논문 21-46-05-09 The Journal of Korean Institute of Communications and Information Sciences '21-05 Vol.46 No.05
https://doi.org/10.7840/kics.2021.46.5.821

런타임 이벤트 및 상태 로깅을 통한 차량용 ECU의 리셋 원인 분석 방법

최 재 형◆, 정 석 용*, 배 재 현**, 유 민 수°

The Methodology for Reset Cause Analysis of Automotive ECUs by Runtime Event-State Logging

Jaehyung Choi◆, Seokyong Jung*, Jaehyun Bae**, Minsoo Ryu°

요 약

차량용 ECU에서 발생하는 예측하지 못한 리셋은 신뢰성과 안정성을 해치는 중요한 문제이다. 하지만 리셋은 초기화를 동반하기 때문에 발생 후 원인 분석을 위한 단서를 찾기 힘들다. 외부 저장 장치를 사용하더라도 리셋은 원인이 다양하기 때문에 어떤 정보들이 원인 분석에 필요할지 선별하기 어렵다. 원인 분석을 위해 사용 가능한 기존의 디버깅 방법들은 리셋 원인 분석에 불충분한 정보를 수집하거나 분석의 범위가 한정되어 있다. 본 논문에서는 차량용 ECU의 런타임에서 정보를 수집하여 로그를 생성하고 이를 통해 원인을 규명하고 가장 먼저 조사해야 할 로그를 선별해주는 리셋 원인 분석 플랫폼을 제안한다. 제안하는 플랫폼은 가장 근원적인 수준의 원인을 8가지로 한정하고 로그를 분석하여 어떤 종류의 원인인지 판단하여 준다. CPU 3%, 휘발성 메모리 1.6%, 비휘발성 메모리 1%정도의 Overhead로 동작함을 확인하였고 이를 통한 리셋 원인 분석 사례를 제시한다. 본 연구에서 제안하는 리셋 원인 분석에 관한 방법론은 차량용 ECU 개발 시 빠른 원인 분석을 도와주어 리셋과 관련된 개발 비용을 감소시켜줄 수 있다.

Key Words : Automotive ECU, Real-Time Embedded System, Safety, AUTOSAR, OSEK OS

ABSTRACT

Unpredictable resets occurring in automotive ECUs are an important problem that harms reliability and stability. However, because reset is accompanied by initialization, it is difficult to find clues for cause analysis after occurrence. Even with external storage devices, reset has a variety of causes, making it difficult to select which information will be needed for cause analysis. Existing debugging methods available for cause analysis either collect insufficient information for reset cause analysis or have limited scope of analysis. In this paper, we propose a reset cause analysis platform that collects information from the runtime of an ECU for vehicles, generates logs, thereby identifying the cause and screening the logs that need to be investigated first. The proposed platform limits the most fundamental level of causes to eight, and analyzes the logs to determine what kind of causes they are. We have confirmed that it operates with an overhead of 3% CPU, 1.6% volatile memory, and 1% non-volatile memory, and we present examples of reset cause analysis. The methodology for reset cause analysis proposed in this work can help with fast cause analysis in the development of an ECU for vehicles, reducing the development cost associated with reset.

※ 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00590, 대규모 노드에서 블록단위의 효율적인 거래 확정을 위한 최종성 보장 기술개발)
◆ First Author : Hyundai AutoEver Corp, Jaehyung.Choi@hyundai-autoever.com, 정회원
° Corresponding Author : Hanyang University Department of Computer Software, msryu@hanyang.ac.kr, 정회원
* Hanyang University Department of Computer Software, syjung@hanyang.ac.kr, 학생회원
** Hanyang University Department of Automotive Control Engineering, baejh0517@naver.com, 학생회원
논문번호 : KICS2021-02-034-B-RU, Received February 4, 2021; Revised February 17, 2021; Accepted February 17, 2021

Ⅰ. 서 론

차량용 ECU의 예측하지 못한 리셋 현상은 시스템의 초기화를 동반하기 때문에 심각한 결함으로 이어질 수 있다. 이는 시스템의 신뢰성과 안정성을 해치는 중요한 문제이기 때문에, 리셋 현상은 재발 방지를 위해 원인 분석이 이루어져야만 한다. 하지만 리셋은 여러 단계에 걸쳐 HW와 SW를 초기화하므로 발생한 후에는 원인 분석을 위한 단서를 찾기 힘들다. 이 때문에 초기화되지 않는 저장 수단에 실행 정보들을 남기더라도 리셋 유발 원인이 다양하여 추후 어떤 정보들이 해당 원인 분석에 필요할지 선별하기가 어렵다. 또한 원인 분석 시 직접 런타임 실행 정보를 검사하여 리셋 유발원을 예측해야 한다는 문제가 있다.

본 논문에서는 차량용 ECU의 런타임에서 원인 분석에 필요한 정보를 선별하여 수집하고 리셋 발생 시 단계적으로 원인을 분석하도록 도움을 주는 REDL(Reset Event & Data Logger) SW 아키텍처를 제안한다. 제안하는 REDL은 차량용 ECU의 런타임에서 리셋 원인 분석에 필요한 정보를 선별하여 로그를 생성한다. 오버헤드가 크거나 CPU를 활용할 수 없는 순간에도 수집해야 하는 정보는 트리거를 받아 동작하는 DMA 수집 방법을 사용하여 원인 분석 시 해당 정보의 무결성을 확보한다. 이후 제어기에서 리셋이 발생했을 경우 수집된 로그를 분석하여 리셋 트리거, 트리거가 발생한 상황, 8가지로 특정한 근원적인 리셋 원인과 리셋과 가장 관련이 깊은 의심 로그를 판단하여 분석 결과 로그를 생성한다. 분석 후에는 결과 로그와 수집한 로그들을 비휘발성 메모리에 저장한다. 차량용 ECU로 많이 사용되는 Infineon의 AURIX 플랫폼에서 가상의 환경을 만들어 구현하였고, 3%정도의 CPU 오버헤드와 1% 정도의 메모리 오버헤드로 주요 기능을 수행할 수 있음을 확인하였다.

본 연구에서 제안하는 REDL은 차량용 ECU에 적용되어 예측하지 못한 리셋 발생 시 HW나 OS에서 리셋에 관해 기본적으로 제공되는 것보다 더 많은 단서를 제공하여 준다. 이를 통해 예측하지 못한 리셋의 원인을 단계적이고 신속하게 분석하여 차량용 ECU의 품질 향상에 기여한다.

Ⅱ. 관련 연구

일반적인 시스템의 디버깅을 위해 런타임에서 시스템의 정보를 남기는 기술들에 관한 여러 연구가 있다. [1,2]에서는 SW 실행 경로, 메모리 영역 접근 카운팅을 통해 메모리 누수나 실행 경로 등을 파악하는 방법이 있다. [3-5]는 코드나 데이터 주소에 원하는 정보들을 기록하는 동작을 삽입하게 하였다. [6]은 가상화를 이용한 모니터링 시스템을 제안한다. [7]에서는 비결정적인 요소들을 포함하는 SW를 결정적으로 재현시키기 위해 인풋과 이벤트 순서 등을 기록한다. 위의 방법들로 기록하는 정보들은 리셋의 원인 분석 시 직접적인 도움을 주는 정보들이 부족하고, 실시간성과 메모리 공간의 제약 등을 고려해야 하는 실시간 차량용 ECU에 그대로 적용되기는 어렵다. [8]에서는 실시간 시스템을 모니터링하기 위해 태스크의 활성화, 선점, 종료, 시스템 콜, 인터럽트 같은 이벤트 정보를 Timestamp와 PC값과 함께 기록한다. 하지만 SW 실행에 관한 정보만을 기록하기 때문에 SW만으로는 파악이 힘든 원인으로 리셋이 발생한 경우를 대비해 수집 정보에 관한 추가적인 선별 방안이 필요하다.

[9]에서는 임베디드 시스템에서 지원하는 공통적인 API에 대해 Wrapper를 호출하도록 만들어 개발언어나 OS에 의존적이지 않는 오류 로깅 및 검사 방법을 제안하였다. 해당 방법은 Wrapper를 통해 오류를 로깅하므로 공통 API와 관련 없는 원인으로 리셋이 발생할 경우 원인 추적에 큰 도움을 줄 수 없다. [10]에서는 OS나 응용SW가 비정상적으로 종료되는 순간 메모리나 레지스터의 내용을 로그로 기록하여 어떤 종류의 불량인지 판단한다. 그러나 비정상적인 종료의 유형만을 판단하기 때문에 원인 분석을 위한 정보들은 로깅하지 않아 추가적인 원인 분석이 불가능하다.

차량용 ECU에서는 에러 상황의 심각도에 따라 효율적인 핸들링을 위해 여러 종류의 리셋 유형을 가진다. 리셋 유형에 따라 초기화 범위도 다르고 발생 상황도 다르기 때문에 리셋의 원인 분석을 위해서는 리셋 유형마다 발생하는 상황에 관한 정보를 런타임에서 충분히 수집하는지 고려해야 한다. 기존의 디버깅 정보 수집 기술과 관련된 연구들은 이런 사항들이 고려되지 않았기 때문에 수집한 정보들을 통해 리셋의 원인 분석 시 리셋 유발원에 대한 직접적인 단서를 산출해내기가 힘들다. 따라서 차량용 ECU의 리셋 발생 상황에 따라 원인 분석에 도움이 되는 정보를 선별하여 수집하고, 해당 정보를 활용하여 원인을 분석하는 방안에 관한 연구가 필요하다.

Ⅲ. 차량용 ECU에서 발생하는 리셋 유형

3.1 AUTOSAR의 리셋 유형

차량용 SW의 표준화를 위해 OEM과 Supplier들이

표 1. AUTOSAR에 정의된 리셋의 종류와 설명
Table 1. Type and description of the reset defined by AUTOSAR

Reset Name	Description in AUTOSAR Documents
SW-C Reset	An SW-C is found to be faulty and is reset in order to get it back into a safe state. The reset takes place at the application level.
Application Reset	If it is not sufficient to just reset single SW-Cs, it may be necessary to restart the whole application so that it can resume its normal service. The reset affects several SW-Cs at the application level, and may involve SW-Cs at multiple ECUs.
ECU Reset	If all else fails, it may be necessary to reset the entire ECU on which the fault or error has been found. This kind of reset will affect all applications that have SW-Cs located on the ECU as well as the BSW. The reset will also likely be visible to other ECU's on the network.

모여 배포하는 AUTOSAR 표준 명세서에는 표준 플랫폼을 사용한 ECU의 에러 핸들링에 관한 내용이 명시되어 있다[11]. 제어기에서 에러가 발생했을 경우, 대체 값을 설정하는 방법 등으로 복구를 시도하게 된다. 하지만 이러한 복구 방법이 사용 불가능하고 제어기를 정상적인 상태로 돌리기 위해 시스템의 완전한 초기화가 필요한 경우가 있는데, AUTOSAR에서는 이런 상황을 단계적으로 구분하여 대응한다. 표 1은 AUTOSAR에 정의된 리셋의 유형을 나타낸다.

SW-C 리셋은 개별 SW 컴포넌트들에 문제가 발생하여 해당 컴포넌트의 초기화만으로 문제가 해결되는 경우 사용한다.

Application 리셋은 문제 해결을 위해 Application 레벨의 초기화가 필요할 때 사용되며, 여러 SW-C 리셋들을 포함한다.

ECU 리셋은 문제의 해결을 위해 전체 ECU 레벨의 리셋이 필요한 경우 사용된다. 이 리셋은 일반적으로 ECU를 초기화시키는 부팅 과정을 포함하기 때문에 치명적인 결함의 대응 방법으로 사용된다.

3.2 AUTOSAR에 대응하는 AURIX 아키텍처의 리셋 유형

Infineon의 AURIX 아키텍처는 자사의 Tricore CPU를 사용하고, 차량용 제어기를 타겟으로 개발된 32-bits 멀티코어 제품군이다. AURIX에서는 제어기 개발 시 ISO 26262의 ASIL 상위 등급 충족을 위한 Lockstep 연산, 메모리 보호, HW기반 암호화 기술인 HSM(HW Security Module) 등을 지원하기 때문에 차량용 실시간 제어기 개발 시 많이 채택된다. 본 연구에서도 AURIX 아키텍처를 사용한 제어기를 타겟으로 하며, 표 2는 AURIX에서 발생 가능한 리셋의 종류와 초기화되는 모듈을 나타낸다. 리셋 트리거는 일반적인 상황에서 발생 가능한 것들만 고려한다.

각 리셋 종류에 따라 초기화하는 모듈이 구분되고 트리거들이 다르기 때문에 AUTOSAR의 ECU 리셋을 AURIX의 Cold Power-on 리셋으로, AUTOSAR의 Application 리셋을 AURIX의 Application/System/Warm Power-on 리셋으로 대응하여 AUTOSAR 표준을 준수하는 제어기를 개발한다.

각각의 리셋 트리거들은 리셋 상황 발생 시 리셋을 담당하는 AURIX의 RCU(Reset Control Unit)에게

표 2. Infineon AURIX 아키텍처에서 발생 가능한 리셋의 종류와 초기화되는 모듈
Table 2. Types of reset and modules that are initialized in the Infineon AURIX architecture

Reset Name	Reset Trigger	Module affected by Reset
Application Reset	• SMU • ESR • SW Reset • ...	• All CPUs • All Peripherals • Port pins in reset state • Parts of SCU • RAMs (Cache Memory)
System Reset	• SMU • ESR • SW Reset • ...	• Module initialized by Application Reset • Flash Memory • XTAL/Osc/PLL • ESR pins
Warm Power-on Reset	• PORST pad asserted	• Module initialized by System Reset • JTAG Interface • OCDS/MCDS • SMU-FSP pin
Cold Power-on Reset	• Startup • EVR • SWD	• Module initialized by Warm Power-on Reset • EVR • Internal Clocks • RAMs (Scratch Pad RAM, LMU/BMU)

신호를 보낸다. 이후 RCU가 초기화 대상 모듈들에게 리셋 신호를 전달한다.

다음은 AURIX에서 리셋 트리거 별로 리셋이 발생하는 상황들에 대한 예시이다.

- SMU: ECC 메모리, 다이의 온도, WDT Overflow나 Access error 등 Infineon에서 정의한 Safety Mechanism을 위반한 경우
- SW Reset: 프로그래밍된 제어기의 로직을 위반하여 리셋 요청을 위해 코드에서 SWRSTCON 레지스터에 특정 값을 설정하는 경우
- ESR: 외부에서 ESR 포트로 신호를 보내는 경우

Warm Power-on Reset을 발생시키는 트리거의 리셋 발생 상황은 다음과 같다.

- PORST: 외부 모듈이 ECU와 주고받는 External Health Check Logic를 실패하는 경우

Cold Power-on Reset을 발생시키는 트리거의 리셋 발생 상황은 다음과 같다.

- EVR: 칩 내부의 EVR Regulator가 과전압이나 저전압을 만들어내는 경우
- SWD: 칩에 공급되는 Supply Voltage가 과전압이나 저전압일 경우

각각의 리셋은 하위 단계의 리셋을 발생시키며 최하위 단계의 리셋은 Application Reset이다. AURIX에서는 Warm Power-on 이상의 리셋에서 주로 사용되는 Scratch Pad RAM을 초기화시키도록 기본 설정이 되어있지만, PMU(Program Memory Unit)의 PROCOND 레지스터 설정을 통해 Warm Power-on Reset에서도 표와 같이 Scratch Pad RAM 영역이 초기화되지 않도록 설정할 수 있다.

모든 종류의 리셋 발생 시 RCU는 모든 CPU에게 Shutdown 요청을 보낸다[12]. 요청을 받은 뒤 CPU는 Shutdown Routine에 진입하고 Routine이 끝나면 다른 리셋 작업이 마무리될 때까지 wait 명령어를 통해 대기하게 된다. SMU의 Safety Mechanism이 위반되거나 ESR로 인해 리셋이 요청되는 경우 Shutdown Routine 도중 CPU가 NMI Trap Handler에 진입하도록 할 수 있으며, 핸들링 코드를 통해 리셋에 초기화되는 레지스터 정보를 리셋 발생 전 수집할 수 있다. SW 리셋의 경우 Wrapping하는 함수를 만들어 정보를 수집하고 리셋을 발생시킬 수 있다. 물리적으로 트리거 되는 PORST로 인해 리셋이 요청되는 경우 리셋 전 원하는 코드를 실행시킬 수 없기 때문에 리셋 발생 후 로그를 통해서만 원인을 분석한다. 모든 메모리 영역을 초기화시키는 EVR, SWD는 공급 전압에 이상이 생겨 발생하지만 리셋을 발생시킬 정도의 전압 이상은 Safety Mechanism의 이상 전압 감지 범위에 포함되어 있다. 따라서 SMU를 통해 CPU가 NMI Trap Handler에 진입할 수 있고 리셋 발생 전 비휘발성 메모리에 해당 리셋 상황을 나타내는 정보를 남겨 원인 분석에 사용한다.

Ⅳ. 리셋 원인 추적 방법

4.1 리셋 원인 추적 흐름도

그림 1은 본 연구에서 로그 분석 시 사용하는 원인

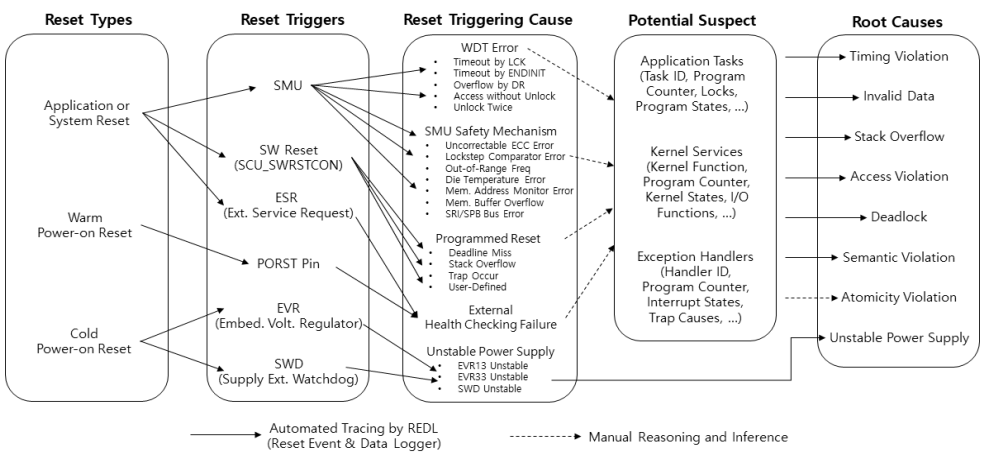


그림 1. 리셋 원인 추적 흐름도
Fig. 1. Reset Cause Tracking Flow Diagram

추적 흐름도이다. 추적 순서는 다음과 같으며 리셋 유형의 경우 리셋 트리거에 종속적이므로 분석 결과로 남기지는 않는다.

- 리셋 유형: 발생 가능한 리셋의 유형
- 리셋 트리거: 리셋을 발생시킨 트리거의 종류
- 리셋 트리거 원인: 리셋 트리거를 발생시킨 상황
- 잠재 원인: Application, Kernel, Exception 레벨에서 리셋 트리거 원인을 유발하는 잠재 요소들
- 리셋 근원: 일반적인 수준의 문제 상황을 지칭하는 근원적인 수준의 리셋 원인

4.2 리셋 근원 별 발생 사례

4.2.1 Timing Violation

제어기의 Task나 Interrupt가 실행 지연 등의 이유로 설계된 데드라인을 지키지 못했을 경우 OS에 의하여 리셋이 요청되는 상황이다. 실시간 시스템은 대부분 데드라인을 일정 수준 위반하는 경우 리셋이 발생하도록 설계되어 있다.

4.2.2 Invalid Data

특정 데이터의 값이 프로그램의 로직을 위반하여 리셋을 요청하는 상황이다. 스로틀 밸브 각도나 휠 속도 같은 중요한 데이터를 연산하기 전 검사하는 로직 등에서 리셋이 발생한다.

4.2.3 Stack Overflow

Task별 할당된 스택 이상으로 메모리를 사용하는 상황을 나타낸다. AUTOSAR에서 사용하는 OSEK OS 표준에서는 태스크 종료 후 컨텍스트 스위칭 시점에서 Stack Overflow 발생 여부를 감지한다[13].

4.2.4 Access Violation

AURIX에서 Access Violation과 관련된 Trap 발생 상황을 나타낸다. Trap 발생 시 분기하게 되는 Trap Handler에서 SW 리셋을 요청한다고 가정하며, 해당하는 Trap의 자세한 상황들은 표 3과 같다.

4.2.5 Deadlock

프로그램 실행 도중 인터럽트 비활성화, 동기화가 필요한 작업의 잘못된 사용 등으로 Deadlock이 발생하여 제어기의 External Health Check Logic에 응답을 하지 못하는 상황을 나타낸다.

4.2.6 Semantic Violation

제어 프로세서의 정해진 레지스터 설정 규칙 등을 위반하여 리셋이 발생하는 상황이다. 본 연구에서는 AURIX 플랫폼의 WDT Error로 발생하는 리셋을 타겟으로 한다. AURIX의 WDT는 일반적인 타이머 기능 외 중요한 레지스터의 접근을 관리하는 ENDINIT (End of Initialization) 기능을 제공한다. ENDINIT 기능 사용 시 WDT Control Register의 설정 순서는 다

표 3. Access Violation에 해당하는 Trap 종류와 설명
Table 3. Trap type and description of Access Violation

Reset Name	Description in AUTOSAR Documents
ALN Trap (Data Address Alignment)	• When attempting to access a memory address that is not aligned with the address Alignment Rule on the AURIX. • When using Circular Addressing of the AURIX, you attempt to access it with the size, length, etc. out of alignment.
MEM Trap (Invalid Memory Address)	• When attempting to access a segment different from the segment in the Base Address • Access through Efficient Address for AURIX address indexing spans two or more segments. • When attempting to access a CSFR address with a command other than mtcr or mfcr. • When attempting to access a non-Local DSPR range with Load or Store commands
DSE Trap (Data Access Synchronous Error)	• Load attempt outside of DSPR range • When attempting to access a specific address in segment C (C1000000 - C7FFFFFF) • The Load command causes an error in Bus. • When an error occurs from Bus during Cache refilling • When loading an invalid Overlay Address
DAE Trap (Data Access Asynchronous Error)	• The Store command causes an error in Bus. • When an error occurs from Bus during Cache writeback or Cache flush • When Store Invalid Overlay Address
MPR, MPW Trap (Memory Protection Read, Write)	• When attempting to access other than the accessible memory area set by the MPU. • When attempting to access a null address with a Load or Store command.

음과 같다.

① LCK 비트가 1이라면 0으로 변경
② LCK 비트를 1로 변경하며 ENDINIT 비트를 0으로 변경 후 중요한 레지스터 접근
③ LCK 비트가 1이라면 0으로 설정
④ LCK 비트를 1로 변경하며 ENDINIT 비트를 1로 설정

일반적인 타이머 기능은 ENDINIT 비트가 0인 상태에서 DR비트를 0으로 설정 시 작동하고 다시 위 과정을 거쳐 DR비트를 1로 변경하여 멈춘다. WDT에서는 3가지 종류의 Error로 리셋이 발생할 수 있다.

- Access Error: ①-④의 순서를 지키지 않고 WDT 설정을 시도하는 경우
- Timeout Error: ①-④의 단계별 실행 간격이 WDT에 설정된 시간 내에 이뤄지지 않는 경우
- Overflow Error: DR 비트를 0으로 설정 후 WDT에 설정된 시간 내에 1로 설정되지 못하는 경우 (일반적인 WDT Timer Overflow)

4.2.7 Atomicity Violation

프로그램이 Atomic하게 실행되어야 하지만 Atomic하게 이뤄지지 않아 리셋의 원인을 유발하는 경우이다. 일반적으로 멀티코어 환경에서 Atomic하게 동작해야 하는 모든 범위를 정확하게 파악하기가 힘들기 때문에, 해당 원인으로 SW가 예상치 못했던 동작을 할 수 있다. 그림 2는 Atomic하게 실행되어야 하는 부분을 잘못 파악하여 리셋이 유발되는 예시이다. incre_cnt()는 타이머 값을 읽고 내부 연산 후 global counter에 저장한다. ①~④에 실행되는 Critical Task는 global_cnt가 선형적으로 증가하는 경우에만 Critical Logic을 실행하며 해당 로직이 실행되지 않을 시 리셋을 발생시킨다. ①에서 Core 0이 STM 값을 읽고, ②와 ③ 시점에서 Core 1은 각각 STM 값을 읽어 증가 연산 후 global_cnt에 저장한다. ④ 시점에서 Core 0이 global_cnt 쓰기 작업을 마치면 cnt 검사 로직에 의해 리셋이 발생하게 된다.

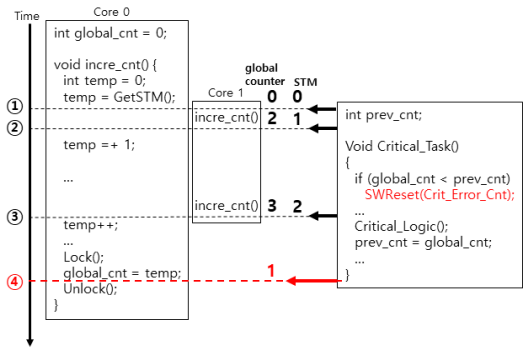


그림 2. Atomicity Violation 사례
Fig. 2. A Case of Atomicity Violation

4.2.8 Unstable Power Supply

EVR의 각 전압을 모니터링 하는 모듈(EVR13, EVR33, SWD, STBY)에서 저전압 및 과전압이 발생 시 리셋이 발생하는 상황이다. 각 모듈의 전압이 EVR13과 EVR33은 각각 기준 전압이 1.17V, 2.97V 이하이고 SWD는 외부 전압이 2.97V 이하일 경우에는 RCU에 우선적으로 리셋 요청을 보낸다.

V. REDL 설계 및 구현

그림 3은 REDL의 SW 아키텍처이다. 4개의 컴포넌트로 구성되어 있으며 각 컴포넌트 별 역할은 다음과 같다.

- Probe: Application, Kernel, HW에 삽입되어 리셋 원인 규명을 위한 정보를 수집
- Log Collector: Probe가 수집한 정보에 추가 정보를 더해 Packing하여 로그를 생성하여 Buffer들에 저장
- Log Analyzer: Global, Core Buffer들에 담긴 로그들을 분석하여 리셋 원인을 판단 후 결과를 Analysis Buffer에 저장
- Log Writer: Analysis, Core Buffer에 저장된 로그들을 EEPROM에 저장

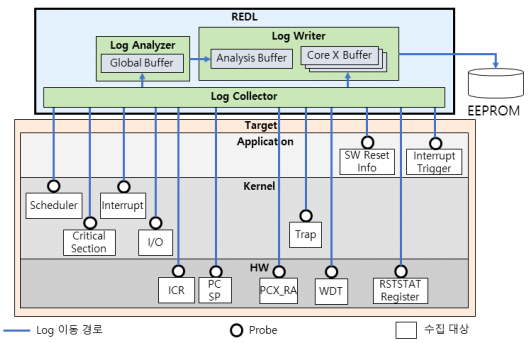


그림 3. REDL SW 아키텍처
Fig. 3. REDL SW Architecture

5.1 Probe

5.1.1 SW Reset Info Probe

SW 리셋 요청을 위해 SCU_SWRSTCON에 값을

설정하는 코드에 삽입한다. SW 리셋을 발생시킨 조건과 그 값을 수집한다. 해당 정보로 생성된 로그는 Global Buffer에 저장된다.

5.1.2 Interrupt Trigger Probe

다른 코어와 동기화가 필요한 인터럽트를 트리거하는 코드에 삽입한다. 인터럽트를 보낸 후 해당 인터럽트에 대한 수신 확인 작업이 필요한 인터럽트 등이 포함된다. 동기화가 필요한 코드는 인터럽트 비활성화 등의 상황으로 Deadlock을 유발할 수 있다. 인터럽트가 트리거되는 코어와 우선순위를 수집한다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 원인 분석 시 자원 대기 구조를 파악하여 Deadlock을 탐지하기 위해 필요하다.

5.1.3 Scheduler Probe

Task와 ISR2의 선점 및 종료 이벤트 발생 시 컨텍스트 스위칭을 실행하는 코드에 삽입한다. 로그 생성을 위해 컨텍스트 스위칭 대상 Task나 ISR2의 ID를 수집한다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 원인 분석 시 SW 실행 순서 파악을 위해 필요하다.

5.1.4 Interrupt Probe

ISR1의 선점 및 종료 이벤트 발생 시 컨텍스트 스위칭을 실행하는 코드에 삽입한다. 컨텍스트 스위칭 대상 Interrupt Priority 정보를 수집한다. 종료 시 해당 ISR1으로 인해 선점당한 컨텍스트가 Task나 ISR2일 경우 로그 생성을 위해 Task ID를 수집한다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 원인 분석 시 SW 실행 순서 파악을 위해 필요하다.

5.1.5 Trap Probe

Trap 발생 시 분기하는 Trap Handler에 삽입한다. Trap의 종류를 나타내는 Trap Class와 ID, 추가적으로 Access Violation 종류의 Trap이라면 로그 생성을 위해 접근 위발 발생 영역 등에 대한 추가적인 정보를 담는 레지스터를 수집한다. NMI Trap의 경우는 위반한 Alarm ID를 추가로 수집한다. 해당 정보로 생성된 로그는 Global Buffer에 저장된다. 원인 분석 시 트랩 발생 경위 파악을 위해 필요하다.

5.1.6 Critical Section Probe

OSEK OS에서 제공하는 Spinlock과 Resource API 코드에 삽입되어 로그 생성을 위해 Get 또는 Release 여부, Spinlock 또는 Resource의 ID를 수집한다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 원인 분석 시 공유 자원을 사용했던 현황 파악을 위해 필요하다.

5.1.7 I/O Probe

I/O 함수 호출 과정의 최상단 함수와 최하단 함수에 삽입되어 로그 생성을 위해 I/O 함수 종류, I/O 함수의 인자값을 수집한다. I/O 사용으로 인해 리셋이 유발되는 경우 수집하는 해당 정보들은 리셋이 유발된 시점에 대한 정보를 제공해 줄 수 있다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 원인 분석 시 I/O 사용 현황 파악을 위해 필요하다.

5.1.8 WDT Probe

WDT 사용을 위한 Control Register를 변경하는 코드에 삽입하여 로그 생성을 위해 레지스터에 설정하려는 값을 수집한다. 해당 정보를 통해 리셋 발생 시 런타임에서 WDT 설정이 어떤 시점에 어떻게 변경되었는지 파악할 수 있다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 원인 분석 시 WDT 사용 순서 파악을 위해 필요하다.

WDT으로 인한 에러 발생 시 SMU가 NMI Trap을 발생시키는데, 분기하는 Handler에서 발생한 Error에 관한 정보를 담고 있는 WDT Status Register 값을 수집하여 리셋 원인 분석에 사용한다. 해당 정보로 생성된 로그는 Global Buffer에 저장된다. 원인 분석 시 WDT으로 인해 어떤 에러가 발생되었는지 파악하는데 필요하다.

5.1.9 RCU Probe

리셋 원인을 판단하는 Log Analyzer 코드에 삽입하여 리셋 트리거 정보를 담고 있는 RCU의 RSTSTAT 레지스터 값을 수집하여 리셋 트리거 원인 분석에 사용한다.

5.1.10 ICR Probe

Interrupt를 비활성화 시키는 코드에 삽입하여 로그 생성을 위해 비활성화 여부 정보를 수집한다. 리셋 원인이 인터럽트 비활성화와 관련되었을 경우 해당 정보를 통해 비활성화 시점 또는 다시 활성화 되었는지 파악 가능하다. 원인 분석 시 인터럽트 비활성화로 인해 인터럽트 요청을 수행하지 못하였는지 파악하기 위해 필요하다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다.

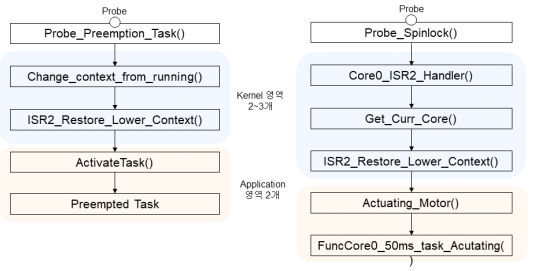


그림 4. 태스크 선점 및 스핀락 사용에서의 Return Address 정보
Fig. 4. Information of Return Address in case of Task Preemption or using Spinlock

5.1.11 PCX_RA Probe

AURIX에서는 스위칭 대상 컨텍스트를 PCX(Previou Context) Register라는 특정 레지스터에 담아 관리한다. 또한 Application에서 사용한 컨텍스트와 Kernel에서 사용한 컨텍스트를 각각 상위 컨텍스트와 하위 컨텍스트로 나누어 관리한다. 로그 생성을 위해 Probe들이 정보를 수집하는 시점에서 각 PCX의 RA(Return Address) 목록을 같이 수집한다. 본 연구에서 사용하는 Erika3 RTOS 기준으로 각 Probe마다 RA 정보를 기록 시 Application에서 사용한 상위 컨텍스트까지 도달하는데 최소 3개에서 최대 5개의 RA 정보가 포함되어 있었다. 따라서 PCX_RA Probe는 각 Probe마다 최소 3개에서 최대 5개의 RA 정보를 수집한다. 그림 4는 태스크 선점 및 스핀락 사용 시 Return Address의 정보를 나타낸다. 이를 이용하여 리셋 발생 시 각 Probe에서 기록하기 전 태스크 및 인터럽트의 선점 및 함수 호출 등 전반적인 상황을 알 수 있다. 해당 정보는 대부분의 로그에 덧붙여진다.

5.1.12 PC(Program Counter), SP(Stack Pointer)

PC는 프로그램의 거동 파악에 필수적이고, OS를 통해서는 Stack Overflow가 발생한 태스크 정보와 컨텍스트 스위칭 시점의 SP 정도만 얻을 수 있기 때문에 Stack Overflow 발생 시점 판단을 위해 주기적으로 수집된 SP는 필수적인 정보이다. REDL에서는 DMA를 통해 SP와 PC를 주기적으로 같은 시점에서 기록하기 때문에 Overflow 발생 지점 조사 시 후보가 되는 PC 범위를 제공하여 준다. 이 2가지 정보는 계속해서 변하기 때문에 빠른 주기로 수집해야 하는데, 다른 Probe들처럼 CPU를 통해 수집 시 컨텍스트 스위칭 오버헤드가 모여 제어기 성능에 영향을 미치게 된다. 또한 PC 같은 정보는 리셋을 유발하면서 코드

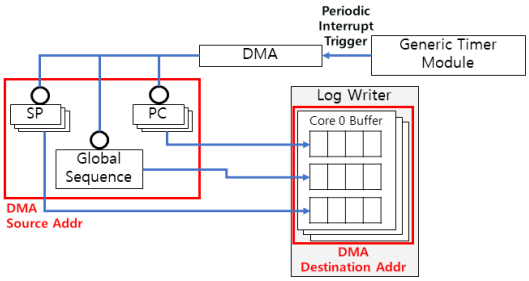


그림 5. DMA를 통한 PC와 SP 수집 방법
Fig. 5. Collecting Method of PCs and SPs through DMA

가 진행이 안 되는 Deadlock 같은 상황에서도 수집 가능해야 해당 상황에서 리셋 발생 시 원인 추정이 가능하다. 이런 한계들을 극복하기 위해 DMA를 사용하여 로그 생성을 위한 2가지 정보를 수집한다.

그림 5는 DMA를 통해 주기적으로 PC와 SP 정보를 수집하는 방법을 나타낸다. AURIX의 DMA모듈은 CPU처럼 직접 HW로부터 인터럽트를 받아 서비스 루틴을 수행할 수 있다. DMA는 타이머 모듈에게 주기적으로 인터럽트 트리거를 받아 PC와 SP, 기록 순간의 로그 순번인 Global Sequence를 각각 버퍼에 저장한다. 해당 정보로 생성된 로그는 Core 버퍼에 저장된다. 해당 정보는 기본적인 스케줄러나 인터럽트 정보 수준보다 더 상세하게 해당하는 SW의 흐름을 조사할 필요가 있는 경우 Global Sequence 기준으로 코드의 어느 부분이 실행되고 있었는지 파악하게 해준다.

5.2 Log Collector

Log Collector는 Probe가 수집한 정보를 Packing하여 로그로 생성한다. Packing 시 추가되는 정보는 표 4와 같다.

Core-specific Buffer의 로그들은 이전에 생성된 로그들을 더 작은 크기로 저장해놓기 위해 RA_CHAIN을 제외한 축약 형태로 바꾼 History Log의 형태로 저장한다. Global Buffer의 로그들은 한 번의 리셋 이벤트에 동일한 종류의 로그가 여러 번 생성되지 않기 때문에 History Log를 생성하지 않는다.

그림 6은 Log Collector가 생성하는 Log의 구조를 나타낸다. History Log는 로그 생성 빈도를 고려해 스케줄링과 인터럽트 로그는 50개, 나머지 로그는 10개를 저장한다. Global Buffer의 로그들은 이후에 Log Analyzer가 리셋 원인을 분석하는데 사용되며 EEPROM에 저장되지 않는다. SMU Alarm 로그는 NMI Trap Handler에서 생성되어 Global Buffer에 저

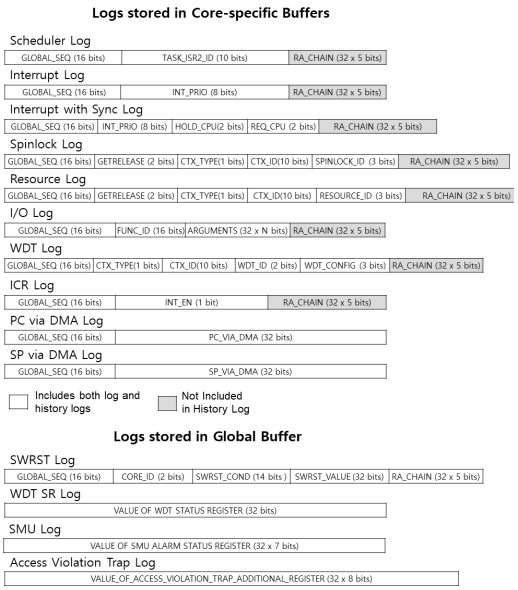


그림 6. Log Collector가 생성하는 로그의 구조
Fig. 6. The structure of the log that Log Collector generates

표 4. Log Packing 시 추가되는 정보
Table 4. Additional Data for Log Packing

Name	Description	Logs that are Packed Together	Size
Global Sequence	• Global variables that record the log generation order • Because PCs and SPs are collected by DMA, they do not increase Global Sequence when generating logs.	All Logs	2 bytes
RA_CHAIN	• Collection of call stack's return address at log generation point	All logs except PC and SP	20 bytes

장되지만 저전압 또는 과전압과 관련된 Alarm의 경우 Cold Power-on Reset을 발생시켜 해당 로그를 유실할 수 있다. 따라서 저전압 또는 과전압으로 인해 발생한 SMU Alarm 로그는 생성 즉시 EEPROM에 저장한다. AURIX에서 Cold Power-on Reset 발생 시 CPU에게 80 us 정도의 리셋 준비 시간이 있기 때문에[12] 해당 시간에 EEPROM에 8 bytes 정도의 정보를 기록할 수 있다. 기록된 저전압이나 과전압 발생 정보는 리셋 후 원인 분석에 사용된다.

5.3 Log Analyzer

Log Analyzer는 로그들을 분석하여 리셋 원인을 판단 후 결과를 Analysis Buffer에 저장한다. 리셋 트리거에 대한 정보를 담고 있는 RSTSTAT 레지스터는 Cold Power-on Reset에만 초기화되기 때문에 해당 레지스터를 먼저 검사하고, Log Collector가 생성한 로그들을 분석하여 상세한 원인을 판단한다. 그림 7은 Log Analyzer가 리셋 원인을 분석하는 로직을 나타낸다.

먼저 RSTSTAT 레지스터 값이 초기화되어 있다면, EVR이나 SWD 트리거로 인한 리셋이 발생했다고 판단한다. 해당 트리거들로 리셋이 발생했을 경우 SMU Alarm이 같이 발생하여 EEPROM에 어떤 종류의 전압 이상인지 남기기 때문에 이후 해당 정보들을 검사하여 리셋 트리거 원인과 리셋 근원을 판단한다.

RSTSTAT 레지스터 값이 SMU로 인한 리셋을 가리킬 경우 SMU 로그에서 위반한 Safety Mechanism이 무엇인지 판단한다. WDT와 관련된 위반이라면 로그를 통해 WDT 사용 순서를 검사한다. 이후 WDT SR 로그를 통해 Overflow, Timeout, Access 중 어떤 에러인지 판별하여 리셋 트리거 원인을 판단한다. 각 에러들은 특정 조건을 설정한 뒤 약속된 동작을 제 시간 내 설정하지 않기 때문에 발생하므로, 로그를 검사하여 다음 약속된 동작이 버퍼에서 발견되지 않는 로그를 찾아내어 의심 로그로 선정한다. WDT과 관련되지 않은 Safety Mechanism이 위반되었다면 위반한 상황을 담은 SMU 로그를 의심 로그로 선정한다. 리셋 근원은 모두 Semantic Violation으로 판단한다.

RSTSTAT 레지스터 값이 ESR로 인한 리셋을 가리킬 경우 설계한 ECU의 로직에 의존적으로 리셋이 발생하기 때문에 의심 로그는 선정할 수 없으며 리셋 트리거 원인은 External Health Checking Failure로 판단한다.

RSTSTAT 레지스터 값이 SW 리셋을 가리킬 경우 SW 리셋 로그를 통해 발생한 상황에 따라 리셋 트리거 원인을 판단한다. 대부분의 경우 발생 상황과 값이 적혀 있는 SW 리셋 로그를 의심 로그로 선정한다. Stack Overflow의 경우 오버플로 발생 지점에서 DMA가 수집한 PC값과 직전 주기에서 DMA가 수집한 PC값을 찾아 의심 로그로 선정한다.

RSTSTAT 레지스터 값이 PORST 리셋을 가리킬 경우 트리거 원인을 External Health Check Failure로 판단한다. 해당 상황은 SW가 정상적으로 진행되지 못해서 발생했을 가능성이 크다. 따라서 Deadlock 발생 가능성을 염두에 두고 순환 대기 상황이 발생했는

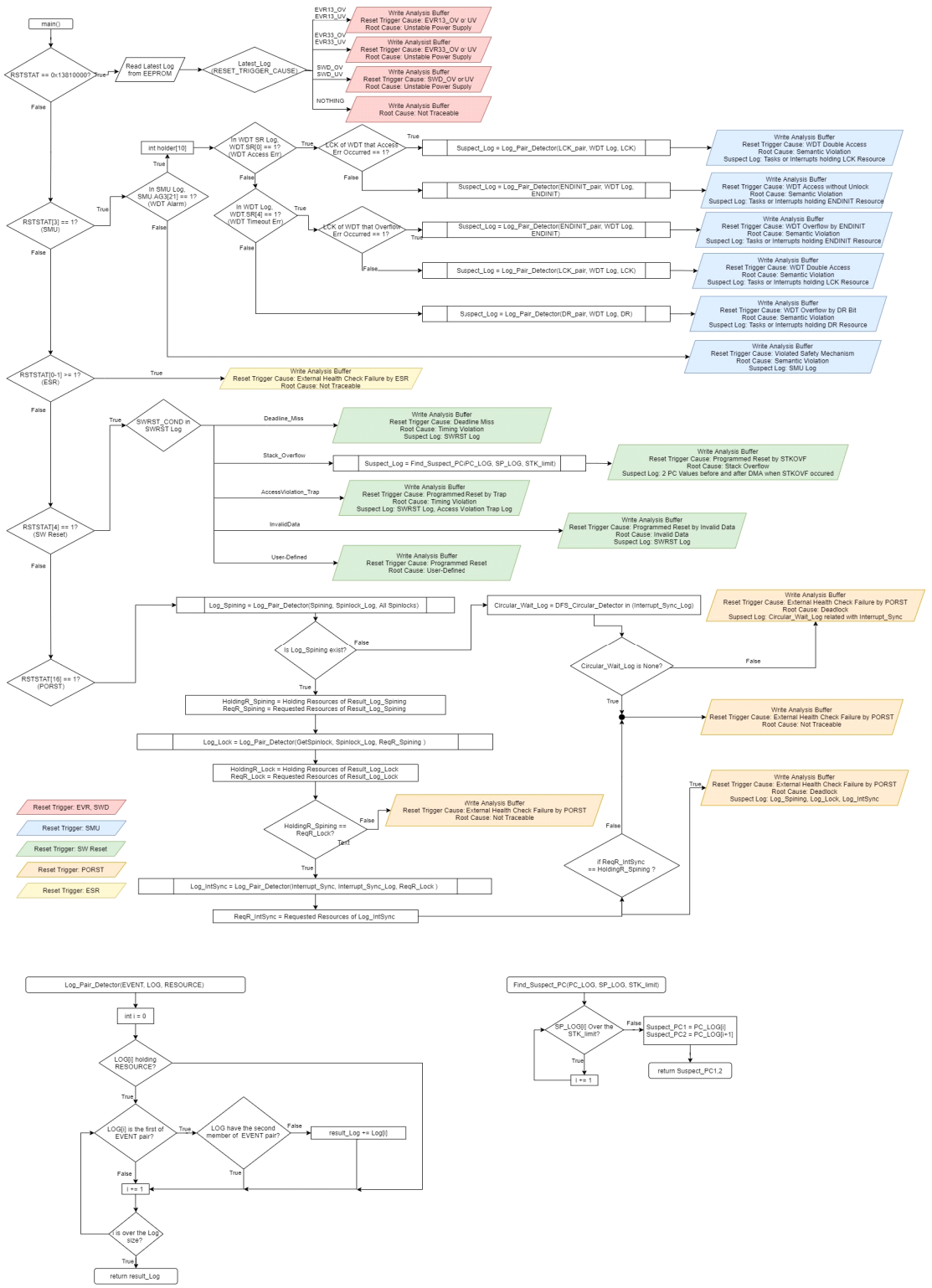


그림 7. Log Analyzer의 리셋 원인 분석 로직
Fig. 7. Reset Cause Analysis Logic of Log Analyzer

지 판단하기 위해 로그를 검사한다. OSEK에서는 동일 코어 내 프로세스 간의 공유자원은 PCP[14] 방식을 사용하기 때문에 순환 대기가 발생하지 않는다. 따라서 Spinlock과 동기화를 위해 다른 코어로부터 수신 확인 작업이 필요한 인터럽트 로그들만을 대상으로 순환 대기를 검사한다. 로그들로 자원 할당 그래프를 만들었을 때 Cycle이 존재할 경우, 리셋 근원을 Deadlock으로 판단하고 자원을 획득하지 못하고 대기 상태인 로그들을 의심 로그로 선정한다. 만약 시스템에서 Spinlock과 동기화를 요하는 인터럽트 외 자원 대기가 발생하는 요소를 사용 중이라면 해당 로그를 순환 대기 검사 목록에 추가함으로써 Deadlock을 감지할 수 있다.

Log Analyzer를 통해 리셋 근원을 판단하지 못하더라도 리셋 트리거와 리셋 트리거 원인은 항상 판단 가능하기 때문에 원인 분석의 단서를 제공할 수 있다.

5.4 Log Writer

Log Writer는 Analysis Buffer와 CPU Buffer의 로그들을 외부 저장 장치에 저장하며, 로그가 담긴 버퍼별로 헤더를 만들어 같이 저장한다. 로그 헤더 구조는 그림 8과 같으며 어떤 리셋에 관한 로그인지 분류하기 위해 Log Analyzer가 분석한 결과와 리셋 발생 후 로그의 마지막 Global Sequence를 담는다.

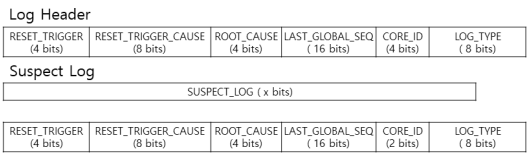


그림 8. 로그 헤더 구조
Fig. 8. The Structure of Log Header

Ⅵ. 실험 결과

표 5는 본 논문에서 가상의 ECU 실험 환경과 REDL이 런타임에서 정보를 수집하여 로그를 생성하는 오버헤드를 나타낸다. AURIX TC277에서 OSEK OS 표준을 따르는 Erika3 Enterprise OS를 사용하였다.

리셋 원인 분석의 구현은 Deadlock 상황을 꾸며 리셋을 발생시키고 분석 결과로 얻을 수 있는 로그를 통해 검증한다. 그림 9는 동기화 툴과 인터럽트 비활성화가 적절하게 사용되지 못하여 Deadlock이 발생한 사례이다. Core 0은 주기적으로 외부 MCU가 보내는 Query에 응답을 보내는 Health Check Logic을 수행

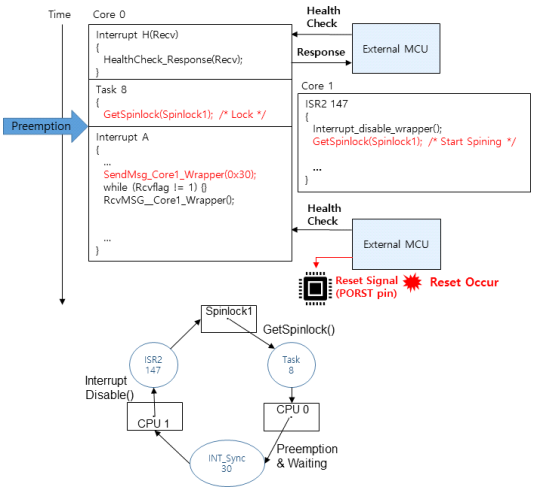


그림 9. Deadlock 발생 사례
Fig. 9. The Case of Deadlock Occurrences

표 5. ECU 실험 환경과 REDL 오버헤드
Table 5. ECU Environment and REDL Overhead

	Core 1	Core 2
Number of Task periods	9	9
Task & ISR Count	31	10
CPU Load	62%	61%
CPU Load with REDL	64.8% (+2.8%)	62.3% (+1.3%)
RAM Usage of REDL	1.992KB (1.6%)	1.992KB (1.6%)
EEPROM Usage of REDL	4KB (1%)	

한다. Task A에서 Spinlock1을 획득하고, Core 1에서 Interrupt disable 후 Spinlock1 획득을 시도한다. 이후 Core 0에서 선점이 발생하고 동기화가 필요한 인터럽트를 트리거한다. Core 1의 인터럽트가 Disable 되어 있기 때문에 해당 인터럽트에 대한 수신 응답은 받을 수 없으므로 Task A의 코드가 실행될 수 없고, 이는 Core 1에게 Spinlock1을 획득할 수 없게 하여 Deadlock이 발생한다.

해당 상황으로 PORST 리셋이 발생한 상황에서 초기화 시 Log Analyzer의 로그 분석 결과로 EEPROM에 저장된 부분은 그림 10과 같다.

로그 분석 결과로 3개의 의심 로그가 생성되었다. 각 의심 로그 앞의 8 bytes는 로그 헤더를 나타내며, 로그를 통해

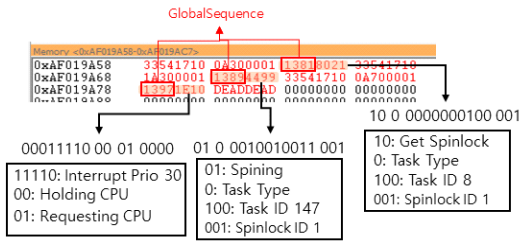


그림 10. EEPROM에 저장된 로그 분석 결과
Fig. 10. Log Analysis Results Stored on EEPROM

1. Core 0의 Task 1이 Spinlock1을 획득
2. Core 1의 Task 147이 Spinlock1 획득 요청
3. Core 0에서 Priority 30 Interrupt가 Core 1 획득 요청

의 순서로 이벤트가 발생하여 Deadlock이 발생했음을 알 수 있다.

이후 추가적인 로그 분석을 통해 Core 1에서 disable이나 Core 0에서 Preemption이 일어난 상황, 진행되지 않고 있던 PC 등을 파악할 수 있다. 이를 통해 리셋이 발생한 상황을 정확히 파악하고 재발 방지를 위한 원인 분석을 수월하게 할 수 있다.

Ⅶ. 결 론

본 논문에서는 차량용 ECU의 런타임에서 정보 수집 및 로그 생성, 생성한 로그를 통해 원인을 분석하는 리셋 원인 분석 플랫폼 REDL을 제안하였다. Infineon TC277과 AUTOSAR에서 채택한 OSEK/VDX 표준을 따르는 Erika3 RTOS를 사용하여 구현하였고 CPU 3%, 휘발성 메모리 1.6%, 비휘발성 메모리 1%정도의 Overhead로 동작이 가능함을 확인하였다.

차량용 ECU에서 발생 가능한 리셋의 원인은 매우 다양하기 때문에 예측하지 못한 리셋의 원인 분석이 제대로 되지 않을 경우 품질과 관련된 개발 비용이 증가한다. REDL은 차량용 ECU 환경에서의 리셋 원인 분석 시 기존의 디버깅 방법들에 비해 단계적이고 명료한 분석 가이드를 제시한다. 만약 리셋 근원의 정확한 판단이 불가능하더라도 리셋 트리거 원인, 로그들을 통해 대략적인 상황을 판단할 수 있다. 이러한 특징들은 수집 정보들의 초기화 여부가 고려되지 않거나 원인 분석에 불충분하게 정보들을 수집하고, 분석 범위가 한정된 기존의 디버깅 방법들에 비해 리셋과 관련된 차량용 ECU의 개발 비용을 감소시켜줄 수 있다.

차세대 차량용 ECU는 SW가 점점 복잡해지고, 차량의 ECU 간 연결성이 강해지는 추세이다. 유기적으로 동작하는 구조에서 한 ECU의 예측하지 못한 리셋은 예상치 못한 큰 문제로 번질 가능성이 높다. 따라서 ECU 리셋 현상의 원인 분석은 차량용 SW의 안전성과 관련하여 점점 더 중요한 기술이 될 것이다.

향후 과제로는 더 다양한 수준의 리셋 근원에 대해 원인 분석을 가능케 하는 연구가 필요하다. 본 연구에서는 리셋 근원을 8가지로 특정하였지만 모든 리셋을 다루기엔 가짓수가 적고, 리셋으로 이어지는 로직이 시스템 의존성이 심한 경우 로그를 분석하여 기계적으로 추적하기가 어렵다. 또한 현재는 원인을 분석하고 결과와 가장 먼저 조사해야 할 로그를 제시하여 주지만, 원인을 좀 더 면밀하게 분석하고 해당 리셋의 재발 방지 가이드까지 기계적으로 분석 가능한 연구가 이루어진다면 차량용 ECU의 리셋과 관련된 개발 비용을 획기적으로 감소시킬 수 있을 것이다.

References

[1] R. Gupta, D. A. Berson, and J. Z. Fang, "Path profile guided partial redundancy elimination using speculation," in *Proc. 1998 Int. Conf. Comput. Lang.*, pp. 230-239, Chicago, IL, USA, May 1998.

[2] T. Ball and J. R. Larus, "Efficient path profiling," in *Proc. 29th Annu. IEEE/ACM Int. Symp. Microarchitecture MICRO 29*, pp. 46-57, Paris, France, Dec. 1996.

[3] *GDB*, Retrieved Oct. 23, 2020, from http://www.gnu.org/software/gdb/

[4] *Lauterbach Trace32*, Retrieved Oct. 23, 2020, from https://www.lauterbach.com/

[5] *gprof*, Retrieved Oct. 23, 2020, from https://sourceware.org/binutils/docs/gprof/

[6] I. Han and S. Lim, "Monitoring frameworks and debugging systems for embedded systems in virtualized environments," *J. Inf. Soc. Comput.*, vol. 21, no. 12, pp. 792-797, Dec. 2015.

[7] K.-C. Tai, R. H. Carver, and E. E. Obaid, "Debugging concurrent Ada programs by deterministic execution," *IEEE Trans. Softw. Eng.*, vol. 17, no. 1, pp. 45-63, Jan. 1991.

[8] H. Tokuda, M. Kotera, and C. Mercer, "A real-time monitor for a distributed real-time operating system," in *Proc. 1988 ACM*

SIGPLAN and SIGOPS Wkshp. Parallel and Distrib. Debugging, pp. 68-77, Wisconsin, USA, May 1988.

[9] Samsung Electronics Co., Ltd., "*How to detect errors in embedded software*," Patent application number 1020060120954, Application date Dec. 01. 2006, Registration date Nov. 2008.

[10] Samsung Electronics Services Co., Ltd., "*Reset diagnostic devices and diagnostic methods of mobile telecommunication terminals*," Patent application number 1020140190251, Patent application date Dec. 12. 2014., Registration date Jan. 6. 2016.

[11] AUTOSAR GbR, *Explanation of Error Handling on Application Level V1.0.0*(2009), Retrieved Oct. 23, 2020.

[12] Infineon, *AURIX TC 27X D-Step 32-Bit Single-Chip Microcontroller User Manual V2.2*(2014), Retrieved Oct. 23, 2020.

[13] AUTOSAR GbR, *Specification of Operating System V5.3.0*(2014), Retrieved Oct. 23, 2020.

[14] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.,* vol. 39, no. 9, pp. 1175-1185, Sep. 1990.

최 재 형 (Jaehyung Choi)

2019년 2월 : 광운대학교 로봇학부 학사

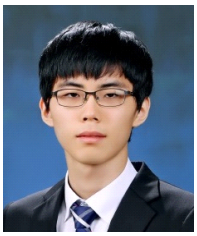
2021년 2월 : 한양대학교 자동차전자제어공학과 석사

2021년 3월~현재 : 현대오토에버 연구원

<관심분야> RTOS, 차량용 임베디드 시스템, AUTOSAR 플랫폼

[ORCID:0000-0003-1173-2787]

정 석 용 (Seokyong Jung)



2011년 2월 : 한양대학교 컴퓨터공학부 학사

2015년 3월~현재 : 한양대학교 컴퓨터·소프트웨어학과 석박사통합과정

<관심분야> 운영체제, 매니코어 확장성, 블록체인 합의 알고리즘

[ORCID:0000-0002-7227-2182]

배 재 현 (Jaehyun Bae)

2020년 2월 : 명지대학교 전자공학과 학사

2020년 3월~현재 : 한양대학교 자동차전자제어공학과 석사과정

<관심분야> RTOS, 차량용 임베디드 시스템, AUTOSAR 플랫폼

[ORCID:0000-0002-8253-4991]

유 민 수 (Minsoo Ryu)

1995년 2월 : 서울대학교 제어계측공학과 학사

1997년 2월 : 서울대학교 전기컴퓨터공학부 석사

2002년 2월 : 서울대학교 전기컴퓨터공학부 박사

2003년 3월~현재 : 한양대학교 컴퓨터소프트웨어학부 교수

<관심분야> 운영체제, 임베디드 시스템, 실시간 시스템, 소프트웨어공학

[ORCID:0000-0002-4137-3052]