MPSoC용 임베디드 소프트웨어의 PSM 모델링 및 시뮬레이션

(Modeling and Simulation of Platform Specific Model in MPSoC Environment)

송 인 권 † 오 기 영 †† 홍 장 의 ††† 배 두 환 ††††
(In-Gwon Song) (Gi-Young Oh) (Jang-Eui Hong) (Doo-Hwan Bae)

요 약 임베디드 소프트웨어는 탑재될 하드웨어 아키텍처에 매우 의존적이기 때문에 플랫폼 특성을 고려한 소프트웨어 설계가 이루어져야 한다. 본 연구에서는 MPSoC(Multi Processor System On Chip)용 플랫폼에 탑재될 임베디드 소프트웨어의 PIM(Platform Independent Model)을 PSM(Platform Specific Model)에 매핑하기 위한 기법을 제안하고, 매핑 결과에 대한 시뮬레이션을 통해 매핑 기법의 유효성을 검사하였다. 제안하는 방법은 UML(Unified Modeling Language) 기반의 객체지향 모델로부터 태스크를 도출하여 이 기종의 하드웨어 컴포넌트로 구성된 MPSoC 플랫폼에 할당하기 위한 것으로써, 할당의 정확성 및 신속성과 소프트웨어 병렬성을 극대화 할 수 있는 장점을 제공한다.

키워드 : 임베디드 소프트웨어, MPSoC, PIM, PSM, 태스크 할당 알고리즘

Abstract Since embedded software is very dependent for target hardware architecture, characteristics of the platform must be considered when designing the software. Furthermore, MPSoCs consists of heterogeneous hardware components that are specified in micro level. Thus mapping of embedded software for MPSoCs should be considered the characteristics. In this paper, we provide an approach to automatic mapping PIM (Platform Independent Model) of an embedded software to PSM(Platform Specific Model) for MPSoC(Multi Processor System On Chip) and verify its effectiveness with simulation. In the proposed approach, tasks are derived from an object oriented model based on the UML (Unified Modeling Language). And then the types of the derived tasks are identified. With the identified types and inter relationship between tasks, the tasks are assigned to appropriate heterogeneous hardware components. We expect that the approach improve accuracy of the assigning and concurrency of the deployed software.

Key words : Embedded software, MPSoC, PIM, PSM, Task assigning algorithm

1. 서 론

임베디드 시스템 개발에 있어서는 소프트웨어가 탑재될 하드웨어 플랫폼을 고려하는 것이 매우 중요하다. 특히 다수의 프로세서가 한 칩 안에 집적되는 MPSoC 환경에서는 소프트웨어가 갖는 모듈의 독립성이나 병렬성을 고려하여 대상 시스템을 설계, 개발 하는 것이 무엇보다 중요하다[1]. MPSoC 기반의 하드웨어 플랫폼은 프로세서 코어나 DSP(Digital Signal Processor) 등과 같은 다양한 타입의 프로세서를 포함하고, 지역 메모리, 공유 메모리 등과 같은 장치들을 갖기 때문에MPSoC용 임베디드 소프트웨어 개발 시 소프트웨어 요구사항을 중심으로 이루어지는 PIM 모델링과 하드웨어 아키텍처를 고려하는 PSM 모델링이 구분되어 설계되어야 한다[2,3]

본 연구는 객체지향 개념과 UML 언어[4]를 이용하여 임베디드 소프트웨어를 개발하고자 하는 경우, 어떻게 PSM 모델링을 수행할 것인가에 대하여 제안한다. 특히 MPSoC용 임베디드 소프트웨어의 개발에 있어서 PIM을 하드웨어 아키텍처에 어떻게 할당 할 것인가에 대한

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성, 지원사업의 연구결과로 수행되었음
† 학생회원 : 한국과학기술원 전자전산학과
igsong@se.kaist.ac.kr
†† 학생회원 : 충북대학교 전기전자컴퓨터공학부
ohgy@selab.chungbuk.ac.kr
††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
jehong@selab.chungbuk.ac.kr
†††† 종신회원 : 한국과학기술원 전자전산학과 교수
: bae@se.kaist.ac.kr
논문접수 2007년 4월 25일
심사완료 2007년 6월 28일

방법을 제시하는 데 연구의 주안점이 있다. 또한 하드웨어 아키텍처에 할당된 태스크의 수행에 대한 시뮬레이션을 통해 PSM의 적합성에 대하여 검사하였다.

일반적으로 트랜잭션 중심의 소프트웨어 개발에 있어서 PSM은 EJB(Enterprise JavaBeans) 아키텍처나 CORBA와 같이 소프트웨어가 탑재될 구현 환경을 고려한 모델이며, PIM으로부터 PSM으로의 변환 과정을 거쳐 상세 설계 과정이 진행된다[5]. 이와 마찬가지로 임베디드 소프트웨어 개발도 유사한 변환 과정이 요구되는데, 특히 MPSoC용 환경에서는 프로세서간의 병렬성 및 자원 제약 사항을 고려한 보다 정밀하고 세밀한 마이크로 레벨에서의 매핑 과정이 요구된다. 이와 같은 매핑과정의 필요성 및 특성에 대하여 정리하면 다음과 같다.

- MPSoC용 하드웨어 플랫폼은 프로세서 이외에도 DSP, 디바이스 컨트롤러와 같은 다양한 종류의 하드웨어 요소(IP(Intellectual Property))들로 구성된다[6].
- 식별된 태스크를 하드웨어 구성요소에 할당 할 때, 태스크의 특성뿐만 아니라 하드웨어 IP의 특성도 고려한 매핑이 필요하다.
- 임베디드 시스템의 경우, 때때로 운영체제를 포함하지 않을 수도 있다. 이러한 경우에는 태스크 수행에 대한 제어가 보다 정밀하게 이루어지기 때문에 태스크 할당의 중요성이 부각된다.
- 식별된 태스크의 수가 많은 경우, 모델러가 일일이 수작업으로 태스크를 하드웨어 요소로 할당하는 일은 번거롭고, Error-Prone한 작업이기 때문에 자동화된 방법이 필요하다.

위와 같은 필요성에 따라 본 연구에서는 PIM을 MPSoC용 하드웨어 아키텍처인 PSM에 할당하기 위한 알고리즘을 제시한다. 제시한 알고리즘은 대규모의 임베디드 시스템 개발 시, PIM과 PSM 간의 매핑을 용이하게 지원할 뿐 만 아니라, PIM이 갖는 병렬성이 PSM에서 충분히 지원하는 가를 검사하기 위한 목적으로 활용될 수 있다.

본 논문의 2장에서는 기존의 관련 연구들에 대한분석을 통해 제시한 연구의 차별점을 설명하고, 3장에서는 모델 매핑을 위한 시스템을 정의하며, 4장에서는 제안하는 할당 알고리즘을 설명할 것이다. 5장에서 시뮬레이션을 이용한 실험을 통해, 본 논문에서 제시하는 알고리즘의 유효성을 보일 것이다.

2. 관련 연구 분석

기존의 관련 연구들에 대한 분석은 크게 두 가지 측면에서 이루어질 수 있는데, (1) 객체지향 개념을 중심으로 하는 MDA 기반 모델링에서 PIM을 PSM으로 변환하는 과정에 대한 기존 연구들과, (2) 멀티 프로세서 환경에서 태스크를 할당하기 위한 알고리즘에 대한 연구들에 대한 것이다.

MDA 기반 모델 변환 과정은 일반적으로 요구사항을 기반으로 하는 추상화 수준에서 모델러의 휴리스틱(Heuristics)에 의존적인 경우가 많다. MDA 기반의 PIM에서 PSM으로 매핑하기 위한 연구는 Master[7]에 의해 제시되었는데, 이 연구에서는 PIM과 PSM의 구성요소들을 컴포넌트화하여 저장소를 유지하고, 이들 컴포넌트의 매핑 과정에서 매핑 규칙을 OCL(Object Constraint Language) 언어로 정의하도록 하였다. 그러나 이 연구에서는 MPSoC용 임베디드 플랫폼 환경에서의 연구이기보다는 응용 어플리케이션 개발에서의 모델 매핑을 위한 자동화 지원에 대한 연구이다.

또한 기존의 멀티 프로세서에 대한 태스크 할당 알고리즘들은 대부분 동일한 프로세서 타입을 갖는 아키텍처에 태스크를 할당하기 위한 것들로써, 태스크의 병렬 수행에 대한 응답 시간의 만족이나, 통신의 오버헤드를 줄이기 위한 목적으로 연구되었다.

Plishker[8]의 연구에서는 멀티 쓰레드를 갖는 멀티 프로세서 임베디드 시스템의 IXP1200 네트워크 프로세서에 태스크를 자동할당하기 위한 방법을 제시하였는데, 총 6개의 처리기에 태스크를 할당하는 방법을 제시하였으나 동일한 프로세서에 대하여 실행 시간만을 고려한 할당이라는 점과 실행 코드 기반의 할당이라는 점에서 본 연구와 차이점이 있다.

Benini[9]의 연구에서는 MPSoC 환경에 존재하는 동일 타입의 프로세서들로 태스크를 할당하기 위한 방법을 제시하였다. 태스크의 할당 과정에서 자원에 대한 제약사항을 고려하여 태스크 할당이 이루어지도록 하였으나, 이 기종의 프로세서 타입이 MPSoC에 존재할 수 있으며, 이에 따른 태스크 할당이 이루어져야 한다는 측면에서 고려되지 못하였다. 그 밖에도 Hong[10]의 연구는 동일한 크기의 상호 독립적인 태스크들을 분산된 컴퓨팅 환경으로 할당하기 위한 알고리즘을, Paulin[11]의 연구는 StepNP 멀티프로세서 SoC 플랫폼을 갖는 분산 객체 환경에서 메시지 패싱과 태스크 스케쥴링을 고려하는 병렬처리 프로그래밍 모델을 제시하였다. 그러나 이러한 연구들은 네트워크 프로토콜을 갖는 분산 환경에서의 태스크 할당 및 접근 방법들이라고 할 수 있다.

3. 매핑 시스템 정의

본 연구는 MPSoC플랫폼에 탑재될 임베디드 소프트웨어를 UML 기반 모델로 설계하기 위한 개발 프로세스 및 방법론[12,13]의 개발 과정에서 수행되었으며, 그림 1과 같은 절차를 갖는다.

본 연구는 그림 1에서의 PSM 모델링에 대한 연구로

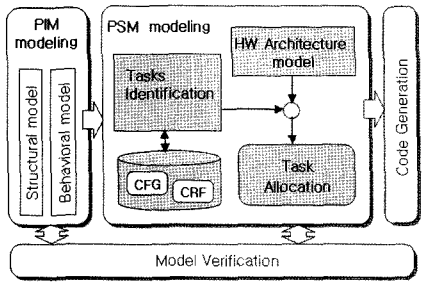


그림 1 본 연구에서 제안하는 PSM 모델링 절차

서, 특히 태스크 할당 즉 매핑에 초점이 맞추어져 있다.

본 연구에서 제안하고자 하는 PIM으로부터 PSM으로의 매핑 시스템을 정형화하여 기술하면 정의 1과 같다.

정의 1. 매핑 시스템 S_m = <S_i, D_t, A_{ux}, α, L_m>으로 구성된다. 여기서

- S_i : 매핑의 대상이 되는 소스 모델
- D_t : 매핑 시스템에 의해 생성되는 타겟 모델
- A_{ux} : 소스 모델로부터 타겟 모델로의 매핑 과정에서 사용하는 변환 함수들의 집합
- α : 매핑을 수행하는 액션의 집합
- L_m : 매핑 위한 할당 함수로서, 액션 α들의 조합으로 구성된다. □

매핑 시스템의 구성 요소들에 대한 상세한 정의를 3.1절, 3.2절, 그리고 3.3절에 각각 설명하였다. 참고로 구성요소들에 대한 설명을 위해 NIST의 FIPS-197 AES (Advanced Encryption Standard) 알고리즘을 사용하는 SFT(Secure File Transfer) 소프트웨어[14]를 예제 시스템으로 사용하였다.

3.1 소스 모델

매핑 시스템에서의 소스 모델은 사용자 요구사항으로부터 생성된 UML 기반의 PIM이다. PIM이 생성되면, 이로부터 하드웨어 아키텍처의 구성요소들로 매핑 할 태스크들이 도출되는데, 소스 모델 S_i에 대한 정의는 다음과 같다.

정의 2. 매핑 시스템에서의 소스 모델, S_i는 다음과 같은 요소들의 집합으로 정의한다.

S_i = {C, I, S}

- C : UML2.0으로 표현된 클래스 다이어그램
- I : UML2.0으로 표현된 Interaction Overview Diagram의 집합
- S : UML2.0으로 표현된 Sequence Diagram들의 집합 □

정의 2에 나타난 바와 같이 본 연구에서 제시하고 있는 PIM의 구성은 클래스 다이어그램, IOD(Interaction Overview Diagram), 그리고 SD(Sequence Diagram)로 구성되고 있다. 정의 2는 UML2.0에서 제시된 표준

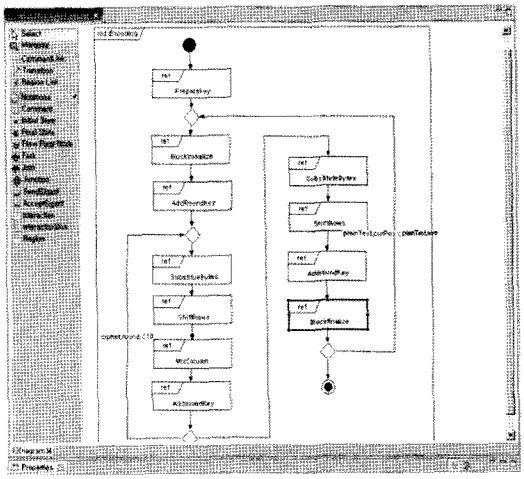


그림 2 예제시스템의 IOD

적인 구문과 의미를 모두 포함하고 있음을 가정한다.

그림 2는 주어진 예제 시스템, SFT에 대한 IOD의 일부 예제 모델이다. 안전한 파일 전송을 위해서는 파일에 대한 암호화 작업이 진행되어야 하며, 이에 대한 과정을 모델링한 것이다.

3.2 타겟 모델

타겟 모델은 소스 모델인 S_i로부터 최종적으로 얻고자 하는 결과 모델이다. 다시 말해서 PIM으로부터 태스크를 추출하고, 이를 MPSoC용 하드웨어 아키텍처에 매핑한 모델로써, 정의 3에서와 같이 정의한다. 이를 위해서는 먼저 MPSoC 하드웨어 아키텍처에 대한 모델링이 이루어져야 한다.

그림 3은 MPSoC용 HAD(Hardware architecture diagram)의 예를 보여준다. 이 다이어그램은 아키텍처에 대한 메타 모델에 의해 정의되며, <<CPU>>,

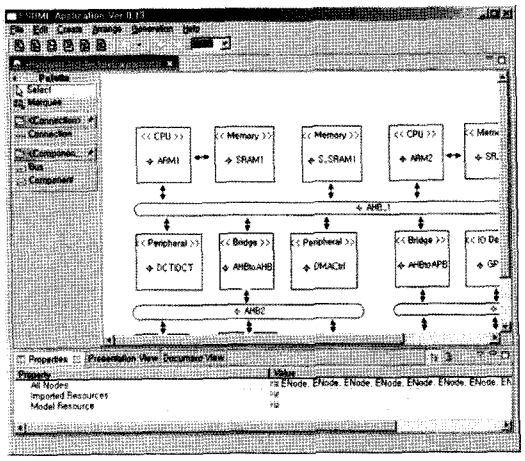


그림 3 HAD예제

<<DSP>>, <<BUS>>, <<Local Memory>>, <<Shared Memory>>, <<DevController>> 등과 같은 스테레오타입을 갖는 컴포넌트들로 구성된다.

그림 3에서 보여지는 요소 중에 BUS를 제외한 하드웨어 아키텍처의 구성요소들에 태스크를 매핑하여 타겟 모델을 생성하는데, 타겟 모델에 대한 정의는 다음과 같다.

정의 3. 타겟 모델, Dt는 다음과 같은 요소들의 집합으로 정의한다.

$D_t = \{h_i, P_{hi}, t_k\}$

- h_i : 하드웨어 아키텍처 구성요소들의 집합
- P_{hi} : 요소 h_i의 타입
- t_k : 요소 h_i에 할당된 태스크들의 집합 □

정의 3으로부터 h_i는 그림 3에서의 사각형으로 나타나는 하드웨어 구성요소로서, 태스크를 할당 가능한 요소만을 포함한다. P_{hi}는 <<CPU>>, <<DSP>> 등과 같은 하드웨어 요소들의 유형을 의미한다. MPSoC의 구조에서는 하나의 유형에 다수의 h_i가 존재할 수 있다. 타겟 모델에서의 t_k는 매핑 시스템의 L_m 요소에 의해 할당된 태스크들의 모임을 의미한다.

3.3 변환 함수

소스 모델로부터 타겟 모델로의 매핑하는 과정에서는 먼저 (1) UML로 표현된 PIM을 쓰레드의 개념을 갖는 CFG(Control Flow Graph)로 생성하는 함수를 거쳐 (2) CFG로부터 최종의 태스크를 식별하고, (3) 식별된 태스크들간의 상관인자(CRF, Correlation Factor)를 계산하게 된다. 이러한 세가지 함수를 거쳐 메타 정보를 생성한 후, 이를 이용하여 최종적으로 매핑함수 L_m에 의해 매핑 액션 α가 수행된다.

(1) CFG 생성함수

$$A_{ux} : f\ (PIM) = \{\ CFG_i \mid i = 1..n\ \}$$

CFG생성함수는 PIM으로부터 시스템 행위의 실행 패스를 찾아 CFG를 반환하는 함수로써, 초기에 PIM이 갖는 태스크의 수인 n 만큼 생성한다. CFG는 UML의 IOD와 SD가 갖는 각 요소를 노드로 하고, 노드 간의 실행 가능한 패스를 에지(edge)로 갖는 방향성 그래프로서[15], 이의 정의는 다음과 같다.

정의 4. Control Flow Graph $G = (V, E, N_i, N_f)$는 다음과 같이 정의된다.

- V : 노드(vertex)들의 집합(이벤트, Fork, Join, Decision, Merge 노드 등이 포함)
- $E \subseteq V \times V$: 노드간의 엣지들의 집합.
- N_i : 시작 노드들의 집합
- N_f : 최종 노드들의 집합 □

위의 정의 4로부터 CFG 생성을 위해 예를 살펴보자.

그림 4는 일반적인 Producer-Consumer의 동작을 SD로 모델링 한 것이다. 이때, 메시지를 주고받는 지점

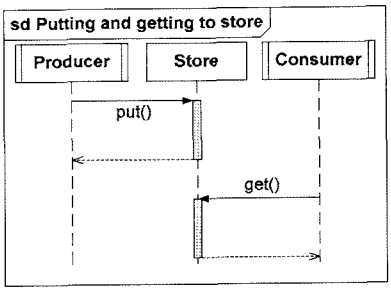


그림 4 Producer-Consumer의 예제 SD

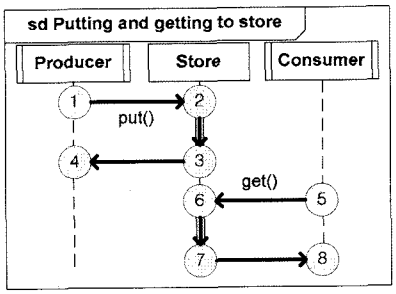


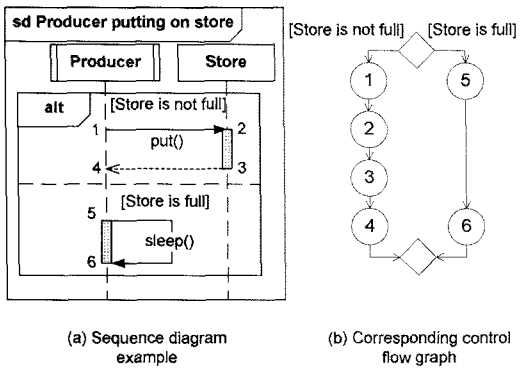
그림 5 그림 4에서 각 요소의 인과적 순서

과 Exec. Occurrence에 나타내는 실행의 시작지점과 끝 지점이 각각 CFG의 노드가 된다. 이 노드들 간의 순서는 그들간의 인과적 관계로 정의된다. 이 인과적 관계는 SD에서의 Time Line에 근거하여 해석된다.

그림 5는 그림 4의 각 모델 요소에 대한 순서를 화살표로 나타낸 것 이다. 이와 같은 방법으로 각각의 SD에서의 CFG가 생성된다.

IOD에서 CFG생성은 두 단계로 이루어지는데, SD간의 연결 흐름이 있는 경우에는 비동기적인 관점에서의 연결(asynchronous concatenation)[15]이 이루어지며, Combined Fragment 혹은 IOD의 제어 노드에 의해 SD가 연결되는 경우에는 Fork, Join, Decision 그리고 Merge노드 등의 제어노드를 사용하여 결합한다.

그림 6은 Combined Fragment에 "alt" 연산자가 부여되었을 경우 CFG생성을 나타내고 있다. 그림 6(a)의 SD는 Store가 꽉 차있을 경우 Producer가 Sleep하고 그렇지 않은 경우 Store에 물품을 넣는 경우를 나타내고 있다. 이에 대한 CFG를 생성하기 위해서는 먼저, SD의 Combined Fragment에 Operand로 주어진 부분에 대한 CFG를 각각 생성한다. 그림 6에서는 각각 1→2→3→4와 5→6으로 operand의 CFG가 나타나게 된다. 생성된 각각의 CFG는 "alt"연산자의 의미에 따라 "Decision"노드와 "Merge"노드로 묶여서 그림 6의 (b)와 같이 변환되게 된다.



(a) Sequence diagram example (b) Corresponding control flow graph

그림 6 Combined fragments로부터의 CFG 변환 예제

이와 같은 기법에 의해 CFG는 하나의 SD에 대하여 하나의 CFG를 생성하게 되며, IOD가 갖는 전체적인 모델의 행위를 따라 CFG가 합성되어 태스크로 도출된다.

(2) 태스크 식별함수

$$A_{ux} : f\ (CFG) = \{\ task\ \}$$

PIM인 클래스 다이어그램, IOD, 그리고 SD로부터 CFG가 생성되면, IOD에 나타나는 액티브 클래스의 행위를 중심으로 CFG가 하나의 태스크로 식별된다. 이들은 독립적인 후보 태스크로 정의하며, 이들로부터 다음의 두 가지 작업을 통해 최종의 태스크를 결정하게 된다.

- 식별된 태스크의 CFG로부터 병렬성이 존재하는 부분을 식별하여 그래프 분할을 수행하고, 분할된 그래프는 세부 태스크로 정의한다.
- 각 태스크들에 대하여 태스크 타입을 정의한다.

이와 같은 작업을 통해 얻어지는 최종의 태스크 집합은 다음과 같은 세가지 요소로 구성된다.

- t_i : 식별한 태스크들의 집합
- P_{ti} : 태스크 t_i의 타입
- C_{ti} : 태스크 t_i에 포함된 객체들의 집합

태스크 타입 P_{ti}는 식별된 각 태스크에 대한 타입을 의미하는 것으로서, 태스크 자체가 지닌 속성이 어느 타입에 속하는 것인지를 정의하는 것이다. 본 연구에서는 태스크 타입은 제어타입, 데이타 연산타입, 디바이스 드라이버 타입, 그리고 데이타 개체 타입 의 네 가지 타입으로 구분한다. 이중에서 앞의 세가지는 연산 타입의 태스크이며, 나머지 하나는 데이타 타입이다. 태스크에 포함되는 클래스의 집합 C_{ti}는 PIM으로부터 태스크를 식별하는 과정에서 클러스터링된 클래스들의 모임을 의미한다.

이와 같은 과정에 따라 주어진 예제 시스템, SFT에 대해 식별된 태스크 목록은 그림 7과 같다.

그림 7로 부터 AT*는 액티브 태스크를 의미하며, ET*는 데이타 엔티티를 의미한다. 또한 AT3.1로부터

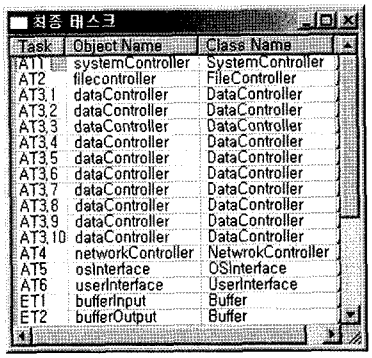


그림 7 SFT 소프트웨어의 태스크 목록

AT3.10은 AT3 태스크가 갖는 병렬성으로 인하여 분리된 태스크들이다. 이와 같이 하나의 태스크가 내부적인 병렬성을 갖는 경우, 이들은 서로 다른 태스크로 분리되어 식별 될 수 있다. 이렇게 함으로써, 프로세서간 병렬처리를 가능하게 한다.

(3) CRF 계산함수

$$A_{ux} : f\ (t_i, t_j) = \{\ CRF_{ij} \mid i = 1..n\ X\ j = 1..n\ \}$$

최종의 태스크 목록이 생성되면, 각 태스크들에 대하여 상관성을 측정하게 된다. 이러한 상관성의 측정은 두 개의 태스크 t_i와 t_j에 대한 결합력을 의미하며, 높은 값일수록 상관성이 높음을 의미한다. 결론적으로 말하면 상관성이 높다는 의미는 두 태스크가 하나의 프로세서로 할당될 수 있음을 의미한다.

태스크간의 상관성을 나타내는 CRF에 대한 산정은 다음 정의와 같다.

정의 5. Task Correlation Factor, CRF는 다음과 같이 정의된다.

$$CRF(t_1, t_2) = MRF(t_1, t_2) + W_f\ X\ FRF(t_1, t_2)$$

이로부터 MRF는 두 개의 태스크 t_1, t_2간에 존재하는 메시지에 의한 상관성을 나타내며, 다음과 같이 계산된다.

(a) 태스크 t_1과 t_2가 Active 태스크인 경우

$$\sum_{m \in M_{async}(G(t_1),G(t_2))} (W_j^{L(m)} P(m)(S_{mid} + \sum_{p \in Par(m)} size(p)))$$

(b) Active 태스크가 아닌 경우

$$\sum_{m \in M_{async}(G(t_1),G(t_2))} (W_j^{L(m)} P(m)(S_{mid} + size(Ret(m)) + \sum_{p \in Par(m)} size(p)))$$

여기서 나타나는 주요한 함수들을 정의하면 다음과 같다.

- $M_{sync}(G_1,G_2)$: CFG의 G_1에서 G_2로 가는 모든 동기 메시지의 집합
- $M_{async}(G_1,G_2)$: CFG의 G_1에서 G_2로 가는 모든 비동기 메시지의 집합
- W_1 : 메시지가 속해있는 루프가 반복하는 횟수를 나타내는 weight factor. 모든 루프가 determini-

stic하게 정의 될 수 없으므로 평균적으로 반복되는 횟수를 사용한다.

- L(m) : 메시지가 속해 있는 루프의 개수를 카운트하는 함수. 몇 개의 중첩된 루프 안에 메시지가 정의되어 있는 지를 파악한다.
- P(m) : CFG에서 메시지 m을 보내는 노드를 만날 확률. 루프를 계산하지 않고, If-then-else를 나타내는 Decision 노드와 Merge노드의 관계만을 이용하여 계산한다.
- S_{mid} : 메시지 식별자(mid)의 크기
- Par(m) : 메시지 m에 포함되는 파라메터의 수
- Size(p) : 타입 P의 메모리 사이즈

그리고, FRF는 Fork/Join에 의한 태스크간의 상관성이다. 이때 W_f는 FRF에 대한 가중치로써, Fork 및 Join 시에 발생하는 메시지의 크기이다. FRF의 값은 $t_1(t_2)$가 $t_2(t_1)$을 Fork하면 1의 값을 아니면, 0의 값을 갖는다. □

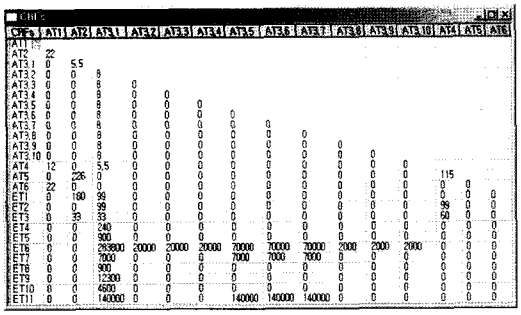


그림 8 예제 시스템의 태스크들에 대한 CRF 값

위와 같은 CRF 값에 대한 정의로부터 예제 시스템에 대하여 생성된 CRF 테이블은 그림 8과 같다. ET5의 데이타 개체 타입의 태스크는 단지 AT3.1 data-Controller 태스크와 상관성이 존재하기 때문에 AT3.1 태스크가 할당되는 프로세서의 지역 메모리로 할당될 것이며, AT5 osInterface 태스크는 AT2와 226의 상관성, 그리고 AT4와 115의 상관성을 갖기 때문에 만약 AT2와 AT4가 서로 다른 프로세서에 할당되었다면 AT2가 할당된 프로세서로 AT5가 할당될 것이다.

4. 할당 알고리즘

할당 알고리즘은 정의 1에 나타난 매핑시스템의 할당 함수 L_m을 정의한 것이다. 할당 함수는 앞서 간략히 설명한 것처럼 태스크의 타입, 태스크간의 상관성을 나타내는 CRF 값, 그리고 하드웨어 구성요소의 타입을 입력으로 받아 태스크를 하드웨어 구성요소에 할당하는 과정을 처리한다.

4.1 알고리즘 개요

PIM으로부터의 태스크를 MPSoC하드웨어 구성요소에 할당하기 위한 개념은 그림 9와 같다.

그림 9에서 보는 바와 같이 태스크들의 집합으로부터 할당 대상이 되는 태스크 t_i가 선택되고, 이의 태스크 타입에 대응되는 하드웨어 요소 h_i가 선택된다. 이때 MPSoC의 아키텍처상에는 동일한 유형을 갖는 h_i가 다수 존재할 수 있기 때문에 - 예를 들면, 동일한 ARM-925ejs 코어 프로세서가 다수 존재할 수 있기 때문에 어떠한 특정 프로세서에 할당 할 것인가를 결정해야 한다. 이는 제어변수로 입력되는 CRF 값에 의해 결정 될 수 있다.

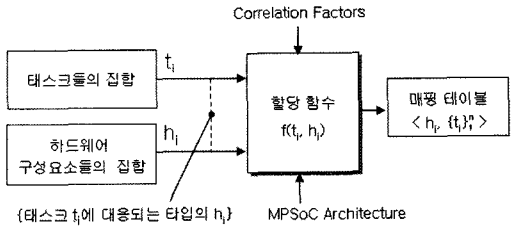


그림 9 태스크 할당 함수의 처리과정

4.2 하드웨어 아키텍처 정의 변수

그림 9에서 정의하는 할당 알고리즘을 구현하기 위해서 하드웨어 아키텍처에 대한 구조를 정의하는 변수를 그림 8과 같은 배열 구조로 정의하였다.

위의 그림이 갖는 자료 구조의 의미는 먼저 배열의 [0]번째 컬럼은 하드웨어 구성요소의 타입 별로 보드에 존재하는 요소들의 수를 의미한다. ARM 코어 같은 범용의 프로세서(GPP)는 MPSoC 보드상에 4개 존재하고, DSP 프로세서는 2개 존재함을 의미한다.

그림 10의 [1]번 컬럼은 각 타입의 첫 번째 구성요소에 대한 정보이다. Null 값을 갖는다면 더 이상의 구성요소가 존재하지 않는다는 의미이며, null 값이 아니고 1이면 해당 프로세서에 지역 메모리(LM)가 존재함을 의미하고, 0이면 지역 메모리가 없음을 의미한다. 따라서 DSP 프로세서는 2개 존재하는데, 첫 번째 DSP는 지역 메모리를 가지고 있으나 두 번째 DSP에는 지역

	[0]	[1]	[2]	[3]	[4]	...	[n]
GPP	4	1	1	0	1	null	
DSP	2	1	0	null			
SM	1	0	null				
DC	3	0	0	0	null		

그림 10 하드웨어 구조를 표현하는 자료구조

메모리가 없음을 의미한다. DC(Device Controller)의 경우는 특정 디바이스를 제어하기 위한 특수 목적용 처리장치가 3개 존재함을 의미한다.

4.3 할당 의사결정 테이블

그림 10의 하드웨어 구조를 표현하는 정보로부터 MPSoC가 갖는 구조를 다음과 같은 8가지의 경우로 구분하여 살펴볼 수 있다.

(1) GPP가 1개(#GPP = 1)만 존재하는 경우
(2) #GPP > 1만 존재하는 경우
(3) #DSP = 1가 존재하는 경우
(4) #DSP > 1가 존재하는 경우
(5) GPP에 LM이 존재하는 경우
(6) DSP에 LM이 존재하는 경우
(7) SM이 존재하는 경우
(8) Device Processor가 존재하는 경우

위의 8가지 하드웨어 구조를 이용하여 생성 할 수 있는 MPSoC 아키텍처와 이들에 대한 할당 내용을 정리하면 표 1과 같다.

표 1 태스크 할당을 위한 조건-액션 테이블

	Condition (AND)								Action(α)
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	
	T	.	.	.	T	.	.	T	$\alpha_4, \alpha_5, \alpha_8$
O	T	.	T	.	T	T	.	T	$\alpha_0, \alpha_2, \alpha_5, \alpha_8$
R	T	.	.	T	T	T	.	T	$\alpha_0, \alpha_3, \alpha_6, \alpha_8$
	.	T	T	.	T	T	T	T	$\alpha_1, \alpha_2, \alpha_6, \alpha_7, \alpha_8$
	.	T	.	T	T	T	T	T	$\alpha_1, \alpha_3, \alpha_6, \alpha_7, \alpha_8$
	.	T	T	.	T	T	.	.	$\alpha_1, \alpha_3, \alpha_6$
	.	T	.	T	.	.	T	.	$\alpha_1, \alpha_3, \alpha_7$

표 1은 MPSoC 하드웨어 갖는 아키텍처의 구성 요소들의 조합에 대하여 서로 다른 타입의 태스크를 매핑하는 액션들과의 관계를 표현한 것이다. 표 1에 나타나는 액션 α는 AND 조건 즉, 해당 줄(row)의 조건이 T에 해당되는 하드웨어 요소들이 존재하는 경우에 태스크를 할당하는 액션들을 정의한 것이다.

태스크 할당을 위한 액션은 α_0부터 α_8까지 정의하였는데, 이들에 대한 처리는 4.4절의 알고리즘 부분에 기술하였다. 여기서 한가지 고려할 사항은 하드웨어 구성요소에 대한 모든 조합을 고려하는 경우 표 1에서 보는 것보다 더 많은 조합이 생겨날 수 있으나, MPSoC 하드웨어 아키텍처를 설계하는 과정에서 극단적으로 고려되지 않는 조합에 대해서는 생략하였다.

4.4 알고리즘

표 1의 조건-액션(Condition-Action) 테이블에서 제시하고 있는 액션 부분에 대한 구현 알고리즘은 그림 11과 같다. 그림 11에서 보는 바와 같이 α_0 액션은 태스크 타입 Pti가 제어타입(CL)인 모든 태스크들을 존재하는 하나의 GPP타입 프로세서에 모두 할당하는 동작이며, α_1 액션은 모든 제어타입의 태스크들을 다수의 GPP 타입 프로세서에 라운드 로빈 방식으로 할당한다. 라운드 로빈 방식의 할당은 단순히 돌아가면서 할당하는 것을 의미하는 것이 아니라, CRF값이 높은 태스크가 할당 된 프로세서에 태스크를 할당하는 방식이다. 이렇게 했을 경우 모든 태스크가 하나의 프로세서에 할당될 수 있으므로 일정 쓰래쉬홀드를 줄 수 있도록 하였다. 본 연구에서 사용하는 예제 SFT의 경우는 그러한 현상이 일어나지 않으므로 쓰래쉬홀드를 따로 설정하지 않았다.

Algorithm L_m ;
{
The set of tasks = { CL, DI, DE, DD} which means that CL is Control type, DI is Data-Intensive type, DE is Data Entity type, and DD is Device Driver type.

Mapping Functions , <<all>>, <<rr>, and <<x>> means that
<<all>> : allocate all tasks of the given type into the target hardware components.
<<rr>> : allocate the tasks of the given type into the target hardware components by round-robin approach with considering the CRF value.
<<x>> : some tasks allocated to the corresponding hardware component, respectively

α_0 : $\forall(P_{ti} = CL) \xrightarrow{<<all>>} \exists(P_{hi} = GPP)$
α_1 : $\forall(P_{ti} = CL) \xrightarrow{<<rr>>} \forall(P_{hi} = GPP)$
α_2 : $\forall(P_{ti} = DI) \xrightarrow{<<all>>} \exists(P_{hi} = DSP)$
α_3 : $\forall(P_{ti} = DI) \xrightarrow{<<rr>>} \forall(P_{hi} = DSP)$
α_4 : $\forall(Pti = CL \vee Pti = DI) \xrightarrow{<<all>>} \exists(P_{hi} = GPP)$
α_5 : $\forall(P_{ti} = DE) \xrightarrow{<<all>>} \exists(P_{hi} = LM_{GPP} \mid Phi = LM_{DSP})$
α_6 : $\forall(P_{ti} = DE) \xrightarrow{<<rr>>} \forall(P_{hi} = LM_{GPP} \mid Phi = LM_{DSP})$
α_7 : $\forall(P_{ti} = DE) \xrightarrow{<<all>>} \exists(P_{hi} = SM)$
α_8 : $\forall(P_{ti} = DD) \xrightarrow{<<x>>} \forall(P_{hi} = DC)$
}

그림 11 할당 액션에 대한 처리 알고리즘

그림 11에서 제시한 알고리즘에 대하여 주어진 예제 시스템 Secure File Transfer 소프트웨어에 대한 태스크들의 하드웨어 구성요소로 할당을 수행하면 그림 12와 같은 매핑 결과의 XMI 파일 형태를 얻을 수 있다.

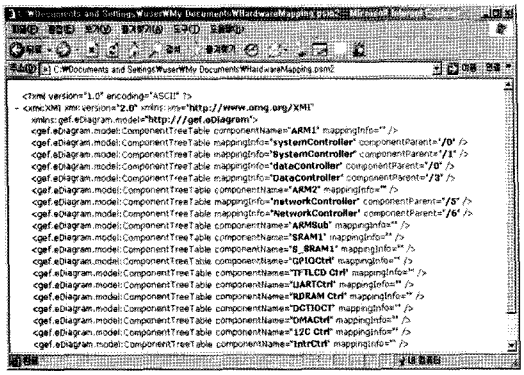


그림 12 태스크 할당 결과에 대한 XMI 파일

5. 실험 및 분석

본 논문에서 제안하는 분할 매핑 기법의 효율성과 효용성을 검증하기 위해서, 본 절에서는 시뮬레이션 기법을 이용한 실험을 수행한다.

5.1 실험환경

실험 시 사용된 예제는 앞서 설명에서 예제로 사용된 SFT(Secure File Transfer) 소프트웨어이다.

그림 13은 SFT 설계의 전체적인 모습을 보여주고 있다. 먼저 유저로부터 전송할 파일이름과 전송할 대상의 IP주소를 받은 뒤에 Initializing 단계에서 초기화를 수행한다. 그 뒤에 DataReading과 Encryping, Network-Sending이 동시에 일어나게 되는데, DataReading에서는 파일을 읽어 들이는 작업을 하고, Encryping에서는 읽어 들인 파일을 암호화하며 NetworkSending에서는 암호화된 파일을 네트워크로 전송한다.

SFT는 암호화 시에 AES 암호화 알고리즘을 사용하는데 실험에서는 128bit키를 사용하는 모드로 설계 되었으며, 암호화 하는 대상 데이타는 256byte의 크기를 갖

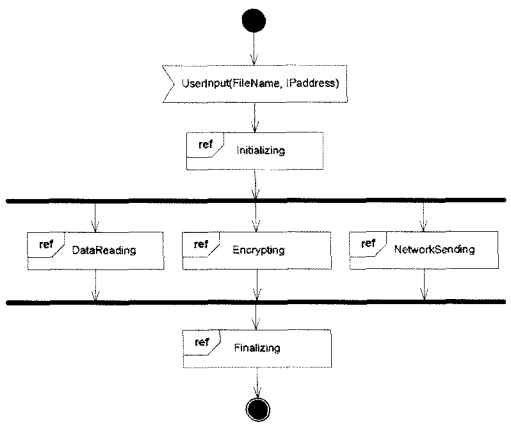


그림 13 시스템 전체 관점에서의 IOD

표 2 실험을 위한AES알고리즘 수정표

	본래의 AES	수정된 AES
암호화round 횟수	10	2
Key준비 round 횟수	44	9

는 파일로 시뮬레이션 되었다. 한편, 실험에서 사용되기에 수행시간이 너무나 길어져, 데이타 분석이 용이하지 않기에 AES알고리즘을 축소시킨 상태로 시험하였다. 축소시킨 부분은 표 2와 같다.

한편 SFT가 적용될 대상이 되는 HAD는 그림 14와 같이 정의 되었다. GPP에 해당하는 CPU가 두 개인 경우를 가정하고, 암호화를 위한 DSP는 4개인 경우를 생각하였다. 각 하드웨어 컴포넌트 사이의 버스와 직접 연결의 속도는 매우 빠르다고 가정하였다.

실험에서 사용할 시뮬레이터는 그림 15와 같은 구조로 개발되었다. 이 시뮬레이터는 시뮬레이션 대상이 되는 모델에서 나타나는 최소 요소들을 UML에서 정의하는 인과적 관계[4]에 의해 하나씩 실행하면서 시뮬레이션을 수행하게 된다. 이 때 이런 인과적인 관계는 3.3 (1)에서 언급된 CFG에 의해 정의된다.

그림 15에서 보는 바와 같이 시뮬레이터는 CFG에서 각각의 태스크를 프로세스로 보고, 가상 프로세서가 할당된 프로세스를 수행해 나간다. 각 프로세서의 특성을 반영하기 위해 처리 대상이 되는 각 모델 요소(CFG의 노드)마다 처리시간을 다르게 시뮬레이션 하도록 설계되었다. 즉 GPP에서 효율적으로 동작하는 모델요소는 GPP에 할당했을 때 더 빠르게 동작하고, DSP에서 효

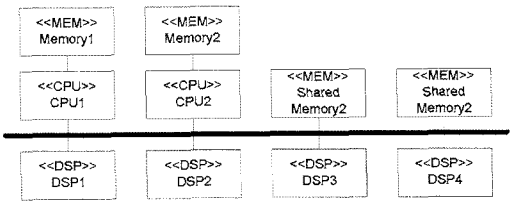


그림 14 SFT가 적용될 HAD

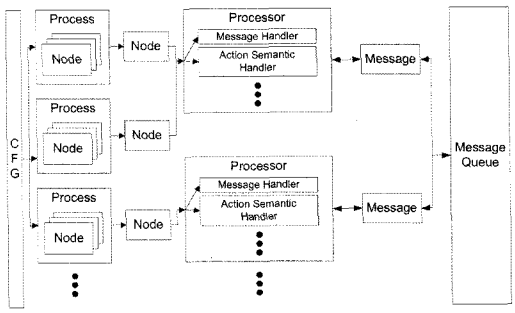


그림 15 시뮬레이터 구조

율적으로 동작하는 모델 요소는 DSP에 할당하였을 때 더 빠르게 동작하도록 하였다.

프로세서에 따라 얼마나 실행속도의 차이를 둘 것인가에 대해서 [17]은 다음과 같은 표를 제시하였다.

표 3 프로세서 타입 별 퍼포먼스 비교표

PE Type	Region					
	A1	A2	A3	A4	A5	A6
ASP	10	1	0.5	0.5	0.5	0.5
GPP	1	1	1	1	1	1

위 표에서 A_1부터 A_6까지의 region은 각각 A_1부터, 가장 응용대상에 대해 밀착된 태스크부터 가장 범용적인 작업들에 대해서 수행하는 A_6까지 태스크를 의미한다. 또한 ASP(Application Specific Processor)는 응용대상에 최적화된 프로세스를 의미하는 것으로 본 논문에서 사용하는 DSP가 이에 해당한다.

이 실험에서는 표 3을 바탕으로 GPP에 적합한 모델 요소를 A_3~A_6로 보고, GPP에서 실행될 때 보다 DSP에서 실행될 때 2배의 실행 지연이 있도록 하였다. 반대로 DSP에서 실행되기 적합한 모델요소를 A_1으로 보고, DSP에서 실행될 때 10배의 실행 지연이 있도록 조정하였다.

5.2 실험 설계

본 논문에서 제안하고 있는 알고리즘은 MPSoC 환경과 같이 매핑대상이 되는 하드웨어 컴포넌트의 종류가 다양한 경우에 대해서 적합하도록 고안이 되었다. 이에 비해 기존의 연구들은 모델 레벨에서 동일한 특성을 갖는 여러 하드웨어 노드에 매핑하는 것이 주된 것이었다.

본 실험에서는 본 논문에서 제안하는 알고리즘이 다양한 하드웨어 컴포넌트가 있는 경우에, 기존의 연구에 비해서 얼마나 더 효과적인 결과를 얻어내는지에 대해 알아보고자 한다. 시뮬레이션 기반의 실험을 위해 준비된 내용을 정의하면 다음과 같다.

- 첫 번째 실험은 본 연구에서 제시한 알고리즘이 적용된 경우로써, 이 경우는 태스크 및 하드웨어 IP의 타입과 CRF를 고려하여 할당하는 경우이다.
- 두 번째 실험은 식별된 태스크들에 대한 처리 속성이나 타입을 고려하지 않고, 프로세서의 부하만을 고려하여 태스크를 할당하는 경우이다.

5.3 실험 결과 및 분석

개발된 시뮬레이터를 이용하여 SFT 알고리즘에 대한 PSM 모델의 태스크 실행 과정에 대하여 시뮬레이션 하였다. 시뮬레이션 수행 결과는 Time Tick을 기준으로 태스크 실행의 각 행위에 대하여 로깅(logging)되었다. 부하의 계산은 프로세서에서 해당 tick에 작업을 하였을 경우 1, 그렇지 않고 태스크 간의 동기화를 기다리거나 처리할 태스크가 없을 경우 0으로 하고 200tick에 대해서 평균을 낸 값을 부하로 보았다. 그림 16은 본 연구에서 제시한 알고리즘을 기반으로 실험한 경우이다.

그림 16에서 보는 바와 같이 ARM1의 경우는 File을 읽어 들이는 태스크들이 할당되었기 때문에 초기에 부하가 집중되고, 마지막에 파일을 닫기 위해 잠시 부하가 다시 늘어나는 것을 볼 수 있다. ARM2의 영우는 네트워크로 암호화된 데이타를 전송하는 태스크들이 할당되어있다. SFT의 설계에 의하면 이 태스크들은 암호화가 끝나기를 기다리면서 계속 polling을 하게 되는데, 이로 인해서 ARM2의 경우 지속적으로 부하가 걸리는 것을 볼 수 있다. 한편, 각 그래프마다 0.0까지 Utilization이 떨어지는 경우들이 있는데, 이것은 병렬화되면서 각 태스크가 서로 동기화(join)하기 위해 생기는 유휴시간(Idle time)으로 볼 수 있다. 그림 16에서 DSP3와 DSP4에 대한 시뮬레이션 결과는 DSP2와 동일한 패턴을 보이기 때문에 지면상 생략하였다.

그림 17은 프로세서 부하만을 고려하는 것으로써, 이는 단위 태스크의 실행 시간과 태스크간의 상호 작용만을 고려하는 경우이다. 즉 태스크 및 하드웨어 IP 등의 타입 특성을 반영하지 않은 경우이다. 이 경우에서도 DSP3와 DSP4가 DSP1과 동일한 수행 패턴을 보이기 때문에 시뮬레이션 그래프를 생략하였다.

그림 17의 경우, 본 논문에서 제안하는 자동 매핑 알고리즘을 사용한 경우보다 부하가 고르게 분산된 것을 볼 수 있다. 그러나 암호화 알고리즘에 사용되는 연산에 최적화된 DSP에 암호화 관련 태스크를 할당하고, 범용적인 제어 루틴이나 입출력에 관련된 태스크를 GPP에 할당한 자동 매핑 알고리즘을 사용한 경우에 비해서 전체 처리시간이 오히려 증가하는 것을 볼 수 있다.

한편 그림 16에서 ARM1의 경우 부하가 전혀 걸리지 않은 부분이 많으므로, OS가 존재하는 시스템의 경우 ARM1에 OS를 탑재시키는 것을 고려할 수 있다.

위와 같은 실험을 통해 정리된 시뮬레이션 결과의 수행시간 비교는 표 4와 같이 정리하였다. 표 4로부터 두 방법의 수행시간의 40%정도의 차이가 있음을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 소프트웨어의 요구사항을 기준으로 작성되는 PIM으로부터 하드웨어 아키텍처를 고려하는 PSM으로의 매핑을 위한 방법을 제시하였다. 기존에 분산 환경이나 멀티 프로세서 시스템에서의 태스크 할당 기법에 대한 연구들이 많이 제시되었으나, 모델에 레벨에서의 태스크 할당 기법이 아니거나 플랫폼 - 태스크가 할당되는 환경 및 하드웨어 요소 - 이 하나의 네트

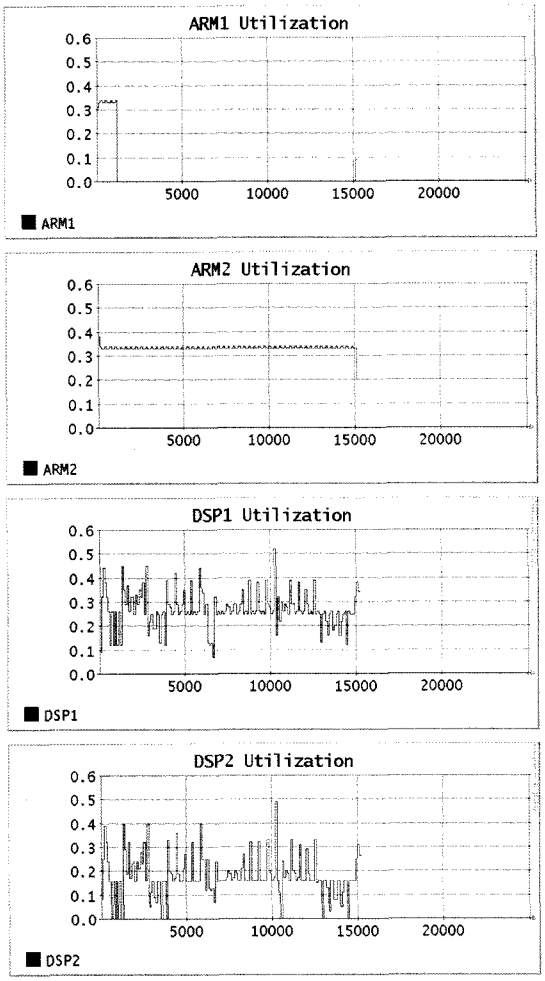


그림 16 제시한 알고리즘에 대한 시뮬레이션 결과

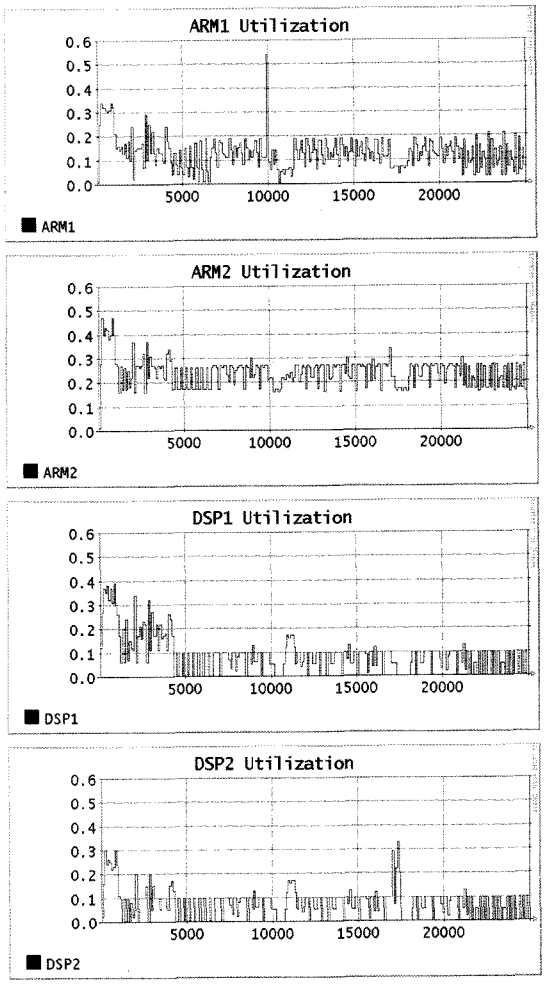


그림 17 프로세서 부하만을 고려하는 시뮬레이션결과

표 4 시뮬레이션에 대한 수행시간 비교표

	제시한 할당기법	프로세스 부하만을 고려한 할당기법
Elapsed tick	15211	24154

워크 노드로서 메모리와 컴퓨팅 유닛을 동시에 갖는 경우에 대한 연구들이었다. 또한 OO(Object Orientation) 기반이 아닌 직접적으로 태스크를 추출할 수 있는 경우에 대한 연구들이 대다수를 이루고 있다.

그러나 본 연구에서는 OO기반의 모델을 기초로 CFG를 생성하고, 이를 통해 태스크를 도출하였다. 또한 도출된 태스크들로부터 병렬성을 찾아내어 태스크를 더 작은 단위로 쪼개었다. 그 다음 각 태스크가 갖는 특성별로 태스크를 식별하였고, 그것을 CRF와 태스크의 특성을 기준으로 하여 알맞은 하드웨어 노드에 할당하는 알고리즘을 제시하고 있다. 이는 MPSoC의 특성이라 할 수 있는 다양한 하드웨어 노드의 존재와 더불어 메모리와 컴퓨팅 유닛의 분리된 상황을 반영하기 위한 것으로 볼 수 있다.

향후 연구로서는 주어진 모델로부터 실행 가능한 소스 코드를 생성하기 위한 방법들에 대하여 정의하고, 코드 생성을 위한 자동화된 도구를 개발하는 것이다. 또한 이를 기반으로 메모리 사용량을 비롯한 임베디드 시스템의 자원 요구량 등을 분석하기 위한 방법들도 연구가 필요하다.

참 고 문 헌

[1] Douglass, B.P. "Real-Time UML Workshop for Embedded Systems," Newnes, 2007.

[2] 하순회, "MPSoC용 임베디드 소프트웨어 설계 및 검증을 위한 모델 기반 프레임워크", 정보과학회회지, Vol.24, No.8, pp. 12-18, Oct. 2006.

[3] 홍장의, 배두환, "멀티프로세서용 임베디드 소프트웨어의 MDA 기반 개발", 정보과학회지, Vol.24, No.8, pp. 19-25, Oct. 2006.
[4] OMG. "UML 2.0 Superstructure Specification," 2004. Doc #ptc-04-10-20.
[5] Lewis, G. A., Meyers, B. C. and Wallanu, K. "Workshop on Model-Driven Architecture and Program Generation," Software Engineering Institute, 2006. CMU/SEI-2006-TN-031.
[6] Jerraya, A. A. and Wolf, W. "Multiprocessor Systems on Chips," Morgan Kaufmann, 2005.
[7] MASTER. "PIM to PSM mapping techniques. "Information Society Technologies, 2003. MASTER-2003-D5.1-V1.0-PUBLIC.
[8] Plishker, W., et al. "Automated Task Allocation on Single Chip, Hardware Multithreaded, Multiprocessor Systems," Workshop on Embedded Parallel Architectures (WEPA-1), 2004.
[9] Benini, L., et al. "Measuring Efficiency and Executability of allocation and scheduling in Multi-Processor Systems-on-Chip," Vol.2, No.3, 2005.
[10] Hong, B. and Prasanna, V. K. "Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput," Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004.
[11] Paulin, P.G., et al. "Parallel Programming Models for Multi-Processor SoC Platform Applied to High-Speed Traffic Management," ACM Press New York, NY, USA, 2004. Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 48-53.
[12] 전상욱, 홍장의, 배두환, "ESUML: UML 2.0 기반 임베디드 소프트웨어 모델링 방법론", 2005. 정보과학회 학술대회. pp. 343-345.
[13] Jeon, S.U., Hong, J.E. and Bae, D.H. "Interaction Based Modeling of Embedded Software," 2006. Proceedings of ISORC'06. pp. 351-355.
[14] NIST. "Advanced Encryption Standard," NIST, 2001. FIPS-PUB-197.
[15] Song, I.G., et al. "Implied Scenario Analysis in UML 2.0 Scenario Specification," Proceedings of System Reliability and Requirement Integrity 2006, 2006.
[16] 한아림, 홍장의, 배두환. "OCL을 이용한 UML2.0 행위모델의 시간 일관성 분석", 2006. KCC 2006. pp. 181-183.
[17] Paul, JoAnn M. and Meyer, Brett H. "Systems, Speedup and Heterogeneity," 2004. Workshop on Application Specific Processors.

송 인 권
2005년 한국과학기술원 전자전산학과 졸업(학사). 2005년~ 한국과학기술원 전자전산학과 석사과정. 관심분야는 시나리오 합성, MDA, 시뮬레이션

오 기 영
2006년 충북대학교 전기전자컴퓨터 공학부 졸업(학사). 2006년~ 충북대학교 전기전자컴퓨터 공학부 석사과정. 관심분야는 임베디드 소프트웨어 모델링

홍 장 의
1988년 충북대학교 전산학과(이학사) 1990년 중앙대학교 전산학과(석사). 2001년 한국과학기술원 전산학과(박사). 2002년 국방과학연구소 선임연구원. 2004년 (주)솔루션링크 기술연구소장. 2004년~ 충북대학교 전기전자컴퓨터 공학부 조교수. 관심분야는 임베디드 소프트웨어 공학, 소프트웨어 개발방법론, 소프트웨어 품질, 소프트웨어 프로세스 개선

배 두 환
1980년 서울대학교 조선공학 학사. 1987년 Univ. Of Wisconsin-Milwaukee 전산학 석사. 1992년 Univ. Of Florida 전산학 박사. 1995년~ 한국과학기술원 전자전산학과 전산학전공 교수. 관심분야는 소프트웨어 프로세스, 객체지향 프로그래밍, 컴포넌트 기반 프로그래밍, 임베디드 소프트웨어 설계, 관점 지향 프로그래밍