DOI: 10.3745/KIPSTD.2009.16D.6.871

임베디드 소프트웨어의 소모전력 분석을 위한 에너지 컴포넌트 라이브러리

김 두 환[†] · 홍 장 의[††]

요 약

임베디드 소프트웨어의 복잡성 및 대형화로 인하여 기능적 요구사항뿐만 아니라 소모전력 관리와 같은 비기능적 요구사항이 중요시되고 있다. 본 연구에서는 소스 코드 기반의 소모 전력을 분석하는 기존의 접근 방법과 달리 UML 2.0 기반의 모델을 중심으로 하는 임베디드 소프트웨어의 소모전력 분석 기법을 제시한다. 특히 소모 전력 분석을 위해 요구되는 에너지 컴포넌트에 대한 라이브러리 구축에 대하여 제시한다. 제시하는 라이브러리는 모델 기반의 소모전력 분석을 가능하도록 지원할 뿐만 아니라, 임베디드 응용의 변경에 따른 라이브러리 적용이 쉽게 이루어질 수 있다는 장점을 제공한다.

키워드 : 임베디드 소프트웨어, 소모전력 분석, 에너지 라이브러리

Energy Component Library for Power Consumption Analysis of Embedded Software

Doo-Hwan Kim[†] · Jang-Eui Hong[††]

ABSTRACT

Along with the complexity and size growth of embedded software, it is critical to meet the nonfunctional requirements such as power consumption as well as functional requirements such as correctness. This paper, apart from the existing studies of source code-based power analysis, proposes an approach of model-based power analysis using UML 2.0. Specially, we focus on the development of energy library to analyze the power consumption of embedded software. Our energy library supports model-based power analysis, and also supports the easy adaption for the change of embedded application.

Keywords : Embedded Software, Power Consumption Analysis, Energy Library

1. 서 론

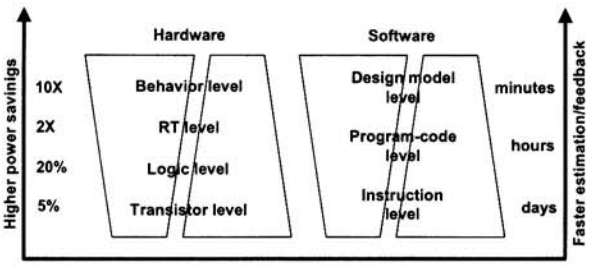
임베디드 소프트웨어는 적용 영역에 따라 기능적 요구사항의 만족뿐만 아니라, 비 기능적 요구사항의 만족이 매우 중요하다. 특히 DMB 폰, MP3 플레이어, 센서 네트워크 등과 같은 모바일 장치에서는 소모전력 특성을 만족해야 하는 비기능적 요구사항의 중요성이 강조되고 있다.

임베디드 시스템에서의 소모 전력의 중요성은 오래전부터 강조되어 왔으며, 보다 긴 수명의 베터리 개발과 저전력을 소모하는 하드웨어 소자 개발에 많은 연구들이 있었다[1]. 그러나 최근 5~6년 전부터는 임베디드 시스템의 서비스 제공에 있어서 소프트웨어가 차지하는 비중이 증가하고, 다양화, 복잡화 되면서 소프트웨어 자체에 대한 소모 전력을 감소시키기 위한 노력이 강조되었다[2].

임베디드 소프트웨어에 대한 소모전력 분석기법에 대한 연구들은 기존에 명령어 및 소스코드 기반으로 수행되어 왔다. 비록 코드 수준의 소모전력 분석이 정확한 분석 결과를 제공한다는 장점을 갖지만, (그림 1)에서 보는 바와 같이 소모 전력 분석을 위해 소요되는 시간이 길고, 분석 결과의 반영(feedback)을 위한 노력도 많이 요구된다[3].

반면 설계 모델의 소모전력 분석은 코드 기반에 비하여 분석결과의 정확성은 다소 떨어질 수 있으나, 분석에 소요되는 시간이 단축되고 피드 백 노력이 감소할 수 있다는 장점이 있다. 이러한 장점으로 인하여 설계 단계에서 임베디드 소프트웨어에 대한 소모전력 분석은 코드 기반의 소모전

※ 본 논문은 2008학년도 충북대학교 학술연구지원 사업의 연구비지원에 의해 연구되었음.
† 준 회 원 : 충북대학교 컴퓨터과학과 석사과정
†† 종신회원 : 충북대학교 컴퓨터공학 부교수(교신저자)
논문접수 : 2009년 6월 23일
수 정 일 : 1차 2009년 8월 5일
심사완료 : 2009년 8월 22일



(그림 1) HW, SW 추상화에 따른 전력분석 효과[3]

력 분석에 앞서 선행적으로 전력 소모에 대한 요구사항을 점검할 수 있다는 유용성을 제공한다.

설계 모델 기반의 소모전력 분석기법은 소스 코드가 표현하고 있는 소프트웨어의 행위적 특성을 보다 상위 수준으로 추상화하고, 이를 기반으로 하는 소모전력 분석 예측 기법[3]이다. 이를 위해서는 추상화된 설계 요소와 하위 수준의 소프트웨어 행위를 연결하기 위한 에너지 라이브러리 구축이 요구된다.

임베디드 소프트웨어 소모전력 분석을 위하여 많은 연구들이 라이브러리 기반의 분석 기법을 사용하고 있다. Yue[4]나 Qu[5]의 연구에서는 응용 객체(Entity) 또는 프로그램 코드에 대한 소모 전력을 산출하고, 이를 라이브러리로 구축하여 소모 전력을 분석하였으며, Muttreja[6]와 Tan[7-8]의 연구에서는 프로그램 코드를 분석하여 매크로 함수를 도출하고, 함수에 의한 소모 전력을 분석하도록 라이브러리를 구축하였다.

그러나 이러한 연구들이 갖는 공통점은 전력 소모량 예측을 위하여 프로그램 코드를 수행한 결과를 근거로 소모 전력량을 수집하고, 소모 전력 분석시 이를 참조하는 방식으로써, 라이브러리의 내용이 특정 응용에 의존적이라는 것이다. 이는 새로운 응용에 대한 소프트웨어 소모 전력 분석을 위하여 별도의 데이터 수집이 요구될 수 있다는 것을 의미한다. 따라서 본 연구에서는 임베디드 소프트웨어 개발과정에서 작성되는 UML 다이어그램을 이용하여 소모 전력을 분석하도록 함으로써, 에너지 라이브러리 구성 요소들을 명령어 수준보다 추상화 시킨 모델 요소 단위로 구성하였다. 비록 모델 요소 단위의 소모 전력을 프로그램 코드를 수행하여 얻은 결과를 기반으로 하고 있지만, 코드가 아닌 모델 요소를 기반으로 에너지 라이브러리를 구성하여, 라이브러리 컴포넌트를 효율적으로 관리할 수 있을 뿐만 아니라 새로운 응용에 대하여 별도의 노력없이 라이브러리를 활용하여 소모 전력을 분석할 수 있는 장점을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 라이브러리 기반의 소모전력 분석 기법에 대한 기존 연구의 분석, 3장에서는 본 연구에서 제시하는 모델 기반의 소모전력 분석 절차에 대한 간략한 기술, 그리고 4장에서는 에너지 라이브러리의 구성 및 구축 절차에 대하여 제시한다. 5장에서는 라이브러리를 적용한 사례를 제시하여 구축한 라이브러리의 유용성을 보인다. 그리고 6장에서는 결론 및 향후 연구에 대하여 제시하였다.

2. 관련 연구

임베디드 소프트웨어의 소모 전력을 분석하기 위하여 모델 기반 및 라이브러리 구축을 통해 소모전력을 분석하는 연구들은 쉽게 찾아보기 어렵다. 이들 중에서 Tan의 연구[7]는 SAG (Software Architecture Graph)를 이용하여 응용 소프트웨어의 아키텍처를 표현한 후, 그래프를 구성하는 노드, 즉 태스크간의 통신에 대한 소모 전력을 측정하였다. 태스크간의 통신에 대한 소모 전력을 분석하기 위하여 운영체제가 제공하는 필수적인 명령어(또는 함수)들에 대한 소모 전력량을 라이브러리로 구축하였다. 이 연구는 라이브러리 활용에 있어서 IPC를 지원하는 명령어 또는 함수 단위로 라이브러리를 접근하는 특성을 갖는다.

Jun의 연구[9]는 소프트웨어 컴포넌트가 갖는 내부 행위를 오토마타 네트워크로 표현하고, 오토마타를 구성하는 노드와 노드간의 전이에 대한 에너지 소모량을 임의로 지정하여 에너지 오토마타를 모델링 하였다. 에너지 오토마타 모델은 합성되어 전체 소프트웨어에 대한 소모전력을 산출하기 위해 사용된다. Jun의 연구에서는 특별히 라이브러리를 활용하지는 않았지만, 오토마타 모델 기반의 소모전력 분석을 수행한 대표적인 연구이다.

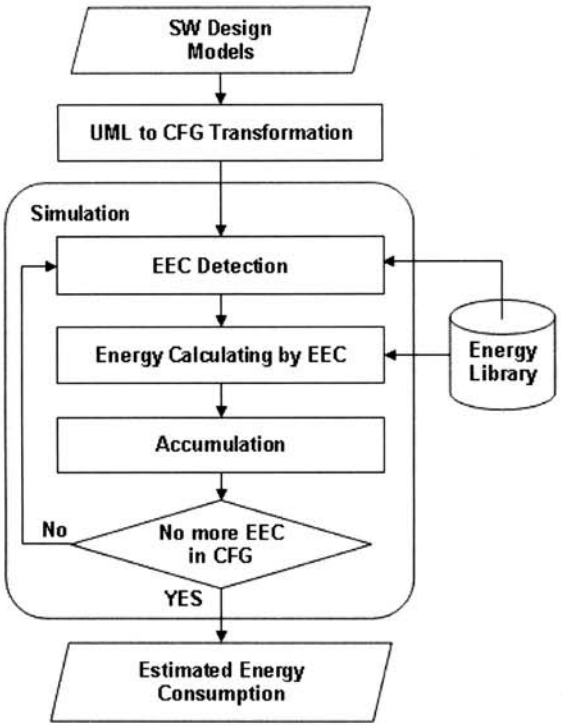
Qu의 연구[5]는 응용 프로그램 코드를 시스템 함수와 사용자 정의 함수로 분류하여 각 함수들에 대한 전력 소모량을 실측하고 이를 Power data bank라는 라이브러리에 저장한다. 저장된 측정 자료는 소프트웨어가 소모하는 전력량을 산출하기 위해 코드 기반의 시뮬레이션 과정에서 기반 자료로 사용한다. Qu의 연구에서도 소스 코드의 소모전력을 분석하기 위하여 함수 단위로 라이브러리를 접근한다는 특성이 있다.

또한 Muttreja의 연구[6]에서는 먼저 주어진 프로그램 모듈에 대한 전력 소모량을 나타내는 매크로 함수를 생성하고, 이를 기본적인 명령어들과 함께 라이브러리에 유지 관리한다. 소프트웨어에 대한 전력 소모량을 산출하기 위해 소스 코드 시뮬레이션을 수행하며, 이 과정에서 명령어 및 매크로 함수 단위로 라이브러리를 접근한다.

본 논문에서 제시하는 연구는 UML 설계 모델을 입력으로 소모 전력을 분석한다는 점이 기존의 연구들과 큰 차별점이다. 기존의 연구와 비교할 때, 라이브러리를 구성하는 정보 수집의 접근 방법이 기존 방법과 유사한 방식으로 이루어지지만, 기존의 라이브러리들이 명령어나 함수들의 소모전력 정보를 직접 사용하도록 제공하는 반면, 본 연구에서는 UML 모델 요소 단위로 라이브러리를 접근할 수 있도록 하였다. 이는 명령어나 함수의 정보를 설계 모델 요소와 매핑하는 에너지 모델을 라이브러리가 포함하고 있기 때문이다.

3. 모델 기반 소모전력 분석

서론에서 간략히 언급했듯이 본 연구는 임베디드 소프트



(그림 2) 모델기반 소모전력 분석 절차

웨어의 소모 전력을 예측하기 위하여 모델 기반의 분석 방법을 제안하고 있다. 특히 최근 재사용 및 제품공학의 도입으로 인하여 객체지향 개념이 널리 활용되고 있는 추세이기 때문에 UML 기반의 설계 모델을 활용하였다. (그림 2)는 본 연구에서 정의한 모델 기반의 소모전력 분석 절차이다.

(그림 2)에서 제시한 소모전력 분석 절차의 각 단계별 개략적인 설명은 다음과 같다.

3.1 소프트웨어 설계 모델 입력

본 논문에서 제안하는 임베디드 소프트웨어의 전력 소모량 예측은 UML 2.0의 행위 모델(behavioral model)을 중심으로 수행한다. UML Sequence diagram은 소프트웨어가 갖는 세부 행위를 시간의 흐름에 따라 순차적으로 표현할 수 있으며, Action Language를 이용하여 특정 함수(메소드)의 내부 행위를 기술할 수 있다. 따라서 본 연구에서는 UML의 class diagram(CD), sequence diagram(SD), Interaction overview diagram(IOD), 그리고 action language(AL)를 기반으로 임베디드 소프트웨어의 행위를 모델링하고, 이를 기반으로 소모 전력을 분석한다. 이와 같은 행위 기반의 소모전력 분석은 임베디드 소프트웨어가 실세계에서 수행할 때, 발생하는 기능 수행의 과정을 직접적으로 표현할 수 있다는 장점을 갖는다.

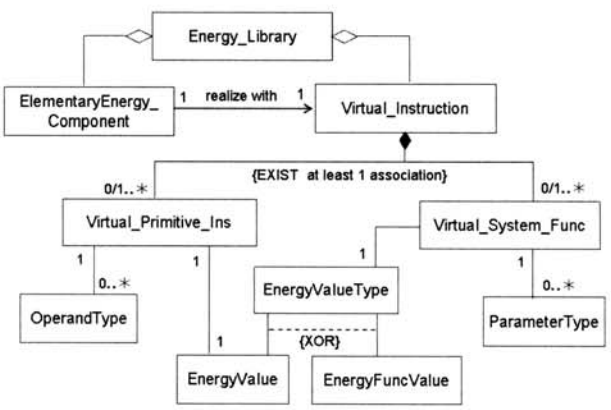
3.2 UML로부터 CFG 생성

UML을 이용하여 작성된 소프트웨어 설계 모델을 입력으로 소모전력 분석을 위한 CFG(Control Flow Graph)[10]를 생성한다. CFG는 UML 모델이 갖는 소프트웨어 행위모델의 2차원적 요소들을 세분화하여 순차적인 제어 흐름으로 표현한 그래프이다. 변환된 CFG는 분석을 위해 적합하며, 전력 소모 인자를 식별하기에 매우 유용한다. UML 모델로부터의 CFG 생성 알고리즘은 Garousi의 연구[10]에 자세하게 제시되어 있다.

3.3 에너지 라이브러리

에너지 라이브러리는 소모전력 분석을 위해 선행적으로 구축되어야 하는 인프라의 구성요소이다. 본 논문에서 제시하는 에너지 라이브러리는 EEC(Elementary Energy Component)라는 에너지 소모의 기본 단위들로 이루어져 있다. 에너지 라이브러리의 구성 요소는 (그림 3)과 같이 표현할 수 있다.

에너지 라이브러리는 크게 행위 유발 요소인 EEC와 EEC의 에너지 소모량을 결정하기 위한 가상 명령어(VI, Virtual Instructions) 그리고 이들 간의 매핑 정보들로 구성된다. 소프트웨어 행위를 표현하는 CFG로부터 EEC를 탐색하면, 이를 이용하여 에너지 라이브러리에 저장된 가상 명령어의 에너지 값을 얻게 된다.



(그림 3) 에너지 라이브러리 메타 모델

3.4 EEC 탐색

Garousi[10]가 제시한 것과 같이 UML 모델의 한 요소는 CFG에서 다수의 노드로 표현되기 때문에 생성된 CFG 노드를 따라 가면서 EEC를 찾아낸다. 하나의 EEC에 대한 탐색은 CFG의 시작 노드를 출발하여 해당 EEC가 끝나는 지점까지 검색하며, 검색된 EEC는 소모 전력 값을 얻기 위하여 에너지 라이브러리로 보내진다.

3.5 전력 소모량 산출

에너지 라이브러리로부터 얻은 전력 소모량은 CFG의 최종 노드에 도착할 때까지 누적된다. CFG로부터 더 이상의 EEC가 존재하지 않는 최종 노드에 도달하게 되면, 소프트웨어의 전력 소모에 대한 최종 합을 구할 수 있게 된다. 구해진 최종 소모량은 사용자에게 다양한 형태로 그래프화하여 출력한다.

4. 에너지 라이브러리

모델 기반 임베디드 소프트웨어의 소모 전력 예측기법에서 에너지 라이브러리는 모델 요소들에 대한 에너지 소모량에 대한 정보를 유지 관리하는 정보 저장소이다. 실제로 에너지 소모량을 예측하기 위해서는 이러한 라이브러리를 사용하지 않더라도 수리적 모델을 정의하여 직접 예측하는 방법[7, 11]이나 실측 자료를 통해 생성한 모델을 직접적으로 적용하는 방법[12-13] 등의 여러 가지 방법들이 있으나, 본 연구에서는 소프트웨어 모델이 변화하더라도 별도의 업데이트 없이 라이브러리에 저장된 정보만을 참조하여 기민하게 적용할 수 있도록 에너지 라이브러리(ENEL, ENErgy Library)를 설계 구축하였다.

4.1 라이브러리 구성 요소

에너지 라이브러리의 구성요소는 3.3절에서 간략히 언급하였다. (그림 3)에 제시한 것과 같이 라이브러리는 크게 EEC와 VI, 그리고 이들 간의 매핑 테이블로 구성된다.

○ EEC(Elementary Energy Component)

EEC는 UML 모델 요소들의 행위 분석을 통하여 도출된 전력 소모 산출의 기본 단위이다. UML 2.0 Superstructure[14]에 정의된 SD와 IOD, 그리고 AL을 구성하는 모델 요소들을 파싱하여 행위 유발 요소를 식별하고, 이들을 EEC로 정의한다. 도출된 EEC의 목록은 <표 1>과 같으며, 이들에 대한 세부 도출 과정은 4.2절에 정의하였다.

〈표 1〉 정의된 EEC 목록

Diagram	EEC
SD	Message(send, receive), MessageSort(SynchCall, AsynchCall, Signal), InteractionOperator(alt, opt, loop, par)
IOD	Fork, Join(wait), Invocation(Create), InterruptableActivityRegion
AL	ExecutionOccurrence, ValueSpecification, Constraint 등에 포함되거나 모델의 상세 정보를 기술하기 위해 사용되는 단위 연산자 (+, -, *, /, %, <<, |, !, ==, &&, >= 등)와 함수연산자(read, write, malloc, 등).

○ 가상 명령어(Virtual Instruction)

가상 명령어는 소프트웨어 명령어가 실행될 때, 하드웨어의 동작에 따른 소모 에너지를 측정하기 위해 정의된 일반화된 명령어들의 집합으로써 VPI(Virtual Primitive Instruction)와 VSF(Virtual System Function)로 구성된다. VPI는 ARM 프로세서 상에서 AL을 컴파일 한 결과로 생성된 기계어를 참조하여 정의하였고, VSF는 임베디드 리눅스에서 제공하는 시스템 함수들을 참조하여 정의하였다.

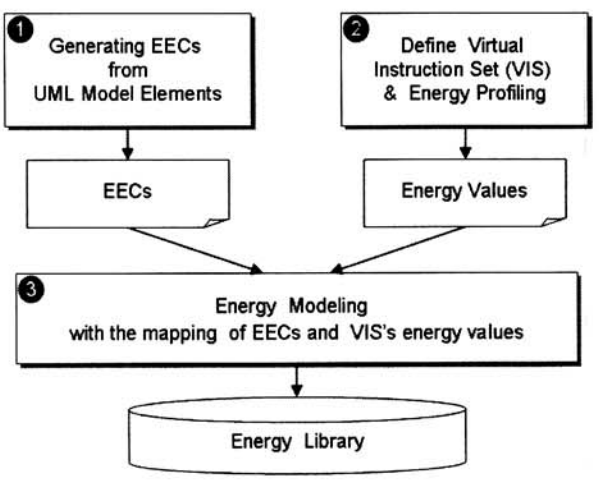
에너지 라이브러리에 포함된 VPI는 Load, Add, Store 등의 기본 명령어 16개를 포함하고 있으며, VSF는 wait(), fork() 등을 포함한 시스템 함수 22개를 포함하고 있다. 이들 각각의 명령어들은 실 전력 소모량의 값을 갖는데, 이들 값은 Public Domain에서 활용되고 있는 EMSIM 2.0 시뮬레이터[15]를 사용하여 측정하였다. 이에 대한 세부 사항은 4.2절에서 설명한다.

○ 매핑 테이블

UML 모델로부터 식별된 EEC는 VI를 구성하는 VPI와 VSF로 표현될 수 있다. 예를 들면, UML Sequence diagram에 나타나는 메시지 패싱은 VSF를 구성하는 msgsnd() 함수와 msgrcv() 함수로 매핑되고, "alt"와 같은 Sequence diagram의 Combined fragment는 load, compare, branch와 같은 VPI의 조합으로 표현된다. 이러한 조합의 정보가 매핑 테이블에 유지 관리된다.

4.2 에너지 라이브러리 구축절차

본 논문에서 제시하는 에너지 라이브러리, ENEL을 구축하기 위한 절차는 (그림 4)와 같이 크게 (1) EEC 식별 단계, (2) VI 에너지 프로파일링 단계, 그리고 (3) 에너지 모델링 단계로 구성된다.



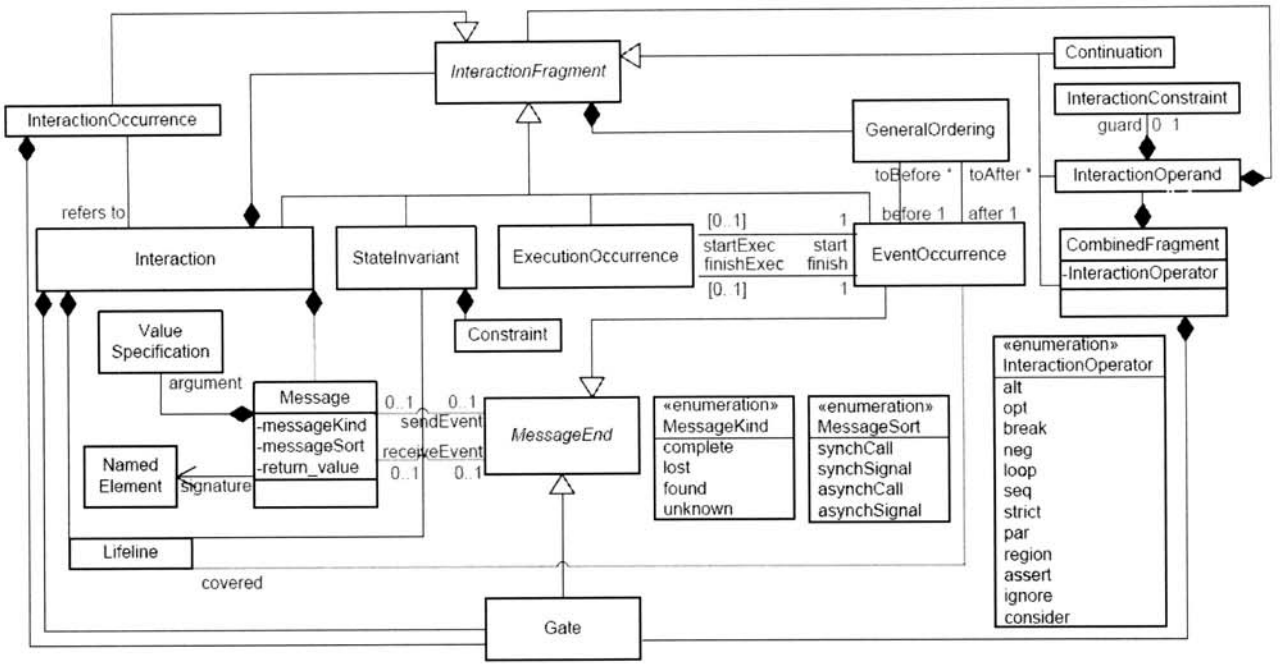
(그림 4) 에너지 라이브러리 구축 과정

4.2.1 EEC의 식별 단계

EEC는 모델 기반의 소모 전력 예측을 위한 기본 단위로써, EEC 식별은 UML 다이어그램의 메타 모델을 구성하는 클래스 중에서 행위 요소를 갖는 모든 Concrete 클래스를 찾아내는 것이다. (그림 5)는 UML Sequence diagram의 메타 모델[14]을 보여 준다.

UML 다이어그램으로부터 EEC를 식별하는 과정은 다음과 같다.

(1) 먼저 메타 모델을 구성하는 클래스들을 유형별로 분류한다. 클래스의 유형은 Abstract 클래스, Control structure 클래스, Behavior execution 클래스, None behavior 클래스, Action Language 클래스 등이다.



(그림 5) UML Sequence Diagram 메타 모델

(2) 클래스 유형 분류에서 Abstract 클래스와 None behavior 클래스의 유형은 전력 소모를 유발하는 행위를 갖지 않기 때문에 EEC 대상에서 제거된다.

(3) 다음은 클래스 유형별로 모델 요소를 세분화한다. 즉, 하나의 메시지는 메시지 송신과 메시지 수신으로 구분될 수 있다.

(4) 마지막으로 메타 모델에 제시된 <<enumeration>> 타입의 속성들을 각각 EEC의 대상으로 세분화 한다.

〈표 2〉 Sequence diagram으로부터의 EEC

메타모델요소	EEC	설명
InteractionFragment	X	Abstract Class
InteractionOccurrence	X	Abstract Class
Interaction	X	Abstract Class
Message	O	Behavior Exec.
StateInvariant	X	None Behavior
ExecutionOccurrence	O	Action Language
EventOccurrence	X	Abstract Class
Continuation	X	Abstract Class
InteractionOperand	X	Abstract Class
CombinedFragment	X	Abstract Class
MessageSort	O	Behavior Exec.
InteractionOperator	O	Control Structure
Lifeline	X	Abstract Class
ValueSpecification	O	Action Language
Constraint	O	Action Language
Gate	X	Abstract Class

이와 같은 과정을 거쳐 (그림 5)의 Sequence diagram 메타 모델에 대한 EEC 식별 결과는 <표 2>와 같다.

(그림 5)로부터 메타 모델의 Interaction 클래스는 Message, Lifeline, Gate, 그리고 InterationFragment의 집합으로 표현된다. 그러나 Interaction 클래스에서 응용 모델링 과정에 표현되는 행위 유발 요소는 Messge 뿐이며, 나머지는 행위를 포함하지 않는 추상 클래스들이다. 클래스 StateInvariant는 행위 유발 요소가 아니며, ExecuOccurrence 클래스는 Action language에 의해 표현되는 특정 행위를 포함하기 때문에 행위 유발 요소에 포함된다. InteractionFragment로부터 상속받는 클래스 Continuation, InteractionOperand, CombinedFragment 중에서 클래스 Continuation은 단순 연결자를 의미하기 때문에 제외되며, InteractionOperand 클래스는 다시 InteractionFragment들의 모임으로 정의되기 때문에 응용 모델에 나타나는 직접적인 요소가 아니다. 따라서 클래스 CombinedFragment에 나타나는 alt, opt, loop 등의 InteractionOperator 속성 변수가 행위 유발 요소가 된다. 이렇게 응용 모델에 표현되는 행위 유발 요소들은 소모 전력을 분석하기 위한 기본 단위인 EEC가 된다.

위와 같은 방법에 의하여 IOD로부터 추출된 행위 유발 요소는 ForkNode, JoinNode, InvocationNode, InterruptableActivityRegion, Constraint 클래스가 해당되며, 이들은 대체적으로 시스템 함수에 의해 정의될 수 있다. EEC의 식별 결과는 <표 1>에 정의 하였다. 본 논문에서 제시하는 EEC은 SD에 대하여 9개, IOD에서 4개, 그리고 AL으로부터 32개가 정의되었다. 물론 AL 차원에서 추가적인 C 언어 기반의 라이브러리 함수들을 더 포함하여 정의할 수 있기 때문에 전체 EEC 수가 한정되는 것은 아니다.

4.2.2 VI 에너지 프로파일링 단계

에너지 프로파일링 단계에서는 정의된 가상 명령어에 대한 소모 전력을 측정한다. 이를 위해 먼저 가상 명령어를 정의하고, 이를 시뮬레이션하여 소모 전력량을 산출한다.

(1) VI 집합은 명령어가 실행될 때, 하드웨어의 동작에 따른 소모 전력을 측정하기 위해 정의된 일반화된 명령어들의 집합으로써, Bammi[16]가 제시한 가상 명령어의 개념을 VPI와 VSF로 확장한 것이다. 본 연구를 통해 식별된 가상 명령어는 최대한 하드웨어 의존성을 줄이기 위해 일반화되었으며, 이는 추후 모델 수준에서의 변경 영향을 줄이는데 그 목적이 있다. 정의된 VI 목록은 <표 3>과 같다.

(2) <표 3>에 제시된 VI들에 대한 에너지 소모량을 측정하기 위해서 Tan이 개발한 코드 기반의 전력 소모량 측정 도구인 EMSIM 2.0[15] 시뮬레이터를 사용하였다.

EMSIM 2.0을 이용해 전력 소모량을 분석할 때는 실험을 수행하고자 하는 프로그램 코드를 작성하여, EMSIM 2.0의 전력 소모량 대상 프로그램 코드로 지정한다. 이때, 주의할 점은 측정을 원하는 명령어나 시스템 함수가 다른 프로그램 코드에 의해 영향을 받지 않도록 최소한의 기능성을 갖도록 구현해 주어야 한다는 것이다.

구현이 완료된 프로그램 코드는 리눅스 소스코드와 함께 ARM 교차 컴파일(cross compile)과정을 거쳐 RAM 디스크 이미지로 작성된 후, ROM 이미지를 생성하여 시뮬레이션하게 된다. 시뮬레이션이 완료되면, EMSIM 2.0은 전력 분석 수행 결과를 함수와 함수를 수행하기 위해 사용된 명령어 단위로 리포팅 한다. 시뮬레이션이 종료되면 출력 결과에서 측정하고자 했던 명령어나 시스템 함수에 대한 소모 전력 정보만을 추출한다. 이러한 과정은 매우 다양하게 구현된 소스 코드를 20회 이상 반복하고, 이들의 평균값을 취하게 된다.

(3) EMSIM 2.0을 통해 가성 명령어들에 대한 소모전력 측정결과는 <표 4> 및 <표 5>와 같다.

〈표 3〉 제안하는 가상 명령어 목록

구 분	명령어	
VPI	load, store, add, sub, call divide, call modulo, mult, compare, convertType, branch, bitNOT, bitAND, bitOR, bitXOR, bitShiftLeft, bitShiftRight	
VSF	Process Manager	fork(), waitpid(), wait(), signal()
	IPC	msgsnd(), msgrcv(), msgget(), msgctl(), semget(), semctl(), semop(), pipe(), pipe open(), pipe write()
	File System	File open(), File close(), File read(), File write()
	Memory Manager	shmget(), shmat(), shmdt(), shmctl()

〈표 4〉 VPI에 대한 전력 소모특성 측정결과

VPI	Type	Energy (nJ)	VPI	Type	Energy (nJ)
load	char	14.0362	call modulo	char	24.25
	short	28.0724		short	31.3863
	int	13.016		int	
	float	13.0157		float	N/A
	double	13.7019		double	
store	char	13.1765	mult	char	11.515
	short	26.3530		short	
	int	12.4897		int	
	float	12.8237		float	13.0358
	double	13.1638		double	13.0354
add	char	10.8692	compare	char	10.1905
	short			short	
	int			int	
	float	10.6376		float	13.0162
	double	10.978		double	
sub	char	11.2089	convertType	int -> float	11.3184
	short			int -> double	11.658
	int			double, float ->int	12.2281
	float	10.9779	branch	N/A	1.91309
	double	11.3176	bitNOT	only int	11.209
call divide	char	26.7458	bitAND	only int	10.5293
	short	29.6097	bitOR	only int	11.2089
	int		bitXOR	only int	10.8691
	float	25.4667	bitShiftLeft	only int	11.8885
	double	25.4669	bitShiftRight	only int	12.2282

〈표 5〉 VSF에 대한 전력 소모특성 측정결과

VSF	Parameters	Macro-models(nJ)	VSF	Parameters	Macro-models(nJ)
fork()	-	E = 132202.6	msgget()	-	E = 4351.019417
waitpid()	-	E = 25077.3	msgctl()	IPC_SET	E = 3334.7
wait()	-	E = 24986.6		IPC_STAT	E = 3835.7
signal()	-	E = 9460.6		IPC_RMID	E = 3603.6
File open()	-	E = 17886.6	semget()	-	E = 4557.2
File close()	-	E = 8262.2	semctl()	GETPID	E = 2573.4
File read()	c bytes	E = 5.1c + 49308.5		GETALL	E = 2938.0
File write()	c bytes	E = 5.6c + 32022.3		GETZCNT	E = 2795.5
shmget()	-	E = 126358.83		IPC_SET	E = 2784.4
shmat()	-	E = 11599.9		IPC_STAT	E = 4021.4
shmdt()	-	E = 14603.2		IPC_RMID	E = 3924.2
shmctl()	IPC_SET	E = 3211.5		SET_ALL	E = 2976.4
	IPC_STAT	E = 4010.1		SETVAL	E = 2660.8
	IPC_RMID	E = 17760.3	semop()	lock	E = 7322.8
msgsnd()	c bytes	E = 4.0c + 4554.0		unlock	E = 3688.1
msgrcv()	c bytes	E = 4.4c + 4818.0	pipe()	-	E = 31947.2
			pipe read()	c bytes	E = 4.7c + 18439.8
			pipe write()	c bytes	E = 2.8c + 14720.0

4.2.3 에너지 모델링 단계

에너지 모델링은 EEC의 상세 행위분석을 통해 해당하는 VPI와 VSF들을 매핑하는 과정이다. 매핑 과정은 다음과 같은 절차에 따라 진행된다.

(1) EEC와 VI의 패핑을 위하여 이들간의 상관성을 마크(mark)하는 매트릭스 테이블을 정의한다. EEC와 VI간의 상관성은 하나의 EEC와 하나의 VI 명령어로 매핑되는 일대일 관계와 하나의 EEC가 다수의 VI로 대응하는 일대다 관계가 존재한다. <표 6>은 EEC와 VI의 매핑 매트릭스를 보여준다. 매핑 매트릭스의 각 셀(cell)에 마크하기 위해서는 EEC에 대한 행위 상세화 과정이 요구된다.

(2) EEC와 VI의 일대다 대응 관계를 찾아내기 위하여

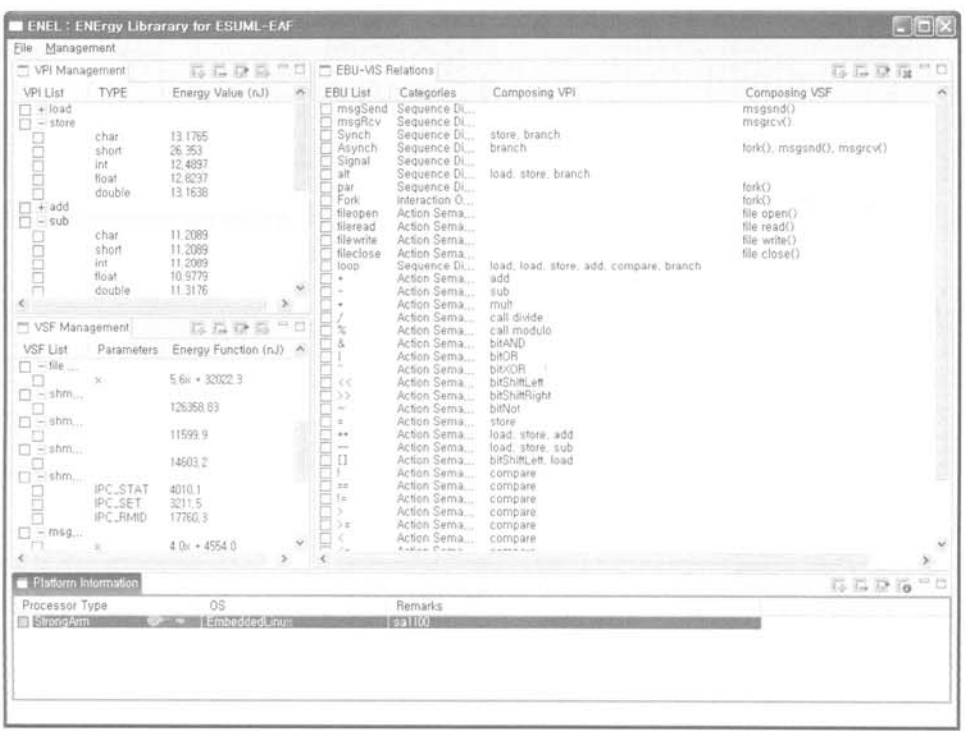
〈표 6〉 EEC와 VI 매핑 매트릭스

VI \ EEC	msg Send	msg Rcv	Sync	Asyn	Signal	Alt	par	Fork	...	>>	read
load						✓					
store			✓								
add											
⋮											
compare						✓					
brench			✓	✓		✓					
bitAND											
bitShiftRight										✓	
fork( )				✓			✓	✓	...		
wait( )											
signal( )					✓						
msgsnd( )	✓			✓							
msgrcv( )		✓		✓							
⋮											
file read( )											✓
shmctl( )											

EEC에 대한 상세화 과정이 필요하다. EEC를 상세화하기 위해서는 먼저 EEC에 해당되는 모델 요소를 프로그램으로 구현한 후에 컴파일 과정을 거쳐 목적 코드를 생성하다. 생성된 목적 코드의 분석을 통하여 VPI 또는 VSF로의 매핑을 수행한다. 이와 같은 과정을 거쳐 생성된 EEC의 세부 행위에 분석 내용은 다음과 같으며, 이들은 <표 6>의 매트릭스(음영 부분)에 반영된다.

- SynchCall : 반환위치를 저장하는 행위 + 분기하는 행위
- AsynchCall : 메시지를 전송하는 행위 + 프로세스를 생성하는 행위 + 분기하는 행위 + 메시지를 수신하는 행위
- alt : 조건변수를 로드 하는 행위 + 비교하는 행위 + 분기하는 행위
- loop : [조건변수 로드행위 + 비교행위 + 분기행위 + 인덱스변수 로드 + 인덱스변수 증가 + 인덱스변수 저장 + 내부 실행 블록(Action language에 의한 표현)] * 반복 횟수

이와 같은 과정을 거쳐 구축된 에너지 라이브러리의 화면은 (그림 6)과 같다.



(그림 6) 에너지라이브러리의 구현 화면

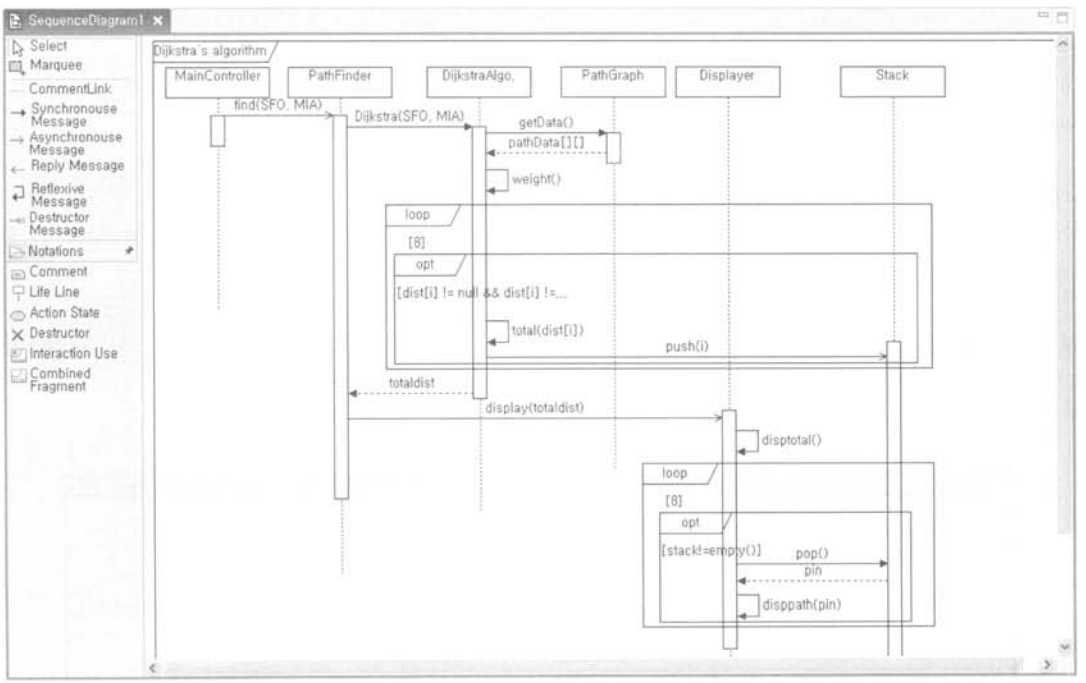
5. 에너지 라이브러리 적용

구축된 라이브러리, ENEL을 적용하여 간단한 UML 모델 기반의 소모 전력 분석에 대한 실험을 수행하였다. 실험은 먼저 UML 다이어그램으로 작성된 임베디드 소프트웨어의 설계모델로부터 EEC를 추출하고, 이를 이용하여 에너지 라이브러리를 통한 소모 전력을 산출하였다. 예제 시스템은 T.K. Tan의 연구[15]에서 제시한 소스 코드기반의 소모전력 분석 도구인 EMSIM 20.0을 이용한 결과와 비교 분석하였다. 사용한 예제 시스템은 (그림 7)과 같이 자동차용 네비게이션 소프트웨어에서 많이 사용되고 있는 Dijkstra의 최단 경로 선정 알고리즘이다.

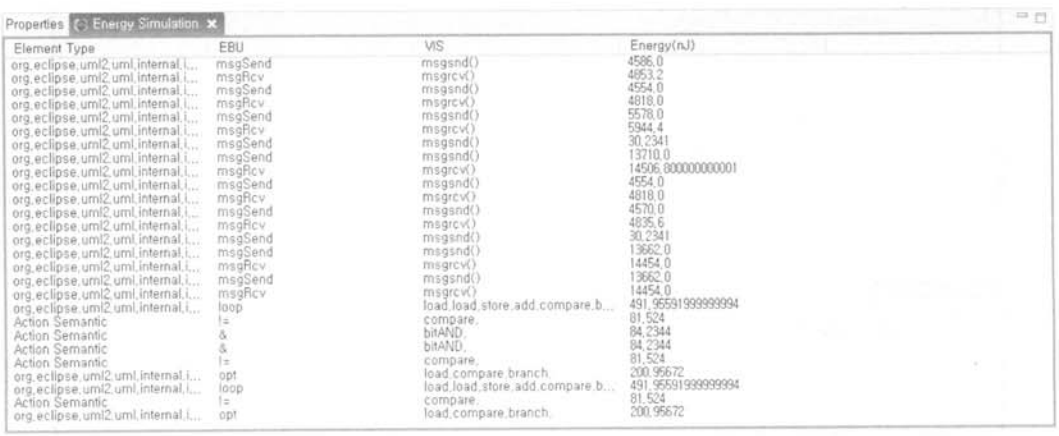
(그림 7)의 Sequence diagram을 이용하여 EEC[(그림 8)의 EBU 필드]를 선정한 결과는 (그림 8)과 같다. (그림 8)의 오른쪽 칼럼은 에너지 라이브러리로부터 검색한 각 EEC별 에너지 소모량을 보여주고 있다.

실험 결과로부터, UML 다이어그램으로부터 산출된 Dijkstra 알고리즘의 소모전력량은 135419.3343(nJ)의 값을 얻었으며, Tan의 연구로부터 산출한 코드 기반의 소모전력량은 136982.1117(nJ)의 값을 얻었다. 또한 임베디드 응용에서 많이 사용되는 탐색과 정렬 알고리즘에 대한 모델 기반의 소모전력 분석으로부터 <표 7>과 같은 결과를 얻었다.

<표 7>로부터 Tan의 소스코드 기반 분석과 본 연구에서 제시하는 UML 모델 기반 연구의 실험결과 적게는 1.15%, 크게는 11.8% 정도의 오차율을 보였다. <표 7>에 지시된 버블 정렬(Bubble_Sort) 알고리즘에서의 노드의 수(입력 데이터의 수)에 따른 소모 전력량의 비교 그래프는 (그림 9)와 같다.



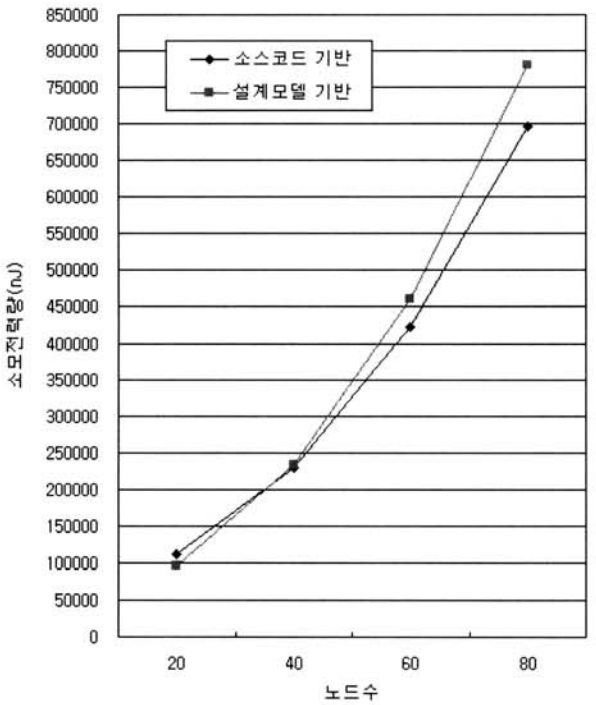
(그림 7) Dijkstra 최단경로선정 알고리즘의 Sequence Diagram



(그림 8) 최단경로선정 알고리즘의 모델 기반 시뮬레이션 결과

〈표 7〉 코드기반과 모델기반의 소모전력 분석 결과

알고리즘	Tan의 코드기반 분석(nJ)	UML 모델기반 분석(nJ)	오차율(%)
Shortest_Path	136982.1117	135419.3343	1.15
Bsearch	76822.7912	69370.9544	9.7
Bubble_Sort	113203.6165	99836.8189	11.8



(그림 9) 버블정렬 알고리즘의 소모 전력량 비교

노드 수의 변화에 따른 버블 정렬 알고리즘의 소모전력량 분석에서 20개 노드의 경우 모델 기반의 분석이 소스코드 기반의 분석보다 다소 적은 소모 전력을 갖는 것으로 예측되었으나, 40개 이상의 경우부터는 모델 기반 분석이 점진적으로 더 많은 전력을 소모하는 것으로 산출되었다. 이는 UML 다이어그램이 갖는 모델 요소의 추상화 수준에서 비롯된 것으로 보인다. (그림 9)에 나타난 두 개의 그래프간의 평균 오차율은 9.89%로써 입력 데이터 값의 분포에 따라 소모 전력량의 차이가 발생함을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 임베디드 소프트웨어 개발에 있어서 필수적으로 고려되고 있는 저전력 소모에 대한 요구사항을 만족시키기 위하여 소프트웨어의 설계 모델 기반의 소모 전력 예측 기법에 대하여 설명하였다. 특히 모델 기반의 소모 전력 예측을 위해 요구되는 에너지 컴포넌트 라이브러리의 구축에 대하여 구체적으로 설명하였다.

기존의 몇몇 연구에서 모델 기반의 소모 전력 분석에 대한 연구가 수행되었고, 또한 라이브러리 기반의 분석이 이루어졌으나, 이들은 소프트웨어 개발을 위해 별도의 추가적인 모델을 개발하여 소모 전력을 분석한다는 점과 라이브러리에 구축된 소모전력 정보를 대상 어플리케이션에 따라 다수 수집해야 하는 문제점이 있다. 그러나 본 연구에서는 소프트웨어 개발 과정에서 작성되는 UML 다이어그램을 그대로 사용하여 소모 전력을 분석하였으며, 가상 명령어 개념을 통한 라이브러리 구성의 유연성을 높였다.

향후의 연구는 임베디드 미들웨어, 메모리 데이터베이스 등에 대한 명령어 분석을 통해 에너지 라이브러리를 보강하고, 다양한 응용 역역에 적용할 수 있도록 확장하는 것이다.

참 고 문 헌

[1] R. G. Carvajal, et. al., "The flipped voltage follower: a useful cell for low-voltage low-power circuit design," IEEE TOCS, Vol.52, Issue 7, pp.1276-1291, 2005.

[2] V. Tiwari, S.Malik, and A.Wolfe, "Power analysis of embedded software: A first step towards software power minimization," IEEE Transactions on VLSI systems, Vol.2, No.4, pp.437-445, Dec., 1994.

[3] T. K. Tan, et al, "Software Architectural Transformations: A New Approach to Low Energy Embedded Software," Proc. of DATE, 2003, pp.1046-1051, Dec., 2003.

[4] Xiong Yue, Zhou Xuehai, et al, "OOEM: Object-Oriented Energy Model for Embedded Software Reuse," IEEE International Conference on Information Reuse and Integration, pp.551-558, 2003.

[5] G. Qu, et al, "Code Coverage-based power estimation techniques for microprocessors," Journal of Circuits, Systems, and Computers, Vol.11, No.5, pp.557-574, 2002.

[6] A. Muttreja, et al, "Hybrid Simulation for Energy Estimation of Embedded Software," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.26, No.10, pp.1843-1854, 2007.

[7] T. K. Tan, A. Raghunathan, et al, "High-Level Energy Macromodeling of Embedded Software," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.21, No.9, pp.1037-1050, Sep., 2002.

[8] T. K. Tan, A. Rahunathan, et al, "Energy Macromodeling of Embedded Operating Systems," ACM Transactions on Embedded Computing Systems, Vol.4, Issue 1, pp.231-254, 2005.

[9] H. Jun, et. al., "Modelling and Analysis of Power Consumption for Component-Based Embedded Software," EUC Workshop 2006, pp.795-804, 2006.

[10] V. Garousi, L. Briand, and Y. Labiche, 'Control flow analysis of UML 2.0 Sequence Diagrams, Carleton University,' Technical Report SCE-05-09, 2005

[11] D. Sarta, D. Trifone, and G. Ascia, "A Data Dependent

Approach to Instruction Level Power Estimation," IEEE Alessandro Volta Memorial Workshop on Low Power Design, pp.182-190, 1999.

[12] Vivek Tiwari, et al, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," IEEE Transactions on VLSI Systems, Vol.2, No.4, pp.437-445, Dec., 1994.

[13] A. Sinha and A. P. Chandrakasan, "JouleTrack - A Web based Tool for Software Energy Profiling," Proc. 38th IEEE Conference on Design Automation (DAC'01), pp.220-225, June, 2001.

[14] OMG, 'Unified Modeling Language: Superstructure,' version 2.1.2 (formal /2007-11-02) edition, 2007. available from: http://www.omg.org

[15] T. K. Tan, A. Raghunathan, et al, "EMSIM: An Energy Simulation Framework for an Embedded Operating System," Proc. of International Symposium of Circuits and Systems, pp.464-467, 2002.

[16] J. R. Bammi, Edwin Harcourt, et al, "Software Performance Estimation Strategies in a System-Level Design Tool," Proc. Of the CODES'00, pp.82-86, 2000.

김 두 환

e-mail : dhkim@selab.cbnu.ac.kr

2007년 충북대학교 컴퓨터공학과(학사)

2007년~현 재 충북대학교 컴퓨터과학과 석사과정

관심분야 : 임베디드 소프트웨어 모델링, 모델기반 전력분석, 소프트웨어 품질공학

홍 장 의

e-mail : jehong@chungbuk.ac.kr

1988년 충북대학교 전산학과(이학사)

1990년 중앙대학교 컴퓨터공학과(공학석사)

2001년 한국과학기술원 전산학(공학박사)

2002년 국방과학연구소 선임연구원

2002년~2004년 (주)솔루션링크 기술연구소장

2004년~현 재 충북대학교 컴퓨터공학 부교수

관심분야 : 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어 품질공학, 소프트웨어 프로세스