論文97-34C-3-1

유전자 알고리즘을 이용한 분할 버스 아키텍처의 상위 수준 합성

(A Genetic-Algorithm-Based High-Level Synthesis for Partitioned Bus Architecture)

金 容 主 *, 崔 起 榮 *

(Yongjoo Kim and Kiyoung Choi)

요 약

본 논문에서는 분할 버스 아키텍처의 상위 수준 합성을 위한 한 가지 방법을 제안한다. 본 방법에서는 제어 단계의 수를 최소화하거나 주어진 제약 조건을 만족하는 등과 같은 통상적인 합성 목표는 물론 데이터 전송 길이와 버스의 수를 최소화하는 것을 주된 목표로 한다. 데이터 전송 길이와 버스의 수를 최소화하는 것은 상호 연결로 인한 지연 시간과 소요되는 면적이 설계하고자 하는 칩의 지연 시간과 면적에서 상당한 비중을 차지하는 딥 서브 마이크론 기술의 응용에 있어 매우 중요한 과제가 되고 있다. 분할 버스 아키텍처에서 이러한 합성 목표들을 만족하는 최적의 해를 얻기 위해서는 스케줄링과 할당 및 바인딩 뿐만 아니라 노드들의 세그먼트별 분할과 세그먼트의 순서 정의 문제들도 동시에 고려되어야 한다. 이러한 추가적인 합성 목표로 인해 기존의 상위 수준 합성 문제의 복잡도는 훨씬 증가하게 된다. 이러한 복잡도의 증가에 대처하여 좋은 합성 결과를 얻기 위해서, 본 합성 방법에서는 두 가지 아이디어, 즉 데이터 전송에 소요되는 버스 요구를 완화하기 위해 타겟 아키텍처를 확장하고 유전자 알고리즘을 중심적인 설계 공간 탐색 방법으로 이용하는 것을 채택하였다. 실험 결과는 본 방법이 분할 버스 아키텍처를 위한 효과적인 상위 수준 합성 방법이 될 수 있음을 보여준다.

Abstract

We present an approach to high-level synthesis for a specific target architecture - partitioned bus architecture. In this approach, we have specific goals of minimizing data transfer length and number of buses in addition to common synthesis goals such as minimizing number of control steps and satisfying given resource constraint. Minimizing data transfer length and number of buses can be very important design goals in the era of deep submicron technology in which interconnection delay and area dominate total delay and area of the chip to be designed. In partitioned bus architecture, to get optimal solution satisfying all the goals, partitioning of operation nodes among segments and ordering of segments as well as scheduling and allocation/binding must be considered concurrently. Those additional goals may impose much more complexity on the existing high-level synthesis problem. To cope with this increased complexity and get reasonable results, we have employed two ideas in our synthesis approach - extension of the target architecture to alleviate bus requirement for data transfer and adoption of genetic algorithm as a principal methodology for design space exploration. Experimental results show that our approach is a promising high-level synthesis methodology for partitioned bus architecture.

* 正會員, 서울大學校 電氣工學部

(School of Electrical Engineering, Seoul National University)

接受日字:1997年1月31日, 수정완료일:1997年3月14日

I. 서 론

상위 수준 합성은 디지털 시스템의 동작에 대한 알고리즘 수준의 사양 입력으로부터 이 동작을 구현하기 위한 레지스터 전송 수준의 구조를 합성하는 것을 가

리킨다. 합성 시스템은 입력 사양으로부터 기능 유닛, 레지스터, 멀티플렉서, 버스 등과 같은 레지스터 전송 수준의 요소들이 연결된 회로를 생성한다. 합성 시스템은 제어부의 사양도 생성해야 한다. 동기 디지털 시스템에서는 제어는 마이크로코드, PLA 프로파일, 또는 랜덤 논리 형태로 된 한 개 또는 그 이상의 유한 상태기(finite state machine)들에 의해 제공된다[12].

일반적으로 합성 과정을 단순화하기 위해 특정한 타겟 아키텍처를 상위 수준 합성을 위한 합성 모델로 가정하는 경우가 대부분이다. 타겟 아키텍처는 기능 유닛과 저장 유닛 간의 데이터 전송을 위한 연결 형태에 따라 멀티플렉서 지향 아키텍처와 버스 지향 아키텍처로 분류될 수 있다[1]. 버스 지향 아키텍처는 멀티플렉서 지향 아키텍처에 비해 할당 과정이 다소 복잡하나 소요 면적과 상호 연결의 활용도 측면에서는 보다 우수하다.

본 논문에서는 버스 지향 아키테처의 특별한 경우인 분할 버스 아키텍처의 상위 수준 합성을 하나의 통합된 과정으로 실행하는 방법을 제안한다. 이 합성 과정에서는 제어 단계의 수를 최소화하는 등과 같은 통상적인 합성 목표 외에 버스를 통한 데이터 전송의 길이와 요구되는 버스의 수를 최소화하는 것을 목표로 한다. 특히 딥 서브 마이크론 설계에서는 버스 등을 위한 상호 연결의 지연 시간 및 면적이 트랜지스터 소자들에 비해 훨씬 크므로, 버스 상의 데이터 전송 길이와 버스 수를 최소화하는 것은 칩 면적 절약과 속도 개선에 매우 중요하다. 뿐만 아니라 버스의 분할을 통해 데이터 전송 길이를 최소화하는 것은 전력 소모를 줄이고 증대된 하드웨어 병렬성을 활용하는 추가적인 이점도 제공한다. 분할 버스 아키텍처에서 상기의 합성 목표들을 달성하기 위해서는 스케쥴링과 할당 및 바인딩 뿐만 아니라 노드들의 세그먼트별 분할과 세그먼트의 순서를 정하는 문제들도 동시에 고려되어야 한다. 이러한 추가적인 합성 목표들로 인해 기존의 상위 수준 합성 문제의 복잡도는 더욱 증가하게 된다. 복잡도의 증가에 대처하며 우수한 합성 결과를 얻기 위해서, 본 합성 방법에서는 아래와 같은 두 가지 아이디어를 사용한다.

(i) 타겟 아키텍처의 변경: 버스 요구를 분산함으로써 결과적으로 요구되는 버스의 수를 줄일 수 있도록 타겟 아키텍처를 확장한다.

(ii) 유전자 알고리즘의 채택: 유전자 알고리즘을 중심적인 설계 공간 탐색 방법으로 채택한다. 합성 과정을 각 단계별로 나누어 순차적으로 실행하는 대신 분할, 세그먼트의 순서 정의(ordering), 스케쥴링, 기능 유닛, 레지스터, 버스의 할당 및 바인딩의 모든 단계를 유전자 알고리즘에 통합하여 동시에 고려되도록 한다. 실험 결과는 본 방법이 분할 버스 아키텍처를 위한 효과적인 상위 수준 합성 방법이 될 수 있음을 보여준다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. II절에서 본 논문의 연구와 관련이 있는 기존의 연구에 대해 살펴본 다음, III절과 IV절에서는 타겟 아키텍처의 변경과 합성 전략 및 알고리즘에 관해 자세히 기술한다. V절에서는 본 논문에서 제시된 합성 방법을 두 개의 벤치마크 예제에 대해 실험한 결과를 소개하고 VI절에서 결론과 아울러 추후 연구과제에 대하여 언급한다.

II. 기존 연구

본 논문에서 제시하는 합성 방법과 관련된 기존 연구는 분할 버스 아키텍처의 상위 수준 합성, 유전자 알고리즘을 이용한 상위 수준 합성, 상위 수준 합성에서 스케쥴링과 할당의 결합등, 세가지 방향에서 살펴볼 수 있다.

(1) 분할 버스 아키텍처의 상위 수준 합성

최근, 분할 버스 아키텍처의 상위 수준 합성에 관한 몇 가지 연구가 수행되었다. Ewering [2]은 데이터 흐름 그래프를 스케쥴한 다음 스케쥴된 노드들을 세그먼트 별로 분할한다. 그 다음 세그먼트들의 순서를 정하고 버스 세그먼트를 할당한다. 반면 Moshnyaga[3] 등은 분할과 순서 결정 과정을 차례로 거친 후 버스에 의해 구동되는 스케쥴링과 레지스터 할당을 수행한다. Frank 등[13]은 스케쥴링과 모듈 할당이 끝난 후 메모리와 상호 연결 할당이 수행되기 전에 배치 과정을 수행한다. 배치 과정을 상위 수준 합성에 결합함으로써 면적과 네트의 길이의 예측이 조기에 가능해 진다. Duncan 등[14]은 분할 버스 아키텍처와 약간 다른 칼럼 기반(column-based) 타겟 아키텍처와 광역 배선이 필요 없는 셀 내부 배선(in-the-cell routing)을 이용하여 면적이 작은 도면을 생성한다. 탐욕적인 스케쥴링(greedy scheduling)을 이용하여 스케쥴된 데이터 흐름 그래프는 세가지 순차적 과정 - 할당/바인딩, 레지

스터/통신 할당, 칼럼 순서 결정 - 을 통해 타겟 아키텍처에 매핑된다. 이들 연구들은 합성의 각 과정의 순서는 다르지만 모든 과정들을 순차적으로 실행한다는 점에서는 동일하다. 분할 버스 아키텍처와 같이 제한된 구조를 위한 데이터 경로의 합성을 순차적인 과정으로 수행하는 것은 보다 큰 설계 공간을 탐색하여 보다 최적의 해를 얻을 가능성을 배제하게 되는 문제점이 있다.

(2) 유전자 알고리즘을 이용한 상위 수준 합성

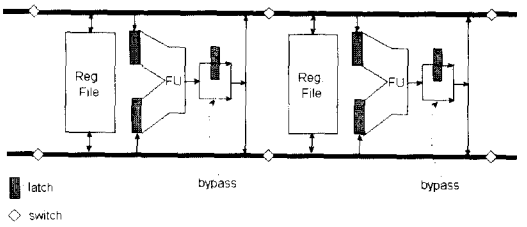
Dhodhi 등[8]은 데이터 경로 합성을 위한 문제 공간 유전자 알고리즘 (problem-space genetic algorithm : PSGA)을 제안하였다. 하드웨어 자원과 실행 시간에 대한 비용 함수를 최소화하기 위해 스케줄링과 할당을 동시에 실행한다. PSGA를 이용함으로써 빠른 합성 시간과 비교적 큰 문제를 다룰 수 있다는 장점이 있지만 멀티플렉서 지향 타겟 아키텍처를 가정하고 있으며 상호 연결의 최적화 문제를 명시적으로 다루지 않았다. Heijligers 등[19]의 유전자 알고리즘에서는 스케줄링 동안 보충적인 자원을 할당할 수 있는 인코딩 기법을 사용한다. 이들 역시 주의깊게 설계된 분석 방법에 의해 합성 시간을 빠르게 했지만 상호 연결의 최적화는 고려하지 않았다.

(3) 상위 수준 합성에서 스케줄링과 분할의 결합

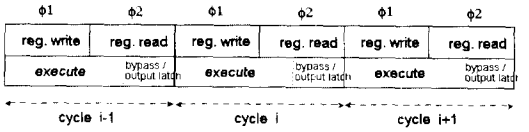
Devadas 등[15]은 스케줄링과 할당 문제를 마이크로 인스트럭션을 시공간상에 배치하는 2차원 배치 문제로 구성하여 시뮬레이티드 어닐링을 이용한 알고리즘으로 해를 구하였다. 레지스터, 기능 유닛, 상호 연결의 할당과 스케줄링을 동시에 풀 수 있지만 실행 시간이 많이 걸리는 단점이 있다. Cloutier 등 [16]은 스케줄링, 할당, 매핑을 반복적인 포스 구동 (force-directed) 스케줄링 알고리즘에 결합하였다. 포스 구동 스케줄링이 가지는 거시적인 안목과 아울러 포스를 이용하여 스케줄링과 매핑의 결과를 예측한다. 이러한 포스는 자원과 상호 연결의 활용도 측면에서 우수한 해를 찾도록 스케줄링을 인도한다. 그들은 멀티플렉서 지향 아키텍처를 가정하였다. Sharma 등 [20]은 스케줄링과 기능 유닛, 저장 유닛, 연결 유닛들의 할당을 하나의 단계로 결합하였다. 데이터 값의 수명(lifetime)이 다 결정되지 않은 부분적인 스케줄에서 레지스터 최적화를 위해 레지스터 상태(자유, 사용 중, 미정)의 개념을 이용한다. 재사용 가능한 데이터 값과 방송(broadcast)을 사용하거나 비임계 경로(non-critical path)상의 연산 노드를 선택적으로 지연시킴으로써 버스 충돌을 완화한다. 자원 할당, 설계 공간 전지(design space pruning)와 합성 결과의 평가를 위해 예측 도구를 사용한다.

Ⅲ. 타겟 아키텍처

본 절에서는 분할 버스 아키텍처의 구조와 동작을 소개한 후 이 아키텍처를 변경하게 된 동기와 내용을 설명한다.



(a)



(b)

그림 1. 분할 버스 아키텍처: (a) 데이터 경로 세그먼트들, (b) 클럭 사이클 내의 각 단계별 마이크로연산을 나타내는 타이밍 다이어그램

Fig. 1. Partitioned bus architecture: (a) data path segments, (b) timing diagram showing micro-operations of each phase within a cycle.

(1) 분할 버스 아키텍처

본 연구에서는 그림 1과 같이 분할 버스 아키텍처를 타겟 아키텍처로 사용하며 2단계 클럭킹(two-phase clocking)을 가정한다. 분할 버스 아키텍처는 버스 지향 아키텍처의 특별한 경우로, 버스는 세그먼트로 분할되어 있고 기능 유닛들과 저장 유닛들은 일렬로 배치되며 버스 세그먼트가 이들 유닛의 양쪽 옆이나 유닛 위에 위치한다 [2]. 세그먼트들 간의 데이터 전송은 버스 세그먼트들간의 스위치에 의해 제어된다. 따라서, 상호 연결 할당은 버스 세그먼트들 간의 스위치의 제어를 통해 데이터 전송을 허용하거나 금지하는 것을

통해 달성된다. 이렇게 선형으로 분할된 연결 형태에는 장점과 아울러 단점이 있다. 장점으로는 비트 슬라이스 도면 스타일에 적합한 선형 및 규칙적인 구조로 인한 면적의 절감과 연결이 끊어진 버스 세그먼트들 사이에 내재하는 병렬성으로 인한 높은 버스 활용도를 들 수 있다. 반면 한가지 단점으로는 각각의 버스 세그먼트는 인접한 세그먼트들과만 직접 통신할 수 있기 때문에 통신이 상당히 제약된다는 점이다.

클럭 주기의 각 단계별 마이크로 연산(micro-operation)은 다음과 같다[1,21]. (그림 1):

- 레지스터 쓰기(register write): 출력을 레지스터 파일로 쓴다. 출력은 이전 사이클에서 실행된 연산의 결과로 출력 래치에 저장되어 있는 값이다.

- 레지스터 읽기(register read): 입력 피연산자를 레지스터 파일에서 입력 래치로 읽어 들인다. 입력 피연산자는 다음 사이클에서 실행될 연산에 사용된다.

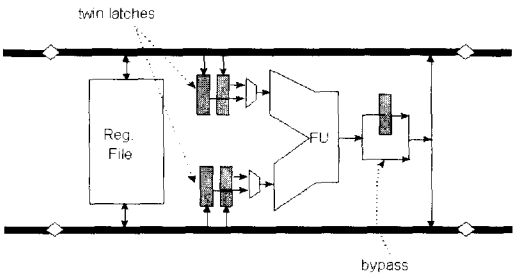
- 실행(execute): 해당되는 세그먼트의 기능 유닛을 이용하여 연산을 실행한다.

- 래치로 출력(output latch): 현재 사이클의 실행의 결과를 출력 래치로 저장한다.

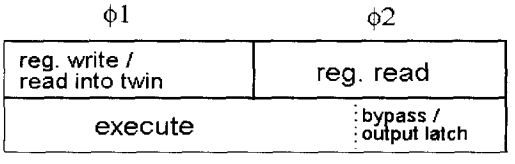
- 바이패스(bypass): 현재 사이클의 연산의 실행 결과를 입력 래치로 곧바로 전달한다. 이 마이크로 연산은 연속되는 사이클에 실행되어야 할 연산 노드들간에 데이터 의존성(data dependency)이 존재할 때 필요하다. 즉 어떤 사이클에서의 연산 결과를 바로 다음 사이클에서 피연산자로 사용할 때, 연산 결과를 레지스터 파일에 써두었다가 읽어들여야할 필요없이 곧 바로 해당 기능 유닛의 입력 래치로 전달(forwarding)함으로써 연속되는 사이클에서 데이터 의존성(data dependency)이 존재하는 연산 노드들의 실행이 가능하도록 한다. 바이패스가 많을수록 레지스터 읽기 및 쓰기가 줄어들기 때문에 결과적으로 레지스터가 줄어들게 된다. 이들 마이크로 연산들 중에서 버스 세그먼트가 필요한 데이터 전송 연산은 레지스터 쓰기, 레지스터 읽기, 바이패스 등이다.

(2) 변경된 분할 버스 아키텍처

본 연구에서는 그림 2와 같이 타겟 아키텍처를 변경하였다. 표준적인 분할 버스 아키텍처와 클럭킹과의 가장 큰 차이는 트윈 래치(twin latch)를 사용한다는 점이다. 트윈 래치는 원래의 분할 버스 아키텍처의 입력 래치를 보조하는 역할을 한다.



(a)



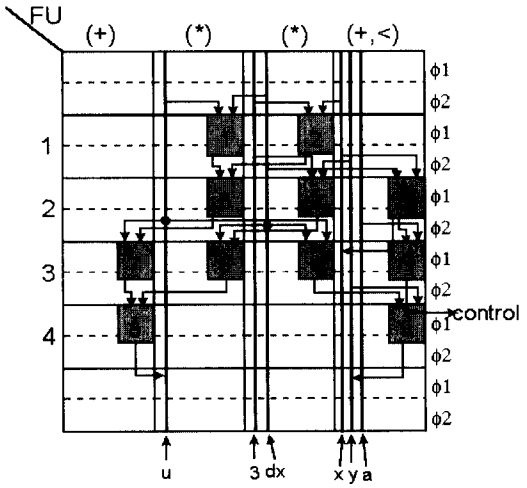
(b)

그림 2. 변경된 분할 버스 아키텍처: (a) 데이터 경로 세그먼트, (b) 타이밍 다이어그램

Fig. 2. Modified partitioned bus architecture: (a) one data path segment, (b) timing diagram.

트윈 래치는 클럭 사이클 내의 1단계(φ1)에서만 피연산자를 읽어 들일 수 있다는 점에서 정상적인 입력 래치와 구별된다. 트윈 래치의 사용은 많은 경우에, 클럭 사이클 내의 2단계(φ2)에서 레지스터로부터 피연산자를 읽거나 실행 결과의 바이패스를 위한 데이터 전송 횟수가 1단계(φ1)에서 출력 결과를 레지스터에 쓰기 위한 데이터 전송 횟수보다 훨씬 많다는 관찰에 기초한다. 이러한 각 단계에서의 데이터 전송 횟수의 불균형은 비효율적인 버스 활용의 원인이 되어 결과적으로는 요구되는 버스의 수를 증가시킨다. 트윈 래치를 사용하면 원래 피연산자를 읽어야 할 사이클 또는 그 이전 사이클의 1단계(φ1)에서도 피연산자를 읽어 들일 수 있다. 단, 이 경우 동일한 단계에서 동일한 레지스터 파일에 데이터를 쓰거나 트윈 래치로 데이터를 가져오는 것을 인해 데이터 해저드(data hazard)[10]가 발생해서는 안된다. 이러한 기법을 이용하면 모든 사이클의 단계에 대해 데이터 전송이 고르게 분포되게 함으로써 버스의 활용도를 높이고 결과적으로 요구되는 버스의 수를 줄이게 된다. 레지스터 읽기/쓰기, 바이패스와 같은 마이크로 연산과 마찬가지로 트윈 래치로 읽기(read into twin) 연산에도 버스 세그먼트가 필요하다.

변경된 아키텍처의 대략적인 클럭 주기는 $\max(t_{op}, (t_r + \max(tt_{win}, t_w))$으로 원래 아키텍처의 주기인 $\max(t_{op}, (t_r + t_w))$에 근접하는 크기인데, 이는 tt_{win}이 t_w과 거의 비슷한 크기를 가지기 때문이다. t_{op}, t_r, t_w, tt_{win}은 4가지 마이크로 연산 - 실행, 레지스터 읽기, 레지스터 쓰기, 트윈 래치 읽기 - 에 걸리는 시간을 나타낸다.



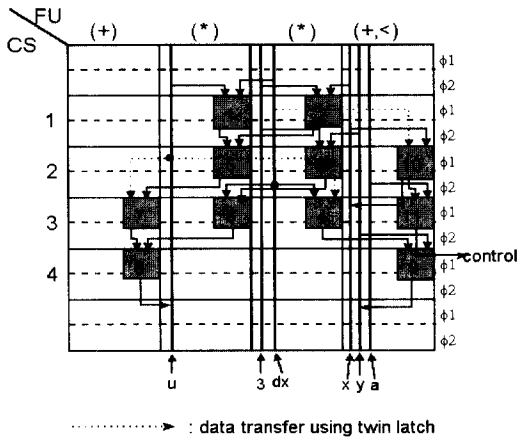
(a)

CS	FU (+)	(*)	(*)	(+,<)	
					φ1
	0	2	3	1	φ2
1	0	0	0	0	φ1
	0	2	4	3	φ2
2	0	0	0	0	φ1
	2	4	3	2	φ2
3	0	0	0	1	φ1
	2	1	1	2	φ2
4	0	0	0	1	φ1
	0	0	0	0	φ2
	1	1	0	1	φ1
					φ2

(b)

그림 3. 트윈 래치 읽기 마이크로 연산을 사용하지 않고 버스를 데이터 전송에 할당/바인딩한 결과: (a) 버스 바인딩, (b) 각 세그먼트별로 요구되는 버스의 개수

Fig. 3. Allocation/binding of bus to data transfers without twin latch read operation: (a) binding of bus, (b) the number of buses required for each segment.



(a)

CS	FU (+)	(*)	(*)	(+,<)	
					φ1
	0	2	3	1	φ2
1	0	0	1	1	φ1
	0	2	3	2	φ2
2	1	1	1	0	φ1
	1	3	2	2	φ2
3	0	0	0	1	φ1
	2	1	1	2	φ2
4	0	0	0	1	φ1
	0	0	0	0	φ2
	1	1	0	1	φ1
					φ2

(b)

그림 4. 트윈 래치 읽기 마이크로 연산을 사용하여 버스를 데이터 전송에 할당/바인딩한 결과: (a) 버스 바인딩, (b) 각 세그먼트별로 요구되는 버스의 개수

Fig. 4. Allocation/binding of bus to data transfers with twin latch read operation: (a) binding of bus, (b) the number of buses required for each segment.

그림 3과 4는 트윈 래치의 유용성을 보여준다. 이들 그림에서 합성의 결과는 시간 (제어 단계)와 공간(세그먼트) 축을 따라 2차원 다이어그램으로 표현한다. 여기에서 사용된 예는 본 논문에서 사용된 벤치마크 예제의 하나인 미분방정식(differential equation) [5]이다. 트윈 래치를 사용하지 않고 각각의 데이터 전송에 대한 버스의 할당과 바인딩을 한 결과 요구되는 버스 수 - 하나의 세그먼트에서 병렬로 일어나는 데이터 전송 수의 최대값 - 는 그림 3과 같이 4인 반면, 트윈 래치

를 사용한 경우에는 그림 4와 같이 3으로 줄어든다.

이러한 분할 버스 아키텍처의 변경은 Aloqeely 등[11]이 사용한 시퀀서에 기초한(sequencer-based) 데이터 경로와 다소 비슷한 점이 있으나 아키텍처 변경의 목적과 기본 원리에 있어서는 근본적으로 다르다.

Ⅳ. 합성 전략 및 알고리즘

(1) 합성 목표와 전략

본 연구에서 구현한 상위 수준 합성기는 제어-데이터 흐름 그래프(control-data flow graph: CDFG)로부터 분할 버스 데이터 경로의 레지스터 전송 수준의 구조와 제어 유닛을 위한 마이크로 연산의 순서를 생성한다. 현재는 자원에 대한 제한이 유일한 제약 조건(constraint)으로 기능 유닛의 수와 타입으로 주어진다. 본 상위 수준 합성기의 합성 목표는 자원 제약 조건을 만족하면서 다음의 변수 값들을 최소화하는데 있다. 이들 변수들은 합성 결과의 품질을 평가하는데 사용된다.

- 버스 전송 길이의 최대값. 즉, 데이터 전송에 사용되는 버스 세그먼트 수의 최대 값.
- 요구되는 버스의 수. 즉, 세그먼트별로 병렬로 일어나는 데이터 전송 수의 최대값.
- 제어 단계의 수
- 레지스터 개수

위의 합성 목표들을 달성하기 위해서는 스케줄링과 할당 및 바인딩 뿐만 아니라 노드들의 세그먼트별 분할과 세그먼트의 순서 정의 문제들도 동시에 고려되어야 한다. 이러한 추가적인 합성 목표들로 인해 기존의 상위 수준 합성 문제의 복잡도는 더욱 증가하게 된다. 본 논문에서는 유전자 알고리즘을 설계 공간의 탐색을 위한 중심적인 방법으로 사용하며 다음의 전략들을 유전자 알고리즘에 결합하였다.

- 버스 전송 길이와 버스 개수의 최소화를 위해서는 분할된 세그먼트의 순서가 매우 중요하다. 본 연구에서는 세그먼트의 순서를 별도의 염색체로 인코딩하여 유전자 알고리즘에서 사용한다.
- 버스 개수의 추가적인 감소를 위해 트윈 래치를 적극적으로 사용한다.
- 제어 단계의 수를 최소화하기 위해 초기 개체군(initial population)의 생성시 리스트 스케줄링(list scheduling)을 사용한다.
- 레지스터 수를 최소화하기 위해 기존의 레지스터 최소화 방법 외에 가능하면 바이패스를 많이 사용한다.

```
Main_Genetic() {
    /* Generate an initial population */
    Do list scheduling;
    For each individual to be generated in the
    initial population :
        Initialize the list of available FUs for
        every c-steps;
        Place all nodes in a queue in random
        order;
        For each node :
            Pop one node at a time from the head
            of the queue;
            Randomly select a FU from available
            list of FUs;
            Update FU list;
        end for;
    end for;

    /* Main loop of genetic algorithm */
    For each generation :
        For each individual :
            Bind_and_Optimize_Bus_and_
            Register();
            Calculate cost and fitness;
        end for;
        Save the best individual;
        Create next generation by genetic ope-
        rations (select, crossover, mutation);
    end for;

    Postprocess the best individual and
        output RT-level result and micro-
        operation;
}

Bind_and_Optimize_Bus_and_Register() {
    Determine the segment position of each va-
    riable;
    Assign bus segment(s) to each data transfers
    (read, write, bypass);
    Optimize bus number using twin latches;
    Bind variables to registers and optimize the
    number of registers;
}
```

그림 5. 합성 알고리즘의 개요
Fig. 5. Overview of synthesis algorithm.

(2) 합성 알고리즘

본 합성 알고리즘의 세부적인 구현은 다음과 같다.

1) 알고리즘 개요: 본 합성 알고리즘의 개요는 그림 5에 나타나있다. 합성 알고리즘의 중추는 유전자 알고리즘이다. 유전자 알고리즘은 VLSI의 합성 문제들[18,19,22]에도 응용된 강력한 최적화 알고리즘이다. 유전자 알고리즘은 초기 개체군이라 불리는 초기의 무작위적인 구성에서 시작하여 이들의 진보를 위해 생물적인 진화와 유사한 과정을 사용한다. 유전자 알고리즘이 적용되는 구성원들의 집합을 개체군이라 부른다. 개체군의 각각의 개체의 특성은 염색체에 들어있는데 염색체는 곧 최적화 문제에 대한 해를 나타내는 심볼 스트링으로 표현된다. 세대(generation)라 불리는 각각의 반복 단

계에서 현재의 개체군 내의 개체들에 대해 휴리스틱 함수를 적용하고 정해진 척도를 사용하여 적합도(fitness)를 평가한다. 이러한 적합도에 근거하여 한 번에 두 개씩 어버이(parent)를 선택한다. 이때, 루울렛 휠 선택(roulette wheel selection)과 같은 선택 방법을 사용함으로써 적합도가 높은 개체일수록 선택될 확률이 높아진다. 선택된 어버이들에 대해 유전자 연산을 가하여 두 어버이의 특징을 결합하여 후손(offspring)이라 불리는 새로운 개체를 생성한다. 유전자 알고리즘에서 사용하는 가장 보편적인 유전자 연산으로는 생물적인 진화 과정에 대한 유추에 의해 유도된 교차(crossover), 돌연변이(mutation), 역위(inversion) 연산 등이 있다.

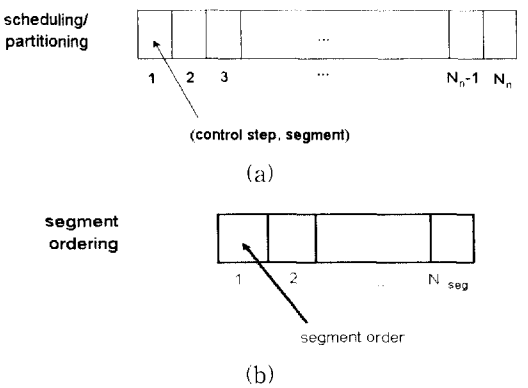


그림 6. 각 개체의 염색체: (a) 스케줄링/분할 염색체, (b) 세그먼트 순서 염색체

Fig. 6. Chromosomes for each individual: (a) chromosome for scheduling/partitioning, (b) chromosome for segment ordering.

2) 염색체 코딩, 초기 개체군의 생성, 유전자 연산: 그림 6은 각 개체의 염색체를 나타낸다. 각 개체는 스케줄링/분할(여기서 분할은 노드들을 세그먼트로 할당하는 것을 뜻함)을 표현하는 염색체(이후 SP염색체라 부름)와 세그먼트들의 순서를 표현하는 염색체(이후 SO염색체라 부름)의 두 가지 염색체를 가지고 있다. SP염색체는 그 길이(즉 염색체 내의 유전자의 수)가 CDFG의 노드 수와 같고 염색체 내의 각각의 유전자는 2-튜플(2-tuple)로 각 노드의 제어 단계와 이 노드의 실행을 위한 기능 유닛이 들어 있는 세그먼트를 나타낸다. 그리고 SO염색체는 그 길이가 세그먼트의 수(즉 자원 제약 조건으로 주어진 기능 유닛의 개수)와 같고 염색체 내의 각각의 유전자는 각 세그먼트의 순서를 나타낸다. 따라서, 각각의 개체는 분할, 스케줄링, 할당/바인딩, 순서 결정 등이 결합된 복합적인 문제의 해를 나타낸다.

초기 개체군은 최소 개수의 제어 단계와 추후의 유리한 설계 공간 탐색을 위해 리스트 스케줄링 [8]과 각 노드의 연산을 수행할 기능 유닛의 무작위 선택을 통해 생성된다.

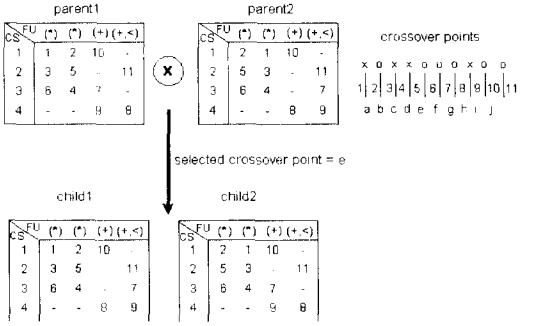


그림 7. 스케줄링/분할 염색체에 대한 교차 연산: 'o'은 가능한 교차 점을 나타냄

Fig. 7. Crossover operation for scheduling/partitioning chromosomes: 'o' represents a possible crossover point.

본 연구에서는 각각의 유전자 연산 후에 많은 시간이 소요되는 염색체 교정(chromosome repair) 과정을 거치지 않도록 유전자 연산을 구성하였다. 예를 들면, SP염색체의 교차 연산 시에는 선택된 두 개의 어버이에 대해 교차 가능한 교차 점(crossover point)들을 먼저 구한 다음 이들 중에서만 교차 점을 무작위 선택함으로써 교차로 생성되는 후손의 염색체가 하드웨어 자원 제약 조건과 노드들간의 데이터 의존성을 만족하는 실현 가능한 해(feasible solution)를 보장하도록 한다. 그림 7은 이 과정을 보여 준다. SO염색체에 대해서는 순서 교차(order crossover) [4,22]를 행한다. SP염색체에 대한 돌연변이 연산은 임계 경로상의 노드들에 대해서는 다른 기능 유닛을 할당하고 비임계 경로상의 노드들에 대해서는 다른 제어 단계와 기능 유닛이 할당되도록 함으로써 설계 공간의 다른 영역을 탐색할 가능성을 제공한다. SO염색체에 대한 돌연변이 연산은 현재로서는 구현되지 않았는데 이는 SO염색체의 길이가 짧기 때문에 돌연변이 연산으로 인한 효과를 크게 기대할 수 없다고 판단했기 때문이다. 그리고

역위 연산도 현재로서는 지원되지 않는다.

3) 버스/레지스터 최적화: 본 유전자 알고리즘은 버스 및 레지스터 최적화를 위하여 다음과 같은 과정을 수행한다.

- 레지스터에 관련된 데이터 전송의 길이가 최소화되도록 각 레지스터가 속할 세그먼트를 결정한다.
- 각 데이터 전송을 위해 버스 세그먼트를 할당한다.
- 트윈 래치를 최대한 활용하여 버스의 수를 최적화한다.
- 레지스터 최적화 알고리즘 [9]을 이용하여 레지스터 개수를 최적화한다.

이들 과정은 그림 5의 합성 알고리즘 개요에서 Bind_and_Optimize_Bus_and_Register() 함수로 구현된다.

4) 비용 및 적합성 평가: 각 개체의 비용 함수는 다음 식과 같이 파라미터 값들에 가중치를 곱한 것의 합으로 주어진다.

$$C = \alpha L_{max} + \beta N_b + \gamma N_r \qquad (1)$$

여기서 L_{max}, N_b, N_r은 데이터 전송 길이의 최대값, 요구 버스의 수, 레지스터의 개수를 각각 나타내며 α, β, γ는 각각의 가중치를 나타낸다. 이 식에서 제어 단계의 수는 포함하지 않았는데 이는 초기 개체군의 생성시에 리스트 스케쥴링을 이용하여 정한 제어 단계의 개수를 항상 유지하기 때문이다. 적합도 함수는 각 세대의 각각의 개체에 대해 다음 식과 같이 계산된다.

$$f_i = (C_{\max} - C_i)^{\pi} / \sum_{j=1}^{j=N} (C_{\max} - C_j)^{\pi} \qquad (2)$$

표 1. 벤치마크 예제의 데이터
Table 1. Data of benchmarks.

benchmarks	data		
Elliptic Wave Filter	# nodes	* / + / - / <	6 / 2 / 2 / 1
		total	11
	# edges	constant / variable	7 / 18
		total	25
Differential Equation	# nodes	* / +	8 / 26
		total	34
	# edges	constant / variable	8 / 68
		total	76

f_i, C_{max}, C_i, π, N은 i번째 개체의 적합도, 현재 세대의 최대 비용, i번째 개체의 비용, 다양성 인수(variety factor), 개체군의 크기를 각각 나타낸다.

표 2. 합성 결과와 비교
Table 2. Synthesis results and comparison.

benchmark	resource constraints		InSyn [20]	ALPS [7]	PARBUS [2]	Moshnyaga [3]	Ours
Elliptic Wave Filter	1*(p), 2+	cs	23	19	19	19	19
		bus/seg/len	3/-/-	5/-/-	2/6/-	2/6/-	2/6/3
		reg	8	19	12	11	10
	1*, 2+	cs	22	21	-	21	21
		bus/seg/len	4/-/-	6/-/-	-	2/6/-	2/6/3
		reg	8	19	-	10	10
	2*, 2+	cs	18	18	-	18	18
		bus/seg/len	6/-/-	5/-/-	-	2/8/-	2/8/3
		reg	8	15	-	11	10
	3*, 3+	cs	18	17	-	17	17
		bus/seg/len	6/-/-	6/-/-	-	2/12/-	2/12/3
		reg	8	16	-	11	11
Differential Equation	1*(p), 1A	cs	-	-	4	4	4
		bus/seg/len	-/-/-	-	2/4/2	2/4/2	2/4/2
		reg	-	-	5	5	5
	2*(s), 1+, 1A	cs	-	-	-	4	4
		bus/seg/len	-	-	-	2/8/4	2/8/2
		reg	-	-	-	8	6

V. 실험 결과

제안된 합성 방법을 Sun Sparc 20에서 C 언어로 구현하였다. 벤치마크 데이터로 미분방정식 [5]과 5차 디지털 엘립틱 필터 [6]를 이용하여 합성한 결과를 기존 연구에서의 합성 결과와 비교하였다. 표 I은 이들 벤치마크들의 합성 관련 데이터를 나타내었다. 표 II는 다양한 하드웨어 자원 제약 조건에 대해 합성한 결과를 보여 주고 있다. 덧셈기(+), ALU(A), 단일 사이클 곱셈기(*(s)), 복수 사이클 곱셈기(*), 파이프라인된 곱셈기(*(p))의 지연 시간은 각각 1, 1, 1, 2, 2 사이클이다. 이 표에서 cs, bus, seg, reg, len은 제어 단계의 수, 버스의 개수, 버스 세그먼트의 개수, 레지스터 개수, 세그먼트의 개수로 표현된 데이터 전송의 최대 길이를 각각 나타낸다. 비교에 사용된 기존 방식 중 InSyn [20]와 ALPS [7]은 일반적인 버스 아키텍처를 사용하였으며, PARBUS [2]와 Moshnyaga [3]은 본 연구와 같이 분할 버스 아키텍처를 사용하였다. 이 표에서 보는 바와 같이 본 논문에서 제안한 합성 방법을 이용하여 버스의 수와 데이터 전송 길이의 측면에서 우수한 결과를 얻을 수 있었다. 특히 동일한 아키텍처를 사

용하는 PARBUS와 Moshnyaga의 결과에 비해 버스의 수는 같으나 데이터 전송 길이가 짧아짐을 볼 수 있다. 이는 이들의 연구에서는 각 합성 단계를 순차적으로 실행한 반면, 본 연구에서는 유전자 알고리즘을 이용하여 각 합성 단계를 통합하여 각 합성 단계의 목표들을 동시에 고려한 결과이다. 현재의 합성기에서는 실제 레이아웃 상에서의 정확한 버스 길이를 고려하는 예측 기능을 아직 구현하지 않았기 때문에 대략적인 예측이긴 하지만 버스 전송 길이를 버스 세그먼트의 개수로만 나타내었다. 레지스터 개수가 작은 것은 레지스터 엑세스를 필요로 하지 않은 바이패스 데이터 전송을 적극적으로 사용한 결과이다.

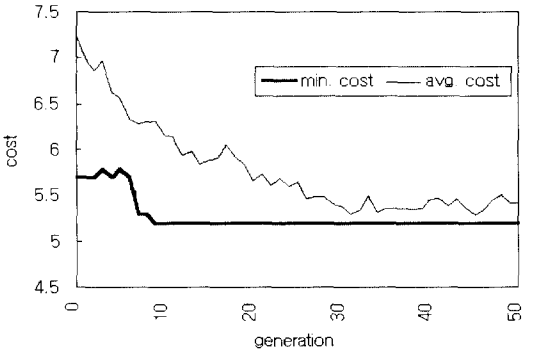


그림 8. 합성 알고리즘의 수렴
Fig. 8. Convergence of our synthesis algorithm.

각 합성에 소요된 시간은 집단의 크기 (30 - 100 개체), 세대 수 (30 - 100 세대), CDFG의 노드와 에지의 개수에 따라 달라지며 4 내지 40초의 CPU 시간이 소요되었다. 이에 비해 PARBUS와 Moshnyaga의 경우에는 Sun 3에서 5초 미만의 CPU 시간이 소요되었다. 실험 결과 집단의 크기는 30 - 100, 세대 수는 30 - 100, 교배 확률(P_{xover})은 0.6 - 0.8, 돌연변이 확률(P_m)은 0.005 - 0.02일 때 좋은 결과를 얻을 수 있었다. 그림 8은 본 논문의 합성 방법에서 유전자 알고리즘을 중추적인 설계 공간 탐색 방법으로 사용할 경우 잘 수렴함을 보여주고 있다.

VI. 결론 및 추후 과제

본 논문에서는 분할 버스 아키텍처를 위한 유전자 알고리즘에 바탕을 둔 상위 수준 합성 방법을 제안하였다. 실험 결과는 이 방법이 버스 연결로 인한 배선 지연 시간이 전체 지연 시간에서 차지하는 비율이 큰 딥 서브 마이크론 기술을 이용한 회로 설계에 적합한 분할 버스 아키텍처를 위한 효과적인 상위 수준 합성 방법이 될 수 있음을 보였다.

추후의 연구과제는 다음과 같다.

(i) 보다 정확한 결과를 얻기 위해 본 상위 수준 합성 방법을 레이아웃 자동 생성 과정과 연결시킴.

(ii) 본 방법을 확장하여 클럭 주기 선택, 저전력 합성, 조건 분기와 루우프 등과 같은 제어 구문 처리 기능 등을 수행할 수 있도록 함.

참 고 문 헌

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

[2] Ewering, "Automatic high level synthesis of partitioned busses", *Proc. ICCAD-90*, pp. 304-307, Nov. 1990.

[3] V. G. Moshnyaga, F. Ohbayashi, and K. Tamaru, "A scheduling algorithm for synthesis of bus-partitioned architectures", *Proc. ASP-DAC95*, pp. 43-48, Aug. 1995.

[4] Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, chapter 1-3, Reading, MA, Addison-Wesley, 1989.

[5] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs", *IEEE Trans. on CAD*, vol. 8, no. 6, pp. 661-679, June 1989.

[6] S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*, Prentice Hall, pp. 256-264, 1985.

[7] J.-H. Lee, Y.-C. Hsu, and Y.-L. Lin, "A new integer linear programming formulation for the scheduling problem in data path synthesis", *Proc. ICCAD-89*, pp. 20-23, Nov. 1989.

[8] P. Gutberlet, H. Kramer, and W. Rosenstiel, "CASCH - a scheduling algorithm for high level synthesis", *Proc. EDAC*, pp. 311-315, Feb. 1991.

[9] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and

Y.-C. Hsu, "Data path allocation based on bipartite weighted matching", *Proc. 27th DAC*, pp. 499-504, June 1990.

[10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, pp. 260-268, Morgan Kaufmann Publishers, 1990.

[11] M. Aloqeely and C. Y. R. Chen, "Sequencer-based data path synthesis of regular iterative algorithms", *Proc. 31th DAC*, pp. 155-160, June 1994.

[12] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems", *Proceedings of IEEE*, vol. 78, no. 2, Feb. 1990.

[13] E. Frank and T. Lengauer, "Applause: area and performance optimization in a unified placement and synthesis environment", *Proc. ICCAD-95*, pp. 662-667, Nov. 1995.

[14] A. A. Duncan and D. C. Hendry, "DSP data path synthesis eliminating global interconnect", Proc. *Euro-DAC93*, pp. 46-51, 1993.

[15] S. Devadas and A. R. Newton, "Algorithm for hardware allocation in data path synthesis", *IEEE Trans. on CAD*, vol. 8, no. 7, pp. 768-781, July 1989.

[16] R. J. Cloutier and D. E. Thomas, "The combination of scheduling, allocation, and mapping in a single algorithm", *Proc. 27th DAC*, pp. 71-76, June 1990.

[17] B. S. Haroun and M. I. Elmasry, "Architectural synthesis for DSP silicon compilers", *IEEE Trans. on CAD*, vol. 8, no. 4, pp. 431-447, April 1989.

[18] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker, "Data path synthesis using a problem-space genetic algorithm", *IEEE Trans. on CAD*, vol. 14, no. 8, pp. 934-944, Aug. 1995.

[19] M. J. M. Heijligers, L. J. M. Cluitmans, and J. A. G. Jess, "High-level synthesis scheduling and allocation using genetic algorithm", *Proc. ASP-DAC95*, pp. 61-66, Aug. 1995.

[20] A. Sharma and R. Jain, "InSyn: integrated scheduling for DSP applications", *Proc. 30th DAC*, pp. 349-354, June 1993.

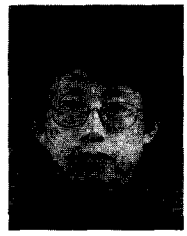
[21] Y.-C. Hsu and Y.-L. Lin, High level synthesis in the THEDA system, as Chap. 12 in *High-Level VLSI Synthesis* (ed. R. Camposano and W. Wolf), pp. 283-306, Kluwer Academic Publishers, 1991.

[22] K. Shahookar and P. Mazumder, "A genetic approach to standard cell placement using meta-genetic parameter optimization", *IEEE Trans. on CAD*, vol. 9, no. 5, pp. 500-511, May 1990.

저 자 소 개

金 容 主(正會員)

1957년 6월 6일생. 1979년 서강대학교 전자공학과 졸업. 1982년 서울대학교 전자공학과 석사. 1994년 서울대학교 전자공학과 박사과정 수료. 1983년 ~ 1997년 현재 한국전자통신연구원 근무. 주관심분야는 High-level synthesis, Codesign, System synthesis, VLSI 설계 등임.



崔 起 榮(正會員)

1955년 8월 30일생. 1978년 서울대학교 전자공학과 졸업. 1980년 한국과학원 전기 및 전자공학과 석사. 1989년 미국 Stanford대학 전기공학과 박사. 1978년 ~ 1983년 (주)금성사 중앙연구소 근무. 1989년 ~ 1991년 미국 Cadence Design System, Inc. 근무. 1991년 ~ 1995년 서울대학교 반도체공동연구소 및 전자공학과 조교수. 1995년 ~ 현재 서울대학교 반도체공동연구소 및 전기공학부 부교수. 주관심분야는 CAD, VLSI 설계 등임.