Journal of Korea Multimedia Society Vol. 16, No. 1, January 2013(pp. 75-86)
http://dx.doi.org/10.9717/kmms.2013.16.1.075

마이크로컨트롤러 인터럽트를 사용한 임베디드시스템의 다중 상태기계 모델링 기반 구현 기법

이 상 설†

요 약

본 논문은 많은 주변장치와 인터페이스 되는 단일칩 마이크로컨트롤러로 구현되는 임베디드시스템을 인터럽트를 사용하여 다중 유한상태기계로 모델링하고 구현하는 방법을 제시한다. 다중 상태기계 모델은 하드웨어설계에 사용되는 FSMD 구조와 인터럽트에 의한 흐름제어 특징을 이용한다. 주 프로그램은 주상태기계에 대응하고, 부상태기계는 인터럽트 서비스루틴에 대응한다. 따라서 주변장치에서 발생하는 인터럽트는 부상태기계에서 신속히 처리될 수 있다. 유한상태기계 사이의 인터페이스는 요청과 응답 변수를 사용한다. 주상태기계와 부상태기계 사이의 콘텍스트 스위칭은 인터럽트에 의한 하드웨어 흐름제어로 대치될 수 있어 별도의 운영체제가 필요하지 않다. ASM 차트를 사용하여 다중 유한상태기계로 모델링된 임베디드시스템은 C언어 프로그램으로 변환시켜 쉽게 구현될 수 있다. 이 구현 방법은 모델링이 구체적고 부상태기계에서 인터럽트에 신속히 반응할 수 있어 하드웨어가 많이 사용되는 내장형시스템에 쉽게 적용될 수 있다.

An Embedded Systems Implementation Technique based on Multiple Finite State Machine Modeling using Microcontroller Interrupts

Sang Seol Lee†

ABSTRACT

This paper presents a technique to implement embedded systems using interrupts of the one-chip microcontroller with many peripherals based on a multiple finite state machines model. The multiple finite state machine model utilizes the structure of FSMD used for hardware design and the features of flow control by interrupts. The main finite state machine corresponds to the main program and the sub-state machines corresponds to the interrupt subroutines. Therefore, interrupts from the peripherals can be processed immediately in the sub-state machines. The request and reply variables are used to interface between the finite state machines. Additional operating system is not necessary for the context switching between the main state machine and the sub-state machine, because the flow-control caused by interrupt can be replaced with the switching. An embedded system modeled on multiple finite state machine with ASM charts can be easily implemented by the conversion of ASM charts into C-language programs. This implementation technique can be easily adopted to the hardware oriented embedded systems because of the detail description of the model and the fast response to the interrupt events in the sub-state machine.

Key words: Embedded Systems(임베디드시스템), Microcontroller(마이크로컨트롤러), Multiple Finite State Machines(다중 상태기계), ASM chart(ASM 차트), FSMD

※ 교신저자(Corresponding Author) : 이상설, 주소 : (570-749) 전북 익산시 신용동 344-2, 원광대학교 전기·정보통신공학부, 전화 : 063) 850-6889, FAX : 063) 850-6889, E-mail : slee@wonkwang.ac.kr

접수일 : 2012년 9월 10일, 수정일 : 2012년 10월 30일
완료일 : 2012년 11월 12일

† 정회원, 원광대학교 전기·정보통신공학부

1. 서 론

산업, 가전, 의료 등의 특수한 목적을 수행하기 위한 여러 분야에서 하드웨어 의존성 높은 임베디드시스템 개발이 활성화되어 있다. 임베디드시스템 처리 장치로는 단일칩 마이크로컨트롤러가 많이 사용되고 있다. 최근 성능이 높아진 마이크로컨트롤러 혹은 SoC 형태의 마이크로컨트롤러 단일 칩 안에는 CPU 코어 외에 프로그램 메모리와 데이터 메모리 그리고 인터럽트 기능을 갖는 다양한 주변장치 등을 내장하고 있다[1,2]. 그림 1은 마이크로컨트롤러를 이용한 임베디드시스템의 하드웨어 구성을 나타내고 있다. 다양한 인터럽트 기능을 갖는 내장된 주변장치 외에 별도로 필요한 주변장치를 마이크로컨트롤러 외부에 연결하여 임베디드시스템 하드웨어가 구성된다.

구성된 하드웨어를 임베디드시스템 목적에 맞게 구동시키기 위해서는 프로그램을 프로그램 메모리에 다운로드 시켜야 한다. 프로그램 개발 방법으로는 동작을 플로차트로 그래픽하게 도식화하여 기술한 후 프로그램으로 옮겨야 사용하므로 코드 변환이 중요하다[3]. 플로차트는 동작을 직관적으로 기술하기 용이하지만 구체적인 기술이 쉽지 않다. 플로차트에서 자동 코드변환에 관한 연구가 진행되고 있지만 구체적인 알고리즘 기술보다 단순한 코드 변한에 치중되고 있다[4]. 반면 유한상태기계를 이용하면 시스템에 대한 동작과 테스트를 위한 기술을 구체적으로 할 수 있어 여러 분야에서 사용될 수 있다[5,6]. 그러나 대부분 단일 유한상태기계에 기반한 방법이 주로 사용되고 있다[7]. 마이크로컨트롤러 내부와 외부에 많은 하드웨어 주변장치가 연결된 임베디드시스템

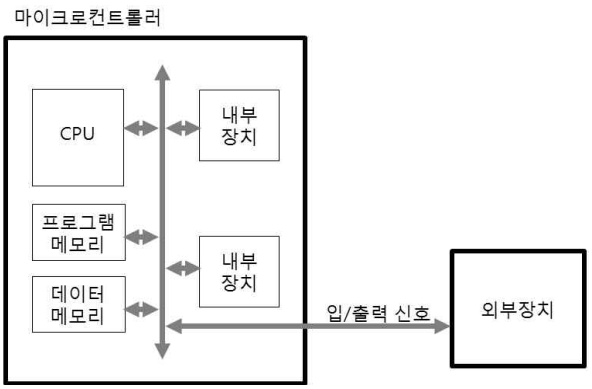


그림 1. 마이크로컨트롤러를 이용한 임베디드시스템 하드웨어 구성

은 주변장치에서 사건이 발생했을 경우 신속하게 그에 필요한 작업을 수행해야 한다. 단일 유한상태기계를 사용하는 임베디드시스템은 사건 처리 상태로 천이된 후에 필요한 작업을 수행할 수 있으므로 시간지연이 있어 신속하게 대응하기 어렵다.

본 논문에서는 하드웨어 의존적인 임베디드시스템에서 인터럽트를 이용한 다중 유한상태기계 기반 임베디드시스템 구현 기법을 제안한다. 마이크로컨트롤러 명령어의 실행 흐름은 리셋에서 시작되는 주 프로그램 실행 흐름과 인터럽트 발생에 의한 인터럽트 서비스루틴의 실행 흐름이 있다. 인터럽트로 프로그램의 흐름제어가 하드웨어적으로 변경되고 주변장치와 관련된 조건변수를 갱신시킬 수 있는 특징을 이용하여 임베디드시스템 작업을 다중화된 유한상태기계로 모델링하고 각 유한상태기계의 동작을 기술한 후 프로그램으로 변환한다. 주 프로그램은 주상태기계로 모델링하고 인터럽트 서비스루틴은 부상태기계로 모델링한다. 주상태기계와 부상태기계 사이는 변수를 이용하여 요청과 응답형식으로 인터페이스 한다. 각 유한상태기계는 ASM(Algorithm State Machine) 차트로 기술된 후 프로그램으로 변환된다. 2장에서는 임베디드시스템의 인터럽트 기반 다중 유한상태기계 모델링과 마이크로컨트롤러 대응 동작을 다룬 후 다중 유한상태기계의 프로그램 변환을 살펴본다. 3장에서는 센서 인터럽트와 타이머 인터럽트에 의해 동작되는 리프트 시스템으로 적용사례를 다룬다. 4장에서는 결론을 맺는다.

2. 인터럽트를 이용한 임베디드시스템의 다중 유한상태기계 모델링과 프로그램 변환

2.1 연구 배경

임베디드시스템은 특정한 작업을 수행하기 위해 디지털시스템을 내장시켜 타깃 장치를 목적에 맞게 제어한다. 다양한 주변장치를 단일 칩에 포함시킨 마이크로컨트롤러를 사용하면 소형화, 개발시간 단축 등 여러 장점을 갖는다. 마이크로컨트롤러에는 인터럽트 기능을 갖는 다양한 주변장치를 내장하고 있고 필요한 경우 외부에 인터럽트와 연동된 주변장치를 추가로 연결할 수 있어 하드웨어 의존적인 임베디드시스템 구현이 용이하다. 주변장치에서 발생하는 인터럽트는 주변장치의 사건발생을 하드웨어적으로

감지할 수 있는 가장 빠른 신호다.

단일 유한상태기계 모형에서 주변장치로부터 인터럽트가 발생하면 관련된 조건변수를 갱신하고 그 변수를 검사하는 상태로 천이한 후에야 갱신된 변수를 시스템에 적용할 수 있다. 그림 2는 상태 A에서 사용되는 조건변수1이 상태 C인 상황에서 인터럽트에 의해 변경되었다 하더라도 상태 C를 거쳐 상태 A로 도달될 때까지 갱신된 조건변수1은 시스템에 반영되지 못한다. 단일 유한상태기계 모형에서는 동시다발적으로 발생하는 사건들의 발생 시점과 유한상태기계 적용 시점 사이에 시간지연이 길어질 수 있어 신속한 처리가 요구되는 임베디드시스템 환경에서 제약을 갖는다.

인터럽트 사건발생에 대한 응답을 담당하는 별도의 유한상태기계를 갖는 다중 유한상태기계 방법으로 임베디드시스템을 설계하면 시스템 작업을 여러 곳에서 다중으로 수행 할 수 있어 사건에 신속하게 응답할 수 있는 설계가 가능하다. 그런데, 일반적으로 임베디드시스템에서 다중 유한상태기계 시스템 운영을 위해서는 부가의 일반 운영체제 혹은 실시간 운영체제가 필요하다[8,9]. 단일칩 마이크로컨트롤러로 구성된 하드웨어 의존적인 임베디드시스템에서 주변장치 사건발생에 신속하게 반응하기 위해 인터럽트와 연동된 다중 유한상태기계 모형에 의한 구현 기법을 제안한다. 인터럽트는 하드웨어적으로 프로그램의 흐름을 변경시킨다[10]. 제안하는 다중 유한상태기계 모형은 주상태기계와 부상태기계로 구성된다. 인터럽트로 주상태기계에서 부상태기계로 스위칭되게 하여 다중 유한상태기계를 부가의 운영체제 없이 사건에 신속히 응답할 수 있는 하드웨어 의존성 높은 임베디드시스템을 구현할 수 있다. 인터럽트가 발생되면 주상태기계에서 부상태기계로 하드웨어적으로 스위칭되고, 인터럽트 서비스루틴이 종료하면 주상태기계로 다시 스위칭된다. 따라서, 주변장치에서 인터럽트가 발생하면 신속하게 부상태기계의 상태에 따라 장치제어와 상태 천이가 가능하게 된다. 주상태기계와 부상태기계 사이의 정보교환은 유한상태기계 사이에 공유하는 변수에 대한 읽기와 쓰기로 이루어진다.

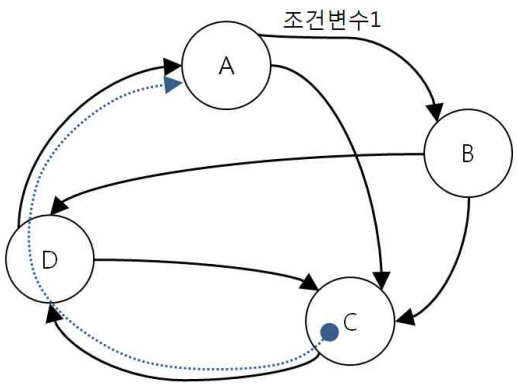


그림 2. 인터럽트 발생으로 갱신되는 조건변수에 의한 상태 천이

하드웨어 시스템에 대한 행위기술(behavioral description)은 합성절차에 의해 데이터경로(datapath)와 제어부(controller)를 포함하는 레지스터 전송수준(RTL)의 하드웨어로 변환된다[11]. 하드웨어는 FSMD(Finite State Machines with Data Path)로 정의하여 설계될 수 있으며 제어부는 유한상태기계로 변환되고 데이터경로를 통해 신호가 메모리(레지스터) 사이를 오가며 연산된다[11]. FSMD는 $<Q, q0, I, V, O, f, h>$로 정의될 수 있고, $Q = \{q0, q1, ..., qn\}$ 인 유한한 제어 상태 집합, $q0 \in Q$ 인 초기 상태, I 는 원시 입력신호 집합, V는 저장 변수 집합, O는 원시 출력신호 집합, f는 상태천이 함수, h 는 출력신호와 저장 변수에 대한 연산 함수를 나타낸다.[12]. 그림 3은 FSMD를 클록에 동기되어 작동하는 유한상태기계를 사용하여 나타낸 것이다.

FSMD는 하드웨어 시스템 행위를 신호에 기반한 시스템으로 정의한 표현이다. 유한상태기계는 입력장치에서 입력되는 원시 입력신호 I, 출력장치로 출력되는 원시 출력신호 O, 메모리와 입출력되는 V 신호와 인터페이스하는 제어동작을 수행한다. 그림 1과 같이 다양한 주변장치를 포함한 단일칩 마이크로컨트롤러에서 수행되는 시스템 행위도 FSMD 개념으로 표현할 수 있다. 그림 4는 그림 1에 해당되는

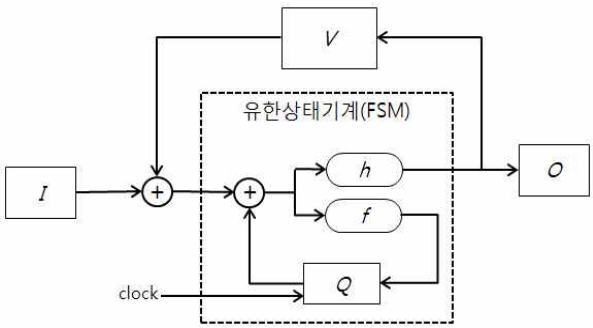


그림 3. 유한상태기계로 표현한 FSMD

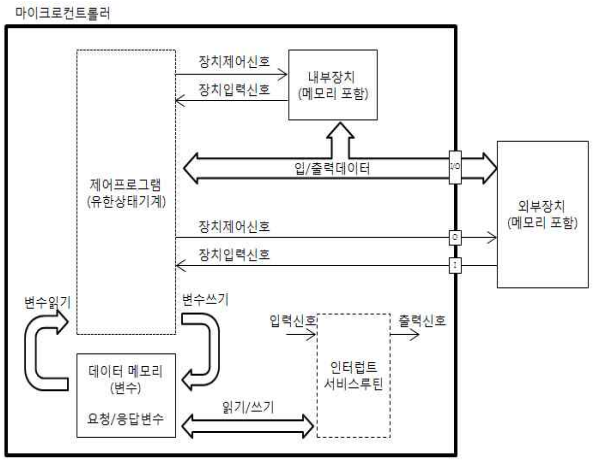


그림 4. 원칩 마이크로컨트롤러를 사용한 시스템의 단일 유한상태기계 기반 FSMD 모델링

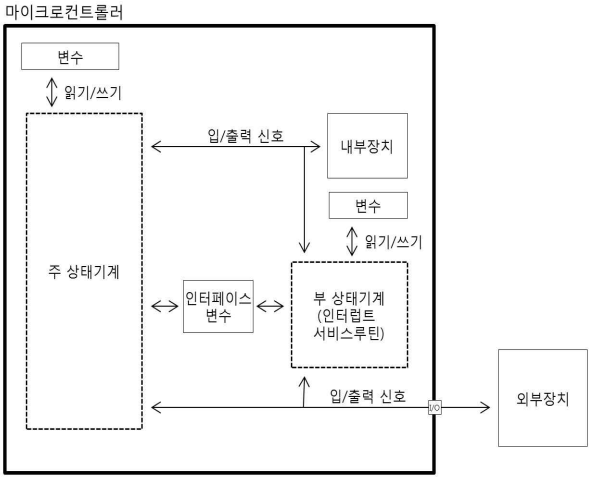


그림 5. 인터럽트를 이용한 다중 유한상태기계로 모델링된 임베디드시스템

단일칩 마이크로컨트롤러를 사용한 임베디드시스템 구성을 단일 유한상태기계를 사용한 FSMD 형식으로 모델링한 것이다[13]. 장치입력신호는 I, 장치제어신호는 O, 그리고 데이터메모리, 내부장치 메모리, 외부장치 메모리에 대한 데이터 입출력은 V에 대한 입출력 신호에 대응된다. 인터럽트서비스 루틴은 메모리에 있는 변수를 갱신하거나 특정 입출력 함수동작을 수행한다. 인터럽트 발생으로 갱신된 변수가 제어프로그램인 유한상태기계에서 반영되기 위해서는 변수가 사용되는 상태까지 천이되어야 하므로 시간지연이 있게 된다.

2.2 인터럽트기반 다중 유한상태기계 모델링과 마이크로컨트롤러 동작

최근 다양한 주변장치를 내장한 단일칩 마이크로컨트롤러는 많은 하드웨어 인터럽트 기능을 갖고 있어 하드웨어 구성이 간단하고 프로그래밍으로 구현할 수 있어 편리하다. 이런 단일칩 마이크로컨트롤러 상에서 임베디드시스템 작업을 하드웨어 설계방법에 이용되는 FSMD 구조를 적용하여 다중 유한상태기계 모형으로 구현할 수 있다. 마이크로컨트롤러가 시작되면 실행되는 주프로그램을 FSMD 구조의 주상태기계로 대응시키고 인터럽트 발생으로 동작하는 인터럽트 서비스루틴을 FSMD 구조의 부상태기계로 대응시킨다. 임베디드시스템을 주상태기계와 부상태기계로 동작하는 다중 유한상태기계 작업으로 분할 구성하고 유한상태기계 사이는 저장 변수로 인터페이스하여 모델링할 수 있다. 그림 5는 그림 1의 임베디드시스템을 다중 유한상태기계로 모델링한 것이다. 주상태기계는 시스템 동작을 총괄 제어하는 동기식 유한상태기계로 설계하고, 인터럽트 서비스루틴을 기반으로 하는 부상태기계는 주변장치의 신호발생에 의한 비동기식 유한상태기계로 설계한다. 부상태기계는 인터럽트 서비스루틴을 기반으로 설계되므로 제안되는 다중 유한상태기계 모형은 인터럽트에 신속히 반응할 수 있어 하드웨어 의존적인 임베디드시스템에 효과적으로 적용할 수 있다.

IC 설계에서 많이 사용하는 ASM 차트로 유한상태기계를 구체적으로 기술하여 임베디드시스템을 다중 유한상태기계로 모델링할 수 있다. ASM 차트로 설계된 다중 유한상태의 임베디드시스템은 C-언어 프로그램으로 쉽게 변환될 수 있다. 주상태기계 혹은 부상태기계의 유한상태기계는 상태를 천이하면서 주변장치 혹은 메모리 저장 변수와 데이터를 입출력하며 동작한다. 주상태기계와 부상태기계 사이의 실행흐름은 주변장치에 의한 하드웨어 인터럽트, 소프트웨어 인터럽트 혹은 타이머 인터럽트에 의해 스위칭된다. 일반 운영체제 콘텍스트 스위칭에서는 복귀시킬 태스크에 대한 정보를 저장하고 복원해야 하지만, 제안하는 다중 유한상태기계 모델에서는 운영체제를 사용하지 않고 인터럽트 발생에 의한 하드웨어 흐름제어로 처리된다. 그림 3의 FSMD 구조로 모델링 되는 유한상태기계는 상태변수 Q 만의 정보를 가지므로, C-언어로 구현할 때 상태변수를 전역변수 혹은 정적변수로 선언하면 별도의 저장과 복원없이 스위칭된다. 다만, 인터럽트에 의해 스위칭될

표 1. FSMD 구성요소와 스위칭에 대한 마이크로컨트롤러의 대응 동작

유한상태기계 입출력신호와 스위칭	마이크로컨트롤러 대응 동작
Q	• 상태기계에 선언되는 상태변수
q0	• 상태변수 초기화
I	• 내외부장치 신호
V	• 메모리에 선언되는 변수 - 유한상태기계에서만 참조하는 변수 - 유한상태기계 사이의 인터페이스 변수
O	• 내외부장치 출력신호
f	• 현재상태에서 내외부장치 신호와 저장변수를 읽어 다음상태를 결정한 후 상태변수에 쓰기
h	• 현재상태에서 내외부장치 신호와 저장변수를 읽어 연산한 결과를 내외부장치에 출력하거나 저장변수에 쓰기
주상태기계에서 부상태기계로 스위칭	• 인터럽트에 의해 부상태기계 실행 - 주변장치의 하드웨어 인터럽트 - 소프트웨어 인터럽트 - 주기적인 타이머 인터럽트
부상태기계에서 주상태기계로 스위칭	• 인터럽트 서비스루틴 종료

때 컴파일러의 최적화로 결정된 레지스터가 자동으로 저장되고 복원된다. 표 1은 그림 5와 같이 다중 유한상태기계로 모델링 되는 임베디드시스템에서 유한상태기계의 입출력 신호와 스위칭에 대응되는 마이크로컨트롤러의 동작을 나타낸다. 부상태기계는 인터럽트에 의해 시작되므로 ASM 차트로 설계된 부상태기계의 원활한 동작을 위한 설계규칙이 필요하다.

2.3 주상태기계와 부상태기계 ASM 차트 설계

그림 5에서 주상태기계는 마이크로컨트롤러 내부와 외부의 변수 메모리 그리고 내부장치와 외부장치에 연결된 입출력 신호와 함께 그림 3의 FSMD 구성을 갖는다. 주상태기계는 인터럽트 서비스루틴이 실행되지 않을 경우 항상 프로그램 실행 제어권을 갖게 되므로 반복적으로 상태천이 개시를 위한 조건변수를 검사한다. 주상태기계는 여러 개의 부상태기계를 총괄하도록 설계한다. 부상태기계는 아래의 규칙에 따라 인터럽트가 발생했을 때 상태천이 개시를 위한 조건변수를 검사한다. 부상태기계는 주상태기계 혹은 다른 부상태기계와 변수를 이용하여 인터페이스하면서 하드웨어 의존적인 작업을 담당하도록 설계한다. 인터럽트에 의해 스위칭되는 부상태기계는 아래 규칙에 따라 ASM 차트로 설계한다.

(규칙 1) 부상태기계는 인터럽트 발생에 의해 스위칭되어 실행한다. 인터럽트는 하드웨어 인터럽트와 소프트웨어 인터럽트를 사용한다.

하드웨어 인터럽트는 타이머를 포함한 주변장치 작동 결과에 따라 발생하여 하드웨어와 관계된 변수를 갱신하고 부상태기계가 실행된다. 소프트웨어 인터럽트는 주상태기계에서 부상태기계의 조건변수를 갱신하고 부상태기계로 스위칭이 필요할 때 사용한다. 부상태기계가 실행중일 때 부상태기계가 중복실행되지 않도록 인터럽트 발생을 비활성화 시킨다. 주기적으로 부상태기계로의 스위칭이 필요한 경우 하드웨어 타이머 인터럽트를 사용한다.

(규칙 2) 상태천이 개시를 결정하는 조건변수는 인터럽트가 발생될 때 검사되고, 한 번의 인터럽트로 한 번의 상태천이를 한다.

인터럽트 서비스루틴 진입 후 상태천이 개시 조건변수가 만족되면 한 번의 상태천이만 수행한 후 인터럽트 서비스루틴을 종료하고 주상태기계로 복귀한다. 그림 6은 ASM 차트로 표현된 부상태기계다. 상태천이 개시를 위해 최초로 검사되는 조건변수1, 조

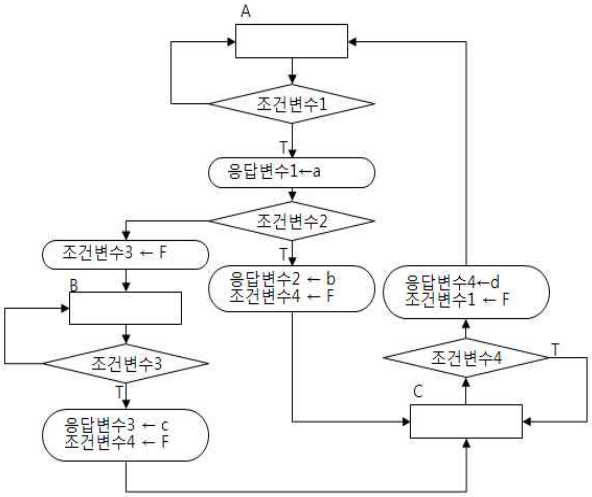


그림 6. ASM 차트로 표현된 부상태기계

건변수3, 조건변수4는 인터럽트와 연동되어 갱신되어야 한다. 인터럽트와 연동되어 갱신되는 경우는 내장된 하드웨어 발생으로 갱신되는 조건변수, 소프트웨어 인터럽트 발생전 갱신되는 조건변수, 주기적인 타이머 인터럽트 발생 전에 갱신되는 조건변수가 있을 수 있다. 상태 A에서 검사되는 조건변수2는 조건변수1 뒤에 검사되므로 인터럽트와 상관없이 갱신되어도 된다.

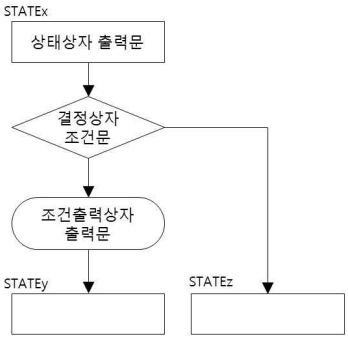
(규칙 3) 상태천이 개시 조건을 만족하지 못하는 경로는 현재상태로 향하게 하고 조건출력상자를 놓지 않는다. 상태상자에서는 출력동작을 하지 않는다.

인터럽트가 발생하지 않으면 (규칙 2)에 의해 상태천이 개시를 위한 조건변수를 검사할 수 없어 천이조건을 만족하지 못하므로 천이조건이 만족되지 않는 상태천이 방향은 현재 상태로 유지한다. 이 때 출력동작도 없어야 하므로 상태천이 개시 조건을 만족하지 못하는 경로에는 조건출력상자를 사용하지 않고, 상태상자 안에도 출력동작을 하지 않는다. 그림 6의 상태 A에서 상태 B로 천이될 때 조건변수3을 (규칙 3)에 맞게 설정하고 인터럽트 서비스루틴을 종료한다. 인터럽트가 발생할 때 까지 상태 B에서 자신의 현재상태 B로 출력없이 천이되는 상황을 유지시킨다. 인터럽트가 발생하면 부상태기계는 상태천이 개시 조건변수3을 검사하여 상태 B에서 다른 상태로 천이될 수 있다.

(규칙 4) 주상태기계와 부상태기계는 저장 변수로 요청응답 인터페이스 한다.

부상태기계가 갱신한 인터페이스 변수는 부상태기계가 종료되면 주상태기계가 실행되면서 해당 변수를 사용한다. 주상태기계가 갱신한 인터페이스 변수는 부상태기계의 상태천이 개시 조건변수인 경우와 그렇지 않은 경우가 있다. 상태천이 개시 조건변수가 아닌 경우 다른 조건변수에 의해 부상태기계가 상태를 천이할 때 인터페이스 변수가 사용된다. 상태천이 개시 조건변수인 경우 소프트웨어 인터럽트로 부상태기계를 실행시키거나 타이머 인터럽트에서 주기적으로 부상태기계를 실행시켜 갱신된 조건변수를 검사하게 한다.

(규칙 1)에 따라 설계되는 다중상태기계는 인터럽트 서비스루틴 실행과 함께 부상태기계로 진입되면서 스위칭되므로 별도의 운영체제를 사용하지 않는다. 인터럽트 발생에 의해 진입된 부상태기계에서 검사되는 조건변수에 의해 부상태기계는 상태를 천이하면서 필요한 작업을 즉시 수행할 수 있어 다중 유한상태기계 상에서 인터럽트에 신속하게 반응할 수 있다. 주변장치가 많이 사용되는 하드웨어 의존적인 임베디드시스템에 적용하여 신속하게 작업을 수행할 수 있다. (규칙 2)에 따라 설계되는 상태천이 개시 조건변수는 부상태기계가 실행되어야 검사될 수 있다. ASM 차트에서 상태천이 개시 조건변수는 상태상자에서 처음 분기되는 결정상자에서 사용된다. 하드웨어 인터럽트가 발생하면서 상태천이 개시 조건변수를 갱신하고 곧바로 부상태기계를 실행시킨다. 소프트웨어 인터럽트는 주상태기계에서 갱신하는 인터페이스 변수가 부상태기계의 상태천이 개시 조건변수 일 때 사용한다. 주상태기계는 실행시킬 부상태기계를 지정하고 소프트웨어 인터럽트를 유발시켜 그 부상태기계를 실행시킨다. 다른 방법으로 타이머 인터럽트에서 주기적으로 부상태기계를 실행시켜 갱신되는 상태천이 개시 조건변수를 검사하게 할 수 있다. (규칙 3)에 따라 설계되는 부상태기계는 인터럽트가 없으면 실행되지 않으므로 상태상자에서는 출력동작을 않게 하고, 상태천이 개시 조건변수를 만족하지 않은 경로는 조건출력상자 없이 현재상태로 향하게 한다. (규칙 4)에 따라 주상태기계와 부상태기계 사이에 인터페이스하며 작업을 수행할 때 프로그램 제어권을 많이 갖는 주상태기계에서는 계산



(a) ASM 차트

```
while(1){
   switch(state){
     ...
     case STATEx:
        상태상자 출력문
        if( 조건상자 조건문){
           조건출력상자 출력문
           state = STATEy;
        }else
           state = STATEz;
        break;
     ...
   }
}
```

(b) 주상태기계로 변환된 프로그램

```
switch(state){
  ...
  case STATEx:
     상태상자 출력문
     if( 조건상자 조건문){
        조건출력상자 출력문
        state = STATEy;
     }else
        state = STATEz;
     break;
 ...
}
```

(c) 부상태기계로 변환된 프로그램

그림 7. ASM 차트의 C언어 프로그램 변환

량이 많은 작업을 실행하게 하고, 부상태기계에서는 짧게 수행되는 하드웨어 의존적인 작업을 수행하게 한다.

2.4 ASM 차트를 이용한 다중 유한상태기계 모형의 프로그램 변환

다중 유한상태기계로 모델링 되는 임베디드시스템이 구현되기 위해서는 프로그램으로 변환하는 과정이 필요하다. ASM 차트는 VLSI 하드웨어를 구현할 때 유한상태기계를 그래픽하게 기술하는 방법으로 많이 사용되고 있다[14]. 본 논문에서 제안하는 다중 유한상태기계의 각 유한상태기계를 ASM 차트로 구체적으로 기술하여 프로그램으로 변환될 수 있다. ASM 차트는 상태상자, 결정상자, 조건출력상자의 요소로 구성되므로 각 구성요소를 대응되는 C언어의 구문으로 치환하면 프로그램으로 쉽게 변환된다.

리셋과 함께 시작되는 주상태기계를 기술한 ASM 차트는 C언어의 while(1) 무한 반복문 안에서 상태상자는 switch-case 문에서 상태변수로 분기시키고, 조건상자는 분기된 상태 위치에서 놓여지는 if 문에 해당된다. 조건출력상자 명령은 if 조건식의 참 혹은 거짓에 따라 실행되는 명령에 해당된다. 다음 상태로 천이되는 화살표는 if 문의 마지막 위치에서 상태변수에 새로운 상태 값을 할당하는 동작으로 대응된다[13]. 그림 7(a)에 해당되는 ASM 차트는 주상태기계인 경우 그림 7(b)와 같이 변환되고 부상태기계인 경우 그림 7(c)와 같이 C언어 프로그램으로 변환된다.

주상태기계의 while(1) 반복문이 1회 반복하는 것은 FSMD 내의 동기식 유한상태기계의 1회의 동기 클록 발생과 대응된다. 부상태기계가 실행되면 (규칙 2)와 같이 한 번의 상태천이만 되어야 하므로 그림 7(c)에서 반복문을 갖지 않는다. 인터럽트 발생은 비동기 순차회로에 대응되는 부상태기계의 상태천이 개시 신호가 활성화 된 것에 해당된다. 인터럽트가 발생하면 주상태기계의 실행이 잠시 정지하고 인터럽트 서비스루틴이 실행되면서 부상태기계로 스위칭된다. 하드웨어 인터럽트가 발생되어 HW1 번호를 갖는 부상태기계로 스위칭하는 경우는 그림 8(a)와 같이 인터럽트 서비스루틴에서 주변장치와 관련된 변수를 갱신시킨 후 해당 부상태기계를 호출한다. 주상태기계에서 부상태기계로 스위칭시킬 때는 주상태기계가 스위칭을 원하는 부상태기계 번호를 switch_ssm 변수에 기록한 후 소프트웨어 인터럽트를 유발한다. 그림 8(b)의 소프트웨어 인터럽트 서비스루틴은 기록된 switch_ssm 번호의 부상태기계를 실행시킨다.

부상태기계의 상태천이 개시 조건변수가 인터럽트에 의해 갱신되지 않는 경우 타이머 인터럽트에서

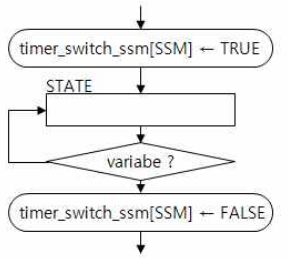
```
ISR(HW1_vect)
{
    hw1_variable = DEVICE_REG;
    sub_state_machine[HW1]();
}
```

(a) 하드웨어 인터럽트에 의한 스위칭

```
ISR(SW_vect)
{
    sub_state_machine[switch_ssm]();
}
```

(b) 소프트웨어 인터럽트에 의한 스위칭

그림 8. 하드웨어 인터럽트와 소프트웨어 인터럽트에 의한 부상태기계로 스위칭



(a) SSM 아이디의 부상태기계 실행 설정

```
ISR(TIMER_vect)
{
    ...
    if( timer_switch_ssm[SSM] == TRUE)
        sub_state_machine[SSM]();
    ...
}
```

(b) 타이머 인터럽트에서 활성화된 부상태기계 실행

그림 9. 타이머 인터럽트에 의한 주기적인 부상태기계로 스위칭

부상태기계를 주기적으로 실행시켜 변수의 갱신을 검사할 수 있다. 그림 9는 timer_switch_ssm[] 배열을 사용하여 부상태기계에서 주기적인 실행여부를 설정하고 타이머 인터럽트에서 실행시키는 방법을 나나내고 있다. 그림 9(a)는 부상태기계의 조건결정 상자에서 SSM 아이디를 갖는 부상태기계 실행을 위해 timer_switch_ssm[SSM]을 활성화로 시킨다. 상태천이 개시 조건변수가 만족되어 상태 천이가 시작되면 타이머 인터럽트에서 부상태기계를 실행시키지 않아도 되므로 조건출력상자에서 비활성화 시킨다. 그림 9(b)와 같이 주기적으로 실행되는 타이머 인터럽트 서비스루틴에서는 부상태기계 실행을 판단하는 배열의 timer_switch_ssm[SSM]값이 활성화 되어있으면 해당 부상태기계를 실행시킨다.

3. 적용사례 및 측정결과

그림 10은 2층을 오가는 리프트 모형시스템에 대하여 제안한 인터럽트 기반의 다중 상태기계 모형으로 구현하기 위한 시스템 구성도다. 호스트는 주상태기계에 직렬통신으로 연결되어 있어 리프트를 1층 혹은 2층으로 이동하는 명령과 리프트 상태를 관찰하기 위한 명령을 내릴 수 있다. 리프트 이동명령인 경우 내장된 PWM 파형발생장치로 모터의 상승 혹은 하강을 위한 동작을 하고, 각 층의 부상태기계에서 리프트 도착을 감지하여 필요한 동작을 하도록 요청한다. 리프트 관찰 명령인 경우 부상태기계에서 갱신한 인터페이스 변수를 이용하여 리프트 상태를 호스트에 전달한다. 각 부상태기계는 주상태기계의 요청에 따라 1층과 2층 센서에서 외부인터럽트로 리프트 도착을 감지하면 즉시 모터를 정지 시키고 문을 여닫으며 진행 중인 작업 상황을 인터페이스 변수로 주상태기계에 알린다. 문을 여닫을 때 필요한 지연시간은 타이머 인터럽트를 사용한다.

그림 11(a)는 주상태기계에 대한 ASM 차트를 나태내고 있다. 주상태기계가 FLOOR1 상태일 때 1층 부상태기계가 req_rep[F1] 인터페이스 변수를 WAITING에서 다르게 갱신했다면 리프트가 1층에

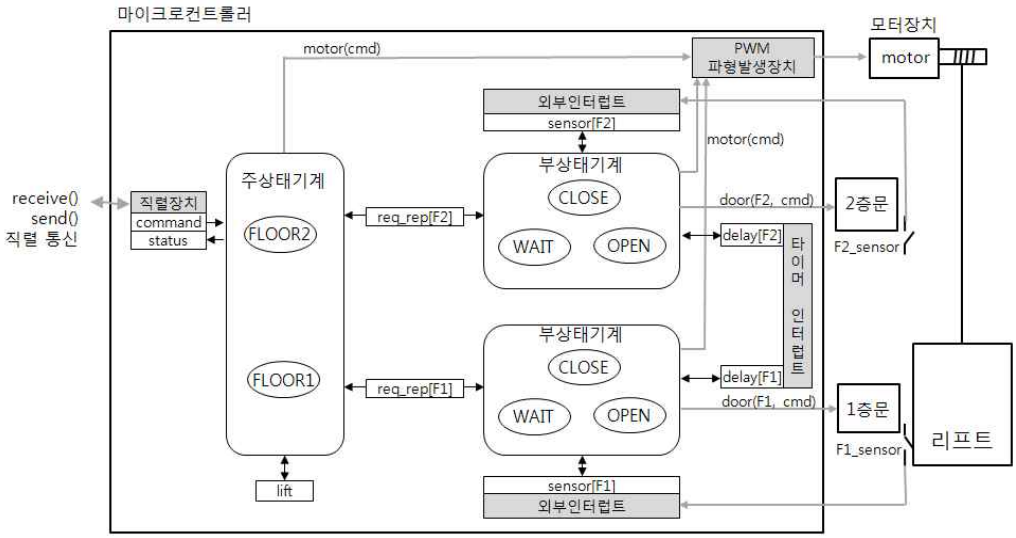
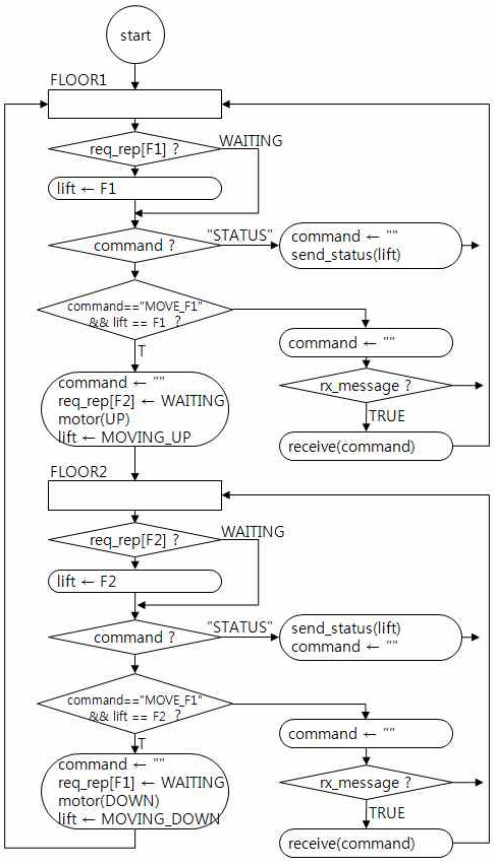


그림 10. 인터럽트 기반의 다중 상태기계 모형으로 설계된 리프트 시스템 구성도

도착된 경우이므로 lift를 F1으로 갱신한다. 호스트에서 수신된 명령이 "STATUS" 일 때는 주상태기계의 저장변수 lift에 있는 리프트 상태정보를 send_status() 함수로 호스트로 전달한다. 리프트가 1층에 있는 상황에서 호스트의 "MOVE_F2" 명령을 받으면 인터페이스 변수 req_rep[F2]를 WAITING으로 갱신시켜 2층 부상태기계가 리프트의 도착을 준비하게 한다. 그리고, 모터 상승명령 motor(UP)을 실행하고 리프트의 저장변수 lift를 MOVING_UP으로 갱신한 후 FLOOR2 상태로 천이한다. FLOOR2 상태에서도 FLOOR1 상태로 천이되기 까지 같은 방법이 사용된다. 그림 11(b)는 ASM차트를 while(1) 문 안에서 주기적으로 상태천이가 되게 하고 상태상자, 결정상자, 조건출력상자를 C-언어 구문으로 대응시켜 프로그램으로 변환한 것이다. 주상태기계는 인터럽트 서비스루틴이 종료하면 프로그램 실행 제어권을 갖게 되어 반복적으로 상태 천이 개시 조건변수를 검사한다.

그림 12(a)의 2층 부상태기계 ASM 차트는 CLOSE 상태에서 주상태기계가 갱신하는 상태천이 개시 조건변수 req_rep[F2]를 검사한다. 이를 위해 타이머 인터럽트가 주기적으로 부상태기계를 실행시키는 방법을 사용하고 있다. 부상태기계가 CLOSE 상태로 천이하기 직전 조건출력상자에서 timer_switch_ssm[F2] 변수를 활성화 시켜 타이머 인터럽트가 주기적으로 2층 부상태기계를 실행시킬 수 있도록 설정한다. CLOSE 상태에서 상태천이가 개시되면 timer_switch_ssm[F2] 변수를 비활성화 시켜 주기적인 부상태기계 실행을 중지시킨다. WAIT 상태에서 리프트 도착으로 인터럽트가 발생하면 상태천이가 시작된다. 조건출력상자에서 신속히 motor(STOP) 명령으로 리프트를 정지시키고 door(F2, OPEN) 명령으로 문을 연다. delay[F2] 변수에 지연시간을 설정하고 인터페이스 변수에 상황을 기록한다. OPEN 상태에서는 타이머 인터럽트가 지연시간이 경과한 후 부상태기계를 실행시키므로 상태천이가 되면서



(a) 주상태기계 ASM 차트

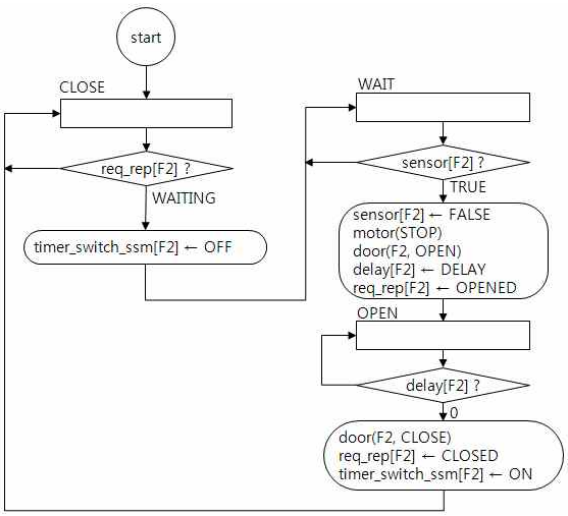
```
while(1){
        switch(main_state){
        case FLOOR1:
                if( req_rep[F1] != WAITING)
                        lift = F1;
                        if( !strcmp(command, "STATUS") ){
                        command[0] = '\0';
                        send_status(lift);
                }else if( !strcmp(command, "MOVE_F2") && lift==F1){
                        command[0] = '\0';
                        req_rep[F2] = WAITING;
                        motor(UP);
                        main_state = FLOOR2;
                }else{
                        command[0] = '\0';
                        if( rx_message() )
                                receive(command);
                }
                break;

        case FLOOR2:
                if( req_rep[F2] != WAITING)
                        lift = F2;

                if( !strcmp(command, "STATUS") ){
                        command[0] = '\0';
                        send_status(lift);
                }else if( !strcmp(command, "MOVE_F1") && lift==F2){
                        command[0] = '\0';
                        req_rep[F1] = WAITING;
                        motor(DOWN);
                        main_state = FLOOR1;
                }else{
                        command[0] = '\0';
                        if( rx_message() )
                                receive(command);
                }
                break;
        }
}
```

(b) C-코드로 변환된 ASM 차트

그림 11, 주상태기계의 ASM 차트와 변환된 프로그램

문을 닫고 인터페이스 변수에 상황을 기록한다. 다음 상태에서는 주상태기계가 갱신하는 req_rep[F2]가 상태천이 개시 조건변수로 되므로 타이머 카운터가 주기적으로 부상태기계를 실행시킬 수 있도록 timer_switch_ssm[F2]를 활성화 한다. 그림 12(b)는 ASM 차트의 상태상자, 결정상자, 조건출력상자를 대응되는 C-언어 구문으로 변환한 프로그램이다.

그림 13(a)는 리프트의 도착을 감지하는 1층과 2층의 센서에 의한 인터럽트 서비스루틴이다. 인터럽트 서비스루틴에서는 sensor[] 변수를 갱신한 후 곧바로 부상태기계 sub_state_machine[]()를 실행시키므로 상태기계에서 필요한 작업을 신속히 수행할 수 있다. 그림 13(b)는 타이머 인터럽트 서비스루틴이고 부상태기계 실행 결정을 위해 delay_switch_ssm[]와 timer_switch_ssm[]를 사용하고 있다. delay_switch_ssm[]은 부상태계가 시간지연을 delay[] 변수로 요청한 경우 시간지연 경과 여부에 따라 활성화 시켜 부상태기계 실행 여부를 판단한다. timer_switch_ssm[] 변수는 부상태기계의 다음 상태에서 상태천이 개시 조건변수가 인터럽트에 의해 갱신되지 않은 경우 사용된다. 부상태기계의 CLOSE 상태에서 req_rep[] 변수가 상태천이 개시 조건변수면서



(a) 부상태기계 ASM 차트

```
void   ssm_F2(void)
{
       switch( ssm_state[F2] ){
       case CLOSE:
              if( req_rep[F2] == WAITING){
                     timer_switch_ssm[F2] = OFF;
                     ssm_state[F2] = WAIT;
              }
              break;

       case WAIT:
              if(sensor[F2] == TRUE){
                     sensor[F2] = FALSE;
                     motor(STOP);
                     door(F2, OPEN);
                     delay[F2] = DELAY;
                     req_rep[F2] = OPENED;
                     ssm_state[F2] = OPEN;
              }
              break;

       case OPEN:
              if( delay[F2] == 0){
                     door(F2, CLOSE);
                     req_rep[F2] = CLOSED;
                     timer_switch_ssm[F2] = ON;
                     ssm_state[F2] = CLOSE;
              }
              break;
       }
}
```

(b) C-코드로 변환된 ASM 차트

그림 12. 주상태기계의 ASM 차트와 변환된 프로그램

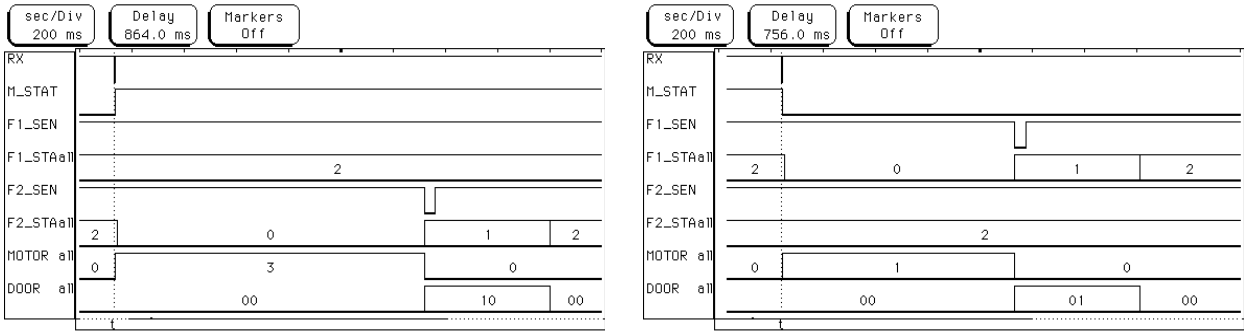
```
/////////// sensor[F2] interrupt
ISR(F2_vect)
{      sensor[F2] = TRUE;
       sub_state_machine[F2]();
}

/////////// sensor[F1] interrupt
ISR(F1_vect)
{      sensor[F1] = TRUE;
       sub_state_machine[F1]();
}
```

(a) 센서 인터럽트 서비스루틴

```
ISR(TIMER_vect)
{   int       ssm;

    for(ssm=0; ssm<MAX_SSM; ssm++){
       if( delay[ssm] > 0){
          delay[ssm]--;
          delay_switch_ssm[ssm] = ( delay[ssm] == 0) ? ON : OFF;
       }

       if( timer_switch_ssm[ssm] | delay_switch_ssm[ssm] )
          sub_state_machine[ssm]();
    }
}
```

(b) 타이머 인터럽트 서비스루틴

그림 13. 인터럽트 서비스루틴과 부상태기계로 스위칭



(a) "MOVE_F1" 명령에 대한 동작 신호 (b) "MOVE_F2" 명령에 대한 동작 신호

그림 14. 인터럽트를 이용한 다중 유한상태기계로 구현된 리프트 시스템의 동작 신호

주상태기계에 의해 WAITING으로 갱신되므로 주기적으로 타이머 인터럽트에 의해 스위칭되어 상태천이 개시 여부를 검사하게 된다.

그림 14는 리프트 모형시스템을 단일칩 마이크로컨트롤러 ATmega128로 구현한 후 동작되는 상황의 입출력 신호와 상태를 측정한 것이다. 그림 14(a)에서 주상태기계(M_STAT)는 호스트로부터 RX 신호로 "MOVE_F2" 명령을 수신하자 FLOOR1(0)인 상태에서 모터(MOTOR)를 상승회전(3) 시키고 FLOOR2(1) 상태로 갱신되고 있다. 이 때 주상태기계가 req_rep[F2]에 기록한 WAITING을 보고 2층 부상태기계(F2_STA)는 CLOSE(2) 상태에서 WAIT(0) 상태로 천이되되고 있다. 중후반부에 2층 도착센서(F2_SEN)가 감지되자 2층 부상태기계(F2_STA)는 모터를 정지(0)시키고 문(DOOR)를 열고(10) 시간지연 후에 닫고(00)있다. 문을 닫은 후 2층 부상태기계는 CLOSE(2) 상태로 천이되고 있다. 그림 14(b)는 2층 상태에서 RX 신호로 "MOVE_F1" 신호를 수신한 후 동작되는 주상태기계와 부상태기계 그리고 입출력 신호의 움직임을 측정한 것으로서 "MOVE_F2"와 유사하게 작동됨을 알 수 있다.

4. 결 론

본 논문에서는 단일칩 마이크로컨트롤러를 사용하는 임베디드시스템을 인터럽트를 사용한 다중 유한상태기계로 모델링한 후 구현하는 방법을 소개하였다. 하드웨어 설계 과정에 많이 사용되는 FSMD 구조를 적용하여 인터럽트를 사용하는 임베디드시스템을 다중 유한상태기계로 모델링 할 수 있다. 다중 유한상태기계는 주상태기계에 대한 ASM 차트와 부상태기계에 대한 ASM 차트로 구성된다. 주상태기계는 주프로그램으로, 부상태기계는 하드웨어 의존적인 인터럽트 서비스루틴으로 대응시킴으로서 인터럽트 발생 즉시 부상태기계로 스위칭되어 사건발생에 신속하게 응답할 수 있는 장점을 갖는다. 유한상태기계 사이의 스위칭은 인터럽트에 의한 하드웨어 흐름제어를 사용하므로 부가적인 운영체제를 탑재하지 않고 단순하게 구현할 수 있다. ASM 차트로 기술된 유한상태기계 사이는 요청과 응답 인터페이스 변수로 정보를 교환한다. 주상태기계 ASM 차트는 동기식 유한상태기계에 대응되고 무한 반복 while(1) 문 내부에서 상태상자, 결정상자, 조건출력상자 동작을 치환 수준의 C언어로 쉽게 변환할 수 있다. 부상태기계 ASM 차트는 비동기 유한상태기계로 대응되어 인터럽트 발생마다 스위칭므로 반복문 없이 변환한다. 제안된 방법은 인터럽트 기능을 갖는 다양한 주변장치를 내장한 단일칩 마이크로프로세서를 사용하는 임베디디시스템 구현에 효과적이다. 인터럽트를 이용한 다중 유한상태기계 모델은 ASM 차트로 임베디드시스템을 구체적으로 표현할 수 있고 사건에 신속하게 응답할 수 있으며 간단한 변환과정을 통해 프로그램으로 쉽게 구현될 수 있다.

참 고 문 헌

[1] G.N.T. Huong, Y.Na, and S.W. Kim, "Applying Frame Layout to Hardware Design in FPGA for Seamless Support of Cross Calls in CPU- FPGA Coupling Architecture," *Microprocessors and Microsystems,* Vol. 35, Issue 5, pp. 462-472, 2011.

[2] Atmel Cooperation , *8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash*, Datasheet, 2007.

[3] W. Gong, M. Qu, X. Wu, and P. Ma, "The Verification of Structure Identification Algorithm and Error Detection Strategies for Structured Flowchart," *International Conference on Future Computer Supported Education*, pp. 880-887, 2012.

[4] K. Charntaweekhun and S. Wangsiripitak, "Visual Programming using Flowchart," *Int. Symp. on Digital Object Identifier*, pp. 1062-1065, 2006.

[5] 배정호, 채흥석, "계약 기반 정형 명세로부터 이해도 높은 상태 기계 자동 구축 기법," 정보과학회 논문지(소프트웨어 및 응용), 제39권, 제9호, pp. 734-749, 2012.

[6] 정소영, 노혜민, 유철중, "SysML 상태 기계 다이어그램 기반 테스트 케이스 생성 기법," 한국정보기술학회 논문지, 제9권, 제3호, pp. 31-39, 2011.

[7] 김기훈, 최현택, 이종무, "유한 상태 기계를 이용한 자율 무인 기뢰 처리기의 다중센서 융합기반 수중 유도 항법시스템 설계," 전자공학회 논문지, 제47권 SC편, 제 6 호, pp. 373- 382, 2010.

[8] J. Cecilio, P. Furtado, "A State-Machine Model for Reliability Eliciting over Wireless Sensor and Actuator Networks," *Procedia Computer Science,* Vol. 10, pp. 422-431, 2012.

[9] 전태건, 김창수, "임베디드 시스템에서 실시간성과 결함허용을 보장하는 스케줄러 설계," 멀티미디어학회논문지 제14권 제1호, pp. 76-84, 2011.

[10] M.H. Samadzadeh and L.E. Garalnabi, "Hardware/Software Cost Analysis of Interrupt Processing Strategies," *IEEE Micro*, Vol. 21, Issue. 3, pp. 69-76, 2001.

[11] D.D. Gajski, N.D. Dutt, A.C.. Wu, and S.Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer, Boston, 1992.

[12] C. Karfa, D. Sarkar, and C. Jandal, "Verification of Datapath and Controller Generation Phase in High-Level Synthesis of Digital Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 29, No. 3, pp. 479-492, 2010.

[13] 이상설, 마이크로컨트롤러 AVR ATmega 128, 한빛미디어, 서울, 2011.

[14] D.D. Givone, *Digital Principles and Design*, McGraw-Hil, New York, 2002.

이 상 설

1984년 2월 고려대학교 전자공학과 공학사
1989년 2월 한국과학기술원 전기 및전자공학과 공학석사.
1994년 2월 한국과학기술원 전기 및전자공학과 공학박사

1994년 3월~현재 원광대학교 전기·정보통신공학부 교수
관심분야 : 임베디드시스템, 인간 컴퓨터 상호작용