150
논문 99-01-19

ASIC 설계를 위한 새로운 레지스터 전송 단계 합성 방법

A New Register Transfer Level Synthesis Method for ASIC Design

印 致 虎*
(Chi-Ho Lin*)

요 약

본 논문에서는 기존의 레지스터 전송 단계 합성기들이 가지고 있는 단점을 개선하는 새로운 레지스터 전송 단계 합성 방법을 제안한다.

기존의 레지스터 전송 단계 합성기들은, 하드웨어 기술 언어로 기술된 설계 사양을 순서회로로 변환하는 과정에서 불합리한 변환을 수행하고 순서 회로를 최적화 하는 과정에서 순서회로를 구성하는 레지스터와 조합회로를 분리하여 조합회로부만을 최적화 한다. 본 논문에서는 이러한 방식의 레지스터 전송 단계 합성기들이 가지는 단점을 지적하고, 이런 단점을 극복하기 위한 새로운 레지스터 전송 단계 합성 방법을 제안한다. 또한, 제안된 방법을 감시용 시스템의 콘트롤러 설계 및 8 비트 부호화 곱셈기에 적용한 결과를 제시함으로써 본 논문에서 제안하는 방법의 유용성을 입증한다.

Abstract

This paper presents a new register transfer level synthesis method to overcome the disadvantages of the previous register transfer level synthesis systems.

The previous register transfer level synthesis systems first translate from a hardware description language to sequential circuits inadequately. Secondly, the systems separate registers and combinational circuits and then optimize only combinational circuits. This paper describes their disadvantages and then proposes a new method to overcome their shortcomings. This paper also shows the effectiveness of the proposed method by using the proposed method at designing the controller of a surveillance system and the 8-bit signed multiplier.

I. 서 론

VLSI를 설계하는 방법에는 스키매틱 에디터(schematic editor)를 이용하여 회로도를 직접 입력하는 방법과 설계 사양을 하드웨어 기술언어로 기술한 후 이것으로부터 논리 회로를 자동 합성하는 방법이 있다. 최근에는 급변하는 시장 상황에 효율적으로 대처하기 위하여, VLSI를 설계하는데 하드웨어 기술언어

* 世明大學校 컴퓨터科學科
(Dept. of Computer Science Semyung Univ.)
接受日: 1999年3月29日, 修正完了日: 1999年7月16日

와 논리합성 시스템을 이용하는 사례가 늘어나고 있으며, 칩의 제작에는 FPGA (Field Programmable Gate Array) [10], [12-14], [16], ASIC (Application Specific IC) 등을 이용하는 사례가 늘어나고 있다.

하드웨어 기술언어를 사용하여 VLSI를 설계할 때에는 주로 VHDL [15]을 사용하여 회로의 동작을 기술한 후, 논리 합성 시스템을 이용하여 자동으로 논리 회로를 얻거나 최적화 한다. 그런데, VHDL로 VLSI의 동작을 기술하고 자동 논리 합성함에 있어서 회로의 면적이나 속도는 자동 논리 합성 시스템의 영향도 많이 받지만 VHDL로 회로의 동작을 기술하는 방법에도 많은 영향을 받는다 [2], [6], [8]. 또한, VHDL로 회로의 동작을 기술함에 있어서는 게이트 레벨에서 회로를 기술할 수도 있지만 많은 경우에 레지스터 전송 단계에서 회로의 동작을 기술한다.

레지스터 전송 단계의 VHDL 기술에서는 신호선 이나 카운터 (counter), 비교기 (comparator), 덧셈기 (adder), 뺄셈기 (subtracter), 곱셈기 (multiplier)등의 비트 수는 VHDL로 회로의 동작을 기술하는 VLSI 설계자가 결정한다. 레지스터 전송 단계 합성 시스템은 VHDL 기술을 레지스터와 조합회로로 구성된 순서회로로 변환하고, 제약조건 - 면적, 속도, 테스트 용이성 등 - 을 만족하는 순서회로를 얻기 위하여 레지스터와 조합회로부를 분리한 후 조합회로부에 대하여 논리 합성 툴 [3], [4]를 이용하여 논리 최적화를 수행한다.

그런데, 이와 같은 방법을 사용하는 기존의 레지스터 전송 단계 합성 시스템들은, 하드웨어 기술 언어로 기술된 설계 사양을 순서회로로 변환하는 과정에서 불합리한 변환을 수행하고 순서 회로를 최적화 하는 과정을 두어 순서회로를 구성하는 레지스터와 조합회로를 동시에 고려하면 순서회로를 보다 효과적으로 최적화 할 수 있음에도 불구하고 레지스터와 조합회로를 분리하여 조합회로부 만을 최적화 하는 단점을 가지고 있다. 본 논문에서는 기존의 레지스터 전송 단계 합성 시스템들이 가지고 있는 문제점들을 기술한 후, 기존 시스템이 사용하는 방법의 문제점들을 극복할 새로운 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 기존의 레지스터 전송 단계 합성 방법이 가지는 문제점에 대하여 검토한다. 3절에서는 이를 해결하기 위한 방법을 제안한다. 4절에서는 감시 시스템의 콘트롤러 설계에 본 논문의 방법을 적용했을 때 얻은 실험결과를 제시한다. 5절에서는 결론과 함께 향후 연구 방향에 대하여 기술한다.

II. 일반적인 레지스터 전송 단계 합성 방법

본 장에서는 기존의 레지스터 전송 단계 합성 시스템의 단점을 고찰한다.

2.1 기존의 레지스터 전송단계 합성

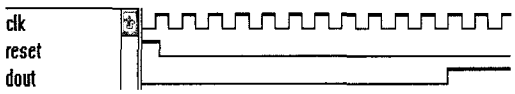
이를 위하여 우선 그림 1에 주어진 VHDL 기술을 예제로 설명한다. 그림 1(a)의 VHDL 기술은 8비트 카운터, 비교기, 플립플롭으로 구성된 간단한 회로의 동작을 표현하고 있지만, 이와 같은 형태의 회로는 비디오 신호 처리시 화상 데이타가 들어있는 주사선 (scan line)을 골라내는데 자주 사용된다 [17]. 회로의 기능은 간단하여 리셋 (reset) 신호가 HIGH로 되면 clk 신호에 관계없이 카운터의 값은 0, dout의 값은 LOW로 되고, 리셋 신호가 LOW이고 clk 신호의 상승 에지에서 카운터 값이 증가되다가 카운터의 값이 십진수로 11이 되면 dout이 HIGH로 되는 회로이다. 그림 1(b)는 그림 1(a)의 VHDL 기술을 시뮬레이션 한 결과이다.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
    port (clk, reset: in std_logic;
          dout: out std_logic);
end counter;
architecture rtl of counter is
 signal count: std_logic_vector (7 downto 0);
begin
    p1: process (reset, clk)
    begin
        if( reset = '1' ) then
```

```
            count <= "00000000";
        elsif rising_edge(clk) then
            count <= count + 1;
        end if;
    end process p1;
    p2: process (reset, count)
    begin
        if (reset = '1') then
            dout <= '0';
        elsif (count = "00001011" ) then
            dout <= '1';
        end if;
    end process p2;
end rtl;
```

(a) VHDL 기술 1

(a) A VHDL description 1.



(b) 시뮬레이션 결과 1

(b) A simulation result 1.

그림 1 VHDL 기술1과 시뮬레이션 결과 1

Fig. 1 A VHDL description1 and a simulation result 1.

그림 1(a)와 같이 VHDL로 기술한 후, 기존의 레지스터 전송단계 합성 시스템을 이용하여 순서회로로 번역하면 그림 2와 같은 회로를 얻는다.

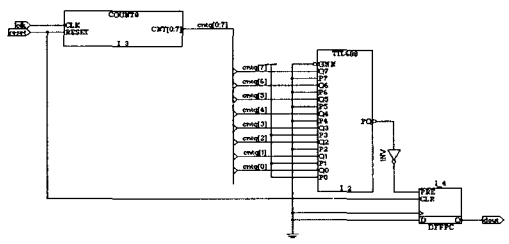


그림 2 순서회로 1

Fig. 2 A sequential circuit 1.

또, 그림 2의 회로를 레지스터와 조합회로로 분리하여 조합회로를 최적화하면 그림 3의 회로를 얻는다. 그림 3의 회로를 96개의 로직셀을 가진 QuickLogic사의 FPGA QL8*12b [14]에 구현하면 11.5%의 로직셀을 이용하고 최대 지연시간이 13.5ns인 회로가 된다.

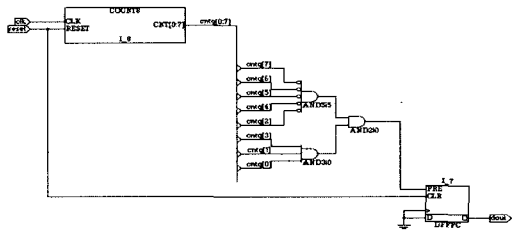


그림 3 기존 방법에 의해 합성된 순서회로 1

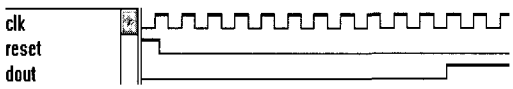
Fig. 3 A synthesized sequential circuit 1 by the previous method.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
    port (clk, reset: in std_logic;
          dout: out std_logic);
end counter;
architecture rtl of counter is
 signal count: std_logic_vector (3 downto 0);
begin
    p1: process (reset, clk)
    begin
        if( reset = '1' ) then
            count <= "0000";
        elsif rising_edge(clk) then
            count <= count + 1;
        end if;
    end process p1;
    p2: process (reset, count)
    begin
```

```
        if (reset = '1') then
            dout <= '0';
        elsif (count = "1011" ) then
            dout <= '1';
        end if;
    end process p2;
end rtl;
```

(a) VHDL 기술 2

(a) A VHDL description 2.



(b) 시뮬레이션 결과 2

(b) A simulation result 2.

그림 4 VHDL 기술 2와 시뮬레이션 결과 2

Fig. 4 A VHDL description 2 and a simulation result 2.

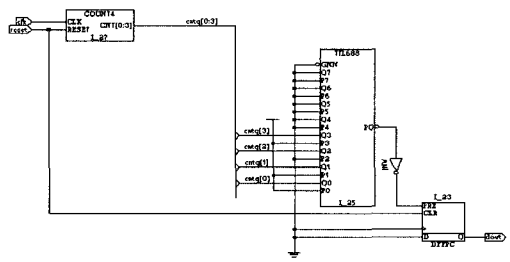


그림 5 순서회로 2

Fig. 5 A sequential circuit 2.

그림 4(a)의 VHDL 기술은 그림 1(a)의 VHDL 기술과 동일한 동작을 기술하고 있으며 그림 4(b)는 시뮬레이션 결과이다. 그림 4(a)와 같이 VHDL로 기술한 후 기존의 레지스터 전송단계 합성시스템을 이용하여 순서회로로 번역하면 그림 5의 회로를 얻는다.

또, 그림 5의 회로를 플립플롭과 조합회로로 분리하여 조합회로를 최적화 하면 그림 6의 회로를 얻는다.

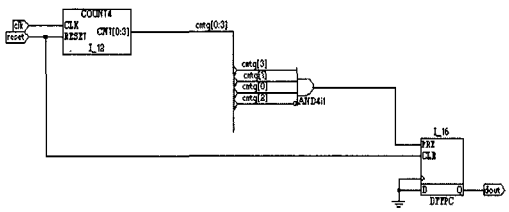


그림 6 기존 방법에 의해 합성된 순서회로 2

Fig. 6 A synthesized sequential circuit 2 by the previous method.

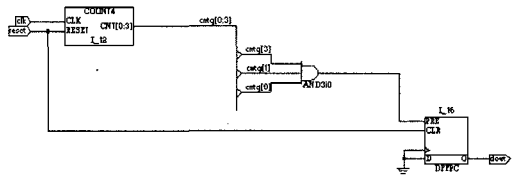


그림 7 무관조건을 고려한 순서회로

Fig. 7 A sequential circuit minimized using don't-care conditions.

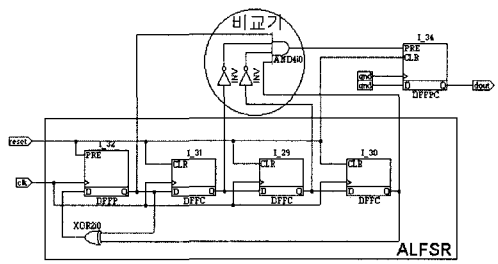


그림 8 ALFSR을 이용한 순서회로

Fig. 8 A sequential circuit3 using an ALFSR

그림 6의 회로를 96개의 로직셀을 가진 QuickLogic사의 FPGA QL8*12b [14]에 구현하면 6.3%의 로직셀을 이용하고 최대 지연시간이 11.0ns인 회로가 된다.

2.2 기존의 VHDL 번역 방법의 문제점

그림 1(a)의 VHDL 기술을 관찰하면 프로세스 p1은 비

동기 리셋 (asynchronous reset)이 있는 8 비트 카운터이고, 프로세스 p2는 RS 플립플롭과 비교기로 구성되는 회로이다. 8비트 카운터의 값이 "00001011"이면 dout이 셋 (set) 된다.

그리고, 8비트 카운터의 값은 오직 프로세스 p2에서 비교기의 입력으로만 사용된다. 따라서, 카운터는 "1011"까지만 셀 수 있으면 되므로 4비트 이상의 임의 비트의 카운터로 바꾸어도 그림 1(a)의 동작은 달라지지 않는다. 회로의 면적 최소화를 위해서는 카운터의 크기를 최소로 하는 바람직하기 때문에 4비트 카운트로 바꾸는 것이 바람직하다. 그러므로 그림 1(a)의 VHDL 기술이 주어졌을 때 그림 1(a)의 VHDL 기술을 그림 2의 순서회로로 직접 번역하는 대신에 그림 4(a)의 VHDL 기술로 변환한 후 그림 4(a)의 VHDL 기술을 그림 5의 순서회로로 번역하고 순서회로를 플립플롭과 조합회로로 분리하여 조합회로부를 최적화 한 후 그림 6의 회로를 얻는 것이 이상적이다. 그럼에도 불구하고 현재 사용되고 있는 VHDL 기술을 순서회로로 번역하는 프로그램 [11]은 그림 1(a)의 VHDL 기술은 그림 2의 순서회로로, 그림 4(a)의 VHDL 기술은 그림 5의 순서회로로 번역한 후 플립플롭과 조합회로를 최적화 함으로서 설계자가 VHDL로 어떻게 기술하느냐에 따라 즉, 그림 1(b)의 파형으로 정의되는 동작을 VHDL로 그림 1(a)처럼 기술하느냐, 그림 4(a)처럼 기술하느냐에 따라 회로의 크기가 확연히 다른 회로를 생성하는 문제점을 가지고 있다.

2.3 기존의 순서회로 최적화 방법의 문제점

그림 1(a)의 VHDL 기술로부터 번역된 그림 2의 순서회로를 최적화하든, 스키매틱 에디터로 입력된 그림 2의 순서회로를 최적화하든 기존의 최적화 방법에서는 플립플롭과 조합회로부를 분리한 후 조합회로부만을 최적화하고 조합회로부를 플립플롭과 재결합한다. 이런 방식의 최적화 방법을 그림 2의 순서회로에 적용한다면 그림 3의 순서회로를 얻을 수밖에 없다. 그러나, 그림 2의 순서회로를 잘 관찰하면 첫째, 카운터와 비교기는 8비트용이지만 비교기가 비교하는 값이 "00001011"이고 둘째, 카운터의 값이 다른 곳에서 사용되지 않고 셋째, 비교기의 출력이 셋 단자에 연결되어 있다는 사실로부터 카운터와 비교기의 크기를 4비트로 만들 수 있다.

그러므로 그림 2의 순서회로를 플립플롭과 조합회로부로 분리하기 전에 그림 2의 순서회로를 그림 5의 순서회로로 변환할 수 있다. 또, 그림 5의 순서회로를 관찰하면 RS 플립플롭이 셋 되는 때의 카운터의 값이 "1011"이므로 카운터의 값이 "1011"때 RS 플립플롭을 셋하고, 카운터의 값이 "1011"보다 큰 동안에는 한번 이상 RS 플립플롭을 셋 하든 하지 않든 회로의 동작은 변하지 않는다. 그러므로 그림 5의 순서회로를 플립플롭과 조합회로로 분리하여 조합회로를 최적화 할 때 무관조건 (don't-care condition) - 여기서는 카운터의 값이 "1011"보다 클 때에는 비교기가 출력이 HIGH가 되든 LOW가 되든 상관없다는 사실 - 을 조합회로 최적화 툴 에 제공하면 그림 7의 순서회로를 얻는다. 그림 5의 순서회로를 최적화 할 때 무관조건을 생성하지 않으면 조합회로부의 최적화 후 그림 6의 순서회로를 얻게 되는데 이 경우에는 그림 7의 순서회로보다는 비교기 부분에서 AND 게이트의 입력수가 1개 많은 순서회로가 되어 불리하다.

또, 그림 5의 순서회로를 그림 7과 같은 순서회로로 최적화 할 수도 있지만 카운터의 값이 비교기의 입력으로만 사용된다는 사실을 이용하여 그림 5의 순서회로를 그림 8의 순서회로로 변환할 수 있다. 그림 5의 순서회로에서 비교기의 출력은 리셋이 LOW로 바뀐 후, 몇 번째 clk에서 HIGH로 되는가만 중요하고 clk가 들어옴에 따라서 카운터의 값이 증가해야 한다거나 감소해야 한다는 조건이 없다. 이런 사실을 이용하면 의사난수 (pseudo-random)를 생성하는 ALFSR (Autonomous Linear Feedback Shift Register) [5], [7]로 카운터를 대체하고 리셋이후 11번째 clk에서 비교기의 출력이 HIGH가 되도록 비교기의 고정된 입력 값을 적당히 조정하면 그림 8의 순서회로를 얻을 수 있다. 그럼에도 불구하고 현재 사용되고 있는 순서회로 최적화 방법은 순서회로를 플립플롭과 조합회로로 분리한 후 조합회로부만을 최적화 함으로서 그림 2의 순서회로를 그림 5의 순서회로로 변환할 수 없으며, 그림 5의 회로를 그림 8의 순서회로로 변환할 수도 없다. 또한, 순서회로의 분리 시 무관조건을 제대로 생

성하지 못함으로서 그림 5의 회로로부터 그림 7의 회로를 얻지 못하는 단점을 가지고 있다.

III. 제안된 레지스터 전송 단계 합성 방법

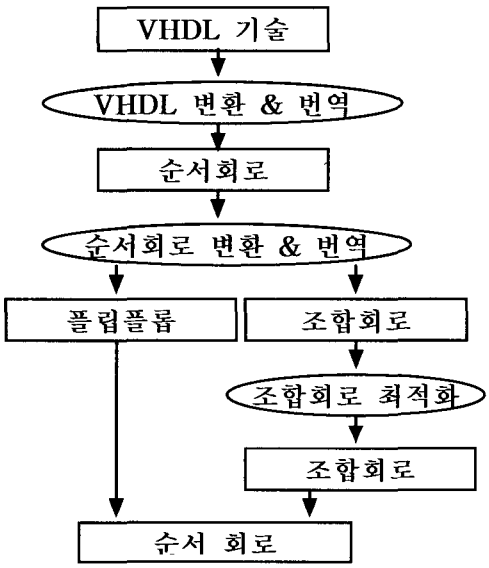


그림 9 제안된 레지스터 전송 단계 합성 방법

Fig. 9 A proposed register transfer level synthesis method

그런데 이런 종류의 문제를 해결함에 있어서는 회로의 전체 특성을 고려해야 하므로 가능하면 하위 레벨보다는 상위 레벨에서 이 문제를 다루는 것이 바람직하다. 그러므로, VHDL 기술을 단순히 순서회로로 번역하고, 번역된 순서회로를 플립플롭과 조합회로로 분리한 후 조합회로부만을 최적화 하는 기존의 레지스터 전송단계 합성방법에서 탈피하여 VHDL 기술을 효과적으로 변환한 후 VHDL 기술을 순서회로로 번역하고, 번역된 순서회로를 면적과 성능이 우수한 순서회로로 변환한 후 순서회로를 플립플롭과 조합회로부로 효과적으로 분리할 수 있는 방법이 요구된다. 본 논문에서는 VHDL 기술로부터 순서회로의 분리에 이르는 과정에 적용 가능한 방법을 기술한다.

먼저 3.1절에서 VHDL 기술을 효과적으로 변환하고 순서회로로 번역하는데 사용하면 유용한 방법을 기술하고, 3.2절에서 순서회로의 변환 및 분리에 적용 가능한 방법을 기술한다. 그림 9에 제안된 방법의 전반적 구성이 있는데 본 논문에서 제안한 방법은 VHDL 변환, VHDL 번역, 순서회로 변환, 순서회로 분리 등에 관련된 것이다.

3.1 VHDL의 변환

레지스터 전송단계에서의 VHDL 기술을 순서회로로 효과적으로 번역하기 위해서는 제어흐름분석 (control flow analysis), 데이터 흐름 분석 (data flow analysis)을 수행할 필요가 있으나, 기존 컴파일러 이론의 코드 최적화 기법 [1]을 그대로 적용하면 계산 시간이 많이 걸리고 하드웨어 기술을 순서회로로 번역하는 것과 소프트웨어를 최적화 하는 것과는 약간의 차이가 있으므로 본 논문은 실제 회로의 설계 시 흔히 사용하는 구문을 대상으로 계산시간에 부담을 느끼지 않는 범위 안에서 회로의 면적을 대폭 줄이는 방법을 기술한다.

3.1.1 곱셈의 변환

곱셈을 수행하는 문장에는 A = B * C와 같이 오퍼랜드 2개 모두 변할 수 있는 경우도 있지만 A = B * 상수와 같이 오퍼랜드 1개가 고정된 경우도 있다. 특히, 오퍼랜드 1개가 상수로 고정된 경우는 2가지로 분류할 수 있다. 상수가 2^n인 경우와 그렇지 않은 경우로 나눌수 있는데 우선 상수가 2^n인 경우부터 기술한다.

A = B * 2^n과 같은 곱셈의 변환에서는 그림 10처럼 B의 비트를 MSB(Most Signicant Bit) 측으로 n비트 올려서 연결하고 LSB (Least Significant Bit)측의 n비트는 GND에 연결한다. 즉, B * 2^n는 B<<n으로 변환한다. 컴파일러 이론에서의 코드 최적화에는 쉬프트(shift)하는 시간이 소요되지만 하드웨어 기술언어의 변환에서는 전혀 하드웨어가 증가하지 않는다.

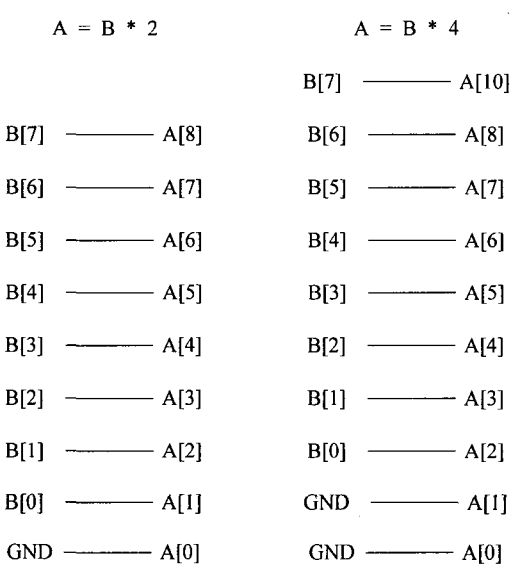


그림 10 A = B * 2^n의 곱셈

Fig. 10 Multiplication A = B * 2^n.

또, A = B * 상수와 같은 곱셈의 변환에서는 그림 11과 같이 변환한다. 예를 들어, A = B * 9와 같은 문장의 변환에서는 A = B + (B<<3)로 변환한다. 즉, A = B * 상수와 같은 곱셈의 변환에서는 곱셈을 B와 B를 쉬프트(shift)한 수를 더하는 연산으로 변환한다. 그런데, A = B * 상수의 변환 시 필요로 하는 덧셈과 쉬프트의 수는 다수일 수 있다. 예를 들어, A = B * 11과 같은 곱셈의 경우는 A = B + (B<<1) + (B<<3)로 변환되어 2개의 덧셈과 2개의 쉬프트를 필요로 한다. 여기서도 역시 쉬프트 연산에 의한 회로의 면적 증가는 없다. 다만, 오퍼랜드 B의 팬 아웃 (fan-out)이 증가한다.

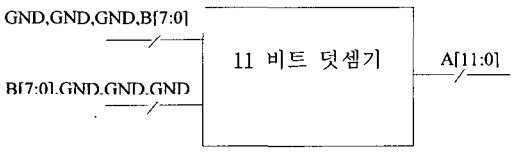


그림 11 A = B * 9의 곱셈

Fig. 11 Multiplication A = B * 9.

3.1.2 나눗셈의 변환

A = B / 2^n과 같은 나눗셈의 번역에서는 그림 12처럼 B의 비트를 LSB 측으로 n비트 올려서 연결하고 MSB 측의 n비트는 GND에 연결하면 된다. 즉, B / 2^n은 B>>n으로 변환한다.

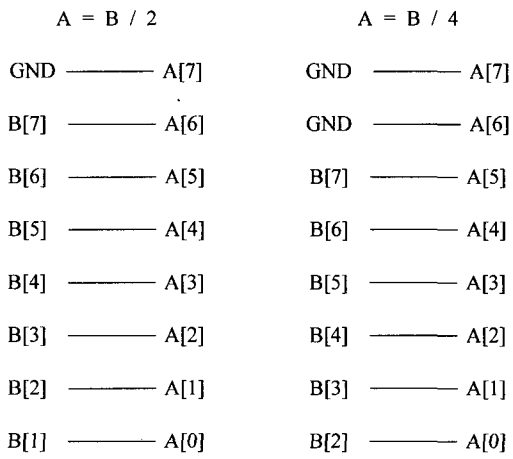


그림 12 A = B / 2^n의 나눗셈

Fig. 12 Division A = B * 2^n.

3.1.3 공통 부분식의 제거

그림 13과 같이 기본 블럭간에 나타나는 공통 부분식이 오직 1번만 계산되도록 하면 회로의 면적을 줄일 수 있다. 이때에도 하위레벨에서 공통 부분식을 찾기보다는 상위레벨에서 공통부분식을 추출하는 것이 바람직하다.

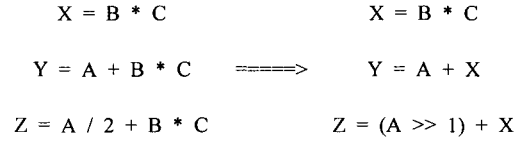


그림 13 공통 부분식의 제거

Fig. 13 Removal of common subexpressions.

3.1.4 상수 폴딩 (folding)

Z = X * Y의 연산이 있을 때, 그림 14(a)처럼 오퍼랜드 X, Y가 초기화되어 있으면 곱셈을 행한 후에 직접 Z = 상수로 대치한다. 또, 그림 14(b)처럼 오퍼랜드 X, Y 중 하나만 초기화되어 있으면 3.1.1절의 곱셈 변환 방법을 함께 사용해 변환한다.

```
X = 1                    X = 1
Y = 2      =====>        Y = 2
Z = X * Y                Z = 2
```

(a) 두 오퍼랜드가 초기화되었을 때의 상수 폴딩

(a) Constant folding with two initialized operands.

```
Y = 2      =====>        Y = 2
Z = X * Y                Z = X<<1
```

(b) 하나의 오퍼랜드가 초기화되었을 때의 상수 폴딩

(b) Constant folding with an initialized operand.

그림 14 상수 폴딩

Fig. 14 Constant folding.

3.1.5 VHDL 기술의 핍홀 최적화

VHDL 기술의 핍홀 (peephole) 최적화는 컴파일러 이론의 핍홀 최적화 기법을 사용하여 행한다. 핍홀 최적화란 원래 코드 생성기에 의해 생성된 목적코드를 부분적인 관점에서 관찰하며 코드를 최적화 하는 방법이다. 여기서 핍홀 이란 목적코드에 만들어지는 작은 움직이는 윈도우 (window)이다. 이러한 컴파일러 이론의 코드 최적화 기법을 VHDL의 최적화에 적용한다.

최적화 과정은 윈도우를 조금씩 움직이면서 개선될 수 있는 VHDL 코드의 패턴을 찾는다. 여기서 탐색하는 VHDL 코드의 패턴은 그림 1(a)의 VHDL 기술에서와 같이 카운터, 비교기, RS 플립플롭으로 구성되는 VHDL 코드에 국한한다. 대상 VHDL 코드의 패턴을 찾으면 카운터, 비교기, RS 플립플롭으로 구성되는 VHDL 코드에서 카운터의 값이 다른곳에서 사용되지 않는 경우 카운터 및 비교기의 크기를 그림 5(a)와 같이 축소 변환한다.

3.2 순서회로 최적화

3.2.1 순서회로의 핍홀 최적화

컴파일러 이론의 핍홀 최적화는 순서회로의 최적화에도 적용할 수 있다. 최적화 과정은 우선, 윈도우를 조금씩 움직이면서 개선될 수 있는 순서회로의 패턴을 찾는다. 대상 회로 패턴을 발견하면 효과적인 회로 패턴으로 대치한다. 여기서 탐색하는 패턴은 다음의 2가지 순서회로 패턴에 국한한다.

패턴 1) 카운터, 비교기, RS 플립플롭으로 구성되는 순서회로

패턴 2) 카운터와 비교기로 구성되는 순서회로

탐색 대상 패턴이 2개 이상인 경우 어떤 패턴을 먼저 변환하는가에 따라서 그 결과가 달라진다. 따라서, 본 논문에서는 패턴 1을 먼저 탐색하고 변환한다. 그림 5의 순서회로에서 패턴 1의 회로를 발견하면 그림 7로 변환한 후와 그림 8로 변환한 후의 회로 크기 및 성능을 비교하여 그림 7의 순서회로로 변환할 것이지 그림 8의 순서회로로 변환할 것인지 결정한다. 패턴 1의 회로에 대한 탐색 및 변환이 끝나면 패턴 2의 회로에 대한 탐색과 변환을 수행한다. 패턴 2의 회로에 대한 변환은 무조건 ALFSR과 비교기로 변환한다.

이와 같은 방법을 사용하면 그림 5의 순서회로를 그림 7이나 그림 8의 순서회로로 변환할 수 있다.

3.2.2 순서회로의 분리

VHDL 기술을 레지스터와 조합회로로 구성된 순서회로로 변환하고, 면적, 속도, 테스트 용이성 등의 제약조건을 만족하는 순서회로를 얻기 위하여 레지스터와 조합회로부를 분리한 후 조합회로부에 대하여 논리 최적화를 수행한다.

순서회로의 조합회로부를 최적화하기 위하여 순서회로를 플립플롭과 조합회로로 분리하는데 이때, 조합회로의 최적화 결과에 막대한 영향을 미치는 무관조건(don't-care condition)을 제대로 생성하지 못하면 조합회로의 최적화를 제대로 수행하기 어렵다. 예를 들어 그림 5의 순서회로를 3.2.1절에서 기술한 핍홀 최적화 방법으로 순서회로를 최적화지 않고 그림 5의 순서회로를 플립플롭과 조합회로부로 무조건 분리하고 조합회로부만을 최적화 하면 그림 6의 순서회로를 얻을 수밖에 없다. 하지만, 그림 5의 순서회로를 플립플롭과 조합회로부를 분리할때 무관조건 - 여기서는 카운터의 값이 "1011"보다 클 때에는 비교기가 출력이 HIGH가 되든 LOW가 되든 상관없다는 조건 - 을 생성하면 그림 5로부터 그림 7의 순서회로를 얻는다. 그러므로, 순서회로의 분리 시 무관조건을 생성할 필요가 있다. 그럼에도 불구하고 회로로부터 무관조건을 생성하기 힘들다는 고정관념으로부터 탈피하지 못함으로써 순서회로의 분리 시 무관조건을 생성하지 않는다.

본 논문의 방법은 순서회로 전반에 적용 가능한 종합적인 무관조건 생성 방법이 아니고, 그림 5와 같이 특정 회로 패턴을 가진 순서회로에 국한하여 적용할 수 있는 방법이다. 이 방법은 비교기의 출력단이 RS 플립플롭에 연결된 회로패턴을 발견하면 그 비교기가 비교하는 값보다 큰 값의 입력신호는 모두 무관조건으로 생성한다.

즉, 본 논문에서 제안한 순서 회로 최적화 방법은 조건 변수의 무관조건을 제대로 고려하지 않았을 때 생기는 문제점을 지적하고, 이 문제의식에 근거하여 3장에서 열거한 합성방법을 도입하였으며, 3장의 합성방법이 새롭다는 것은 아니고 순서회로 최적화에 도입하여 기존의 레지스터 전송 단계 합성 시스템들이 가지고 있는 문제점들을 기술한 후, 기존 시스템이 사용하는 방법의 문제점들을 극복할 새로운 방법을 제안하여 구현하였다.

IV. 실험결과

본 논문에서 제안하는 방법의 유용성을 입증하기 위하여 본 논문의 방법을 QuickLogic사의 합성 툴 SpDE [14]와 비교한다. 또, 회로의 면적이나 지연시간의 직접적인 비교를 위하여 주어진 회로는 QuickLogic사의 FPGA QL8*12b, QL12*16b에 구현한 후 비교한다. FPGA는 speed grade 2인 FPGA를 사용하였으며, VHDL 기술이나 순서회로로부터 최종회로를 얻는데는 버전 5.1의 SpDE을 사용하였다.

최종회로를 얻는 방법은 다음과 같다. 본 논문의 결과는 VHDL 기술이 주어지면 본 논문의 방법으로 최적화 한 후 SpDE로 합성하여 최종회로를 얻었고, SpDE의 결과는 VHDL 기술을 그대로 SpDE로 합성하여 최종회로를 얻었다.

우선 그림 1의 VHDL 기술을 본 논문의 방법과 SpDE로 합성한 결과가 표1에 있다. 표 1의 결과를 얻는데 사용한 FPGA는 로직셀 이 96개 있는 1000 게이트급 QL8*12b 이다. 예상했듯이 본 논문의 방법이 SpDE보다 회로의 면적과 지연시간 측면에서 우수한 결과를 얻는다. 이와 같은 결과는 본 논문의 방법이 3.2절에서 기술한바와 같이 순서회로 최적화 시 플립플롭과 조합회로부의 상관관계를 고려하는데 비해 SpDE는 순서회로 합성 시 플립플롭과 조합회로의 관계를 고려하지 못하고 무조건 플립플롭과 조합회로부를 분리하여 조합회로부만을 최적화하기 때문이다.

표 1 그림 1의 VHDL 기술의 합성결과

Table 1 The synthesis result of the VHDL description in Fig. 1.

방법 / 회로	SpDE	제안된 방법 & SpDE
Logic Cells	11.5%	6.3%
Routing Resources	2.6%	1.1%
Vialink Resources	0.1%	0.1%
최대 지연시간	13.5ns	11.0ns

또, CCTV 시스템의 콘트롤러 [9]에 본 논문의 방법과 SpDE를 적용한 결과가 표 2에 있다. 표 2의 결과를 얻는데 사용한 FPGA는 로직셀이 192개 있는 2000게이트급 QL12*16 이다. 실험에 사용된 CCTV 콘트롤러는 그림 2와 유사한 회로를 많이 포함하고 있는데, 비

디오 신호의 수직 동기 신호 V_SYNC를 리셋 신호로 가지고 있으며, 수평동기 신호 H_SYNC를 clk 신호로 가지고 있다.

표 2 CCTV 시스템 콘트롤러의 합성결과

Table 2 The synthesis result of the controller for a CCTV system.

방법 / 회로	SpDE	제안된 방법 & SpDE
I/O 수	56개 (46.7%)	56개 (46.7%)
Logic Cells	143개 (73.7%)	88개 (45.8%)
Routing Resources	21.4%	13.6%
Vialink Resources	1.1%	0.4%
최대 지연시간	18.6ns	16.3ns

그리고 8 비트 부호화 곱셈기 [12]에 본 논문의 방법과 SpDE를 적용한 결과가 표 3에 있다. 표 3의 결과를 얻는데 사용한 FPGA는 표 2와 같이 로직셀이 192개 있는 2000게이트급 QL12*16 이다. 본 논문의 결과는 VHDL 기술이 주어지면 본 논문의 방법으로 최적화 한 후 SpDE로 합성하여 최종회로를 얻었고, SpDE의 결과는 VHDL 기술을 그대로 SpDE로 합성하여 최종회로를 얻었다. 표 1과 표 2의 결과에서 예상했듯이 본 논문의 방법이 SpDE보다 회로의 면적과 지연시간 측면에서 우수한 결과를 얻는다.

표 3. 8 비트 부호화 곱셈기

Table 3 The synthesis result of the 8-bit signed multiplier.

방법 / 회로	SpDE	제안된 방법 & SpDE
I/O 수	68개 (56.7%)	68개 (56.7%)
Logic Cells	153개 (79.6%)	110개 (57.2%)
Routing Resources	39.6%	27.7%
Vialink Resources	2.8%	1.2%
최대 지연시간	27.8ns	22.9ns

V. 결 론

본 논문에서는 기존의 레지스터 전송 단계 합성 시스템들이 하드웨어 기술언어로 기술한 회로를 순서회로로 번역하는 과정 또는 논리 회로 합성 시스템이 순서회로를 최적화 하는 과정에서 플리플롭들과 조합회로 상호간의 관계를 고려하지 않음으로써 야기되는 문제점들을 지적하고 이를 해결할 수 있는 새로운 방법의 레지스터 전송 단계 합성 방법을 제안하였다. 또한, 실제로 사용되고 있는 감시 시스템의 콘트롤러 설계에 본 논문의 방법을 적용하여 회로의 면적을 약 38.5%정도 줄일 수 있음을 입증하였으며, 그리고 8 비트 부호화 곱셈기에 적용한 결과 약 35.9% 정도의 회로 면적을 줄일 수 있는 효율적인 결과를 얻었다.

향후 연구과제로는 VHDL 기술을 순서회로로 번역하는 과정에서 카운터, 비교기, RS 플립플롭 등의 논리 블록을 효율적으로 찾는 방법에 대한 연구가 필요하다. 또한, 논리 회로 최적화 시스템에서도 순서회로가 주어지면 순서회로를 레지스터들과 조합회로로 분리한 후 최적화를 수행할 것이 아니라 레지스터들과 조합회로의 상호 관계를 검토하는 과정을 기존의 논리 회로 합성과정의 전처리 단계로 삽입하는 연구가 필요하다.

참 고 문 헌

[1] A. V. Aho and J. D. Ullman, *Principles of compiler design*, Addison-Wesley, 1987.

[2] J. R. Amstrong and F. G. Gray, *Structural logic design with VHDL*, Prentice-Hall, 1993.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*, Kluwer Academic Publishers, Boston, 1984.

[4] R. K. Brayton, G. D. Hachtel, and A. Sangiovanni-Vincentelli "Multilevel logic synthesis," *Proceedings of IEEE*, Vol.78, No.2, pp. 264-300,

Feb. 1990.

[5] E. J. McCluskey, *Logic design principles with emphasis on testable semicustom circuits*, Prentice-Hall, 1986.

[6] D. L. Perry, *VHDL*, McGraw-Hill, 1993.

[7] W. W. Peterson and E. J. Weldon, *Error-correcting codes*, Colonial Press, Inc., 1972.

[8] 박현철, *VHDL 회로설계와 응용*, 한성출판사, 1995.

[9] *월간 전자기술*, (주) 첨단, 1995. 9.

[10] *Data Book, Altera*, 1995.

[11] *Design Analyzer Reference Manual*, Synopsys, 1994.

[12] *FPGA Databook and Design Guide*, Actel, 1994.

[13] *Lattice Databook*, Lattice, 1994.

[14] *Quickworks Version 5 User's Guide*, QuickLogic, 1995.

[15] *Standard VHDL language reference manual (IEEE Std1076-1987)*, IEEE, 1988.

[16] *The Programmable Gate Array Design Handbook*, Xininx, 1986.

저 자 소 개

印 致 虎 (正會員)

1985년 한양대학교 전자공학과 학사, 1987년 한양대학교 대학원 공학석사, 1996년 한양대학교 대학원 공학박사, 1992년 ~ 현재 세명대학교 컴퓨터과학과 부교수

관심분야 : VLSI CAD, ASIC 설계, CAD 알고리듬 등.