Vol.2015-SLDM-171 No.8
2015/5/14

実時間組み込みソフトウェア解析のための HW/SW 協調検査

木村 悠介[1,a)]　ガラバギ アミル マスード[1,b)]　藤田昌宏[2,c)]

概要：組込みシステムのソフトウェアが複雑化する中で、ソフトウェアの検証をより効率よく行うことが重要になっている。割り込み処理は組み込みシステムを特徴付ける機能の 1 つであり、本稿ではこの割り込み処理の検証方法に注目する。通常割り込み処理を含むソフトウェアは、手動でのテストパターン入力によって検証されている。割り込み処理信号を自動でソフトウェア実行中に網羅的に入力して解析するツールはない。本研究では、コンコリック実行ツールを用いて割り込み処理を網羅的に解析する手法を紹介する。また、膨大な実行パス数を削減するために、検証対象の Simulink モデルで得られた入出力データから制約情報を抽出し、それを検証に利用する手法も提案する。実験では、組み込みソフトウェアの実行パスを網羅するテストパターンを実際に生成する。また、制約情報を用いてテストパターン数が大幅に削減でき、効率よくソフトウェアを解析出来ることを示す。

キーワード：実時間組込みシステム, 記号実行, コンコリック実行

Interrupt Analysis for Realtime Software based on HW/SW Co-Testing using Concolic Test

Kimura Yusuke[1,a)]　Gharehbaghi Amir Masoud[1,b)]　Fujita Masahiro[2,c)]

Abstract:
As the software of embedded system becomes more complex, its verification should be much more efficient. Interrupt is one of characteristic functions of real time embedded systems, and the paper is focusing on verification of interruptions. There has been no automatic verification tool to comprehensively analyze interrupts. In this paper, we propose a comprehensive analysis method using Concolic Execution tool. In order to reduce the number of huge test patterns, we extract the restrictons from Simulink model for the target embedded system, and apply them to the verification. We conducted an experiment and gernerated the test patterns to analyze embedded system's interrupts. The experimental results show that the restrictions significantly reduce the number of test patterns and make the verification much more efficient.

Keywords: Realtime Embedded System, Symbolic Execution, Concolic Execution

1. はじめに

組込みシステムは、特定の機能を実現するために装置内に内蔵される電子機器のことである。家電や産業機械には組込みシステムが内蔵されて高度な機能が実現されており、医療機器などでも導入されている [1], P.44。普及が広まっている理由の 1 つに、これまで機械的に実現されていたものが電子制御に置き換わることで、開発効率や動作の

[1] 東京大学大学院工学系研究科電気系工学専攻
Dept. of Electrical Engineering and Information Systems, The University of Tokyo
[2] 東京大学大規模集積システム設計教育研究センター
VLSI Design and Education Center, The University of Tokyo
[a)] kimura@cad.t.u-tokyo.ac.jp
[b)] amir@cad.t.u-tokyo.ac.jp
[c)] fujita@ee.t.u-tokyo.ac.jp

柔軟性が向上することが挙げられる。特に、実時間組込みシステムは決められた時間通りに応答するよう設計されたシステムであり、自動車のエンジン制御や医療機器の制御など人命に関わるシステムの制御に利用されている。

組込みシステムの利用が広がり、実現するべき機能が増えて高機能・多機能化が進むほど、組込みシステムに含まれる不具合も問題になっている。しかし組込みシステムを設計する企業には、製品をなるべく早く市場に投入したいという意向があり、システムの信頼性解析に時間をかけすぎると収益機会を逃してしまうことになる。市場投入までの時間の短縮化と十分な信頼性解析という相反する目標を達成するためには、信頼性解析にかかる時間の短縮技術を開発することが重要である。例えば、仕様の設計段階から不十分な点や矛盾した部分を取り除くために、モデル設計が行われている。また、実際にハードウェア上で検証を行う際に、テストパターンを入れ替えたり圧縮することで検証が素早く行えるようにする技術などがある。

本研究は、組込みシステムのソフトウェアの信頼性解析を対象し、割り込み処理を含むソフトウェアのテストパターンを生成・削減する手法を提案する。なお、組込みシステムソフトウェアでは様々なプログラミング言語が用いられているが、本研究では最も広く用いられている C 言語を対象とする [1], P.84。

2. 組込みソフトウェアの開発フローと割り込み処理を含むソフトウェアの既存解析手法

ソフトウェアのバグは多種多様である。IPA（情報処理推進機構）がまとめている脆弱性データベースでは、脆弱性を 23 に分類してバグの管理を行っている [2]。割り込み処理の場合に特に問題となるのは競合状態、リソース管理などといったものであるである。競合状態とは同一リソースに複数のプロセスやスレッドなどが同時にアクセスしてしまうことが原因となる問題である。リソース管理に関する問題とは、メモリーなどの資源の不足により発生する問題のことである。

例えば、多重割り込みを許すシステムで、割り込み時に実行される関数内で新たな変数を宣言したりメモリーを確保する関数を使用していると、たくさんの割り込み処理を行った場合にメモリーが足りなくなりソフトウェアが正常に動かなくなる。これはリソース管理に関連する脆弱性と言える。

2.1 静的解析

静的解析ツールは、与えられたプログラム中の配列の範囲外アクセスや NULL ポインタ参照、未初期化変数発見などの誤りを基本的にプログラムを実行せずに発見するためのツールである。発見できるバグは多岐にわたるが、競合状態やリソース管理の問題については静的解析では発見し切れない場合がある。また、一般に静的解析ツールは、False Warning と呼ばれる、プログラムは実際には間違っていないにも関わらず、エラーメッセージを出してしまう場合も少なくない。

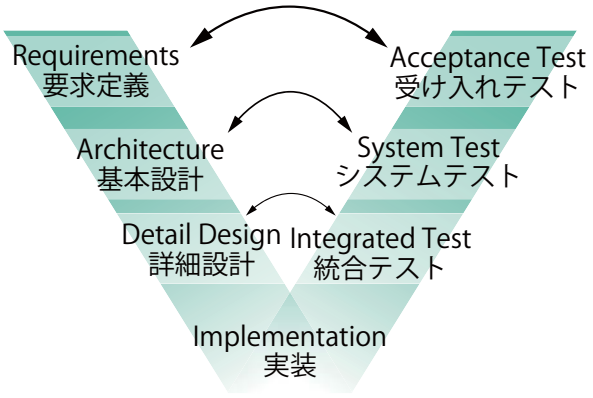

図 1　組込みシステム開発の V 字モデル

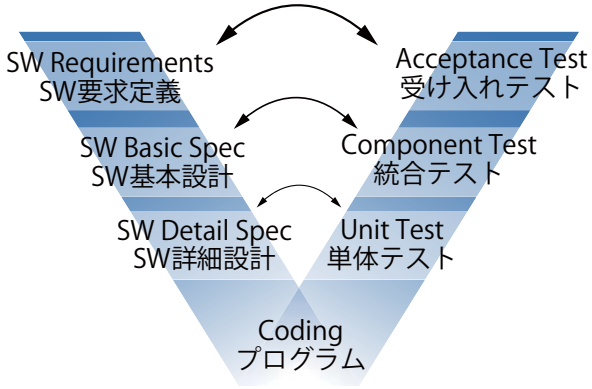

図 2　組込みソフトウェア開発の V 字モデル

ソースコード 3　割り込みモデル化記述

```
1 /* 割り込み信号の有無を確認し、
      interrupted 変数に代入 */
2 scanf("%d", &interrupted);
3
4 /* もしinterrupted=1ならば､割り込み処理
      interrupt_handler 関数を実行 */
5 if (interrupted){
6     interrupt_handler();
7 }
```

組込みソフトウェアのテストでも静的解析は活用される。しかし、アセンブリ記述を含むコードは静的解析ツールでは通常認識できないため、利用には工夫が必要である。既存の静的解析ツールで割り込み処理も含めて解析する方法の一つに､。割り込みのモデルを C 言語で記述するというものがある [3]。

静的解析の問題点として、その精度がある。解析ツールが C 言語レベルでの解析を行う場合、1 つの C 言語記述は

複数の命令に分割される場合があり、割り込みモデル化記述ではその命令の途中で割り込み処理が入った場合の解析ができない。

2.2 動的解析

静的解析では実際にプログラムを実行しないので、検出できるバグには限りがある。例えば、メモリーリークなどのバグは、実際に実行してはじめて問題が表面化することがある。近年ソフトウェアはマルチコア・マルチスレッド対応が進んでいるが、デッドロックや競合状態を見つけるには実際にプログラムを実行しないと難しい。また、ソフトウェア内に複雑なポインタ参照がある場合、ポインタが何をさしているか不明なこともある。このように、実際にプログラムを実行して解析する手法のことを動的解析と呼ぶ。

組込みシステムの開発は図 1 のように V 字型に表すことが出来る。動的解析では、テストするに当たって、入力すべき値のセットとそれに対応する望ましい応答を事前に定義する必要がある。理想的な組み込み開発現場の設計フローが V 字型で表されるのは、前述のテスト仕様定義とテスト実行が左右の矢印で対応していることを表すためである。例えばソフトウェアの単体テストで行われるテストは SW 詳細設計時に策定されたテスト方法を利用する。このような計画的なテストを行うことで、動的解析は効果的な検証手法となる。

問題点として、動的解析ツールには割り込みを網羅的に解析するものはない点が挙げられる。そもそも組み込みソフトウェアの検証は人手によって生成されたテストパターンに対する応答を調べることで行われることが多く、テストパターン生成が自動化された手法がとられることはない。このため、バグの発見が遅れたり、直されないまま製品が出荷される可能性がある。クラスや関数などの小さな単位で検証で、後述の Concolic Execution ツールが用いられることはあるが [4]、割り込み処理の検証にまでは用いられていない。

また、動的解析は、静的解析と違い入力テストパターン数によって検査の質が変化する。必要十分なテストを行うのが難しい点も問題である。

3. Concolic Execution

3.1 Symbolic Exectuion

Symbolic Execution(記号実行)[5] は、ソフトウェア動作の網羅的な解析のための手法の 1 つである。Symbolic Execution の基本的なアイデアは、ソフトウェアに入力する変数に具体的な値を利用するのではなく、抽象的な記号を利用するというものである。

Symbolic Execution では、条件分岐の条件などから特定の数式が生成され、それが成立するか否かをチェックすることが繰り返される。条件成立の可否を様々に組み合わせることで、可能な限りたくさんの種類の実行パスをとる入力値の条件を生成することができる。

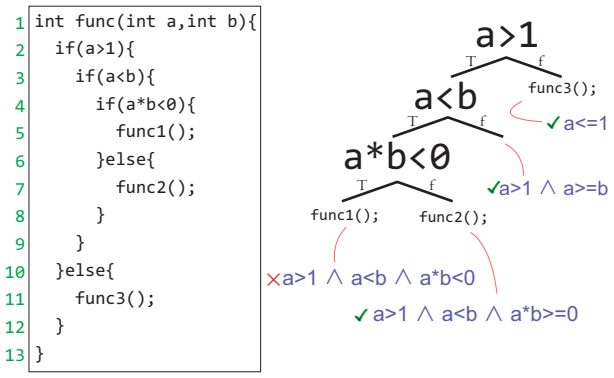

図 4 Symbolic Execution

例えば、図 4 の左側に示したプログラムをもとに、右側に示したような木構造を考えることができる。各結節点の条件の True/False を考えて AND をとると、青文字の数式が 4 つ得られる。この 4 つの数式の成立可否を調べ、候補となる a,b の値を 1 つずつ列挙すれば、パスカバレッジが最大になるようなテストパターンが得られる。図 4 の例では、$a > 1 \wedge a < b \wedge a * b < 0$ という条件式は成立しないので、それ以外の数式についてはパターンが生成できる。$a > 1 \wedge a < b \wedge a * b >= 0$ に対しては $a = 2, b = 3$ などが得られる。

3.2 Concolic Execution

Concolic Execution[4] はソフトウェアテスト手法の 1 つである。なお concolic は”concrete” と”symbolic” の 2 語を合成した造語である。前項で紹介した Symbolic Execution は高いブランチカバレッジと少ないテストパターンを両立する手法であるが、ここで得られたテストパターンを自動でソフトウェアに入力し、実際にどのようなパスを通るのかを調べる手法が Concolic Execution である。このため、Symbolic Execution のみを用いる場合と比べて、より実行パスの網羅性が向上する。

Concolic テストツールは、内部の形式的な解析に SMT ソルバーを用いることが多い。Concoclic テストツールは条件分岐や実行パスの監視を行い、その情報をもとに SMT ソルバが次の入力パターンを生成するということを繰り返すことで、実行パスを網羅的に解析することを実現している。実験で用いる CREST[6] は SMT ソルバーには Yices[7] を用いている。

4. 提案するテスト手法

4.1 動的解析における割り込み処理のモデル化記述

組込みシステムモデルを用いる場合、CPU の動作をソフトウェア上で再現する必要がある。組込みシステムで使

ソースコード 5　動的解析のための割り込みモデル化記述

```
1  void interrupt_handle(int isSig[NUM]){
2    int i,pri;
3    for(pri=0;pri<NUM;pri++){
4      for(i=0;i<NUM;i++){
5        if((priority[i]==pri)*isEnabled[i]*
             isSig[i]){
6          func[i]();
7          goto CLEAR;
8    }}}
9   CLEAR:
10 }
```

われる CPU は仕様が様々であるため、使う CPU の種類ごとにその動作を再現することは大変なことである。また、テストの実行速度の観点から CPU をモデル化せずに、x86_64 アーキテクチャ向けにコンパイルしたバイナリファイルを用いることもあるが、このような場合、各アーキテクチャ特有のアセンブリ記述は実行できない。

そこで、ソースコード 5 のような動的解析のための割り込み処理のモデル化記述を提案する。これを用いることで、動的解析ツールや Concolic ツールのような既存のツールで割り込み処理を扱うことが可能となる。

上述のモデル化記述は ARMv7 や SuperH などのアーキテクチャよりも単純である。実際には割り込みマスクや割り込み保留ビットなどのレジスタを考慮したモデルにする必要がある。

4.2　テストパターン削減手法

4.2.1　通常使用時の信号値解析による基準信号解析

本研究では、基準となる割り込み信号を決め、その間に、割り込みがどのように入るか調査するものとする。実デバイスやソフトウェアモデルで観測される信号を監視することで、規則を抽出する。基準とする信号の選定には、回数や規則性、優先度などが尺度として考えられる。

基準とする割り込み信号をもとに、以下を抽出することとした。

- 割り込みの有無
- 割り込みの種類
- 割り込みの回数・頻度

ネストの深さや、基準信号間の割り込みの回数の制限を Concolic テストツールに与えることにより、テストパターンの削減が可能である。ただし、解析によって得られた情報は、人が確認して採用・不採用を決める必要がある。割り込みが発生する回数は少ないものの重要な割り込み信号がある場合などに対応するためである。

4.2.2　解析したい変数に特化したテストパターン生成

組込みシステムのソフトウェア検証で、問題のある箇所が既に分かっているが原因が見つからない場合がある。このような場合に活用できる方法として、特定の変数と関連する割り込み信号のみを解析し、それ以外の割り込み信号を無視するという解析手法を提案する。

まず、組込みソフトウェアを解析し、変数・関数間の依存関係を調査する。調査は次のような手順をとればよい。

(1) 解析対象の変数を変更する関数を記録する。対象の変数への代入文が条件分岐内にある場合、条件分岐を決定する変数も記録する。
(2) 手順 1 で得られた条件分岐を決定する変数について、解析対象の変数と同様に 1 の作業を行う。
(3) 解析対象の変数がなくなった後、割り込みベクタに登録された関数と、割り込み関数内で呼び出される関数のリスト 1 を用意する。
(4) リスト 1 に含まれた関数で、手順 2 までで得られた変数の値を変更するものをリスト 2 に登録する。
(5) リスト 2 に含まれる関数を実行する割り込み関数が、解析すべき割り込み関数である。

このような関係のない割り込み関数がある場合には、前節に表示したソースコード内の「割り込み種類ごとの回数制限」の該当箇所を 0 にすればよい。これを Concolic テストツールで実行して、テストパターンを生成する。

表 1　混入させたバグ

バグ	内容	検出するための記述
1	割込優先度のミス	assert(error_sig==1 && motor==1);
2	変数の指定ミス	assert(limit_sugar > sugar);

5.　実験による提案手法の評価

5.1　インスリンポンプ

5.1.1　システムの概要

インスリンポンプとは、糖尿病患者に自動でインスリンを注入する医療機器である [8]。インスリンポンプに搭載された電子機器の内部構成は、Amrita 大学で作成されたインスリンポンプを参考にした [9]。

この機器は主に 2 つのコントローラと周辺機器から成り立っている。1 つ目の LSI はポンプのモータを制御するコントローラであり、もう 1 つはセンサーや外部入力を制御するコントローラである。本研究では、これらの情報を参考に MATLAB/Simulink 上でインスリンポンプモデルを作成し、モデルは図 6 のようになった。

本研究ではセンサー・UI を制御するコントローラのソフトウェアを検証対象とするものとする。コントローラで実行されるソフトウェアは、5 種類の割り込み信号を持ち、200 行程度の C 言語プログラムである。なお、ソフトウェア中には表 1 のように意図的にバグを混入させた。更に、各バグに対応する assert 文も挿入した。

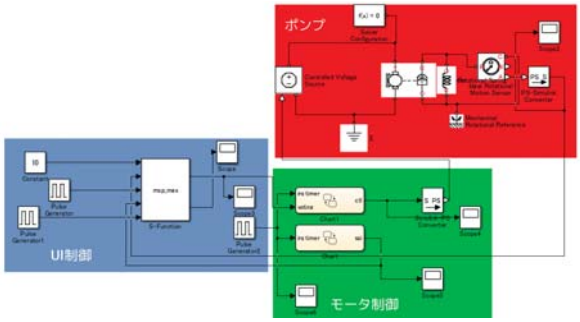

図 6　Simulink モデル

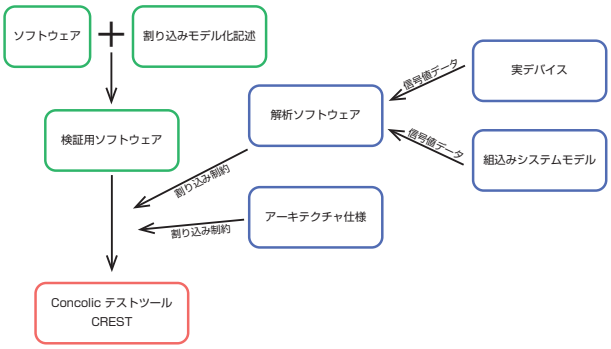

図 7　テストパターン削減手法

5.1.2　実験の方法

実験では、図 7 のような流れでテストパターンを作成する。まずソフトウェアを Concolic テストツールで解析可能な形に修正し、アセンブリ記述の代わりとして後述の割り込みモデル化記述を挿入する。本実験では、C 言語記述 1 行ごとにモデル化記述を挿入した。これと並行して、実デバイスやモデルを用意して組込みシステムを再現する。本研究では Simulink 上にモデルを作成した。この環境で組込みシステムを一般的な使い方で動作させ、LSI の信号の入出力を記録する。この結果を解析すると、割り込み処理に関して一定の規則や頻度を抽出することができる。これを踏まえて Concolic テストツールを利用すればテストパターンを削減することが可能である。Concolic テストツールとしては、CREST を利用した [6]。

5.1.3　結果

Simulink モデルを実行して検証対象のソフトウェアに入力される割り込み信号を記録し、規則を抽出した。基準信号は、タイマー割り込み信号とした。

- 割り込み処理 1(電源管理) と割り込み処理 2(設定管理) は同時に発生しない＜ルール 1 ＞
- 割り込み処理 4(エラー信号) はない＜ルール 2 ＞
- 2 つの基準信号の間には最大 2 つの割り込みが発生。＜ルール 3 ＞
- 2 つの基準信号の間で、同じ割り込みが発生することはない。＜ルール 4 ＞

このうちルール 2 は、検証の上では不適切である。よって他 3 つの制約条件を用いることとする。

表 2　様々な条件下での基準信号間のテストパターン数

割り込み回数	パターン数	Bug1	Bug2
1	50	×	○
2	2279	○	○
3	105294	○	○
4	5142351	○	○
提案手法 2 (各 1)	1659	○	○

表 3　特定の変数に着目したテストパターン生成

変数	関連割込信号	パターン数	Bug1	Bug2
なし	5 つ	1659	○	○
変数 1	1 つ	1	×	×
変数 2	3 つ	376	×	○
変数 3	5 つ	1659	○	○

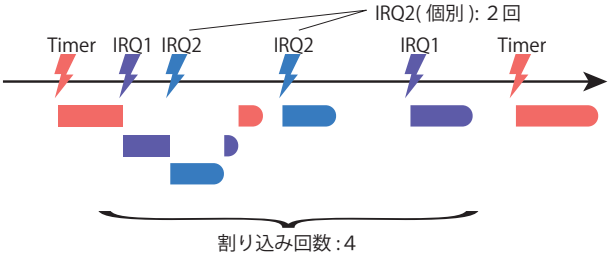

図 8　各パラメータの説明

ここで、基準信号間の信号数と、基準信号間の各割り込み信号数に様々な上限を設定し、テストパターン数を計測した。図 8 にパラメータの説明を附した。結果は表 2 の通りである。基準信号間の割り込み数に着目し、提案手法時のみ各割り込み信号は 1 回のみとしている。Bug1,Bug2 はバグが発見出来たかどうかを示している。

さらに、解析したい変数をランダムに選び、提案手法を適用した。結果は表 3 のとおりである。

表 2 からは、提案手法の 1 つである割り込みモデル化記述のみであると、テストパターン数が膨大になってしまうことがわかる。基準信号間の割り込み回数が 5 回以上になると、検証出来ないほどたくさんのテストパターンが生成される。逆に割り込み回数が小さすぎるとパターン数が少なくはなるがバグが発見出来ないことになる。提案手法は、合理的な制約情報に基づいて必要最小限のテストパターン数を実現していることが分かる。表 3 では、特定の変数に着目すれば、当然テストパターン数が減少することが分かる。

5.2　飛行船制御プログラム

5.2.1　システムの概要

もう 1 つの例題として飛行船制御プログラムを用意した。これはルネサスエレクトロニクスの M16C マイクロプロセッサプロセッサで動作することを前提に書かれたプログラムである。コンパイラはルネサスエレクトロニクスの

NC30 を前提としている。

ソフトウェアは C 言語で 4000 行程度で記述されており、6 つの外部割り込み信号を持つが、ソフトウェアにより多重割り込みは禁止するものとなっている。

5.2.2 結果

飛行船制御プログラムでは、Concolic テストツールを正常に動作させることができず、「予測不能」というエラーメッセージに対応することができなかった。

飛行船制御プログラムには、UART 通信部などにビット演算が多用されており、これらの計算に Concolic ツールが対応していないためである。また、変数宣言部にビットフィールドが多用されており、これも予測不能となってしまう原因である。

6. 結論と今後の課題

本研究では、通常使用時の入出力データを活用して、組込みシステムソフトウェアのテストパターンを削減する手法を考案した。パターンの生成には Concolic テストツールを用いるが、そのツールが割り込みを処理できるよう動的解析における割り込みモデル化記述も提案した。これらの提案手法を用いて生成したテストパターン数は、従来手法と比べて数百分の一と非常に少ないことが確認できた。なお、これらのパターンでは多重割り込みも考慮されている。さらに、解析したい変数に特化したテストパターン生成を提案した。これは、解析したい変数に影響を及ぼさない割り込み信号をマスクすることで、テストパターン数を削減する手法である。変数によってはテストパターンを削減することが可能であることが確認できた。

従来の網羅的な解析ではパターン数が膨大になってしまうという問題があったが、これらの提案手法を用いることでテストパターンを削減することができた。また、通常使用時のデータの解析結果から得られる複数の制約条件は、採用・不採用を人手で判断することができ、テスト精度とテスト時間のトレードオフを提供するという点でも有用である。

今後の課題として、割り込みモデル化記述の挿入箇所の削減が挙げられる。プログラム内の変数同士の依存関係を調べ、不要なモデル化記述を除去する仕組みを実装したい [10]。さらに、ビット演算などの特殊な演算を Concolic テストツールで扱うことが出来ないという点が分かったので、Concolic ツールの改修などで対応したい。ソフトウェアによっては複雑な四則混合演算を行うことがあるが、これも既存の Concolic ツールは扱えないことが分かっているので、同様に対応したい。

参考文献

[1] 経済産業省：平成 22 年度中小企業システム基盤開発環境整備事業事業報告書，http://www.meti.go.jp/policy/mono_info_service/joho/chusho_ESIR/2011/01.pdf.
[2] 独立行政法人情報処理推進機構：IPA 共通脆弱性タイプ一覧 CWE 概説，http://www.ipa.go.jp/security/vuln/CWE.html.
[3] 都井　敬：ソフトウェアにおける割り込み処理に対する静的解析に関する研究，東京大学電気系卒業論文 (2014).
[4] Godefroid, P., Klarlund, N. and Sen, K.: DART: directed automated random testing, *ACM SIGPLAN Notices*, Vol. 40, No. 6, p. 213 (2005).
[5] King, J. C.: Symbolic execution and program testing, *Communications of the ACM*, Vol. 19, No. 7, pp. 385–394 (1976).
[6] Burnim, J. and Sen, K.: Heuristics for Scalable Dynamic Test Generation, *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, IEEE, pp. 443–446 (2008).
[7] Dutertre, B.: Yices 2.2, *Computer-Aided Verification (CAV'2014)* (Biere, A. and Bloem, R., eds.), Lecture Notes in Computer Science, Vol. 8559, Springer, pp. 737–744 (2014).
[8] 宣彦斎藤：ここから始める糖尿病レクチュア，文光堂，東京 (2014).
[9] Civera, P., Macchiarulo, L., Rebaudengo, M., Reorda, M. and Violante, M.: FPGA-based fault injection for microprocessor systems, *Proceedings 10th Asian Test Symposium*, IEEE, pp. 304–309 (2001).
[10] 稲森　豊，山田信幸：検出漏れの無い割込み干渉検出システムの開発，*SEC journal*, Vol. 7, No. 1, pp. 6–9 (2011).