組込ソフトウェア向けインターフェース シミュレータ（SILS）による開発効率の向上

Improvement of development efficiency by embedded software for interface simulator (SILS)

京谷 智哉* 中越 敏典*
Tomoya Kyotani, Toshinori Nakagoshi

昨今、車載向け組込ソフトウェアは、EV（Electric Vehicle）、ADAS（Advanced Driver Assistance System）開発の盛り上がりによって、ますます複雑化・大規模化するとともに、機能安全要求の高まりによる高品質化、及び、開発競争による更なる開発期間短縮が求められている。

当社もそのような環境の中にあるが、対応策の１つとして PC 上で動作する SILS（Software In the Loop Simulation）に分類されるシミュレータを開発し、開発効率を飛躍的に向上させることができた。

本稿では、このシミュレータを開発するに至った背景、シミュレータの概要、及び導入効果について紹介する。

Recently automotive embedded software is increasingly complicated and large scale due to the excitement of EV(Electric Vehicle) and ADAS(Advanced Driver Assistance System) development. In addition, it is required both higher quality by increasing functional safety requirement and further shortening of development period by development competition.

Although we are also in such an environment, as one of the countermeasures, we developed a simulator classified as SILS (Software In the Loop Simulation) that runs on PC, and improved the development efficiency dramatically.

In this paper, we introduce the background of the development of this simulator, the outline of the simulator, and the introduction effects.

1. まえがき

車載向け組込ソフトウェアの開発は、車両メーカー／車載組込機器ベンダーからの要求による車両機能改善・機能追加に伴う仕様変更や、複数機種への機能水平展開による機能移植、車載組込機器ベンダーからの要求によるコストダウン、及び性能改善に伴うハードウェア回路変更による仕様変更等の様々な開発要件に対応してソフトウェアを変更する内容となる。

開発の流れとしては、これら複数の開発要件に対応したソフトウェア変更を車両開発における試作車イベントごとの要求に合わせて順々に行い、複数回のリリースを経て開発完了へ向かうという形が一般的である。特に、車両と密な関係を持つ ECU（Electronic Control Unit：電子制御ユニット）に関しては、実車評価結果に応じてソフトウェア変更を行う場面が増え、短期間に複数回のリリースを行う状況が発生しやすい。

ソフトウェア開発を行うために必要な環境としては、大きく、設計ツール（文書作成ツール、モデリングツール等）、ソフトウェア開発環境（コンパイラ）、テスト環境が挙げられる。設計ツールやソフトウェア開発環境に関しては、よほど特殊な場合を除いて汎用 PC 上で動作するツールを使用するが、テスト環境に関しては、開発したソフトウェアを実行する環境に依存する。

車載向け組込ソフトウェア開発においては、主となるテスト環境が開発対象ハードウェアの実機環境であり、これが唯一のテスト環境となる場合が多い。

このような状況の中、数年前までは、少数の開発担当者が「設計⇒実装⇒テスト」のサイクルを短期間で回しながら開発を進めることができた。

しかしながら、昨今の先進運転支援システムをはじめとする車両電装化技術の急速な発展により、機能安全に対応した高い品質が求められる機能の増加、マイコンの高性能化に伴い可能となった複雑・大規模な機能要件の

＊関西事業部 第六技術部

実装、グローバル展開の流れからの機種数の増加、及び競争激化によるリードタイムの圧縮等々への対応が求められ、少なくとも、今までの開発体制では、これらに対応することが難しくなっている。対応策として開発要員を増やし、開発サイクルを複数の開発担当者が並行して回すも、品質・コスト・工期に関する多くの問題を抱えている。

当社では、車載向け組込ソフトウェア開発の中でもアプリケーションソフトウェアを中心とした受託開発を行っているが、その中で、これらの問題解決策の１つとしてシミュレータの開発を行った。

2. インターフェースシミュレータ開発の背景

当社が車載向け組込ソフトウェア開発の受託を開始した当時は、まだ機種数が少ないうえ、開発規模も小さく、少人数での開発が主であったが、委託元での開発機種のグローバル展開が一気に加速し、機種数が増え始め、開発規模が増大すると、少人数体制では全ての開発に対応するのが難しい状況となった。

そこで、開発要員の増員を行い、開発体制強化を図ったが、複数の担当者が並行して開発を進める中で、テスト環境に起因する表１に示す問題点が浮上してきた。

以降に、これらの問題点の原因を示す。

2.1　問題点の原因

2.1.1　設計・実装完了後、すぐにテスト開始できない

この問題は、ソフトウェアのテスト環境として使用するハードウェア実機環境の不足が主な原因であり、以下の状況で発生した。

(1)　開発対象のハードウェアの不足

新システム開発初期や、原低対応でのハードウェア構成変更の試作段階では、開発対象ハードウェアの製造数が限られるため、構築可能なハードウェア実機環境の数が限られる。

(2)　ハードウェア実機環境で必須の測定機器の不足

機種展開が増え、複数機種の開発が並行して行われる状況が増えると、車載向け組込システムの評価で一般的に使用する測定機器も不足する。

これらの測定機器の中でも特に必須のものは価格が高く、費用の面から容易に台数を増やすことが難しい。

(3)　ハードウェア実機環境の使用可能時期、期間の制限

ハードウェア実機環境は当社が主に受託しているアプリケーションソフトウェア開発だけでなく、ハードウェアドライバソフトウェア開発、システムテストの検証でも使用するため、各担当者の使用希望時期が重なり、使用可能時期、期間が制限される状況が発生しやすい。

表１　テスト環境に起因する問題点

問題点	影響を受けるプロジェクト管理項目		
	品質	コスト	工期
設計・実装完了後、すぐにテスト開始できない		○	○
客先試験場への移動（出張）が必要		○	
デバッグが容易に行えない	○	○	○
フォールト注入テスト実施のしきいが高い	○	○	
統合検証・回帰テストの網羅不足（試験実施に時間がかかり、短期間で頻繁にリリースが必要な場合に発生）	○	○	○

2.1.2　客先試験場への移動（出張）が必要

前項2.1.1で述べた原因により、ハードウェア実機環境の使用が客先試験場に制限される場合に、テスト業務のため、頻繁に客先試験場への移動が必要となった。

特に当社の場合、客先が遠隔地であったこともあり、テスト業務における客先への移動時間が大きなウェイトを占めることになり、開発コストに影響を与えることとなった。

2.1.3　デバッグが容易に行えない

ソフトウェア開発においてデバッグは重要な作業項目であるが、以下のようなデバッグが容易に行えない状況が発生した。

(1)　ハードウェア実機環境の稼働率が高い

前項で述べた原因から、ハードウェア実機環境の稼働率が高い状況が発生しやすく、デバッグ実施に当たっては実機環境を使用するための調整が必要となることや、使用できる期間も制限されることが多く、気軽にデバッグできない。

(2)　デバッグの準備に時間がかかる

ハードウェア実機環境でのデバッグにおいては、マイコン端子へのデバッグ用モニタ出力を測定器でモニタできるようにする等、ハードウェア回路への細工が必要な場合が多い。

また、バグ改修やデバッグコードの挿入等によるソフトウェア変更の度に、実機へ書き込むオブジェクトの生成を行うためのコンパイルを実施しなければならないが、機種によっては、このコンパイルに非常に時間がかかるものがあり、デバッグを長期化させる要因となった。

(3)　ソフトウェアデバッグが難しい

ソフトウェアデバッグにおいては、ソフトウェアを実行停止→変数値を操作（動作条件変更）→実行再開の流れで操作したい場面が多いが、ハードウェア実機環境においては、デバッガ等を用いてソフトウェアの実行を停止させたとしてもハードウェア回路は動作し続ける

ため、ソフトウェアの実行を再開させた際に、想定しない状態になる場合が多い。そのため、デバッグを行うためのハードウェア知識が必要であり、経験の浅い担当者であると、デバッグを実施すること自体が難しい。

2.1.4　フォールト注入テスト実施のしきいが高い

ハードウェア実機環境でのデバッグにおいてフォールト注入テストを行う場合には、ハードウェア回路の改造を必要とする、又は特殊な治具を製作する必要がある。そのため、テスト実施に当たっては、相応のハードウェア知識を持つ担当者が必要であり、準備にも時間がかかる。

また、誤操作によりハードウェアを故障させてしまう場合もある。

2.1.5　統合検証・回帰テストの網羅不足

ハードウェア実機環境上でのテストにおいては、テスト項目に応じた試験系の準備が必要であり、１つのテスト項目を実施するために相応の時間がかかる。

そのため、短期間で頻繁にリリースを繰り返す開発においては、時間的制約からリリースまでに実施する回帰テスト項目を優先順位に応じて絞って対応せざるを得ない場面があり、網羅範囲が不十分であることが原因で、後工程でバグが見つかる等、問題発覚が遅れるケースがあった。

2.2　問題点への対策

前節で挙げたテスト環境に起因する問題の原因は、主として活用できるテスト環境がハードウェア実機環境のみであることによる。

当社が受託開発する車載向け組込ソフトウェアの開発対象は、ハードウェア回路構成への依存度が少ないアプリケーションに該当する部分が大半を占めており、ソフトウェアのデバッグ、テストにおいてハードウェアとの結合が必要な項目が少なく、必ずしもハードウェア実機環境を必要としない内容が多い。

また、ソフトウェア開発においては、昨今の開発規模増加傾向の中で、開発期間の確保も重要となってきている。そのため、システム開発初期段階でハードウェア開発が並行に進んでいる状況において、ハードウェア実機環境の使用に制約がある場合でも、ソフトウェアロジックに関するデバッグ・評価を、品質を確保しつつ効率よく進めたい状況にある。

これらを踏まえて、ハードウェア回路構成に依存しないミドルウェアやアプリケーション等のソフトウェア（以降、本稿では、これらを総称したものをアプリケーションと呼ぶ）をデバッグ・評価することを主目的とした、専用ハードウェアを必要としない汎用 PC 上で動作するシミュレータを構築し、ソフトウェア開発に適用することが、車載向け組込ソフトウェア開発における問題に対する解決策の１つになると考えた。

3. インターフェースシミュレータの概要

本章では、開発したシミュレータで採用したシミュレーション手法及びシミュレータの適用範囲について述べる。

3.1　シミュレーション手法

今回、シミュレータを構築するために採用したシミュレーション手法は SILS と呼ばれるものである。

以降に一般的なシミュレーション手法、及び手法として SILS を選択した理由を述べる。

3.1.1　一般的なシミュレーション手法

特殊ハードウェアを使用せず汎用 PC 上で動作させるシミュレータを構築するためのシミュレーション手法としては、一般的に、大きく以下の３つがある。

(1)　MILS　：Model In the Loop Simulation
(2)　SILS　：Software In the Loop Simulation
(3)　SPILS　：Simulator based Processor In the Loop Simulation

MILSは、モデル記述された制御モデルをテスト対象とし、制御モデルの処理内容の妥当性の検証を行うために用いる手法である。主にシステム開発の上流設計の検証で用いられる。

SILS は、C 言語等で記述されたソースコードをテスト対象とし、ソースコードのロジック検証を行うために用いる手法である。主にソフトウェア開発フェーズでの検証で用いられる。

SPILS は、実マイコンに書き込み動作させるものと同じオブジェクトコードをテスト対象とし、命令セットシミュレータ上で実マイコン上に書き込む全てのプログラムを動作させ、実動作と等価な検証を行うための手法である。SILS と同様にソフトウェア開発フェーズでの検証に用いられるが、性能評価に関する検証でも用いることが可能な点が異なる。

これらのシミュレーション手法の構成概要を表２に示す。

3.1.2　SILS 手法を選択した理由

今回、シミュレータを構築する際の手法として SILS を採用した理由は以下によるものである。

(1)　ソースコードでのロジック検証が必須

車載向け組込ソフトウェア開発では、ソースコードで

記述されたレガシーコードをベースに機能変更・追加を行う派生開発が中心であり、ソースコードのロジック検証が必須である。

また、昨今、モデルベース開発によりモデル記述を使用してソフトウェアの開発を進める場面が増えつつあるが、その多くは、モデル記述から生成したソースコードをレガシーコード上に組み込み、デバッグ、評価を行う必要があり、その場合もソースコードのロジック検証が必須となる。

(2)　導入コストを抑えることができる

今回の問題の解決に当たっては、複数の担当者が並行して開発を進められることが前提となり、テスト環境を各担当者に割り当てることを考慮する必要がある。この場合、テスト環境の導入・維持に必要なコストが重要になる。

MILS の場合はモデル記述及びその検証を行うためのモデリングツールが、SPILS の場合はマイコン対応の専用命令セットシミュレータがそれぞれ必要となるが、どちらも初期導入費、維持費が高額なものが多く、かつ使用するユーザごとにライセンス購入等が必要になる場合が大半であり、必要数に応じた導入コストを要する。

SILS に関しては、手法実現に当たって安価な汎用ソフトウェアを用いて構築することができ、大幅な追加コストをかけずに、必要な数のテスト環境を立ち上げることが可能である。

(3)　シミュレータ変更の頻度を抑えることができる

SILS と SPILS を比べると、マイコン上で組み込む全てのプログラムを動作させることができ、性能を含めた多くの検証が可能な SPILS が魅力的に見える。

しかし、SPILS では、マイコン仕様に依存した命令セットシミュレータを使用する必要があること、シミュレータ内に組み込む電子回路シミュレータをマイコン仕様に従って構築する必要があることから、シミュレータ自体がマイコンのハードウェア回路構成に密に依存する構成となる。

そのため、マイコン及びその周辺回路の仕様変更に伴いシミュレータを変更しなければならない。特に、マイコンを全くシリーズの異なるものに置き換える場合は、大規模な変更が必要となる。

アプリケーションのロジック検証を目的とした場合、アプリケーション自体がマイコン仕様に依存することがほとんど無いため、マイコン仕様の変更によってシミュレータの変更が発生する状況は避けたい。

SILS では、プラントモデル内に組み込む電子回路シミュレータ、制御対象シミュレータの構成に関しては、特にハードウェア回路構成に依存しなければならない等の制約が無く、シミュレータ構築時にハードウェア回路構成への依存度を調整することができ、シミュレータ変更の頻度を少なくすることが可能である。

表２　シミュレーション手法と構成概要

手法	構成概要
MILS	テスト対象（モデル記述）／制御対象シミュレータ：制御対象モデル
SILS	テスト対象（ソースコード）／電子回路シミュレータ：回路IFモデル／制御対象シミュレータ：制御対象モデル
SPILS	命令セットシミュレータ：テスト対象（オブジェクトコード）／電子回路シミュレータ：メモリマップIFモデル・回路IFモデル／制御対象シミュレータ：制御対象モデル

※ 命令セットシミュレータ、メモリマップ IF モデル、回路 IF モデルの一部は、使用するマイコン仕様に従ったものとする必要がある。

3.2　シミュレータ適用範囲

想定しているシミュレータの適用範囲を図１に示す。

シミュレータの主な適用範囲は、ソフトウェア開発 V 字モデルの右側、具体的には、検証フェーズでのソフトウェア結合テスト、ソフトウェアテストに関するソースコードレベルでのロジック検証及びソースコード対象のデバッグである。

また、ソフトウェア開発 V 字モデルの右側だけでなく、左側の設計フェーズでのソフトウェア設計、及びソフトウェア構築におけるコーディング等においても、設計変更における影響分析や、コーディング時のソースコード ロジックの事前検証等での活用が可能である。

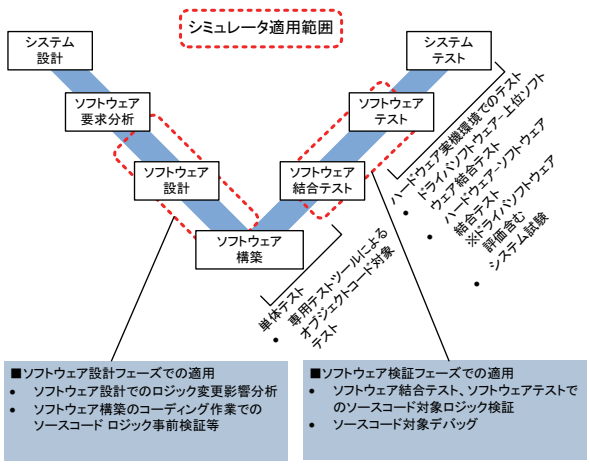

図１　シミュレータ適用範囲

機能安全対応の要求がある場合、オブジェクトコードでのテストが求められるが、オブジェクトコードを対象とした単体テストでソースコードとオブジェクトコードの動作一致を確認できていれば、アプリケーションのロジック検証に関しては、シミュレータ上での評価結果をハードウェア実機環境上での評価と同等に扱うことも可能と考えており、機能安全対応においてもシミュレータの活用が期待できる。

4. インターフェースシミュレータの特徴

4.1　シミュレータの特徴

インターフェースシミュレータの特徴を以下にまとめる。

(1)　テスト対象

車載向け組込ソフトウェア内のハードウェア回路構成への依存度が少ないアプリケーションをテスト対象とする。

(2)　簡易モデルを組み込んだシミュレータ

一般的な SILS では、厳密にハードウェア動作やプラント動作を模擬するモデルを組み込むのに対して、インターフェースシミュレータでは、テスト対象ソフトウェアのインターフェース観点で、ソフトウェアのロジック検証を行う上で最低限必要な論理的な振る舞いを模擬する簡易モデルを組み込む。

本シミュレータを“インターフェースシミュレータ”と呼ぶ理由もこの特徴によるものである。

また、この特徴から、機能的には同じであるがハードウェア回路構成が異なる他機種へのシミュレータ流用が行いやすい。

(3)　シミュレータの構築が容易

インターフェースシミュレータは、シミュレータ構築で必須の以下の作業を容易に行うための仕組みを持つ。

(a)　テスト対象とシミュレータ間のインターフェース接続

(b)　シミュレータ内へのモデル組み込み

これらにより、シミュレータ導入までの期間を短縮する。

(4)　同じテスト項目を同条件で何度でも実施可能

シミュレータの振る舞いを制御するためのシナリオを記述したファイルを作成し、シミュレータに読み込ませることで、シナリオに従ったテストを実行することができる。

これにより、同じテスト項目を同条件で何度でも実施することが可能である。

(5)　テスト結果をファイル出力可能

テスト対象内の変数値や、モデルの動作状態をモニタファイルとして出力することが可能である。

このファイルを利用することでテスト結果のエビデンス作成を効率的に行える。

(6)　導入のしきいが低い

Windows 搭載の汎用 PC 上で動作するシミュレータであるため、特殊な環境が必要ではない。

また、必要なツールが特殊用途の専用ツールでは無く、一般的にソフトウェア開発で使用されるソフトウェア統合開発環境の Visual Studio のみであるため、導入しやすい。

4.2　シミュレータの制約

(1)　割込み処理等のイベント処理に制約がある

現状のインターフェースシミュレータでは、テスト対象が持つ定周期処理、割り込み処理等のイベント処理を順番に実行する仕様であるため、実環境で発生する周期処理実行途中での割込み処理実行を行うことができない。

よって、イベント処理の実行順に関して、ハードウェア実機環境と異なることを考慮してテストを実施する必要がある。

4.3　シミュレータの構成

本節では、シミュレータの構成について説明する。

インターフェースシミュレータの構成概要を図２に示す。

(1)　プラットフォーム

シミュレータの中核となる部分であり、テスト対象及びシミュレータ内の各機能の実行制御を行う。

(2)　インターフェースアタッチメント

シミュレータとテスト対象間のインターフェースを

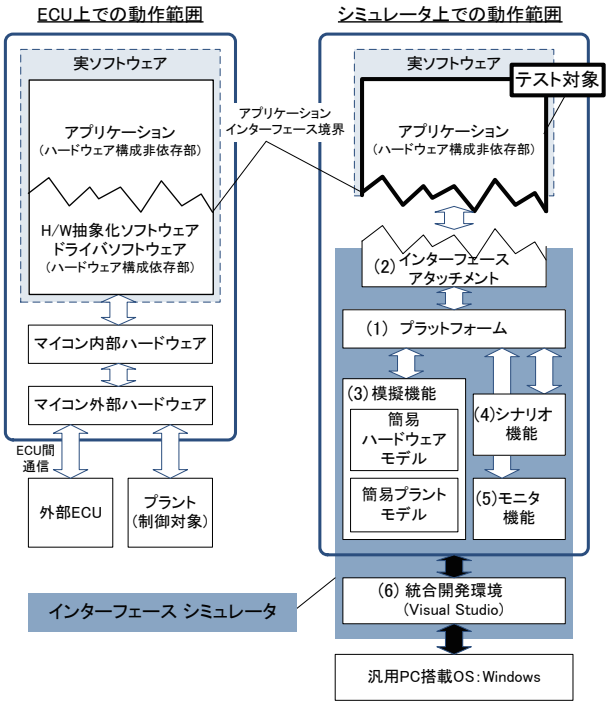

図２　インターフェースシミュレータ構成概要

接続する。また、テスト対象とのインターフェース差異を吸収する役割を持つ。

(3)　模擬機能

簡易ハードウェアモデル及び簡易プラントモデルが組み込まれ、インターフェースアタッチメントを介して、テスト対象のインターフェースに対する振る舞い模擬を行う機能である。

また、シミュレータ構築の際にモデルを容易に組み込むための仕組みを持つ。

(4)　シナリオ機能

ファイルで入力されるシナリオ記述に従い、模擬機能内のモデルの振る舞いを制御する。

(5)　モニタ機能

テスト対象内の変数やシミュレータ内モデル内の動作状態をモニタする。モニタ結果をファイルとして出力する。

(6)　統合開発環境

Windows のソフトウェア統合開発環境である Visual Studio 上で、テスト対象、及びシミュレータを動作させ、デバッグ機能として Visual Studio を利用する。

4.4　シミュレータの用途

インターフェースシミュレータの用途を以下にまとめる。

(1)　ソフトウェア結合テスト

アプリケーション内のモジュール結合テスト、機能結合テスト等、ソフトウェアの組合せに関するロジック検証で使用する。

ただし、4.2 節で述べた制約があるため、複数の周期処理間、割り込み処理間でのインターフェース確認を行う必要があるテスト項目については、特定条件下という制約付きで検証を行うことになる。

(2)　ソフトウェアテスト

ソフトウェアテストで行う妥当性確認に関しても、ハードウェア実機環境と併用することで、シミュレータの適用が可能である。

例えば、ハードウェア実機環境上で確認が困難な複雑な組合せパターンの検証に関してはシミュレータを適用し、性能面の確認で必要な最低限のパターンをハードウェア実機環境上で確認する等のテスト戦略が考えられる。

(3)　回帰テスト

シミュレータのシナリオ機能により、過去に行ったテストで使用したシナリオを実行することで、アプリケーションのソースコード ロジック観点での回帰テストが可能である。

(4)　フォールト注入テスト

テスト対象の入力又はテスト対象内へのフォールト注入テストにおいて使用可能である。

シミュレータのシナリオ機能、又はデバッグ機能（統合開発環境 Visual Studio の機能）を使用して、変数値を任意の値に書き換えることにより、フォールト注入を行う。

シミュレータを使用したテストでは、ハードウェア動作との厳密な連携動作を検証するのではなく、フォールト発生時に行うべきソフトウェアの機能連携に着目したロジック検証を行うこととなる。

フォールト注入テストに関連する機能の多くは、機能安全に関係したものであり、車両自体の振る舞いに影響するものであるため、その最終的な検証及び妥当性確認を車両環境と同等な環境で行う必要があるが、事前にシミュレータ上で十分なソフトウェアのロジック検証を行うことにより、後工程での手戻りを抑えることができる。

図３にシミュレータを使用してのフォールト注入イメージを示す。

(5)　ソフトウェアデバッグ

設計フェーズ、検証フェーズを問わず、ソースコードロジックのデバッグで使用可能である。

統合開発環境 Visual Studio 上でテスト対象、及びシミュレータを動作させる構成であるため、Visual Studio が持つ豊富な機能を利用して、容易にデバッグを行うことができる。

5. インターフェースシミュレータ導入効果

インターフェースシミュレータ導入により、導入前に発生した問題の多くを改善することができた。以下にその効果についてまとめる。

(1)　テスト環境のボトルネック解消

開発担当者が自席で動作可能なシミュレータ環境を持ち、必要な時期に必要な期間、ソフトウェアデバッグ、ソフトウェア結合テスト、及びソフトウェアテストを実施するための環境を使用できるようになった。これにより、複数の開発担当者を投入して並行開発を行う様な局面においても効率的に開発を進められるようになった。

ハードウェア実機環境での検証が全く不要となる訳ではないが、ハードウェア実機環境を必要とする場面を

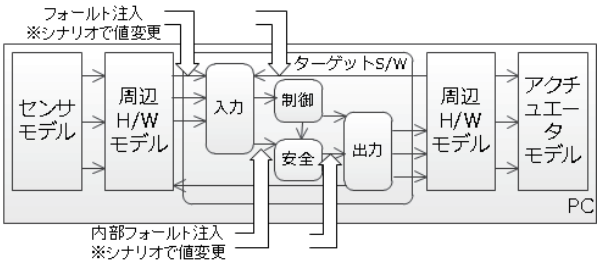

図３　シミュレータを使用してのフォールト注入イメージ

大幅に削減でき、該当機種開発に閉じる問題だけでなく、測定機器不足等を要因とする複数の開発に跨るハードウェア実機環境不足問題も大幅に改善された。

(2)　デバッグ効率の向上

インターフェースシミュレータ導入により、高度なデバッグ環境を使用できるようになったため、実装時や不具合調査でのデバッグ効率化が図れた。

特に、不具合調査時の効果は大きく、ハードウェア実機環境上で再現が難しい現象を、シミュレータ上で比較的容易に再現でき、原因特定の効率化が図れた。

(3)　フォールト注入テストの実施効率の向上

従来、フォールト注入テストはハードウェア実機環境を用いて行っていたため、ハードウェアへの細工やテストコードの埋め込み等の処置が必要であり、環境準備の難しさや、作業効率の問題があったが、シミュレータ適用によりそれらの問題が解消された。

また、ハードウェア実機環境上での作業機会を必要最小限に減らすことで、ハードウェア故障のリスクを低減できた。

(4)　回帰テストの網羅率向上

従来、回帰テストはハードウェア実機環境を用いて人手によって行っていたため、実施期間及びテストの再現性という点で問題があったが、シミュレータ適用により、短期間での実施が可能となり、回帰テスト項目の網羅率を上げることができ、さらに、テスト再現性についても確保することができるようになった。

また、CI（Continuous Integration）環境を構築し、更なる効率化を実現することができた。

6. 今後の展開

インターフェースシミュレータの導入により、テスト環境の制約によって引き起こされる多くの問題を改善することができ、その有効性を確認することができた。しかし、シミュレータ導入によって見えてきた課題もあった。その多くは、シミュレータを更に有効活用するための課題であり、例えば、シミュレータに読み込ませるシナリオの作成の効率化や、テスト結果検証の効率化に関するものである。

また、今後、新規機種へのインターフェースシミュレータ導入を進めるに当たっては、シミュレータ構築の効率化が鍵となり、構築を容易にする更なる工夫、仕組みが必要となる。

今後、これらの課題の改善を進めながら、新規機種への展開を行っていく計画である。

7. むすび

本稿では、当社で開発したインターフェースシミュレータについて、導入背景、特徴、用途、導入効果をソフトウェア検証の観点で述べた。しかし、シミュレータの活用については、機能改善を行うことで、まだまだ領域を拡大できると考えている。また、新たな活用方法を見出すためにも、改善と合わせて活用の普及も継続的に行っていきたいと考えている。

参考文献

(1)　仮想マイコン応用推進協議会／ vECU-MBD WG：仮想 ECU 活用のためのユーザ向け導入検討支援ガイド，Rev3.1
http://www.vecu-mbd.org/document/

執筆者紹介

京谷 智哉
1997 年入社。官公庁・防衛関連システムの開発を担当後、2009 年から車載向け組込システムの開発に携わる。

中越 敏典
1992 年入社。通信・防衛関連システムの開発を担当後、2010 年から車載向け組込システムの開発に携わる。