http://dx.doi.org/10.3745/KIPSTA.2012.19A.1.023

오토마타 이론을 적용한 Simulink 실행 모듈 구현

김 경 준[†] · 정 기 현[††] · 최 경 희[†††]

요 약

본 연구에서는 오토마타 이론을 적용하여 Simulink 모델의 각종 정보를 보여주고 시뮬레이션 결과를 출력하는 모듈 구조를 제안하고, 이를 구현하였다. 이 모듈은 사용자로부터 명령을 받아서 Simulink 모델의 각종 내부 정보와 시뮬레이션 결과를 보여주는 기능을 제공하는 메인 프로그램과 Simulink 모델의 시뮬레이션을 제어하기 위한 별도의 스레드, 그리고 실제로 시뮬레이션을 수행하는 Simulink 모델의 세 가지 독립적인 시스템으로 이루어져 있다. 오토마타를 이용하여 각 시스템과 전체 모듈을 설계 및 검증하고, C# 과 MATLAB을 통해 이를 구현한다. 그리고 구현된 모듈의 동작을 실제 모델을 통한 실험으로 검증하였다.

키워드 : 시뮬링크 실행 모듈, 오토마타, 시뮬링크 모델 설계 테스트, 시뮬링크 모델 디버그

Implementation of the Simulink Execution Module by Applying Automata Theory

Kyungjoon Kim[†] · Kihyun Chung[††] · Kyunghee Choi[†††]

ABSTRACT

This paper suggests that implementation of the Simulink execution module controls Simulink simulation and shows simulation results by applying Automata theory. This module is composed of three independent systems that the main program to accept user commands, the thread to control a simulation of Simulink model and the Simulink model to execute simulation. This paper designs each module and entire system by applying Automata theory, and implements it with C# and MATLAB language. And the Simulink execution module implemented will be verified through the experiment.

Keywords : Simulink Execution Module, Automata, Simulink Model Design Test, Simulink Model Debug

1. 서 론

Simulink는 MATLAB의 확장자로 동적 시스템의 시뮬레이션 및 모델 기반 설계를 위한 플랫폼이다[1]. 미리 정의된 방대한 블록 세트를 가지고 있어 블록 다이어그램 형태의 설계가 가능하고, 모델링에 있어 높은 추상화를 제공한다. 이러한 장점들로 인해 Simulink는 시스템 모델링 분야에서 널리 사용되고 있다. 최근에는 알고리즘 검증 단계에서의 자원을 그대로 하드웨어 설계 및 소프트웨어 설계에서 사용하고 최종적으로 검증에까지 사용하는 추세이다[2]. Simulink 모델을 이용한 검증작업이 많아짐에 따라 이를 기반으로 하는 테스트 작업의 자동화도 요구되고 있다.

† 준 회 원 : 아주대학교 전자공학과 공학석사
†† 정 회 원 : 아주대학교 전자공학부 교수
††† 정 회 원 : 아주대학교 정보통신전문대학원 교수

논문접수 : 2010년 11월 24일
수 정 일 : 1차 2011년 5월 17일
심사완료 : 2011년 6월 15일

본 연구진에서 개발한 R-bench의 TSG(Test Script Generator)는 요구 사항 혹은 Simulink/Stateflow 기반으로 자동 테스트 케이스를 생성하는 기능을 제공한다[3]. 본 논문에서 제안하는 모듈은 이 중 Simulink/Stateflow 기반으로 생성된 테스트 케이스를 사용하여 Simulink 모델의 동작을 검증하는데 매우 유용하게 사용될 수 있다. 모델 설계자는 이 모듈을 이용하여 생성된 테스트 케이스로 Simulink 모델을 실제로 실행시켜서 돌아가는 모습을 눈으로 확인해 볼 수 있으며 이 과정에서 Simulink 모델의 시뮬레이션 명령과 이 모듈의 시뮬레이션 명령, 양쪽에 자유롭게 접근하면서 모델을 디버깅하는 것이 가능하다.

본 논문의 2장에서는 구현하려는 Simulink 실행 모듈의 기능과 구조를 정리하고, 오토마타를 통한 모듈 내부의 각 시스템의 설계 및 전체 모듈의 검증 과정을 기술한다. 3장에서는 2장에서 정리하였던 모듈의 기능들과 오토마타를 반영하여 실제 모듈을 구현하는 방법을 설명하고, 4장에서는 구현된 모듈을 2장에서 검증한 전체 시스템 오토마타의 상

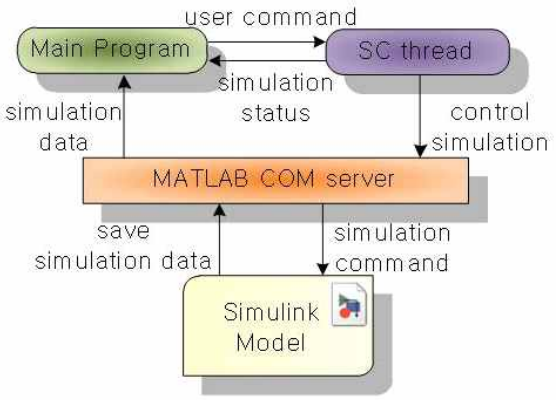
태와 이벤트들을 기준으로 동작을 확인한 내용을 기술할 예정이다.

2. Simulink 실행 모듈 구조

본 연구에서 구현하려는 모듈은 다음과 같은 기능 들을 제공할 것이다.

(1) TSG에서 생성된 테스트 케이스를 입력으로 Simulink 모델의 시뮬레이션
(2) 시뮬레이션 시 Simulink 모델의 출력 정보
(3) 시뮬레이션 시 Stateflow 내부 데이터 정보
(4) 시뮬레이션 시 원하는 시점에서의 잠시 멈춤 및 재시작
(5) Main Program의 GUI 명령 버튼과 Simulink 모델의 명령 버튼을 통한 시뮬레이션 제어

이를 위하여 (그림 1)과 같이 사용자의 명령을 받고 여러 정보를 출력해주는 메인 프로그램과 Simulink 모델의 시뮬레이션의 제어를 위한 별도의 Simulation Control 스레드(이하 SC스레드), 시뮬레이션을 수행하는 Simulink 모델의 3가지 병행적인 모듈 구조를 제안한다. 사용자는 Main 프로그램에서 제공하는 UI를 통해 Simulink 모델을 시뮬레이션 할 수 있고, 프로그램 내부적으로SC 스레드를 통해 Simulink 모델을 제어하며 모델의 변화를 감지한다. 필요하다면 직접 Simulink 모델의 시뮬레이션 명령도 사용할 수 있다. 이러



(그림 1) Simulink 실행 모듈

한 병행 프로그램의 설계 및 구현에는 여러 가지 어려운 점들이 있는데, 오토마타 이론을 적용할 경우 적절한 설계 및 검증이 가능해진다[4][5].

본 논문에서는 위 3가지 시스템에 대해 오토마타를 이용하여 설계하고, 전체 시스템을 검증한다. 이 모듈이 제공하려는 모든 기능을 오토마타를 이용하여 표현할 경우 전체 시스템이 매우 복잡하고 해석하기 부담스러운 모델이 된다. 또한, 시뮬레이션 제어를 제외한 기능들은 굳이 오토마타를 이용하여 검증할 필요가 없다. 따라서 3가지 시스템의 상태가 복합적으로 고려되어야 하는 시뮬레이션 제어 기능을 위주로 오토마타를 이용한 설계를 할 것이다. 그 외의 기능들에 대해서는 3장Simulink 실행 모듈 구현 부분에서 추가적으로 설명하도록 한다.

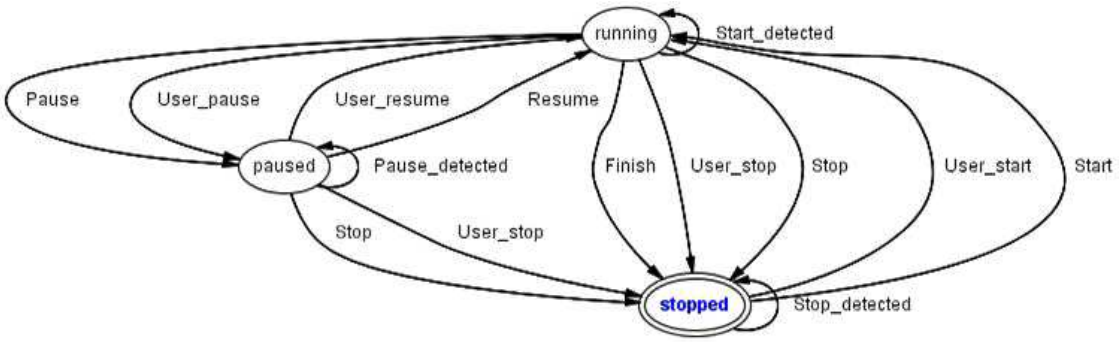
각 시스템의 오토마타 표현과 전체 시스템 Parallel Composition에는 DESUMA라는 프로그램을 사용하였다[6]. DESUMA는 미시간 대학(University of Michigan)에서 개발된 UMDES library와 마운트 앨리스 대학(Mount Allison University)에서 개발된 GIDDES를 통합한 프로그램으로 유한 상태 오토마타를 이용한 이산 사건 시스템의 모델링 시 모델 구축, 고장 진단, 검증, 분산형 제어 등에 활용할 수 있다[7].

2.1 Simulink 모델 오토마타

Simulink 모델은 실제로 stopped, updating, initializing, running, paused, terminating, external 총 7가지의 상태를 가지고 있다[8]. 하지만 Simulink 모델의 시뮬레이션 제어 시 의미를 갖는 것은 stopped, running, paused 3가지 상태뿐이다. 따라서 Simulink 모델의 오토마타는 이 3가지 상태만 정의한다. Simulink 모델의 오토마타 표현은 (그림 2)에 나타내었다.

〈표 1〉 Simulink 모델 오토마타 상태

상 태	설 명
stopped	Simulink 모델의 시뮬레이션이 정지된 상태
running	Simulink 모델의 시뮬레이션이 실행중인 상태
paused	Simulink 모델의 시뮬레이션이 잠시 멈춤 된 상태



(그림 2) Simulink 모델 오토마타

〈표 2〉 Simulink 모델 오토마타 이벤트

이벤트	설 명
Start	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션 시작 명령이 모델에 전달한다.
Stop	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션 중단 명령이 모델에 전달된다.
Pause	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션이 잠시 멈춤 명령이 모델에 전달된다.
Resume	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션이 재시작 명령이 모델에 전달된다.
Finish	모델에 설정된 End time까지 시뮬레이션이 실행된 후 종료된다.
User_start	사용자가 Simulink 모델의 버튼을 통해 시뮬레이션을 시작한다.
User_stop	사용자가 Simulink 모델의 버튼을 통해 시뮬레이션을 중단한다.
User_pause	사용자가 Simulink 모델의 버튼을 통해 시뮬레이션을 잠시 멈춘다.
User_resume	사용자가 Simulink 모델의 버튼을 통해 시뮬레이션을 재시작한다.
Stop_detected	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션이 종료됨을 감지한다.
Start_detected	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션이 시작되어 동작 중임을 감지한다.
Pause_detected	SC 스레드가 MATLAB 함수를 사용하여 시뮬레이션이 잠시 멈춤을 감지한다.

사용자는 메인 프로그램의 버튼을 통해 Simulink에 시뮬레이션 명령을 내릴 수 있다. 이렇게 전달된 명령은SC 스레드에 의해 Simulink 모델에 전달된다. 사용자는 Simulink 모델에서 직접 시뮬레이션 시작, 중단 등의 명령을 내려 시뮬레이션을 실행시킬 수도 있다. <표 2>의 Start, Stop, Pause, Resume이 전자를 표현한 이벤트이고, User_start, User_stop, User_pause, User_resume이 후자를 표현한 이벤트이다. 또한, Simulink 모델에 정의되어 있는 End time까지 시뮬레이션 실행 후에는 시뮬레이션이 자동 종료되어 stopped 상태로 가게 되는데 이러한 이벤트를 Finish로 정의해 두었다. Finish 이벤트에 의한 stopped 상태로의 변화를 파악할 수 있도록 Stop_detected 이벤트를 stopped상태에서만 발생 가능하도록 정의하였고, 이는 User_stop 이벤트에 의한 상태 변화에도 대응할 수 있다. 구현하려는 모듈은 시뮬레이션 명령을 메인 프로그램의 버튼과 Simulink 모델의 버튼으로부터 동시에 받아들여 적절한 동작을 하려고 하기 때문에 User_start, User_resume 을 위한 Start_detected 이벤트와 User_pause를 위한 Pause_detected 이벤트도 각각 running 상태와 paused 상태에서만 발생 가능하도록 정의한다.

2.2 메인 프로그램 오토마타

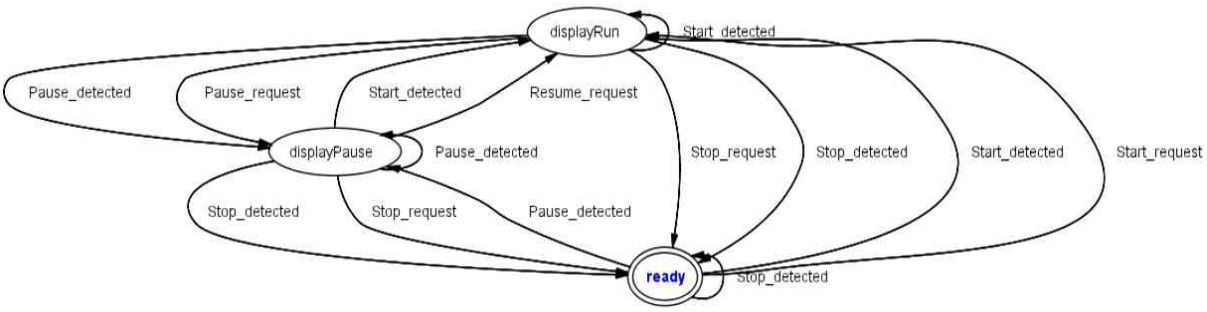
메인 프로그램은 버튼을 통해 사용자로부터 시뮬레이션 명령을 받으며, 테스트 케이스를 입력으로 Simulink 모델을 시뮬레이션하는 동안 그 결과를 화면에 보여준다. 시뮬레이션 제어와 관련된 메인 프로그램의 상태는 크게 ready, displayRun, displayPause의 3가지 상태로 정의할 수 있다. 다음 (그림 3)는 메인 프로그램의 오토마타 표현이다. 이 오토마타에는 <표 4>에 나타낸 이벤트 외에 Stop_detected, Start_detected, Pause_detected 이벤트가 정의되어 있는데, 이에 대한 설명은 Simulink 모델 오토마타와 중복되기 때문에 표에서는 생략하였다.

〈표 3〉 메인 프로그램 오토마타 상태

상 태	설 명
ready	시뮬레이션 결과 정보를 화면에 출력하고 있지 않는 상태
displayRun	시뮬레이션이 수행되는 동안의 결과 정보를 화면에 출력하고 있는 상태
displayPause	시뮬레이션이 잠시 멈춤된 상태의 결과 정보를 화면에 출력하고 있는 상태

〈표 4〉 메인 프로그램 오토마타 이벤트

이벤트	설 명
Start_request	메인 프로그램의 버튼을 통해 시뮬레이션 시작 명령이 입력된다.
Stop_request	메인 프로그램의 버튼을 통해 시뮬레이션 시작 명령이 중단된다.
Pause_request	메인 프로그램의 버튼을 통해 시뮬레이션 시작 명령이 잠시 멈춤 된다.



(그림 3) Main Automata

메인 프로그램은 사용자로부터 시뮬레이션과 관련된 명령을 받으며, <표 4>에 이러한 이벤트들이 정의되어 있는 것을 볼 수 있다. 또한, Simulink 모델의 상태의 변화를 감지하여 화면에 출력하는 내용이 변화하여야 하기 때문에 Stop_detected, Start_detected, Pause_detected 이벤트의 정의도 필요하다.

displayRun 상태에서는 현재 시뮬레이션 실행 중인 모델의 데이터 정보들을 화면에 출력할 것이다. 이를 위해서 현재 시뮬레이션 중인 Simulink 모델의 출력 정보, Stateflow 내부 데이터 정보, 시뮬레이션 시간 정보 등을 필요로 한다. 이러한 정보들을 Simulink 로 부터 얻어오는 방법은 3장의 구현에서 설명한다.

2.3 SC 스레드 오토마타

SC 스레드는 메인 프로그램으로부터 입력 받은 명령에 대해 Simulink 모델의 시뮬레이션을 실행시키고, 실행 결과를 확인하여 Simulink 모델과 메인 프로그램이 동기되어 움직일 수 있도록 하는 역할을 한다. 초기 상태인 scInit상태와 각각의 명령을 수행하고, 실제 모델의 시뮬레이션 상태를 확인하는 scStopBeforeRun, scTryRun, scTryStop, scTryResume, scTryPause 총 6가지 상태를 정의하였다.

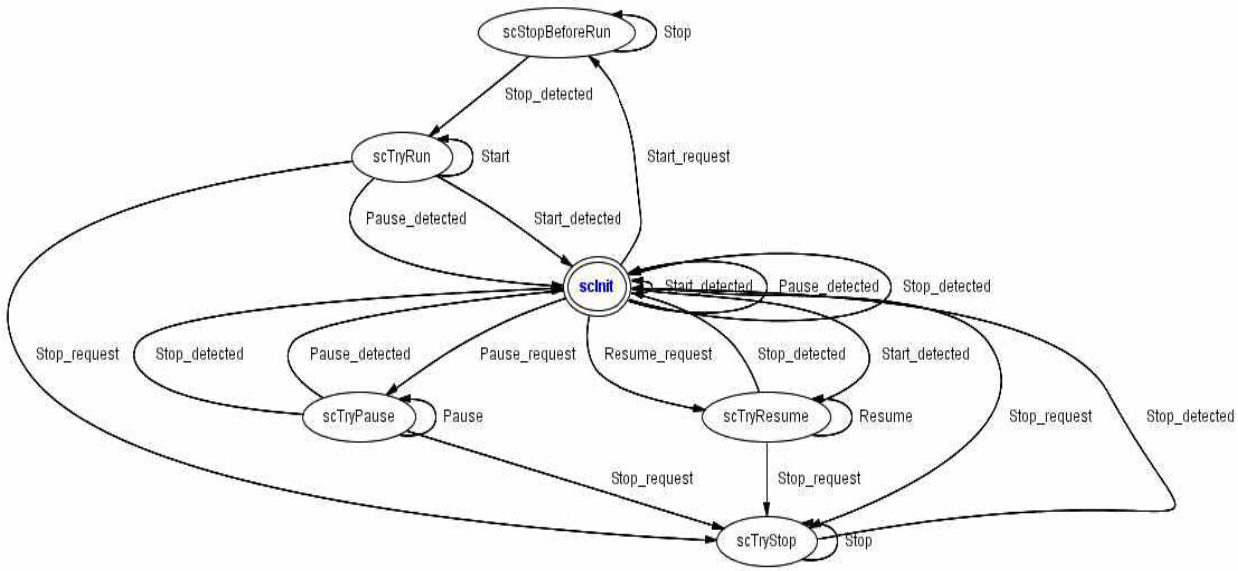
SC 스레드 오토마타에는 Simulink 모델 오토마타에서 정의되어 있던 Start, Stop, Pause, Resume와 메인 프로그램 오토마타에서 정의되어 있던 Start_request, Stop_request, Pause_request, 그리고 Simulink 모델 오토마타의 상태 확인을 위한 Start_detected, Stop_detected, Pause_detected 이벤트가 정의되어 있다. 각 이벤트에 대한 설명은 <표 2>, <표 4>와 동일하므로 생략한다.

Simulink 모델 오토마타에서 정의한 이벤트들과 메인 프로그램 오토마타에서 정의한 이벤트가 동시에 정의되어 있어서 이를 이용하여 Simulink 모델과 메인 프로그램 두 시스템 동기되어 움직일 수 있도록 할 수 있다.

〈표 5〉 SC 스레드 오토마타 상태

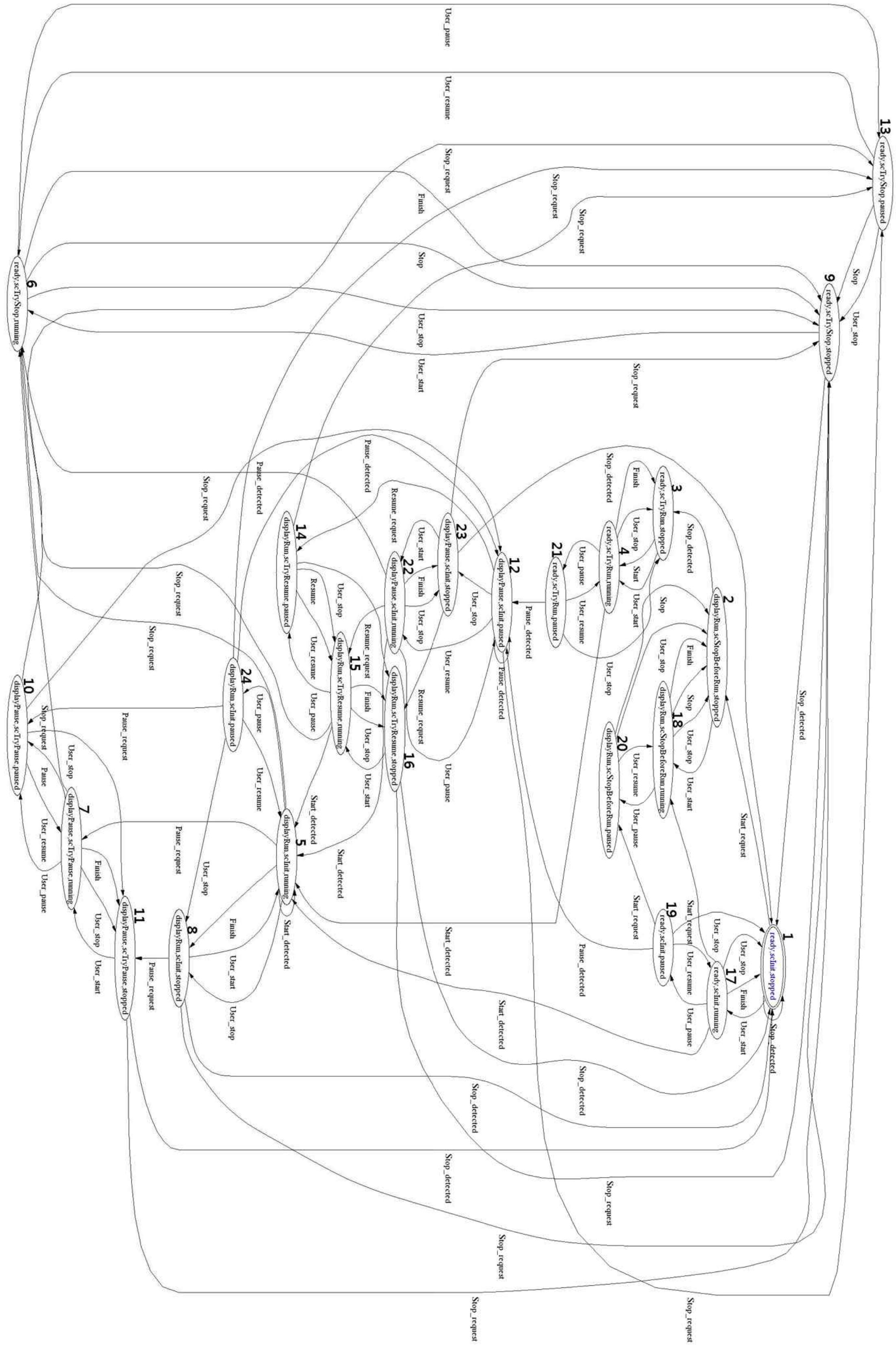
상태	설명
scInit	처리할 시뮬레이션 명령이 없는 상태
scStopBeforeRun	시뮬레이션 시작에 앞서 시뮬레이션 중단을 시도하는 상태, Stop_detected여야 상태 전이 가능
scTryRun	시뮬레이션 시작을 시도하는 상태, Simulink 모델의 상태 변화가 있어야 상태 전이 가능
scTryStop	시뮬레이션 중단을 시도하는 상태, Simulink 모델의 상태 변화가 있어야 상태 전이 가능
scTryPause	시뮬레이션 잠시 멈춤을 시도하는 상태, Simulink 모델의 상태 변화가 있어야 상태 전이 가능
scTryResume	시뮬레이션 재시작을 시도하는 상태, Simulink 모델의 상태 변화가 있어야 상태 전이 가능

Start_request 이벤트에 대해서는 다른 명령과는 다르게 scStopBeforeRun 상태에서 시뮬레이션 종료를 시도하고, Simulink 모델 오토마타의 상태가 stopped임을 확인한 후에 scTryRun 상태로 전이하도록 설계하였다. 구현하고자 하는 Simulink 실행 모듈은 선택된 테스트 케이스를 입력으로 모델을 시뮬레이션 시킬 수 있어야 한다. Simulink 모델은 시뮬레이션 시 모델의 입력 데이터를 모두 받아서 실행되는 형식이기 때문에 이미 시뮬레이션이 실행 중인 상태에서 사용자가 선택한 테스트 케이스를 입력으로 할 수 있는 방법이 없다. 따라서 Start_request 이벤트에 대해서는 모델을 stopped 상태로 만든 후 scTryRun 상태로 이동하도록 설계한다.



(그림 4) SC 스레드 오토마타

2.4 전체 시스템 오토마타



(그림 5) 전체 시스템 오토마타

전체 시스템의 오토마타는 앞서 알아본 3가지 시스템 오토마타의 Parallel Composition을 사용하여 얻을 수 있다. (그림 5)에 전체 시스템에 대한 오토마타를 나타내었다. Simulink 실행 모듈은 User_start, User_stop, User_pause, User_resume 이라는 사용자가 Simulink 모델을 통해 언제든지 발생시킬 수 있는 이벤트들이 존재하기 때문에 dead lock 이나 live lock 이 존재하기 힘든 구조이다. 하지만 Simulink 모델, 메인 프로그램, SC 스레드를 잘못 설계할 경우 앞에서 언급한 4가지 이벤트를 제외하고 상태 전이가 불가능한 경우가 발생한다. 따라서 전체 시스템 오토마타에 대해 dead lock과 live lock외에도 User_* 이벤트만 정의되는 상태들을 우선적으로 제거하고, (그림 4)의 모든 상태에서 허용되어야 할 이벤트가 존재하는가, 허용되지 말아야 할 이벤트가 존재하지는 않는가, 허용되는 이벤트에 대해 정상적으로 상태 전이가 일어나는가를 확인한다. 문제를 발견하면 2.1절부터 2.3절에서 정의된 상태와 이벤트들을 수정하거나 새로운 이벤트를 생각해보아야 한다. <표 6>에 모든 상태에서 이러한 사항들을 확인한 결과를 정리하였다. User_start, User_stop, User_pause, User_resume은 Simulink 모델 상태에 따라서 언제든지 발생 가능하기 때문에 제외하였고, 상태의 이름이 길기 때문에 번호를 사용하여 표현하기로 하겠다. 예를 들어 [ready, scInit, stopped] 상태는 상태1이라고 표현한다.

〈표 6〉 모든 상태에서의 이벤트 및 상태 전이 확인

상태	허용 이벤트 및 상태 전이 확인
1 [ready, scInit, stopped]	Start_request : 현재 상태는 시스템의 초기 상태로 시뮬레이션 시작 명령만 유효해야 한다. 이 이벤트를 제외한 *_request 이벤트들은 불허되어 있고, 이 이벤트에 의해 상태2로 정상 전이된다. Stop_detected : SC 스레드에서 항상 모델의 상태를 확인하도록 설계되어 있어서 Start, Stop, Pause, Resume과 같이 시뮬레이션 명령을 내리는 상태를 제외한 모든 상태에서 적절한 *_detected 이벤트가 항상 발생가능 하다. 초기 상태에서는 이 이벤트가 허용되고, 발생 시 정상적으로 현재 상태에 머문다.
2 [displayRun, scStopBeforeRun, stopped]	Stop_detected : Simulink 모델 시뮬레이션 시작 전에 stopped상태임을 확인해야 하고, 확인 후 상태3으로 전이한다.
3 [ready, scTryRun, stopped]	Start : 실제 Simulink 모델을 시뮬레이션 시작 시키고, 정상적으로 상태 4로 전이된다.
4 [ready, scTryRun, running]	Start_detected : 실제로 Simulink모델이 실행된 것을 확인하면 이 이벤트가 발생하고 상태5로 전이된다.
5 [displayRun, scInit, running]	Start_detected : 시뮬레이션 실행 중으로 Start_detected를 통해 이를 확인한다. 정상적으로 현재 상태에 머물러 있다. Stop_request, Pause_request : 현재 상태는 Simulink모델이 실행 중이고, 이와 같이 메인 프로그램에도 실행 중인 데이터들이 출력되는 상태이다. 이 상태에서 사용자는 시뮬레이션 중단 명령과 잠시 멈춤 명령을 내릴 수 있다. 각각 상태 6과 상태 7로 정상 전이된다. Finish : 시뮬레이션이 End time까지 모두 실행되어 스스로 종료되고 상태 8로 이동한다.
6 [ready, scTryStop, running]	Stop : 실제 Simulink모델의 시뮬레이션을 중단시키고 상태9로 이동한다. Finish : Simluink모델을 중단시키기 전에 시뮬레이션이 End time까지 모두 실행되어 스스로 종료된다. 마찬가지로 시뮬레이션이 중단되기 때문에 상태 9로 정상 전이된다.
7 [displayPause, scTryPause, running]	Pause : 실제 Simulink 모델의 시뮬레이션 잠시 멈춤시키고 상태10으로 이동한다. Finish : SC스레드가 미처 Simulink 모델을 잠시 중단시키기 전에 End time까지 모두 실행되어 스스로 종료될 수 있다. 종료 후에는 상태 11로 정상 전이된다. Stop_request : Simulink 모델을 잠시 중단시키기 전에 종료 명령이 발생하여 상태 6으로 정상 전이된다.
8 [displayRun, scInit, stopped]	Stop_detected : 모델이 종료된 것이 확인되어 상태 1로 전이된다. Stop_request, Pause_request : 모델이 종료된 것이 미처 확인되기 전에 시뮬레이션 종료 명령과 잠시 중단이 가능하며 각각 상태 9와 상태 11로 이동한다.
9 [ready, scTryStop, stopped]	Stop_detected : 이 이벤트에 의해 시뮬레이션 중단이 확인되고 초기상태인 상태1로 이동한다.
10 [displayPause, scTryPause, paused]	Pause_detected : 모델이 잠시 멈춤 상태이므로 이 이벤트에 의해 감지되어 상태12로 이동할 수 있다. Stop_request : 모델의 잠시 멈춤 상태가 미처 확인되기 전에 중단 명령이 들어올 수 있고, 상태 13로 이동한다.

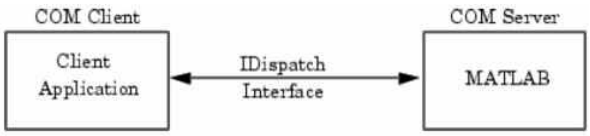
11 [displayPause, scTryPause, stopped]	Stop_detected : 모델이 종료된 것이 확인되어 정상적으로 상태 1로 전이된다. Stop_request : 모델이 종료된 것이 확인되기 전에 이 이벤트가 발생가능하고, 상태 9로 이동한다.
12 [displayPause, scInit, paused]	Resume_request : 잠시 멈춤 상태이므로 재시작 명령이 가능하고 상태 14으로 이동한다. Stop_request : 시뮬레이션 종료 명령도 가능하며 상태 13로 정상 이동한다.
13 [ready, scTryStop, paused]	Stop : Simulink 모델의 시뮬레이션을 중단시키고 상태 9로 이동한다.
14 [dispalyRun, scTryResume, paused]	Resume : 시뮬레이션 재시작 명령에 의해 Simulink 모델을 재시작시키고 상태 15로 이동한다. Stop_request : Simulink 모델이 재시작되기 전에 사용자로부터 종료 명령이 들어오고 상태 13으로 이동한다.
15 [displayRun, scTryResume, running]	Start_detected : 시뮬레이션이 재시작된 것을 확인하고 상태 5로 이동한다. Stop_request, Finish : 시뮬레이션이 재시작된 것을 미처 확인하기 전에는 종료 명령과 Finish 이벤트가 발생 가능하고, 각각 상태 6, 상태 16으로 이동한다.
16 [displayRun, scTryResume, stopped]	Stop_detected : Simulink모델의 시뮬레이션이 중단되었음을 확인하고 상태 1로 이동한다. Stop_request : 시뮬레이션이 중단되기 전에는 시뮬레이션 중단 명령이 가능하고, 상태 9로 이동한다.
17 [ready, scInit, running]	Start_detected : 상태 1에서 User_start이벤트로 도달가능한 상태이다. Simulink 모델이 실행 중이므로 이 이벤트에 의해 상태 5로 이동이 가능하다. Start_request, Finish : 시뮬레이션 실행 중임이 미처 확인되기 전에 시뮬레이션 시작 명령과 Finish 이벤트가 발생가능하고 각각 상태 18과 상태 19로 이동한다.
18 [displayRun, scStopBeforeRun, running]	Stop : 시뮬레이션 시작 명령에 의해 우선 실행 중인 시뮬레이션을 중단시킨다. 상태 2로 이동한다. Finish : 마찬가지로 End time까지 실행 후 스스로 종료되면서 상태 2로 이동될 수 있다.
19 [ready, scInit, paused]	Pause_detected : 잠시 중단 상태임이 감지되어 상태 12로 이동한다. Start_request : 잠시 중단 상태임이 감지되기 전에는 시뮬레이션 시작 명령이 가능해야 한다. 상태 20으로 이동한다.
20 [displayRun, scStopBeforeRun, paused]	Stop : 시뮬레이션 시작 명령에 의해 우선 현재 실행 중인 시뮬레이션을 중단시킨다. 상태 2로 이동한다.
21 [ready, scTryRun, paused]	Pause_detected : 이 상태는 상태 4에서 User_pause이벤트로 도달가능하다. Simulink 모델이 잠시 중단된 상태로 이 이벤트에 의해 상태 12로 전이가 가능하다.
22 [displayPause, scInit, running]	Start_detected : 이 상태는 상태 12에서 User_resume 이벤트로 도달가능하다. 시뮬레이션이 재시작된 것이 감지되어 상태 5로 정상 전이된다. Resume_request, Finish : 재시작 된 것이 미처 감지되기 전에 사용자로부터 재시작 명령이 들어오거나 Fininsh 이벤트가 발생가능하고 각각 상태 15와 상태 23으로 이동한다.
23 [displayPause, scInit, stopped]	Stop_detected : 시뮬레이션 중단이 감지되어 상태 1로 이동한다. Stop_request, Resume_request : 시뮬레이션 중단이 감지되기 전으로 사용자로부터 종료 명령과 잠시 멈춤 명령이 들어올 수 있고 각각 상태 9와 상태 16으로 이동한다.
24 [displayRun, scInit, paused]	Paused_detected : 상태 5에서 User_pause로 도달가능한 상태이다. 잠시 멈춤이 확인되면 상태 12로 이동한다. Stop_request, Pause_request : 잠시 멈춤이 확인되기 전에는 사용자로부터 시뮬레이션 중단 명령과 잠시 멈춤 명령이 가능하여야 하고 각각 상태 13과 상태 10으로 이동한다.

3. Simulink 실행 모듈 구현

3.1 C# 프로그램에서 MATLAB 사용

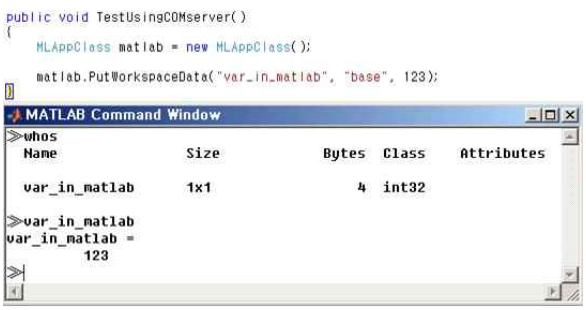
본 논문에서 제시하는 Simulink 실행 모듈에서 우선적으로 구현해야 할 것은 외부 프로그램에서 Simulink 모델을 제어할 수 있는 기능이다. Simulink 모델은 MATLAB의 확장자로 MATLAB프로그램을 이용하여 모델의 정보를 얻어오거나 시뮬레이션을 제어하는 것이 가능하다. 또한, MATLAB은 외부 프로그램에서 MATLAB을 사용할 수 있도록 여러 가지 기능을 제공해주고 있다. 따라서 MATLAB 프로그램으로 필요한 함수를 구현한 후, 외부의 프로그램에서 MATLAB 함수를 호출하는 방법으로 모듈에서 Simulink 모델을 제어하는 기능을 구현할 수 있다.

MATLAB은 외부의 어플리케이션에서 MATLAB으로 구현된 프로그램을 실행시킬 수 있도록 Automation COM server를 제공한다[8]. 예를 들면 Microsoft Excel, Microsoft Access, Microsoft Visual Basic, Microsoft Visual C++ 등의 프로그램에서 사용이 가능하다.



(그림 6) 외부에서의 MATLAB COM server 사용[9]

Simulink 실행 모듈은 이 중 C#을 사용하여 구현하도록 한다. 다음 (그림 7)은 C#에서의 COM server 사용 예시이다. 예시에서는 C#에서 int32형의 123이라는 값을 MATLAB workspace에 "var_in_matlab"이라는 이름의 변수에 저장하고 있다. GetWorkspaceData라는 함수를 사용하면 거꾸로 MATLAB workspace의 데이터를 C#으로 가져오는 것도 가능하다.



(그림 7) C#에서의 MATLAB COM server 사용 예시

3.2 MATLAB 을 통한 Simulink 모델 제어

MATLAB은 Simulink를 위한 여러 가지 함수를 제공한다. Simulink 모델 오토마타에서 정의된 Start, Stop, Pause, Resume의 구현은 그 중에서 set_param 함수를 이용하여 구현할 수 있다. 마찬가지로 Stop_detected, Start_detected, Pause_detected 역시 MATLAB에서 제공하는 get_param 함수를 사용하여 구현할 수 있다[10].

다음 (그림 8)은 set_param과 get_param 함수의 사용 예시이다.

```
>> set_param('model', 'SimulationCommand', 'Start')
>> get_param('model', 'SimulationStatus')
ans =
running
```

(그림 8) set_param & get_param 사용 예시

3.3 SC 스레드 구현

메인 프로그램은 단순히 UI를 이용하여 사용자의 명령을 받고, 모델의 정보를 보여주는 기능을 구현한다. Simulink 모델 열기 버튼이 입력되면 (그림 7)과 같은 방법으로 MATLAB COM server를 실행시키고, SC 스레드를 실행시킨다. Simulink 모델 닫기 버튼이 입력되면 MATLAB COM server를 종료시키고 SC 스레드도 종료시킨다.

Simulink 모델의 시뮬레이션 제어는 모두 SC 스레드에 의해 이루어지는데, 2장에서 검증했던 SC 스레드 오토마타의 상태를 그대로 반영하여 구현하도록 한다. 또한 각 상태에서는 발생하여야 하는 이벤트들을 모두 구현하고, 시뮬레이션 명령에 대해서는 (그림 8)과 같은 MATLAB 함수를 호출한다. 2장에서 설계된 것과 같이 각 이벤트마다 적절한 상태 전이를 발생시킨다.

3.4 MATLAB workspace에 정보 저장

Simulink 실행 모듈은 2장에서 정리한 것과 같이 이 외에도 Simulink 모델 출력 정보, Stateflow 내부 데이터 정보를 제공해줄 수 있어야 한다. 이러한 데이터들을 MATLAB workspace에 저장할 수만 있다면 3.1절에서 설명한 방식으로 C# 프로그램에서 데이터들을 얻는 것이 가능하다. 데이터의 workspace저장에 대한 설명에 앞서 간단한 예제 모델을 먼저 소개한다.

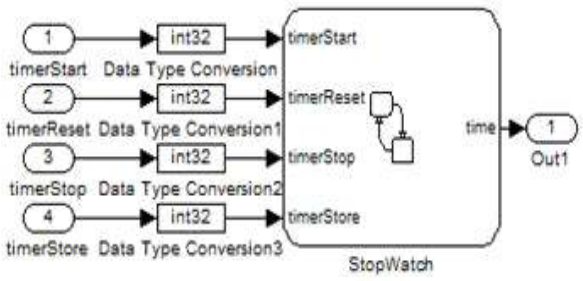
Simulink 모델에는 입력 역할의 Inport 블록 4개와 출력 역할의 Outport 블록 1개가 존재하며, 스톱워치 기능은 Stateflow chart 내부에 구현되어 있다.

Stateflow chart 내부에는 Timer 라는 상태가 있어서 timerStart, timerStop, timerReset 입력값에 의해 카운팅을 시작할 것인가, 종료할 것인가 결정된다. Store 상태에서는 앞의 Timer 상태와는 무관하게timerStore라는 입력값에 의해 현재 time값을 내부의 배열 변수 storage에 저장하는 역할을 한다. Reset 상태에서는 마찬가지로 앞의 Timer상태와 Store상태와는 무관하게 timerReset 입력값에 의해 time변수와 내부 배열 변수 storage를 0으로 초기화 하는 역할을 한다.

Simulink 모델의 출력은 기본적으로 Outport 블록만 사용하면 yout이라는 이름의 변수로 시뮬레이션 결과를 MATLAB workspace에 저장하도록 설정되어 있다. 하지만

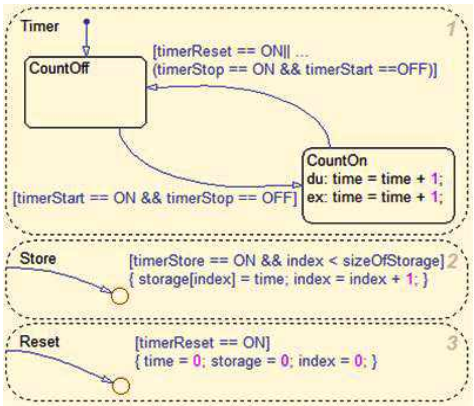
```
다음의 동작을 무한히 반복한다.
  (그림 8)의 방법으로 Simulink 모델의 시뮬레이션 상태를 감지한다.
  SC 스레드 상태에 따라 다음의 동작을 수행한다.
   SC 스레드 상태가 scInit 인 경우
     메인 프로그램에서 Start_request가 발생하였다면 SC 스레드 상태를 scStopBeforeRun 로 전이한다.
     메인 프로그램에서 Pause_request가 발생하였다면 SC 스레드 상태를 scTryResume 로 전이한다.
     메인 프로그램에서 Stop_requrest가 발생하였다면 SC 스레드 상태를 scTryStop 로 전이한다.
   SC 스레드 상태가 scStopBeforeRun 인 경우
     (그림 8)의 방법으로 Simulink 모델에 시뮬레이션 Stop 명령을 내린다.
     시뮬레이션 상태가 Stop 이라면 SC 스레드 상태를 scTryRun 상태로 전이한다.
   SC 스레드 상태가 scTryRun 인 경우
     (그림 8)의 방법으로 Simulink 모델에 시뮬레이션 Start 명령을 내린다.
     메인 프로그램에서 Stop_request가 발생하였다면 SC 스레드 상태를 scTryStop 로 전이한다.
     시뮬레이션 상태가 Start 라면 SC 스레드 상태를 scInit 로 전이한다.
     시뮬레이션 상태가 Pause 라면 SC 스레드 상태를 scInit 로 전이한다.
   SC 스레드 상태가 scTryPause 인 경우
     (그림 8)의 방법으로 Simulink 모델에 시뮬레이션 Pause 명령을 내린다.
     메인 프로그램에서 Stop_requrest가 발생하였다면 (그림 8)의 방법으로 시뮬레이션 명령을 전달한다.
     시뮬레이션 상태가 Pause 라면 SC 스레드 상태를 scInit 로 전이한다.
     시뮬레이션 상태가 Stop이라면 SC 스레드 상태를 scInit 로 전이한다.
   SC 스레드 상태가 scTryResume 인 경우
     (그림 8)의 방법으로 Simulink 모델에 시뮬레이션 Resume 명령을 내린다.
     메인 프로그램에서 Stop_requrest가 발생하였다면 (그림 8)의 방법으로 시뮬레이션 명령을 전달한다.
     시뮬레이션 상태가 Start 라면 SC 스레드 상태를 scInit로 전이한다.
     시뮬레이션 상태가 Stop이라면 SC 스레드 상태를 scInit로 전이한다.
   SC 스레드 상태가 scTryStop 인 경우
     (그림 8)의 방법으로 Simulink 모델에 시뮬레이션 Stop 명령을 내린다.
     시뮬레이션 상태가 Stop이라면 SC 스레드 상태를 scInit로 전이한다.
```

(그림 9) SC 스레드 구현 Pseudo code



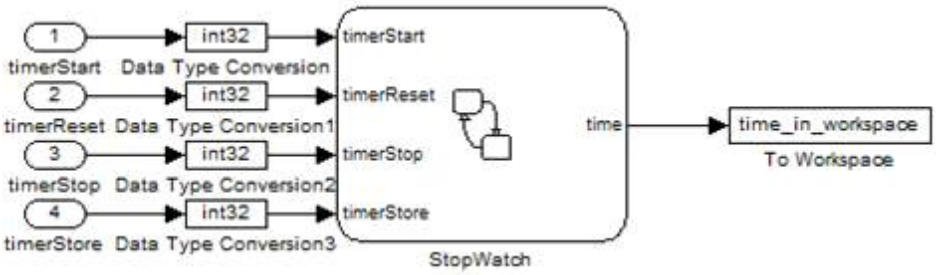
(그림 10) 스톱워치 예제 모델

이 Outport 블록을 사용하는 방법은 모든 출력들이 하나의 2차원 변수로 저장되어 원하는 데이터만 얻는 것이 번거롭기도 하고, 최상위에 Outport 블록을 위치시켜야 된다는 제약 사항이 존재한다. 따라서 이 방법보다는 To Workspace 블록을 사용하여 원하는 데이터를 MATLAB workspace에 저장하기로 한다. (그림 12)은 예제 모델의 출력 역할을 하던 Outport 블록을 To Workspace 블록으로 변경한 예시이다. workspace에 저장될 변수 이름은 C#프로그램에서 마음대로 지정이 가능하다.

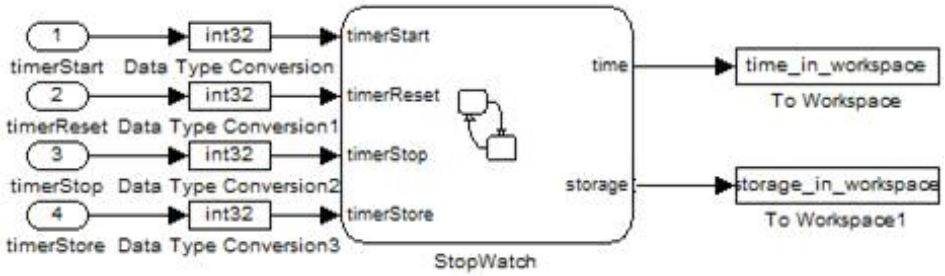


(그림 11) 예제 모델의 Stateflow chart내부

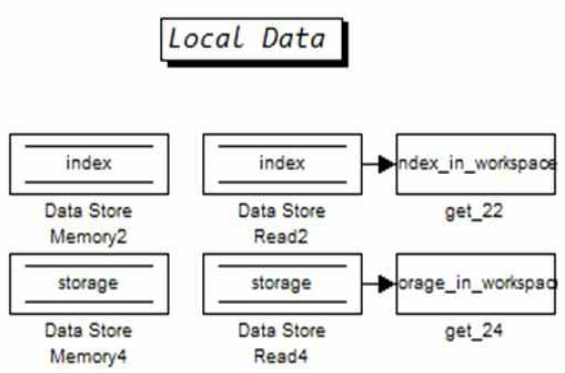
Stateflow 내부의 데이터 값도 시뮬레이션 중 MATLAB workspace에 저장해야 한다. 예제 모델에서는 storage라는 이름의 변수가 내부 데이터로 설정되어 있다. 생각할 수 있는 가장 간단한 방법은 (그림 13)과 같이 storage 변수를 출력 데이터로 변경하여 앞서 사용한 To Workspace 블록으로 저장시키는 방법이다.



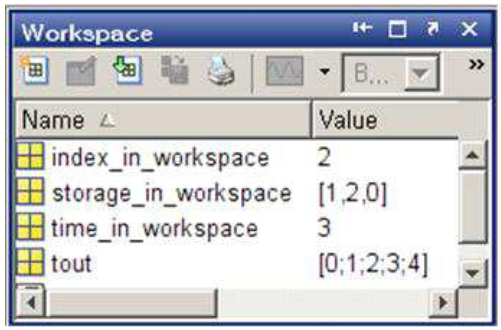
(그림 12) Simulink 모델 시뮬레이션 출력 저장



(그림 13) Stateflow 내부 데이터 변경 예



(그림 14) Statefow 내부 데이터 저장



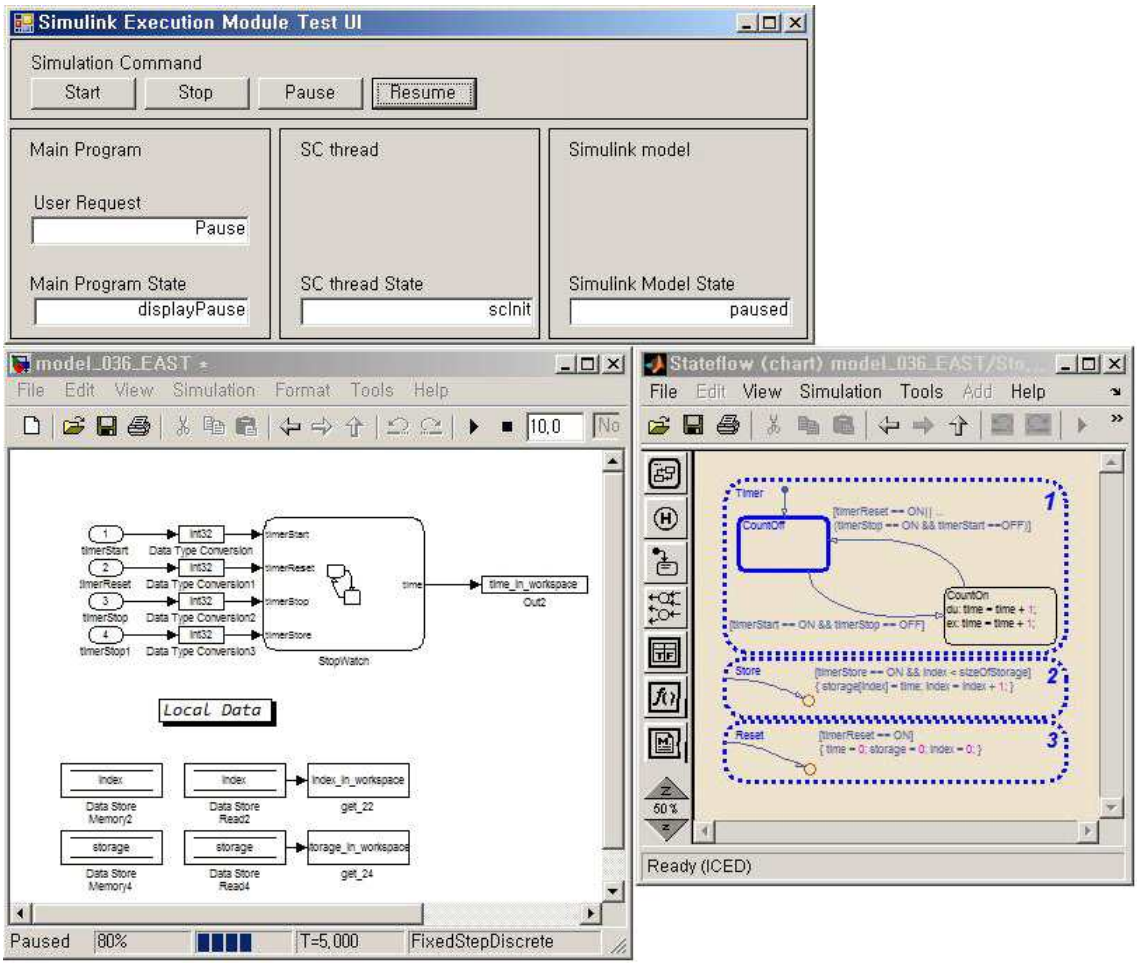
(그림 15) workspace저장된 데이터들

하지만 Stateflow 의 출력은 Stateflow의 타입 옵션에 의해서 변경될 수 있으며, 따로 출력만을 위한 옵션도 존재하여 출력 방식이 변경될 수 있다. 이러한 옵션에 영향을 받지 않던 내부 데이터를 그냥 출력 데이터로 변경하는 경우 원 모델과 상이하게 동작할 가능성이 매우 크다. 따라서 본 논문에서는 Data Store Memory 블록을 사용하여 이러한 문제점을 해결하려고 한다. 이 블록을 사용하면 Stateflow내부의 데이터를 Simulink 모델에서의 신호로 존재하게 설정할 수 있다. (그림 14)은 예제 모델에 이러한 방법으로 적용하여 새로 생성한 블록들의 그림이고, (그림 15)는 이러한 블록들에 의해 시뮬레이션 중 Stateflow 내부의 데이터가 MATLAB workspace에 저장된 그림이다.

4. 실 험

실험의 목적은 3장의 방법을 통해 구현한 Simulink 실행 모듈이 2장에서 오토마타를 통해 검증한 것과 동일하게 동작하는지 확인하는 것이다. 시뮬레이션의 제어를 확인하는 과정으로 모든 Simulink 모델은 시뮬레이션 관점에서 (그림 2)와 동일하기 때문에 오류가 없어서 실행 가능한 모델이라면 어떠한 것이라도 상관없다. 본 논문에서는 3.4 절에서 설명한 스톱워치 모델을 사용하였다.

확인 방법은 2.4절 전체 시스템 오토마타를 통해 동작을 확인하였던 것과 동일한 방법으로 모든 상태와 발생가능 이벤트, 이벤트 후의 상태 전이를 테스트하여 시뮬레이션 제어 기능이 정상 동작하는 것을 확인하는 것으로 결과는 <표 6>과 동일하다. 만약 <표 6>과 동일하지 않다면 원인이 되는 상태와 이벤트를 반영하여 2장의 과정과 실험 과정을 반복하여야 한다. (그림 16)은 실험을 위해 만든 테스트 프로그램 화면과 예제 모델의 그림이다. 테스트 프로그램에서는 MATLAB workspace에 데이터들이 저장되는 것만 확인하고, 화면에 출력하는 구현은 생략되어 있다.



(그림 16) 테스트 프로그램과 예제 모델

5. 결 론

본 논문에서는 오토마타 이론을 적용하여 사용자의 시뮬레이션 명령과 Simulink 모델의 시뮬레이션 명령을 통해 시뮬레이션을 제어하는 Simulink 실행 모듈을 설계하고 이를 구현하였다. 이 모듈은 C#으로 구현한 프로그램과 Simulink 모델은 서로 독립적인 시스템으로 설계에 어려운 점이 있는데 오토마타 이론을 적용하여 이러한 문제점을 해결하였다. 구현한 모듈은 Simulink의 강력한 시뮬레이션 기능을 사용하려는 여러 프로그램들에 적용시킬 수 있고, 특히 Simulink 모델 기반의 테스트에 많은 도움되리라 기대한다.

참 고 문 헌

[1] The Mathworks, http://www.mathworks.com/products/simulink/

[2] 송문빈, 송태훈, 오재곤, 정연모, "효율적인 통합 시뮬레이션에 의한 스피커 연결 시스템의 SoC 설계," 전자공학회논문지, 제43권 SD편, 제10호, pp.671-676, 2006.

[3] BTS Technologies inc., http://www.btstech.co.kr/

[4] 서희석, 정인상, 김병만, 권용래, "Java 다중 스레드 프로그램을 위한 오토마타 기반 테스팅 환경의 설계 및 구현," 정보과학회논문지, 소프트웨어 및 응용 제 29권, 제 11·12호, pp.883 - 894, 2002.

[5] Christos G. Cassandras and Stephane Lafortune, Introdunction to Descrete Event Systems. 2nd edition, Springer, New York, pp.62 - 133.

[6] Electrical Engineering and Computer Science, https://www.eecs.umich.edu/umdes/projects/lib/download_access/submit_desuma.html

[7] DESUMA Software, http://www.eecs.umich.edu/umdes/toolboxes.html

[8] The Mathworks, http://www.mathworks.com/help/toolbox/simulink/ slref/f23-7515.html

[9] The Mathworks, MATLAB® 7 External Interfaces, pp.323 - 452, 2010

[10] The Mathworks, http://www.mathworks.com/help/toolbox/simulink/ ug/f11-61851.html

김 경 준

e-mail : keij81@naver.com
2009년 아주대학교 전자공학부(공학사)
2011년 아주대학교 전자공학과(공학석사)
관심분야 : 임베디드 시스템, 요구사항 모델링, 소프트웨어 테스팅 등

정 기 현

e-mail : khchung@ajou.ac.kr
1984년 서강대학교 전자공학과(학사)
1988년 미국 Ilinois 주립대 EECS(석사)
1990년 미국 Purdue 대학 전기전자공학부 (박사)
1991년～1992년 현대반도체 연구소
1993년～현 재 아주대학교 전자공학부 교수
관심분야 : 컴퓨터 구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등

최 경 희

e-mail : khchoi@ajou.ac.kr
1976년 서울대학교 수학교육과(학사)
1979년 프랑스 그랑데꼴 Enseeiht 대학 (석사)
1982년 프랑스 Paul Sabatier 대학 정보공학부(박사)
1982년～현 재 아주대학교 정보통신전문대학원 교수
관심분야 : 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템 등