네트워크-플로우 방법을 기반으로 한 통합적 데이타-경로 합성 알고리즘

(Integrated Data Path Synthesis Algorithm based on Network-Flow Method)

김 태 환 †

(Taewhan Kim)

요 약 이 논문은 상위 단계 데이타-경로 합성에서 연산 스케쥴링과 자원 할당 및 배정을 동시에 고려한 통합적 접근 방법을 제시한다. 제안한 방법은 스케쥴링 되어있지 않은 데이타-플로우 그래프에 대해서 수행에 필요한 총 clock 스텝 수와 필요한 회로 면적을 동시에 최소화하는 데이타-경로 생성에 특징이 있다. 일반적으로, 연결선의 결정이 합성의 마지막 단계에서 이루어지는 기존의 방법과는 다르게, 우리의 접근 방법은 연산 스케쥴링과 연산의 연산 모듈 배정 그리고 변수의 레지스터 배정 작업을 동시에 수행하여 추가적인 연결선의 수를 매 clock 스텝마다 최적화(optimal) 시킨다. 본 논문은, 이 문제를 최소-비용의 최대-플로우 문제로 변형하여 minimum cost augmentation 방법으로 polynomial time안에 해결하는 알고리즘을 제안한다

Abstract We present an integrated approach to the problems of the scheduling, allocation (register and interconnections), and binding in high-level synthesis. Our algorithm synthesizes a data path from an unscheduled dataflow graph with an objective of minimizing both the number of control steps and total design area. Unlike most of the prior approaches in which interconnections are determined only in the final step of the synthesis process, in our approach, scheduling of operations, binding of operations to functional units, and binding of variables to registers are performed simultaneously so that interconnections are determined optimally for each clock step. The problem is formulated as a *minimum cost maximum flow problem in a network which can be solved in polynomial time* using the minimum cost augmentation method.

1. 서 론

상위 단계 합성은 디지털 시스템의 상위 기술 언어로 표현된 회로를 RT(register-transfer) 단계의 회로로 변환하는 과정을 말한다. 이 과정에서의 중요한 두 가지 작업은 연산 스케쥴링 (scheduling)과 자원 (resource) 할당 (allocation) 및 배정(binding)이다[1]. 스케쥴링 작업은 연산들을 clock 스텝으로 나누어 같은 clock 스텝 내의 연산이 동시에 수행되도록 하는 과정이며, 자원 할당/배정은 자원을 선택하는 작업으로, 산술식이나 논리식을 수행할 모듈이나, 변수를 저장할 레지스터, 그리고 데이타 값을 전송할 연결선을 할당하고 배정하는 과정이다. 최종적인 설계 목표는 수행시간 (총 clock 스텝 수)와 회로 면적(연산 모듈, 레지스터, 연결선 수)를 동시에 최소화하는 것이며, 일반적으로, 연산 스케쥴링 문제는 clock 스텝의 총 개수의 최적에 있고, 자원 할당/배정 문제에서는 연산 모듈, 레지스터, 그리고 연결선의 개수의 최적화에 있다[1]. 특히, 우리가 주목하고자 하는 것은 연결선의 최소화에 더 큰 비중을 두고자 하는 것이다. 그 이유는, RT-단계의 회로가 생성된 후 연결선을 줄이는 문제는 매우 어려우며, 고밀도 집적회로 설계에 따른 연결선이 회로 면적에 차지하는 비율이 점점 증가하고, 따라서 연결선의 라우팅(routing) 문제가 더 복잡해지며, 또한, 연결선의 지연 시간이 전체 회로의 속도에 크게 영향을 주는 상황이 전개되기 때문에,

· 본 연구는 첨단정보기술 연구센터(AITrc)를 통하여 과학재단의 지원을 받았음.

† 종신회원 : 한국과학기술원 전산학과 교수

tkim@cs.kaist.ac.kr

논문접수 : 1999년 12월 15일

심사완료 : 2000년 10월 21일

상위 단계 합성에서 n-비트의 연결선을 하나 더 줄인다는 것은 n 개의 시그널 라인을 줄이는 효과가 있기 때문에 매우 중요하다 하겠다.

자원 할당과 배정 문제에 대한 많은 연구가 있어왔다. 대표적인 연구를 정리하면 다음과 같다. [2]는 이 문제를 3개의 문제로 나누어 해결하고자 하였다. 3 개의 세부 문제란, (a) 최소 개수의 레지스터를 사용한 변수 배정 문제, (b) 연산 배정 문제, (c) 연결선 할당 문제를 말한다. 문제 (a), (b)는 가중된 bipartite matching 문제들로 모델링하여 해결하였으며, 문제 (c)는 greedy한 방법을 사용하였다. 결과적으로, 원래의 할당/배정 문제를 세부문제 (a), (b) (c)의 순서로 풀거나, (b), (a), (c)의 순서로 푸는 것을 제안한 것이다. [3]은 연결선의 개수를 최소화하는 문제를 0-1 정수 선형 프로그래밍 문제로 모델링하였다. 스케쥴된 데이타-플로우 그래프, 할당된 연산 모듈, 레지스터들이 주어진 상태에서, [3]은 연산에 대한 연산 모듈 배정과 변수에 대한 레지스터 배정을 동시에 해결함으로서 연결선의 개수를 최소화하였다. 그러나, 이 방법은 상당히 많은 계산 시간을 필요로 한다. [4]는 배정 문제만을 한 clock 스텝씩 점진적으로 해결하였다. 이 과정에서 간단한 branch and bound과 휴리스틱을 이용하여 연결선의 개수를 최소화하였다. [5,6]은 소위 슬라이싱 트리 (slicing tree)에 기초하여 연산 배정을 점진적으로 수행하여 레이아웃 작업을 용이하게 하고자 시도하였다. 하지만 사용된 예측치는 완전하지 않은 연산 배정 결과로 부터 구했기 때문에 비현실적일 수 있다. [7] 또한 슬라이싱 트리를 이용하여 모듈 배정 문제를 FPGA (field-programmable gate array) 응용에서 풀고자 하였다. 하지만 이 방법은 체인된 연산의 수에 따라 기하 급수적으로 계산 시간이 커지는 단점을 지니고 있다.

위의 언급된 연구등의 근본적인 약점은 스케쥴링을 데이타-경로의 하드웨어 비용을 줄이는데 아무런 고려를 하지 않았다는 것이다. 또한, [2]의 경우 연결선의 할당이 변수와 연산 배정이 끝난 후에야 결정된다는 것이다.

이러한 약점을 극복하기 위한 연구로 스케쥴링과 할당/배정 문제를 동시에 고려한 통합적 합성 시도가 많이 있어 왔다. [8]은 점진적 합성 알고리즘을 제안하였다. 매 clock 스텝에서 수행될 연산의 선택과 자원 할당/배정 문제를 0-1 정수 선형 프로그래밍 문제로 변형하는 방법을 제안하였다. 당연히 계산 시간이 많이 걸리게 되며, 적용 가능한 설계 입력의 크기에 제안이 따르게 된다. [9]는 [10]이 제안한 force-directed 스케쥴링에 기초한 반복적 알고리즘을 제시하였다. 구체적으로, 각 반복 과정에서 한 개의 연산이 선택되고 연 산 모듈에 배정이 된다. 이 과정에서, 필요하다면 추가적인 연결선이 할당될 것이다. cost 함수를 정하는 과정에서 force-directed 스케쥴링의 개념을 확장한 방식을 취하였으며, 필요한 연결선의 개수에 대한 예상치를 함수에 포함하였다. 그러나, 엄밀한 의미에서 변수에 대한 레지스터 할당과 배정이 나중에 따로 이루어지기 때문에 연결선 할당이 스케쥴링과 연산 모듈 할당/배정 과정과 동시에 수행된다고는 말할 수 없다. [11,12]는 simulated annealing에 기초한 통합적 합성 방법을 시도하였다. 합성 문제를 한 축으로는 clock 스텝들을, 다른 한 축으로는 연산 모듈들을 배치한 2차원적 자리 배정 문제로 모델링하였다. 연산들은 2 차원의 각 좌표에 무작위적으로 이동 가능하여, 그때마다 그 상황에서의 사용된 clock 스텝 수, 연산 모듈 수, 레지스터 예상 수 및 연결선 예상 수를 계산하게 된다. 이 방법은 최적의 결과에 접근하는 해결을 가져올 수 있지만, 문제 크기가 증가하면 많은 시간을 필요로 하게 되고, 또 연결선의 결정은 최후로 미루게 되는 단점이 있다. [13]은 스케쥴링, 모듈 할당/배정을 레이아웃과 연관시켜 동시에 해결하고자 하였다. 레이아웃과 하드웨어공유 사이의 이른바 힘(force)의 연결을 예측하고 이를 이용하여 두 문제를 동시해 풀고자 시도하였다. 이 방법 또한 슬라이싱 트리에 기초를 두고 있다. 단점은 연결선의 최소화 문제는 고려하지 않았다는 것이다. [14] 또한 [14]의 것과 비슷한 슬라이싱 트리의 구조적 변형에 의해 모듈을 할당/배정 하고자 하였다. 하지만, 그 구조로 부터 어떻게 모듈 배치 정보를 받아내는지는 분명히 보여주고 있지는 않다.

이제 기존의 통합적 합성 단점을 해결하는 한 방법으로, 본 논문에서는, clock 스텝의 개수와 연결선을 포함한 전체 회로 면적을 동시에 최소화하는 목적을 가지고 새로운 문제 해결 방식을 제안한다. 즉, 강조하고자하는 것은, 연결선을 합성의 마지막 단계에서 결정하는 기존의 방법과는 다르게 우리의 방법에서는 스케쥴링, 그리고 자원 할당/배정 문제를 동시에 고려하여 각각의 *clock 스텝에서 부가적으로 필요한 연결선의 양을 최적화(optimal)* 한다. 한가지 명확히 하고자하는 것은 각 clock 스텝에서 스케쥴링과 연산 모듈, 레지스터 배정을 동시에 고려하여 연결선의 수를 최적으로 구한다는 의미는 이러한 단계적 최적이 반드시 전체적인 최적이란 의미는 아니다. 단지, 통합된 방법으로 연결선의 부분적 최적을 구하여 기존의 방법들의 것보다 연결선을 줄이

는데 매우 효과적임을 본 논문은 입증하고 있는 것이다.

2. 통합적 데이타-경로 합성 문제

우리의 합성 알고리즘의 입력은 산술연산과 논리연산을 나타내는 노드와 노드들 사이의 수행 종속 관계를 나타내는 arc로 이루어진 스케쥴링 되어있지 않은 데이타-플로우 그래프이다. 우리의 알고리즘은 회로에서 사용될 각 종류의 연산 모듈의 개수를 회로 설계자가 이미 결정하였다고 가정한다. 알고리즘은 한 번에 한 clock 스텝 씩 데이타-경로를 점진적으로 만든다. 데이타-플로우 그래프에서 어느 한 연산의 모든 선행 연산들이 이전의 clock 스텝에 스케쥴링 되어있을 때 그 연산을 ready 연산이라고 한다. 알고리즘의 각 반복에서, 현재의 clock 스텝에서 수행될 연산의 집합을 선택해서 연산 모듈에 배정하며, 동시에 생성되는 새로운 변수들을 레지스터에저장하는 일을 다음과 같은 목적을 만족시키는 방향으로 진행시킨다.

(1) 추가로 필요한 레지스터의 개수를 최소화한다.

(2) 추가로 필요한 연결선의 개수를 최소화한다.

(3) 데이타-플로우 그래프내의 모든 연산을 스케쥴링하기 위해 필요한 clock 스텝의 총 개수를 최소화한다.

(1), (2), (3)을 모두 만족시키는데 있어서는 서로 간의 trade-off를 잘 알아야 할 것이다. 사실상, 앞으로 언급되겠지만, 우리의 알고리즘은 추가적인 레지스터나 추가적인 연결선의 개수, 그리고 총 clock 스텝의 개수의 총괄적인 최소화 방향으로 ready 연산들 중에서 적절한 연산을 선택하여 스케쥴링과 자원 배정을 해 나가는 것이다.

구체적으로 제안한 알고리즘은, 현재 진행되는 clock 스텝에서의 연산의 선택, 연산에 대한 연산 모듈 배정, 그리고 변수에 대한 레지스터 할당/배정은 최소-비용의 최대-플로우 문제를 해결함으로써 이루어진다. 따라서, 이러한 플로우 (flow) 문제의 해에 따라 그 clock 스텝까지 수행되는 연산들에 대한 데이타-경로가 형성되게 된다, 이런 반복적인 플로우의 적용은, 데이타-플로우 그래프 안의 모든 연산들이 스케쥴 될 때까지 수행하게 된다.

3. 네트워크-플로우 문제로의 변형

2 절에서 언급하였듯이, 우리의 제안한 알고리즘은 추가적인 레지스터의 개수(목적 1), 추가적인 연결선의 개수(목적 2), 총 clock 스텝 개수(목적 3)가 최소화 되도록 현재 clock 스텝에서의 여러 개의 ready 연산 중에 가능한 많은 연산을 선택한다. t-1번의 clock 스텝을 통하여 데이타-경로가 부분적으로 이미 생성되어 있다고 하자. 이제, 현재의 clock 스텝은 t이다. O를 ready 연산의 집합, F를 사용될 수 있는 연산 모듈의 집합, R을 사용 가능한 레지스터의 집합이라 하자.

네트워크 $G=(N,A)$는 노드의 집합 N과 arc의 집합 A로 이루어진 그래프이다. 집합 A안의 각 arc들은 가중치(weight)와 용량(capacity)이 주어져 있다. 우리의 문제에서, 집합 N은 O, R, 그리고 R의 원소들과 source a와 sink b로 이루어져 있다. 집합 O 의 노드들은 ready 연산들이고, 집합 F의 노드들은 사용 가능한 연산 모듈이며, 집합 R의 노드들은 사용 가능한 레지스터들이다. A는 집합 O의 연산 노드들로부터 집합 F의 해당 연산이 수행 가능한 연산 모듈로의 arc와, 집합 F의 연산 모듈에서 그 해당 연산 모듈의 출력을 저장할 수 있는 집합 R의 레지스터로의 arc, 그리고 노드 a로부터 집합 O의 모든 노드들로의 arc와, 집합 R의 모든 노드들로부터 노드 b로의 arc로 구성되어 있다. (그림 3에 예가 제시되어 있음.)

연산과 연산 모듈과의 관계: 그래프 $G=(N,A)$에서, ready 연산 집합 O의 한 연산 o_i과 집합 F의 한 연산 모듈 f_j간에 arc가 존재하는 것은 연산 모듈 f_j이 연산 o_i를 수행할 수 있다는 것을 의미한다. 집합 O와 F의 노드들과 O로부터 F로의 arc들로 이루어진 그래프 G의 부분 그래프는 bipartite 그래프이고, 우리는 이러한 그래프를 $G_1=(O,F,E_1)$이라 한다. 여기서 주목할 점은 (i) G_1 의 최대 matching의 개수가 현재의 clock 스텝에서의 수행 가능한 연산의 개수라는 사실이다. 또한, 이 개수는 현재 clock 스텝에서 수행 가능한 연산들의 출력을 저장하는 데에 필요한 레지스터의 개수가 된다. 왜냐하면 각 연산은 하나의 출력을 생성하고 그 출력은 서로 다른 레지스터에 저장되어야 하기 때문이다. 이러한 정보는 최소의 추가적인 레지스터의 개수를 계산하는데 이용 될 수 있다. 다음으로 주목할 점은, (ii) 일반적으로, G_1의 최대 matching은 유일하지 않다는 사실이다. 그러므로, 최대 matching의 선택은 주어진 데이타-플로우 그래프의 모든 연산을 스케쥴하는데 필요한 clock 스텝의 전체 개수에 영향을 끼친다. 전체 수행시간에 대해 현재 clock 스텝에서 수행 될 연산 o_i가 미치는 영향의 한 정량적인 cost를 그래프 G_1의 o_i에서 출발하는 각 arc에 표시할 수 있고, 이는 한 최대 matching을 선택하는데 있어서 기준이 된다. s_i를 연산 o_i로 부터 발생하는 각 arc에 할당된 cost라 하자. 이 때, s_i를 연산 o_i의 스케쥴링 cost라 부른다. 연산의 스

케쥴링 cost는 해당 연산과 그 연산의 결과에 종속된 연산들과 critical path의 길이를 이용하여 계산된다. 즉, $s_i=-(l^2+d)$로 표현하며 l은 o_i에서부터 시작하여 출력까지 이르는 최대 연산 길이를 나타내고, d는 o_i 에 종속된 연산 개수를 나타낸다. 따라서, 스케쥴링 cost가 작은 연산을 현재의 clock 스텝에서 선택하여 수행시키는 것이 바람직하다 하겠다. 세 번째로, (iii) 만약 연산 o_i가 연산 모듈 f_j에 배정된다면, f_j의 입력 터미널에 존재하는 연산 o_i의 입력 값을 저장하는 레지스터로부터 추가의 연결선 여부를 알 수 있다. $c^{I}_{i,j}$를 추가로 필요한 연결선의 개수를 나타내며, 이는 연산 o_i가 연산 모듈 f_j에 배정될 때의 추가의 입력 연결선 *cost*라 부른다.

결과적으로, 연산 o_i로부터 기능 모듈 f_j로의 arc $e_{i,j}$에 할당되는 cost는 s_i와 c^{I}_{ij}의 가중된 합으로 다음과 같이 정의된다.

$$w(e_{i,j})=\alpha\cdot s_i/S_0+\beta\cdot c^{I}_{i,j}/C_0$$

이때, S_0와 C_o는 표준(nomalization) factor이고, α와 β는 가중(weighting) factor이다.

연산 모듈과 레지스터와의 관계: 그래프 $G=(N,A)$에서, 집합 F의 노드로부터 집합 R의 노드로 arc가 존재한다. 집합 F와 R의 노드와 F에서 R로의 arc로 이루어진 그래프 G의 부분 그래프는 bipartite graph이 되고 우리는 이를 $G_2=(F,R,E_2)$라고 한다. 만약, 한 ready 연산이 현재 clock 스텝에서 스케쥴링 되고, 또한, 연산 모듈 f_j에 배정되었다면, 연산 모듈 f_j의 출력은 집합 R의 한 레지스터 r_k에 저장될 수 있다. 이때 우리는 집합 R의 레지스터 r_k에 대해, 연산 모듈 f_j의 출력 터미널로부터 r_k까지의 추가적인 연결선이 필요한지 여부를 결정할 수 있다. $c^{O}_{j,k}$에 기능 모듈 f_j과 레지스터 r_k간의 추가적 연결선이 존재하는가에 따라 0 또는 1의 값이 주어진다. 이렇게 연산 모듈 f_j과 레지스터 r_k에 의해 생성된 cost를 우리는 추가의 출력 연결선 *cost*라 부른다. 결과적으로, 그래프 G내의 연산 모듈 f_j과 레지스터 r_k까지의 arc $g_{j,k}$의 연결선 cost는 다음과 같이 주어진다.

$$w(g_{j,k})=\beta\cdot c^{O}_{j,k}/C_o$$

구체적인 $s_i, C^{I}_{i,j}, C^{O}_{j,k}, S_0, C_0$의 자세한 계산 설명은 생략하며, 후의 예에서 간략히 설명한다.

이제, 그래프 G의 다른 arc들의 값은 0으로 놓는다. 그리고, G의 모든 arc의 용량은 1로 놓는다. 결과적으로 우리의 최적화 문제는 그래프 G에서 최소-비용의 최대-플로우(maximum flow of minimum cost)[9]의 답을 찾는 문제로 귀결된다. 최소화되어야 하는 cost 함수는 다음과 같다.

$$\alpha\cdot\sum_{o_i\in A_1}\frac{s_i}{S_o}+\beta\cdot\Big(\sum_{e_{i,j}\in A_2}\frac{C^{I}_{i,j}}{C_0}+\sum_{g_{j,k}\in A_3}\frac{C^{O}_{j,k}}{C_0}\Big)$$

이때, A_1은 현재 clock 스텝의 수행을 위해 선택된 연산들의 집합이고, A_2는 이러한 연산의 수행을 위해 선택된 연산 모듈까지의 arc들의 집합이고, A_3은 네트워크 G내에서 이러한 연산 모듈의 수행을 위해 선택된 레지스터들의 집합이다. 이런 네트워크에서 최적화 문제를 해결하기 위해 우리는 minimum cost augmentation 방법[15]를 적용한다.

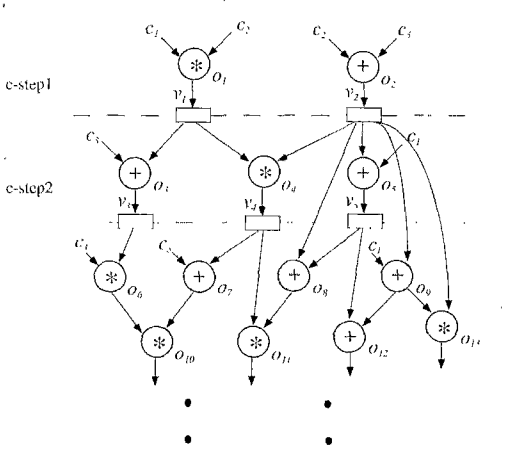


그림 1 데이타 플로우 그래프의 일부분

예를 들어, 그림 1은 데이타-플로우 그래프의 한 부분을 보여준다. c_1,c_2,c_3은 상수이고, v_i는 연산 o_i를 수행하기 위해 생성된 변수들이다($i=1,\cdots,5$). 이 예에서 clock 스텝 1, 2에서 수행될 연산들은 미리 결정되어 있고, 그림 1에서의 박스는 이미 스케쥴 된 연산들의 출력이 저장될 레지스터들을 나타낸다. clock 스텝 1, 2까지의 데이타-경로는 그림 2에 보여지며, 하나의 덧셈, 하나의 곱셈, 그리고, 하나의 ALU 연산기인 f_1,f_2,f_3로, 그리고 레지스터들은 그들이 할당된 시간에 따라서 명명되었다. 또한, 이 데이타-경로는 변수의 레지스터 배정, 연산의 기능 모듈 배정에 대한 정보를 나타내 보이고 있다. 이제, 현재 진행중인 clock 스텝 3의 ready 연산들은 o_6,o_7,o_8,o_9이 된다.

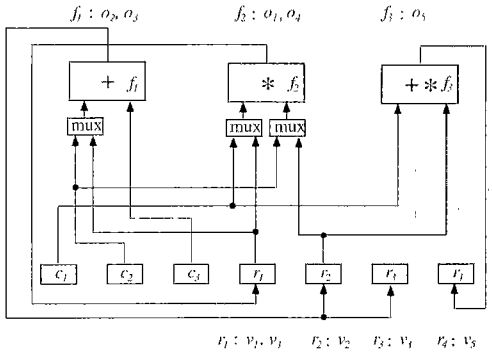
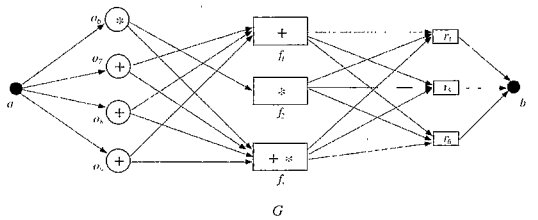


그림 2 그림 1의 데이타-플로우 그래프에 대한 데이타-경로

그림 3은 현재의 clock 스텝에 해당하는 네트워크 $G=(N,A)$를 보여주고 있다. 그림 2의 레지스터 r_1, r_2, r_4는 r_1에는 변수 v_4, r_2에는 변수 v_2, 그리고 r_4에는 변수 v_5가 현재의 clock 스텝에서 유효하기 때문에 현재 다른 변수를 새로이 저장 할 수 없다. 그러나, r_3는 v_3의 유효 저장 시간이 현재 clock 스텝의 write cycle 에서 끝나기 때문에 다시 사용될 수 있다. 하나의 레지스터 r_3만 사용 가능하기 때문에, 새로이 두 개의 레지스터 r_5, r_6이 생성되었음을 그림 3은 보여진다. 그림 3의 표는 스케쥴링 cost s_i, 입력 연결선 cost $c^{I}_{i,j}$, 그리고, 출력 연결선 cost $c^{O}_{j,k}$의 값을 나타내고 있다. 이러한 값들은 그림 1의 데이타-플로우 그래프와 그림 2의 합성된 부분적 데이타-경로로부터 유도된다. 예를 들어, o_6이 f_2에 연결 될 경우 스케쥬링 cost는, $s_6=-(l^2+d)=-(6^2+8)=-44$로 정해진다. 또한, f_1이 r_3에 연결 될 경우 연결선 cost는 $c^{I}_{6,2}=2$이며, $c^{O}_{1,3}=0$이다. S_0 표준 factor의 값은 91이며, 이는 s_i의 값만을 이용한 동일 네트워크에서의 최소-비용의 최대-플로우를 구해서 얻어진 값이다. 비슷한 방법으로, 표준 factor C_0 값은 9이며, 이는 $c^{I}_{i,j}$와 $c^{O}_{j,k}$의 값만을 이용한 동일 네트워크에서의 최소-비용의 최대-플로우를 구해서 얻어진 값이다. 이제 최종적으로, 그래프 G내의 arc $e_{i,j}$의 $w(e_{i,j})$값과, arc $g_{j,k}$의 $w(g_{j,k})$의 값이 계산된다. 그림 3안의 표의 마지막 열은 arc들의 cost를 보여준다. (우리는 여기서 $\alpha=\beta=1$이라고 가정하였다.) 예를 들어, 첫 번째 표에서, $w(e_{6,2})=s_6/S_0+c^{I}_{6,2}/C_0=-44/91+2/9$이고, 두 번째 표에서 $w(g_{1,3})=C^{O}_{1,3}/C_0=0/9$임을 알 수 있다.



$e_{i,j}$	S_i	$c^I_{i,j}$	$w(e_{i,j})$
$o_6 - f_2$	$-(6^2+8)=-44$	2	-44/91+2/9
$o_6 - f_1$	$-(6^2+8)=-44$	2	-44/91+2/9
$o_7 - f_1$	$-(5^2+7)=-32$	0	-32/91+0/9
$o_7 - f_2$	$-(5^2+7)=-32$	2	-32/91+2/9
$o_8 - f_2$	$-(3^2+4)=-13$	2	-13/91+2/9
$o_8 - f_1$	$-(3^2+4)=-13$	1	-13/91+1/9
$o_9 - f_1$	$-(3^2+6)=-15$	2	-15/91+2/9
$o_9 - f_2$	$-(3^2+6)=-15$	0	-15/91+0/9

$g_{j,k}$	$c^O_{j,k}$	$w(g_{j,k})$
$f_1 - r_3$	0	0/9
$f_1 - r_5$	1	1/9
$f_1 - r_6$	1	1/9
$f_2 - r_3$	1	1/9
$f_2 - r_5$	1	1/9
$f_2 - r_6$	1	1/9
$f_3 - r_4$	1	1/9
$f_3 - r_5$	1	1/9
$f_3 - r_6$	1	1/9

그림 3 현재 clock 스텝에 대한 네트워크 $G=(N,A)$

Minimum cost augmentation 방법을 사용하여, 그림 4에서 보이는 바와 같이 최소-비용의 최대-플로우를 구할 수 있다. 얻어진 해에서, 우리는 현재의 clock 스텝에서의 수행을 위해 스케쥴 되어야 할 연산의 집합을 결정할 수 있고, 연산에 대한 연산 모듈 배정을 결정할 수 있으며, 스케쥴된 연산들의 출력을 저장할 레지스터들을 할당/배정 할 수 있고, 또한, 동시에 추가로 필요한 연결선을 결정할 수 있다. 본 예에서의 경우, 연산 o_6, o_7, o_9는 현재의 clock 스텝의 수행을 위해 스케쥴 되었고 각각 2, 0, 0개의 추가 입력 연결선이 할당을 필요로 하는 연산 모듈 f_2, f_1, f_3 로 배정이 되었다. 게다가, 연산 o_6, o_7, o_9의 출력 변수의 값들은 1, 0, 1개의 추가 출력 연결선을 각각 필요로 하는 레지스터 r_5, r_3, r_6로 배정이 되었음을 보여준다.

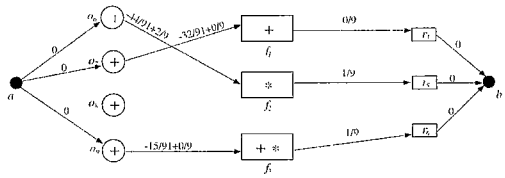


그림 4 최소-비용의 최대-플로우 적용

4. 실험 결과

제안된 알고리즘은 C로 구현되었으며 Sun Sparcstation 10에서 수행되었다. 또한, 여러 개의

benchmark 설계에 적용하여 데이타-경로를 생성하였으며, 이를 다른 알고리즘에 의해 얻은 결과와 비교하였다. 실험에서 우리의 제안한 방법의 결과는 수식 3에 사용된 가중치 α (스케쥴 비용에 대한 가중치), β (연결선 비용에 대한 가중치)의 값들과 밀접한 연관성이 있다. 본 실험에서는 설계에 따라 α와 β의 값을 변경시켜 가면서 기존의 연구 결과와 비교하여 비슷한 clock 스텝 수를 가지는 합성 결과들 가운데서 연결선이 최소인 것을 선택하는 과정으로 실험을 진행하였다.

표 1은 [16]에 나타난 AR 필터의 데이타-플로우 그래프를 내번 펼쳐 (unrolling) 구한 그래프를 (이 그래프를 4AR이라 부르며 64개의 덧셈과 48개의 곱셈으로 구성되어 있음.) 가지고 가중치 α와 β 값을 변화해 가면서 구한 결과를 요약한 것이다. 결과에서 보듯이 레지스터의 수는 연결선과의 가중치의 값들에 따른 반비례적 관계를 잘 따르지 않지만, clock 스텝의 수와 연결선의 수는 가중치의 값의 변화를 통해 잘 제어되고 있음을 보여준다.

표 2는 AR 필터 설계[16]에 대한 결과를 보여주며. 얻은 결과는 [17]의 결과와 비교하였다. 제안한 알고리즘은 입력으로 스케쥴 되어 있지 않은 데이타-플로우 그래프를 받아들인다. 그리고, [17]에서 사용된 연산 모듈과 같은 종류 및 수의 연산 모듈을 사용하였다. 우리의 알고리즘은 레지스터를 하나 덜 사용한 반면, 연결선을 하나 더 사용하였으며, 같은 개수의 clock 스텝을 사용하였다. 그러나, 알고리즘의 수행시간에 있어서는 우리의 것이 상당한 향상을 가지고 오는 것을 볼 수 있다.

표 1 가중치의 값의 변화에 따른 4AR 필터 설계의 결과

스케쥴링 Cost(α)	연결선 Cost(β)	결 과		
		Clockt스텝	연결선	레지스터
1	4	40	45	11
1	3	36	46	11
1	2	34	49	11
1	1	33	49	11
2	1	33	51	11
3	1	32	53	12
4	1	31	55	12

표 2 AR 필터 설계에 대한 비교

	Clock스텝	연결선	레지스터	Time(초)
Ours	9	17	5	7.2
Rim	9	16	6	190.84[a]

a : 스케쥴링 시간을 포함하고 있지 않음

표 3 EW 필터 설계에 대한 비교

	Clock스텝	연결선	레지스터	Time(초)
Ours1	21	39	11	12.4
Ours2	22	37	11	13.0
Ours3	21	45	11	5.5
STAR	c	43	10	10.28[a]
ARYL	21	46	10	0.65[a]
LYRA	21	47	10	3.38[a]
Splicer	21	43	24	~20k
HAL	19[b]	45	12	360

a : 스케쥴링 시간을 포함하고 있지 않음,
b: 파이프라인 된 곱셈기를 사용, c : 알려지지 않았음

표 3은 EW 필터 설계[10]에 대한 결과를 보여준다. (여기서 α = 1.5, β = 1를 사용하였다.) 우리는 대부분의 논문에서 사용하는 [10]의 데이타-플로우 그래프를 이용하였다. 우리의 알고리즘은 입력으로 스케쥴 되어 있지 않은 데이타-플로우 그래프와 스케쥴 되어 있는 데이타-플로우 그래프 모두를 받아들인다. 스케쥴 되어 있지 않은 데이타-플로우 그래프에 대한 우리의 알고리즘의 결과 (표에서 *Ours1*, *Ours2*)와, 스케쥴 되어 있는 데이타 플로우 그래프에 대한 우리의 알고리즘과의 결과를 (표에서 *Ours3*) 비교했을 때, 스케쥴링, 할당/배정을 통합적으로 하여 수행할 경우에 생성된 연결선 cost에 큰 감소를 가지고 오는 것을 알 수 있다. 또한, 우리의 알고리즘의 수행결과를 STAR[18], LYRA&ARYL [2], Splice[4], HAL[10]의 결과와 비교해 보았다. 모든 것들은 [10]에 의해 스케쥴된 데이타-플로우 그래프를 입력으로 받아들였고, point to point 연결선 방식을 적용하였으며, 동일 종류와 같은 수의 연산 모듈 사용을 하였다. 표 3의 비교에서 보이듯이, 우리의 알고리즘은 상당히 좋은 결과를 생성하였다. 스케쥴된 데이타-플로우 그래프를 입력으로 받는 데이타-경로 합성 알고리즘 가운데 여태까지 가장 좋은 결과를 낸 STAR[18]의 것과 비교해서도, 우리의 알고리즘은 동일한 개수의 clock 스텝과 레지스터를 사용하면서, 4개나 적은 연결선을 할당하였다. 이는 우리의 알고리즘이 연결선의 cost를 최소화하는 방향으로 스케쥴링, 할당 및 배정을 동시에 수행하는데 아주 효과적임을 암시한다.

5. 결 론

본 논문에서, 우리는 상위 단계 합성에서 연산 스케쥴링과 자원(연산 모듈, 레지스터, 연결선) 할당 및 배정의

문제를 동시에 해결하는 새로운 접근 방법을 시도하였다. 합성 마지막 단계에서 연결선이 결정되는 종래의 접근 방식과는 다르게, 우리는 각 clock 스텝에서 스케쥴링과 할당/배정을 동시에 처리하는 통합적 방법을 제안함으로서 연결선의 수를 단계적으로 최적화 하는데 그 특징이 있다. 이 문제를 우리는 네트워크 상에서의 최소-비용의 최대-플로우 문제로 변형하여 최적의 해를 구하였다. 그러나, 이러한 단계적 최적이 반드시 전체적 연결선 최적은 아니다. 하지만 통합적 방법으로 구한 연결선의 최적화는 기존의 연구에서는 보이지 않으며, 우리의 제안 방법으로 구현한 결과가 기존의 연결선 최소화 결과보다 우월하다는 것을 입증하였다는데 의의가 있다. 추후 연구과제로, 제안한 비용 공식, 즉, 수식 3의 가중치를 빠른 시간에 자동으로 알아내는 효과적인 방법이 요구된다고 하겠다. 이를 위해서는 다양한 많은 실험적 경험 뿐만 아니라 각 가중치가 알고리즘의 전반적인 효과에 얼마나 큰 역할을 하는지를 알아내는 분석적 방법도 고려해 보아야 할 것이다.

참 고 문 헌

[1] D. Gajski, N. Dutt, A. Wu, and S. Lin (Eds.) *High-level Synthesis - Introduction to Chip and System Design*, Kulwer Academic Publishers, 1992.

[2] C. Huang, Y. Chen, Y. Lin, and Y. Hsu, "Data Path Allocation based on Bipartite Weighted Matching," *Proc. of Design Automation Conference*, pp. 499-504, 1990.

[3] C. Hwang J. Lee, and Y. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10. no. 4, pp.464-475, April 1991.

[4] B. Pangrle, "Splicer: A Heuristic Approach to Connectivity Binding," *Proc. of Design Automation Conference*, pp. 536-541, 1988.

[5] S. Tarafdar, M. Leeser, "The DT-Model: High-level Synthesis using Data Transfers," *Proc. of Design Automation Conference*, pp. 114-117, 1998.

[6] S. Tarafdar, M. Leeser, Z. Yin, "Integrating Floorplanning in Data Transfer Based High-level Synthesis," *Proc. of International Conference on Computer-Aided Design*, pp. 412-417, 1998.

[7] M. Xu and F. Kurdahi, "Layout-driven High-level Synthesis for FPGA Based Architectures," *Proc. of Design and Test Conference in Europe*, 1998.

[8] M. Balakrishnan and P. Marwedel, "Integrated Scheduling and Binding: A Synthesis Approach for Design Space Exploration," *Proc. Design Automation Conference*, pp. 68-74, 1989.

[9] R. Cloutier and D. Thomas, "The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm," *Proc. Design Automation Conference*, pp. 71-76, 1990.

[10] P. Paulin and J. Knight, "Scheduling and Binding Algorithms for High-Level Synthesis," *Proc. Design Automation Conference*, pp. 1-6, 1989.

[11] S. Devadas and A. Newton, "Algorithm for Hardware Allocation in Data Path Synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 7, pp. 768-781, July 1989.

[12] P. Kollig, B. Al-Hashimi, "Simultaneous Scheduling, Allocation and Binding in High-level Synthesis," *Electronics Letters*, vol. 33, no. 18, August 1997.

[13] W. Dougherty and D. Thomas, "Unifying Behavioral Synthesis and Physical Design," *Proc. of Design Automation Conference*, pp. 756-761, 2000.

[14] S. Hassoun, "Fine Grained Incremental Rescheduling Via Architectural Retiming," *Proc. of International Symposium on System Synthesis*, pp. 158-163, 1998.

[15] R. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, 1983.

[16] M. Rim *et al.* "Optimal Allocation and Binding in High-Level Synthesis of VLSI Digital Systems," Tech. Report, Dept. of Electrical and Computer Engineering, University of Wisconsin - Madison, Sept. 1991.

[17] M. Rim *et al.* "Optimal Allocation and Binding in High-Level Synthesis," *Proc. Design Automation Conference*, pp. 120-123, 1992.

[18] F. Tsai and Y. Hsu, "STAR: An Automatic Data Path Allocator," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 9, pp. 1053-1064, Sept. 1992.

김 태 환
정보과학회논문지 : 시스템 및 이론
제 27 권 제 2 호 참조