Vol.2012-SLDM-155 No.11
Vol.2012-EMB-24 No.11
2012/3/2

ハードウェア異常に対応した組込み制御ソフトウェア不具合のモデル検査手法

松原正裕[†]　櫻井康平[†]　成沢文雄[†]　山中久光[††]

組込み制御システムではソフトウェア規模の増大により，処理タイミングによりごくまれに発生する不具合がシミュレーション等のテストでは発見できずに潜在する恐れが高まっている．このような不具合を検出するため，非定常の挙動を含む外部環境モデルと，外部環境との境界調整が容易なソフトウェアのスライシング方式を適用してモデル検査を行う手法を考案した．本手法を自動車用ソフトウェアの非再現不具合の原因調査に適用して不具合要因を発見し，本手法の有効性を確認した．

Model Checking to Find Defects in Embedded Control Software Affected by Hardware Failures

MASAHIRO MATSUBARA[†]　KOHEI SAKURAI[†]
FUMIO NARISAWA[†]　HISAMITSU YAMANAKA[††]

In embedded control systems, potential risks have been increasing because of software defects caused by a timing related problem. These defects are rarely found by simulation and test. To detect such defects, we have proposed model checking application method based on external environment models with unsteady behaviors and the software slicing technique using variable dependency graphs. We have applied the proposed method to an analysis of a non-reproducible defect in automotive control software and demonstrated the usefulness of the method.

1. はじめに

組込み制御システムでは，装置の電動化や診断機能の高度化等に伴う制御用ソフトウェアの規模増大により，ソフトウェア不具合が潜在し，システムの安全性を損なう恐れが増していることを背景として，IEC61508 や ISO26262 などの機能安全規格が制定されている．ハードウェアの不具合がソフトウェアの不具合を誘発するケースもあり，ソフトウェアのテストだけでは不具合を検出しきれないのも難しい点である．組込み制御システムの検証手法として HILS（Hardware In-the-Loop Simulation）のようにハードウェアとソフトウェアを合わせてテストする方法も実用化されているが，テストケースのカバレッジが課題となる．特にハードウェアの動作やソフトウェアの割込み処理等に起因するタイミング関連の不具合は，テストケースが無限に存在しうるため，検出が困難である．

このような問題を解決する手段として，モデル検査[1][2]の適用がある．モデル検査では，検査対象の取り得る状態を網羅的に探索して，検査対象が要求される性質を満たすか否かを判定する．この際，検査対象を適切にモデル化すれば，無限に存在する処理タイミングを有限の組み合わせに落とし込むことができるため，タイミング関連の不具合には特に効果が大きい（図 1）．またソースコードをモデル化することにより，仕様を実装する段階で混入した不具合も検出することができる．

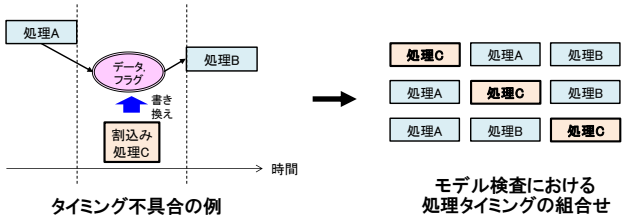

図 1　タイミング関連の不具合現象例
Example of malfunction related to processing timing

ところがハードウェアを含む制御システム全体にモデル検査手法を適用しようとすると，システムの複雑度が大きいため，いわゆる状態爆発により検証が完了しない恐れがある．状態爆発を回避するためには，ソースコードの詳細さをなるべく維持した

[†] (株)日立製作所
Hitachi Ltd.
[††] 日立オートモティブシステムズ(株)
Hitachi Automotive Systems, Ltd.

Vol.2012-SLDM-155 No.11
Vol.2012-EMB-24 No.11
2012/3/2

ままモデル化する範囲を限定し，それ以外の部分を外部環境として簡略化する必要がある．しかし，ソースコードからモデル化する範囲を狭めすぎると，不具合要因がモデル化されず見逃す恐れが発生する．このため，ソフトウェアモデルと外部環境モデルとの境界を検証したい性質に応じて調整する必要がある．

本稿では，ハードウェアとソフトウェアの複合動作により，実機でごくまれに発生するタイミング関連の不具合の検出に主眼を置くモデル検査手法を提示する．本検査手法の特徴は，変数依存関係に基づくスライシング手法により，ソフトウェアと外部環境との境界を柔軟に調整可能とした点にある．また，非定常または故障動作を有する外部環境モデルを利用し，ハードウェアとソフトウェアの複合動作による不具合を検出する点にある．以下 2 章にて本検査手法を説明し，3 章では本検査手法の適用事例を示す．4 章では関連研究との比較を示す．

2. 検査手法

2.1 組込みシステムのモデル化方針

本検査手法における組込みシステムのモデル化方針を図 2 に示す．実機にて発生しうる不具合を検出するため，検証対象とするソフトウェアのモデルはソースコードから変換して構築する．また状態爆発を回避するため，ソースコードから検証点に関係する部分を抽出するスライシング[3]を行う．この際，スライシングは変数間の依存関係に基づき実施する（2.2 参照）．

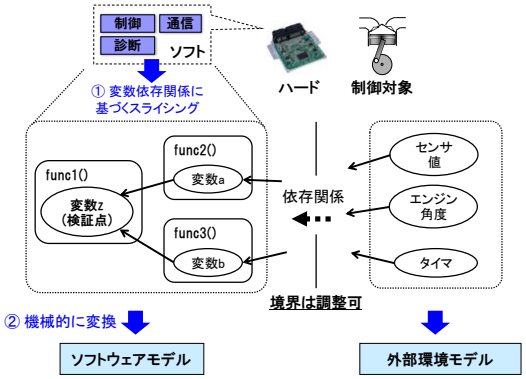

図 2　組込みシステムのモデル化方針
Modeling Policy for Embedded System.

ハードウェア（検証したい性質に関係すれば制御対象機器も），またソフトウェアの一部は，外部環境としてモデル化する．この際，非定常または故障動作の振る舞いを外部環境モデルに持たせる（2.3 参照）．

変数依存関係に基づくスライシングは，ソフトウェアモデルと外部環境モデルとの境界は柔軟に調整可とする．境界を調整する必要性は，検証点に関連するソースコードを全てモデル化すると状態爆発してしまう場合があるためである．境界の調整が必要な例を図 3 に示す．図 3 の事例では，ソフトウェアと外部環境との境界であるドライバ部は不具合に直接関係がない一方で，通信などの複雑な処理が入っており，検証上は省略することが望ましい．この場合，ソフトウェアのドライバ部から外側を外部環境として扱い，実際の処理よりも簡略化するのが適切である．このように，モデルの詳細度は検証対象に応じて適宜変更する必要がある．

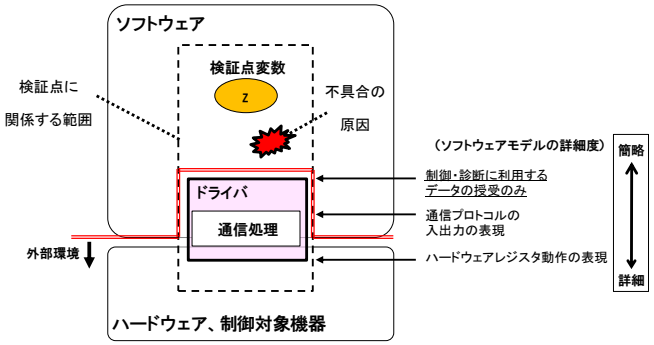

図 3　ソフトウェアと外部環境との境界の抽象化
Figure 3　Abstraction of Boundary between Software and External Environment.

2.2 変数依存関係に基づくスライシング

スライシングを行う際に利用されるデータには通常，プログラムの命令間の依存関係をつないだプログラム依存グラフ（PDG）が利用される[3]．また，スライシング基準として，始点のステートメントと，終点のステートメント及びそのステートメントに含まれる変数を指定する．

これに対し本検査手法では，スライシングに利用するデータとして，変数単位で依存関係をつないだデータフローである変数依存関係グラフを採用する．この理由は，検証者がデータフローを辿ることでモデル境界を調整することが有効なためである．

変数依存関係グラフでは，接点を記述位置で区別した変数とする．プログラム依存

Vol.2012-SLDM-155 No.11
Vol.2012-EMB-24 No.11
2012/3/2

グラフでは接点間の依存関係の種類として,データ依存関係と制御依存関係があるが,これらは変数依存関係グラフでも変数間で存在する.変数依存関係グラフではさらに,代入関係（代入文による依存関係）も扱う.

プログラム依存グラフと変数依存関係とは情報量としては等価と考えられるが，本検査手法では変数依存関係を明示し，外部環境との境界を検証者が調整可とする点に特徴がある.

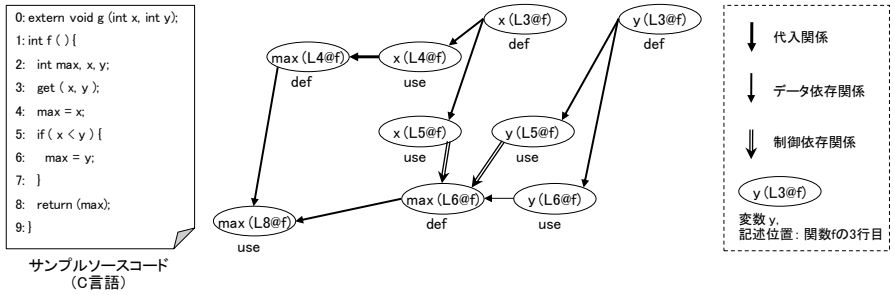

図 4　変数依存関係グラフ
Variable Dependency Graph

スライシングの手順は次の通りである.

1. 検証点とする変数（プロパティやアサーションに記述する変数）を選択する.
2. 選択した変数をソースコードから検索する.
3. 検索した変数を起点とする変数依存関係を解析し，ツリー形式などで明示する.
4. ユーザが変数依存関係上から，除外する変数を指定する.
   除外された変数とその下位変数は，5.でコードから除外される．このため，除外された変数と上位変数とにデータ依存関係または代入関係がある場合，上位変数がモデル境界の変数となる.
5. 変数依存関係に残された変数を含むステートメント,
   およびそれらステートメントを含む関数や制御構文を残す.

検証点とする変数は，スライシング基準における終点の変数に相当する．また境界変数は始点の変数に相当する．変数の除外指定時に，異なる関数に属する変数間で依存関係を切ると，関数単位でのスライシングを実施できる．制約として，変数の含まれないステートメント（特に割込みの許可／禁止や優先度変更など），およびそのステートメントに影響を与えるコードを抽出することは対象外であり，制御フローを別途辿る必要がある.

以上の手順を踏むことの効果として，1. 境界変数は，検証点変数と依存関係を有する変数群，つまり検証点に関係するコードから選択することができる．2. 境界変数の選択により，外部環境との境界を柔軟に調節できる（図 5）．例えば変数依存関係をツリー形式で表示すれば，検証点変数に至るまでの依存関係が一方向になり，境界変数を検証点変数に近く取るほど，ソフトウェアのモデルサイズが小さくなるとわかる．3. 外部環境と境界（インタフェース）となる変数が明確であり，外部環境モデルの新規作成や既存モデルからの選択に有用である．4. 入力値のデータマッピング（値の抽象化）を行う場合，条件式の調整などモデル上でコードの修正を行う必要があるが，データマッピングの影響を受ける変数が明確である.

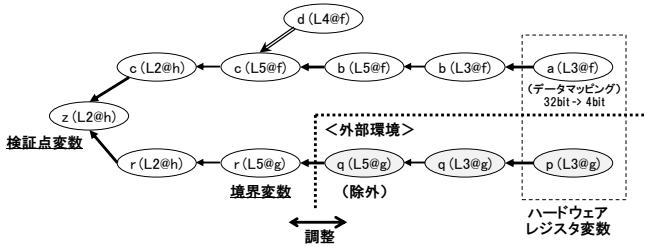

図 5　モデル境界の調整
Adjustment of model boundary

変数依存関係の解析やスライシングは，本稿の適用事例では手作業であるが，計算機で処理を自動化できる．なお本稿ではモデル検査器として SPIN [4][5][6]を用いており，抽出したコードは SPIN のモデル記述言語 Promela へ機械的に変換する.

2.3 非定常・故障動作を有する外部環境モデル

外部環境モデルが必要な理由は，ハードウェアや制御機器対象とソフトウェアの相互作用により誘発される不具合を検出するためであるが，より具体的には，ソフトウェアが不具合を起こす起点となる外部環境の振る舞いを明らかにする，ソフトウェアに影響する外部環境の振る舞いを限定する（例 : 割込みの発生条件），外部環境にて発生する不具合の影響を再現する，などがある.

本検査手法では，外部環境モデルに非定常または故障動作の振る舞いを持たせ，非決定性構文やプロセスによる並列動作によりその動作にランダム性を持たせる.また，各非定常・故障動作の有無を検証毎に切り替える．ある非定常・故障動作があるときに不具合が検出され，ないときには不具合が検出されない場合，その非定常・故障動作が不具合の要因と考えられ，反例の解析時には，その非定常・故障動作を基点に調

Vol.2012-SLDM-155 No.11
Vol.2012-EMB-24 No.11
2012/3/2

査することができる．

またソフトウェアモデルとのインタフェースは，変数依存関係に基づくスライシングにおける境界変数とする．

3. 適用事例

3.1 システム構成

検査対象として，自動車ソフトウェアのうち，コントローラに搭載の電源 IC に対する異常診断処理（図 6）における非再現不具合を選定した．問題の事象は，開発時にごくまれに異常が検出されるも，ソースコード中に不具合が発見されず，原因解明に多大な時間を要していた．

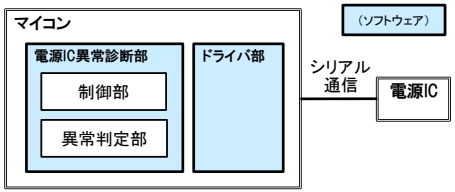

図 6　適用事例のシステム構成（抜粋）
System Structure of Sample Case.

3.2 段階的モデル化

検証は図 7 のように 2 段階に分けた．ステップ 1 ではまず，電源 IC との通信部に焦点を当ててソフトウェア仕様レベルでモデル化した．ここで不具合が検出されなかったので，ステップ 2 では異常診断処理のソフトウェアと電源 IC を全体的にモデル化する代わりに，通信部には不具合がないとして処理を簡略化した．

ステップ 2 のモデル化において，本検査手法を適用した．変数依存関係に基づくスライシングでは，診断結果を示す変数を検証点に選定した．また通信部が省略され，マイコンの電源状態管理が簡略化されるように境界変数を選定した．

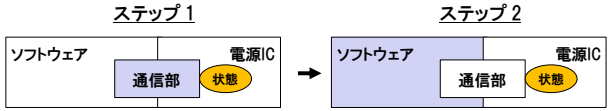

図 7　モデル化と検証のステップ
Steps of Modeling and Verification

3.3 外部環境の非定常動作のモデルと，非再現不具合の原因

外部環境モデルには，マイコンの電源状態として本来のシステムが取る状態遷移のほかに，動作途中で意図せず電源 OFF が入る遷移パスを追加した（図 8）．この電源 OFF は，電源電圧の瞬時低下により誘発されることを想定している．

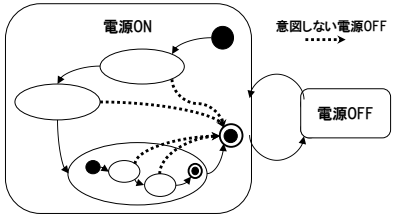

図 8　外部環境モデルの電源状態の遷移（模式図）
Power state transition of external model

モデル検査にて状態数を抑える上で，プロセス数を少なくすることは重要である．ソフトウェアが動作するマイコンと電源 IC とは並行動作するが，異常診断処理は周期処理であるため，外部環境の状態変化は周期処理と同じプロセスにて，周期処理の間に変化するようモデル化した．このモデルで不具合が検出されなければ，外部環境の状態変化を異常診断処理とは別プロセスにする必要がある．また，電源 IC の状態についてデータマッピングを行った．

以上のモデル化により，意図せぬ電源 OFF により誘発されるソフトウェア不具合を 1 件抽出し，そのような電源 OFF が発生した場合には問題とされた非再現不具合の原因となりうることが確認された．表 1 にソースコードとモデルの規模を示す．

表 1　ソースコードとモデルの規模
Size of Source code and model

項目	サイズ
ソースコード全体	約 13 万行
ソフトウェアモデル	約 600 行
システムモデル	約 750 行

不具合発生の仕組みは，電源 OFF により異常診断処理が中断されるのに対し，電源 IC はマイコンの電源 OFF 中も処理が継続するため，電源復帰後に電源 IC の実状態と，

Vol.2012-SLDM-155 No.11
Vol.2012-EMB-24 No.11
2012/3/2

異常診断処理が持つ電源 IC の状態とが不一致になるという現象であった. またこの現象は，ごく短い時間しか出現しない特定状態（変数の値）において，異常診断処理の実行中に電源 OFF が発生したときだけ発生する，というものであった.

このような現象の検出はソースコードのレビューでは難しく, また HILS などのシミュレーションでは，不具合の存在を事前に知らなければテスト条件の設定が困難である．本適用事例から，モデル検査がこのようなタイミング関連の不具合で威力を発揮することを確認でき，また本検査手法の実用性を確認した．本検査手法は，変数依存関係に基づくスライシングのツール化や，外部環境モデルのライブラリ化またはテンプレートからの自動生成などにより省力化され，製品開発プロセスでの実用化が期待できる.

4. 関連研究

ソフトウェアのソースコードをモデル検査する手法およびツールとして，ソースコードをモデル記述言語に明示的に変換（モデル変換）するものと，ソースコードを直接入力とし内部で抽象化するものとがある．前者に属するものとして，Modex/Feaver [7] は C ソースコードに対しスライシングとモデル変換を行う．また Bandera[8]は，Java ソースコードに対し，プロパティに記載された変数やステートメントからスライシング基準を規定してスタティックスライシングを施し，モデル変換を行う．稲盛ら[9]は，ソフトウェアの割込みに特化してソースコードから独自手法によるスライシングを行い，モデル変換を行っている．割込みは不具合要因として非常に重要であるが，本稿の適用事例からもわかるように，ハードウェアが絡む不具合原因は割込みに限らない．後者に属するものとして，青島ら[10]は C ソースコードからメモリリークの検出を行っている．モデル検査器に掛けるソースコードは，検査対象として選定した関数から，一定の呼び出し深さまで関数単位で残すスライシングを行う.

外部環境を含めてソースコードをモデル検査する手法およびツールとしては，BLAST[11]がある．外部環境はソフトウェアのスタブである．Pattabiraman ら[12]は，ハードウェア（プロセッサ）とアセンブラで記述されたソフトウェアについて，ハードウェアにフォルト注入して記号実行とモデル検査を行っている.

悦田ら[13]は，変数依存関係を解析し，Java ソースコードと連携表示することでソフトウェアに対する理解を容易にする手法を提示している．ただし解析を簡易にするため，変数はクラスやメソッド内では記述位置で区別されていない.

以上に対し，本研究はソフトウェア（C 言語ソースコード）とハードウェアや制御機器対象とからなるシステムのモデル検査を行う上で，ソフトウェアと外部環境との境界を柔軟に調整してモデル化することが容易なスライシングと，非定常・故障動作を含む外部環境モデルとを組み合わる検査手法を提案しており，状態爆発を避けつつ不具合原因を含むモデルを構築して検査することの実用化に寄与する.

5. おわりに

本稿ではソフトウェアと，ハードウェアや制御対象機器等からなる外部環境とを合わせたシステムにて，ごくまれに発生しうる不具合を検出する手法を提示した．その手法は, ソフトウェアと外部環境との境界調節が容易なスライシング手法と, 非定常・故障動作を含む外部環境モデルとを組み合わせ，モデル検査を行うものである．また本手法を実機の非再現不具合に適用し，不具合要因を抽出することにより，上記手法の実用性を示した.

今後の課題は，上記スライシング手法のツール化，検証者が境界変数を指定する際の基準作成，非定常・故障動作を含む外部環境モデルのライブラリ化または自動生成による，検査のさらなる容易化である.

参考文献

1) 米田友洋, 梶原誠司, 土屋達弘: ディペンダブルシステム, 共立出版(2005).
2) 中島震: ソフトウェア工学の道具としての形式手法, NII-2007-007J. pp.27-48 (2007).
3) 下村隆夫: プログラムスライシング技術と応用, 共立出版 (1995)
4) G. J. Holzmann: The SPIN Model Checker, Addison-Wesley Pub. (2003).
5) 中島震: SPIN モデル検査, 近代科学社 (2008)
6) 萩谷昌己, 吉岡信和, 青木利晃, 田原康之: SPIN による設計モデル検証, 近代科学社 (2008)
7) G. J. Holtzmann: Model Extraction with Feaver/Modex, SPIN Workshop 2005
8) James C. Corbett, Mattew B. Dwyer et al.: Bandera : Extracting Finite-state Models from Java Source Code, Proceedings of the 22nd International Conference on Software Engineering (2000)
9) 稲森豊, 山田信幸: 静的コード解析による検出漏れのない割込み干渉検出手法の開発, 情報処理学会シンポジウム論文集, Vol.2010, No.10, pp.113-118 (2010)
10) 青島武伸, 伊藤智祥, 春名修介: メモリリーク検出システムλtrace, SEC journal No.11, pp.6-15 (2007)
11) Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar: The software model checker BLAST, Journal on Software Tools for Technology Transfer, pp.505-525 (2007)
12) Karthik Pattabiraman et al.: SymPLFIED: Symbolic Program-Level Fault Injection and Error Detection Framework, DSN2008.
13) 悦田翔悟, 石尾隆, 井上克郎: 変数間データフローグラフを用いたソースコード間の移動支援, Vol.2011-SE-171, No.12 (2011)