저전력 데이타-경로를 위한 효율적인 고수준 합성 알고리즘

(An Efficient Data Path Synthesis Algorithm for Low-Power)

박 채 령 † 김 영 태 †† 김 태 환 †††
(Chaeryung Park) (Young-Tae Kim) (Taewhan Kim)

요 약 이 논문은 회로 설계의 상위 단계에서 저 전력 회로 합성을 위한 효율적인 알고리즘을 제시한다. 풀고자하는 문제는, 상위 단계 회로 합성의 두 가지 주요 작업인 스케줄링과 하드웨어 할당 과정에서 최소의 전력 소모를 가지는 데이타-경로를 합성해 내는 것이다. 이 문제의 해결 방안으로, 우리는 과거 연구 결과에서 도출된 전력 소모를 줄이기 위한 기존의 요소들을 기초로 하여, 상위 단계 회로 합성에서 최적(optimal)의 전력 소모를 가지는 데이타-경로를 얻기 위한 IP (integer programming) 표현을 유도하였다. 그 다음, 유도된 IP 식을 이용하여 최소 전력 소모의 회로 합성을 위한 스케줄링과 하드웨어 할당 작업을 빠른 시간에 수행하기 위한 단계적 근사치 계산 방법을 고안하였다. 실제, 우리는 실험을 통해 제안된 알고리즘이 매우 빠르며, 상위 단계에서의 데이타-경로 합성에서 전력 소모량을 줄이는데 매우 효과적임을 보여준다.

Abstract In this paper, we present a new high-level data path synthesis algorithm which solves the two design problems, namely, scheduling and allocation, with power minimization as a key design parameter. From the observations in previous works on data path synthesis for low power, we derive an integer programming (IP) formulation for the problem, from which we then develop an efficient heuristic to carry out the scheduling and allocation simultaneously. Our experimental results demonstrate that the proposed algorithm is very effective in saving power consumption of circuits significantly.

1. 서 론

90년대에 들어서면서 눈부신 반도체 공정기술의 발전에 따라, VLSI 설계에서 전력 소모를 최소화시키는 문제는 매우 중요한 고려 사항으로 대두되고 있다[1]. 즉, 고집적 회로 설계는 휴대 가능한 전자 제품 (예: 휴대폰, 파워-PC 등)의 사용을 일상 생활화하게 하였으며, 이에 따라 회로의 전력소모로 인해 생기는 열 발산에 따른 회로 성능의 보존과 회로 패키징 문제가 중요하게 다루어지고 있다. 이러한 문제를 극복하기 위한 근본적인 해결로 저전력 소모의 회로 합성은 매우 중요하며, 최근까지 저전력 회로 설계에서 회로 내부의 전력 소모를 줄이기 위한 많은 기술들이 제안되었다. 그 중 상위 단계 회로 합성에서 대표적인 두 가지 기법으로 *기능모듈 선택 (functional module selection)*[2,3]과 *기능모듈의 선택적 폐쇄 (selective shutdown of functional modules)*[4,5]를 들 수 있으며, 우리가 제안할 합성 알고리즘은 이 두 가지의 저전력 기법을 기초로 하고 있다.

1.1 기능모듈 선택

일반적으로, 주어진 연산은 특정한 기능모듈에 의해 수행되며, 각 기능모듈들은 다양한 실행 지연 시간, 면적 및 전력 소모를 갖는다. 예를 들어, [2]의 데이타에

† 비 회 원 : 미국 시놉시스 연구원
cpark@synopsys.com
†† 비 회 원 : 한국과학기술원 전기전산학과 및 첨단정보기술연구센터
young@vlsisyn.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전기전산학과 및 첨단정보기술연구센터 교수
tkim@cs.kaist.ac.kr
논문접수 : 1999년 7월 12일
심사완료 : 1999년 12월 15일

의하면 32-비트 덧셈연산이 Ripple-carry adder (*RCA*)로 실행될 경우에는 평균 22*mW*의 전력 소모와 20*ns*의 수행 시간을 필요로 하며, Carry-lookahead adder (*CLA*)로 실행될 경우는 평균 37.3*mW*의 전력 소모와 10*ns*의 수행 시간을 필요로 한다. 또한, 32-비트 곱셈연산이 Booth multiplier (*BOOTH*)로 실행될 경우 평균 84*mW*의 전력 소모와 160*ns*의 수행 시간을, 그리고 Array multiplier (*ARR*)로 실행될 경우 평균 295.5*mW*의 전력 소모와 100*ns*의 수행 시간을 필요로 한다. 이렇게 한 연산에 대해 실행 가능한 여러 개의 모듈들이 주어져 있을 때, 회로의 전체 전력 소모를 줄이기 위해서는 회로의 허용된 수행 시간을 만족시키는 범위 내에서 각 연산을 전력 소모가 적은 기능모듈들로 구현하는 것이 바람직하다. 한 예로, 그림 1과 그림 2는 *BOOTH* 곱셈 연산이 2 clock cycle의 수행 지연 시간을 갖고, *ARR*이 1 clock cycle의 수행 지연 시간을 갖는다고 가정할 때, 주어진 dataflow 그래프에 대한 두 가지 다른 스케줄을 보여준다. (dataflow 그래프에서 *R*로 명시된 노드는 입력-포트 작동을, *W*로 명시된 노드는 출력-포트 작동을 나타낸다.) 그림 1의 스케줄 *A*는 *ARR* 기능모듈 만을 사용할 경우의 스케줄을 보여주며, 그림 2의 스케줄 *B*는 *ARR*과 *BOOTH* 기능모듈 모두를 사용할 경우의 스케줄을 보여준다. 이 경우, 스케줄 *A*는 2개의 *ARR*를 사용하여 4138.4*mW*의 전력 소모를 가져오며. 스케줄 *B*는 1개의 *ARR*와 2개의 *BOOTH*를 사용하여 2657.2*mW*의 전력 소모만을 가져온다. 결국 스케줄 *B*는 스케줄 *A*에 비해 평균 35% 적은 전력 소모를 가지게 된다.

[2]와 [3]은 기능 모듈의 선택을 이용한 대표적인 연구들이다. [2]는 시뮬레이션에 의해 입력값들에 대한 통

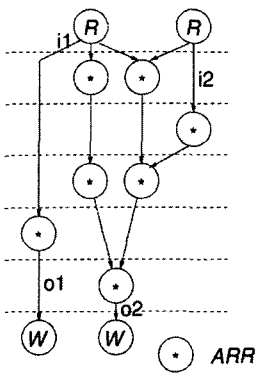


그림 1 스케줄 *A*

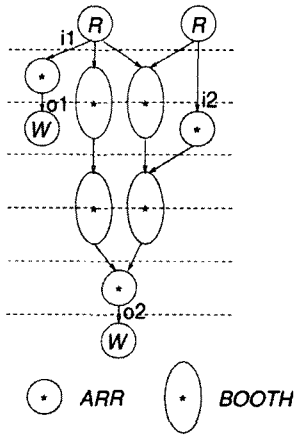


그림 2 스케줄 *B*

계학적 분포에 기초하여, 각 기능모듈의 전력 소모를 미리 구하고 회로의 수행시간이 주어져 있는 조건하에, 적은 전력 소모를 가지는 스케줄링과 기능모듈 선택 결과를 보였으나, 이에 대한 구체적인 해결은 제시하지 않았으며, [3]은 스케줄링 후 저전력 소모를 위한 기능 모듈 선택 문제를 Max-cost Multi-Commodity Flow문제로 변형하여 최적의 결과를 얻었다. 본 논문의 방법은 [2], [3]의 것과 달리 기능모듈 선택을 스케줄링 단계에서 어떻게 잘 활용하여 최적의 전력 소비를 하는 회로를 생성하는가하는 방법에 초점을 두고 있다.

1.2 기능모듈의 선택적 폐쇄

최근 연구[2]에 따르면 CMOS 디지털 회로에서 clock 시그널에 의한 전력 소모는 회로 전체 전력 소모의 15%에서 45%정도에 이른다고 한다. clock 시그널은 모든 기능모듈에 고루 연결되어 있다. 그러나, 대부분의 경우 회로의 기능모듈들은 작동하지 않는 clock step을 가지고 있고, 그 동안 기능모듈들은 clock 시그널을 필요로 하지 않는다. 또한, 기능모듈에 의한 전력 소모 뿐 아니라 clock 시그널 자체에서 발생하는 전력 소모를 줄이기 위해서라도, 특정 기능모듈에 대해서 clock 펄스가 필요한 clock step 동안만 clock 시그널을 발생하도록 하는 기법이 요구된다.

그림 3은 기능모듈의 작동과 비작동 시간에 따라 clock 시그널을 조절 발생시키는 clock gating 기법을 이용한 clock 재발생기 (clock regenerator)를 보여준다. 재발생된 clock 시그널을 이용하면, 그림 2의 스케줄 *B*에서는 한 개의 기능모듈 *ARR*가 3번 작동하고, 두 개의 기능모듈 *BOOTH*가 각각 2번씩 작동하기 때문에 전체적으로 1222.8*mW* (= 295.5×3 + 84×2×2)의 전력을 소

모하게 된다. 그러므로, 스케쥴 A의 전력 소모와 비교했을 때 70% 가량 적은 결과를 가지게 된다. 그러나, clock 재발생기 또한 회로 면적과 전력 소모를 가지는 추가적인 회로 구성요소이기 때문에, 각 기능모듈마다 clock 재발생기를 할당한다는 것은 바람직하지 않다. 따라서, 우리는 같은 유형의 모듈들을 한꺼번에 제어하는 방식으로 각 모듈 유형 당 하나의 clock 재발생기를 할당하는 방법을 채택한다.

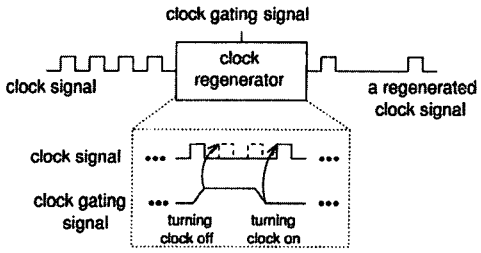


그림 3 clock 재발생기

이제 우리가 할 일은, 각 연산에 대해 구현 가능한 여러 개의 기능모듈 및 그것들의 수행 속도, 면적, 평균 전력 소모량이 포함된 기능모듈 라이브러리가 주어져 있고, dataflow 그래프 내의 모든 연산들을 완전히 수행하기 위해 필요한 clock cycle 수 또한 설계자에 의해 주어져 있다고 할 때, 전체 기능모듈에 의한 전력 소모의 최소화를 위해 (1) 각 연산들에 대한 실행 스케쥴을 결정하고 (2) 각 연산이 구현될 기능모듈을 할당하고, 마지막으로 (3) 사용된 기능모듈의 유형마다 재발생 될 clock 시그널을 이끌어내는 것이다.

기능모듈의 선택적 폐쇄는 실제 회로 합성 도구[4]로 활용되고 있는 기법이며, 대표적인 연구로 [5]를 들 수 있다. [5]에 의하면, 스케쥴링이 끝난 후 메모리의 구성요소 (레지스터)들을 그것들의 작동/비작동 시간에 따라 잘 묶어 각 그룹별로 활동 시간대에만 clock을 생성하여 전력 소모를 줄이는 기법을 제시하였다. 이와 달리 본 연구는 기능모듈에 대한 선택적 폐쇄 방법을 스케쥴링과 함께 고려하여 최상의 결과를 구하고자 시도하였다.

2. 용어 정의

우리의 알고리즘은 주어진 HDL(VHDL, Verilog 등) 표현의 내부적인 구조 형태, 즉, 데이타-경로를 HDL로 기술한 dataflow 그래프를 입력으로 받는다. 스케쥴링에서 사용될 총 clock cycle 수는 T로 주어져 있는 것으로 가정한다. 본문에서 우리는 dataflow 그래프내의 각 연산들과 그 연산들 사이의 수행 선행관계를 '<'라는 기호를 사용하여 나타낸다. 예를 들어, $O=\{op_1, op_2, \ldots, op_N\}$ 를 연산들의 집합이라 할 때, (여기서, N은 dataflow 그래프 내의 연산 총수를, L은 라이브러리 내의 기능 모듈의 총수를 각각 나타낸다.) $op_i < op_j$는 op_j의 실행이 시작하기 전에 op_i가 반드시 끝나야 한다는 것을 의미한다. 우리는 기능모듈 라이브러리가 각 유형의 모듈에 관한 평균 전력 소모, 실행 시간, 면적 등을 가지고 있는 것으로 가정한다. 이를 표현하기 위해, $\Lambda=\{1,2,\ldots,L\}$ 를 기능모듈 라이브러리를 나타내는 기호로 사용하며, 유형-k 기능모듈에 대하여 e_k는 clock cycle 단위(0보다 큰 정수)로 표현된 실행 시간을, c_k는 기능모듈의 면적을, d_k는 기능모듈의 clock cycle당 전력 소모량을 의미하는 기호로 사용한다. $O(k)$는 유형-k의 기능모듈로 실행될 수 있는 연산들의 집합을, 그리고 $U(op_i)$는 주어진 연산 op_i가 실행될 수 있는 기능모듈 유형의 집합을 나타내는 기호로 사용한다. 마지막으로, 어떤 연산 op_i의 *time frame*, $[s_i, t_i]$는 op_i가 수행 가능한 clock step의 범위를 나타낸다. 물론 이 경우 이들은 T 개의 clock step을 넘지 않아야 한다는 조건을 만족해야 한다. 여기서 한 연산에 대한 *time frame*은 그 연산이 가장 일찍 수행될 수 있는 시간을 *ASAP (As Soon As Possible)* 스케쥴링으로 구할 수 있고, 같은 방식으로 그 연산이 총 clock step T에 위배되지 않으면서 가장 늦게 수행될 수 있는 시간을 *ALSP (As Late As Possible)* 스케쥴링으로 구하여 얻는다[6].

3. 문제 공식화

이번 절에서는, 전력 소비 최소화를 위한 스케쥴링과 하드웨어 할당 방법을 IP (integer programming) 문제로 변환하여 해결하는 방법에 대해 자세히 소개한다. 먼저, IP에 사용될 몇 가지 변수들에 대해 정의한다. 각 연산 op_i에 대해 다음과 같은 값을 가지는 변수 o_{ijk}, ($1\le i\le N$, $s_i\le j\le t_i-e_k+1$, $k\in U(op_i)$)를 다음과 같이 정의한다.

$$o_{ijk}=\begin{cases}1, & \text{연산 } op_i\text{가 유형}-k\text{의 기능모듈로}\\ & \text{clock step } j\text{에서 실행을 시작할 경우,}\\ 0, & \text{그외의 경우.}\end{cases}$$

다음으로 각 유형의 기능모듈이 사용된 개수를 결정하기 위해 f_{kj}와 F_k ($1\le k\le L, 1\le j\le T$)를 정의한다. f_{kj}는 clock step j에서 수행되어야 하는 유형-k의 기능모듈

개수이고, F_k는 모든 clock step에 걸쳐 요구되어지는 유형-k의 기능모듈 개수이다.

마지막으로, 각 유형의 기능모듈이 작동 가능한 clock step과 작동이 가능하지 않은 clock step을 표현하기 위해, 0 또는 1 값을 가지는 변수 a_{kj} ($1 \le k \le L$)를 다음과 같이 정의한다.

$$a_{kj} = \begin{cases} 1, & \text{어떤 유형}-k\text{의 기능 모듈이 clock step } j\text{에서 작동하는 경우,} \\ 0, & \text{그외의 경우.} \end{cases}$$

이제, IP에서 만족되어야 할 제약 조건 식들을 제시한다. 먼저, 각 연산은 정확히 한 번만 수행되어야 하므로, 각 연산 op_i ($1 \le i \le N$)에 대해 다음과 같은 식이 필요하다.

$$\sum_j \sum_k o_{ijk} = 1. \qquad (1)$$

두 번째로 연산에 대한 수행 의존 관계를 선형 제약 식들로 나타낸다.

$F(i,j) = \{O_{ij'k} | s_i \le j' \le j - e_k + 1,\ k \in U(op_i)\}$ 라고 하면, 다음과 같은 제약 식이 만족되어야 한다.

$$\sum_{o \in F(i,j)} o = \begin{cases} 1, & \text{연산 } op_i\text{의 실행이 clock step } j\text{나 그 이전에 끝난 경우,} \\ 0, & \text{그외의 경우.} \end{cases}$$

마찬가지로, $I(i,j) = \{O_{ijk} | k \in U(op_i)\}$ 라고 하면, 다음과 같은 제약 식이 만족되어야 한다.

$$\sum_{o \in I(i,j)} o = \begin{cases} 1, & \text{연산 } op_i\text{의 실행이 clock step } j \text{ 에서 시작한 경우,} \\ 0, & \text{그외의 경우.} \end{cases}$$

그리고, 두 연산 op_a와 op_b 가 수행 의존 관계가 있다면 (즉, $op_a < op_b$), 다음과 같은 제약식 또한 만족되어야 한다.

$$\sum_{o \in F(a,j)} o \ge \sum_{o' \in I(b,j+1)} o', \qquad s_a \le j \le t_a - 1. \qquad (2)$$

이 식은 op_b의 실행이 clock step $j+1$ 에서 시작하였다면 op_a의 실행이 clock step j나 그 이전의 clock step에서 반드시 끝나야 함을 의미하는 제약 식이다.

이제, 변수 o_{ijk}의 값이 결정되면 이를 이용해 f_{kj}와 F_k 값을 다음과 같이 구할 수 있다.

$$f_{kj} = \sum_{\{i | op_i \in O(k)\}} \left(\sum_{j'=e_k+1}^{j} o_{ij'k} \right), \qquad (3)$$

$$F_k = \max_j f_{kj}. \qquad (4)$$

마지막으로, f_{kj}의 값이 결정되면 할당된 유형-k의 기능모듈들이 작동하지 않을 clock step또한 결정된다. 따라서, a_{kj}의 값은 다음과 같이 구할 수 있다.

$$a_{kj} = \begin{cases} 0, & f_{kj} = 0 \text{ 인 경우,} \\ 1, & \text{그외의 경우.} \end{cases} \qquad (5)$$

다시 말해, 어떤 유형 k의 기능 모듈들 중 적어도 하나가 clock step j ($f_{kj} \ge 1$)에서 작동되었다면 유형-k의 모든 기능모듈이 clock step j에서 작동될 것이며, 이것은 a_{kj}의 값이 1임을 뜻한다.

이제 우리가 최소화시키고자 하는 식은 다음과 같다.

$$F = \alpha \left(\sum_k c_k F_k \right) + \beta \sum_k \sum_j a_{kj} d_k F_k + \gamma \sum_k \sum_j a_{kj}.$$

우변의 첫 번째 항은 기능 모듈에 대한 하드웨어 비용을, 두 번째 항은 기능모듈에 의한 전력 소모량을, 그리고 마지막 항은 clock 재발생기의 작동 비용을 각각 나타내고 있다. 그리고, α, β, γ는 각각의 항에 대한 가중치이다.

4. 근사치 계산 알고리즘

3절에서 유도한 IP는 최적(optimal)의 해결을 가져오지만, 풀고자하는 문제의 크기가 클 경우 매우 많은 시간이 요구되거나, 계산이 불가능할 수 있다. 이를 해결하기 위해 이 절에서는 3절에서 얻은 IP를 이용한 근사치 계산 방법을 제시한다. 핵심 아이디어는 다음과 같다. IP에 사용된 변수 중 값이 결정되어야 하는 주된 변수는 o_{ijk}이며, 이것은 결국 0이나 1 값 중 하나를 취해야 한다. 하지만 중간 계산 결과에서는 o_{ijk}가 정수 값이 아닐 수도 있다. 이때 우리는 이러한 정수가 아닌 변수 값들을 단계적으로 예상 측정하여 빠르게 결과를 계산해 낸다. (이러한 방법은 본 연구에서 처음 고안 시도하였으며 나중의 실험 결과에서 나타나듯이 결과의 질을 희생함이 없이 빠른 수행 시간 안에 문제를 풀 수 있음을 보여 준다.)

구체적인 설명으로, 변수 o_{ijk}중 하나의 값을 1로 해보자. 이러한 선택은 제약 조건 (1)과 (2)에 의해 다른 0-1변수들의 값을 결정하는 것을 가능하게 한다. 이 계산 절차를 그림 4의 예제를 이용하여 설명하도록 한다. 총 clock step 수를 5라고 할 때 IP 표현으로부터 다음과 같은 제약식들을 구할 수 있다.

제약 조건 (1)로 부터:

a) $o_{111} + o_{121} = 1$

b) $o_{221} + o_{222} + o_{231} + o_{232} + o_{241} = 1$

c) $o_{321} + o_{322} + o_{331} = 1$

d) $o_{431} + o_{441} + o_{451} = 1$

e) $o_{531} + o_{532} + o_{541} = 1$

f) $o_{641} + o_{651} = 1$

과 같은 제약식을 구할 수 있으며, 또한 제약 조건 (2)로부터 다음과 같은 제약식들을 구할 수 있다.

g) $o_{111} \geq o_{221} + o_{222}$

h) $o_{111} \geq o_{321} + o_{322}$

i) $o_{221} \geq o_{431}$

j) $o_{221} + o_{231} + o_{222} \geq o_{441}$

k) $o_{321} \geq o_{531} + o_{532}$

l) $o_{531} \geq o_{641}$

먼저 o_{111}을 1로 정했다고 하자. 그러면 제약 *a)*에 의해 $o_{121}=0$ 이 된다. 또 다른 방법으로 o_{121}을 1로 정했다고 가정할 수도 있다. 이 경우 제약 *a)*에 의해 우리는 $o_{111}=0$ 을 얻어낼 수 있다. 그러면, $o_{111}=0$ 이므로 수행의존 제약 *g)*와 *h)*에 의해 $o_{221}=o_{222}=0$, $o_{321}=o_{322}=0$ 이 된다. 또한, $o_{321}=0$ 이므로, 제약 *k)*에 의해 $o_{531}=o_{532}=0$이 된다. 그리고, $o_{531}=0$ 이므로, 제약 *l)*에 의해 $o_{641}=0$ 이 된다.

이런 절차로 하나의 0-1변수의 값을 1로 정하면, 다른 몇 개의 0-1변수의 값이 제약 조건(1)과 제약 조건(2)에 의해 저절로 결정된다. 이 경우, 결정이 되지 않은 나머지 변수 o_{ijk}의 값은 제약 조건 등식 (1)을 만족해야 하므로, 등식 안의 변수들이 동일한 값을 가진다고 가정하여 이 변수들의 값을 결정한다. 이렇게 얻어진 값들과 이미 결정된 값들을 이용하여 3절에서 정의한 f_{kj}, F_j, a_{kj}의 값들을 측정해 낸다.

마지막으로 F의 값을 계산한다. 다음은 변수 o_{ijk} 들을 각기 1로 정할 때 그에 따른 F 근사 측정값을 보여주고 있다. (여기서 가중치는 $\alpha=1$, $\beta=1$, $\gamma=3$ 로 하였다.)

setting $o_{111}=1$: F = 148.92,
setting $o_{121}=1$: F = 152.26,
setting $o_{111}=1$: F = 148.92,

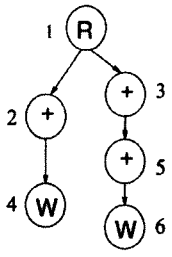
⋮

setting $o_{641}=1$: F = 161.72,
setting $o_{651}=1$: F = 128.16.

F의 근사값을 검사하여 이들 변수 o_{ijk} 중 가장 큰 영향을 미치는 것을 선택하여 1로 둔다. 즉, 모든 변수 o_{ijk} 가운데 F 근사치의 값을 최소로 하는 변수가 1로 정해지고, 이에 따라 다른 변수의 값들도 정해짐으로써 중간 결과가 얻어진다. 위의 예제에서는 o_{651}이 1로 선택되었고 중간 결과는 다음과 같이 된다.

$o_{111} = o_{121} = \frac{1}{2}$,
$o_{221} = o_{222} = o_{231} = o_{232} = o_{241} = \frac{1}{5}$,
⋮

이 중간 결과를 이용하여 위의 과정을 반복한다. 중간 결과에서 정수가 아니라고 가정한 모든 0-1변수들 중 하나를 1로 정한다. 그리고, 다른 변수들의 값들도 같은 방식으로 도출해 내어 F의 근사값을 계산한다. 이러한 반복 과정을 통해 0-1 변수라 가정한 모든 변수들이 0 이나 1의 값을 갖게 되고 정수로 가정한 모든 변



	CLA	RCA
type	1	2
hardware cost	2	5
power consumption	4	1
execution time	1	2
operations	+	+

그림 4 알고리즘 적용의 예

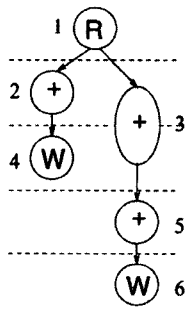


그림 5 그림 4의 예에 대한 알고리즘 적용 결과

수들이 정수값을 가질 때 우리는 최종 결과를 얻는다. 예를 들어, 그림 5는 그림 4에서 모든 0-1변수의 값이 결정되었을 때 얻어진 dataflow 그래프의 스케쥴을 보여준다.

다음은 위의 근사 알고리즘의 흐름을 요약한 것이다. 이 알고리즘의 시간 복잡도는 총 clock step이 T, 총 연산 수가 N, 총 기능 모듈이 L일 때 $O((T \cdot N \cdot L)^2)$이 되며, 한 변수의 결정으로 다른 많은 변수들의 값들이 결정되기 때문에 제약식의 개수가 빠르게 줄어듬으로서, 실제 문제에서 걸리는 시간은 이보다 훨씬 작다.

```
repeat
  /* 정수가 아닌 변수 o_ijkp에 해당하는 중간 결과를 결정한다
     */
  for ( 정수가 아닌 변수 o_ijkp에 대해)
      o_ijkp = 1;
      제약식들로부터 다른 변수들의 값을 계산한다;
      최종적으로 구하고자하는 F의 값을 계산한다;
  end_for;
  F 가 최소인 중간 결과를 선택한다;
until ( 모든 변수가 정수값을 가질 때)
```

5. 실 험

우리는 본문에서 제시된 알고리즘을 실제로 구현하고, 이를 여러 개의 상위 단계 합성 benchmark들에 적용하였다. 표 1은 제시된 알고리즘에 의해 사용된 기능 모듈 라이브러리이다. 라이브러리의 내용은 다양할 수 있으며, 설계자에 의해 사용된 technology library로부터 얻어질 수도 있다. 표 2는 실험 결과를 정리한 것이다. 설계 *df.5* 와 *df.7* [6]은 총 clock step을 각각 5와 7로 정하고 얻은 미분방정식이고, *ar.8*과 *ar.10* [6]은 총 clock step을 각각 8과 10으로 정하고 얻은 AR-lattice 필터이다. 그리고, *ewf.18*과 *ewf.20* [6]은 총 clock steps을 각각 18과 20으로 정하고 얻은 Elliptic-wave 필터이다. (실험에서 우리는 3장의 문제 공식의 최소화 값의 가중치 값은 α=1, β=1, γ=2로 두었으며, 그에 따른 결과를 구하였다. 이러한 가중치의 값은 설계의 특성과 설계자의 경험에 따라 자유로이 변경 될 수 있으며, 일반적인 선호 값의 기준은 없다고 하겠다.)

표 2의 '면적의 최적화' 열은 설계 면적만을 최적화한 결과를 보여주며, '전력의 최적화' 열은 전력 소모량만을 최적화한 결과를 보여준다. 마지막으로, '면적과 전력 최적화' 열은 전력 소모량과 회로 면적을 동시에 최적화하는 상위 단계 합성 결과를 보여준다. 세 열에 나타난 결과의 비교는 우리의 알고리즘이 설계의 하드웨어 면적과 전력 소모를 잘 절충했음을 보여준다. 이러한 결과를 얻는데 걸린 시간은 각 설계마다 3초를 넘지 않았다.

다음으로, 우리의 제안한 알고리즘에 의해 생성된 결과가 최적의 결과에 얼마나 가까운지를 알아보기 위해 3장에서 표현한 IP를 풀어야 하였다. 이를 위해 사용된 비선형 목적식 (non-linear objective function)을 먼저 La Grange first order 조건들을 사용해 선형의 식에 의해 근사값을 구하도록 한 후 ILP를 적용 풀었다. 이 문제를 풀기 위해 우리는 IBM3081에서 LINDO 패키지를 사용하였다. 표 3은 *df.5*에 대해 ILP와 우리의 알고리즘에 의해 구한 결과를 보여 준다. 결과에서 보여 주듯이 우리의 알고리즘은 최적의 값을 생성했으며, ILP가 1시간 이상의 걸린데 비해 우리의 알고리즘은 단지 2.15초 걸려 결과를 구하였다. 그와 나머지 benchmark에 대해는 이것에 해당하는 ILP 식이 너무 크게 되었으며, 따라서, LINDO는 어떠한 결과도 생성해 낼 수 없었다. 하지만, 여기서 강조하고자 하는 것은 우리의 알고리즘이 항상 최적의 결과를 생성하는 것은 아니다. 엄밀히 말하면, 제안된 알고리즘은 greedy한 성질을 가지기 때문에, 설계 규모가 증가 함에 따라 local minimum

표 1 사용된 기능모듈 라이브러리

유형	기능 모듈	면적	평균전력 소모량	실행 시간	기능
1	*rpl*	5	1	1	+, −, >
2	*cla*	2	4	1	+, −, >
3	*booth*	15	4	2	×
4	*arr*	6	15	1	×

표 2 알고리즘에서 얻은 결과

설계	면적의 최적화		전력의 최적화		면적과 전력 모두 최적화	
	면적	전력	면적	전력	면적	전력
df.5	54	170	88	84	57	125
df.7	50	133	71	95	66	81
ar.8	88	544	94	252	94	252
ar.10	76	418	130	140	139	140
ewf.18	90	414	153	124	107	195
ewf.20	110	460	158	125	125	106

에 걸리는 단점이 있다. 따라서, 제안한 알고리즘의 효과는 표 2, 3의 결과에서 예시하듯이 최적화에 가까운 결과를 유지하면서, 빠른 시간 안에 회로 면적과 전력 소모의 상대적 가중을 잘 절충하는 장점을 가지는데 있다고 할 수 있다.

표 3 df.5의 ILP와 제안한 알고리즘의 결과

	ILP	Ours
기능모듈의 작동 step 수	20	20
전력	125	125
실행시간 (sec.)	4208	2.15

6. 결 론

본 논문에서 우리는 데이타-경로의 전력 소모를 줄이기 위한 스케쥴링과 하드웨어 할당 문제에 효과적으로 접근하기 위한 상위 단계의 알고리즘을 제시하였다. 이 문제를 먼저 최적 결과를 얻기 위한 IP (integer programming) 표현으로 변환하였고, 이를 효율적으로 풀기 위한 근사치 계산 방법을 제안하였다. 우리의 문제 해결 방법은 하드웨어 비용과 전력 소모량 사이의 trade-off를 효과적으로 활용하는 특징을 가지고 있으며, 실험 결과는 이러한 특징을 잘 나타내고 있다. 또한, 제안한 근사치 계산 방법은 다른 복잡한 IP, 혹은 ILP (integer linear programming) 문제를 효율적으로 풀기 위한 시도에 사용되어 질 수 있다. 앞으로의 연구과제로, 실제 제안한 기법을 보완하여 현재의 local minimum를 효과적으로 극복할 수 있는 연구가 필요하다고 하겠다.

참 고 문 헌

[1] S. Gary, P. Ippolito, G. Gerosa, C. Dietz, J. Eno and H. Sanchez, "PowerPC 603, A Microprocessor for Portable Computers," *IEEE Design & Test of Computers*, Vol. 11, No. 4, pp. 14-23, 1994

[2] R. Martin and J. Knight, "Power-Profiler: Optimizing ASICs Power Consumption at the Behavioral level," *Proc. of Design Automation Conference*, pp. 42-47, June 1995

[3] J. Chang and M. Pedram, "Module Assignment for Low Power," *Proc. of European Design Automation Conference*, pp. 376-381, September 1996

[4] Synopsys Inc., *Design Power User Manual*, 1998

[5] A. Farrahi, G. Tellez and M. Sarrafzadeh, "Memory Segmentation to Exploit Sleep Mode Operation," *Proc. of Design Automation Conference*, pp. 36-41, June 1995

[6] P. Paulin and J. Knight, "High-level Synthesis Benchmark Results using a Global Scheduling Algorithm," *Logic and Architecture Synthesis for Silicon Compilers*, North-Holland, pp. 211-228, 1989

박 채 령

1987년 서울대학교 컴퓨터 공학 학사. 1989년 서울대학교 컴퓨터 공학 석사. 1995년 미국 일리노이 주립대 전산학 박사. 현재 미국 시놉시스 연구원. 관심분야는 VLSI 설계 자동화, 컴퓨터 구조, 계산이론

김 영 태

1999년 한국과학기술원 전산학 학사. 1999년 ~ 현재 한국과학기술원 전기전산학과 전산학전공 석사과정중. 관심분야는 저전력 VLSI 회로 설계, 상위수준 회로 합성

김 태 환

1985년 서울대학교 전산통계학 학사. 1987년 서울대학교 전산학 석사. 1993년 미국 일리노이 주립대 전산학 박사. 현재 한국과학기술원 전기전산학과 전산학전공 조교수. 관심분야는 VLSI 설계 자동화, 계산이론