임베디드 시스템에 대한 모델 기반의 명세 기법 및 정적 시간 분석 기술

박해우, 김진우, 하순회
서울대학교 전기컴퓨터공학부
{starlet, jwkim, sha}@iris.snu.ac.kr

Model-based specification and static timing-analysis of embedded systems

Hae-woo Park, Jinwoo Kim, Soonhoi Ha
School of EECS, Seoul National University, Korea

요 약

임베디드 시스템은 그 복잡도가 나날이 증가하고 있는데, 특히 기능성 외에 주어진 시간 제약 조건을 만족해야 한다는 점에서 개발 및 검증이 어렵다는 특징을 갖고 있다. 특히 검증에 대해서는 많은 경우에 적당한 시나리오들을 잡아 반복적인 시뮬레이션을 하는 방법을 사용하는데, 이 방법은 많은 시간이 걸리며, 적당한 시나리오들을 잡기 어렵다는 문제 또한 가지고 있다. 본 논문에서는 데이터플로우 모델과 유한상태기계 모델을 확장하여 시스템에 자원 사용 정보 및 시간 제약 조건을 명세하고, 이 모델들에 기반한 정적 시간 분석 방법을 제시하고 있다. 본 논문에서 제시한 방법을 통해 검증 시 자동으로 필요한 시나리오들에 대해 검증을 수행할 수 있으며, 시뮬레이션 등 긴 시간이 걸리는 검증 방법을 최소한으로 사용할 수 있다는 이점을 얻을 수 있다.

1. 서론

임베디드 시스템은 그 복잡도가 나날이 증가되고 있으나 시장에서 요구하는 개발 시간은 점차 짧아지고 있다. 그러나 짧은 시간 동안 복잡한 시스템을 개발하는 것에는 어려움이 있었고, 이에 따라 정형적인 모델에 기반을 둔 통합 설계 방법들이 연구되었다. 이는 데이터플로우(dataflow) 모델, 유한상태기계(FSM) 모델 등 다양한 모델들을 사용하여 시스템을 설계하고 이러한 모델들에 기반하여 분석을 수행하는 설계 방법이다. UC Berkeley의 Ptolemy[1], MathWorks社의 Simulink[2], 서울대학교의 PeaCE[3]와 같은 개발 도구들은 이러한 연구의 결과물들이다.

임베디드 시스템의 개발을 어렵게 하는 요소 중 하나는 임베디드 시스템이 올바른 기능성(functionality) 외에 상황에 따른 시간 제약 조건 역시 만족시켜야 한다는 점인데, 이는 어떤 입력에 대한 결과가 정확하면서도 주어진 시간 내에 나와야 한다는 것이다. 이러한 시간 제약 조건 만족 여부에 대한 검증은 기능성 검증과 달리 많은 시간이 걸리는데, 이는 모든 시나리오에 대해 실제로 프로파일링을 수행하는 것이 일반적이며 이러한 프로파일링이 보통 느리다고 알려져 있는 가상 프로토타입 환경에서 이루어지기 때문이다. 또한 모든 시나리오를 얻어 입력으로 넣어주는 것조차 어려운 경우가 많이 있다.

예를 들어 [그림 1]과 같이 동작하는 전형적인 임베디드 시스템을 생각해보자. 이 시스템은 외부에서 명령과 데이터를 입력으로 받아 시스템의 상태를 바꾸고 명령과 데이터에 따라 적절한 처리를 수행하는 일을 반복한다. 이들 중 어떤 입력에 대한 처리는 특정 시간 내에 완료되어야 할 것이며 상황에 따라 앞선 입력에 의해 처리 시간이 바뀔 수도 있다. 예를 들어, 앞선 입력에 대응하는 처리기가 DMA에 명령을 전송하고 완료되었다면 현재 입력에 대응하는 처리기는 DMA가 완료될 때까지 DMA에 명령을 전송하지 못하고 대기하여야 할 것이며, 정해진 시간 내에 처리를 완료하지 못할 가능성이 있다. 이러한 경우 모든 입력 시나리오에 대해 시뮬레이션을 수행하는 것은 매우 어렵고 시간이 많이 걸리는 일이 될 것이다.

```
while (true) {
    cmd = take_command();
    arg = take_argument();

    change_machine_status(cmd, arg);
    switch (cmd) {
        command_1 : { ... }
        command_2 : { ... }
            :
        command_n : { ... }
    }
}
```

그림 1. 전형적인 임베디드 시스템의 동작 예시

본 연구에서는 데이터플로우 모델과 유한상태기계

모델에 기반하여 설계된 시스템에 시간 제약 조건과 관련한 정보를 추가하는 방법을 제시하고, 추가된 정보를 기반으로 하여 문제가 발생할 수 있는 시나리오를 예측하여 장시간의 시뮬레이션을 반복하지 않고 시간 제약 조건 만족 여부를 얻어내는 방법을 제안하였다. 이는 정적인 분석을 통해 미리 문제를 발견하여 시뮬레이션 기반의 동적 분석 시간을 줄이는 데에 그 목표가 있다.

이후 2장에서는 관련 연구를, 3장에서는 시스템을 명세하는 방법과, 제안하는 시간 제약 조건 명세 방법을 다루며, 4장에서는 앞서 이루어진 명세를 바탕으로 정적 시간 분석을 수행하는 방법을 다룬다. 5장에서는 본 논문에서 제시한 방법을 사용한 실험을 다루며, 6장에서는 결론을 맺고자 한다.

2. 관련 연구

D. Kim, *et al.* [4] [5] 에서는 멀티미디어 응용 프로그램을 데이터플로우 모델과 확장된 유한상태기계 모델을 기반으로 하여 명세하고, 유한상태기계 모델을 기반으로 분석을 수행, 전체 시스템의 프로그램 코드를 생성하는 연구가 수행되었다. 본 논문에서 시스템을 명세하는 데 사용된 모델들은 이 논문들에서 제안한 모델들에 기반하고 있다.

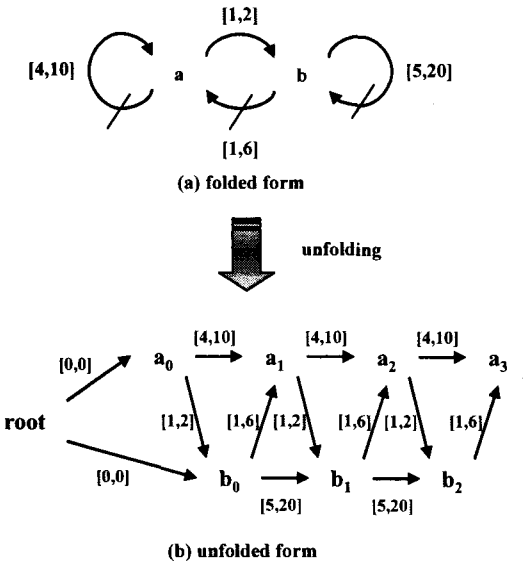

그림 2. 프로세스 그래프의 예 (from [7])

T. Amon, *et al.* [6] 과 H. Hulgaard, *et al.* [7] 에서는 동시성(concurrency)을 가지는 시스템을 프로세스 그래프(process graph)라고 부르는 유향 순환 그래프로 기술하고 이를 기반으로 시스템의 시간 분석을 수행하는 연구가 이루어졌다. 이 논문은 발생할 수 있는 이벤트들 사이의 시간 간격 범위를 기술하고 이를 바탕으로 분석을 수행하여 임의의 두 이벤트 사이의 시간 간격 범위를 구할 수 있게 하는 방법이다. 본 논문에서는 시스템의 명세와 시간 정보를 가지고 프로세스 그래프의 펼쳐진 형태로 새로운 그래프를 구성하여 이 논문에서 제안한 시간 분석 방법을 이용하여 시간 제약 조건 위반을 검사하고 있다.

[그림 2]는 프로세스 그래프의 예인데, (a)의 a, b는 이벤트들을 의미한다. 이벤트 사이의 화살표는 이벤트 간의 의존성을 의미하는데, b_k는 a_k 발생 후 1~2의 지연이 있은 후에 발생 가능하고, a_{k+1}는 b_k 발생 후 1~2의 지연이 있은 후에 발생 가능하다. 첨자의 차이는 화살표에 빗금으로 표시한다. (b)는 (a)를 펼친 형태로 각 이벤트의 시간 의존성을 첨자를 포함하여 나타내고 있다.

[6] [7] 에서는 여러 수식을 이용하여 이 펼쳐진 그래프에서 임의의 두 이벤트 사이에 발생할 수 있는 시간 간격의 최대값, 최소값을 구할 수 있는 알고리즘을 제공하며, 본 논문에서는 프로세스 그래프의 펼친 형태를 생성하고 이 논문에서 제시하는 알고리즘을 적용하였다.

3. 시스템 명세

3.1. 전체 시스템 명세

시스템은 전체적으로 데이터플로우 모델로 명세된다. 각 데이터플로우 블록은 입력에 대한 처리를 주로 수행하는 연산 블록과 입력에 대해 시스템의 상태를 변경하는 유한상태기계 블록 중 한 가지가 된다.

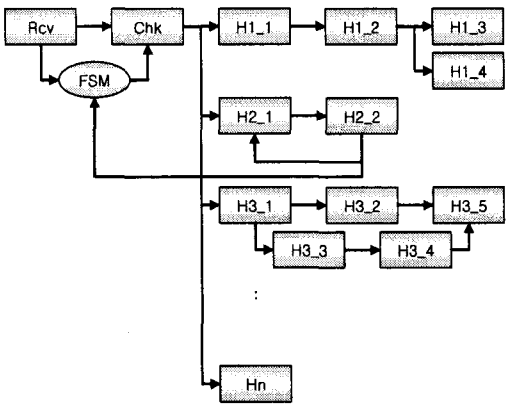

그림 3. 전체 시스템 명세의 예

[그림 3]은 [그림 1]의 일을 하는 시스템을 데이터플로우 모델에 기반하여 명세한 예인데, 사각형은 연산 블록을, 타원은 유한상태기계 블록을 각각

나타낸다.

예제를 간단히 설명하면, Rcv 블록은 입력을 받아 FSM 블록으로 하여금 시스템의 상태를 업데이트하도록 하고, Chk 블록은 입력 및 시스템 상태에 따라 처리를 수행할 블록에 데이터를 보내주어 처리를 하게 한다. H2_2 블록처럼 처리 이후 FSM 블록으로 하여금 시스템의 상태를 업데이트하도록 하는 경우도 있다. 이러한 형태는 외부 입력에 반응하는 많은 임베디드 시스템에서 흔히 볼 수 있는 구조이다.

3.2. 자원 사용 시간 명세

각 데이터플로우 블록에는 해당 블록이 사용하는 자원에 대한 정보를 넣어주게 된다. 여기에는 각 입력에 대한 한 번의 시뮬레이션을 통해 프로파일링된 정보를 기입하게 된다. 여러 입력의 나열인 시나리오가 아닌 각 입력 종류별 시뮬레이션으로 충분하기 때문에 여기에는 많은 시간이 걸리지 않는다.

각 블록이 사용하는 자원에 대해서는, 자원의 이름, 블록이 시작된 지 몇 사이클이 지나서 해당 자원이 사용되기 시작할 것인지의 최소/최대값, 해당 자원이 얼마 동안 사용될 것인지의 최소/최대값을 명세하게 된다.

예를 들어 한 블록이 CPU를 100~200사이클 사용하고, DMA를 해당 블록이 시작된 지 10~20사이클 이후에 50~150사이클 사용하고, CRC 처리기를 해당 블록이 시작된 지 20~30사이클 이후에 10~20사이클 동안 사용한다고 하면 이는 [그림 4]와 같은 형태로 명세될 것이다.

```
{cpu 0 0 100 200}
{dma 10 20 50 150}
{crc 20 30 10 20}
```

그림 4. 자원 사용 시간 명세의 예

이러한 자원 사용 명세는 모든 블록들에 대해 각각 이루어지게 된다.

3.3. 시간 제약 조건 명세

일반적으로 시스템이 특정 상태에 진입하게 되면 그 상태(state)에 해당하는 연산 및 처리를 수행하게 된다. 예를 들어 카메라 내의 임베디드 시스템이라면, 셔터가 눌렸을 때 시스템의 상태 전이가 일어나고 외부로부터 빛을 받아들여 저장하는 처리가 일어나게 될 것이다.

여기서는 이러한 점에 착안하여 시스템의 상태에 따라 처리하는 연산에 시간 제약 조건을 할당할 수 있도록 유한상태기계 모델을 확장하였다. 즉, 시간 제약 조건을 검사해야 할 상태에 대해 검사 시작 지점, 검사 완료 지점, 시간 제약 조건을 명세할 수 있게 하였다.

[그림 5]는 유한상태기계의 한 상태 S에 대해 시간 제약 조건을 명세한 예이다. 이 예에서 S에 명세된 내용은 연산 블록 'T1'의 '시작'부터 연산 블록 'T4'의 '끝'까지 64사이클 이내에 실행되어야 함을 나타내고 있다.

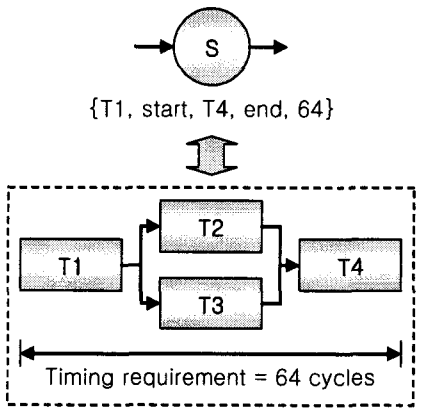

그림 5. 시간 제약 조건 명세의 예

4. 정적 시간 분석

4.1. 정적 시간 분석에서 자원 사용 고려의 필요성

1장에서 언급한 대로, 어떤 블록은 단독으로 실행되었을 때는 시간 제약 조건을 만족하지만, 앞서 다른 블록이 처리된 상황에서는 시간 제약 조건을 어기게 될 수 있다.

[그림 6]는 이러한 경우의 예이다. 그림에서 (a)와 (b)는 두 종류의 블록이 각각 실행되는 경우인데, 두 블록 모두 CPU가 입력 이벤트를 받아 처리를 수행하다가 DMA에 명령을 보내고 CRC 처리기에 명령을 보낸 다음 외부에 완료 이벤트를 보내는 방식이다. 두 블록 모두 정해진 데드라인(d1, d2) 이전에 완료 이벤트를 발생하므로 문제가 없다.

[그림 6] (c)의 경우는 (a)와 (b)가 연속적으로 처리되는 경우이다. (a)의 블록은 주어진 시간 제약 조건을 만족하지만, (b)의 블록은 DMA와 CRC가 (a)의 블록에 의해 선점되어 있어서 CPU가 DMA 및 CRC에 명령을 전송하는 것이 늦어지고 따라서 시간 제약 조건(d2)을 위반하게 되는 것을 볼 수 있다.

따라서 (b)의 블록의 경우 시간 분석 시 (a)의 블록이 앞서 처리되는 경우가 발생할 수 있는지 확인하여야 한다. 본 논문에서는 이러한 경우를 시스템의 유한 상태 기계 블록을 검사하여 찾아내게 된다.

4.2. 연산 블록 별 그래프 조각 생성

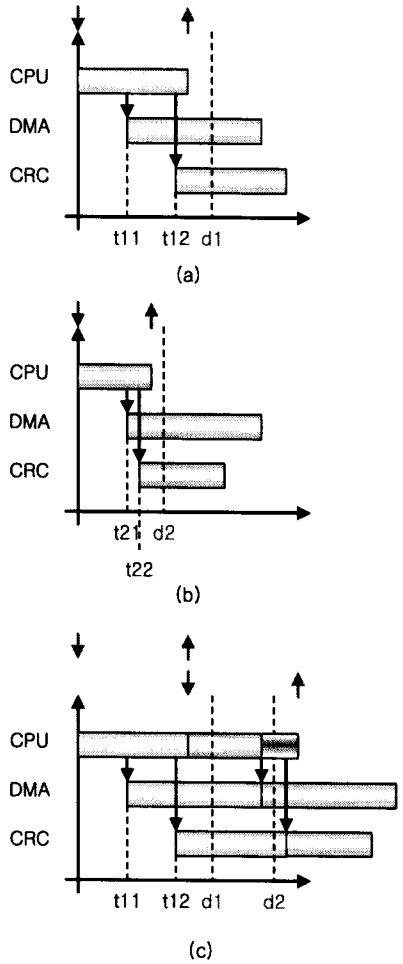

그림 6. 여러 블록의 연속적인 실행에서 자원 점유로 인해 발생하는 시간 제약 조건 위반의 예

프로세스 그래프 분석 방법에 기반한 정적 시간 분석을 위해서는 먼저 연산 블록들의 자원 사용 명세로부터 프로세스 그래프를 얻어야 하는데, 여기서는 먼저 각 연산 블록마다 프로세스 그래프의 조각을 얻어내게 된다.

CPU는 주어진 자원을 동작시키는 역할을 하며, CPU가 다른 자원을 동작할 수 없는 상황인 경우 CPU의 동작 시간은 더 길어질 수 있다. 이러한 조건들은 CPU와 다른 자원들 사이의 의존성으로 표현할 수 있으며 이는 프로세스 그래프의 특징과 일치한다.

상기 조건들을 고려하여 [그림 4]에 주어진 자원 사용 명세를 펼쳐진 형태의 프로세스 그래프로 표현하면 [그림 7]과 같이 된다.

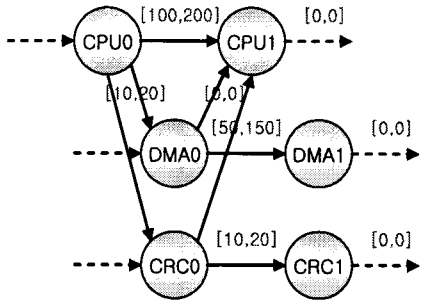

그림 7. [그림 4]에서 생성된 프로세스 그래프 조각

이는 자원의 수가 많아져도 같은 형태로 생성할 수 있는데, (CPU0 → 자원0), (CPU0 → CPU1), (자원0 → CPU1), (자원0 → 자원1) 의 의존성 정보를 유지하면서 생성하면 된다.

4.3. 시간 제약 조건으로부터의 프로세스 그래프 조립과 시간 분석

유한상태기계 블록에 기입된 시간 제약 조건에는 시작 블록과 끝 블록만 나타나 있으므로, 각 블록들의 수행 순서를 분석하여 시작 블록과 끝 블록 사이에 실행되어야 할 블록들을 얻어내어야 한다. 이 문제는 블록들 사이의 분기나 피드백 등에 의한 루프가 있을 경우 복잡도가 높은 문제인데, R. W. Floyd [9] 의 알고리즘을 수정하여 블록들 사이의 의존성 정보를 얻고 변형된 넓이 우선 탐색을 적용하여 찾을 수 있다. 예로, [그림 5]의 경우 그래프 탐색을 통해 T1→T2→T3→T4 의 블록 실행 순서를 얻어낼 수 있다. 이 그래프 탐색 방법은 본 논문의 범위를 벗어나므로 자세하게 다루지는 않는다.

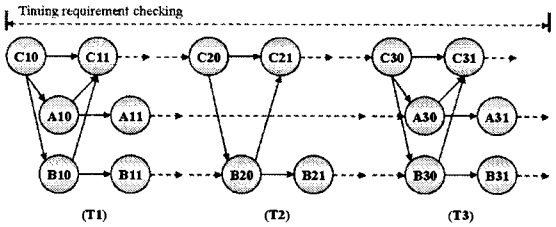

그림 8. 프로세스 그래프 조립

블록 실행 순서를 얻어내고 나면, 실행 순서에 따라 각 블록의 프로세스 그래프 조각을 연결하게 되는데, 만일 T1→T2→T3의 블록 실행 순서가 있다고 하면 [그림 8]과 같은 펼쳐진 형태의 프로세스 그래프를 얻어낼 수 있다.

이후 이 그래프에 [6] [7]의 시간 분석 방법을 적용하면 시간 제약 조건 만족 여부를 파악할 수 있다.

4.4. 연속되는 외부 이벤트에 대한 시간 분석

3.3에서 언급하였듯이 외부 이벤트가 연속적으로 들어오는 경우, 단일 이벤트에 대한 시간 분석 외에 앞선 이벤트에 대한 고려가 필요한 경우가 있다. 이러한 경우에 대해서는 각 이벤트에 대한 프로세스 그래프를 결합하여 시간 제약 조건을 분석할 수 있다.

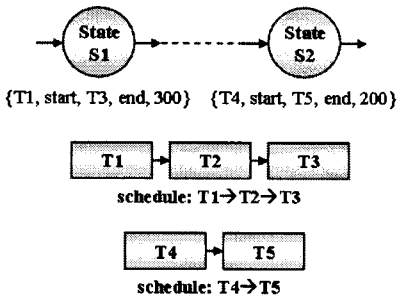

그림 9. 유한상태기계 블록에 기술된 연속된 이벤트 및 연관된 연산 블록들의 예

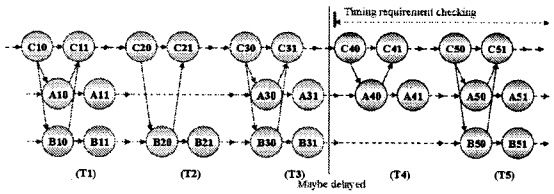

그림 10. [그림 9]에서 생성된 프로세스 그래프

[그림 9]는 연속될 수 있는 두 이벤트에 대응하는 유한상태기계 블록 내의 두 상태와 그와 연관된 연산 블록들을 나타낸 것이다. 각 이벤트에 대해서는 앞 절에서 설명한 방법으로 프로세스 그래프가 생겨났으며, 두 이벤트의 연속된 처리를 위해서는 [그림 10]과 같은 프로세스 그래프로 병합한 다음 앞의 이벤트 처리가 완료된 다음 뒤의 이벤트 처리가 얼마나 지연되는지 분석하게 된다. 여기에는 앞 절에서 언급한 것과 동일한 알고리즘을 적용한다.

5. 실험

5.1. 실험 환경

본 논문에서 제안한 분석 방법은 서울대학교에서 개발된 PeaCE[3] 개발 도구의 확장을 통해 구현하였다. 데이터플로우 모델로는 BP 모델을, 유한상태기계 모델로는 fFSM 모델을 사용하였다.

개발에 사용한 컴퓨터는 Dual-Xeon 3GHz를 CPU로 사용하고 4GB의 메모리를 장착한 서버이다.

5.2. 실험 예제

실험 예제로는 3.1에서 다룬 형태의 시스템을 사용하였다. [그림 11]은 데이터플로우 모델로 기술된 시스템의 전체 구조를 나타낸 것이며, [그림 12]는 시스템의 상태 전이를 나타낸 유한상태기계이다. [표 1]은 데이터플로우 모델의 각 블록에 기술된 자원 사용 명세와 유한상태기계에 명세된 시간 제약 조건을 각각 나타낸 것이다.

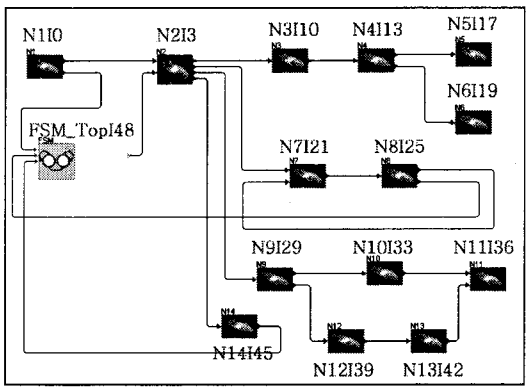

그림 11. 데이터플로우 모델로 기술된 실험 예제

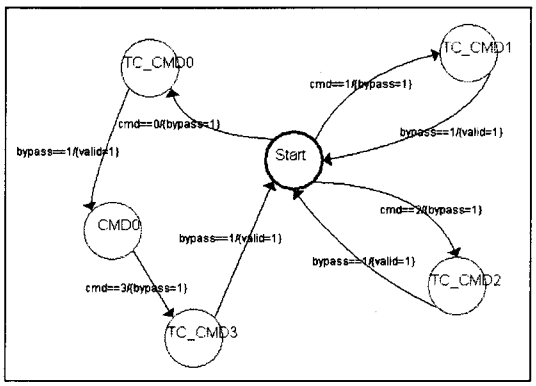

그림 12. 유한상태기계 모델로 기술된 시스템 상태 전이도

예제를 간략히 설명하면, 시스템은 외부에서 이벤트를 받아서 상태를 전이하고 관련된 블록들을 실행하는 형태이다. 유한 상태 기계에는 초기 상태에서는 명령 0, 1, 2를 받을 수 있고 명령 0을 받은 후에는 명령 3만 받을 수 있다는 것이 표현되어 있다. 명령 0은 N3로 시작하는 블록 나열을, 명령 1은 N7로 시작하는 블록 나열을, 명령 2는 N9로 시작하는 블록 나열을, 그리고 명령 3은 블록 N14를 수행하게끔 되어 있다.

본 실험 예제를 작성한 후 검사 도구를 실행하면

[그림 13]의 결과가 나온다.

표 1. 실험 예제의 자원 사용 및 시간 제약 조건 명세

연산 블록	자원 사용 명세
N1	{cpu 0 0 50 100}
N2	{cpu 0 0 50 50}
N3	{cpu 0 0 100 100} {r1 0 50 100 150}
N4	{cpu 0 0 100 100} {r1 0 50 100 100} {r2 25 25 50 75}
N5	{cpu 0 0 100 100} {r1 0 0 50 50} {r2 25 25 50 75}
N6	{cpu 0 0 100 100} {r2 50 50 125 150}
N7	{cpu 0 0 100 100} {r1 25 50 25 75}

연산 블록	자원 사용 명세
N8	{cpu 0 0 100 150}
N9	{cpu 0 0 100 100} {r1 50 50 200 200}
N10	{cpu 0 0 100 100} {r2 50 50 25 75}
N11	{cpu 0 0 100 100} {r1 50 100 50 100}
N12	{cpu 0 0 100 100} {r1 50 50 100 150}
N13	{cpu 0 0 100 100} {r2 50 75 50 75}
N14	{cpu 0 0 100 100} {r1 0 50 100 150} {r2 50 75 25 50}

상태	시간 제약 조건
TC_CMD0	{n3 start n5 end 300}
TC_CMD1	{n7 start n8 end 200}

상태	시간 제약 조건
TC_CMD2	{n9 start n11 end 500}
TC_CMD3	{n14 start n14 end 100}

```
(+) Performing timing analysis
-- Task lists for state TC_CMD1: N7I21 N8I25
-- Task lists for state TC_CMD2: N9I29 N10I33 N12I39 N13I42 N11I36
-- Task lists for state TC_CMD0: N3I10 N4I13 N5I17
-- Task lists for state TC_CMD3: N14I45
*. Check for consecutive timing requirement specifications.
( 1)TC_CMD1         -> (1)TC_CMD1 (2)TC_CMD2 (3)TC_CMD0
( 2)TC_CMD2         -> (1)TC_CMD1 (2)TC_CMD2 (3)TC_CMD0
( 3)TC_CMD0         -> (5)TC_CMD3
( 5)TC_CMD3         -> (1)TC_CMD1 (2)TC_CMD2 (3)TC_CMD0
*. WARNING: TIMING VIOLATION MAY BE OCCURRED.
    State          : TC_CMD1
    Tasks          : N7I21 N8I25
    Requirement    : 200
    Calculated     : 250
*. WARNING: TIMING VIOLATION MAY BE OCCURRED.
    State          : TC_CMD0
    Previous state : TC_CMD2
    Tasks          : N9I29 N10I33 N12I39 N13I42 N11I36 N3I10 N4I13 N5I17
    Requirement    : 300
    Calculated     : 350
*. WARNING: TIMING VIOLATION MAY BE OCCURRED.
    State          : TC_CMD0
    Previous state : TC_CMD3
    Tasks          : N14I45 N3I10 N4I13 N5I17
    Requirement    : 300
    Calculated     : 350
```

그림 13. 실험 예제의 검사 결과

결과를 살펴보면 각 TC_CMDx 상태에서 각각 실행되어야 하는 블록들의 목록이 기대한 바대로 나오고 있으며 루프 및 피드백이 있는 복잡한 연결 구조에서도 반드시 들어가야 할 블록들만 들어가 있는 것을 확인할 수 있다.

또한 연속적으로 선택될 수 있는 명령, 즉 명령 1 이후에는 명령 1, 2, 00I, 명령 2 이후에는 명령 1, 2, 00I, 명령 0 이후에는 명령 30I, 그리고 명령 3 이후에는 명령 1, 2, 00I 올 수 있음을 보여주고 있다.

그리고 명령 1에 의해 실행되는 블록들이 시간 제약 조건을 어기고 있다는 것, 그리고 명령 00I 명령 2 혹은 명령 3 이후에 들어오면 시간 제약 조건을 어길 수 있다는 것을 결과로 보이고 있다. 직접 손으로 블록들을 나열해보면 모두 정확한 결과이며 다른 경우들은 문제가 없다는 것을 확인할 수 있다.

또한 이 분석은 1초 미만의 시간이 걸려, 한 번 시뮬레이션을 수행하여 자원 사용 정보를 얻기만 하면 그 후로는 적은 시간을 들일 수 있음을 확인할 수 있었다.

6. 결론

본 논문에서는 자원 사용 및 시간 제약 조건에 대한 명세를 할 수 있도록 데이터플로우 모델과 유한상태기계 모델을 확장하였다. 또한 제시된 정형적 명세를 바탕으로 하여 임베디드 시스템에서 다루기 어려운 문제인 시간 제약 조건 만족 여부를 검사할 수 있는 정적인 시간 분석 방법을 제시하였다.

제시된 정적 시간 분석 방법을 사용하면 미리 문제가 발생할 수 있는 상황을 감지할 수 있고, 시뮬레이션과 같이 시간이 오래 걸리는 검증을 최소한으로 사용할 수 있다는 이점을 얻을 수 있다.

■ 감사의 글

본 연구는 BK21 프로젝트, 과학기술부 도약연구지원사업(R17-2007-086-01001-0), 삼성전자에 의해 지원되었다. 또한 서울대학교 컴퓨터신기술연구소와 IDEC은 본 연구에 필요한 기자재들을 지원해 주었다. 본 연구는 또한 한국전자통신연구원의 SoC 핵심설계인력양성사업에 의해 부분적으로 지원되었다.

■ 참고 문헌 및 자료

[1] http://ptolemy.eecs.berkeley.edu
[2] http://www.mathworks.com/products/simulink
[3] http://peace.snu.ac.kr/research/peace
[4] D. Kim and S. Ha, "System level specification for multimedia applications," APCHDL-99, 1999.
[5] D. Kim and S. Ha, "Static Analysis and Automatic Code Synthesis of flexible FSM Model," ASP-DAC 2005, 2005.
[6] T. Amon, H. Hulgaard, S. M. Burns, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," In IEEE Int. Conf. on Comp. Design, pp.166-173, 1993.
[7] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "Practical applications of an efficient time separation of events algorithm," Proc. in Int. Conf. on Computer-Aided Design, pp.146-151, 1993.
[8] Edward A. Lee and David G. Messerschmitt, " Synchronous Data Flow," Proceedings of the IEEE, vol. 75, no. 9, September, 1987.
[9] Robert W. Floyd, "Algorithm 97: Shortest path," Communication of the ACM, vol.5, no.6, 1962.