**디지털회로설계 HW #6**

**Technology Mapping**

20161453 전자공학과

김규래

1. What is Technology Mapping

일반적인 상용 CAD 프로그램들은 두 가지 종류가 있다. Technology Independent Mapping과 Technology Mapping이다. 전자의 경우 물리적으로 칩을 구현할 때 활용 가능한 리소스들을 고려하지 않는 최적화를 말하며, 후자는 실제로 활용 가능한 리소스들을 고려한 최적화를 말한다. Technology Mapping의 대표적인 결과는 AND와 OR들로 이루어져있는 회로가 NAND로만 이루어진 회로로 바뀌는 것이다.

1. Technology Mapping in Synthesis

Synthesis 과정은 총 3급(Class)로 분류된다. Abstract Representation, Logic Description, Actual Implementation. 1급은 High-Level Synthesis, 2급은 Logic-Level Synthesis, 3급은 Physical-design Synthesis 이다. Logic-level Synthesis가 앞서 말한 Technology Dependent와 Independent로 나눠진다.

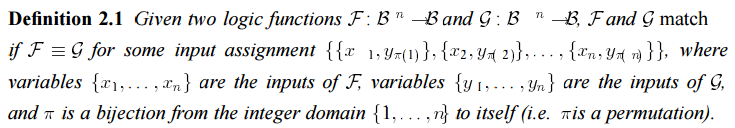
우리 교재에서는 Synthesis를 3단계로 분류하였는데, 순서대로 Netlist Generation, Gate Optimization, 마지막으로 Technology Mapping이다. 이하에서는 Netlist Generation 이후 Gate Optimization 을 High-Level Synthesis와 Logic-level Synthesis로 구분하여 말하겠다.

1. The goal of Technology Mapping

Technology Mapping의 목적은 Technology-independent한 Circuit description을Technology-dependent한 Circuit description으로 바꾸는 것이다. Technology-dependent 구현은 딜레이나 면적 등의 비용들을 최적화해야 한다. Logic-Level Synthesis에서 Technology-independent operation을 거친 결과물은 Boolean 방정식의 형태로 나타난다. Target technology들은 Technology specific library로 나타나며, 이들은 Target technology들에서 구현돼있는 Logic gate의 형태인데, Technology mapping을 통해서 이 Target technology 라이브러리들이 요구하대로 Boolean 방정식들을 바꾸는 것이다. 많은 Target technology 들은 라이브러리를 일반적으로 단일 출력 Combinational gate로 요구한다.

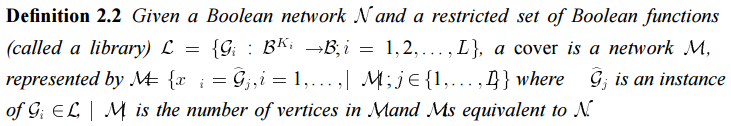
Technology-independent 한 디자인에서 Technology dependent한 디자인으로 변환하는 과정은 두 가지 연산을 필요로 한다. 두 개의 서로 다른 디자인이 Logically equivalent 하다는 것을 판별하는 연산과, Logically equivalent 한 회로들 중에서 현재 constraint들을 만족하는 최상의 회로를 판별하는 연산이다. 전자를 Matching 이라고 하며, 후자를 Covering이라고 한다.

1. Matching

Matching 연산의 형식 정의는 다음과 같다.

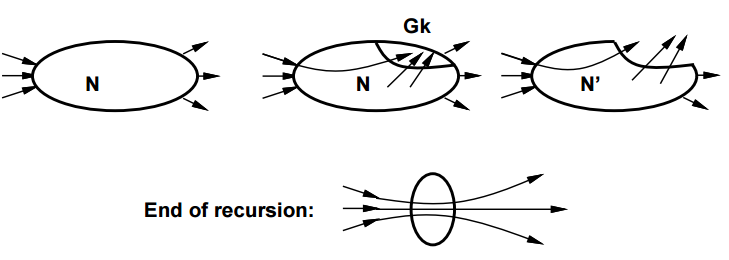
요약하자면, N개의 입력의 모든 조합들이 논리함수 F와 G에서 일치한다는 것을 확인해야 한다는 것이다.

1. Covering

Covering 연산의 형식 정의는 다음과 같다.

어떠한 네트워크 N 의 출력 에 관해서, 인 라이브러리의 부분집합 는

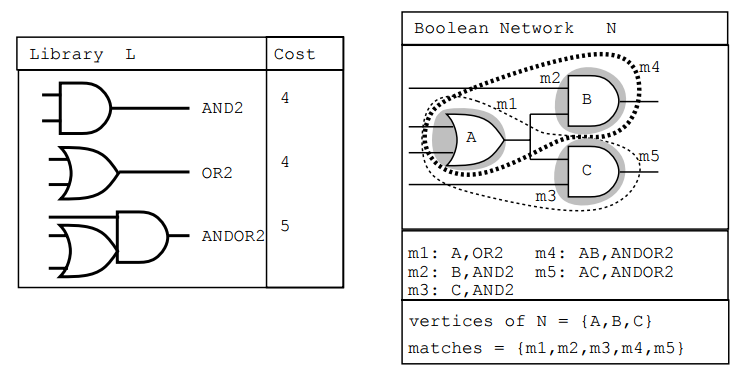
아웃풋이 와 동일하다. 이때 의 인풋들은 네트워크 N의 부분집합이거나 N 내부 변수들의 부분집합이다. 이때 는 출력의 모든 일치하는 조합들의 집합이다. 인 조합 가 선택됐을 때, 의 모든 변수들은 다른 의 부분집합들의 조합을 통해 구현될 수 있어야 한다.

이제 N의 출력 를 제거하고 의 모든 인풋들이 새로운 네트워크 의 아웃풋이라고 하자. 이때 N` 의 아웃풋에 해당하는 조합을 구하는 과정에서 N의 인풋들이 결정되고, 이 작업은 재귀적으로 반복된다.

<figure 1> Covering 작업을 나타내는 그림

실제 활용예를 보자면, <figure 2>는 아래에 라이브러리 집합 L을 통해서 네트워크 N을 새로운 네트워크 M으로 변환하는 과정을 보여준 것이다.

N의 각 vertex는 L의 원소 최소한 하나로 표현될 수 있다. 또한 L의 원소 하나는 N의 vertex 최소한 하나를 표현할 수 있으며 그 이상을 표현할 수도 있다. 따라서 M 은 L의 원소들의 다양한 조합으로 표현될 수 있다.

<figure 2>에서 네트워크 N의 원소 A를 표현할 수 있는 방법은 2가지가 있다. 라이브러리 L에서 OR2, 또는 B 또는 C와 함께 묶여서 ANDOR2로 표현될 수 있다. 이때 각 조합에 해당하는 변수를 만든다면 m1 = {OR2} for A, m2 = {AND2} for B, m3 = {AND2} for C, m4 = {ANDOR2} for AB, m5 = {ANDOR2} for AC 로 표현될 수 있다. A를 OR2로 표현하는 것을 선택한다면 m1 = 1, 선택하지 않는다면 m1 = 0 이라고 하자.

<Figure 2>

A는 따라서 () 라는 Boolean 수식으로 표현될 수 있다. 마찬가지로, B와 C는 (), ( 로 표현될 수 있다. 원래 회로의 모든 Vertex들을 커버하기 위해서는 위에서 언급한 모든 수식들이 동시에 참이여야 한다는 조건이 필요하다. 따라서 (() () () = 1) 이다.

이러한 조건들과 함께 특정 조합은 다른 조합의 아웃풋을 필요로 하는 의존성관계가 성립하는 경우가 있다. 예를 들어 = 1, = 1 인 경우에, 의 출력을 가 필요로 한다. 이 관계를 로 표현하자. 비슷하게 로 표현할 수 있다. 따라서 우리의 예시 문제를 해결하기 위해서는

= 1

로 수식을 확장할 수 있다. 그리고 이를 다시

= 1

로 변환하고, 이를 마지막으로

)

로 정리할 수 있다.

각 조합들의 비용은 각 원소의 비용 C를 곱한 것과 같다.

결론적으로 각 조합의 비용은

(12, 13, 13, 10)

이고, 따라서 최선의 조합은

이다.

1. Methods of Technology Mapping

이러한 Technology Mapping 과정의 복잡성으로 인해서 Heuristic 이나 더 단순화한 방법론을 활용하는 것이 일반적이다. Technology Mapping은 보통 두 가지로 분류를 하며 각각 Rule-based 와 Algorithmic이다.

Rule-based의 경우에는 몇 개의 규칙이 가능한 변환들을 추상화하는 방식이며, 이 규칙들에 의거해서 변환을 하는 것이다. 이 규칙들이라는 것은 보통의 경우 Target Technology 에 특화된 것들이며 따라서 Technology에 따라서 정해진 규칙의 집합에 새로운 규칙을 추가한다는 것은 쉽지 않다.

Algorithmic-based 방식에서는 일반적인 Boolean-logic을 순차적으로 Technology-specific한 디자인으로 바꾼다.

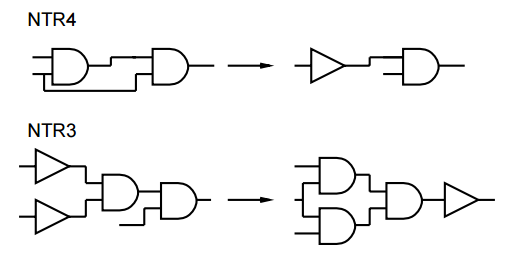
1. Rule-based systems

Rule-based system은 대표적으로 LSS와 Socrates가 대표적인 방식이다. Rule-based 방식의 장점들은 Technology Specific한 디자인 패턴들과, 회로 디자이너들의 개인적인 노하우 및 디자인패턴들을 적용할 수 있다는 장점이 있다.

Rule-based 에서는 Target Technology를 기준으로 정해진 몇가지 Rule들에 따라서 매우 미시적인 관점으로 변환을 진행한다.

일반적인 Rule 들은 다음과 같은 형태로 정의될 수 있다.

Rule들은 순서대로 3가지 등급, General rules, Timing rules, Area rules 로 분류될 수 있으며, 앞선 Rule이 더 우선시된다. General rule에 경우 모든 경우에 일반적으로 적용되는 변환 규칙을 뜻하며, Timing rule은 Delay minimization, Area rules는 Area minimization을 위한 변환 규칙들을 뜻한다.

각 Rule들을 적용하기 위해서는 각 단위에 대해 Match: 어떠한 규칙이 적용될 수 있는지를 판단하고, Cost: 적용될 수 있는 Rule들의 비용을 계산해보며, Select: 최선의 Rule을 선택한 다음, Replace: Replacement configuration을 적용한다.

<figure 3>

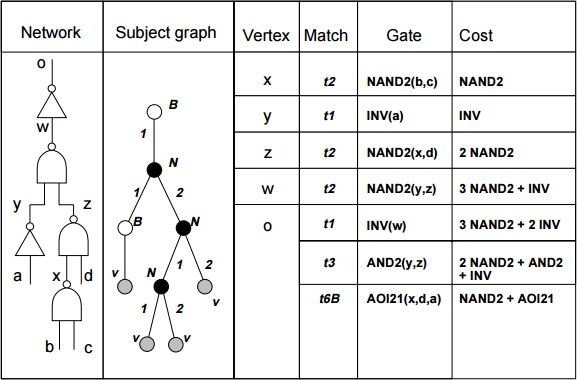
<figure 3>은 LSS 시스템에서 실제로 사용됐던 대표적인 General rule을 나타낸 것이다.

Socrates 시스템에서는 일반적인 Rule-based 시스템들과 다르게 State space search tree 알고리즘을 적용했다. 기존의 Rule-based 시스템들이 미시적인 관점에서 작동했던 만큼, 거시적인 관점에서 최적의 결과를 판별하지 못해낼 수 있다는 문제가 있었는데, Socrates는 이러한 문제를 해결한 것이다. 이 뿐만 아니라 Socrates는 Technology-dependent한 부분은 Algorithm-based로, Technology-dependent한 부분은 Rule-based를 적용한 Hybrid 형태였다는 점에서 당대에는 혁신적이었다.

이와 같은 Rule-based 시스템들은 다음과 같은 단점들이 있다. 기존의 Rule들과 어떻게 상호작용할지를 판단해야 한다는 점에서 1) Rule을 새로 생성해내고, 유지하기가 어렵다. 2) Technology Dependence가 굉장히 높아서 일반화하여 다른 Technology에 적용하기가 어렵다. 3) Rule들의 질과 Rule 선정과정에 결과물의 품질이 굉장히 의존적이다.

1. Algorithm-Based Approach

Algorithm-based Approach는 소프트웨어 프로그래밍 언어들의 컴파일 과정과 비슷하다. 프로그래밍언어들이 컴파일 과정에서 Abstract Syntax Tree, AST로 표현되는 것과 비슷하게 회로는 Directed Acyclic Graph, DAG로 표현된다. 이 DAG에서 각 Node는 임의의 연산을 표현하고, 각 Edge는 Data dependency를 표현한다. 최종적인 DAG에서는 각 노드의 연산들은 Restricted class에 포함되는 것들이여야 한다. 이 경우 Restrcited class는 Target Technology에서 제공하는 연산들이다. 또, 최종 DAG는 비용이 최소인 최적이다.

Pattern matching 에 있어서 DAG는 굉장히 연산난이도가 높다는 것이 알려져, 일반적으로는 Tree matching이 사용된다. 라이브러리 내의 연산들은 트리로 표현되고, Tree matching automaton을 각 라이브러리별로 생성한다.

<figure 4>

Boolean 네트워크는 이런 트리들의 집합, 숲으로 표현될 수 있다. <figure 4>는 NAND와 인버터들의 네트워크를 트리 형태로 표현한 모습이다.