韓國 컴퓨터情報學會 論文誌
第13卷 第3號, 2008. 5.

2008-13-3-1-5

μC/OS-II 운영체제환경을 고려한 SDL 명세로부터의 내장형 C 코드 자동 생성

곽 상 훈*, 이 정 근**

Automatic SDL to Embedded C Code Generation Considering μC/OS-II OS Environment

Sang-Hoon Kwak *, Jeong-Gun Lee **

요 약

내장형시스템의 복잡도 증가로 인하여 내장형 소프트웨어의 자동생성 및 하드웨어-소프트웨어 통합설계 방법론 등이 크게 이슈화 되고 있다. 자동화된 설계 방법론에 있어서 공통적인 요구사항은 시스템 설계를 효과적으로 모호성 없이 기술 할 수 있도록 정형화된 설계 언어를 제공하는 것과 설계 언어로 부터 자동으로 원하는 소프트웨어 코드를 생성하는 방법의 개발이다. 본 논문에서는 시스템 기술언어로 ITU-T에 의해 표준으로 권고되어 널리 사용되고 있는 SDL (Specification and Description Language)로부터 실시간 운영체제 uC/OS-II에서 수행될 임베디드 C코드를 자동으로 생성하는 자동화된 방법론을 제시한다. 연구 개발된 자동 내장형 C코드 생성기는 하드웨어-소프트웨어 통합설계환경에서 소프트웨어 설계의 한 축으로 이용될 수 있으며 SDL 시뮬레이터나 검증기를 통하여 개발하고자 하는 code의 기능을 초기 모델 수준에서 평가하고 검증 할 수 있다.

Abstract

Due to the increasing complexity of embedded system design, automatic code generation of embedded software and hardware-software co-design methodologies are gaining great interest from industries and academia. Such an automatic design methodologies are always demanding a formal system specification languages for defining designer's idea clearly and precisely. In this paper, we propose automatic embedded C code generation from SDL (Specification and Description Language, ITU-T recommended the SDL as a standard system description language) with considering a real-time uC/OS-II operating system. Our automatic embedded C code generator is expected to provide a fast specification, verification and performance evaluation platform for embedded software designs.

▸ Keyword : 임베디드 시스템(embedded system), 코드 자동 생성(Automatic Code Generation), SDL (Specification and Description Language)

• 제1저자 : 곽상훈 교신저자 : 이정근
• 접수일 : 2008. 3. 27, 심사일 : 2008. 4. 3, 심사완료일 : 2008. 5. 24.
* 광주과학기술원 정보통신공학과 **한림대학교 컴퓨터공학과

Ⅰ. 서 론

최근 디지털시스템 설계 복잡도의 증가, 하드웨어와 소프트웨어가 혼재된 시스템의 동시설계 등으로 인하여 시스템 수준 설계 방법론, 시스템 수준 사양 기술 언어 (Specification Description Language) 등에 관한 관심이 고조되고 있다 [1]. 시스템 수준 명세 언어를 이용하여 시스템을 기술할 경우 시스템의 복잡한 세부 사항들을 나타내지 않고 구현과 독립적인 시스템 기술이 가능하다. 또한 최상위 수준에서 조기에 시스템의 동작을 검증할 수 있는 장점이 있다. 추상적인 시스템 기술언어로부터 실제 구현언어로 변환시키는 과정은 점차 방대해지는 시스템 규모와 조기 시장 진입이 관건인 내장형 시스템 설계분야의 흐름으로 볼 때 전체 시스템 설계과정 중 필수적인 부분이라 할 수 있다. 시스템수준 기술언어로는 SDL, LOTOS, SystemC, SpecChart 등과 같이 여러 종류의 언어들이 존재하나 본 논문에서는 정형적으로 구문의 의미가 잘 정의되어 있고, 국제기관에 의해 표준으로 권고 되고 있으며, 구현 독립적인 장점을 가지는[2] SDL(Specification and Description Language)로부터 내장형 시스템에서 수행될 C코드를 자동 생성하는 방법을 제안하고자 한다.

SDL은 1976년 CCITT(ITU-T의 전신)에 의해 처음 표준으로 권고된 이래 주로 통신시스템의 정형적 명세에 널리 이용되어 왔으며 최근에는 내장형시스템, 실시간시스템의 설계에 적용하려는 연구가 활발히 진행되고 있다 [3]. 본 논문에서는 제어중심의 내장형 시스템 설계 시에 SDL을 이용하여 시스템사양을 기술하고 그 사양으로부터 우선순위기반의 선점형 운영체제 상에서 수행될 소프트웨어 부분을 C 코드로 자동 생성해주는 방법을 제안한다. 내장형 시스템의 전형적인 구조에서, 하드웨어 부분은 MPU (micro- processor unit) 또는 MCU (micro-controller unit) 코어와 데이터 처리 등에 있어서 MPU 및 MCU를 지원하는 ASIC (Application Specific Integrated Circuit), 그리고 주변 장치로 구성되어 있으며, 소프트웨어 부분은 커널의 크기가 작고 실시간 요구사항을 만족하는 실시간 운영체제 상에서 응용프로그램이 수행되는 형태이다.

본 논문에서는 소프트웨어 구현을 위해 C코드를 생성할 때 운영체제에 의존하지 않는 프로세스 내부 행동 부분(process behavior)과 운영체제에 의존하는 프로세스 간 통신 부분을 분리하여 코드를 생성하였다. 더불어, SDL 수행모델을 지원하기 위한 추가적인 제어프로세스를 두지 않고 C 코드와 운영체제 서비스 함수만을 이용하여 원래의 SDL 로 기술된 code가 소프트웨어로 수행되게 하였다. 앞서 언급한 전형적인 내장형 시스템의 구조에 적합하도록 많은 실시간 운영체제에서 채택하고 있는 우선 순위기반의 선점형 스케줄링 방식을 고려하였고, 운영체제의 시스템 서비스를 이용하는 긴밀한 통합(tight integration)을 통해서 내장형 시스템 개발에 실질적인 도움이 되는 시스템을 구현하였다.

본 논문은 다음과 같이 구성되어 있다. 2절에서는 관련연구 및 SDL 시스템의 개념적인 수행모델과 SDL-C 변환모델을 소개하고, 3절에서는 운영체제에 의존적인 SDL 시스템의 요소들을 살펴보고, SDL 프로세스간 시그널 전송과 SDL 다중 프로세스의 구현을 위한 운영체제 스케줄링 기법의 이용방법에 대해 살펴보았다. 4절에서는 SDL 사양으로부터 SDL2C 코드변환기를 통해 구현한 Access Control 시스템의 예를 소개하고 5절에서는 결론을 제시한다.

Ⅱ. SDL명세의 C코드로의 변환

그림 1은 SDL로부터 C코드를 생성하는 전체 과정을 보여준다. 중간 형태로 Telelogic사의 SDL Access [4]라는 자료구조 및 API를 이용하여 초기 SDL 코드를 파싱한 후 얻어진 access 파일로부터 그와 동등한 내장형 C코드를 생성한다.

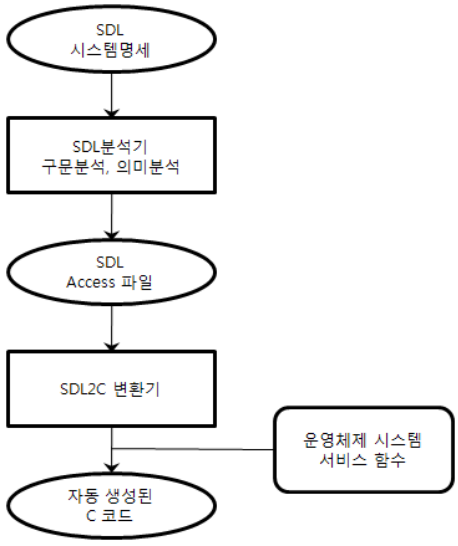


그림 1. SDL-C 변환 흐름도
Fig. 1. A flow chart of SDC-C transformation

C 코드 생성 시에 목적 운영체제의 시스템 서비스 함수를 이용하면 SDL 시그널 전송 등의 동작을 좀 더 효율적으로 구현할 수 있다. 본 논문에서는 목적 운영체제로 uC/OS-II라는 실시간 운영체제를 사용하였다. 대부분의 운영체제에서 프로세스생성, 소멸, 메시지전송, 세마포어와 같은 대표적인 종류의 운영체제 시스템함수와 자료구조를 지원한다. 따라서, 본 논문에서 제안하는 SDL2C 변환기는 표준 C 코드 (ANSI C)를 기준으로, 사용될 운영체제의 서비스 함수만을 사용하여 이식성(portability)을 높이고자 하였으며, 다른 운영체제 환경으로 이식하는 경우 운영체지 의존적인 코드, 즉 이식하고자 하는 운영체제의 의존적인 서비스 함수만을 고려하여 수정함으로써 새로운 운영체제 환경에 맞는 C 코드를 쉽게 생성할 수 있다.

2.1 관련연구

내장형 시스템 설계에 있어 많은 연구가 SDL 언어를 이용하여 진행되어 왔다 [5-13]. N. E. Zergainoh과 G. F. Marchiroro 등은 하드웨어-소프트웨어 통합설계 환경의 시스템 기술 언어로 SDL을 이용하여, C언어와 VHDL로 각각 변환한 후 소프트웨어 컴파일, 하드웨어 합성을 통하여 원하는 시스템을 구현하고 있고 [5, 6], F. Slomka 등은 pSOS 기반 운영체제에서 수행될 C코드를 SDL로부터 생성하는 방법을 제시하고 있으나 실제 행위(behavior)를 가진 SDL 프로세스 이외에 SDL 시스템의 실행모델을 실제 소프트웨어 수행 시 제어, 감독하기 위한 메타프로세스를 생성하므로 부가적인 오버헤드와 성능저하의 우려가 있다 [8]. 또한 J. M. Alvarez등은 실시간 시스템 설계를 위해 SDL을 초기 사양 기술언어로 채택하여 C코드를 생성하고 실시간 응답성 보장에 중점을 둔 SDL 수행모델을 제시하였으나 기본적인 실시간 운영체제의 상위에 SDL 시스템의 소프트웨어 수행을 지원하는 전용 미들웨어를 구성하므로 구현된 SDL 소프트웨어 시스템에 추가적인 부담을 부과한다 [9][10][11]. 따라서 본 논문의 목표는 변환된 C코드가 원래의 SDL 사양이 나타내는 행위 이외에 SDL 수행모델을 지원하는 미들웨어나 부가적인 코드가 최소가 되도록 C코드를 생성하는 것이다. 따라서 본 논문에서 제안된 C코드 생성방법을 이용하여 SDL 사양기술로부터 표준 C코드(ANSI C)와 운영체제 서비스 함수만으로 구성되는 효율적인 소프트웨어 구현을 얻을 수 있다. J. M. Daveau, O. Bringmann과 C. Drosos 등은 SDL로부터 하드웨어를 구현하기 위한 방법론을 제시하고 있으며 SDL명세로부터 VHDL등과 같은 하드웨어 기술언어로의 변환을 통한 내장형시스템의 하드웨어 부분 설계자동화를 지원하고 있다 [12][13][14].

2.2 SDL 수행모델

SDL 사양기술에서 시스템 설계자가 기술하려고 하는 실제 시스템은 system이라는 하나의 단위로 표현된다. 그리고 구조적으로는 한 system 내부에 여러 개의 block이 존재할 수 있으며 하나의 블록 내부에는 또 다시 하나의 block 또는 하나의 process를 기술하는 방식으로 한 시스템 내부의 계층성을 표현한다. SDL 기술에서 시스템의 동작을 나타내는 기본단위는 프로세스이며, 한 프로세스의 내부행위는 FSM(Finite State Machine)과 상태 전이 시에 데이터 연산을 표현할 수 있는 EFSM(Extended Finite State Machine) 모델을 기반으로 하고 있다 [15]. 프로세스들 사이의 상호작용은 프로세스들 사이에 시그널 전송과 그 시그널에 대해 종속적으로 정의된 데이터 교환으로 기술할 수 있다. 그리고 여러 개의 프로세스를 포함할 수 있는 블록이라는 단위로 시스템의 계층성을 표현한다. 따라서 추상적인 SDL 모델로부터 실제 소프트웨어로 구현될 코드를 자동으로 생성하기 위해서는, 실제 생성된 코드의 실행환경에 적합하도록 원래 사양기술에 나타나 있지 않은 묵시적인 요소들을 지정해 줄 필요가 있다. 예를 들어 SDL 기술에서는 한 process 내에서 시그널 전송 시에 그 시그널을 받는 process를 명시적으로 지정하지 않아도 signalroute, channel 정의에 의해 해당 시그널을 받아들일 수 있는 process가 결정되며, 프로세스마다 무한한 크기를 가지는 입력큐가 존재하여 그 프로세스로 전송된 모든 시그널들을 프로세스의 입력큐로부터 읽어 들일 수 있다. 따라서 이러한 개념적인 시그널전송 모델은 실제 소프트웨어 구현 시에는 메시지 전달, 전역변수 읽기/쓰기 등 여러 가지 방법으로 구현가능하며, 코드생성기에서 이러한 구현방법을 지정하여 생성된 C코드에서는 원래의 사양기술의 의미를 소프트웨어 동작에서 실현하게 된다.

SDL 구문에는 산술연산, 프로세스 정의 등 C언어와 일대일 대응이 가능한 구문도 있으나 위에서 설명한 바와 같이 하나의 구문이 여러 개의 C언어 구문과 운영체제 수준에서 제공하는 함수까지도 이용해서 구현해야 하는 구문들도 존재한다. 시그널 전송, 타이머시그널 전송과 같은 구문과 다중 SDL process의 동시수행 등은 운영체제 기능을 이용하지 않고는 사실상 효율적인 수행이 불가능한 구문들이다. 따라서 본 논문에서는 일대일로 C언어와 대응할 수 있는 SDL 요소와 운영체제 제공 기능을 이용해야 하는 SDL 요소를 분리하여 C코드를 생성하고 이외에 추가적인 제어를 위한 코드 없이 SDL 문맥을 멀티태스킹 기능을 지원하는 운영체제 상에

서 동작하는 소프트웨어로 가능한 일대일 대응되게 하였다. 따라서 부가적인 관리, 제어를 위한 코드가 존재하지 않으므로 좀 더 효율적이고 동작을 이해하기 쉬운 C코드를 생성할 수 있다.

본 논문에서 제안한 SDL2C코드 생성기로부터 생성된 C코드는 다음과 같은 실행환경을 가정하고 있다.

- 하나의 SDL process는 운영체제에서 하나의 task로서 수행된다.
- SDL 기술에서의 계층구조(프로세스-블록-시스템)는 평탄화(flattening)되어 C코드에서 표현된다.
- 값의 범위에 대한 제한이 없는 SDL 자료형은 변환된 C언어 자료형의 범위로 값이 제한된다.
- SDL 수행모델에서 각 프로세스마다 존재한다고 가정되는 무한 크기의 입력버퍼는 생성되는 C언어에서 유한크기의 큐로 구현된다.

2.3 SDL 시스템의 소프트웨어변환 모델

2.2 절에서 언급한 바와 같이 본 논문에서는 SDL 구문 중 C코드와 일대일 대응할 수 있는 구문요소와 운영체제 기능을 이용하는 부분으로 나누어 C코드를 생성한다. 그림 2는 SDL-C 변환 모델을 나타낸다.

◆ 운영체제 의존부

운영체제 의존부에는 운영체제 IPC를 이용하여 수행되는 시그널 전송과 데이터 인자 전달, 타이머, 다중프로세스의 생성 및 소멸 등과 같은 기능들이 포함된다. 이 기능들은 운영체제에서 기본적으로 제공하는 기능들과 밀접하게 관련되어 있어 이 기능들을 이용하면 보다 간단하고 명료한 C언어 코드를 생성할 수 있다. 본 논문에서는 공개된 실시간 운영체제인 uC/OS-II [16] 환경에서 실행될 C코드를 생성한다.

◆ 운영체제 비의존부

자료형 정의, process 내부행위, 조건부 상태전이 등은 그와 가장 유사한 C언어 코드로 직접 변환한다. 이 부분은 운영체제 기능과 독립적으로 C언어 구문수준에서 표현이 가능하므로 그에 대응되는 C언어 구문으로 변환한다.

위에서 간략히 언급한 운영체제 의존부/비의존부 분리모델의 각 요소들을 좀 더 자세히 살펴보면 다음과 같다.

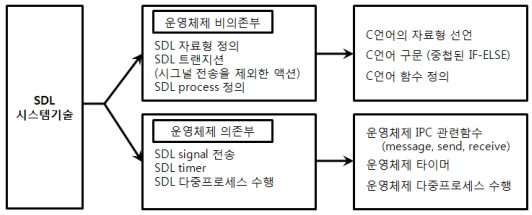


그림 2. 운영체제 의존부 분리에 의한 SDL-C 변환모델
Fig. 2. SDL-C transformation according to the dependency of an operating system

◆ SDL의 계층구조

SDL 사양기술에서는 block-process를 이용하여 계층구조를 표현한다. block 내부에서는 channel을 통하여 하위 block 또는 실제 행동을 기술하는 process들 사이의 커뮤니케이션 경로만을 나타내고, 실제 행동은 process 내부에서 기술한다. 이 block 구조는 계층구조를 나타내기 위한 하나의 개념적인 틀(template)이다. 생성되는 C코드에서는 이 계층구조를 평탄화 시키므로, 모든 프로세스들이 동일한 층위에서 수행된다는 가정 하에서 소프트웨어 모델을 작성하였다.

◆ SDL 자료형

SDL의 기본 자료형에는 integer, boolean, real, character, charstring이 있고, 이들은 각각 C언어의 int, int, float, char, char [20]으로 변환된다.

◆ SDL process 정의

SDL process의 본체 정의는 C언어의 함수 정의와 대응된다. C언어의 함수정의로 변환된 SDL process의 실제 수행은 운영체제의 프로세스 생성기능에 의해 시작된다.

◆ SDL 트랜지션

SDL 상태 전이 시에 발행하는 액션은 대부분 C언어 코드로 일대일 대응된다. 산술연산, 프로시저 호출 등이 이에 해당되며 시그널 전송 구문은 운영체제 기능을 이용하여 변환된다. 또한 상태전이 시 변수조건에 따라 다른 상태로 전이하는 decision 구문은 C언어의 if-else 문을 이용하여 변환한다.

◆ SDL 다중프로세스 수행

SDL process의 다중수행은 SDL process 본체의 정의 부분이 C 언어 함수정의 본체로 변환된 후 운영체제에서 제공하는 프로세스 생성 서비스 함수를 이용하여 수행한다. 내장형 시스템에서 우선순위 기반의 선점형 운영체제가 널리 이용되고 있으므로,

본 논문에서는 우선순위 기반의 선점형 운영체제 상에서 SDL 다중 프로세서가 교착상태가 발생하지 않는 SDL 수행

모델을 제시하였다. 이에 관해서는 3장에서 자세히 살펴보기로 한다.

◆ SDL 타이머

SDL 타이머 시그널은 운영체제의 타이머 서비스 함수를 이용하여 수행된다. 이 경우 타이머의 정밀도 및 타이머 서비스 함수의 구현에 따라 타이머를 이용한 처리에 미세한 오차가 발생할 수 있으며, 높은 수준의 정밀도를 요구하는 타이머를 이용하기 위해서는 하드웨어 타이머 신호를 직접 이용하여야 한다. 그러나 본 논문에서는, 프로세서 하드웨어의 발달에 따른 타이머의 정밀도가 증가와 운영체제에서 제공하는 서비스 함수의 기능이 향상됨에 따라 직접 하드웨어 타이머를 이용하는 것보다는 코드 자동 생성에 있어 보다 중요한 설계 요소인 "이식성" 향상을 위해 운영체제가 제공하는 타이머 서비스 함수를 이용한다.

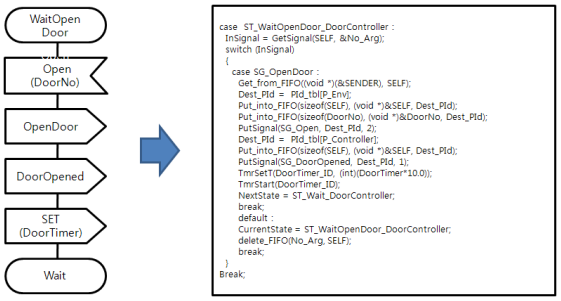


그림 3. SDL 명세에서의 한 상태전이를 C코드로 변환한 예
Fig. 3. A C code example for state transitions in SDL specification

앞 절에서 설명한 바와 같이 하나의 SDL process는 EFSM 동작을 표현한다. EFSM 동작 구조는 C언어의 중첩된 switch - case 문을 이용하여 표현이 가능하고, 상태전이 시에 발생하는 데이터연산은 그와 동일한 C언어 연산으로 변환이 가능하다. 그림 3은 SDL 프로세스 내부에서의 한 상태전이를 C코드로 변환한 결과이다. 그림 3의 왼쪽 부분은 SDL 프로세스에서의 한 상태전이를 그래픽 표현으로 나타낸 것이고 오른쪽 부분은 변환된 C코드이다. GetSignal(), PutSignal() 함수는 운영체제의 메시지 메일박스와의 데이터 송수신을 위한 함수로써 SDL 시그널 교환을 구현한 것이며, Get_from_FIFO() 함수와 Put_into_FIFO() 함수는 하나의 시그널과 함께 교환되는 데이터를 프로세스의 입력큐로부터 읽거나 입력큐에 쓰는 함수이다.

III. 실시간 운영체제 환경에 대한 고려

SDL2C 변환기는 SDL 구문을 C의 구문으로 변환해주는 구문중심적(syntax oriented)번역을 수행하기보다는 생성될 C코드가 수행될 환경을 고려하여 C코드를 생성한다. 대부분의 실시간 운영체제에서 선점형 우선순위기반의 스케줄링 기법이 사용되는 것을 고려하여, 실제 운영체제 상에서 다중프로세스로 수행이 되도록 SDL의 다중프로세스 수행모델을 구현하였다. 또한, SDL의 시그널 송수신은 의미상 대응하는 운영체제의 IPC 함수를 이용하여 구현하였다.

3.1 운영체제 의존적인 SDL의 시스템 요소들

앞 장에서 설명한 바와 같이, 운영체제 의존적인 SDL의 시스템 요소에는 다중프로세스 수행, 타이머, 프로세스간 시그널과 데이터송수신 등이 있다. 이 절에서는 이 시스템 요소들을 uC/OS-II가 제공하는 운영체제 제공함수나 자료구조를 이용하여 구현하기 위하여 SDL의 수행모델의 의미를 자세히 살펴보고, 이 시스템 요소들을 효율적으로 표현할 수 있는 uC/OS-II 운영체제의 기능들을 제시하여 변환 관계를 설정하고자 한다.

(1) 다중프로세스 수행

SDL process들은 SDL 수행모델에서 병행적으로 수행되지만, 실제 수행환경에서는 process들에 대한 스케줄링 및 타이밍에 의해 수행된다. 스케줄링 및 타이밍에 관한 사항은 구현 시에 설계자에 의해 결정된다. 본 논문에서는 uC/OS-II 즉, 선점형 우선순위기반 운영체제 방식에서의 다중 프로세스 수행환경에 따라 원래 SDL 원시코드의 수행환경을 구현한다.

SDL process는 시그널을 전송할 때 SDL process 이름만으로 해당 시그널을 수신하게 될 process를 식별하는 것이 가능하며, 운영체제는 각 process의 우선순위를 할당하기 위하여 SDL process address를 이용할 수 있다. 그림 4와 같이 프로세서 이름을 address로 사용하여 각 process에 할당된 우선순위를 저장하고 있는 프로세스 식별자 테이블을 보여준다.

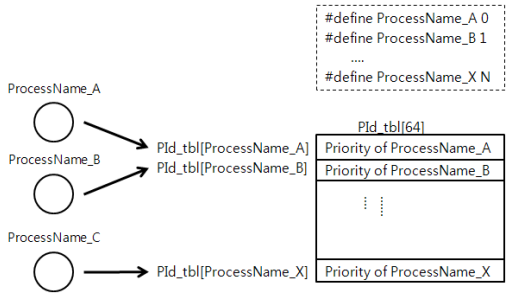


그림 4. process 이름을 통한 식별을 위한 process ID 테이블
Fig. 4. A process ID table for identifying process names

```
void ActivateSystem_SystemName(void)
{
        TaskData[Curr_Prio][0] = 0;
        TaskData[Curr_Prio][0] = Curr_Prio;
        PId_tbl[P_Env]();
        OSTaskCreate(&ProcessName1,
       (void *) & TaskData[Curr_Prio][0],
     &TaskStack[Curr_Prio][STACKSIZE-1], Curr_Prio);
}
```

그림 5. C코드에서 다중프로세스 수행을 위한 초기 프로세스 생성루틴
Fig. 5. A C code routine creating an initial process for multi-process execution

또한 SDL 수행 환경에서는 수행 중인 프로세스를 생성시키거나 소멸시키는 것이 가능하며, 본 논문에서 개발한 C코드 생성기 역시 동적 process의 생성과 소멸 기능을 지원한다. 변환된 C코드에서는 실행시작 시부터 수행될 프로세스들을 생성하기 위해 그림 5와 같이 초기화부분에서 필요한 함수들을 fork시킨다.

(2) 타이머 시그널

SDL에서 타이머는 signal 메커니즘을 이용한다. 즉 지정된 시간이 지나 시간만료가 되면 해당 타이머 signal을 생성하여 그 signal이 선언된 프로세스의 입력큐로 전달한다. signal은 하나의 SDL process 안에서 선언되고 그 process로만 전달되며 다른 process로는 전달될 수 없다. 즉, 타임아웃이 되어 시그널이 생성되면 외부 프로세스에서 전달되는 입력시그널과 동일하게 동작하게 된다.

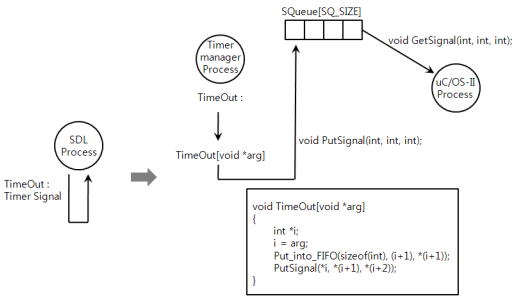


그림 6. SDL타이머 시그널의 uC/OS-II 타이머로의 변환
Fig. 6. Transformation of a SDL timer signal to a uC/OS-II timer

본 논문에서는 uC/OS-II 애플리케이션 라이브러리[17]에서 제공하는 타이머를 사용하여 타이머 시그널을 구현하였다. 이 소프트웨어 타이머는 타이머 관리를 위한 프로세스가 존재하여 타이머 인터럽트에 의해 여러 개의 타이머를 제어하는 방식으로 동작하므로 프로세스의 개수와 타이머 프로세스의 우선순위 등의 조건에 따라 타이머 만료시간이 정확하게 동작하지 않을 수도 있다. 하드웨어 타이머를 이용할 경우 좀 더 정확한 타이머 제어가 가능하지만 하드웨어에 의존적이 되므로 코드의 이식성이 떨어지고, 여러 개의 타이머를 사용할 경우 각 타이머를 제어하기 어렵다는 단점이 있다. 따라서, 본 논문에서 하드웨어 타이머를 사용하지 않고 소프트웨어 타이머를 사용함으로써 생기는 시간 정밀도의 차이가 개발하고자 하는 시스템의 실시간 처리에 문제를 일으키지 않는 시스템만으로 "코드 자동 생성"을 위한 플랫폼으로 고려하며, 이 경우 보다 간결하고 "이식성 높은 C코드 생성"을 위하여 타이머 매니저 프로세스가 제어하는 소프트웨어 타이머만을 이용할 수 있다. 만약 초정밀도의 실시간 처리가 요구되는 응용분야라면, 설계자는 자동 생성된 코드를 기반으로, 타이머가 사용되는 코드 부분을 하드웨어 타이머를 사용하여 재 수정함으로써 개발하고자 하는 시스템의 실시간 처리 조건을 만족 시켜야 한다. 그림 6에서와 같이 SDL의 한 process 내부에서 정의된 타이머 시그널은 타임아웃이 되면 그 프로세스로 시그널이 전송되는 행동은 uC/OS-II에서 타이머매니저 프로세스에서 타임아웃시 실행되는 함수 TimeOut에서 PutSignal() 함수와 Put_into_FIFO() 함수를 호출하여 시그널 전송을 하도록 구현하였다.

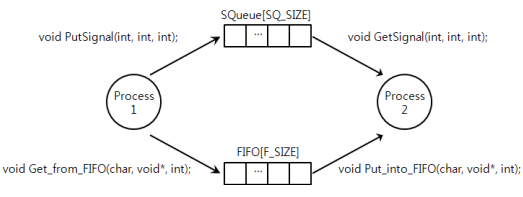


그림 7. SDL시그널 전송의 소프트웨어 구현
Fig. 7. Software implementation of SDL signal transmission

(3) IPC를 이용한 시그널(데이터)송수신

SDL 수행모델에서 프로세스간의 상호작용은 상태 전이 시 발생하는 출력시그널과 그 시그널에 종속되어 정의된 데이터를 다른 프로세스로 전송함으로써 이루어진다. 그림 7과 같이 운영체제상의 메시지메일박스와 FIFO를 이용하여 프로세스 간 상호작용을 구현하였다. SDL signal 전송을 위해 uC/OS-II에서의 메시지 메일박스 OS_EVENT* 타입의 SQueue[SQ_SIZE]를 정의하였고, 그 signal에 연관된 인자들을 전달하기 위해 전역 배열 FIFO[F_SIZE]를 정의하였다. 메시지 메일박스에 메시지를 송신·수신하기 위해서 uC/OS-II의 API인 OSQPost(), OSQPend()를 이용하였다. 본 코드생성기에서는 GetSignal() 함수와 PutSignal() 함수를 이용하여 SDL signal 통신을 구현하였다. 그림 8은 SDL signal과 데이터인자의 송수신을 위한 자료구조와 C코드이다. 여기서 SDL signal에 연관된 데이터는 먼저 전역변수에 저장하였다. 이는 하나의 signal이 임의의 SDL process에 전달되었을 때에 그 signal과 연관된 데이터는 이미 그 SDL process에 전달되었다는 가정을 만족시키기 위함이다.

```
void GetSignal(int Pid, int *arg_no)
{
        int result;
        INT8U err;

        result = (int) OSQPend(SQueue[Pid], 0, &err);
        result = (int) OSQPend(SQueuue[Pid], 0, & err);
}

void PutSignal(int signal, int Pid, int no_arg)
{
        int result, err;
        result = OSQPost(SQueue[Pid], (void *)signal);
        if (result = OSQFULL)
        {
           printf("Buffer for process %d is full\n", Pid);
        } else
        {
           result = OSQPost(SQueue[Pid],
                            (void *)no_arg);
           if (result = OSQFULL)
           {
             printf("Buffer for process %d if full!\n", Pid);
           }
        }
}
```

그림 8. 시그널 전송과 수신을 위한 PutSignal(), GetSignal() 함수
Fig. 8. PutSignal() and GetSignal() functions for transmitting and receiving signals

3.2 우선순위기반 스케줄링에서 SDL 수행모델의 적용

SDL 프로세스의 내부행위는 EFSM 모델을 기반으로 하고 있으므로 "상태전이 - 입력시그널 대기"의 두 단계를 반복하게 된다. 우선순위기반 스케줄링기법에서는 우선순위가 가장 높은 하나의 프로세스가 CPU 시간을 얻어 수행되며, 실행중인 프로세스가 시그널 혹은 세마포어가 활성화되기를 기다리게 되거나 프로세스 지연 함수 등이 실행되면 대기상태로 전환된다. 그 후에는 다음 우선순위를 가지며 준비리스트에 속한 프로세스가 CPU 시간을 할당받아 실행되게 된다. 이러한 우선순위기반 스케줄링기법에서 SDL 다중프로세스 수행모델은 다음과 같은 방식으로 실행된다.

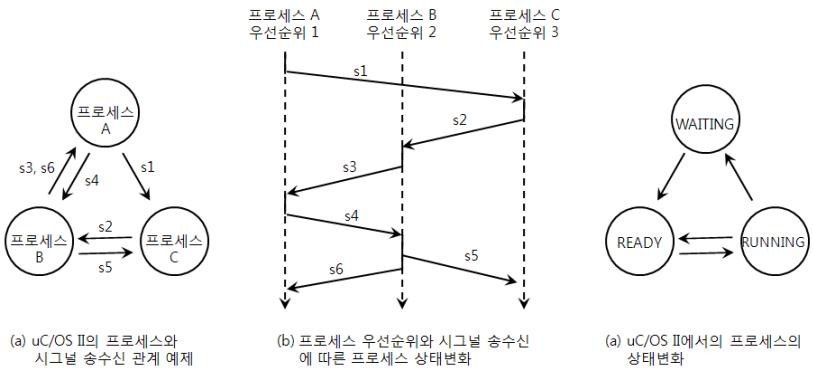


그림 9. 시그널송수신에 따른 우선순위기반 스케줄링 운영체제에서의 프로세스 상태변화
Fig. 9. Process state transitions according to the signal transmissions scheduled by a priority based OS

1. 현재 우선순위가 가장 높은 프로세스가 수행도중 상태 전이가 발생하여 출력시그널을 전송한다.
2. 출력시그널을 전송한 후 최우선순위 프로세스는 입력시그널을 대기하며 프로세스 대기상태로 전환되어 다음 우선순위를 가지는 프로세스가 수행될 수 있는 상태가 된다.
3. 차우선순위를 가진 프로세스가 대기상태에서 상태전이를 일으키는 시그널을 입력받았을 때에 프로세스의 상태는 실행상태가 되어 프로세스는 스케줄러에 의해 실행된다.
4. 차우선순위를 가진 프로세스가 대기상태에서 상태전이를 일으키는 시그널을 입력받지 못했다면 우선순위기반 스케줄링방법에 따라 그 다음 우선순위를 가지는 프로세스가 실행이 된다.

그림 9는 위의 시나리오를 표현한 것이다. 각 프로세스의 실행스레드에서 실선은 프로세스의 수행상태를 나타내고 점선은 프로세스가 대기상태에 있음을 나타낸다. 그리고 s1부터 s6은 각 프로세스가 주고받는 시그널들이다.

(1) 교착상태에 대한 고찰

SDL 사양기술을 C코드 변환기의 입력으로 사용하여 나온 결과코드는 원래의 사양기술에서 존재하지 않는 교착상태나 기아현상이 발생하지 않아야 한다. 본 논문에서 제안한 C코드 생성방법에 의해 생성되는 C코드는 원래의 사양기술에 존재하지 않는 교착상태나 기아상태가 존재하지 않는다.

(증명 1) 생성된 C코드에서 교착상태 발생가능성

한 시스템 S에 속하는 프로세스 P1, P2, …, Pi가 자원 R1, R2, …, RN을 사용하고 있을 때 시스템 S가 원하는 작업을 진행하지 못하고 멈춰있는 교착상태는 다음 세 가지 조건을 모두 만족할 경우 발생한다.

1. 프로세스 P1, P2, …, Pi가 각각 자원 R1, R2, …, Ri를 점유하고 있고 또한 P1은 R2를 대기하고 있으며, P2는 R3를, Pi는 R1을 대기하는 순환형태의 대기상태가 존재한다.
2. 각 프로세스 Pi는 자원 Ri를 한번 점유하면 Pi가 자발적으로 Ri를 릴리스하지 않는 한 계속해서 점유한다.

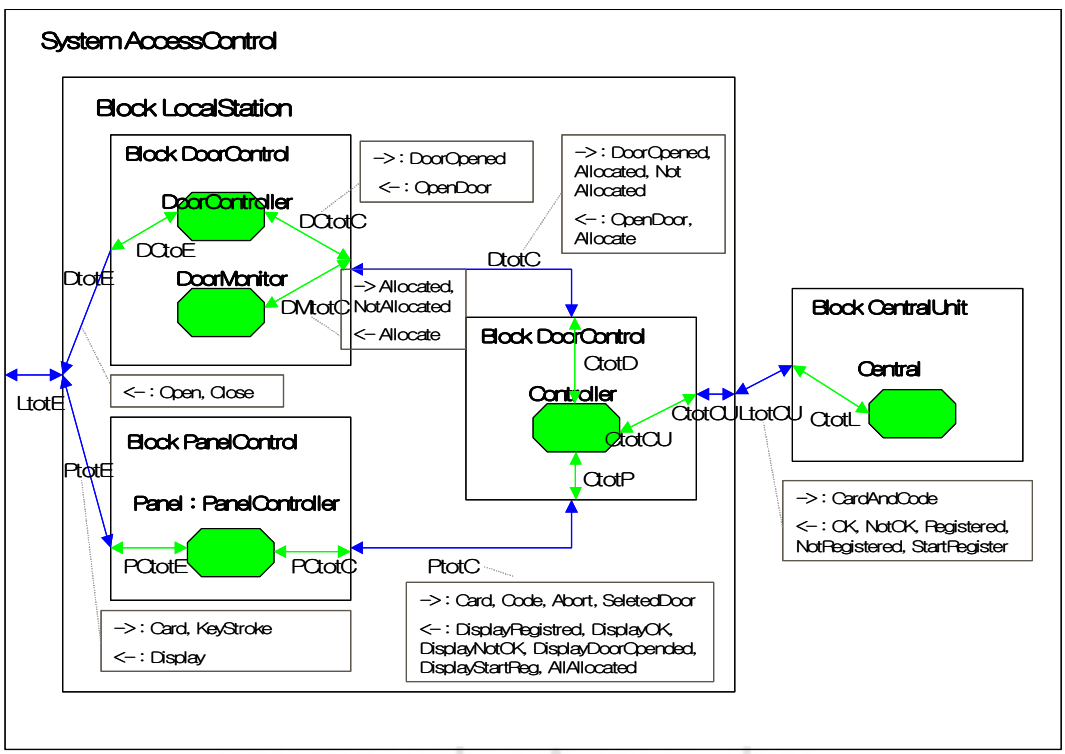


그림 10. Access Control 시스템의 SDL 블록구조도
Fig. 10. A SDL block structure of an access control system

3. 각 프로세스 Pi는 다른 프로세스 Pj에 의해 선점되지 않는다.

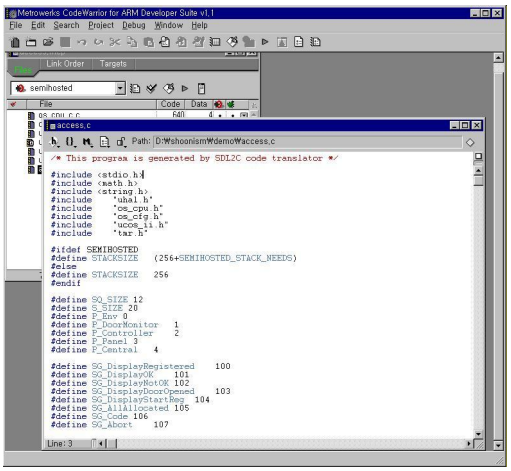


그림 11. SDL사양명세로부터 생성된 Access Control 시스템에 대한 C코드
Fig. 11. A C code for the access control system generated from SDL specification

본 C코드 생성기를 통해서 생성된 C코드의 경우, 한 프로세스의 입력시그널을 받아들이는 입력 큐가 항상 Full일 수는 없다고 가정한다. CPU 사이클이 자원이라 할 경우 최상위 우선순위 프로세스가 CPU사이클을 점유한다고 하더라도 "상태전이-시그널대기"의 순환을 반복하는 과정에서 반드시 임의의 시간 후에는 시그널대기 상태에 도달한다. 이 상태에서 프로세스는 운영체제의 프로세스 상태 중 대기 상태에 있게 되므로 두 번째 조건인 자원 점유 후 해당 작업이 끝날 때까지 계속해서 자원을 점유하는 상태가 발생하지 않는다. 따라서 원래의 SDL 사양에서 교착상태가 발생하지 않는다면 본 C코드생성기를 통해서 생성된 C코드에서도 교착상태가 발생하지 않는다는 것이 보장된다.

이와 같이 우선순위기반 운영체제상의 소프트웨어로 구현된 SDL 프로세스는 원래의 SDL사양기술에서 교착상태(deadlock)가 발생하지 않는다면 변환된 소프트웨어 환경에서도 교착상태가 발생하지 않는다는 것이 보장된다.

(2) 기아상태(starvation) 발생 가능성에 대한 고찰

원래의 SDL 수행환경은 모든 SDL process들이 병렬적으로 동시에 수행될 수 있는 이상적인 환경을 가정하고 있으나, 변환된 C코드가 수행되는 환경은 하나의 마이크로프로세스에서 운영체제의 도움을 받아 다중프로세스가 실행되는 환경이다. 특히 라운드로빈 스케줄링방식이 아닌 우선순위기반 스케줄링방식을 사용하는 대부분의 실시간 운영체제에서는 우선순위할당에 따른 기아현상이 발생할 가능성이 존재한다.

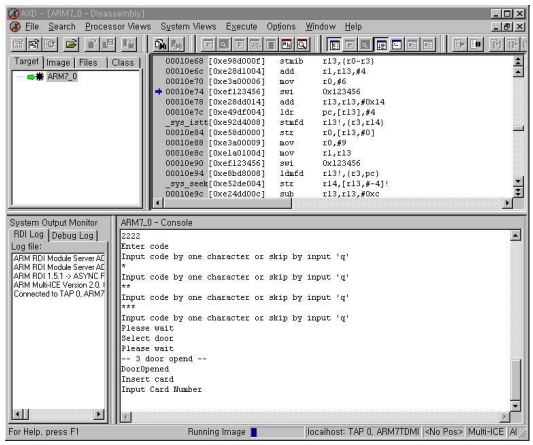


그림 12. Access Control 시스템의 실행화면
Fig. 12. A screen shot of running the access control system

Ⅳ. 코드변환 예제

SDL 사양기술로부터 C코드를 생성하는 SDL2C 변환기의 입력으로 Access Control 시스템을 사용하였다. 이 시스템은 출입문 접근시에 카드키를 이용하여 인증된 사용자에게 출입을 허용하는 시스템으로 전형적인 제어중심 시스템이다. 그림 10은 Access Control 시스템의 SDL 블록구조도이다.

SDL2C 변환기를 통하여 C코드를 생성한 후, 프로세서로는 ARM7TDMI 코어를 사용하고 운영체제로 uC/OS-II를 운영하는 내장형시스템 구조에서 생성된 C코드를 디버깅하고 실행하였다. 컴파일 및 디버깅에는 ARM Development Suite상의 C 컴파일러와 AXD(Arm Extended Debugger)를 이용하였다. 그림 11은 SDL 사양명세로부터 SDL2C 변환기를 거쳐 얻은 AccessControl 시스템에 대한 C코드를 ADS의 Code Warrior상에서 컴파일하기 전의 화면이고, 그림 12는 실제 Access Control 시스템을 ARM7TDMI 프로세서에 적재하여 실행중인 화면이다.

Ⅴ. 결론

SDL 사양기술로부터 실시간 운영체제에서 수행될 C코드를 자동으로 생성하는 방법을 제시하였다. 본 논문에서 제시

한 SDL2C 변환 방법은 현재 내장형시스템의 소프트웨어 개발환경을 고려하여, uC/OS-II에서 사용하는 우선순위기반 선점형 스케줄링과 적절히 융화될 수 있는 기본틀을 제시하고 그에 맞게 uC/OS-II에서 제공하는 운영체제 서비스 함수들을 이용하여 C코드를 생성한다. 따라서 생성된 C코드는 원래의 SDL 사양과 동일한 동작을 수행하며, 운영체제 기능들을 이용하여 효율적으로 수행된다. 이 때 timer, 다중프로세스 수행, 시그널 전송과 같이 운영체제가 제공하는 서비스 함수들이 사용되어 SDL 수행의미(semantic)가 효율적으로 구현될 수 있다.

본 C코드 생성기를 이용하여, 시스템수준언어를 이용한 추상적인 시스템의 동작기술을 실제 구현언어인 C로 바꾸고 그 과정에서 uC/OS-II라는 내장형시스템용 실시간운영체가 제공하는 서비스함수를 포함하는 C코드를 생성하였다. 따라서 시스템수준기술언어-하드웨어 또는 소프트웨어 구현언어(C 또는 VHDL)로 연결되는 설계흐름의 한 축을 제공하며 앞서 언급한 바와 같이 시스템수준에서 조기에 대상시스템을 검증하고 복잡한 시스템을 구현언어를 이용하여 설계할 때 보다 자동화된 코드 생성기를 이용함으로써 보다 신속한 시스템 설계를 통한 개발품의 조기시장진입을 가능하게 한다.

현재는 초기사양을 동일한 동작을 수행하는 C코드 생성에 주안점을 두고 있지만, 생성된 소프트웨어 시스템의 응답성 및 CPU 이용도 등을 척도로 하는 고성능 실시간 소프트웨어 생성이 실시간 운영체제용 C코드 생성을 위해 앞으로 고려해야 할 사항이다. 또한 SDL로 시스템을 기술한 후 하드웨어-소프트웨어 통합설계 시에 이용할 경우 하드웨어 부분과의 인터페이스를 자동으로 생성해 줄 수 있는 방법도 차후 연구과제이다.

참고문헌

〔1〕 Daniel D. Gajski, Frank Vahid, Sanjiv Narayan, and Kie Gong, "Specification and Design of Embedded System", Prentice Hall , 1994

〔2〕 J. Staunstrup, W. Wolf, "Hardware/Software Co-Design : Principles and Pracitce", Kluwer Academic Publishers, 1997

〔3〕 J. Ellsberger, D. Hogrefe, A. Sarma, "SDL : Formal Object-oriented Language for Communication Systems", Prentice Hall, 1997

〔4〕 "Telelogic Tau 3.2 User's Manual", Telelogic AB, Sweden, 1997

〔5〕 N. E. Zergainoh, G. F. Marchioro, and A. A Jerraya, "Using SDL for Hardware/Software Co-Design of an ATM Network Interface Card", Proceedings of 1st Workshop of the SDL Forum Society on SDL and MSC, Berlin, Germany, 1998

〔6〕 G. F. Marchiroro, J.M. Daveau, T.B. Ismail, A. A. Jerraya, "Transformational partitioning for codeign", IEE proceeding of Computers and Digital Techniques, pp. 181-195, 1998

〔7〕 Y. Huang, M. Hughes, "Using SDL in embedded systems design: a tool for generating real-time OS pSOS based embedded systems applications software", Proc. of the 11th IEEE Workshop on Real-Time Operating Systems and Software, pp. 39 -43, 1994

〔8〕 F. Slomka, M. Dorfel, and R. Munzenberger, "Generating Mixed Hardware/Software Synthesis from SDL Specificatoins", Proceedings of the Ninth International Symposium on Hardware/Software Codesign, pp. 116 -121, 2001

〔9〕 J. M. Alvarez, M. Diaz, L. Llopis, E. Pimentel, J.M. Troya, "Deriving hard real-time embedded systems implementations directly from SDL specifications", Proc. of the 9th International Symposium on Hardware/Software Codesign, pp. 128-133, 2001

〔10〕 J. M. Alvarez, M. Diaz, L. Llopis, E. Pimentel, and J.M. Troya, "Schedulability Analysis in Real-Time Embedded Systems Specified in SDL", Proceedings of 25 IFAC/IFIP Workshop on Real-Time Programming, Elsevier pp. 125-131, 2000.

〔11〕 Jose Maria Alvarez, Manuel Diaz, Luis Llopis, Ernesto Pimentel, and J.M. Troya, "An Analyzable Execution Model for SDL for Embedded Real-Time Systems", In proceedings of 24 IFAC/IFIP Workshop on Real-Time Programming, pp. 117-123, Elsevier, 1999

〔12〕 J. M. Daveau, G. F. Marchioro, C. A. Valderrama, and A. A. Jerraya, "VHDL genrations from SDL

specifications", Proceedings f the IFIP conference on Hardware Description Languages and their Applications, pp. 192-201, 1997

〔13〕 O. Bringmann, A. Muth, F. Slomka, W. Rosenstiel, G. Farber, and R. Hofmann, "Mixed Abstraction Level Hardware Synthesis from SDL for Rapid-Prototyping", Proceedings of IEEE International Workshop on Rapid System Prototyping, pp. 114-119, 1999

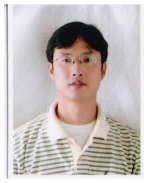
〔14〕 C. Drosos, M. Zayadine, D. Metafas, "Real-Time Communication Protocol Development Using SDL for an Embedded System On Chip Based on ARM MicroController", 13th Euromicro Conference on Real-Time Systems, pp. 89-94, 2001

〔15〕 J. Ellsberger, D. Hogrefe, A. Sarma, "SDL : Formal Object-Oriented Language for Communicating Systems", Prentice Hall, 1997

〔16〕 Jean J. Labrosse, "MicroC/OS-II : The Real-Time Kernel", R&D Books, 1999

〔17〕 Jean J. Labrosse, "Embedded System Building Blocks", R&D Books, 2000

저 자 소 개

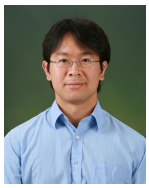


곽 상 훈

2000년 2월 광주과학기술원 정보통신공학과 공학석사

2000년 3월~현재 광주과학기술원 정보통신공학과 박사과정

〈관심분야〉 VLSI/CAD 설계, 임베디드 시스템 설계, 비동기 회로 설계



이정근

2005년 2월 광주과학기술원 정보통신공학과 공학박사

2005년 5월~2007년 6월 University of Cambridge, 컴퓨터 연구소, 박사후 연구원

2007년 7월~2008년 2월 광주과학기술원 정보통신공학과 연구교수

2008년 3월~현재 한림대학교 컴퓨터 공학과 조교수

〈관심분야〉 임베디드 시스템 설계, 비동기 회로 설계, 병행 시스템 설계, 컴퓨터 구조, 설계 자동화