데이타 경로 합성에서의 연결선 최적화를 위한 다중포트 메모리 할당 알고리즘

(A Multiport Memory Allocation Algorithm for Optimizing Interconnections in Data Path Synthesis)

김 태 환 † 홍 성 백 ††

(Taewhan Kim)(Sung-Pack Hong)

요 약 상위단계 합성에서 데이타 저장을 위한 메모리 할당 문제는 중요하게 다루어지는 영역의 하나이다. 이 논문에서는, 다중포트(multiport) 메모리 할당 문제에 대한 새로운 방법을 제안한다. 문제의 복잡도를 줄이기 위해, 기존의 연구들은 요약하면, 두 단계의 과정으로 이루어지고 있다. 첫 번째 단계에서는 변수들을 몇 개씩 묶어서 하나의 메모리를 형성한다. (즉, 메모리 최적화 문제를 푼다.) 두 번째 단계에서는, 메모리들과 기능모듈들 간의 연결선을 최적화시킨다. (즉, 연결선 최적화 문제를 푼다.) 이 경우 심각한 단점은 연결선의 비용을 최소화하는 데는 한계가 있다는 것이다. 다시 말해, 연결선의 비중이 점점 중요하게 되어지는 설계 추세에서 기존의 방법은 다중포트 메모리 사용을 통해 얻을 수 있는 연결선 최소화를 극대화하는데 한계가 있음을 뜻한다. 이를 극복하기 위해, 우리는 새로운 할당 방법을 제시한다. 구체적으로 먼저, 연결선 최소화를 해결하고, 그 다음에, 메모리 최적화를 시도한다. 또한, 제안한 알고리즘은 연결선 최소화 과정 동안 다음 단계에서 결정될 메모리 비용도 적절히 고려한다. 우리는 다양한 실험을 통해, 우리의 제안한 방법이 기존의 연구보다 상당히 효율적인 것임을 보인다.

Abstract Memory allocation is one of the most important areas in high-level synthesis. In this paper, a new approach to the problem of allocation of multiport memories for data storage is presented. To reduce the complexity of the problem, previous approaches solve the problem in two steps; First, the variables are grouped to form memories (i.e., optimizing memories). Then, the interconnections between the memories and functional units are determined (i.e., optimizing connections). One serious limitation of the approaches is that there is no easy way to predict the result of the second step during the first step. The situation is becoming worse since the importance of interconnections is increasing today. To overcome the limitation, we propose a new approach. Specifically, we minimize the cost of interconnections first and then group the variables to form memories later. In addition, we take into account the results of memory allocation during the connection minimization step in a proper way. From experimentations using a set of benchmark designs, we show that the proposed approach outperforms the pervious ones in reducing both interconnections and memories.

1. 서 론

상위단계 합성(High-level synthesis)은 주어진 Behavioral Description에서 요구된 하드웨어 및 수행시간을 만족하는 RTL(Register-Transfer Level) 설계를 생성해 내는 것을 말한다. 상위단계 합성에서의 두 가지 주요 작업은, 연산 수행 단계를 결정하는 스케줄링(Scheduling)과 하드웨어 할당(Hardware Allocation)이다 [1]. 하드웨어 할당 단계에서는, 계산 수행 도중에 생성된 변수(variable) 값들을 저장하기 위한 하드웨어로 레지스터(register)의 할당이 필요하다. 이때의 주된 최적화는 사용될 레지스터 개수를 줄이는 할당을 하는 것이다. 이에 대해서는 많은 연구들이 있어왔다[1].

· 본 연구는 한국과학재단 특정기초연구(과제번호 1999-2-302-002-3)의 지원을 받았음.

† 종신회원 : 한국과학기술원 전자전산학과 교수
tkim@cs.kaist.ac.kr
†† 학생회원 : 한국과학기술원 전자전산학과
hongsup@vlsisyn.kaist.ac.kr

논문접수 : 1999년 11월 22일
심사완료 : 2000년 7월 10일

본 논문에서는, 데이타 저장을 위한 하드웨어로서 개별의 레지스터가 아닌 다중포트(multi-port)를 가진 메모리 할당 문제를 다룬다. 다중포트 메모리는 레지스터들의 그룹으로 이루어지는데, 그 레지스터들은 메모리에 있는 여러 개의 (읽기전용(read-only), 쓰기전용(write-only), 읽기/쓰기(read/write)) 포트를 공유하게 된다. 이는 메모리 속에 있는 여러 레지스터가 동시에 엑세스(access)될 수 있고, 한 레지스터가 각기 다른 clock 스텝에서 각기 다른 메모리 포트를 통해서 데이타 엑세스를 할 수 있음을 말한다. 이러한 레지스터들의 포트 공유는 그 메모리와 기능모듈들(functional units)간의 연결선(interconnection)의 개수, 멀티플렉서의 개수, 그리고 tri-state 버퍼의 개수를 줄이는데 매우 효과적일 수 있다.

데이타 경로 합성에서 메모리를 활용하는 문제에 대한 여러 연구가 있어왔다[2, 3, 4, 5, 6, 7, 8, 9, 10]. [2]는 레지스터의 동시 엑세스 요구를 표현한 그래프에서, 노드들의 크리크 분할(clinique partition)에 기초한 방법을 제시하였다. 그러나, 이 방법은 연결선 개수의 최소화는 고려하지 않았으며, 단일포트 메모리에 제한하고 있다. MIMOLA[3]에서는 멀티포트 메모리의 활용을 고려했지만, 연결선 개수 최적화의 문제는 역시 다루지 않았다. [4]는 에지 색 지정(edge coloring) 알고리즘에 기초하여, (변수들을 레지스터에 할당하기 전에) 변수들을 몇 개씩 묶어 메모리를 만들어 내는 방식을 제시하였다. 일단 변수의 묶음이 끝난 다음에는, 모의 담금질(simulated annealing) 기법을 적용하여 연결선의 최소화를 시도한다. 이 방법 역시 단일포트 메모리에 제한되어 있다.

[5]는 레지스터들을 몇 개씩 묶어 다중포트 메모리를 형성하는 문제를 0-1 정수 선형 프로그래밍(0-1 integer linear programming) 문제로 표현하였다. 변수들이 레지스터에 할당된 다음에 가능한 그룹 중 가장 큰 그룹을 선택하여 다중포트 메모리로 형성한다. 그 다음, 남아 있는 레지스터들에 이 과정을 반복한다. 일반적으로는 이러한 반복 수행은 전체적인 다중포트 메모리 비용을 최적화하지 못한다. 연결선 최소화 문제 역시 0-1 정수선형 프로그래밍 문제로서 표현하였다. 이 수식 역시 한번에 하나의 다중포트 메모리에 적용된 관계로 전체 연결선 수를 최적화하지는 못한다고 할 수 있다. MAP[6]에서는 [5]의 접근 방법의 한계들을 극복하도록 확장하였다. 먼저 변수들을 그룹으로 나누고, 0-1 정수 선형 프로그래밍 수식을 적용하여 최소 개수의 다중포트 메모리을 결정한다. 그 다음, 메모리들과 기능모듈 사이의 연결선 개수를 최소화하는데, 가환(commutative) 연산에 의해 생겨난 추가적인 가능성도 고려하였다. 이 최소화 문제 역시 0-1 정수 선형 프로그래밍으로서 표현하였다. 요약하면, 이전 접근방법들[2, 3, 4, 5, 6]은 다중포트 메모리 할당 문제를 두 단계로 풀었다. 즉, 단계 1에서, 변수(레지스터)들을 몇 개씩 묶어서 메모리을 형성하고, 그 다음으로, 단계 2에서 메모리들과 기능모듈들 사이의 연결선 개수를 최적화하였다. 연결선의 개수는 변수들을 어떻게 묶어 한 메모리에 저장하는가에 좌우되므로, 단계 1의 결과가 단계 2에 강한 영향을 미친다. 그러나, 단계 1의 과정에서 단계 2의 결과를 예측하기란 쉽지 않다.

또한, 단수의 변수를 메모리에 배정하는 문제가 아니고, array 별로 메모리에 배정하는 문제를 다룬 방법들이 최근 많이 소개되었다. [7]은 embedded 시스템에서 On-칩과 Off-칩 메모리를 할당 최적화하는 기법들을 소개하였으며, [8]은 여러 개의 array를 하나의 메모리에 배열하는 최적화 방법을 다루었다. 여기서의 최적화 목표는 어떤 주어진 수행 제약시간 안에서 메모리 크기를 최소화하는 array mapping을 구하는 것이다. [10]은 소스 코드에 있는 array 엑세스들의 흐름 구조를 변형하여, 주어진 On-칩 메모리의 크기에 맞는 최적의 배정을 구하는 방법을 제안하였다. 하지만 [7, 8, 9, 10]의 방법들은 연결선의 비용은 고려되지 않았으며, 단지 단계 1의 할당할 메모리의 최적화에 그 목적을 두고 있다.

종래의 방법과 달리, 변수들에 대한 메모리 최적화 방법으로 우리는 메모리 비용보다는 연결선 최적화에 우선순위를 둔다. 다시 말하면, 연결선을 최소화하는 것을 먼저 결정하고, 그 다음, 메모리 최적화 시도를 한다. 따라서, 우리의 방법은 어떤 특정한 방법으로 변수들을 묶는 일없이 연결선 최소화를 수행하며, 그렇게 하는 동안, 연결선 비용과 메모리 비용의 균형을 또한 적절히 고려하게 된다.

2. 대상 아키텍처

우리의 설계 시스템은 선형(linear)과 임의(random) 위상(topology) 아키텍처 양쪽 모두에 적용 가능하다. 우리의 선형 위상 아키텍처는 MAP [6]에서의 그것과 같으며, 유일한 버스(bus)가 각각의 메모리 포트에 연결되어 있다. 그림 7은 선형 아키텍처 구조를 보여주고 잇다. 두 개의 메모리가 할당되어 있으며, 하나는 3 개의 포트를, 다른 하나는 2 개의 포트를 가지며, 각 포트는 고유의 버스에 직접 연결되어 있다. 더욱이, 우리의 시스템은 one-phase clock(한 clock에 변수의 읽기와 쓰기에 대해 동일 메모리의 사용이 불가함) 과 two-phase clock(한 clock에 변수의 읽기와 쓰기에 대해 동일 메모리의 사용이 가능함)방식 모두를 지원한다.

3. 메모리 모듈 할당 알고리즘

계산수행 단계의 스케쥴링과 각 오퍼레이션을 수행할 기능모듈 배정은 이미 이루어진 것으로 우리는 가정한다. 즉, 기능 모듈들과 각 clock 스텝에서 어떤 변수들이 엑세스될 것인가는 알고 있다. 그렇지만, 변수의 데이타 값들이 메모리들의 어느 레지스터에 저장될 것인지와 메모리들과 기능모듈들 사이의 연결은 아직 결정되지 않았다. 또한, 우리는 하나의 변수는 꼭 하나의 레지스터에 할당되는 것으로 가정한다. (다시 말하면, 데이타의 복사를 고려하지는 않는다.) 예를 들어, 그림 1은 스케쥴링과 기능모듈 배정이 이루어진 계산수행을 보여준다. 구체적으로, clock 스텝 1에서는 v1, v2, v3, v4, v5, v6, v18, v19, v20의 변수들이 엑세스되며, v1과 v2는 기능모듈 FU1의 두 입력 터미널에 연결되어야 하며, v3과 v4는 기능모듈 FU2의 두 입력 터미널에 연결되어야 함을 의미한다.

cstep 1: $v18 = v1 +_a v2;\ v19 = v3 -_b v4;\ v20 = v5 /_c v6;$

cstep 2: $v22 = v9 -_a v3;\ v24 = v11 *_b v1;\ v21 = v7 -_c v8;$

cstep 3: $v29 = v19 -_a v20;\ v37 = v18 /_b v20;\ v27 = v2 +_c v6;$
$v28 = v18 -_d v19;$

cstep 4: $v30 = v20 /_a v21;\ v31 = v21 -_b v22;\ v32 = v5 *_c v4;$

cstep 5: $v35 = v27 -_a v28;\ v38 = v30 /_b v21;\ v36 = v28 -_c v29;$

cstep 6: $v39 = v31 -_a v32;\ v40 = v24 -_b v33;\ v43 = v35 /_c v28;$

cstep 7: $v45 = v29 -_a v37;\ v46 = v37 *_b v30;\ v44 = v18 +_c v28;$

cstep 8: $v47 = v38 +_a v31;\ v48 = v32 -_b v24;$

a :FU1, b :FU2, c:FU3, d:FU4

그림 1 스케줄된 연산 순서

이제 우리가 최소화하고자 하는 비용은 다음과 같이 표현 될 수 있다.

$$Conn_{cost} + w \cdot M_{cost} \quad (1)$$

여기서 $Conn_{cost}$는 총 연결선의 비용, M_{cost}은 총 사용된 메모리의 비용, w는 가중치이다.

우리의 알고리즘은 다음의 두 과정을 반복 수행함에 의해 위의 표현한 비용의 최소화를 시도한다.

과정 1 : 가상의 커다란 한 개의 메모리가 있다고 생각한다. 이 메모리의 포트는 t1 개의 읽기전용, t2 개의 쓰기전용, 그리고 t3 개의 읽기/쓰기 포트로 구성되어 있다고 하자. (t1, t2, t3 값의 결정은 3.1 절과 3.3 절에서 명확히 설명한다.) 각 clock 스텝마다, 엑세스 될 각 변수가 사용하게 될 메모리의 포트를 결정해야 하며, 이 결정으로 인해 배정된 포트에서 그 변수가 생성된 (또는 사용될) 기능모듈의 터미널로 연결선이 만들어지게 된다. 여기서 주목하고자하는 것은, 동일의 변수가 서로 다른 clock 스텝에서, 다른 메모리 포트를 통해 터미널들에 연결될 수 있다는 것이며, 이는 한 연결선의 반복적 공유를 가능하게 하여 결국 변수들을 메모리 포트에 잘 배정 연결함으로서, 메모리 포트와 기능모듈의 터미널 사이의 최종 연결선을 최소화할 수 있다는 것이다.

과정 2 : 과정 1 에서 얻은 가상의 메모리에서 행한 변수들에 대한 메모리 포트 배정의 결과를 가지고, 한 변수는 꼭 한 메모리에 저장되어야 한다는 본래 제약을 충족하도록, 그 가상의 메모리를 포트 단위 별로 작게 분할하여 그 분할된 것들을 다시 합치는 과정을 통해 메모리 비용을 최소화시킨다.

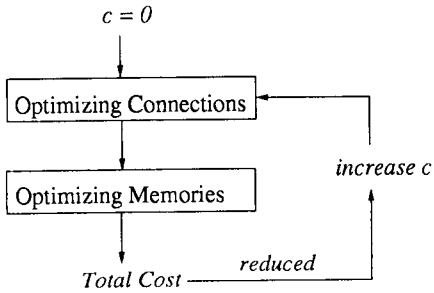


그림 2 제안한 알고리즘의 흐름도

그림 2는 우리의 제안한 두 과정의 알고리즘 흐름을 보여준다. c는 제어 매개변수로서 연결선 최소화 과정에서 메모리에 대한 예측 비용을 고려하는데 사용된다. 각 반복수행에서, 위의 두 과정이 실행되고 나면, 전체 비용 $Conn_{cost} + w \cdot M_{cost}$을 계산하며, 그 전의 반복과정에서 구한 전체 비용보다 감소가 없을 때까지 수행이 계속된다. 이 두 과정에 대해 이제 3.1절(연결선 최적화)과 3.2절(메모리 최적화)에서 자세히 다룬다.

3.1 변수에 대한 메모리포트 배정 및 포트와 기능모듈 사이의 연결선 할당

설명의 복잡성을 피하기 위해 우리는 one-phase clock 방식을 가정한다. 또한, 모든 메모리 포트들이 읽기/쓰기 포트라고 가정하자. (읽기전용, 쓰기전용, 읽기/쓰기를 가진 메모리로의 일반화는 3.3절에서 다룬다.) 변수들을 포트에 할당하는데 있어서, n 개의 레지스터와 m 개의 포트로 이루어진 하나의 (가상적인) 커다란 메모리를 생각하자. 각각의 clock 스텝에서 변수들이 그 메모리 안에서 마치 별개의 레지스터들에 할당되어 있는 것처럼 간주하

고, 변수들을 메모리 포트들에 배정시킨다.따라서, n은 단순히 스케쥴된 연산 순서에서의 변수의 총 개수가 되고, m은 한 clock스텝에서 동시에 엑세스되는 변수의 최대 개수와 같다.

만약 변수 vi가 한 clock 스텝에서 포트 pj에 배정되었다면, 변수 vi와 포트 pj에 내부(internal) 연결선이 있다고 말한다. 예를 들어, 변수 v1이 두 개의 다른 clock 스텝에서 각기 포트 p1과 p2를 통해서 엑세스된다고 해보자. 이 때 v1과 두 개의 포트 사이에는 두 개의 내부 연결이 있게 된다. 또한, 포트 p1과 p2는 과정 2에서 변수들과 포트들을 분할할 때 같은 메모리에 할당되어 지도록 해야 하는 제약을 받게 된다. (만약 그렇게 되지 않으면, 변수 v1은 두 개의 메모리에 자리잡게 된다.) 반면에, v1이 두 clock 스텝 모두에서 포트 p1을 통해 엑세스 되도록 배정하였다면, v1과 p1 사이의 내부 연결은 하나만 있게 된다. 나중에 다시 자세히 언급되겠지만, 내부연결의 개수는 메모리 전체 비용과 밀접하게 연관되어 있다. 즉, 적은 수의 내부연결은 낮은 메모리 비용을 이루게 하는 경향이 매우 강하다는 것이다. 반면, 외부연결은 실제 회로 설계에서 말하는 연결선을 의미하며 메모리 비용과 더불어 그 수를 줄이는 것은 우리의 목표이다.

변수 vi가 포트 pj에 배정되어 있다고 하자. 만약 vi가 가환(commutative) 연산을 수행하는 (예: +, *) 기능모듈의 입력이라면, pj는 이 기능모듈의 두 입력 터미널 중 어느 곳에도 연결될 수 있다.[1] 반면에, vi가 불가환 연산을 수행하는 (예. -) 기능모듈의 입력이거나, 기능모듈의 출력일 경우에는 pj는 그 기능모듈의 특정 터미널에만 연결되게 된다.

포트 pj가 터미널 tk와 연결될 경우에, 우리는 pj와 tk사이에 외부(exnternal) 연결선이 있다고 말한다. 변수들을 (가상의 메모리) 포트에 배정하고, 이로 인해 메모리 포트들을 기능모듈의 터미널에 연결될 때, 우리가 최소화하고자 하는 비용은 다음과 같다.

$$\alpha \cdot \sum_{j=1}^{m} EXT_j + c \cdot \beta \cdot \sum_{i=1}^{n} INT_i$$

여기서 EXTj는 메모리 포트 pj와 기능모듈 터미널들 사이의 외부연결의 개수이고, INTi는 변수 vi와 메모리 포트들 사이의 내부연결의 개수이다. 그러므로, 모든 포트 p1, p2, ... , pm에 대해서 합한 $\sum_{j=1}^{m} EXT_j$는 외부연결의 총 개수이고, 모든 변수 v1, v2, ..., vn에 대해서 합한 $\sum_{i=1}^{n} INT_i$는 내부연결의 총 개수이다. α는 외부연결 하나의 비용이고, β는 내부연결 하나의 비용이다.[2] c는 제어 매개변수로서 외부연결의 비용과 메모리 비용의 균형 잡는데 사용된다. (메모리 비용은 내부연결의 총 개수와 밀접히 연관되어 있다.) 문제 크기가 충분히 작을 때는 0-1 정수 선형 프로그래밍 수식을 사용하여, 모든 clock 스텝에 대해서 식 (2)의 최소 값을 찾을 수 있다. 그러나, 문제 크기가 커지게 되면 변수들을 포트에 배정하고 포트들을 터미널에 연결하는데, clock 스텝 하나씩 차례로 하는 방식으로, 각각의 clock 스텝에서 식 (2)의 추가적인 값을 최소화하도록 하는 배정을 시도한다. 즉, V = {v1, v2, ...}가 어떤 한 clock 스텝에서 액세스되는 변수들의 집합이라고 하자. V의 변수들이 포트들에 배정되고, 포트들이 관계되는 터미널들과 연결되었을 때, ΔEXT 가 포트들과 터미널들 사이에 새로 생겨난 외부연결의 개수라고 하자. (몇몇의 포트로부터 몇몇의 터미널들로의 외부연결은 그 전의 clock 스텝들에서 이미 만들어져 있다.) ΔINT 가 변수들과 포트들 사이에 새로 생겨난 내부연결의 개수라고 정의한다. 또한,

$\Delta TOT = \alpha \cdot \Delta EXT + c \cdot \beta \cdot \Delta INT$ 라고 하면, ΔTOT 는 추가적 비용이라고 할 수 있다. 문제는 변수들을 포트에 할당하고, 포트들을 터미널에 연결하되, 증가된 비용 ΔTOT 가 최소가 되도록 하자는 것이다. 이 문제는, 일반적인 최적화 문제를 푸는 식으로 0-1 정수 선형 프로그래밍 수식을 만들 수 있다. 그러한 수식은 MAP [6]에서와 유사하기 때문에 (단지 내부연결의 개수를 추가로 고려해 두었다는 점이 다르다.) 자세한 수식은 이 논문에서 생략하도록 한다.

3.2 메모리 모듈 비용 최소화

이제 우리는 변수들과 (3.1절에서 사용한 가상의 큰) 메모리의 포트들을 분할하여, 실질 메모리들을 형성하고자 한다. 목표는 총 메모리 비용을 최소화하는 것이다. 3.1 절에서 변수들을 포트에 할당한 결과를 이용하여 포트들 사이의 관계를 그래프로 표현할 수 있다. 예로서, 그림 3은 그림 1의 스케쥴된 연산 순서에서의 변수들을 메모리 포트에 배정한 결과이다. 그림 3 결과로부터 그림 4 (a)의 그래프가 구성된다; 그래프의 각각의 노드들은

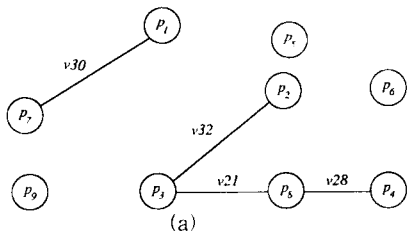
1) 우리의 논의에서는 모든 가환 연산은 이항(binary) 연산이라고 가정한다. 이 가정은 쉽게 n항 연산으로 일반화 될 수 있다.

2) α 와 β 는 실질 설계자의 설계 구조에 대한 중요도와 관계 있다. 예로서, 메모리 비용보다 설계시 routing의 용이를 위해서는 연결선에 보다 더 큰 비중을 두게 된다. 그러나, 절대적인 α 와 β 값은 많은 실험을 통해 경험적으로 얻게 되며, 사용하는 target technology에 따라서 달라진다.

포트를 나타내고, 두 개의 노드들은 그 노드들에 해당하는 두 포트 모두를 통해서 엑세스되는 변수가 있을 경우 서로 연결된다. 이를테면, 변수 v21은 clock 스텝 2와 4에서 포트 p3에 배정되고, clock 스텝 5에서는 포트 p8에 배정되어 있다. 따라서, 노드 p3과 p8사이에는 v21이라고 이름 붙은 에지가 있게 된다.

port:	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
c-step: 1	v1	v2	v18	v3	v4	v19	v5	v6	v20
2	v1	v8	v21	v3	v7	v22	v11	v9	v24
3	v27	v2	v18	v28	v29	v19	v37	v6	v20
4	v30	v32	v21	v31	v4	v22	v5		v20
5	v27	v38	v35	v28	v29	v36	v30	v21	
6	v39	v32	v35	v31	v33	v40	v43	v28	v24
7	v30	v46	v18	v28	v29	v45	v37	v44	
8	v47	v38	v32	v31		v48			

그림 3 그림 1의 변수들에 대한 메모리 포트 배정 예



(a)

unit1 (4,7) :({p2, p3, p4, p8}, {v2, v8, v32, v38, v46, v18, v21, v35, v6, v9, v28, v44, v3, v31})
unit2 (2,4) :({p1, p7}, {v1, v27, v30, v39, v47, v5, v11, v37, v43})
unit3 (1,4) :({p6}, {v19, v22, v36, v40, v45, v48})
unit4 (1,3) :({p5}, {v4, v7, v29, v33})
unit5 (1,2) :({p9}, {v20, v24})

(b)

그림 4 그림 3으로부터의 유닛 생성

이렇게 구성된 그래프는 상호 연결된 구성요소들로 나뉘어질 수 있다. 모든 변수들은 단 하나의 레지스터에만 저장되어야 한다는 조건으로 인해, 같은 상호 연결된 구성요소에 있는 모든 포트들과 변수들은 동일 메모리에 있어야 한다. 이러한 포트들과 변수들의 모음을 기본 유닛(unit)이라고 부른다. 하나의 유닛에서 변수들을 저장하기 위해 필요한 레지스터의 개수는 변수들의 생존시간(lifetime)을 분석해보면 알 수 있다 [1, 11]. 유닛의 크기는 포트의 개수와 레지스터의 개수의 순서쌍으로 정의된다. 그림 4 (b)는 그림 4 (a)의 그래프의 상호 연결 부분에 해당하는 다섯 개의 유닛들의 크기들을 보여준다.

예를 들어, 유닛 1의 크기 (4,7)은 유닛 1은 4개의 포트와 최소 7개의 레지스터가 있어야 변수 {v2, v8, v32, v38, v46, v18, v21, v35, v6, v9, v28, v34, v3, v31}을 저장할 수 있음을 말한다. 설계자는 사용 가능한 메모리들의 라이브러리를 가지고 있다고 우리는 가정한다. 목표는 유닛들을 메모리들에 맞추어 넣는데, 할당한 메모리 총 비용을 최소로 하는 것이다.

먼저 균일한 크기의 메모리들이 충분히 공급되는 특별한 경우를 고려해 보자. 균일한 크기의 메모리들에 유닛들을 맞추어 넣는 것은, 이차원 bin-packing 문제[12]로 볼 수 있다. 자세히 말하자면, 메모리들은 이차원 bin으로서 메모리 포트의 개수와 레지스터의 개수가 두 차원을 이룬다고 볼 수 있다. 유닛들은 모서리에서 모서리로 대각선방향으로 메모리를 가로질러 배열되게 된다. 이러한 유형의 bin-packing 문제에 대해서는 잘 연구된 휴리스틱들이 있다[15]. 우리는 그 중 시간 복잡도가 상수의 상한 값을 가지는 first-fit decreasing 알고리즘을 우리의 문제에 적용한다 [12, 13]. 먼저 계수 ρ를 계산하는데, ρ는 다음과 같이 정의된다.

$$\rho = \sum_{unit\ i} (\#ports\ in\ i) / \sum_{unit\ i} (\#registers\ in\ i) \quad (3)$$

각각의 유닛에 대해서, 두 개의 숫자, 즉, 포트의 개수와 ρ·(레지스터개수)중에서 큰 쪽을 결정한다. 이때, 이 결정된 값을 그 유닛의 표준화 스칼라 크기라 부른다. 유닛들은 표준화 스칼라 크기에 따라 큰 순서대로 배열되어 있고, 그 순서대로 메모리에 놓여질 것이다. first-fit decreasing 알고리즘은, 한 유닛을 메모리에 배치하는데, 그 유닛을 수용할 수 있는 첫 번째 메모리 모듈에 위치시킨다. 그리고 이러한 과정을 유닛이 배열된 순서대로 모든 유닛에 반복하여 적용한다.

우리의 bin-packing 문제가 일반적인 bin-packing 문제와는 한가지 중요한 점에서 다르다: 두 개나 그 이상의 유닛들이 메모리에 자리잡고 나면, 그것들이 사용하는 포트 개수는 각 유닛의 포트 개수들의 합과 같은 반면에, 유닛들이 사용하는 레지스터의 총 개수는 각 유닛의 레지스터의 개수의 합보다 작아질 수 있다. 이것은 왜냐하면 유닛의 변수들을 모두 한 묶음으로 생각함으로써, 레지스터 할당을 좀 더 잘 할 수 있는 방법이 있기 때문이다. 그러므로, 일단 유닛들이 메모리에 자리잡고 나면, 유닛들의 변수들에 필요한 레지스터들의 개수는 변수들의 생존시간에 맞추어 다시 계산한다. 이는 차후의 유닛의 packing을 더 유리하게 할 수 있는 공간을 제공할 수 있기 때문이다.

식 (2)에서 제어 매개변수 c가 과정 1에서 얻어지는 유닛들의 개수에 영향을 미치고 있음을 알 수 있다. 작은 c값은 큰 크기의 유닛들을 생성하게 하고, 그 반대도 성립한

다. 그러므로, 만약 유닛들을 메모리들에 맞추어 넣을 수 없거나, 낭비가 별로 없도록 딱 맞추어 넣지 못한 경우에는, c 값을 증가시킨 다음 과정 1을 다시 반복하여 변수들을 포트에 배정하는 것을 다르게 시도해 보도록 한다.

그림 1의 예를 계속 살펴보자. 3개의 읽기/쓰기 포트와 7개의 레지스터를 가지는 메모리가 많이 있다고 해 보자. c=2로 놓았을 때, 우리는 그림 3에서 나타난 배정을 얻었다. (이때 $\alpha=\beta=1$로 놓았다.) 이에 해당한 유닛들이 그림 4에 나타나 있는데, 이것들은 주어진 메모리들에 맞추어 넣을 수 없음을 알 수 있다, 왜냐하면, 크기 (4,7) 짜리 유닛이 있기 때문이다. c의 값을 3으로 증가 시켰을 때에 얻어진 유닛들을, 표준화 스칼라 크기가 큰 순서로 정렬하여, 그림 5에 나타내었다. 그림 6(a)는 이 유닛들을 메모리들에 꾸려 넣은 모습을 보여준다. 유닛 3과 유닛 5가 같은 메모리 M3에 놓여졌을 때, 레지스터의 개수가 그림 6(b)에 나타난 것처럼, 다시 계산되었음을 알 수 있다.

메모리들이 균일한 크기가 아닐 때에는, 각각의 메모리들의 표준화 스칼라 크기를 계산하여, 메모리들을 큰 순서대로 늘어놓는다. 그리고 나서, 유닛들은 first-fit decreasing 알고리즘에 따라 처리한다.

unit1 (3,6) :{(p2, p3, p4), (v2, v7, v32, v38, v46, v18, v21, v28, v3, v31)}
unit2 (2,5) :{(p1, p7), (v1, v27, v30, v39, v47, v5, v11, v37, v33)}
unit3 (1,4) :{(p6), (v19, v22, v36, v40, v45, v48)}
unit4 (1,2) :{(p5), (v4, v9, v29, v43)}
unit5 (1,2) :{(p8), (v6, v8, v35,v44)}
unit6 (1,2) :{(p9), (v20, v24)}

그림 5 더 작은 크기의 유닛들의 생성

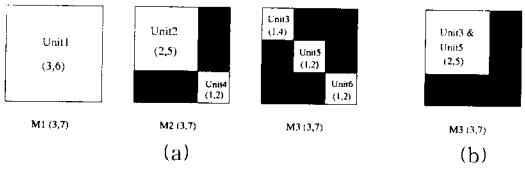


그림 6 (a) 유닛들을 메모리 모듈에 꾸려 넣는 모습 (b) 레지스터 개수 재 계산

3.3 메모리 포트 타입의 일반화

언급한 바와 같이 메모리는 읽기/쓰기 뿐만 아니라 읽기전용, 쓰기전용의 포트도 가질 수 있다. trk, twk를 각각 clock 스텝 k에서의 변수 읽는 것과 쓰는 것의 수라고 하자. 또한, t1, t2, t3를 각각 과정 1에서 사용한 가상의 큰 메모리가 가지는 읽기전용, 쓰기전용, 읽기/쓰기 포트의 개수라고 하자. 그러면, 우리는 t1, t2, t3을 각 clock 스텝에서 다음 조건들을 만족하도록 하는 어떤 값들로 선정할 수 있다 :

$$t_r^k \le t_1 + t_3,$$
$$t_w^k \le t_2 + t_3,$$
$$(t_r^k - t_1) + (t_w^k - t_2) \le t_3$$

예를 들어, 그림 1에서 t1, t2, t3을 각기 2, 1, 6으로 정하면, 위의 제약 식들을 만족한다. 과정 2의 유닛들의 packing은 단지 메모리의 가용한 읽기전용, 쓰기전용, 읽기/쓰기 포트 수에 의해 제약을 받게 된다.

4. 실 험

우리의 알고리즘을, 여러 벤치마크 설계들에 적용하여 그 결과를 다른 기존의 시스템에 의해 만들어진 결과와 비교하였다. 우리의 알고리즘에서는, 오퍼레이션을 기능 모듈에 배정하는 문제를 Huang의 가중치를 가진 bipartite matching 알고리즘[14]으로 해결하였으며 연결선 비용을 고려해 넣었다.

표 1은 [15]에서의 6-order bandpass elliptic filter에 대한 결과이다. 우리는 이 결과를 모든 상수들은 ROM에 저장되어 있다는 가정 하에서 MAP[6]와 ADPS[15]의 것과 비교하였다. 우리의 알고리즘은 MAP에 비하여 더 적은 수의 멀티플렉서 입력과 tri-state 버퍼를 사용하였다. (MAP에 의해 사용된 두 개의 메모리 모듈들의 포트 구조는 알 수 없었다.) ADPS는 메모리 모듈을 사용하지 않는데, ADPS와 비교한 것은 참고를 위한 것이다.

표 2는 [16]에서의 5-order elliptic wave filter에 대한 결과이다. 우리의 알고리즘은 MAP[6] 보다 적은 수의 멀티플렉서 입력을 사용하였고, STAR[17]와 SPAID[18]보다는 훨씬 적은 수의 멀티플렉서 입력을 사용하는데, 이들은 단일 포트 메모리를 사용하였다. 표의 SPAID 열에서 괄호 안에 나타나 있는 숫자는 메모리 모듈들과 버스 사이의 멀티플렉서 입력의 개수를 포함했을 때의 개수를 나타낸다. 그림 7에서는 실제로 우리의 알고리즘에 의해 표 2에 해당하는 합성된 데이타 경로를 보여주고 있다.

표 3은 메모리를 단일의 포트를 가진 것으로 제약한 상황에서 구한 결과를 STAR[17], SPAID[18], EASY[4]의 것과 비교한 것이다. 여태까지 가장 효율이 좋다고 알려진 STAR의 것과 비교하여서도 멀티프레스 입력과 tri-state 버퍼 수를 상당히 많이 줄인 것을 볼 수 있다. 연결선 하나를 줄이는 것은 word-단위인 관계로 bit 수로 환산하면 상당히 많은 양에 해당한다고 할 수 있다. 따라서, 이는 분명, 연결선 최소화에 중점을 둔 우리의 알고리즘이 매우 효과적임을 잘 나타내고 있다.

표 1 6-order bandpass elliptic filter에 대한 결과

	Ours	MAP	ADPS
#c_steps	11	11	11
#ALU	2	2	2
#Multipliers	1	1	1
#MUX_inputs	10	12	27
#Tri state Buf.	6	7	N/A
#Buses	5	5	N/A
#Registers	11	11	11
#RAMs	2[a]	2[b]	N/A
#ROMs	1[c]	1[c]	N/A

a: 3개의 읽기/쓰기 포트를 가진 메모리와 2개의 읽기/쓰기 포트를 가진 메모리 g라당

b: 미상 c: 1개의 읽기전용 포트를 rwls 메모리 할당

표 2 5-order elliptic wave filter에 대한 결과

	Ours	MAP	STAR	SPAID
#c_steps	19	19	19	19
#ALU	2	2	2	2
#Multipliers	1	1	1	1
#MUX_inputs	9	10	17	17(22)
#Tri state Buf.	5	5	16	N/A
#Buses	5	5	5	5
#Registers	12	14	13	19
#RAMs	2[a]	2[b]	5[d]	5[d]
#ROMs	1[c]	1[c]	1[c]	1[c]

a: 3개의 읽기/쓰기 포트를 가진 메모리와 2개의 읽기/쓰기 포트를 가진 메모리 g라당

b: 미상 c: 1개의 읽기전용 포트를 rwls 메모리 할당 d: 1개의 읽기/쓰기 포트를 가진 메모리 할당

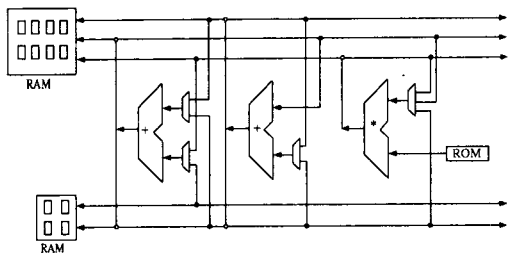


그림 7 5-order elliptic wave filter의 합성된 데이타 경로

표 3은 메모리를 단일의 포트를 가진 것으로 제약한 상황에서 구한 결과를 STAR[17], SPAID[18], EASY[4]의 것과 비교한 것이다. 여태까지 가장 효율이 좋다고 알려진 STAR의 것과 비교하여서도 멀티프레스 입력과 tri-state 버퍼 수를 상당히 많이 줄인 것을 볼 수 있다. 연결선 하나를 줄이는 것은 word-단위인 관계로 bit 수로 환산하면 상당히 많은 양에 해당한다고 할 수 있다. 따라서, 이는 분명, 연결선 최소화에 중점을 둔 우리의 알고리즘이 매우 효과적임을 잘 나타내고 있다.

표 3 단일 포트를 가진 메모리 사용하에서 5-order elliptic wave filter에 대한 결과

	Ours	STAR	SPAID	EASY1	EASY2
#c_steps	19	19	19	19	19
#ALU	2	2	2	2	2
#Multipliers	1	1	1	1	1
#MUX_inputs	14	17	17(22)	19	25
#Tri state Buf.	12	16	N/A	N/A	N/A
#Buses	5	5	5	8	8
#Registers	14	13	19	17	15
#RAMs	5[a]	5[a]	5[a]	8[a]	[illegible][a]
#ROMs	1[b]	1[b]	1[b]	1[b]	1[b]

a: 1개의 일기/쓰기 포트를 가진 메모리 할당

b: 1개의 일기전용 포트를 가진 메모리 할당

주목할 것은 과정 1의 변수들의 메모리 포트 배정은 매 clock 스텝마다 0-1 정수 선형 식을 풀어 해결하였다. 그러므로, 최종 결과는 최적(optimal)이라고 할 수 없다. 전체의 clock 스텝을 통한 0-1 정수 선형 식을 벤치마크 설계들에 적용하였을 때 우리는 24 시간 안에 해답을 구하기가 불가능함을 발견하였다. 하지만, 표들의 비교에서 보듯이 기존의 방법들에 의한 결과보다는 분명 향상된 것임을 알 수 있었다.

5. 결 론

본 논문에서 우리는 데이타 경로 설계에서 다중포트 메모리를 사용하여서 메모리 비용 및 연결선을 최적화하는 새로운 방법을 제시하였다. 기존의 접근 방식들은 최적화 문제를 두 단계로 나누었었다, 즉, 먼저, 변수들을 몇 개씩 묶어서 메모리을 형성한다. 그 다음, 메모리들과 기능모듈들 간의 연결선 최소화를 시도하였다. 이와는 달리, 우리의 방법은, 메모리 비용보다는 연결선의 비용에 주된 중점을 두었다. 이를 위해, 먼저 가상의 큰 메모리에 있는 포트들에 변수들을 자유로이 배정함으로서 연결선의 최소화를 해결하고, 그 다음, 변수들과 포트들을 메모리에 할당하는 방법을 시도하였다. 실험결과를 통해서 이러한 접근 방법이 메모리 기반 아키텍처에 있어서 매우 효과적임을 보여 주었다.

마지막으로 우리의 제안한 알고리즘는 greedy한 반복 수행을 하기 때문에 결과가 local 최적화에 머물 수 있다.

이를 보완하기 위해서는 현재의 제시된 알고리즘에서는 많은 반복 수행으로 극복될 수 있다. 하지만, 근본적 문제 해결을 위해서는 보다 심도있는 global 최적화 연구가 앞으로 진행되어야 할 것이다.

6. 감사의 글

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

참 고 문 헌

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

[2] C.-J. Tseng and D.P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," IEEE Trans. on Computer-Aided Design, Vol. CAD-5, No. 3, pp. 379-395, July 1986.

[3] P. Marwedel, "The MIMOLA design system: Tools for the Design of Digital Processors," Proc. Design Automation conference, pp. 587-593, 1984.

[4] L. Stok, "Interconnection Optimization During Data Path Allocation," Proc. European Design Automation Conference, pp. 141-145, 1990.

[5] M. Balakrishnan et al., "Allocation of Mutiport Memories in Data Path Synthesis," IEEE Trans, on Computer-Aided Design, Vol. 7, No. 4, pp. 536-540, April 1988.

[6] I. Ahmad and C. Y. Roger Chen, "Post-Process for Data Path Synthesis using Multiport Memories," Proc. International Conference on Computer-Aided Design, pp. 276-279, 1991.

[7] P. R. Panda, N. Dutt, and A. Nicolau, "Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration," Kluwer Academic Publishers, Morwell, MA, 1999.

[8] L. Ramachandran, D. D. Gajski, and V. Chaiyakul, "An algorithm for Array Variable Clustering," Proc. of European Design Automation conference, pp. 262-266, 1994.

[9] H. Schmit and D. E. Thomas, "Synthesis of Application Specific Memory Designs," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 5, No. 3, pp. 101-111, March 1997.

[10] P. R. Panda, "Memory Bank Customization and Assignment in Behavioral Synthesis," Proc. International Conference on Computer-Aided Design, pp. 477-481, 1999.

[11] F. J. Kurdahi and A. C. Parker, "REAL: A Program for Register Allocation," Proc. Design Automation Conference, pp. 80-85, 1987.

[12] E. G. Coffman et al., "Approximation Algorithms for Bin Packing - An Updated Survey" in Algorithm Design for Computer System Design, M. Luccertini, G. Ausiello and P. Serafiri, Springer Verlag, 1984.

[13] M. R. Garey, R. L. Graham, D. S. Johnson and A. C. Yao, "Resource Constrained Scheduling as Generalized Bin Packing," J. combinatorial Theory, Ser. A21, pp. 257-298, 1976.

[14] C. Y. Huang, Y. S. Chen, Y. L. Lin, Y. C. Hsu, "Data Path Allocation Based on bipartite Weighted Matching," Proc. Design Automation Conference, pp. 499-504, 1990.

[15] C. A. Parpachristou and H. Konuk, "A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnection Optimization Algorithm," Proc. Design Automation Conference, pp. 77-83, 1990.

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

[17] F. S. Tsai and Y. C. Hsu, "Data Path Construction and Refinement," Proc. International Conference on Computer-Aided Design, pp. 308-311, 1990.

[18] B. S. Haroun and M. I. Elmasry, "Architecture Synthesis for DSP Silicon Compiler," IEEE Trans. on Computer-Aided Design, Vol. CAD-8, No. 4, pp. 431-447, April 1989.

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

홍 성 백
1999년 한국과학기술원(KAIST) 전산학과 학사. 1999년 ~ 현재 한국과학기술원 (KAIST) 전산학과 석사과정 재학.