Regular paper http://dx.doi.org/10.13067/JKIECS.2015.10.5.539

듀얼 버스 시스템에서의 공유 메모리 커널 모듈 구현

문지훈* · 오재철**

Implementation of Kernel Module for Shared Memory in Dual Bus System

Ji-Hoon Moon* · Jae-Chul Oh**

요 약

본 논문에서는 프로세서별 서로 다른 버스에 서로 다른 운영체제를 갖는 멀티코어 시스템에서 공유 메모리 기능을 구현하고, 임베디드 리눅스 시스템을 통하여 두 프로세서 사이에서 공유 메모리 기능을 실험하였다. 듀얼 버스 구조에서 공유 메모리 구현을 위해 메모리 컨트롤러를 이용하였으며, 리스트 자료구조를 통하여 공유 메모리 세그먼트를 관리한다. AMP 멀티 코어 실험을 위하여 2개의 프로세서 코어에 리눅스 운영체제를 탑재하도록 하였다. 그리고 공유 메모리 테스트를 위하여 구현된 커널 모듈을 이용하여 공유 메모리 생성 및 이용이 가능함을 확인 하였다.

ABSTRACT

In this paper, shared memory feature was developed in multi-core system with different OS for different processor-specific bus, while conducting an experiment on shared memory feature between the two processors based on embedded Linux system. For the purpose of developing shared memory in dual bus structure, memory controller was used, while managing shared memory segment through list data structure. For AMP multi-core test, Linux OS was installed in 2 processor cores. In addition, it verified the creation and use of shared memory by using kernel module implemented to test shared memory.

키워드

Embedded System, Dual Core, Shared Memory
임베디드 시스템, 듀얼 코어, 공유 메모리

Ⅰ. 서 론

최근 우리 생활 속에서 임베디드 기기의 사용은 일상화 되어가고 있다. IT 환경의 발달로 3D-TV, VOD 등과 같은 다양한 영상정보매체 기술은 매우 다양한 분야로 발전을 거듭하고 있다[1-3]. TV 시청을 위해 셋톱박스를 이용하여 실시간 방송 및 VOD 서비스를 이용하며, 현재 스마트 폰은 임베디드 프로세서 및 하드웨어의 발전으로 전화뿐만 아니라 예전의 PC에서 가능했던 작업이 가능하게 되었다. 이러한 작업을 수행하기 위해서 최근 임베디드 시스템은 고성능화로 인해 단순한 주변기기를 벗어나 다양한 사용자가 원하는 복잡한 기능을 충분히 소화해낼 만큼의 뛰어난 능력을 필요로 하고 있다[4].

듀얼 코어는 하나의 프로세서 안에 여러 개의 독립적인 실행코어를 둠으로써 프로그램을 병렬적으로 처

* 이니텍 보안개발2본부 DB보안팀 차장(jihoon.moon@initech.com)
** 교신저자(corresponding author) : 순천대학교 컴퓨터과학과 교수(ojc@sunchon.ac.kr)
접수일자 : 2015. 03. 18 심사(수정)일자 : 2015. 05. 13 게재확정일자 : 2015. 05. 23

리할 수 있게 한 프로세서이며, 병렬처리 방식으로는 pipelining, multi-processor, multi-threading 등의 방법이 있다[5]. 이러한 듀얼 코어 처리 방식은 단일 코어만 사용하였을 경우 발생되었던 프로세서 사용 병목 현상에 대한 문제를 해결할 수 있다[6]. 복잡한 프로그래밍 환경에서 프로세서 사이의 데이터 처리 효율성을 증진하기 위한 방법으로 공유 메모리를 이용하여 하나의 프로세스가 처리해야 하는 데이터를 두 개의 코어에서 처리하여 성능 향상을 증진 시킬 수 있다[7].

기존 논문의 경우는 AMP(Asymmetric Multiprocessing) 기반 환경에서 프로세서 성능 개선을 위해서 두 코어가 공유할 수 있는 공유 메모리 방식을 이용하였다[8-10]. 이 방식의 일반적인 경우는 프로세서 코어가 동일 버스를 사용하므로 프로세서별 코어가 바라보는 메모리 주소가 동일하게 되므로, 같은 공유 메모리 주소를 이용하다. 위의 경우 동일 버스를 이용하는 방식이므로 공유 메모리를 이용하여 성능을 향상 시킬 수 있다. 하지만 프로세서의 성능 개선을 위해서 프로세서 코어별 버스를 가지는 경우라면 물리 메모리 주소가 다르게 되는 문제로 운영체제를 수정하지 않는 경우라면 공유 메모리 방식을 이용하지 못하게 된다. 본 논문에서의 개선 방안으로 두 프로세서가 모두 접근할 수 있는 메모리를 추가하여, 이를 통하여 프로세서 사이의 데이터를 공유할 수 있는 메모리 영역으로 이용하도록 하였다. 하드웨어에서 설계된 공유 메모리 기능을 운영체제에서 사용하기 위해서 리눅스 커널 모듈을 이용하여 구현 하였으며, 리스트 자료구조를 통하여 세그먼트 생성, 삭제 및 프로세스 관리를 하였다.

논문의 구성은 다음과 같다. 2장에서는 멀티 코어를 위한 공유 메모리를 설명한다. 그리고 3장에서 논문에서 제안하는 듀얼 버스 시스템에서의 공유 메모리 커널 모듈을 구현한다. 4장에서는 실험을 통하여 프로세서별 버스를 가지는 시스템에서 공유 메모리 기능을 실험하며, 마지막 5장에서 결론을 맺는다.

II. 멀티 코어를 위한 공유 메모리

공유 메모리의 기본 개념은 여러 프로세스가 동시에 접근할 수 있는 메모리 영역을 의미한다. 그림 1은 일반적으로 이용되는 공유 메모리 구조를 나타낸다. 프로세스1이 새롭게 공유 메모리를 생성하게 되면, 운영체제로부터 공유 메모리 세그먼트 아이디를 획득하게 되며, 이를 이용하여 공유 메모리 영역에 데이터 읽기 및 기록이 가능하다.

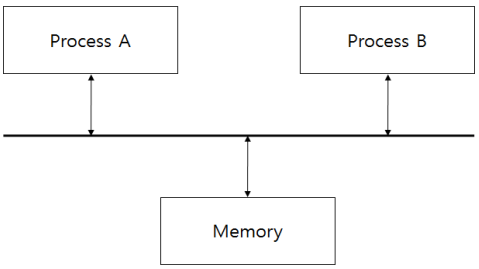

그림 1. 공유 메모리 구조
Fig. 1 Shared memory structure

프로세스2는 프로세스1에서 생성한 공유 메모리 영역을 이용하는 프로세스이며, 해당 메모리 영역에 접근하기 위해서 프로세스1에서 생성한 아이디를 이용하여 해당 메모리 영역의 공유가 가능하다.

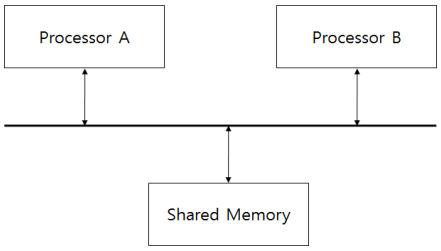

그림 2. 단일 버스 구조의 멀티 코어
Fig. 2 Multi-core of single bus structure

그림 2는 단일 버스에서의 하나 이상의 프로세서가 연결되어 있는 구조를 나타낸다. 프로세서A와 B에서 사용되는 메모리 영역이 동일하기 때문에 각 프로세서에서 바라보는 메모리는 동일한 주소를 갖게 된다. 위의 경우 프로세서마다 운영체제가 탑재되는 AMP 방식에서는 큰 어려움 없이 공유 메모리 사용이 가능하게 된다. 하지만 그림 2는 몇 가지의 문제점을 가지고 있다. 각 프로세서 코어가 이용하는 디바이스가 작은 경우는 문제가 없지만, 복잡한 디바이스를 제어하는 프로세서인 경우는 문제점을 가지게 된다. 프로세서에 연결된 버스를 통하여 데이터를 처리하게 되는

데, 많은 양의 디바이스를 이용하는 버스의 경우 시스템 병목 현상으로 성능에 문제점이 발생하게 된다.

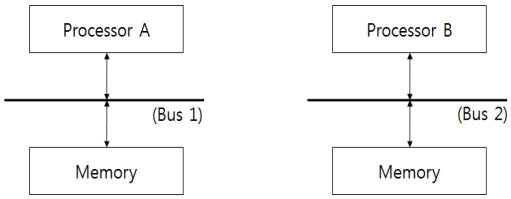

그림 3. 프로세서별 별도의 버스 형태
Fig. 3 Processor-specific separate bus type

그림 3은 각 버스에 연결되어 있는 프로세서는 자신의 메모리 컨트롤러를 갖으며, 실제 물리적인 메모리 디바이스를 갖는 형태를 나타낸다. 위의 경우 특정 프로세서의 버스 사용으로 인하여 다른 프로세서의 성능 저하 현상이 발생하지 않는 특징을 갖게 된다. 하지만 프로세서별로 이용되는 물리 메모리 주소가 다르게 되므로, 두 프로세서에 동시에 접근할 수 있는 공유 메모리 영역을 이용할 수 없게 된다.

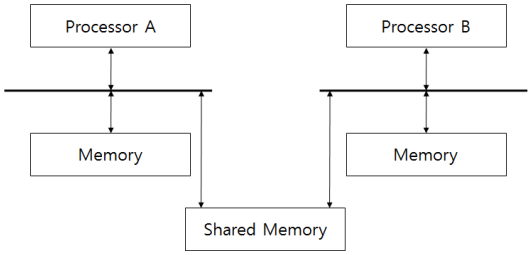

그림 4. 두 프로세서에서 이용 가능한 메모리 영역
Fig. 4 Memory area available between dual processors

그림 4는 프로세서별로 버스가 분리되어 있는 구조에서 데이터를 공유하기 위한 방법으로 새롭게 공유 메모리 블록을 추가 하였다. 메모리 블록이 프로세서 A와 B 버스에 연결되어 있으므로, 각 프로세서와 데이터 공유가 가능하게 된다.

III 공유 메모리 커널 모듈 구현

그림 5는 프로세서별 버스를 가지는 듀얼 버스 기반 시스템 구성도를 나타낸다. 하나의 버스 라인에 다수의 프로세서가 연결된 시스템의 경우, 특정 프로세서에 의해서 시스템 병목 현상이 발생할 수 있다. 이러한 문제점을 해결하기 위해서 본 논문에서는 AMP 시스템에서 프로세서별 버스를 이용하는 듀얼 버스 방식으로 시스템을 구성하였다.

프로세서별 운영체제를 가지는 AMP의 특성 상, 롬으로부터 초기 부팅에 필요한 코드를 로드하기 위하여 각 프로세서마다 32 킬로바이트의 SRAM 메모리 블록을 이용하였다. 그리고 임베디드 시스템 특성상 하드웨어를 줄이기 위하여 각 프로세서에서 공통으로 이용할 수 있는 디바이스를 공용으로 구성하였으며, 대상 하드웨어는 DDR 메모리, 공유 메모리, 낸드 플래시 메모리 컨트롤러이다.

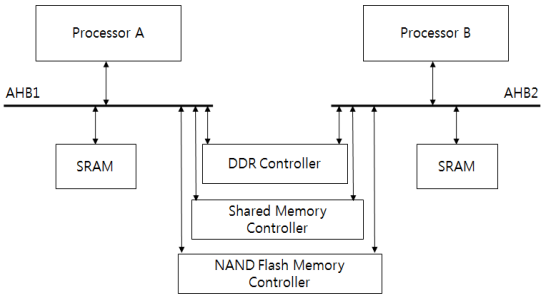

그림 5. 듀얼 버스 기반 시스템 구성도
Fig. 5 Dual bus-based system structure

공용 컨트롤러 중에서 공유 메모리 컨트롤러 부분이 논문으로 다루어지는 주제이며, 내부적으로 32 킬로바이트 SRAM 메모리를 이용하며, 8개의 논리적 세그먼트로 분할하여 프로세서 사이의 공유 메모리를 관리한다.

그림 6은 공유 메모리 컨트롤러 구조를 나타내며, 임베디드 메모리 컨트롤러를 통하여 실제 사용할 SRAM 디바이스와 연결하였다. 그러므로 프로세서A에서 B로 데이터를 공유하고자 할 경우, EMC에 연결되어 있는 SRAM 메모리를 이용하여 사용이 가능하다. 새로운 프로세스에 의해서 공유 메모리 영역의 세그먼트를 할당 받을 경우 세그먼트 아이디를 할당받게 되는데, 논문에서는 세그먼트 아이디 생성 시에 4KByte 메모리 공간을 할당 하였다.

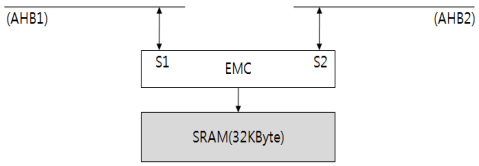

그림 6. 공유 메모리 컨트롤러
Fig. 6 Shared memory controller

표 1은 공유 메모리 세그먼트 메모리 번지를 나타낸다. Segment name은 공유 메모리에서 이용되는 세그먼트명을 나타내며, SEG1부터 SEG8까지 이용이 가능하다. 그리고 Physical memory address는 해당 세그먼트에서 이용되는 실제 물리 메모리 주소를 나타낸다.

표 1. 공유 메모리 세그먼트 메모리 번지
Table 1. Segment memory address of shared memory

Segment name	Physical memory address
SEG1	0xA200_0000 - 0xA200_0FFF
SEG2	0xA200_1000 - 0xA200_1FFF
SEG3	0xA200_2000 - 0xA200_2FFF
SEG4	0xA200_3000 - 0xA200_3FFF
SEG5	0xA200_4000 - 0xA200_4FFF
SEG6	0xA200_5000 - 0xA200_5FFF
SEG7	0xA200_6000 - 0xA200_6FFF
SEG8	0xA200_7000 - 0xA200_7FFF

예를 들어 특정 프로세스가 SEG4를 할당 받았다는 것은 공유 메모리 세그먼트 아이디 4를 할당 받았음을 의미하며, 공유 메모리에 데이터를 읽거나 쓸 경우 물리 메모리 주소는 0xA200_3000에서부터 4킬로바이트이다.

새로운 프로세스에 의해서 프로세서 사이의 공유 메모리를 이용하고자 하는 일반적인 경우, 자료구조를 이용하여 사용 가능한 공유 메모리 자원을 관리하게 된다. 하지만 논문에서 제안하는 방식은 프로세서에서 이용하는 버스가 서로 다른 방식이므로, 공유 메모리 세그먼트 관리뿐만 아니라 하드웨어적으로 특정 프로세서에게 IRQ를 전달하여 프로세서 사이의 데이터 확인이 가능하기 위해서 그림 7과 같이 메모리 컨트롤 레지스터를 추가하였다.

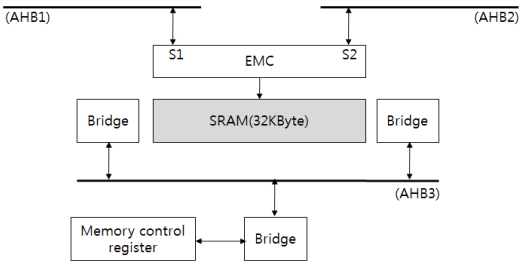

그림 7. 메모리 컨트롤 레지스터 블록 다이어그램
Fig. 7 Block diagram of memory control register

이 레지스터를 통하여 공유 메모리 컨트롤러의 세그먼트 정보뿐만 아니라, 세그먼트를 생성한 프로세서 정보가 기록되어 있다.

표 2. 메모리 컨트롤 레지스터
Table 2. Memory control register

Segment	Direction	Status
SEG1	reg[2]	reg[1:0]
SEG2	reg[5]	reg[3:4]
SEG3	reg[8]	reg[6:7]
SEG4	reg[11]	reg[9:10]
SEG5	reg[14]	reg[12:13]
SEG6	reg[17]	reg[15:16]
SEG7	reg[20]	reg[18:19]
SEG8	reg[23]	reg[21:22]

표2는 메모리 컨트롤 레지스터 비트 값을 나타내며, 이 정보를 이용하여 공유 메모리 커널 모듈은 현재 이용 가능한 세그먼트 정보를 얻는다. 각 세그먼트는 direction과 status로 구성된다. direction은 세그먼트 아이디를 생성한 프로세서 정보를 나타낸다. 프로세서 A에 의해서 세그먼트를 생성했다면 해당 세그먼트의 direction값은 0이 되며, 프로세서 B인 경우에는 이 값은 1이다. 그리고 status는 현재 세그먼트 이용 유무에 대하여 설정하는 값이다. 이 값이 0이라면 현재 세그먼트 영역을 이용하는 프로세스가 존재하지 않음을 의미하므로 새롭게 할당될 수 있는 메모리 영역이 된다. 그리고 status 값이 1이면 다른 프로세스에 의해서 세그먼트 아이디가 할당된 상태이지만, 프로세스에 의해서 데이터가 기록된 상태가 아님을 나타낸다. 따라서 이 세그먼트 영역은 다른 프로세스에

의해서 새롭게 생성할 수 없는 세그먼트를 나타낸다. 마지막으로 이 값이 2라면 프로세스에 의해서 해당 메모리 영역에 데이터가 기록된 상태임을 나타낸다. 그러므로 메모리 컨트롤 레지스터를 통하여 이용 가능한 세그먼트 정보를 얻을 수 있다.

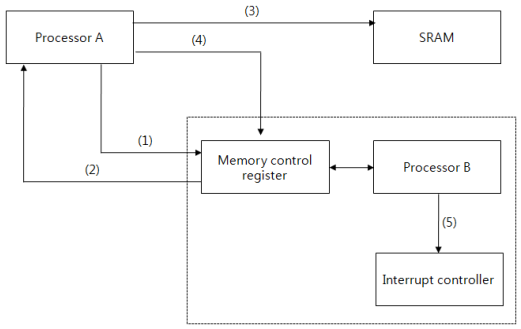

그림 8. 인터럽트 컨트롤러를 통한 IRQ 전달
Fig. 8 IRQ assert through interrupt controller

그림 8은 공유 메모리를 이용하여 프로세서 사이에서 IRQ를 전달하는 과정을 나타낸다. (1)과 (2)는 프로세서 A에 의해서 메모리 컨트롤러에 새로운 세그먼트를 할당하며, (3)은 해당 세그먼트 영역의 물리 메모리 주소에 데이터를 기록한다. 데이터 기록을 완료한 후, 메모리 컨트롤 레지스터의 해당 세그먼트의 status 비트 값을 2로 변경하여 세그먼트 데이터 기록이 완료되었음을 나타낸다. 이 시점에서 프로세서 내부적으로 해당 세그먼트의 direction 비트를 읽어 들여 IRQ를 assert할 프로세서에게 알린다. 호출을 받은 프로세서는 내부적으로 인터럽트 컨트롤러의 해당 IRQ에 인터럽트를 발생 시킨다. 위의 방법을 이용하여 운영체제와 메모리 버스가 다른 환경에서 인터럽트 컨트롤러를 이용하여 공유 메모리에 변경된 데이터가 있음을 알 수 있다.

그림 9는 공유 메모리 세그먼트 관리를 위한 자료구조를 나타낸다. 새롭게 세그먼트가 등록되면 자료구조 커널 모듈에 의해서 세그먼트 아이디를 할당받게 되며, 아이디에 해당되는 메모리 주소를 저장한다. 그리고 Master processor는 세그먼트를 생성한 프로세서를 나타내는데, 만일 프로세서 A인 경우 0, 프로세서 B인 경우는 1이 설정된다. time은 세그먼트가 생성된 시간을 저장하는 용도로 이용되며, PID는 세그먼트를 사용하는 프로세스 아이디를 의미한다. 마지막 use 항목은 세그먼트의 이용 횟수를 누적하는 정보를 담고 있다. 이는 이용 가능한 세그먼트가 제한적이기 때문에 이에 대한 문제를 해결하기 위해서 논문에서는 LFU(:Least Frequently Used) 알고리즘을 적용하여 자료구조에서 use 값이 가장 낮은 세그먼트를 반환 하도록 구현하였다.

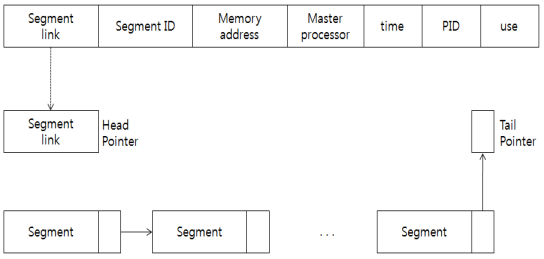

그림 9. 공유 메모리 저장 자료구조
Fig. 9 Stored data structure of shared memory

그림 10은 두 프로세서가 이용 가능한 공유 메모리 커널 모듈 메커니즘을 나타낸다. open 시스템 콜을 이용하여 사용 가능한 공유 메모리 세그먼트 아이디를 획득하게 된다. 이 경우, 프로세서 내부의 메모리 컨트롤 레지스터 중에서 이용 가능한 세그먼트를 찾게 된다. 메모리 컨트롤 레지스터에서 찾은 세그먼트 정보를 리스트 자료구조에 추가하게 된다. 자료구조에 저장되는 정보는 세그먼트 물리 메모리 영역, 세그먼트를 생성한 프로세서 정보, 시각 정보 등을 저장하게 된다.

데이터를 기록할 경우 write 시스템 콜을 이용하며, fd를 이용하여 자료구조에서 해당 세그먼트 노드를 검색하게 된다. 일치하는 노드가 검색 되었다면, 해당 물리 메모리 주소에 데이터를 저장하게 된다. 데이터 저장 시, 자료구조의 알고리즘 처리를 위해서 시각 정보 등을 업데이트 한다. 데이터 읽기의 경우 데이터 기록과 유사하게 open 시스템 콜에서 얻은 fd를 이용하여 자료구조에서 fd에 해당되는 세그먼트 노드를 찾은 후, 물리 메모리 영역으로부터 데이터를 읽어 들이게 된다. 읽어 들인 데이터를 사용자 메모리 영역으로 전달하여 공유 메모리 읽기 동작이 완료된다.

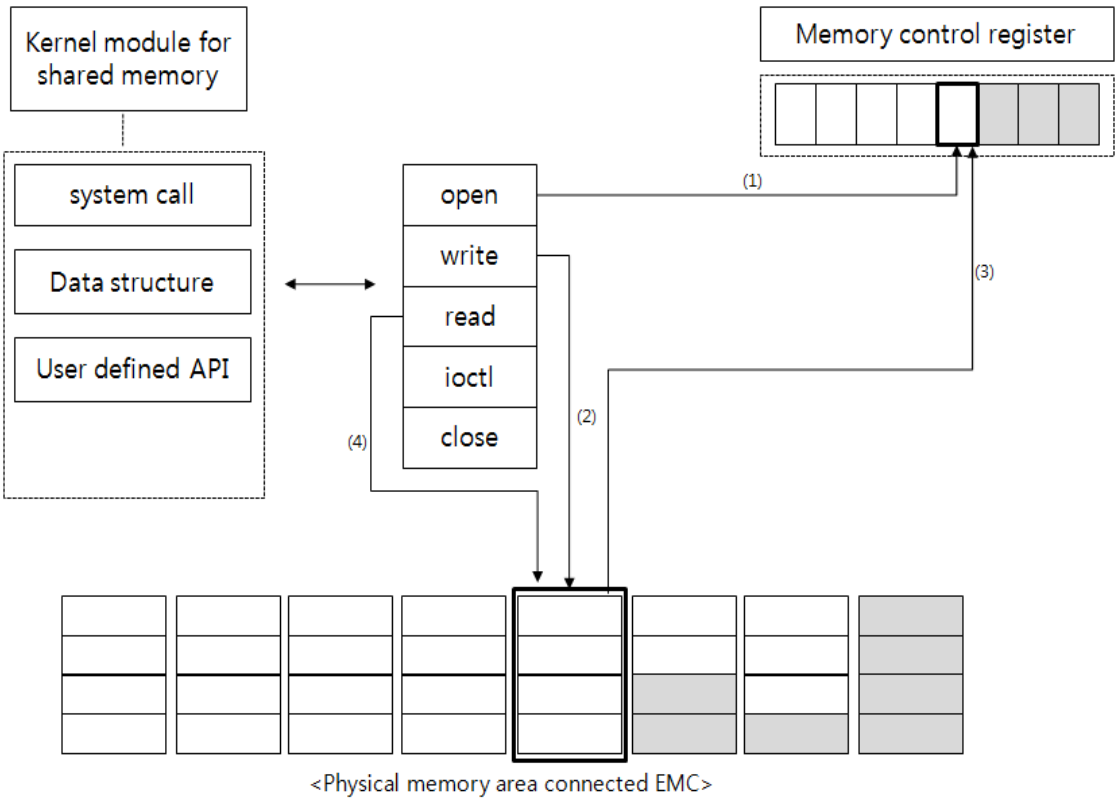

그림 10. 공유 메모리 커널 모듈 메커니즘
Fig. 10 Kernel module mechanism for shared memory

IV. 실 험

구현된 공유 메모리 커널 모듈 실험을 위해서 프로세서 코어마다 리눅스 운영체제를 탑재 하였으며, 컴파일러는 ARM gcc, 리눅스 커널 버전은 2.6.28을 이용하였으며, 파일 시스템은 ramfs를 이용하였다.

그림 11은 프로세서 코어별 버스를 이용하는 시스템에서 공유 메모리 테스트 결과를 나타낸다. 각 프로세서마다 운영체제가 탑재되는 AMP 멀티 코어 방식을 이용 하였으며, 리눅스 운영체제를 포팅 하였다. 표 3은 프로세서 코어별 수행되는 운영체제의 메모리 영역을 나타낸다. 프로세서 A에 해당하는 부분이 그림 11의 왼쪽 화면이며, 프로세서 B가 그림 11의 오른쪽 화면이다. 현재 사용되는 메모리는 256MByte를 이용하였으며, 하나의 메모리를 이용하여 두 운영체제가 동시에 수행 되도록 하기 위해서 상위와 하위 메모리를 구분하였다. 따라서 프로세서 A는 DDR의 0x0 - 0x7fffffff 메모리 번지를 사용하며, 프로세서B는 0x08000000 - 0x0fffffff를 이용 하도록 하였다. 그리고 시스템이 부팅될 때, 낸드 플래시 메모리에서 운영체제 관련 이미지를 해당 DDR 메모리 영역에 로드한다. 그러므로 프로세서 A에서 사용되는 IRQ 및 FIQ Stack은 128MByte이며, 프로세서 B는 오프셋 128MByte 이상의 메모리 주소로부터 128MByte 메모리 영역을 사용함을 알 수 있다.각 프로세서에서 사용되는 커널 이미지가 낸드 메모리에서 커널 영역으로 정상적으로 로드되면, 표 3의 DDR load 위치로 각 PC 값을 이동하여 프로세서마다 자신의 운영체제를 수행하게 된다.

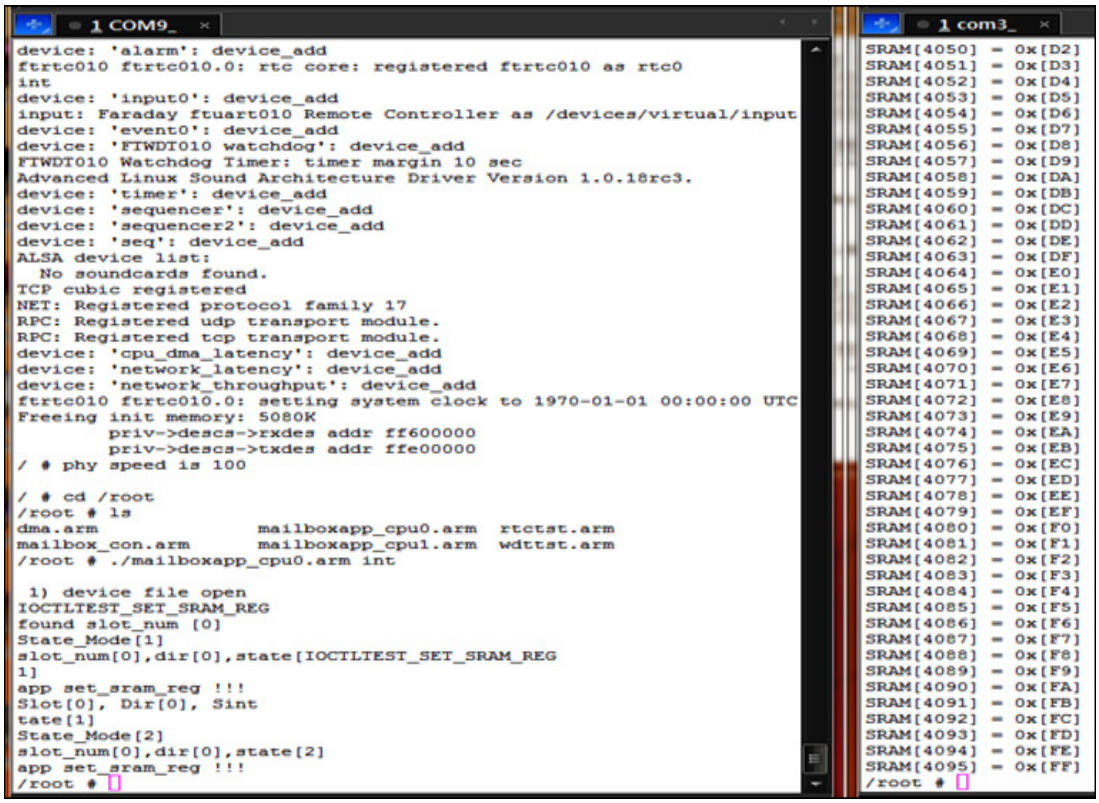

그림 11. 듀얼 버스에서의 공유 메모리 테스트
Fig. 11 Shared memory test of dual-bus

표 3. 프로세서별 메모리 위치 확인
Table 3. Verification of processor-specific memory address

프로세서 A	프로세서 B
IRQ Stack : 006fff7c	IRQ Stack : 086fff7c
FIQ Stack : 006f7f7c	FIQ Stack : 086f7f7c
DDR load : 0x8000	DDR load : 0x8008000

그림 11의 왼쪽 화면은 프로세스에서 공유 메모리 커널 모듈를 이용하여 새롭게 공유 메모리 세그먼트 아이디를 할당 받은 후, 해당 데이터 영역에 데이터 기록을 나타낸다. 그림에서 오른쪽은 다른 프로세서의 프로세스에 의해서 할당 받은 공유 메모리 아이디를 이용하여, 해당 메모리 값을 읽어 들이는 과정을 나타낸다.

그림 12의 시뮬레이션 화면은 프로세서 A에서 데이터를 공유 메모리 컨트롤러에 기록 후, 프로세서 B의 공유 메모리 컨트롤러에서 기록된 데이터가 정상적으로 읽기가 가능한지에 대한 파형을 나타낸다. 1)은 system clock을 나타내며, 현재 FPGA의 AHB는 50Mhz를 이용한다. 그리고 2)는 chip select를 나타내는데, 이용 중인 상태이면 이 값은 high로 설정되며, 그렇지 않는 경우는 low로 설정된다. 4)는 address를 나타내며, 이는 SRAM의 메모리 크기에 비례하게 된다. 그리고 5)는 입력 데이터 클럭을 나타내며, 마지막 6) 항목은 데이터 출력 클럭을 나타낸다. 그림 13은 공유 메모리 컨트롤 레지스터 초기 시뮬레이션 결과를 나타낸다.

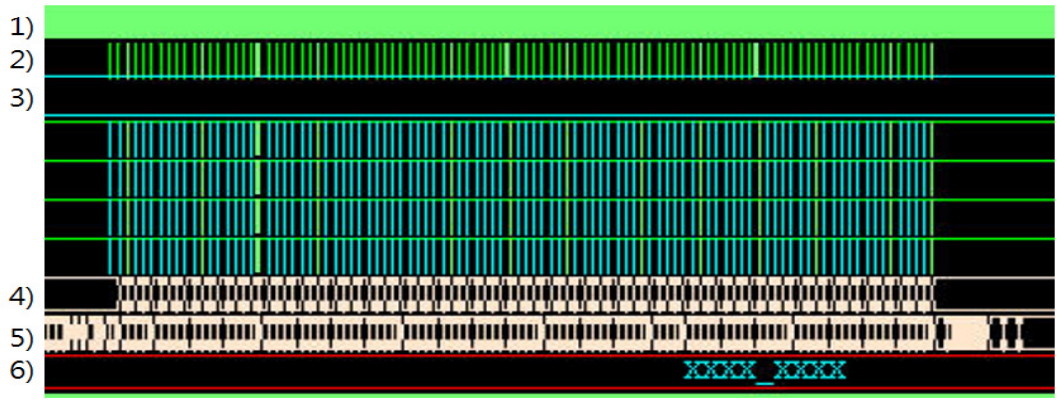

그림 12. 프로세서 A에 대한 공유 메모리 컨트롤러 시뮬레이션
Fig. 12 Simulation of shared memory controller for processor A

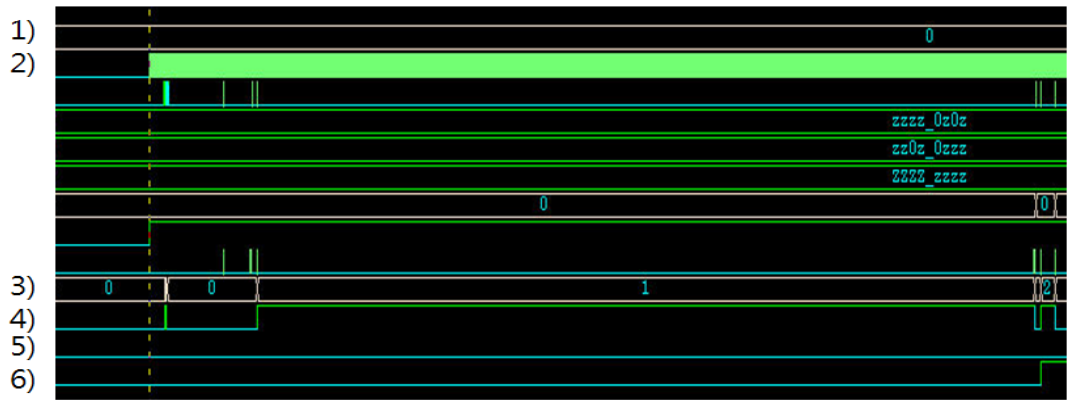

그림 13. 공유 메모리 컨트롤 레지스터 초기 시뮬레이션 화면
Fig. 13 Screen capture of shared memory control register initial simulation

1)은 address[15:0] 비트를 사용한다. address는 데이터를 기록할 메모리의 주소를 얻기 위해서 사용되며, 2)는 메모리 컨트롤 레지스터의 클럭을 나타낸다. 3)은 address에서 지정한 메모리 셀 영역의 데이터를 나타낸다. 그리고 4) 항목은 데이터를 기록하기 전 enable 신호를 나타낸다. 3) 항목에서 0->0->1->2 데이터 값이 데이터 라인으로 들어왔음을 알 수 있다. 실제 데이터가 메모리 셀에 저장되는 시점은 4)의 PWRITE 신호가 active high가 되는 지점을 그림을 통하여 알 수 있다. 위의 방법으로 메모리 셀에 데이터를 기록할 경우 클럭 동기화를 통하여 데이터가 정상적으로 저장됨을 알 수 있다.

표 4는 그림 13의 시뮬레이션에 대한 주요 신호 값을 나타낸다. 표 4의 5, 6 항목은 인터럽트를 나타낸다. 프로세서별로 버스를 가지는 듀얼 버스 구조에서 프로세서 사이의 공유 메모리를 이용할 경우 새로운 데이터가 생성되었음을 알리기 위해서 IRQ 이용하였다. 5) 항목은 프로세서 A에 대한 인터럽트 발생 유무를 나타내며, 6) 항목은 프로세서 B를 나타낸다.

실험을 통하여 응용 프로그램에서 공유 메모리 커널 모듈을 이용하여, 공유 메모리 세그먼트 생성 및 해당 메모리 영역에 사용자의 데이터를 정상적으로 저장하였다. 구현된 SRAM 방식의 공유 메모리 영역에 세그먼트 아이디를 알고 있는 다른 프로세서의 프로세스가 해당 메모리 영역의 데이터를 읽기, 쓰기가 가능함을 확인하였다.

표 4. 메모리 컨트롤러의 주요 신호
Table 4. Main signal of memory controller

item	description
1	PADDR[15:0]
2	PCLK
3	PWDATA[31:0]
4	PWRITE
5	int_a
6	int_b

V. 결 론

프로세서 코어별 버스를 가지는 듀얼 버스 환경에서 시스템 성능 향상을 위하여 공유 메모리를 이용하는 경우, 두 프로세서에서 사용되는 물리 메모리 주소가 다르게 되는 문제로 인하여 기존 운영체제에서 제공해주는 공유 메모리 사용이 불가능하게 된다. 위의 문제를 해결하기 위해서 논문에서는 2개의 버스와 연결이 가능한 메모리 컨트롤러를 이용하여 각 프로세서에서 공유 메모리로 이용 가능한 메모리 블록을 추가하였다. 추가된 메모리 블록을 이용하여 공유 메모리 구현을 위해서 커널 모듈을 구현하였다. 운영체제에서 제공해주는 형식이 아닌 특성으로, 이용 가능한 공유 메모리를 관리하기 위해서 새롭게 리스트 형식의 자료구조를 구현 하였다. 프로세서 코어마다 버스가 다른 구조에서 프로세서 내부에 추가한 메모리 컨트롤러를 이용하여 두 프로세서 사이에서 데이터가 정상적으로 공유되었음을 실험을 통하여 확인하였다.

References

[1] J. Moon and J. Oh, “Design of the Virtual SD Memory Card System on the Embedded Linux,” *J. of the Korea Institute of Electronic Communication Sciences*, vol. 9, no. 1, 2014, pp. 77-82.

[2] J. Moon and J. Oh, “Design of Shared Memory Controller Device Driver in Embedded System,” *J. of the Korea Institute of Electronic Communication Sciences*, vol. 9, no. 6, 2014, pp. 703-709.

[3] J. Moon and J. Oh, “Design of the SD Protocol Analyzer,” *J. of the Korea Institute of Electronic Communication Sciences*, vol. 8, no. 11, 2013, pp. 1697-1706.

[4] J. Jeong and C. Kim, “Improvement Method and Performance Analysis of Shared Memory in Dual Core Embedded Linux system,” *J. of the Korean Society for Internet Information*, vol. 11, no. 4, pp. 95-106.

[5] J. Kim, J. Moon, G. Im, G. Jeong, and G. Choi, “A Study on Buffer and Shared Memory Optimization for Multi-Processor System,” *J. of the Korea Information Processing Society,* vol. 9, no. 2, 2002, pp. 147-162.

[6] Hai Huang, Padmanabhan Pillai, and K. Shin, “Improving wait-free algorithms for interprocess communication in embedded real-time system,” *Proc. of the Usenix Annual Technical Conf.,* Monterey, CA, vol. 27, no. 5, Jun. 2002, pp. 303-316.

[7] Avi Siberschatz, Peter Baer Galvin, and Greg Gagne, *Operating System Principles 7th edition.* New York: John Wiley & Sons, 2006, pp. 87-105.

[8] J. Jeong, “Improvement Method and Performance Analysis of Shared Memory in Dual Core Embedded Linux System,” Master’s Thesis, *KongJu National University*, 2010.

[9] J. Jung, K. Lee, J. Kim, and C. Kim, "Performance Analysis on Dual-core Embedded System Using High Speed IPC Technique," *Proc. of the Korea Information and Communication Society*, Seoul, Korea, Nov. 2008, pp. 1494-1497.

[10] S. Jang, E. Choi, D. Kang, G. Lee, D. Kim, and J. Kim, "A Study of Performance Enhancement for the Shared Memory in the Linux O.S," *Proc. of the Korea Institute of Information Scientists and Engineers Fall Conf.*, Busan, Korea, vol. 34, no. 2, Oct. 2007, pp. 324-329.

저자 소개

문지훈(Ji-Hoon Moon)

2002년 동의대학교 컴퓨터공학과 졸업 (공학사)
2004년 동의대학교 대학원 컴퓨터공학과 졸업(공학석사)
2015년 순천대학교 대학원 컴퓨터과학과 이학박사
2013년~현재 : 이니텍 보안개발2본부 DB보안팀 차장
※ 관심분야 : 정보보안, 운영체제

오재철(Jae-Chui Oh)

1978년 전북대학교 컴퓨터공학과 졸업 (공학사)
1982년 전북대학교 대학원 컴퓨터공학과 졸업(공학석사)
1988년 전북대학교 대학원 컴퓨터공학과 졸업(공학박사)
1984년~1986년 기전대학교 전자계산학과 전임강사
1986년~현재 순천대학교 컴퓨터공학과 교수
※ 관심분야 : 임베디드시스템, USN, 네트워크 설계 및 분석