BDD를 이용한 다단계 리드뮬러회로의 합성

장 준 영† · 이 귀 상††

요 약

본 논문에서는 BDD(Binary Decision Diagrams)를 이용한 다단계 리드뮬러회로 합성 방법을 제시한다. 기존의 다단계 논리 합성 도구인 FACTOR에서는 논리 함수를 입력 분할에 의해 맵형태의 행렬로 표현하고 행렬 연산을 통해 다단계 회로를 합성한다. 이 방법은 논리 합성의 입력으로 맵을 사용하기 때문에 입력 수에 따라 기억 공간이 지수적으로 증가하고 이에 비례하는 연산 시간이 필요하게 되어 대규모 회로에서 잘 동작되지 않는다. 이러한 단점을 해결하기 위해 기존의 방법과는 다른 새로운 시도로서 BDD에 의한 합성 방법을 제시한다. 이 방법은 BDD를 이용하여 논리 함수를 간결하게 표현하므로 합성에 필요한 기억 공간과 연산 시간의 문제가 해결되어 대규모 회로를 쉽게 표현, 처리할 수 있다. BDD 표현에 의해 최선의 패턴을 선택하므로 최소화된 다단계 리드뮬러회로를 구현한다. 본 논문에서 제시한 방법을 사용한 benchmark 회로의 실험 결과, 대부분의 회로에서 기존의 결과[2]에 비해 개선된 결과를 보인다. 특히, 대칭 함수에 대해서는 최적에 가까운 결과를 보인다. 대규모 회로에서 합성 결과를 개선하기 위해 최선의 입력 분할을 고려하므로 기존의 결과보다 개선된 결과를 얻었다.

Synthesis of Multi-level Reed Muller Circuits using BDDs

Chang June Young† · Lee Guee Sang††

ABSTRACT

This paper presents a synthesis method for multi-level Reed-Muller circuits using BDDs(Binary Decision Diagrams). The existing synthesis tool for Reed Muller circuits, FACTOR, is not appropriate to the synthesis of large circuits because it uses matrix(map-type) to represent given logic functions, resulting in the exponential time and space in number of input to the circuits. For solving this problems, a synthesis method based on BDD is presented. Using BDDs, logic functions are represented compactly. Therefore storage spaces and computing time for synthesizing logic functions were greatly decreased, and this technique can be easily applied to large circuits. Using BDD representations, the proposed method extract best patterns to minimize multi-level circuits. Therefore resulting circuits are with AND/XOR gate which form a unique class of multi-level Reed Muller circuits with good performance in area optimization and testability. Experimental results using the proposed method show better performance than those using previous methods[2]. For large circuits of considering the best input partition, synthesis results have been improved.

1. 서 론

디지털 회로를 설계하고 표현하는데 SOP(Sum Of Products) 표현 방법이 많이 이용되어 왔으나 최근 들어 AND/XOR 게이트를 이용한 XOP(eXclusive-sum Of Products) 표현 즉, 리드뮬러 표현과 처리에 많은 관심이 집중되었다. AND/XOR 게이트로 구현된 회

*본 논문은 1995년도 한국과학재단 연구비지원(과제번호:951-0917-119-1)에 의한 결과임.
† 정 회 원:대불대학교 컴퓨터학부
†† 종신회원:전남 대학교 자연 대학 전산학과
논문접수:1995년 10월 2일, 심사완료:1996년 3월 15일

로를 리드뮬러회로(Reed-Muller circuits)라 한다. 논리 표현의 간결성, 내재된 높은 검증도의 장점에도 불구하고 리드뮬러회로는 많이 이용되지 않았다[1, 2]. 왜냐하면 리드뮬러 표현이 동등한 SOP 표현보다 많은 비용이 들었다. 그러나 FPGA(Field Programmable Gate Array)와 같은 집적 기술의 발달로 인하여 XOR 게이트가 다른 게이트와 비교하여 가격과 속도면에서 동등한 구현이 가능하게 되었다. 따라서 산술, 통신, 에러 제어, 동기화 시스템 테스팅회로등에서는 XOP 표현을 이용한 리드뮬러회로를 이용하고 있고 이러한 회로를 합성하기 위한 알고리즘에 관한 연구가 진행되고 있다[1, 2, 3, 4]. 이 단계 리드뮬러 합성은 1950년대 이후 계속 연구되고 있다. 그러나 다단계 리드뮬러회로 합성에 대한 연구는 아직까지 초기 상태이다. 최근에 다단계 리드뮬러회로를 합성하는데 MISII와 같은 기법을 이용한 Saul[2]의 결과가 현재 문헌에 나타난 유일한 것이다.

최근에 제안된 입력 분할에 의한 합성 도구인 FACTOR[8, 9]에서는 입력 함수를 입력 분할(input partition)에 의해 진리표 형태의 맵(map)으로 표현하고, 이를 입력으로 하여 행렬 연산(matrix operation)을 통해 다단계 회로를 합성한다. 그러나 FACTOR는 입력 수가 증가하므로 기억 공간이 지수적으로 증가하고 이에 비례하는 연산 시간이 필요하게 되어 대규모 회로를 처리하는데 적합하지 않다. 이러한 문제를 해결하기 위해 BDD를 이용한 다단계 리드뮬러회로 합성 방법을 제시한다. 이 방법은 논리 합성의 시작을 지수적인 기억 공간이 필요한 맵대신에 BDD를 이용하여 간결하게 표현하고, 이를 FACTOR의 패턴 추출(pattern extraction) 과정인 행렬 연산에 적용한다. 따라서 이 방법은 합성에 필요한 기억 공간과 합성 시간이 감소하게 되어 대규모 회로를 쉽게 처리할 수 있다.

2장에서는 논리 함수를 표현하기 위한 자료 구조인 BDD에 대해서 설명하고 3장에서는 FACTOR의 행렬 연산에 의한 패턴 추출 과정에 대해서 기술한다. 4장에서는 FACTOR의 패턴 추출 과정인 행렬 연산을 BDD 연산에 적용하여 다단계 리드뮬러회로를 최소화하는 과정을 설명한다. 또한 5장에서는 합성 결과를 개선하기 위한 최선의 패턴 선택 방법에 대해서 기술하고, 6장에서는 benchmark 회로에 대한 실험 결과에 대해서 설명한다.

2. BDD에 의한 논리 함수 표현

논리 함수를 표현하고 조작하는 알고리즘에서 논리 함수의 표현은 매우 중요하다. 일반적으로 동일한 함수를 표현하기 위한 논리 함수의 표현은 NP-complete 문제로 알려져 왔다[7]. 진리표로 논리 함수를 표현할 경우에 진리표의 각 항목이 함수의 미치는 모든 효과를 고려해야 한다. 전통적인 표현 방법에서는 n개의 입력 변수를 갖는 논리 함수를 표현하는데 $O(2^n)$개의 공간이 필요한 정규 SOP(Sum of Products) 형태나 정규 POS(Product of Sums) 형태 또는 진리표, 카르노 맵(Karnough map)을 사용하였다. 반면에 다른 표현 방법으로 논리 함수를 표현하면 지수적 기억 공간이 필요한 전통적인 표현 방법보다 작은 기억 공간이 필요할 수 있다. 따라서 전형적인 논리 함수 표현 방법을 이용하면 상당히 많은 기억 공간과 처리 시간이 필요하게 되므로, 이를 해결하기 위해 그래프 이용한 새로운 표현 방법인 BDD가 제시되었다. 대표적인 BDD의 종류에는 ROBDD[6]과 ROCBDD[10]이 있다.

2.1 ROBDD(Reduced Ordered Binary Decision Diagram)

ROBDD[6]은 함수 그래프의 부분 그래프가 중복되지 않도록 간략화하고, 루트 노드에서 단말 노드까지의 경로에 나타나는 변수들이 정해진 순서를 만족한다. ROBDD는 임의의 함수에 대해서 유일한 표현을 제공하므로 다음과 같은 장점을 갖게 된다.

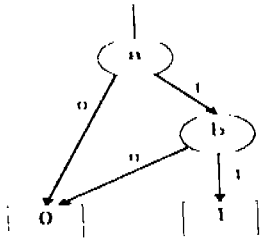
(1) ROBDD는 정규 표현이므로 함수의 비교가 쉽다.

(2) 대부분의 논리 합성 문제는 함수 표현의 크기에 영향을 받는데 ROBDD는 이상적인 크기를 갖는다. 즉, 대부분의 경우 입력 변수의 증가에 따라 그 함수에 대응하는 BDD의 크기가 비지수적으로 증가한다.

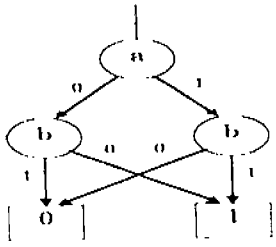
(3) 연산 시간은 ROBDD의 크기에 비례한다.

2.2 ROCBDD(ROBDD with Complemented Edges)

ROCBDD[10]은 ROBDD에 보수 간선을 첨가한 것으로 BDD 내의 노드 수를 ROBDD에 비해 7% 정



(a) $f1 = a \cdot b$



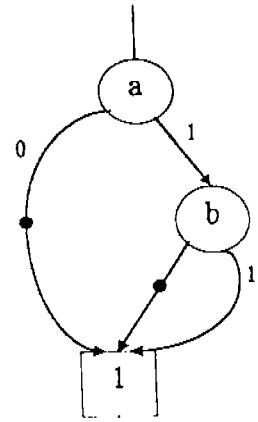
(b) $f2 = a \oplus b$

(그림 1) ROBDD의 표현
(Fig. 1) Representation of ROBDD

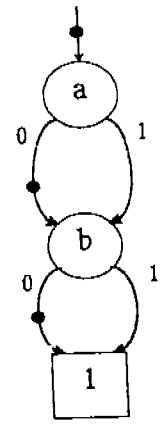
도 줄이는 효과가 있으며 강한 정규성등의 장점이 있는 것으로 보고되었다. 이것은 보수 관계가 있는 부분 그래프를 보수 간선으로 축소한 것이며 정규성은 보수 표현을 ELSE 간선으로 제한함으로써 보장한다. 보수 간선이란 BDD 간선의 일부에 점(•)을 추가하여 표시하며, 임의의 루트로부터 단말노드까지의 경로, 또는 그에 해당하는 큐브가 0과 1중 어떤 단말노드를 선택할 것인지를 결정한다. 논리 함수 $f1 = a \cdot b$과 $f2 = a \oplus b$를 ROCBDD로 표현하면 (그림 2)와 같다.

ROCBDD는 임의의 경로가 짝수 개의 보수 간선을 가지면 1이고 홀수 개의 보수 간선을 가지면 0으로 해석한다. 예를 들어 (그림 2(b))의 루트에서부터 1 -〉1인 경로는 홀수 개의 보수 간선을 가지므로 0으로 해석된다.

본 논문에서는 다양한 BDD 중에서 가장 적은 기억 공간을 사용하는 ROCBDD를 이용하여 논리 함수를 표현한다.



(a) $f1 = a \cdot b$



(b) $f2 = a \oplus b$

(그림 2) ROCBDD의 표현
(Fig. 2) Representation of ROCBDD

3. FACTOR의 행렬 연산[8]

본 논문에서 제시한 BDD를 이용한 다단계 합성 방법은 FACTOR의 행렬 연산과 밀접한 관계를 가지고 있다. 따라서 먼저 FACTOR[8]의 기본적인 행렬 연산에 대하여 설명한다. FACTOR의 논리 합성의 기본 개념은 회로의 각 부분 회로 사이의 통신복잡도(communication complexity)를 최소화하는 것이다. 이것은 집적회로의 면적의 극대점(upper bound)을 계산하는 데에 회로를 표현하는 논리 함수의 리터럴의 개수보다도 통신복잡도를 이용한다는 점이다. 다시

말하면 회로 면적을 결정하는 것은 통신복잡도라는 점에 착안한 것이다. 통신복잡도는 두개의 어떤 회로가 주어진 작업을 수행하기 위해 정보 교환을 하는데 필요한 최소한의 연결 회선의 개수이다. 이것은 부분 회로로 구분된 회로에 적용될 때, 각 부분 회로들 사이의 회선의 개수를 의미한다. 따라서 FACTOR의 논리 합성은 전체 회로를 좌, 우와 중앙의 부분 회로들로 구분하여 좌, 우와 중앙의 회로 사이의 연결선을 최소화하는 입력 분할을 찾는 것이다. 회로에 대한 함수가 주어지면 이 함수에 대한 행렬을 만들고 행렬을 여러 개의 부분 행렬로 변환한다. 즉, 주어진 함수에서 구해진 행렬(이를 M행렬이라 하자)의 좌우에 단위 행렬을 첨가한다. 따라서 $M=I\cdot M\cdot J$(I, J는 단위 행렬)가 되며 이제 M 행렬을 대각 행렬로 만들어 나간다. 만들어 나가는 과정에서 행렬 M에서 추출된 패턴을 좌, 우측 단위 행렬인 I와 J에 기록한다. 이러한 행렬 연산 과정에서 얻어지는 것은 행렬 M의 차수(rank)이다. 이 차수는 좌, 우와 중앙의 회로 사이의 연결선의 수 즉, 패턴의 수를 나타낸다. 따라서 이 주어진 함수의 차수를 줄임으로써 통신복잡도를 최소화시킨다. 이러한 절차를 예제를 통하여 설명한다.

예를 들어, 주어진 논리 함수가 $f=a'b'c'd'\oplus a'b'cd\oplus abc'd'\oplus abcd$ 이라 하자. 이를 진리표 형식의 행렬로 표현하면 다음과 같다.

ab \ cd	00	01	10	11
00	1	0	0	1
01	0	0	0	0
10	0	0	0	0
11	1	0	0	1

행렬 연산에 이용되는 연산자는 [덧셈 mode 2]와 [곱셈 mode 2]를 사용하는데 이것은 정확히 XOR와 AND 연산과 같다. 행렬 M의 행 연산이 $[R_i(M)=R_i(M)\oplus R_j(M)]$을 수행하면 행렬 I는 열 연산 $[C_j(I)=C_i(I)\oplus C_j(I)]$를 수행하는데 이는 행렬 M에 의해 표현된 논리 함수의 함수적 성질은 그대로 유지된다. $C_i(K)$와 $R_j(K)$는 행렬 K의 i번째 열과 j번째 행을 나타낸다. 비슷하게 행렬 M의 열 연산이 $[C_j(M)=C_i(M)\oplus C_j(M)]$을 수행하면 행렬 J의 행 연산은 $[R_i(J)=R_i(J)\oplus R_j(J)]$를 수행한다. 초기에 주어진 행렬 M과 단위 행렬 I와 J가 행렬 곱의 형태인 $I\cdot M\cdot J$를 구성한다. 행렬 M의 행과 열 연산에 의해 단위 행렬 I와 J가 원래 함수의 성질을 유지하기 위해 연속적으로 변해 간다. 이러한 과정은 다음 두 가지 연산 규칙에 의한다.

규칙 1. M의 $R_i=R_i\oplus R_j$, I의 $C_j=C_i\oplus C_j$
규칙 2. M의 $C_i=C_i\oplus C_j$, J의 $R_j=R_i\oplus R_j$
(C_i는 i번째 열, R_j는 j번째 행을 나타냄)

이러한 규칙을 적용하면 행렬들의 곱 $I\cdot M\cdot J$의 결과는 변하지 않음을 확인해 볼 수 있다.

$$=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \quad (1)$$

$=C_1(I)\cdot R_1(J)\oplus C_1(I)\cdot R_4(J)\oplus C_4(I)\cdot R_1(J)\oplus C_4(I)\cdot R_4(J)$
(C는 column, R은 row를 나타내고, 숫자는 첨자이다.)

$$=\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\cdot[1\ 0\ 0\ 0]\oplus\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\cdot[0\ 0\ 0\ 1]$$

$$\oplus\begin{bmatrix}0\\0\\0\\1\end{bmatrix}\cdot[1\ 0\ 0\ 0]\oplus\begin{bmatrix}0\\0\\0\\1\end{bmatrix}\cdot[0\ 0\ 0\ 1] \quad (2)$$

$$=a'b'\cdot c'd'\oplus a'b'\cdot cd\oplus ab\cdot c'd'\oplus ab\cdot cd \quad (3)$$

식 (1)에서 I 행렬의 각 행과 열은 입력 패턴 (a, b)를 나타내고 J 행렬의 각 행과 열은 입력 패턴 (c, d)를 나타낸다. 또한 식 (2)에서 M 행렬의 각 항은 I 행렬의 열과 J 행렬의 열의 곱으로 나타낸다.

함수 f의 기능을 그대로 유지하면서 M 행렬의 행 연산 $R_1(M)\oplus R_4(M)\rightarrow R_4(M)$와 I 행렬의 열 연산 $C_4(I)\oplus C_1(I)\rightarrow C_1(I)$을 하면 식 (4)와 같이 된다. 여기서 식 (3)과 식 (5)를 비교해 보면 원래 함수 f의 논리식이 분할된 것을 알 수 있다.

$$= \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad (4)$$

$$= C_1(I) \cdot R_1(J) \oplus C_1(I) \cdot R_4(J)$$

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot [1\ 0\ 0\ 0] \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot [0\ 0\ 0\ 1]$$

$$= a'b' \oplus ab) \cdot c'd' \oplus (a'b' \oplus ab) \cdot cd \quad (5)$$

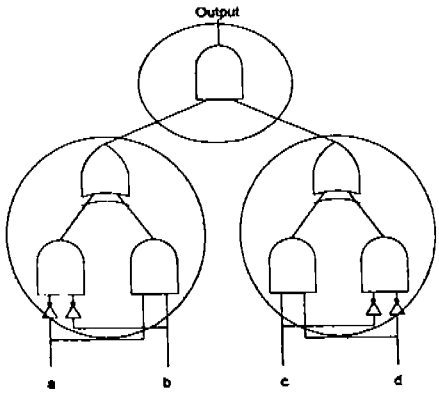
비슷하게, M 행렬의 열 연산 $C_1(M) \oplus C_4(M) \rightarrow C_4(M)$와 행 연산 $R_4(J) \oplus R_1(J) \rightarrow R_1(J)$에 의해 최종 결과 식 (6)과 같이 된다. 식 (7)은 행렬 연산의 결과에 의해 얻어진 최종 논리식이다.

$$= \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad (6)$$

$$= C_1(I) \cdot R_1(J)$$

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot [1\ 0\ 0\ 1]$$

$$= (a'b' \oplus ab) \cdot (c'd' \oplus cd) \quad (7)$$



(그림 3) FACTOR의 회로
(Fig. 3) Resulting circuits of FACTOR

지금까지, 행렬 연산에 의해 다단계 리드뮬러회로를 합성하는 과정을 살펴보았다. 최소화된 다단계 리드뮬러 표현식으로 구성된 결과 회로는 (그림 3)과 같다.

기존의 FACTOR는 회로 면적의 최적화, 지연 시간의 최소화, 결함 검출 능력의 극대화와 같은 논리 합성의 요구를 만족하는데 반해 입력 수에 따라 지수적인 기억 공간과 연산 시간이 필요하므로 대규모 회로에 적용하기 어렵다. 이를 해결하기 위해 논리 함수를 맵 대신에 BDD로 표현하고 BDD를 행렬 연산 과정에 그대로 적용한 BDD를 이용한 다단계 리드뮬러 합성 방법을 제시한다. 또한 FACTOR의 임의 패턴을 선택하는 방법을 개선한 최선의 패턴을 선택해 가는 방법을 제시한다.

4. BDD를 이용한 다단계 회로의 최소화

논리 합성의 시작을 행렬 대신에 BDD로 표현된 논리 함수를 이용한다. BDD로 표현된 논리 함수를 행렬 연산의 입력으로 사용하므로 대규모 회로의 논리 표현의 문제점을 개선한다. 본 논문의 핵심은 진리표 형태의 행렬을 입력으로한 행렬 연산을 어떻게 BDD를 이용한 행렬 연산으로 전환시키느냐에 달려 있다. 먼저 4장에서 사용되는 용어를 정의하고, 다음으로 BDD를 이용한 다단계 회로의 최소화 방법에 대해서 설명한다.

4.1 용어 정의

기존의 FACTOR[8]은 논리 함수가 주어지면 이를 진리표 형태의 행렬로 표현하고 최소의 차수를 갖는 입력 분할을 구한 다음 [AND, XOR] 연산자에 의한 행렬 연산을 이용하여 최소의 패턴을 추출한다. 회로를 구성하는 패턴의 수를 최소화함으로 회로를 구성하는 게이트 수가 최소화되고 회로의 면적이 최소화된다. 결과적으로는 최소화된 다단계 리드뮬러회로가 구성된다.

【정의 4.1】 피벗 원소

진리표 형태의 맵으로 표현된 논리 함수에서 패턴을 추출하기 위해 기준이 되는 원소를 피벗 원소라 한다.

【정의 4.2】 패턴

열 벡터와 행 벡터의 곱으로 구성되는 행렬 P를 패턴이라 한다.

$$P=\begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{mi} \end{bmatrix}[a_{j1}\ a_{j2}\cdots a_{jn}]$$

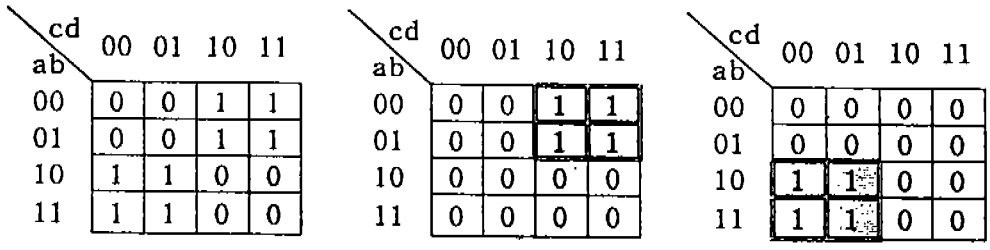
예를 들어, 논리 함수를 표현하는 맵이 (그림 4 (a))와 같이 주어지면 임의의 피벗 원소인 a′b′cd′는 패턴 a′c를 구하기 위한 기준이 되는 원소이고 ab′c′d는 패턴 ac′를 구하기 위한 피벗 원소이다. 따라서 이와 같은 피벗 원소를 선택하면 2개의 패턴으로 구성된 논리 함수 $f=a'c\oplus ac'=a\oplus c$을 구할 수 있다.

4.2 BDD 연산에 의한 패턴 추출

BDD에 의한 다단계 회로의 최소화 방법은 진리표 형태의 맵에 행렬 연산을 BDD를 이용한 행렬 연산으로 확장한 방법이다. BDD에 의한 패턴 추출 방법은 BDD를 이용하여 논리 함수를 간결하게 표현한다. 진리표 형태의 맵 대신에 BDD를 행렬 연산에 그대로 적용하여 패턴을 추출한다. 논리 함수는 맵 대신에 BDD로 표현되고, BDD에 의해 피벗 원소가 선택된다. FACTOR에서 이용한 [bit-AND, bit-XOR] 연산자를 이용한 행렬 연산 대신에 BDD(AND, XOR) 연산에 의해 패턴을 추출한다. BDD를 이용하여 패턴을 추출하는 연산은 맵 연산보다 매우 빠르므로 연산 시간을 향상시킬 수 있다. 따라서 기존의 FACTOR에서 기억 공간이나 연산 시간의 제약으로 인해 처리하지 못한 대규모 회로를 합성할 수 있는 중요한 의미를 갖는 방법이다. 먼저, 임의의 패턴을 선택하여 다단계 회로를 합성하는 방법에 대해서 설명하고, 다음으로 합성 결과를 개선하기 위한 최선의 패턴을 선택하여 다단계 회로를 최소화하는 방법에 대해서 설명한다.

4.2.1 BDD에 의한 논리 함수 표현과 입력 분할

논리 함수 f(a, b, c, d)=∑m(1, 2, 4, 5, 8, 9, 10, 11, 12, 14)이 주어졌을 때 입력 변수를 [a, b/c, d]로 분할하고 이를 진리표 형태의 맵으로 표현하면 (그림 5 (a))와 같이 표현되고, 입력 순서가 a〉b〉c〉d〉인 BDD(즉, ROCBDD)로 표현하면 (그림 5 (b))와 같이 표현된다. 입력 순서가 a〉b〉c〉d인 BDD에서 각 노드는 입력 변수를 나타내고 각 경로는 곱항을 의미한다. (그림 5 (a))에서 a′b′cd′인 항은 (그림 5(b))의 BDD에서 간선 0－〉0－〉1－〉0을 의미하고 보수 간선이 홀수개이므로 정점 0이 1로 해석된다. 입력 변수를 분할하는 방법은 입력 변수가 겹치지 않게 동일 크기로 분할하는 방법을 사용한다. 입력 변수의 수가 짝수개일때는 좌, 우 입력 변수가 동일하게 n/2로 나누어지고 홀수개일때는 좌 입력 변수가 n/2＋1, 우 입력 변수가 n/2개로 구성된다. BDD가 구성되면 함수를 구성하는 패턴을 추출하기 위해 기준이 되는 피벗 원소를 선택한다. 피벗 원소를 선택하는 방법은 BDD의 각 간선을 탐색하면서 처음으로 만나는 민텀(minterm)인 a′b′c′d를 피벗 원소로 선택한다. 모든 경우의 민텀을 고려하면 최적의 결과를 얻을 수 있으나 입력 수가 많은 대규모 회로에서는 매우 어려운 문제이다. 여기서는 FACTOR에서 이용하는 가장 간



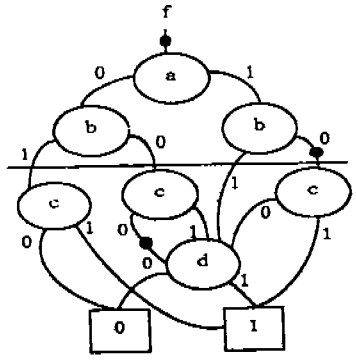
(a) 논리 함수 맵 표현　(b) a′b′cd′에 의한 a′c패턴　(c) ab′c′d에 의한 ac′패턴

(그림 4) 피벗 원소에 의한 패턴 선택
(Fig. 4) Pattern selection by pivot element

단한 패턴 선택 방법인 첫 번째 민텀을 피벗 원소로 선택한다.

cd / ab	00	01	10	11
00	0	1	1	0
01	1	1	0	0
10	0	1	1	1
11	1	0	1	0

(a) 논리 함수의 맵 표현



(b) 논리 함수의 BDD 표현

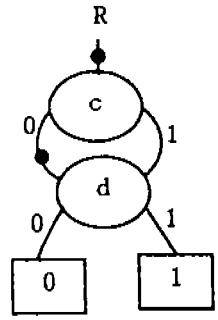
(그림 5) BDD에 의한 논리 함수의 표현
(Fig. 5) Representation of logic function by BDDs

4.2.2 패턴 추출 과정

BDD에서 피벗 원소를 선택하고 피벗 원소에 의해 ab 패턴과 cd 패턴을 추출하고 BDD(AND, XOR) 연산에 의해서 BDD를 변화시킨다. 먼저, BDD에서 cd 패턴을 구하는 방법에 대해서 설명한다. 입력 순서가 a〉b〉c〉d 인 BDD(f)를 구성하면 (그림 5 (b))와 같다. BDD에서 a′b′ 패스에 연결된 c와 d의 노드들로 구성된 부분-BDD는 cd 패턴을 의미한다. cd 패턴을 맵으로 표현하면 (그림 6 (a))와 같고, BDD로 표현하면 (그림 6 (b))와 같다. 이에 해당하는 논리식은 c′d ⊕cd′이다. 따라서 BDD의 피벗 원소에 의해 cd 패턴이 추출된다.

cd / ab	00	01	10	11
00	0	1	1	0
01	0	1	1	0
10	0	1	1	0
11	0	1	1	0

(a) cd 패턴의 맵 표현

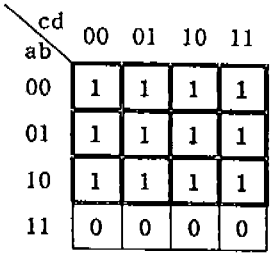


(b) cd 패턴의 BDD 표현

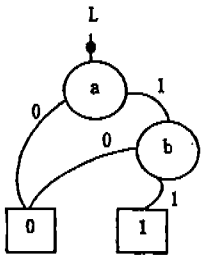
(그림 6) cd 패턴 추출
(Fig. 6) Extraction of cd pattern

다음으로 BDD에서 ab 패턴을 구하는 방법에 대해서 설명한다. 입력 순서가 d′〉c′〉b′〉a′로 구성된 BDD를 BDD(g)라 하자. 입력 순서가 a〉b〉c〉d인 BDD(f)나 입력 순서가 d′〉c′〉b′〉a′인 BDD(g)는 동일한 BDD를 나타낸다. 역순으로 구성된 BDD(g)를 구성하는 이유는 (그림 5 (b))와 같은 BDD에서 ab 패턴을 구하는 것은 매우 어려우나 d′〉c′〉b′〉a′로 구성된 BDD(g)에서는 ab 패턴을 쉽게 구할 수 있기 때문이다. 따라서 c′d 패스에 연결된 노드 a와 b로 구성된 부분-BDD는 ab 패턴을 의미하고 이를 맵으로 표현하면 (그림 7 (a))와 같고, BDD로 표현하면 (그림 7 (b))와 같다. 이에 해당하는 논리식은 a′b′ a′b ab′이다. 따라서 BDD의 피벗 원소에 의해 ab 패턴이 추출된다.

ab 패턴의 BDD와 cd 패턴의 BDD를 AND 연산(·)하여 XOR 패턴을 구성한다. 이를 맵으로 표현 (그림 8 (a))와 같고 BDD로 표현하면 (그림 8 (b))와 같다.

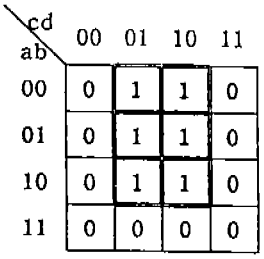


(a) ab 패턴의 맵 표현

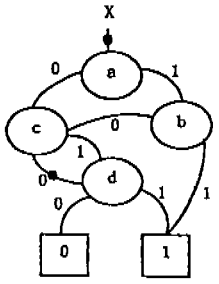


(b) ab 패턴의 BDD 표현

(그림 7) ab 패턴 추출
(Fig. 7) Extraction of ab pattern



(a) XOR 패턴의 맵 표현



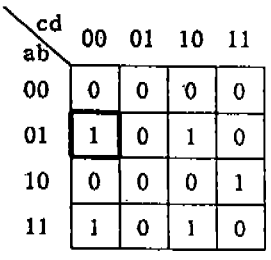
(b) XOR 패턴의 BDD 표현

(그림 8) XOR 패턴
(Fig. 8) XOR pattern

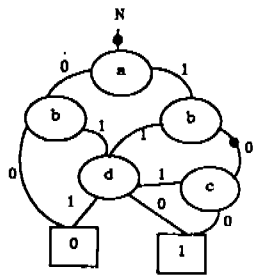
이에 해당하는 논리식은 (a′b′ ⊕a′b ab′)·(c′d ⊕cd′)으로 표현된다.

4.2.3 추출된 패턴에 의한 함수의 변화

생성된 XOR 패턴을 원래 함수의 BDD에 XOR 연산( )을 한다. XOR 연산을 한 후의 변화된 논리 함수에 대한 맵 표현은 (그림 9 (a))와 같고, BDD로 표현하면 (그림 9 (b))와 같다. BDD 연산에 의한 패턴 추출 과정은 3장에서 설명한 [bit-AND, bit-XOR] 연산자를 이용한 행렬 연산을 이용한 패턴 추출 과정과 동일함을 알 수 있다. 다음으로 (그림 9 (b))와 같이 변화된 BDD에서 새로운 피벗 원소를 선택하여 피벗 원소가 없을 때까지 동일한 과정을 반복한다. 따라서 일련의 피벗 원소의 선택에 의해서 논리 함수를 구성하는 패턴이 추출되고 이에 따라 함수가 변화되어 간다. 이 연속적인 과정을 맵으로 표현하면 (그림 10)과 같다. (그림 10 (a))에서 피벗 원소 a′b′c′d를 선택하고 피벗 원소와 관련된 ab 패턴은 a′b′ ⊕a′b ⊕ ab′=1 ⊕ab=(ab)′이고, cd 패턴은 c′d ⊕cd′=c ⊕d

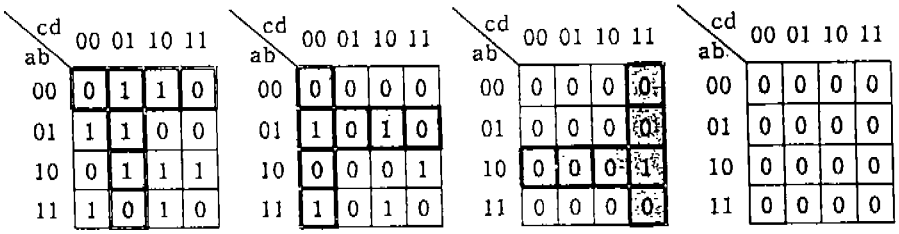


(a) 변화된 논리 함수의 맵 표현



(b) 변화된 논리 함수의 BDD 표현

(그림 9) XOR 연산에 의한 함수의 변화
(Fig. 9) Modification of logic function by XOR operation



(a) $(ab)' \cdot (c \oplus d)$ 패턴 (b) $b \cdot d$ 패턴 (c) $ab' \cdot cd$ 패턴

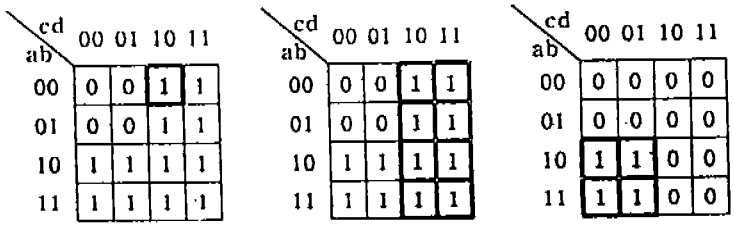
(그림 10) 패턴 추출과 논리 함수의 변화
(Fig. 10) Pattern extraction and modification of logic function

이다. 이 두 개의 패턴을 AND(·) 연산하면 패턴 $(ab)'(c \oplus d)$가 추출된다.

이 추출된 패턴을 원래 함수 f에 XOR($\oplus$) 연산하면 (그림 10 (b))가 된다. 따라서 추출된 3 개의 패턴에 의해서 함수가 구성되고 최소화된 결과는 $f=(ab)'(c \oplus d) \oplus bd \oplus ab'cd$ 이다. 피벗 원소에 의해서 추출된 XOR 패턴을 원래 함수에 XOR 연산하는 과정은 3장에서 설명한 행렬 연산 과정과 동일한 과정이다.

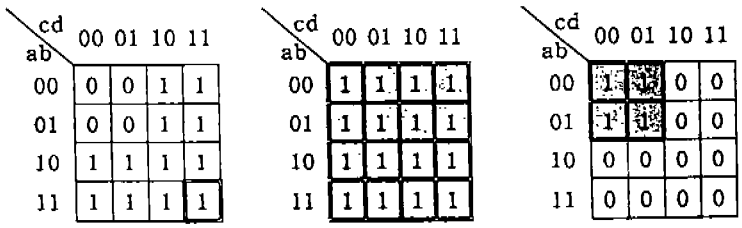
5. 최선의 패턴에 의한 다단계 회로의 최소화

BDD에 의한 다단계 회로 합성 방법에서 BDD로 표현된 논리 함수에서 피벗 원소를 선택하는 과정이 중요하다. 그 이유는 어떤 피벗 원소를 선택하느냐에 따라서 생성되는 패턴의 형태가 달라지고 합성 결과도 달라진다. 기존의 FACTOR에서는 논리 함수를 맵형태의 행렬로 표현하고 임의의 패턴 즉, 첫 번째



(a) 피벗 원소 a'b'cd' (b) c 패턴 추출 (c) ac' 패턴 추출

(그림 11) 임의의 패턴 선택에 따른 최악의 결과
(Fig. 11) Worst solution by selection of random pattern



(a) 피벗 원소 abcd (b) 상수 1인 패턴 추출 (c) a'c' 패턴 추출

(그림 12) 최선의 패턴 선택에 따른 최선의 결과
(Fig. 12) Best solution by selection of best pattern

민텀을 피벗 원소로 선택하여 패턴을 추출하는 방법을 이용한다. 그러나 BDD에 의한 합성 방법에서는 BDD를 이용하여 논리 함수를 표현하고 BDD를 이용하여 최선의 패턴을 구하므로 임의의 패턴을 선택하는 합성 결과보다 개선된 결과를 구할 수 있다.

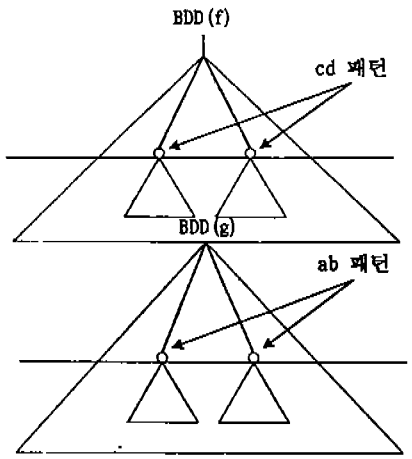
피벗 원소 선택의 중요성을 예를 들어 설명한다. (그림 11 (a))와 같이 논리 함수를 표현하는 맵에서 첫 번째 민텀 a′b′cd′를 피벗 원소로 선택할 경우 즉, 임의의 패턴을 선택할 경우 c 패턴과 ac′ 두 개의 패턴이 추출된다. 이에 해당하는 논리식은 $f=c \oplus ac'$ 이다.

그러나 (그림 12 (a))와 같이 피벗 원소를 abcd를 선택하면 상수 1인 패턴과 a′c′ 패턴이 구해진다. 이에 해당하는 논리식은 $f=1 \oplus (a'c')'=(a'c')'$ 이다. 따라서 임의의 패턴을 선택하는 것보다 상수 1인 패턴을 먼저 선택하므로 합성 결과가 개선된다.

분명히 최선의 결과를 구하는 패턴이 존재함에도 불구하고 임의의 패턴을 선택하므로 좋은 결과를 구하지 못하고 있다. 이를 개선하기 위해 최선의 패턴에 의한 다단계 리드뮬러회로의 최소화 방법을 제시한다.

5.1 BDD에서 최선의 패턴 선택

FACTOR에서는 최선의 패턴을 선택하기 어려우나 BDD를 이용하면 합성 결과를 개선하는 최선의 패턴을 쉽게 구할 수 있다. BDD를 이용하여 최선의 패턴을 선택하는 방법에 대해서 설명한다.



(그림 13) BDD에서 최적의 패턴 선택
(Fig. 13) Selection of best pattern in BDD

먼저 임의의 논리 함수가 주어지면 이를 두개의 BDD(f)와 BDD(g)로 구성한다. BDD(f)와 BDD(g)는 동일한 논리 함수 f와 g를 나타내는 BDD이다. 동일한 논리 함수에 대해서 입력 순서가 다른 두 개의 BDD를 구성하는 이유는 BDD(f)에서 cd 패턴을 쉽게 구할 수 있고 BDD(g)에서는 ab 패턴을 쉽게 구할 수 있기 때문이다. BDD(f)는 입력 순서가 a〉b〉c〉d 순으로 구성된 BDD이고, BDD(g)는 d′〉c′〉b′〉a′로 구성된 BDD이다. BDD(f)에서 입력 분할에 의해 BDD를 분할한 다음 분할된 곳의 노드 이하 부분-BDD는 패턴을 나타낸다 (그림 13)의 BDD(f)에서 분할된 곳의 노드는 cd 패턴을 나타낸다. BDD(g)의 분할된 곳의 노드는 ab 패턴을 나타낸다. BDD의 분할된 곳에서의 패턴의 수(차수)는 같으나 어떤 노드를 선택하느냐에 따라 합성 결과는 달라진다. 따라서 BDD를 이용하여 합성 결과를 개선하기 위해 최선의 패턴을 선택한다. 합성 결과에 영향을 미치는 최선의 패턴을 구하기 위해 BDD를 탐색하면서 분할된 곳의 패턴들 중에서 1) 상수 1인 패턴, 2) 보수 패턴, 3) 노드 수가 가장 적은 패턴을 선택한다. 따라서 임의의 패턴을 선택하는 방법보다 합성 결과를 개선할 수 있는 패턴을 먼저 선택하므로 합성 결과를 개선할 수 있다. 예를 들어, (그림 12 (b))와 같은 상수 1인 패턴을 먼저 선택하므로 패턴을 줄일 수 있다. 최선의 패턴을 선택하기 위한 알고리즘은 (그림 14)와 같다.

```
/* BDD의 분할된 최선의 패턴을 선택 */
/* 입력 : BDD로 표현된 논리 함수 f */
/* 출력 : 다단계 리드뮬러회로를 최소화하는 최선의 패턴 p */

find_best_pattern(f) {
  /* 패턴들 중에서 상수 패턴을 선택*/
  if ( is_constant(p) )
      return (select pattern);
      /* 패턴들 중에서 보수 패턴을 선택 */
      else if ( is_complement(p) )
          return (select pattern);
          /* 패턴들중 노드 수가 가장 적은 패턴을 선택 */
          else if ( mini_node_cnt(p) )
              return (select pattern);
}
```

(그림 14) 최선 패턴 선택 알고리즘
(Fig. 14) Best pattern selection algorithm

(그림 14)의 알고리즘에서 find_best_pattern()는 BDD로 표현된 논리 함수 f를 입력으로 하여 다단계 회로를 최소화하는 패턴을 추출하는 함수이다. is_constant()는 입력 분할에 의해 분할된 곳의 패턴들 중에서 상수 패턴 p를 선택하는 함수이고, is_complement()는 보수 패턴 p를 선택하는 함수이고, mini_node_cnt()는 패턴들 중에서 최소의 노드를 갖는 패턴을 선택하는 함수이다. 이 알고리즘에 대한 설명을 예를 들어 설명한다.

5.2 최선 패턴 선택의 예

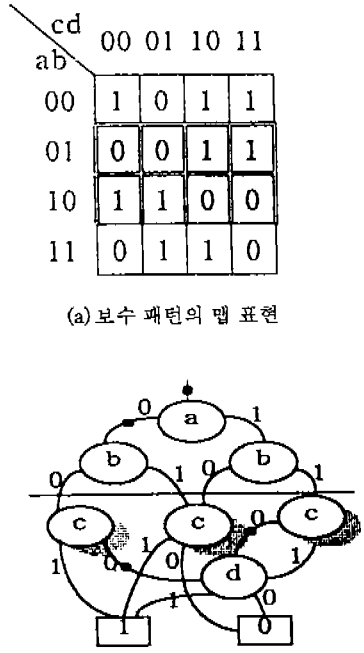
5.2.1 상수 1인 패턴 선택

BDD에서 상수 1인 패턴을 갖는 민텀을 피벗 원소로 선택하므로 XOR 게이트를 줄일 수 있는데 예를 들어 $1 \oplus cd = (cd)'$이기 때문이다. BDD상에서 패턴이 상수 1인가를 검사하는 것은 BDD 특성상 매우 간단하다. BDD(f)에서 상수 1인 패턴인가를 검사하는 방법은 분할된 곳의 노드에서 상수 1로 가는 간선이 있는가를 탐색하면 알 수 있다. 이를 맵으로 설명하면 (그림 15)와 같다. 따라서 민텀 abcd를 피벗 원소로 선택하면 상수 1인 패턴을 추출할 수 있다. 이에 해당하는 논리식은 $f = 1 \oplus a'd' \oplus a'bcd' \oplus ab'c'd$이다. 여기서 $1 \oplus a'd'$는 $(a'd')'$이므로 XOR 게이트가 하나 줄어든다. 이와 같은 상수 1인 패턴을 피벗 선택 과정에 고려함으로 합성 결과를 개선할 수 있다.

5.2.2 보수 패턴 선택

BDD에서 두개의 패턴이 보수(complement) 관계에 있는가를 검사하는 것은 BDD 특성상 매우 간단하다. BDD의 분할된 곳의 노드에 연결된 간선이 보수 간선인가 아닌가를 검사하면 쉽게 알 수 있다. (그림 16 (b))에서 a'b 간선과 ab' 간선에 연결된 cd 패턴은 보수 관계에 있다. (그림 16(a))의 맵에서 패턴 [0011]과 [1100]은 보수 관계를 가지고 있다.

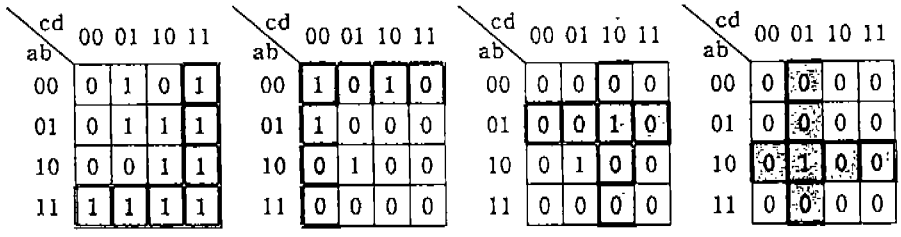
피벗 원소를 선택할 때 보수 패턴을 먼저 선택하므로 생성된 하나의 패턴으로 두 개의 보수 패턴을 표현하므로 패턴 하나를 줄일 수 있다. 예를 들어, 임의



(a) 보수 패턴의 맵 표현

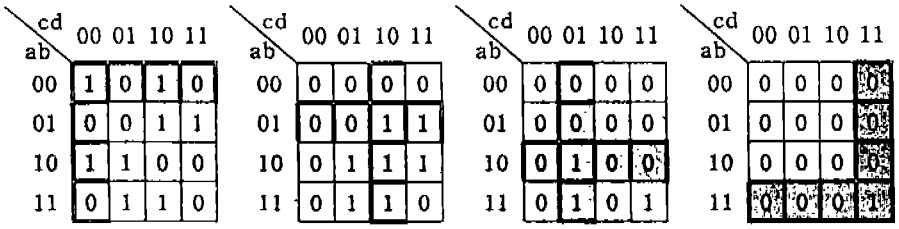
(b) 보수 패턴의 BDD 표현

(그림 16) 보수 패턴의 선택
(Fig. 16) Selection of complement pattern

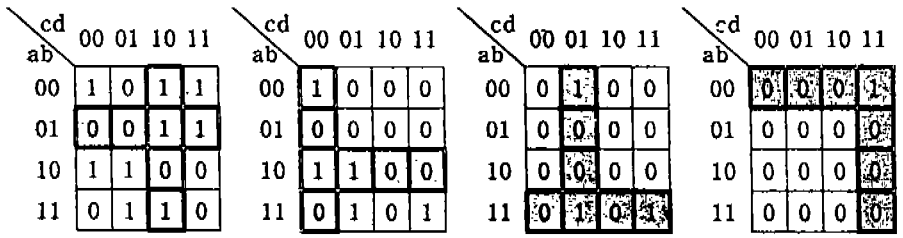


(a) 상수 1 패턴 선택 (b) a'd' 패턴 선택 (c) a'b·cd' 패턴 선택 (d) ab'·c'd 패턴 선택

(그림 15) 상수 1인 패턴 선택에 의한 결과
(Fig. 15) Solution by selection of constant-1 pattern



(그림 17) 임의 패턴 선택에 의한 결과
(Fig. 17) Solution by selection of random pattern



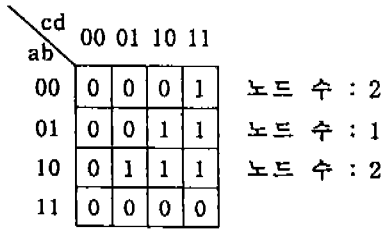
(그림 18) 보수 패턴 선택에 의한 결과
(Fig. 18) Solution by selection of complement pattern

의 패턴을 선택했을 때 생성되는 패턴의 형태는 (그림 17)과 같고 각각 4개의 패턴이 생성된다.

보수 패턴을 선택했을 때 생성되는 패턴의 형태와 함수의 변화는 (그림 18)과 같다. 생성되는 ab 패턴은 4개이고 cd 패턴의 수는 3이다. 왜냐하면 [0011] 패턴과 [1100] 패턴은 하나의 패턴만으로 표현이 가능하기 때문이다. 따라서 임의의 패턴을 선택했을 때 보다 cd 패턴이 하나 감소된다.

5.2.3 BDD에서 노드 수가 적은 패턴 선택

BDD의 분할된 곳의 노드에 노드의 크기 즉, 노드의 수를 계산하여 노드에 기억시켜 둔다. 이 노드들 중에서 노드의 크기가 가장 적은 노드를 갖는 패턴을 선택한다. (그림 19)는 논리 함수를 나타내는 맵으로 3개의 cd 패턴인 [0001], [0011], [0111]로 구성되어 있다. 패턴 [0001]은 논리식으로 cd를 의미하는데 BDD로 표현하면 2개의 노드 (c 노드, d 노드)가 필요하다. 또한 두 번째 cd 패턴인 [0011]은 c를 의미하는데 BDD로 표현하면 1개의 노드 (c 노드) 필요하다. 어떤 민텀을 피벗 원소로 선택하느냐에 따라서 합성 결과가 달라진다. 임의의 패턴을 선택하는 것보다 패턴의 수를 줄이는데 영향을 미치는 노드의 수를 고려하여 패턴을 추출하므로 합성 결과를 개선시킬 수 있다.



(그림 19) 패턴의 노드 수
(Fig. 19) Number of node of pattern

논리 함수를 표현하는데 BDD를 이용하므로 회로를 최소화시키는 패턴을 BDD를 탐색하므로 쉽게 찾을 수 있다. 즉, BDD에서 상수 패턴이나 보수 패턴 또는 가장 적은 노드를 갖는 패턴을 빠른 속도로 찾아내므로 연산 속도가 개선되고, 또한 최선의 패턴을 추출하므로 합성 결과도 개선된다.

6. 실험결과

본 논문에서 제시한 BDD에 의한 다단계 회로 합성 알고리즘은 IBM 486 PC의 LINUX 시스템에서 C 언어로 구현되었으며 Carnegie Mellon 대학의 BDD 공개 라이브러리 CMUBDD를 사용하였다. 구현된 다단계 합성 알고리즘을 BRM(Multi-level Reed Muller Minimizer using BDD)이라 한다. 이 합성 도구는 기존의 입력 분할에 의한 합성 도구인 FACTOR의 확장 버전으로 FACTOR의 표현 공간과 연산 시간의 문제를 해결한 다단계 리드뮬러회로 합성 도구이다.

본 논문에서 제시된 알고리즘의 효율성을 평가하기 위해 benchmark 회로에 의한 실험 결과를 Saul[2]의 결과와 XOR 게이트 수(항의 수)를 기준으로 하여 비교한 결과를 〈표 1〉에 제시하였다. BRM(random)은 논리 함수가 주어지면 이를 BDD로 표현하고 BDD를 탐색하면서 맨 처음 만나는 임의의 민텀을 피벗 원소로 선택하는 방법이다. 실험 결과, 입력 수가 16개이하인 회로에서 Saul의 결과보다 개선된 결과를 보이나 일부 회로(5xp1, clip, f51m, z4ml)는 개선된 결과를 보이지 않는다. 이는 패턴을 선택하는데 합성 결과를 개선할 수 있는 패턴이 존재함에도 불구하고 임의의 패턴을 선택하기 때문이다.

합성 결과를 개선하기 위해 최선의 패턴에 의한 합성인 BRM(best)을 제시하였다. 이 방법에서 패턴을 선택할 때 임의의 패턴을 선택하는 대신에 상수 1인 좌우 패턴, 보수 패턴, 노드의 수가 가장 적은 패턴을 선택하는 휴리스틱 방법을 사용하였다. 실험 결과, 임의의 패턴을 선택하는 BRM(random)의 결과보다 모든 회로에서 개선된 결과를 보이며, Saul의 결과에 비해 대부분의 회로에서 개선된 결과를 보인다. 특히, 격자 형태를 갖는 대칭 함수(9sym, rdxx)에서 최적에

〈표 1〉 Benchmark 회로에 의한 실험 결과

〈Table 1〉 Experimental results of benchmark circuits

Benchmark 회 로		Saul[2]의 결과		BDD based synthesis					
				BRM(random)		BRM(best)		BRM(partition)	
회로	PI/PO	#XOR	time [sec]	#XOR	time [sec]	#XOR	time [sec]	#XOR	time [sec]
5xp1	7/10	42	8.2	50	0.33	37	1.32	29	37.5
9sym	9/1	98	74.5	26	0.51	24	1.69	24	52.4
clip	9/5	113	50.7	131	1.27	115	4.08	42	118.0
con1	7/2	7	0.6	10	0.06	7	0.20	5	0.05
f51m	8/8	34	4.6	36	0.29	31	1.21	27	0.38
misex1	8/7	29	3.1	42	0.19	24	0.65	19	17.9
rd53	5/3	14	1.3	11	0.11	10	0.40	8	13.5
rd73	7/3	55	13.2	26	0.49	20	1.95	18	62.1
rd84	8/4	82	37.1	35	1.54	26	4.59	21	151.6
z4ml	7/4	12	0.8	31	0.31	25	0.94	10	27.5
bw	5/28	80	26.2	73	0.39	62	1.46	55	49.4
sao2	10/4	94	32.4	57	0.67	48	2.30	39	58.05
vg2	25/8	104	81.5	549	11.1	273	10.2	100	402.39
duke2	22/29	117	42.4	276	4.81	167	5.50	123	256.22
misex2	25/18	11	5.0	19	0.31	11	0.89	11	34.2
total		892	381.6	1372	22.38	880	37.38	531	1281.2

가까운 결과를 보인다.

그러나 대규모 회로(misex2, vg2, duke2)에서 misex2는 동일한 결과를 보이나 vg2, duke2 회로에서는 개선된 결과를 보이지 않는다. BRM은 입력 분할에 민감하게 동작되어지고, vg2 회로의 형태가 random하게 구성되어 있어 입력 분할에 매우 민감하게 반응한다. 따라서 모든 경우의 입력 분할을 고려하면 최적의 결과를 구할 수는 있으나 이는 매우 어려운 일이다. 따라서 BRM(partition)에서는 100개 이상의 입력 분할을 고려할 만큼 빠른 연산이 가능하므로 임의의 입력 분할 100개를 선택하여 가장 좋은 결과 구하는 방법을 사용한다. BDD를 이용한 합성 방법에 최선의 입력 분할을 고려하므로 합성 결과가 대폭 개선되었다. 〈표 1〉에 있는 benchmark 회로들과 같이 입력 분할에 민감한 회로에 가장 좋은 입력 분할을 고려하므로 좋은 결과를 구할 수 있고, vg2와 같은 대규모 회로에서 Saul의 결과보다 더 좋은 결과를 보인다.

7. 결 론

본 논문에서는 BDD를 이용한 다단계 리드뮬러회로 합성 방법을 제시하였다. 논리 함수를 맵 대신에 BDD로 표현하고 BDD 연산에 의해 다단계 회로를 합성한다. 논리 합성의 시작을 행렬 대신에 BDD를 이용하므로 기존의 행렬 연산에 의한 패턴 추출 과정이 BDD 연산에 의한 패턴 추출 과정으로 확장되었다. 따라서 이 방법은 FACTOR에서 대규모 회로를 합성하는데 발생되는 표현 공간과 연산 시간의 문제점을 해결한 방법이다. 이 방법은 BDD를 이용하여 회로의 면적을 최소화하는 최선의 패턴 (즉, 상수 패턴, 보수 패턴, 노드 수가 가장 적은 패턴)을 선택한다. 따라서 임의의 패턴을 선택하는 기존의 방법에 비해 다단계 회로를 구현하는 패턴의 수가 최소화되고 결과적으로는 최소의 면적으로 구성된 다단계 리드뮬러회로가 합성된다. Benchmark 회로에 의한 실험 결과, 대부분의 회로에서 기존의 결과에 비해 개선된 결과를 보인다. 특히, benchmark 회로중에서 대칭 함수에서는 매우 좋은 결과를 보인다. 대규모 회로에 최선의 입력 분할을 고려하므로 기존의 결과에 비해 개선된 결과를 얻었다. 앞으로 연구해야 할 내용으로는 1) BRM의 패턴 선택 과정에 각 출력에 대해 공통 패턴을 먼저 고려하므로 각 출력에 대한 패턴의 수를 최소화하는 다중 출력을 고려한 합성 방법을 현재 연구 중이다. 2) 또한 BRM을 현재 많은 연구가 진행되고 있는 FPGA(Field Programmable Gate Array)에 적용하여 회로를 구현하는 기본 셀인 LUT (Look-up Table) 수에 의해서 기존의 결과와 성능을 비교할 예정이다. 따라서 현재, 게이트 형태로 되어 있는 BRM을 LUT 형태의 회로로 합성하는 방법에 관한 연구가 진행 중이다.

참 고 문 헌

[1] J. Saul, "An Improved Algorithm for the Minimization of Mixed Polarity Reed Muller Representation," in Proceedings of the IEEE International Conference on Computer Design, pp. 372-375, Sep. 1990.

[2] J. Saul, "An Algorithm for the Multi-level Minimization of Reed-Muller Representations," in Proceedings of the IEEE International Conference on Computer Design, pp. 634-637, Oct. 1991.

[3] A. Sarabi and M. Perkowiski, "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks," in Proceedings of the 29th ACM/IEEE Design Automation Conference, pp. 30-35, 1992.

[4] M. Helliwell, M. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-muller Forms," in Proceeding of the 25th ACM/IEEE Design Automation Conference, pp. 427-432, 1988.

[5] S. B. Akers, "Binary Decision Diagrams," IEEE Transactions on Computers, C-27(6), pp. 509-516, June. 1978.

[6] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in Proceedings of the 27th ACM/IEEE Design Automation Conference, pp. 40-45, June. 1990.

[7] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," IEEE Trans-

actions on Computers, C-35(8), pp. 677-691, Aug. 1986.

[8] T. T. Hwang, R. M. Owens and M. J. Irwin, "Exploiting Communication Complexity for Multi-level Logic Synthesis," IEEE Transactions on Computer Aided Design, Vol. CAD-9, pp. 1017-1027, Oct. 1990.

[9] G. S Lee, J. Y Chang, Robert M. Owens, Mary Jane Irwin, "Synthesis of Multi-level Reed Muller Circuits using Matrix Transformations," IFIP Workshop on Applications of the Reed Muller Expansion to Circuits Design, Hamburg, Germany, Sep. 1993.

[10] Seh-woong Jeong, Binary Decision Diagrams and Their Applications to Implicit Enumeration Techniques in Logic Synthesis, Ph. D. Thesis, University of Colorado at Boulder, 1992.

[11] 장준영, 이형수, 천승환, 이귀상, "다단계 리드뮬러회로의 합성," 한국정보과학회 봄 학술 발표지, Vol. 20, No. 1, pp. 905-908, 1993.

[12] 장준영, 이귀상, "BDD를 이용한 논리 함수의 분할," 한국정보과학회 봄 학술 발표지, Vol. 22, No. 1, pp. 803-806, 1995.

장 준 영

1985년 전남대학교 전산학과 졸업(이학사)

1987년 중앙대학교 대학원 전자산학과 졸업(이학석사)

1995년 전남대학교 대학원 전산학과 졸업(이학박사)

1988년~1989년 조선대학교 산업대학 전산학과 시간강사

1991년~1995년 전남대학교 자연대학 전산학과 시간강사

1995년~현재 대불대학교 컴퓨터학부 전임강사

관심분야: VLSI 설계자동화, 논리합성, FPGA 합성, 컴퓨터구조, 테스팅

이 귀 상

1980년 서울대학교 전기공학과 졸업(공학사)

1982년 서울대학교 컴퓨터공학과 졸업 (공학석사)

1991년 Pennsylvania State University 전기공학과(공학박사)

1982년~1983년 금성통신연구소 연구원

1983~현재 전남대학교 전산학과 부교수

관심분야: VLSI 설계자동화, 테스팅, 논리합성, 신경회로망