그래프를 분석하여 연산유닛의 개수를 결정하고 이에 따라 각 연산을 제어스텝에 할당한다. 스케줄링 과정 중에 상대적으로 부족한 연산유닛과 여유 있는 연산유닛을 구별하여 연산유닛의 수를 조절한 후 반복적으로 성능개선을 시도하게 된다. 제안된 알고리즘의 성능을 평가하기 위하여 모의실험을 수행하였고 그 결과는 기존의 방법들에 비해 우수함을 알 수 있었다.

Abstract

Scheduling for digital system synthesis is assigning each operation in a control/data flow graph(CDFG) to a specific control step without violating precedence relation. It is one of the most important tasks due to its direct influence on the performance of the hardware synthesized. In this paper, we propose a resource-constrained scheduling algorithm. Our algorithm first analyzes the given CDFG to determine the number of functional units of each type, then assigns each operation to a control step while satisfying the constraints. It also tries to improve the solution iteratively by adjusting the number of functional units using the results collected from the previous scheduling. Experiments were performed to test the performance of the proposed algorithm, and results are presented

Key words : Scheduling, High level synthesis, Control/data flow graph, functional unit

I. 서론

VLSI 기술의 눈부신 발달로 수백만개 이상의 트랜지스터가 하나의 칩에 집적되고 있으며 회로 시뮬레이션, 배치, 할당, 라우팅 등의 하위레벨에서의 여러 가지 설계 문제를 성공적으로 해결하고 있다. VLSI 집적도의 증가와 발달된 기술은 주문형 반도체(Application Specific Integrated Circuits)의 크기와 복잡도를 크게 증가시켰다. 이러한 이유로 하드웨어 설계에서 게이트 또는 트랜지스터 단위가 아닌 functional unit 단위의 설계를 통하여 회로의 기능 명세 및 최적화를 달성하기 위한 상위레벨 회로합성 기술이 그 중요성을 더하고 있다.

상위레벨 회로합성은 control data flow graph (CDFG)를 통하여 회로의 동작을 기술하고 이로부터 최적화된 레지스터 전송레벨의 구조를 얻는 과정을 의미한다. 회로가 구현할 알고리즘을 나타내는 CDFG에서 노드는 연산을 나타내고 두개의 노드를 잇는 간선은 데이터 값이나 제어흐름의 의존을 나타낸다. 회로합성 결과는 알고리즘을 구현하는 datapath 회로와 각 제어스텝에 따라 datapath 회로의 데이터 흐름을 제어하는 제어회로이며 제어회로는 주로 finite state machine 으로 표현된다.

*충북대학교 컴퓨터교육과
접수 일자 : 2005. 1. 4 수정 완료 : 2005. 1. 25
논문 번호 : 2005-1-1

회로합성작업은 분할, 스케줄링, 할당 등으로 나누어질 수 있다. 분할은 회로를 각 구성 부분들이 최적의 기능을 가지도록 하면서 2개 이상의 그룹으로 나누는 것을 말한다. 스케줄링이란 각각의 CDFG 노드를 데이터와 제어흐름의 의존도에 따라 제어스텝에 할당하는 과정이다. 스케줄링의 목적은 CDFG로 표현된 알고리즘을 수행하는데 필요한 제어스텝의 수를 줄이거나 요구되는 하드웨어를 최소화하는데 있다. 할당에서 하는 일은 CDFG의 노드와 간선들을 자원충돌이 생기지 않고 연산을 수행할 수 있도록 레지스터, 연산유닛 및 버스와 같은 하드웨어의 각 부분에 할당한다.

본 논문에서는 상위레벨 회로합성에 필요한 스케줄링 알고리즘을 제안한다. 스케줄링 문제는 크게 시간제한 스케줄링과 자원제한 스케줄링으로 분류될 수 있다. 시간제한 스케줄링의 경우는 정해진 제어스텝수 안에 주어진 연산을 모두 마쳐야 하는 제한이 따르며 주어진 조건을 만족하면서 필요한 하드웨어 자원을 최소화하는 것이 목표이다. 실시간 시스템 또는 DSP(Digital Signal Processing) 등에 필요한 하드웨어 설계에 주로 사용된다. 자원제한 알고리즘의 경우에는 주어진 하드웨어 자원만을 사용하여 최소한의 제어스텝수 안에 주어진 연산을 모두 마치도록 하는 것이 목표이다.

대부분의 자원제한 스케줄링 알고리즘은 각 종류의 연산을 수행할 수 있는 연산유닛 수가 주어진 것으로 가정하고 이에 따라 최적의 해를 찾으려 시도한다. 그러나 주문형 반도체 제작시 제한된 실리콘 공간에 정해진 알고리즘을 수행하는 회로를 구현해야 한다면 이에 필요한 최적의 연산유닛 수를 결정하는 것은 쉽지 않은 일이다. 본 논문에서는 가용한 실리콘 공간의 크기만이 주어진 경우 각 연산 종류별 연산유닛의 개수를 결정하고, 이에 따라 주어진 CDFG내의 연산들을 스케줄링하는 알고리즘을 제안한다. 제안된 알고리즘에서는 각 연산 종류의 연산유닛을 만드는데 필요한 실리콘 공간의 크기와 CDFG를 분석한 결과를 이용하여 전체 실리콘 공간에 구현할 수 있는 각 연산유닛의 개수를 결정하고 이를 이용하여 연산들을 제어스텝에 할당한다. 초기 스케줄링 후 연산유닛수를 약간 수정하여 더 나은 스케줄링이 가능한가를 반복적인 방법을 통하여 확인한다.

본 논문은 구성은 다음과 같다. 다음장에서는 관련 연구를 소개하고, 3장에서는 스케줄링 문제를 공식적으로 정의한다. 4장에서는 제안된 상위레벨 회로합성을 위한 자원제한 알고리즘을 상세히 기술한다. 5장에서는 본 논문에서 제안한 알고리즘의 성능 평가를 위한 실험 결과를 제시하고 끝으로 6장에서 결론을 기술한다.

II. 관련연구

스케줄링은 상위레벨 회로 합성에 있어 합성되는 회로의 성능에 가장 직접적인 영향을 미치는 중요한 단계이다. 스케줄링시에는 연산유닛의 크기, 사용빈도, 각 연산마다 서로 다른 제어스텝 수 등의 많은 어려움이 있다. 따라서 합성되는 회로의 성능개선을 위하여 많은 연구가 이루어 졌다. 이 장에서는 그중 관련된 몇 개를 정리하였다.

Sllame과 Drabek은 리스트 스케줄링을 기반으로한 새로운 스케줄링 알고리즘을 제안하였다.[1] 이 알고리즘은 CDFG 구조로부터 얻어진 정보를 이용하여 스케줄러가 최적에 근사한 해를 얻도록 해준다. CDFG 구조분석 결과는 노드의 선행연산 후행연산 관계, 총 후행연산의 수, 구성된 그래프의 출력으로부터 얻어지는 트리 등이 있다.

Yang과 Peng은 스케줄링과 datapath 할당을 통합하여 접근하는 방법으로 제안했다.[2] 위의 두 태스크를 통합함으로써 스케줄링과 할당의 효과가 보다 더 효율적으로 활용되었다. 이러한 접근은 VHDL의 동작적 기술로부터 RT(Register Transfer) 레벨의 효과적 구현을 위한 설계에 일련의 시맨틱 보존 변환을 적용하는 알고리즘에 기초를 두고 있다.

Achatz는 상위레벨 회로합성에서의 스케줄링 문제에 대한 새로운 접근방법을 제안했다.[3] 최대한 병렬적인 CDFG로부터 시작하여 연속적으로 에지를 그래프에 더해가는 방법으로 모든 하드웨어적 제한을 지키도록 했으며 여러 가지의 다양한 스케줄링 선택을 만들어낸다. 이어지는 바인딩 알고리즘은 이들을 이용하여 보다 더 낳은 해를 찾게 된다.

Kolling, Al-Hashimi와 Abbott는 시간제한적 스케줄링 문제에 대한 heuristic 알고리즘을 소개하였다.[4] 이 알고리즘은 분산 그래프에 기반을 두고 있으며 mean square error 함수를 연산의 순차적인 스케줄링에 사용한다.

스케줄링이 상위레벨 회로 합성에 있어 가장 중요한 단계라 할 수 있는 만큼 많은 선행 연구가 있었으나 [5][6][7], 자원이 제한되는 경우 각 연산 종류별 연산유닛 수의 결정에 관한 연구는 아직 없었다. 본 논문에서는 합성되는 회로가 높은 효율을 내도록 각 연산 종류별로 연산유닛의 수를 결정한 다음 이에 따라 주어진 CDFG를 스케줄링하는 알고리즘을 제시한다.

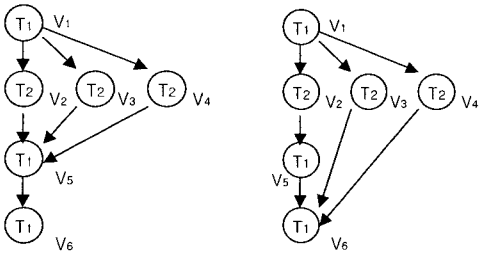
III. 자원제한 스케줄링 문제 정의

스케줄링이란 CDFG에서 노드로 나타내지는 각 연산에 그 연산이 수행될 제어스텝을 할당하는 과정을 말한다. 하나 이상의 연산이 같은 하드웨어에서 같은 제어스텝 동안에 수행될 수 없으므로 연산에 제어스텝을 할당할 때에는 연산들의 상호 의존관계와 더불어 하드웨어 자원의 가용여부도 함께 고려하여야 한다. 스케줄링 문제는 제한요소와 최적화 대상에 따라 시간제한 스케줄링과 자원제한 스케줄링으로 나눌 수 있다.

시간제한 스케줄링에서는 정해진 제어스텝 내에 CDFG가 나타내는 알고리즘을 최소한의 자원만을 사용하여 수

행하는 것을 목표로 한다. 자원제한 스케줄링에서는 주어진 하드웨어 자원만을 사용하여 최소한의 제어스텝내에 연산의 수행을 마치는 것이 목표이다. 이때 하드웨어 자원은 각 연산유닛의 개수로 주어지는 것이 일반적이나 주문형 반도체 제작시 제한된 실리콘 공간에 정해진 알고리즘을 수행하는 회로를 구현해야 한다면 연산유닛의 수가 주어진 다는 것은 지나친 가정일 수 있다. 실제로 각 연산유닛의 수에 따라 알고리즘을 마치는데 필요한 제어스텝수가 크게 달라질 수 있고 최소의 제어스탭수를 제공하는 최적의 연산유닛 수를 결정하는 것은 대단히 어려운 문제일 것이다.

그림 1.에서는 종류별 연산유닛의 개수가 알고리즘의 수행을 마치는데 필요한 제어스텝수에 어떤 영향을 미치는가 보여주고 있다. 여기서 T_1과 T_2는 연산의 종류를 나타내고 $V_1 - V_6$까지 6개의 연산이 있다.



(a) (b)

그림 1. CDFG의 두 가지 예

Fig. 1. Two examples of CDFG

그림 1.의 (a) 와 (b)는 모두 연산 T_1, T_2를 각각 3번 수행한다. 그러나 (a)를 위해 T_1, T_2 연산유닛을 각각 2개씩 만들 경우 V_2, V_3, V_4 모두가 연산유닛의 부족으로 한 제어스텝에 수행되지 못하고 두 제어스텝이 요구된다. 따라서 이 경우 총 제어스텝 수는 5가 된다. T_1을 1개, T_2를 3개 만들었다면 4개의 제어스텝만에 모든 연산이 이루어질 수 있음을 쉽게 알 수 있다. (b)의 경우 V_3, V_4가 두 번째 제어스텝에서 수행되지 않아도 전체 제어스텝수를 늘리지 않으므로 T_1 1개, T_2 2개만 주어지면 충분함을 알 수 있다. 이처럼 필요한 연산유닛의 개수는 CDFG내의 연산 개수뿐만 아니라 연산의 의존관계와 특정연산이 어느 위치에 편중되어 있는가에 의하여 복합적으로 결정된다.

IV. 제안된 스케줄링 알고리즘

1. 연산 종류별 연산유닛의 초기 개수 결정

먼저 연산 종류별 연산유닛 개수의 적절한 초기값을 구하기 위하여 주어진 CDFG에 ASAP(As Soon As Possible) 스케줄링과 ALAP (AS Late As Possible) 스케줄링을 수행한다. ASAP 스케줄링에서는 자원에 제한을 두지 않고 CDFG 노드를 최대한 일찍 스케줄링 하게 된다. ASAP 스케줄링에 의해 정해진 노드 v_i의 제어스텝을 E_i로 표기한다. ALAP 스케줄링에서는 주어진 총 제어스텝 수를 초과하지 않으면서 각 노드를 최대한 늦게 스케줄 한다. ASAP 스케줄링으로부터 얻어진 총 제어스텝수는 자원에 제한이 없을 경우 얻을 수 있는 최소 제어스텝수가 된다. 이 제어스텝수를 ALAP 에 적용하여 얻은 노드 v_i의 제어스텝을 L_i로 표기한다. 두 방법 모두 널리 알려진 알고리즘이며 여러 문헌에서 찾아볼 수 있으므로 자세한 기술은 생략한다.[8] 노드 v_i의 E_i값과 L_i 값이 같다면 이 노드는 CDFG에서 임계경로(critical path)상에 놓인 것을 의미하고 이 연산의 수행이 지연되면 총 제어스텝수가 늘어나게 된다. 따라서 자원을 충분히 할당하여 이 연산의 수행이 지연되지 않도록 하는 것이 총 제어스텝수를 최소화 하는데 유리하다. E_i값과 L_i 값의 차이가 크면 연산의 수행이 지연되어도 총 제어스텝수를 늘리지 않을 가능성이 크다고 할 수 있다. 이를 이용하여 각 연산유닛 수의 초기치를 구하기 위해 다음과 같은 기호를 사용하여 아래의 절차를 수행한다.

Control/Data Flow Graph - $G(V,E)$

가용한 총 실리콘 공간 크기 - A

연산종류 수 - m

연산타입 t_k를 위한 연산유닛을 만드는데 필요한 실리콘 크기 - a_{t_k}

연산타입 t_k를 위한 연산유닛 개수 - n_{t_k}

스케줄링 함수 - $f: O \rightarrow S$ (O : Set of operations; S : Set of control steps s_i)

Algorithm FUNCTIONAL UNITS
Call ASAP(G)
Call ALAP(G)
for each node v_i
  for each j $E_i \le j \le L_i$

$$P^j_{(type\ of\ v_i)} = P^j_{(type\ of\ v_i)} + \frac{1}{L_i - E_i + 1}$$

for each t_k do

$$P_{t_k} = max_j P^j_{t_k}$$

endAlgorithm

위의 알고리즘에서는 제어스텝 j에 대한 각 연산 종류의 상대적 유닛수 $P^j_{t_k}$를 구하고 모든 제어스텝에 대하여 연산의 종류 별로 최대값을 구하였다. 이렇게 구해진 값 P_{t_k}는 정수가 아닐 수 있고 연산유닛수의 상대적 비율만을 나타낸다. 이 값과 주어진 실리콘 공간의 크기를 이용

하여 다음과 같은 계산으로 연산타입 t_k를 위한 연산유닛 개수 n_{t_k}를 구한다.

for each t_k do

$$q_{t_k} = \frac{P_{t_k} a_{t_k}}{\sum_{t_k} P_{t_k} a_{t_k}} \cdot A$$

$$n_{t_k} = \left\lfloor \frac{q_{t_k}}{a_{t_k}} \right\rfloor$$

endfor

n_{t_k}값이 결정된 후 남아있는 실리콘 공간을 활용하기위해 아래의 과정을 수행한다.

$$R = A - \sum_{t_k} n_{t_k} a_{t_k}$$

while $(R > a_{t_1}\ or\ R > a_{t_2}\ or \cdots or\ R > a_{t_m})$

for each operation type t_k

if $R \geq a_{t_i}$ then

$R = R - a_{t_i};\ n_{t_i} = n_{t_i} + 1$

endfor

endwhile

2. 반복적 개선을 이용한 스케줄링

위에서 얻어진 각 연산유닛의 개수를 활용하여 주어진 CDFG를 스케줄링 한다. 스케줄링 방법으로는 널리 알려진 리스트 스케줄링을 사용한다. 각 연산유닛의 개수가 한정되어 있으므로 노드들이 스케줄링될때 특정 연산유닛이 부족하여 전체적인 제어스텝의 수가 늘어날 수 있다. 이럴 경우 상대적으로 덜 부족한 연산유닛 개수를 줄이고 가장 부족하다고 판단되는 연산유닛의 개수를 늘리면 보다 적은 수의 제어스텝내에 스케줄링을 마칠 가능성이 있다. 이를 위하여 리스트 스케줄링을 하면서 각 제어스텝에서의 연산 종류별 대기연산(실행가능 하지만 가용한 연산유닛이 없어서 실행되지 못한 연산)의 개수를 구하고 모든 제어스텝에 대해서 연산 종류별로 대기연산 개수의 합을 계산한다. 그리고 각 해당 연산유닛의 개수로 나누어 대기연산의 평균값을 구한다. 이렇게 구해진 평균값이 가장 큰 연산 종류의 연산유닛을 한개 이상 늘릴 수 있을 때까지 R 값을 고려하면서 대기연산의 평균값이 가장 작은 연산 종류의 연산유닛의 개수를 줄인다. 이때 모든 연산 종류별로 연산유닛이 반드시 한개 이상은 있어야 한다. 이런 방법으로 확보된 실리콘 공간을 이용하여 대기연산의 평균값이 가장 큰 연산유닛을 최대한 만든다. 그런 후 다시 R 값을 계산하고 추가로 만들 수 있는 연산유닛이 있다면 최대한 만든다. 위의 과정을 연산유닛의 개수를 조정할 수 없거나 (대기연산의 평균값이 가장 작은 연산 종류의 연산유닛이 한개 일 경우) 연산유닛의 개수를 조정했음에도 불구하고 제어스텝수가 줄지 않을 때까지 반복한다. 위의 과정을 아래의 알고리즘으로 상세히 기술한다. 아래의 알고리즘에서 $Plist_{t_i}$는 선행연산이 모두 스케줄링 되어 즉시 스케줄링이 가능한 타입 t_i 연산의 리스트를 나타낸다. 함수 INSERT_REAY_OPS는 노드의 집합 V를 검색하여 선행연산이 없는 노드들을 찾아 리스트에 넣고 집합에서 제거한다. SCHEDULE_OP $(S_{current}, FIRST(PList_{t_k}), C_{step})$에서는 현재까지 스케줄링 상태 $S_{current}$ 에 $FIRST(PList_{t_k})$을 C_{step} 번째 제어스텝에 할당한다. 여기서 함수 $FIRST$는 리스트의 첫 번째 원소를 제공한다. 함수 DELETE에서는 주어진 연산을 리스트에서 제거한다. 위의 함수들은 주로 리스트관리를 위한 것이므로 상세한 기술을 생략한다. 사용된 변수들 중 $R_OP_{t_{\min}}$과 $R_OP_{t_{\max}}$는 각각 연산유닛 당 대기연산의 최대값과 최소값을 나타내고 $t_{\min}, t_{\max}$는 해당연산의 타입을 표시한다.

Algorithm SCHEDULING

$Prev\_S\_length = \infty$

repeat

INSERT_REAY_OPS

$(V, PList_{t_1}, PList_{t_2} \ldots PList_{t_m})$;

$C_{step} = 0$

while(($PList_{t_1} \neq \varnothing$) or ... or ($PLis_{t_m} \neq \varnothing$)) do

$C_{step} = C_{step} + 1$;

for $k = 1$ to m do

for $funit = 1$ to n_{t_k} do

if $PList_{t_1} \neq \varnothing$ then

SCHEDULE_OP

$(S_{current}, FIRST(PList_{t_k}), C_{step})$

$PList_{t_1} =$

DELETE $(PList_{t_k}, FIRST(PList_{t_k}))$

endif

endfor

endfor

for $k = 1$ to m

$R\_OP_{t_k}$ = $|Plist_{t_k}|$ //리스트내의 연산 개수 //

endfor

INSERT_READY_OPS

$(V, PList_{t_1}, PList_{t_2} \ldots PList_{t_m})$

endwhile

$R\_OP_{t_{\min}} = Min_{t_k}(R\_OP_{t_k}/n_{t_k})$

$R\_OP_{t_{\max}} = Max_{t_k}(R\_OP_{t_k}/n_{t_k})$

REALLOCATE$(R, t_{\min}, t_{\max})$

$S_{prev} = S_{current}$

until $(C_{step} \geq Prev\_S\_length)$

$S = S_{prev}$

endAlgorithm

서브루틴 REALLOCATE($R, t_{\min}, t_{\max}$)는 대기연산의 평균값이 가장 작은 연산유닛의 개수를 줄여 대기연산의 평균값이 가장 큰 연산유닛의 개수를 늘려주는 함수로 아래에 상세히 기술되어 있다.

REALLOCATE($R, t_{\min}, t_{\max}$)
$temp = a_{t_{\max}} - R$
repeat
  $temp = temp - a_{t_{\min}}$
  $n_{t_{\min}} = n_{t_{\min}} - 1$
until ($temp \le 0$)
$R = R + temp$
$n_{t_{\max}} = n_{t_{\max}} + (R/n_{t_{\max}})$ // 정수 나눗셈 //
$R = R - (R \mod a_{t_{\max}})$
while ($R > a_{t_1}$ or $R > a_{t_2}$ or $\cdots$ or $R > a_{t_m}$)
  for each operation type t_k
    if $R \ge a_{t_k}$ then
      $R = R - a_{t_k}$; $n_{t_k} = n_{t_k} + 1$
  endfor
endwhile

V. 모의실험

제안된 스케줄링 알고리즘의 성능을 평가하기 위하여 다음과 같이 모의실험을 실시하였다. 먼저 스케줄링에 사용될 CDFG를 무작위하게 생성하였다. 대부분의 연산이 피연산자가 한개 또는 두개인 것을 고려하여 그래프를 생성할 때 모든 노드의 선행노드 개수를 2 이하로 제한하였고 연산자의 종류는 세 가지로 가정하였다. 본 논문에서 제안한 스케줄링 알고리즘의 성능 비교 평가를 위하여 다음과 같은 리스트 스케줄링을 기반으로한 두 가지 알고리즘을 사용하였다. 첫 번째 방법으로 그래프 내에서 연산 종류별 연산 개수의 비례로 연산유닛을 만든 다음 리스트 스케줄링을 적용하였다. 두 번째 방법으로는 연산 종류별 연산 개수와 관계없이 동수로 연산유닛을 만든 다음 첫 번째 방법처럼 리스트 스케줄링을 적용하였다. 두 방법 모두에서 주어진 실리콘 공간과 각 연산유닛을 만드는데 필요한 실리콘 공간크기에 따라 연산유닛의 수가 결정되었다. 위의 두 방법들과 본 논문에서 제안된 방법을 각각 L_1, L_2, L_3 으로 표기하기로 한다.

표 1은 그래프내에 총 연산수의 변화에 따른 세 알고리즘의 성능을 보여주고 있다. 각각의 경우에 대하여 3종류의 연산이 있는 100개의 그래프를 생성하였고 스케줄링 후 총 제어스텝수의 평균을 구하였다. 실험결과에 따르면 본 논문에서 제안된 자원제한 알고리즘(L_3)이 총 연산수에 관계없이 모두 우수한 결과를 보여 주고 있다. 연산 종류별로 연산유닛을 동수로 한 경우(L_2)가 가장 스케줄링 효율이 낮았으며 특히 그래프가 커질수록 성능이 나빠지는 것을 확인할 수 있다. 연산 개수 비례인 경우(L_1)는 L_3에 가까운 비교적 고른 스케줄링 효율을 보여주고 있다.

표 1. 연산개수 변화에 따른 평균 제어스텝 수
Table 1. Average control steps for different numbers of operations

연산 개수	L_1	L_2	L_3
50	10.4	11.4	10.5
100	15.7	18.5	15.1
200	22.1	26.2	21.0
300	25.8	34.0	24.7
400	30.2	42.3	29.5

표 2는 그래프내에 연산개수의 비율 변화에 따른 세 알고리즘의 성능을 보여주고 있다. 세 종류의 T_1, T_2, T_3의 비율이 10:20:70과 같이 큰 차이를 보이는 경우부터 30:40:30처럼 근소한 차이를 보이는 경우에 대하여 실험이 이루어졌다. 예상대로 본 논문에서 제안된 자원제한 알고리즘(L_3)은 비율차가 클 때 다른 두 방법에 비하여 상대적으로 높은 성능을 보이는 것을 확인할 수 있다. 이는 그래프의 특성에 따라 적응적으로 연산유닛의 개수를 변화시키는 전략 때문이라 생각된다. 연산 종류별로 연산유닛을 동수로 한 경우(L_2)는 비율차가 클수록 스케줄링 길이가 비례해서 커지는 것을 볼 수 있다. L_1과 L_3의 성능차이는 상대적으로 크지 않지만 일단 하드웨어로 구현되고 나면 같은 계산이 수없이 많은 횟수 반복되는 것을 고려할 때 하나의 제어스텝수라도 줄이는 것은 대단히 중요하다.

표 2. 연산개수의 비율 변화에 따른 평균 제어스텝 수
Table 2. Average control steps for different ratios of operation types

연산 종류별 연산 개수의 비율(%)			L_1	L_2	L_3
T_1	T_2	T_3			
10	20	70	28.8	57.8	26.8
20	20	60	28.6	49.0	27.0
30	40	30	28.1	29.4	27.1
50	20	30	27.7	37.0	26.9

VI. 결론

상위레벨 회로 합성은 회로의 동작을 기술하고 분할, 스케줄링, 할당 등의 단계를 거쳐 회로로 완성된다. 그 중 본 논문에서는 스케줄링 단계에서 자원(실리콘 공간크기)이 제한되는 경우 주어진 그래프에 대하여 각 연산 종류별로 연산유닛의 개수를 결정하여 스케줄링하는 알고리즘을 제시하였다. 널리 알려진 ASAP, ALAP 알고리즘 등을 이용하여 그래프를 분석하고 어떤 연산이 특정 제어스텝에 집중되어 보다 많은 연산유닛이 필요한지를 결정하였다. 이와 같이 결정된 연산유닛들을 이용하여 그래프에 리스트 스케줄링 알고리즘을 적용한다. 스케줄링 도중에 각 연산의 종류별로 연산유닛의 부족으로 실행되지 못한 대기연산의 개수의 합을 구하여 부족한 연산유닛과 상대적으로 여유 있는 연산유닛을 확인하게 된다. 이렇게 하여 대기연산이 가장 적은 종류의 연산유닛을 줄이고 대기연산이 가장 많은 연산유닛의 개수를 늘린 후 다시 스케줄링하는 과정을 반복하며 최대의 효율을 내는 연산 종류별 연산유닛의 개수를 찾았다. 이 알고리즘의 성능을 평가하기 위하여 모의실험을 수행하였고 비교의 대상으로 삼은 방법들에 비하여 우수한 스케줄링 결과를 제공하는 것을 확인하였다.

참 고 문 헌

[1] Azeddien M. Sllame, Vladimir Drabek, "An Efficient List-Based Scheduling Algorithm for High-Level Synthesis", Proceedings of the Euromicro Symposium on Digital System Design, Sept. 2002, pp. 316-323.

[2] Tianruo Yang, Zebo Peng, "An Efficient Algorithm to Integrated Scheduling and Allocation in High-Level Test Synthesis", Proceedings of Design Automation and test in Europe, Feb. 1998, pp. 74-81.

[3] Hans Achatz, "Generating Several Solutions for the Scheduling Problem in High-Level Synthesis", Proceeding of Euro-Design Automation Conference, Sept. 1995, pp. 66-71.

[4] P. Kollig, B. M. Al-Hashimi, K. M. Abbott, "Effective scheduling of behavioral descriptions in high-level synthesis", IEE Proc. Comput. Degit. Tech. Vol. 144, No. 2, Mar. 1997, pp. 75-82.

[5] Pierre G. Paulin, John P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's", IEEE Transactions on Computer-Aided Design. Vol. 8, No. 6, Jun. 1989, pp. 661-679.

[6] Anatoly Prihozhy, "Net Scheduling in High-Level Synthesis", IEEE Design & Test of ComputerI Vol. 13, No. 1, Spring 1996, pp. 26-35.

[7] Robert A. Walker, Samit Chaudhuri, "Introduction to the Scheduling Problem", IEEE Design & Test of Computer Vol. 12, Issue. 2, Summer 1995, pp. 60-69.

[8] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, Steve Y-L Lin, "High-level synthesis Introduction to Chip and System Design", Kluwer Academic Publishers, 1992.

황 인 재(In-Jae Hwang)

1986년 충북대학교 컴퓨터공학과 공학사

1991년 Univ. of Florida 전산학과 공학석사

1994년 Univ. of Florida 전산학과 공학박사

1995년~현재 충북대학교 컴퓨터교육과 부교수

관심분야 : 병렬 및 분산처리, 알고리즘