論文 97-22-8-1

대규모 조합문제를 해결하기 위한 효율적인 논리함수 처리 시스템의 개발과 순서회로 설계에의 응용

正會員 권 용 진*

Development of an efficient logic function manipulation system for solving large-scale combination problems and its application to logic design of sequential circuits

Yong Jin Kwon* *Regular Member*

※이 논문은 1995학년도 한국학술진흥재단의 공모과제(신진연구) 연구비에 의하여 연구되었음

요 약

계산기상에서 논리함수를 효율적으로 처리하기 위한 논리함수 처리 방법에 대해 많은 연구가 활발히 진행되고 있다. 본 논문에서는 논리 함수의 내부 표현으로서 이분결정도를 채용하고 이를 객체지향적으로 구현하여 쉽게 논리함수를 표현할 수 있는 논리함수 처리 시스템을 제안한다. 이분 결정도의 크기를 줄이기 위해서, 종래의 어드레스 공간의 축소를 초래하는 속성비트 수법 대신에, 본 논문에서는 filtering 함수의 함수적인 성질을 응용하여 해와 무관한 불필요한 노드를 제거하는 수법을 포함하고 있다. 그로 인해 어드레스 공간의 증대가 가능하게 되어 제안한 시스템에서는 2^{27}개의 노드까지 사용할 수 있다. 그리고 이 filtering 수법을 비동기 순서회로의 One-Shot 상태할당 문제에 적용해서 이분결정도의 크기를 줄이는 데 효과적임을 실험결과를 통해서 확인한다.

ABSTRACT

Many studies on internal data expression to process logic functions efficiently on computer have been doing actively. In this paper, we propose an efficient logic function manipulation system made on the Objected-Oriented manner, where Binary Decision Diagrams(BDD's) are adopted for internal data expression of logic functions. Thus it is easy to make BDD's presenting combinational problems. Also, we propose a method of applying filtering

*한국항공대학교 항공통신정보공학과
論文番號:97012-0113
接受日字:1997年 1月 13日

function for reducing the size of BDD's instead of attributed bits, and add it to the manipulation system. As a result, the space of address is expanded so that the number of node that can be used in the manipulation system is increased up to 2^{27}. Finally, we apply the implemented system to One-Shot state assignment problems of asynchronous sequential circuits and show that it is efficient for the filtering method to reduce the size of BDD's.

Ⅰ. 서 론

최근 VLSI(Very Large-Scale Integration) 기술의 진보에 따라 대규모의 복잡한 논리회로가 실현가능하게 되어 논리회로 설계 및 합성, 조회 등의 논리설계 지원시스템(CAD)이 널리 사용되고 있다. 효과적으로 논리회로를 설계 및 해석하기 위해서, 논리회로를 표현하고 있는 논리함수를 계산기상에서 효율적으로 처리하기 위한, 논리함수의 성질을 고려한 내부데이타 표현방식은 큰 비중을 차지하고 있다. 또한 논리함수를 다루는 일은 VLSI 설계 시스템 뿐만 아니라, 인공지능 및 조합문제와 같은 정보공학의 많은 분제들에서도 매우 중요시 되고 있다.

논리함수를 계산기상에서 효율적으로 처리하기 위한 중요한 열쇠는, 효율적인 내부데이타 표현방식과 그에 대응하는 적절한 알고리즘이다. 그러나 기존의 논리함수를 표현하고 다루는 수법들에는 진리치표, 카노맵, 해석트리(parse trees), 정규곱-합표현, 정규합-곱표현 등이 있지만, 대규모 문제에 적용할 경우 논리연산의 복잡함, 등가성 판정의 어려움 등, 많은 문제점을 내포하고 있다. 그래서 최근의 연구동향으로, 논리함수를 효과적으로 표현하고 처리하기 위한 연구가 활발히 진행되어 왔으며, 그 중 대표적인 것으로 이분결정도(Binary Decision Diagrams)가 있다. 이분결정도는 논리함수의 그래프적인 표현법으로서, 논리함수을 표현하고 처리하는데 좋은 특성들[1][2][3]을 가지고 있다. 이분결정도의 기본적인 개념은 1978년 Akers[1]에 의해 소개되었으며 효과적인 연산방법(manipulation method)은 1986년 Bryant[2]에 의해 제안되었다. 이분결정도는 진리치표와 같이 입력논리변수 n개에 대해서 항상 2^n개의 기억공간을 필요로 하지 않으며, 함수의 종류에 따라서 필요한 기억공간의 크기는 변한다. 일반적으로 많은 실용적인 함수들이 입력논리변수의 다항식 크기의 이분결정도로 표현가능하다.

본 논문에서는 논리함수의 내부 데이터 표현으로서 이분결정도를 채용하고 이를 객체지향적으로 구현하여 쉽게 논리함수를 표현할 수 있는 논리함수 처리 시스템을 제안, 개발하였다. 그리고 실제 응용에서 이분결정도의 크기가 계산기의 계산한계(기억공간의 한계)에 도달하게 되어, 계산이 불가능한 경우가 발생하여, 원하는 결과를 얻어내지 못하는 경우가 있다. 종래에는 이 문제를 해결하기 위해서 이분결정도를 컴팩트하게 압축하는 속성비트를 사용하였다. 그러나 이 수법은 어드레스 공간의 축소를 초래하여 사용가능한 노드수를 감소시키는 요인이 되고 있다. 본 논문에서는 filtering 함수를 응용해서 이분결정도의 크기를 감소시키는 방법을 제안하고 이 filtering 수법을 연산의 형태로 논리함수 처리 시스템 포함시켰다. 그 결과 어드레스 공간의 증대가 가능하게 되어, 제안된 시스템에서는 2^{27}개의 노드까지 표현할 수 있게 되어서 대규모의 문제에 적용가능하다. 또한 제안된 filtering 수법의 효과를 확인하기 위해서, 비동기 순서회로의 One-Shot 상태할당 문제에 적용하였다. One-Shot 상태할당 문제는 일반적인 상태할당 문제와는 달리, 비동기순서회로의 고유의 문제(race와 hazard)를 고려한 상태할당 문제이다. 이 상태할당문제는 하나의 상태에 다수의 이진벡터가 할당되며, 할당되는 이진벡터는 인접한 상태에 할당된 이진벡터를 고려해야 만하는 등, 조합문제중에서도 pspace이상의 복잡도를 갖고 있는 어려운 문제중의 하나이다. 실험결과를 통해서 제안된 filtering 수법이 이분결정도의 크기를 효과적으로 감소시키는 방법임을 보인다.

Ⅱ. 이분 결정도

논리함수를 표현하는 방법인 진리치표, 카노맵, 정규합-곱 표현, 정규곱-합 표현이 입력 논리변수 n에 대해서 $O(2^n)$의 기억공간을 필요로 하면서도 논리연산등에 비효율적이지만, 논리함수를 그래프적으로

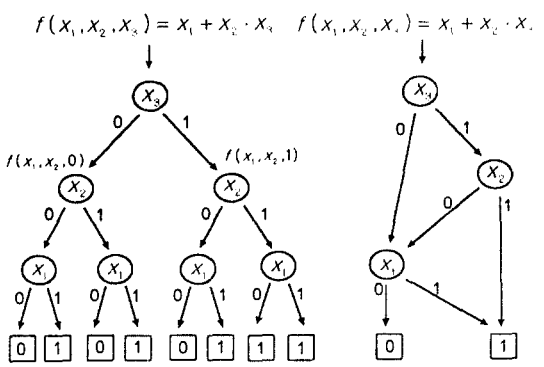
표현한 이진결정도는 이에 반해 다음과 같은 특징이 있다.

(1) 같은 그래프를 중복해서 생성하지 않으므로, 함수의 등가성 판정이 노드를 가리키는 포인터의 일치 판정으로 가능하다.

(2) 그래프의 공유에 의해, 불필요한 기억량을 줄이는 것이 가능하다.

(3) 논리함수의 연산이, 포인터의 연산 등에 의해 이루어므로 고속이다.

본 논문에서는 n개의 논리변수 $n_1, n_2, \cdots, x_n$을 가진 논리함수를 $f(x_1, x_2, \cdots, x_n)$으로 표시한다. Binary Decision Tree(BDT)는 두 형태의 노드(node), 즉 단말노드(terminal node)와 비단말노드(nonterminal node)을 가지는 이진 트리(binary tree)이다(그림 1(a)). 단말노드는 0 혹은 1로 할당되어 있고, 각각은 논리 함수값 0과 1을 나타낸다. 비단말노드는 입력 변수 x_1, $x_2, \cdots, x_n$ 중의 하나가 할당되어 있다. 모든 비단말노드는 0 혹은 1로 표시된 2개의 edge을 갖고, 각 edge는 노드에 할당된 변수의 값에 대응한다.



a) Binary Decision Tree (b) 이분결정도

그림 1. Binary Decision Tree와 이분결정도
Fig. 1 Binary Decision Tree and 이분결정도

함수 $f(x_1, x_2, \cdots, x_n)$의 입력 변수 $x_i (1 \le i \le n)$를 0 혹은 1로 대치하는 것을 함수 $f(x_1, x_2, \cdots, x_n)$의 제한(restriction)이라 한다. 샤논의 전개를 이용해서, 변수 x_i에서의 함수 f를 다음과 같이 표현할 수 있다.

$$f(x_1, x_2, \cdots, x_i, \cdots, x_n) = x_i f(x_1, x_2, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) + \bar{x}_i f(x_1, x_2, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)$$

논리함수 $f(x_1, x_2, \cdots, x_n)$에 대해서 위의 샤논의 전개를 재귀적으로 반복함으로써, 논리함수 $f(x_1, x_2, \cdots, x_n)$에 대한 BDT가 얻어진다. 한 예로 논리함수 $f(x_1, x_2, x_3) = x_1 + x_2 x_3$를 나타내는 BDT를 그림 1(a)에 나타낸다.

이분결정도는 다음의 변형규칙 (1), (2), (3), (4)이 더 이상 적용되지 않을 때까지 BDT에 적용함으로써 얻어지는 directed acyclic graph이다.

(1) 비단말노드의 0 edge와 1 edge가 함께 동일한 단말노드를 가리키면 그 비단말노드를 제거한다.

(2) 동형인 부분 그래프를 공유한다.

(3) 비단말노드의 0 edge와 1 edge 모두가 똑같은 비단말노드를 가리키면 그 비단말노드를 제거한다.

(4) 단지 2개의 단말노드, 0 단말노드와 1 단말노드만 남겨둔다.

위의 변형규칙을 적용해서 그림 1(a)에 있는 BDT를 이분결정도로 변형한 것이 그림 1(b)에 있다. 그리고 위의 변형규칙을 통해서 얻어진 이분결정도는 변수의 순서가 고정되어 있을 때 논리함수에 대한 canonical 형태를 제공하며 이를 Reduced Ordered 이분결정도(RO이분결정도)라 한다. 이 특성은 실제 응용에서 매우 중요하다. 이 특성으로부터 두 개의 논리함수에 대한 등가성 판정은 단지 그들 RO이분결정도의 동형성만 검사하면 된다. 이분결정도에 관련된 설명은 RO이분결정도에 기반을 두고 있으며 본 논문에서는 간편함을 위해서 RO이분결정도를 이분결정도라 표기한다.

일반적으로 많은 실용적인 함수들을 입력변수의 다항식 크기의 이분결정도로 표현할 수 있다. 그리고 함수에 대한 논리연산의 계산량이 연산되고 있는 함수들의 그래프의 크기의 곱에 의해 제한된다.

함수 f와 g를 표현하는 이분결정도가 주어졌을 때, binary logic operation, $f \langle op \rangle g$의 결과를 나타내는 이분결정도를 생성하는 알고리즘은 [2]에서 자세하게 설명하고 있다.

Ⅲ. 이분결정도의 객체지향적 구현

3.1 데이터 구조

본 논문에서 이분결정도의 모든 노드들은 세 개의 기본적인 데이터, 즉 입력 변수의 인덱스, 0 edge가 가리키는 노드의 포인터, 1 edge가 가리키는 노드의 포인터를 가진다. 몇 개의 추가적인 데이터와 카운터가 노드의 효율적인 관리를 위해서 노드 구조에 부가되고 있다. 본 논문에서는 이 노드를 다음과 같은 Class로 표현한다.

```
Class BDDNode {
    private:
        bddp f;
        ..
        ..
    public:
        ..
     노드에 대한 여러 가지 기능 함수들
        ..
}
```

본 논문의 논리함수 처리 시스템에서 모든 노드들은 주메모리상에 하나의 테이블에 저장된다. 위 class에서 f는 노드에 대한 포인터로서 노드가 저장 되어 있는 테이블상의 실제 위치를 가리키고 있다. 그리고 f에 부정 edge 기능, 입력 반전 edge 기능 등을 나타내는 속성비트들을 포함하여 부정연산의 계산시간과 이분결정도의 노드를 줄이는데 효과를 가져올 수 있도록 하였다. f로부터 노드의 테이블상의 실제 위치를 알아내어 그 노드의 입력변수의 레벨, 0 edge 포인터, 1 edge 포인터를 알아낼 수 있다. 본 논리함수 처리 시스템은 기본 데이터의 크기가 32비트인 계산기에서 사용하도록 설계되었고, f의 구체적인 데이터 구조는 그림 2와 같다.

노드의 수를 줄일 수 있는 수법들이 활발히 연구 발표되고 있는데, 본 논문에서는 노드를 줄이는 효과와 구현의 복잡도를 고려해서 채택했다. 그 이유는 본 논문에서 노드의 개수를 줄이는 수법을 제안하고 있는 데, 제안된 방법은 속성 비트를 통해서 이분결정도를 컴팩트하게 압축하는 수법이 아니고, 논리함수의 특징을 이용해서 불필요한 노드를 제거하는 수법이다. 따라서 이용 가능한 노드의 개수를 확대하기 위해 꼭 필요한 속성비트만을 채용했다.

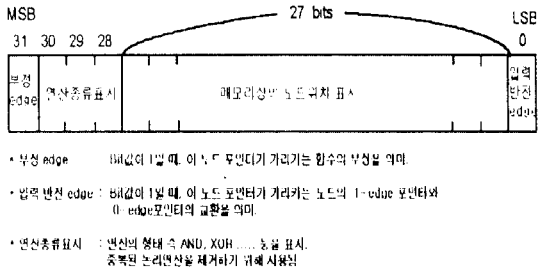


그림 2. 노드 포인터의 데이터 구조
Fig. 2 Data structure of node pointer

이분결정도에서 동형인 부분 그래프는 공유되어야 한다. 이 특성은 모든 노드들을 해쉬 테이블에 기록함으로써 유지된다. 새로운 노드를 만들기 전에 배번 해쉬 테이블을 조사해서 만약 그 노드와 똑같은 노드가 존재하면 그 노드를 만들지 않으며 이미 존재하는 노드를 공유해서 사용한다. 또한 연산결과 테이블, 즉 임의의 두 함수의 연산과 그 연산 결과 생긴 함수를 해쉬 테이블에 저장해 둠으로써 임의의 두 함수에 대한 똑같은 연산이 수행하지 않고 재이용하도록 했다.

3.2 이분 결정도의 생성

본 논문에서는 이분결정도를 생성하는 과정에서 기존 이분결정도 패키지에서 사용하는 방법을 개선하였다. 예를 들어 기존의 이분결정도 패키지에서는 함수 $f = x_1x_2x_3x_4 + x_5x_6 + x_7$을 이분결정도로 생성하는 데는 아래와 같은 방법을 사용해야만 한다.

$Andx1x2 = bddand(x_1, x_2)$;
$Andx3x4 = bddand(x_3, x_4)$;
$Andx1x2x3x4 = bddand(Andx1x2, Andx3x4)$;
$Notx5 = bddnot(x_5)$;
$AndNOtx5x6 = bddand(Notx5, x_6)$;
$Or = bddor(Andx1x2x3x4, AndNotx5x6)$;
$f = bddor(Or, x_7)$;

그러나 본 논문에서는 논리함수 처리 시스템의 구현에 객체지향적인 수법을 도입함으로 해서, 즉 클래스 개념, 생성자(constructor)와 소멸자(destructor), 연산자 및 함수의 overload를 이용해서, 논리함수 $f = x_1x_2x_3x_{41} + x_5x_6 + x_7$는 다음과 같은 방법으로 이

분결정도가 생성된다.

$f = x_1 * x_2 * x_3 * x_4 + (\sim x_5) * x_6 + x_7;$

위와 같이 본 논문의 논리함수 처리 시스템은 논리함수를 쓰는 방식으로 이분결정도를 생성시킬 수 있다. 즉 기존 이분결정도 패키지의 사용법상의 어려움을 해결해 주며 쉽게 논리 함수를 이분결정도로 생성할 수 있게 해준다.

3.3 메모리 관리

본 논문의 논리함수 처리 시스템은 기본적인 데이터의 크기가 32비트인 계산기에서 사용할 수 있도록 설계되었고, 각 노드에 대해서 약 32바이트의 메모리를 필요로 한다. 본 논문의 이분결정도 패키지는 2^{27}개의 노드를 표현할 수 있다. 이분결정도의 연산에서 각 부분적인 연산에 대한 결과를 나타내는 많은 이분결정도가 일시적으로 생성되는데, 이와같은 일단 사용되고 난 이분결정도를 효율적으로 처리하는 것이 필요하다. 만약 이분결정도가 다른 이분결정도와 함께 sub-graph를 공유한다면 그 sub-graph는 제거할 수 없다. 그래서 본 논문의 논리함수 처리 시스템에서는 공유 카운터 기법을 이용하여 노드가 공유될 때는 그 노드의 공유카운터 수를 증가시키며 불필요하게 될 때는 공유카운터 수를 감소시킨다. 그래서 그 노드의 공유카운터 수가 0으로 되었을 때만 그 노드를 메모리에서 해방시키는 것이다. 그러나 노드의 공유카운터 수가 0으로 된 바로 그 시점에서 그 노드를 해방시키는 것이 아니라 일단 보존해 두었다가 나중에 재이용을 기다리는 것이다. 그래서 기억량이 한계에 도달했을 때 공유카운터 수가 0인 노드를 한꺼번에 해방한다. 노드를 해방할 때에는 전 노드를 차례로 조사해서 공유카운터 수가 0인 노드를 모두 해방해서 다시 그들을 모아서 새로운 빈 노드로서 사용하는 Garbage Collection 방법을 실현하고 있다. 이 방법에 의해서 바로 다음에 필요하게 되는 노드가 해방되는 것을 피할 수 있다. 또한 Garbage Collection실행 이외에는 노드가 해방되지 않으므로 연산결과 테이블을 효과적으로 이용할 수 있다.

Ⅳ. 이분결정도의 크기를 줄이는 수법

본 논문에서는 filtering함수를 응용하여, 조합문제를 표현하고 이분결정도로 부터 해와 관련이 없는 불필요한 노드를 제거함으로 해서 이분결정도의 크기를 줄이는 수법을 제안한다. 우선, filtering함수에 대해서 간단히 설명한다.

[정의 1] n변수의 논리함수 $f(x_1, x_2, ..., x_n)$가 그것의 입력 변수 중 임의의 변수를 서로 치환하더라도 불변이면 이 때 함수 f를 symmetric (totally symmetric)이라 한다. ■

그리고 함수 f가 집합 $\{x_1, x_2, ..., x_n\}$의 부분집합 $\{x_i, x_j\}$에서 변수 x_i와 x_j를 교환해도 함수 f가 변하지 않으면 함수 f를 partially symmetric이라 한다. 본 논문에서는 단지 totally symmetric에 대해서만 관심을 갖는다.

Symmetric 함수 $f(x_1, x_2, ..., x_n)$는 입력 변수중 $a_i (0 \le a_i \le n)$개가 1일 때만 f의 값이 1이 되도록 하는 수의 집합 $\{a_1, a_2, ..., a_k\}$에 의해서 기술되며 이때 그 수의 집합 $\{a_1, a_2, ..., a_k\}$을 a-numbers라 부른다. 그래서 Symmetric 함수 f는 $S_{a_1, a_2, ..., a_k}(x_1, x_2, ..., x_n)$으로 표시하는 경우가 많다. 예로서 함수 $f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}\,x_3 + \overline{x_1}\,x_2\,\overline{x_3} + x_1\,\overline{x_2}\,\overline{x_3}$은 세개의 입력변수 중 임의의 변수 하나가 1일 때만 값 1을 가진다. 그래서 이 함수는 $S_1(x_1, x_2, x_3)$로 표시된다. 비슷하게 symmetric 함수 $S_{1,3}(x_1, x_2, x_3)$은 $f(x_1, x_2, x_3) = x_1\,x_2\,x_3 + \overline{x_1}\,\overline{x_2}\,x_3 + \overline{x_1}\,x_2\,\overline{x_3} + x_1\,\overline{x_2}\,\overline{x_3}$이다.

[정의 2] n변수 함수 $f(x_1, x_2, ..., x_n)$가 symmetric 함수 $S_{0, 1, 2, ..., a_k}(x_1, x_2, ..., x_n)$ $(0 \le a_k \le n)$일때 함수 f를 입력 vector의 Hamming weight가 a_k 이하 일 때 함수값이 1이 되는 light-weight pass filtering 함수$(=T_n^k)$라 한다. 비슷하게 입력 vector의 Hammming weight가 a_k 이상 일 때 함수값이 1이 되는 즉 $S_{a_k, a_{k+1}, ..., a_n}(x_1, x_2, ..., x_n)$인 함수 f를 heavy-weight pass filtering function$(=\overline{T_n^{k-1}})$이라 한다. 또한 함수 f가 입력 vector의 Hamming weight가 $a_k, a_{k-1}, a_{k-2}, ..., a_{k-w}$일때만 함수 f의 값이 1이 되는 즉 $S_{a_k, a_{k-1}, a_{k-2}, ..., a_{k-w}}(x_1, x_2, ..., x_n)$인 함수 f를 bandwidth w를 갖는 band-weight pass filtering 함수$(=T_n^{k,w})$라고 한다. ■

논리함수 $f(x_1, x_2, x_3, x_4)$가 입력 벡터의 Hamming weight가 2이하인 모든 입력 vector에 대해서 $f(x_1, x_2, x_3, x_4)$가 1이 되는 T_4^2라면, 함수 $f(x_1, x_2, x_3, x_4)$는 다음과 같이 표현된다.

$$f(x_1, x_2, x_3, x_4) = T_4^2 = x_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3x_4 + \ldots + x_1\bar{x}_2x_3x_4 + \bar{x}_1x_2x_3x_4 + \ldots + x_1x_2x_3x_4$$

그림 3은 각 filtering 함수의 예를 보여주고 있다. 본 논문에서 처음으로 이분결정도를 이용한 논리함수 처리 시스템에 filtering 함수를 응용한 수법을 구현하는 다음의 함수들을 포함했다.

- $T_n^k = ltpassfilter(n, k)$
- $\overline{T_n^k} = \sim ltpassfilter(n, k)$
- $T_n^{k,w} = bdpassfilter(n, k, w)$
- $f \cdot T_n^k = f * ltpassfilter(n, k)$
- $f \cdot \overline{T_n^k} = f * (\sim ltpassfilter(n, k))$
- $f \cdot T_n^{k,w} = f * bdpassfilter(n, k, w)$

어떤 임의의 n변수 함수 f에 대해서 우리가 원하는 해가, positive literal 갯수가 upper bound μ를 넘지 않는 입력 vector에 대해서만 고려의 대상이 된다고 가정하자. 즉, upper bound μ를 초과하는 positive literal들을 포함하고 있는 입력 vector는 고려하지 않는다고 하면, 함수 f를 이분결정도로 표현한 경우, 단말노드 "1"로 가는 path중에서 upper bound μ를 초과하는 단말노드 "1"로 가는 path들을 f를 표현한 이분결정도로부터 제거하기 위해 우리는 Light-weight pass filtering 함수 T_n^μ를 다음과 같이 적용해서,

$$f \cdot T_n^\mu$$

upper bound μ를 초과하는 positive literal의 수를 가지고 있는 path들을 이분결정도로부터 제거시킬 수 있다. 따라서 언제나 해와 관련이 있는 노드만을 갖고 문제를 표현하므로 항상 이분결정도의 크기를 최적산산출할 수 있어 효율적으로 문제를 해결할 수 있게 된다.

$x_4\ x_3\ x_2\ x_1$	f	T_4^2	$\overline{T_4^2}$	$T_4^{2,1}$	$f \cdot T_4^2$	$f \cdot \overline{T_4^2}$	$f \cdot T_4^{2,1}$
0 0 0 0	0	1	0	0	0	0	0
0 0 0 1	1	1	0	1	1	0	1
0 0 1 0	0	1	0	1	0	0	0
0 1 0 0	0	1	0	1	0	0	0
1 0 0 0	1	1	0	1	1	0	1
0 0 1 1	1	1	0	1	1	0	1
0 1 0 1	1	1	0	1	1	0	1
1 0 0 1	1	1	0	1	1	0	1
0 1 1 0	0	1	0	1	0	0	0
.	1	1	0	1	1	0	1
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	1	1	0	1	1	0	1
1 1 0 0	1	1	0	1	1	0	1
0 1 1 1	1	0	1	0	0	1	0
.	0	0	1	0	0	0	0
.	1	0	1	0	0	1	0
.	1	0	1	0	0	1	0
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
1 1 1 0	1	0	1	0	0	1	0
1 1 1 1	1	0	1	0	0	1	0

그림 3. filtering 함수의 예
Fig. 3 Examples of filtering function

V. 순서회로 설계에의 응용

비동기 순서회로의 상태할당은 critical race가 상태변수간에 존재하지 않도록 binary vector를 각 상태에 할당하는 것이 필요조건이다. One-Shot 상태할당은 각 내부-상태 천이에 대해서 단지 하나의 상태변수만 변하도록 상태할당을 하는 것이다. 예를 들어, 상태 S_i에서 $S_j (i \neq j, S_i, S_j \in$상태의 집합)로의 천이가 있다고 해보자. Binary vector 001과 110을 상태 S_i에 할당하면 상태 S_j에 할당할 수 있는 binary vector들은 011과 111이 있다. One-Shot 상태 할당은 구현된 순서회로의 동작속도의 고속화와 내부 소비전력을 최소화 할 수 있는 방법의 하나이다. 본 논문에서는 비동기 순서회로의 One-Shot 상태할당의 최소해를 얻어내는 방법으로 propositional calculus을 minimal One-Shot 상태할당을 표현하기 위해 사용하고, 이분결정도를 propositional formulas의 내부표현으로서

사용한다.

Propositional calculus에 의해 One-Shot 상태할당 문제를 풀기 위해 사용되는 논리변수를 소개한다. 여기서 S=n 상태의 집합, B=binary vector, $(b_0,\ldots, b_i,\ldots,b_n)$, $b_i\in\{0, 1\}$의 집합이라 하면, Binary vector $b\in B$가 상태 $X\in S$에 할당될 때 논리변수 $P_b^X\in\{0, 1\}$을 다음과 같이 정의한다.

$$P_b^X=\begin{cases}1, & \text{만약 } \textit{binary vector}, b\in B\text{가 상태}\\ & X\in S\text{에 할당되면}\\ 0, & \text{그렇치 않으면}\end{cases}$$

예로서 만약 다음과 같이 주어지는 propositional formula Q가 있다면 $Q=1$은 binary vector $b\in B$가 상태 $X_i\in S$에 할당되고 임의의 다른 상태 $X_j\in S(i\neq j$인 모든 j에 대해서)에 binary vector b가 할당되지 않는 것을 표현한다.

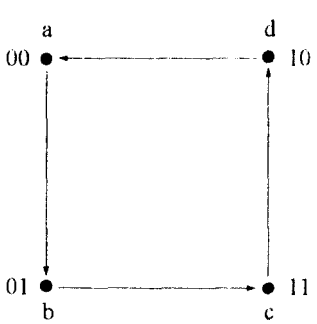
$$Q=P_b^{X_i}\cdot\prod_{i\neq j}\overline{P_b^{X_j}}$$

Propositional calculus에 의해서 One-Shot 상태할당을 표현할 경우 다음의 두 조건을 표현해야 한다.

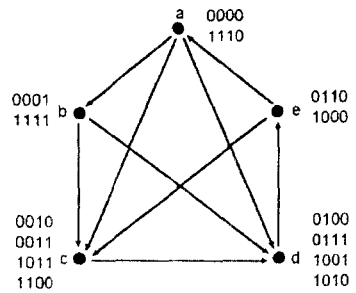
(1)어떤 상태에 할당된 binary vector는 임의의 다른 상태에 할당 되지 말아야 한다는 일반적인 상태할당 조건.

(2)어떤 상태에 할당된 binary vector는 그 상태와 인접한 상태에 Hamming distance가 1인 binary vector를 가져야 한다는 One-Shot 상태할당의 고유조건.

예를 들어 그림 4(a)에서 binary vector 00이 상태 a에 할당된다면 이 때의 One-Shot 상태할당, $P_{a,\,00}$은 다음과 같이 얻어진다.



(a) 단순한 One-Shot상태



(b)복잡한 One-Shot 상태 할당

그림 4. One-Shot 상태할당의 예

Fig. 4 Examples of the One-Shot state assignment

$$P_{a,\,00}=\overline{P_{00}^a}+(P_{01}^b\,\overline{P_{01}^a}\,\overline{P_{01}^c}\,\overline{P_{01}^d}+P_{10}^b\,\overline{P_{10}^a}\,\overline{P_{10}^c}\,\overline{P_{10}^d})$$

그리고 위와 같이 모든 propositional formula $P_{i,\,j}$ $(i\in S,\ j\in B)$를 만들고 모든 $P_{i,\,j}$의 곱을 P라 하자:

$$P=\prod_{all(i,\,j)}P_{i,\,j}$$

일반성의 손실없이 binary vector 00을 상태 a에 할당하고 상태 a에 할당한 binary vector 00과 Hamming distance가 1인 binary vector 01이 상태 a에 binary vector 00을 할당한 결과로서 상태 b에 할당된다면 이 proposition은 다음과 같이 Q로 표시된다.

$$Q=P_{00}^a\,\overline{P_{00}^b}\,\overline{P_{00}^c}\,\overline{P_{00}^d}\,P_{01}^b\,\overline{P_{01}^a}\,\overline{P_{01}^c}\,\overline{P_{01}^d}$$

그러면 그림 4(a)에 대한 One-Shot 상태할당은 $O=P\cdot Q$로 표시된다. 상태변수의 수를 최소로 가지는 최소해는 상태변수의 개수를 하나씩 증가시켜 갈 때 모순이 아닌 O가 첫번째로 발견될 때 얻어진다. Binary vector 수의 관점에서 최소해는 $O=1$일 때 최소의 positive literal의 수를 가지는 곱항에 포함된 positive literal의 집합이다. 본 논문에서는 O의 내부 표현으로서 이분결정도를 사용한다. 이 경우 그림 4(a)에 대한 One-Shot 상태할당의 최소해는 노드 "1"로 가는 path 중에서 최소의 positive literal의 수를 가지는 path에 포함된 positive literal의 집합이다. 그림 4(b)에 One-Shot 상태할당이 일반적인 상태할당과 구별됨을 보이기 위한 예를 제시한다.

본 논문에서는 비동기 순서회로의 One-Shot 상태할당문제를 표현한 propositional formulas의 내부 표현으로서 사용된 이분결정도의 크기를 줄이기 위해 4절에서 소개한 filtering 수법를 적용하였다. 먼저 filtering 수법를 적용하기 전에 최소해에 포함되는 binary vector의 갯수에 대한 upper bound를 계산할 필요하다. upper bound에 대한 구체적인 설명은 [4]을 참조하기 바란다. μ를 최소해에 대한 upper bound라 하자. One-Shot 상태할당의 내부 표현인 이분결정도는 이분결정도의 크기에 비례하는 계산시간안에 최소해를 발견해 낼 수 있다. 그러나 고려해야 할 binary vector의 수가 많아지면 이분결정도의 크기는 커지게 되고, 제한된 기억공간때문에 이분결정도를 이용해서 propositional formulas를 표현하기가 불가능하게 된다. 그러나 노드 "1"로 가는 그리고 μ보다 더 많은 positive literal의 수를 가지는 path는 최소해가 발견할 때 명백히 고려의 대상이 되지 않는다. 따라서 이들 path들은 이분결정도로 부터 제거될 수 있다. 이상으로 우리는 이분결정도의 크기를 줄이기 위해 filtering 함수 T_n^{μ}를 사용한다. 여기서 T_n^{μ}는 입력 vector의 Hamming weight가 μ 이하일 때 함수값이 "1"이 되는 light-weight pass filtering 함수이다. 그러면 propositional formula $P_{i,j}$을 표현하는 이분결정도를 만드는 것을 $P_{i,j} \cdot T_n^{\mu}$을 표현하는 이분결정도를 만드는 것으로 대치함으로써 고려할 필요가 없는 path는 제거되게 된다. T_n^{μ}을 표현하는 이분결정도는 $(\mu+1)(m-\mu)$개의 노드를 가지므로 (m은 논리변수의 수), 이 filtering 함수를 곱하는 일은 추가적인 계산 없이 수행될 수 있다.

본 논문에서는 상태할당문제를 그림 4처럼 directed graph로 모델링해서 몇몇 directed graph들에 대해서 상태할당을 해 보았다. 그 결과가 표1에 있다. 표1은 SUN SPARC 20 WORKSTATION(256Mbytes)에서 실험된 결과이다. 표1에 나타나는 것과 같이 filtering 수법이 이분결정도의 노드의 수를 감소시키는 데 효과적임을 알 수 있다. 특별한 경우, dg505(vertex:5개, edge:5개)에 대해서는, 노드의 개수가 97.66% 감소하였고, 계산시간도 72.30%로 감소하였다.

표 1. 실험결과
Table 1. Experimental results

이름	#s	#b	No-filtering		filtering	
			#Node	cpu(sec)	#N L	cpu(sec)
dg303	3	6	115	2.89	65	2.9
dg404	2	4	12	2.63	16	2.64
dg405	3	8	45	3.16	45	3.22
dg406	3	8	32	3.15	32	3.25
dg505	4	10	270667	225.75	6321	62.53
dg506	4	10	66479	162.79	958	55.88
dg509	4	14	368	16.04	98	37.34
dg609	4	9	13605	36.27	230	70.22

[Notes] 이름:dg405는 4개의 vertex와 5 edge를 가진 directed graph이다. #s:상태변수의 수. #b:binary vector의 수. #Node:논리 변수 P_j^i를 사용하는 노드의 최대수. #N L: Light-weight pass filtering 함수를 사용한 경우의 노드의 최대수. cpu:초.

Ⅵ. 결 론

본 논문에서는 내부 데이터 표현방식으로 사용한 이분결정도를 객체지향적으로 구현하여 쉽게 논리함수를 표현할 수 있도록 한 논리함수 처리 시스템을 개발하였다. 본 논문에서는 이분결정도의 크기를 줄이는 방안으로서 종래의 어드레스 공간의 축소를 초래하는 속성비트 대신에 filtering 함수를 응용하여, 불필요한 노드를 제거함으로 해서 언제나 최적의 이분결정도를 유지하는 수법을 제안했다. 그로 인해 어드레스 공간이 증대하여 2^{27}개의 노드까지 표현가능하게 되었다. 또한 구현한 논리함수 처리시스템을 비동기 순서회로의 One-Shot 상태할당 문제에 적용하였다. 실험 결과에서 볼 수 있듯이, filtering 수법을 적용하여 표현한 이분결정도의 크기는 기존의 이분결정도의 크기보다 매우 작아졌음을 알 수 있다. 즉 filtering 수법이 이분결정도의 크기 감소에 효과적임을 의미한다. 그리고 이 filtering 수법을 논리함수 처리 시스템안에 연산의 하나로 포함시켰기 때문에 일반적인 대규모 조합문제를 해결할 때 이분결정도의 크기를 대폭 감소시킬 수 있을 것이다. 앞으로의 연구과제로는 64비트를 기본 데이터로 하는 계산기상의 효과적인 구현과 각종 테이블을 효과적으로 관리하기 위한 해싱 함수의 개발이다. 해싱 함수에 대한

연구는 암호분야에도 중요시 되고 있다.

참 고 문 헌

1. S.B. Akers, "Binary decision diagram," IEEE Transactions on Computers, C-27(6):509-516, June 1978.
2. R.E. Bryant, "Graph-Based algorithms for boolean function anipulation," IEEE Transactions on Computers, C-35(8):677-691, August 1986.
3. Shin-ichi Minato, "Binary Decision Diagrams and Their Application for VLSI CAD," 박사학위논문, 일본쿄토대학, 12,1994.
4. 권용진, "State Assigments for Asychronous Sequential Circuits Using Graph Techniques and Binary Decision Diagrams," 박사학위논문, 일본쿄토대학, 12, 1993.
5. Ichiro Semba, "Combinatorial algorithms using boolean processing," 박사학위논문, 일본쿄토대학, 9, 1994.
6. 이귀상, 천승환, "이단계 리드뮬러 회로의 최소화를 위한 BDD의 응용," 한국정보과학회 95년 춘계학술발표회 논문집, 501-504, 4월 1995.
7. 권오형, 김종, 홍성제, "다변수 출력 논리회로에서 Binary Decision Diagram 입력 변수 배열," 한국정보과학회 95년 춘계학술발표회 논문집, 477-480, 4월 1995.
8. 장용희, 권용진, "Filtering 함수를 이용한 효율적인 논리함수 처리," 한국정보과학회 96년 추계학술발표회 논문집, 1691-1694, 10월 1996.
9. 장용희, 권용진, "Filtering 함수를 이용한 논리함수 처리 시스템의 구현과 그 응용," 한국항공대학교 부설 전자정보통신연구소 논문지, 제2권 2호, 25-35, 10월 1996.
10. Tony Stornetta and Forrest Brewer, "Implemention of an Efficient Parallel BDD Package," 33rd DAC proceeding, pp. 635-640, 1996.
11. J. V. Sanghavi, R. K. Ranjan, R. K. Brayton and A. Sangiovanni-Vincentelli. "High Performance BDD Package By Exploiting Memory Hierarchy," 33rd DAC proceeding, pp. 629-634, 1996.

권 용 진(Yong Jin Kwon) 정회원

1986년 2월 : 한국항공대학교 항공전자공학과 졸업

1990년 3월 : 일본 쿄토대학 대학원 정보공학과 졸업 (공학석사)

1994년 3월 : 일본 쿄토대학 대학원 정보공학과 졸업 (공학박사)

1994년 3월~현재 : 한국항공대학교 항공통신정보공학과 조교수

※주관심분야 : 논리회로 설계 및 합성, 알고리즘 개발, 정보보안