論文95-32A-11-12

효율적 Pseudoexhaustive Testing을 위한 다단 논리합성

(Multi-level Logic Synthesis for Efficient Pseudoexhaustive Testing)

李 永 浩 *, 鄭 正 和 *

(Young Ho Lee and Jong Wha Chong)

요 약

본 논문에서는 pseudoexhastive testing이 용이한 다단 논리 회로를 합성하는 새로운 방법을 제안한다. 이 방법은 다음과 같이 4단계로 구성되어 있다. 1 단계에서는 다출력 함수의 각 출력에 대해서 가능한 여러가지의 minimum variable support를 생성한다. 2 단계에서는 다단 논리회로를 구현하는데 사용하면 pseudoexhaustive testing이 어렵게 되는 minimum variable support를 제거한다. 3 단계에서는 각 출력을 구현하는데 알맞은 minimum variable support와 논리 (정논리 또는 부논리)를 결정한다. 4 단계에서는 세번째 단계에서 정해진 minimum variable support와 논리를 이용하여 각 출력을 다단 논리합성한다. 제안된 방법의 유용성을 평가하기 위하여 56개의 벤치마크 예제에 대하여 실험했는데, 실험결과는 본 논문의 방법이 회로의 testability 측면에서 56개의 예제 모두에 대해 MIS 보다 우수하거나 MIS와 대등한 회로를 생성함을 보여준다. 더구나, 본 논문의 방법이 testability를 높이기 위하여 개발되었음에도 불구하고 회로의 면적 측면에서 19개의 예제에 대해 MIS보다 우수한 회로를, 12개의 예제에 대해 MIS와 대등한 회로를 얻는다.

Abstract

In this paper, we present a new multi-level logic synthesis method for producing the multi-level circuits which can be easily tested by the pseudoexhaustive testing techniques. The method consists of four stages. In the first stage, it generates the minimum variable supports for each output of a multiple-output function. In the second stage, it removes the minimum variable supports which if used to implement the outputs, lead to inefficient pseudoexhaustive test. In the third stage, it determines the minimum variable support and logic (uncomplementary or complementary logic) for each output. In the fourth stage, it performs the multi-level logic synthesis so that each output has the minimum variable support and logic determined in the third stage. To evaluate the performance and quality of the proposed method, we have experimented on the 56 benchmark examples. The results show that for 56 examples, our method obtains better results than MIS in terms of testability. Moreover, the method produces better results for 19 examples and the same results for 12 examples compared with MIS in terms of literal count although it has been developed to improve the testability.

* 正會員, 漢陽大學校 CAD 및 通信回路 研究室
(CAD & Communication Circuit Lab., Hanyang Univ.)

接受日字: 1994年8月16日, 수정완료일: 1995年10月26日

Ⅰ. 서 론

VLSI 제조 기술이 발달함에 따라 단일 칩 상에 수백만개의 트랜지스터를 집적할수 있게 되었으며, 앞으로 이러한 추세는 계속될 전망이다. 따라서, VLSI 칩의 설계 및 테스트가 점점 어려워지리라 예상된다. 이러한 문제를 해결하고자 VLSI 칩의 설계 및 테스트 방법에 관한 연구가 활발히 진행되고 있다. 특히, VLSI 칩의 테스트 문제는 그 심각성이 두드러져서 회로 설계시 테스트를 고려하지 않으면 생산된 칩의 테스트가 불가능한 경우도 발생한다.

이러한 상황에 대처하고자 개발된 종전의 해결책은, 설계 사양 - 회로의 기능, 면적, 성능 - 을 만족시키는 회로를 설계한 후 테스트가 용이하도록 회로를 변형하는 것이다. Ad hoc method, Structured method, BIST (Built-In Self Test) 등은 이 부류에 속하는데, Ad hoc method는 설계자의 경험과 판단에 따라 회로의 일부를 테스트가 용이하도록 변형시키는 방식이다. Structured method에는 LSSD (Level Sensitive Scan Desing) [15] 등이 있는데, 회로 내부의 기억소자를 스캔 가능한 기억소자로 교체하여 기억소자들 사이에 흩어져 있는 조합 회로 부분을 기존의 조합회로 테스트 방법을 이용하여 테스트 할 수 있도록 변형시키는 방법이다. BIST는 외부 테스트 장비에 대한 요구사항을 줄이고 테스트 패턴의 생성, 적용 및 응답 분석을 CUT (Circuit Under Test) 자체가 수행하도록 회로를 변형하는 방식이다. 이러한 방식들은 모두 회로의 면적을 증가시키고 회로의 성능을 저하시키는 단점을 가지고 있다.

최근, 회로의 기능 설계후 테스트를 고려하는 방식 대신에 회로의 기능과 테스트를 동시에 고려하여 회로를 설계하는 방식에 대한 연구가 시작되었으며, 이러한 추세에 따른 설계 방법은 크게 순서회로 설계 방법과 조합회로 설계 방법으로 나뉘어 연구되고 있다. 테스트를 고려한 순서회로 설계 방법은 FSM (Finite State Machine) 합성 [14]을 중심으로 이루어지고 있으며, 테스트를 고려한 조합회로 설계방법[6-8][12]도 많이 발표되고 있다. 하지만, VLSI 칩의 설계에 많이 사용되는 다단 논리 회로의 pseudoexhaustive testability를 고려한 논리회로 합성방법은 발표된 바 없다. 이에 본 논문에서는 pseudoexhastive testing이 용이한 다단 논리 회로를 합성하는 새로운 방법을 제안한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 효율적 pseudoexhaustive testing을 수행하기 위하여 논리회로가 갖추고 있어야할 조건을 기술하고, 3, 4, 5, 6장에서는 pseudoexhaustive testing이 용이한 회로를 합성하는 방법을, 그림 1에 주어진 논리함수를 사용하여 단계별로 자세히 기술한다. 7장에서는 56개의 벤치마크 예제에 대한 실험을 통하여 제안된 방법의 실용성을 평가하고, 기존의 다단 논리합성에서도 고려하면 좋을 것으로 기대되는 사항들을 지적한다. 8장에서는 앞으로의 연구 방향을 제시하고, 결론을 맺는다.

```
.type fr
.i 4
.o 3

#xxxx fff
#0123 012
 0000 101
 0001 -11
 0010 10-
 0101 010
 0110 0-1
 0111 -10
 1000 000
 1001 -01
 1011 011
 1101 100
 1110 000
```

그림 1. 4개의 입력과 3개의 출력을 가진 논리함수의 Boolean cover
Fig. 1. Boolean cover of a function with 4 inputs and 3 outputs.

Ⅱ. Pseudoexhaustive Testing

n개의 입력과 m개의 출력을 가지는 논리회로 ($f_0, \ldots, f_{m-1}$)를 테스트하기 위한 가장 간단한 방법은 모든 가능한 입력 패턴을 회로에 인가하고 회로의 응답을 기대되는 응답과 비교하는 것이다. 그러나, 이러한 exhaustive test 방법은 n개의 입력을 갖는 회로에 대해 2^n개의 테스트 패턴을 요구한다. 따라서, 이 방법은 n이 증가하면 테스트 패턴의 수가 폭발적으로 증가하므로 대규모 회로의 테스트에 사용할 수 없다. 하지만, 많은 회로에 있어서 각 출력이 n보다 훨씬 작

은 k개의 입력에 의존한다. 이러한 경우에는 m개의 출력을 $m*2^k$개의 테스트 패턴으로 완벽하게 테스트할 수 있으며, 이런 테스트 방법을 pseudoexhaustive test [9]라 부른다. Pseudoexhaustive test에서 각 출력을 하나씩 테스트할 수도 있지만 모든 출력을 동시에 테스트할 수도 있다. 특히, pseudoexhaustive test 중에서 모든 출력을 동시에 테스트하는 방법을 검증 테스팅 (verification testing) [10]이라 부른다. 가령 parity generator인 74LS630과 같은 회로는 23개의 입력과 10개의 출력을 가지고 있지만, 검증 테스트 방법 [10]에 따라 테스트시에 입력들을 몇 개씩 적당히 연결하여 테스트함으로써 2^{10}개의 테스트 패턴으로 모든 출력을 완벽하게 테스트할 수 있다. 지금까지 pseudoexhaustive test에 대하여 간략히 기술했는데, 자세한 내용은 [9] 및 [10]에서 찾아볼 수 있다.

그런데, 논리 합성시 pseudoexhaustive testability를 고려하지 않으면 pseudoexhaustive test가 용이한 회로가 생성될 가능성이 거의 없다는 사실이 발표되었다[7]. 한 예로서 그림 1의 incompletely specified boolean function으로부터 product term의 수가 최소인 논리회로를 합성한다면 그림 2와 같이 3가지 방법으로 논리회로를 합성할수 있는데 이 3가지 회로는 회로의 면적 측면에서는 동일하지만 pseudoexhaustive testability 측면에서는 그림 2(c)의 회로가 가장 우수하다.

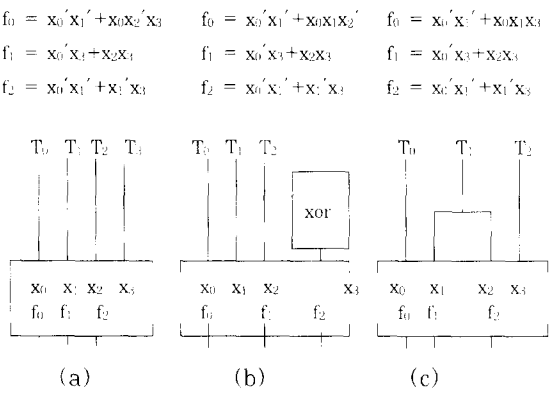


그림 2. 그림 1의 논리함수를 만족하는 3 회로와 검증테스팅 방법

Fig. 2. Three implementations and verification testing schemes. of the function in Fig 1.

위의 예에서 알수 있듯이,

1) 논리회로의 각 출력의 variable support의 크기가 최소이고
2) 논리회로의 출력들의 variable support들의 교집합이 최대

이면 회로의 효율적 pseudoexhaustive test가 가능하다. 문제는 상기의 조건을 어떻게 만족시키는가이다. 3, 4, 5, 6장에서는 pseudoexhaustive testing이 용이한 회로를 합성하는 방법을, 그림 1에 주어진 논리함수를 사용하여 단계별로 자세히 기술한다. 본 논문에서 제안하는 논리합성은 다음과 전체구성으로 되어 있다.

1) 논리함수의 각 출력에 대하여 여러가지 minimum variable support 생성
2) 부적합한 minimum variable support의 제거
3) 각 출력의 논리합성에 사용할 minimum variable support 및 phase의 결정
4) 제약조건하에서의 다단논리회로의 합성

Ⅲ. Minimum Variable Support의 생성

2 장에서는 효율적인 pseudoexhaustive test를 위하여 회로가 가져야할 조건들을 살펴보았는데, 조건 1을 만족시키는 것은 비교적 간단하다. 왜냐하면 minimum variable support를 사용하여 각 출력을 구현하면 되기 때문이다. 하지만, 조건 2를 만족시키기 위해서는 각 출력의 모든 variable support를 구한 후, variable support들간의 교집합을 조사하고 서로 상충되는 조건을 만족시켜야 한다. 그러나, 막대한 계산 시간과 메모리 요구 때문에 각 출력의 모든 variable support를 생성하기도 어렵고, 설령 각 출력의 모든 variable support를 생성했다고 하더라도 서로 상충되는 조건을 만족시키기 힘든다. 따라서, 본 논문에서는 Boolean cover 형태로 주어진 논리함수 ($f_0, \ldots, f_{m-1}$)의 각 출력 f_j에 대하여 f_j의 가능한 여러가지 minimum variable support들을 생성한 후, 조건 2를 가장 잘 만족시킬 것으로 기대되는 minimum variable support를 사용하여 각 출력을 구현한다.

지금까지 다단 논리 회로의 합성에 사용된 mini-

mum variable support 알고리듬[11]은 주어진 출력에 대하여 오직 한개의 minimum variable support를 구하기 때문에, 본 논문에서는 각 출력의 가능한 다수의 minimum variable support들을 생성하는 새로운 알고리듬을 기술한다.

1. 정의

우선 minimum variable support 생성 알고리듬의 기술에 필요한 용어를 정의한다.

정의 1: support sup(f_j)란 출력 f_j가 의존하는 입력변수들의 집합이다. |sup(f_j)|란 sup(f_j)의 원소의 수이다.

정의 2: 출력 f_j를 표현하는데 있어서 입력변수 x_k와 x_k' 가 나타나지 않는 방법으로 f_j를 표현할수 없으면 x_k는 essential input variable이다. 출력 f_j의 essential variable support esup(f_j)란 f_j의 essential input variable들의 집합이다. 출력 f_j를 표현하는데 있어서 x_k와 x_k'가 나타나지 않는 방법으로 f_j를 표현할수 있으면 x_k는 redundant input variable이다. 출력 f_j의 redundant variable support rsup(f_j)란 f_j의 redundant input variable들의 집합이다.

f_j의 주어진 입력변수 x_k가 essential input variable인지 redundant input variable인지는 f_j의 ON-set f_j^{ON}과 OFF-set f_j^{OFF}에서 입력변수 xk를 -(don't care)로 만든 후의 ON-set과 OFF-set의 orthogonality 여부를 조사하면 쉽게 알수 있다. 예를 들면, f_0의 입력변수 x_0가 essential input variable인지 redundant input variable인지를 조사하기 위해서는 f_0의 ON-set과 OFF-set인

$$f_0^{ON} = \begin{matrix}0000\\0010\\1101\end{matrix} \quad 과 \quad f_0^{OFF} = \begin{matrix}0101\\0110\\1000\\1011\\1110\end{matrix}$$

에서 x_0를 -(don't care)로 만든

$$ff_0^{ON} = \begin{matrix}-000\\-010\\-101\end{matrix} \quad 과 \quad ff_0^{OFF} = \begin{matrix}-101\\-110\\-000\\-011\\-110\end{matrix}$$

의 orthogonality를 조사하면 되는데, ff_0^{ON}와 ff_0^{OFF}는 -000, -101에서 공통되는 부분이 존재하므로 orthogonal하지 않다. 따라서, f_0의 입력변수 x_0는 essential input variable이다. 또, f_0의 입력변수 x_2가 essential input variable인지 redundant input variable인지를 조사하기 위해서는 f_0의 ON-set과 OFF-set에서 x_2를 -(don't care)로 만든

$$ff_0^{ON} = \begin{matrix}00\text{-}0\\00\text{-}0\\11\text{-}1\end{matrix} \quad 과 \quad ff_0^{OFF} = \begin{matrix}01\text{-}1\\01\text{-}0\\10\text{-}0\\10\text{-}1\\11\text{-}0\end{matrix}$$

의 orthogonality를 조사하면 되는데, ff_0^{ON}와 ff_0^{OFF}에는 공통되는 부분이 없으므로 orthogonal하다. 따라서, f_0의 입력변수 x_2는 redundant input variable이다. 이와같은 방법으로 그림 1에 주어진 논리함수 f의 각 출력 f_j에 대하여 esup(f_j), rsup(f_j)를 구하면 표 1과 같다.

표 1. 예제 함수의 esup(f_j)와 rsup(f_j)

Table 1. esup(f_j) and rsup(f_j) of the example function.

f_j	esup(f_j)	rsup(f_j)
f_0	{x_0,x_1}	{x_2,x_3}
f_1	{x_0,x_2,x_3}	{x_1}
f_2	{x_0,x_1,x_3}	{x_2}

표1에서 주의할 점은, fj^{ON}와 fj^{OFF}에서 rsup(f_j)에 속하는 입력변수 하나만을 -(don't care)로 만들었을 때는 ffj^{ON}와 ffj^{OFF}가 orthogonal하지만, rsup(f_j)에 속하는 2개 이상의 입력변수를 동시에 fj^{ON}와 fj^{OFF}에서 -(don't care)로 만들었을 때는 ffj^{ON}와 ffj^{OFF}가 orthogonal하지 않을 수도 있다는 것이다. 예를 들면, essential input variable과 redundant variable의 정의에 의하여 rsup(f_0)의 원소중 x_2만

을 -(don't care)로 만들거나 x_3만을 -(don't care)로 만들면 ff_0^{ON}와 ff_0^{OFF}가 orthogonal하지만, x_2와 x_3을 f_0^{ON}와 f_0^{OFF}에서 동시에 -(don't care)로 만든

ff_0^{ON} = 00--
00--
11--

과

ff_0^{OFF} = 01--
01--
10--
10--
11--

는 00--, 11--에서 공통되는 부분이 존재하므로 직교(orthogonal)하지는 않다. 이러한 예로부터 어떤 함수 f_j를 표현하는데는, esup(f_j)에 속하는 입력변수 모두와 rsup(f_j)에 속하는 입력변수들의 일부가 필요함을 알수 있다. 따라서, f_j의 여러가지 minimum variable support들을 생성하는 문제는 rsup(f_j)에 속하는 입력변수를 최소로 사용하여 f_j를 표현하는 문제이다. 즉, rsup(f_j)에 속하는 입력변수들 중에서 f_j의 표현에 사용하지 않아도 되는 입력변수의 수를 최대화하는 문제이다.

정의 3: 출력 f_j의 partially redundant input variable이란 f_j의 minimum variable support의 생성중에 esup(f_j)에 더해지는 rsup(f_j)의 원소이다. 출력 f_j의 partially redundant variable support prsup(f_j)란 f_j의 partially redundant input variable들의 집합이다. 출력 f_j의 totally redundant input variable이란 f_j의 minimum variable support의 생성중에 esup(f_j)에 더해지지 않는 rsup(f_j)의 원소이다. 출력 f_j의 totally redundant variable support trsup(f_j)란 f_j의 totally redundant input variable들의 집합이다.

정의 4: 출력 f_j의 dependency family S_j란 f_j의 minimum variable support들의 집합이고 $|S_j|$란 dependency family S_j의 원소의 수를 나타낸다. 논리함수 $f=\{f_0, f_1, \ldots, f_n\}$의 dependency space S란 논리함수 f의 dependency family S_j의 집합이다.

정의 5: 출력 f_j의 dependency family S_j에 대하여 $rcsup(f_j) = \cap_i \{sup(f_j)_i\}$ 이다.

```
Minimum_Variable_Support(f)
{ /* N : 논리함수 f의 입력 수 */
  for fj∈f {
    fj^ON = ON-set of fj; fj^OFF = OFF-set of fj;
    esup( fj) = essential variable support of fj;
    rsup( fj) = redundant variable support of fj;
    prsup( fj)=φ;trsup( fj)=φ;Dj=φ; Sj=φ;C=N;
    Single_MVS( fj^ON, fj^OFF,esup( fj),rsup
          ( fj),prsup( fj),trsup( fj));
    for s∈ Dj /* s: a variable support */
      if( |s| = C ) Sj = Sj ∪ {s};
}

Single_MVS( fj^ON, fj^OFF,esup( fj),rsup( fj),prsup(
fj),trsup( fj))
{ /* U : 논리함수 f의 입력변수들의 전체집합 */
  if( |esup( fj) ∪ prsup( fj)| > C ) return;
  if( rsup( fj) = φ ) {
   sup( fj) = esup( fj) ∪ prsup( fj);
   Dj = Dj ∪ {sup( fj)};
   if( |sup( fj)| < C ) C = |sup( fj)|;
  }
  else {
   X = Select_TRI ( fj^ON, fj^OFF, esup( fj), rsup
          ( fj),prsup( fj),trsup( fj));
   for x∈X { /* x : 1개의 입력, X : U의 부분집합 */
     ffj^ON = fj^ON; ffj^OFF = fj^OFF;
     ffj^ON와 ffj^OFF에서 입력 x를 -(don't care)로 만든
          다;
     trsup( ffj) = trsup( fj) ∪ {x};
     rsup( ffj) = redundant variable support of ffj;
     prsup( ffj) = U - esup( fj) - trsup( ffj) -
          rsup( ffj);
     Single_MVS( ffj^ON, ffj^OFF,esup( fj),rsup
          ( ffj),prsup( ffj),trsup( ffj));
   }
  }
}
```

2. 알고리듬

3.1절에서 정의를 사용하여 기술한 minimum variable support 생성 알고리듬은 Minimum_Variable_Support이다.

이 알고리듬의 기본 전략은, 각 출력 f_j에 대하여 esup(f_j)와 rsup(f_j)를 생성한후, |prsup(f_j)|가 최소가 되도록 rsup(f_j)의 원소들을 trsup(f_j)와

prsup(f_j)로 나누고, 최종적으로 sup(f_j)$_i$=esup (f_j) ∪prsup(f_j)를 구한다. Single_MVS내의 Select_TRI는 rsup(f_j)로부터 1개 이상의 원소(totally redundant input variable)를 선택한다. 선택방법은 다음과 같다. 선택한 원소(입력변수) x를 fj^{ON}와 fj^{OFF}에서 -(don't care)로 만들었을때 얻어지는 함수 ff_j의 redundant variable support의 크기 |rsup (ff_j)|가 최대가 되도록 totally redundant input variable을 선택한다. 이런 선택방법은 변화하는 rsup(f_j)에 보다 많은 원소를 남겨둠으로써 차후에 제거되는 변수의 수를 최대로 하기 위한 것이다.

그림 1의 함수를 예제로하여 Minimum_Variable_Support 알고리듬을 설명하면, 3, 4 라인의 수행후 각 출력 f_j에 대하여 표1의 esup(f_j)와 rsup(f_j)를 얻은 후, Single_MVS 알고리듬을 call한다. f_0의 minimum variable support를 생성하는 과정을 설명하면 다음과 같다. Minimum_Variable_Support에서는 f_0에 대하여

f_0^{ON} = 0000 f_0^{OFF} =0101 esup(f_0) = {x0,x1} D_j = {}
0010 0110 rsup(f_0) = { x_2, x_3}
1101 1000 prsup(f_0) = {}
1011 trsup(f_0) = {}
1110 C = 4

인 상태에서 Single_MVS 알고리듬을 call한다. 첫번째로 call된 Single_MVS에서는 8 라인의 휴리스틱 함수인 Select_TRI로 X={ x_2, x_3}를 얻은 후, X의 각 원소에 대하여 10-15 라인을 수행한다. 먼저 x_2에 대해서 10-14 라인을 수행하고 난후 다음과 같은 상태

ff_0^{ON} = 00-0 ff^{OFF} = 01-1 esup(f_0) = { x_0,x_1} D0= {}
00-0 01-0 rsup(f_0) = {}
11-1 10-0 prsup(f_0) = { x_3}
10-1 trsup(f_0) = { x_2}
11-0 C = 4

에서 Single_MVS 알고리듬을 다시 (순환적으로) call한다. 두번째로 call된 Single_MVS에서는 rsup (f_0)={}이므로 2-5라인을 수행하여 글로벌 변수인 D_j={{ x_0, x_1, x_3}}, C=3로 update한후 첫번째로 call된 Single_MVS로 복귀한다. 첫번째로 call된 Single_MVS로 복귀하여 x_3에 대해 10-14 라인을 수행한후 다음과 같은 상태

ff_0^{ON} =000 ff^{OFF} =010- esup(f_0)={ x_0,x_1} D_0={{ x_0,x_1,x_3,}}
001- 011- rsup(f_0) = {}
110- 100- prsup(f_0) = { x_3}
101- trsup(f_0) = { x_2}
110- C = 3

에서 Single_MVS 알고리듬을 다시 (순환적으로) call한다. 세번째로 call된 Single_MVS에서는 rsup (f_0)={}이므로 2-5라인을 수행하여 글로벌 변수인 D0={{ x_0,x_1, x_3},{ x_0,x_1, x_2}}로 update한후 첫번째로 call된 Single_MVS로 복귀한다. 첫번째로 call된 Single_MVS로 복귀하고 나면 처리해야할 X의 원소가 더이상 남아있지 않으므로 Minimum_Variable_Support로 복귀한다. Minimum_Variable_Supprt로 복귀하여 7-8라인을 수행하면 S0={{ x_0,x_1, x_3},{ x_0,x_1, x_2}}를 얻는다. 이와 같은 방법을 f_1,f_2에 대해 적용하면 표2의 결과를 얻는다. 표2의 resup (f_j)는, 각 출력 f_j에 대하여 표2에 주어진 sup $(f_j)_i$의 교집합으로 구한 것이다. 예를들면, 표2의 resup (f_0)는 다음과 같은 방법으로 구한다.

$$\begin{aligned} \text{resup}(f_0) &= \text{sup}(f_0)0 \cap \text{sup}(f_0)1 \\ &= \{x_0, x_1, x_2\} \cap \{x_0, x_1, x_3\} \\ &= \{x_0, x_1\} \end{aligned}$$

표 2. 예제 함수의 dependency space S
Table 2. Dependency space S of the example function.

S_0 j,i	prsup(f_j)$_i$	trsup(f_j)$_i$	sup(f_j)$_i$	resup(f_j)	R_{sv}	VTS O	VTS S	VTS M	SUP A	SUP Q	PHASE
0,0	{x_3}	{x_2}	{x_0,x_1,x_3}	{x_0,x_1}	5	3	-	5	-		ON
0,1	{x_3}	{x_2}	{x_0,x_1,x_3}		5		2				
1,0	{}	{x_3}	{x_0,x_1,x_3}	{x_0,x_1,x_3}	6	1	-	3	-		OFF
2,0	{}	{x_3}	{x_0,x_1,x_3}	{x_0,x_1,x_3}	7	2	-	3	-		OFF

VTS: 부적합한 minimum variable support 제거 과정
SUP: minimum variable support의 결정 과정
PHASE: 할당된 출력 phase
O: 출력들이 VTS에서 선택되는 순서
S: VTS에서 variable support가 제거되는 단계
A: 2단 논리 회로의 literal의 수
+: VTS 또는 SUP에서 선택된 variable support

-: VTS 또는 SUP에서 삭제된 variable support
ON: 정논리
OFF: 부논리

Ⅳ. 부적합한 minimum variable support의 제거

여기서는, 전술한 알고리듬에 의해 생성된 minimum variable support sup $(f_j)_i$들 중에서 다단 논리합성에 사용하면 pseudoexhaustive test가 어렵게 되는 minimum variable support를 제거한다. 바람직하지 못한 minimum variable support를 제거하는 알고리듬은 논문 [12]의 입력 분할 알고리듬을 사용하였으므로 본 논문에서는 그 기본 아이디어만을 간략히 기술하고, 표2에 주어진 minimum variable support들을 예제로 pseudoexhaustive test에 부적합한 minimum variable support를 제거하는 방법을 설명한다. 이 알고리듬은 dependency space S로부터 검증 테스팅 (verification testing) 계획을 가상적으로 마련해봄으로써, 효율적 pseudoexhaustive test에 적합하지 않은 minimum variable support를 제거한다.

예를들어 표2의 minimum variable support sup$(f_j)_i$ 중 원소를 최대로 많이 가진 것은 3개의 원소를 가지고 있다. 표2에 주어진 minimum variable support sup $(f_j)_i$는 출력 f_j를 최소의 입력변수로 구현할때 필요한 변수들이므로 그림 1로부터 얻어진 회로의 pseudoexhaustive test에는 적어도 3개의 테스트 신호선이 필요하다. 그래서 표3처럼 테스트 신호선을 우선 3개 만들고, 입력변수들을 이 3개의 테스트 신호선에 분산하여 연결하되, 동일한 테스트 신호선에 연결된 입력변수를 1개 이하씩 포함하는 minimum variable support sup $(f_j)_i$가 표2의 각 출력에 1개 이상 존재하도록 연결한다. 물론, 3개의 신호선만 가지고 이러한 연결이 불가능할 수도 있는데 이때는 테스트 신호선을 1개 증가시킨후, 상기의 과정을 반복한다. 또, 출력이 여러개 존재하고, 각 출력에는 여러개의 minimum variable support가 있어 어느 출력의 어떤 minimum variable support의 원소(입력변수)를 먼저 테스트 신호선에 연결하는가에 따라서 pseudoexhaustive test에 필요한 테스트 신호선의 수가 달라질수 있다. 본 논문에서 사용하는 알고리듬[12]은 다음과 같은 기준을 적용하여 어느 출력의 어떤 minimum variable support의 원소(입력변수)를 먼저 테스트 신호선에 연결할지를 정한다.

1) 우선적으로 처리할 출력
   a) 삭제되지 않고 남아있는 minimum variable support의 수가 최소인 출력
   b) $|\sup(f_j)_i|$가 큰 출력
   c) $|\mathrm{resup}(f_j)|$가 큰 출력

2) f_j의 선택되는 minimum variable support
   a) f_j의 삭제되지 않고 남아있는 minimum variable support 중에서 $R_{j,i}=\sum_k |\mathrm{resup}(f_k)\cap \sup(f_j)_i|$가 최대인 minimum variable support

조건 2에서 $R_{j,i}=\sum_k |\mathrm{resup}(f_k)\cap \sup(f_j)_i|$는 2장에서 기술한 조건 2 (minimum variable support의 교집합이 최대)를 수치화한 것이다. 표 2와 표 3은, 그림 1에 주어진 예제함수의 dependency space S로부터 검증 테스팅 계획을 마련하면서 pseudoexhaustive test에 바람직하지 못한 minimum variable support를 제거하는 과정을 자세히 보여준다.

표 3. 각 단계에서 동일한 검사 신호선에 연결되는 입력 신호들
Table 3. Input signals connected to the same test signals at each step.

테스트 신호선	step1	step2	step3
T_0	$\{x_0\}$	$\{x_0\}$	$\{x_0\}$
T_1	$\{x_2\}$	$\{x_1,x_2\}$	$\{x_1,x_2\}$
T_2	$\{x_3\}$	$\{x_3\}$	$\{x_3\}$

예를들면, 표2의 dependency space를 조사하여 pseudoexhaustive test에 필요한 테스트 신호선이 적어도 3개라는 사실로부터 표3과 같이 테스트 신호선을 3개 만들고, 우선적으로 처리할 출력을 결정한다. 제일 먼저 step1에서 1)의 조건에 따라 f_1을 선택하고, 2)의 조건에 따라 f_1의 sup $(f_1)_0$을 선택한후, sup $(f_1)_0$의 원소(입력변수)를 테스트 신호선 T_0, T_1, T_2에 분산하여 연결한다. 이 상태를 표3의 step1열에 나타냈다. 두번째로 step2에서 1)의 조건에 따라 f_2을 선택하고, 2)의 조건에 따라 f_2의 sup

$(f_2)_0$을 선택한후, sup $(f_2)_0$의 원소(입력변수)를 테스트 신호선 T_0, T_1, T_2에 분산하여 연결하는데, 이때 x_0와 x_3은 이미 테스트 신호선에 연결되어 있으므로 x_1을 T_1에 연결한다. 이것이 가능한 것은 전술한 바와같이 x_1을 T_1에 연결해도 동일한 테스트 신호선에 연결된 입력변수를 1개 이하씩 포함하는 minimum variable support sup(f_1)i가 표2의 각 출력에 1개 이상 존재하기 때문이다. x_1을 T_1에 연결한 후에는 동일한 테스트 신호선에 연결된 입력변수를 2개 이상 포함하는 minimum variable support를 삭제한다. 표2의 예에서는 T_1에 연결된 2개의 입력변수 x_1, x_2를 포함하는 sup $(f_0)_1 = \{x_0, x_1, x_2\}$를 삭제한다. 세번째로 step3에서 1)의 조건에 따라 f_0을 선택하고, 2)의 조건에 따라 f_0의 sup $(f_0)_0$을 선택한후, sup $(f_0)_0$의 원소(입력변수)를테스트 신호선 T_0, T_1, T_2에 분산하여 연결한다.

V. Minimum Variable Support의 결정과 출력 Phase 할당

제거되지 않고 남아있는 minimum variable support라면 어떠한 support를 사용하여 출력을 합성해도 pseudoexhaustive test의 측면에서는 동일한 다단 논리 회로를 얻을 수 있다. 하지만, 어떤 support를 사용하느냐에 따라서 최종적으로 얻어지는 다단 논리 회로의 크기가 변하고, 동일한 support를 사용하여 다단 논리 회로를 실현해도 사용하는 논리(정논리 또는 부논리)에 따라 회로의 크기가 변한다 [13]. 따라서, 각 출력을 구현하는데 알맞은 minimum variable support와 논리를 정할 필요가 있다.

그러나, minimum variable support와 논리 (정논리 또는 부논리)의 모든 조합에 대해서 다단 논리 합성을 해보지 않고는 그 결과를 정확히 예측할 수 없으며, 모든 조합에 대해 다단 논리 회로를 합성하는 것은 막대한 계산시간이 요구되므로 사실상 불가능하다. 따라서, 다음과 같은 휴리스틱 방법을 개발했다. 이 휴리스틱 방법은, 2단 논리 회로 합성을 통해 다단 논리 회로의 크기를 예측함으로써 짧은 시간에 다단 논리 합성에 적합한 minimum variable support와 논리를 결정한다. 그러나, 이 알고리듬은 다단 논리 회로에 미칠 영향을 2단 논리 회로를 근거로 판단하기 때문에 적은 계산시간을 요구하지만 다단 논리 합성에 미칠 영향을 정확히 예측할 수 없다는 단점을 가지고 있다. 그 방법은 다음과 같다.

1) 다단 논리 합성에 사용해도 pseudoexhaustive testability가 저하되지 않는 minimum variable support를 이용하여 각 출력에 대해 2단 논리회로를 구현하되 정논리와 부논리를 사용하여 2단 논리회로를 구현한다.
2) 구현된 2단 논리 회로를 조사하여 리터럴 수가 제일 작은 회로를 각출력에 대해 하나씩 선택한다.
3) 선택된 논리회로들을 취합한다.

상기의 방법에 따라 취합된 회로는 주어진 다출력 함수를 구현한 2단 논리 회로이기 때문에 다단 논리 합성기의 초기해로 사용한다. 표 2는 그림 1에 주어진 논리함수의 각 출력을 구현하는데 사용한 논리와 minimum variable support를 리터럴 수와 함께 표시하고 있다. SUP 열의 +는 다단 논리 합성기의 초기해를 구성하는데 사용한 minimum variable support를 의미하고, PHASE 열의 ON은 해당 출력을 정논리로, OFF는 부논리로 구현했음을 나타낸다. 그림 1의 예제 함수에 대하여, 이와같은 방법으로 취합된 2단 논리 회로가 그림 3에 주어져 있다.

```
.i 4
.o 3
#.phase 100
.p 6
11-1 100
00-- 100
1-0- 010
---0 010
1--0 001
-1-- 001
```

그림 3. 취합된 2단 논리 회로
Fig. 3. Collected two-level circuit.

VI. 제약조건하에서의 다단 논리 합성

여기서는, 전술한 방법에 의하여 얻어진 2단 논리 회로로부터 효율적 pseudoexhaustive testing이 보

장되는 다단 논리 회로를 얻는 방법을 기술한다.

다단 논리 합성 과정에서 pseudoexhaustive testability가 저하되는 것을 방지하기 위해서는 앞에서 결정한 각 출력의 support를 변경하면 안된다. 이를 위해서 본 논문에서는 다단 논리 합성시 algebraic operation만을 한다. 이러한 제약조건은 다단 논리 합성시 Boolean operation의 사용을 금지시키기 때문에 면적이 더 작은 회로를 얻는 기회를 상실하게 만든다. 그러나, Boolean operation을 사용하면 많은 계산시간이 요구되므로 이런 제약 조건은 현실적으로 큰 제약이 되지 않는다. Algebraic operation만을 이용하여 다단 논리 합성하면 그림 3의 2단 논리 회로로부터 그림 4의 다단 논리 회로를 얻는다.

```
INORDER = x0 x1 x2 x4;
OUTORDER = f0 f1 f2;
f0 = x0*x1*x3 + !x0*!x1;
f1 = ![40];
f2 = ![41];
[40] = x0*!x2 + !3x;
[41] = x0*!x3 + x1;
```

그림 4. 최종적으로 얻어진 다단 논리 회로
Fig. 4. Final multi-level circuit.

Ⅶ. 실험 결과

제안된 방법의 유용성을 평가하기 위하여, 3장, 4장, 5장에서 기술한 알고리듬을 Sun4/330상에 C언어로 구현한 후, 56개의 벤치마크 예제 함수에 대하여 실험하고 그 실험결과를 표 4에 나타냈다. pseudoexhaustive test를 고려한 다단 논리 회로의 합성 방법은 아직 발표된 바 없으므로 기존의 다단 논리 회로 합성기인 misII (version 2.2) [13]의 결과와 제안된 알고리듬 Veteran의 결과를 비교하였다. misII의 결과는 espresso [12]를 사용하여 얻은 2단 논리회로로부터 misII -f script.algebraic으로 다단논리회로를 얻었고, Veteran의 결과는 espresso를 사용하여 얻어진, 그림 3과 같이 취합된 2단논리회로로부터 misII -f script.algebraic으로 다단논리회로를 얻었다. 표4의 테스트 신호선의 수는 모든 출력을 동시에 테스트하는 pseudoexhaustive test인 verification test에 필요한 테스트 신호선의 수이다.

표 4. misII와 Veteran의 비교
Table 4. Comparison of misII and Veteran.

논리함수			misII		Veteran	
이름	입력수	출력수	테스트 신호선수	리터럴수	테스트 신호선수	리터럴수
addm6	12	7	12	96	12	96
adr4	8	5	8	48	8	48
alu1	12	8	1	41	4	38
alu2	10	8	10	107	10	99
alu3	10	8	10	106	10	97
apla	10	12	10	140	9	124
bca	26	46	16	1064	16	
bcb	26	39	16	961	16	
bcc	26	45	16	884	16	
bcd	26	38	16	731	16	
bc0	26	11	20	792	20	
chkn	29	7	26	426	26	436
co14	14	1	14	76	14	76
cps	24	109				
dc1	4	7	4	46	4	48
dc2	8	7	7	136	7	114
dist	8	5	8	398	8	391
dk17	10	11	10	93	8	67
dk27	9	9	8	35	4	27
dk48	15	17	15	89	6	56
exep	30	63	30	545	23	
f51m	8	8	8	154	8	161
gary	15	11	15	463	15	551
in0	15	11	15	457	15	564
in1	16	17	15	603	15	1320
in2	19	10	19	477	19	436
in3	35	29	27	344	27	
in4	32	20	31	499	31	
in5	24	14	22	333	22	473
in6	33	23	17	294	17	
in7	27	10	21	135	21	
jbp	36	57	17	497	17	
misg	56	23	15	101	15	
mish	94	43	94	126	8	
mlp4	8	8	8	396	8	400
opa	17	69	17	488	17	
radd	8	5	8	48	8	48
rckl	32	7	32	200	32	196
rd53	5	3	5	62	5	59
rd73	7	3	7	122	7	145
risc	8	31	8	119	7	116
root	8	5	8	189	8	171
sqn	7	3	7	137	7	101
sqr6	6	12	6	168	6	148
ti	47	72	47	912	24	
tial	14	8	14	1222	14	760
vg2	25	8	25	97	25	
wim	4	7	4	42	4	27
x1dn	27	6	27	101	27	157
x2dn	82	56	82	218	25	
x6dn	39	5	38	339	36	451
x7dn	66	15	66	369	28	
x9dn	27	7	27	101	27	160
z4	7	4	7	42	7	42
Z5xp1	7	10	7	168	7	143
Z9sym	9	1	9	207	9	306

표 4에 나타난 바와같이, 본 논문의 방법은 회로의 pseudoexhaustive testability 측면에서 56개의 예제 모두에 대해 misII보다 우수하거나 misII와 대등한 회로를 생성한다. 더구나, 본 논문의 방법이 testability를 높이기 위하여 개발되었음에도 불구하고 회로의 면적 측면에서 19개의 예제에 대해 misII보다 우수한 회로를, 12개의 예제에 대해 misII와 동일한 회로를 얻는다.

특히, 논리 함수 "mish"에 대하여 misII는 veri-

fication test를 위해 94개의 테스트 신호선, 즉 2^{94}개의 테스트 패턴을 요구하는 다단 논리 회로를 생성하는 반면, 제안된 방법은 8개의 테스트 신호선, 즉 2^{8}개의 테스트 패턴만으로 verification test할 수 있는 다단 논리 회로를 생성한다. 그러나, pseudoexhaustive testability 측면에서 본 논문의 방법이 misII보다 항상 우수한 다단논리회로를 생성하지는 않는다. 왜냐하면 어떤 함수, 예를들면 addr6의 경우에는 어떤 출력의 minimum variable support의 원소의 수가 입력변수의 수와 동일하기 때문이다.

또, 어떤 함수, 예를들어 x6dn에 대해서는 Veteran도 pseudoexhautive test하기 힘든 회로를 생성하므로 partitioning technique [11]을 적용하여 테스트 신호선의 수를 더욱 줄여야만 효율적 pseudoexhaustive test를 수행할 수 있다. 그러나, partitioning technique을 필요로하는 다단회로를 Veteran이 misII보다 적게 생성한다. 따라서, Veteran에 의해서 생성된 다단회로를 pseudoexhaustive test하는데 필요한 테스트 신호선의 수가 현저하게 줄어든 경우에는 리터럴 수가 다소 늘어났다하더라도 회로의 면적 측면에서 Veteran의 결과가 misII의 결과에 비하여 나쁘다고 평가하면 안된다. 왜냐하면, 많은 테스트 신호선을 필요로 하는, misII에 의해서 생성된 회로를 효율적으로 pseudoexhaustive test하기 위해서는 partitioning technique [11]을 적용해야 하는데, 이렇게하면 segmentation 셀이 증가하여 결국 회로의 면적이 증가하기 때문이다.

제안된 알고리듬은 testability를 개선하기 위하여 개발되었음에도 불구하고 몇몇 회로에 대하여 회로의 면적 측면에서 misII보다 우수한 결과를 생성한다. 이러한 결과는 다음과 같은 사실에 기인한다. 첫째, 제안된 알고리듬은 2단 논리 회로의 리터럴 수를 근거로 다단 논리 회로의 각 출력을 구현하는데 사용할 논리와 minimum variable support를 결정한다. 둘째, 기존의 다단 논리 합성 툴이 각 출력에 대해서 오직 한개의 minimum variable support를 생성하여 다단 논리 합성을 행하는 것과는 달리, 본 논문의 방법은 여러 minimum variable support를 생성한 후 논리 회로의 리터럴 수를 고려한다. 따라서, 기존의 다단 논리 합성기에서 회로의 면적을 최소화하기 위하여 각 출력에 대해서 여러 minimum variable support를 생성한후 다단 논리 합성에 사용하는 minimum variable support에 따른 리터럴 수의 증감을 조사할 필요가 있는 것으로 추측된다.

Ⅷ. 결 론

본 논문에서는 pseudoexhastive testing이 용이한 다단 논리 회로를 합성하는 새로운 방법을 제안하고 56개의 벤치마크 회로에 대한 실험을 통하여 제안된 알고리듬의 유용성을 입증했다. 또, 실험결과를 분석함으로써 기존의 다단 논리 합성 알고리듬을 개발함에 있어서 고려해야 할 사항을 제시했다.

앞으로의 연구과제는 효율적 peudoexhaustive testability와 회로의 면적 최소화를 위한 FSM 합성 방법에 대한 연구와 본 논문의 연구 결과를 BIST와 결합하여 순서회로에 적용하는 것이다.

감사의 글

본 논문이 나오기까지 염려하며 격려해주신 대전산업대학교 이재홍 교수님과 실험을 위해 필요한 것을 제공해주신 한국전자통신연구소의 배영환님께 감사드립니다.

※ 본 연구는 91년도 한국 과학 재단의 연구비 지원을 받아 수행되었음. 과제 번호: 91-01-00-09

참 고 문 헌

[1] R. K. Brayton, "Algorithms for Multi-level Logic Synthesis and Optimization," in G. De Micheli, A. Sangiovanni-Vincentelli and P. Antognetti, *Design Systems for VLSI Circuits*, Martinus Nijhoff Publishers, Boston, 1987.

[2] R. K. Brayton, G. D. Hactel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.

[3] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli and A. R. Wang, "MIS: A Multiple-level Logic

Optimization System," *IEEE Trans. on Computer-Aided Design*, pp. 1062-1081, Nov. 1987.

[4] S. Devadas and K. Keutzer, "A Unified Approach to the Synthesis of Fully Testable Sequential Machines," *Proc. 23rd Annual Hawaii International Conference*, pp. 427-435, Jan. 1990.

[5] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testing," *Proc. 14th Design Automation Conference*, pp. 462-468, Jun. 1977.

[6] A. Krasniewski, *Verification Testing Oriented Logic Synthesis*. Technical Report no. 135, Warsaw Univ. of Technology, Institute of Telecommunications, Sept. 1989.

[7] A. Krasniewski, "Design for Verification Testability," *Proc. 1st European Design Automation Conference*, pp. 664-648, Mar. 1990.

[8] A. Krasniewski, "Logic Synthesis For Efficient Pseudoexhaustive Testability," *Proc. 28th Design Automation Confe rence*, pp. 66-72, Jun. 1991.

[9] E. J. McCluskey and S. Bozorgui-Nesbat, "Design For Autonomous Test," *IEEE Trans. on Computers*, pp. 866-875, Nov. 1981.

[10] E. J. McCluskey and S. Bozorgui-Nesbat, "Verification Testing - A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers*, pp. 561-546, Nov. 1984.

[11] Rajagopalan Srinivasan, Sandeep K. Gupta and Melvin A. Breuer, "An Efficient Partitioning Strategy for Pseudo-Exhastive Testing," *Proc. 30th Design Automation Conference*, pp. 242-248, Jun. 1993.

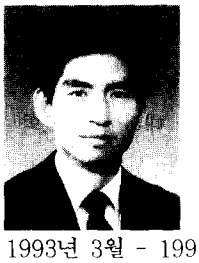
[12] 이영호, 정정화, "검증 테스팅을 위한 새로운 설계방법," 대한전자공학회 논문지, 제 29권 A편 제 2호, pp.91-98, 1992년 4월

[13] 이영호, 정정화, "SHARE: 효율적인 출력 PHASE를 이용한 PLA 최소화," 대한전자공학회 논문지, 제 30권 A편 제 12호, pp.1-9, 1993년 12월

저 자 소 개

李 永 浩 (正會員)

1965년 8월 7일생. 1989년 2월 한양대 전자공학과 학사학위 취득. 1991년 2월 한양대 전자공학과 석사학위 취득. 1994년 2월 한양대 전자공학과 박사과정 수료. 1993년 2월 제 2회 젊은 공학도를 위한 반도체 워크샵 장려상 수상. 현재 한양대 산업과학연구소 연구원. 주관심분야는 VLSI 설계 및 CAD 특히, Logic Synthesis, Testing, 화상처리 시스템 설계 등임.



鄭 正 和 (正會員)

1950년 3월 10일생. 1975년 2월 한양대 전자공학과 학사학위 취득. 1977년 2월 한양대 전자공학과 석사학위 취득 1981년 3월 와세다 대학 전자공학과 박사학위 취득. 1993년 3월 - 1995년 2월 CAD 및 VLSI 설계연구회 위원장. 현재 한양대 전자공학과 교수. 주관심분야는 VLSI 설계 및 CAD 특히, ASIC Emulator, MPEG 회로설계 등임.