

# SDV 검증을 위한 Virtual ECU 병렬 빌드 및 SILS 자동화 구축 방안

김 승 현 · 박 경 인 · 김 예 진 · 오 태 인\* · 정 지 현

에이치엘 만도, SW Beacon Lab, Platform SW Team

## A Study on Parallel Virtual ECU Build and SILS Automation for SDV Validation

Seunghyun Kim · Kyungin Park · Yejin Kim · Taein Oh\* · Jihyun Jung

Platform SW Team, SW Beacon Lab, HL Mando, 255 Pangyo-ro, Bundang-gu, Seongnam-si, Gyeonggi 17962, Korea

(Received 25 March 2026 / Revised 15 April 2026 / Accepted 29 April 2026)

**Abstract :** The transition toward the use of software-defined vehicles (SDVs) has prompted verification lead time following application software (ASW) changes as an increasingly critical issue in ECU development processes. In conventional V-model-based development, immediate functional verification after ASW modification is challenging, while Real ECU-based HIL (Hardware-in-the-Loop) testing introduces major bottlenecks in DevOps environments due to high equipment costs and physical resource constraints. To address these limitations, this paper proposed a Level 3 virtual ECU (V-ECU) environment that eliminates hardware dependency by applying a Virtual MCAL, along with a context-aware automatic patch engine and a Jenkins-based Dual-Build pipeline that enables parallel builds of Real ECUs and V-ECUs and automated execution of software-in-the-loop simulation (SILS). Experimental results using representative ASW change scenarios confirm that the V-ECU-based SILS results are functionally equivalent to those of the real ECU, while reducing verification time and effort. The study's primary objective is, through these validations, to demonstrate that the proposed V-ECU-based early verification framework reduces overall verification lead time by lowering verification effort and effectively alleviates major bottlenecks in DevOps environments, thereby providing a rapid ECU development and verification framework suitable for SDV systems.

**Key words :** Virtual ECU(가상 전자 제어 장치), SIL(Software-in-the-Loop, 소프트웨어 인 더 루프), AUTOSAR (Automotive Open System Architecture, 오토사), Virtual MCAL(Virtual Microcontroller Abstraction Layer, 가상 마이크 로컨트롤러 추상화 계층), Context-aware patching(문맥 기반 패치), Dual-build pipeline(병렬 빌드 파이프라인), Jenkins(젠킨스)

### Nomenclature

ASW : application software

AUTOSAR: automotive open system architecture

BSW : basic software

CAN : controller area network

CI : continuous integration

ECU : electronic control unit

EPC : ecu parameter configuration

FMU : functional mock-up unit

HIL : hardware-in-the-loop

MCAL : microcontroller abstraction layer

OS : operating system

RTE : run-time environment

SDV : software-defined vehicle

SILS : software-in-the-loop simulation

SWC : software component

### 1. 서론

최근 자동차 산업은 SDV (Software-Defined Vehicle)로 빠르게 전환됨에 따라, ECU (Electronic Control Unit) SW (소프트웨어)의 복잡도와 변경 빈도가 기하급수적으로

\*Corresponding author, E-mail: [taein.oh@hlcompany.com](mailto:taein.oh@hlcompany.com)

This is an Open-Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(<http://creativecommons.org/licenses/by-nc/3.0>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium provided the original work is properly cited.

증가하고 있다. OTA(Over-the-Air) 기반의 기능 업데이트 수요는 빈번해지는 반면, 하드웨어 개발과 소프트웨어 검증이 순차적으로 진행되는 전통적인 V-Model 개발 방식은 빠른 대응에 한계를 보이고 있다. 특히 물리적인 제어기(Real ECU)와 고가의 장비를 필요로 하는 HIL (Hardware-in-the-Loop) 기반 검증은 전체 개발 흐름에서 주요한 병목 요소로 작용한다.<sup>1)</sup> 높은 장비 비용, 물리적 자원의 부족, 그리고 실시간 제약으로 인해 개발자가 코드를 수정한 직후 즉각적인 피드백을 받는 ‘Shift-Left’ 검증을 수행하기에는 현실적인 어려움이 따른다.

이러한 하드웨어 종속성을 탈피하기 위해 가상 환경에서 ECU 기능을 검증할 수 있는 Virtual ECU 기반의 SILS(Software-in-the-Loop Simulation)가 대안으로 주목받고 있다.<sup>2)</sup> 특히 BSW(Basic Software)와 Virtual MCAL (Microcontroller Abstraction Layer)을 포함하는 Level 3 Virtual ECU는 실제 양산 코드에 가까운 검증 커버리지를 제공한다는 장점이 있다.<sup>3,6)</sup> 저자들의 선행 연구에서는 SbW(Steer-by-Wire) 제어기를 Level 3 수준의 Virtual ECU로 개발하고, SIL 환경에서 OS 동작, CAN 통신, 진단 통신 및 고장 검출 기능에 대한 검증을 수행하여 Virtual ECU의 신뢰성과 적용 가능성을 확인한 바 있다. 이와 유사하게 다수의 선행 연구에서도 Virtual ECU의 동작 정합성과 검증 도구로서의 활용 가능성이 보고되어 왔다.

그러나 실무적 관점에서, 이미 구축된 Virtual ECU를 지속적으로 운영하는 문제는 별도의 과제로 남아 있다. Real ECU를 위한 코드(Hardware-dependent MCAL)와 가상 환경을 위한 코드(Virtual MCAL)가 서로 다르기 때문에, 개발자는 코드가 변경될 때마다 두 가지 버전의 프로젝트를 각각 수작업으로 유지·보수해야 하는 부담을 안게 된다. 이러한 반복 작업은 변경 누락이나 휴먼 에러를 유발할 수 있으며, Virtual ECU를 구축해 두고도 CI 흐름에서 충분히 활용하지 못하게 만드는 요인이 된다. 여기에 V-Model 기반의 순차적 개발 방식에서는 ASW 변경 이후 검증까지의 리드 타임(Lead time)이 길어 개발 초기 단계에서의 빠른 피드백 확보가 어렵다는 구조적 한계까지 더해져, Virtual ECU의 실무적 활용도를 높이기 위해서는 자동화 기반의 지원 체계가 요구된다.

본 연구는 이러한 실무적 과제에 초점을 맞추어, ASW 변경 사항을 Real ECU와 Virtual ECU 프로젝트에 자동으로 반영하는 Context 기반 자동 패치 엔진과, 두 타겟을 하나의 Jenkins 파이프라인에서 동시에 빌드하는 병렬 빌드 구조를 제안한다. 이때 본 연구의 목적은 Real ECU용 MCAL과 Virtual ECU용 MCAL의 구조적 차이를 제거하거나 두 환경의 완전한 동일성을 주장하는 데 있지 않다. 이미 선행 연구를 통해 확보된 Level 3 Virtual ECU 자산

을 실제 개발 프로세스에 보다 효율적으로 편입하기 위해, 동일한 ASW 변경 사항을 양측에 일관되게 반영하는 작업을 자동화하여 검증 리드 타임과 운영 부담을 줄이는 실무형 구조를 제안하는 데 있다. 이를 검증하기 위한 실험은 대표적인 ASW 변경 시나리오를 대상으로, 제안된 자동화 구조가 기대한 기능 동작을 재현하는 방식으로 수행한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 V-Model 프로세스와 Virtual ECU 기반 검증 연구의 동향을 정리하고, 기존 Jenkins 기반 Real ECU 빌드 구조를 설명한다. 제 3장에서는 본 연구의 핵심 요소인 Context 기반 자동 패치 엔진의 설계를 중심으로, ASW 변경 사항을 소스 코드에 자동 반영하기 위한 Patch Tool의 구조와 구현 방법을 설명한다. 제 4장에서는 EPC 파일 기반의 Virtual MCAL 및 Virtual ECU 프로젝트 생성 방법과 Jenkins 환경에서 Real ECU와 Virtual ECU를 병렬로 생성하는 기법을 소개하며, SILS 검증을 자동 수행하기 위한 전체 시스템 통합 과정을 다룬다. 제 5장에서는 ASW 변경 시나리오를 기반으로 한 실험을 통해, 제안된 자동 패치 엔진과 병렬 빌드 파이프라인의 유효성을 검증하고, Real ECU와 Virtual ECU 간의 테스트를 통해 실험 결과를 분석한다. 마지막으로 제 6장에서는 연구 내용을 종합하여 결론을 도출하고, 향후 연구 방향에 대해 논의한다.

## 2. 배경 및 관련 연구

### 2.1 V-Model 기반 검증 체계

임베디드 시스템, 특히 자동차 전자제어장치(ECU)와 같은 고신뢰성 시스템의 SW 개발에는 체계적이고 구조화된 개발 방법론의 적용이 필수적이다. 이 중 V-Model은 요구사항 분석에서부터 설계, 구현 및 통합에 이르기까지 각 개발 단계와 그에 대응하는 검증 활동을 명확히 연결하는 계층적 개발 프로세스로서, 현재까지 널리 활용되고 있다.<sup>7)</sup>

V-Model은 Fig. 1 좌측의 개발 단계(요구사항 분석, 시

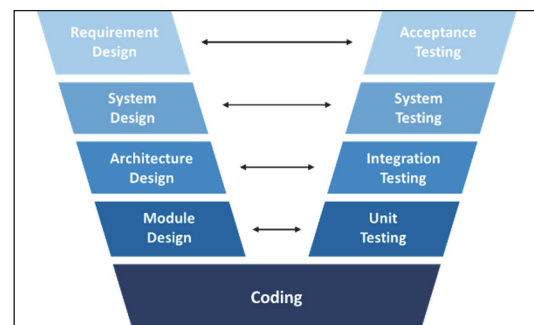


Fig. 1 V-Model development process

시스템 설계, 소프트웨어 설계, 구현 등)와 우측의 검증 단계(단위 테스트, 통합 테스트, 시스템 테스트 등)가 V자 형태로 대응되도록 구성되어 있다. 이러한 구조는 각 단계별 산출물에 대해 정형화된 검증이 가능하도록 하며, 품질 확보와 결함의 조기 식별에 유리한 구조를 제공한다.

## 2.2 Virtual ECU 기반 검증 연구

AUTOSAR(Automotive Open System Architecture) 기반 개발 환경에서는 Virtual MCAL을 포함하는 Level 3 수준의 Virtual ECU를 구성하여 기능 검증에 활용한 사례가 보고되었으며, 차량 시스템 수준에서도 통신 구조와 기능을 사전에 검토하는 수단으로 활용 가능성이 논의되어 왔다.<sup>9)10)</sup> 하지만 Virtual ECU는 Real ECU와 실행 환경이나 MCAL 구조가 다르기 때문에, 하드웨어 인터럽트나 레지스터 레벨의 동작까지 동일하게 재현하는 데에는 한계가 있다. 이러한 이유로 Virtual ECU를 Real ECU의 완벽한 대체재로 보기보다는, SW 변경 사항을 빠르게 확인하기 위한 별도의 검증 수단으로 보는 것이 적절하다.

저자들의 선행 연구에서도 Level 3 Virtual ECU를 구축해 CAN 통신 기반의 Task 주기 동작 확인과 로직 검증을 수행하여, Virtual ECU를 기능 검증 도구로서의 가능성을 확인한 바 있다.<sup>11)</sup> 다만, 이 연구는 Virtual ECU 개발과 신뢰성 확인에 목표를 두었기 때문에, 실제 개발 프로세스에서 반복적으로 발생하는 SW 변경을 자동화하고 SILS 실행까지 연계하는 운영 구조는 후속 과제로 남아 있었다.

현업에서는 SW가 수정될 때마다 Real ECU와 Virtual ECU의 프로젝트 형상을 각각 관리해야 하다 보니, 이 과정에서 업무 부담이 커지고, 변경 사항이 누락되거나 검증이 지연되는 일이 자주 발생한다. 기존 연구들 역시 가상 검증 환경의 구성 방법이나 활용 사례에 주로 초점을 맞췄을 뿐, 코드 변경을 자동으로 반영하고 빌드부터 SILS 실행까지 하나의 CI 파이프라인으로 연결하는 구조에 대해서는 구체적인 방법이 제시되지 않았다. 본 연구는 이러한 점에 착안하여, 선행 연구에서 확보한 Virtual ECU 자산을 바탕으로 Context 기반 자동 패치 기법과 Jenkins 병렬 빌드 구조를 결합하여 실무 운영 효율을 높이고자 한다.

## 2.3 기존 Jenkins 기반 Real ECU 빌드 구조

기존 ECU 개발 환경에서는 Jenkins를 활용한 CI (Continuous Integration) 빌드 파이프라인이 구성되며, ASW 및 AUTOSAR 설정 코드 변경 이후 ECU용 바이너리 파일을 생성하는 것을 목적으로 한다.<sup>12)13)</sup> 해당 파이프라인은 Fig. 2에 나타난 바와 같이 4단계로 구성되며, ECU 단일 타겟에 대한 자동화된 빌드를 수행하는 전통

적인 Jenkins 기반 구조이다.

빌드 흐름은 Developer가 ASW 및 AUTOSAR 설정 코드를 수정하여 PTC Repository에 Check-in 하는 시점부터 시작된다. 이후 빌드 요청이 발생하면 Jenkins Build Node에서 파이프라인이 실행되며, 사전에 정의된 단계에 따라 빌드 절차가 순차적으로 수행된다. Fig. 2의 1) Checkout 단계에서는 ECU 프로젝트 코드와 관련 파일을 Jenkins 작업 공간으로 가져온다. 이어서 2) Code Generation 단계에서는 EB tresos를 이용해 ARXML 기반의 MCAL 및 BSW 코드를 생성하며, 생성된 결과물은 이후 ECU 빌드를 위한 입력으로 사용된다. 다음으로 3) Build 단계에서는 Tasking Compiler를 이용해 Real ECU 빌드를 수행하고, ECU용 Hex 파일을 생성한다. 마지막으로 4) Artifact 단계에서는 생성된 바이너리 파일(Hex)과 빌드 로그를 Jenkins 산출물로 보관하여 이후 조회 및 다운로드가 가능하도록 저장한다.

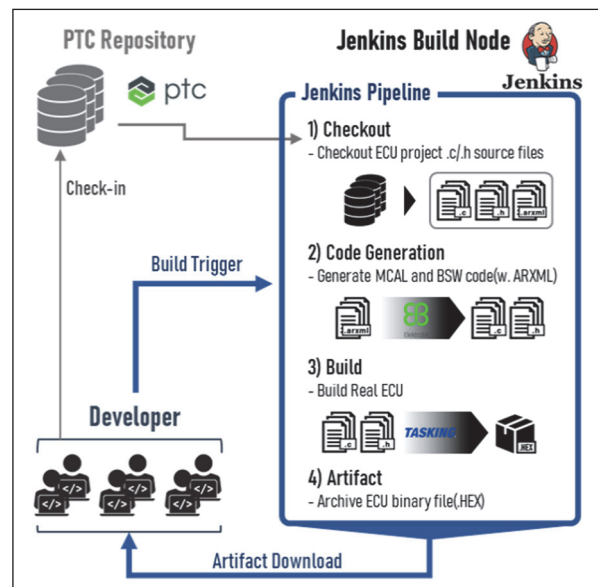


Fig. 2 Jenkins-based real ECU build process

## 3. Context 기반 자동 패치 엔진 설계 및 구현

양산 ECU SW 개발 과정에서는 요구사항 변화와 시스템 통합 조건의 복잡성으로 인해 ASW 구성 변경이 빈번히 발생한다. 변경 사항을 Real ECU와 Virtual ECU 프로젝트에 각각 수작업으로 반영하는 기존 방식은 반복 작업에 따른 비효율과 변경 누락, 휴먼 에러를 유발하여 SW 구성을 일관적으로 유지하는 데 한계가 있다. 본 연구에서는 이러한 문제점을 해결하기 위해, ASW 변경 정보를 JSON 형태의 구조화된 데이터로 관리하고, 이를 기반으로 Real ECU 및 Virtual ECU 프로젝트에 자동으로 코

드에 반영하는 Patch tool 기반 변경 반영 구조를 제안한다. 전체 과정은 ‘리소스 동기화(Resource synchronization)’, ‘코드 정제(Code pruning)’, 그리고 ‘문맥 기반 패치(Context-aware patching)’의 3단계 파이프라인으로 구성된다.

### 3.1 리소스 동기화 (Resource Synchronization)

리소스 동기화 단계는 Real ECU 프로젝트에서 생성된 최신 헤더(Include) 및 소스(Source) 파일을 Virtual ECU 타겟 디렉터리로 복사하여, 두 환경 간의 기반 코드 형상을 동일하게 유지하는 과정이다. Fig. 3은 코드 동기화를 위한 리소스 복사 절차를 보여준다. 이 단계에서는 디렉터리 내의 파일을 재귀적으로 순회하며, Real ECU 산출물을 기준으로 Virtual ECU가 동일한 기반 코드(Base code)를 공유할 수 있으며, 이후 자동 패치 적용의 일관성을 보장하고 Virtual ECU 환경에서 최신 ASW 구성을 확보하기 위한 기반을 제공한다.

| Algorithm 1 Resource Synchronization                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 1: <b>procedure</b> SYNCHRONIZERESOURCES (srcDir, buildDir) 2:   <b>for all</b> (root, dirs, files) in srcDir (recursive) <b>do</b> 3:     <b>for all</b> file in files <b>do</b> 4:       targetPath ← join(buildDir, relpath(root, file, srcDir)) 5:       create parent dir of targetPath if needed 6:       copy file to targetPath 7:     <b>end for</b> 8:   <b>end for</b> 9: <b>end procedure</b> </pre> |

Fig. 3 Pseudo-code of resource synchronization

### 3.2 코드 정제 (Artifact Pruning)

코드 정제 단계는 Fig. 4와 같은 구조로 Virtual ECU 시뮬레이션 환경에서 불필요하거나 하드웨어 종속적인 파일들을 자동으로 제거하는 과정이다. 삭제 대상은 별도의 설정 파일로 관리되며, 주석 및 공백 라인은 무시하고,

| Algorithm 2 Prune Artifacts                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 1: <b>procedure</b> PRUNEARTIFACTS (inPath, targetFolder) 2:   <b>if</b> inPath not exist <b>then return</b> 3:   <b>end if</b> 4:   absTarget ← abs path of targetFolder 5:   <b>for all</b> line in readLines(inPath) <b>do</b> 6:     skip if comment/empty; normPath ← clean(line) 7:     candidate ← join(targetFolder, normPath) 8:     <b>if</b> candidate in absTarget <b>then</b> 9:       delete file or dir if exists 10:    <b>end if</b> 11:  <b>end for</b> 12: <b>end procedure</b> </pre> |

Fig. 4 Pseudo-code of artifact pruning

디렉터리 경로나 범위를 벗어나는 경로는 자동으로 필터링되도록 구현하였다. 이를 통해 삭제 대상을 유연하게 변경할 수 있으며, SILS 환경에 필요한 최소 구성 요소를 포함함으로써 실행 환경의 경량화와 안정성을 동시에 달성할 수 있다.

### 3.3 문맥 기반 패치 (Context-aware Patching)

문맥 기반 패치(Context-aware patching) 단계는 본 Patch tool의 핵심 기능으로, ASW 변경 사항을 기술한 JSON 데이터를 입력으로 받아 지정된 코드 파일에 대해 자동 수정 작업을 수행한다. 각 JSON 규칙 파일은 Context, Delete line, Add line의 세 항목으로 구성되며, 코드 내 특정 문맥의 문자열을 기준으로 삽입 또는 삭제 위치를 식별하는 방식으로 동작한다. 구체적으로는 대상 파일에서 Context 문자열을 탐색한 후, 해당 위치를 기준으로 Delete line과 일치하는 코드가 존재할 경우 이를 제거하고, Add line을 삽입한다. 이 과정은 라인 번호에 의존하지 않고 문맥 기반 탐색을 통해 수행되기 때문에, 코드 생성 과정에서 라인 수나 위치가 변경되더라도 패치 규칙의 적용 가능성을 유지할 수 있다. 모든 수정은 변경이 실제로 발생한 경우에만 반영되며, 이를 통해 Virtual ECU 프로젝트가 최신 ASW 구성에 맞춰 자동으로 재구성된다.

| Algorithm 3 Apply Context-Aware Patch                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 1: <b>procedure</b> PATCHCONTEXT (targetRoot, ruleRoot) 2:   <b>for all</b> rulePath in ruleRoot (json files recursively) <b>do</b> 3:     targetPath ← corresponding path in targetRoot 4:     <b>if</b> targetPath not a file <b>then continue</b> 5:   <b>end if</b> 6:   rule ← load(rulePath); src ← read(targetPath) 7:   <b>if</b> context in src <b>then</b> 8:     <b>if</b> delLines match after context <b>then delete them</b> 9:   <b>end if</b> 10:    <b>if</b> addLines exist <b>then insert them after context</b> 11:  <b>end if</b> 12:    overwrite targetPath with modified src 13:  <b>end if</b> 14: <b>end for</b> 15: <b>end procedure</b> </pre> |

Fig. 5 Pseudo-code of context-aware patching

또한 Patch tool은 Fig. 5의 의사 코드와 같이 각 JSON 규칙에 대해 대상 파일 존재 여부, Context 탐색 성공 여부, Delete line 매칭 여부, Add line 삽입 여부를 로그로 기록한다. Jenkins 기반 CI 파이프라인에서는 로그를 바탕으로 Patch summary를 생성하며, 이를 통해 패치 적용 결과를 단계별로 확인할 수 있다. 필수 규칙이 적용되지 않은 경우에는 후속 Virtual ECU 빌드 또는 SILS 실행 단계에

서 이를 확인할 수 있도록 구성하였다. 이를 통해 개발자는 패치 수행 여부뿐만 아니라, 필요한 변경사항이 실제로 반영되었는지를 검토할 수 있다. 다만 본 방식은 라인 번호 변화에는 비교적 강건하지만, 코드 변경으로 인해 Context 문자열 자체가 크게 달라지는 경우에는 적용 한계가 있을 수 있다.

#### 4. DevOps 기반 병렬 빌드 파이프라인 구현

본 파이프라인은 EB tresos, dSPACE SystemDesk/ VEOS, Tasking Compiler, Jenkins를 기반으로 구성된다. EB tresos는 ARXML 기반 Real ECU용 MCAL 및 BSW code generation을 수행하고, SystemDesk는 EPC 기반 Virtual ECU 및 FMU 생성을 담당한다. Tasking Compiler는 Real ECU용 실행 파일을 생성하며, VEOS는 SILS 실행 환경을 제공한다. Jenkins는 전체 빌드 및 실행 절차를 Orchestration 한다. 본 논문에서는 이러한 도구 흐름 사이에 Resource Synchronization, Artifact Pruning, Context-aware Patching 절차를 적용하고, Real ECU와 Virtual ECU를 병렬로 처리한 뒤 SILS 실행까지 연계하는 전체 자동화 흐름을 구성하였다. 이를 통해 양산 개발 환경에서 요구되는 반복적 변경 대응과 Virtual ECU 기반 조기 검증 적용을 보다 일관된 절차로 수행할 수 있도록 하였다.

##### 4.1 EPC 기반 Virtual MCAL 및 Virtual ECU 생성

본 절에서는 dSPACE사의 SystemDesk 도구를 활용하여, 양산 ECU 프로젝트에서 생성된 AUTOSAR EPC(ECU Parameter Configuration) 파일을 기반으로 Virtual MCAL을 생성하고, Level 3 수준의 Virtual ECU를 구축하는 절차를 설명한다.

Virtual ECU 생성은 Real ECU 프로젝트의 EPC 파일을 SystemDesk에 불러오는 것으로 시작되며, 해당 정보를 기반으로 Virtual MCAL이 생성된다. 이후 ASW 및 BSW에 해당하는 소스 코드와 헤더 파일을 SystemDesk 프로젝트에 통합함으로써, Virtual ECU가 실제 양산 SW와 동일한 로직을 수행할 수 있도록 한다. 이후 PC 시뮬레이션 타겟 (예: Windows/Linux)에 맞춰 OS (Operating System) 및 TASK 스케줄링을 조정하고, 최종적으로 가상 환경에서 실행 가능한 FMU (Functional Mock-up Unit) 형태의 Virtual ECU를 생성한다. 생성된 Virtual ECU는 SILS 환경 내에서 통신 메시지 송수신, 내부 변수 모니터링 등이 가능한 독립적 시뮬레이션 단위로 활용된다. 단, 본 연구는 Virtual MCAL의 내부 생성 로직 구현보다는, 상용 도구에서 제공하는 기능을 활용해 ASW 변경 자동 반영 및 검증 자동화에 중점을 두었다.

##### 4.2 Real ECU - Virtual ECU 병렬 빌드 파이프라인

해당 연구에서는 기존 Jenkins 기반 ECU 단일 빌드 구조를 확장하여, Real ECU와 Virtual ECU를 하나의 파이프라인 내에서 병렬 처리하는 구조를 구현하였다. 제안 구조의 핵심은 동일한 코드 형상 기반에서 ECU 빌드 흐름을 유지하되, Virtual ECU 생성을 위한 별도 빌드 경로를 추가하여 두 타겟을 동시에 처리하는 데 있다. 개발자가 ASW 및 AUTOSAR 설정 코드를 PTC Repository에 Check-in 하면 Jenkins Build Node에서 파이프라인이 실행되며, Real ECU와 Virtual ECU 빌드가 분기되어 병렬로 수행된다. ECU는 기존 방식과 동일하게 ARXML 기반 MCAL·BSW·ASW를 결합하여 Hex 파일을 생성하고, Virtual ECU는 Virtual MCAL을 적용하여 SystemDesk를 통해 FMU 파일로 빌드 된다. 이처럼 하나의 빌드 파이프라인에서 Hex와 FMU 파일을 동시에 생성함으로써, 동일 변경 시점을 기준으로 Real ECU와 Virtual ECU를 병렬 처리하고 SILS 실행까지 연계하는 자동화된 검증 환경을 구성하였다. 이러한 흐름은 산업 현장에서 요구되는 빠른 변경 대응과 Virtual ECU 기반 검증 적용을 보다 효율적으로 지원할 수 있다.

##### 4.3 Virtual ECU 기반 SILS 구성 및 실행 자동화

병렬 빌드 파이프라인을 통해 Virtual ECU FMU가 생성되면, 이를 기반으로 SILS 환경이 자동 실행된다. Fig. 6은 Real ECU와 Virtual ECU의 병렬 빌드 이후 SILS 환경이 연속적으로 구성 및 실행되는 흐름을 나타낸다. Fig. 6의 하단에 도시된 SILS Environment Set-Up & Test 절차에 따라, 먼저 1) Import Virtual ECU FMU 단계에서는 FMU 파일을 dSPACE VEOS 환경에 로드하여 Virtual ECU 인스턴스를 구성한다. 이어서 2) Import Virtual Plant Models 단계에서는 Simulink 기반의 플랜트, 센서, 액추에이터 모델을 함께 Import 한다. 이후 3) Configure Virtual CAN Bus & Connect 단계에서는 DBC(Database CAN) 파일을 기반으로 Virtual CAN(Controller Area Network) Bus를 구성하고, Virtual ECU와 각 모델 간 통신 및 Signal 연결을 설정한다. 이를 통해 SILS 환경 구성이 완료되면, 4) Execute Test & Logging 단계에서 정의된 테스트 시나리오에 따라 시뮬레이션이 자동 실행되며, 이때 발생하는 로그 및 출력 Signal은 서버 환경에 저장되어 이후 동작 검증 및 분석을 위한 데이터로 활용된다. 이와 같이 병렬 빌드와 SILS 실행이 하나의 연속 흐름으로 구성됨으로써, Real ECU 및 Virtual ECU 빌드 결과가 동일한 자동화 흐름 내에서 시뮬레이션 기반 실행 단계로 연결되도록 구현하였다.

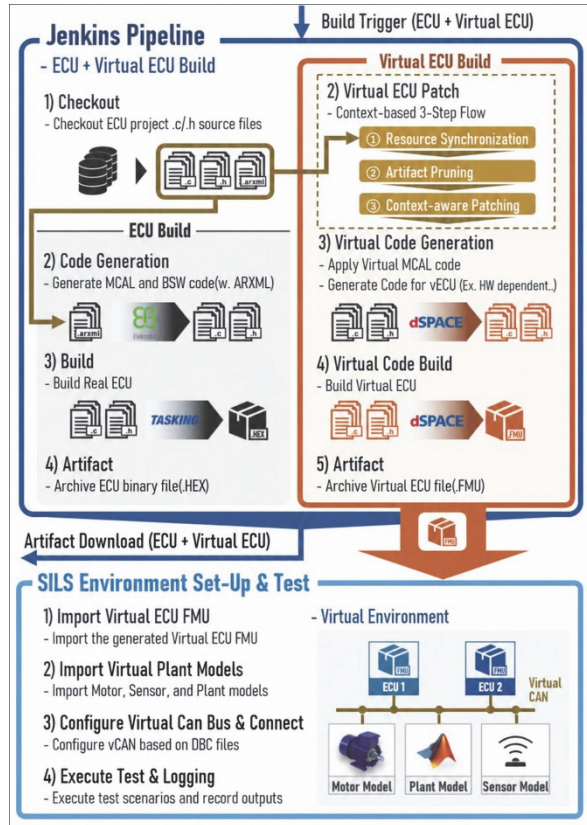


Fig. 6 Overview of dual-build and SILS automation pipeline

## 5. 실험 및 검증 결과

본 장에서는 앞서 제안한 자동화 기법이 실제 SW 개발 단계에서 자주 발생하는 사례에 대해서 유효하게 적용될 수 있는지 검증한다. 이를 위해 구축된 CI/CD 기반 자동화 환경을 활용하여, ASW 변경에 따른 자동 Patch tool을 적용하고, 병렬 빌드 파이프라인을 통해 Real ECU와 Virtual ECU를 빌드한다. 이후 실제 환경 및 가상 환경 양측에서 미리 상정한 테스트 시나리오를 적용하여, Jenkins 파이프라인에서 자동 생성된 Virtual ECU와 Real ECU SW의 동작 일치 여부를 확인하고 자동화 환경의 효과를 입증하고자 한다.

### 5.1 Test Case

검증을 위한 테스트 케이스로 SW 개발 단계에서 빈번하게 발생하는 RTE (Run-Time Environment) Port 연결 변경과 CAN 메시지 수정 사례를 선정하였다. 테스트 대상이 되는 시스템은 Redundant 구조를 가지며, 2개의 ECU와 2개의 Battery로 구성된다. 각 ECU는 서로의 정보를 포함하여 Battery 전압 정보를 Inter-ECU CAN Bus를 통해 송수신한다. Fig. 7은 해당 시스템 내 Signal 흐름을 간략화한 구조도이다.

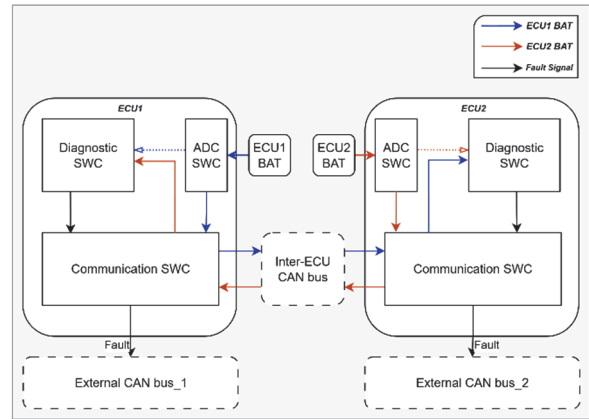


Fig. 7 System signal flow diagram

각 ECU는 자체 Battery 전압 정보를 SW ADC를 통해 측정하고, 이를 Diagnostic SWC (Software Component)에 Sender-Receiver Interface를 통해 전달한다. 본 테스트 케이스에서는 기존 RTE Port 연결을 제거하고, 상대 ECU의 Battery 전압 정보를 이용하여 고장 진단을 수행하도록 RTE Port 간 연결을 변경하였다. 또한, 고장 상태를 CAN 통신을 통해 송출하기 위해 새로운 CAN Signal인 ECUx\_Fault를 정의하였으며, 이에 따라 DBC 파일과 ASW 코드도 변경하였다.

요약하면, 각 ECU는 자신의 Battery 전압이 아닌 상대 ECU의 Battery 전압 정보를 이용해 고장 여부를 판단하고, 그 결과를 CAN Bus를 통해 전송하도록 구성된다. 테스트 시나리오에 따른 단계별 시퀀스는 Table 1과 같다.

Table 1 Test scenario sequence

| Test case: High voltage injection test |              |              |
|----------------------------------------|--------------|--------------|
| Seq                                    | ECU1_BAT (V) | ECU2_BAT (V) |
| 1                                      | 13.5         | 13.5         |
| 2                                      | 13.5         | 23.0         |

각 시퀀스에 대한 기대 동작 결과는 Table 2와 같다. 예를 들어, Sequence 2에서는 ECU2의 Battery 고장을 의도적으로 주입하고, ECU1은 ECU2의 전압 정보를 기반으로 고장을 판단하여 CAN Tx Signal로 고장 상태를 송신한다. 반면, ECU2는 ECU1의 전압 정보를 기반으로 고장

Table 2 Expected result

| Expected result: High voltage injection test |                      |                      |
|----------------------------------------------|----------------------|----------------------|
| Seq                                          | Chassis1: ECU1_Fault | Chassis2: ECU2_Fault |
| 1                                            | No Fault             | No Fault             |
| 2                                            | Fault                | No Fault             |

을 판단하기에 정상 상태로 진단하고, CAN Tx Signal로 정상을 표출한다.

추가로, Jenkins 파이프라인에서 자동화 Patch tool이 단계별로 의도한 코드 수정을 정확히 수행했는지 확인하기 위해, 실행된 패치 로그를 분석하였다.

## 5.2 Test 환경 및 구성

Real ECU SW 테스트 환경은 Photo 1과 같다. Photo 1의 ①은 테스트 대상인 ECU1 및 ECU2를 나타내며, ②와 ③은 각각의 ECU 전원 라인에 연결된 Power supply로, Battery 전압을 조절할 수 있도록 하였다. ④는 VN1640 장비로, 본 장비를 활용하여 CAN Bus 모니터링을 수행한다.

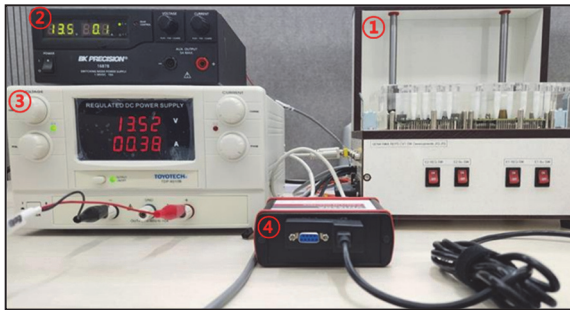


Photo. 1 Real ECU test setup

Virtual ECU SW 테스트 환경은 Fig. 8과 같이 Plant model을 통해 Virtual ECU에 Battery 값을 입력하고, Fault 발생 시 Virtual ECU가 Fault Signal을 Virtual CAN Bus를 통해 출력하도록 구현하였다. 또한, 가상의 Inter-ECU CAN Bus

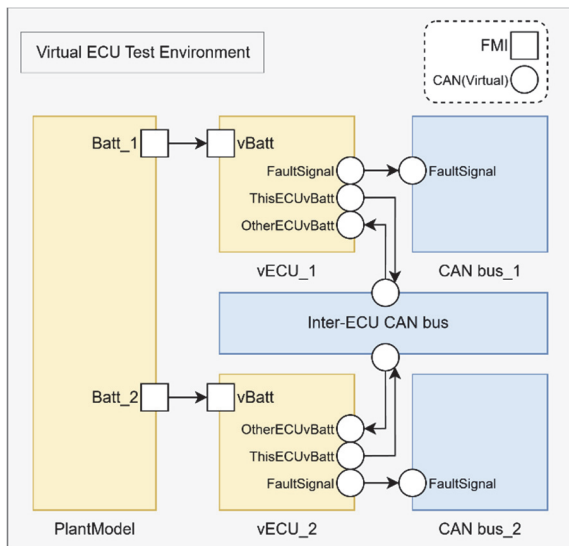


Fig. 8 Virtual ECU simulation architecture

를 통해 서로의 Battery 전압 정보를 주고 받도록 구성하였다.

## 5.3 Test 결과 및 고찰

SW 변경 사항에 대해 동일 프로젝트 형상을 기반으로 Jenkins Job을 구성하고, 자동화 Patch tool 적용 후 Real ECU와 Virtual ECU를 병렬 빌드하였다.

자동화 Patch tool 적용 결과, 리소스 동기화 단계에서는 빌드 시 생성되는 코드 675건과 프로젝트 종속 코드 125건을 포함한 총 800건의 파일이 모두 정상 동기화되었다. 이어 코드 정제 단계에서는 기 설정한 45건의 제거 대상 파일이 모두 정제되었으며, 마지막 문맥 기반 패치 단계의 경우 설정된 11개 파일에 대해 모두 패치가 적용되어 100 % 적용률을 확인하였다(Table 3).

Table 3 Patch tool application results

| Type                   | Target | Applied | Ratio |
|------------------------|--------|---------|-------|
| Resource Sync          | 800    | 800     | 100 % |
| Code pruning           | 45     | 45      | 100 % |
| Context-aware patching | 11     | 11      | 100 % |

이후, Real ECU SW를 실제 테스트 환경에서 검증하기 위해 Reprogram 하였고, Virtual ECU는 dSPACE VEOS 환경에 Import 하여 SILS 실행 환경을 구성하였다. 이후 사전에 정의한 테스트 케이스에 따라 실험을 수행하였다. 그 결과, Fig. 9와 같이 Real ECU와 Virtual ECU 모두 ECU2의 Battery 전압을 조절하였을 때 ECU1이 고장 상태로 판단하여 CAN Signal을 통해 고장을 표출하였다. 반면 ECU1의 Battery 전압은 정상 상태로 유지되었기 때문에, ECU2는 정상 상태로 판단하여 CAN Signal을 출력하지 않았다. 이는 Table 2 Expected result와 일치하는 결과이다.

추가로, 고장 진단 로직의 신뢰성 검증을 위해, Battery 전압이 22V 이상으로 500 ms 이상 유지되었을 때 고장으로 판단하는 요구 사항을 적용하였다. Fig. 9의 주요 부분을 확대한 Fig. 10을 참고하면 Real 및 Virtual ECU 모두 해당 조건에서 CAN Signal을 통해 고장이 표출되었음을 확인하였으며, 이를 통해 진단 로직이 정상적으로 동작함을 입증하였다.

이상의 실험 결과를 통해, Real ECU SW에 적용된 CAN 메시지 수정 사항과 RTE Sender-Receiver Port 연결 변경 사항이 정상적으로 반영되었음을 확인하였다. 또한 자동화 Patch tool이 의도한 대로 동작하여 Virtual ECU에도 동일한 변경 사항이 정확히 적용되었음을 확인하였다. 이후 병렬 빌드까지 성공적으로 수행되었으며, 빌드 산

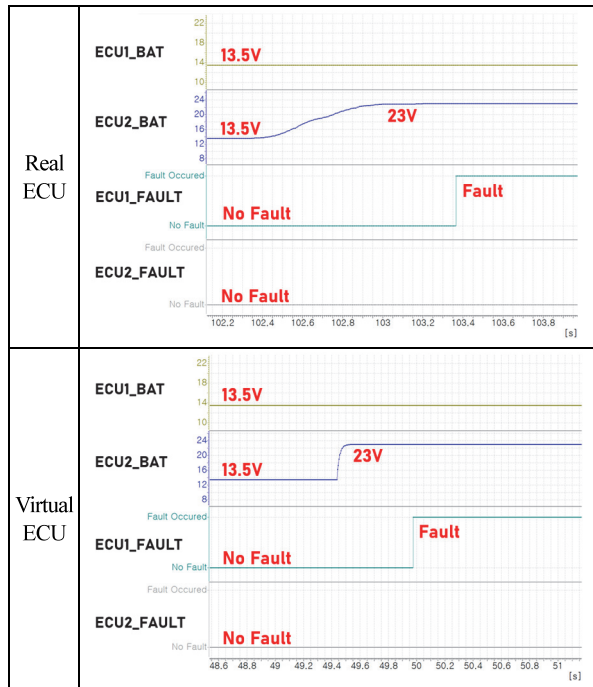


Fig. 9 Comparison of fault detection results for real and virtual ECUs

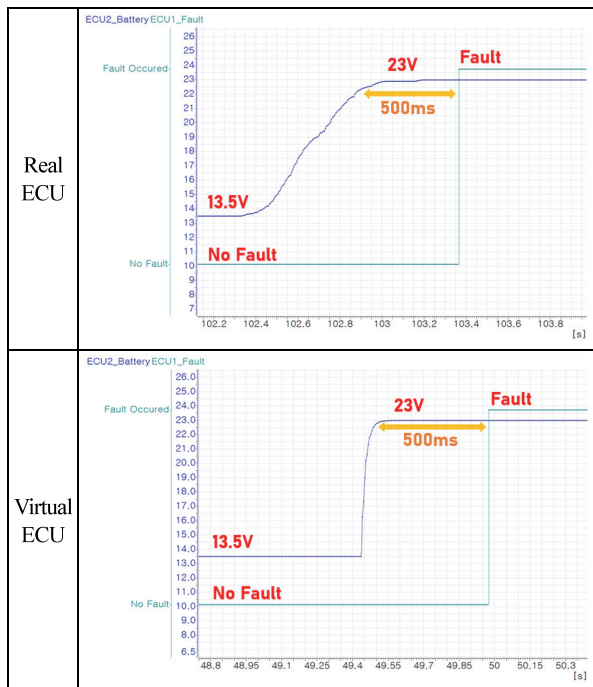


Fig. 10 Fault diagnosis under battery voltage conditions

출물을 활용한 테스트를 기반으로 Jenkins 파이프라인을 통해 생성된 Virtual ECU에 대한 동작 신뢰성과 유효성을 확인하였다.

## 6. 결론

본 연구에서는 AUTOSAR 기반 ECU SW 개발 과정에서 발생하는 ASW 변경 후 검증까지의 긴 리드 타임과 Real ECU 및 Virtual ECU 프로젝트를 각각 수작업으로 유지·관리해야 하는 실무적 부담을 경감하기 위해, Virtual ECU를 활용한 SILS 자동화 구조를 제안하였다.

이를 위해 선행 연구에서 확보한 Level 3 수준의 Virtual ECU를 바탕으로 Python 기반 자동 Patch tool을 설계하여 리소스 동기화, 코드 정제, 문맥 기반 패치를 단계별로 적용함으로써 Virtual ECU 생성을 자동화하였다. 또한 Jenkins 기반의 병렬 빌드 파이프라인을 구축하여, 동일한 소스를 기준으로 Real ECU와 Virtual ECU를 동시에 빌드할 수 있는 환경을 마련하였다. 이를 통해 Virtual ECU 측 변경 사항 반영과 SILS 실행 준비 과정을 자동화할 수 있었으며, 기존 Virtual ECU 자산을 실제 개발 흐름에 보다 효율적으로 편입할 수 있는 실무적 접근 방안을 제시하였다.

실험 결과, SW 개발 단계에서 주로 발생하는 ASW 변경 사례(RTE Port 연결 변경, CAN 메시지 내용 추가)에서 자동화 Patch tool이 정상 동작하였으며, 병렬 빌드 및 SILS 자동 실행 흐름 내에서 변경된 로직이 Real ECU와 Virtual ECU 양쪽 모두 기대한 결과로 재현됨을 확인하였다. 이는 본 연구에서 제안한 구조가 Virtual ECU 자산을 기반으로 변경 사항을 양쪽 프로젝트에 일관되게 반영하고, 검증 준비에 소요되는 수작업과 시간을 줄이는 데 실질적으로 기여할 수 있음을 보여준다. 즉, DevOps 기반의 빌드-검증 흐름을 차량용 SW 개발 환경에 적용함으로써, Real ECU 중심 검증 이전 단계에서 SW 초기 결함을 개발 단계에서 조기에 확인하고 수정할 수 있는 기반을 마련한다.

향후 연구에서는 Multi-ECU 간 협조 기반의 SILS 확장, 자동 테스트 유닛 생성 및 시나리오 기반 검증 자동화, 실차 환경과의 등가성 확보를 위한 Virtual ECU 모델의 고도화 등 다양한 방향으로의 확장이 기대된다.

## 후 기

이 연구는 2026년도 산업통상자원부 및 한국산업기술기획평가원(KEIT) 연구비 지원에 의한 연구임(RS-2024-00432265, 도로노면상태 판단을 위한 임베디드 AI 제어기 개발).

## References

- 1) M. Ruehl and F. Bronner, "Creation and Usage of Virtual Control Units (V-ECUs) in SIL and HIL for Development

- and Validation Especially for Software-Defined Vehicles (SDVs)," Stuttgart International Symposium on Automotive and Engine Technology (ISSYM 2024) Proceedings, pp.371-383, Springer Vieweg, Wiesbaden, 2024.
- 2) S. Yadav, R. T. Ugale and R. Goyal, "Development of Virtual Test Environment for Vehicle Level Simulation," 3rd Asian Conference on Innovation in Technology (ASIANCON), Ravet, India, pp.1-6, 2023.
  - 3) S. Singh, R. K. Nema, P. Swarnkar, S. K. Gautam, M. K. A. Ansar and S. Shukla, "Simulation Based Actuator Model Creation and Commissioning on Virtual Test Bed," IEEE 3rd International Conference on Electrical Power and Energy Systems (ICEPES), Bhopal, India, pp.1-6, 2024.
  - 4) G. H. Park, Y. M. Lee, J. Y. Kim and J. H. Bae, "Virtual Testing Environment with Virtual FCU," KSAE Annual Conference Proceedings, pp.1430-1435, 2023.
  - 5) K. Morishima, S. Oho and S. Shimada, "A Virtual ECU and Its Application to Control System Analysis – Power Window System Demonstration," SAE 2016-01-0022, 2016.
  - 6) A. Junghanns, J. Mauss and M. Seibt, "Faster Development of AUTOSAR Compliant ECUs through Simulation Embedded Real Time Software and Systems," ERTS2014, 2014.
  - 7) L. Pestana and B. Miranda, "A Rapid Review on Advanced Software-in-the-Loop Methods for Automotive Software Verification," SAST 2024 Proceedings, pp.1-10, 2024.
  - 8) B. Liu, H. Zhang and S. Zhu, "An Incremental V-Model Process for Automotive Development," 23rd Asia-Pacific Software Engineering Conference (APSEC), pp.225-232, 2016.
  - 9) K. J. Yong, S. T. Seon, L. I. Ho, H. W. Jin, S. W. Hyun, K. Hyeong and C. Jung, "Virtual ECU Supporting for CAN-FD: Implementation and Integration of MCP2517FD," Proceedings of Symposium of the Korean Institute of Communications and Information Sciences, pp.376-378, 2023.
  - 10) K. J. Yong, S. W. Hyun, H. W. Jin, Y. Anna and C. Jung, "Measurement of vCAN-FD Network Timing Accuracy for QEMU-Based Virtual ECUs," Proceedings of Symposium of the Korean Institute of Communications and Information Sciences, pp.408-409, 2024.
  - 11) S. Kim, K. Park, B. Yoon, Y. Kim, J. Jung and T. Oh, "Research on Virtual ECU Verification Methods for Steer-by-Wire," Transactions of KSAE, Vol.33, No.5, pp.335-342, 2025.
  - 12) J. Won, J. Seok, S. Lee and J. Yoo, "Software-in-the-Loop Simulation Implementation Methodology for Autonomous Vehicle System Development," Transactions of KSAE, Vol.33, No.11, pp.967-977, 2025.
  - 13) M. R. Kabir and S. Ray, "Virtualization for Automotive Safety and Security Exploration," IEEE 16th Dallas Circuits and Systems Conference (DCAS), pp.1-4, 2023.