특 집

상위 테스트합성 기술의 개발 동향

신 상 훈, 박 성 주
한양대학교 전자계산학과

요 약

시스템을 단일 칩에 구현함에 따라서 반도체 칩은 수백만 게이트를 내장할 정도로 고집적화 되어 가고 있다. 이러한 고집적도의 칩을 제작하는 데 소요되는 고가의 테스트비용을 최소화하기 위해 설계의 각 단계 별로 다양한 테스트설계기술이 개발되고 있다. 합성 후 회로구조가 테스트에 용이하도록 하기 위하여 상위 및 논리 합성 단계에서 테스트기능을 추가하고 있다. 합성된 회로에 대하여는 스캔, 테스트점 삽입, 및 BIST 등의 테스트설계 기술이 사용되고 있다. 본 논문에서는 VHDL 등으로 기술되는 상위 기능정보와 상위 구조합성 과정에서 고려되고 있는 다양한 테스트합성 기술을 소개하고자 한다.

I. 상위 테스트합성 기술의 개요

반도체 제품의 수명은 계속 짧아지고 초고집적화 되어감에 따라서 빠른 시간에 체계적으로 설계할 수 있는 VHDL을 이용한 상위설계 기술이 널리 사용되고 있다. 상위설계는 Verilog, VHDL, 및 C언어 등의 HDL로 기술하여 시뮬레이션하고 상위구조 합성 및 논리합성 과정을 거친다. 상용화된 테스트합성 도구는 HDL을 컴파일한 후 기술의존적인 게이트 혹은 기술 독립적인 게이트회로를 생성해 준다. HDL로 기술된 정보에 테스트기능을 추가하여 전체 기능회로 및 테스트회로를 최적화 시키는 연구가 활발히 진행되고 있다. 상위수준에서 부분스캔 구현시 세 개의 플립프롭만 스캔하면 되는데 상위 합성 후 생성된 게이트 회로에서는 이십 개의 플립프롭을 스캔해야 동일한 점검도를 이룰 수 있는 경우가 보고되었다[1]. 상위수준에서의 테스트합성은 물리적 결함에 대한 기능적 고장모델의 설정이 어렵지만 제어부 및 데이터 통로 등의 기능정보와 구조정보가 복합적으로 포함되어 있다. 반면에 게이트 수준에서는 물리적 결함에 대한 다양한 고장모델을 설정할 수 있지만 기능정보가 배제되어 있어서 테스트설계의 최적화를 이룰 수가 없다. 즉, 게이트수준에서 행해지고 있는 대부분의 체계적인 테스트설계 기술은 쉽게 적용할 수는 있지만 상위수준 정보와의 결합 없이는 최적의 방법이라고 볼 수 없다. 〈표 1〉은 상용 CAD 도구에서 회로합성의 여러 단계에서 테스트기능을 삽입하는 예를 보여주고 있다[2]. 어떤 도구는 테스트합성을 위하여 칩 및 모듈단위의 네트리스트 정보를 필요로 하고, 어떤 도구는 기능정보, RTL, 기술독립적, 기술의존적인 정보 등을 필요로 한다.

상위 구조합성은 스케쥴링과 바인딩 두 단계로 구성되어 있다. 예를 들어 〈그림 1〉과 같은 미분방정식 해석기를 VHDL로 설계했다면 컴파일 후에 〈그림 2〉와 같은 Control Data Flow Graph (CDFG)가 생성된다. CDFG에서 연산자간의 데이터 의존도가 실선 화살표(edge)로 표시되며 점선 화살표는 제어도를 나타낸다[3]. CDFG는 수행시간 및 자원을 고려하여 〈그림 3〉과 같이 스케쥴

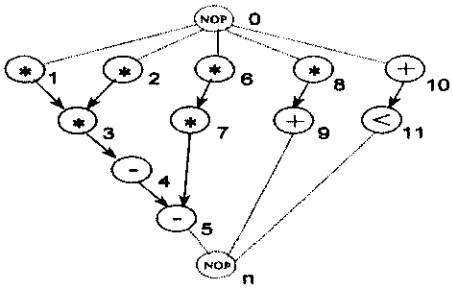
* 본 연구는 정보통신연구단의 대학기초연구지원사업으로 수행하였습니다.

〈표 1〉 상용 CAD 도구에서 테스트 삽입 기능

이 름	합성 기반	테스트 기능 삽입 수준
Mentor	Autologic II	technology-independent
LogicVision	Synopsys HDL&Design Compiler, BIST 합성기	HDL
IBM	Booledozer	tech-independent or tech-dependent
Synopsys	Synopsys HDL&Design Compiler, Viewlogic(Sunrise)	HDL and technology dependent
Compass	ASIC Synthesizer	technology dependent
AT&T	Synovation	HDL and technology dependent

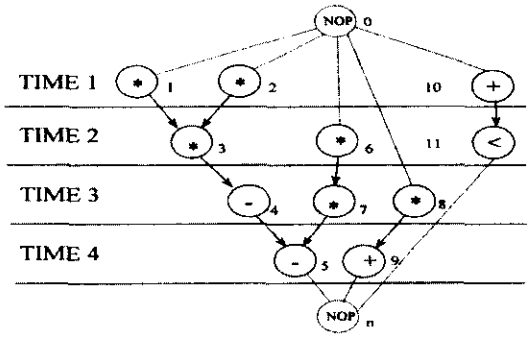
```
package mypack is
  subtype bit8 is integer range 0 to 255;
end mypack;

use work.mypack.all;
entity DIFFEQ is
prot(    dx_port, a_port, x_port, u_port : in bit;
         y_port : inout bit8;
         clock, start : in bit );
end diffeq;

architecture BEHAVIOR of DIFFEQ is
begin
process
         variable x, a, y, u, dx, xl, ul, : bit8;
begin
         wait until start'event and start = '1';
         x := x_port; y := y_port; a := a_port;
         u := u_port; dx := dx_port;
         DIFFEQ_LOOP
         while ( x < a ) loop
                  wait until clock'event and clock = '1';
                  xl := x+dx;
                  ul := u-(3*x*u*dx)-(3*y*dx);
                  yl := y+(u*dx);
                  x := xl; u := ul; y := yl;
         end loop DIFFEQ_LOOP;
         y_port <= y;
end process;
end BEHAVIOR;
```

〈그림 1〉 미분방정식 해석기의 VHDL 코드

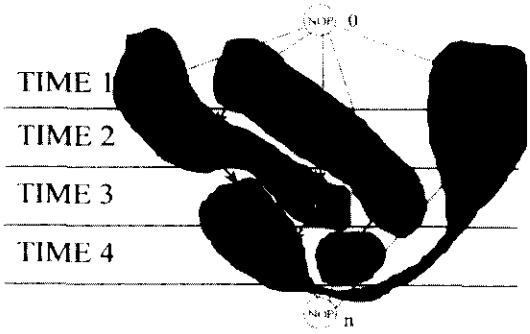


〈그림 2〉 미분방정식 해석기의 CDFG

링하고 〈그림 4〉와 같이 바인딩한다. 바인딩 시에는 각 변수가 레지스터를 최대한으로 공유할 수 있도록 하고 여러 연산자가 자원을 공유할 수 있도록 한다. 이상의 상위합성을 통하여 〈그림 1〉의 VHDL 회로를 구현하는데는 11개의 연산에 대하여 곱셈기 2개, ALU 2개, 및 10개의 변수에 대한



〈그림 3〉 스케쥴링 된 미분방정식 해석기



〈그림 4〉 스케쥴링과 바인딩 된 미분방정식 해석기

5개의 레지스터를 필요로 하며 입력된 데이터를 수행하는데는 총 4 사이클(latency)의 시간이 걸린다.

이상의 상위합성 단계별로 테스트기능을 삽입하는 기술이 개발되고 있다. 기능을 기술한 VHDL 코드를 분석하여 테스트조정도를 높이는 기술, 상위 구조합성 후의 루프의 개수를 최소화 하기 위하여 스케쥴링 및 바인딩하는 방법의 개발, BIST 및 BILBO를 위한 레지스터 공유 방법 등이 개발되고 있다.

II. 상위 기능정보에 대한 테스트기능의 추가

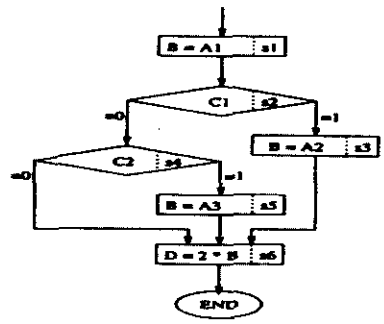
[4]에서는 〈그림 5〉와 같이 Symbol table과 Statement List로 기능을 기술한다. 이 Statement List의 Statement가 vertex가 되는 Control Flow Graph(CFG)를 〈그림 6〉과 같이 만들어 테스트 가능도 분석을 위해 이용한다. 검사하기 어려운 부분을 찾아내어서 테스트점 삽입에 알맞은 장소를 파악하기 위해서는 통로, 변수 var를 정당화할 수 있는 통로의 집합인 JPath(var), 통로를 정당화하기 위해 사용되어진 변수의 집합인 JContPath(p_i, var), Complete Controllability(CC)를 사용한다. 모든 값을 가질 수 있는 변수를 CC 형태라고 한다. NonComplete Controllability(NCC) 변수는 CC가 아닌 변수를 말하고, 이러한 NCC 변수가 있는 곳에 테스트점을 삽입한다.

입력단에서 출력단에 이르는 모든 통로를 열거하고 변수를 추출하며 모든 변수(레지스터)의 크기와 정당화(Justification) 가능도를 통로에 기준하여 측정하고 CC 형태와 NCC 형태로 구별한다. 여기서 NCC 형태의 변수를 모두 제거하는 데 필요한 최소한의 테스트점을 선정하게 된다. 이러한 테스트점은 Test Statement로 삽입되어서 테스트 모드에서 NCC 변수를 CC 변수로 바꾸어 테스트 가능도를 높여주게 된다.

〈그림 7〉은 최대공약수(GCD)를 구하는 회로의

```
.Symbol Table
        A1[0:7], INPUT;
        A2[0:7], INPUT;
        A3[0:7], INPUT;
        B[0:7], INT;
        C1[0:0], INPUT;
        C1[0:0], INPUT;
        D[0:7], OUTPUT;

Statement List
        B = A1; ..... s1
        IF C1  s2
            THEN B = A2; ..... s3
            ELSE IF C2 ..... s4
                THEN B = A3; ..... s5
        D = 2*B; ..... s6
END
```

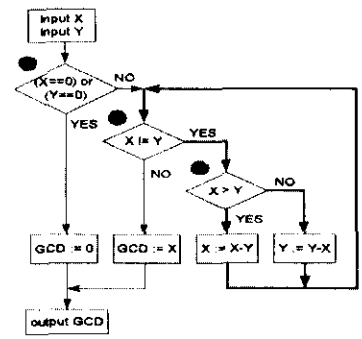
〈그림 5〉 회로의 기능적 표현



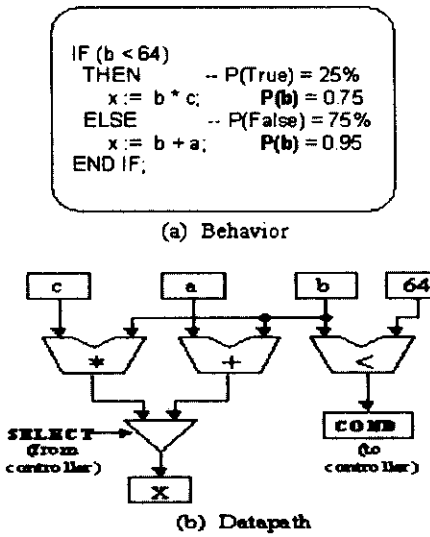
〈그림 6〉 CFG

기능수준 VHDL 코드와 Control Data Flow Graph(CDFG)를 보여주고 있다. CDFG에서 A 및 C는 if-then-else를 B는 while 루프의 조건문을 나타낸다. 상위수준의 기능정보에서 제어 흐름을 증진시켜서 테스트가능도를 높이는 연구가 최근에 활발히 진행되고 있다[4, 5, 6]. While 루프만을 고려하는 [5]에서는 입력 포트까지의 상태문장 개수를 비교하여 최대개수를 필요로 하는 B의 while 루프에 테스트점을 삽입하였다. If-then-else를 고

```
process
  begin
      X := PortX;
      Y := PortY;
      if ( X == 0 ) or ( Y == 0 ) then
         GCD := 0;
      else
         while ( X != Y ) loop
            if ( X > Y ) then
               X := X - Y;
            else
               Y := Y - X;
            end if;
         end loop;
         GCD := X;
      end if;
      portGCD <= GCD;
end process;
```



〈그림 7〉 최대공약수 회로의 상위수준 기술 코드 및 CDFG



〈그림 8〉 기능도 및 데이터 통로도

려하는 [6]에서는 A 혹은 C의 제어문에 XOR 형태 테스트점을 추가하였다.

변수의 정당화 가능도 대신에 시뮬레이션에 의하여 제어가능도를 산출하여 AND, OR 및 XOR 등의 테스트점 가운데 최적의 테스트점을 선별 삽입하도록 한다.

조건제어문 내에 위치한 변수들에 대하여 다음과 같이 조정가능도를 정의한다. 설명을 돕기 위하여 〈그림 7〉의 소스코드에서 내부 if-then-else문을 〈그림 8〉과 같이 변경하였다. [6]에서는 조건제어문 내에 관심 있는 변수 b에 대한 조정가능도를 다음과 같이 산출하였다.

- 방법 1 : 변수 b는 8비트 레지스터로 구현될 것이며 then 이하의 true 문에서는 b<64 이므로 8비트 가운데 상위 2비트는 항상 '00'이 되어서 완전제어가 되지 않아 b의 조정가능도는 개략적으로 6/8=75(%)가 된다.

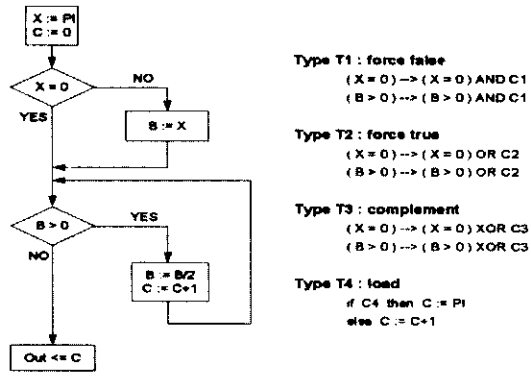
상기 방법은 제어가능도(controllability)를 '0'과 '1'로 세분하지 않았으며 단지 true와 false문 내에서 변수에 대한 균등한 제어도를 유지하기 위하여 XOR 테스트점만을 사용하였다는 단점이 있다. 이러한 단점을 다음과 같은 방법으로 보완될 수 있다.

- 방법 2 : 〈표 2〉에서 보여주는 바와 같이 b 레지스터의 각 비트에 대하여 0-조정도와 1-조정도를 산출해낸다. b 레지스터의 0-조정도는 5이며 1-조정도는 3이 되므로 〈그림 9〉에서의 OR형태와 같은 테스트점을 삽입하여 1-조정도를 증진시킨다.

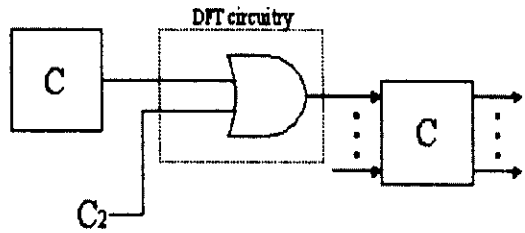
〈표 2〉 레지스터의 각 비트에 대한 0 및 1 테스트 가능도

	b7	b6	b5	b4	b3	b3	b1	b0
0-조정도	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.5
1-조정도	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5

〈그림 9〉는 삽입되는 4가지 다른 종류의 테스트점을 보여주고 있다. T1은 0-조정도를 증진시킬 수 있는 AND 형태의 테스트점을, T2는 1-조정도를 증진시킬 수 있는 OR 형태의 테스트점을, T3은 true와 false문 내에서 변수의 조정도를 반전시킬 수 있는 XOR 형태의 테스트점을 그리고 T4는 변수에 필요한 값을 직접 입력받을 수 있는 테스트점을 나타낸다. 〈그림 10〉은 T2 테스트점의 게이트수준 회로도를 보여주고 있다.



〈그림 9〉 테스트점 삽입



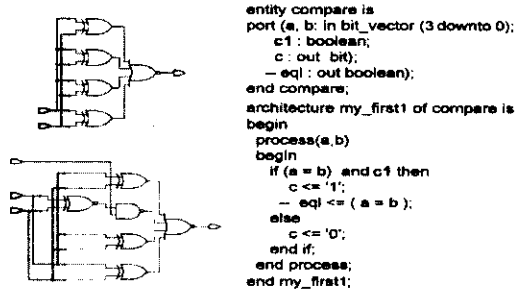
〈그림 10〉 게이트 수준의 테스트점 삽입

〈표 3〉 테스트점을 삽입한 회로의 고장점검도 비교

GCD			
While(x>y)		While(y>=0)	
Test pins	Fault Coverage	Test pins	Fault Coverage
c1	73.31	c1	77.82
c2	69.31	c2	73.01
c,1 c2	68.35	c1, c2	78.30
c3	73.36	c3	74.83
IF (x=0) AND (y=0) THEN에 c3 사용		71.77	
테스트점 미사용		69.40	

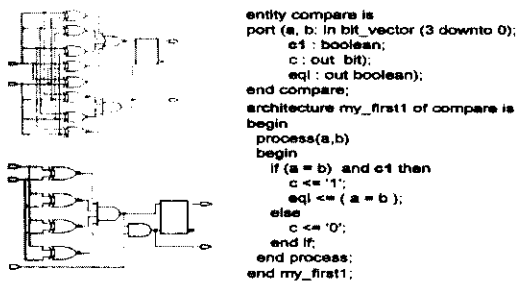
GCD 회로에 테스트점을 삽입한 후 합성한 회로의 고장점검도는 〈표 3〉에 나타나 있다. 결과를 보면 일반적으로 테스트점을 사용하지 않았을 때보다 고장점검도가 증가된다. 그러나 테스트점의 종류에 따라서 고장점검도의 증가에 차이가 있고, 어떤 경우는 거의 변화가 없는 경우도 볼 수 있다.

테스트점을 삽입한 후에 합성된 회로의 결과를 살펴보기로 한다. 다음은 간단한 코드에 대해 테스트점 삽입 전후의 합성된 모습에 대한 예를 보여준다. 〈그림 11(a)〉는 원래의 소스 코드에 대해 우측과 같이 테스트점 C1을 삽입한 후에 합성한 결과를 보여주며 게이트가 증가함을 알 수 있다. 〈그림 11(b)〉는 〈그림 11(a)〉와는 조금 다른 코드에 테스트점을 넣은 경우를 보여주며 합성 후에 오히려 게이트 개수가 감소함을 알 수 있다.



```
entity compare is
port (a, b: in bit_vector (3 downto 0);
      c1 : boolean;
      c : out  bit);
    – eql : out boolean);
end compare;
architecture my_first1 of compare is
begin
 process(a,b)
 begin
   if (a = b)  and c1 then
      c <= '1';
    – eql <= ( a = b );
   else
      c <= '0';
   end if;
 end process;
end my_first1;
```

(a)



```
entity compare is
port (a, b: in bit_vector (3 downto 0);
      c1 : boolean;
      c : out  bit);
      eql : out boolean);
end compare;
architecture my_first1 of compare is
begin
 process(a,b)
 begin
   if (a = b)  and c1 then
      c <= '1';
      eql <= ( a = b );
   else
      c <= '0';
   end if;
 end process;
end my_first1;
```

(b)

〈그림 11〉 테스트점 합성후 게이트 수준

III. 테스트를 고려한 상위 구조합성 기술

상위 구조합성 과정에서 테스트기능을 추가하는 기술은 합성 후 회로의 루프를 최소화하여 부분스캔이 용이하도록 하는 기술과 BIST 등에 필요한 레지스터의 개수를 최소화하는 기술로 구분될 수 있다. 테스트에 필요한 자원을 최소화하기 위해 다양한 스케쥴링 및 바인딩 기술이 개발되고 있다.

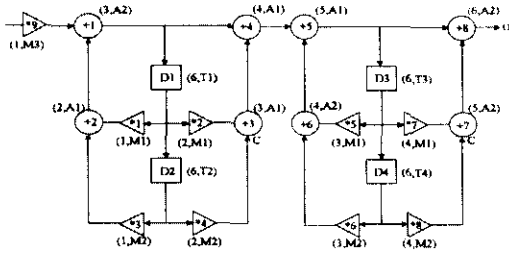
1. 부분스캔을 위한 상위합성 기술

완전스캔으로 인한 추가영역 및 테스트패턴의 주입시간을 최소화하기 위한 부분스캔 기술이 개발되고 있다. 순차회로에서 테스트패턴 생성이 어려운 큰 이유중의 하나는 루프 때문이며 게이트회로에서 루프를 최소화하기 위한 부분스캔 기술이 개발되었다. 합성 후 게이트회로에서 루프를 제거하는 데 최소한의 플립프롭만 스캔할 수 있도록 상위 합성의 바인딩과 스케쥴링을 하는 기술을 소개해 본다.

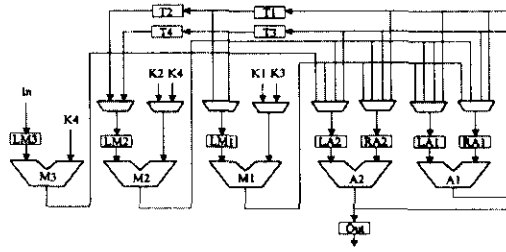
스케쥴링과 바인딩하는 방법에 따라서 부분 스

캔을 할 경우 선택되어야할 스캔 플립프롭의 개수가 달라지게 된다[7].

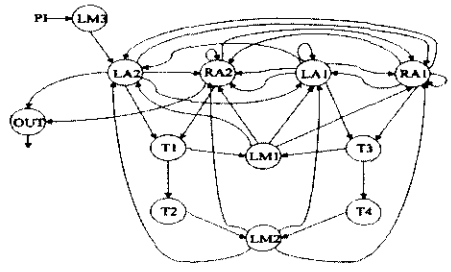
〈그림 12(a)〉는 스케쥴과 바인딩이 종료된 CDFG를 나타낸다. +1에서 +8는 덧셈을, *1에서 *9는 곱셈을, D1에서 D4는 지연시간를 나타낸다. 여기서 각 연산이 한 사이클의 동작시간을 필요로 한다면 최장 연산은 6 사이클이 필요하다. 〈그림 12(a)〉에서 +2는 A1 덧셈기에 할당되고 두번째 사이클에 스케쥴된다는 것을 (2, A1)로 나타낸다. 〈그림 12(b)〉는 〈그림 12(a)〉에 대해 합성된 데이터 통로를 나타낸다. 〈그림 12(b)〉에서 LA2와 RA2는 각각 A2 덧셈기의 왼쪽 레지스터와 오른쪽 레지스터를 나타낸다. 〈그림 12(c)〉는 〈그림 12(b)〉에 대한 S-그래프를 나타내고 이것은 데이터의 의존도를 나타내 준다. 이 S-그래프에서 모든 루프를 제거하는데는 적어도 3개(LA1, LA2, LM1)의 레지스터를 스캔해야 한다.



(a) 스케쥴과 할당이 끝난 CDFG



(b) 데이터 통로

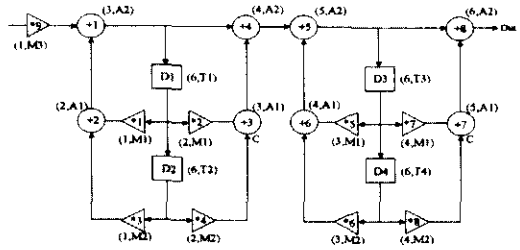


(c) S-그래프

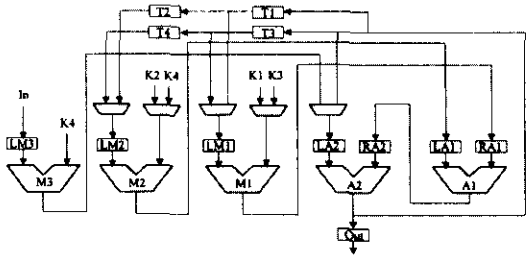
〈그림 12〉 4차 IIR 필터

CDFG에 대해 합성된 데이터 통로에는 여러 가지 형태의 루프가 만들어 질 수 있는데, 여기서는 데이터 의존도 루프, 할당 루프, 순차 거짓 루프, 레지스터 파일 클릭를 검사한다. 이를 위해 CDFG의 각 연산에 대한 mobility와 overlap 값을 구하고 이것으로 compatibility를 계산하여 Data-Dependency and Compatibility Graph(DDCG)를 만든다.

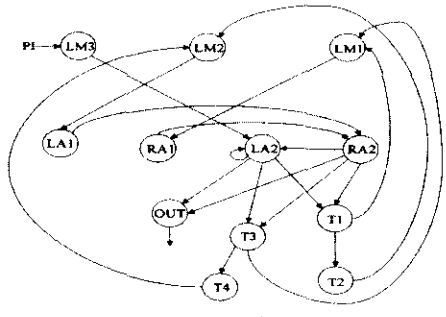
CDFG 루프를 제거하기 위해 필요한 스캔 플립



(a) 스케쥴과 할당이 끝난 CDFG



(b) 데이터 통로



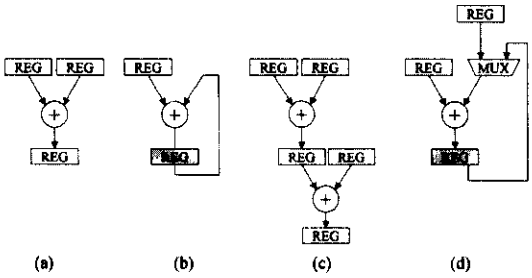
(c) S-그래프

〈그림 13〉 테스트가능도와 자원 활용을 고려하여 합성된 4차 IIR 필터

프롭의 개수를 최소화하기 위해 적당한 스케쥴링과 바인딩을 하게 된다. 이러한 방법을 통하여 구조 합성한 CDFG는 〈그림 13(a)〉같이 나타나고 〈그림 13(b)〉과 같은 데이터 통로를 가지며 〈그림 13(c)〉같은 S-그래프를 형성하게 되는데, 이 S-그래프의 모든 루프를 제거하는데는 1개(RA2)의 레지스터만 있으면 된다. 즉, 3개 대신 1개의 레지스터만 스캔하면 모든 루프를 제거할 수 있는 것이다.

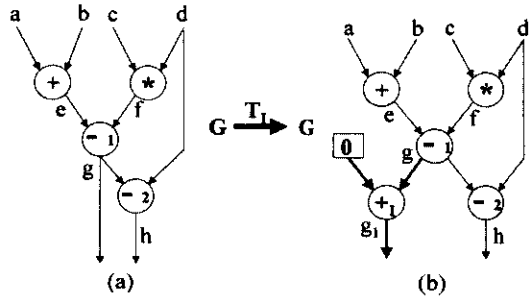
2. BIST를 고려한 상위합성 기술

BIST는 테스트패턴 자동생성기(TPG)와 테스트출력 압축기(SA) 등으로 구성되어 있으며 내장된 메모리 및 블럭 등을 자동으로 테스트하는데 널리 사용된다. 합성된 회로에 BIST 기능을 추가하기 위해서는 레지스터 일부가 테스트기능을 갖도록 변경되어야 한다. BIST 동작을 하기 위해서는 데이터 통로가 테스트모드 시에 TPG 및 SA로 사용될 수 있도록 재구성되어야 한다. 각 조합회로 블럭은 한 개 이상의 레지스터에서 생성된 테스트패턴을 입력받고 연산이 끝난 출력은 다른 레지스터로 주입하여 압축한다. 대부분의 상용 BIST 구조는 레지스터의 초기화 및 테스트패턴 주입을 위하여 부분스캔 및 완전스캔 플립프롭을 이용한다. 이러한 BIST 동작을 위하여 별도의 입력과 출력 전용 레지스터가 있으면 문제는 없지만 한 개의 레지스터가 입력과 출력 두 가지 용도로 사용되어진다면 BIST 동작이 불가능해진다. 예를 들어 〈그림 14〉의 (a)는 각 레지스터가 덧셈연산에 대하여 입력 혹은 출력으로만 사용되므로 자체 테스트가 가능한 회로이다. 반면에 (b)에서 빗금친 하단부의 레지스터는 출력 및 입력 두 가지 용도로 사용되므로 부분적으로만 자체테스트가 가능하다. 이렇게 두 가지 용도로 사용되는(self-adjacent) 레지스터가 생성되지 않도록 바인딩하는 기술[8,9]이 개발되었다.



〈그림 14〉 BIST용 독립적인 레지스터와 self-adjacent 레지스터

BIST 영역은 데이터 통로 모듈을 테스트하는데 필요한 TPG와 SA의 개수를 줄임으로서 감소될 수 있다[10]. 레지스터의 값이 특정 데이터 통로를 지난 후에도 값이 변하지 않는 통로를 I-path라고 정의하였으며, I-path를 활용하면 모든 레지스터를 테스트 레지스터로 사용하지 않아도 됨을 보였다. 테스트 레지스터를 널리 공유함으로써 BIST로 인한 추가영역을 최소화 할 수 있었다. 최근에는 원래의 기능에 중복 연산(redundancy operation)을 추가하여 BIST 테스트 레지스터를 줄이는 연구가 발표되었다. 각 모듈이 동작하지 않고 있는 시간대에 〈그림 15〉와 같이 중복 연산을 삽입하여 테스트 통로를 여러 모듈이 공동으로 사용할 수 있게 하였다[11].



〈그림 15〉 I-변형

IV. 결 론

본 논문에서는 테스트비용을 줄이기 위해서 최근에 사용되고 있는 다양한 상위 테스트합성 기술을 소개하였다. VHDL로 기술된 기능정보에 대하여 테스트점 삽입하는 기술과 부분스캔 및 BIST에 필요한 테스트 회로를 최소화하기 위한 상위 테스트합성 기술을 살펴보았다.

상위합성뿐만이 아니라 논리합성과정 중에서도 테스트 기능을 추가하고, 합성 후 게이트 수준의 회로에서 최종적으로 테스트 설계가 이루어졌을 때 테스트회로 영역 및 고장점검도가 최적화되리라고 사료된다.

참 고 문 헌

[1] V. Chickermane, J. Lee and J. Patel, "A Comparative Study of DFT Methods Using High-Level and Gate-Level Descriptions," Proceedings, ICCAD, pp. 620-624, 1992.

[2] K. D. Wagner and S. Dey, "High-Level Synthesis for Testability: A Survey and Perspective," Proceedings of the Design Automation Conference, pp. 131-136, 1996.

[3] G. De Micheli, "Synthesis and Optimization of Digital Circuits", McGRAW-HILL International Editions, 1994.

[4] C. Chen, T. Karnik, and D. G. Saab, "Structural and Behavioral Synthesis for Testability Techniques," IEEE Trans. on Computer-Aided Design, Vol. 13, No. 6, pp. 777-785, Jun. 1994..

[5] F. F. Hsu, E. M. Rudnick, and J. H. Patel, "Enhancing High-Level Control-Flow for Improved Testability," Proc. Int'l Conf. on Computer-Aided Design, pp. 322-328, Nov. 1996.

[6] K. A. Ockunzzi and C. A. Papachristou., "Testability Enhancement for Behavioral Descriptions Containing Conditional Statements", Proceedings, Int'l Test Conf., pp. 236-245, Nov. 1997.

[7] M. Potkonjak, S. Dey, R. K. Roy, "Behavioral Synthesis of Area-Efficient Testable Designs Using Interaction Between Hardware Sharing and Partial Scan," IEEE Trans. on Computer-Aided Design, Vol. 14, No. 9, pp. 1141-1154, Sep. 1995.

[8] C. Papachristou, S. Chiu and H. Harmanani, "A Data-Path Synthesis Method for Self-Testable Designs," Proceedings of the Design Automation Conference, pp. 378-384, 1991.

[9] T. Lee, W. Wolf and N. Jha, "Behavioral Synthesis for Easy Testability in Data-Path Scheduling," Proceedings of the International Conference on Computer Aided Design, pp. 52-55, 1988.

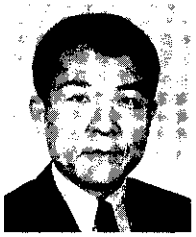
[10] I. Parulkar, S. Gupta, and M. A. Breuer, "Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead," Proceedings of the Design Automation Conference, pp. 395-401, 1995.

[11] I. Parulkar, S. Gupta, and M. A. Breuer, "Introducing Redundant Computations in a Behavior for Reducing BIST Resources," Proceedings of the Design Automation Conference, pp. 395-401, 1998.

저 자 소 개

申 尙 勳

1998년 한양대학교 전자계산학과 학사, 1998년~현재 한양대학교 전자계산학과 석사과정, 〈주관심 분야: 테스트 합성, Scan Design, ASIC 설계, CAD, 그래프이론 등〉



朴 成 柱

1983년 한양대학교 전자공학과 학사, 1983년~1986년 금성사 소프트웨어개발, 1992년 Univ. of Massachusetts 전기 및 컴퓨터공학과 박사, 1992년~1994년 IBM Microelectronics 연구스텝, 1995년~현재 한양대학교 전자계산학과 조교수, 〈주관심 분야: 테스트 합성, Built-In Self Test, Scan Design, ATPG, ASIC 설계, 고속 신호처리 시스템 설계, 그래프이론 등〉