論文93-30A-11-16

디지탈 CMOS VLSI의 범용 Test Set 분할 생성 알고리듬

(Divided Generation Algorithm of Universal Test Set for Digital CMOS VLSI)

金 東 郁

(Dong Wook Kim)

要 約

CMOS회로의 고집적도는 CMOS회로의 특성에 의한 FET고장(Stuck-On고장과 Stuck-Off고장)과 더불어 설계및 제조과정에 필요한 test비용을 기하급수적으로 증가시키고 있다. 본 논문에서는,디지탈 CMOS회로의 설계및 제조과정에서 필요한 여러 test단계를 일원화하고 static회로와 dynamic회로에 공히 적용할 수 있는 test set을 임의의 크기의 회로에 대해 용이하게 생성할 수 있는 알고리듬을 제안한다. 이 알고리듬은 임의의 크기의 회로를 계층적 또는 수평적으로 회로분할하고 분할된 각 자회로를 입력재원으로 사용한다. 또한 이 알고리듬은 분할된 자회로들 중 출력측에서 생성된 test set을 입력측으로 구동시키는 방법을 사용하고 있으며 구동시 구동되는 자회로와 연결된 자회로만을 구동재원으로 사용한다. 따라서 이 알고리듬은 VLSI와 같은 대형회로의 test비용을 상당히 절감시킬 수 있으리라 기대된다.

Abstract

High Integration ratio of CMOS circuits incredibly increases the test cost during the design and fabrication processes because of the FET faults(Stuck-on faults and Stuck-off faults) which are due to the operational characteristics of CMOS circuits. This paper proposes a test generation algorithm for an arbitrarily large CMOS circuit, which can unify the test steps during the design and fabrication procedure and be applied to both static and dynamic circuits. This algorithm uses the logic equation set for the subroutines resulted from arbitrarily dividing the full circuit hierarchically or horizontally. Also it involves a driving procedure from output stage to input stage, in which to drive a test set corresponding to a subcircuit, only the subcircuits connected to that to be driven are used as the driving resource. With this algorithm the test cost for the large circuit such as VLSI can be reduced very much.

Ⅰ. 서 론

극소 전력소모와 고집적 용이성의 CMOS회로 설계및 제조공정에서 요구되는 test비용은 회로의 집적도가 증가함에 따라 기하급수적으로 증가하고 있다.[1] 더우기 CMOS회로의 동작특성(dual동작)[2]은 Stuck-Off(S-Off)고장과 Stuck-On(S-On)고장[3]의 새로운 FET고장 발생으로 CMOS회로의 testability를 더욱 저하시키고 있다. 집적도의 증가는 test생성에 있어 그복잡도(Complexity)[4]를 기하급수적으로 증가시키며, test비용의 증가율은 회로의

* 正會員,光云大學校 電子材料工學科
(Dept of Elec. Materials Eng., Hwangwoon Univ.)

接受日字 : 1992年 12月 21日

증가율에 비해 더 한층 가속되고 있다. 또한, FET고장은 각 FET에 발생하는 영구고장의 형태이어서 그 test는 FET-레벨에서 취급되어져야 하는 바 설계와 제조공정상에서 요구되는 게이트-레벨 또는 functional 레벨의 simulation/test에 필요한 입력 벡터들은 따로 생성되어져야 한다.

FET-레벨과 게이트-레벨의 test를 모두 수행할 수 있는 test set은 본 논문의 저자에 의해 [2] 에 제안되었으며, 본 논문에서는 이 test set을 임의의 크기의 회로에서 용이하게 생성할 수 있는 새로운 알고리듬을 제안한다. 대형회로를 가정, 그 회로를 분할하여 test생성 알고리듬을 적용하는 방법은 [5] [6] 등에서 제안되었으나, path tracing [5] 이나 group function 또는 nodal function의 합성과정 [6] 에서 결국 회로전체를 고려하여야 하므로 알고리듬의 복잡도를 크게 향상시키지 못한다. 본 논문에서는 임의의 대형회로를 임의적으로 수직적(계층적) 또는 수평적으로 분할하여, 분할된 각 자회로(subcircuit)을 이용, 전체회로를 참조하지 않고 전체회로의 test set을 생성하는 알고리듬을 제안한다. 이 알고리듬은 [2] 에서 제안된 test set을 생성하며 이 test set의 목적인 FET-레벨 test와 게이트-레벨의 test의 일원화를 위하여 각 자회로의 논리함수만을 알고리듬의 입력재원으로 사용한다. 본 논문에서는 분할된 각 자회로가 최소화(irreucible and irredundant)된 것으로 가정하며, 고려되는 고장은 FET-레벨 고장(S-Off고장과 S-On고장)과 게이트-레벨 고장(Stuck-At 고장: S-A고장)이고, 단일고장 발생을 가정한다.

Ⅱ. Test 패턴 Set (R3P4T)

본 장에서는 [2] 에서 제안된 디지탈 CMOS회로의 test set을 간략히 요약한다. 이 test set은 세 개의 test패턴을 한 개의 test단위로 사용하며 그 구성은 (T1,T2,T1)의 형태를 취한다. 이 test단위는 양방성을 가지므로 (T1,T2,T1)과 (T2,T1,T2)는 test수행에 있어 그 효과가 동일하다. 이 test 단위는 그 방향성에 상관없이 〈T1,T2〉 또는 〈T2,T1〉으로 나타내며 한 개의 test단위를 Robust Three Patterns for Four Tests(R3P4T)라 한다. [2] [7] 한 R3P4T의 T1과 T2는 단일 bit만 서로 다른 값을 가지며 그 bit는 test되는 FET의 입력변수에 해당한다.

R3P4T에 의한 test는 FET-레벨에서 단일 FET에 대해서가 아니라 CMOS회로의 특징인 두 개의 dual network에서 서로 대응되는 두 개의 FET를 동시에 고려하여 수행된다. 한 개의 R3P4T는 네 개의 FET고장(두 개의 dual FET에 대해 각 FET의 S-Off고장과 S-On고장)과 해당 게이트-레벨 회로에서 해당 신호선의 S-A-0고장과 S-A-1고장을 test한다. 한 R3P4T의 T1 또는 T2는 고장이 발생한 FET(FET-레벨) 또는 지점(게이트-레벨)에서 관찰가능한 출력으로 단일 경로를 연결 또는 차단하는 One Path Activation and Deactivation(OPAD) 또는 One Gate Activation and Deactivation (OGAD)의 성질을 갖고 있으며 이로 인해 Complex-gate 회로와 Primitive-gate 회로뿐만 아니라 일반적인 cascade형태의 회로에 R3P4T의 적용이 가능하다. 또한 이 OPAD 또는 OGAD의 성질은 R3P4T의 두 패턴에 의해 연결-차단되는 두 경로상에서 cascade되는 FET들을 cascade시키는 FET의 R3P4T로 test할 수 있게 하며 따라서 cascade시키는 FET들의 R3P4T들로 전체 회로를 test할 수 있다. Cascade시키는 FET들에 대한 정보는 주어진 논리함수의 논리식에 모두 포함되어 있으므로 주어진 회로의 R3P4T set은 그 회로의 논리식만으로 생성가능하다. 결국 R3P4T는 회로의 설계와 제조과정에서 게이트 회로나 FET 회로의 참조없이 생성가능하며 논리식은 게이트 회로나 FET 회로 보다 선행되는 과정이므로 R3P4T를 게이트 회로와 FET 회로에 모두 사용가능하다.

이 R3P4T의 구성과 존재여부는 [2] 의 정리, 조건, 보조정리에 나타나 있으며 R3P4T 생성 조건으로는 고려되는 회로가 최소화되어 있다는 것이다.

Ⅲ. R3P4T의 분할 생성 알고리듬

Ⅱ장에서 설명한 R3P4T set은 [1] 에서 그 생성 알고리듬을 제안하였다. 그러나 이 알고리듬은 입,출력 수가 그리 많지 않은 비교적 간단한 회로에서 더욱 효과적으로 사용될 수 있으며, 만약 VLSI와 같은 대형회로의 경우 알고리듬의 run time과 memory 용량등에 의해 제약을 받을 수 있다. 따라서 본 장에서는 임의의 대형회로에 효과적으로 적용할 수 있는 R3P4T 생성 알고리듬을 제안한다.

VLSI와 같은 대형회로는 주로 전체회로를 여러 개의 자회로로 분할(수평적 분할)하거나 계층적인형태로 분할(수직적 분할)하여 설계하게 된다. 그러나 CMOS회로의 기본형태는 다입력 일출력이므로 본 논문에서는 분할된 각 자회로가 다입력 일출력의 형태를 갖는 것으로 가정한다. 예로써 그림 1에 다입력 일출력의 형태로 수평분할된 74169(Synchronous 4-bit Up/Down Binary Counter) 회로를 나타내

었다. 이 회로는 sequential회로이므로 sequential 동작을 제어하기 위하여 LSSD [8] 의 Double-Latch 설계방식을 택하고 있다. 따라서 74169의 모든 D플립플롭을 [8] 의 LSSD 래치로 대치하였으며 LSSD 래치의 출력들(Q1, Q2, Q3, Q4)은2차입력(Secondary Inputs)으로써 test수행시 test입력단자로 사용된다. 이 회로는 네 개의 단(stage)로 나누어 지며 18개의 자회로(B10에서 B45)로 분할되었다. 분할된 각 자회로에 대한 CMOS FET회로는 [7] 에 나타나 있다.

R3P4T 생성 알고리듬은 그림 2에 나타낸 바와 같이 두 개의 procedure({R3P4T_GEN}와 {R3P4T_DRV})와 한 개의 결정단계(decision maker)로 구성된다. {R3P4T_GEN}은 R3P4T의 생성 procedure이며 {R3P4T_DRV}는 R3P4T의 driving procedure이다. 이 두 procedure는 각각 세 개씩의 subroutine({R3P4T_GEN}은 {T1_GEN}, {SER_T1_GEN}, {PAR_T2_GEN}, {R3P4T_DRV}는 {DRV_VAR}, {DRV_PAT}, {DRV_DRV})을 갖고 있으며 {R3P4T_GEN}은 [2] [9] 에 상세히 나타나 있다. 이 알고리듬의 적용방법은, 먼저 전체회로의 출력단의 각 자회로에 대하여 {R3P4T_GEN}을 적용하여 출력단의 각 자회로의 R3P4T set을 구한 다음 이 R3P4T set을 출력단에서 입력단으로 drive하게 된다. Driving procedure는 한 개의 단을 기준으로 하여 수행되며 입력변수가 모두 외부변수(external variable)일 때까지 계속된다.

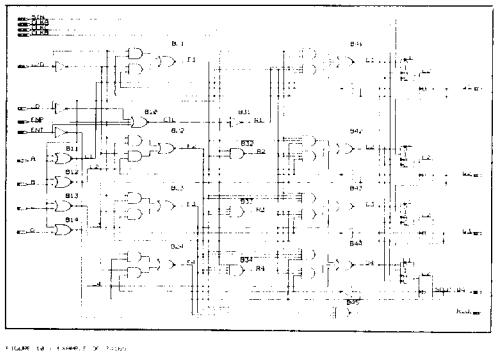

그림 1. 분할 생성 알고리듬을 위한 예제(74169)
Fig. 1. Example Circuit(74169) for Divided Generation Algorithm.

Ⅱ장에서 밝힌 바와 같이, 이 알고리듬은 대형회로의 test생성 용이성 뿐 아니라 설계및 제조과정중 test절차의 일원화 또한 그 목적이므로 {R3P4T_GEN} 뿐만 아니라 {R3P4T_DRV}의 입력재원으로 분할된

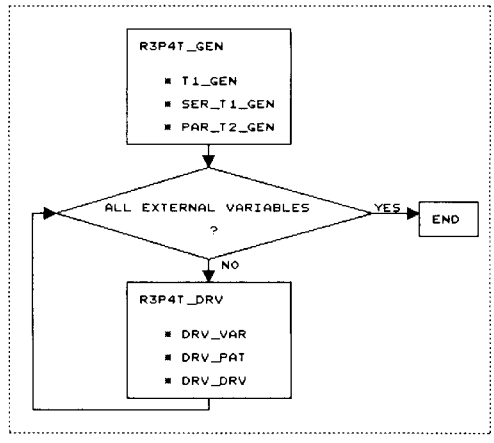

그림 2. 분할 생성 알고리듬의 구성
Fig. 2. Structure of Divided Generation Algorithm.

자회로들의 논리식들을 사용한다. 그림 1의 예제에 대한 논리식의 set을 표 1에 나타내었다.

표 1. 그림 1 회로의 자회로들에 대한 논리식
Table 1. Logic Equation Set for the Subcircuits of the Circuit in Fig. 1.

```
<STAGE-1>
!CTL =!LD + ENP + ENT
!L1  = LD + A
!L2  = LD + B
!L3  = LD + C
!L4  = LD + D

<STAGE-2>
!F1  = U/D*Q1 + !U/D*!Q1
!F2  = U/D*Q2 + !U/D*!Q2
!F3  = U/D*Q3 + !U/D*!Q3
!F4  = U/D*Q4 + !U/D*!Q4

<STAGE-3>
!R1  = CTL
!R2  = CTL*F1
!R3  = CTL*F1*F2
!R4  = CTL*F1*F2*F3

<STAGE-4>
!D1  = CTL*Q1 + R1*LD*!Q1 + L1
!D2  = CTL*F1*Q2 + R2*LD*!Q2 + L2
!D3  = CTL*F1*F2*Q3 + R3*LD*!Q3 + L3
!D4  = CTL*F1*F2*F3*Q4 + R3*LD*!Q4 + L4
!RCO = F1*F2*F3*F4*!ENT

KEY : ! = NOT, * = AND, + = OR
```

표 2. 그림 1 회로에 대한 driving 경로
Table 2. Driving Path Set for the Circuit in Fig. 1.

PAT_B41 -> DRV_11			DRV_13
{B41}	{B31, B11}		{B10}
PAT_B42 -> DRV_21	DRV_22		DRV_23
{B42}	{B32, B12}	{B21}	{B10}
PAT_B43 -> DRV_31	DRV_32		DRV_33
{B43}	{B33, B13}	{B21, B22}	{B10}
PAT_B44 -> DRV_41	DRV_42		DRV_43
{B44}	{BN34, B14}	{B21, B22, B23}	{B10}
PAT_B45		DRV_52	
{B45}		{B21, B22, B23, B24}	

Driving의 전 단에서 drive된 결과는 다음 drive 과정의 입력으로 사용되며 따라서 회로 분할시 drive되는 자회로의 수가 최소가 되게 회로를 분할하면 이 drive과정을 보다 효율적으로 수행할 수 있다. 그림 1의 예제에 대한 driving 경로는 표 2에 나타내었다. 이 표의 경로는 각 {R3P4T_GEN} 또는 {R3P4T_DRV}의 결과로 생성된 ASCII file로 표시되어 있으며 각 과정에서 사용된 자회로는 중괄호로 표시하였다.

{R3P4T_GEN} procedure와 이에 속한 세 개의 subroutine을 그림 2, 3, 4, 5에 각각 나타내었다.

```
PROCEDURE {R3P4T_DRV}

Objective : Driving R3P4Ts backward to input stage
Inputs : 1. Logic Equations for the Variables to be Driven
         2. R3P4Ts from Non-hierarchical Algorithm
Outputs : All possible R3P4Ts

begin
    until all input variables are external
    begin
        for each set of R3P4Ts,
        begin
            Call {DRV_VAR} ;
            Call {DRV_PAT} ;
            Call {DRV_DRV} ;
        end
    end
end
```

그림 3. {R3P4T_DRV} Procedure
Fig. 3. {R3P4T_DRV} Procedure.

```
SUBROUTINE {DRV_VAR}

begin
    for each variable (A) which is not external in R3P4Ts ,
    begin
        Find corresponding logic equation whose output is A ;
        Find all possible input combinations to make A logic 0 ;
        Put them into the list (DRV_A_0) ;
        Find all possible input combinations to make A logic 1 ;
        Put them into the list (DRV_A_1) ;
    end
end
```

그림 4. {DRV_VAR} Subroutine
Fig. 4. {DRV_VAR} Subroutine.

```
SUBROUTINE {DRV_PAT}

begin
    for each combination of variables (ABC = ijk) which are
            not external in R3P4Ts,
    begin
        Combine the lists (DRV_A_i, DRV_B_j, DRV_C_k) of
            corresponding variables and values by applying the
            operator # for each common variable ;
        if conflict,
            Delete the pattern ;
        if no conflict,
            Put it into the list (DRV_P_ijk) ;
    end
end
```

그림 5. {DRV_PAT} Subroutine
Fig. 5. {DRV_PAT} Subroutine.

```
SUBTOUTINE {DRV DRV}

begin
    for each R3P4T in the R3P4T set,
    begin
        Replace the external variables with the list (DRV P ijk)
        for each pair of T1 and T2,
        begin
            for each variable,
            begin
                Apply the operator # between T1 and T2 ;
            end
            if only one conflict,
                Put the pair of T1 and T2 into the set of new
                    R3P4Ts for the conflicted variable ;
            else,
                Delete the pair of T1 and T2 ;
        end
    end
end
```

그림 6. {DRV_DRV} Subroutine
Fig. 6. {DRV_DRV} Subroutine

R3P4T의 driving procedure는,

{그 전 과정의 R3P4T Set} ∩ {R3P4T Driving 조건} (1)

을 수행하는데, R3P4T driving 조건은,

1. Drive되는 변수에 해당하는 자회로에서 drive되는 변수를 유발하는 입력조건
2. Drive되는 변수가 두 개 이상일 경우 drive되는 자회로들의 공통변수들이 서로 다른 값을 가지지(Conflit) 않을 조건
3. Drive될 자회로들의 입력변수 조합에서 T1과 T2간의 conflict 조건은 그 변수에 해당하는 단일 conflict

로 요약될 수 있으며, 이 세 조건은 {DRV_VAR}, {DRV_PAT}, {DRV_DRV}에서 각각 수행된다. {R3P4T_DRV}의 주 procedure에서는 {R3P4T_GEN} 또는 전 과정의 {R3P4T_DRV}의 결과를 입력으로 받아들여 세 개의 subroutine을 제어한다.

{DRV_VAR}은 조건 1을 수행하는데 그방법은 drive되는 변수가 출력인 자회로에서 그 변수를 논리-0과 논리-1로 만드는 각각의 입력조건을찾는 것이다. 조건 2는 {DRV_VAR}의 결과로 {DRV_PAT}에서 수행되는데, 조건 2와 3의 처리를 위해서 다음의 연산자를 이용한다.

#	0	1	X
0	0	$	0
1	$	1	1
X	0	1	X

표 3. 그림 1의 회로에 대한 한 R3P4T Set
Table 3. One R3P4T Set for the Circuit in Fig. 1.

T1												T2											
U/D	L	E	E	A	B	C	D	Q	Q	Q	Q	U/D	L	E	E	A	B	C	D	Q	Q	Q	Q
	D	N	N	T				1	2	3	4		D	N	N	T				1	2	3	4
X	0	1	X	1	X	X	X	0	X	X	X	X	1	1	X	1	X	X	X	0	X	X	X
X	0	1	X	0	X	X	X	1	X	X	X	X	1	1	X	0	X	X	X	1	X	X	X
X	0	0	0	1	X	X	X	1	X	X	X	X	1	0	0	1	X	X	X	1	X	X	X
X	1	0	0	X	X	X	X	0	X	X	X	X	0	0	0	X	X	X	X	0	X	X	X
X	1	1	0	X	X	X	X	1	X	X	X	X	1	0	0	X	X	X	X	1	X	X	X
X	1	0	1	X	X	X	X	1	X	X	X	X	1	0	0	X	X	X	X	1	X	X	X
X	0	X	X	1	X	X	X	X	X	X	X	X	0	X	X	0	X	X	X	X	X	X	X
X	1	0	0	X	X	X	X	0	X	X	X	X	1	0	0	X	X	X	X	1	X	X	X
X	1	1	X	X	X	X	X	1	X	X	X	X	1	1	X	X	X	X	X	0	X	X	X
1	1	0	0	X	X	X	X	1	1	X	X	0	1	0	0	X	X	X	X	1	1	X	X
0	1	0	0	X	X	X	X	1	0	X	X	1	1	0	0	X	X	X	X	1	0	X	X
0	1	0	0	X	X	X	X	0	1	X	X	1	1	0	0	X	X	X	X	0	1	X	X
1	1	0	0	X	X	X	X	0	0	X	X	0	1	0	0	X	X	X	X	0	0	X	X
0	0	0	0	X	1	X	X	1	1	X	X	0	1	0	0	X	1	X	X	0	0	X	X
X	1	1	X	X	0	X	X	1	X	X	X	X	0	1	X	X	0	X	X	X	1	X	X
0	0	0	0	X	1	X	X	1	1	X	X	0	1	0	0	X	1	X	X	1	1	X	X
1	1	0	0	X	X	X	X	0	0	X	X	1	1	1	0	X	X	X	X	0	0	X	X
X	0	X	X	X	1	X	X	X	X	X	X	X	0	X	X	X	0	X	X	X	X	X	X
1	1	0	0	X	X	X	X	1	1	X	X	1	1	0	0	X	X	X	X	0	1	X	X
0	1	0	0	X	X	X	X	0	1	X	X	0	1	0	0	X	X	X	X	1	1	X	X
0	1	0	0	X	X	X	X	1	0	X	X	0	1	0	0	X	X	X	X	1	1	X	X
X	1	1	X	X	X	X	X	1	X	X	X	X	1	1	X	X	X	X	X	0	X	X	X
1	1	0	0	X	X	X	X	1	0	1	X	0	1	0	0	X	X	X	X	1	1	1	X
0	1	0	0	X	X	X	X	1	1	0	X	1	1	0	0	X	X	X	X	1	0	0	X
1	1	0	0	X	X	X	X	0	1	1	X	0	1	0	0	X	X	X	X	1	1	1	X
0	1	0	0	X	X	X	X	1	1	0	X	1	1	0	0	X	X	X	X	0	1	0	X
0	1	0	0	X	X	X	X	0	1	1	X	1	1	0	0	X	X	X	X	0	0	1	X
0	0	X	X	X	X	1	X	0	X	0	X	0	1	X	X	X	X	1	X	0	X	0	X
X	1	1	X	X	X	0	X	X	X	1	X	X	0	1	X	X	X	0	X	X	X	X	X
0	1	0	0	X	X	X	X	1	0	1	X	0	1	0	0	X	X	X	X	1	1	1	X
0	1	0	0	X	X	X	X	1	1	0	X	1	1	0	0	X	X	X	X	1	1	1	X
0	1	0	0	X	X	X	X	1	1	0	X	0	1	0	0	X	X	X	X	0	1	0	X
1	1	0	0	X	X	X	X	1	0	0	1	0	1	0	0	X	X	X	X	1	1	1	1
0	1	0	0	X	X	X	X	1	1	1	0	1	1	0	0	X	X	X	X	1	1	1	0
1	1	0	0	X	X	X	X	0	1	0	1	0	1	0	0	X	X	X	X	1	1	1	1
0	1	0	0	X	X	X	X	1	1	1	0	1	1	0	0	X	X	X	X	1	1	1	0
1	1	0	0	X	X	X	X	0	0	1	1	0	1	0	0	X	X	X	X	1	1	1	1
0	1	0	0	X	X	X	X	1	1	1	0	1	1	0	0	X	X	X	X	1	1	1	0
0	1	0	0	X	X	X	X	0	1	1	1	1	1	0	0	X	X	X	X	0	0	1	1
0	0	X	X	X	X	X	X	1	0	X	X	0	1	X	X	X	X	1	X	0	X	0	X
X	1	1	X	X	X	X	X	0	X	X	X	X	0	1	X	X	X	X	X	0	X	X	X
0	0	0	0	X	X	1	1	1	1	1	1	0	1	0	0	X	X	X	X	1	1	1	1
0	1	0	0	X	X	X	X	1	1	1	0	0	1	1	0	X	X	X	X	0	1	1	0
X	0	X	X	X	X	1	X	X	X	X	X	0	1	0	0	X	X	X	X	1	1	1	1
0	1	0	0	X	X	X	X	1	1	1	0	0	1	0	0	X	X	X	X	1	1	1	1
0	1	X	X	X	X	X	X	0	X	X	1	0	1	X	X	X	X	X	X	0	X	X	0
1	0	X	X	X	X	X	X	1	0	0	0	0	0	X	X	X	X	X	X	1	1	1	1
1	0	X	X	X	X	X	X	0	1	0	0	0	0	X	X	X	X	X	X	1	1	1	1
1	0	X	X	X	X	X	X	0	0	1	0	0	0	X	X	X	X	X	X	1	1	1	1
1	0	X	X	X	X	X	X	0	0	0	1	0	0	X	X	X	X	X	X	1	1	1	1
0	0	X	X	X	X	X	X	0	1	1	1	1	0	X	X	X	X	X	X	0	0	0	0
0	1	X	X	X	X	X	X	1	1	1	1	0	0	X	X	X	X	X	X	1	1	1	1
1	0	X	X	X	X	X	X	0	0	0	1	1	0	X	X	X	X	X	X	0	0	0	0
0	0	X	X	X	X	X	X	1	1	1	0	0	0	X	X	X	X	X	X	1	1	1	1

여기서 0(1)은 논리-0(1)을, X는 don't care 조건을 각각 나타내며, $는 conflict를 의미한다. 이 연산자는 drive되는 변수에 해당하는 자회로의 모든 입력변수에 대해 적용되며 이 subroutine에서는 conflict가 허용되지 않는다. 즉 {DRV_PAT}에서는 {DRV_VAR}의 결과에 따른 각각의 입력조건을 전단계의 driving과정 또는 {R3P4T_GEN}의 결과로 conflict없이 결합한다. {DRV_PAT}의 결과는 {DRV_DRV}에 사용되고 자회로의 입력변수들에 대한 새로운 R3P4T set의 조건은 {DRV_DRV}에서 적용된다. {DRV_DRV}에서도 연산자 #가 사용되나, 여기서는 II 장의 R3P4T 조건에서 밝힌 바와 같이 한 개의 입력조합에 대해 각 변수에 해당하는 bit에서만의 conflict가 존재하여야 한다. {DRV_DRV}의 출력은 해당 {R3P4T_DRV}의 출력이며 {DRV_DRV} 출력을 구성하는 입력변수가 모두 외부변수일 때까지 {R3P4T_DRV}를 반복한다. 최후의 {R3P4T_DRV} 결과는 주어진 회로전체에 대한 가능한 모든 R3P4T set이며 실제의 test입력은 그 중 한 set에 해당한다. 따라서 CMOS회로의 기본형태인 다입력 일출력의 형태에 따라 주어진 임의의 대형회로를 분할하여 이 알고리듬에 적용하면 전체회로에 대한 가능한 모든 R3P4T set을 구할 수 있다. 예로써 그림 1의 예제 회로에 대해 본 논문의 알고리듬을 적용하여 구한 R3P4T의 한 set을 표 3에 나타내었다. 이 set은 58개의 R3P4T로 구성되며 X는 don't care 조건을 의미한다.

Ⅳ. R3P4T의 Dynamic회로에의 적용

집적도의 증가는 한 IC 칩내에 포함되는 회로의 양을 증가시키고 있으며 따라서 회로전체에 대한 신호의 흐름에 있어 timing의 어려움 또한 큰 문제점으로 대두되고 있다. 이러한 문제의 해결책으로회로의 동기화 방식을 많이 사용하는데 그 방법은 주로 dynamic회로의 형태를 취한다. 통용의 dynamic회로는 전체회로를 다수의 자회로로 분리하여 실현하는 것이므로 본 논문의 취지인 test의분할 생성에 맞추어 본 장에서는 CMOS dynamic회로에 R3P4T를 적용하는 방법을 다룬다.

[9] [10] [11] [12] 에서 보는 바와 같이 dynamic회로는 주로 여러 개의 회로단(stage)이 cascade된 형태를 이루고 회로실현의 방식에 따라 필요한 회로를 부가하게 되는데 dynamic회로의 단은 그림 7에 나타낸 두 가지가 사용된다. Dynamic 회로는 한 개 또는 여러 개의 clock(CLK)주기에 의

해 입력에서 출력으로 신호가 전달되며 각 단은 precharge(PR)와 evaluate(EV)의 두 동작이 한 CLK 주기동안 수행된다. 그림 7의 두 stage는 CLK=0일 때 n(p)MOS 단의 출력이 논리 1(0)로 PR되고 CLK=1이 되면 입력조건에 따라 출력의 논리값이 결정되는 EV동작이 수행된다. 지금까지 발표된 dynamic CMOS회로는 이 두 논리단을 사용하고 있으므로 본 논문에서는 이 두 단에의 R3P4T 적용방법에 대해서만 언급한다(기본 dynamic회로형태, Domino CMOS 회로, NORA회로, Zipper CMOS 회로 등의 실제회로에 대한 적용은 [7] 참조).

그림 7에 나타낸 두 논리단은 CMOS회로의 기본형태인 dual network 대신 단일 network과 PR FET(MP), EV FET(ME)로 구성된다. 이 두 회로의 생략된 logic block을 재고려하여 modeling하면 nMOS 논리단의 경우 MP와 평행하게 pMOS logic block을 부가하고 pMOS 논리단 에는 MP에 직렬로 nMOS logic block을 삽입하여 CMOS의 가본형태와 같이 구성할 수 있다. 재구성된 회로의 논리식은,

$$F = CLK' + f(inputs) \quad (2)$$

$$F' = CLK \cdot f'(inputs) \quad (3)$$

로 된다. 여기서 f(inputs)은 해당 논리단에 대한 논리함수를 나타낸다. 식 (2)와 (3)은 CMOS회로의 기본형태를 그대로 나타내고 있으므로 dynamic회로 역시 R3P4T의 적용이 가능하다. 그러나 modeling을 위해 부가된 logic block의 test는 불필요하므로 R3P4T의 적용방법에 있어 수정이 요구된다. Ⅱ장에서 설명한 바와 같이 R3P4T는 두 개의 dual FET를 기본단위로 적용된다. 그러나 dynamic회로의 한 논리단은 단일 network으로 구성되어 있어 R3P4T의 기본단위인 세 패턴을 모두 적용할 경우 패턴의 redundant가 발생하여 그 효율이 상당히 저하된다. 따라서 본 논문에서는 R3P4T의 세 패턴중 두 패턴(T1과 T2)만 적용하는 방법을 제시한다. 또한 이 방법은 회로의 정상동작과 test동작의 일치성을 위해 test적용시 PR와 EV의 동작을 변화시키지 않는다. Ⅱ장에서 설명한 R3P4T의 OPAD성질은, static회로에서 T1과 T2가 각각 한 개 씩의 신호경로를 도통시키는 것에 반해, dynamic회로에서는 T1과 T2가 모두 한 개의 경로에만 관계를 하며 두 패턴중 한 패턴은 그 경로를 연결시키고 다른 패턴은 차단시키는 동작을 한다. 그러나 dynamic회로는 단일network만 포함하므로 단일경로의 연결과 차단만으로 test를 충분히 수행할 수 있다.

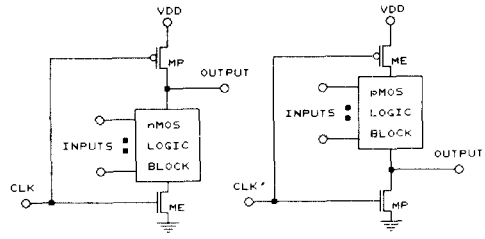

그림 7. Dynamic CMOS 회로의 두 논리단
(a) nMOS 논리단
(b) pMOS 논리단

Fig. 7. Two Logic Stages of Dynamic CMOS Circuits.
(a) nMOS Logic Stage,
(b) pMOS Logic Stage.

먼저 logic block에 있는 FET를 고려한다. PR기간 동안 그 단의 출력은 precharge되고 CLK의 변화로 EV기간으로 천이한다. 이 때 test되는 FET (FET Under Test: FUT)에 S-On고장이 발생하였다고 가정하자. EV기간 이 전에 이 FET에 대한 R3P4T의 T1을 가하면, 정상동작시 해당 logic block의 모든 경로가 차단되어 PR된 논리치를 출력이 보유하여야 하나 S-On고장에 의해 그 logic block에 경로가 연결되어 출력은 PR된 것과 반대의 논리치를 갖게 된다. 이 논리치는 OPAD의 성질에 의해 외부출력(visible output)으로 전달되어 S-On 고장이 검출된다. FUT에 S-Off고장이 발생한 경우는 T1 대신 T2를 적용한다. 정상동작시 T2는 logic block의 전류경로를 연결하여 그 단의 출력을 PR된 것과 반대의 논리치로 변화시킨다. 그러나 S-Off고장이 발생한 경우에는 이 전류경로가 S-Off고장에 의해 차단되고 출력은 PR된 논리치를 보유하게 된다. 이 결과는 OPAD성질에 의해 외부출력으로 전달되어 이 고장은 검출가능하다. 따라서 한 쌍의 (T1, T2)로써 해당 FET의 S-On고장과 S-Off고장이 검출되며, static회로에서와 달리 dynamic회로에서는 logic block내의 FET S-On고장에 대한 검출방법이 전류측정(current monitoring)이 아니라 전압측정(voltage monitoring)으로 수행된다. Logic block내의 FET에 대한 test수행절차는 표 4에 요약하여 나타내었다.

MP와 ME를 test하기 위하여 입력변수의 논리치를 PR기간의 초기에 적용하는 것으로 가정한다. 먼

표 4. Logic Block내의 FET에 대한 Test
Table 4. Test for an FET in a Logic Block.

	TEST PATTERNS		COVERED FAULTS	
	CLK	VARIABLES	S-OFF	S-ON
EACH FET IN A LOGIC BLOCK	PRECHARGE	CORR. T1		⟨V⟩
	EVALUATE	"		
	PRECHARGE	CORR. T2	⟨V⟩	
	EVALUATE	"		

⟨V⟩ : VOLTAGE MONITORING

표 5. MP와 ME에 대한 Test
Table 5. Tests for MP and ME.

	TEST PATTERNS		COVERED FAULTS	
	CLK	VARIABLES	S-OFF	S-ON
MP & ME FOR A LOGIC BLOCK	PRECHARGE	CORR. T2	ME⟨V⟩	ME⟨I⟩
	EVALUATE	"		MP⟨I⟩
	PRECHARGE	CORR. T1		
	EVALUATE	"	MP⟨V⟩	

⟨V⟩ : VOLTAGE MONITORING
⟨I⟩ : CURRENT MONITORING

저 MP의 S-Off고장(MP/Off)를 가정한다. MP/Off는 일단 stage의 출력이 PR된 논리치의 반대로 EV되면 그 후는 PR동작을 수행할 수 없게 한다. 따라서 이 고장의 test는 출력을 PR된 논리치와 반대로 초기화한 다음 PR동작을 수행하여 고장의 효과를 출력에 유발시키고 그 다음의 EV기간동안 외부출력으로 전달하여야 한다. 이는 T2로써의 EV, T1으로써의 PR, T1으로써의 EV로 수행될 수 있다. MP의 S-On고장(MP/On)은 logic block에 전류경로가 연결된 경우의 EV기간동안 정전류가 VDD에서 GND로 흐르게 된다. 따라서 MP/On은 T2를 적용한 EV기간 동안 전류측정을 수행함으로써 test할 수 있다. ME/Off는 evaluate 경로를 항상 차단시킴으로써 논리단의 출력을 PR된 논리치로 고정시킨다. 따라서 이 고장은 PR동작을 수행한 후 logic block에 전류경로를 연결하는 패턴으로써의 EV로 test되며 이는 T1패턴으로써의 PR과 EV로 가능하다. ME/On은 logic block에 경로를 연결시키는 패턴을 적용한 PR기간동안 정전류를 흐르게 하므로 T2에 의한 PR기간 동안 전류측정으로 test할 수 있다. MP와 ME에 대한 test절차는 표 5에 요약하여 나타내었다.

표 4와 5에 나타난 바와 같이 R3P4T는 static회로 뿐 아니라 dynamic회로에도 적용가능하며 단지 static회로에서 세 패턴 모두를 적용하는 것과는 달

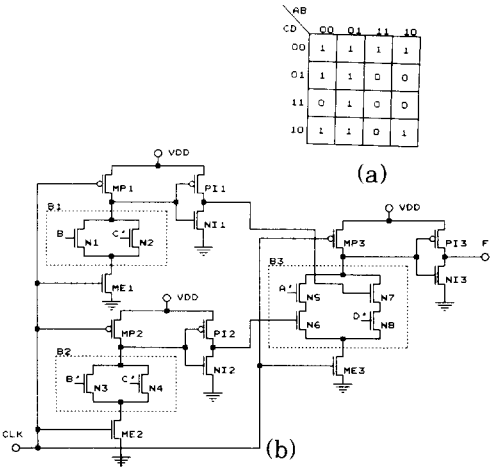

그림 8. Domino CMOS의 예제회로
(a) 논리함수
(b) Domino CMOS로 실현
Fig. 8. Example Circuit of Domino CMOS.
(a) Logic Function,
(b) Domino CMOS Implementation.

표 6. 그림 8의 회로에 대한 Test
Table 6. Tests for the Circuit in Fig. 8.

FETs	TEST PATTERNS		COVERED FAULTS	
	CLK	ABCD	S-OFF	S-ON
N1	PRECHARGE	0011		N1,P2 ⟨V⟩
	EVALUATE	0011		
	PRECHARGE	0111	N1,P2,ME1,PI1 ⟨V⟩	ME1,PI1⟨I⟩
	EVALUATE	0111	MP1 NI1 ⟨V⟩	MP1,NI1⟨I⟩
N2	PRECHARGE	0011		N2,P2 ⟨V⟩
	EVALUATE	0011		
	PRECHARGE	0001	N2,P2 ⟨V⟩	
	EVALUATE	0001		
N3	PRECHARGE	1110		N3,P3 ⟨V⟩
	EVALUATE	1110		
	PRECHARGE	1010	N3,P3,ME2,PI2 ⟨V⟩	ME2,PI2⟨I⟩
	EVALUATE	1010	MP2 NI2 ⟨V⟩	MP2,NI2⟨I⟩
N4	PRECHAGRE	1110		N4,P3 ⟨V⟩
	EVALUATE	1110		
	PRECHARGE	1100	N4,P3 ⟨V⟩	
	EVALUATE	1100		
N5	PRECHARGE	1001		P1⟨V⟩
	EVALUATE	1001		
	PRECHARGE	0001	P1,ME3,PI3 ⟨V⟩	ME3,PI3⟨I⟩
	EVALUATE	0001	MP3 NI3 ⟨V⟩	MP3,NI3⟨I⟩
N8	PRECHARGE	0011		P4⟨V⟩
	EVALUATE	0011		
	PRECHARGE	0010	P4⟨V⟩	
	EVALUATE	0010		

⟨V⟩ : VOLTAGE MONITORING
⟨I⟩ : CURRENT MONITORING

리 dynamic회로에서는 두 패턴(T1,T2)에 대해 정상동작과 같이 PR과 EV를 각 패턴에 적용한다. 또한 Domino회로의 Inverter [10] , NORA회로의 clocked CMOS stage [11] , Zipper CMOS의 clock driving 회로 [12] 도 MP 또는 ME의 test와 유사한 방법으로 test할 수 있으며 [7] , 따라서 모든 dynamic CMOS회로에 R3P4T가 적용가능하다. 단, 표 4에 나타난 것과 같이 일반적으로 T1과 T2의 적용순서를 임의로 했을 경우 최소 두 개 이상의 FET가 한 logic block에 존재하여야 한다. 이는 논리함수의 분할시 충분히 해결될 수 있는 문제이므로 Ⅲ장에서 제안된 알고리듬은 static회로 뿐 아니라 dynamic회로에 대해서도 적용할 수 있음을 알 수 있다.

예로써 그림 8에 간단한 Domino CMOS 회로를 나타내었다. 이 회로에 대한 Ⅲ장의 알고리듬 결과는 표 6에 나타나 있으며, 이 R3P4T들을 적용하는 방법은 표 7에 표 4와 5를 기준으로 나타내었다.

Ⅴ. 결 론

VLSI와 같은 대형 CMOS 디지탈 회로에 대한 test 생성의 용이성을 위하여 본 논문에서는 전체회로를 임의의 수의 다입력 일출력 자회로로 분할하여 test를 생성하는 알고리듬을 제안하였다. 이 알고리듬의 결과는 전체회로에 대한 Robust Three Patterns for Four Tests(R3P4T)라는 새로운 형태의 test set이며 이 R3P4T set은 세 패턴을 test 단위로 하고 한 개의 R3P4T는 dual network의 두 dual FET에 대해 적용된다. 이 알고리듬의 입력재원으로는 분할된 자회로들에 대한 논리함수의 set이 사용되며 이는 R3P4T의 목적인 FET-레벨의 test와 게이트-레벨의 test를 일원화하기 위함이다. 종래에 발표된 방식과는 달리 본 논문에서 제안된 방법은 전체회로의 참조없이 drive되는 자회로들의 논리식만을 사용하여, 출력단에서 생성된 R3P4T set들을 입력단으로 drive하므로 알고리듬의 복잡도(Complexity)가 상당히 저하될 수 있으며 한 번에 처리하여야 하는 data양이 적어 memory등의 hardware적인 제약 또한 상당히 완화될 수 있다.

분할 설계의 예로써 dynamic CMOS회로에 대해 static CMOS를 가정하여 제안된 본 논문의 알고리듬을 적용하여 dynamic CMOS회로에서의 적용가능성을 보였다. Static 회로에서와는 달리 dynamic회로에서는 dual network 대신 단일 network으로 회로가 구성되므로 R3P4T의 세 패턴중 두 패턴만 필요하며 static 회로에서의 R3P4T 성질(OPAD)이 dynamic회로에서도 그대로 적용되므로 본 논문에서 제안한 알고리듬은 static 회로 뿐 아니라 dynamic 회로에 대해서도 적용할 수 있다. Dynamic 회로에 있어 각 logic block은 최소 두 개 이상의 FET를 포함하여야 하며, 분할된 각 자회로는 최소화된 것으로 가정하였다.

參 考 文 獻

[1] Jonah McLeod, "Growth spurt opens the door for U.S. VLSI test system markets," *J. Electronics*, pp 89-101, 1988.

[2] 김 동욱, "디지탈 CMOS 회로의 Multi-Level Test를 위한 범용 Test Set 생성," 대한전자공학 회 논문지, 제 30권, A편, 제 2호, pp 64-75, 1993년 2월.

[3] R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and nMOS Integrated Circuits," The Bell System Technical Journal, pp 1449-1474, May-June 1978.

[4] Sreejit K. Chakravarty, "On the Complexity of Computing Tests for CMOS Gates," *IEEE Trans.* Computer-Aided Design, Vol.8, N0.9, pp 973-980, Sept. 1989.

[5] Kuang-Wei Chiang, Zvonko G. Vranesic, "On Fault Detection in CMOS Logic Networks," IEEE/ACM Design Automation Conference, pp 50-57, 1983.

[6] Chantal Vivier, Georges Fournie, "Automatic Modeling of MOS Transistor Networks for Test Pattern Generation," *IEEE Test Conference*, pp 340-348, 1986.

[7] Dong-Wook Kim, "CMOS Digital Circuit Test Generation for Transistor-Level and Gate-Level Implementations," ph.D. Dissertation, Georgia Institute of Technology, July 1991.

[8] E. B. Eichelberger, T. W. Williams, "A Logic Design Structure for LSI Testability," IEEE Design Automation

Conference, pp 462-468, 1979.

[9] David J. Myers, Peter A. Ivey, "A Design Style for VLSI CMOS," *IEEE J. Solid State Circuits*, vol.SC-20, no.2, pp 741-745, April 1985.

[10] Jacoubs A. Pretorius, et. al., "Analysis and Design Optimization of Domino CMOS Logic with Application to Standard Cells," *IEEE J. Solid-State Circuits*, vol.SC-18, no.2, pp 523-530, June 1983.

[11] Nelson F. Concalves, Hugo J. DeMan, "NORA: A Racefree Dynamic CMOS Techniques for Pipelined Logic Structure," *IEEE J. Solid-State Circuits*, vol.SC-18, no.3, pp 261-266, June 1983.

[12] Charles M. Lee, Ellen W. Szeto, "Zipper CMOS," *IEEE Circuits and Devices Magazine*, pp 10-16, May 1986.

著 者 紹 介

金 東 郁(正會員)

1983年 2月 한양대학교 전자공학과 졸업. 1985年 2月 한양대학원 전자공학과 졸업. 1991年 9月 Georgia Institute of Technology (Georgia 공과대학) Electrical Engineering (전기과)에서 박사학위 취득. 1992年 3月 부터 현재 광운대학교 전자재료공학과 재직중. 주관심분야는 Digital VLSI System. VLSI Testability 등임.