MPSoC 플랫폼의 버스 에너지 절감을 위한 버스 분할 기법
(Bus Splitting Techniques for MPSoC to Reduce Bus Energy)

정준목† 김진효‡ 김지홍†
(Chun-Mok Chung) (Jinhyo Kim) (Jihong Kim)

† 서울대학교 컴퓨터공학부
School of CSE, Seoul Nat'l Univ.

‡ 삼성전자
Samsung Electronics Co.

요약 버스 분할 기법은 통신이 많은 모듈들을 가까이 배치하고 필요한 버스 단편만 사용함으로 버스 에너지 소비를 줄인다. 그러나, MPSoC와 같은 다중 프로세서 플랫폼에서는 캐쉬 일관성을 유지하기 위하여 모든 프로세서에서 버스 트랜잭션을 알아야 하므로, 기존의 버스 분할 기법을 적용할 수 없다. 본 논문에서는 공유 메모리 기반의 MPSoC 플랫폼에서 버스 에너지를 절감시키기 위한 버스 분할 기법을 제안한다. 제안된 버스 분할 기법은 비 공유 메모리와 공유 메모리의 버스를 분할함으로써, 캐쉬 일관성을 유지하며 비 공유 메모리를 참조할 때 소비하는 버스 에너지를 최소화시킨다. 또한, 태스크 별 버스 트랜잭션 횟수를 기반하여 태스크를 할당함으로써, 공유 메모리를 참조할 때 소비하는 버스 에너지를 절감시키는 캐쉬 일관성을 고려한 태스크 할당 기법을 제안한다. 시뮬레이션을 통한 실험에서 제안된 버스 분할 기법은 비 공유 메모리 참조시의 버스 에너지를 최대 83%까지 절감시키며, 태스크 할당 알고리즘은 공유 메모리 참조시의 버스 에너지를 최대 36%까지 절감시키는 효과가 있음을 보여준다. 그럼으로, 다중 프로세서 시스템에서도 버스 분할 기법을 적용하여 버스 에너지 절감 효과를 볼 수 있으며, 캐쉬 일관성을 고려한 태스크 할당 기법을 통해 추가적으로 버스 에너지를 절감할 수 있음을 보여준다.

Abstract Bus splitting technique reduces bus energy by placing modules with frequent communications closely and using necessary bus segments in communications. But, previous bus splitting techniques can not be used in MPSoC platform, because it uses cache coherency protocol and all processors should be able to see the bus transactions. In this paper, we propose a bus splitting technique for MPSoC platform to reduce bus energy. The proposed technique divides a bus into several bus segments, some for private memory and others for shared memory. So, it minimizes the bus energy consumed in private memory accesses without producing cache coherency problem. We also propose a task allocation technique considering cache coherency protocol. It allocates tasks into processors according to the numbers of bus transactions and cache coherence protocol, and reduces the bus energy consumption during shared memory references. The experimental results from simulations say the bus splitting technique reduces maximal 83% of the bus energy consumption by private memory accesses. Also they show the task allocation technique reduces maximal 30% of bus energy consumed in shared memory references. We can expect the bus splitting technique and the task allocation technique can be used in multiprocessor platforms to reduce bus energy without interference with cache coherency protocol.

1. 서론

멀티미디어 등의 응용 프로그램의 복잡성 증가와 하나의 기기에서 복합적인 기능을 지원하는 요구사항 등으로 인하여 임베디드 시스템은 기존의 한 가지 기능만을 제공하는 수준에서 복합적인 기능의 여러 가지 프로그램을 동시에 수행시키는 방향으로 발전하고 있다. 다행히도 반도체 공정 기술의 발전으로 집적률이 증가하여 하나의 칩 안에 다수의 프로세서와 메모리가 집적되는 MPSoC (Multiprocessor System-on-Chip)가 개발되었고, 고성능 모바일 시스템 등에서 적용되는 사례가 확산되고 있다 [1]. 이러한 모바일 시스템 (예를 들면, 핸드폰, PDA, 휴대용 게임기, PMP 등)의 대부분은 배터리를 기반으로 하기 때문에, 저전력 소비는 시스템 설계의 중요한 고려 요소가 된다. 왜냐하면, 이들의 배터리 용량은 한정되어 있으므로, 에너지 소비가 시스템의 동작 시간에 직접적으로 영향을 주기 때문이다. 더구나, MPSoC와 같은 고집적, 고성능 플랫폼은 소비 에너지가 증가하므로, 시스템 설계에서 저전력 설계의 중요성은 더욱 증가하고 있다.

공유 버스는 간단한 망 구조, 낮은 비용 등의 장점으로 인해 SoC 플랫폼에서 통신 네트워크로 널리 사용되는 방식이다 [2]. 온 칩 버스에서 소비되는 전력은 프로세서에서 소비되는 전력의 37%에 달하며, 저전력 반도체 기술이 발전함에 따라 그 비중은 더욱 높아질 것이다. 따라서, 버스 에너지를 줄이는 연구가 중요성을 가지게 되었고, 분할 버스는 이러한 버스 에너지를 줄이기 위한 제안된 기법이다 [3],[4]. 분할 버스는 여러 개의 컴포넌트들이 연결되어 있는

버스를 분할자(splitter)를 이용하여 여러 개의 버스 단편(bus segment)들로 나누어 놓은 구조로 되어 있다. 통신이 많은 모듈들을 가까이 배치하고 모듈의 통신을 수행할 때 전체 버스를 모두 사용하지 않고 컴포넌트들이 연결되어 있는 버스 단편들에만 신호가 전송되도록 분할자를 제어하여 통신시에 필요한 버스 단편들만 사용함으로 버스 에너지 소비를 줄인다.

그러나, 기존의 연구들은 단일 프로세서서 기반의 플랫폼에 버스 분할을 적용하는 방법을 제안하며, 캐쉬 일관성 문제가 발생하는 경우를 고려하지 않아, MPSoC에 적용하기에 문제점을 가진다. 일반적으로 MPSoC의 각 프로세서는 성능과 에너지상의 이득을 위해 캐쉬를 포함하고 있고, 캐쉬 일관성 문제를 해결하기 위하여 스누피 프로토콜과 같은 캐쉬 일관성 프로토콜을 사용하고 있다. 캐쉬 일관성을 유지하기 위해서는 모든 프로세서에서 버스 트랜잭션의 내용을 알아야 하므로, 캐쉬 일관성 문제가 발생하는 공유 데이터에 대한 버스 트랜잭션에서는 버스 분할을 적용할 수 없다.

본 논문에서는 캐쉬 일관성 프로토콜을 사용하는 MPSoC 플랫폼에 버스 분할 기법을 적용하여 버스 에너지를 절감시키는 기법을 제안한다. 메모리 참조 형태에 따라 버스 분할자를 제어하여 캐쉬 일관성 프로토콜과 버스 분할 기법간의 충돌을 해결한다. 그럼으로써, 캐쉬 일관성 문제가 없는 버스 트랜잭션의 소비 에너지를 절감시킨다. 또한, 캐쉬 일관성 문제가 발생하는 버스 트랜잭션의 에너지를 최소화시키기 위하여 캐쉬 일관성 프로토콜을 고려한 태스크 할당 기법을 제안한다. 시뮬레이션을 통한 실험을 통해 제안된 기법들이 버스 에너지를 감소시킴을 보여준다.

이후의 논문 구성은 다음과 같다. 2장에서는 관련된 연구들을 살펴보고, 이들을 MPSoC에 적용하기 어려운 문제점을 명시한다. 3장에서는 타겟 MPSoC 환경을 명시하고, 4장에서는 본 논문에서 제안하는 버스 에너지 절감 기법들을 자세히 설명한다. 5장에서는 실험을 통해 제안된 기법들의 버스 에너지 절감 효과를 평가하고, 끝으로 6장에서 결론을 맺는다.

2. 관련 연구

버스를 여러 개의 버스 단편으로 분할시켜 에너지 소비를 줄이는 버스 분할 기법은 Chen 에 의해 처음으로 제안되었다 [3]. Chen은 패스 트랜지스터로 버스를 여러 개의 버스 단편으로 나누는 방식을 제안하였다. 그리고, 그래프 알고리즘을 사용하여 통신량이 많은 장치들을 인접한 버스 단편에 위치시켜 버스 에너지 소비를 최소한으로 줄이는 최적의 버스 위상을 찾았다. Hsieh는 분할 버스 구조에서 최소 에너지 소비를 위해 버스에 연결된 여러 개의 모듈들을 어떻게 두 부분으로 분할할 것인가를 확률적인 모델을 기반으로 하는 방안을 제시하였으며 [4], 실험을 통해 공유 버스와 비교해 분할 버스가 16%~50% 정도의 에너지 절약 효과를 가져온다고 주장하고 있다. 그러나 저자들은 하나의 분할자를 사용하는 구조만을 고려했으며, 분할자를 제어하는 동작 모델을 설명하지 않고 있다. 최근에는 다중 동시 버스 접근이 가능한 분할 버스 구조가 Lu에 의해 제안되었다 [5]. 이 구조에서는 다중 동시 접근이 가능하도록 중재자에 새로운 회로를 추가하였다. 그러나, 이 방식에서는 다중 동시 접근을 통한 버스의 성능 향상에 초점이 맞추어져 있다. Lahiri는 실제 AMBA 버스의 사양을 시뮬레이터로 구현하여, 여러 가지 저전력 버스 기법들의 효율성을 실험을 통해 제시하였다 [6]. 실험 결과는 버스 분할 기법을 통해 전체 버스 시스템에서 16%의 에너지 절감효과가 있고, 버스 라인에 대해서는 70% 가량의 에너지 절감효과가 있는 것을 보여주고 있다.

그러나, 기존의 연구들은 단일 프로세서 환경만을 고려하여, 캐쉬 일관성 프로토콜을 사용하는 MPSoC에 버스 분할 기법을 적용하기 위해서는 캐쉬 일관성 프로토콜과 버스 분할 기법간의 충돌 문제를 해결해야 하며, 본 논문은 이러한 충돌 문제를 해결하여 MPSoC에서 버스 분할 기법을 적용하는 방법을 제안한다. 본 논문의 기여 부분은 다음과 같다. 공유 메모리 기반의 MPSoC의 메모리 참조 특성을 고려하여, 버스 분할 기법을 통한 버스 에너지 절감 방법을 제안한다. 메모리 참조 형태에 따른 버스 분할자의 제어 방식을 통해 버스 분할 구조와 캐쉬 일관성 프로토콜의 충돌 문제를 해결함으로써, 캐쉬 일관성 프로토콜을 사용하는 MPSoC 환경에서 버스 분할 기법을 적용하여 버스 에너지 절감 효과를 제공한다. 또한, 캐쉬 일관성 프로토콜을 고려하여 캐쉬 일관성 문제가 발생하는 버스 트랜잭션의 소비 에너지를 최소화시키는 태스크 할당 기법을 제안한다. 태스크 할당 기법은 캐쉬 일관성을 고려한 태스크 별 버스 트랜잭션 횟수를 분석하고, 이를 기반하여 버스 트랜잭션이 많은 태스크를 공유 메모에 가까운 프로세서에 할당함으로써, 프로세서가 공유 메모리를 참조할 때 소비하는 버스 에너지를 절감한다.

3. 타겟 MPSoC 플랫폼의 구조

본 절에서는 본 논문에서 기본 모델로 가정하는 타겟 MPSoC 플랫폼의 구조를 기술한다. 다음절에서 설명되는 버스 에너지 절감 기법들은 타겟 MPSoC와 같은 구조를 가지는 MPSoC 플랫폼에서의 버스 에너지 절감을 위하여 제안된다.

그림 1은 타겟 플랫폼인 공유 메모리 기반 MPSoC의 구조를 보여준다 [7]. 타겟 MPSoC는 다수의 프로세서, 메모리 모듈, 중재자로 구성되며 (MPSoC에는 이외에도 외부 장치 제어기, 인터럽트 제어기 등의 추가적인 모듈이 존재하나, 표현의 단순화를 위하여 메모리 참조와 관련된 모듈만 표시하였다), 모듈들은 하나의 공유 버스로 연결되어 있다. 프로세서로는 ARM, MIPS 등의 범용 프로세서나 DSP, HW 가속기 등과 같은 특수 목적용 프로세서가 사용될 수 있으나, 본 논문에서는 모든 프로세서가 동일한 형태의 프로세서라고 가정한다. 각 프로세서는 캐쉬를 포함하고 있으며, 캐쉬 일관성 유지를 위하여 라이트-스루 무효화 (write-through invalidation) 방식의 스누피 기반 프로토콜 [8]을 사용한다. 메모리는 하나의 프로세서에서 독점적으로 사용되는 비 공유 메모리와 프로세서간 통신에 사용되는 공유 메모리로 구분된다.

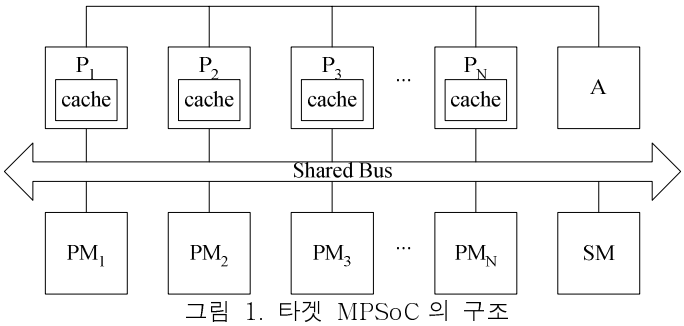

그림 1. 타겟 MPSoC 의 구조

그림에서 P_1~P_N은 프로세서, PM_1~PM_N은 프로세서의 비

공유 메모리(private memory), SM은 공유 메모리(shared memory), A는 중재자(arbitor)를 나타낸다.

4. MPSoC에서의 버스 에너지 절감 기법들

단일의 공유 버스 구조에서는 프로세서가 참조하는 메모리가 비 공유 메모리거나 공유 메모리에 상관없이 메모리를 참조할 때마다 버스의 모든 부분이 사용되고, 전체 버스에 해당하는 에너지를 소비하는 단점을 가진다. 본 절에서는 버스 에너지를 절감시키기 위한 두 가지 기법들을 제안한다. 하나는 프로세서의 메모리 참조 형태를 고려한 버스 분할 기법이며, 다른 하나는 태스크의 공유 메모리 참조 특성과 캐쉬 일관성을 함께 고려한 태스크 할당 기법이다.

4.1 버스 분할을 통한 버스 에너지 절감 기법

타겟 MPSoC와 같은 공유 메모리 환경에서는 비 공유 메모리 영역과 공유 메모리 영역이 명확하게 구분되므로, 프로세서에서는 참조하려는 주소값 분석을 통해 쉽게 비 공유 메모리 참조와 공유 메모리 참조를 구분할 수 있다. 예를 들어, 32비트 어드레스 버스를 사용하는 환경에서 각 프로세서의 비 공유 메모리가 0x00000000 ~ 0x7FFFFFFF, 모든 프로세서의 공유 메모리가 0x80000000 ~ 0xFFFFFFFF로 매핑된다면, 어드레스 버스의 최상위 1비트만을 분석하면, 비 공유 메모리 참조인지, 공유 메모리 참조인지를 구분할 수 있다. 비 공유 메모리의 데이터는 하나의 프로세서에서만 사용되므로, 다른 프로세서와 캐쉬 일관성 문제가 발생하지 않으므로, 비 공유 메모리와 공유 메모리를 서로 다른 버스로 분할하면, 프로세서가 비 공유 메모리를 참조하려고 할 때는 프로세서와 비 공유 메모리가 속한 버스 단편만을 동작하도록 분할자의 동작을 제어하여 버스 에너지를 절감시킬 수 있으며, 캐쉬 일관성과의 충돌 문제도 발생하지 않는다.

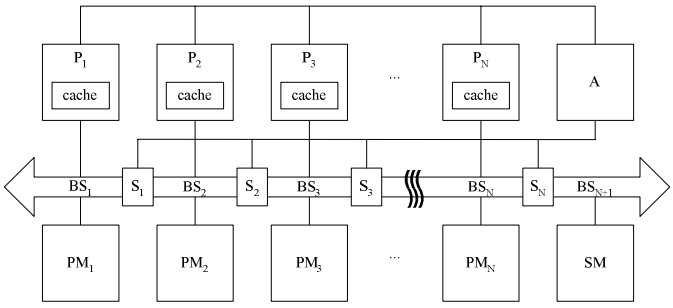

그림 2. 버스 분할 기법을 적용한 MPSoC 의 구조

그림 2는 이러한 구조적 특징을 바탕으로 N개의 프로세서로 구성된 MPSoC에 버스 분할 기법을 적용하여 비 공유 메모리와 공유 메모리를 서로 다른 버스로 분할한 구조를 보여준다. 그림에서 S_1~S_N은 버스 분할자(splitter)를, BS_1~BS_{N+1}은 버스 단편(bus segment)을 나타낸다. 버스는 분할자들로 구분된 여러 개의 버스 단편들로 이루어진다. 하나의 프로세서와 해당 프로세서의 비 공유 메모리는 동일한 버스 단편에 속해있으며, 하나의 버스 단편에는 프로세서와 비 공유 메모리가 짝을 이루어 연결되거나, 공유 메모리가 연결되어 있다. 분할자는 이웃한 두 개의 버스 단편을 연결하며, 두 이웃한 버스 단편간에 신호가 전달되거나, 전달되지 않도록 한다. 예를 들면, P_3이 PM_3을 참조할 때는 BS_3만을 사용하도록 S_2와 S_3를 제어하여 버스 에너지를 절감시킨다.

프로세서의 메모리 참조 형태에 따라 버스 분할자를 제어하는 역할은 중재자가 담당한다. 중재자는 프로세서로부터 메모리 참조를 요청하는 정보를 받아 분할자들의 동작을 제어한다. 그림 3은 프로세서, 중재자, 분할자의 연결 관계를 자세히 보여준다. 캐쉬에서 미스가 발생하여 메모리 참조가 필요하면, 프로세서는 중재자에 요청 신호(Request)를 보내 버스 사용을 요청한다. 요청 신호와 동시에 프로세서는 비 공유/공유 신호(Private/Shared)와 읽기/쓰기 신호(Read/Write)를 중재자에게 보낸다. 비 공유/공유 신호가 0이면 비 공유 메모리를, 1이면 공유 메모리를 의미하며, 읽기/쓰기 신호가 0이면 읽기를, 1이면 쓰기를 의미한다. 중재자는 현재 버스가 사용 중이 아니면, 프로세서에 허가 신호(Grant)를 보내서 버스 사용을 허가하고, 동시에 분할자들에게 제어 신호(Control)를 보내어 분할자가 버스를 분할하거나 연결하도록 한다. 중재자로부터 허가 신호를 받은 프로세서는 버스를 사용하여, 메모리 참조를 시작한다.

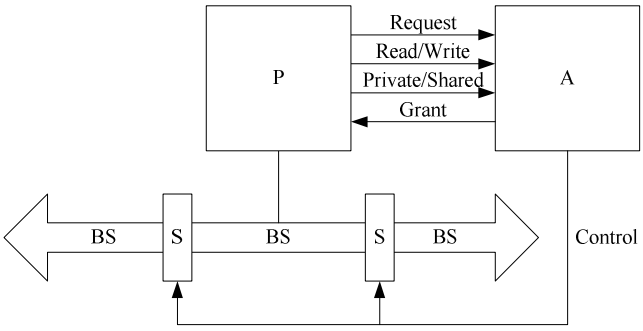

그림 3. 중재자의 분할자 제어 방식

4.2 분할 버스의 에너지 모델

공유 버스에서 소비되는 평균 에너지에 대한 식은 다음이 나타낼 수 있다 [9].

$$E_{MBUS} = 0.5 \times C_{bus} \times V_{dd}^2 \quad (1)$$

수식에서 C_{bus}는 버스 정전 용량(capacitance)을, V_{dd}는 버스의 전압을 나타낸다. 버스 정전 용량은 버스의 길이에 비례하므로, 공유 버스가 $N+1$개의 버스 단편들로 나누어지고, 모든 버스 단편의 길이가 동일하다고 가정하면, 하나의 버스 단편이 소비하는 에너지는 다음과 같이 나타낼 수 있다.

$$E_b = E_{MBUS} / (N+1)$$

분할 버스는 버스 단편들을 연결하는 분할자들을 포함하므로, 분할 버스의 소비 에너지는 아래와 같이 버스 단편들의 에너지와 분할자들의 에너지의 합으로 나타낼 수 있다.

$$E_{SBUS} = E_b \times D + E_s \times S \quad (2)$$

수식에서 D는 버스 트랜잭션에 사용된 버스 단편의 개수를, E_s는 분할자 하나가 소비하는 에너지를, S는 버스 트랜잭션에 사용된 분할자의 개수를 나타낸다. 분할 버스를 사용하는 MPSoC에서 N개의 프로세서가 동작하며 소비하는 버스의 에너지는 다음과 같이 나타낼 수 있다.

$$E_{SBUS,N} = \sum_{i=1}^{N}\sum_{j=1}^{Ki}(E_b \times D_{i,j} + E_s \times S_{i,j}) \quad (3)$$

수식에서 K_i는 프로세서 P_i의 버스 트랜잭션 횟수를, $D_{i,j}$는 P_i의 j번째 버스 트랜잭션에 사용된 버스 단편의 개수를, $S_{i,j}$는 분할자의 개수를 나타낸다. 수식(3)을 메모리 참조 형태에 따른 구분으로 변경하면, 버스 에너지는 아래와 같이 비 공유 메모리 참조시 소모 에너지와 공유 메모리 참조시 소모에너지로 구분할 수 있다.

$$E_{SBUB,N} = E_{PM} + E_{SMR} + E_{SMW}$$

수식에서 E_{PM}는 비 공유 메모리 참조에 의한 에너지, E_{SMR}은 공유 메모리 읽기에 의한 에너지, E_{SMW}은 공유 메모리 쓰기에 의한 에너지이다. 비 공유 메모리 읽기와 비 공유 메모리 쓰기에는 동일한 크기의 버스 단편만을 사용하므로 구분하지 않고 통일하여 표시할 수 있다. 비 공유 메모리를 참조할 때는 하나의 버스 단편과 하나 또는 두 개의 분할자가 사용되므로, 이를 수식(3)에 적용하면, 아래와 같이 나타낼 수 있다.

$$E_{PM} = \sum_{i=1}^{N}\{K_{i,pm} \times (E_b + E_s \times S_i)\} \; where, \; S_i = \begin{cases} 1 & if\ i = 1 \\ 2 & if\ 2 \le i \le N \end{cases}$$

수식에서 $K_{i,pm}$은 프로세서 P_i의 비 공유 메모리 참조를 위한 버스 트랜잭션 횟수를, S_i는 분할자의 개수를 나타낸다. 공유 메모리 읽기 시에는 필요한 버스 단편들과 분할자들만 사용된다. 이를 수식(3)에 적용하면, 공유 메모리 읽기시의 버스 에너지는 아래와 같이 나타낼 수 있다.

$$E_{SMR} = \sum_{i=1}^{N}\{K_{i,smr} \times (E_b \times D_i + E_s \times S_i)\}$$

수식에서 $K_{i,smr}$는 프로세서 P_i의 공유 메모리 읽기를 위한 버스 트랜잭션 횟수를 나타낸다. 공유 메모리 쓰기 시에는 위에서 설명했듯이 캐쉬 일관성을 위하여 모든 버스 단편들이 사용해야 하므로, $N+1$개의 버스 단편과 N개의 분할자가 모두 동작한다. 이들 버스 단편 수와 분할자수를 수식(3)에 적용하면, 공유 메모리 쓰기시의 버스 에너지는 다음과 같이 나타낼 수 있다.

$$E_{SMW} = \sum_{i=1}^{N}[K_{i,smw} \times \{E_b \times (N+1) + E_s \times N\}]$$

수식에서 $K_{i,smw}$는 프로세서 P_i의 공유 메모리 쓰기를 위한 버스 트랜잭션 횟수를 나타낸다.

4.3 캐쉬 일관성을 고려한 태스크 할당 기법

분할된 버스에서는 버스 에너지가 버스 단편의 수와 분할자의 수에 비례하므로, 공유 메모리에 가까운 프로세서일수록 공유 메모리를 한 번 참조하는데 소비되는 버스 에너지가 작다. 따라서, 여러 개의 태스크를 동시에 수행할 때는 공유 메모리 참조가 많은 태스크를 공유 메모리에 가까운 프로세서에 할당할 때 공유 메모리 참조에 의한 버스 에너지가 최소화된다. 그러나, 이는 캐쉬 일관성 프로토콜로 인한 버스 트랜잭션을 고려하지 않았을 때만 유효하다. 라이트-스루 무효화 프로토콜을 사용할 때는 캐쉬 일관성을 위하여 공유 메모리 쓰기를 위한 버스 트랜잭션에서는 모든 프로세서가 쓰여지는 메모리의 주소를 알게 하기 위하여, 모든 버스 단편들을 사용해야 하기 때문이다. 본 절에서는 이러한 캐쉬 일관성 프로토콜을 고려하여 분할된 버스에서 공유 메모리 참조에 의한 버스 에너지를 최소화 시키는 태스크 할당 기법을 제안한다.

프로세서는 공유 메모리를 참조하기 위하여 어드레스 버스와 데이터 버스를 각각 한번씩 사용하므로, 프로세서에서 수행되는 태스크의 버스 사용 횟수는 공유 메모리 참조 횟수의 두 배가 된다. 그러나, 라이트-스루 무효화 프로토콜을 사용할 때는 캐쉬 일관성을 위하여 공유 메모리 쓰기를 위한 버스 트랜잭션에서는 어드레스 버스의 모든 버스 단편을 사용하여야 하므로, 태스크가 어느 프로세서에 할당되더라고 전체 버스 에너지에 영향을 주지 않는다. 따라서, 캐쉬 일관성을 고려했을 때 태스크 T_i의 공유 메모리 참조를 위한 버스 트랜잭션 횟수 중에서 실질적으로 버스 에너지에 영향을 미치는 버스 트랜잭션 횟수를 나타내는 항목인 유효 버스 트랜잭션 횟수 M_i는 다음과 같이 나타낼 수 있다.

$$M_i = 2K_{i,smr} + K_{i,smw}$$

그림 4는 프로세서 개수와 태스크 개수가 동일한 경우, 캐쉬 일관성을 고려한 태스크 할당 집합을 구하는 알고리즘을 보여준다. 알고리즘의 동작 과정을 보면, 우선, 모든 태스크에 대하여 공유 메모리 참조를 위한 버스 트랜잭션 횟수를 이용하여 유효 버스 트랜잭션 횟수 M_i를 계산한다. 또한, 모든 프로세서에 대해 공유 메모리와 프로세서 사이의 버스 단편의 개수 D_i를 구한다. M_i가 큰 순서대로 태스크들을 정렬하고, D_i가 작은 순서대로 프로세서들을 정렬한 후, 태스크 집합의 태스크 순서대로 프로세서 집합의 프로세서에 할당함으로, 태스크 할당 집합을 생성한다. 알고리즘의 소요시간은 태스크 집합과 프로세서 집합의 정렬 시간에 비례하므로, N개의 태스크에 대한 태스크 할당 알고리즘의 수행 시간은 $O(NlnN)$으로 나타낼 수 있다.

```
입력 ;   T: 태스크 집합, P: 프로세서 집합,
         N: 태스크 개수(프로세서 개수)
출력 ;   A: 태스크 할당 집합
정의 ;   Ti ∈ T, Pi ∈ P
 1:  for i := 1 to N
 2:        Ki,smr := Ti의 공유 메모리 읽기용 버스 트랜잭션 횟수
 3:        Ki,smw := Ti의 공유 메모리 쓰기용 버스 트랜잭션 횟수
 4:        Mi := 2 × Ki,smr + Ki,smw
 5:        Di := Pi에서 공유 메모리까지의 버스 분할 개수
 6:  end for
 7:  T := { Ti | Mi가 감소하는 순서로 정렬, 1 ≤ i ≤ N }
 8:  P := { Pi | Di가 증가하는 순서로 정렬, 1 ≤ i ≤ N }
 9:  A := {}
10:  for i := 1 to N
11:        A := A ∪ {(Ti, Pi)}
12:  end for
```

그림 4. 캐쉬 일관성을 고려한 태스크 할당 알고리즘

그림 5는 버스 분할 기법을 적용한 타겟 MPSoC에서 캐쉬 일관성을 고려하지 않았을 때와 이를 고려하였을 때, 버스 에너지 최적화를 위하여 태스크를 할당한 예제를 보여준다. 그림에서 하나의 버스 단편에서 소비되는 에너지는 1E, 분할자의 에너지는 0으로 가정하고 버스 에너지를 계산하면, 기존의 방식대로 공유 메모리 참조 횟수만을 고려하여 태스크를 할당하였을 경우 버스의 에너지는 357E이나, 캐쉬 일관성을 고려하여 태스크를 할당하였을 때는 338E로 기존의 방식보다 5%정도의 버스 에너지 절감 효과가 있음을 알 수 있다.

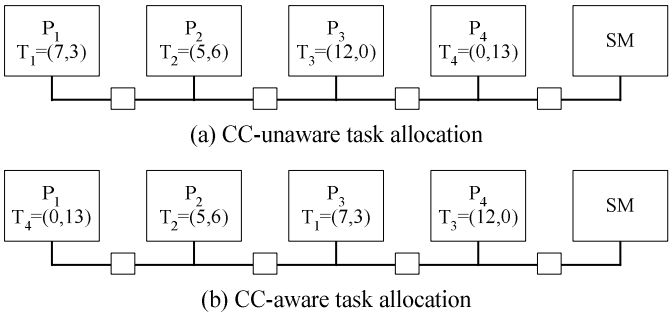

그림 5. 캐쉬 일관성을 고려한 태스크 할당 예제

5. 성능 평가

본 절에서는 시뮬레이션을 통해 제안한 버스 분할 기법과 캐쉬 일관성을 고려한 태스크 할당 기법의 버스 에너지 절감 효과를 평가한다.

5.1 실험 환경

MPSoC 플랫폼의 버스 소모를 측정하기 위하여 시뮬레이션 기반의 버스 에너지 측정 환경을 구축하였다. 그림 6은 실험에 사용된 시뮬레이션 환경을 보여준다. MPSoC 시뮬레이터는 실행할 태스크 집합과 프로세서의 개수를 입력 받아, 태스크를 각 프로세서에 할당하고 실행한다. 실험에서는 예제의 태스크 수에 일치하도록 시뮬레이터의 프로세서 수를 입력하였다. MPSoC 시뮬레이터는 실행 후, 각 태스크 별 버스 트랜잭션 정보를 출력한다. 이 정보는 비 공유 메모리와 공유 메모리에 대하여 읽기, 쓰기를 위한 각각의 버스 트랜잭션 횟수로 구성된다. 태스크 할당기는 이전에 기술한 알고리즘에 따라 캐쉬 일관성을 고려한 태스크 할당을 수행하여, 태스크 할당 정보를 버스 에너지 측정기에 전달한다. 버스 에너지 측정기는 앞에서 정의한 버스 에너지 모델과 분할자 하나의 소비 에너지를 파라미터로 입력 받는다. 이는 분할자의 에너지에 의한 분할 버스에서의 에너지 절감 효과를 검증하기 위해서이다. 버스 에너지 측정기는 버스 에너지 모델에 각 태스크의 버스 트랜잭션 횟수와 태스크 할당 정보를 적용하여 버스에서 소모된 에너지를 측정한다.

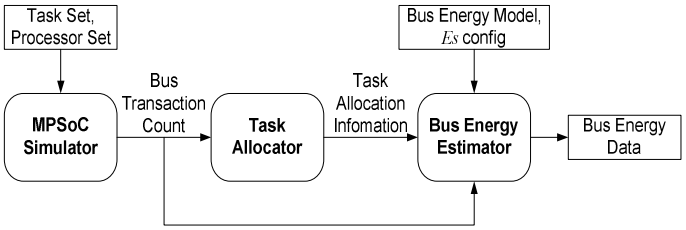

그림 6. 시뮬레이션 기반의 버스 에너지 측정 환경

본 실험에서 사용된 MPSoC 시뮬레이터는 SystemC [10]기반으로 구현되었다. 프로세서 시뮬레이터로는 캐쉬 모듈을 추가한 ARMulator [11]를 이용하였고, 캐쉬는 32바이트 블록 크기, 4웨이, 128 집합으로 설정하였다. 프로세서 이외의 모듈들-메모리, 버스, 중재자, 캐쉬 일관성 프로토콜 등-은 SystemC로 구현되었다. 온 칩 버스의 구조와 동작은 AMBA [12] 규격을 따른다.

시뮬레이션 벤치마크로는 임베디드 시스템용 벤치마크로 널리 사용되는 MiBench [13]의 jpeg, susan, adpcm, FFT에 대하여 버스 에너지를 측정하였다. 이들 프로그램들의 원본이 단일 태스크로 작성되었기 때문에 MPSoC 환경에 적용하기 위하여 다중 태스크로 수정하였다.

5.2 버스 분할 기법에 의한 버스 에너지 절감

표 1은 실험에 사용된 벤치마크 프로그램에서 발생하는 메모리 참조 유형에 따른 버스 트랜잭션 비율을 보여준다. 전체 메모리 참조중의 68%~97%에 해당하는 메모리 참조가 자신의 비 공유 메모리를 참조하며, 공유 메모리 읽기는 2%~16%, 공유 메모리 쓰기는 1%~16%에 불과하다. 이는 코드와 내부 데이터 등 비 공유 메모리에 대한 참조가 공유 데이터에 대한 참조보다 훨씬 많기 때문이다. 이중 비 공유 메모리 참조와 공유 메모리 읽기 부분에서 버스 분할에 의한 버스 에너지 절감이 기대된다.

표 1. 버스 트랜잭션 분포 비율

	K_{pmr}	K_{pmw}	K_{smr}	K_{smw}
susan	11%	86%	2%	1%
adpcm	2%	94%	3%	2%
FFT	1%	91%	6%	2%
jpeg	16%	52%	16%	16%

먼저, 제안된 버스 분할 방식에 의한 버스 에너지의 최대 감소량을 확인하기 위하여, 분할자의 소모 에너지를 무시했을 때의 버스 에너지를 측정하였다. 그림 7은 버스 에너지의 최대 감소량을 보여준다. 두 개의 막대 중에서 mono는 공유 버스 구조일 때 소모된 에너지를, split은 분할 버스에서 소모된 에너지를 나타낸다. 각 막대는 공유 버스일 때의 소모에너지를 1로 봤을 때의 상대적인 값을 나타낸다.

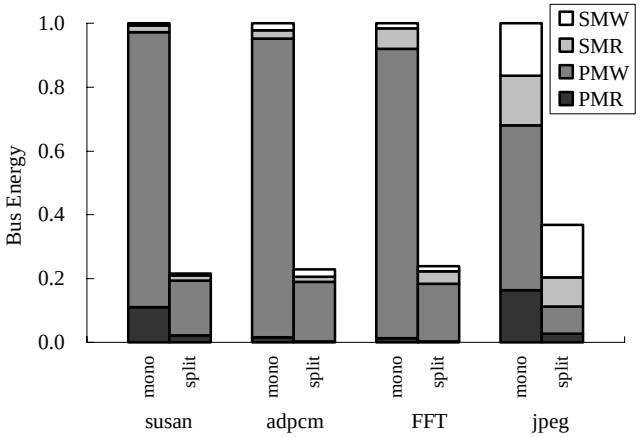

그림 7. 버스 분할 이전과 이후의 버스 에너지 비교

분할자에 의한 에너지 소모를 무시할 경우, 제안된 분할 버스에서 소모된 에너지는 공유 버스의 22%~37%로 감소하였다. 메모리 참조 형태에 따른 감소폭을 보면, 비 공유 메모리를 읽거나 쓸 때 버스 에너지가 공유 버스의 17%~20%로 감소한다. 이는 비 공유 메모리 참조 시에는 한 개의 버스 단편만 사용하기 때문이다. 공유 메모리 읽기 시에는 프로그램에 따라 58%~73%로 감소한다. 이는 P_1의 경우, 공유 메모리 참조를 위해 모든 버스 단편을 사용하는 반면에 P_N은 공유 메모리 참조시에 두 개의 버스 단편만 사용하기 때문이다. 그러나, 공유 메모리 쓰기 시에는 캐쉬

일관성을 위하여 어드레스 버스에서는 모든 버스 단편을 사용하므로, 버스 에너지에 미치는 영향이 작다.

분할 버스에서 버스 에너지는 버스 단편과 분할자에서 소모되는 에너지의 합으로 계산되므로, 분할 버스의 에너지를 정확히 측정하기 위해서는 분할자의 소비 에너지도 고려하여야 한다. 다만, 하나의 분할자가 소비하는 에너지는 고정적이나, 버스 단편 하나의 소비 에너지는 버스의 길이에 따라 가변이므로, 분할자의 에너지를 버스 단편에 기준한 상대적 값으로 변화시켜 분할 버스에서의 버스 에너지 소모를 측정하였다. 그림 8은 분할자의 에너지를 포함했을 때 분할 버스에서 소비되는 에너지를 보여준다. 5개의 막대가 하나의 프로그램에 대한 버스 에너지를 나타낸다. 5개의 막대들 중에서 mono는 공유 버스를 사용했을 때의 버스 에너지를 나타내며, 이후 S1~S4는 분할자의 소모 에너지 비율이 변경되었을 때의 분할 버스의 에너지를 나타낸다. S1은 분할자의 에너지가 무시할 정도로 작을 때, S2는 분할자가 버스 단편의 0.01배의 에너지를 소모할 때, S3은 0.1배의 에너지를 소모할 때, S4는 버스 단편과 같은 에너지를 소모할 때 분할 버스에서 소모하는 에너지를 보여준다.

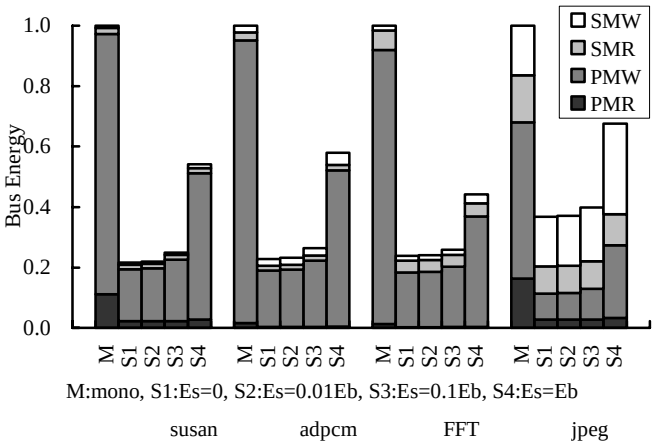

그림 8. 분할자 에너지 변화에 따른 버스 에너지 비교

분할자의 에너지 소모가 커지면, 분할 버스에서 소모되는 에너지도 커진다. 이는 분할 버스 구조에서는 버스 단편이 사용될 때, 이에 연결된 분할자도 함께 사용되기 때문이다. 분할자의 에너지 영향은 메모리 읽기보다는 쓰기시의 버스 에너지에 영향을 많이 준다. 이는 공유 메모리 읽기에는 프로세서와 공유 메모리 사이의 분할자만 사용되나, 공유 메모리 쓰기에는 모든 분할자가 사용되기 때문이다. 분할자가 버스 단편과 같은 양의 에너지를 소모하는 경우에도 분할 버스에서 소모하는 에너지는 공유 버스의 44%~68% 정도를 소모하는데, 이는 분할자의 에너지 소비가 사용되지 않는 버스 단편에 의한 에너지 절감보다 작기 때문이다.

5.3 태스크 할당 기법에 의한 버스 에너지 절감

캐쉬 일관성을 고려한 태스크 할당 기법에 의한 에너지 절감 효과를 검증하기 위하여, 다양한 태스크 할당 기법에 따른 버스 에너지를 측정하였다. 그림 9는 다양한 태스크 할당 기법에 따른 버스 에너지의 변화를 보여준다. 태스크 할당 기법은 공유 메모리 참조에 소비되는 버스 에너지를 절감시키는 방식이므로, 그림에서는 공유 메모리 참조에 의한 버스 에너지만을 보여준다. 그림에서 mono는 공유 버스에서의 에너지 소모를 나타낸다. 공유 버스에서는 모든 프로세서에서의 버스 트랜잭션 당 소비되는 버스 에너지가 동일하므로, 특별한 할당 방식을 적용하지 않았다. sequential은 분할 버스 구조에서 첫 번째 태스크를 첫 번째 프로세서에 할당하는 순차적 할당 기법을 적용했을 경우의 버스 에너지를 보여주며, CC-aware는 캐쉬 일관성을 고려한 할당 기법을 적용했을 때의 버스 에너지를 나타낸다.

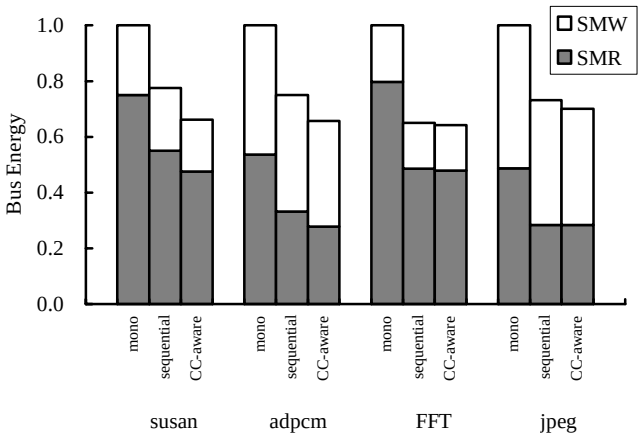

그림 9. 태스크 할당 방식에 따른 버스 에너지 비교

분할 버스는 공유 메모리 참조시에 필요한 버스 단편만 사용하므로, 순차적 할당 방식의 태스크 할당에서도 버스 에너지는 공유 버스의 65%~77% 수준으로 낮아진다. 캐쉬 일관성을 고려한 할당 방식은 공유 메모리 읽기에 더 높은 비중을 둔다. 공유 메모리 쓰기 시에는 캐쉬 일관성을 위하여 모든 버스 단편을 이용해야 하므로, 태스크 할당 방식에 의한 영향이 작기 때문이다. 캐쉬 일관성을 고려한 할당 방식의 경우, 순차적 할당 방식보다 더 낮아진 64%~70% 수준의 에너지만을 소비한다. 이는 태스크 중에 공유 메모리 참조 횟수가 많은 태스크가 공유 메모리에 가까운 프로세서에 할당되기 때문이다.

제안한 태스크 할당 기법에서 에너지 절감 효과에 영향을 미치는 요소들을 분석하기 위하여 다양한 버스 트랜잭션 횟수를 가진 태스크들을 임의로 생성하여 버스 에너지를 분석하였다. 태스크 개수 4개, 6개, 8개에 대하여, 각각의 경우에 소비되는 버스 에너지를 살펴보았다. 프로세서의 개수는 태스크의 개수와 동일하게 설정하였고, 각 태스크의 메모리 참조에 의한 버스 트랜잭션 횟수는 태스크 별로 차등하여 설정하였다. 각 태스크 별 버스 트랜잭션 횟수 분포 모델로는 균등 분포(uniform distribution), 다항식 분포(polynomial distribution), 지수 분포(exponential distribution) 이렇게 세가지 모델을 적용하였다. 아래의 식은 각 태스크 별 버스 트랜잭션 횟수에 대한 분포식이다.

$$Uniform: \ comm\_num(T_i) = \alpha$$
$$Polynomial: \ comm\_num(T_i) = \alpha + \beta i$$
$$Exponential: \ comm\_num(T_i) = \alpha + \beta^i$$
$$comm\_num(T_i): \text{ 태스크 } T_i \text{ 의 버스 트랜잭션 횟수}$$

분포 식에서 α값과 계수(coefficient)인 β값에 따라 태스크의 버스 트랜잭션 횟수가 달라진다. 예를 들어, 태스크가 네 개인 시스템에서 α값이 10이고, β값이 2라고

가정하면, 각 태스크 T_1, T_2, T_3, T_4의 버스 트랜잭션 횟수는 균등 분포 일 경우에는 10으로 모두 동일하고, 다항식 분포일 경우에는 12, 14, 16, 18이고, 지수 분포일 경우에는 12, 14, 18, 26로 변화한다. 실험에서는 α값을 10으로 고정시키고, β값을 4, 5, 6, 7로 변화시켜 가면서 결과를 분석해 보았다. 결과 그래프에 표시된 에너지는 공유 메모리의 읽기, 쓰기 시 소모되는 버스 에너지만을 고려한 그래프이다. 각 에너지 그래프는 상대적 값으로 표시되는데, 이 때 기준이 되는 값은 모든 경우의 태스크 할당에 따른 에너지 소모의 평균값이다. 즉, 평균적인 에너지 소모에 대비한 최소 에너지 소모 값을 막대 그래프로 표시하였다.

먼저, 분포 식에 따른 에너지 소모 분석 및 태스크 개수에 따른 에너지 소모를 분석하였다. 그림 10은 각 태스크 별로 통신량을 분포 식에 따라 다르게 주었을 경우, 캐쉬 일관성을 고려한 태스크 할당 기법을 적용하였을 때 소비되는 버스 에너지를 보여준다. 각 태스크의 읽기와 쓰기는 모두 같은 횟수로 수행된다고 가정하였다.

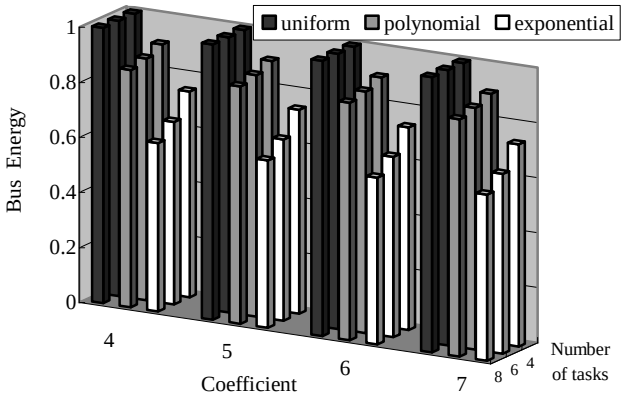

그림 10. 트랜잭션 분포에 따른 버스 에너지 비교

각 태스크의 버스 트랜잭션 분포가 균등 분포인 경우에는 모든 태스크가 동일한 통신량을 가지므로, 태스크 할당에 상관없이 동일한 버스 에너지를 소모한다. 따라서, 상대적 에너지 값도 항상 1로 고정되어 있음을 확인할 수 있다. 그러나, 버스 트랜잭션 분포가 다항식 분포와 지수 분포가 되면 에너지 소모가 확연하게 줄어드는 것을 볼 수 있다. 다항식 분포의 경우, 제안한 태스크 할당 방법을 사용했을 경우 평균 18%정도의 에너지 절감 효과가 있음을 알 수 있다. 지수 분포의 경우에는 50%정도의 에너지 절감 효과가 있다. 지수 분포의 경우가 다항식 분포의 경우보다 태스크 간의 버스 트랜잭션 횟수의 차이가 현격하게 나기 때문에, 태스크 할당을 통한 에너지 절감에 있어서도 보다 효과적인 결과를 보여준다. 그리고, 태스크(프로세서)의 개수에 따라서도 에너지 절감효과가 다르게 나타난다. 균등 분포와 다항식 분포의 경우에는 큰 변화가 없지만, 지수 분포의 경우에는 태스크의 개수가 많아질수록 에너지 절감 효과가 커진다. 이는 태스크가 많아질수록 태스크의 버스 트랜잭션 횟수의 차가 많아지는 효과와 함께, 프로세서와 공유 메모리 사이의 거리도 프로세서 별로 그 차이가 크게 나타나기 때문이기도 하다. 그러나, 각 버스 트랜잭션 분포 식의 α값과 계수인 β값에 따라서는 별다른 차이를 보이지 않고 있다. 그러므로, 전체 버스 트랜잭션 횟수의 크기와 태스크 할당 기법을 통한 에너지 절감 효과와는 상관 관계가 없음을 알 수 있다.

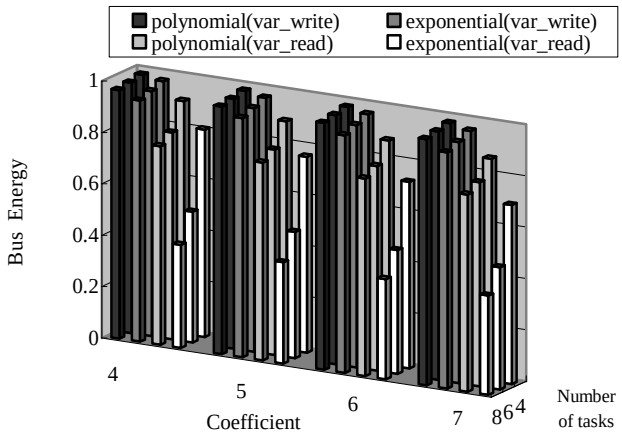

그림 11. 메모리 읽기, 쓰기 비율에 따른 버스 에너지 비교

다음으로, 공유 메모리 읽기와 쓰기 횟수에 따른 버스 에너지 소비의 차이를 분석하였다. 그림 11은 읽기와 쓰기의 비율에 따라서 태스크 할당을 통한 에너지 절감 효과가 어느 정도 되는지를 보여주는 결과이다. var_write는 읽기에 의한 버스 트랜잭션 횟수는 고정시키고 쓰기에 의한 버스 트랜잭션 횟수만 분포 식에 따라 변화시켜 가면서 통신량을 만들어 냈다. 마찬가지로 var_read는 쓰기에 의한 버스 트랜잭션 횟수는 고정시키고 읽기에 의한 버스 트랜잭션 횟수만 분포 식에 따라 변화시켜 가면서 통신량을 만들어 냈다. 각 고정 값은 50으로 동일하게 하였고, 분포 식은 이전의 실험과 동일하다. 그림 11에서 보여주는 결과는 쓰기보다 읽기에 의한 버스 트랜잭션 횟수가 에너지 소모에 보다 많은 영향을 미치고 있음을 보여준다. 이는 스누피 기반의 캐쉬 일관성 프로토콜의 경우 쓰기보다 읽기가 버스 에너지에 영향을 주는 요소가 크기 때문이다. 즉, 쓰기의 경우는 캐쉬 일관성을 위해 어드레스 버스에서는 항상 동일한 에너지를 소모하게 되고, 데이터 버스에서만 태스크 할당 방식에 따라 소모하는 에너지가 다르다. 그에 반해, 읽기의 경우에는 캐쉬 일관성에 아무런 영향을 미치지 않기 때문에 어드레스 버스와 데이터 버스 모두 태스크 할당 방식에 따라 소모하는 에너지가 다르게 된다.

6. 결론

본 논문에서는 공유 메모리 기반의 MPSoC 플랫폼에서 소비되는 버스 에너지를 절감시키기 위한 버스 분할 기법을 제안하고 동작 모델을 설명하였다. 또한, 캐쉬 일관성을 고려한 태스크 할당 기법을 함께 제안하였다. 제안된 버스 분할 기법은 비 공유 메모리와 공유 메모리의 버스를 분할함으로써, 캐쉬 일관성을 유지하며 비 공유 메모리를 참조할 때 소비하는 버스 에너지를 최소화시며, 태스크 할당 기법은 태스크 별 유효 버스 트랜잭션 횟수를 기반하여 태스크를 할당함으로써, 공유 메모리를 참조할 때 소비하는 버스 에너지를 최소화시킨다. 시뮬레이션을 통한 실험에서 제안된 버스 분할 기법은 비 공유 메모리 참조를 위한 버스 에너지를 기존의 공유 버스에 비해 최대 83%까지 절감시키며, 캐쉬 일관성을 고려한 태스크 할당 기법은 공유 메모리를 위한 버스 트랜잭션에서 최대 36%의 버스 에너지를 절감시킴을 알 수 있었다. 그럼으로, MPSoC와 같은 다중 프로세서 환경에서도 버스 분할 기법을 적용하여 버스 에너지 절감 효과를 볼 수 있으며, 더불어 캐쉬 일관성을 고려한

태스크 할당 기법을 통해 추가적으로 버스 에너지를 절감할 수 있음을 보여준다.

감사의 글

본 논문의 실험에 사용한 시뮬레이터의 원본 코드를 제공해주고, 실험을 위한 시뮬레이터의 수정에 조언을 주신 서울대학교 전기공학부의 정진용씨께 감사 드립니다. 본 논문은 정보통신부 및 정보통신 연구진흥원의 정보통신 선도기반기술 개발사업의 연구 결과로 수행되었습니다.

참고 문헌

[1] E. Arts and R. Roovers. “IC Design Challenges for Ambient Intelligence,” DATE, pp. 3-7, 2003.

[2] Vijay Raghunathan, Mani B. Srivastava, and Rajesh K. Gupta. “A Survey of Techniques for Energy Efficient On-Chip Communication,” DAC, pp. 900-905, 2003.

[3] J. Y. Chen, W. B. Jone, S. Wang, H. I. Lu, and T. F. Chen. “Segmented Bus Design For Low-Power Systems,” IEEE Trans. On VLSI Sys., Vol.7, No.1, pp.25-29, March 1999.

[4] Cheng-Ta Heish and Massoud Pedram. “Architectural Energy Optimization by Bus Splitting,” IEEE Trans. on CAD of Integrated Circuits and Sys., Vol.21, No.4, April 2002.

[5] Ruibing Lu and Cheng-kok Koh. “A High Performance Bus Communication Architecture through Bus Splitting,” ASP-DAC, pp.751-755, 2004.

[6] Kanishka Lahiri and Anand Raghunathan. “Power Analysis of System-Level On-Chip Communication Architectures,” CODES+ISSS, pp. 236-241, 2004.

[7] Mirko Loghi and Massimo Poncino. “Exploring Energy/Performance Tradeoffs in Shared Memory MPSoCs: Snoop-Based Cache Coherence vs. Software Solutions,” DATE, pp. 508-513, 2004.

[8] Per Stenstrom. “A Survey of Cache Coherence Schemes for Multiprocessors,” IEEE Computer, Vol. 23, No. 6, pp. 12-24, June 1990.

[9] Gu-Yeon Wei, M.A. Horowitz, and J. Kim, “Energy-efficient design of high-speed links,” Chapter 8 of Power Aware Design Methodologies, Editors: M. Pedram and J. Rabaey, Kluwer Academic Publishers, Norwell, MA, 2002.

[10] Synopsys, Inc.,“SystemC Language,”http://www.systemc.org.

[11] GNU, “ linux-arm-elf Tool Chain, ” http://ecos.sourceware.org/tools/linux-arm-elf.html.

[12] ARM Ltd., “AMBA Specification (Rev 2.0),” May 1999.

[13] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” WWC, pp. 3-14, 2001.