시분할 FPGA 합성에서 LUT 개수에 대한 하한 추정 기법

(A Lower Bound Estimation on the number of LUT's in Time-Multiplexed FPGA Synthesis)

엄 성 용 †

(Seong Yong Ohm)

요 약 주어진 논리 회로를 시분할 FPGA 칩으로 효과적으로 합성하기 위해서는 전체 회로를 여러 개의 부분회로로 나눈 후, 각 부분 회로가 동일한 하드웨어 회로를 시간차를 두고 공유하도록 하여야 한다. 이를 위해 칩에 대한 시간별 재구성 정보를 미리 만들어, 칩 내부의 특정 메모리 영역에 저장하여 두었다가 정해진 시간대가 되면 칩 전체를 재구성하도록 하여야 한다. 그런데, 시분할 FPGA 합성에서 사용하는 세부적인 재구성 기법(일반적으로 스케쥴링이나 다중 회로 분할 기법을 사용)에 따라 동일 시간대에 필요한 LUT의 개수, 즉 FPGA의 용량이 달라질 수 있다. 본 논문에서는 입력되는 논리 회로를 직접 합성하지 않고서도 그 회로가 필요로 하는 전체 LUT 개수에 대한 하한을 추정함으로써 재구성 기법에 관계없이 필요한 최소한의 LUT 개수를 파악한다.

만일, 기존의 재구성 결과가 본 연구에서 추정된 하한과 일치할 경우, 그 결과는 최적의 결과를 의미한다. 반면에, 하한과의 차이가 있는 경우에는 기존의 연구 결과에 비해 더 좋은 재구성 결과가 존재하거나, 또는 본 연구에서 추정한 하한보다 더 좋은(큰, 정확한) 하한이 실제 존재함을 의미한다. 따라서 이러한 비교 분석을 통해, 기존 연구의 결과가 최적인지, 또는 개선의 여지가 있는지를 판단하는 좋은 지표를 제공할 수 있다.

실험 결과, 실험한 대부분의 예제에서, 기존의 연구 결과에서 출력한 결과와 본 논문에서 제안한 방법으로 추정한 하한이 정확히 일치하는 것을 발견할 수 있었는데, 이는 기존의 합성 시스템에서 생성한 결과의 최적성을 확인하게 하는 한편, 본 논문에서 제안한 하한 추정의 정확성을 반증하는 것으로 해석될 수 있다.

키워드 : DPGA, FPGA, 시분할 FPGA, 하드웨어 합성, 하한 추정, 스케쥴링, 분할

Abstract For a time-multiplexed FPGA, a circuit is partitioned into several subcircuits, so that they temporally share the same physical FPGA device by hardware reconfiguration. In these architectures, all the hardware reconfiguration information called contexts are generated and downloaded into the chip, and then the pre-scheduled context switches occur properly and timely. Since the maximum number of the LUT's required in the same time determines the size of the chip used in the synthesis, it needs to be minimized, if possible. Many previous work use their own approaches, which are very similar to either scheduling method in high level synthesis or multi-way circuit partitioning method, to solve the problem.

In this paper, we propose a method which estimates the lower bound on the number of LUT's without performing any actual synthesis. The estimated lower bounds help to evaluate the results of the previous work. If the estimated lower bound on the number of LUT's exactly matches the number of LUT's of the result from the previous work, the result must be optimal. In contrast, if they do not match, the following two cases are expected : the more exact lower bound may exist, or we might find the new synthesis result better than the result from the previous work.

· 이 논문은 2000년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF-2000-041-E00247)

† 종신회원 : 서울여자대학교 정보통신공학부 교수

osy@swu.ac.kr

논문접수 : 2002년 2월 26일

심사완료 : 2002년 4월 19일

Experimental results show that our lower bound estimation method is very accurate. In almost all cases experimented, the estimated lower bounds on the number of LUT's exactly match those of the previous synthesis results respectively, implying that the best results from the previous work are optimal as well as our method predicted the exact lower bound for those examples.

Key word : DPGA, FPGA, Time-Multiplexed FPGA, Hardware Synthesis, Lower Bound Estimation, Scheduling, Partitioning

1. 서 론

FPGA (Field Programmable Gate Array)는 기본적인 연산을 수행할 수 있는 논리 블록들로 구성된 논리 블록 영역과 이 셀(블록)들의 입출력 신호들을 상호 연결시키기 위한 연결 영역으로 구성되어 있다. FPGA 칩은 논리 블록을 구현하기 위한 세부 방법에 따라 MUX (Multiplexor)를 이용하는 방식과 LUT (Look-Up Table)를 이용하는 방식으로 나눌 수 있는데, 최근에는 회로 기능을 진리표를 통해 구현하는 LUT 방식이 주류를 이루고 있다.[1, 2] 일반적으로, FPGA 칩을 별도의 장비에 장착할 경우, 그 기능을 쉽게 구현, 삭제 또는 변환(즉, 프로그래밍)할 수 있어, 시제품 제작 등 하드웨어 구현이 단기간에 요구되는 분야에서 많이 활용되고 있다.

하지만, 최근 들어, 하드웨어 회로가 전원이 켜져 동작하는 상태에서 그 회로의 내부를 변경 가능하게 함으로써 구현된 하드웨어의 기능을 수시로 변경할 수 있는 '동적 재구성이 가능한 FPGA (Dynamic Reconfigurable FPGA)' 또는 간략히 'DPGA'라고 불리우는 칩이 개발되어 많은 관심을 끌게 되었다.[3-9] DPGA는 주어진 논리 회로를 하드웨어로 구현할 수 있다는 점에서는 일반 FPGA 칩과 동일하지만, 칩이 동작되는 도중에도 회로의 전체 또는 일부를 다른 내용으로 신속히 변경할 수 있는 동적인 특성이 있다. 즉, 하나의 하드웨어가 시간대별로 다른 기능을 수행하도록 할 수 있다는 것이다. 이는 하드웨어는 고정된 작업을 계속 반복한다는 일반적인 관념을 벗어나는 것으로 하나의 하드웨어가 시간대 또는 주위 상황에 따라 다른 기능을 수행할 수 있음을 말하며, 또한 동일 기능의 하드웨어를 기존의 방법에 비해 보다 작은 용량의 칩으로 구현하거나, 보다 적은 수의 칩만으로도 구현 가능하게 됨을 의미한다. 이러한 기능은 마치 시스템 소프트웨어 분야에서 제한된 주기억장치를 최대한 활용하기 위해 사용하는 가상메모리 (Virtual Memory) 기법과 유사하기 때문에 이를 ***'가상하드웨어'*** (Virtual Hardware) 기법이라고 부르기도 한다.

DPGA는 여러 가지 구조[3-9]로 제안되었는데, 재구성 정보의 저장 위치에 따라 크게 두 가지 방식으로 나누어 볼 수 있다. 첫째는 FPGA의 내부 논리 블록과 라우팅 설정 정보의 일부를 칩의 동작 중에 칩 외부로부터 수시로 입력받는 방식이다. 이러한 형태의 FPGA는 칩이 동작하는 중에 회로의 일부가 계속 새로운 기능으로 대체되므로 한 개의 칩 용량 이상의 논리 용량을 구현할 수 있으나, 재구성 정보가 칩 외부로부터 전달되어 와야 하므로 재구성에 따른 지연 시간이 크다는 단점이 있다. 이와는 달리, 칩에 대한 시간별 재구성 정보를 미리 만들어, 칩 내부의 특정 메모리 영역에 저장하여 두었다가 정해진 시간대가 되면 칩 전체를 재구성하는 방식이 있다. 이 방식은 칩 내부에 구성 정보를 저장할 메모리를 위한 면적이 추가적으로 필요하고 재구성 정보의 양이 제한되는 면이 있으나, 일반적으로 추가되는 재구성 정보의 저장에 필요한 메모리가 그리 크지 않을 뿐 아니라, 재구성에 필요한 데이타 이동이 칩 내부의 메모리로부터 이루어지므로 재구성 시간이 매우 빠르다는 장점이 있다. 이러한 구조의 대표적인 예는 Xilinx의 ***'시분할 FPGA'*** (Time Multiplexed FPGA)[3,4]이다. 본 논문에서는 Xilinx의 시분할 FPGA를 대상으로 한다.

그런데, 칩 내부에 저장될 재구성 정보를 미리 생성하는 시분할 FPGA 합성 문제를 해결하기 위해서는 다음의 두 가지 사항을 결정하여야 한다. 먼저, 주어진 전체 논리 회로 중, 어느 논리 회로 부분을 각각 하나의 LUT로 구현할 것인지를 결정하여야 한다. 또한 그 LUT로 구현될 회로 그룹을 어느 시간대에 활성화시킬 것인지를 미리 결정하여야 한다. (여기서 회로 그룹이 하나의 LUT로 구현되기 위해서는 단지 하나의 출력 신호만을 가지도록 묶여져야 한다) 하지만, 전자의 경우, 그 경우의 수가 너무 많아, 모든 경우를 고려하기 어렵기 때문에 기존 연구에서는 어느 회로 그룹이 동일한 LUT에 의해 수행될 지는 다른 방법이나 도구에 의해 이미 결정된 것으로 가정한 다음, 후자의 경우를 해결하는 데 중점을 두고 있다. 후자의 경우, 재구성 정보 생성시 특정 시간대에 필요한 LUT의 개수가 칩 내의 LUT 개수보다 많을 경우, 전체 회로의 구현이 불가능해 지므로, 구현에 필요한 LUT 개수를 최소화시키기

위한 노력이 필요하다.

이러한 DPGA 합성에서 사용하는 세부적인 재구성 기법에 따라 구현에 필요한 LUT의 개수가 달라질 수 있기 때문에, 본 논문에서는 입력되는 논리 회로를 분석하여 그 회로가 필요로 하는 전체 LUT 개수에 대한 하한(Lower Bound)을 추정함으로써 어느 재구성 기법을 사용하더라도 최소로 필요한 LUT 개수(즉, FPGA의 용량)를 파악하고자 한다. 만일, 기존의 재구성 결과가 본 연구에서 추정된 하한과 일치할 경우, 그 결과는 최적의 결과를 의미한다. 반면에, 하한과의 차이가 있는 경우에는 기존의 연구 결과에 비해 더 좋은 재구성 결과가 존재하거나, 또는 본 연구에서 추정한 하한보다 더 좋은(큰, 정확한) 하한이 실제 존재함을 의미한다. 따라서 이러한 비교 분석을 통해, 기존 연구의 결과가 최적인지, 또는 개선의 여지가 있는지를 판단하는 좋은 지표를 제공할 수 있다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 본 논문에서 주 대상으로 고려하는 시분할 FPGA를 위한 재구성 모델 및 이와 관련된 기존 연구에 대해 설명한다. 3장에서는 전체 LUT 개수에 대한 하한을 추정하는 기법에 대해 설명하며, 4장에서는 본 연구의 실험 내용과 기존 연구와의 결과 비교 분석을 기술한다.

2. 시분할 FPGA의 재구성 모델 및 용어 정의

2.1 시분할 FPGA의 내부 구조 및 동작

시분할 FPGA는 제한된 실제 하드웨어에 구현될 수 없는 거대한 회로를 일의 우선 순위 또는 데이타 흐름을 고려하여 시간대별로 적절히 나눈 후 시간대별로 수행하는 방법이다.[3, 4] 각 시간대별로 수행할 회로에 대한 구성 정보를 ***'context'***라고 하며, 모든 *context*는 미리 내부 메모리에 저장되어 있다가, 시간대에 따라 해당 *context*로 자동 변경된다. 그림 1은 Xilinx사의 시분할 FPGA의 간소화된 내부 구조를 보여준다.[3] 그림에서 하나의 *context*란 LUT로 구성된 각 논리 함수(logic function)의 기능을 결정하는 진리표 값과 회로간의 연결을 결정하는 MUX 각각에 대한 선택신호 값으로 구성되는 정보로서, 전체 회로의 기능을 결정한다. 이 예의 경우, 해당 논리 함수에 대한 진리표 값과 MUX 선택 신호는 각각 8개의 다른 값을 가지고 있는데, 이는 8개의 *context*로 구성되어 있음을 의미한다. 예를 들어, G1, G2, G3, G4를 입력으로 하는 논리 함수는 16 비트의 진리표 값에 따라 함수의 기능이 정의되는데, 그 함수는 시간대별로 새로이 구성되어야 하므로, 전체 *context* 수와 일치하는 총 8개의 16비트 진리표 값(전체적으로 16비트×8의 정보)이 그 함수의 시간대별 구성을 위해 미리 결정되어 있음을 알 수 있다.

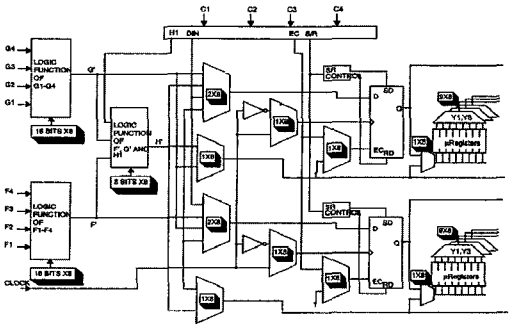

그림 1 시분할 FPGA의 내부 구조 예

주어진 논리 회로를 시분할 FPGA로 구현하기 위해서는 k개(주로 DPGA의 구조에 따라 미리 결정되어 있음)의 *context*를 구성하여야 한다. 예를 들어, 그림 1의 경우, 총 8개의 *context*를 가진다.(즉, k = 8) 이러한 각 *context*는 시간대별 회로 구성 정보를 의미하기 때문에 k개의 시간대 각각을 하나의 ***'micro cycle'***이라고 하며, k개의 *micro cycle*로 이루어진 전체 주기를 하나의 ***'user cycle'***이라고 한다. *user cycle*은 논리 회로 설계자 입장에서 볼 때 일반적인 수행 주기, 즉, 하나의 클록(clock)을 의미한다. 한편, *context*가 변경되면 이전의 회로는 모두 사라지고 새로운 회로가 구성되기 때문에, 어느 시간대에 생성된 결과(조합회로의 출력)가 그 이후에 다시 사용되는 경우, 이러한 값들은 ***'micro register'***라는 별도의 기억장치에 저장하여야 한다. 즉, 다른 *context*간의 정보 전달이 *micro register*를 통해 이루어진다.

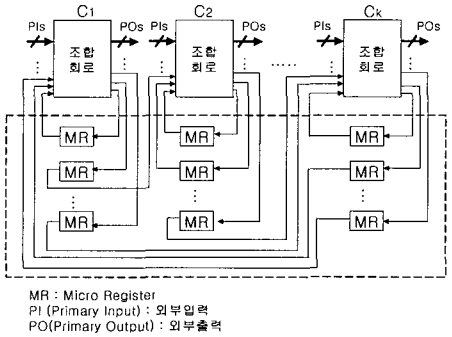

그림 2 시분할된 회로의 내부 통신 구조

그림 2는 시분할 FPGA의 내부 통신 및 저장 구조를 보여준다. 그림 2에서 각 *micro cycle*에 구성된 조합 회로는 외부 입력 신호(PI : Primary Input) 및 이전 구성에서 *micro register*에 저장된 값들을 입력으로 하여, 해당 기능을 수행한 후, 필요한 값들을 출력한다. 출력된 일부 값은 외부 출력 신호 (PO : Primary Output)로 직접 전달되기도 하며, 일부는 다음 구성에서 사용되기 위해 *micro register*에 저장되기도 한다. 결과적으로 *k*개의 *micro cycle* 단계(즉, 하나의 *user cycle*)를 거친 출력 결과는 주어진 논리 회로에서 하나의 클록이 수행된 후의 출력 결과와 동일하다.

논리 회로 설계자 입장에서는 외부 클록인 *user cycle*만이 의미가 있다. 실제 칩보다 더 큰 회로가 구현되기 위해서는 DPGA 재구성 기법에 의해 입력 회로에 대한 *context* 생성 결과가 칩에 다운로드(download)되어야 한다. 일단, 다운로드가 되고 나면, 칩 내부적으로 정해진 시간대(즉, *micro cycle*)별로 *context*들 간의 변경이 자동적으로 일어나면서 *k*개의 단계를 거친다. 이러한 내부적인 *context* 변경은 사용자에게는 투명(transparent)하게 진행되므로 칩의 사용자는 일반 FPGA를 사용할 때와 차이점을 느끼지 못한다.

2.2 시분할 FPGA 합성 문제의 재정의

본 논문에서 다루는 시분할 FPGA 합성 문제는 다음과 같은 문제로 해석할 수 있다.

- **입력** :
  1) 구현할 조합 회로에 대한 그래프 G
  2) k 값 (전체 *context*의 수, 즉 user cycle 내의 *micro cycle* 개수)
- **결과** : $R = \{ (u, S(u)) \mid u \in V \}$ (즉 각 노드에 대한 S 값을 결정)
  단, $P(u) < P(w)$인 임의의 두 노드 u, w에 대해, $S(u) \leq S(w)$라는 관계식이 반드시 성립하여야 한다.
- **최적화 목표** : $\max_{t=1,\ldots,k} (|C(t)|)$의 최소화

본 논문에서는 시분할 FPGA로 합성될 논리 회로는 조합회로로서 다른 도구에 의해 그래프 G = (V, E)로 변환되어 주어진다고 가정한다. 그래프 G에서 V는 노드들의 집합으로 각 노드는 하나의 LUT로 구현될 수 있는 회로 그룹을 의미하며, 반드시 하나의 출력 값을 가진다. 하나의 노드 즉, 하나의 LUT로 구현되는 회로 그룹을 '*가상 LUT*" (virtual LUT)[4]라고 부르기도 한다. E는 노드간의 입출력 연결 관계를 나타내는 것으로, E의 원소 (u, w)는 노드 u로부터 출력된 신호가 노드 v에 의해 사용됨을 의미한다. 즉, 노드 u의 기능을 실행하는 LUT의 출력이 w의 기능을 수행하는 LUT의 입력으로 사용됨을 뜻한다. 서로 다른 w1, w2에 대해 (u, w1)∈E와 (u, w2)∈E가 성립하는 경우, u의 동일한 출력 신호가 fanout되어 사용됨을 의미한다. k는 허용되는 전체 *context* 개수를 의미하는 상수로서, 한 user cycle 내의 *micro cycle* 개수를 의미한다.

또한 함수 P(u)는 노드 u의 실행 순서로서, 노드 u로부터 노드 w로 가는 경로가 존재하는 경우, '노드 u는 노드 w에 선행한다'라고 하며, P(u)<P(w)라고 표기한다. 또한 S(u)는 노드 u가 LUT로 실제 구현되는 *context* 번호, 즉 *micro cycle* 번호를 의미하며, 1부터 k 사이의 정수값을 갖는다. 합성 결과가 유효하기 위해서는, P(u)<P(w)인 임의의 u, w에 대해, 반드시 S(u) ≤S(w)가 성립하여야 한다. 함수 C(t)는 해당 노드의 S 값이 t인 노드들의 집합, 즉 $C(t)=\{u \mid u\in V \text{ and } S(u)=t\}$를 의미한다. C(t)는 *context* t에 실행되는 노드들의 집합으로서, C(t)의 크기에 따라 구현이 필요한 FPGA의 LUT 용량이 결정된다. 따라서 DPGA 합성은 노드 간의 선행관계를 유지하는 조건 하에서, C(t)의 최대 값이 최소화되도록 각 노드에 대한 S 함수 값을 결정하는 문제라고 볼 수 있다.

그림 3에서 그림 6까지의 그림들은 동일한 예제 그래프에 대한 다양한 시분할 FPGA의 합성 결과를 보여준다. 그림 3, 그림 4, 그림 5는 모두 올바른 재구성 결과이지만, 그림 3이나 그림 4는 적어도 3개의 LUT를 필요로 하지만, 그림 5는 2개의 LUT만을 요구하게 된다. 또한 *context*의 수가 그래프의 임계 경로 길이보다 작은 경우, 부득이 순차적인 수행이 요구되는 여러 개의 LUT들을 하나의 *micro cycle*에 배정할 수도 있다. 이러한 것을 ***'스케쥴링 압축'*** (scheduling compression) [4, 9]이라고 한다. 즉, 스케쥴링 압축은 P(u)<P(w)이고 (u, w)∈E인 두 노드 u와 w를 필요에 따라 S(u) < S(w)

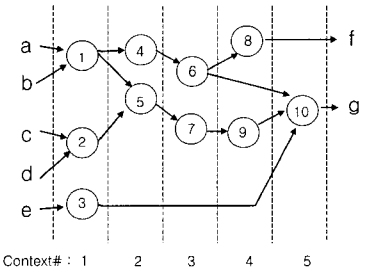

그림 3 각 노드를 가능한 가장 빠른 *context*에 재구성한 결과

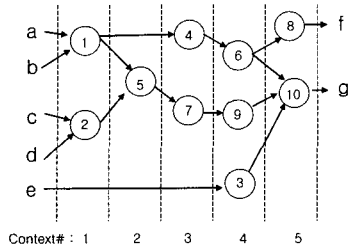

그림 4 각 노드를 가능한 가장 나중 *context*에 재구성한 결과

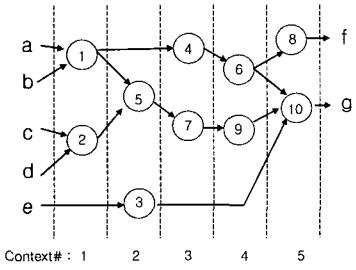

그림 5 각 노드를 여러 *context*에 적절하게 재구성한 결과

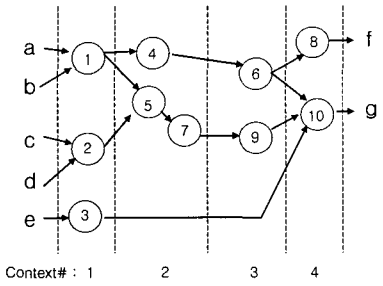

그림 6 *context* 수 감소를 위해 순차적인 노드들을 동일 *context*에 재구성한 결과

가 아니라 S(u)=S(w)가 성립하도록 재구성하는 것을 뜻한다. 그림 6은 스케줄링 압축이 적용된 결과를 보여준다. 단, 이 경우, 지연 시간 증가에 따라 *micro cycle*의 크기가 늘어날 수 있다. 반대로, *context* 수가 충분한 경우, *micro cycle*의 개수를 늘려, 동일 시간대에 배정되는 노드의 개수를 줄일 수도 있는데, 이를 ***'스케줄링 확장'*** (scheduling extension)[4]이라고 한다.

2.3 스케줄링 문제와의 연관성

앞 절에서 설명한 용어에서 보다시피, 이러한 시분할 FPGA의 합성 문제는 기존에 상위단계합성[10] 분야에서 연구되어 오던 스케줄링(scheduling) 문제[10-14] 또는 회로 분할 (circuit partitioning) 문제[10, 15]와 매우 유사하다. 표 1은 시분할 FPGA 합성 문제와 스케줄링 문제와의 유사성을 설명하고 있다. 두 문제 모두 그래프를 입력으로 하여, 각 노드가 실제 실행될 시간을 결정하는 기법으로서, 스케줄링에서는 수행할 각 종류별 연산을 하나의 노드로 보는 반면, FPGA 합성에서는 LUT로 구현될 회로 그룹을 하나의 노드로 취급한다. 또한 스케줄링 문제에서는 일반 클록을 기준으로 하는 반면, FPGA 합성 문제에서는 *micro cycle*을 기준으로 함을 알 수 있다. 그리고 스케줄링 문제에서는 여러 가지 연산자가 존재하므로 각 연산자별로 필요 개수를 계산하지만, FPGA 합성에서는 LUT라는 특수한 연산지 한 종류만이 사용된다고 볼 수 있다.

한편, 스케줄링 문제라는 관점에서 보면, 그림 3과 그림 4는 각각 ASAP 스케줄링과 ALAP 스케줄링 결과이며, 그림 6은 노드의 체이닝을 고려한 스케줄링 기법을 적용한 결과로 볼 수 있다.[10-14]

표 1 스케줄링 문제와 시분할 FPGA용 합성간의 유사성 비교

비교 항목	기존의 스케줄링 문제	시분할 FPGA 합성 문제
입력정보	수행할 연산들과 그들 간의 선행 관계를 나타내는 방향성 그래프	하나의 LUT로 구현 가능한 노드들과 이들간의 입출력 연결 관계를 나타내는 방향성 그래프
실행결과	각 연산들의 실행 시기, 즉 각 연산이 연산자에 의해 수행될 클록주기 결정	각 노드가 재구성되어 실행되는 시기, 즉 노드가 실제 LUT로 구현되는 *micro cycle* 결정
제약조건	선행 관계 유지	선행 관계 유지
최적화 대상	종류별 연산자수(동일 클록에 스케줄링된 종류별 연산 노드들의 개수)	LUT의 개수(동일 *micro cycle*에 배정된 노드들의 개수)

2.4 관련연구

시분할 FPGA 합성과 스케줄링과의 유사성으로 인해 기존의 많은 연구에서는 기존에 상위단계 합성분야의 연구에서 개발된 각종 스케줄링 기법 또는 다중 회로 분할 기법들을 수정하여 적용하고 있다. 스케줄링 기법 및 회로 분할 기법 자체에 대한 자세한 설명은 참고문헌[10-15]을 참조하기 바란다.

참고문헌 [4], [5], [6] 등의 연구에서는 기존의 스케줄링 기법을 변형한 방법을 사용한다. [4]에서는 기존의 리스트 스케줄링 기법을 합성 문제에 알맞게 변형하여

적용하여 일단 초기 결과를 얻은 후, 스케줄링 압축 및 스케줄링 확장 기법을 이용하여 그 결과를 반복적으로 향상시킨다. [5]에서는 회로의 임계 경로를 고려하는 스케줄링 기법 (Precedence Constraint Scheduling) 기법을 사용하는데, 초기 결과를 토대로 반복적인 재스케줄링(rescheduling : 부분적인 스케줄링 결과 조정)을 적용하여 보다 좋은 결과를 찾는다. 특히, 이 방법에서는 *micro register* 및 IO의 최소화를 위해, 오히려 LUT의 개수가 증가되는 것을 의도적으로 허용한다. [6]에서는 가장 빠른 회로 구현을 위해, FDS 스케줄링 (Force-Directed Scheduling)[12]을 변형한 알고리즘을 사용하며, LUT의 개수 이외에 *micro register*의 최소화도 고려한다.

반면에 [8]과 [9] 등의 연구에서는 기존의 회로 분할 기법을 변형하여 적용한다. [8]은 *micro register*의 최소화에 중점을 두고 있으며, 이를 위해 다중 회로 분할이 가능한 '네트워크 흐름 방식의 회로 분할' (network flow based circuit partitioning) 기법을 활용한다. [9]에서는 향상된 결과를 얻기 위해 최적의 스케줄링 압축 기법을 제안하고 있다.

3. LUT 개수에 대한 하한 추정 기법

시분할 FPGA 합성에서, 구현에 필요한 FPGA의 최소 용량은 동일 *micro cycle*에 스케줄링(할당)되는 노드들의 최대 개수, 즉 동일 *micro cycle*에서 동시에 사용되는 LUT의 최대 개수에 의해 결정된다. 그런데, 최적의 스케줄링 결과를 얻는 것은 NP-hard 문제로 알려져 있기 때문에 이를 보장한다는 것은 매우 어려운 일이다. 하지만, 특정한 조건을 만족하는 경우, 이에 대한 보장을 할 수 있다. 그러한 방법 중의 하나가 하한 추정 기법[13-14, 16]을 이용하는 것이다. 즉, 만일, 주어진 문제에 대한 (이론적) 하한이 알려져 있고, 또한 어느 한 결과가 그와 동일한 결과를 낸다면, 그 결과는 분명히 최적의 결과라고 할 수 있다. 이에 본 논문에서는 주어진 입력 정보에 대한 분석을 통해 동일 *micro cycle*에서 동시에 사용되는 LUT의 최대 개수에 대한 하한을 추정함으로써 기존의 여러 합성 기법에 의해 생성된 결과가 최적인지를 판단하는 지표로 삼고자 한다.

LUT 개수에 대한 하한 추정의 이론적인 배경은 다음과 같다. 만일, 주어진 그래프 G=(V, E)에서 모든 노드가 k개의 *micro cycle*에 자유롭게 스케줄링될 수 있다고 가정하면, 적어도 $\lceil (|V| / k) \rceil$개의 노드가 스케줄링된 하나 이상의 *micro cycle*이 반드시 존재한다는 것이다. 여기서 $\lceil (|V| / k) \rceil$는 (|V| / k)의 계산 값보다 크거나 같은 최소의 정수 값을 의미한다. 따라서 이 값을 LUT 개수에 대한 하한으로 추정할 수 있다. (이를 본 논문에서는 ***'단순 하한'*** (trivial lower bound)이라고 부른다.)

하지만, 일반적인 경우, 그래프의 노드간에는 연결 관계(선행 관계)가 존재하기 때문에 대부분의 경우에서는 각 노드들이 각 *micro cycle*에 걸쳐 균등하게 분포되지 못하는 경우가 많다. 따라서 본 논문에서는 이러한 연결 관계를 잘 이용하면 앞에서 구한 단순 하한보다는 훨씬 정확한 하한, 즉, 보다 큰(정확한) 하한 값을 추정할 수 있다. 예를 들어, 노드간의 선행 관계 때문에 전체 노드 중 m개는 반드시 *micro cycle* 범위 $[from, to]$ 내에서 수행되어야 한다면, $\lceil (m / (to-from+1)) \rceil$개 이상의 노드가 스케줄링되는 *micro cycle*이 하나 이상 반드시 존재한다. 따라서, 만일 이 값이 앞에서 구한 하한에 비해 크면, 이를 새로운 하한으로 결정할 수 있다.

주어진 범위 $[from, to]$에 반드시 스케줄링되는 노드의 개수 m은 다음과 같은 방법을 통해 쉽게 구할 수 있다. 먼저, 주어진 전체 그래프에 대해 각 노드를 최대한 빠른 *context*에 배정하는 ASAP 스케줄링과 가장 늦은 *context*에 배정하는 ALAP 스케줄링을 실행하여, 각 노드 u가 스케줄링 가능한 최대 범위 $[A_u, B_u]$를 구한다. 그러면, 적용되는 스케줄링 기법에 관계없이, 범위 $[from, to]$에 반드시 스케줄링되는 노드들의 집합은 $\{u \mid u \in V, from \le A_u \le B_u \le to\}$로 표현될 수 있다. 즉, 자신의 스케줄링 가능 범위$[A_u, B_u]$가 주어진 범위$[from, to]$에 포함되는 임의의 노드 u는 반드시 주어진 그 범위 내에 스케줄링된다. 따라서 이러한 집합의 크기를 해당 범위 $[from, to]$에 대한 m으로 결정할 수 있다.

본 논문에서는 보다 정확한 하한을 추정하기 위해 설정 가능한 모든 micro cycle 범위 $[from, to]$ ($1 \le from \le to \le k$, k는 전체 *context*의 수)에 대해 위의 과정을 반복하여 가장 큰 값을 LUT 개수에 대한 하한 추정 값으로 최종 선정한다. 여기서 반드시 알아야 할 매우 중요한 특성은 이렇게 추정한 하한은 주어진 문제에 대한 이론적 하한에 비해 항상 같거나 작은 성질이 있다는 것이다. 또한 이러한 하한 추정은 복잡한 스케줄링을 수행하지 않고 단지 그래프에 대한 분석만을 통해 산출한다는 점이다.

다음의 그림 7은 본 논문에서 제안한 하한 추정 알고리즘을 간략하게 요약한 것이다.

만일, 기존의 연구에서 제시한 합성 결과가 본 연구에서 추정된 하한과 일치할 경우, 그 결과는 최적의 결과라고 분명히 말할 수 있다. 반면에, 추정된 하한과의 차

1. 주어진 그래프 G=(V, E)와 *context* 수 k를 입력으로 받는다.
2. G에 대한 ASAP 스케쥴링과 ALAP 스케쥴링을 각각 수행하여, 각 노드 u에 대해, 그 노드가 스케쥴링될 수 있는 *micro cycle* 범위 $[A_u, B_u]$를 구한다.
3. 단순 하한을 구한다. 즉,
   LB = $\lceil(|V|/k)\rceil$, k는 주어진 *context*의 개수;
4. 설정 가능한 모든 범위 $[from, to]$ $(1 \le from \le to \le k)$에 대해,
   $m = |\{ u \mid u \in V, from \le A_u \le B_u \le to \}|$;
   LB = max (LB, $\lceil m / (to-from+1) \rceil$);
5. LB를 추정된 하한 값으로 출력한다.

그림 7 간소화된 하한 추정 알고리즘

이가 있는 경우에는 기존의 연구 결과에 비해 더 좋은 합성 결과가 존재하거나, 또는 본 연구에서 추정한 하한보다 더 좋은(큰, 정확한) 하한이 실제로 존재할 수 있음을 의미한다. 따라서 이러한 비교 분석을 통해, 기존 연구의 결과가 최적인지, 또는 개선의 여지가 있는지를 판단하는 좋은 지표를 제공할 수 있다.

표 2는 앞에서 설명한 그림 3, 4, 5에서 사용한 동일한 예제에 대해, 각 범위별 LB 값을 보여준다. 한 예로, 범위 [1,3]의 경우를 살펴보면, 노드 1, 2, 4, 5, 7은 반드시 이 범위 내에 스케쥴링되어야 한다. 반면에 노드 3과 노드 6은 각각 *context* 4에도 스케쥴링될 가능성이 있으므로 반드시 그 범위 내에 스케쥴링된다고 볼 수 없다. (여기서는 설명의 간소화를 위해, 스케쥴링 압축 및 확장은 고려하지 않는다고 가정한다.) 이러한 사실은 노드들에 대한 ASAP 및 ALAP 스케쥴링 결과로부터 쉽게 구할 수 있다. 이 경우에 계산되는 LB 값은 $\lceil 5 / (3-1+1) \rceil = 2$이 된다. 표 2에 나타난 바와 같이, 모든 범위에 대해 이와 같은 LB들이 각각 결정되면, 이중 가장 큰 값를 추정된 하한으로 결정한다. 이 경우는 최대 값이 2가 나오므로, LUT 개수에 대한 하한은 2라고 추정한다. 결과적으로 그림 3와 그림 4에 대해서는 추정된 하한이 해당 결과의 LUT 개수 보다 작으므로 그 결과가 최적인 지를 판단하지 못한다. 이 경우는 추정된 하한이 부정확하거나 아니면, 그 결과가 개선의 여지가 있음을 판단할 수 있다. 반면에 그림 5의 경우, 그 두 값이 일치하므로 그림 5의 결과는 최적이라고 분명히 확신할 수 있다.

4. 실험 결과 및 분석

본 논문에서 제안한 하한 추정 기법은 UNIX 운영체제 (Solaris 2.6 이상)에서 C 프로그래밍 언어로 구현되었다. 개발된 하한 추정 시스템의 성능을 검사하고, 또한 기존 연구 결과의 정확성을 분석하기 위해, MCNC 벤치마크를 대상으로 실험하였다. 표 3은 본 논문에서 실험한 내용 중 주요 결과를 보여준다. 표에 제시된 값들은 실험된 각 예제에 대한 1) 예제명, 2) 예제의 크기 (노드의 수), 3) 임계 경로의 길이, 4) 기존의 스케쥴링 결과중 문헌에 제시된 최상의 합성 결과에서 필요한 LUT 개수 (참고문헌 [4] 및 [5]에서 발췌), 5) 단순 하한 (즉, $\lceil$ (전체 노드수) / (전체 *context* 개수) $\rceil$), 6) 본 논문에서 추정한 LUT 개수에 대한 하한, 7) 추정된 하한과 단순 하한의 차, 8) 추정된 하한과 기존 결과의 차이를 각각 나타낸다. 표에서 보면, 거의 모든 예에서 기존의 스케쥴링 결과와 본 논문에서 추정한 하한이 정확히 일치하는 것을 볼 수 있는데, 이는 기존 결과가 최적임을 확인시켜 줌과 동시에, 본 논문에서 제안한 하한 추정 기법이 매우 정확함을 반증하는 것으로 해석할 수 있다. 특히, 표의 마지막 두 칸에 제시된 정보는 본 논문에서 제안한 하한 추정 기법이 단순 하한에 비해 향상된 정도와 기존 스케쥴링 결과에 대한 하한 추정의 정확성을 보여준다. 한편, 본 연구는 현재로선 조합 회로에 국한되고, 순차 회로에 대해서는 적용할 수 없는 한계가 있기 때문에, 참고문헌에 제시된 기존의 연구 결과 중 조합 회로에 대한 부분만을 발췌[4, 7]하여 결과를 비교하였다.

표 2 각 *context* 범위 $[from, to]$에 대해 추정한 하한 ($\lceil m/(to-from+1) \rceil$)

from \ to	1	2	3	4	5
1	⌈2/(1-1+1)⌉=2	⌈3/(2-1+1)⌉=2	⌈5/(3-1+1)⌉=2	⌈8/(4-1+1)⌉=2	⌈10/(5-1+1)⌉=2
2		⌈1/(2-2+1)⌉=1	⌈3/(3-2+1)⌉=2	⌈5/(4-2+1)⌉=2	⌈7/(5-2+1)⌉=2
3			⌈1/(3-3+1)⌉=1	⌈3/(4-3+1)⌉=2	⌈5/(5-3+1)⌉=2
4				⌈1/(4-4+1)⌉=1	⌈3/(5-4+1)⌉=2
5					⌈1/(5-5+1)⌉=1

* 최대 LB 값 = 2 (추정된 하한)

표 3 기존의 합성(스케쥴링) 결과와 하한 추정 결과의 비교

예제	전체 노드수 (N)	임계경로 길이(C)	LUT 개수 (개)			추정하한과 단순하한의 차이 (LB-⌈N/C⌉)	추정하한과 기존결과의 차이 (Z-LB)
			문헌상 최상의 스케쥴링 결과 (Z)	단순 하한 (⌈N/C⌉)	추정 하한 (LB)		
C2670	320	11	30	30	30	**0**	**0**
C3540	1080	14	127	78	127	49	**0**
C5315	592	10	60	60	60	**0**	**0**
C6288	618	28	33	23	33	10	**0**
C7552	1039	11	108	95	108	13	**0**
C880	221	12	19	19	19	**0**	**0**
alu4	590	11	69	54	69	15	**0**
dalu	605	6	150	101	150	49	**0**
des	1437	7	206	206	206	**0**	**0**
i8	632	6	114	106	114	8	**0**
i9	212	5	115	43	115	72	**0**
i10	1508	16	95	95	95	**0**	**0**
k2	1177	6	285	197	285	88	**0**
t481	408	8	123	51	123	72	**0**
vda	301	6	61	51	58	7	3
x3	419	5	84	84	84	**0**	**0**
합계			1,679	1,293	1,676	383	3

5. 결 론

시분할 FPGA 합성 문제는 상위단계 합성의 주요 단계인 스케쥴링 문제 (또는 다중 회로 분할 문제)와 매우 유사하다. 따라서 본 논문에서는 상위단계 합성에서 스케쥴링 결과 분석에 유용하게 사용되는 하한 추정 기법을 도입하여, 시분할 FPGA 합성 문제에 적용함으로써, 기존의 시분할 FPGA 합성 결과의 최적성 분석 및 결과 향상 가능성 분석 등에 사용하고자 하였다.

실험 결과, 대부분의 경우 기존의 연구 결과와 동일한 하한이 추정되는 것을 발견할 수 있었는데, 이는 기존의 합성 시스템에서 생성한 결과의 최적성을 확인하게 하는 한편, 본 논문에서 제안한 하한 추정의 정확성 및 성능을 직접적으로 보여준다.

하지만, 기존의 시분할 FPGA 합성 연구에서는 조합 회로는 물론 순차 회로도 고려해야하는 반면, 본 논문에서는 하나의 클록 내에서 동작하는 조합 회로만을 대상으로 하였기 때문에, 여러 클록에 걸쳐 수행되는 순차 회로에 대해 실험한 기존 연구 결과에 대한 분석이 제대로 이루어질 수 없는 문제가 있다. 향후, 조합 회로는 물론 순차 회로의 경우에도 적용할 수 있도록 본 논문에서 제안한 하한 추정 기법을 확장하는 연구가 필요하다. 또한 LUT 개수만 아니라, *micro register*의 개수도 중요한 요소로 고려하기 위한 연구도 필요하다.

참 고 문 헌

[1] R. Murgai, K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Synthesis for Field Programmable Gate Arrays," Kluwer Academic Publisher, 1995.

[2] Kang Yi, Seong Y. Ohm, and Chu S. Jhon, "An Efficient FPGA Technology Mapping Tightly Coupled with Logic Minimization," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E80-A, pp.1807-1812, Oct. 1997.

[3] S. Timberger, "A Time-Multiplexed FPGA," Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[4] S. Trimberger, "Scheduling Designs into a Time-Multiplexed FPGA," Proceedings of International Symposium on the Field Programming Gate Array, 1998, pp. 153-160.

[5] D. Chang et al, "Buffer Minimization and Time-Multiplexed I/O on Dynamically Reconfigurable FPGAs," ACM/SIGDA International Synposium on Field Programmable Gate Arrays, 1997, pp. 142-148.

[6] D. Chang at al, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," ACM/ SIGDA International Synposium on Field Programmable

Gate Arrays, 1998. pp. 161-167.
[7] H. Liu and D.F. Wong, "Circuit Partitioning for Dynamically Reconfigurable FPGAs," ACM/SIGDA International Synposium on Field Programmable Gate Arrays, 1999. pp.187-194.
[8] H. Liu and D.F. Wong, "Network Flow Based Circuit Partitioning for Time-Multiplexed FPGAs," Proceedings of International Conference on the Computer Aided Design, 1998, pp. 497-504.
[9] H. L. and D. F. Wong, "A Graph Theoretic Optimal Algorithm for Scheduling Compression in Time-Multiplexed FPGA Partitioning," Proceedings of International Conference on the Computer Aided Design, 1999, pp. 400-405.
[10] M. C. McFarland, A. C. Parker, and R. Composano, "Tutorial on High Level Synthesis," Proceedings of the 25th Design Automation Conference, pp.330-336, June 1988.
[11] C. T. Hwang, J. H. Lee, and Y. C. Chu, "A Formal Approach to the Scheduling Problem in High Level Synthesis," IEEE Transactions on Computer-Aided Design, pp.464-475, April 1991.
[12] P. G. Paulin and J. P. Knight, "Force-directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Transactions on Computer-Aided Design, pp.661-679, June 1989.
[13] 엄성용, 전주식, "하한 비용 추정에 바탕을 둔 최적 스케쥴링 기법", 대한전자공학회 논문지, 제28권 A편 제12호, pp.73-87, 1991년 12월.
[14] Seong Y. Ohm, Chu S. Jhon, and Fadi J. Kurdahi, "An Optimal Scheduling Approach using Lower Bound in High-Level Synthesis," IEICE Transactions on Information and Systems, Vol. E78-D, No.3, pp.231-236, March 1995.
[15] Hyun-Chul Shin and Chung-Hee Kim, "A Simple Yet Efficient Techniques for Partitioning," IEEE Transactions on VLSI Systems, Vol. 1, No. 3, pp.380-386, 1993.
[16] Seong Y. Ohm, Fadi J. Kurdahi, Nikil Dutt, "A Unified Lower Bound Estimation Technique for High-Level Synthesis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 5, pp.458-472, May 1997.

엄 성 용
1985년 서울대학교 컴퓨터공학과 졸업(학사). 1987년 서울대학교 대학원 컴퓨터공학과 졸업(석사). 1992년 서울대학교 대학원 컴퓨터공학과 졸업(박사). 1992년 ~ 1993년 컴퓨터신기술공동연구소 특별연구원. 1993년 ~ 1995년 University of California, Irvine에서 Post-Doc. 1996년 ~ 현재 서울여자대학교 정보통신공학부 부교수. 관심분야는 상위단계합성, VLSI/CAD, 컴퓨터그래픽스, 디지털 방송, 홈네트워킹