한국산업정보학회논문지 제13권 제2호(2008. 6)

임베디드 시스템을 위한 소프트웨어 시험 환경 구축

(A Software Test Environment for Embedded Systems)

강병도*
(Byeong-do Kang)

요 약 최근에 임베디드 시스템에 첨가되는 기능들은 대부분의 응용분야에서 점점 다양화되고 복잡해지고 있다. 임베디드 소프트웨어가 점점 더 많은 기능을 포함하게 되고, 그 크기도 점점 커지고 있다. 따라서 소프트웨어 개발자들이 제시간에 개발기간을 맞추는 것이 점점 더 어려워지고 있다. 그러므로 임베디드 소프트웨어 개발에 적합한 설계 및 시험 기법이 요구된다. 이 논문에서 우리는 임베디드 소프트웨어를 위한 소프트웨어 구조 형태를 제안한다. 이 구조는 재사용 가능한 기능의 조립을 촉진시키며 개발자가 개발기간을 단축하는 데 도움이 된다. 또한 우리는 목적시스템에서 동작하는 임베디드 소프트웨어를 시험하는 기법과 도구를 제안한다.

핵심주제어 : 임베디드 소프트웨어, 임베디드 소프트웨어 시험 기법, 목표시스템 시험

Abstract In recent years, the functions added to an embedded system have grown various and complex in most parts of the applications. Embedded software comes to include more functions and is getting bigger. Therefore it is getting difficult for software developers to keep the development time to market on time. This leads to a need for adequate design and test technique for embedded software. In this paper, we propose a software architecture style for embedded software. It facilitates the composition of reusable functions and helps developers reduce development time. We also propose a test method and tools for embedded software running on target systems.

Key Words : Embedded Software, Embedded Software Test Method, Target System Test

1. 서 론

임베디드 시스템은 장치 내부에서 사용되는 특정 목적의 컴퓨터이다. 예를 들면, 마이크로웨이브 오븐은 임베디드 시스템을 포함하고 있는데, 이 임베디드 시스템은 패널로부터 입력을 인식하고, LCD 화면을 제어하며, 음식을 요리하는 가열 장치를 켜고 끄는 기능을 수행한다. 임베디드 시스템은 일반적으로 마이크로프로세서를 사용한다. 이 마이크로프로세서는 하나의 장치위에 컴퓨터의 여러 기능을 포함하고 있다. 모토롤라나 인텔 회사가 잘 알려진 마이크로프로세서를 생산하고 있다. 과거에는 임베디드 시스템을 개발할 때 소프트웨어 개발자들은 호스트(Host) 운영체제 없이 마이크로프로세서 하드웨어에 직접 프로그램 하였다.

그러나 임베디드 시스템이 점차 지능화되고 네트워크 기능과 같은 더 일반적인 컴퓨터 기능들을 포함하기 시작함에 따라 호스트 운영체제를 사용하는 것이 일반화되었다. 리눅스와 임베디드용 윈도우즈가 임베디드 시스템을 구현하는 데 이용되는 운영체제의 예이다[1].

최근에는 임베디드 시스템에 첨가되는 기능들은 대부분의 응용분야에서 점점 다양화되고 복잡해지고 있다. 이에 따라 개발비와 개발기간이 증가하고 있다. 이러한 문제는 개발자들이 차후에 발생할 요구사항 변경을 고려하여 유연한 구현을 하도록

* 대구대학교 컴퓨터 · IT공학부 교수

만들었다. 변경되는 기능을 하드웨어로 구현하는 것이 소프트웨어로 구현하는 것보다 유연성이 적다고 생각하기 때문에, 임베디드 소프트웨어가 점점 더 많은 기능을 포함하게 되고, 그 크기도 점점 커지고 있다. 따라서 소프트웨어 개발자들이 제시간에 개발기간을 맞추는 것이 점점 더 어려워지고 있다. 임베디드 소프트웨어 개발비는 증가하는 기능의 복잡도 때문에 크게 증가하고 있다. 그러므로 임베디드 소프트웨어 개발에 적합한 설계 및 시험 기법이 요구된다.

이 논문에서 우리는 임베디드 소프트웨어를 위한 설계 기법과 시험기법을 제안한다. 2장에서는 관련 연구를 간단히 소개하고, 3장에서는 임베디드 소프트웨어를 위한 설계 및 시험 기법을 제안한다. 마지막 4장에서는 요약과 함께 결론을 맺는다.

2. 관련 연구

2.1 POLIS

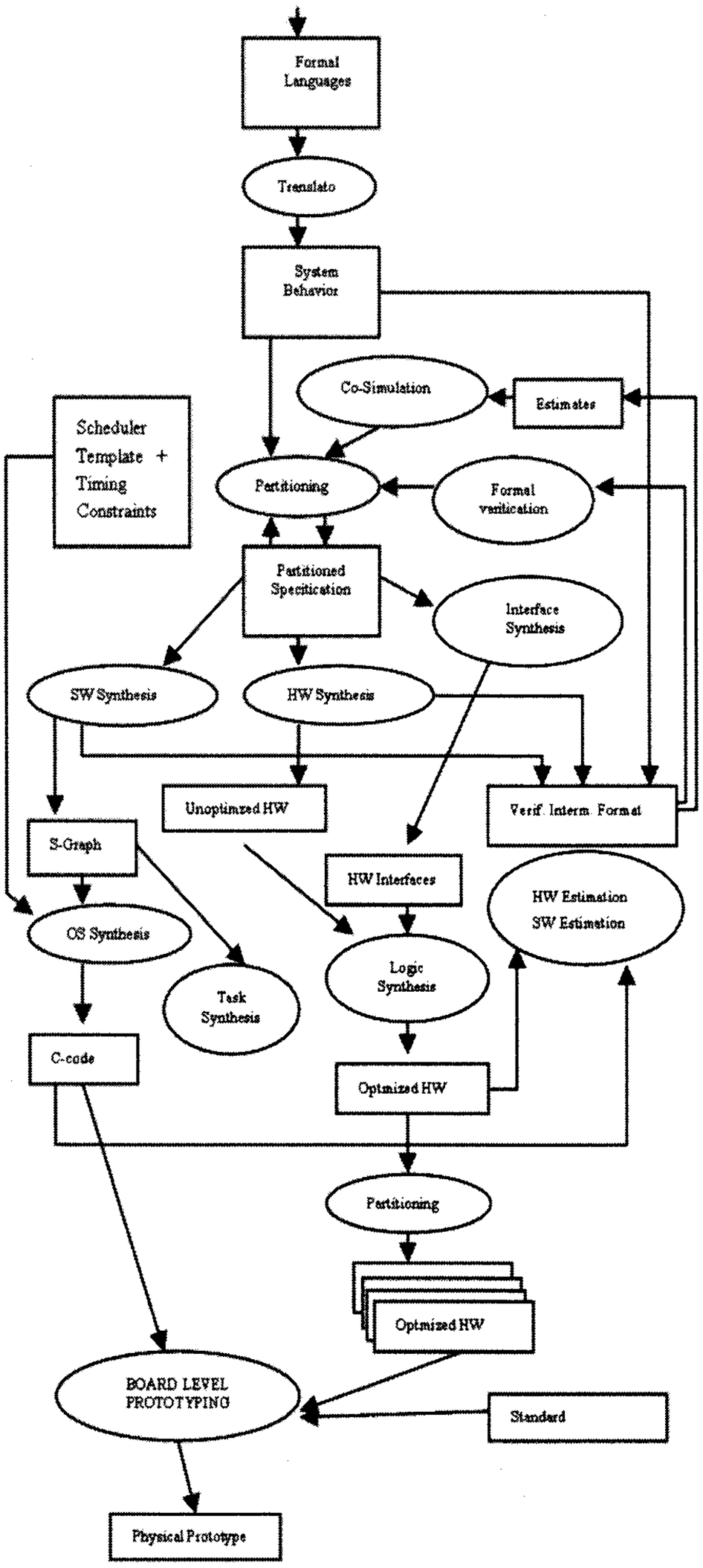
POLIS[2]는 임베디드 시스템의 통합된 하드웨어-소프트웨어 표현, 자동 합성, 검증 등이 가능한 명세 기법이다. POLIS를 이용하여 설계자들은 병행설계 유한상태 기계(CFSM: Co-design Finite State Machine)로 직접 번역이 가능한 고급언어로 명세서를 작성할 수 있다. 고전적인 유한상태 기계(FSM: Finite State Machine)처럼 CFSM은 유한개의 내부 상태로써 입력 집합을 출력 집합으로 변환한다. 이 두 모델간의 차이는 고전적 FSM의 동기적 통신 모델이 유한 반작용시간, 비제로(Non-zero) 반작용시간, 무제한 반작용시간 등으로 표현되어 CFSM 모델로 대체되는 것이다. CFSM 네트워크의 각 요소들이 모델링되는 시스템의 컴포넌트들을 묘사한다. CFSM 명세는 하드웨어나 소프트웨어 구현의 한쪽에 치우치지 않는다. POLIS에 내재된 형식 명세나 합성 방법은 FSM에 기반한 기존의 형식 검증 알고리즘으로 직접 지원되는 것이 가능하다. POLIS는 CFSM을 FSM으로 번역하는 번역기가 있으며 이에 대한 검증 기능도 갖추고 있다.

시스템 수준에서 하드웨어와 소프트웨어를 동시 시뮬레이션 하는 기능은 설계자들에게 설계 시 하드웨어와 소프트웨어의 기능 할당, CPU 선택, 스케쥴러 선택 등과 같은 결정을 하는 데 도움을 준다. 이러한 결정은 설계자의 경험에 많이 좌우된다. CFSM 명세 중 하드웨어로 구현될 부분은 논리 합성 기법으로 구현되어 적합하게 된다. 소프트웨어로 구현될 부분은 각 CFSM을 처리하는 프로시쥬어를 포함하는 소프트웨어 구조로 대응된다. 서로 다른 구현간의 인터페이스는 POLIS내에서 자동으로 합성된다. 이 인터페이스들은 합성된 구현에 내장되어 상호작용하는 회로와 소프트웨어 프로시쥬어 형태로 나타난다. (그림 1)은 POLIS를 이용한 설계 절차를 나타낸다.

2.2 UML 기반 설계 기법

UML[3]의 장점과 SDL[4]의 장점을 결합하여 임베디드 실시간 시스템을 시스템 수준에서 설계하는 기법이 있다[5]. 이 기법의 주된 개념은 하드웨어와 소프트웨어의 컴포넌트를 명세 하는 데 객체지향 기법을 사용하는 것이다.

이 기법을 이용하여 명세서를 작성하는 절차는 4단계로 구성된다. 시스템 수준의 설계 절차의 각 단계는 산출물들의 집합에 의해 구분된다. 시스템 설계와 명세서 도출을 위한 첫 번째 정형화 단계에서는 UML과 UML 모델링 과정이 사용된다. 두 번째 단계에서는 UML 2.0 에서 정의된 SDL 개념이 추가적 패턴으로 사용된다. UML과 SDL의 정적 및 동적 관점은 시스템 상호간 통신과 상호작용의 명세를 위하여 사용된다. 세 번째 상세 설계 단계에서는 모듈과 다른 요소들의 내부 명세를 기술한다. 전체 기능 명세는 상세 SDL로 기술된다. 네 번째 단계에서는 시스템의 기능을 분산하여 전개하는 목표구조를 명세 하는 것이다. 실시간 임베디드 시스템을 설계할 때 객체지향 개념을 사용하는 것은 필요할지 모르지만, 많은 시간이 소요되고 개발비가 증가되는 요인이 되기도 한다.



(그림 1) POLIS를 이용한 설계 절차[11]

3. 임베디드 소프트웨어 개발 및 시험

3.1 임베디드 소프트웨어

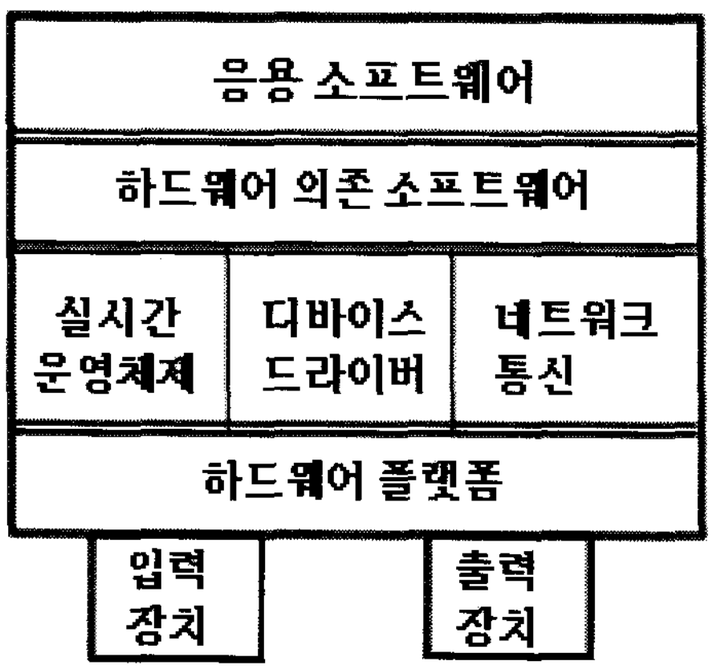
임베디드 소프트웨어를 개발하는 방법은 개인용 컴퓨터나 워크스테이션 또는 대형 컴퓨터에서 동작하는 응용소프트웨어나 시스템소프트웨어와 차이가 있다. 응용소프트웨어와 시스템소프트웨어는 일반적으로 그래픽 사용자 인터페이스를 갖춘 소프트웨어 개발환경에서 고성능의 호스트 컴퓨터(Host Computer)에서 개발된다. 반면에 임베디드 시스템이 동작하는 목표 프로세서(Target Processor)는 비용과 공간을 최소화하기 위하여 전형적으로 목적에 적합한 성능과 크기로 최소화되어 있다. 그러므로 임베디드 시스템의 목표 하드웨어(Target Hardware)는 잘 정의된 방법론을 지원하는 소프트웨어 개발 도구들이 탑재되어 동작할 여유가 없다. 그러므로 목표 시스템에서 동작하는 임베디드 프로그램은 처음에 개인용 컴퓨터나 워크스테이션 같은 호스트 컴퓨터에서 개발된다. 그 다음에 교차컴파일러(Cross-Compiler)와 링커(Linker)가 임베디드 시스템에서 실행될 목표 코드(Target Code)를 생성한다. 마지막으로 이 목표 코드가 목표 프로세서에 다운로드(Download)된다. 호스트 환경(Host Environment)과 목표 환경(Target Environment)은 사용자에게 제공하는 기능과 인터페이스가 각각 상이하다. 호스트 환경에서는 사용자가 그래픽 인터페이스를 통하여 다양한 설계 및 시험도구를 사용할 수 있지만, 목표 시스템이 동작하는 목표 환경에서 사용할 수 있는 도구들은 드물다.

(1) 임베디드 시스템의 구조

임베디드 시스템은 큰 시스템 또는 기계의 일부분으로 특성화된 컴퓨터 시스템이다. 전형적인 임베디드 시스템은 ROM에 프로그램을 저장한 마이크로프로세서 보드(Microprocessor Board)로 제작된다. 임베디드 시스템의 응용 예로서 소비자 가전제품, 통신기, 자동차 제어기, 공장 제어기 같은 것들을 들 수 있다. 이 예들은 응용 분야가 서로 다르더라도 기능 구성면에서는 공통된 구조를 가지고 있다. (그림 2)는 임베디드 시스템의 공통된 계층 구조를 나타내고 있다. 임베디드 시스템의 공통 기능 모듈로는 하드웨어 플랫폼, 하드웨어 의존 소프트웨어(Hardware-dependent Software), 응용소프트웨어, 응용프로그램 인터페이스 (Application Programming Interface) 등이 있다[6][7].

응용프로그램 인터페이스는 하드웨어 의존 소프

트웨어와 응용소프트웨어 계층사이의 통신을 위하여 필요하다. 하드웨어 의존 소프트웨어는 물리적 하드웨어와 네트워크 장치와 밀접한 관계가 있다. 실시간 운영체제(Real-time Operating Systems)와 장치 드라이버(Device Drivers)는 하드웨어 플랫폼(Hardware Platform)과 연결되어 있다. 응용분야 측면에서 보면, 성능과 크기가 하드웨어 플랫폼을 결정하는 제한사항이 된다. 특정 응용분야에서는 프로세서가 최저 속도를 만족하여야 하며, 메모리 시스템도 최저 크기를 만족하여야 한다.



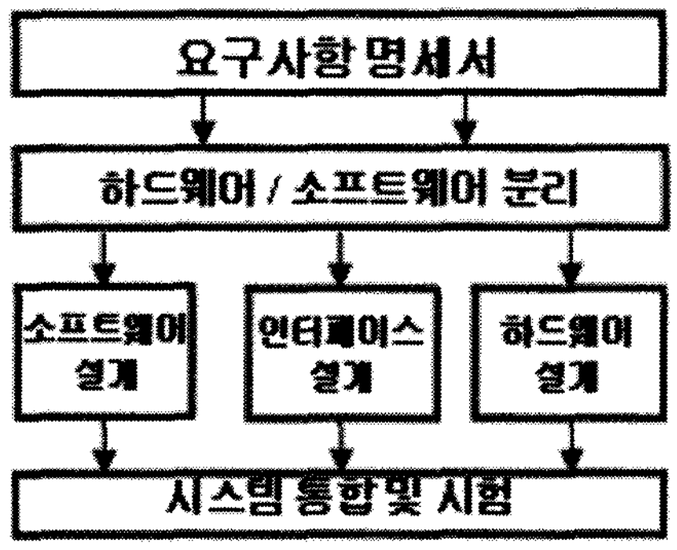
(그림 2) 임베디드 시스템의 기능별 계층 구조

(2) 임베디드 시스템 개발 과정

임베디드 시스템을 설계하는 과정에는 다음과 같은 기본적인 6개의 단계가 포함되어 있다:

- 요구사항 명세
- 하드웨어와 소프트웨어 기능 분리
- 소프트웨어 설계
- 하드웨어 설계
- 인터페이스 설계
- 시스템 통합 및 시험

첫 번째 단계에서는 사용자 요구사항이 분석되고 명세언어로 기술된다. 개발자들은 이 요구사항에 의해서 요구되는 기능들을 유도해낸다. 이 기능들은 하드웨어와 소프트웨어로 할당된다. 하드웨어와 소프트웨어는 동시에 개발되며, 이들 간의 인터페이스도 개발된다. 모든 하드웨어와 소프트웨어가 개발된 후에는 통합되어 시험된다. (그림 3)은 임베디드 시스템 개발 과정을 나타낸다.



(그림 3) 임베디드 시스템 개발 과정

3.2 임베디드 소프트웨어 개발 기법

시스템 수준의 개발 과정에서 임베디드 소프트웨어 개발을 위한 기법을 제안하고자 한다. 우리가 제안하는 기법은 소프트웨어 구조(Architecture)와 서브시스템(Subsystem) 개념을 적용하여 임베디드 소프트웨어 설계 시에 시간과 비용을 절감할 수 있도록 하였다. 본 논문에서 제안하는 임베디드 소프트웨어 개발 과정은 다음과 같은 4개의 기본 단계로 구성 된다:

- 소프트웨어 구조 설계
- 소프트웨어 서브시스템정의
- 기능 블록(Functional Block) 설계
- 타스크(Task) 구현

(1) 단계 1: 소프트웨어 구조 설계

임베디드 소프트웨어를 개발하는 첫 단계는 소프트웨어 구조를 설계하는 것이다. 소프트웨어 구조는 소프트웨어 시스템 전체의 구성을 의미하며, 작업의 단위인 컴포넌트(Component)와 컴포넌트 간의 커넥션(Connection)으로 묘사된다[8]. 컴포넌트는 더 작은 단위로 계층적으로 분할될 수 있으며, 커넥션은 컴포넌트간의 통신 정보를 나타낸다. 이때 도출되는 소프트웨어 구조 형태는 컴포넌트 간의 구조적 배치에 의해서 데이터 공유(Shared Data) 구조, 블랙보드(Black Board) 구조, 계층적(Layer) 구조, 상호 호출 (Invocation)구조, 파이프와 필터(Pipe and Filter) 구조 등과 같이 분류된다.

소프트웨어 개발팀은 이 단계에서 목표 시스템을 이해하고 제안된 하드웨어 구조를 검토한다. 목

표 시스템의 구조를 제작하기 위하여 사용자 입장에서 목표 시스템에서 요구되는 특징 선택사항을 결정한다. 목표 시스템의 모든 특징이 선택된 후에는 목표 시스템의 구조는 적합한 구조 형태에 기반 하여 만들어 진다[9]. 이 소프트웨어 구조는 소프트웨어 시스템을 표현하게 된다. 설계된 소프트웨어 구조는 소프트웨어 서브시스템 정의 단계에서 정제된다.

(2) 단계 2: 소프트웨어 서브시스템 정의

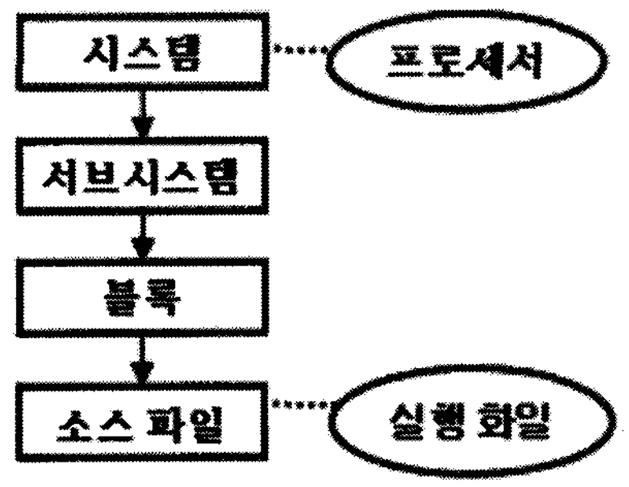
단계 1에서 도출된 전체 소프트웨어 시스템 구조에 나타난 컴포넌트들은 서브시스템으로 분할되어 하향식으로 점차 구체화 된다. 서브시스템은 개발될 논리적 단위 또는 물리적 단위이다. 하나의 서브시스템은 시스템이 제공해야 하는 하나의 기능을 포함하고 있다. 하나의 기능은 실행 단위인 하나 이상의 타스크들에 의해서 구현된다.

(3) 단계 3: 기능 블록 설계

단계 3에서 도출된 각 서브시스템은 기능 특성과 개발 및 유지보수에 용이한 크기에 기반 하여 하나 이상의 기능 블록으로 하향식으로 세분화 되어 설계된다. 하나의 블록은 실제 목표 시스템의 프로세서에 탑재되어 기능을 수행하는 하나 이상의 타스크로 구성된다. 블록 설계 시에는 타스크 사이에 발생하는 메시지 전달 내용도 명세 한다. 프로세서간의 통신은 메시지 전달 방식을 통하여 이루어진다. 타스크와 메시지 인터페이스는 이 단계에서 상세히 정의된다. 메시지 인터페이스를 통하여 전달되는 모든 자료와 가능한 값, 타이머 값들이 정의된다.

(4) 단계 4: 타스크 구현

기능의 가장 기본 단위인 타스크는 프로그래밍 언어에 의해 구현된다. 구현된 프로그램은 컴퓨터 소스 파일로 저장된다. (그림 4)는 설계되고 구현되는 소프트웨어의 논리적 계층구조를 나타낸다. 하나의 소프트웨어 시스템은 하나 이상의 서브시스템으로 구성된다. 서브시스템은 기능에 기반 하여 관리하기 용이한 크기의 하나 이상의 기능 블록으로 분할 될 수 있다. 블록 내부에 포함된 타스크들은 프로그래밍 언어로 구현되어 실행 가능한 화일로 저장된다. 이 실행화일은 목표 시스템의 프로세서에 탑재되어 실행된다.



(그림 4) 구현된 소프트웨어의 논리적 계층구조

우리가 제안하는 설계 기법은 임베디드 소프트웨어 개발 시 소프트웨어 구조와 설계 패턴을 이용하는 것이다. 이것은 기능의 재사용성이 높은 임베디드 소프트웨어의 특성을 고려할 때 이점이 있다. 그리고 하나의 소프트웨어 시스템을 기능을 바탕으로 유지보수하기 쉬운 기능 블록 단위로 구현함으로서 복잡도를 줄이고, 재사용성과 신뢰성을 높였다. POLIS 기법과 같은 방법에서 제안하는 설계절차에 따라 명세서를 번역하고 개발 대상물을 분리하여 개발 후 합성하는 것보다는 우리가 제안하는 소프트웨어 개발과정이 임베디드 소프트웨어 개발에는 더 이점이 있다.

시간제한(Time Constraints) 개념이 포함된 실시간 임베디드 시스템에서는 시스템 설계 시 시간 정보를 명세 하는 것이 상당히 중요하다. 올바른 처리 결과라도 시간제한 보다 늦게 발생하면 잘못된 영향을 미친다. UML 같은 기법에서는 객체간의 메시지 전달에 의한 상호작용에 의해 원하는 결과를 얻게 된다. 따라서 일련의 기능 수행을 위한 소요시간 정보 산출에 어려움이 있다. 본 연구에서 제안한 기능 블록 개념에서의 간략화된 메시지 인터페이스는 UML과 POLIS 기법보다 실행 소요시간이 적다.

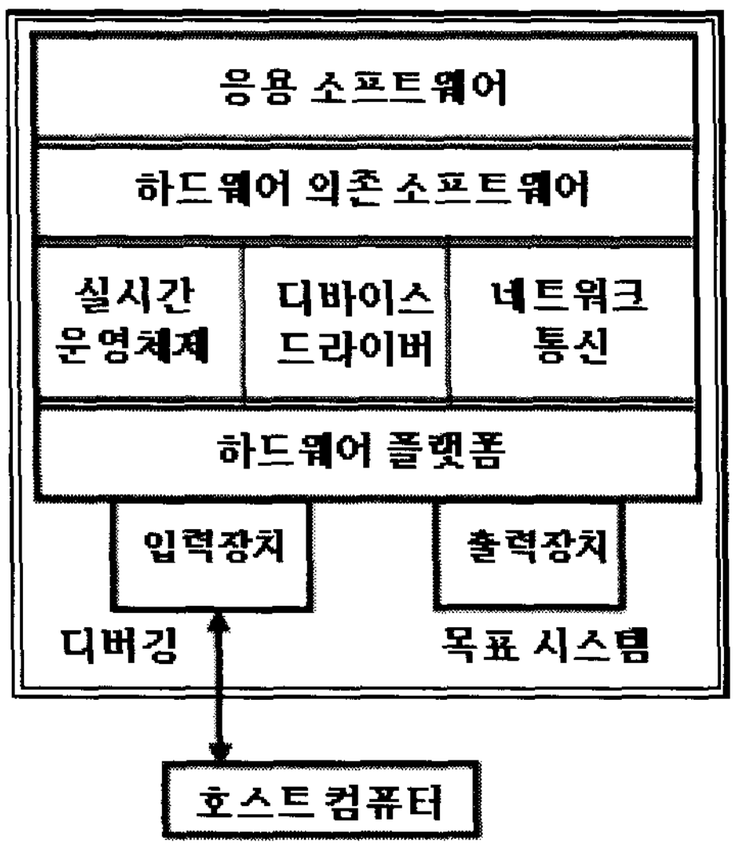
3.3 임베디드 소프트웨어 시험

임베디드 시스템은 소프트웨어를 시험하고 디버깅 하는 것에 어려움이 있다. 임베디드 시스템은

대개 맞춤식 하드웨어 형상위에서 개발되기 때문에, 어떤 임베디드 시스템에 적용할 수 있는 시험 도구와 기법이 다른 임베디드 시스템에는 적용되지 않을 수도 있다[10].

(1) 목표 디버거(Target Debugger)

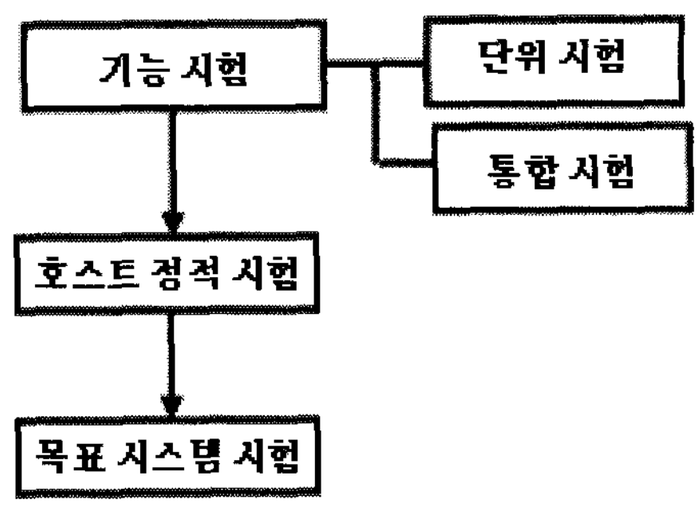
디버깅 시스템은 대개 2개의 프로세스를 가진다. 하나는 시험대상인 프로그램을 실행시키는 데 이용되고, 다른 하나는 디버거를 실행시키는 데 이용된다. (그림 5)는 목표 디버거의 실행 환경을 나타낸다. 목표 디버거는 목표 시스템에 내장되어 실행되는 임베디드 소프트웨어를 시험하는 데 이용되며, 이 목표 디버거는 목표 시스템에서 실행되는 것이 아니라 호스트 컴퓨터에서 실행된다. 그 이유는 목표시스템은 임베디드 시스템에 적합한 최소한의 컴퓨터 자원으로 구성되어 있으므로, 시험 도구가 실행될 만큼의 컴퓨터 용량을 갖추고 있지 못하기 때문이다. 개발하고자 하는 임베디드 소프트웨어가 내장된 임베디드 시스템이 목표 시스템이 되며, 이때 임베디드 소프트웨어는 목표 소프트웨어 또는 목표 프로그램이 된다. 개발자들은 호스트 컴퓨터와 목표 시스템 사이의 인터페이스를 이용하여 목표시스템에서 실행되는 프로그램을 제어하면서 디버깅할 수 있다. 이때 디버거에서 미리 정의된 명령문을 이용하게 되며, 명령문을 이용하여 실행제어, 브레이크 포인트(Breakpoints), 자료의 현재 값 조회 등을 할 수 있다.



(그림 5) 목표 디버거 환경

(2) 임베디드 소프트웨어 시험 및 디버깅

우리는 임베디드 시스템에 내장될 임베디드 소프트웨어를 시험하기 위하여 3개의 시험단계를 제안한다. 첫 번째 단계는 기능시험이다. 이 기능시험은 단위시험과 통합시험으로 구성된다. 이 기능시험은 개발자들이 임베디드 소프트웨어를 개발하는 호스트 컴퓨터에서 수행된다. 개발자들은 각자가 개발한 기능을 위한 단위 시험을 수행한 후에, 전체 기능을 조합하여 통합시험을 수행한다. 통합시험이 성공적인 결과를 얻더라도 목표시스템에 탑재되었을 때 올바르게 동작한다는 보장은 없다. 그 이유는 실행되는 하드웨어 형상이 바뀌기 때문이다. 두 번째 단계는 호스트 정적 시험이다. 마지막 단계는 임베디드 시스템을 구성하는 하드웨어와 소프트웨어의 인터페이스를 포함한 목표 시스템 시험이다. 이 시험은 호스트 컴퓨터에서 인터페이스를 이용하여 목표시스템의 실행을 제어하면서 수행된다. (그림 6)은 임베디드 소프트웨어 시험 및 디버깅 과정을 나타낸다.



(그림 6) 임베디드 소프트웨어 시험 단계

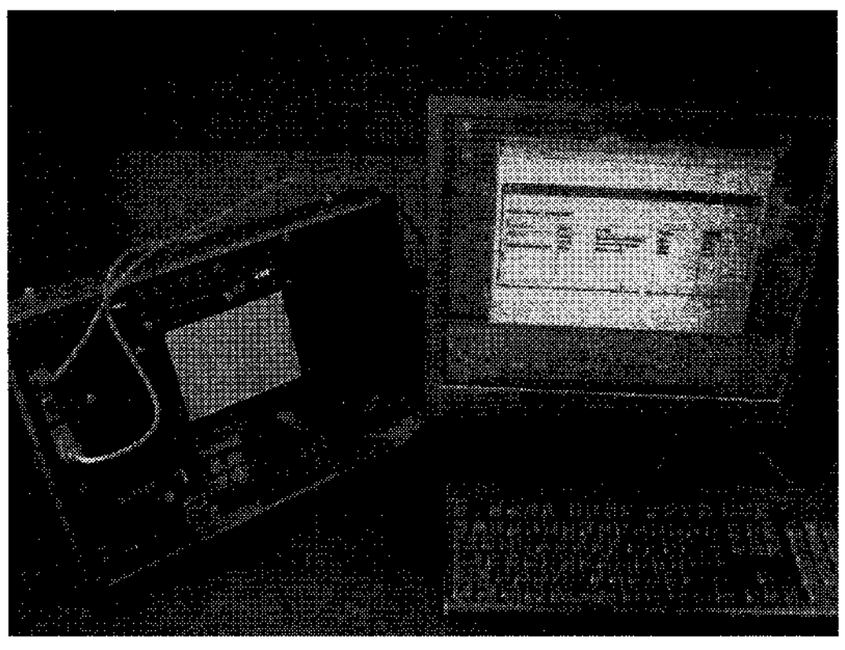
- 기능 시험: 사용자 요구사항에 맞게 기능들이 구현되어 있는지를 시험한다. 개발자들은 자신이 개발한 각 기능블록이 완벽하게 구현되었는지를 검사하기 위하여 단위시험을 수행한다. 단위시험이 성공적인 결과를 얻게 되면, 자신에게 할당되었던 기능 블록을 통합하여 통합시험을 수행하여 요구사항의 기능이 만족되는 지를 검사한다.
- 호스트 정적 시험: 각 개발자들이 개발한 모든 기능 블록들을 통합하여 구성한 소프트웨어 시스템을 호스트 컴퓨터에서 실행시키면서 시험한다. 이때 호스트 컴퓨터에서 작동하는 디버거를 사용하여 변수 값, 제어흐름,

메시지 전달경로 등을 추적할 수 있다.

- 목표 시스템 시험: 호스트 정적시험이 끝난 후에 소프트웨어는 목표 시스템에 탑재된다. 이때 소프트웨어는 목표 시스템을 구성하는 하드웨어와 통합된다. 이 시험단계에서 호스트 컴퓨터에서 실행되면서 인터페이스를 통하여 목표시스템을 제어할 수 있는 목표 디버거를 사용한다. 이 목표 디버거는 목표시스템에서 실행되는 임베디드 소프트웨어, 즉 목표 소프트웨어의 실행상태를 추적할 수 있다. 특정 지점에 브레이크 포인트를 설정하여 실행 중간상태를 검사할 수 있다.

3.4 목표 임베디드 소프트웨어 시험

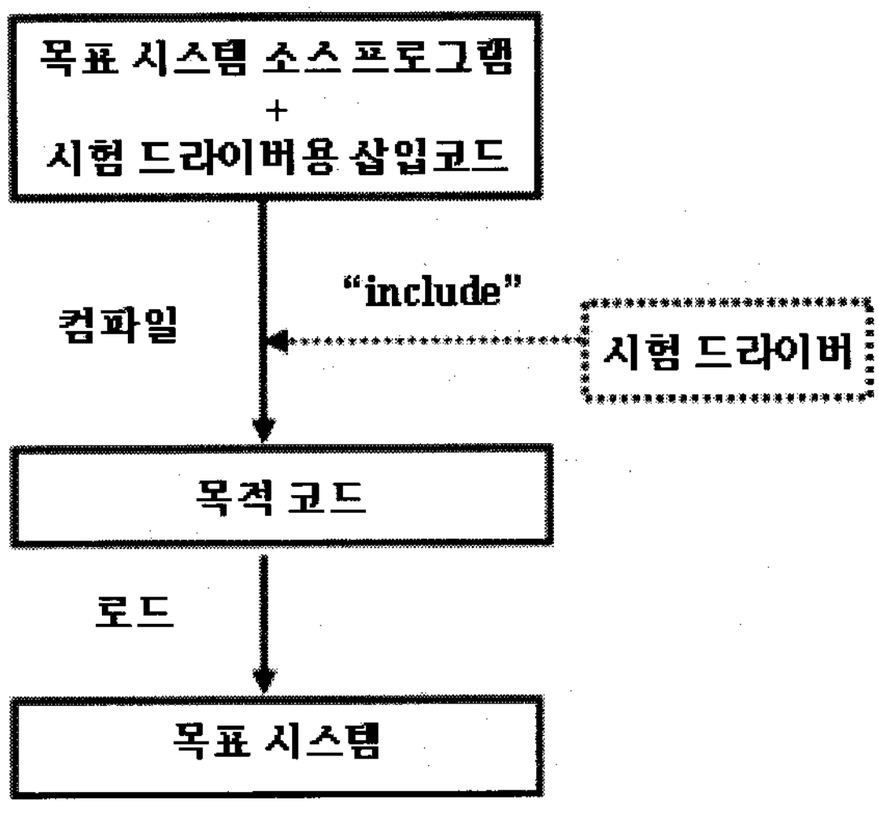
임베디드 시스템에 탑재되는 임베디드 소프트웨어는 임베디드 시스템, 즉 목표 시스템에서 개발되는 것이 아니고 우수한 성능을 가진 소프트웨어 개발도구들이 실행될 수 있는 호스트 컴퓨터에서 개발된다. 목표 시스템은 호스트 컴퓨터가 가지고 있는 고성능 및 대용량의 컴퓨터 하드웨어를 갖추고 있지 않으므로 개발자들은 호스트 컴퓨터에서 소프트웨어를 개발하고 컴파일 하여 실행화일을 생성한 후에 목표 시스템의 메모리에 탑재한다. 그러나 목표 시스템에서 작동하는 디버거가 없기 때문에, 호스트 컴퓨터에서 작동하는 디버거를 목표 시스템의 인터페이스를 통하여 동작시켜서 목표 시스템에 탑재된 소프트웨어의 실행상태를 추적하게 된다. (그림 7)은 호스트 컴퓨터에 인터페이스를 통하여 연결된 목표 시스템의 상태를 보여주고 있다.



(그림 7) 호스트 컴퓨터(우)와 목표 임베디드 시스템(좌)

(1) 목표 소프트웨어 시험방법

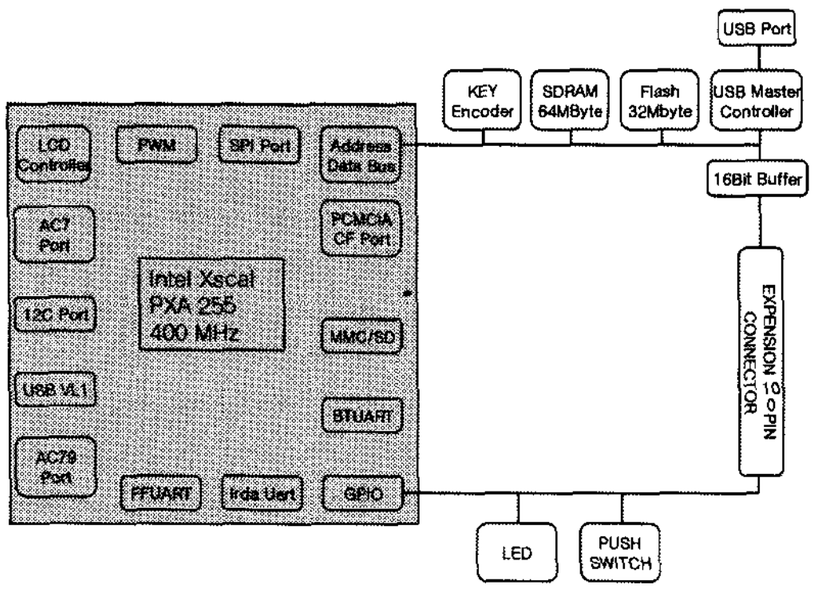
개발하고자 하는 목표 시스템인 임베디드 시스템 내부에 탑재된 소프트웨어를 시험하기 위해서는 호스트 컴퓨터와 목표 시스템을 인터페이스를 통하여 연결한다. 그리고 호스트 컴퓨터에서 시험하고자 하는 목표 소프트웨어의 실행상태를 추적할 수 있는 디버거를 이용하여야 한다. 본 논문에서는 목표 시스템에서 소프트웨어가 실행하는 상태를 추적할 수 있도록, 임베디드 시스템에 탑재될 목표 프로그램에 삽입할 수 있는 시험드라이버를 개발하였다. 이 시험 드라이버는 추적하고자 하는 정보대상에 따라 달리 개발되어야 하며, 목표 프로그램을 컴파일 할 때 포함되어 같이 컴파일 된다. 시험 드라이버 프로그램에는 실행되는 조건, 입출력 상태, 행동, 관계되는 장치 등이 출력되는 명령문이 포함되어 있다. 그리고 목표 소스 프로그램 내부에는 브레이크 포인트를 설정하여, 그 순간에 시험 드라이버에서 정의된 내용을 출력하도록 한다. 이러한 방법으로 시험 대상인 목표 소프트웨어의 실행상태를 추적할 수 있다. 그러나 시험 드라이버는 추적하고자 하는 정보의 대상 및 종류에 따라 브레이크 포인터를 달리 설정하여야 하며, 시험 시나리오에 따라 개발되어야 하므로 사전에 시험 범위 및 대상을 결정해야 하는 단점이 있다. 실행 중 출력되는 결과는 인터페이스를 통하여 호스트 컴퓨터 화면에서 확인할 수 있다. (그림 8)은 임베디드 소프트웨어에 시험 드라이버를 삽입하는 과정을 나타낸다.



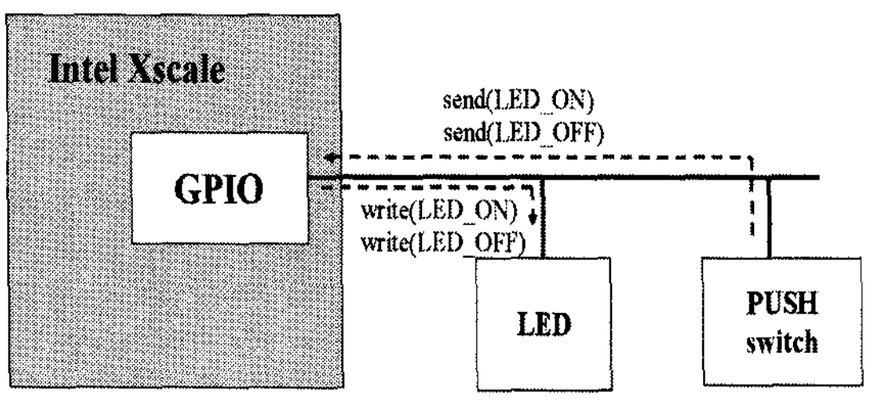
(그림 8) 임베디드 시스템에 탑재된 소프트웨어 시험 방법

(2) 임베디드 소프트웨어 시험 사례

본 연구에서 제안한 시험기법을 실제 임베디드 소프트웨어에 적용하여 보았다. (그림 9)는 시험할 임베디드 소프트웨어가 탑재되는 목표 임베디드 시스템의 논리적 구조이다. 이 목표 시스템은 Intel PXA 255 CPU, 물리적 장치, 그리고 소프트웨어 모듈 등으로 구성된다. 소프트웨어 모듈인 GPIO는 2개의 물리장치 LED 와 PUSH switch와 상호작용한다. LED 불빛을 켜거나 끄기 위해서는 PUSH switch 가 소프트웨어 모듈인 GPIO에 시그널을 보내야 한다. 그러면 GPIO는 PUSH switch의 시그널에 의해서 LED 불빛을 제어한다. (그림 10)은 소프트웨어 모듈인 GPIO와 물리적 장치인 LED와 PUSH switch간의 상호작용을 나타낸다.



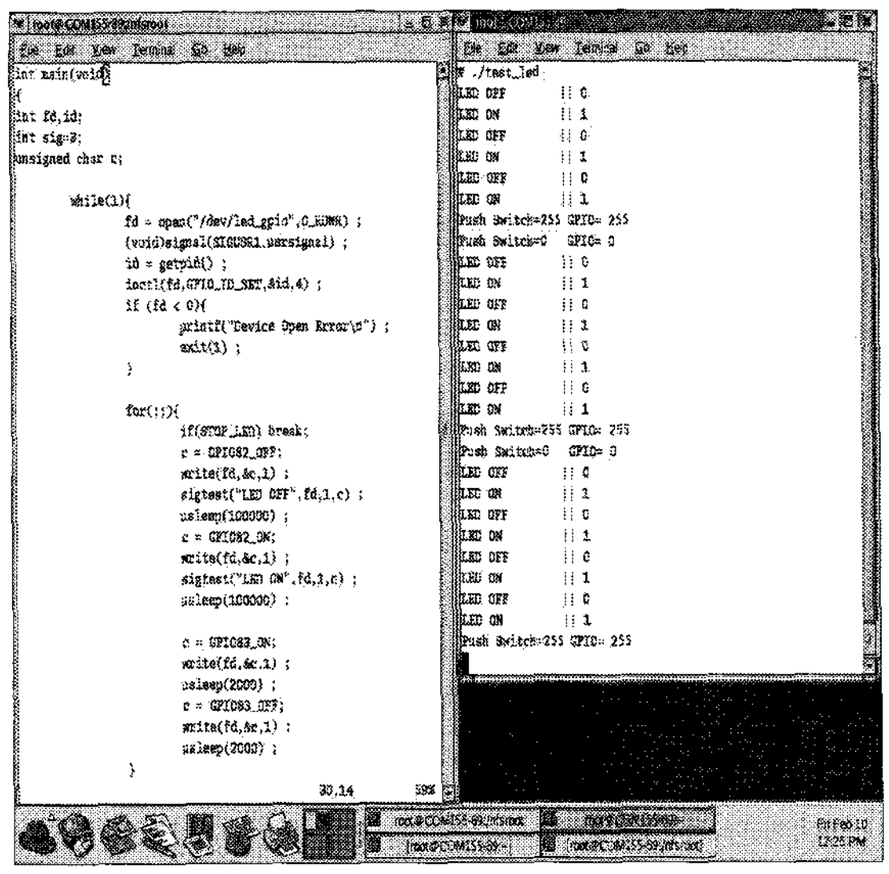
(그림 9) 목표 임베디드 시스템 예



From	To	Signals	Functions
GPIO	LED	LED_ON & LED_OFF	Turn on/off LED lights
PUSH switch	GPIO	LED_ON & LED_OFF	Asking GPIO for LED on/off

(그림 10) 임베디드 소프트웨어 GPIO 와 물리적 장치간의 상호작용 시험

목표 소프트웨어인 GPIO의 실행상태를 추적하기 위하여, 우리는 GPIO가 LED와 PUSH switch와 통신하는 정보를 출력하는 시험 드라이버를 개발하였다. 이 시험 드라이버는 소프트웨어 함수이며 입력, 출력, 행동, 연관된 장치를 출력하는 기능을 가지고 있다. 그리고 GPIO 소스프로그램 내부에 브레이크 포인트를 설정하여 그 순간을 지날 때마다 시험 드라이버에서 정의한 정보가 호스트 컴퓨터 화면에 출력되도록 하였다. GPIO 소스 프로그램과 시험 드라이버를 컴파일 하여 목표 시스템에 탑재한다. 그리고 목표시스템의 LED와 PUSH switch를 작동시키면 시험 드라이버에서 미리 정의된 정보가 호스트 컴퓨터 화면에 출력된다. (그림 11)은 목표 시스템이 작동하는 동안 호스트 컴퓨터 화면에 출력되는 정보를 나타내고 있다. 이 출력되는 정보를 바탕으로 목표 소프트웨어인 GPIO의 실행 흐름을 파악할 수 있다.



(그림 11) 실행상태를 보여주는 호스트 컴퓨터 화면

4. 결 론

임베디드 소프트웨어가 점점 더 많은 기능을 포함하게 되고, 그 크기도 점점 커지고 있다. 따라서 소프트웨어 개발자들이 제시간에 개발기간을 맞추는 것이 점점 더 어려워지고 있다. 따라서 본 연구에서는 임베디드 소프트웨어 개발에 적합한 설계 및 시험 기법을 제안하였다. 우리가 제안하는 개발

기법은 소프트웨어 구조와 서브시스템 개념을 적용하여 임베디드 소프트웨어 설계 시에 시간과 비용을 절감할 수 있도록 하였다. 제안하는 임베디드 소프트웨어 개발 과정은 소프트웨어 구조 설계, 소프트웨어 서브시스템 정의, 기능 블록 설계, 타스크 구현 등과 같은 4개의 기본단계로 구성된다. 우리가 제안하는 설계 기법은, 임베디드 소프트웨어 개발 시 소프트웨어 구조와 설계 패턴을 이용하는 것이다. 이것은 기능의 재사용성이 높은 임베디드 소프트웨어의 특성을 고려할 때 상당히 이점이 있다. 그리고 하나의 소프트웨어 시스템을 기능을 바탕으로 유지보수하기 쉬운 기능 블록 단위로 구현함으로써 복잡도를 줄이고, 재사용성과 신뢰성을 높였다.

우리는 임베디드 시스템에 내장될 임베디드 소프트웨어를 시험하기 위하여 3개의 시험단계를 제안하였다. 첫 번째 단계는 기능시험이다. 이 기능시험은 단위시험과 통합시험으로 구성된다. 이 기능시험은 개발자들이 임베디드 소프트웨어를 개발하는 호스트 컴퓨터에서 수행된다. 개발자들은 각자가 개발한 기능을 위한 단위 시험을 수행한 후에, 전체 기능을 조합하여 통합시험을 수행한다. 통합시험이 성공적인 결과를 얻더라도 목표시스템에 탑재되었을 때 올바르게 동작한다는 보장은 없다. 그 이유는 실행되는 하드웨어 형상이 바뀌기 때문이다. 두 번째 단계는 호스트 정적 시험이다. 마지막 단계는 임베디드 시스템을 구성하는 하드웨어와 소프트웨어의 인터페이스를 포함한 목표 시스템 시험이다. 이 시험은 호스트 컴퓨터에서 인터페이스를 이용하여 목표시스템의 실행을 제어하면서 수행된다.

개발하고자 하는 목표 시스템인 임베디드 시스템 내부에 탑재된 소프트웨어를 시험하기 위해서는 호스트 컴퓨터와 목표 시스템을 인터페이스를 통하여 연결한다. 본 연구에서는 목표 시스템에서 소프트웨어가 실행하는 상태를 추적할 수 있도록, 임베디드 시스템에 탑재될 목표 프로그램에 삽입할 수 있는 시험 드라이버를 개발하였다. 이 시험 드라이버는 시험 시나리오에 의해 개발되어야 하며, 목표 프로그램을 컴파일 할 때 포함되어 같이 컴파일 된다. 시험 드라이버 프로그램에는 실행되는 조건, 입출력 상태, 행동, 관계되는 장치 등이 출력되는 명령문이 포함되어 있다. 그리고 목표 소스 프로그램 내에는 브레이크 포인트를 설정하여, 그 순간에 시험 드라이버에서 정의된 내용을 출력하도록 하였다. 이러한 방법으로 시험 대상인 목표 소프트웨어의 실행상태를 추적할 수 있다.

우리가 제안한 임베디드 소프트웨어 시험방법으로 시험 드라이버를 개발하여, 임베디드 시스템을 구성하는 소프트웨어 모듈과 물리적 장치간의 입출력 정보를 추적하는 시험을 적용하여 보았다.

감사의 글

이 연구는 대구대학교 연구지원에 의해 이루어졌습니다.

참 고 문 헌

[1] 송태훈, ARM9 Core를 이용한 시스템온칩 및 임베디드시스템 설계, 홍릉과학출판사, 2004.

[2] The POLIS System, Release 4.0, http://www.cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

[3] B. P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems, 2nd Edition, Addison Wesley, 2000.

[4] R. Saracco, J. Smith, and R. Reed.: Telecommunications Systems Engineering Using SDL, NewYork: North-Holland, 1989.

[5] Gjalt de Jong.: A UML-Based Design Methodology for Real-Time and Embedded Systems.Proceedings of the 2002 Design, Automation, and Test in Europe Conference and Exhibition, 2002.

[6] Alberto Sangiovanni-Vincentelli and Grant Martin.: Platform-Based Design and Software Design Methodology for Embedded Systems. IEEE Design & Test of Computers, November-December, 2001, pp.23-33.

[7] M. Sgroi, L. Lavagno, and A. Sangiovanni-Vincentelli.: Formal Models for

Embedded Systems Design. IEEE Design & Test of Computers, April-June, 2000, pp.2-15.
[8] Mary Shaw and David Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.
[9] H. Gomaa and G. A. Farrukh.: Composition of Software Architectures from Reusable Architecture Pattern. Proceedings of the ISAW3, Orlando, USA, 1998, pp.45-48.
[10] Harry Koehnemann and Timothy Lindquist.: Towards Target-Level Testing and Debugging tools for Embedded Software. Proceedings of the Conference on TRI-Ada'93, Seattle, USA, October, 1993, pp.288-298.
[11] "A framework for hardware-software co-design of embedded systems" retrieved on August 25th, 2006, http://embedded.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

강 병 도(Byeong-do Kang)

- 종신회원
- 1995년 서울대학교 이학박사 (전산과학 전공)
- 1988년~1998년 한국전자통신연구원 선임연구원
- 2004년 미국 CMU Research Associate
- 1998년~현재 대구대학교 부교수
- 관심분야 : 소프트웨어구조, 소프트웨어 프로세스, 소프트웨어 개발방법론, 소프트웨어 시험기법, 인터넷 윤리