RC-006

シミュレータによる非実機車載ソフトウェア検証環境の統合制御コントローラ
An unified-controller for simulator-based automotive control software testing environment

伊藤 康宏 † 勝 康夫 † 太田 亙宏 ‡ 下澤 昌史 § 和田 健二 ‡
Yasuhiro Ito Yasuo Sugure Nobuhiro Ohta Masafumi Shimozawa Kenji Wada

要旨

自動車制御ソフトウェアの実機レス検証環境 vHILS に対して, 任意のシステム統合テストを高速高精度に実現する Unified Controller を開発した。本機構は, 異種複数シミュレータで構成されている vHILS 環境で, 各シミュレータの違いを意識することなく, ECU の内部状態操作及び観測を可能にした。更に, 非同期通信プロトコルを用いて実装することにより, シミュレータ実行と独立して動作可能なため, シミュレータ処理負荷の低減を可能とした。本手法を実際の車載制御用ソフトウェアの製品検証に適用し, 従来 275 時間を要していたテストの実行及び結果解析を, 1.1 時間まで削減する見通しを得た。

1. 研究の背景

近年, 組込み制御システムは, 交通, 産業, 医療, 家電など幅広い分野に応用されている。特に車載制御分野では, 燃費向上や排ガス対策のような環境対応や, 機能安全の保証のため, 性能・機能・信頼性の更なる高度化が求められている。その結果, 車載組込み制御システムに搭載するソフトウェアが複雑化している。

また,2011 年度に策定された安全規格 ISO26262 への準拠にあたり電子制御システムを構成する要素が故障した場合について安全性を確かめる必要がある。この対象には, 実機では検証者の望む通りの故障を起こすことが難しいメモリやマイコン等の電子デバイスの故障も含まれているため, シミュレータを用いた故障注入試験を行うことが強く推奨されている。

以上の要求により, ソフトウェア挙動に関する, 性能・機能・信頼性のあらゆる方面からの検証が必要となり, その工数が年々増加する傾向にあり, 検証工数の削減が期待されてきた。この課題を解決するため, 我々はソフトウェア挙動検証の着手前倒し及び高速化による開発工数の短縮を目的に, 組込みシステムの実機レス開発環境を計算機上に構築し, ソフトウェアの挙動をテストするシステム Virtual HILS(vHILS) を開発してきた。

従来 vHILS を運用するに当たり, 対象となる制御システムに対し作用する外部環境モデルや, データ取得用のライブラリは vHILS 適用対象ごとに変わるシミュレータの種類やインスタンス数, テスト項目に合わせてテストエンジニアがその都度専用モデルとして作り込む必要があった。このため, 大量のテスト項目を多種多様な車載システムに対して実行するためには, 非常に工数がかかる点が課題となっていた。

† （株）日立製作所
‡ （株）日立アドバンストデジタル
§ （株）日立ソリューションズ

本研究では, このテスト項目の挿入及び, モデル内部状態観測機能を一般化し, テスト項目用モデルの作り込みを不要とする事を目的とし, 上記二つの機能を最低限度の速度オーバーヘッドで実現するためのシミュレーション操作ライブラリ, Unified Controller(以下 UC) を開発した。UC は, vHILS を構成するモデルと直接データ交換を行うことが可能なように, ユーザ定義のシミュレーションモデルとして動作し, またシミュレーション外部からの指示や, 内部状態の送受信が可能な通信プロトコルを備えている。UC とシミュレータ外部のソフトウェアは非同期の TCP/IP 通信によって接続されているため, また異なる時間間隔をもつイベントを精度を保ちつつ実行しても, シミュレーション速度の低下を抑制可能である。

2. 従来の車載ソフトウェア検証環境

2.1.Hardware In the Loop Simulation (HILS)

現在車載システムの開発プロセスで広く用いられている, HILS の構成に関して述べる. 図 1 に示す通り, HILS は車両の機械的特性を模擬するための車両モデルを実時間で駆動するための特殊計算機である。この特殊計算機は機械系部品と同じ I/O を持ち, 実際に制御コンピュータを接続することで, 実製品と同じ環境を模擬することが可能である。また HILS には, その動作をユーザが対話的に操作するためのソフトウェアが付属しており, ユーザの手入力, または HILS 操作ソフトを自動化するスクリプトによる操作が可能である. 実物の制御コンピュータを必要とし, モデル化が難しい機械系部品の実物を用いるケースもあることから, HILS の実行環境は必ずある一定の設置する空間を必要とするため, 資源的制約を発生させる.

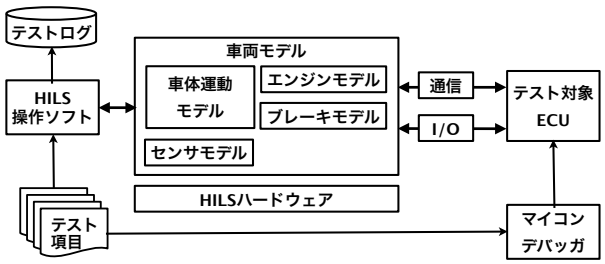

図 1: HILS 環境の構成

また, テスト条件の設定では電子系コンポーネント内部または, 通信でやり取りされる値が操作が困難である事から, 外部環境等の初期条件を車両モデル中で設定し

た後に, 電源を投入し所定のテスト条件に至るまで待機する必要が有る. また, 実行されているソフトウェアの内部状態を取得する場合は, マイコン部分に対しインサーキットエミュレータ (ICE) を挿入し, ソフトウェアの実行を一旦中断して取得する必要がある. そのため, テストの実行に時間がかかり実行可能なテスト条件が制限されることが問題である.

HILS で用いられるトレースは車両モデル, マイコン内部, 通信, マイコンボード上と, その発生箇所によって異なる手法を用いて採取される。そのため, これらのトレースはテスト実行中に解析処理を行うことは難しく, テスト終了後にすべてのトレースを集約し, 結果解析を行う事が主である。このとき, 個別の手法で収集されたトレースデータの時間依存関係を合わせるため, 各トレースデータにイベントの同期を行うようなコードを打ち込む手法が用いられる。

上記のように HILS では, 実機上でトレース可能な条件を用いて, 間接的に結果解析タイミングを指定する必要が有るため, 実行可能なテストの精度の向上が難しいという課題を持つ。

2.2.Virtual HILS(vHILS)

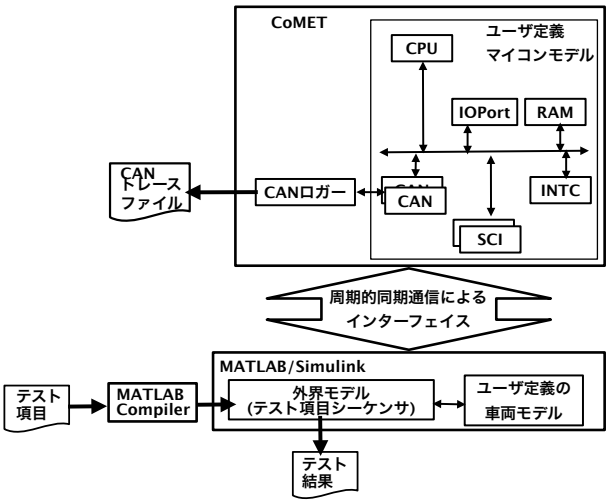

図 2: 従来の vHILS 構成

vHILS とは，マイコンもしくは SoC の実物の代わりに CPU, メモリ, バス, 周辺機器のモデルを用い, 実物と同じソフトウェアを実行し, その動作の正しさを検証する手法である。シミュレータを用いることにより, 実物のハードウェアが無い状態であっても実装コードをシミュレータ上で実行させることで，ソフトウェアの挙動を詳細に解析することが可能である。vHILS の構成要素としては, 図 2 に示すように, マイコンモデル, 車両モデルの他, シミュレーション外部より, テスト条件を指定するための外界モデルが必要とされる。この外界モデルは vHILS を用いて実行するテスト項目により, 介入可能な内部状態の種類が異なる。

これまでに構築してきた, 自動車走行制御システムをビークルとした vHILS 環境 [1] では, 自動車の走行状態における通信パケット内容と車両モデル中のブレーキ, エンジン挙動と比較的上位階層のデータのみをテスト項目で用いていた。このため, 外界モデルをテスト項目シーケンサとして, 車両モデルと接続する形で実装した。テスト項目シーケンサはテスト項目で指定された内部状態の操作や観測操作を順次実行する MATLAB/Simulink モデルとして, テスト項目毎に自動生成する方式を採用した。

本 vHILS は出力結果として, 車載ネットワークの一種である, Controller Area Network(CAN) のトレースと, 上記の車両モデル側で動作するテスト項目の結果をファイルとして出力し, ユーザはこれらのファイルを組み合わせて, テスト結果の解析を行っていた。この時, 例えば CAN のパケット内容の変化と車両モデル上での動作の変化の因果関係を調査するなど, 複数種類のログファイルにまたがった解析で常に人手が必要となっていた点が vHILS の課題点とされていた。

3.vHILS を統一的に操作する Unified Controller

前章で述べた既存 vHILS のテスト項目実行と結果収集解析に関わる課題に加え, 本報告で対象とする自動車エンジン制御に適用される vHILS では, 検証対象が ECU 内部のソフトウェア実行順序や, 割り込み発生タイミング等のマイコン内部状態と, バッテリー電圧等の相互作用など検査するなど, 複数のシミュレータにまたがって, ハードウェアに近いコンポーネントの動作データにアクセスする必要があるテスト項目をサポートする必要がでてきた。

これらを解決するため, 以下の 3 つ特徴を持つ vHILS 汎用テストコントローラとして UnifiedController(UC) を開発した。

1. vHILS に含まれるシミュレータの種類やインスタンス数, 求められる操作の種類によらず統一的インターフェイスを提供すること。
2. シミュレータを横断して, テスト項目の時間依存性を維持すること。
3. 同一シミュレータ内部でも, 異なる時間スケールのテスト項目を最小の処理負荷でサポートすること。

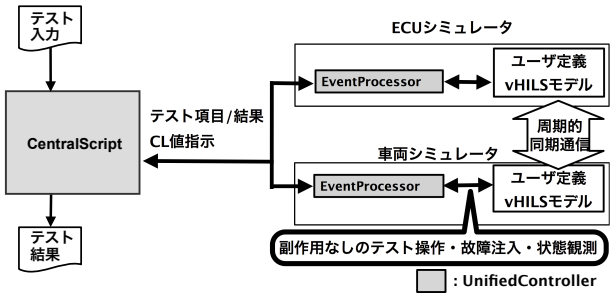

図 3: UC の構成概要

1 点目の統一的インターフェイスの提供のためには, これまで vHILS ユーザがシミュレータ毎に個別にやってきた内部状態の観測, 操作を肩代わりする必要がある。そこで, 図 3 に示すように, vHILS に含まれる全てのシミュレータに内蔵され, 実際に内部状態の観測/操作を行うプログラム (以下, EventProcessor, E.P. と略記) と, 全ての E.P. とソケットを介し接続されるプログラム (以下, CentralScript, C.S. と略記) という形で UC を定義した。ユーザには vHILS 全体に対するテスト操作を C.S. に入力するコマンド列として記述させ, C.S. が各コマンドに含まれるアクセス対象である内部状態から, 適切な E.P. を振り分ける方式を採用した。その接続方式の詳細は以下の'UC 導入済み vHILS 構成' の節にて解説する。

2 点目のテスト項目間の時間依存性に関して, 各 E.P. は独立して動作するため, それぞれに振り分けられるテスト項目の実行順序を確保することは難しい。そのため, C.S. と E.P. 間の通信でテスト項目実行順序の同期を確保する必要がある。本研究で開発した UC では, 各 E.P. に割り振られる C.S. との通信スロット内でシミュレータが進捗するタイムステップ (以下, CommunicationLatency, C.L. と略記) を定義することで, 同期の確保を実現した。同期方法の詳細は, 以下の'E.P. 動作フロー' の節にて解説する。

3 点目に関して, vHILS のテスト項目の特徴として, 各項目ごとに実行する時間スケールが大きくばらつく点が挙げられる。これに対し, 固定的なタイムステップを採用すると, 時間スケールの大きなタイマオーバーフローイベントの発生などで, 不必要に時間を浪費する。さらに, 2 点目で述べた同期方式を採用した場合, 詳細時間スケールの項目を同期させようとすると, シミュレーション時間オーバーヘッドが副作用として増大化する傾向にある。

そこで, 各 E.P. の C.L. を動的に調整する機能を実装し, 実行するテスト項目種類やその前後の内容の組合せから, 適切な C.L. を自動設定する機能を C.S. に実装することで, 強い時間依存関係をもつテスト項目の時のみ, 細かい同期間隔を適用することを可能にした。最適化の詳細は以下の'Unified Controller 動作パラメータ自動最適化' の節にて解説する。

3.1.UC でサポートされるテスト項目

本研究で開発した UC は, 車両モデルだけではなく, 制御ユニットの内部状態観測及び操作をサポートする必要があった。本研究で vHILS の適用対象となった, 自動車エンジン制御システムでは, 走行制御, ボディ装備などからの通信や, 基板上の監視用マイコンなどとメインの SH マイコンが, CAN, SCI などを通じて連動している。 このシステムに適用する vHILS ではこのような通信, 基板上の配線の仕様上の不具合に対するソフトウェア挙動を確かめるようなテストが実行される。また, RAM 異常や演算器異常に対応するフェールセーフ機能等の実機上で任意に発生させることが困難なケースにも対応するため, メモリ値の書換や故障注入もサポートする必要がある。

本研究では vHILS のテスト項目処理手順を図 4 に示すように設定した。先に述べた, 操作系命令の複数個実行した後, シミュレーション内部状態の変化または時間経過をトリガ条件とする, トリガ命令を用いて, シミュレータ上の観測タイミングを測り, 最後に期待値判定を行う状態取得系命令を複数実行する構成である。この３つの命令グループを繰り返す事によって, 任意のテスト項目の記述を可能にした。例えば, 関数呼び出し等の特定タイミングに依存して内部状態の操作を行う場合は, 状態取得系命令に何も実行しない命令 (NOP) を指定すれば良い。UC でサポートするコマンドを以下の表 1 に纏める。

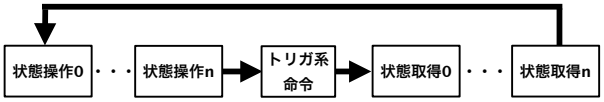

図 4: UC でサポートする命令の組み合わせ

表 1: UC がサポートするテストコマンド

コマンド名	概要
Logic/Analog 値設定	I/O, 配線を指定された値に設定, 一定時間の値を固定
PWM 設定	指定された周期, duty 比の PWM 信号発生
メモリ故障注入	メモリアクセス時にメモリ値を反転
メモリ値 HOLD	指定メモリ番地の値を固定
メモリ値設定	指定メモリ番地の値を上書き, マスク値適用
CAN パケット送信	指定された CAN パケットを送信
CAN 周期, 加算設定	周期的な CAN パケット送信設定
シリアル送信	対象チャネルでシリアルのパケットを送信
クロック周波数変更	クロック周波数の変更や, 供給停止
Logic/Analog トリガ	I/O, 配線の値が条件式を満たすまで待機
CAN トリガ	CAN パケット内容が条件式を満たすまで待機
ソフトウェアトリガ	指定 PC の通過または関数呼び出しまで待機
メモリトリガ	指定アドレス値が条件式を満たすまで待機
待ち時間	指定時間分次のテスト項目の実行を遅延
Logic/Analog 値取得	指定された I/O, 配線の値を取得
メモリ値取得	指定アドレス値を取得, 読出しマスクの適用
CAN パケット取得	指定 CAN パケットの最新値を取得
シリアル受信	シリアル通信のパケットを取得
レイテンシ調整	C.L. を指定値に設定

3.2.UC を導入した vHILS 構成

前節で挙げたコマンドは 1 種類の E.P. ではなく, 下記 5 種類の E.P. に分割して実装された。(1)AmpiEventProcessor : CPU シミュレータにおける Logic/Analog, メモリ, クロック関連コマンド, (2)CANOperator : CAN 関連コマンド, (3)SCIOperator : シリアル通信関連コマンド, (4)SoftEventProcessor : ソフトウェアトリガ, (5)SimulinkEventProcessor : 車両モデル側での Logic/Analog 関連コマンド, これは, 以下の二つの課題を解決するためである。

1. 全ての機能をひとまとめに実装した場合, たとえ利用されない場合でも, これらの特殊な接続部分がシミュレータ内部にインスタンスとして残り, シミュレータの処理負荷を増加させる事を防ぐため
2. ソフトウェアトリガに関しては, 他の E.P. と違い, 仮想的なシミュレーションモデルとしてでなく, シ

ミュレータ付属の実行トレース取得ツールの一部として実装する必要があるため

この機能分割によって, C.S. は実行するべきコマンドの種類と, 実行中の vHILS に含まれる E.P. のインスタンスから適した割振り先を選択する必要がある。

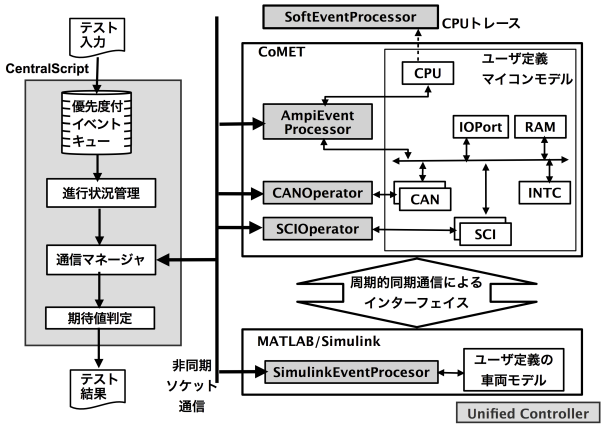

図 5: UC を導入した vHILS の構成

図 5 に UC を導入した際の自動車エンジン制御システム向け vHILS 構成図を示す。この vHILS は CPU シミュレータの CoMET と車両シミュレータの MATLAB/Simulink の協調シミュレーションによって構成される。CoMET 上ではユーザ定義のマイコンモデルが動作し, 前述の AmpiEventProcessor, CANOperator, SCIOperator がシミュレータ内部の配線を用いて接続される。また, SoftEventProcessor は ECU シミュレータの CPU トレース取得機構のプラグインとして接続されている。もう一つの車両シミュレータでは, この vHILS 用 ECU モデルと周期的な同期通信インターフェイスを通じて協調動作する, ユーザ定義の車両モデルが存在し, それらとは前述の SimulinkEventProcessor が接続される。

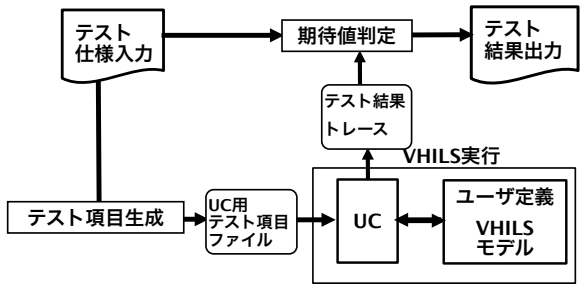

図 6: vHILS 自動実行システムの運用形態

これらの E.P. は C.S. とそれぞれ異なるプロセス間通信ソケットにて接続され, C.S. からのテスト項目の送信待受けと, 実行結果の返送を行う。E.P. と C.S. 間の通信は非同期の TCP/IP を用いて実装されており, ユーザ定義モデル間の同期通信とは全く独立の通信である。よって, 実行するべきテスト項目が存在していない場合, 両者の間に通信が発生しないため, 通信負荷を抑制可能である。

図 7 に E.P. と C.S. でやり取りされるコマンド列の形式を示す。コマンド列は, シミュレータ中に複数存在する E.P. から実行対象を指定するための ID フィールド, コマンドの種類を指定するための Type フィールド実行時間を指定する Time フィールド, コマンドの引数を格納する可変長の arg フィールドをもつ。また E.P. での該当命令の実行後はその実行したタイムスタンプ, メモリ値, I/O ピンの値の実行結果が Time フィールド, arg フィールドに上書きされて, C.S. に返送される。このように, 定型化されたコマンド列によって, 任意のテスト項目のやり取りを実施可能である。

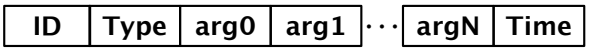

図 7: E.P. の通信プロトコル

C.S. はテスト項目を実装したコマンド列を受取り, そこに記述されたコマンドの実行時刻を優先度とする待ち行列に格納し, 実行時刻が小さいものから実行する。E.P. が返送する, トリガ系, 観測系命令の返り値を集め, 期待値判定してテスト出力ファイルを出力する。上記構造によって, ユーザからは異種複数のシミュレータによって構成される vHILS が, C.S. を介することで予め定義された入出力コマンド列を処理する単一のシミュレータをみなすことが可能となる。

以下に C.S. の入力ファイルの記述例と, 出力ファイルの例を示す。ユーザは図 8 に示すようなスプレッドシート形式で, 実行するべきコマンドを記述する。個々でユーザは操作を望むシミュレータ上の I/O ピンのインデックスを指定可能な他, プログラム上の変数名, 通信パケット上のビットフィールドを指定して, その値を操作したり, トリガ条件式を設定することが可能である。これらのターゲット指定や実行する命令の指定は, 入力ファイルを C.S. が受け取った際に, ターゲットソフトウェアのコンパイラ中間生成物, 車両モデルの構成情報から, 対応するメモリアドレス, I/O ピンのインデックス等を割り出し, E.P. が解釈可能な情報に変換する。

期待値判定命令は, シミュレータ内部の状態取得指定だけではなく, 取得する値を用いた期待値判定が可能なように条件式を与えることが可能である。状態取得指定は, トリガ系命令や操作系命令と同じく C.S. によってコンパイルされ, E.P. で処理される. 条件式は C.S. に貯められ, E.P. で処理された状態取得指定の結果を用いて, 該当命令の実行後に利用される. 条件式は, C.S. の実装に用いる言語で利用可能な計算式を用いて記述する。結果及びトリガ系命令の実行時間は, 図 9 に示す通り, ユーザが入力したテスト項目指定に追記する形で出力される。期待値判定が False の際は, 実際に観測された値が出力される。

状態操作		トリガ条件			期待値判定0			
命令	引数	命令	引数	発火時間	命令	引数0	判定式	結果
Logic値	TI0A	nop			nop			
Analog値	ブレーキ圧 =300	関数トリガ	ADCTask		メモリ値	ADCReg	$1==0xAA	
	20ms	CANトリガ	ID1AA = 0xAABB		メモリ値	Val1	$1 % 0xF == 8	

図 8: C.S. の入力例

状態操作		トリガ条件			期待値判定0			
命令	引数	命令	引数	発火時間	命令	引数0	判定式	結果
Logic値	TI0A	nop			nop			
Analog値	ブレーキ圧 =300	関数トリガ	ADCTask	1.294	メモリ値	ADCReg	$1==0xAA	0xBB
	20ms	CANトリガ	ID1AA = 0xAABB	1.347	メモリ値	Val1	$1 % 0xF == 8	OK

図 9: C.S. の出力例

この実装により, C.S. から上流の処理フローを図 6 に示すように共通化可能になった。さらに UC の入出力機能である, テスト項目の生成, 及びテスト結果の収集と期待値判定の各機能を前述の vHILS 自動分散システムの動作仕様に合わせて設計することで UC を vHILS の統一的なインターフェイスとして扱うことで, マイコンファミリの違い, 実行するテスト項目の種類の違い等に影響される事無く, シミュレーション操作を可能にした。

3.3.E.P. 動作フロー

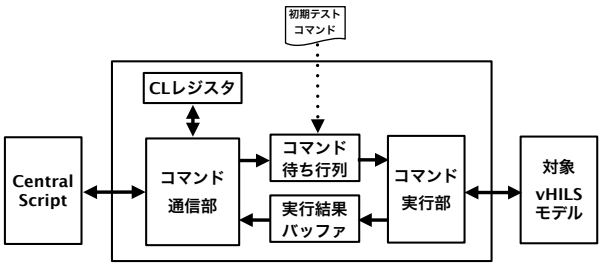

図 10: E.P. の共通部分実装

次に, 各 E.P. で共通実装部分の構造とそれを用いた, イベント処理方法に関して解説する。図 10 に E.P. で共通であるイベント処理部分の構造と C.S. との接続を示す。E.P. の内部は二つのスレッドで構成されており, コマンド通信部が含まれるコマンド通信スレッドと, コマンド実行部が含まれるコマンド実行スレッドに分けられ, それぞれ独立して動作する。両スレッド間では, コマンド待ち行列及び, 実行結果バッファが共有されており, これによりコマンドの共有が可能となっている。またコマンド待ち行列は, デバッグ時や, C.S. からの入力と無関係に固定的にコマンドを発生させたい場合に対応して, コマンド列をファイル内容から格納することが可能である。コマンド通信部は C.S. とソケット通信で接続され, 実行するコマンドや実行結果のやりとりを行う, また, コマンド通信部には後述する C.L. 値を保存するための CL レジスタが付いている。前述の UC でサポートする命令のうちレイテンシ調整命令は CL レジスタに格納される値を更新する。コマンド実行部は対象となる, vHILS モデルと直接接続される場所であり, この部分の実装, サポートするコマンドや, シミュレータ種類によって変更することが可能である。

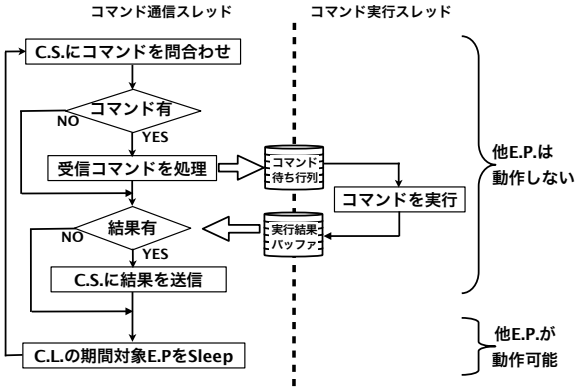

図 11: E.P. におけるイベント処理フロー

図 11 に E.P. によるイベント処理フローに関して解説する。前述のとおり, E.P. 内部処理は, コマンド通信スレッドと, コマンド実行スレッドの二つに分割されており, 独立して動作する。

コマンド通信スレッドは, シミュレーション実行中繰り返し, C.S.(以下, 図中でも C.S. と略記) に対し非同期通信のリクエストを続ける。まずコマンド受信部は C.S. に対し当該 E.P. の実行するコマンドの有無を問い合わせる, これは非同期ソケットの Read を利用しているので, もし実行すべきコマンドが存在しない場合, 受信処理をスキップする。もし, コマンドを受信した場合, このコマンドをコマンド待ち行列に格納する。 この待ち行列はコマンド毎に設定された, 当該コマンドを実行するべきタイムスタンプを優先度として挿入時にコマンド列のソートが可能である。次に, 実行結果バッファの有効エントリ数を参照し, 返送するべき結果が存在した場合のみ C.S に対し通信を行う。最後に, 前述の CL レジスタに格納された C.L. に対応したシミュレーション時間分 E.P. を Sleep させる。

コマンド実行スレッドは, コマンド待ち行列への新しいコマンドの挿入をトリガとして自身の Sleep を解除する。その時点でのシミュレータタイムスタンプが前述のコマンドに設定されたタイムスタンプより若い場合, 前述のコマンドを実行するべきタイムスタンプまで再度自身を Sleep させる。コマンド実行部はコマンド待ち行列の優先度最高のエントリを一件取得し, そのコマンドを実行し, 終了したコマンドがトリガ系及び観測系コマンドであった場合, モデルより取得できる値とシミュレータタイムスタンプを返り値として, 実行結果バッファに格納する。

上記のプロセスをまとめると, E.P. は, 非同期の送受信と C.L. 分の Sleep, コマンドの実行を 1 セットを繰

り返している。また, vHILS に用いる CPU シミュレータでは, E.P. の Sleep 時以外はシミュレータの制御自体を E.P. が独占する仕様となっているため, コマンドの送受信, 実行中は他の E.P. の動作を停止させることが可能であり, その結果 C.S. で定義したコマンドの実行順序を保つことが可能になる。

4.Unified Controller 動作パラメータ自動最適化

前章にて C.L. を用いた E.P. 間の同期手法を紹介した。これによって, ユーザが定義したテスト項目の実行順序の確保が可能となった。その一方で, 下記に述べるシミュレーションの速度及び精度の問題が副作用として生じた。本説ではこの問題を解決するために開発した, C.L. の動的な調整を高速に行う手段に関して述べる。

4.1.Communication Latency のシミュレータ精度への影響

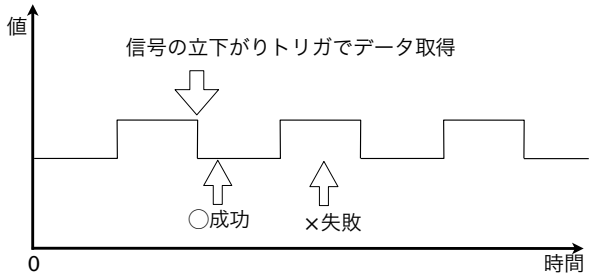

図 12: CL 値の精度への影響評価実験内容

表 2: 複数 CL 化での精度/速度評価

C.L. 値 (ps)	成功率	実行時間
1M	16/17	1min24sec
200K	17/17	2min31sec

C.L. の値は, 各 E.P. が単位シミュレーション時間辺りにシミュレータの処理を独占する回数と関連するため, 小さい C.L. 下では, シミュレーション精度が高くなる反面, シミュレータの処理負荷が高くなる傾向がある。C.L. がシミュレーション開始時に静的に決定される環境化では, テスト項目のうち最も時間スケールの細かいコマンドに合わせる必要がある。

図 12 は C.L. の影響を定量評価するために行った実験内容を示す。1 ミリ周期で立上がりと立下がりを繰り返す信号の立下がりをトリガとしてその信号値を取得するコマンド列を E.P. に与え, その実行タイミングを評価した。立下がり区間中にデータが取得できた物を成功とし, 遅延により次の立上がり後にデータを取得した場合を失敗と定義した。このテストコマンドを 1M と 200k, 2 種類の C.L. 値で行った結果を表 2 に纏めた。前述のテストコマンドを 17 回実行し, その成功率とシミュレーション実行時間を評価した。この時, C.L. を 1M に設定したシミュレーションでは実行ノードへの負荷集中時にタイミングエラーを発生させたのに対し, 200k 条件下では, 全ての試行でデータの取得に成功した, その一方で,1M 条件下と比較しシミュレーション実行時間が約 80%長くなっている。このように, C.L. 値の設定は特にソフトウェア挙動の検証に用いる vHILS では重要となる。

4.2.C.L. 自動調整機構

実行するテスト項目に応じて動的に C.L. を更新することにより, 最適なシミュレーション速度と精度を達成することが可能となる。前述のとおり全ての E.P. はその中で保持している, C.L. 値を更新する命令をもつ。しかしこの命令を多用した場合, 高頻度の通信を C.S. 間に集中させてしまい, その結果前述の高速化の効果を損なってしまう。そこで, Latency 調整命令の発行を最低限に抑制しつつ高頻度の調整を可能にする事を目的として, 本節で解説する自動調整機構を導入した。

前節の評価で提示した実験の成功条件から, UC を導入した vHILS の精度は, トリガ系コマンドとのその後に連なるコマンド間に求められる時間間隔に帰着できる。これと, UC のコマンド列がトリガ系コマンドの後に操作, 観測系コマンドが続く仕様を組合せ, 以下の方法で精度と速度の両立が可能になる。コマンド列中のトリガ系命令から次の有時間処理を伴うコマンド, 即ちトリガ系, 通信系送信, 待ち時間挿入等のコマンドの実行直前までは, C.L. は可能な限り小さくし, 時間精度を確保する。一方で有時間処理を伴うコマンドの実行開始から終了までは, いかなるコマンドの時間依存性も存在しないため, シミュレータで許される最大の C.L. 値を設定することで UC のオーバーヘッドを低減させる。

この調整アルゴリズムを単一シミュレータ内部で C.S. の介入無く, 自動調整を行う CLBUS を開発し, さらにシミュレータを跨いだ時間依存関係に関し, 適応的に調整を行うテスト項目自動生成プロセスを開発した。

4.3.CLBUS:シミュレータ内部の自動調整方法

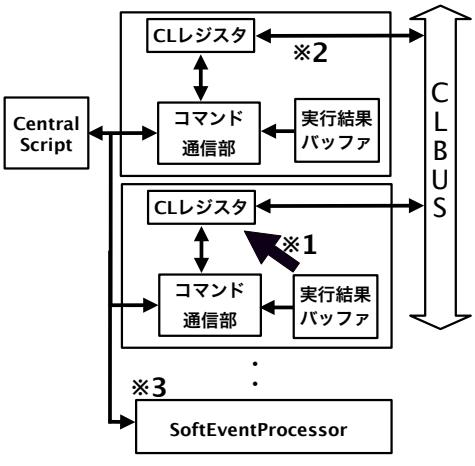

図 13: CLBUS の構造

CLBUS は単一シミュレータ内部に複数導入された E.P. 間で完結した, C.L. の共有及び自動調整を目的として導入した。図 13 にその構造を示す。前述の E.P. の CL レジスタを, シミュレータ内部の配線を用いて相互結合させた。これにより, 最後に CL レジスタが更新された時の値が, 同一シミュレータ内部の全 E.P. で共有される。さらに, 前述の CL 調整アルゴリズムを適用させ, 実行結果にトリガ系コマンドが挿入された時点で, 即座に CL レジスタにシミュレーションを減速させて, 動作精度を向上させるための値 (本実装では 1) を書込む機構と, コマンド待ち行列にトリガ系, 通信系送信, 待ち時間挿入等のコマンドが挿入された場合, CL レジスタに予め与えられたシミュレーションを加速して, 再開するための値を書込む機構を追加した。

これらの一連の処理はシミュレーションモデルの結合に利用する配線を用いており, CL レジスタ書込みと同じシミュレーションタイムスタンプですべての C.L. の更新を可能にするため, シミュレータ挙動に影響のある処理遅延を発生させない。

4.4. シミュレータ間の自動調整

単一シミュレータ内部でも SoftEventProcessor とその他の E.P., あるいは他シミュレータインスタンスをまたぐ C.L. の共有は前述の CLBUS ではサポートすることが難しい, 例えばシミュレータ間周期的同期通信で交換する値に CLBUS を加えることは可能だが, その場合シミュレータ間同期周期以下でのコマンド時間依存性を保つことが難しい。 また SoftEventProcessor と E.P. 間の CL 値同期の場合も, C.S. を経由するためソケット通信の送受信各 1 回のうちに, ユーザが所望した状態が更新される恐れがある。

そこで本研究では CLBUS の機能を補完する形で, シミュレータ間の CL 自動調整プロセスを追加した。本プロセスはソフトトリガコマンドや, トリガ系コマンド後にシミュレータを跨いだコマンドが存在した場合, C.S. がレイテンシ調整命令を自動で挿入する。その時の引数としては, トリガ系コマンドの次にトリガ系コマンドの成立と CL レジスタ値更新までの時間経過値を用いる。

これにより, CLBUS によってトリガ条件の成立前後で過度にシミュレーションが高速化する事象を抑制することが可能で, その結果, シミュレーション精度を確保することが可能である。

5.UC 導入効果の評価

5.1.UC を導入した vHILS の速度評価

前述の UC と UC における C.L. 制御を導入した vHILS の速度及び精度評価を行った。評価に当たり, 前述の HCM 向け vHILS を用い, 入力としては既にデバッグ済のターゲットソフトウェア, 実製品の検証に用いる BIOS 層ソフトウェア検証シナリオのうち前述のソフトウェアで期待値判定が全て真となる物を抽出して用いた。これによって, 評価中に期待値判定に問題が出た場合, C.L. の調整不良により生じた精度問題とすることができる。評価方法として, 期待値判定が発生しないよう C.L. 値を調整した状態で, 各自動制御の有無を切り替えてシミュレーションを 10 度実行し, 実行速度の平均を算出した結果を下記の表 3 に纏めた。

理論的な性能限界を示すため, 導入されていた UC を全て取り除いた状態での速度も併記した。ただし, この時テスト操作に必須な外部環境が無いため, それ以外の試行とはソフトウェアの動作範囲が限定されることにより, 実際よりも高速に実行された結果が出力される。また, 速度比を実機との比較を示すためプログラムとしての実行時間をテスト終了時のシミュレータタイムスタンプで除算した値と定義して記載している。テスト終了時のシミュレータタイムスタンプは全てのシミュレーション条件で同一の値を返す。

表 3: UC と C.L. 制御による vHILS 速度評価

UC 動作条件	実行時間 (sec)	速度比
静的 CL	20.98	27.62
CLBUS	13.04	17.20
CLBUS+CL 自動調整	8.379	11.03
テスト項目ハードコード	5.275	6.943

CLBUS の導入によって, 39%シミュレーションが高速化されている。先に述べた通り, 以前の実装ではテスト項目間の時間スケールの差のため, シミュレーション時間を浪費していたことがわかる。また CL 自動調整を用いる事により 36%のシミュレーション高速化がされている。これは, 採用したテストケースで, 割込みハンドラ実行をトリガとしてメモリ値の操作/観測を多用しており, その結果, SoftEventProcessor と他の E.P. をまたぐテストコマンドが頻出したためと結論付けられる。

この場合例えば, トリガ条件をマイコンの INTC からの割込み要求線の値に変更することで, CLBUS のみ導入時の動作速度を向上することが可能で, その一方, CL 自動調整機能の効果は限定的になる。また UC を全て無効化し, 注入するテストコードを全てモデル中にハードコードした場合の動作速度は実機比 6.9 倍で, CLBUS+CL 自動調整の動作速度の約 1.5 倍であり, これはこれまでに述べた最適化機構により, UC の速度オーバーヘッドを 1.5 倍までに抑制可能であることを示す。

vHILS の有効な適用先として, マイコン内部故障のシステム挙動への結果解析が考えられるが, この時, マイコンの演算器や RAM 等の時間スケールの小さな事象をトリガとして, 機械挙動等の時間スケールの大きな系の挙動を考える必要がある。本章で示した結果から, UC は一度に実行するテストケース間の時間スケールのばらつきが大きいほど, シミュレーション速度をより適切に保つことが可能である。よって, 上記用途において UC が高い効果を示す。

5.2.UC を導入した vHILS による Turn around time(TAT) 評価

UC を導入した vHILS によるソフトウェア検証 TAT の削減効果を定量的に評価した。評価にあたり同じソフトウェア検証項目を HILS と vHILS とで実行した際の所要時間の比較を以下の図 14 にまとめた。UC を導

入しない vHILS の場合 HILS で実行可能であるテスト項目が実行できないため, 比較対象から除外した。また比較対象とする HILS のデータとして, HILS の機材の予約と運搬, 計測装置などの接続, HILS の実行, 実行結果を解析してレポート作成を行うまでの時間の総計が 275.7 時間であると, 過去の開発実績を解析した結果判明した. それに対して UC を導入した vHILS では vHILS 自体の実行は, 同じテスト項目数を 1.1 時間で実行し, 実行結果の解析からレポート出力まで約 5 分という結果であった。

従来, HILS では一つのテストの中でも, 観測する状態に応じてマイコンの動作を止め, 計測機器を変更するなど常に人手による作業が必要であり, トレース情報取得の都度, 仕様との合致性を判断することも難しく, 後処理工程で物理系, 制御系動作の時間依存関係を人手で調整して解析する必要があった一方, UC を導入することにより, vHILS はテスト項目の実行から, 期待値判定までを全て自動的に, シミュレータ実行中に実行可能であることが大幅な高速化につながったと結論付けられる。

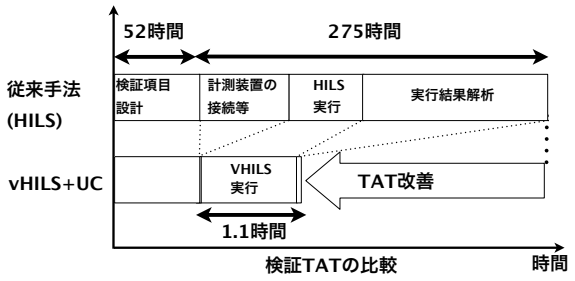

図 14: ソフトウェア検証プロセスの TAT 評価

6. まとめ

本報告では, 仮想システムを用いたソフトウェア検証環境 vHILS に対し, 特殊なモデルの作り込み無しで異種複数シミュレータの差異を意識する必要なく内部状態操作と観測を可能にする汎用シミュレーション操作ライブラリ, Unified Controller(UC) を開発した。本ライブラリは下記の要求仕様を満たすよう設計された。

1. ユーザが実行するテスト操作の種類やシミュレータに依存したい統一的インターフェイスの提供。
2. シミュレータを横断して, テスト項目の時間依存性を維持すること。
3. 異なる時間スケールのテスト項目を混合して実行しても, 性能低下を起こさない

上記仕様を満たすため, UC はテスト項目の割振りや実行順序の動機を行う C.S. と実際にシミュレータ内部で仮想的なモデルとして, テスト項目を実行する E.P. とが非同期通信で結合された構成を採用した。この UC を組込んだ vHILS を実際の自動車エンジン制御システムのソフトウェア検証に向けて実装し, その性能を評価した。その結果 UC を組込まない場合と比べ, 1.5 倍シミュレータ速度へのオーバヘッドで, 複数階層のシミュレータ内部情報をその違いを意識せず容易に横断的に扱うことが可能と解った。

ソフトウェア検証工程の TAT を比較した結果, 従来の HILS と比べ UC を組込んだ vHILS は約 1/300 に検証の TAT を削減可能であることが示された。これは従来の HILS と vHILS で, テスト項目の注入や実行結果の解析に関し, 人手で計測機器の切替えや, データの付きあわせが必要であったところを UC が全てシミュレーション実行と同時に処理可能としたことが寄与している。

本ライブラリは, 一度に実行するテストケース間の時間スケールのばらつきが大きいケースほど, 従来手法に比べ, シミュレーション速度をより適切に保つことが可能である。例えば車載システム全体レベルの故障注入テストのように, ナノ秒オーダでマイコン系の誤動作を発生させ, 秒オーダで動作する車両系の動作を評価するケースのように, 今後安全規格として求められる, システム全体の安全性確認の有効な手段の一つとして利用されるのが望ましい。

参考文献

[1] 伊藤, 勝, 於保 Virtual HILS : システム全体仮想化による組込みソフト検証の高効率化, 電子情報通信学会技術研究報告. CPSY, コンピュータシステム 110(473), 243-247, 2011-03-11

[2] Yasuo Sugure, et al., “ Virtual Engine System Prototyping with High-Resolution FFT for Digital Knock Detection Using CPU Model-Based Hardware/Software Co-Simulation, ” SAE World Congress (2009-01-0532) 2009

[3] George Saikalis, et al., Virtual Embedded Mechatronics System, SAE World Congress (2006-01-0861) 2006

[4] Makoto Ishikawa, et al., CPU Model-based Hardware/Software Co-design for Real-Time Embedded Control Systems, SAE World Congress (2007-01-0776) 2007

[5] Y.Sugure, et al., Virtual Engine System Prototyping with High-Resolution FFT for Digital Knock Detection Using CPU Model-Based Hardware/Software Co-Simulation, SAE World Congress (2009-01-0532) 2009