論文94-31A-6-24

VHDL 표현으로부터의 시간 지연 합성

(Timing Synthesis from VHDL Description)

朴 相 憲*, 崔 起 榮**

(Sanghun Park and Kiyoung Choi)

要 約

일반적으로 하드웨어를 설계할 때 시스템 클락의 주기와 비교하여 매우 긴 시간 지연을 필요로 하는 경우에는 타이머를 사용하게 된다. 이 논문은 이러한 시간 지연 조건을 만족하는 하드웨어를 VHDL 표현으로부터 타이머를 이용하여 합성하는 방법을 제시한다. 일반적으로 타이머는 큰 하드웨어 비용을 필요로 하기 때문에 가능한 한 효율적으로 사용되어야 한다. 이 문제의 해를 구하기 위하여 그래프 모델을 정의하고 이를 이용하여 타이머의 개수를 최소화하는 알고리듬을 제안한다. 또한 예비적인 실험 결과를 통하여 제안된 알고리듬이 최소의 타이머 개수로 필요한 모든 시간 지연을 구현함을 보인다.

Abstract

Timers are commonly used in hardware design for time delays that are to be much longer than the system clock period. In this paper, we present a method by which we can synthesize a hardware containing timers that implement long time delays described in VHDL. Because, in general, timers require high hardware cost, they must be utilized as efficiently as possible. To solve this problem, we define a graph model and propose an algorithm that uses the graph model to minimize number of timers. A preliminary experimental results show that the algorithm implements all required time delays using minimum number of timers.

I. 서론

전자 시스템과 VLSI의 크기와 복잡도가 커지고 개발 요구 기간이 짧아짐에 따라 그 설계에서 상위 단계 합성(high-level synthesis)이나 논리 합성(logic synthesis)의 중요성은 점점 높아지고 있으며 이미 이에 대한 많은 연구와 개발이 이루어져 왔다. 그러나 아직 미흡한 것 중의 하나가 시간 지연을 포함하는 표현으로부터의 합성이다. 특히 제어기(controller)를 설계할 때에는 시간 지연 표현이 중요하므로 이에 대한 합성(시간 지연 합성)은 제어기의 설계를 자동화 하는 데에 필수적인 기능이 된다. 오늘날 대부분의 제어기들은 시간 지연 제어를 위하여 하나 이상의 타이머들을 가지고 있으며, 특히 범용의 제어

* 正會員, 서울大學校 電子工學科
(Dept. of Elec. Eng., Seoul Nat'l Univ.)
** 正會員, 서울大學校 半導體共同硏究所
(Dept. of Elec. Eng., Seoul Nat'l Univ.)
※본 연구는 과학재단 '92 핵심전문 연구지원비에 의하여 이루어짐.
接受日字 : 1993年 11月 4日

기들은 보통 여러 개의 타이머를 포함하고 있다.[1,2] 그러나 특수 목적의 제어기를 설계하고자 할 때에는, 타이머들을 새로 설계하거나 또는 라이브러리로부터 가져 와서 손 작업으로 설계하고자 하는 제어기에 삽입해야 한다. 기존의 합성 시스템들[3,4]은 이러한 과정의 자동화를 고려하고 있지 않은데 여기에 타이머의 자동 설계 기능이 추가되면 제어기의 설계 시간이 많이 단축될 수 있을 것이다.

시간 지연 합성에서 발생하는 문제의 핵심은 필요한 타이머의 개수와 크기의 결정이다. 타이머의 개수는 공유 가능성에 의하여 결정된다. 하나의 타이머로 몇 개의 서로 다른 시간 지연 표현들을 만족시킬 수 있으면 타이머의 총 개수를 줄일 수 있다. 타이머의 크기는 클락 주기에 의하여 결정되며 클락 주기는 필요한 정밀도에 의하여 결정된다. 높은 정밀도가 요구될 수록 클락 주기는 작아야 하며 따라서 타이머는 커지게 된다. 높은 정밀도가 요구되지 않는 경우에는 시스템 클락을 분주하여 주기가 큰 클락을 새로 만들어 이를 여러 개의 타이머가 같이 사용하도록 함으로써 각 타이머의 크기를 줄일 수도 있다.이러한 것을 고려할 때 타이머를 이용한 시간 지연 합성 문제는 다음과 같이 정의될 수 있다.

각 시간 지연에 대하여 요구되는 정밀도가 만족되는 범위에서 타이머가 차지하는 실리콘의 총면적이 가장 작도록 타이머의 개수와 각 타이머의 크기를 구한다.

이 논문은 시간 지연 명세로부터 타이머를 생성하여 원래의 설계에 이를 삽입하는 시간 지연 합성 시스템을 제안한다. 그림 1은 제안된 시스템의 구성을 보여 준다. 이 시스템에서는 두 가지 유형의 입력 명세가 가능하다. 첫째 유형은 기능 명세와 별도의 시간 지연 명세로 구성된다. 여기에서 시간 지연 명세는 신호들 사이의 최소와 최대 지연 시간을 명시하여 지연 시간 오차의 크기를 제한한다. 둘째 유형은 기능 명세와 시간 지연 명세가 섞여 있는 표현으로 이루어지는데 여기에서는 시간 지연 오차가 무시할 정도로 작다는 가정을 하며 시스템 클락을 분주하여 여러 개의 타이머가 분주된 클락을 공유하는 방법은 사용하지 않는다. 둘째 유형의 장점은 별도의 시간 지연 명세가 필요 없으며 VHDL의 구문만으로도 시간 지연 명세가 가능하다는 점이다. 이 논문에서는 두 번째 유형의 입력 명세로부터의 시간 지연 합성에 중점을 둔다. 이 경우에 합성 과정은 최적화와 타이머 생성 및 삽입의 두 단계로 이루어진다.

다음 절에서는 제안된 시스템에서 필요로 하는 시간 지연 명세와 함께 이를 구현하기 위한 타이머의 구조를 설명한다. Ⅲ 절에서는 입력 명세로부터 최적화 알고리듬의 수행에 필요한 그래프 모델을 추출하는 방법을 설명한다. Ⅳ절에서는 주로 타이머의 개수를 줄이기 위한 타이머 공유 최적화 알고리듬을 설명한다. Ⅴ절에서는 제안된 합성 시스템의 구현과 실험 결과에 대하여 고찰한다. 마지막으로 결론과 함께 앞으로의 연구 방향에 대하여 기술한다.

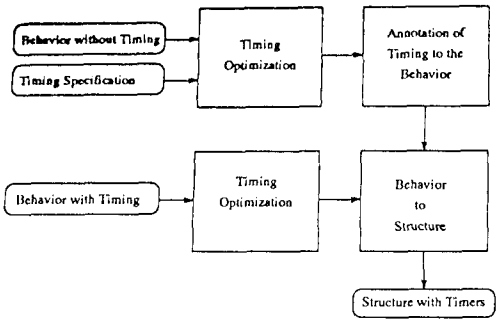


그림 1. 시간 지연 합성 시스템
Fig. 1. Timing synthesis system.

Ⅱ. 시간 지연 명세와 타이머의 구조

시간 지연 명세는 '정해진 조건 하에서 두 이벤트(event) 사이의 지연 시간'의 집합으로 주어진다. 본 논문에서는 이 두 이벤트를 각각 소스(source) 이벤트와 타겟(target) 이벤트로 부른다. 지연 시간을 표현하는 방법은 그 범위를 명시하는 유형에 따라 세 가지로 분류할 수 있다. 첫째 방법은 최대 지연 시간을 지정하는 것으로 소스 이벤트가 발생하고 나서 주어진 시간 내에 타겟 이벤트가 발생해야 함을 뜻한다. 이는 기존의 합성 시스템에서 합성된 하드웨어의 최소한의 성능을 지정하기 위한 방법으로 널리 사용하고 있다. 둘째로 생각할 수 있는 방법은 최소 지연 시간을 지정하는 것으로 소스 이벤트가 발생하고 나서 적어도 주어진 시간은 지나야 타겟 이벤트가 발생함을 뜻한다. 이는 매우 제한된 응용 범위를 가지며 널리 사용되지는 않는다. 마지막으로 최소와 최대 지연 시간을 모두 지정하는 것으로 소스 이벤트가 발생하고 나서 일정 시간이 흐른 후에 타겟 이벤트가 발생해야 하는 경우에 사용할 수 있다. 이 때, 최소와 최대값을 줌으로써 허용 오차를 표현할 수도 있고 주어진 허용 범위 내의 오차는 무시해도 좋다는 가정하에 두 값을 같은 값으로 지정할 수도 있다. 세번째 방법은 상위 단계 합성에서 스케쥴링 목적으로 사용되기도 한다.[5]

본 논문에서는 세 번째 유형의 시간 지연을 다룬

다. 이런 유형의 시간 지연을 구현하기 위하여 지연 소자나 타이머의 사용을 고려할 수 있다. 지연 소자의 사용은 시스템 클락 주기에 비하여 상대적으로 작은 시간 값을 갖는 시간 지연을 구현하는데 적당하고 타이머의 사용은 시스템 클락에 비하여 상대적으로 큰 시간 값을 갖는 시간 지연을 구현하는 데 적당하다. 한 개 또는 몇 개의 시스템 클락 주기에 해당하는 시간 지연을 갖는 경우는 기존의 합성 시스템[5,6]에서와 같이 FSM(Finite State Machine)의 control state로 쉽게 구현할 수 있다. 그러나 제어기에서 흔히 요구되는 것과 같이 시스템 클락에 비하여 매우 큰 값을 갖는 시간 지연을 구현하려면 control state가 많아지고 FSM이 커지게 되어 설계가 어려워지며 비효율적인 설계가 되기 쉽다. 여기에서는 control block과는 독립적으로 타이머를 사용하여 시간 지연을 구현하는 방법을 제안한다. 타이머도 일종의 FSM으로 하나의 작은 control block으로 생각할 수도 있다. 그러나 다른 control block과는 그 성격이 많이 다르므로 독립적으로 설계하는 것이 쉽고 효과적이다. 예를 들면 최적화된 카운터 구조, 즉, 래치나 플립플롭의 클락으로 시스템 클락이 아니라 다른 래치나 플립플롭의 출력을 연결하는 구조를 사용함으로써 쉽게 최적화된 타이머를 구현할 수 있다. 또한 높은 정밀도가 요구되지 않는 곳에서는 시스템 클락보다 큰 주기를 갖는 별도의 타이머 클락을 사용할 수도 있으며 이 경우 필요한 state수가 줄며 따라서 하드웨어의 크기를 많이 줄일 수 있다.

본 논문에서 제안하는 방법은 시간 지연을 표현할 수 있는 HDL이면 마찬가지로 적용될 수 있으나 여기에서는 IEEE의 표준으로 되어 있는 VHDL[7]을 사용하여 시간 지연을 표현한다. 합성을 고려할 때 VHDL의 시간 지연 구문이 갖는 문제점은 지연 시간의 허용 오차를 표현할 수 없다는 점이다. 그러므로 여기에서는 VHDL 표현과 함께 허용 오차를 입력하도록 한다. VHDL의 시간 지연 구문은 오차의 허용 범위만 여유 있게 주어지면 본 논문에서 제안하는 시간 지연 합성의 방법을 이용하여 그 의미의 바꿈이 없이 그대로 합성할 수 있다.

VHDL의 시간 지연 구문으로는 다음과 같은 것이 있다.

wait for time_expression;

after time_expression;

Wait for 구문은 process문이나 서브프로그램 내에서 순차적인 실행 사이의 시간 간격을 표현한다. 실행 도중에 이 구문을 만나면 시간 수식(time expression)이 갖는 시간 값만큼 기다린 후에 다음 문장을 실행한다. After 구문은 signal assignment 구문이나 disconnection 구문에 사용되는데 전자의 경우 신호의 값이 바뀌는 시점을, 후자의 경우 guarded signal과 driver사이의 연결이 끊어지는 시점을 표현한다.[7] 시간 수식의 값은 시간에 따라 변할 경우 합성이 어려워지므로 본 논문에서는 elaboration 과정까지는 그 값을 결정할 수 있는 정적 수식(static expression)으로 제한한다.

VHDL에는 시간 지연 모델로서 inertial 지연과 transport 지연이 정의되어 있다. 그림 2에서 (a)는 inertial 지연 모델을 사용하는 concurrent signal assignment 문에 대한 시간 지연 도표이다. Driver

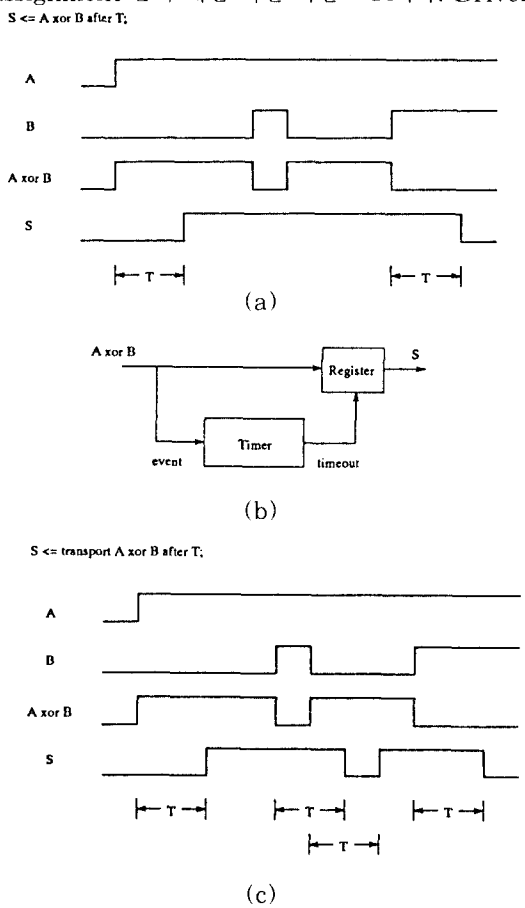


그림 2. VHDL의 지연 모델과 타이머를 이용한 합성 (a) inertial 지연 모델 (b) 합성된 하드웨어 (c) transport 지연 모델

Fig. 2. Delay models in VHDL and the synthesis with timers. (a) inertial delay model, (b) synthesized hardware, (c) transport delay model.

의 연산 결과가 T 시간 후에 타겟 S에 전달되는데 만일 그 값이 T 시간 동안 유지되지 않으면 그 값은 타겟에 전달되지 않는다. (b)는 이 구문을 타이머를 사용하여 합성했을 때의 가상적인 하드웨어이다. 이 하드웨어에서 타이머는 입력 신호에 이벤트가 발생했는가를 탐지하여 카운터를 동작시키고 T 시간에 해당되는 만큼 클락을 센 후에 종료(timeout) 신호를 레지스터에 보내어 바뀐 값을 출력과 동시에 기억시킨다. 만일 타이머가 동작하고 있는 동안에 입력에 새로운 이벤트가 탐지되면 카운터를 초기화하고 클락을 처음부터 다시 세기 때문에 먼저 발생했던 이벤트는 출력 S에 전달되지 않게 된다. 따라서 이러한 하드웨어를 이용하면 inertial 지연 모델을 쉽게 구현할 수 있다. 다만, 입력 신호가 비동기 신호인 경우 이벤트가 발생하는 시점과 타이머가 클락을 세기 시작하는 시점 사이에 시간 간격이 생길 수 있다. 그러나, 이로 인한 오차는 클락의 한 주기보다 작으며 지연 시간은 클락의 한 주기에 비해 훨씬 길다는 가정을 고려하면 그 정도의 오차는 무시해도 좋을 것이다 (현재 구현된 합성 시스템에서는 설계자가 허용 오차를 주게 되며 합성 결과 오차가 허용치를 벗어나게 되면 에러 메시지를 발생시킨다) 이와는 달리 transport 지연 모델은 (c)에서 보듯이 입력 신호의 모든 이벤트를 T 시간 후에 출력 S에 전달해야 한다. 따라서 모든 입력 이벤트에 대하여 지연 시간 T를 계산하여 기억하고 있어야 하므로 타이머를 이용하여 구현하기 어렵다. 본 논문에서는 inertial 지연 모델에 의한 시간 지연 명세만 고려한다.

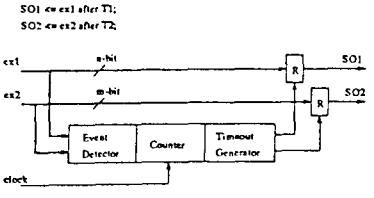


그림 3. 타이머 구조
Fig. 3. Timer Architecture.

시간 지연 명세의 하드웨어 구현을 위해서는 그림 3에서 보듯이 시간 지연을 위한 타이머와 지연 시간 동안 출력 신호의 값을 유지하기 위한 레지스터가 필요하다. 이 그림은 하나의 타이머가 두 시간 지연 구문에 의하여 공유될 수 있다고 가정했을 때의 예를 보여 준다. 타이머 자체는 입력 신호의 이벤트를 탐지하는 이벤트 탐지기, 클락 펄스를 세기 위한 카운터, 그리고 출력 측 레지스터에 지연 시간 후 종료 신호를 내보내는 종료 신호 발생기로 되어 있다. 종료 신호는 레지스터의 클락 입력에 가해져서 입력 신호의 값을 출력 신호에 전달시키는 역할을 한다. 그림에서 두 레지스터의 bit 크기는 각각 SO1과 SO2의 bit 크기이고 이벤트 탐지기의 개수는 SO1과 SO2의 bit 크기의 합과 같다. 종료 신호 발생기의 개수는 서로 다른 시간 지연의 개수와 같다.(이 예에서는 두 개) 각 시간 지연마다 별개의 타이머를 사용할 경우에는 종료 신호 발생기뿐만 아니라 카운터도 여러 개 필요하게 되는데 이를 하나의 타이머로 구현할 경우에는 여러 개의 카운터 중에 가장 큰 지연 시간을 위한 것 한 개만 있으면 되므로(종료 신호 발생기는 여러 개 필요함) 하드웨어의 크기를 많이 줄일 수 있다.

카운터는 입력 신호에서 이벤트가 탐지 되면 초기화되고 클락 펄스를 세기 시작한다. 지연 시간과 카운터가 세는 클락 펄스의 수와의 관계는 오차를 무시하면 다음과 같다.

지연 시간 = 클락 주기 × 클락 펄스의 수

Ⅲ. 이벤트 의존 그래프

타이머를 이용하여 시간 지연 합성을 할 때 타이머의 개수를 줄이기 위해서는 하나 이상의 시간 지연 구문들 사이의 타이머 공유 가능성을 조사할 필요가 있다. 예를 들면, 서로 다른 시간 지연 구문을 구현하기 위한 타이머의 이용 시간이 전혀 중첩되지 않거나 또는 타이머 구동 시점이 항상 일치한다면 타이머 이용상의 충돌이 발생하지 않으므로 공유가 가능해진다. 여기에서 타이머 이용상의 충돌이란 서로 다른 두 개 이상의 시간 지연 구문이 공유하고 있는 타이머가 한 시간 지연 구문에 의하여 동작하고 있을 때 다른 시간 지연 구문에서 발생하는 이벤트에 의해 초기화 상태가 됨으로써 원하는 시간에 종료 신호를 발생시키지 못하는 경우를 말한다. 어떤 시간 지연 구문들이 하나의 타이머를 공유할 수 있는지 알기 위해서는 시간 지연 구문들과 이들을 활성화시키는 이벤트들 사이의 의존성(dependency)을 조사할 필요가 있다. 본 논문에서는 다음과 같이 EDG(이벤트 의존 그래프, Event Dependency Graph)를 정의하여 사용한다.

$G(V_s, V_d, E)$: 이벤트 의존 그래프

V_s : 소스 이벤트 또는 타겟 이벤트를 나타내는 vertex (signal vertex)

V_d : 신호의 driver가 가지는 값을 생성해 내는 수식을 나타 내는 vertex (driver vertex) 로 assignment 문의 오른 쪽에 있는 수식

을 뜻함

E : V_s와 V_d 사이의 의존성을 나타내는 directed edge

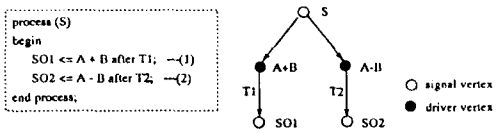


그림 4. EDG의 예

Fig. 4. Example of EDG.

그림 4는 주어진 process 문으로부터 생성된 EDG의 예를 보여 준다. Process 문은 sensitivity list에 있는 신호 S의 값이 변하면 활성화되고 따라서 두 assignment 문 (1)과 (2)가 수행되어 타겟 SO1과 SO2의 driver 값이 바뀌게 된다. 새로운 이벤트는 지정된 시간(T1과 T2) 후에 타겟에 나타나게 된다. 소스 이벤트 S와 타겟 이벤트 SO1, SO2는 EDG의 signal vertex로 표현되고 각 assignment 문의 수식은 driver vertex로 표현된다. 또한 소스 이벤트 S가 발생해야 driver 값이 바뀌고 driver 값이 바뀌어야 지정된 시간 후에 타겟에 이벤트가 발생하므로, S로부터 각 driver vertex로, 그리고 각 driver vertex로부터 타겟으로 directed edge의 연결이 필요하다.

EDG를 만들 때 고려해야 할 VHDL의 구문에는 process 문(sensitivity list), signal assignment 문, wait 문 등이 있다. 모든 concurrent signal assignment는 동등한 process문으로의 변환이 가능하므로 여기에서는 process 문에 대해서만 설명한다.

Process 문의 활성화에 영향을 주는 구문은 sensitivity list와 wait 문이다. Sensitivity list는 wait on 구문이 process 내의 마지막 순차문(sequential statment)으로 쓰인 것과 동등하게 처리하면 된다. Wait on 구문에 포함된 신호들 또는 그러한 신호가 없는 경우 wait until 구문에 포함된 신호들은 모두 process 문을 활성화하는 신호이므로 EDG에서 소스 이벤트로 표현된다 Wait for 구문은 그 위치에서 지정된 시간만큼 process를 정지시켰다가 다시 활성화시키는 구문으로 EDG 상의 소스 이벤트로 작용하므로 이를 위하여 dummy 신호를 만들어 사용한다. 그림 5는 wait for 구문이 있는 경우의 EDG 예를 보여 준다. 이 예에서는 dummy 신호 TS와 이 신호를 T2 시간마다 반전시키는 assignment 문 "TS <= not TS after T2;"를 만들어 사용한다. 따라서 TS는 소스 이벤트인 동시에 타겟 이벤트로서 EDG에서 signal vertex가 되고 "not TS"는 EDG에서 driver vertex가 된다.

Ⅳ. 타이머 공유 최적화 알고리듬

시간 지연 명세에 포함된 모든 시간 지연을 각각 다른 타이머를 사용하여 구현한다면 불필요 하게 큰 하드웨어가 되기 쉽다. 가능하면 많은 시간 지연이 타이머를 공유하도록 하는 것이 하드웨어의 크기를 줄이는 한 가지 방법이다. 여기에서는 EDG를 이용하여 타이머의 개수를 최소화 하는 알고리듬을 제안한다. 이 알고리듬은 지역(local) 타이머 공유 최적화와 전역(global) 타이머 공유 최적화의 두 단계로 구성된다.

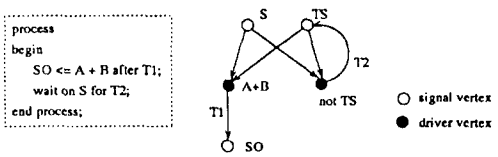


그림 5. Wait for 구문이 있는 process 문의 EDG.

Fig. 5. EDG for a process statement containing a wait for construct.

1. 지역 타이머 공유 최적화

지역 타이머 공유 최적화는 한 process 문 내에 여러 개의 순차문이 있고 그 사이에 wait 문이 없을 때 순차문 사이의 타이머 공유 가능성을 찾는 단계이다. 이때에는 각 순차문들 사이의 천이 일치(transition coherency)를 조사한다. 여기에서 천이 일치라는 것은 서로 다른 driver의 값이 항상 동일한 시점에서 변하여 타이머 구동 시점이 항상 일치하는 것을 말한다. 그림 6은 천이 일치가 없는 경우의 예와 관련 신호의 파형을 보여 준다. 다른 조건이 없는 한 이 예는 세 개의 타이머를 필요로 한다. 반면에 그림 7의 (a)는 세 개의 driver 값이 동일한 시점에서 변하는 예를, (b)는 두 개의 driver 값이 동일한 시점에서 변하는 예를 보여 준다. 이 경우에는 세 개 또는 두 개의 시간 지연을 한 개의 타이머로 만족시킬 수 있다.

천이 일치가 만족되는 모든 경우를 알아 내는 것은 쉽지 않다. 그러나 쉽게 알 수 있는 경우만 고려해도 도움이 된다. 그림 7 (a)와 같이 driver의 값이 정적 수식에 의하여 바뀌는 경우는 천이 일치가 만족되어 타이머를 공유할 수 있다. 이러한 경우는 수식에 신

호, 변수, 함수 등이 포함되어 있는가를 조사함으로써 쉽게 알 수 있다. 또한 driver의 값이 같은 수식에 의하여 변하거나 그림 7 (b)와 같이 이진 값을 갖는 수식에 not 연산만 추가되는 경우도 쉽게 알 수 있는 경우로 천이 일치가 만족되어 타이머를 공유할 수 있다.

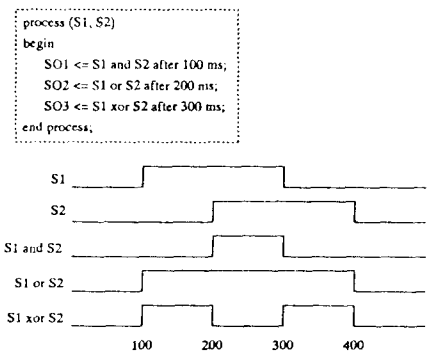


그림 6. 천이 일치가 없는 구문의 예

Fig. 6. Statements without transition coherency.

```
process (S1, S2)
begin
    SO1 <= 0, 1 after 100 ms;
    SO2 <= 1, 0 after 200 ms;
    SO3 <= 0, 1 after 300 ms;
end process;
```

(a)

```
process (S1, S2)
begin
    SO1 <= S1 and S2 after 100 ms;
    SO2 <= not (S1 and S2) after 200 ms;
end process;
```

(b)

그림 7. 천이 일치가 있는 구문의 예 (a) 정적 수식의 경우 (b) 같은 수식에 not 연산이 추가된 경우

Fig. 7. Statements with transition coherency. (a) case of static expression (b) case of not operation on the same expression.

2. 전역 타이머 공유 최적화

전역 타이머 공유 최적화는 하나 이상의 process 문을 고려하여 타이머 공유 최적화를 수행하는 단계이다. 여기에서는 다음과 같은 사실을 고려한다.

> 한 process 문 내에서, 중간에 wait 문이 삽입되어 있지 않는 여러 개의 assignment 문이 한 번 수행되고 나서 다시 수행될 때까지의 시간이 그 assignment 문이 갖는 지연 시간보다 항상 크다는 것이 보장되는 경우 그러한 모든 시간 지연들은 하나의 타이머로 구현할 수 있다. 중간에 wait 문이 삽입되어 있는 경우는 wait 문을 기준으로 하여 그 전에 오는 assignment 문과 후에 오는 assignement 문의 두 그룹으로 나누어 각각을 따로 고려하면 된다.

이 사실은 다음과 같이 설명할 수 있다.

우선 한 process 내에서 중간에 wait 문이 없고 마지막에만 wait 문이 있는 경우를 생각해 보면 다음과 같다. Process 내의 assignment 문들은 순차적이지만 모두 한 시뮬레이션 사이클 내에 수행되므로 타이머를 동시에 구동시키게 된다. 위의 사실에서 언급한 조건이 항상 보장되면 모든 assignment 문들이 수행되고 나서 다시 수행되기 전에 driver에 스케줄 되어 있는 모든 값들이 타겟 신호에 전달되므로 반복 수행될 때마다 타이머를 동시에 새로 구동시키게 된다. 따라서 하나의 타이머를 사용해도 충돌은 발생하지 않는다. 만일 어떤 assignment 문의 지연 시간이 길어서, 반복 수행될 때 driver에 어떤 값이 스케줄되어 있는 채로 남아 있는 경우에는 그 값이 새로 스케줄하고자 하는 값과 같으면 기존의 스케줄을 유지해야 한다.[7] 따라서 타이머를 다시 구동시키지 말고 계속해서 클락 펄스를 세도록 해야 하므로 이 경우에는 별도의 타이머를 사용해야 한다.

Process 문의 중간에 wait 문이 있는 경우에는 그 wait 문에 포함된 조건이 만족되어야 그 구문 다음에 오는 순차문들이 수행된다. 따라서 wait 문 전의 시간 지연들과 후의 시간 지연들이 타이머를 구동시키는 시점에는 차이가 있으므로 서로 다른 타이머가 필요하다. 그러나, 위의 조건이 보장되는 한 wait 문 전과 후의 모든 시간 지연들은 각각 하나의 타이머를 공유할 수 있다. 단, 중간에 wait 문이 하나만 있고 마지막에는 wait 문이 없는 경우에는(process 문의 sensitivity list도 없어야 함) wait 문 다음에 오는 순차문이 수행되는 즉시 wait 문 전의 순차문이 수행되므로 위의 조건이 만족되는 모든 시간 지연을 하나의 타이머로 구현할 수 있다 또한 전역 타이머 공유

최적화 단계에서는 다음과 같은 사실도 이용한다.

앞의 시간 지연 구문에서 스케쥴 된 값이 지연 시간 후에 타겟에 나타나고 이를 받아서 뒤의 시간 지연 구문이 활성화되어 스케쥴 된 값이 지연 시간 후에 타겟에 나타날 때까지 새로운 이벤트가 발생하지 않는다는 것이 보장되면 타이머를 공유할 수 있다.

특히 EDG상에 입력이 없는 사이클이 있으면 위의 조건이 만족되므로 그 사이클 안에 포함된 모든 시간 지연 구문은 wait 문의 유무에 상관 없이 하나의 타이머를 공유할 수 있다.

이와 같이 전역 타이머 공유 최적화 단계에서 이용하는 두 가지 사실 중 첫번째 사실은 하나의 타이머를 여러 개의 시간 지연이 동시에 구동시키는 경우이며 두번째 사실은 하나의 타이머를 여러 개의 시간 지연이 중첩되지 않게 사용하는 경우가 된다.

이상으로부터 다음과 같은 전역 타이머 공유 최적화 알고리듬을 만들어 낼 수 있다.(알고리듬의 흐름도는 그림 8 참고) 여기에서 *MEP*(최소 이벤트 주기, Minimum Event Period)라고 하는 것은 주어진 signal에 연이어 발생하는 event의 최소 시간 간격을 말한다.

1. VHDL 표현으로부터 EDG를 만든다.
2. EDG의 모든 입력 signal vertex(in-degree가 0인 signal vertex)의 *MEP*를 0으로 초기 화하고 다른 signal vertex의 *MEP*는 가능한 최대의 값(*tmax*)으로 초기화한다.
3. 각 입력 signal vertex로부터 recursion 방법으로 directed edge를 따라 EDG를 통과하면서 방문하게 되는 signal vertex의 *MEP*를 다음과 같이 정한다.
   - 바로 전에 방문한 signal vertex의 *MEP*를 MEP_{i-1}, 바로 전에 통과한 edge의 시간 지연 값을 DEL_{i-1}, 현재 방문 중인 signal vertex의 *MEP*를 MEP_i라고 할 때, $MEP_i = tmax$인 경우는 처음 방문하는 경우이므로 $MEP_i = \min(MEP_i, \max(MEP_{i-1}, DEL_{i-1}))$로 *MEP*를 갱신하고 현재의 가지에서의 갱신 과정을 계속한다.
   - $MEP_i < tmax$인 경우는 이미 방문한 signal vertex를 다시 방문하게 되는 경우이며 이벤트가 어떤 간격으로 들어올지 알기 어려우므로 그 signal vertex의 MEP 를 0으로 한 후 현재의 가지에서의 갱신 과정을 계속한다.

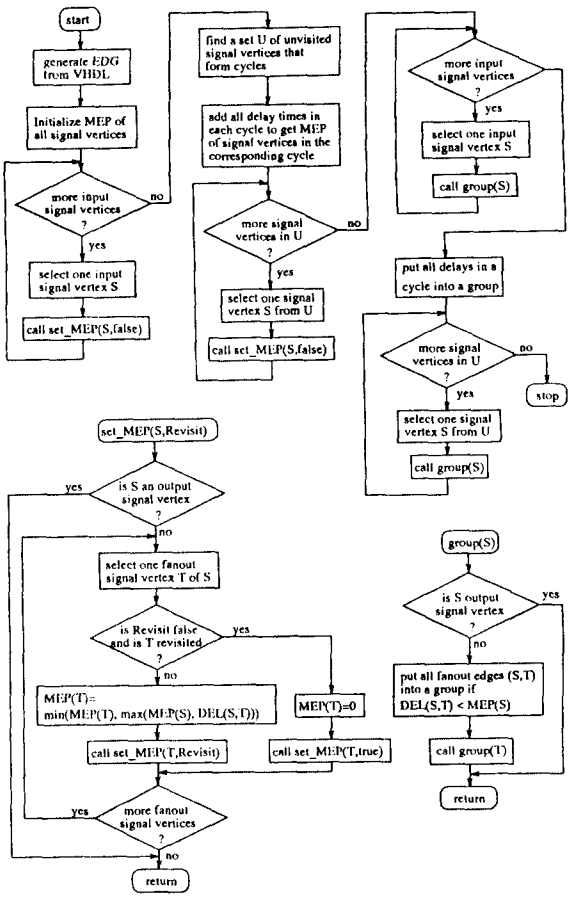


그림 8. 전역 타이머 공유 최적화 알고리듬의 흐름도.

Fig. 8. Flow chart of the global optimization algorithm for sharing timers.

   - 출력 signal vertex에 도달하면 현재의 가지에서의 갱신 과정을 마친다.
   - 모든 입력 signal vertex에 대하여 위의 과정을 거친다.
   - 방문하지 않은 입력으로 하기 때문이며 각 사이클에 포함된 모든 지연 시간의 합 을 구하여 그 사이클을 이루는 모든 signal vertex들의 *MEP*로 하고 이로부터 그 사이클의 출력 signal vertex들의 *MEP*를 구해 나간다.
4. EDG를 다시 recursion 방법으로 통과하면서 다음과 같이 시간 지연들을 그룹으로 묶는다.
   - 현재 방문 중인 signal vertex를 소스로 하는 edge의 시간 지연 중 그 값이 소스 signal vertex의 *MEP*보다 작은 것들은 모

두 하나의 그룹으로 묶는다.

- 입력이 없는 사이클을 이루는 모든 시간 지연들은 모두 하나의 그룹으로 묶는다.

5. 하나의 그룹에 속하는 시간 지연들은 모두 하나의 타이머로 구현한다.

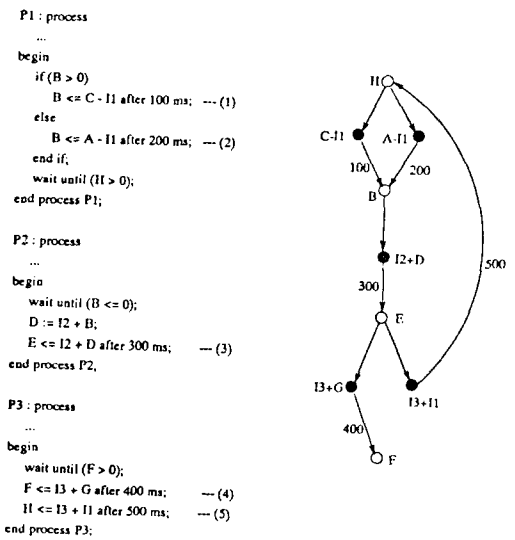


그림 9. 전역 타이머 공유 최적화 알고리듬의 적용 예
Fig. 9. Example for an application of the global optimization algorithm.

그림 9는 전역 타이머 공유 최적화 알고리듬을 적용한 예이다. 그림에서 EDG는 입력이 없는 하나의 사이클을 가지며 최소 주기는 900 ms가 된다. 따라서 그 사이클을 구성하는 모든 signal vertex의 *MEP*는 900 ms가 되며 모든 edge의 시간 지연은 하나의 그룹으로 묶인다. 또한 signal vertex E를 소스로 하고 signal vertex F를 타겟으로 하는 edge의 지연 시간 400 ms는 signal vertex E의 *MEP*인 900 ms보다 작으므로 같은 signal vertex를 소스로 하고 타겟이 H인 edge(지연 시간 500 ms)와는 같은 그룹에 속한다. 결과적으로 다섯 개의 시간 지연 모두가 하나의 그룹에 속하게 되어 하나의 타이머로 구현할 수 있게 된다. 이때 시간 지연 구문 (4)와 (5)는 타이머를 동시에 구동시키며 다른 시간 지연 구문들은 각각 다른 시점에 타이머를 구동시킨다. 시간 지연 구문 (1)과 (2)는 EDG에서 해당 edge가 병렬로 연결되어 있지만 if 문의 조건에 따라 서로 배타적으로 수행되므로 타이머의 구동 시점은 서로 달라진다.

V. 시간 지연 합성 시스템의 구현 및 실험 결과

본 논문에서 제안하는 방법은 VHDL의 시간 지연 구문을 최적화 과정을 거쳐 타이머의 instance로 바꾸어 주는 것이다. 이는 시간 지연 명세에 해당되는 부분만 합성하는 것이며(시간 지연 합성) 기능 명세 부분은 따로 합성해야 한다.(기능 합성) 그림 10은 하나의 process 문 내의 두 시간 지연 구문이 하나의 타이머로 구현될 때 기능 부분과 시간 지연 부분의 합성된 모양을 보여준다. VHDL 표현을 보면 시간 지연 명세 부분이 타이머 instance로 변환된 것을 알 수 있다. 또한 각각의 driver에 대하여 새로운 신호를 정의하여 타이머를 구동시키는 신호로 사용한다. 지연 시간은 타이머 component의 generic으로 하여 필요한 타이머 instance를 만들어 낼 수 있도록 한다. 입출력 신호의 개수와 type에 따라서 각각 다

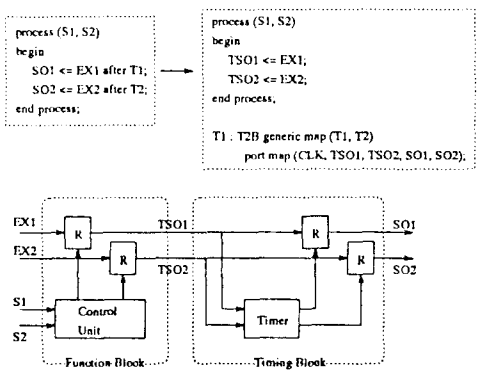


그림 10. 기능 합성과 시간 지연 합성의 예
Fig. 10. Example of function synthesis and timing synthesis.

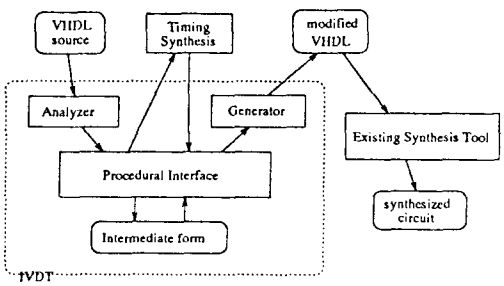


그림 11. 시간 지연 합성 시스템의 환경
Fig. 11. Timing synthesis system environment.

른 타이머를 라이브러리로부터 가져 오게 되는데 이것도 generic을 이용하면 라이브러리를 더욱 간단하게 만들 수도 있다 제안된 시스템은 이미 개발되어 있는 툴키트인 IVDT [8] 를 이용하여 C 프로그래밍 언어로 구현하였다. 그림 11은 시간 지연 명세가 있는 VHDL구문을 위한 합성 시스템의 환경과 합성 과정을 보여 준다. VHDL 소스 파일은 분석기

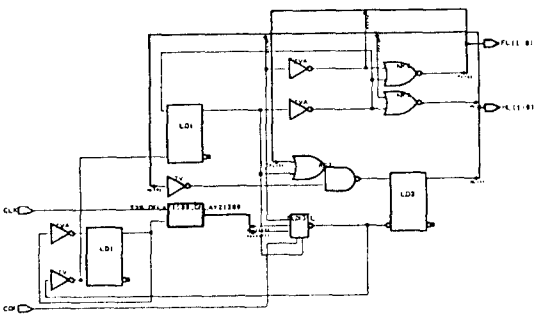
```
process (ST)
begin
   TOL <= '0', '1' after Long_Time;
   TOS <= '0', '1' after Short_Time;
end process;
```

→

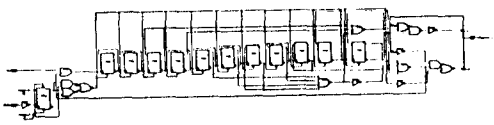
```
process (ST)
begin
   TOL_1 <= '0'; TOL_2 <= '1';
   TOS_1 <= '0'; TOS_2 <= '1';
end process;

T1 : T2S generic map (180, 1200)
        port map (clk, ST, tout);

TOL <= TOL_2 when tout(2)='1'
          else TOL_1;
TOS <= TOS_2 when tout(1)='1'
          else TOS_1;
```

그림 12. 교통 신호 제어기 1의 시간 지연 합성 결과

Fig. 12. Timing synthesis results of the traffic light controller 1.



(a)

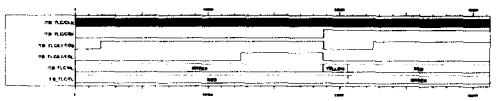


(b)

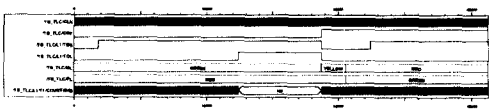
그림 13. 교통 신호 제어기 1의 합성 회로. (a) 전체 회로 (b) 타이머의 내부 회로

Fig. 13. Synthesized circuit of the traffic light controller 1. (a) whole view of the circuit (b) circuit of the timer block.

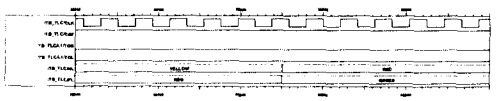
(analyzer)를 통해 분석되어 중간 형태(intermediate form)로 만들어진다. 시간 지연 합성기는 중간 형태의 코드로부터 IVDT의 procedure들을 이용하여 EDG를 만들어 시간 지연 합성을 하고 그 결과로부터 역시 IVDT의 procedure들을 이용하여 중간 형태를 변경한다. [8] 즉, 중간 형태에서 모든 시간 지연 구문을 타이머의 instance로 바꾸어준다. 이렇게 변경된 중간 형태로부터 재생기(generator)를 통해 변경된 VHDL 파일을 생성해 낸다. 생성된 VHDL 파일은 원래 VHDL의 시간 지연 구문으로 되어 있는



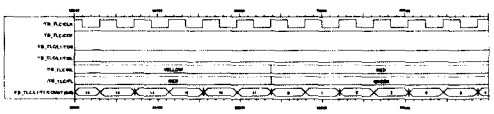
(a)



(b)



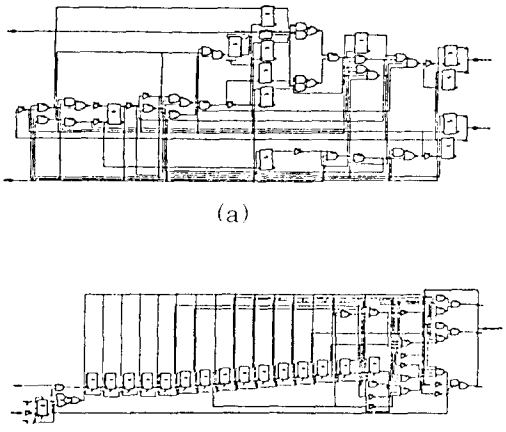
(c)



(d)

그림 14. 교통 신호 제어기 1의 모의 실험 결과 (a) 시간 지연 합성 전 (b) 시간 지연 합성 후 (c) 시간 지연 합성 전(확대한 파형) (d) 시간 지연 합성 후(확대한 파형)

Fig. 14. Simulation results for the traffic light controller 1. (a) before timing synthesis (b) after timing synthesis (c) before timing synthesis(zoomed-in waveforms) (d) after timing synthesis(zoomed-in waveforms)



(a)

(b)

그림 15. 교통 신호 제어기 2의 합성 회로
(a) 전체 회로 (b) 타이머의 내부 회로

Fig. 15. Synthesized circuit of the traffic light controller 2. (a) whole view of the circuit (b) circuit of the timer block

것을 타이머의 instance로 바꾸어 더 이상 VHDL의 시간 지연 구문을 포함하지 않으므로 기존의 합성기[3]를 이용하여 합성할 수 있다. 이와 같이 시간 지연 구문을 허용하지 않는 기존의 합성기를 이용하여 시간 지연 구문을 허용하는 합성 시스템을 쉽게 구현할 수 있다.

구현된 시스템을 이용하여 교통 신호 제어기[9]를 합성해 보았다. 여기에서 타이머의 클락 주파수는 60 Hz로 하였다. 그림 12는 교통 신호 제어기를 VHDL로 표현한 후 타이머를 이용하여 시간 지연 합성을 했을 때 VHDL 표현에서 변경된 부분을 보여 준다.

그림에서 Long_Time(20 초)과 Short_Time(3 초)의 시간 지연이 한 개의 타이머 instance T1으로 구현됨을 알 수 있다. 따라서 이 시간 지연들을 각각 다른 타이머로 구현하는 것보다 작은 면적을 차지하게 된다. 이 예에서는 지역 타이머 공유 최적화 과정만 적용된다. 그림 13에 보인 것은 시간 지연 합성 결과로부터 기능 합성 과정을 거쳐서 생성해낸 회로이다. 그림에서 (a)는 합성된 전체 회로이며 (b)는 여기에 사용된 타이머(전체 회로에서 T2S_DELAY1180_DELAY21200)의 내부 회로이다.

그림 14는 합성하기 전과 합성한 후의 VHDL표현을 가지고 모의 실험을 하여 얻은 파형을 보여 준다. (a)와 (b)에서 시간의 미세한 차이는 볼 수 없으나 전체적으로 합성 전과 합성 후의 출력 파형이 일치하는 함을 확인할 수 있다. (c)와 (d)는 확대한 파형으로 합성 전과 합성 후의 출력 파형이 한 클락 주기 이내의 오차를 가질 수 있음을 볼 수 있다.

그림 15는 기능이 좀 더 복잡한 교통 신호 제어기 [10] 의 합성 결과이다. 이것은 5 개의 시간 지연 구문을 가지며 지연 시간의 값은 20 초, 180 초, 360 초의 세가지가 있는데 모두 하나의 타이머(T3S DELAY11200_DELAY210800_DELAY321600)로 구현되었다. 여기에서도 지역 타이머 공유 최적화만 적용되었다.

그림 16은 간소화한 승강기 제어기의 합성 결과이다. 이 예에서 승강기 제어는 버튼 제어, 출입문 제어, 모터 제어로 구성되며 출입문이 열려 있는 시간, 모터의 구동과 출입문 제어 사이의 시간 간격 등 네 개의 시간 지연 구문을 갖고 있는데 그림에서 보듯이 세 개의 종료 신호를 발생하는 하나의 타이머로 구현되었다. 여기에서는 전역 타이머 공유 최적화가 적용되었다. 이 예에서는 외부의 event dependency를 함께 표현하여 타이머의 개수를 줄였는데 예를 들면 출입문을 닫는 신호가 출력이 되어야만 출입문이 닫혔다는 감지 신호가 입력된다는 표현이 dummy process 문으로 추가되었다. 이렇게 하면 EDG에 사이클이 형성되어 전역 타이머 공유 최적화의 적용이 가능해진다. 물론 이러한 dummy process는 기능 합성 과정에서는 제외된다. 이와 같이 설계자가 설계하는 시스템의 외부 조건을 함께 표현해 주면 하드웨어의 크기를 더욱 줄일 수 있게 된다.

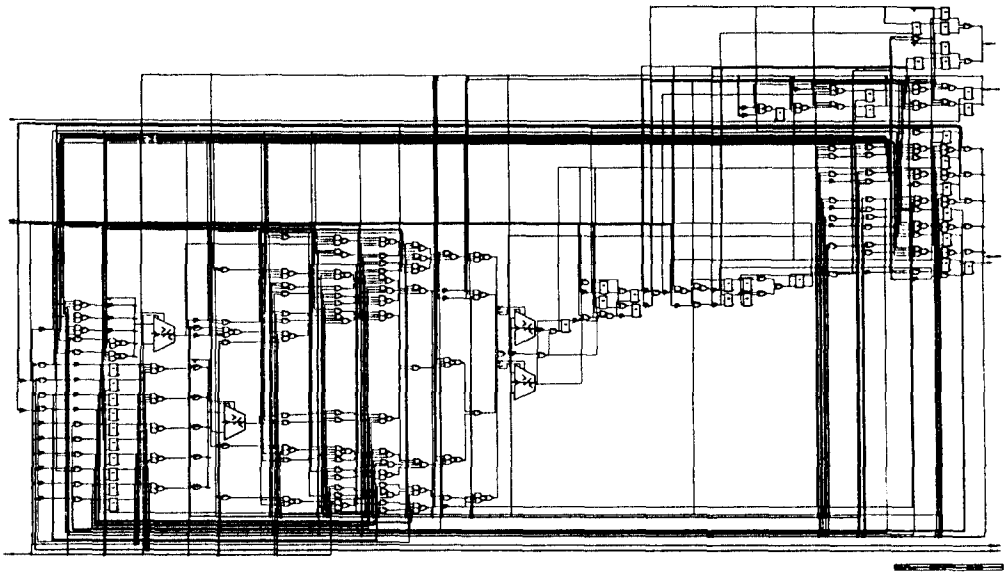
모든 실험에서 원래의 설계에 대한 시뮬레이션 결과와 시간 지연 합성 및 기능 합성을 거쳐 생성된 회로에 대한 시뮬레이션 결과는 한 클락 주기의 오차 범위 내에서 일치함을 확인하였다.

표 1. 기존 방법과 제안된 방법의 성능 비교 (기존/제안)

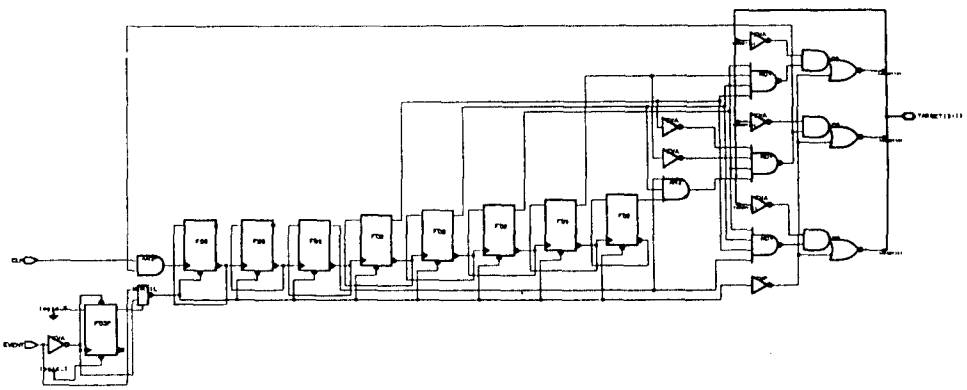
Table 1. Performance of the proposed method as compared with existing one (existing/proposed).

example	no. of cells	no. of nets	area	cpu time (sec)
traffic light controller 1	53/31	55/40	139/116	56.2/177.1
traffic light controller 2	117/54	120/76	297/238	6268.8/234.4
elevator controller	-/343	-/365	-/900	-/429.5

표 1은 본 논문에서 제안하는 방법과 기존의 합성 시스템[3]만을 이용하여 FSM 합성을 한 것을 비교한



(a)



(b)

그림 16. 승강기 제어기의 합성 회로
(a) 전체 회로 (b) 타이머의 내부 회로
Fig. 16. Synthesized circuit of the elevator controller.
(a) whole view of the circuit, (b) circuit of the timer block.

것이다. 후자의 경우 VHDL 표현으로는 합성이 불가능하므로 지연 시간에 해당되는 클락 수를 계산하여 state table을 만든 후 FSM 합성 기능을 이용하여 합성하였다. 기술 사상(technology mapping)은 standard cell library(lsi_10k)를 이용하였다. 표를 보면 기존의 방법과 비교하여 본 논문에서 제안하는 방법이 더 작은 면적 (〈 85 퍼센트)에 구현할 수 있음을 알 수 있다. 여기에서 제안된 방법이 cell 수는 훨씬 작은데 면적은 그만큼 차이가 나지 않는 이유는 cell 중에 면적을 많이 차지하는 latch의 수가 증가하기 때문이다. 이들 latch는 timer가 공유될 때 어떠한 시간 지연을 구현하고 있는가를 기억하는 데 사용된다. 교통 신호 제어기 1의 경우 본 논문에서 제안하는 방법이 더 오래 걸리는데 이는 타이머를 이용한 시간 지연 합성에 걸리는 시간이 추가로 걸리기 때문이다. 그러나 좀 더 복잡한 교통신호 제어기 2의 경

우 제안하는 방법이 기존의 방법보다 26배 정도 빠르다. 이는 시간 지연을 기존의 방법으로 FSM 합성을 할 때 state가 많고 회로가 복잡하여 합성 시간이 급증하기 때문이다. 이러한 경향은 설계가 복잡할 수록 심해져서 승강기 제어기의 경우는 기존의 방법으로는 너무 오래 걸려서 결과를 얻지 못하였으나 제안하는 방법으로는 쉽게 결과를 얻을 수 있었다. 다른 실험 결과도 기존의 방법으로 합성할 경우 합성 시간이 너무 오래 걸려서 시스템 클락을 1/60초에서 1초로 늘여서 얻었다.

Ⅵ. 결론

시스템 클락의 주기와 비교하여 매우 긴 시간 지연을 필요로 하는 하드웨어를 타이머를 사용하여 합성하는 방법을 제안하였다. 이 방법은 타이머를 이용하지 않는 기존의 FSM 합성 방법에 비하여 복잡한 설계의 경우 더 작은 면적에 훨씬 빠르게 합성한다. 제안된 방법은 VHDL 표현으로부터 타이머 공유 최적화 과정을 거쳐 타이머의 개수를 최소화 하면서 주어진 시간 지연을 만족하는 하드웨어를 합성한다. 타이머 공유 최적화 알고리듬은 VHDL 표현으로부터 추출하게 되는 그래프 모델을 이용하여 가능하면 적은 수의 타이머로 모든 시간 지연을 만족시키는 방법을 찾아 낸다. 이 알고리듬은 VHDL 툴키트인 IVDT와 시간 지연 구문이 허용되지 않는 기존의 합성기를 이용하여 구현하였다. 그 효과는 몇가지 예비적인 실험을 통하여 확인하였다 본 논문에서 제안하는 방법은 기존의 합성 시스템이 효율적으로 구현하지 못하는 긴 시간 지연을 타이머를 이용하여 효율적으로 합성하며 또한 VHDL의 시간 지연 구문을 그 의미대로 합성할 수 있다는 데 의의가 있다. 물론 이 방법이 VHDL 표현에만 국한되는 것은 아니며 시간 지연 구문을 포함하는 다른 HDL 표현에도 마찬가지로 적용될 수 있다. 그러나 현재 구현된 시스템은 시스템 클락의 주기를 정해 놓고 합성을 시작해야 하며 모든 시간 지연에 대하여 단 하나의 허용 오차만 표현하는 등의 몇 가지 부족한 점을 갖고 있다. 현재 각 시간 지연을 최소값과 최대값으로 표현하고 기능은 VHDL 표현으로 주어졌을 때 이로부터 합성하는 방법을 연구 중이다. 또한 본 논문에서는 타이머의 개수를 최소화하는 방법만 다루고 있는데 타이머 각각의 크기를 최소화하는 방법도 연구 중이다. 이를 위하여 두가지 방법을 생각할 수 있는데 그 중 한가지는 허용 오차 범위 내에서 종료 신호 발생기 회로를 간소화하는 것이며 다른 한 가지는 시스템 클락을 분주하여 얻은 새로운 클락을 여러 개의 타이머 클락으로 사용할 수 있도록 하여 오차 범위 내에서 타이머의 크기를 최대한 줄이는 것이다.

본 논문에서 제안하는 방법은 타이머를 이용하여 합성하는 방법은 시간 지연의 구현을 하드웨어에 의존하는 것이지만 앞으로의 연구 방향으로 같은 개념을 소프트웨어에 의한 시간 지연의 구현에 적용하거나 하드웨어와 소프트웨어의 동시 설계로 확장하는 것을 고려하고 있다.

參考文獻

[1] Intel Corporation, Mt. Prospect, Illinois, *8-Bit Embedded Controllers*, 1991.

[2] Samsung Electronics, Kiheung, *KS88 C0016 8-Bit CMOS Microcontroller User's Manual*, 1993.

[3] Synopsys Inc., Mountain Wiew, California, *Design Compiler R eference Manual*, 1991.

[4] Viewlogic Systems Inc., Marlboro, Massachusetts, *VHDL User's Guide*, 1989.

[5] D. Ku and G. D. Micheli, "Relative scheduling under timing constraints," in Proc. *27th ACM/IEEE Design Automation Conference*, (Orland), pp. 59-64, June 1990.

[6] A. Stoll and P. Duzy, "High-level synthesis from VHDL with exact timing constraints," in Proc. *29th ACM/IEEE Design Automation Conference*, (Anaheim), pp 188-193, June 1992.

[7] The Institute of Electrical and Electronics Engineers, Inc., New York, NewYork, *IEEE Standard VHDL Language R eference Manual, IEEE Std* 1076-1987, 1988

[8] K. Choi, "ISRC VHDL developer's toolkit," Tech. Rep. ISRC 92-E-0015, Inter-University Semiconductor Research Center, Seoul National University, 1992.

[9] R. Lipsett, C. Schaefer, and C. Ussery, *VHDL: Hardware Description*

and Design. Norwell, Massachusetts: Kluwer Academic Publishers, 1989.
[10] J.-M. Berge, A. Fonkoua, and S. Maginot, *VHDL Designer's Reference*. Dordrecht, Netherlands: Kluwer Academic Publishers, 1992.

著 者 紹 介

崔 起 榮(正會員)
1955年 8月 30日生. 1978년 서울대학교 전자공학과 학사. 1980년 한국과학원 전기 및 전자공학과 석사. 1989년 미국 Stanford 대학 전기공학과 박사. 1978년 - 1983년 (주)금성사 중앙연구소 근무. 1989년 ~ 1991년 미국 Cadence Design System, Inc 근무. 1991년 ~ 현재 서울대학교 반도체공동연구소 및 전자공학과 조교수. 주관심 분야는 CAD, VLSI 설계 등임.

朴 相 憲(準會員)
1969年 10月 21日生. 1992년 전남대학교 전자공학과 학사. 1994년 서울대학교 전자공학과 석사. 현재 동 대학원 전자공학과 박사과정. 주관심 분야는 상위수준합성, 논리합성 등임.