The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 18, No. 3, pp.111-117, Jun. 30, 2018. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2018.18.3.111
JIIBC 2018-3-15

효과적인 이기종 다중코어 응용 개발을 위한 SMP기반 이기종 다중코어 시뮬레이터

Heterogeneous multi-core simulator based on SMP for the efficient application development at the heterogenous multi-core environment

사공준*, 신동하**

June SaKong*, Dongha Shin**

요 약 서로 다른 기능을 가진 코어들을 집적시킨 이기종 다중코어 환경은 더욱 복잡하고 다양해지는 임베디드 시스템의 요구사항들을 만족시키는 강력한 도구이며 특화된 응용을 위해 상이한 코어 상에서 별개의 운영체제를 수행하여 적합한 환경을 구성한다. 그러나 이런 이질성은 개발 환경을 더욱 복잡하게하고 프로그래밍을 어렵게 하며 개발과 디버깅을 쉽지 않도록 만든다. 본 논문에서는 이기종 다중코어 환경을 단일 다중코어 환경으로 매핑 가능함을 보이고 이기종 다중코어 환경에서 프로세스 간 통신에 사용하는 RPMsg를 리눅스 기반으로 구축하여 여러 단계의 개발과정을 축소할 수 있음을 보인다. 이러한 단순화를 통해 이기종 다중코어 환경에서의 개발 기간을 대폭 줄여줄 수 있는 시뮬레이션 방법을 제안한다.

Abstract Heterogeneous multi-core environment integrated with different functional cores is the powerful tool for the embedded system that became more complex and diverse. Specialized application requires one chip solution with different operating system over different cores. But this heterogeneity causes difficult configuration of the development environment, makes hard to develop and test software. We show the environment of heterogeneous multi-core processing can be mapped to symmetric multi-core environment. We construct Linux based RPMsg for the data exchange between processes similar with the heterogeneous multi-core RPMsg and experiment that the proposed environment can be used to reduce the steps of the heterogeneous multi-core application development. With this simplification, we suggest simulation method for easy development and debugging the heterogeneous multicore environment that makes complex steps to simple.

Key Words : heterogenous multicore, RPMsg, simulation, easy development, debugging

Ⅰ. 서 론

어플리케이션 프로세서와 GPU, 실시간 처리용 마이크로콘트롤러, DSP, 그 외의 프로세서들을 단일 칩에 집적시키고 메모리를 공유하면서 다양한 주변기기를 동시에 가동하는 이기종 다중코어(heterogeneous multicore) 환경은 더욱 복잡하면서도 강력한 처리 능력을 요구하는 임베디드 시스템을 구성하는 새로운 수단이다.[1][2] IoT 서버, 스마트 미터, 다기능 의료기기[3], 위폐 감별기 등의 응용에서 실시간 처리 기능과 고해상도 이미지 처리, 편리한 사용자 인터페이스를 통합한 솔루션을 경제적으로 제공할 수 있는 방법이지만 이런 이기종 다중코어 환경

*정회원, 경북대학교 컴퓨터학부 대학원
**정회원, 상명대학교 융합공과대학(교신저자)
접수일자: 2018년 3월 26일, 수정완료: 2018년 5월 10일
게재확정일자: 2018년 6월 8일

Received: 26 March, 2018 / Revised: 10 May, 2018
Accepted: 8 June, 2018
*Corresponding Author: dshin@smu.ac.kr
Dept. of Computer Science, Sangmyung University, Korea

은 서로 다른 코어 프로세서들이 서로 다른 운영체제를 기반으로 메모리를 공유하면서 동일한 주소 공간을 운용하므로 특화된 응용 개발과 그에 따른 디버깅을 더욱 어렵게 한다.[4] 이러한 이기종 다중코어 환경에서의 개발을 SMP(Symmetric Multi-Processing)환경에서의 개발로 대체하면 개발 절차를 줄여서 응용프로그램 개발 기간을 효율적으로 단축할 수 있다. 본 논문에서는 먼저 관련 기술들과 이기종 다중코어 환경에서의 데이터 교환을 위한 가상 버스인 RPMsg(Remote Processor Messaging)를 살펴본다. 다음으로 RPMsg와 관련 기술을 사용해 이기종 다중코어 환경을 SMP환경으로 대체할 수 있음을 보이고 단일 운영체제 SMP 환경에서 이기종 다중코어 응용 프로그램을 쉽게 개발하기 위한 시뮬레이터 환경을 구축하여 관련 응용 개발을 보다 단순화할 수 있음을 확인하며 구현된 결과를 제시한다.

II. RPMsg 및 관련 기술

1. 이기종 다중코어 시스템 개발 환경

최근 출시된 TI, NXP, Mentor, Xilinx, Altera[5][6] 등의 단일 칩 이기종 다중코어 SoC들은 ARM Cortex A시리즈의 코어와 Cortex M4 코어, DSP 코어 등을 하나로 결합하였다. TI AM5728의 경우 하나의 칩 내부에서 Cortex A15 코어 2개, DSP코어 2개, Cortex M4 2개, 듀얼코어 3D GPU 등에 리눅스, TI-RTOS, 안드로이드 등의 운영체제를 각각 수행하면서 별개의 작업들을 효율적으로 조율할 수 있도록 하였다.[5] 각각의 운영체제에서 필요한 프로세스를 생성하여 수행을 하고, 경우에 따라 프로세스 간에 그림 1처럼 버스를 통한 통신을 진행한다.

이기종 다중 코어 환경에서의 프로그램 개발은 개발 환경 설정이 까다로우며 번거로울 뿐만 아니라 다음과 같은 복잡한 문제들을 포함한다.[4]

1) 아키텍쳐 : 각 코어, GPU, 메모리, I/O 등의 자원과 관련하여 운영체제와 응용프로그램을 설치하는 여러 가지 옵션이 있다.

2) 환경 설정 : 환경 설정은 자동화하기 어렵고 시간을 많이 요구하며 초기 설정을 하였어도 요구사항의 변화 등으로 환경을 재설정해야 하는 경우도 있다.

3) 부팅 : 각 운영체제를 해당 코어와 관련 하드웨어에 부팅해야하며 여러 운영체제를 부팅 시 지켜야 할 순서가 있고 시스템의 요구 조건에 따라 하드웨어 공유 등을 고려해야 할 수도 있다.

4) 디버깅 : 시스템 통합 시 각 운영체제와 응용 프로그램이 작동하는 방식을 전체로 이해해야 하며, 자원 경쟁이 발생하는 부분과 프로세서나 버스 혹은 장치들의 포화 상태 도달 여부를 알아야 하고, 부분적인 작동이 미치는 부작용을 고려하면서 전체 시스템의 효율을 최적화해야한다.

5) 분리 : 시스템의 한 부분이 오동작하거나 잘못되었을 때 통합 시스템에 영향을 주지 않도록 해야 한다.

6) 장치 공유 : 제한된 기능의 하드웨어 장치는 공유되어야 하고 공유 시 그 기능이 영향을 받지 않아야 한다.

7) IPC : 통합된 시스템에서의 각 코어에서 수행되는 여러 응용프로그램이 수행 중에 서로 통신이 가능해야 한다.

8) 보안 : 하드웨어 장치들의 공유는 추가적으로 발생하는 보안 문제를 고려해야 한다.

다중코어 SoC 상에서 수행되는 프로그램에 문제가 발생하였을 경우 이를 확인하는 과정은 각각의 코어와 운영체제를 고려해야 하므로 디버깅 환경 역시 복잡해진다. 주 프로세서에서는 리눅스가 수행이 되고 실시간 코어에서는 실시간 운영체제가 수행이 되며 운영체제 없이 직접 수행하는 베어 메탈(Bare metal)도 통합될 수 있다. 이런 각각의 프로그램들을 따로 따로 컴파일하여 SD 카드에 운영체제와 함께 만들어 넣고 다중코어 시스템을 부팅하여 결과를 확인해야 한다. 프로그램에 문제가 있으면 이 과정을 다시 반복해야하는 어려움이 있다.

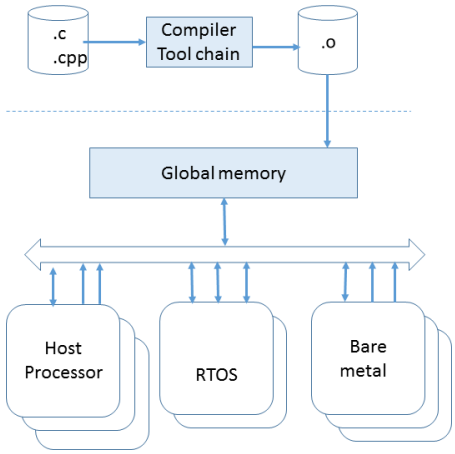

그림 1. 다중코어에서의 데이터 교환
Fig. 1. Data exchange of heterogenous multicore

2. RPMsg

이기종 다중코어 사이의 메시지 전달은 RPMsg를 사용한다. RPMsg는 이기종 다중 코어를 기반으로 하는 비대칭 다중처리(asymmetric multiprocessing) 시스템에서 서로 다른 프로세서 사이에 메시지를 전달하는 수단으로 Linux의 I/O 가상화에 사용하는 virtio를 기반으로 만들어진 가상 버스이다.[7]

RPMsg는 안드로이드 4.0(Ice Cream Sandwich)을 위하여 2011년 리눅스 커널 3.0에서부터 추가되었고 TI의 리눅스와 RTOS(SYS/BIOS)에서 라이브러리로 제공되고 있으며 공유메모리(shared memory)와 코어 간 인터럽트를 사용하여 구현된다. 현재 RPMsg는 OpenAMP의 버전[8], TI의 버전[9], NXP의 버전[10] 등이 있다.

3. RPMsg 구성

RPMsg는 물리적 계층, 매체 접근 계층, 전송 계층의 3개 계층으로 이루어져있다. 물리적 계층(Physical layer)은 공유메모리, 코어 간 인터럽트로 구성된다. 공유메모리에는 통신을 위한 vring과 메시지 버퍼를 구현한다. 코어 간 인터럽트는 데이터 전달 상황을 상대방 프로세서에게 알리기 위하여 상대 프로세서에게 발생시키는 인터럽트이며 옵션 형태로 구현된다.

매체접근계층(Media access layer)은 공유메모리 상에 구현된 vring 및 vqueue, 메시지 버퍼로 구성된다. 전송계층(Transport layer)은 채널과 엔드포인트로 구성된다. 그림 2는 이러한 3개 계층을 요약한 것 이다.[8]

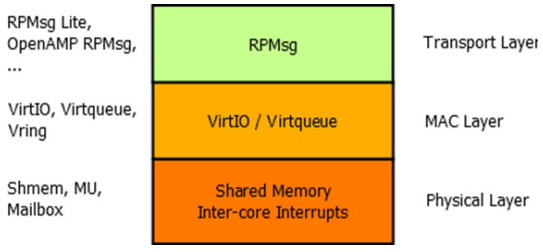

그림 2. RPMsg의 3계층 표현
Fig. 2. 3-layer of RPMsg

채널은 두 프로세서가 통신하기 위하여 각 프로세서에 생성하는 논리 장치이다. 채널의 종류에는 주 채널(master channel)과 원격채널(remote channel)이 있으며 하나의 채널은 양방향 통신을 위하여 TX용 vqueue와 RX용 vqueue를 사용한다. 주 채널의 TX 용 vqueue의 vring과 원격 채널의 RX 용 vqueue의 vring은 공유메모리 상의 같은 vring이다.

메시지를 보낼 때 채널 id와 그 채널의 엔드포인트 id를 상대방 주소로 사용한다. 각 엔드포인트는 그 엔드포인트로 전달된 메시지 버퍼들 중 응용 프로그램이 아직 가져가지 않은 메시지 버퍼들의 주소들을 저장하는 큐(queue)를 가지고 있다. 그림 3에 메시지 전달과정을 나타내었다.[11]

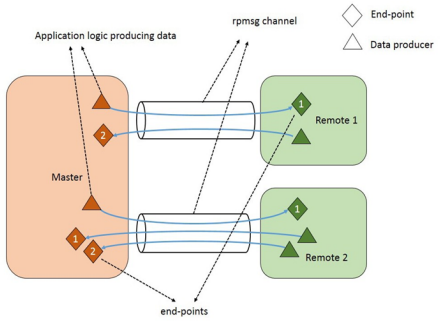

그림 3. RPMsg에서의 메시지 전달과정
Fig. 3. Message transfer using RPMsg

공유메모리 상의 vring은 단일 쓰기, 단일 읽기(single-writer single-reader) 특성을 가지는 환형 큐 자료 구조이다. 예를 들면, 프로세서 1이 vring_avail에 쓰기 가능하면 프로세서 2는 vring_avail에 읽기만 가능하다.

메시지 버퍼는 공유메모리 상에 저장되며 RPMsg를 통하여 보내거나 받는 메시지 자체를 저장한다.

Ⅲ. 이기종 다중코어 프로그램 개발 환경의 단순화

SMP환경의 단일 운영체제에서 RPMsg와 공유메모리를 활용하면 이기종 다중코어 간 데이터 교환을 프로세스 간 데이터 교환으로 단순화할 수 있다.

표 1에서 응용프로그램 개발을 위한 필요 요소들에 대해 이기종 다중코어 환경과 SMP 환경을 비교하였다. 이기종 코어의 기능은 동종의 코어에서도 수행이 가능하며 이기종 코어들이 사용하는 글로벌 메모리는 공유메모리로 매핑이 가능하다. 데이터 전달에 사용하는 버스도 매핑이 가능하고 프로세스 간 통신에 사용하는 IPC도 동일한 기능의 RPMsg를 사용하면 각 기능들을 SMP 환경에서도 대체 수행이 가능하다. 따라서 이기종 다중코어 환

경에서의 프로그램 수행을 SMP 환경에서도 실행할 수 있다. SMP환경에서 공유메모리를 활용하여 RPMsg를 구현하면 이기종 다중코어 환경을 매핑하여 시뮬레이션 할 수 있다. 주 프로세스에서 가상 버스 환경에 해당하는 RPMsg를 통해 원격 프로세스와 통신을 하는 것은, 이기종 다중코어 환경에서 서로 다른 프로세서가 통신하는 것과 동일한 효과를 가진다.

표 1. 이기종 다중코어 환경과 SMP 환경 매핑
Table 1. Mapping heterogeneous multi-core environment with SMP environment

이기종 다중코어 환경	SMP 환경
이기종 다중코어	다중코어
multi-process	multi-process
global memory	shared memory
다중 버스	가상 버스
AMP RPMsg	Linux RPMsg
multi-core application	여러 개의 single-core application

Ⅳ. 구현 및 실험 결과

본 논문에서는 x86환경에서 리눅스 운영체제를 기반으로 간략한 RPMsg를 구현하고 공유 메모리를 통해 메시지 교환을 수행한다. cortex-A에서의 프로세스는 리눅스에서의 프로세스 A_1, A_2,...로, cortex-M에서의 프로세스는 리눅스에서의 프로세스 B_1, B_2,...로 대치하여 실행이 된다. 서로 다른 코어 및 커널들 사이의 데이터 교환은 SMP환경에서 프로세스 간 데이터 교환으로 치환하여 진행한다. 이렇게 실험 및 디버깅 후 이기종 다중코어 환경으로 컴파일하고 실제 환경에서 실행하여 문제없이 수행이 잘 되었다. 그림 4는 이러한 환경을 요약하여 표현한 것이다.

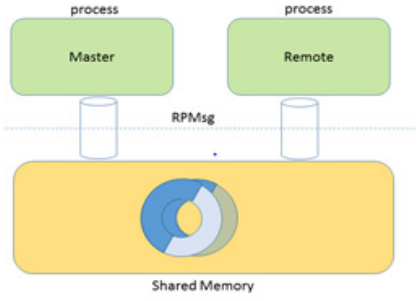

그림 4. SMP 환경에서 RPMsg를 적용
Fig. 4. RPMsg with SMP environment

1. vring 및 descriptor 구현

구조체 vring은 공유메모리 상에 구성되는 하나의 vring에 대한 정보를 표현하는 구조체이다. 구조체 vring 형을 가지는 변수 자체는 공유메모리 상에 있지 않고 해당 프로세서의 메모리에 있다.

표 2. 구조체 vring
Table 2. vring structure

필드	설명
uint64_t addr	공유메모리 상에 저장되는 메시지 버퍼의 시작 주소(guest physical address)
uint32_t len	메시지 버퍼의 크기(byte 수)
uint16_t flags	• bit 0: 1이면 아래 next 필드가 사용됨을 의미 • bit 1: 0이면 현재 버퍼가 읽기 전용이고, 1이면 현재 버퍼가 쓰기 전용임을 의미 • bit 2: 1이면 이 버퍼가 descriptor의 리스트를 가지고 있음을 의미 • bit 15 -3 : 사용하지 않음
uint16_t next	다음 미사용 버퍼 descriptor의 인덱스를 저장

구조체 vring_desc는 공유메모리 상에 저장되는 하나의 메시지 버퍼를 표현하는 descriptor를 저장하는 구조체이다.

구조체 vring_avail은 사용가능한 vring의 버퍼 링에 대한 정보를 저장하는 구조체이다.

구조체 vring_used는 사용한 vring의 버퍼 링에 대한 정보를 저장하는 구조체이다.

표 3. 구조체 vring_desc
Table 3. vring_desc structure

필드	설명
unsigned int num	공유메모리 상에 구성되는 vring의 최대 descriptor의 개수(default로 256)
struct vring_desc *desc	공유메모리 상에 구성되는 vring_desc 배열의 시작 주소
struct vring_avail *avail	공유메모리 상에 구성되는 vring_avail의 시작 주소
struct vring_used *used	공유메모리 상에 구성되는 vring_used의 시작 주소

2. 메시지 전달 과정

그림 5는 구현된 메시지 전달 과정 중 주 채널에서 원격 채널로 메시지를 전달하는 과정을 도식화 한 것이다.

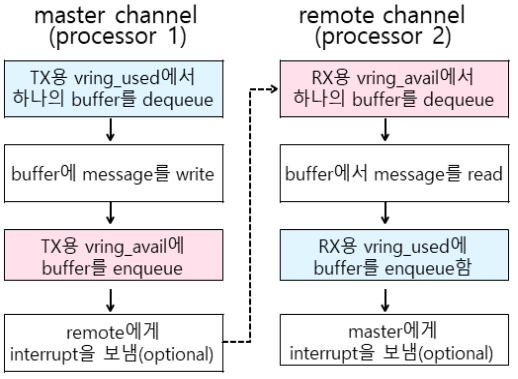

그림 5. 주 채널에서 원격 채널로의 메시지 전달
Fig. 5. Message transfer from master to remote

주 채널에서 원격 채널로의 메시지 전달 순서는 다음과 같다.

1) 주 채널에서 원격 채널로 메시지를 전달하기 위해 송신용 vring_used에서 버퍼를 하나 가져온다.
2) 가져온 버퍼에 메시지를 기록한다.
3) 송신용 vring_avail에 메시지가 기록된 버퍼를 큐에 넣는다.
4) 필요하면 원격 프로세서에게 인터럽트를 발생시킨다.
5) 원격 채널에서 수신용 vring_avail에서 버퍼를 하나 가져온다.
6) 가져온 버퍼에서 메시지를 읽는다.
7) 수신용 vring_used에 읽은 후의 버퍼를 큐에 넣는다.
8) 필요하면 원격 프로세서에게 인터럽트를 발생시킨다.

원격 채널에서 주 채널로의 메시지 전달 과정은 TX와 RX가 바뀌어서 역순으로 진행된다.

3. 구현 결과

이기종 다중코어 환경에서 응용 프로그램을 개발하는 과정은 그림 6처럼 각 코어에 맞추어 컴파일을 수행하고 타겟보드의 리눅스로 실시간 응용프로그램을 전송 후 실시간 운영체제를 가동하는 절차를 반복하면서 진행해야 한다. 테스트 및 디버깅 과정에서 하나의 콘솔로 메시지가 출력되는 경우에는 각 코어에서의 실행 결과가 섞여서 출력되기 때문에 더욱 번거롭다.

SMP환경에서는 그림 7처럼 각 절차를 단순화하여 충분히 테스트 및 디버깅한 후 타겟 보드에서 실제 환경 적용 테스트를 통해 결과를 검증한다.

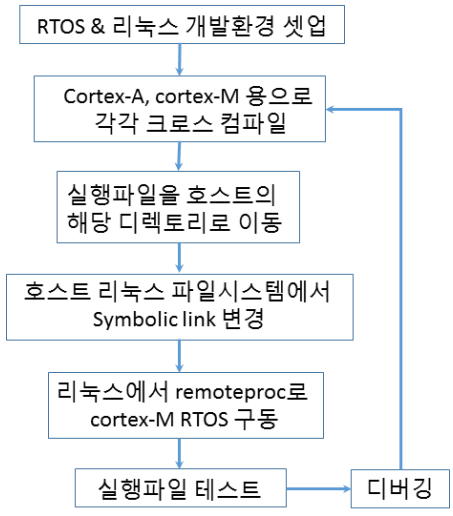

그림 6. 이기종 다중코어 응용프로그램 개발 절차
Fig. 6. Application programming steps for heterogeneous multicore system

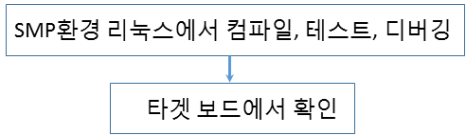

그림 7. SMP환경에서의 응용프로그램 개발 절차
Fig. 7. Application programming steps for SMP environment

구현 결과 테스트는 이기종 다중코어 프로그램 개발에 TI-AM5728 IDK 보드와 66AK2Gx 보드를 사용하고 리눅스는 Yocto/OpenEmbedded 와 Ubuntu ver. 16.04를 사용하였다. SMP 환경의 리눅스는 Intel core i7-6700 CPU 기반에서 Ubuntu ver. 16.04를 적용하였다.

구현된 시뮬레이션 환경에서 개발과 디버깅을 진행하고, 최종 결과를 실제의 실행환경에서 확인하여 별도의 컴파일, 포팅, 리부팅 등의 과정을 줄일 수 있어서 개발 효율을 높일 수 있음을 확인하였다.

실제 구현된 환경에서 주 프로세스와 원격 프로세스가 서로 데이터를 주고받는 모습을 그림 8, 9에서 보여주고 있다. 최종 결과를 TI-AM5728 IDK 개발 보드에서 cortex-A15(리눅스), cortex-M4(RTOS)에 각각 수행하여 문제없이 실행 가능하였으며 시뮬레이션 환경에서 개발과정의 대부분을 완성할 수 있었다.

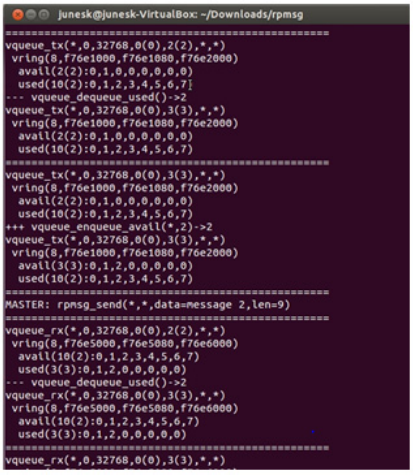

그림 8. 시뮬레이션 결과 1
Fig. 8. Results of simulation 1

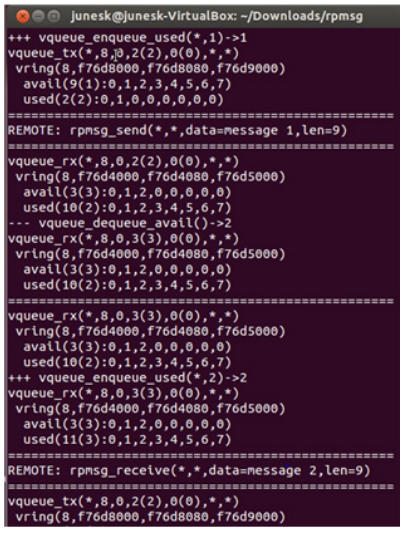

그림 9. 시뮬레이션 결과 2
Fig. 9. Results of simulation 2

A회사의 B프로젝트 중 C모듈의 경우 cortex-A15에서 리눅스 환경의 호스트 프로세스를 수행하고 7개의 실시간 프로세스를 cortex-M4에서 수행하였다. B프로젝트의 경우 중요 모듈 하나를 최종 개발하려면 평균 3~400회의 디버깅이 필요하였다. 이기종 다중코어 환경에서는 시간당 2회 디버깅이 가능했으며 유사한 복잡도를 가진 D모듈의 경우 본 논문에서 제안한 방법론을 적용하여 시간당 12회 이상의 디버깅이 가능하였다. 1일 평균 2시간씩 디버깅에 투입하였을 때 이기종 다중코어 환경의 경우 약 88일이 소요되었고 SMP 환경에서 개발 절차 단순화 방식을 적용하였을 때 약 15일이 소요되었다. 다만 SMP환경의 경우 이기종 다중코어 환경과 달리 실제 환경 적용 테스트가 필요하여 2일 정도 추가 소요 기간이 필요하였다. 표 4에 그 결과를 요약하였다.

표 4. 적용 결과
Table 4. Results of experiment

	이기종 다중코어 환경에서 개발	SMP 환경에서 개발
모듈명	C	D
코어사용	cortex-A15, cortex-M4	x86
프로세스 수	8	8
1시간당 디버깅횟수	2	12
총 디버깅 횟수	350	350
실적용 테스트	-	4
1일 2시간 투입 시 총 개발일수	88	17

이러한 실험 과정을 통하여 개발 기간을 줄일 수 있음을 확인하였으나 명령어 수행시간, 소프트웨어 실행 싸이클 횟수 계산, 메시지 전달 시간, 에너지 사용량 등 세부적인 성능평가를 위해서는 실제 개발 환경에서 수행을 해야만 가능하다. 또한 반드시 32비트 환경으로 컴파일을 수행해야 이기종 다중코어 환경과 SMP환경 사이의 호환성을 유지할 수 있다.

V. 결 론

임베디드 시스템 개발에서 더욱 다양해지는 고도의 요구 사항들을 만족시키기 위한 이기종 다중 코어 환경은, 서로 다른 응용들을 하나로 묶어서 효율적으로 운용 가능하게 한다. 여러 코어와 서로 다른 운영체제를 하나의 칩에 통합, 운용하여 개발과 디버깅이 복잡해지고 많은 시간이 필요하다. 본 논문에서는 가상 버스인 virtio를 기반으로 한 RPMsg 환경을 응용하여 서로 다른 이기종 다중코어 실행 환경을 SMP 실행 환경에서 시뮬레이션 가능함을 보이고, 각 기능을 추상화하고 대체하여 개발 절차를 단순화할 수 있었다. 이러한 절차 단순화를 통해 이기종 다중코어 환경에서 응용프로그램 개발 단계를 대폭 축소하여 개발 기간을 단축하는 방법을 제시하고 실험하여 결과를 확인하였다. 이러한 개발 방법론을 적용

한 결과 1/4정도의 시간으로 개발기간을 줄일 수 있었다.

추후 시뮬레이션 환경에서 실행 싸이클 횟수를 계산하여 개발과정에 반영함으로써 전체 효율을 높이는 연구를 추가적으로 수행하여 보다 유용한 방법론으로 발전시킬 수 있기를 기대한다.

References

[1] Bryon Moyer, "Real World Multicore Embedded Systems", Elsevier, pp.33-73, 2013.

[2] Jongbok Lee, "A Performance Study of Asymmetric Embedded Multi-Core Processors", The Journal of The Institute of Internet, Broadcasting and Communication, VOL. 16 NO. 1, pp.233-238, February 2016.
DOI: https://doi.org/10.7236/JIIBC.2016.16.1.233

[3] Felix Baum, Arvind Raghuraman. "Making Full use of Emerging ARM-based Heterogeneous Multicore SoCs", 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016.

[4] Warren Kurisu, "Addressing Design Challenges in Heterogeneous Multicore Embedded Systems" Mentor Graphics Corporation, 2015

[5] Texas Instruments, "AM572x Technical Reference Manual",,,Oct 2014.

[6] Freescale Semiconductor Inc., "i.MX 8 Family of Applications Processors",
https://www.nxp.com/docs/en/fact-sheet/IMX8FAMFS.pdf,,,Oct 2016.

[7] Rusty Russell, "virtio: towards a de-facto standard for virtual I/O devices", ACM SIGOPS Operating Systems Review, Vol. 42 Issue 5, pp.95-103, July 2008.

[8] The Multicore Association® (MCA) Open Asymmetric Multi Processing (OpenAMP), "OpenAMP source" ,, https://github.com/OpenAMP/open-amp,,,2016.

[9] OASIS consortium, "Virtual I/O Device (VIRTIO) Version 1.0, Committee Specification Draft 01", http://docs.oasis-open.org/virtio/virtio/v1.0/csd01/virtio-v1.0-csd01.html#x1-260001,, Dec 2013.

[10] Freescale Semiconductor Inc., "NXP RPMsg-lite" , https://github.com/codeauroraforum/rpmsg-lite,,, 2016.

[11] Arvind Raghuraman, "Toward Easier Software Development for Asymmetric Multiprocessing Systems", Xcell Journal issue 93, pp.58-65, Oct. 2015.

저자 소개

사 공 준(정회원)

• June SaKong received the B.S. and M.S. degree in Computer Engineering from Kyung-pook National University in 1983 and 1995 respectively. His research interests include embedded computing, HMP computer architecture and sustainable energy system.

신 동 하(정회원)

• Dongha Shin received the B.S. degree in Computer Engineering from Kyung-pook National University in 1980, the M.S. degree in Computer Engineering from Seoul National University in 1982 and the Ph.D. degree in Computer Science from University of South Carolina in 1994. During 1982-1996, he worked in ETRI as a technical staff to study expert systems, word processing systems, file systems and language processing systems. During 1997-current, he is a professor of Computer Science and Electronics Department in Sangmyung University. His research interests include real-time kernels, ARM-based computer architecture, inter processor communication, dynamic software updating, functional and logic languages and compiler construction.