論文95-32A-10-9

# MRM: 상징행렬을 이용한 다단계 리드뮬러회로의 합성 도구

# (MRM : A Synthesis Tool for Multi-level Reed Muller Circuits using Symbolic Matrix)

李 貴 相 *, 張 峻 榮 *

(Gueesang Lee and Juneyoung Chang)

요 약

본 논문에서는 행렬연산을 이용한 다단계 리드뮬러회로 합성 도구인 MRM(Multi-level Reed Muller Minimizer)에 대해 기술한다. 최근에 제안된 행렬연산을 이용한 논리합성방법은 칩면적의 최소화, 회로의 지연시간 최소화, 결함검출 능력의 극대화등의 요구를 수용하는 장점에 비해 진리표 형태만을 입력으로 사용하기 때문에 논리합성을 위한 공간과 속도면에서 그 효과가 반감되고 있다. 이러한 단점을 극복하기 위해 기존의 행렬연산을 그대로 이용하면서 진리표 형식의 입력대신에 2 단계 최적화를 거친 결과, 즉 큐브(cube)들을 이용한다. 따라서 행렬연산의 입력행렬의 크기가 감소되어 대규모회로를 쉽게 표현, 처리할 수 있다. 2단계 표현을 다단계 표현으로 변환시키기 위해서, 서로 다른 입력패턴으로 상징행렬을 구성하고 이를 행렬연산 방법에 적용한다. 이 방법을 이용한 결과, 기존의 결과보다 좋은 결과를 보여주었다. 본 논문에서 제시한 방법은 리드뮬러 합성 분야에서는 새로운 방법이며 확고한 수학적 기초를 갖는다.

**Abstract**

In this paper, a synthesis tool using matrix operations for designing multi-level Reed Muller circuits is described which has been named as MRM (Multi-level Reed Muller Minimizer). The synthesis method which uses matrix operations has advantages in effectively minimizing chip area, delay optimization and fault detection capability. However, it uses only truth-table type maps for inputs, synthesizing only small circuits. To overcome the weakness, our method accepts two-level description of a logic function. Since the number of cubes in the two-level description is small, the input matrix becomes small and large circuits can be synthesized. To convert two-level representations into multi-level ones, different input patterns are extracted to make a map which can be fed to the matrix operation procedure. Experimental results show better performance than previous methods. The matrix operation method presented in this paper is new to the society of Reed Muller circuits synthesis and provides solid mathematical foundations.

## I. 서 론

최근들어 리드뮬러표현을 사용하여 논리함수를 구현하고 설계하는 데에 많은 관심이 기울여지고 있다[4,5,6,7]. 리드뮬러표현은 일반적으로 AND/OR 연산 대신 AND/XOR 연산을 사용한 논리함수의 표현을 일컫는다. 리드뮬러표현을 이용하여 AND/XOR 게이트로 구현된 회로를 리드뮬러회로라 한다. 리드뮬러회로는 논리함수 표현의 간결성과 내재된 높은 검증도의

* 正會員, 全南大學校 電算學科

(Dept. of Computer Science, Chonnam Univ.)

接受日字:1994年3月25日, 수정완료일:1995年9月28日

장점에도 불구하고 SOP(Sum-Of-Products) 표현과 비교하여 구현에 있어서의 복잡성과 비용때문에 비교적 자주 언급되지 않았다. 그러나 최근들어 VLSI 기술의 발달로 인해 다른 게이트에 비해 XOR 게이트가 가격과 속도에서 동등한 구현이 가능하게 되었다[9].

리드뮬러형식을 이용한 AND/XOR 구현의 주된 응용과 그 장점은 [15]에 잘 기술되어 있다.

리드뮬러회로의 합성은 크게 이단계 합성과 다단계 합성으로 구분할 수 있다. 이단계 리드뮬러 회로의 합성에 관한 연구는 1950년대 이후 많은 연구결과가 발표되고 있으나[4,5,6,9] 다단계 리드뮬러회로에 관한 연구[7,8,11,12,14]는 Saul[7,8]의 연구가 대표적인 것이라 할 수 있다. 그러나, 최근에 제안된 행렬연산을 이용한 논리합성 도구인 FACTOR[2] 는 주어진 회로의 함수를 진리표 또는 행렬형태로 표현하여 이를 행렬연산(Matrix operation)을 통해 회로를 합성하며, 이 때 [+,*] 연산대신 [bit-AND, bit-XOR] 연산을 이용한다. 합성된 회로는 일반적인 게이트들을 사용하여 구현된다.

그러나 엄밀히 구분한다면 FACTOR 는 다단계 리드뮬러회로의 합성도구의 하나라 볼 수 있다. 이 논리합성도구는 회로 면적의 최소화, 지연 시간의 감소, 결함검출 능력의 극대화[13]등의 요구를 수용하는 장점을 갖고 있으나 행렬연산의 입력으로 진리표 형태의 행렬만을 사용하므로 합성할 수 있는 회로는 소규모 회로로 제한되고 있다.

본 논문에서는 이 문제점을 해결하기 위한 상징행렬을 이용한 다단계 리드뮬러회로 합성 방법을 제시한다. 이 방법에 의해 구현된 합성도구를 MRM(Multi-level Reed Muller Minimizer)라 명명하였다. 이 방법은 기존의 행렬연산을 그대로 사용하면서 진리표 형태의 행렬입력 대신에 이단계 최적화를 거친 결과인 큐브(Cube)들로 구성된 상징행렬을 이용한다. 따라서 기존의 진리표를 이용한 행렬연산이 상징행렬을 이용한 논리식연산(Logic operation)으로 변환되므로 다단계 리드뮬러회로의 합성을 위한 기억공간(Space)이 감소되고 처리시간(Time)이 향상되어 대규모회로를 쉽게 표현, 처리 할 수 있다.

다음 Ⅱ 장에서는 상징행렬을 이용한 다단계 리드뮬러회로 합성과 그 알고리즘에 대해서 설명한다. Ⅲ 장에서는 실험결과에 대해서 기술하고, 마지막 Ⅳ 장에서는 결론을 제시한다.

## Ⅱ. MRM : 상징행렬에 의한 다단계 리드뮬러회로 합성

### 1. 상징행렬의 구성

기존의 행렬연산을 이용한 합성 방법에서는 0과 1의 배열인 진리표 형태의 행렬을 입력으로 행렬연산 과정을 통해 다단계 회로로 합성한다. 진리표 형태를 이용한 행렬연산는 입력 수에 따라 입력행렬의 크기가 지수승($2^n$ ,n:입력 수)인 기억공간이 필요하게 되므로 입력행렬의 크기에 따라 행렬연산에 필요한 기억공간과 처리시간이 증가된다.

이러한 문제를 해결하기 위해 상징행렬을 행렬연산에 적용한 다단계 리드뮬러회로 합성 방법에서는 논리합성의 시작을 진리표 형태의 행렬대신에 이단계 리드뮬러 최소화를 거친 큐브들 즉, 논리식으로 구성된 상징행렬(Symbolic matrix)로 부터 시작한다.

상징행렬을 구성하는 과정을 보면 그림1.과 같이 진리표로 논리함수가 주어졌을 때, 2 단계 리드뮬러 최소화 도구[6,15]에 의해 그림 2.와 같은 상징적인 논리식 즉, 큐브들이 생성된다.

| | cd | 00 | 01 | 01 10 | 11 |
|---|---|---|---|---|---|
| ab | 00 | 0 | 0 | 0 | 0 | 0 |
| | 01 | 1 | 1 | 1 | 1 | 0 |
| | 10 | 0 | 0 | 0 | 0 | 0 |
| | 11 | 0 | 1 | 1 | 1 | 1 |

그림 1. 논리함수의 진리표 표현
Fig. 1. Truth table representation of a logic function.

최소화된 2 단계 리드뮬러 논리식은 $F = a'bcd' \oplus a'b \oplus ab' \oplus ab'cd'$ 이다.

| a b c d o | | 논리식 표현 |
|---|---|---|
| 0 1 - - 1 | ➡ | a'b - - |
| 0 1 1 0 1 | ➡ | a'b c d' |
| 1 0 - - 1 | ➡ | a b'- - |
| 1 0 0 0 1 | ➡ | a b'c'd' |

그림 2. 2 단계 리드뮬러 최소화의 결과
Fig. 2. Results of Two-level AND/XOR minimization.

위의 결과를 그림 3.과 같이 상징적인 논리식으로 구성한 행렬을 본 논문에서는 상징행렬(Symbolic matrix)이라 부르기로 한다.

| | c'd | cd' | -- |
|---|---|---|---|
| a'b | 0 | 1 | 1 |
| ab' | 1 | 0 | 1 |

그림 3. 상징행렬
Fig. 3. Symbolic matrix.

다음 절에서는 상징행렬을 행렬연산에 적용하는 과정과 알고리즘에 대해 설명하고, 이를 예제를 통해서 설명한다.

2. 상징행렬을 적용한 행렬연산 과정

상징행렬을 적용한 행렬연산에서는 주어진 함수를 표현하는 진리표 입력 대신 논리식으로 구성된 상징행렬을 입력행렬 M으로 하여 좌우 측에 단위행렬을 추가하여 $F = I \cdot M \cdot J$ 행렬을 만들고, 이를 연속적인 행렬변환 과정에 의해 $F = I_1 \cdot M_1 \cdot J_1$이 만들어진다. 여기서 입력행렬로 사용된 M 행렬은 주어진 함수의 진리표 형태의 행렬 대신에 2 단계 리드뮬러 최소화의 결과에서 추출된 논리식으로 구성된 상징행렬이다. 따라서 진리표 형식의 행렬연산(matrix operation) 과정이 상징행렬에 의한 논리식연산(logic operation)으로 변환되어 다단계 리드뮬러회로가 합성되어가는 것을 알수 있다. 논리식으로 구성된 상징행렬을 입력으로 하여 행렬연산 과정을 통해 논리식연산을 수행해 가는 과정을 보면 그림 4.와 같다.

주어진 회로의 함수 M
➡ **I • M • J** (M: 상징행렬 I,J: 단위행렬)
➡ 행 연산( Row operation)
행렬 M의 행연산 $[R_i(M) = R_i(M) \oplus R_j(M)]$
행렬 I는 열연산 $[C_j(I) = C_i(I) \oplus C_j(I)]$
➡ 열 연산(Column operation)
행렬 M의 열연산 $[C_j(M) = C_i(M) \oplus C_j(M)]$
행렬 J의 행연산 $[R_i(J) = R_i(J) \oplus R_j(J)]$
➡ **$I_1 \cdot M_1 \cdot J_1$** ($M_1$ 은 대각행렬)

그림 4. 상징행렬에 의한 행렬연산 과정
Fig. 4. Matrix operation by symbolic matrix.

여기에서 $I_1$ 과 $J_1$ 은 각각 그 열과 행에 의해서 부분함수들을 정의한다. 이들을 $f_1, f_2, \ldots, f_r$( $f_i$ 는 $I_1$ 의 열에 해당), $g_1, g_2, \ldots, g_r$( $g_i$ 는 $J_1$ 의 행에 해당) 이라 하자. 이때 r는 $M_1$의 대각원소 중 1의 개수이다. 이때 원래의 함수 F는 다음과 같이 나타낼 수 있다.

$$F = f_1 \cdot g_1 \oplus f_2 \cdot g_2 \oplus \ldots \oplus f_r \cdot g_r$$

만일 이때 $f_i$, $g_i$가 하나의 큐브를 형성한다면 F는 이단계 리드뮬러회로라 할 수 있다. 그러나 대부분의 경우 이 부분함수들은 하나의 큐브를 형성하지 않고 여러 개의 큐브들의 복합체이다. 따라서 각 부분함수들 $f_i$, $g_i$는 다시 그 자신이 행렬 M으로 표현되어 위의 과정을 반복한다.

F = I • M • J ( M:상징행렬, I,J:단위행렬 )

$$= \begin{vmatrix}1&0\\0&1\end{vmatrix} \cdot \begin{vmatrix}0&1&1\\1&0&1\end{vmatrix} \cdot \begin{vmatrix}1&0&0\\0&1&0\\0&0&1\end{vmatrix}$$

( M 의 행연산 )

M 의 $R_1 = (R_1 \oplus R_2)$

I 의 $C_2 = (C_1 \oplus C_2)$

$F = a'bcd' \oplus a'b \oplus ab'c'd' \oplus ab'$

$$= \begin{vmatrix}1&1\\0&1\end{vmatrix} \cdot \begin{vmatrix}1&1&0\\1&0&1\end{vmatrix} \cdot \begin{vmatrix}1&0&0\\0&1&0\\0&0&1\end{vmatrix}$$

( M 의 행연산 )

M 의 $R_2 = (R_1 \oplus R_2)$

I 의 $C_1 = (C_1 \oplus C_2)$

$F = a'bc'd' \oplus a'bcd' \oplus (a'b \oplus ab')c'd' \oplus (a'b \oplus ab')$

$$= \begin{vmatrix}0&1\\1&1\end{vmatrix} \cdot \begin{vmatrix}1&1&0\\0&1&1\end{vmatrix} \cdot \begin{vmatrix}1&0&0\\0&1&0\\0&0&1\end{vmatrix}$$

( M 의 열연산 )

M 의 $C_2 = (C_1 \oplus C_2)$, $C_3 = (C_2 \oplus C_3)$

J 의 $R_1 = (R_1 \oplus R_2)$, $R_2 = (R_2 \oplus R_3)$

$F = a'bc'd' \oplus a'bcd' \oplus (a'b \oplus ab')\ cd' \oplus (a'b \oplus ab')$

$$= \begin{vmatrix}0&1\\1&1\end{vmatrix} \cdot \begin{vmatrix}1&0&0\\0&1&1\end{vmatrix} \cdot \begin{vmatrix}1&1&0\\0&1&1\\0&0&1\end{vmatrix}$$

$= I_1 \cdot M_1 \cdot J_1$

$F = ab'(c'd' \oplus cd') \oplus (a'b \oplus ab')(cd' \oplus 1)$ (2.1)

3. 알고리즘

이 절에서는 상징행렬을 이용하여 다단계 리드뮬러 회로로 합성해 가는 알고리즘에 대해서 설명한다. 2 단계 리드뮬러 최소화의 출력은 trmfile(Two-level Reed-Muller Minimization file) 로 나타내었으며, 이를 상징행렬로 변환하여 행렬연산을 통해 최종적으로 최소화 다단계 리드뮬러 표현식이 게이트 레벨 기술언어인 GLUE[4] 화일로 출력된다. 개괄적인 알고리즘은 그림 5.에 나타나 있다.

```
/* MRM : Multi-level Reed Muller synthesizer */
Main()
{
        read_trm(trmfile);/* trmfile을 읽어 상징
                             행렬 구성 */
        mrm(trmfile);     /* MRM 메인루틴 */
        generate_glue();  /* GLUE 화일 형태로
                             최소화된 표현식 출력 */
}
```

```
mrm(trmfile)  /* MRM 메인루틴 */
{
      if ( number of input <= 1 ) return;
                /* MRM 종료 조건 */
        find_best_partition();
                /* 최적의 입력분할 선택 */
        calculateLMR();
                /* 상징행렬구성 및 행렬연산 */
        make_trm();
                /* 좌우측 trmfile 구성 */
        mrm(left_trmfile);
             /* left trmfile에 의한 MRM 반복 */
        mrm(right_trmfile);
             /* right trmfile에 의한 MRM 반복 */
}
```

그림 5. 상징행렬에 행렬변환 알고리즘
Fig. 5. Matrix transformation algorithm by symbolic matrix.

read_trm()은 2 단계 리드뮬러 최소화 결과인 상징적인 논리식 들로 구성된 trmfile 화일을 읽어서 상징행렬을 만든다. mrm()은 메인루틴으로 read_trm()에 의해서 구성된 상징행렬을 입력으로 재귀적으로(recursively) 행렬연산을 실행한다. 최종적으로 generate_glue() 에 의해 게이트 레벨 기술 언어인 GLUE 형태의 화일로 최적화된 다단계 리드뮬러회로가 출력된다.

여기에서 그림 6.에 나와 있는 것과 같이 find_best_partition()은 적절한 입력분할을 생성하고, 이에 따른 행렬을 구성한 다음 각 행렬에 따른 rank를 계산한다. 입력분할에 따라 논리함수를 구성하는 행렬의 모양과 rank의 수가 달라지게 되므로 모든 경우의 입력분할 형태를 계산하여, 행렬의 크기가 가장 작고 rank 수가 가장 적은 분할을 선택한다.

그리고 make_matrix()에서는 각 입력 분할에 따른 상징행렬을 만들고, get_rank()에서는 구현되는 회로의 크기에 대한 예측으로서[4] 각 상징행렬의 rank를 구한다. 이는 그림 7.과 같다.

```
find_best_partition()  /* 최적 입력분할 선택 */
{
        repeat until best one is found
            make_partition();
                /* 모든 가능한 입력분할 생성 */
            make_matrix();
                /* 각 입력분할에 따른 행렬 구성*/
            get_rank();
                /* 각 행렬의 rank 계산 */
}
```

그림 6. 최적 입력분할의 선택
Fig. 6. Find best input partition.

```
calculateLMR(best partition)
/* 최적 입력분할에 의한 행렬연산 */
{
        make_matrix();
        /* 최적 입력분할에 의한 상징행렬 구성 */
        matrix_operation();
        /* 상징행렬에 의한 행렬연산 */
}
```

그림 7. 상징행렬에 의한 행렬연산
Fig. 7. Matrix operation by symbolic matrix.

그림 8.의 calculateLMR()는 최적분할에 의해 상징행렬을 구성하고 행렬연산을 통해서 다단계 리드뮬러회로를 합성한다. make_matrix()에서는 최적 입력분할에 의해 상징행렬을 만들고, 이를 이용하여 연속적인 행렬연산을 수행한다. 논리식으로 구성된 상징행렬을 행렬연산에 적용하므로 결과적으로는 논리식연산이 수행된다. 여기서 상징행렬로 구성된 함수 $F = I \cdot M \cdot J$ 에 대해서 $F = I_1 \cdot M_1 \cdot J_1$이 만들어지고, make_trm()

에서 $I_1$으로 left_trm 화일을 만들어 다시 mrm() 함수를 반복적으로 실행하고, $I_1$으로 right_trm 화일로 만들어 mrm()을 반복적으로 수행하여 한다. 입력변수의 수가 1 보다 작거나 같을 때까지 부분회로로 분할하여 이 알고리즘을 반복적으로 적용함으로 최적화된 다단계 리드뮬러회로가 합성된다.

## Ⅲ. 실험결과 및 토의

본 논문에서 제안된 행렬연산을 이용한 다단계 리드뮬러회로 합성 방법은 NeXT Station과 Linux System(IBM PC Unix System)에서 C 언어로 구현하였으며, 이를 MRM(Multi-level Reed Muller circuits synthesizer)이라 명명하였다. MRM은 부회로들 사이의 연결선의 수를 제한하지 않으며 상징행렬을 구성하고 처리할 수 있는 기능을 갖는다. 기존의 합성도구 FACTOR 는 부회로들 사이의 제한된 연결선의 수 때문에 대다수의 MCNC 벤치마크회로에서 잘 동작하지 않는다. MRM은 이러한 문제점을 해결하고 상징 표현에 의한 입력을 처리할 수 있도록 확장되었다. 입력 회로는 ESPRESSO형태[11]의 PLA 형식이고(여기서는 TRM file 이라는 용어를 주로 사용하였음) 최소화된 다단계 리드뮬러 표현은 게이트 레벨 기술언어인 GLUE 화일로 출력된다.

실험은 합성 과정에 어떠한 선택사항을 선택하느냐에 따라 실행하고 그 결과를 비교하였다. 먼저 -s option은 모든 가능한 입력중에서 가장 작은 행렬을 만드는 입력분할을 선택하지 않고 맨 처음의 입력분할을 선택함으로써 실행시간을 단축시키는 것이다. 그리고 -m option을 주는 경우는 행과 열 연산을 특정한 순서가 없이 행렬에 주어진 대로 대각원소(pivot elements)를 선택하므로, 이득함수(gain function)에 의해 XOR operator의 수를 단조감소시키는 방법과 비교하고자 하는 것이다. 또한 -f option은 일정숫자와 함께 flatten level을 결정함으로써 어떤 경우에 가장 좋은 합성결과를 보기 위해서 주어진 것이다. flatten이란 주어진 상징행렬을 역으로 진리표형태로 만드는 것으로써 이는 때때로 상징행렬의 크기가 진리표형태보다 훨씬 큰 경우가 존재할 수 있기 때문이다. 또한 행렬의 크기가 비슷하더라도 경우에 따라 진리표형태가 더 좋은 합성결과를 보여주는 경우도 있다.

본 실험에서 비교의 기준은 합성결과의 AND 와 XOR operator 의 갯수이다. 최종 레이아웃(layout)의 크기가 반드시 이러한 gate 의 수에 의해 좌우되지는 않으나 이는 일반적인 논리합성도구 비교의 기준으로 받아들여지고 있다. 또한 합성시간도 제시하였다.

표 1.은 실험에 사용한 benchmark 회로들을 보여준다. 여기에 주어진 회로들은 HERMES[6]와 TRM[15]로 부터 생성된 회로들이다. 이들 이단계 리드뮬러 회로합성도구들은 아직까지 큰 회로들에 대한 benchmark 회로들이 부족하여 임의의 회로들을 추가하여(myex1-myex4) 실험하였다. 그러나 본 논문의 알고리즘은 입력의 크기가 아니라 2 단계 회로의 곱항(product term) 의 수에 관련이 있으므로 입력의 크기는 절대적인 평가의 기준이 될 수 없을 것으로 보인다. 현재 진행중인 대규모회로에 대한 2단계 합성도구 TRM-II(Two-level Reed Muller circuits Minimizer II)가 완성되면 MRM이 본격적으로 활용될 수 있을 것이다[15].

표 1. 실험에 사용한 benchmark 회로
Table 1. Benchmark circuits.

| 회로 | #in | #out | #prd | 회로 | #in | #out | #prd | 회로 | #in | #out | #prd |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5xp1 | 7 | 10 | 47 | 9sym | 9 | 1 | 87/69 | z9sym | 9 | 1 | 85/69 |
| count1 | 8 | 1 | 2 | count2 | 8 | 2 | 4 | count8 | 8 | 8 | 15 |
| clip | 9 | 5 | 143 | con1 | 7 | 2 | 10/14 | col4 | 14 | 1 | 14 |
| dist | 8 | 5 | 115 | sao2 | 10 | 4 | 63 | f51m | 8 | 8 | 33/53 |
| mul3 | 6 | 6 | 20/26 | mul4 | 8 | 8 | 79/13 | misex1 | 8 | 7 | 43 |
| m181 | 15 | 9 | 31 | mlp4 | 8 | 8 | 79 | ex9 | 9 | 1 | 69 |
| myex1 | 30 | 3 | 68 | myex2 | 22 | 9 | 87 | myex3 | 25 | 8 | 110 |
| myex4 | 31 | 5 | 180 | rd53 | 5 | 3 | 17/23 | rd73 | 7 | 3 | 46/96 |
| rd84 | 8 | 4 | 191 | root | 8 | 5 | 47 | root6 | 6 | 3 | 7 |
| sqr4 | 4 | 8 | 12 | sqr5 | 5 | 8 | 32 | sqr6 | 6 | 12 | 36 |
| 9sym | 9 | 1 | 87 | count8 | 8 | 8 | 15 | rd53 | 5 | 3 | 17 |
| bw | 5 | 28 | 27/70 | life | 9 | 1 | 63 | xor5 | 5 | 1 | 16 |
| z4 | 7 | 4 | 30 | z4ml | 7 | 4 | 45 | z5xp1 | 7 | 10 | 61 |

표 2.는 -s option과 -m option을 사용할 때의 결과들을 비교한다. -m option을 쓰지 않고 XOR 항을 단조감소하는 방향으로 행과 열연산을 사용하면 훨씬 좋은 결과를 냄을 알 수 있다. 실행시간도 사실상의 차이점을 볼 수 없었다. 또한 -s option을 사용했을 때, 오히려 더 좋은 결과를 내는 등, 사실상 대부분의 경우 최적의 입력분할을 찾는 노력이 효과적이 아님을 보여주었다. 마지막 열은 flatten level 에 따른 결과를 보여주는 것이다. 이는 상징행렬대신 진리표 형태의 행렬을 이용하는 것을 말한다. 만일 큐브들의 갯수가 2^n(n=#inputs) 보다 큰 경우는 당연히 flatten 한 경우 더 좋은 결과를 보인다.

표 2. HERMES 출력을 이용한 경우
Table 2. Results obtained when HERMES outputs are used.

| 회로 | No option | | | -m option | | | -s option | | | -s -m option | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AND | XOR | 시간 | AND | XOR | 시간 | AND | XOR | 시간 | AND | XOR | 시간 |
| 5xp1 | 69 | 42 | 0.99 | 73 | 43 | 0.82 | 66 | 37 | 0.78 | 100 | 52 | 0.68 |
| 9sym | 156 | 86 | 6.62 | 401 | 313 | 14.72 | 162 | 85 | 4.92 | 57 | 32 | 8.52 |
| bw | 201 | 156 | 3.12 | 203 | 158 | 2.78 | 204 | 155 | 3.27 | 155 | 83 | 3.59 |
| con1 | 24 | 13 | 2.84 | 40 | 31 | 2.83 | 36 | 13 | 0.14 | 38 | 13 | 0.14 |
| con1 | 17 | 7 | 0.13 | 17 | 9 | 0.12 | 17 | 7 | 0.10 | 31 | 16 | 0.09 |
| count1 | 6 | 1 | 0.05 | 12 | 3 | 0.05 | 6 | 1 | 0.05 | 14 | 7 | 0.05 |
| count2 | 7 | 2 | 0.06 | 12 | 3 | 0.06 | 7 | 2 | 0.05 | 18 | 9 | 0.05 |
| count3 | 9 | 7 | 0.16 | 9 | 7 | 0.14 | 9 | 7 | 0.12 | 32 | 16 | 0.09 |
| dist | 253 | 134 | 9.77 | 277 | 153 | 7.97 | 272 | 141 | 12.16 | 315 | 184 | 11.57 |
| f51m | 42 | 33 | 0.68 | 45 | 34 | 0.56 | 45 | 32 | 0.70 | 96 | 54 | 0.53 |
| life | 142 | 60 | 3.60 | 288 | 190 | 7.52 | 164 | 59 | 3.52 | 41 | 18 | 6.77 |
| m181 | 61 | 18 | 26.23 | 63 | 20 | 25.99 | 80 | 25 | 0.89 | 94 | 33 | 2.8 |
| mult3 | 36 | 16 | 0.17 | 38 | 19 | 0.17 | 36 | 16 | 0.15 | 61 | 25 | 0.14 |
| mult4 | 166 | 85 | 4.27 | 177 | 94 | 3.61 | 169 | 88 | 3.31 | 217 | 115 | 2.72 |
| radd | 24 | 19 | 0.33 | 21 | 19 | 0.28 | 66 | 29 | 0.70 | 123 | 53 | 0.58 |
| rd53 | 18 | 14 | 0.15 | 19 | 18 | 0.14 | 22 | 14 | 0.14 | 32 | 15 | 0.14 |
| rd73 | 75 | 44 | 1.18 | 94 | 62 | 1.16 | 72 | 45 | 0.91 | 64 | 34 | 0.74 |
| root | 111 | 57 | 1.50 | 130 | 67 | 1.52 | 105 | 57 | 1.24 | 163 | 93 | 1.00 |
| root6 | 11 | 8 | 0.08 | 12 | 10 | 0.07 | 11 | 8 | 0.07 | 23 | 11 | 0.07 |
| sqr1 | 17 | 7 | 0.10 | 17 | 9 | 0.09 | 17 | 7 | 0.09 | 16 | 5 | 0.09 |
| sqr6 | 68 | 39 | 0.89 | 73 | 42 | 0.80 | 68 | 40 | 0.83 | 108 | 52 | 0.64 |
| z4 | 33 | 25 | 0.40 | 33 | 28 | 0.34 | 46 | 28 | 0.40 | 91 | 45 | 0.44 |

표 3.은 TRM의 결과를 MRM의 입력으로 사용하는 경우의 결과를 표시한 것이다. flatten의 경우, 같은 benchmark 회로는 표 2.의 경우와 같으므로 생략하였다. 일반적으로 TRM 출력을 사용한 경우가 HERMES 출력을 사용한 경우보다 더 좋은 결과를 내었다. 그 이유는 첫째, TRM 출력의 경우 product 갯수가 더 적고, 또한 TRM의 경우 size가 큰 큐브를 많이 만들어 내기 때문인 것으로 보인다.

표 3. TRM의 결과를 이용한 실험결과
Table 3. Experimental results using TRM results.

| 회로 | No option | | | -m option | | | -s option | | |
|---|---|---|---|---|---|---|---|---|---|
| | AND | XOR | 시간 | AND | XOR | 시간 | AND | XOR | 시간 |
| 5xp1 | 69 | 42 | 0.98 | 73 | 43 | 0.84 | 66 | 37 | 0.77 |
| 9sym | 80 | 32 | 2.04 | 86 | 41 | 1.70 | 88 | 38 | 0.73 |
| bw | 121 | 73 | 1.53 | 129 | 78 | 1.49 | 121 | 73 | 1.44 |
| clip | 199 | 112 | 12.00 | 228 | 128 | 8.31 | 247 | 129 | 10.71 |
| con1 | 18 | 8 | 0.15 | 19 | 11 | 0.14 | 22 | 8 | 0.14 |
| f51m | 76 | 44 | 1.54 | 88 | 56 | 1.38 | 70 | 42 | 1.05 |
| misex1 | 60 | 31 | 0.72 | 65 | 32 | 0.62 | 61 | 33 | 0.56 |
| mul3 | 47 | 20 | 0.30 | 50 | 23 | 0.27 | 47 | 20 | 0.27 |
| mul4 | 223 | 119 | 9.72 | 245 | 128 | 8.44 | 234 | 118 | 9.63 |
| rd53 | 32 | 13 | 0.12 | 11 | 11 | 0.10 | 32 | 13 | 0.10 |
| rd73 | 63 | 32 | 1.01 | 50 | 35 | 0.90 | 65 | 31 | 0.49 |
| rd84 | 107 | 53 | 2.94 | 114 | 75 | 2.77 | 107 | 53 | 1.33 |
| squar5 | 48 | 19 | 0.30 | 44 | 20 | 0.26 | 42 | 22 | 0.29 |
| xor5 | 14 | 7 | 0.06 | 0 | 4 | 0.06 | 14 | 7 | 0.05 |
| z4ml | 55 | 30 | 0.63 | 62 | 39 | 0.50 | 88 | 36 | 0.78 |
| z5xp1 | 87 | 49 | 1.65 | 97 | 58 | 1.66 | 87 | 49 | 1.56 |
| z9sym | 80 | 32 | 2.09 | 86 | 41 | 1.70 | 88 | 38 | 0.73 |

표 5는 MRM의 결과와 Saul의 결과와 비교한 것이다. Saul의 결과보다 많은 경우 향상된 결과임을 볼 수 있다. 여기서는 XOR 게이트의 갯수만을 고려하였다. Saul의 결과는 literal 수를 제시하고 있지만, glue output은 최종 gate layout을 보이므로 literal 수를 직접 구해서 비교할 수는 없었다.

표 4. random data 에 의한 결과
Table 4. Experimental results using random data.

| 회로 | size | | | -s option | | | -s -m option | | |
|---|---|---|---|---|---|---|---|---|---|
| | #in | #out | #prod | AND | XOR | 시간 | AND | XOR | 시간 |
| myex1 | 30 | 3 | 68 | 193 | 36 | 3.4 | 196 | 38 | 2.4 |
| myex2 | 22 | 9 | 87 | 308 | 94 | 12.0 | 310 | 93 | 9.3 |
| myex3 | 25 | 8 | 110 | 367 | 98 | 39.0 | 367 | 98 | 30.0 |
| myex4 | 31 | 5 | 180 | 264 | 123 | 23.1 | 259 | 123 | 13.1 |

표 5. Benchmark 회로에 의한 Saul [7] 의 결과와 비교
Table 5. Comparison of Saul's results by benchmark circuits.

| Function | #PI | #PO | Saul results | MRM results |
|---|---|---|---|---|
| | | | # XOR | # XOR |
| 5xp1 | 7 | 10 | 42 | 37 |
| 9sym | 9 | 1 | 98 | 32 |
| bw | 5 | 28 | 80 | 83 |
| clip | 9 | 5 | 113 | 80 |
| con1 | 7 | 2 | 7 | 7 |
| f51m | 8 | 8 | 34 | 32 |
| rd53 | 5 | 3 | 14 | 14 |
| rd73 | 7 | 3 | 55 | 34 |
| rd83 | 8 | 3 | 82 | 37 |
| sao2 | 10 | 4 | 94 | 60 |
| z4ml | 7 | 4 | 12 | 11 |

## Ⅳ. 결 론

본 논문에서는 다단계 리드뮬러회로를 합성하는 새로운 알고리즘을 제안하였다. 그리고 이 알고리즘을 NeXT Station과 IBM PC Unix(Linux System)에서 C 언어로 구현하였으며, benchmark 회로들에 의한 실험결과와 기존의 다른 문헌에 나타난 결과와 비교하여 좋은 결과를 얻을 수 있었다.

합성방법의 기본은 행렬연산으로 하되, 기존의 방법에서처럼 2n( n:#입력)개의 진리표를 입력행렬로 사용하는 대신 2 단계 최소화 과정의 출력인 큐브형태의 상징적인 논리식을 입력행렬로 사용하므로 입력행

렬의 크기를 작게 할 수 있다. 따라서 행렬크기에 따른 필요 저장공간의 증가, 논리합성 시간의 증가,그리고 소규모회로에만 적용된다는 문제점을 해결하였다.

앞으로 연구되어져야 할 분야로는 최종 출력화일인 다단계 리드뮬러회로를 표현하는 최종 출력화일인 GLUE 화일을 그래픽으로 처리하여 화면에 게이트 레벨의 회로도를 출력하는 방법과 다양한 행렬연산 과정을 개발하기 위해 행렬 연산 과정에서 rank를 구하기 위해 특정한 최적화과정 없이 행과 열 연산을 실행하는 방법 대신에 어떻게 하면 가장 최적화된 연산순서를 결정할 수 있는가 하는 이론적인 연구가 수행되어야 할 것으로 보인다. 또한, 어떻게 2 단계 회로를 합성하면 이를 MRM에서 가장 효과적으로 이용할 수 있는지도 좋은 연구 목표가 될 것으로 본다. 그리고 불완전하게 표현된 함수를 처리하는 알고리즘의 개발과 non- ovelapped decomposition만을 실행하는 행렬연산의 한계를 어떻게 극복하여 ovelapped decomposition까지도 실행할 수 있는지의 연구가 현재 진행중이다.

## 참 고 문 헌


[1] R.K.Brayton et al., "Logic Minimization Algorithms for VLSI synthesis," Kluwer Academic Publishers.

[2] T.T. Hwang, R.M. Owens and M.J. Irwin, "Efficiently Computing Communication Complexity for Multi-level Logic Synthesis," IEEE Trans.on CAD., vol. CAD-11, May. 1992, pp.545-554.

[3] R.K. Brayton et al., "MIS: A Multi-level Logic optimization System," IEEE Transactions on CAD, Nov. 1987, pp.1062-1081.

[4] T.Sasao and P. besslich, "On the complexity of Mod-2 Sum PLA's," IEEE transaction on Computers, Vol.39 NO.2, pp. 262-265, Feb. 1990.

[5] M. Helliwell and M.Perkowski, "A Fast Algorithm to Minimized Multi-output Mixed-Polarity Generalized Reed-Muller Forms," Design Automation Conference, pp.427-432, 1988.

[6] J. Saul, "An Improved Algorithm for the Minimization of Mixed Polarity Reed Muller Representation," ICCD, pp.372-375, 1990.

[7] J. Saul, "An Algorithm for the Multi-level Minimization of Reed-Muller Representations," ICCD 1991.

[8] J. Saul, " Logic Synthesis for Arithmetic Circuits Using the Reed- Muller Representation" Europe Design Automation Conference '93, pp.109-113, 1992.

[9] A. Sarabi and M. Perkowski, "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND /XOR Canonical Networks," Design Automation Conference, pp.30-35, 1992.

[10] S.M. Reddy, "Easily Testable Realization for Logic Functions", IEEE trans. on Computers, vol C-21, No. 11, pp. 1183-1188, 1972.

[11] G.S Lee, J.Y Chang, Robert M. Owens, Mary Jane Irwin, "Synthesis of Multi-level Reed Muller Circuits using Matrix Transformations",IFIP '93 workshop, Hamburg, Germany, Sep. 1993.

[12] G.S Lee, J.Y Chang, Robert M. Owens, Mary Jane Irwin, "Logic Synthesis for Multi-level Reed Muller Circuits using Matrix Transformations", 3rd International Conference on VLSI and CAD, Nov. 1993.

[13] G.S.Lee, M.J.Irwin, and R.M. Owens, "Polynomial-time Testability of Circuits Generated by Input Decomposition," IEEE Transactions on Computer, Feb. 1994.

[14] I. Schaefer et al., "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions," IFIP '93 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, Germany, 42-51,Sep. 1993.

[15] 장준영, 이귀상, "이단계 Reed-Muller 회로의 최소화에 관한 새로운 접근", 전자공학회 논문지 - B, 제 30권 B 편 제 9호 pp.1-8, 1993.9

## 저 자 소 개

**李 貴 相(正會員)**

1958년 2월 1일생. 1980년 2월 서울대학교 전기공학과 졸업(B.S.). 1982년 2월 서울대학교 전산기공학과 졸업(M.S.). 1991 Pennsylvania State University 전산학과(Ph.D.). 1982년 2월 - 1983년 3월 금성통신연구소 연구원. 1983년 4월 - 현재 전남대학교 전산학과 조교수. 주관심분야 : VLSI 설계자동화. 테스팅. 논리합성. 뉴럴네트워크 등임

**張 峻 榮(正會員)**

1962년 8월 20일생. 1985년 2월 전남대학교 전산통계학과 졸업(B.S.). 1987년 8월 중앙대학교 전자계산학과 졸업(M.S.). 1992년 3월 - 현재 전남대학교 전산학과 박사과정 수료. 주관심분야 : VLSI 설계자동화. 테스팅. 논리