논문 2022-17-44

차량용 임베디드 프로세서에서 저전력 반응적 제어를 위한 이기종 멀티코어 협력적 스트리밍 온-칩 소프트웨어 구조

(Collaborative Streamlined On-Chip Software Architecture on Heterogenous Multi-Cores for Low-Power Reactive Control in Automotive Embedded Processors)

권 지 수, 박 대 진*
(Jisu Kwon, Daejin Park)

Abstract : This paper proposes a multi-core cooperative computing structure considering the heterogeneous features of automotive embedded on-chip software. The automotive embedded software has the heterogeneous execution flow properties for various hardware drives. Software developed with a homogeneous execution flow without considering these properties will incur inefficient overhead due to core latency and load. The proposed method was evaluated on an target board on which a automotive MCU (micro-controller unit) with built-in multi-cores was mounted. We demonstrate an overhead reduction when software including common embedded system tasks, such as ADC sampling, DSP operations, and communication interfaces, are implemented in a heterogeneous execution flow. When we used the proposed method, embedded software was able to take advantage of idle states that occur between heterogeneous tasks to make efficient use of the resources on the board. As a result of the experiments, the power consumption of the board decreased by 42.11% compared to the baseline. Furthermore, the time required to process the same amount of sampling data was reduced by 27.09%. Experimental results validate the efficiency of the proposed multi-core cooperative heterogeneous embedded software execution technique.

Keywords : Automotive, Embedded system, Heterogeneous execution, Micro-controller unit, Multi-core

Ⅰ. 서론 및 관련 연구

최근 자동차의 패러다임이 내연기관에서 모터, 배터리 제어를 비롯한 전자 제어 장치들로 전환되어감에 따라 차량 내부의 각종 센서 동작을 제어하는 임베디드 소프트웨어에 대한 관심이 높아지고 있다 [1-5]. 차량에서 사용되는 임베디드 소프트웨어는 내부의 여러 전자 제어 유닛 (ECU)에서 프로그래밍된 대로 각각의 역할을 수행한다. 또한, 차량에서 ECU가 차지하는 비중도 점차 커지고 있다. 최근에는 차량의 원가에서 ECU 및 소프트웨어가 차지하는 비중이 50%에 육박하고 있다. 하지만, ECU를 구성하는 마이크로컨트롤러 유닛 (MCU)에 적재될 소프트웨어는 운영체제 위에서 수행되는 일반적인 프로그램이 아니기에 개발 단계에서 효율성 증대를 위해 차량용 어플리케이션에 대한 이해가 필요하다.

ECU의 차량용 소프트웨어는 코드 라인의 증가 및 제어 대상 센서 수의 증가에 따라 수행 복잡도 또한 증가하고 있다. 이러한 복잡성으로 인해 소프트웨어 실행과정에서 발생하는 지연 및 오류는 차량용 소프트웨어의 특성상 큰 피해로 이어질 수 있다. 따라서 차량용 ECU에 사용되는 MCU와 소프트웨어 특성을 바탕으로 효율적 실행이 가능케 하는 연구가 필요하다 [6-11].

본 논문에서는 소프트웨어의 실행 관점에서 동종 (homogeneous) 및 이기종 (heterogeneous) 특성을 정의한다. 소프트웨어 실행에서의 동종 특성은 동일한 제어 흐름에서 동일한 컴포넌트에 접근하고 데이터를 처리하는 것으로 정의한다. 반대로 이기종 특성은 실행 과정에서 데이터 처리 또는 연산을 위해 서로 독립적인 컴포넌트에 접근하는 제어 흐름으로 정의한다. 대부분의 차량용 MCU 임베디드 소프트웨어는 제어 대상을 센싱하고, 취득한 데이터를 처리하여 이를 전송하는 일련의 과정으로 구성된다. 하지만 이러한 동종 소프트웨어 실행 흐름으로 이루어진 상태에서는 다양한 하드웨어 제어 요소들로 인해 많은 오버헤드가 발생한다. 임베디드 소프트웨어에서는 analog-to-digital converter (ADC)를 사용한 샘플링이나, 인터럽트의 처리 등 소프트웨어 실행 흐름에 따라 개발자가 개입하여 최적화할 수 있는 여지가 매우 큰 이기종 특성을 보인다 [12-14].

본 논문은 범용 마이크로컨트롤러가 아닌 차량용 임베디

*Corresponding Author (boltanut@knu.ac.kr)
Received: Oct. 14, 2022, Revised: Oct. 25, 2022, Accepted: Nov. 15, 2022.
J. S. Kwon: Kyungpook National University (Ph.D. Student)
D. J. Park: Kyungpook National University (Assoc. Prof.)
※ 본 논문은 교육부의 재원으로 한국연구재단 (NRF-2018R1A6A1A03025109, NRF-2022R1I1A3069260)의 지원을 받아 수행된 연구임.
※ 본 논문은 과학기술정보통신부의 재원으로 정보통신기획평가원 (No. 2021-0-00944, No. 2022-0-00816, No. 2022-0-01170)의 지원을 받아 수행된 연구임.

ⓒ IEMEK J. Embed. Sys. Appl. 2022 Dec. 17(6) 375-382
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2022.17.6.375

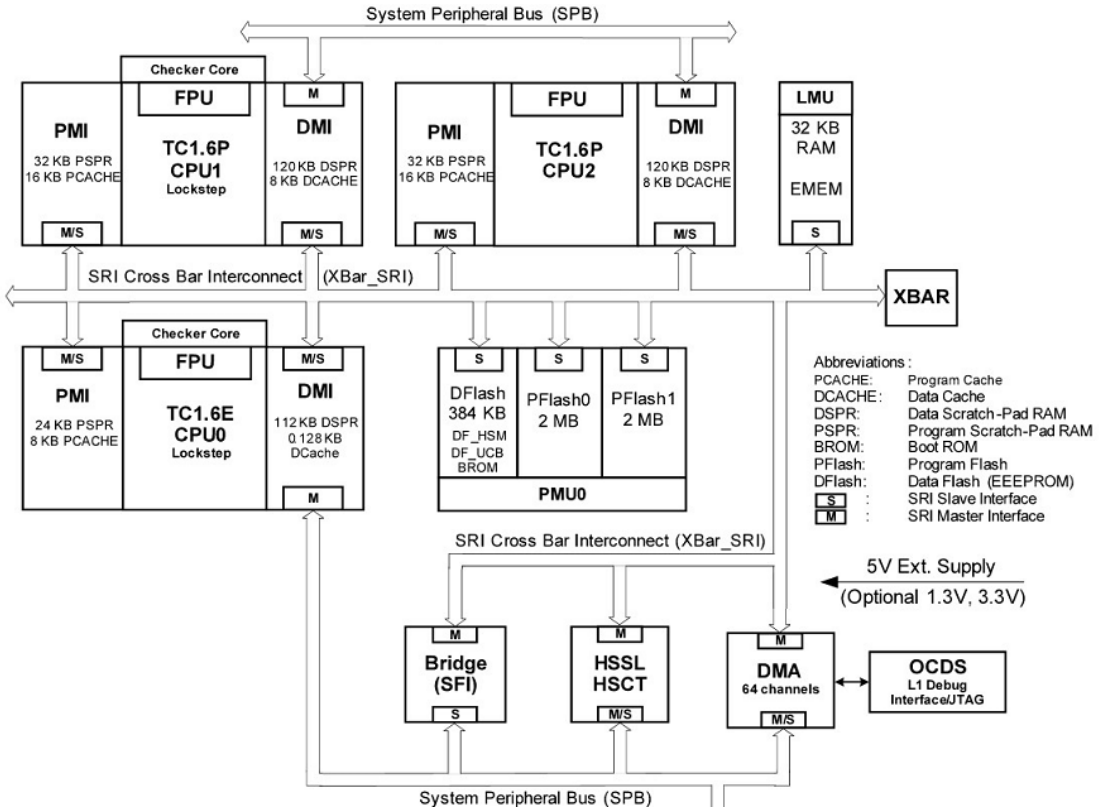

그림 1. TC275 MCU의 내부 구조 블록 다이어그램 [18]
Fig. 1. The internal architecture block diagram of the TC275 MCU

드 프로세서에 특화된 소프트웨어 구조를 제안하고자 한다. 차량용 임베디드 프로세서는 고온, 고압 등 열악한 환경에서 동작하기 위해 범용 MCU와 내부 구조, 개발 방법 등에서 차이가 존재한다. Infineon사의 AURIX™ TriCore™ 차량용 MCU는 독립적으로 소프트웨어 실행이 가능한 복수의 코어를 온-칩에 내장하고 있다. 차량용 MCU의 멀티-코어를 활용하여 한정된 자원을 효율적으로 사용하고자 하는 연구가 활발히 수행되고 있다 [15-17]. 본 논문에서는 멀티-코어 환경에서 동종 실행 흐름으로 구현된 소프트웨어의 태스크들을 차량용 소프트웨어의 이기종 실행 특성에 맞게 각각의 코어에 분담하는 기법을 제안한다. 임베디드 소프트웨어의 전체 실행과정은 크게 데이터 sensing, processing, collecting으로 나눌 수 있는데, 각각의 과정들을 태스크로 할당하고 이를 복수의 코어에서 연속적으로 실행할 수 있도록 코어 간의 코드 실행을 스케줄링한다. 결과적으로 태스크 간의 대기 시간으로부터 발생하는 오버헤드를 최소화함으로써 실행 시간 및 전력 소모를 개선할 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서는 Infineon사의 AURIX™ TriCore™ 임베디드 소프트웨어 개발과정을 소개한다. 3장에서는 동종 실행 흐름으로 구현된 임베디드 소프트웨어에 제안하는 기법을 적용하는 과정을 설명한다. 4장에서는 타겟 보드에서 코어 간의 협력적 이기종 연산을 적용하여 실행 시간 및 전력 소모를 분석한 결과를 설명하고, 마지막 5장에서 결론을 맺도록 한다.

Ⅱ. 차량용 임베디드 소프트웨어 개발

본 논문에서 사용하는 차량용 MCU는 Infineon사의 AURIX™ TriCore™ TC27x 멀티-코어 계열이다. TC27x 계열의 MCU 중, 본 논문에서는 TC275를 대상으로 설명 및 실험을 수행한다. 코어 3개로 이루어진 TC275의 전반적인 구조를 그림 1에서 나타내었다 [18].

하단 SPB (system peripheral bus)에 연결된 여러 가지 주변 장치 (peripheral)들은 그림에서 생략하였다. TC275에는 CPU0부터 CPU2까지 3개의 코어가 존재하는데, 각 코어는 독립적으로 수행할 코드를 저장하기 위한 플래시 메모리를 보유하고 있다. 모든 코어는 동일한 TriCore™ 1.6 아키텍처 기반으로 구현되었지만, 상대적으로 저전력에 초점을 맞춘 E-코어 (efficiency) 1개와 성능에 초점을 맞춘 P-코어 (performance) 2개로 구성된다. E-코어와 P-코어는 캐시 메모리의 크기나 파이프라인 구조에서 서로 다른 특징을 보인다.

멀티-코어 구조에서는 온-칩 메모리 요소에 대한 접근을 관리하는 것이 중요하다. TC275의 각 코어는 독립적으로 사용하는 scratch-pad (SPR), 캐시 메모리뿐만 아니라, cross-bar 버스 (SRI)를 통해서 공유 메모리 자원들에 접근할 수 있다. 코어가 사용할 수 있는 공유 메모리 자원들에는 PMU (Program Memory Unit)의 코드/데이터 플래시 메모리 (PFlash/DFlash), LMU (Local Memory Unit)의 SRAM이

표 1. Infineon AURIXTM TriCoreTM TC275 MCU 사양
Table 1. Specification of Infineon AURIXTM TriCoreTM TC275 MCU

	TC1.6E (Efficiency)	TC1.6P (Performance)
Operating Clock Frequency	up to 200 MHz	up to 200 MHz
Scratch-Pad RAM (SPR)	Data: up to 112 Kbyte (DSPR) Instruction: up to 24 Kbyte (PSPR)	Data: up to 120 Kbyte (DSPR) Instruction: up to 32 Kbyte (PSPR)
Cache	Instruction: up to 8 Kbyte (ICACHE)	Data: up to 8 Kbyte (DCACHE) Instruction: up to 16 Kbyte (ICACHE)
Flash Memory	Data: up to 384 Kbyte (DFLASH) Instruction: up to 4 Mbyte (PFLASH)	
SRAM	32 Kbyte	

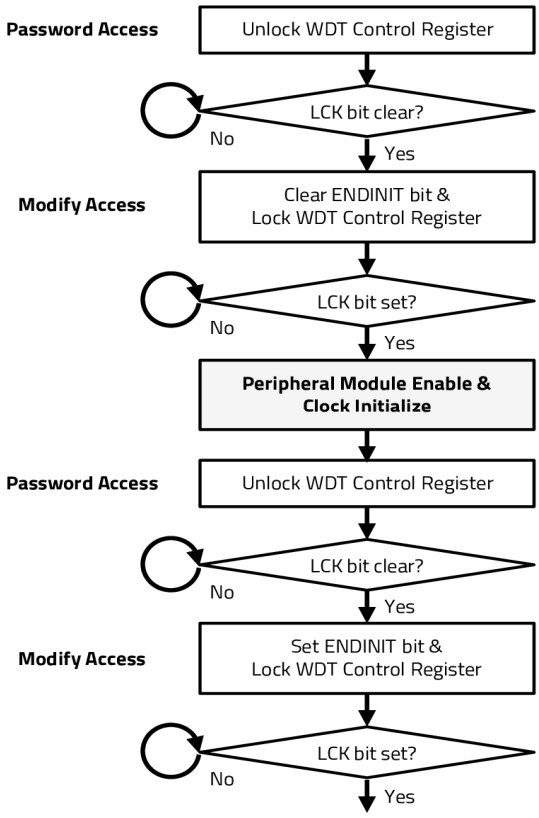

그림 2. 시스템 critical WDT 제어 레지스터 잠금 해제를 통한 주변 장치 모듈 초기화 순서도
Fig. 2. Peripheral module initializing flow chart by system critical WDT control register unlock

있다. TC275 MCU의 세부 사양은 표 1에 기재하였다. TC275의 주변 장치 제어는 SPB를 통한 레지스터 설정으로 이루어진다. 3개 코어 모두 SPB에 연결되어 동일한 조건에서 주변 장치들을 제어할 수 있다. 이때 후술할 초기화 과정에서 특정 주변 장치에 접근하는 코어에 따라 각 코어에 할당된 보호 레지스터 설정이 필요하다.

TC275는 차량용으로 설계된 MCU이기에 일반적인 범용 MCU와 다른 특징이 존재한다. 차량용뿐만 아니라, 범용 MCU에 적재되는 소프트웨어 설계는 대상 하드웨어에 대한

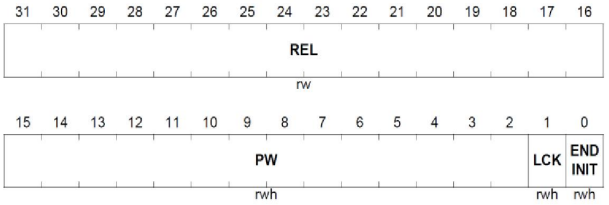

그림 3. 32비트 WDT 제어 레지스터 필드 구조
Fig. 3. 32-bit WDT control register field structure

이해를 바탕으로 개발되어야 한다. 본 논문에서 대상으로 삼고 있는 차량용 임베디드 프로세서의 멀티 코어 협력적 연산을 구현하기 위해서 MCU 하드웨어 제어에 대한 설명을 선행한다. 차량용 임베디드 프로세서의 다양한 특징 중, 주변 장치 제어를 하기 위한 프로그래밍 관점에 대해 설명하도록 한다. 주행 중인 차량의 내부는 높은 온도와 압력, 진동으로 인해 전자 제어 장치의 동작에 노이즈가 발생하기 쉬운 열악한 환경이다. 이러한 환경에서도 차량용 MCU는 외란의 영향을 최소화하며 강인하게 동작해야 차량의 피해가 발생하지 않는다. 따라서 TC275는 사용자의 의도와 달리 MCU의 모듈 컴포넌트가 비정상적으로 동작하는 상황을 방지하기 위해 복잡한 시퀀스 (sequence)를 통해 모듈을 초기화하도록 한다. TC275에서 하드웨어 모듈을 사용하기 위해 초기화하는 과정은 그림 2와 같다.

모듈을 구동하거나 클락 신호를 인가하는 설정에 해당하는 레지스터는 TC275에서 system critical 레지스터에 해당하기 때문에 이를 조작하려면 Password Access, Modify Access 두 단계의 시퀀스가 필요하다. TC275 멀티-코어 구조는 3개의 코어를 내장하는데, 각 코어에서 병렬적인 주변 장치 접근을 보장함과 동시에, 동일한 주변 장치 접근으로 인한 버스 충돌을 방지하기 위해서 코어마다 할당된 WDT (watchdog timer) 제어 레지스터가 존재한다. 그림 3은 첫 번째 코어 CPU0에 할당된 32비트 WDT 제어 레지스터이다.

임의의 주변 장치 모듈에 클락 신호를 공급하고 enable시키기 위해선 WDT 제어 레지스터의 ENDINIT 필드를 set해야 하고, 이를 위해서는 Password Access를 통해 WDT 제어 레지스터의 잠금을 먼저 해제해야 한다. 이때 CPU0

표 2. 시스템 critical 레지스터 접근에 따른 LCK, ENDINIT 비트 조합
Table 2. LCK, ENDINIT bit combination according to system critical register access

	Password Access	Modify Access (Before Init.)	Modify Access (After Init.)
LCK	0	1	1
ENDINIT	1	0	1

에서 주변 장치를 enable하는 코드가 수행 중이라고 가정하면, 서로 다른 코어 사이의 충돌을 방지하기 위해 CPU0에 할당된 WDT 제어 레지스터 WDTCPU0CON0에서 잠금을 해제할 수 있다. Password Access를 위해서는 32비트 전체 WDT 제어 레지스터의 값을 읽되, PW 필드는 값을 반전하여 읽는다. 그리고 레지스터로부터 읽은 값을 그대로 다시 WDT 제어 레지스터에 write하는데 레지스터의 잠금을 풀고자 하는 것이 목적이므로 LCK 비트의 값을 0으로 clear, ENDINIT 비트의 값을 1로 set한다. 유의해야 할 점은 WDT 제어 레지스터에 32비트 전체를 한 번에 write해야 한다는 점이다. WDT 제어 레지스터의 LCK 비트가 0으로 읽히면 잠금이 해제되고 ENDINIT 비트를 수정할 수 있다.

WDT 제어 레지스터의 ENDINIT 비트가 0의 값이어야만 주변 장치 초기화 설정이 가능하므로, Modify Access를 통해 잠금이 해제된 레지스터의 ENDINIT 비트를 수정한다. 앞서 Password Access와 동일하게 WDT 제어 레지스터의 값을 읽고, 그대로 write하되, ENDINIT 비트의 값을 0으로 clear함과 동시에 LCK 비트를 잠그기 위해 1로 set한다. LCK 비트가 1로 읽히면 주변 장치를 제어하는 레지스터에서 해당 모듈을 enable하거나 클락을 인가하도록 설정할 수 있다. 주변 장치 초기화 설정을 완료한 후에는 이전과 동일하게 Password Access에서 WDT 제어 레지스터의 잠금을 해제하고, Modify Access에서 WDT 제어 레지스터의 ENDINIT 비트를 1로 set한다. WDT 제어 레지스터에 접근하는 목적에 따라 3가지의 조합이 존재한다. 각 조합에 따른 LCK, ENDINIT 비트의 값을 표 2에 나타내었다.

Ⅲ. 멀티-코어 협력적 연산

일반적인 차량용 임베디드 소프트웨어의 역할은 센서가 데이터를 수집하고, 이를 연산하여 통신 인터페이스로 전송하는 3가지 단계로 구분할 수 있다. 그림 4는 일련의 과정이 동종 소프트웨어 실행 흐름으로 구현된 경우를 나타내었다. 하나의 소프트웨어는 서로 다른 역할을 하는 복수의 태스크들로 구성되어 간헐적으로 발생하는 이벤트에 대응하는 서비스 복잡도뿐만 아니라, 구동하는 하드웨어도 모두 다르다. 이러한 이기종 특성을 고려하지 않고, 단일 코어 기반 실행 흐름으로 프로그램을 개발하게 되면 각 태스크 사이의 idle, wait으로 인한 불필요한 오버헤드가 발생할 수 있다.임베디드 소프트웨어 실행과정에서 데이터 센싱은 ADC의 샘플링으로 대응된다. ADC는 외부에서 입력되는 아날로그 신호를 샘플링하여 그 결과를 버퍼에 저장한다. 센싱 데이터에 digital signal processing (DSP) 등의 연산을 적용하는 경우는 많은 계산량으로 인해 오버헤드가 더 크게 작용할 수 있다. 본 논문에서 가정한 DSP 연산에는 이동 평균, FIR 필터 등 목표로 하는 도메인에 따라 다양한 알고리즘이 사용될 수 있다. 이러한 DSP 연산 처리 과정에서 ADC 샘플링 지연이나, DSP 연산 적용 범위의 데이터가 확보를 위한 대기로부터 오버헤드가 발생한다. 또한, DSP 처리가 완료된 데이터를 전송하는 통신 인터페이스로는 universal asynchronous receiver / transmitter (UART)가 사용되었다. DSP 연산이 완료되어 버퍼에 저장된 데이터들은 UART 통신으로 호스트에 전송된다. 전체 실행 흐름은 단일 코어에서 동종으로 구현되어 있지만, 세부적인 태스크들이 구동하는 하드웨어 요소, 코드 실행의 순서관계 등을 고려하면 이기종 실행 흐름 특성을 보인다. 이러한 특성을 고려하지 않은 상태로 차량용 임베디드 소프트웨어를 구현한다면 비효율적인 동작을 하게 된다.

본 논문에서 타겟으로 삼은 차량용 임베디드 프로세서에는 메모리 관리 유닛이 존재하지 않으므로 운영 체제 수준에서 제공하는 쓰레드, 멀티 프로세스 등, 기존 멀티 코어 활용을 위한 API를 사용하기 어렵다. 따라서 개발자의 타겟 차량용 임베디드 프로세서 하드웨어에 대한 이해를 바탕으

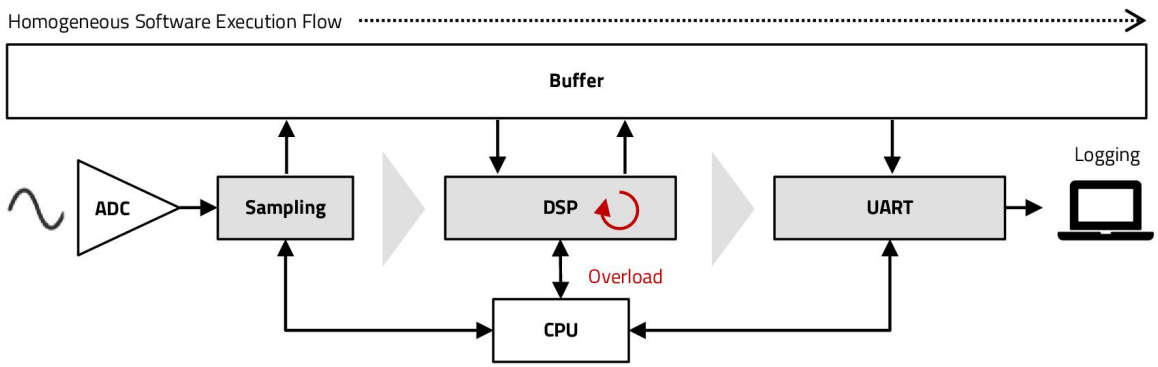

그림 4. 기존 동종 실행 흐름 임베디드 소프트웨어 구조
Fig. 4. Conventional homogeneous execution flow embedded software structure

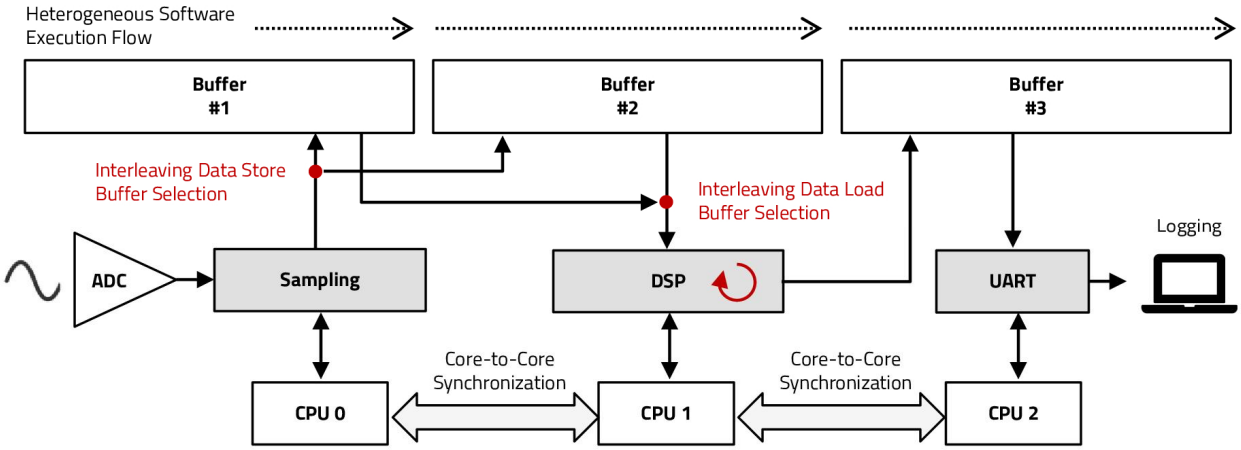

그림 5. 제안하는 멀티-코어의 협력적 연산 기반 이기종 임베디드 소프트웨어 실행 구조
Fig. 5. Proposed multi-core cooperative operation-based heterogeneous embedded software execution structure

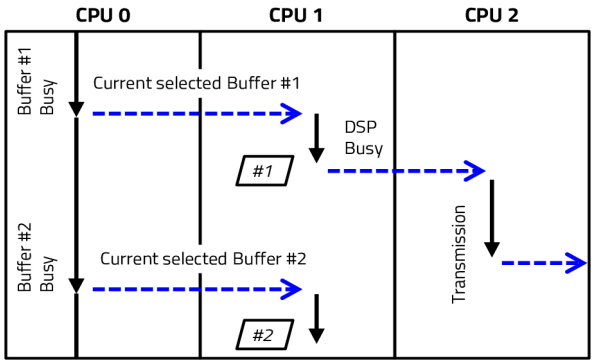

그림 6. 멀티-코어 동기화 개요
Fig. 6. Multi-core synchronization overview

로 한 소프트웨어 설계가 필요하다. 따라서 그림 5와 같이 멀티-코어 구조에서 각 코어에 이기종 태스크들을 할당하여 오버헤드를 최소화하고 연속적인 프로그램 실행이 가능케 하였다. 이를 위해서 기존의 단일 코어가 모든 과정을 관리하는 구조에서 3개의 코어를 사용하고 각 코어가 샘플링, DSP, 전송 태스크들을 나누어 맡도록 하였다. 또한, 기존의 단일 버퍼 구조에서 3개의 버퍼를 사용하도록 변경하였다. 첫 번째 코어 CPU 0은 ADC 하드웨어를 구동하여 샘플링을 수행한다. 샘플링 수행 결과는 버퍼에 저장하되, 버퍼의 사용 여부를 확인하여 첫 번째 버퍼 #1과 두 번째 버퍼 #2를 번갈아 가며 사용한다. 두 번째 코어 CPU 1에서는 앞서 CPU 0이 샘플링 데이터를 저장한 버퍼로부터 데이터를 읽어 DSP 연산을 수행한다. 이때 ADC 샘플링 데이터가 저장되는 버퍼가 2개이므로 CPU 0으로부터 idle 상태의 버퍼 정보를 확인하여 DSP를 수행한다. 예를 들어 현재 CPU 0이 첫 번째 버퍼에 샘플링 데이터를 저장 중이라면, CPU 1은 idle 상태인 두 번째 버퍼로부터 데이터를 읽어 DSP 연산 처리를 한다. CPU 1에서 DSP 연산이 완료된 데이터는 세 번째 버퍼 #3에 저장된다. 세 번째 코어 CPU 2는 버퍼 #3에 저장된 데이터를 UART 통신 인터페이스로 호스트 PC에 전송한다.

현재 멀티-코어를 사용하는 구조에서는 코어 간의 동기화 문제로 underrun, overrun이 발생할 수 있다. 이를 방지하기 위해 각 코어 간에 플래그를 사용하여 버퍼 및 DSP의 상태를 확인할 수 있도록 하였다. 각 코어 간의 동기화 관계를 그림 6에 나타내었다. CPU 1은 CPU 0으로부터 현재 ADC 샘플링 데이터가 저장되는 버퍼 상태를 확인하고, DSP 연산을 적용할 버퍼를 결정한다. 이때 더블 버퍼를 사용함으로써 버퍼 점유로 인해 발생할 수 있는 비효율적인 오버헤드를 제거하였다. 하나의 버퍼가 CPU 0에 의해 점유되는 동안, 다른 버퍼에 대해 CPU 1에서 접근하여 데이터 연산 처리를 위해 읽어감으로써 CPU 0의 연속적인 동작을 보장할 수 있다. CPU 2는 CPU 1로부터 DSP 연산 수행 상태를 확인하고 버퍼에 저장된 데이터 전송 시작 여부를 결정한다. 이러한 과정에서 CPU 1과 2는 CPU 0의 더블 버퍼 동작으로부터 발생하는 오버헤드 시간 동안에 idle 구간을 최소화할 수 있다.

Ⅳ. 실 험

멀티-코어 차량용 MCU에서 이기종 연산의 효율적 실행을 검증하기 위해 사용한 보드는 그림 7과 같다. Infineon 사의 TC275 MCU를 실장하고 있는 Hitex 사의 ShieldBuddy TC275 보드를 사용하였다 [18]. 타겟 보드가 실장하고 잇는 TC275 MCU는 TriCore™ v1.6.1 코어를 내장하고 있으며 최대 200 MHz 속도로 동작한다. 또한 온-칩에 포함된 3개의 코어는 각 코어의 명령어 실행을 위해 4Mbyte의 플래시 메모리를 갖추고 있다. RAM 용량은 472 Kbyte이다. 보드의 아날로그 핀으로부터 ADC 샘플링의 대상이 되는 신호를 입력받도록 하였다. 제안하는 기법을 검증하기 위한 버퍼의 크기는 100개로 설정하였고, 이는 각 코어에서 모두 동일하다. 실험에서 사용될 DSP 연산은 윈도우 크기가 10인 이동 평균 필터로 선택하였다. DSP 필터 연산을 거쳐 통신 인터페이스로 전송된 ADC 샘플링 데이

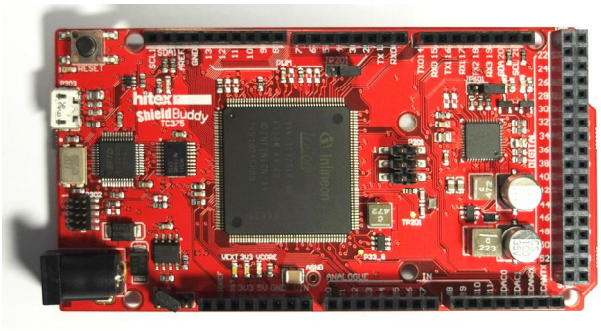

그림 7. 타겟 보드 (Hitex Shieldbuddy TC275)
Fig. 7. Target board (Hitex Shieldbuddy TC275)

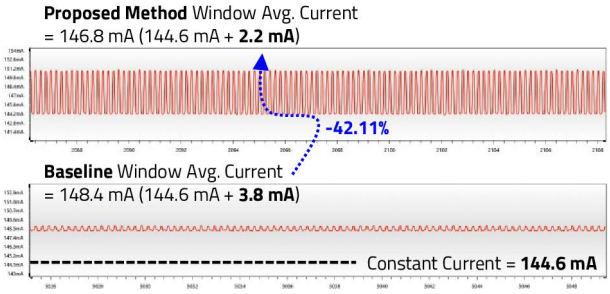

그림 8. 타겟 보드 전류 측정 시각화
Fig. 8. Target board current measurement visualization

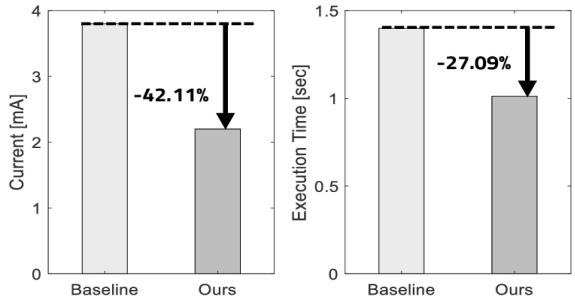

그림 9. 타겟 보드 전류 및 실행 시간 측정 결과
Fig. 9. Target board current and execution time measure result

터는 아날로그 신호에서 노이즈가 감소된 형태를 가진다. TC275 보드에서 호스트 PC로 데이터를 전송할 때는 UART 통신을 사용하도록 하였다.

멀티-코어 이기종 연산의 효율성을 검증하기 위해 비교될 baseline은 동종 실행 구조로 구현된 임베디드 소프트웨어이다. 비교 대상이 되는 baseline 소프트웨어는 ADC 샘플링 데이터가 가득 찬 버퍼에 대해 DSP 연산을 수행하고, DSP 연산이 완료된 버퍼에 대해 UART 인터페이스를 통한 전송을 수행하는 과정이 순차적으로 구현되어 있다. 제안하는 방법을 적용했을 때의 전력 소모량 변화를 비교하기 위해 TC275 보드의 전류를 측정하였다. 실험 과정에서의 전류 측정 구간이 연산 처리 중간에 위치할 경우, 전력 budget의 형태에 따라 잘못된 결과가 관측될 수 있다. 이를 방지하기 위해 전체 작업이 완료되어 충분한 반복이 이루어지기까지의 시간으로 설정하였다. 그림 8의 상단 그래프는 제안하는 기법을 적용한 경우, 하단 그래프는 baseline 경우의 보드에 흐르는 전류를 시각화한 결과이다. 10초 동안의 전류를 측정한 결과, 제안하는 기법을 적용한 경우는 146.8 mA, baseline은 148.4 mA의 평균 전류가 측정되었다. 이때 보드가 아무 동작도 하지 않는 경우 소모하는 상수 전류가 144.6 mA로 측정되어 이를 제외하면 평균 전류는 3.8 mA에서 2.2 mA로 42.11% 감소하였다. 또한, 실행 시간 개선을 검증하기 위해 두 가지 경우에서 500개 샘플을 출력하기까지의 소요 시간을 측정하였다. Baseline 경우는 1.399초가 소요됐지만, 제안하는 기법을 적용한 경우는 1.012초가 소요되어 27.09% 감소한 결과를 보였다. 이러한 결과는 그림 9에 나타내었다.

실험 결과로부터 멀티-코어 차량용 MCU에 임베디드 소프트웨어를 구현하는 경우, 이기종 실행 특성을 고려했을 때 더 효율적인 연산이 가능한 것을 확인할 수 있다. 성능 차이에 대한 원인을 분석하면 다음과 같다. 전력 그래프의 추이를 보면, baseline 경우의 전류는 TC275 보드가 아무 동작도 하지 않을 때의 상수 전류에 비해 큰 오프셋만큼 상승해있다. 반면, 제안하는 기법의 경우는 최대 전류는 baseline에 비해 다소 높지만, 태스크 사이에 idle한 구간에서는 보드의 상수 전류에 가까울 정도로 전륫값이 낮아진다. 그 결과 멀티-코어 이기종 실행방식으로 임베디드 소프트웨어를 구현하면 평균적인 전류는 낮아질 수 있다.

V. 결 론

본 논문에서는 차량용 MCU에 적재되는 임베디드 소프트웨어의 이기종 특징을 고려하여 멀티-코어의 협력적 연산 구조를 제안하였다. 또한, 멀티-코어가 내장된 차량용 MCU에서 사용되는 소프트웨어를 개발하는 일부 과정을 소개하였다. 차량용 소프트웨어는 다양한 하드웨어 구동을 위해 이기종 실행 흐름을 가지는 특성을 가진다. 이러한 특성에 대한 고려 없이 동종 실행 흐름으로 개발된 소프트웨어에서는 코어의 대기 시간이나 코어의 부하로 인해 비효율적인 오버헤드가 발생한다.

제안하는 방법은 멀티-코어를 내장한 차량용 MCU가 실장된 실제 보드에서 검증되었다. ADC 샘플링, DSP 연산, 통신 인터페이스 등 임베디드 시스템의 일반적인 태스크들이 포함된 소프트웨어가 동종 실행 흐름으로 구현된 경우, 제안하는 방법을 적용하여 멀티-코어 협력적 환경에 적합하게 이기종 실행 흐름으로 구현된 경우의 오버헤드 감소를 실증하였다. 제안하는 방법을 사용했을 때 이기종 태스크 사이에 발생하는 idle 상태를 활용하여 보드의 자원을 효율적으로 사용할 수 있었다. 실험 결과, baseline과 비교하여 보드의 소모 전력은 42.11% 감소하였다. 또한, 동일한 양의 샘플링 데이터를 처리하는데 소요된 시간은 27.09% 감소한 결과를 얻었다. 실험 결과로부터 제안된 멀티-코어 협력적

이기종 임베디드 소프트웨어 실행 방법의 효율성을 검증하였다. 향후 차량용 MCU에서 수행할 수 있는 다양한 어플리케이션을 대상으로 효율적인 연산을 가능케 하는 협력적 분산 처리 기법에 관해 연구를 진행할 예정이다.

References

[1] S. Elashri, A. Azim, “An Energy-aware Optimization Model for Real-time Systems Analysis and Design: Work-in-progress,” In Proceedings of the 2021 International Conference on Embedded Software (EMSOFT '21), pp. 45-46, 2021.

[2] J. Chang, S. Oh, D. Park, “Accuracy-Area Efficient Online Fault Detection for Robust Neural Network Software-Embedded Microcontrollers,” In Proceedings of the 2022 International Conference on Embedded Software (EMSOFT '22), 2022.

[3] H. R. Faragardi, B. Lisper, K. Sandström, T. Nolte, "A Resource Efficient Framework to run Automotive Embedded Software on Multi-core ECUs," Journal of Systems and Software, Vol. 139, pp. 64-83, 2018.

[4] M. Uelschen, M. Schaarschmidt, C. Fuhrmann, C. Westerkamp, "Work-in-Progress: PowerMonitor: Design Pattern for Modelling Energy-Aware Embedded Systems," 2019 International Conference on Embedded Software (EMSOFT), pp. 1-2, 2019.

[5] S. H. Aldaajeh, S. Harous, S. Alrabaee, "Fault-Detection Tactics for Optimized Embedded Systems Efficiency," IEEE Access, Vol. 9, pp. 91328-91340, 2021.

[6] N. Navet, F. Simonot-Lion, Automotive Embedded Systems Handbook, CRC Press, 2017.

[7] M. Ashjaei, L. L. Bello, M. Daneshtalab, G. Patti, S. Saponara, S. Mubeen, "Time-Sensitive Networking in Automotive Embedded Systems: State of the Art and Research Opportunities," Journal of Systems Architecture, Vol. 117, pp. 102137, 2021.

[8] F. Salewski, S. Kowalewski, "Hardware/Software Design Considerations for Automotive Embedded Systems," in IEEE Transactions on Industrial Informatics, Vol. 4, No. 3, pp. 156-163, 2008.

[9] G. L. Gopu, K. V. Kavitha, J. Joy, "Service Oriented Architecture based Connectivity of Automotive ECUs," 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), pp. 1-4, 2018.

[10] C. Bradatsch, T. Ungerer, R. Zalman, A. Lajtkep, "Towards Runtime Testing in Automotive Embedded Systems," 2011 6th IEEE International Symposium on Industrial and Embedded Systems, pp. 55-58, 2011.

[11] S. H. Lee, D. K. Lee, P. Choi, D. Park, “Efficient Power Reduction Technique of LiDAR Sensor for Controlling Detection Accuracy Based on Vehicle Speed,“ IEMEK J. Embed. Sys. Appl., Vol. 15, No. 5, pp. 215-225, 2020 (in Korean).

[12] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, K. Li, "Resource Consumption Cost Minimization of Reliable Parallel Applications on Heterogeneous Embedded Systems," in IEEE Transactions on Industrial Informatics, Vol. 13, No. 4, pp. 1629-1640, 2017.

[13] G. N. Khan, J. Levman, J. Alirezaie, "Hardware-Software Co-Synthesis of Heterogeneous Embedded Computer Systems," 2006 Canadian Conference on Electrical and Computer Engineering, pp. 1304-1307, 2006.

[14] Y. H. Fan, J. O. Wu, S. F. Wang, "Software Synthesis of Middleware for Heterogeneous Embedded Systems," 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 2084-2087, 2012.

[15] P. Gai, M. Violante, "Automotive Embedded Software Architecture in the Multi-core age," 2016 21th IEEE European Test Symposium (ETS), pp. 1-8, 2016.

[16] E. Díaz, E. Mezzetti, L. Kosmidis, J. Abella, F. J. Cazorla, "Modelling Multicore Contention on the AURIXTM TC27x," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1-6, 2018.

[17] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, K. Li, "Resource Consumption Cost Minimization of Reliable Parallel Applications on Heterogeneous Embedded Systems," in IEEE Transactions on Industrial Informatics, Vol. 13, No. 4, pp. 1629-1640, 2017.

[18] https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc2xx/aurix-family-tc27xt/

[19] https://www.hitex.com/microcontroller-support/aurix/shieldbuddy-tc275

Jisu Kwon (권 지 수)

2019 Electronics Engineering from Kyungpook National University (B.S.)

2019~Electronic and Electrical Engineering from Kyungpook National University (Integrated Ph.D. Student)

Field of Interests: machine learning, embedded system, microcontroller

Email: kjisu96@knu.ac.kr

Daejin Park (박 대 진)

2001 School of Electronics Engineering from Kyungpook National University (B.S.)

2003 School of Electronics Engineering from KAIST (M.S.)

2003~2014 Research Engineer, SK Hynix/ Samsung

2014 School of Electronics Engineering from KAIST (Ph.D.)

2016~School of Electronics Engineering from Kyungpook National University (Associate Professor

Field of Interests: Low-power SoC Design, Robust Embedded Systems

Email: boltanut@knu.ac.kr