# MPSoC용 임베디드 소프트웨어 설계 및 검증을 위한 모델기반 프레임워크†

서울대학교 하순회

## 1. 서 론

반도체 공정기술의 발전으로 한 칩 안에 하드웨어 로직뿐 아니라 프로세서와 메모리를 탑재하는 SoC (System on Chip)의 제작이 가능하게 되었고 시간이 지남에 따라 집적도는 더욱 증대되어 한 칩 안에 멀티프로세서 시스템이 집적되는 MPSoC(Multi-Processor System-on-Chip)가 차세대 비메모리 반도체의 핵심으로 부각되고 있다. SoC를 개발하기 위한 설계 도구들은 1990년 후반부터 활발하게 개발되어 왔으나 대부분 하드웨어 설계에 초점을 맞추고 있다. 그러나 실제로 **MPSoC를 개발함에 있어서 병목이 되는 것은 하드웨어 개발이 아니라 소프트웨어의 개발이다. 이에 본 연구에서는 MPSoC를 대상으로 임베디드 소프트웨어를 설계하고 검증하는 프레임워크를 개발하는 것을 목표로 한다.**

일반적인 소프트웨어와 비교할 때에 MPSoC용 임베디드 소프트웨어가 갖는 특징을 정리하면 다음과 같다.

(1) **병행성(Concurrency)** : MPSoC용 임베디드 소프트웨어는 병렬프로그램이다. 한 프로세서에서 수행되는 코드는 다른 프로세서뿐 아니라 시스템을 구성하는 하드웨어 컴포넌트와도 서로 통신을 하며 병행적으로 수행된다.

(2) **실시간성(Real-time constraints)** : 임베디드 시스템의 일반적인 특징은 실시간 제약조건이 존재한다는 것이며 이를 해결하기 위하여 태스크의 우선순위에 기반을 둔 실시간 스케쥴링 기법을 사용한다. 널리 알려진 스케쥴 가능성 검사 기법은 단일 프로세서를 대상으로 한 것이기 때문에 MPSoC 와 같은 멀티프로세서 시스템에 직접 적용할 수 없다. 따라서 멀티프로세서 시스템을 대상으로 멀티태스크의 실시간 제약조건 만족 여부를 검증하는 기법을 개발할 필요가 있다.

(3) **자원제약(Resource constraints)** : 임베디드 시스템은 가격대 성능비와 전력소모량이 중요한 성능지표이며 이를 위하여 최적화된 소프트웨어를 개발하는 것이 필요하다.

(4) **검증(Verification)** : MPSoC는 제작 비용이 매우 비싸고 제작 시간이 오래 걸릴 뿐만 아니라 제작 후에는 수정이 불가능하기 때문에 하드웨어를 제작하기 이전에 소프트웨어를 개발하고 가상 프로토타입 시스템 위에서, 전체 시스템이 올바르게 동작하는 지를 확실히 검증해야 한다.

최근 들어 임베디드 소프트웨어를 체계적으로 개발하기 위하여 모델 기반 아키텍처(MDA: Model Driven Architecture)를 적용하는 것이 널리 확산되고 있다[1]. 시스템의 기능을 플랫폼에 독립적인 모델(PIM: Platform Independent Model)로 먼저 개발하고, 하드웨어 플랫폼 명세 사항을 입력으로 플랫폼에 종속적인 모델(PSM: Platform Specific Model)을 생성한 후, 타겟 프로세서에 탑재될 소스코드를 생성하도록 하는 것이다. 이와 같은 모델기반 프로그래밍은 플랫폼에 독립적인 소프트웨어 모듈의 재사용성을 높임으로 해서 임베디드 소프트웨어의 설계 생산성을 크게 높일 수 있다. 특히 반도체 기술의 발전으로 성능이 개선된 새로운 아키텍처가 계속 등장함에 따라 새로운 아키텍처에 최적화된 소프트웨어를 빠르게 작성하기 위해서는 가능한 한 기존의 소프트웨어 모듈을 재사용하도록 하는 것이 좋다. 또한 표준의 변화나 사용자의 요구사항이 계속 변화함에 따라 제품의 수명주기가 짧아지고, 제품의 시장집입시간(Time-to-Market)이 빨라지는 추세이기 때문에 소프트웨어의 생산성을 높이는 것은 매우 중요하다. 따라서 본 연구에서도 모델 기반의 개발 방법론을 사용한다.

현재까지 UML 기반으로 개발된 설계 도구들은 앞

† 본 연구는 정보통신부 선도기반기술사업의 "MPSoC용 임베디드 소프트웨어 설계 및 검증기술 개발" 과제로 수행되었습니다. 본 연구의 수행에 참여하고 있는 모든 연구원들에게 감사의 뜻을 전합니다.

에서 열거한 MPSoC 용 임베디드 소프트웨어의 특징을 만족하지 못하고 있다. 병행성을 고려한 소스코드를 생성하지 못하고 있으며 실시간 제약조건의 만족 여부를 검증하지 못한다. 또한 자동으로 생성된 코드의 효율성이 떨어져서 실제 제품의 생산에 사용되지 못하고 있다. 그리고 소프트웨어가 탑재된 전체 시스템의 정확성을 검증하는 방법이 확립되어 있지 않다. MPSoC용 임베디드 소프트웨어를 개발하기 위해서는 이러한 단점들이 극복되어야 한다. 이를 위해서 본 연구에서는 그림 1과 같은 모델기반의 프레임워크(가칭 HOPES)을 제안한다.

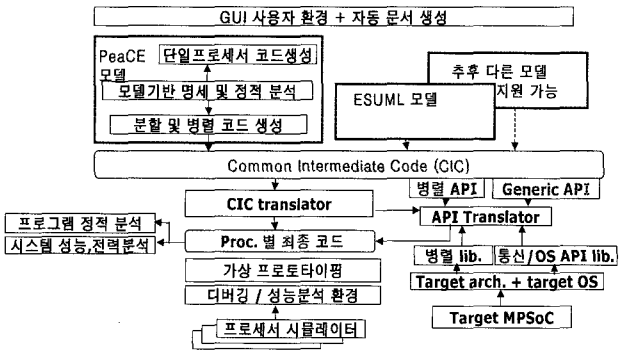


그림 1 HOPES: MPSoC 용 임베디드 소프트웨어의 설계 및 검증 프레임워크

제안하는 프레임워크의 구별되는 특징 4가지는 다음과 같다.

(1) **모델기반 프로그래밍** : UML을 임베디드 소프트웨어 개발에도 사용하는 설계 도구가 개발되어 있는데 IBM의 Rose-RT나 Telelogics 사의 TAU 가 대표적인 예이다. 반면에 학계에서는 임베디드 시스템의 병렬성을 잘 표현하는 액터기반 모델에 대한 관심이 증대하고 있다[2]. 객체지향 모델에서의 클래스는 외부로 부터의 호출에 의하여 메소드가 수행되는 수동적인 컴포넌트임에 반해 액터기반 모델에서의 액터는 데이터 입력을 받아서 각각 독립적으로 수행되는 능동적인 컴포턴트이다. The Mathworks사의 SIMULINK가 액터모델의 한 예이며 Real-Time Workshop 이라는 도구는 SIMULINK 모델로부터 자동차 제어 응용을 위한 임베디드 소프트웨어를 생성하는데 널리 사용되고 있다[3]. 이와 같이 임베디드 소프트웨어 설계를 위한 다양한 모델이 제안되어 왔으나 어느 모델이 가장 좋은지에 대하여 합의가 이루어지지 않고 있다. 이는 다양한 응용에 따라 가장 적합한 모델이 다를 수 있기 때문이다. 따라서 본 연구에서는 특정한 모델을 선택하지 않고 다양한 모델을 지원하는 환경을 개발한다. 현재 개발에 사용하고 있는 모델은 PeaCE 통합설계 환경에서 사용하는 PeaCE 모델 [4]과, UML2.0 기반의 모델이다.

(2) **아키텍처 독립적인 중간코드** : 각 모델로부터 주어진 제약 조건하에서 멀티프로세서로의 분할이 수행된다. 그리고 프로세서별로 분할된 모델로부터 공통의 중간코드(Common Intermediate Code, CIC)가 생성된다. CIC는 태스크별로 각 프로세서로 할당된 소프트웨어 코드를 명세한 것인데, 태스크간 통신을 위해서는 Generic API 를 사용하고 태스크 내부의 데이터 병렬성을 표현하기 위해서는 OpenMP [5] 명세를 사용하기 때문에 특정 플랫폼에 독립적이다. 이러한 태스크 코드 외에 CIC에는 태스크간의 상호의존도 및 하드웨어 정보, 제약조건 등이 명시되어 있다. CIC는 모델로부터 자동 생성될 수도 있으며 수동으로 직접 작성할 수도 있다. 이와 같이 CIC는 병행성과 시간제약조건이 명시된 프로그램이다. 일반적인 MDA 관점에서 보면 PIM 모델로부터 대상 아키텍처의 특징들을 고려하여 분할이 이루어진 다음에 생성된 코드이므로 PSM 수준의 명세로 생각될 수 있다.

(3) **소프트웨어 자동생성 기술** : CIC로부터 주어진 타겟 아키텍처에 최적화된 코드를 자동으로 생성하도록 한다. 이를 위해서 OpenMP pragma 를 MPI library와 통신 API 등을 사용한 병렬 프로그램으로 변환하고, Generic API를 타겟 아키텍처에 최적화한 OS API 나 통신 API로 변환하도록 한다. 통신 API나 OS API는 타겟 아키텍처에 최적화된 라이브러리로 미리 준비되어야 한다. CIC 는 각 태스크별로 분할된 코드가 표현되어 있으므로 여러 태스크가 동일한 프로세서에 할당이 되는 경우, 태스크들의 시간 제약 조건과 자원 사용량을 고려하여 최적의 런타임 시스템을 합성하는 것이 필요하다. OS가 있는 시스템의 경우에는 OS의 스케쥴링을 이용하도록 한다. 이 단계를 거치면 각 프로세서별로 수행될 코드가 생성된다.

(4) **검증 및 디버깅 기술** : 개발하는 소프트웨어의 검증을 위하여 3단계 검증 기법을 사용한다. 우선 모델단계의 검증이다. 모델의 정형성을 이용하여 프로그램의 구문 에러를 분석하고 모델로 부터의 기능 시뮬레이션을 통하여 모델의 기능성을

검증한다. 두 번째 단계는 CIC로부터 각 프로세서 별로 코드가 생성된 다음에 이루어진다. 프로세서 별로 생성된 코드를 개별적으로 분석하여 메모리 오류와 프로세서간의 통신 및 동기화와 관련된 오류를 정적 분석으로 검증한다[6]. 이렇게 정적 분석을 통하여도 걸러지지 않는 오류를 디버깅하기 위하여 마지막 단계로 가상 프로토타입 시스템을 이용한 검증 환경을 제공하도록 한다. 각 프로세서 시뮬레이터별로 디버깅 기능을 제공하도록 하고 병렬 디버깅 기능과 성능 분석 기능도 제공한다.

## 2. 모델기반 프레임워크: HOPES

### 2.1 모델 기반 프로그래밍: PeaCE 모델의 예

제안하는 프레임워크는 다양한 모델을 지원하는 것이 목표이며 우선 UML 2.0 기반의 모델과 PeaCE 모델을 대상으로 개발이 진행되고 있다. PeaCE 모델은 멀티미디어 임베디드 시스템의 설계를 목적으로 개발된 하드웨어-소프트웨어 통합설계 환경인 PeaCE[7]에서 사용하는 모델을 일컫는 용어이다.

PeaCE 모델은 멀티 패러다임 모델링을 사용하고 있으며 다음 3가지 모델로 시스템을 명세한다. 멀티미디어 응용에서 핵심이 되는 신호 처리 알고리즘과 같이 계산 중심의 태스크를 명세하는 위해서 확장된 데이터플로우 모델인 SPDF 모델[8]을 사용한다. 시스템의 제어 기능을 담당하는 제어 태스크의 명세를 위해서는 계층성과 병렬성을 표현한 statechart 의 파생 모델인 fFSM 모델[9]을 사용하다. 그리고 시스템의 기능을 명세하는 최상위 계층에서는 태스크를 기본 블록으로 갖는 Task 모델을 정의하여 사용한다.

그림 2는 PeaCE 모델을 이용하여 Divx player 를 명세한 예를 도시하고 있다. 상위 계층에는 3개의 SPDF 태스크인 "Avi reader", "H.263 decoder", 그리고 "MP3 player" 태스크들로 이루어진 태스크 그래프가 명세되어 있다. 각 SPDF 태스크는 계층적으로 명세되며 맨 하단에는 재사용성을 고려하여 명세된 기능 블록들이 사용된다. 그림 2에서는 H.263 decoder 태스크를 구성하고 있는 일부 계층적인 명세가 도시되어 있다. 이 예에서는 제어 태스크 명세는 사용되고 있지 않다.

데이터플로우 모델은 기능블록들 간의 필수적인 의존도만을 표시하기 때문에 알고리즘이 내포하고 있는 병렬성이 가시적으로 표현된다는 점에서 병렬프로그래밍 모델로 적합하다. 또한 멀티미디어 응용의 경우 데이터병렬성이 많이 존재하는데, PeaCE 모델에서 사용하는 SPDF 모델로 데이터 병렬성을 추출하는 것이 가능하다.

본 연구에서 개발하는 프레임워크에서는 시스템의 기능을 명세하는 것과 별도로 시스템의 하드웨어 아키텍처도 블록 다이어그램으로 명세한다. 프로세서와 하드웨어 블록들을 기본 블록으로 하여 명세된 하드웨어 다이어그램에는 임베디드 소프트웨어의 개발에 필요한 정보들이 명세되도록 한다.

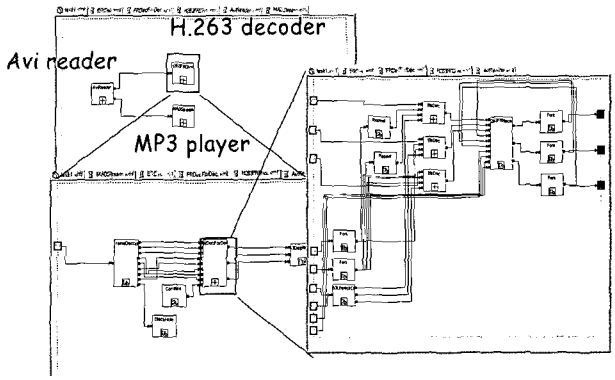


그림 2 PeaCE 모델을 이용한 Divx player의 명세

### 2.2. 아키텍처 독립적인 중간코드(CIC: Common Intermediate Code)

CIC는 여러 모델을 동일한 프로그래밍 환경에서 사용하기 위한 중간단계의 표현이다. 여러 모델을 한 환경에서 사용하기 위한 일반적인 방법은 추상화 수준을 높인 메타모델을 정의하고 그 메타 모델로부터 개별 모델로 특화시키는 것이지만, 제안하는 기법은 그 반대로 추상화 수준을 낮춘 공통의 표현을 정의한 것이다. CIC는 그림 3에서와 같이 "Architecture"와 "Task code"의 두 부분으로 구성된다.

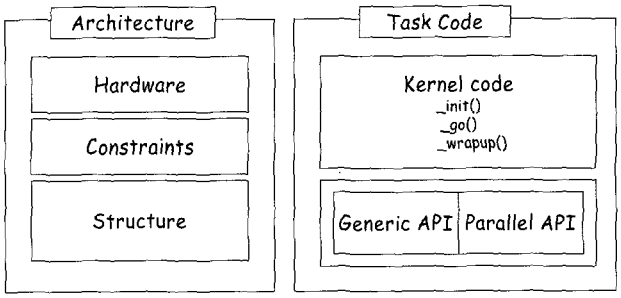


그림 3 CIC (Common Intermediate Code) 포맷

CIC의 Architecture 부분은 타겟 소프트웨어 생성에 필요한 하드웨어의 정보를 담고 있는 부분(Hardware)과 태스크들의 시간제약조건이나 전력소모량의 제약조건을 표시한 부분(Constraints), 그리고 분할된 후의 태스크들 간의 의존관계와 통신 방법들을 명시한 부분(Structure)으로 세분되며 xml 형태

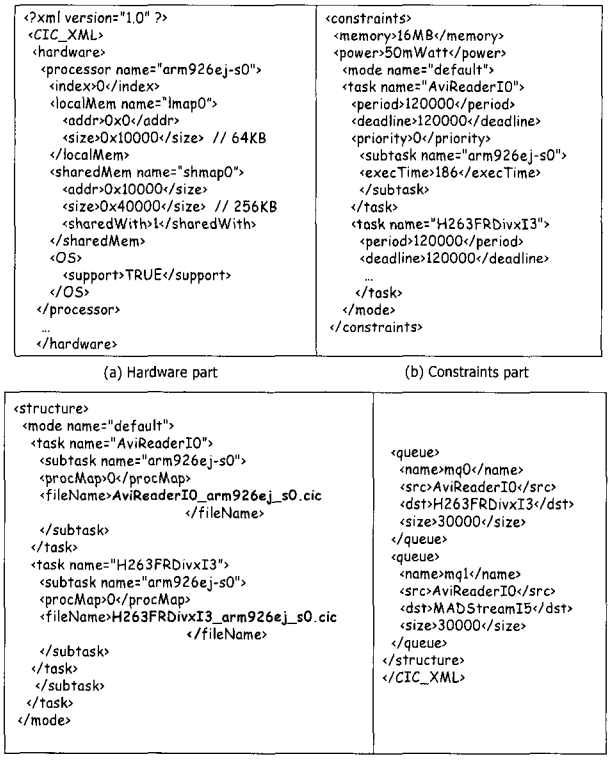


그림 4 CIC 포맷에서의 Architecture 부분

의 파일로 기술된다. 그림 4은 CIC의 Architecture 부분을 기술한 예이다.

"Hardware" 부분에서는 소프트웨어 생성에 필요한 정보를 명세하며 지역메모리와 전역메모리의 주소 공간을 기술하고, 전역메모리인 경우 공유하는 프로세서의 인덱스를 표시한다. 그리고 OS를 사용하는지의 여부도 표시한다. "Constraints" 부분에서는 시스템 전체의 메모리 크기와 전력 제약조건과 아울러 태스크별로 주기, 데드라인, 우선순위를 명세한다. 그리고 프로세서 별로 나누어진 서브태스크들의 수행시간을 명세한다. 서브태스크 별로 명세된 수행시간과 태스크들의 주기, 우선순위 등의 정보를 이용하여 각 프로세서별로 OS의 스케쥴 정책을 정하든지, OS가 없는 경우는 시간제약 조건을 만족하도록 런타임 시스템을 합성하도록 한다. "Structure" 부분에서는 태스크와 태스크 사이의 통신 방법을 기술하는데, 메시지 큐를 이용하는 통신 경로와 공유 메모리를 이용하는 통신 경로를 제공한다. 그림에서는 큐를 이용하여 태스크가 통신하는 것을 기술하였다.

또 "Structure" 부분에서는 각 서브태스크들을 정의하고 있는 "Task code"의 파일 이름을 정의하고 있다. Task code는 ".cic" 의 확장자를 갖는데 그림 3에서 묘사한 것처럼, 다음 3 함수로 나뉘어 기술된다. {서브태스크이름}_init(), {서브태스크이름}_go(), {서브

```
void h263decoder_go (void) {
        ...
        l = MQ_RECEIVE("mq0", (char *)(ld_106->rdbfr), 2048);
        ...
        # pragma omp parallel for
        for(i=0; i<99; i++) {
            //thread_main()
          ....
        }
         // display the decode frame
           dither(frame);
}
```

그림 5 CIC Task code 의 한 예

태스크이름}_wrapup(). _init() 함수는 태스크를 초기화 시키는 코드를 정의하고 _go() 함수가 태스크의 메인 코드를 정의한다. _wrapup() 코드는 수행이 끝난 뒤에 불리워지며 주로 동적으로 할당된 메모리를 삭제하고 열린 파일을 닫는 등의 작업을 한다.

그림 5는 CIC Task code의 한 예로 "_go()" 함수를 보여주고 있다. 여기서 주목할 것은 외부와의 통신을 위해서 "MQ_RECEIVE" 라고 하는 Generic API를 사용한다는 것과 태스크 내부의 데이터 병렬성을 표현하기 위해 OpenMP pragma 를 사용하고 있다는 점이다. OpenMP 변환기는 이 태스크가 사용할 수 있는 프로세서의 개수만큼 thread를 생성하여 태스크를 병렬적으로 수행하도록 한다. Generic API 에 대하여는 2.2절에서 설명한다.

제안하는 프레임워크에서는 각 모델로부터 CIC 코드를 자동으로 생성하도록 한다. 이를 위해서 모델로부터 각 프로세서별로 분할되는 서브태스크를 정의하고 그에 해당하는 Task code를 위에서 정의한 _init(), _go(), _wrapup() 함수들로 나누어 자동으로 생성해 주어야 한다. 그리고 "Constraints" 부분과 "Structure" 부분에서 서브태스크들의 수행시간 정보와 태스크간 통신 방법들에 대한 정보를 기술해 주어야 한다.

### 2.3 소프트웨어 자동생성: CIC 변환기와 API 변환기

CIC는 태스크들이 어느 프로세서로 어떻게 분할되며 분할된 태스크들이 어떻게 서로 통신을 하는지에 대한 정보를 아키텍처 독립적으로 기술하고 있다. 그러나 모델로부터 프로세서별로 태스크를 분할 할 때에는 아키텍처의 특성을 고려하는 것이 필요하다. CIC 변환기는 이러한 아키텍처 독립적인 기술로부터 각 프로세서별 코드를 생성하는 기능을 수행한다. CIC 변환기는 세 단계로 이루어지는데 첫 번째 단계는 OpenMP 변환기를 통하여 병렬프로그램으로 변환하는 단계이다. 타겟 아키텍처가 공유 메모리 구조인 경우에는 멀티스레딩 코드를 생성하고, 분산 메모리 구조인 경우에는 MPI 프로그램으로 변환하도록 한다. 두 번째 단계는 API 변환기를 통하여 Generic API를 아키텍처에 특

화된 API로 변환하는 단계이며 마지막 단계는 프로세서 별로 런타임 시스템을 합성하는 단계이다.

멀티프로세서 시스템의 운영체제는 다양하게 구성될 수 있다. 마스터-슬레이브(Master-Slave) 구조에서는 한 개의 마스터 프로세서에 마스터 OS가 탑재되고 다른 프로세서는 마스터 프로세서의 제어를 받는 슬레이브 OS에 의하여 운영된다. 반면에 슬레이브 프로세서는 OS없이 마스터 프로세서의 코프로세서로 동작을 시킬 수도 있다. 다른 방안은 모든 프로세서에 개별적인 OS를 탑재하여 전체적으로 분산 시스템을 구성하는 것이다. 또 하나의 방법은 멀티프로세서용 OS를 탑재하는 것이다.

범용 멀티프로세서 시스템과 달리 MPSoC 에서는 이형의 프로세서들로 구성되는 경우가 일반적이며 이 경우 마스터-슬레이브 구조나 코프로세서 구조로 구현하는 것이 간단하다. 이같이 다양한 경우에 동일한 태스크 코드를 재사용할 수 있도록 하기 위해서는 OS의 유무에 관계없이 동작하는 코드를 작성하는 것이 필요하며 이를 위하여 본 연구에서는 Generic API를 정의하였다. Generic API는 OS 의 유무에 관계없이 정의되는 API 이며 IEEE POSIX 1003.1-2004 표준을 추상화하여 OS 서비스와 C 라이브러리 함수 중에서 응용 프로그래머가 많이 사용하는 것을 발췌하여 정의하였다. 현재 파일 접근, 입출력, 태스크간 통신, 동기화, 타이밍 등과 관련된 API 70여개가 정의되어 있다.

Generic API는 그림 6의 과정을 거쳐서 변환된다. 타겟 프로세서에 OS가 탑재되어 있는 경우 Generic API는 "OS API translator"의 의하여 타겟 OS에 특화된 OS API로 변환된다. 반면에 OS가 탑재되지 않은 경우는 "통신 API translator"에 의하여 OS 없이 정의된 통신 API로 변환된다. 그림 6에서 OS API는 통신 API를 통하여 구현될 수 있음을 보인다. MPSoC에서는 프로세서와 메모리간의 통신 대역폭이 크고 프로세서간 특별한 통신 경로와 동기화를 위한 하드웨어가 존재할 수 있으며 OS API와 통신 API는 이런 특징을 고려하여 타겟 아키텍처에 최적화된 라이브러리로 구현되어 있어야 한다.

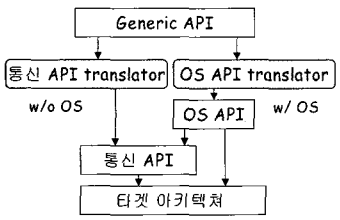


그림 6 Generic API의 변환 과정

Generic API를 OS API로 변환하는 "OS API translator"의 내부 구조를 그림 7에 도시하였다. 변환기의 입력은 CIC 코드와 아울러 변환되어야 할 Generic API의 패턴 정보와 매개 변수, 그리고 API 변환에 필요한 규칙을 명세한 파일이다. 패턴은 Generic API가 코드에서 사용되면서 반복적으로 나타나는 형태를 정의하는데 한 개의 API가 패턴으로 정의되는 것이 일반적이지만 Semaphore를 이용한 동기화의 경우 API의 짝으로 패턴이 정의될 수도 있다. 새로운 API를 추가할 경우, 이들 정보만 추가로 제공을 하면 변환이 이루어지도록 하였다. OS API 변환기는 CIC 코드를 분석하여 Generic API를 검출하고 심볼 테이블을 구성한 다음에 변환을 수행한다.

Generic API 가 변환되면 태스크별로 타겟 아키텍처에 특화된 소프트웨어 코드가 생성된다. 그 다음에는 CIC에서 명세된 태스크별 주기와 데드라인, 그리고 실시간 제약 조건을 고려하여 각 프로세서에 할당된 태스크들의 수행 순서를 결정하는 런타임 시스템을 합성한다. OS 가 있는 경우에는 CIC에서 명시된 태스크별 우선순위를 이용하여 OS 스케쥴링을 실행할 수 있으나 OS가 없는 경우에는 실시간 제약조건을 만족하도록 태스크의 수행을 제어하는 코드를 생성한다. 다양한 OS 구조에 따라 적절한 런타임 시스템을 합성하는 것이 필요하다.

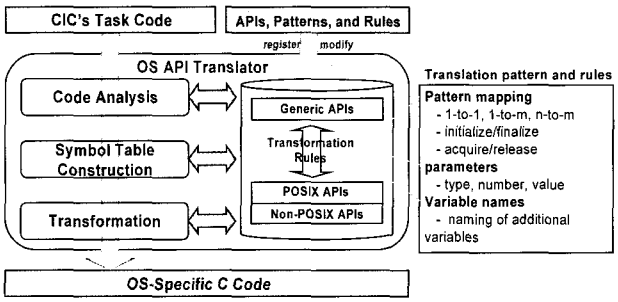


그림 7 OS API translator 의 구조

## 2.4 3단계 검증

MPSoC용 임베디드 소프트웨어 개발환경에서 가장 핵심적인 기능 중의 하나는 소프트웨어 검증 환경이다. 칩이 제작된 다음에는 소프트웨어를 변경할 수 없기 때문에 칩을 제작하기 전에 전체 시스템이 오류없이 수행되는 것을 보장해야 하며 이는 매우 어려운 문제이다. 본 연구에서는 3단계에 걸쳐서 소프트웨어를 검증하도록 하여 오류의 검출율을 최대한 높이도록 한다.

첫 번째 단계는 모델 단계이다. 각 모델로 기술된 프로그램은 모델의 검증 도구를 통하여 모델 수준의 정확성이 검증된다. 예를 들어 PeaCE 모델의 경우에

는 FSM 모델과 데이터플로우 모델을 이용하여 태스크가 표현되는데, 모델 수준의 검증을 통하여 기능 블록간 교착상태의 유무, 버퍼 오버플로우 상태의 유무 등을 점검할 수 있다. 단, 이 단계에서의 검증에서는 기능블록이 오류없이 동작하는 것을 가정한다. CIC 코드는 모델로부터 자동 생성 되므로 CIC 수준에서는 오류가 없는 것을 보장한다.

두 번째 단계는 CIC 변환기를 통하여 프로세서 별로 생성된 코드의 정적 분석을 통하여 이루어진다. C 코드의 정적 분석을 통하여 검출할 수 있는 오류는 buffer overrun error, memory leak, zero de-referencing 등이며, 본 연구에서 개발된 정적 분석기는 오류 가능성이 있는 부분을 찾아서 사용자에게 정보를 제공한다. 사용자는 정적 분석기가 제공한 정보를 바탕으로 오류의 진정성을 검사하여 참 오류를 수정하도록 한다. 정적 분석기의 성능은 오류를 검출하는 속도와 아울러 오류의 진정성이 얼마인가에 의하여 결정된다. 이 단계에서는 기능 블록 내부에 숨어있는 오류를 찾아낼 수 있다.

마지막 단계는 생성된 코드를 가상 프로토타입 시스템을 통해서 코드를 실행하면서 디버깅을 하는 단계이다. 가상 프로토타입 시스템은 실제 하드웨어 컴포넌트를 대신해서 시뮬레이션 모델들을 이용하여 만들어진 시뮬레이션 환경이며, 소프트웨어 디버깅 기능을 추가하여 구성한다. 가상 프로토타입 시스템은 실제 하드웨어의 특성을 얼마나 충실히 모델하는 가를 나타내는 정확성과 시뮬레이션의 속도가 성능 지표가 된다. SoC의 하드웨어 구조가 결정된 다음 가상 프로토타입 시스템을 만들어서 수동으로 작성된 프로그램을 탑재하여 시뮬레이션을 통하여 성능을 점검하는 수준의 상용 도구가 최근에 개발되어 있다. CoWare사의 ConvergenSC, 그리고 ARM(과거 Axys)사의 MaxSim, Mentor Graphisc사의 Seamless CVE등이 이러한 도구들이다. 이러한 시뮬레이션 도구들은 단일 프로세서 코어를 갖는 SoC의 시뮬레이션이 주된 타겟이다. 멀티프로세서 코어를 대상으로 하는 시뮬레이션도 가능하지만 성능과 확장성 면에서 제약점이 있다. 이에 비해 본 연구에서 개발하는 가상 프로토타입 시스템은 멀티프로세서 코어를 대상으로 하는 분산 시뮬레이션 환경이 될 것이며 병렬 프로그램을 디버깅하는 기능을 가진다는 점에서 구별된다.

이와 같은 기능의 검증 및 디버깅 환경 외에 시스템을 구성하는 함수별로 성능과 전력을 예측하는 도구도 개발된다. 제품을 제작하기 전에 시스템의 성능과 전력을 예측하는 것은 주어진 제약조건의 만족여부를 점검할 뿐 아니라 소프트웨어의 최적화를 위한 유용한 정보를 제공할 것이다.

## 3. 결 론

언제, 어디서, 누구나 컴퓨터를 사용할 수 있도록 하는 유비쿼터스 컴퓨팅 시대를 가능하도록 하는 핵심 기술이 MPSoC를 설계하는 기술이며, MPSoC의 설계에 있어서 병목은 더 이상 하드웨어 설계가 아니라 소프트웨어의 설계이다. MPSoC를 위한 임베디드 소프트웨어의 설계가 특히 어려운 점은 병렬 프로그램이기 때문이다. 범용 병렬 컴퓨터를 위한 프로그램 개발 환경은 오랜 시간 연구, 개발되었으나 MPSoC 용 소프트웨어가 가지고 있는 실시간성과 자원 제약조건, 그리고 검증의 필요성 등의 요구사항을 만족하지 못하기 때문에 직접 사용될 수 없다. 따라서 MPSoC의 빠르고 정확한 설계를 위해서 새로운 소프트웨어 설계 환경의 개발이 절실하게 요구되는 시점이다.

최근에 임베디드 소프트웨어의 개발을 위해 모델 기반 프로그래밍 방법론이 널리 확산되는 추세이며 본 연구도 그러한 추세를 따르고 있다. 본 연구가 관련 연구와 구별되는 점을 다시 정리하면 다음과 같다. 첫째, 다양한 모델을 지원하는 융통성 있는 프레임워크라는 점이다. 둘째, 대상 아키텍처의 통신 구조와 OS에 독립적인 공통의 중간코드를 정의하여 다양한 하드웨어 및 소프트웨어 플랫폼에 적용할 수 있도록 하였다. 셋째, 3단계의 검증 기술을 적용하여 소프트웨어의 오류를 검증하는 다양한 기능을 제공한다는 점이다.

본 연구를 통해 개발되는 프레임워크는 시스템의 기능 모델로부터 시작하여 MPSoC의 각 프로세서에 탑재할 소프트웨어를 설계하고 검증하는 모든 과정을 지원하는 통합 개발 환경이다. 연구의 결과물은 MPSoC 뿐 아니라 멀티프로세서를 대상으로 임베디드 소프트웨어를 설계하는데 사용될 수 있다.

현재의 프레임워크는 PeaCE 모델을 이용하여 Divx player 예제를 가지고 제안하는 설계 방법론을 검증한 수준으로 개발된 상태이며, 앞으로 UML 2.0 기반의 모델로부터 CIC 코드가 생성되는 기능, 멀티프로세서용 가상 프로토타입 시스템을 이용하여 디버깅하는 기술, 그리고 OpenMP 코드를 병렬 코드로 치환 기술 등이 추가로 개발될 예정이다.

## 참고문헌

[1] D. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing,

John Wiley & Sons, 2003.
〔2〕 Edward A. Lee and Stephen Neuendorffer, "Concurrent Models of Computation for Embedded Software," IEE Proceedings, Computers and Digital Techniques, Vol. 152, Issue 2, pp.239-250, March, 2005.
〔3〕 http://www.mathworks.com/products/rtw/
〔4〕 Kiseun Kwon, Youngmin Yi, Dohyung Kim, Soonhoi Ha, "Embedded Software Generation from System Level Specification for Multi-Tasking Embedded Systems", ASP-DAC'05 Vol. 1 pp 145-150 Jan 18-21 2005.
〔5〕 OpenMP C and C++ API, version 1.0, http://www.openmp.org, 1998
〔6〕 정영범, 김재황, 신재호, 이광근, "자동 오류 검출을 위한 프로그램 분석기 - 아이락", 마이크로 소프트웨어, pp.178-186, 2005년 6월.
〔7〕 http://peace.snu.ac.kr/research/peace
〔8〕 Chanik Park, Jaewoong Chung and Soonhoi Ha, "Extended Synchronous Dataflow for Efficient DSP System Prototyping," Design Automation for Embedded Systems, Kluwer Academic Publishers Vol.3, pp. 295-322, March, 2002.
〔9〕 Dohyung Kim, Soonhoi Ha, "Static Analysis and Automatic Code Synthesis of flexible FSM Model," ASP-DAC 2005 Jan 18-21 2005.

하 순 회

1985 서울대학교 공과대학 전자공학(학사)
1987 서울대학교 공과대학 전자공학(석사)
1992 U.C.Berkeley EECS(박사)
1992~1993 U.C.Berkeley EECS Post. doctoral researcher
1993~1994 현대전자 산업전자연구소
1994~현재 서울대학교 전기컴퓨터공학부 교수
관심분야 : 임베디드 시스템, 하드웨어-소프트웨어 통합설계, 병렬처리
E-mail : sha@snu.ac.kr