MAC 계층 소프트웨어의 구현 환경을 제공하기 위한 SystemC 기반의 가상 MCU 모듈의 설계 및 구현☆

Design and Implementation of a Virtual MCU Based on SystemC to Provide the Implementation Environment of MAC Layer Software

정 유 진* 박 수 진** 이 호 응*** 박 현 주****
Jeong Yoo Jin Park Soo Jin Lee Ho Eung Park Hyun Ju

요 약

무선통신 프로토콜의 구현에서 MAC 계층은 하드웨어와 소프트웨어를 결합한 시스템-온-칩으로 출시하는 것이 일반적이다. 하지만 이러한 시스템 개발에서 하드웨어의 개발에 많은 시간이 소요되므로 하드웨어의 개발 완료 이전에 소프트웨어의 개발 및 검증하기 위한 환경이 필요하다. 하드웨어와 소프트웨어의 통합 개발에서 하드웨어는 HDL(Hardware Description Level)을 이용한 RTL(Register Transfer Level) 로의 하드웨어 모델링을 통해서, 소프트웨어는 ISS를 통해 시뮬레이션 환경을 제공할 수 있다. 시스템의 개발 복잡도가 점차 증가함에 따라 기존 RTL(Register Transfer Level) 보다 높은 추상 레벨에서의 모델링을 이용하는 ESL(Electronic System Level) 설계가 이루어지고 있다. ESL 설계는 비시간 모델과 시간 모델로 나눌 수 있다. 본 논문에서는 시간 모델이 아닌 비시간 모델 시뮬레이션을 위한 MCU를 설계 및 구현한다. 제안하는 MCU는 비시간 모델에서 정확한 시간이 요구되는 부분 보다는 시스템의 동작을 쉽고 빠르게 검증함으로써 시스템 설계 초기 단계에 시스템의 최적화뿐만 아니라 설계 완료 시점을 앞당길 수 있다. 또한 운영체제를 구동할 수 있는 MCU 모듈을 설계함으로써 MAC 계층의 소프트웨어 부분을 실시간 운영체제 상에서 구현할 수 있는 환경을 제공할 수 있다. 따라서 본 논문에서는 SystemC 기반의 MCU 모듈과 실시간 운영체제 동작을 지원하는 UC/OS-II 모듈을 제안한다.

ABSTRACT

The development of wireless communication MAC layer is usually released as SoC which is a combination in hardware and software. In this system development environment, an environment for software development and verification is necessary because the hardware development takes a lot of time priori to completion. In integrated development of hardware and software, simulation environment of hardware and software provided by hardware modeling using HDL at RTL and ISS respectively. By increasing the development complexity of system, ESL design modeling systems at higher abstraction level than RTL has already prompted. The ESL design is divided untime model and time model. This paper present design and implementation of MCU for untime model simulation, not time model. Proposed MCU can optimize the system at early step of system development and move up the development completion time by verifying the system function easily and rapidly than part required exact time in untime model. In this paper, we present an MCU module based on SystemC and UC/OS-II Module providing real-time operate system.

☞ KeyWords : SystemC, MAC, OS, MCU, SystemC, 매체 접근 제어, 운영체제, 마이크로 컨트롤 유닛

1. 서 론

무선통신 프로토콜의 구현에서 MAC 계층은 하드웨어와 소프트웨어를 결합한 시스템-온-칩으

* 정 회 원 : 국립한밭대학교 정보통신전문대학원 전파공학과 졸업(석사) gilsioun@hanbat.ac.kr
** 정 회 원 : 국립한밭대학교 정보통신전문대학원 전파공학과 석사과정 macsnh@hanbat.ac.kr
*** 정 회 원 : 국립한밭대학교 정보통신전문대학원 전파공학과 박사과정 hoeung@hanbat.ac.kr
**** 정 회 원 : 국립한밭대학교 전파공학과 정교수 phj@hanbat.ac.kr

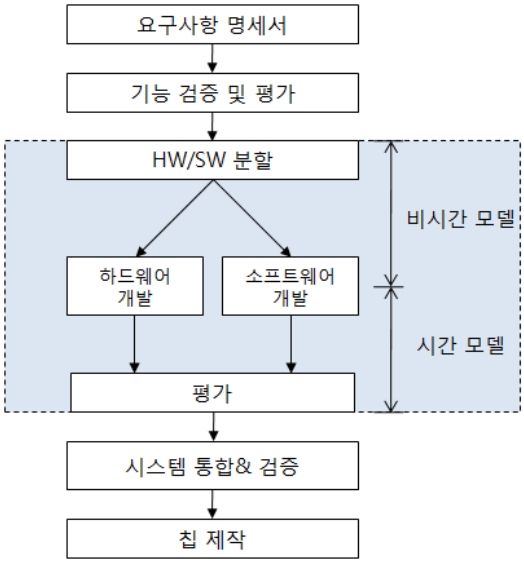
[2009/01/21 투고 - 2009/2/9 심사(2009/05/12 2차심사) - 2009/05/21 심사완료]
☆ 본 논문은 2단계 BK21 사업의 지원을 받아 수행된 연구임.

로 출시하는 것이 일반적이다. MAC 계층의 분할 구현은 일반적으로 메모리가 많이 필요하거나 높은 복잡도가 필요한 알고리즘의 경우 소프트웨어로 구현하고 반복/규칙적인 기능이 필요한 경우와 실시간 성능이 필요한 경우는 하드웨어로 구현한다.

MAC 구현에서 뿐만 아니라 소프트웨어와 하드웨어의 통합 개발에 있어서 하드웨어와 소프트웨어의 개별적인 동작 및 상호동작이 제대로 이루어져야 하며, 시스템 계발 단계에서는 이에 대한 검증도 필요하다. 하지만 하드웨어 설계의 복잡도가 증가함에 따라 일반적으로 하드웨어의 개발 완료시점이 소프트웨어보다 늦어지는 것이 대부분이다. 그 결과 소프트웨어 개발은 하드웨어의 개발이 완료되기 전까지 소프트웨어에 대한 구체적인 개발 및 검증이 어렵다. 즉 소프트웨어를 실제 개발 환경에서 검증해 볼 수 있는 환경을 갖출 때까지 기다려야 하며, 이러한 상황이 지속될수록 시스템 개발 시간은 점점 더 지연된다. 따라서 하드웨어의 개발 완료 이전에 소프트웨어의 개발 및 검증하기 위한 환경이 필요하다

하드웨어와 소프트웨어의 통합 개발에서 하드웨어의 개발 및 검증은 소프트웨어를 통해 하드웨어 시스템을 모델링하는 방법으로 가능하다.[1] 가장 기본적인 형태로 Hardware Description Language(HDL)을 이용한 Register Transfer Level(RTL) 수준의 설계 및 시뮬레이션 수행이 이루어져왔다. 하지만 시스템의 복잡도가 점점 증가하면서 기존 RTL 수준의 하드웨어 시뮬레이션은 시스템 모델링, 성능 검증, 개발 속도 면에 있어서 매우 느린 단점이 있다. 현재 이를 극복하기 위해 기존 RTL 보다 높은 추상 레벨에서의 모델링을 이용하는 ESL(Electronic System Level) 설계 기술이 연구 및 적용되고 있다[2][3].

소프트웨어의 경우 마이크로프로세서의 명령어 단위로 시뮬레이션을 수행하는 Instruction Set Simulator(ISS)를 통해서 가능하다. ISS는 RTL 수준에 비해 빠르고 프로세서의 상태를 확인할 수 있으며 Instruction Level에서 수행하기 때문에 실시간 운영체제 동작 환경을 제공할 수 있다. 하지만 ISS는 다수의 노드가 존재하는 무선 통신 시스템 환경의 특성상 시뮬레이션 수행 속도가 저하되는 단점이 있다. 또한, 특정 마이크로프로세서에 종속된 시뮬레이션 환경을 제공하므로 일반화 개념을 적용하기 어렵다.



(그림 1) SoC 설계 흐름에서의 ESL 설계

일반적으로 시스템 개발 절차는 시스템의 추상적인 설계가 먼저 이루어지며 개발이 점차 진행됨에 따라 시스템의 구체적인 설계가 이루어진다. 즉 시스템 개발 초기에는 시스템의 기능적인 면에서의 설계가 이루어지며 개발이 진행됨에 따라 시스템의 성능을 고려한 설계가 이루어진다. 이는 그림 1의 ESL 설계에서 비시간 모델과 시간 모델에 반영할 수 있다. 비시간 모델 설계 시에는 시간을 고려하지 않은 시스템 동작의 검증이 필요하며, 시간 모델 설계 시에는 정확하게 시간에 따라 제대로 동작하는지에 대한 검증이 필요하다[4]. 따라서 비시간 모델 단계에 하드웨어의 경우 RTL 수준의 설계가 필요하지 않고 보다 높은 추상화 수준의 설계가 필요하다. 또한 시스템 개발

초기 시점부터 특정 CPU로 개발하는 것이 정해지지 않을 수도 있으므로 특정 하드웨어에 종속적인 시뮬레이션을 제공하는 ISS는 비시간 모델 단계에 적합하지 않다. 반면 다양한 정확도의 시뮬레이션을 제공하므로 비시간 모델 검증보다는 시간 모델에 적합하다.

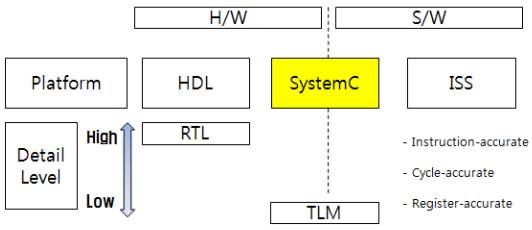
본 논문에서는 시간 모델이 아닌 비시간 모델에서의 검증을 위한 시뮬레이션 환경을 제안한다. 구체적으로 말하면 시스템 설계 언어 중에 하나인 SystemC를 사용하여 MCU 모듈을 설계한다. SystemC는 하드웨어와 소프트웨어 모두 시뮬레이션 환경을 제공하며 RTL 보다 높은 추상화 수준인 Transaction Level Modeling(TLM) 수준으로 시뮬레이션이 가능하다[5][6]. 또한 SystemC 환경 내에 ISS를 모델링 할 수 있으며 높은 추상화 수준을 유지할 수 있어서 효과적인 하드웨어 모델링 및 검증이 가능해진다. 제안하는 MCU 모듈은 소프트웨어 개발 시 정확한 시간이 요구되는 부분보다는 시스템의 동작을 쉽고 빠르게 검증함으로써 시스템 설계 초기 단계에 시스템의 최적화뿐만 아니라 설계 완료 시점을 앞당길 수 있다.

또한 본 논문에서 제안하는 MCU 모듈은 실시간 운영체제를 구동할 수 있는 기능들을 포함하고 있다. 이는 설계한 MCU 모듈 기반 위에 실제 운영체제를 구동시킬 수 있음을 의미한다. 그러므로 제안 모듈을 검증하기 위해서는 제안하는 MCU 모듈위에 실제 운영체제를 구동해야 한다. 따라서 본 논문에서는 제안하는 MCU 모듈을 검증하기 위한 방법으로 대중적으로 많이 사용하는 UC/OS-II를 이식하여 구동한다. 소프트웨어 구현 시 하드웨어와의 상호 작용을 효율적으로 처리하기 위해서는 다수의 태스크 구동 및 하드웨어 자원 관리에 대하여 실시간 처리가 필요하다. 즉 실시간 운영체제를 구동할 수 있는 제안 모듈은 MAC 계층의 소프트웨어 부분을 실시간 운영체제 상에서 구현할 수 있는 환경을 제공할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 하드웨어 모델링을 위한 관련 연구를 살펴본 후, 다양한 요구사항을 도출한다. 3장에서는 2장에서 도출한 요구사항을 바탕으로 MCU 모듈을 설계하고 구현한다. 4장에서는 설계한 MCU 모듈에 실시간 운영체제 구동 환경을 제공하는 UC/OS-II 모듈을 제안하고, 5장에서는 제안하는 MCU 모듈을 평가하기 위한 방법 및 실험 결과를 확인한다. 마지막으로 6장 결론 및 향후연구에서는 향후 본 연구의 검증 방안과 개선 사항들을 나타낸다.

2. 관련 연구

시스템 개발 시 하드웨어와 소프트웨어를 설계 및 검증하기 위한 특화된 시뮬레이터 개발 환경이 존재한다. 그림 2는 하드웨어와 소프트웨어의 시뮬레이터 개발 환경을 나타낸다. 하드웨어의 경우 HDL(Hardware Description Language)을 이용하여 RTL(Register Transfer Level) 수준의 하드웨어를 모델링할 수 있다. 소프트웨어의 경우 프로세서 시뮬레이터인 ISS를 통해 다양한 정확도로 시뮬레이션 할 수 있다. 하드웨어와 소프트웨어가 공존하는 시스템에서는 SystemC를 통해 하드웨어와 소프트웨어의 독립적인 시뮬레이터가 아닌 하나의 시뮬레이션 환경을 제공할 수 있다. 그림 2에서 보는바와 같이 하드웨어와 소프트웨어를 모두 기술할 수 있는 SystemC를 통해 RTL보다 높은 추상화 수준인 TLM(Transaction Level Modeling) 수준의 설계를 할 수 있다. 본 장에서는 각 시뮬레이터 개발 환경에 대해 자세히 살펴보고 비시간 모델에서의 MAC 구현 환경을 도출한다.



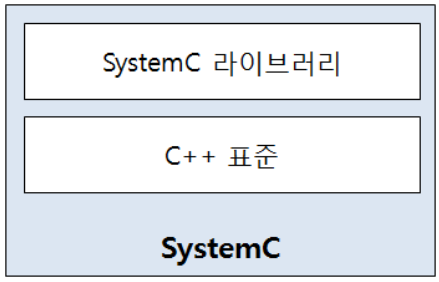
(그림 2) 시뮬레이터 개발 환경

2.1 시뮬레이터 개발 환경

하드웨어의 시뮬레이터 개발 환경은 Verilog, VHDL과 같은 HDL을 이용하여 기존 게이트 수준의 설계보다 더 높은 추상화 수준의 설계인 RTL 설계를 통해 제공할 수 있다. Verilog, VHDL 언어로 설계한 상위 수준 소스코드는 합성기(Synthesizer)에 의해 게이트 수준으로 변환하여 FPGA 또는 CPLD에 적재할 수 있고, 시뮬레이터에 의해 데스크톱 컴퓨터에서 모의실험을 수행할 수 있다. 이러한 시뮬레이션 환경은 복잡한 하드웨어에 대한 검증 용이성 및 생산성 향상을 제공한다. 하지만 RTL 설계는 정확한 검증이 가능한 반면 시뮬레이션 속도가 느리다는 단점이 있다.

소프트웨어의 경우 ISS를 통해 프로세서의 동작을 데스크톱 컴퓨터에서 시뮬레이션 할 수 있다. ISS는 특정 프로세서 상에서 동작하는 소프트웨어의 수행 시간을 측정하거나 캐쉬, 레지스터, 메모리, I/O의 동작에 대한 모든 로그를 확인할 수 있다. ISS는 Instruction 수행 시간 모델링을 제공하므로 특정 S/W의 소요 시간을 예상하는 용도로 활용할 수 있다. 또한 병렬 실행 흐름을 갖는 RTOS 기반 응용 S/W 검증에 유용하게 사용할 수 있다. ISS는 크게 Instruction-accurate, Cycle-accurate, Register-accurate로 분류할 수 있다. Instruction- accurate는 프로세서의 명령어에 대한 산술적인 연산만 관여한다. Cycle-accurate는 프로세서의 파이프라인 동작까지 고려하기 때문에 정확한 소프트웨어 수행 시간을 측정하거나 실시간 소프트웨어의 성능 평가를 수행할 수 있다. 또한, Register-accurate는 프로세서를 Register 동작 수준에서 시뮬레이션 함을 의미한다. 그러므로 Cycle 수준보다 더 정확한 시뮬레이션이 가능하다.

시스템의 복잡도가 점차 증가함에 따라 기존 RTL 수준의 설계 및 검증은 개발 생산성 및 시뮬레이션 수행 속도 저하 문제로 인해 시장 적시성을 만족할 수 없다. 따라서 이러한 문제에 대처하기 위해 RTL보다 추상화 수준이 높은 Transaction Level Modeling(TLM) 수준의 언어인 SystemC가 등장하였다. SystemC는 C++ 언어와 SystemC Library를 결합한 형태의 언어이다[7][8]. 그림 3은 SystemC 언어의 구조를 나타낸다. SystemC Library는 시스템 구조를 모델링 하는데 필요한 요소들을 제공한다. 특히 병렬성, 하드웨어 타이밍, 그리고 반응성 등과 같은 표준 C++에 없는 요소를 포함하고 있어 쉽게 시스템 모델링 할 수 있다. 기존 RTL 시뮬레이션에서 불가능했던 높은 추상화 기능을 제공하기 때문에 신속한 하드웨어 구성 및 시스템 모델링을 할 수 있는 장점이 있다. 또한, 병렬 시뮬레이션 기능을 제공하여 하드웨어 동작을 쉽게 모델링 할 수 있는 환경을 제공한다.



(그림 3) SystemC 언어의 구조

2.2 비시간 모델에서 MAC 계층 구현 환경 도출

본 논문에서 제안하는 MCU 모듈은 비시간 모델에서의 구현 환경 제공을 목표로 한다. 비시간 모델 단계에서는 시스템의 기능적인 면을 검증하는 데에 중점을 두기 때문에 하드웨어의 자세한 설계가 필요 없다. 다시 말하면 하드웨어의 경우 RTL 수준의 설계보다는 높은 추상화 레벨의 설계가 필요하다. 소프트웨어의 경우 ISS를 통한 시뮬레이션은 RTOS 기반 응용 S/W 검증에 유용하지만 특정 하드웨어에 종속적인 시뮬레이션을 제공한다. 하지만 시스템 개발 초기 단계에는 특정 MCU가 정해지지 않을 수 있으므로 일반화된 시뮬레이션 환경이 필요하다. 이는 앞서 설명한 SystemC를 통해 만족할 수 있다. SystemC는 높은

추상화 수준의 설계가 가능하기 때문에 빠른 하드웨어 모델링 및 검증이 가능할 뿐만 아니라 소프트웨어 구현 시 필요한 하드웨어의 동작을 SytemC를 통해 구현함으로써 다양한 마이크로프로세서에 일반화된 시뮬레이션 환경도 제공할 수 있다.

MAC 계층 소프트웨어의 구현은 하드웨어와의 빈번한 상호작용을 효율적으로 처리하기 위해 실시간 처리가 필요하다. 즉 실시간 운영체제를 하드웨어 위에서 동작 시키고 이러한 실시간 운영체제를 기반으로 MAC 계층의 소프트웨어를 구현하고 구동할 수 있는 환경이 필요하다.

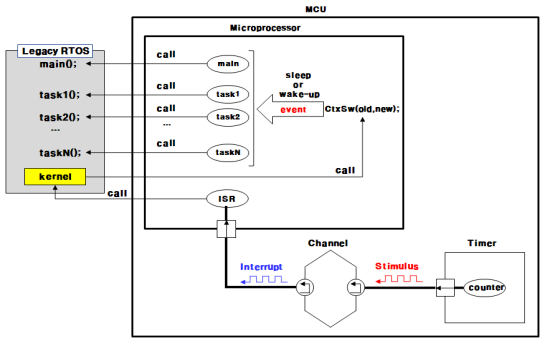
따라서 위의 내용을 요약하면 비시간 모델 단계에서 MAC 계층 구현을 제공하기 위해서는 SystemC를 이용하여 실시간 운영체제를 구동할 수 있는 가상의 MCU 모듈이 필요하다.

3. MCU 모델링

본 논문에서 제안하는 MCU 모듈은 실시간성 동작을 제공하는 위해서 실시간 운영체제를 구동할 수 있는 필수 기능을 모두 포함하여야 한다. 일반적으로 실시간 운영체제는 Tick 단위로 실행되어 스케줄 작업을 진행하며 이러한 Tick의 구현은 MCU의 타이머를 이용하여 쉽게 구현할 수 있다. 타이머에 Tick 시간마다 인터럽트를 발생하는 설정을 하면 인터럽트 서비스 루틴(ISR)에서 운영체제 스케줄러를 실행한다. 운영체제 스케줄러는 자신이 관리하는 모든 태스크들을 검사하여 Context Switch 작업을 진행한다. 결론적으로 실시간 운영체제는 MCU의 타이머와 인터럽트 기능의 지원으로 동작할 수 있으므로 본 장에서는 이러한 기능을 포함하는 MCU 모듈을 설계한다.

3.1 MCU 모듈 설계

본 절에서는 운영체제를 구동할 수 있는 MCU 모듈을 제안한다. 그림 4는 SystemC를 이용하여 MCU 모듈을 설계한 것으로 Microprocessor 모듈, 채널, Timer 모듈을 포함하고 있다.



(그림 4) SystemC를 이용한 장치 독립적 MCU 모델링

Timer 모듈은 Tick을 발생하고 Channel을 통해 Microprocessor 모듈로 주기적인 신호를 전달한다. Microprocessor 모듈의 ISR(Interrupt Service Routine)은 운영체제의 Scheduler를 호출한다. 운영체제의 Task는 SystemC의 프로세스와 1대1로 연결한다. Microprocessor 모듈은 운영체제의 Context Switch 작업을 지원하기 위해 CtxSw() 함수를 제공한다. 이 함수에서 사용하는 2개의 매개변수인 old와 new는 프로세스의 ID를 나타내며, 각각 Sleep 상태와 Active상태로의 전환을 수행한다. SystemC의 프로세스는 운영체제의 태스크와 연결되었기 때문에 SystemC의 프로세스의 상태 변경은 운영체제의 태스크에 반영된다.

3.2 MCU 모듈 구현

본 절에서는 MCU 모듈에 대한 설계를 구현한다. 그림 5은 MCU 모듈에 대한 SystemC 클래스 선언을 나타낸다. MCU 모듈은 3.1절의 설계에서 제시한 바와 같이 마이크로프로세서 모듈, 채널, 타이머 모듈을 포함한다.

그림 6은 MCU 모듈의 구현 부분을 나타낸다. MCU 모듈은 생성자 부분에서 각 모듈에 대한 인스턴스 생성 및 주변 설정을 한다. 본 예제서 타이머 모듈은 100ms 단위로 Tick이 발생할 수 있도록 임시 설정한다. Tick 설정은 실시간 운영체제

상에서 동작하는 응용의 구현에 따라 다르다. Tick 설정 후 각 모듈의 포트와 채널을 연결한다.

```
SC_MODUEL( MCU )
{
    public:
        MCU(sc_module_name  nm);
        ~MCU();
    public:

    public:
        Microprocessor *    m_pMicroprocessor;
        sc_signal<int>      m_Channel;
        Timer *             m_pTimer;
}
```

(그림 5) MCU 모듈의 헤더 소스코드

```
SC_HAS_PROCESS( MCU )
MCU::MCU(sc_mocdule_name    nm)
: sc_module(nm)
{
        m_pMicroprocessor = new UCOS_II("UCOS_II");

        m_pTimer = new Timer("timer");
        m_pTimer->SetInterval( sc_time(100, SC_MS) );

        m_pTimer->m_port( m_Channel );
        m_pMicroprocessor->m_Port( m_Channel );
        sc_trace(g_tracefiel, m_Channel, "m_Channel");
}
```

(그림 6) MCU 모듈의 구현 소스코드

Timer 모듈은 일정 시간을 단위로 주기적인 신호를 내보낸다. 그림 7은 Timer 모듈에 대한 SystemC 클래스 선언을 나타낸다. 이 그림에서 m_Port는 채널과 연결하기 위한 포트이고 SetInterval()은 Tick 간격을 조정하는 함수이다. thread()는 SetInterval()에서 설정한 시간 간격으로 포트에 신호를 송출한다.

그림 8는 Timer 모듈에 대한 구현을 나타낸다. thread() 구현 부분을 보면, for문을 이용하여 약 50회 정도 주기적인 Tick을 발생한다. 이 횟수는 본 논문의 단위 검증을 위해 50회로 실행을 제한한 것이다. Tick의 실행 횟수는 사용자의 응용 소프트웨어 요구사항에 따라 다르게 설정할 수 있고, 향후 디자인 패턴을 적용한다면 보다 높은 수준의 유연한 구조로 Tick 발생 횟수를 제어하는 설계가 가능할 것으로 예상한다.

그림 8는 Timer 모듈에 대한 구현을 나타낸다. thread() 구현 부분을 보면, for문을 이용하여 약 50회 정도 주기적인 Tick을 발생한다. 이 횟수는 본 논문의 단위 검증을 위해 50회로 실행을 제한한 것이다. Tick의 실행 횟수는 사용자의 응용 소프트웨어 요구사항에 따라 다르게 설정할 수 있고, 향후 디자인 패턴을 적용한다면 보다 높은 수준의 유연한 구조로 Tick 발생 횟수를 제어하는 설계가 가능할 것으로 예상한다.

```
#ifndef TIMER_H
#define TIMER_H

    class temp;

    SC_MODULE( Timer )
    {
        public:
            sc_port<sc_signal_inout_if<int> >   m_Port;

        public:
            Timer(sc_module_name    nm);

        public:
            void SetInterval    (sc_time    interval);
            void thread         (void);

        private:
            sc_time     m_Interval;
            int timer_test;
    };
```

(그림 7) Timer 모듈의 헤더 소스코드

```
/*********************************************************/
/* Constructor & Desctructor */
/*********************************************************/
SC_HAS_PROCESS( Timer );
TImer::Timer(sc_module_name     nm)
: sc_moudle(nm)
{
    SC_THREAD( thread );

    timer_test = 0;
    sc_trace(g_tracefile, timer_test, "timer_test);
}

Timer::~Timer()
{
    cout << "Created tiemr.vcd" << endl;
}

/*********************************************************/
/* Method */
/*********************************************************/
void Timer::SetInterval(sc_timer    interval)
{
    m_Interval = interval;
}

/*********************************************************/
/* thread */
/*********************************************************/
void Timer::thread(void)
{
    int     i;

    for( i=0; i<50; i++)
    {
        wain( m_Interval);
        m_Port->write( timer_test++ );
    }
}
```

(그림 8) Timer 모듈의 구현 소스코드

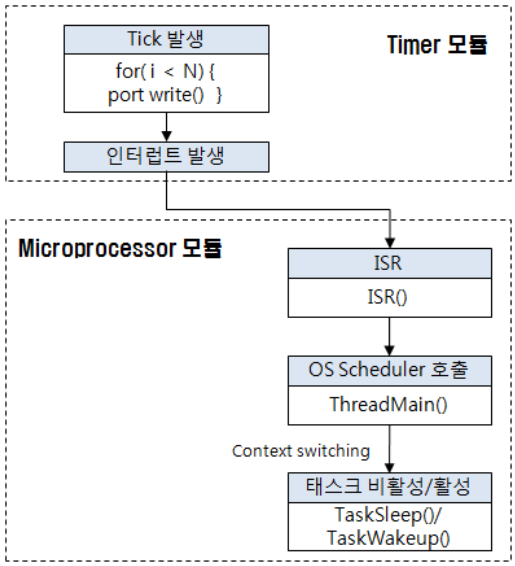
그림 9은 Microprocessor 모듈에 대한 SystemC 클래스 선언을 나타낸다. 이 그림에서 m_Port는 채널과 연결하기 위한 포트를 나타내고, ISR()과 TheadMain()은 스레드 프로세스를 나타낸다. ISR()은 타임로부터 온 Tick 신호에 대한 인터럽트 서비스 루틴을 담당한다. ThreadMain()은 운영체제의 첫 실행 흐름인 main() 함수의 역할이다.

Microprocessor 모듈은 SystemC 언어로 구현한 MCU 모듈을 이용하여 실시간 운영체제 스케줄러를 구동할 수 있도록 UC/OS-II를 이식하기 위해 사용된다. UC/OS-II를 이식하기 위한 구조 및 구현은 다음절에서 설명한다.

```
SC_MODULE( Microprocessor )
{
    public:
        sc_port<sc_signal_inout_if<int> >   m_Port;

    public:
        Microprocessor(sc_module_name   nm);
        ~Microprocessor();

    public:
        virtual void ISR        (void) = 0;
        virtual void ThreadMain (void) = 0;

    public:
        virtual void TaskSleep  (int id) = 0;
        virtual void TaskWakeup (int id) = 0;


    private:
        int         m_ISR_Count;
        sc_event    m_TestEvent[3];
};
```

(그림 9) Microprocessor 모듈의 헤더 소스코드

그림 10은 MCU 모듈의 동작 알고리즘을 Pseudo Code로 나타낸 것이다. Timer 모듈에서 주기적으로 발생하는 Tick이 발생하고, 인터럽트가 발생하면 Microprocessor의 ISR이 실행된다. ISR은 운영체제의 Scheduler를 호출하고 Scheduler는 모든 태스크를 검사하여 문맥 전환을 한 후 실행해야할 태스크를 활성화시킨다.



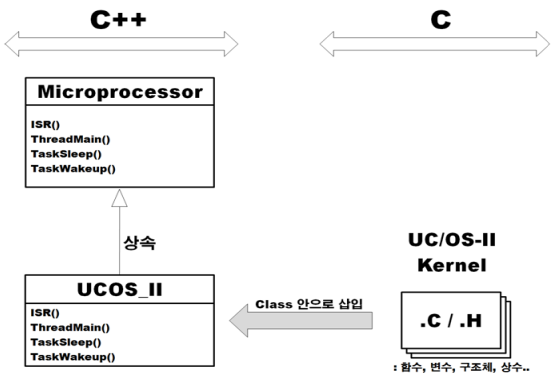
(그림 10) MCU 모듈의 동작 알고리즘

4. UC/OS-II 이식

본 논문에서 제안하는 MCU 모듈은 실시간 운영체제를 구동할 수 있는 기능을 포함하고 있으므로 설계한 MCU 모듈 위에 실시간 운영체제를 동작시킬 수 있다. 설계한 MCU 모듈을 검증하기 위해서는 실제 운영체제를 MCU 모듈에 이식하고 구동해야한다. 그러므로 본 논문에서는 제안하는 MCU 모듈의 동작을 검증하기 위해 대중적으로 많이 사용하는 실시간 운영체제인 UC/OS-II를 제안 모듈에 이식한다. UC/OS-II의 이식은 본 논문에서 제안하는 MCU 모듈의 동작을 검증할 수 있을 뿐만 아니라 MAC 계층의 소프트웨어 부분을 실시간 운영체제 상에서 구현할 수 있는 장치 독립적 환경을 제공할 수 있다. 본 장에서는 설계한 MCU 모듈의 실시간 처리를 위해 기존 UC/OS-II를 MCU 모듈에 이식하기 위한 구조를 설계 및 구현한다.

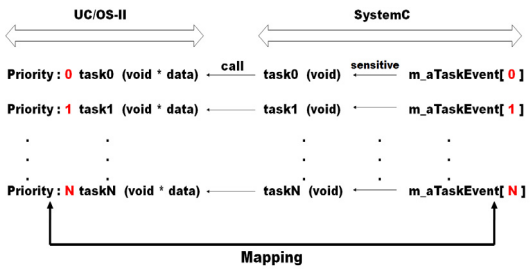
4.1 UC/OS-II 이식 구조

운영체제의 이식 작업은 그림 10의 Microprocessor 모듈을 상속하여 진행할 수 있다. 그림 11은 UC/OS-II에 대한 이식 구조 예제를 나타낸다. 본 논문에서 이식하고자 하는 UC/OS-II는 C언어를 이용하여 구현되어 있으므로 C++ 기반인 SystemC 환경에서 동작하기 위해서는 Kernel을 C++로 변경해야 한다. 이러한 작업은 커널의 모든 함수, 구조체 및 변수 등을 UC/OS-II 모듈에 모두 삽입하는 방식으로 간단히 구현할 수 있다. 따라서 UC/OS-II 모듈은 Microprocessor 모듈의 기능을 계승하면서 UC/OS-II 커널의 기능을 포함한다.



(그림 11) C로 구현된 Kernel의 C++ 변환

UC/OS-II의 태스크는 SystemC의 스레드 프로세스와 1:1로 연결한다. 연결방법은 그림 12와 같다. m_aTaskEvent는 SystemC의 프로세스에 감응할 수 있는 이벤트 데이터형 배열이다. 그림 12에서 나타낸 바와 같이 UC/OS-II 태스크의 Priority 번호는 m_aTaskEvent 배열의 인덱스로 활용한다. 즉, I번째 배열 값과 연결된 스레드 프로세스는 I번째 태스크와 연결하는 구조이다. 그러므로 UC/OS-II의 Priority 번호를 알면 특정 태스크에 대한 제어를 할 수 있다.



(그림 12) 태스크의 Priority 번호와 이벤트 배열의 인덱스 연결

4.2 UC/OS-II 모듈 구현

그림 13은 UC/OS-II 모듈에서 구현한 태스크 제어 메소드 및 컨텍스트 스위치 함수를 나타낸다.

```
void UCOS_II::OSCtxSw()
{
    int HighPrio = (int) (OSTCBHighRdy->OSTCBPrio);
    int CurrPrio = (int) (OSTCBCur->OSTCBPrio);

    OSTCBCur     = OSTCBHighRdy;
    OSPrioCur    = OSPrioHighRdy;

    TaskWakeup( HighPrio );
    TaskSleep( CurrPrio );
}

void COS_II::TaskSleep(int  id)
{
    wait( m_aTaskEvent[ id ] );
}

void COS_II::TaskWakeup(int id)
{
    m_aTaskEvent[ id ].notify( SC_ZERO_TIME);
}
```

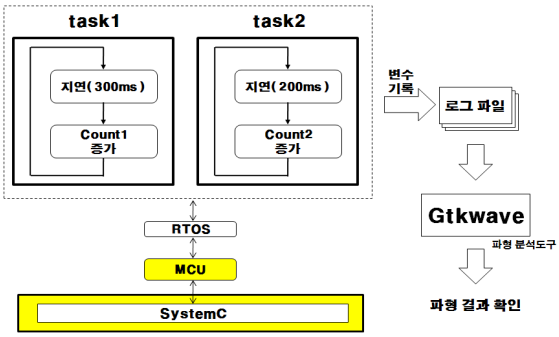
(그림 13) 태스크 제어 메소드 및 컨텍스트 스위치 함수

TaskSleep(int id)은 id에 해당하는 태스크를 정지한다. TaskWakeup(int id)은 id에 해당하는 태스크의 실행을 재개한다. OSCtxSW()는 UC/OS-II의 컨텍스트 스위치 함수를 나타낸다. 이 함수의 구현은 본래 하드웨어에 종속된 코드의 형태로 작성해야한다. 따라서 MCU의 레지스터를 직접 제어하고 어셈블리 언어를 이용한 코드 작성 기술을 요구한다.

하지만 본 논문에서는 SystemC를 이용하여 구현한 MCU 모듈에서 UC/OS-II가 동작하기 때문에, UC/OS-II의 태스크와 연계된 SystemC의 스레드 프로세스를 재개하거나, 중지하는 명령어의 수행으로 구현을 쉽게 할 수 있다.

5. 실험 방법 및 실험 결과

본 논문에서 제안하는 MCU 모듈을 검증하기 위해 실제 운영체제인 UC/OS-II를 이식하였다. 따라서 본 장에서는 제안한 MCU 모듈을 검증하기 위한 방법과 제안 모듈을 기반으로 UC/OS-II를 구동하여 SystemC 기반 환경에서 실시간 운영체제가 동작함을 확인한다.



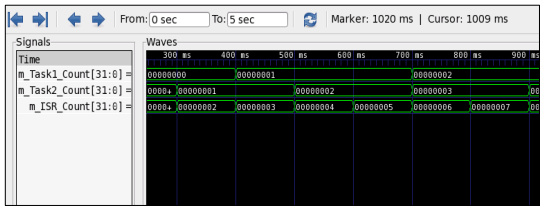
(그림 14) MCU 모듈의 검증 방법

그림 14는 UC/OS-II의 정확한 동작을 검증하기 위한 절차를 나타낸다. 2개의 예제 태스크인 task1과 task2를 실행하고, 각각 300ms와 200ms 단위로 시간 지연을 한 후, 변수 값을 증가시키는 반복 루틴 예제를 실행한다. 변수 값은 SystemC에서 제공하는 변수 추적 라이브러리를 활용하여 값의 변화를 파일에 기록한다. 기록된 파일은 파형 분석 도구인 Gtkwave를 활용하여 결과를 확인한다. 이에 대한 결과로 task1은 300ms 단위로 변수의 값이 증가해야 하고, task2는 200ms 단위로 변수의 값이 증가해야 한다. 이 결과를 만족시킬 수 있다면 다음의 2가지 사항을 신뢰할 수 있다. 첫째, MCU 모듈의 tick 발생이 정상적으로 수행되어 운영체제의 스케줄러가 단위 시간마다 수행되어 지연 태스크들에 대한 시간 관리가 정상적으로 동작한다. 둘째, 태스크의 전환이 정상적으로 동작한다.

```
void UCOS_II::task1(void *  data)
{
    m_Task1_Count = 0;
    sc_trace(g_tracefile, m_Task1_Count, "m_Task1_Count");

    while(1)
    {
        OSTimeDly(3);
        m_Task1_Count++;
    }
}

void UCOS_II::task2(void *  data)
{
    m_Task2_Count = 0;
    sc_trace(g_tracefile, m_Task2_Count, "m_Task2_Count");

    while(1)
    {
        OSTimeDly(3);
        m_Task2_Count++;
    }
}
```

(그림 15) task1과 task2의 예제 소스코드

task1과 task2의 구체적인 소스 코드는 그림 15와 같다. 운영체제의 Tick 주기를 본 예제에서는 100ms로 설정했기 때문에, 이 소스코드에서 보는 바와 같이 300ms 지연과 200ms 지연을 위해, 시간 값을 각각 3과 2로 설정한다. sc_trace 함수는 변수 값의 변화를 추적하는 함수이다. 이 그림에서 보는 바와 같이 sc_trace 함수를 수행하여 m_Task1_Count 변수와 m_Task2_Count 변수를 등록한다.



(그림 16) 변수 로그 파일에 대한 출력 결과

그림 16은 시뮬레이션 수행 후, 변수 값에 대한 로그 파일을 Gtkwave를 이용하여 출력한 화면이다. 본 예제에서는 Tick이 1번 발생할 때마다 m_ISRCount를 증가시키는 명령어를 삽입하였고, Tick의 주기는 100ms이다. 이 그림에서 보는 바와 같이 m_Task1_Count는 300ms 단위로 값이 바뀌는 것을 확인할 수 있고 m_Task2_Count는 200ms 단위로 값이 바뀌는 것을 확인할 수 있다. 결론적으로, SystemC 언어로 구현한 MCU 모듈을 이용하

여 실시간 운영체제의 스케줄러를 구동할 수 있음을 보인다.

6. 결론 및 향후 연구

MAC 계층의 구현은 처리 분야에 따라 하드웨어와 소프트웨어로 분할함에 따라 요구되는 구현 환경도 달라진다. 하드웨어와 소프트웨어의 동시 개발에 있어, MAC 소프트웨어는 호스트 시스템에서 담당하여 처리하고 MAC 하드웨어의 경우에는 실제 하드웨어를 제작하여 처리한다. 하지만 소프트웨어와 하드웨어 사이의 많은 상호 동작이 존재하기 때문에 소프트웨어에 대한 검증은 하드웨어 개발이 완료된 시점에서 이루어질 수 있다. 다시 말하면 하드웨어 제작 시 많은 시간이 소요되며 하드웨어 개발이 늦어질수록 소프트웨어의 개발 및 검증도 늦어진다. 따라서 하드웨어 제작 이전에 소프트웨어를 개발 및 검증할 수 있는 환경이 필요하다.

이는 기존 RTL 보다 추상성이 높은 모델링을 하는 ESL 설계를 통해 만족할 수 있다. 특히 본 논문에ESL 설계에서의 비시간 모델 단계에서의 소프트웨어 구현 환경 제공을 목적으로 한다. 따라서 본 논문에서는 소프트웨어 구현에 필요한 하드웨어 동작을 제공할 수 있는 경량형 MCU 모델을 제안하고 이를 SystemC를 통해 구현했다.

또한 많이 이용하는 운영체제인 UC/OS-II를 제안 모듈에 이식하여 제안한 MCU 모듈의 동작을 검증하였다. 이러한 실시간 운영체제를 구동할 수 있는 MCU 모듈의 설계는 MAC 계층의 소프트웨어 부분을 실시간 운영체제 상에서 구현할 수 있는 환경을 제공할 수 있다.

MCU 모듈의 경우, 본 논문에서 제시한 연구 수준은 SystemC 환경에서 MCU 모듈을 만들고, 이 MCU 모듈 기반에서 운영체제의 정상적인 구동에 문제가 없다는 것을 확인하였다. 향후 연구에서는 제안한 MCU 모듈에 MAC 계층의 동작을 확인해야 한다. 또한, SystemC의 스레드 프로세스를 기반으로 실시간 운영체제의 태스크를 구동하기 때문에 선점형 프로세스는 동작하지 않는다. 따라서 선점형 프로세스까지 구동할 수 있는 MCU 모듈 설계의 연구가 필요하다.

참 고 문 헌

[1] 윤덕용, 기안도, 유우석, 하순희, “플랫폼 기반 시스템 설계 방법론 제안 및 구현”, 한국정보과학회 학술발표 논문집 제 34권 제 2호(B), pp.364-372, October 2007.

[2] 윤덕용, 기안도, 유우석, 하순희, “플랫폼 기반 시스템 설계 방법론 제안 및 구현”, 한국정보과학회 학술발표 논문집 제 34권 제 2호(B), pp.364-372, October 2007.

[3] Alexander Adamov, Karina Mostovaya, Inna Syzonenko, Alexey Melnik, ″Electronic System Level Modeles for Functional Verification of System-on-Chip″, CADSM’07 , February 2007.

[4] Brian Bailey, Grant Martin, Andrew Piziali, ″ESL Design and Verification″, Morgan Kaufmann, March 2007.

[5] David C. Black, Jack Donovan, ″SystemC: From the Ground Up″, Springer, October 2005.

[6] “IEEE Standard SystemC Language Reference Manual″, IEEE Std 1666, March 2006.

[7] David C, Black, Jack Donovan, ″원리부터 배우는 SystemC″, Acorn, February 2007.

[8] Grorker Liao Martin Swan, ″SystemC를 이용한 시스템 설계″, Acorn, March 2003.

◐ 저 자 소 개 ◑

정 유 진

2007년 국립한밭대학교대학교 정보통신공학과 졸업(학사)
2009년 국립한밭대학교 정보통신전문대학원 전파공학과 졸업(석사)
관심분야 : 임베디드 소프트웨어, 무선 통신 프로토콜, 네트워크 시뮬레이터
E-mail : gilsioun@hanbat.ac.kr

박 수 진

2008년 국립한밭대학교 정보통신공학과 졸업(학사)
2008~현재 국립한밭대학교 정보통신전문대학원 전파공학과 석사과정
관심분야 : 무선통신 소프트웨어, 네트워크 시뮬레이터
E-mail : macsnh@hanbat.ac.kr

이 호 응

2004년 국립한밭대학교 정보통신공학과 졸업(학사)
2006년 국립한밭대학교 정보통신전문대학원 정보통신공학과 졸업(석사)
2006~현재 국립한밭대학교 정보통신전문대학원 전파공학과 박사과정
관심분야 : 임베디드 소프트웨어, 무선 통신 프로토콜, 네트워크 시뮬레이터.
E-mail : hoeung@hanbat.ac.kr

박 현 주

1900년 서울 시립대학교 전산통계학과(학사)
1992년 서울대학교 전산과학과 대학원 졸업(석사)
1997년 서울대학교 전산과학과 대학원 졸업(박사)
2004~현재 국립한밭대학교 전파공학과 정교수
관심분야 : 데이터베이스, 무선통신 소프트웨어.
E-mail : phj@hanbat.ac.kr