Esterel에서 동기장치 중복사용 문제 검출시 과잉 경보 줄이기

(Reducing False Alarms in Schizophrenic Parallel Synchronizer Detection for Esterel)

윤 정 한 † 김 철 주 †† 김 성 건 † 한 태 숙 †††
(Jeong-Han Yun) (Chul-Joo Kim) (Seonggun Kim) (Taisook Han)

요 약 Esterel이라는 절차형(imperative) 동기(synchronous) 언어로부터 회로를 합성(synthesis)할 때, 하나의 동기장치(synchronizer)가 한 클럭에 중복사용되는 문제(schizophrenic parallel synchronizer)가 발생할 수 있다. 기존 컴파일러는 동기장치가 중복사용될 경우 동기장치를 복제하여 이 문제를 해결하고 있다. 본 논문은 동기장치가 중복사용되더라도 회로상/기능상 문제가 없는 조건을 제시하고, 이를 기반으로 소스코드를 분석하여 복제해야만 하는 동기장치를 찾아주는 알고리즘을 제안한다. 이 알고리즘은 컴파일러가 중복사용되는 동기장치들 중에서 꼭 복제해야만 하는 것을 알 수 있게 해 주어, Esterel 프로그램을 좀 더 작은 회로로 합성할 수 있도록 한다.

키워드 : Esterel, 동기식 언어, 회로 중복사용, 동기장치, 동기회로, 자원공유

Abstract Esterel is an imperative synchronous language well-adapted to control-intensive systems. When an Esterel program is translated to a circuit, the synchronizer of a parallel statement may be executed more than once in a clock; the synchronizer is called schizophrenic. Existing compilers cure the problems of schizophrenic parallel synchronizers using logic duplications. This paper proposes the conditions under which a synchronizer causes no problem in circuits when it is executed more than once in a clock. In addition we design a detection algorithm based on those conditions. Our algorithm detects schizophrenic parallel synchronizers that have to be duplicated in Esterel source codes so that compilers can save the size of synthesized circuits.

Key words : Esterel, synchronous language, schizophrenia, synchronizer, synchronous circuit, resource sharing

· 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2010-C1090-1031-0004)

† 학생회원 : KAIST 전산학과
jeonghan.yun@gmail.com
seonggun.kim@arcs.kaist.ac.kr
†† 정 회 원 : 삼성전자 Digital Media & Communications R&D Center 책임연구원
chuljoo.kim@gmail.com
††† 종신회원 : KAIST 전산학과 교수
han@cs.kaist.ac.kr

논문접수 : 2010년 4월 13일
심사완료 : 2010년 6월 10일

정보과학회논문지: 소프트웨어 및 응용 제37권 제8호(2010.8)

1. 서 론

동기식 언어[1,2]는 실제 시간을 논리적 단위시간(instant)으로 나누어, 프로그래밍 언어에서 하드웨어 타이밍 관련 문제들을 쉽게 관리할 수 있게 한다.

Esterel[3-5]이라는 절차형 동기식 언어로부터 회로를 생성할 때 동기 언어의 특성과 절차형 언어의 특성이 서로 얽히면서 회로 중복사용 문제(schziophrenia)[3,6]가 발생한다. 이는 Esterel 문장 하나가 한 단위시간에 2번 이상 수행되는 것을 뜻한다. 이런 경우 Esterel 프로그램을 회로로 변환하면, 하나의 기능단위가 한 클럭 안에 2번 이상 사용되어 문제를 일으킬 수 있다.

Esterel 컴파일러에서는 이러한 문제를 해결해야 한다. 기존 연구에서 여러 가지 해결책들[3,6,7]이 제시되어 있는데, 이 방법들의 기본 아이디어는 루프 펼치기(loop unrolling)이다. 루프 내의 코드를 한 번 복사하여

하나의 문장이 한 단위시간에 두 번 이상 수행되지 않도록 하는 것이다.

1.1 동기장치의 중복사용 문제

본 논문에서는 회로 중복사용 문제 중 동기장치 중복사용 문제[3]에 대해 자세히 알아보고자 한다. 기존 연구[3,6]에서는 동기장치가 중복사용되면 항상 문제가 발생할 수 있다고 가정하여 모든 경우에 대해 동기장치를 복제하도록 하였다.

동기장치가 중복사용 되더라도 아무런 문제가 발생하지 않는 조건을 찾아, 복제해야만 하는 동기장치를 찾아내는 알고리즘을 제시하고자 한다. 이는 동기장치의 불필요한 복제를 막아 생성되는 회로를 더욱 작게 만들 수 있게 해 준다.

Esterel에서는 쓰레드(thread)들이 하나의 클럭에 맞추어 동시에 수행된다. 동기장치는 동시에 수행되는 쓰레드들의 시간 동기화를 담당하는 회로이다.

병렬문이 한 단위시간에 여러 번 수행될 경우 해당 동기장치는 2가지 점에서 문제를 일으킬 수 있다.

1. 전기적 안정성: 동기장치의 중복사용으로 인해 불안정한 순환 회로(*cyclic circuit*)가 발생할 수 있다.

2. 기능적 안정성: 동기장치는 쓰레드들의 동기화를 수행해야 한다. 동기장치가 중복수행 될 경우 한 단위시간에 여러 번의 동기화를 동시에 수행해야 하는데, 이 경우 동기화를 잘못 수행할 수 있다.

하지만 동기장치가 중복사용 될 경우 항상 이러한 문제가 발생하는 것은 아니다. 동기장치 중복사용이 아무런 문제를 일으키지 않으면 컴파일러는 중복사용 되는 동기장치를 복제할 필요가 없다. 이런 경우 만약 병렬문이 다른 중복사용 문제의 해결 때문에 복제가 되더라도 해당 동기장치는 복제될 필요가 없다. 복제된 병렬문이 동기장치를 공유할 수 있기 때문이다.

1.2 논문의 구성

본 논문에서는 예제를 통해 동기장치 중복사용이 문제를 일으킬 경우에 대한 원인을 알아본다. 그리고 동기장치가 중복사용 되더라도 회로상/기능상 아무런 문제를 일으키지 않는 조건을 제시한다.

이 조건은 중복사용되는 동기장치들 중 컴파일러가 복제해야 하는 동기장치를 알아내는데 기초가 될 것이다.

본 논문은 다음과 같이 구성하였다. 2장에서 Esterel의 의미 구조와 하드웨어 컴파일 방법, 그리고 회로상에서의 조건 증명에 이용한 회로 모델을 소개한다. 3장에서는 동기장치가 중복사용되는 예제를 이용하여 동기장치가 중복사용 시 문제를 일으키지 않는 조건을 제시한다. 4장에서는 이 조건을 기반으로 중복사용 문제를 일으키는 동기장치를 찾아내는 알고리즘을 소개하고, 마지막으로 5장에서 결론 및 향후 연구 방향을 기술한다.

2. 사전 정보

2.1 Esterel 문법 및 의미

본 논문에서는 Pure Esterel의 문장을 대상으로 한다. 구체적인 문법과 간단한 의미는 그림 1에 소개되어 있다[3,4]. 여기서 p와 q는 문장을 나타내고, S는 신호, T는 선언된 예외를 의미한다.

`nothing`	아무 것도 하지 않음
`pause`	시간 소모
`emit S`	출력
`p; q`	연속
`present S then p else q`	신호 테스트
`loop p end`	루프 (무한히 반복)
`p || q`	병렬
`suspend p when S`	S가 출력되면 일시 중지
`signal S in p end`	지역신호 선언
`trap T in p end`	예외 선언
`exit T`	예외 발생

그림 1 Esterel 문법 및 의미

Esterel 프로그램은 외부 클럭에 맞춰 전체 프로그램의 동기화가 이루어진다. 이때 "pause"를 이용해 동기화 시점과 단위시간의 흐름을 결정할 수 있다.

Esterel 프로그램은 신호의 상태에 따라 동작(reaction)한다. 신호의 상태는 단위시간마다 켜지거나(present) 또는 꺼지게(absent) 되는데, "emit S"에 의해 켜지게 되면 단위시간이 끝날 때까지 그 상태가 유지된다.

2.2 Esterel로부터 회로 생성 규칙

우리는 Berry의 회로 변환 규칙[3]을 기준으로 하였다. 이 규칙은 Esterel 최신 버전[5]에서도 하드웨어 컴파일 규칙의 핵심 알고리즘으로 사용하고 있다[8].

그림 2는 변환 규칙의 일부를 보여준다. (a)는 하나의 Esterel 문장이 회로로 변환될 때의 기본 구조를 보여준다. 각 핀(pin)의 의미는 다음과 같다[3].

- GO: 문장을 시작하라는 입력 제어 신호
- RES: 현 문장의 내부에서부터 다시 수행하라는 입력 제어 신호
- SUSP: 현 문장을 이번 단위시간에 수행하지 못하게 하는 제어 신호
- KILL: 예외선언 블럭을 벗어나는 경우 해당 문장을 다음 단위시간에는 수행하지 못하게 하는 입력 제어 신호
- SEL: 다음 단위시간에 해당 문장이 아직 수행할 것이 남아 있음을 나타내는 출력 제어 신호
- K0, K1, ...: K0은 문장 수행이 끝났음을, K1은 pause가 수행되어 현재 단위시간까지의 수행은 끝났으나 다음에 할 일이 남아 있음을 뜻하는 완료코드(completion code). 그리고 2 이상의 Kn은 (n-1)번째 가까운 예외선언 블럭을 빠져나가는 출력 제어 신호

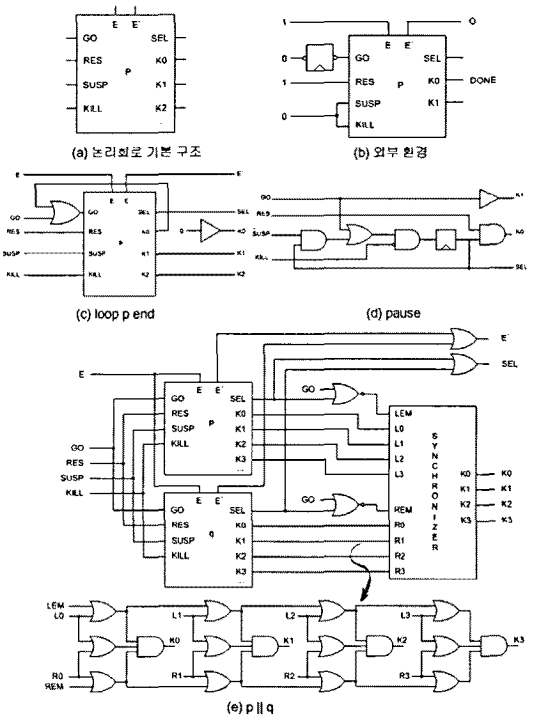

그림 2 Esterel 회로 생성 규칙[2]

• E, E′: 입출력 데이터 신호

프로그램 p를 회로로 변환 시 외부와의 연결은 그림 2(b)와 같이 이루어진다. 첫 번째 클럭에 GO를 켜 준 다음 계속 RES을 켜 주어 매 클럭마다 프로그램을 수행시킨다.

그림 2(c)부터 (f)까지는 루프, pause, 병렬문이 어떻게 회로로 변환하는지 그림으로 나타내고 있다. 그 외 문장에 대한 변환 규칙들은 본 논문에서는 생략한다. 그림 2(c) 루프는 루프가 끝나면(K0=1) 다시 시작한다. 그림 2(d) pause는 시작하면(GO=1) K1이 켜지면서, 다음 클럭에 다시 수행해야 함을 레지스터에 저장해 둔다. 다음 클럭에 RES이 켜지면 레지스터가 1일 때만 K0를 출력하여 수행이 끝났음을 알린다. 그림 2(e) 병렬문은 p, q 두 문장을 동시에 수행시킨다. 병렬문은 동기장치는 두 문장의 완료코드를 받아 그 중 큰 값을 병렬문의 완료코드로 선택하는 역할을 담당한다. LEM과 REM은 현 단위시간 이전에 해당 쓰레드가 종료되었는지를 확인하는데 사용된다.

2.3 회로 분석 모델

우리는 순환 회로 분석에 널리 사용되고 있는 삼치 논리(three-valued logic)[9]를 이용한다. 삼치 논리는 부울(bool) 값인 {0,1}을 세 개의 값 {0,1,X}로 확장한 것이다.

2개의 입력을 받는 AND 게이트(gate)의 경우 둘 중에 하나라도 0이면 나머지 값에 관계없이 결과는 0으로 결정된다. 이 때 0을 AND 게이트의 지배값(controlling value)이라고 한다. 이와 비슷하게 1은 OR 게이트의 지배값이다.

그림 3은 삼치 논리를 이용한 회로 분석 과정을 보여준다. 회로 분석 시작 시 모든 와이어(wire)는 X값이라고 가정한다[10]. 그리하여 회로 전체의 게이트와 와이어가 0 또는 1의 값으로 안정화 되면 그 회로는 순환 경로를 가지고 있더라고 회로상에서 문제를 일으키지 않는다[10,11].

3. 동기장치 중복사용

동기장치가 중복사용될 경우 발생할 수 있는 문제점은 2가지이다. 동기장치에 의해 생기는 회로상의 순환 경로(cyclic path)가 불안정(unstable)하거나, 한 클럭 안에 여러 번의 동기화를 제대로 처리하지 못하는 것이다. 이 장에서는 우리는 이 2가지 문제를 일으키지 않는 조건을 찾고자 한다.

3.1 예제

동기장치는 병렬문의 쓰레드들의 완료코드를 입력 받아 그 중 가장 큰 값을 병렬문 전체의 완료코드로 결정해 준다. 만약 하나의 쓰레드만이 완료코드를 입력한다면 동기장치는 그 값을 선택한다.

동기장치의 중복사용에서 나타나는 첫 번째 문제는 불안정한 순환 회로(unstable cyclic circuit)이다. 병렬문이 루프 안에 있을 경우 병렬문은 한 단위시간에 2번 이상 수행될 수 있다. 이 때 동기장치의 출력값이 루프의 GO와 K0를 지나서 자신의 입력값인 동기장치의 GO에 영향을 줄 수 있다. 이 순환 구조가 회로를 불안정하게 만들 수 있다. 그림 4는 루프 안에 있는 동기장치가 만들 수 있는 순환 구조를 보여 준다.

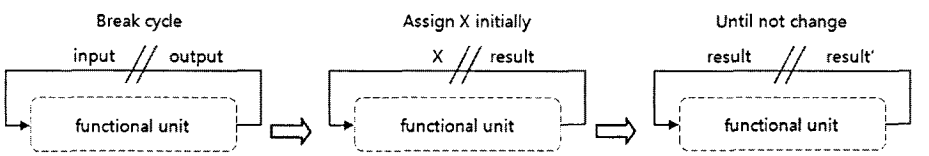

그림 3 삼치 논리를 이용한 회로 분석

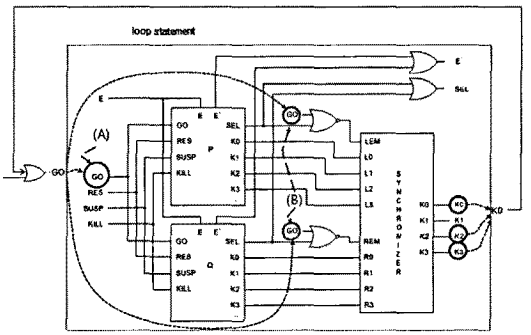

그림 4 동기장치 중복사용시의 순환 경로

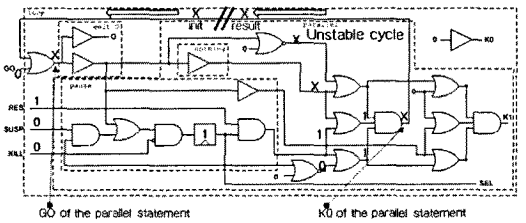

그림 5 "loop emit O; [pause||nothing] end"에 대한 회로

그림 5와 그림 6은 동기장치의 중복사용으로 인한 불안정한 순환회로의 예이다. 그림 5의 회로[3]는 하나의 쓰레드가 nothing이므로 루프가 끝날 때 그 쓰레드의 SEL 값이 0이다. 그래서 병렬문의 GO와 K0가 순환 의존성이 발생한다. 그리하여 그림 4에서의 (A)와 같은 불안정한 순환 경로가 생기고, 병렬문 GO의 상태를 결정할 수 없게 된다.

그림 6의 회로는 그림 5의 각 쓰레드에 pause문을 추가한 것으로, 중복사용된 동기장치에 의한 불안정한 회로의 또 다른 예이다. 여기서는 그림 4의 (A)와 같은 순환 경로는 안정화된다. 그러나 병렬문이 종료할 때 하나의 쓰레드가 이전에 종료되어 있어서, 그 쓰레드의

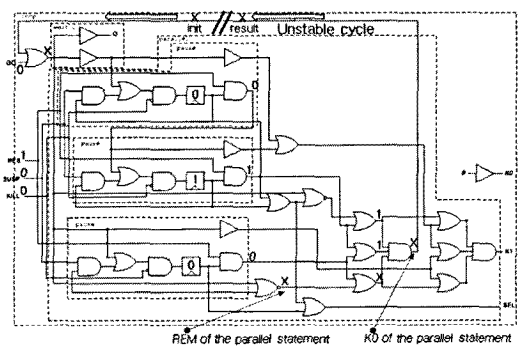

그림 6 "loop emit O; [pause;pause || pause] end"에 대한 회로

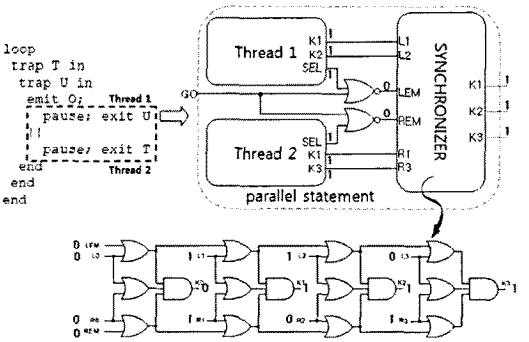

그림 7 동기장치 중복사용으로 인한 기능 오류 예

SEL의 값은 0이다. 그래서, REM과 K0의 순환 의존성(cyclic dependency) 때문에 그들의 값을 정할 수 없게 된다.

그림 7의 회로는 순환 경로가 존재하지만 회로상에서는 문제가 없다. 모든 쓰레드가 병렬문 시작할 때 종료하거나 예외를 발생시키지 않으므로, 그림 4의 (A)와 같은 경로는 레지스터 값에 의해 안정화된다. 게다가 병렬문이 종료할 때 이미 끝난 쓰레드가 없으므로 각 쓰레드의 SEL 값은 1이다. SEL이 LEM과 REM의 지배값이 되어 GO와 관계 없이 LEM과 REM의 값이 0으로 정해진다. 그러므로 그림 4의 (B)와 같은 불안정 순환 경로도 없다.

그림 7의 회로에서 순환 경로들은 비록 모두 안정화되어 있지만, 프로그램 원래의 의미는 올바르게 반영되지 않는다.

동기회로는 두 쓰레드의 완료코드를 받아 둘 중 큰 값을 출력한다. 이는 두 쓰레드가 서로 다른 예외를 발생시켰을 경우 병렬문 전체가 더 먼 예외 선언문에 해당하는 블록을 빠져나가게 하는 역할을 한다.

루프가 끝날 때는 쓰레드들이 각기 다른 예외를 발생시켜서 각각의 완료코드는 exit U에 해당하는 K2와 exit T에 해당하는 K3이다. 그러므로 루프가 끝날 때 동기장치는 K3을 출력해야 한다. 루프가 다시 시작하면, 모든 쓰레드가 그 때 pause를 수행하여 완료코드는 모두 K1이다. 이 때 동기장치는 K1을 출력해야 한다. 즉, 루프가 끝나고 즉시 다시 시작할 때 동기장치의 올바른 출력값은 K1과 K3이다. 하지만, 동시에 두 번의 동기화를 처리하면서 동기장치에 L1, L3, R1, R2가 동시에 입력되어 K1, K2, K3이 출력된다. 이는 그림 7에서 보듯이 두 번의 동기화를 동시에 처리하다가 동기장치의 구조에 따라 생기는 기능 오류이다.

3.2 중복사용 가능한 동기장치

앞에서 살펴본 예제들로부터 우리는 동기장치가 중복사용되더라도 회로상/기능상 문제를 일으키지 않는 조건을 제시한다.

정리 1.
다음 3가지 조건을 만족하는 병렬문은 한 단위시간에 중복사용 되더라도, 병렬문의 동기장치는 불안정한 순환 경로를 만들지 않고, 중복 수행되는 병렬문의 완료코드들만 출력한다.
(1) 병렬문이 시작하는 단위시간에 모든 쓰레드는 pause문을 수행한다.
(2) 병렬문이 종료할 때 모든 쓰레드의 SEL 값은 1이다.
(3) 쓰레드가 하나의 단위시간에 서로 다른 두 개 이상의 예외를 발생시키지 않는다.

증명. 여기서는 증명에 대한 아이디어를 제시하고자 한다.

먼저 회로상의 안정성을 알아보자. 병렬문이 한 단위시간에 2번 이상 수행될 경우 순환 경로는 그림 4의 (A)와 (B), 2가지가 가능하다. 조건 (1)에 의해 모든 쓰레드는 병렬문이 시작하는 단위시간에 pause를 수행하여 완료코드가 1이다. 즉, 병렬문의 GO가 각 쓰레드의 K0, Kn (n>1)에 영향을 미치지 않는다. 그러므로 루프의 GO는 (A)와 같은 경로로 불안정한 순환 경로를 만들지 않는다.

조건 (2)에 의해 병렬문이 끝나거나 예외를 발생시킬 때 각 쓰레드의 SEL 값은 1이다. SEL은 OR 게이트의 지배값이므로 동기장치의 입력인 LEM과 REM은 병렬문의 GO와 관계없이 0이 되어 K0, Kn (n>1)에 영향을 미치지 않는다. 즉, (B)와 같은 경로로도 불안정한 경로가 만들어지지 않는다. 그러므로 조건 (1), (2)에 의해 해당 회로는 본 논문에서 사용하는 회로 모델 상에서 모든 값을 결정할 수 있으므로, 회로상 안정하다.

여기서부터는 동기화 회로의 출력값에 대해 알아보겠다. 조건 (1)에 의해 각 쓰레드는 병렬문이 시작할 때 무조건 pause를 수행하여 완료코드값이 1이다. 또한 조건 (1)을 만족하는 병렬문이 한 단위시간에 중복사용 가능한 횟수는 많아야 두 번이다. 조건 (3)에 의해 병렬문이 두 번 중복사용될 경우 각 쓰레드와 동기장치의 모든 가능한 완료코드 값의 경우의 수는 다음과 같다(C는 예외 발생에 의한 완료코드 값을 뜻한다).

루프 종료할 때		루프 재시작할 때		동기화 결과
쓰레드1	쓰레드2	쓰레드1	쓰레드2	
L0	R0	L1	R1	K0,K1
L1	R1	L1	R1	K1
LC	R1	L1	R1	K1,KC
LC	RC	L1	R1	K1,KC

가능한 경우들을 고려할 때, 쓰레드1의 완료코드가 쓰레드2보다 크거나 같다고 가정하였다(반대 경우는 생략하였다).

모든 경우에 대해서 동기장치는 두 번의 동기화 결과들만을 출력하고 있다. 즉 해당 동기장치는 중복사용 하더라도 기능을 올바르게 수행한다. <증명 끝>

4. 탐지 알고리즘

앞에서 제시한 조건들을 검증하기 위해서는 쓰레드들이 수행되는 모든 경우에 대해 조사해야 한다. 이는 모델체커를 이용한 복잡한 분석 과정이 필요하므로, 분석이 시간과 메모리를 많이 필요로 한다.

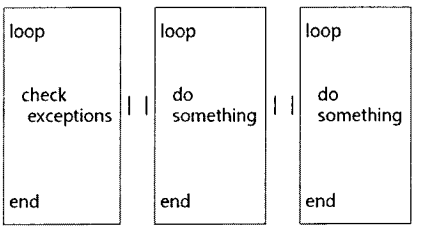

그림 8 병렬문의 프로그래밍 패턴

병렬문을 사용하여 시스템을 디자인하는 경우 대부분 그림 8과 같은 형식을 취한다. 동시에 수행할 일들을 각각의 쓰레드에 분배한 후 각 쓰레드는 해당 작업을 반복 수행한다. 이 때 하나의 쓰레드가 전체 일이 끝날 조건을 체크하는 방식이다.

우리는 실시간에 컴파일러가 사용할 수 있는 가벼우면서도 정확한 분석기 디자인을 위해 위의 조건들을 프로그래밍 패턴을 고려하여 다음과 같이 단순화(abstraction) 하였다.

(1′) 병렬문이 시작하는 단위시간에 모든 쓰레드는 pause 문을 수행한다.
(2′) 모든 쓰레드는 예외 발생이 없이는 끝나지 않는다.
(3′) 예외를 발생시킬 수 있는 쓰레드가 1개뿐이거나, 전체 쓰레드가 발생시킬 수 있는 예외가 1개뿐이다.

(1′)는 조건 (1)을 그대로 유지한다. (2′)는 그림 8과 같이 예외가 발생되지 않으면 각 쓰레드가 종료하지 않음을 의미한다. SEL은 각 쓰레드가 수행 중인지를 확인하는 신호이므로 병렬문 종료 시에도 모든 쓰레드의 SEL 값은 1이다. (3′)는 (3)의 조건을 약화시킨 조건이다. 앞에서 본 프로그래밍 패턴인 하나의 쓰레드가 예외 상황을 조절한다는 것을 반영하여 (3)의 조건을 약화 시킨 것이다.

우리는 이전 연구에서 Esterel에 대해 CFG를 생성하고[12], CFG 상에서 루프의 시작 영역(first-surface)과 끝 영역(last-surface)에 대한 정의 및 계산 알고리즘을 제시하였다[13]. 시작 영역은 루프가 시작하는 단위시간

에 CFG 상에서 수행될 수 있는 노드들의 집합을 말하며, 끝 영역은 루프 바디가 끝날 때 수행될 수 있는 노드들의 집합을 뜻한다. 이 알고리즘을 기반으로 한 그래프 탐색으로 위의 단순화된 조건들을 확인할 수 있다.

본 알고리즘을 이용하여 Esterel로 작성된 Ramesh의 사례연구[14] 중 DLX 프로세서 디자인에 동기회로 중복사용 문제가 있는지 찾아 보았다. 그 결과 중복수행 가능한 병렬문을 5개가 있어서 기존 연구에 따르면 5개 병렬문 모두 동기회로 중복사용 문제가 발생할 수 있다고 할 수 있다. 하지만, 해당 병렬문은 모두 그림 8에서의 프로그래밍 패턴을 따르고 있었다. 그러므로 5개 모두 동기장치 중복수행 문제를 발생시키지 않는다는 것을 알 수 있었고, 이는 우리의 알고리즘으로 확인 가능하였다.

5. 결론 및 향후 연구 과제

우리는 본 논문에서 동기장치가 중복 사용되더라도 정상 동작할 수 있는 조건을 찾고, 이를 바탕으로 중복사용 때문에 복제되어야 하는 동기장치를 찾는 알고리즘을 제시하였다.

우리의 알고리즘은 Esterel 소스 코드 상에서 동작하므로, 컴파일러가 회로 생성 이전에 그 결과를 알 수 있게 한다. 이는 복제되는 동기장치의 개수를 줄여 생성되는 회로의 크기를 줄이는데 도움을 준다. 또한, 회로 생성 이후의 검증보다 그 절차가 훨씬 간편하다.

본 논문을 바탕으로 앞으로 탐지 방법을 기반으로 한 구체적 알고리즘을 구현하고자 한다. 이전 연구[12,13]를 활용하여 구현하여 컴파일러에 적용할 것이다.

참 고 문 헌

[1] N. Halbwachs, Synchronous Programming of Reactive Systems, Kluwer Academic Publishers, 1993.

[2] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," Proceedings of the IEEE Embedded Systems, vol.91 (1), pp.64-83, 2003.

[3] G. Berry. The Constructive Semantics of Pure Esterel. Draft book available at http://www.inria.fr/meije/esterel/esterel-eng.html, 1999.

[4] D. Potop-Butucaru, S. A. Edwards, and G. Berry. Compiling Esterel. Springer, 2007.

[5] Esterel Technologies. The Esterel v7 Reference Manual Version v7.30. initial IEEE standardization proposal. Esterel Technologies, 679 av. Dr. J. Lefebvre 06270 VilleneuveLoubet, France, November, 2005.

[6] O. Tardieu and R. de Simone, "Loops in Esterel," Transactions on Embedded Computing Systems, vol.4, no.4, pp.708-750, 2005.

[7] K. Schneider, J. Brandt, and T. Schuele, "A verified compiler for synchronous programs with local declarations," *Electronic Notes in Theoretical Computer Science*, vol.153, no.4, pp.71-97, 2006.

[8] G. Berry, "Circuit design and verification with Esterel v7 and Esterel Studio," *IEEE International High-Level Design*, Validation, and Test Workshop, pp.133-136, 2007.

[9] M. Yoeli and S. Rinon, "Application of ternary algebra to the study of static hazards," *Journal of ACM*, vol.11, no.1, pp.84-97, 1964.

[10] S. Malik, "Analysis of cyclic combinational circuitsM" *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.13, no.7(7), pp.950-956, 1994.

[11] M. D. Riedel and J. Bruck, "Cyclic boolean circuits," *Journal of Discrete Applied Mathematics*, 2009.

[12] C. Kim, J. Yun, S. Seo, K. Choe, and T. Han, "Over-approximated Control Flow Graph Construction on Pure Esterel," *IEICE Transactions on Information and Systems*, vol.E93-D, no.5, May 2010 (accepted).

[13] J. Yun, C. Kim, S. Seo, T. Han, and K. Choe, "Refining schizophrenia via graph reachability in Esterel," *7th ACM-IEEE International Conference on Formal Methods and Models for Codesign*, 2009.

[14] S. Ramesh, Ramesh's homepage. http://www.cse.iitb.ac.in/~ramesh

윤 정 한
정보과학회논문지 : 소프트웨어 및 응용
제 37 권 제 4 호 참조

김 철 주
정보과학회논문지 : 소프트웨어 및 응용
제 37 권 제 4 호 참조

김 성 건
정보과학회논문지 : 소프트웨어 및 응용
제 37 권 제 4 호 참조

한 태 숙
정보과학회논문지 : 소프트웨어 및 응용
제 37 권 제 4 호 참조