Vol.2009-SE-166 No.19
2009/11/6

組込みソフトウェアのテストを目的とした CPU エミュレータ上での異常注入手法

東　誠[†1]　山本哲男[†2]　早瀬康裕[†1]
石尾　隆[†1]　井上克郎[†1]

組込みソフトウェアの例外処理は，そのソフトウェアが外部デバイスから受け取った入力が特定の条件を満たす際に，その入力を検知して対処する．この例外処理が対処できない入力によって，組込みソフトウェアが正常に動作しないという故障が発生することがある．本研究では，例外処理の効率的なテストを目的として，故障を引き起こす可能性のある入力を組込みソフトウェアに注入する機構を CPU エミュレータに加えることで，組込みソフトウェアに対する異常注入を実現する手法を提案する．具体的には，ソフトウェアが入力を受け取る外部デバイスや，その外部デバイスから受け取る値をソフトウェアの実行前に予め指定しておく．そして，ソフトウェアの実行中に，指定内容に従って，外部デバイスから受け取る入力値の代わりに新しい入力値を設定することにより，ソフトウェアに故障を引き起こす入力を注入する．提案手法を適用して，実際のソフトウェアに故障を引き起こす可能性のある入力を注入したところ，異常注入手法なしには実行が困難な例外処理を実行できることが確認された．

A Method of Fault Injection on a CPU Emulator for Embedded Software Testing

Makoto Higashi,[†1] Tetsuo Yamamoto,[†2]
Yasuhiro Hayase,[†1] Takashi Ishio[†1]
and Katsuro Inoue[†1]

Exception handling in embedded software detects and deals with inputs from I/O devices when those inputs meet conditions of the exception handling. When exception handling cannot deal with an input, the input sometimes causes a failure of embedded software. In this paper, we suppose a method of fault injection by adding the mechanism to inject faults to a CPU emulator for efficient testing of exception handling. At first, before software running, a user sepcifies devices and values which may causes a fault in advance. Then, during softwarerunning, values of inputs from the devices are ignored, and new values are setted as inputs according to the configuration. Applying the method to an open software project shows that CPU emulator with the mechanism to inject faults can execute exception handling that is hard to be executed without the mechanism to inject faults.

1. はじめに

近年，携帯電話やペースメーカーといった組込みシステムが開発されている．組込みシステムとは，システムの外部から受け取った情報を変換する外部デバイスと，その外部デバイスから入力を受け取って処理を行う組込みソフトウェアで構成されているシステムである．

このような組込みシステムには高い信頼性が求められるものがある．例えば，携帯電話などのネットワークを利用する組込みソフトウェアでは，適切なネットワークアクセス許可機能が実装されていないと，個人情報が第三者から読み出されるといったセキュリティ上の問題が発生する．別の例として，もしペースメーカーが動作を停止すると，そのペースメーカーの利用者が人命の危機に陥ってしまう．そのため，これらの組込みシステムの信頼性は可能な限り向上させる必要がある．

組込みシステムの信頼性を向上させる方法の 1 つとして，組込みソフトウェアが特定の状況で特定の外部デバイスから入力を受け取った際に，組込みソフトウェアが正常に動作しなくなるという故障への対処が挙げられる．例えば，携帯電話である FOMA P901i[7] では，着信時に終了キーを連続押下した場合などに，稀に着信履歴が表示されない，または同じ着信履歴が 2 回表示されるという故障が報告されている．また，同じく携帯電話である FOMA D901i[6] では，メール作成中に,「題名」を入力した直後に素早く「本文」にカーソルを合わせて「本文」を入力すると，操作ができなくなる場合があるという故障が報告されている．上記のような組込みソフトウェアの故障を引き起こす可能性のある入力を，以降は**異常な入力**と呼ぶことにする．

このような故障が発生する原因として，組込みソフトウェアに適切な例外処理が実装され

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 立命館大学総合理工学院情報理工学部
Ritsumeikan University Institute of Science and Engineering College of Information Science and Engineering

ていないことが挙げられる．例外処理とは，外部デバイスから受け取った入力が特定の条件を満たすならその入力を異常な入力として検知し，各入力の内容に応じて適切な対処を行う枠組みである．しかし，対処すべき入力が例外処理で検知できなかったり，入力に対する例外処理の内容が不適切だった場合，組込みシステムに故障が発生する．そのため，組込みソフトウェアの例外処理は重点的にテストされる必要があると考えられる．

しかし，外部デバイスから受け取る入力に対する例外処理のテストは困難である．その理由は 3 つ挙げられる．1 つは，例外処理で対処されるべき入力は発生頻度が低いため，その入力を用いたテストを実行するのは困難だからである．2 つは，組込みソフトウェアが外部デバイスから受け取る入力値と，それを受け取る状況の組み合わせの数は膨大であるため，全ての組み合わせを網羅してテストすることが困難だからである．3 つは，例外処理は他の機能に比べて注意が払われ難い[8] ため，例外処理に重点を置いたテストが行われ難いからである．

そこで本研究では，例外処理の効率的なテストを目的として，ソフトウェアに異常な入力を注入する機構を CPU エミュレータに加えることで，組込みソフトウェアに対する異常注入を実現する手法を提案する．具体的には，ソフトウェアが入力を受け取る外部デバイスや，その外部デバイスから受け取る値をソフトウェアの実行前に予め指定しておく．そして，ソフトウェアの実行中に，外部デバイスから受け取る入力値を指定内容に従って変更することにより，ソフトウェアに異常な入力を注入する．このような注入を行う機構を CPU エミュレータに加えることで，その CPU エミュレータ上で動作する様々なソフトウェアに異常な入力を注入することができるようになる．

この機構を物理 CPU ではなく CPU エミュレータに加える理由は 2 つ挙げられる．1 つは，外部デバイスと同時にソフトウェアを開発する際には外部デバイスが未完成の時にもテストをするため，組込みソフトウェアのテストには CPU や外部デバイスのエミュレータを用いることが多いからである．もう 1 つは，ハードウェアよりもソースコードの方が加工が行い易いという点で，物理 CPU よりも CPU エミュレータの方が異常注入機構を加えるのが容易だからである．

以降，2 章では CPU エミュレータに異常注入機構を加えることで異常な入力の注入を実現する手法を提案する．3 章では提案手法を実装した CPU エミュレータについて説明し，4 章ではその CPU エミュレータを用いた実験について説明する．そして 5 章では関連研究を紹介する．最後に，6 章で本研究のまとめと今後の課題について述べる．

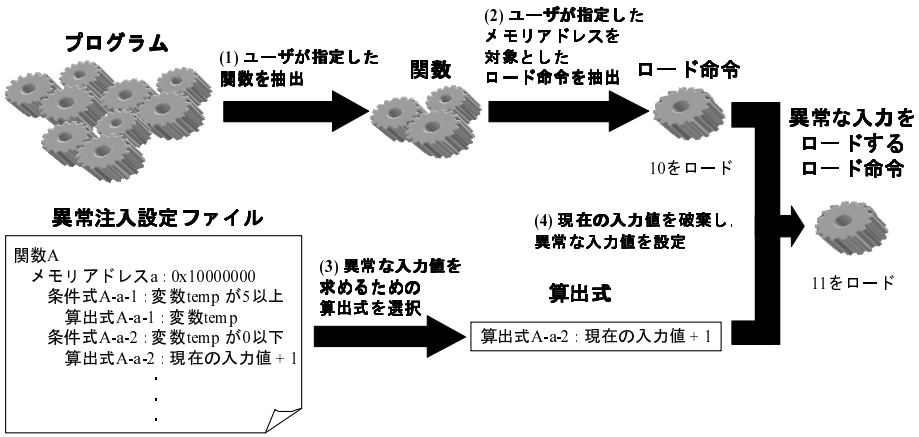

図 1 異常な入力を注入する手順
Fig. 1 the process of fault injection.

2. 提案手法

本章では，CPU エミュレータに異常注入機構を加えることで異常な入力の注入を実現する手法について説明する．

異常注入機構を実装する箇所は，CPU エミュレータがロード命令を実装している箇所である．その理由は，本研究で想定している CPU エミュレータが外部デバイスから入力を受け取る手段がメモリから値を読み出すロード命令だからである．他に CPU エミュレータが外部デバイスから入力を受け取る手段としては，入出力ポートから値を読み出すイン命令が挙げられるが，その場合でも同様の方法で実装が可能である．

ロード命令の実装箇所に以下の手順で処理を行う機構を実装することで，CPU エミュレータ上で動作している組込みソフトウェアに異常な入力を注入することができるようになる．

(1) 外部デバイスなどのユーザが指定した条件が成立しているか判断する
(2) 外部デバイスから受け取った現在の入力値を破棄し，その代わりとして異常な入力値をソフトウェアに受け取らせる

このように異常注入機構が異常な入力を注入する手順の概要を**図 1** に示す．

図 1 のように，異常注入機構の入力は CPU エミュレータ上で動作させるプログラムと，異常注入設定ファイルである．異常注入設定ファイルとはプログラムの実行前に，異常な入

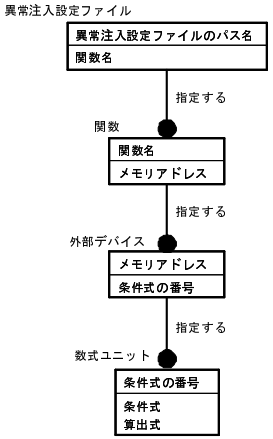

図 2 異常注入設定ファイルの構造
Fig. 2 a structure of configure file to inject faults.

力を注入する条件や注入する入力値をユーザが指定するためのファイルである．異常注入設定ファイルには，以下の情報が記述されている．

- 関数名
- メモリアドレス
- 算出式:異常な入力を計算するための数式．そのオペランドとして利用される値は定数や現在の入力値，プログラムで利用されている変数の値である
- 条件式:利用する算出式を選ぶための数式．そのオペランドとして利用される値は算出式と同様である

また，l これらの情報が異常注入設定ファイルに記述される構造を IDEF1X 表記の ER 図で表すと，**図 2** のようになる．

異常注入機構に対する各入力の用途については 2.1 節以降にて説明する．

- 関数名
- メモリアドレス
- 算出式:異常な入力を計算するための数式．そのオペランドとして利用される値は定数や現在の入力値，プログラムで利用されている変数の値である
- 条件式:利用する算出式を選択するための数式．そのオペランドとして利用される値は算出式と同様である

そして，図 1 のように，異常注入機構が異常な入力を注入する手順は以下のようになっている．

(1) 異常注入機構に入力されたプログラムから，異常な入力の注入を行う条件として指定されている関数を抽出する
(2) 抽出した関数から，ユーザが指定したメモリアドレスから値を受け取るロード命令を抽出する
(3) 異常注入機構に入力された異常注入設定ファイルから，異常な入力値を算出するための算出式を選択する
(4) (2) で抽出したロード命令のオペランドを破棄し，その代わりに (3) で抽出した算出式の結果をロード命令の新しいオペランドとして設定する

以降，各手順の詳細について説明する．

2.1 指定された関数の抽出

まず，異常注入設定ファイルに記述された関数名を持つ関数を異常注入機構に入力されたプログラムから抽出する．そして，その関数が実行されている間のみ，異常な入力を注入可能にする．

このように，異常な入力の注入が可能な状況を指定された関数が実行されている間のみに限定する理由は 2 つ挙げられる．1 つは，異常な入力が注入可能な全ての状況で異常な入力を注入すると，ソフトウェアの動作が極端に遅くなったり，ユーザにとって望ましくない状況で異常な入力がソフトウェアに注入されたりするという弊害が発生するからである．もう 1 つは，特定の機能を実行している時に異常な入力を受け取ると組込みソフトウェアの故障が発生するケースが報告されている[7] ため，1 つの機能に該当する関数は異常な入力をソフトウェアに注入する状況を指定する条件として適切だと考えられるからである．

このような関数を異常注入機構に入力されたプログラムから抽出する方法は，以下の通りである．まず，異常注入設定ファイルに記述された関数名を抽出し，その関数がロードされているメモリのアドレスの範囲を調べる．そして，このアドレスの範囲に現在のプログラムカウンタの値が含まれている場合，異常な入力が注入可能な関数が実行されていると判断する．

2.2 指定されたロード命令の抽出

次に，異常な入力値を設定するためのロード命令を，2.1 節で抽出した関数から抽出する．このロード命令を抽出するには，関数で実行されるロード命令のロード元となっている外部

デバイスを特定し，それが異常注入設定ファイルに記述されているかを判断する．

本研究では，ロード元となる外部デバイスをメモリアドレスで指定する．その理由は，本研究で想定している CPU エミュレータが外部デバイスから入力を受け取る手段がメモリから値を読み出すロード命令だからである．他に CPU エミュレータが外部デバイスから入力を受け取る手段としては入出力ポートから値を読み出すイン命令が挙げられるが，その場合はポート番号を指定する．

2.3 異常な入力値の算出に必要な算出式の選択

次に，異常な入力値を算出するための算出式を，異常注入設定ファイルから選択する．この算出式を選択するには，その算出式に対応する条件式を評価し，それが成立するか判断する．図 2 のように，条件式と算出式は 1 対 1 の対応関係にあるため，条件式の内容が成立すれば，それに対応する算出式を異常な入力値を算出するための数式として選択できる．

条件式のオペランドには定数と現在の入力値，プログラムで利用されている変数の値が利用可能である．このうち，現在の入力値を条件式のオペランドに利用することで，特定の入力値を異常な入力値に確実に変更することが可能になる．また，プログラムで利用されている変数を条件式のオペランドに利用することで，プログラムの状況をより詳細に考慮して異常な入力を注入する条件を指定することが可能になる．例えば，以前の入力値を格納する変数があれば，その変数を条件式のオペランドとして利用することで特定のシーケンスで入力を注入することが可能になる．

2.4 異常な入力値の計算と設定

最後に，2.2 節で抽出したロード命令のオペランドを破棄し，その代わりに 2.3 節で選択した算出式の結果をロード命令の新しいオペランドとして設定する．

算出式のオペランドには定数と現在の入力値，プログラムで利用されている変数の値が利用可能である．このうち，現在の入力値を算出式のオペランドに利用することで，現在の入力値と確実に異なる値を異常な入力値として設定できる．また，プログラムで利用されている変数を算出式のオペランドに利用することで，外部デバイスから受け取った入力値による条件分岐をプログラムが含む場合，特定の分岐を実行することが可能になる．

3. 異常注入機構の実装

本研究の実験では異常注入機構を CPU エミュレータの skyeye[11] に加えた．この異常注入機構に必要な各機能は，以下の方法で実装した．

- 関数に対応するメモリのアドレスの範囲を求める機能:dwarf2[3] を利用し，CPU エミュレータ上で動くソフトウェアの実行前に異常注入設定ファイルで指定された関数名を持つ関数に対応するメモリアドレスの範囲を取得した．そして，この指定された関数名を持つ関数が実行されているかを判断する機能を，ソースファイル skyeye/arch/arm/common/armemu.c で記述されている ARMul_Emulate32 関数と ARMul_Emulate26 関数でプログラムカウンタの値を更新する箇所に実装した
- 異常注入設定ファイルで指定されたロード命令を抽出し，そのロード命令に新しいオペランドを設定する機能:ロード命令の抽出やそれに新しいオペランドの設定を行う関数を C 言語で独自に作成した．そして，その関数を，ソースファイル skyeye/arch/arm/common/armemu.c で記述されている LoadWord 関数や LoadHalfWord 関数，LoadByte 関数，LoadMult 関数，LoadSMult 関数から呼び出すことで，異常な入力の注入を実現した．なお，この独自に作成した関数の引数として外部デバイスから受け取った現在の入力値を用いることで，その入力値を条件式や算出式のオペランドとして用いることを可能にした
- 異常注入設定ファイルで指定された条件式，算出式を構文解析して数式として利用する機能:yacc と lex[12] を利用し，文字列を数式として解釈する構文解析プログラムを作成した
- プログラム中で利用されているグローバル変数の値を取得する機能:dwarf2[3] を利用しており，プログラムの実行前に各グローバル変数のアドレスを取得することで，そのアドレスからプログラムの実行中にグローバル変数の値を取得することを可能にした．なお，ローカル変数の値を取得する機能を実装していない理由は，ローカル変数のスコープを管理するのが困難だからである。ローカル変数はプログラムから利用できる状況が限られているため，その状況を把握した上で値を取得しないと，誤った値をローカル変数の値として取得する危険があるため，本研究では実装しなかった

4. 実　　験

提案した異常注入手法によって外部デバイスから受け取った入力に対する例外処理が実際にテストできることを確認するため，実験を行った．この章では，その実験内容や結果，それに対する考察について説明する．

4.1 実 験 対 象

異常注入機構を加えた CPU エミュレータ上で動作させるソフトウェアとしては，Rockbox[10] を利用した．そして，異常な入力を注入する対象は，ソースファイル rock-

box/firmware/drivers/serial.c で記述されている SERIAL0 関数から呼び出される，以下の関数のロード命令とした．SERIAL0 関数については，4.1.1 項で詳しく説明する．

- rx_rdy 関数で実行され，メモリアドレス 0x70006014 をロード元とするロード命令
- rx_readc 関数で実行され，メモリアドレス 0x70006000 をロード元とするロード命令

ただし，今回の実験では，CPU エミュレータ上で動作させるために，Rockbox に以下の変更を加えた．

- 外部デバイスの初期化用関数を除去した．その理由は，今回の実験では外部デバイスを用意できなかったため，それを初期化する関数を実行するとプログラムの動作が停止するためである
- SERIAL0 関数をソースファイル apps/main.c で記述されている main 関数から呼び出すようにした．その理由は，skyeye で動作できるようにコンパイルした Rockbox では，rockbox/firmware/drivers/serial.c がコンパイルされず，SERIAL0 関数が呼び出せないからである

4.1.1 関数 SERIAL0 の動作

SERIAL0 関数はシリアルネットワークのデータ転送率であるボーレートを変更する関数である．この関数では，現在のボーレートを決定するためのデバイザラッチレジスタの値が取得可能かを rx_rdy 関数によって判断した後，現在のデバイザラッチレジスタの値を rx_readc 関数によって取得する．

SERIAL0 関数および，それから呼び出される rx_rdy 関数，rx_readc 関数のソースコードを**図 3** に示す．

図 3 のように，SERIAL0 関数は以下の順で処理を行う．なお，SERIAL0 を呼び出す前の時点で，グローバル変数である autobaud に格納されている値は 2 であり，0x70006000, 0x70006014 番地のメモリに格納されている値はそれぞれ 0x0D, 0x00 である．

(1) rx_rdy 関数を呼び出し，デバイザラッチレジスタの値が取得可能か判断する．そして，メモリアドレス 0x70006014 に格納されている値の最下位ビットが 1 ならばデバイザラッチレジスタの値が取得可能だと判断して (2) へ進み，0 ならば SERIAL0 関数を終了する．

(2) rx_readc 関数を呼び出し，0x70006000 番地のメモリに格納されているデバイザラッチレジスタの値をローカル変数 temp に格納する．

(3) autobaud の値によって利用する switch 文を選ぶ

(4) temp の値に対応する case 文または default 文を実行する

```
void SERIAL0(void)
{
   static int badbaud = 0;
   static bool newpkt = true;
   char temp;

   while(rx_rdy())
   {
      temp = rx_readc();
      if (newpkt && autobaud > 0)
      {
         if (autobaud == 1)
         {
            switch (temp)
            {
                     (省略)
            }
         } else {
            switch (temp)
            {
                     (省略)
            }
         }
      }
   }
                     (省略)
}
```

```
#define SER0_LSR
(*(volatile unsigned long*)(0x70006014))
#define SER0_RBR
(*(volatile unsigned long*)(0x70006000))
```

```
int rx_rdy(void)
{
   if((SER0_LSR & 0x1))
      return 1;
   else
      return 0;
}
```

```
unsigned char rx_readc(void)
{
   return (SER0_RBR & 0xFF);
}
```

図 3 実験対象とした関数のソースコード
Fig. 3 source code of the functions that are treated as target of examination.

(5) (1) へ戻る

これらのうち，(4) で実行される case 文または default 文の内容を**表 1** に示す．表 1 で実行される処理のうち，default 文がデバイザラッチレジスタの値に対する例外処理であり，case 文は正常系の処理である．

また，表 1 のように，case 文によってはデバイザラッチレジスタの値を変更するために，rx_readc 関数で読み出される 0x70006000 番地に新たな値を格納する処理もある．このため，以前の case 文の実行結果が次に実行される case 文の選択に影響することもある．また，autobaud の値は 5 回以上 default 文を実行することで変更される．

4.2 実験の目的と内容

本実験の目的は，提案した異常注入手法が組込みソフトウェアの例外処理のテストに利用可能なことを示すことである．そのためには，異常注入機構がない時にはテストが困難な処理を，異常注入機構によって実行できることを確認する必要がある．

そして，その目的を達成するため，CPU エミュレータに加えた異常注入機構を用いて Rockbox に異常な入力を注入することによって，4.1.1 項の case 文と default 文の両方を実行することを目標とする．異常注入機構を加えていない CPU エミュレータ上で Rockbox を実行した場合，Rockbox がシリアルネットワークに接続されているなら，デバイザラッチレジスタの値によって case 文に分岐することが多く，default 文を実行することは困難である．また，Rockbox がシリアルネットワークに接続されていないなら，デバイスラッチレジスタの値によって常に default 文に分岐し，case 文を実行することは困難である．そこで，異常注入機構を加えた CPU エミュレータ上で Rockbox を実行させ，デバイザラッチレジスタの値の代わりに異常注入設定ファイルで指定した値を Rockbox に受け取らせることにより，case 文と default 文の両方を実行させる．

この目標を達成するために異常注入設定ファイルに記述した条件文および算出式を**表 2** に示す．各条件式および算出式を用いる理由は，以下のようになっている．

まず，図 3 から，case 文や default 文を実行するには rx_rdy 関数の返り値を 1 にする必要がある．そのためには，メモリアドレス 0x70006014 に格納されている値の最下位ビットが 1 である必要がある．そこで本研究では，メモリアドレス 0x70006014 に格納されている値の代わりに 1 を Rockbox に受け取らせることにより，case 文や default 文を実行できるようにした．

そして，rx_rdy 関数の返り値であるデバイザラッチレジスタの値の代わりに各 case 文や default 文に分岐する値を受け取らせることにより，特定の case 文や default 文を実行できるようにした．この時，各 case 文や default 文を実行するための算出式を選ぶ条件式のオペランドには，現在の入力値であるデバイザラッチレジスタの値を利用している．これは，case 文の中にはデバイザラッチレジスタの値を変更するものがあるため，それを利用して次に実行する case 文を指定することが可能だからである．

なお，本研究の実験では temp の値が 0xFF または 0x55 の時の case 文を実行していない．その理由は，これらの case 文ではデバイザラッチレジスタの値を変更しないため，これらの case 文を実行すると他の case 文が実行できなくなるからである．これは case 文と default 文の両方を実行するという本実験の目標に反するため，今回は実行しなかった．

表 1 関数 SERIAL0 で実行される case 文，default 文の内容
Table 1 the contents of case and default in the function named SERIAL0

autobaud の値	temp の値	badbaud の値	case 文，default 文の内容
2	0xFF	無関係	何もしない
2	0x55	無関係	何もしない
2	0xFE	無関係	0x70006000 番地に 0x1A 格納
2	0xFC	無関係	0x70006000 番地に 0x27 格納
2	0xE0	無関係	0x70006000 番地に 0x4E 格納
2	上記以外	4 以下	badbaud に 1 加算，0x70006000 番地に 0x0D 格納
2	上記以外	5 以上	autobaud に 1 代入，0x70006000 番地に 0x1A 格納
1	0xFF	無関係	何もしない
1	0x55	無関係	何もしない
1	0xFC	無関係	0x70006000 番地に 0x4E 格納
1	0xE0	無関係	0x70006000 番地に 0x9C 格納
1	上記以外	4 以下	badbaud に 1 加算，0x70006000 番地に 0x1A 格納
1	上記以外	5 以上	autobaud に 2 代入，0x70006000 番地に 0x0D 格納

4.3 実験結果

上記の実験を行ったところ，以下の順で case 文や default 文が実行されていた．

(1) autobaud が 2，temp が 0xFE の時の case 文が 1 回実行される
(2) autobaud が 2，temp が 0xFC の時の case 文が 1 回実行される
(3) autobaud が 2，temp が 0xE0 の時の case 文が 1 回実行される
(4) autobaud が 2，badbaud が 4 以下の時の default 文が 5 回実行される
(5) autobaud が 2，badbaud が 4 以下の時の default 文が 1 回実行され，autobaud が 1 になる

表 2 異常注入設定ファイルの内容
Table 2 the content of configure file to inject faults.

関数名	メモリアドレス	条件式	算出式
rx_rdy	0x70006014	常に真	0x1
rx_readc	0x70006000	autobaud が 2 かつ，現在の入力値が 0x0D	0xFE
rx_readc	0x70006000	autobaud が 2 かつ，現在の入力値が 0x1A	0xFC
rx_readc	0x70006000	autobaud が 2 かつ，現在の入力値が 0x27	0xE0
rx_readc	0x70006000	autobaud が 2 かつ，現在の入力値が 0x4E	0x00
rx_readc	0x70006000	autobaud が 1 かつ，現在の入力値が 0x1A	0xFC
rx_readc	0x70006000	autobaud が 1 かつ，現在の入力値が 0x4E	0xE0
rx_readc	0x70006000	autobaud が 1 かつ，現在の入力値が 0x9C	0x10

(6) autobaud が 1，temp が 0xFC の時の case 文が 1 回実行される
(7) autobaud が 1，temp が 0xE0 の時の case 文が 1 回実行される
(8) autobaud が 1，badbaud が 4 以下の時の default 文が 5 回実行される
(9) autobaud が 1，badbaud が 4 以下の時の default 文が 1 回実行され，autobaud が 2 になり，(1) に戻る

これにより，temp が 0xFF と 0x55 以外の時の case 文は全て実行できたことが確認された．

4.4 考　　察

本節では，実験結果を基に，提案手法を利用したテスト手法と，提案手法によって例外処理のテストが改善される点について考察する．

4.4.1 提案手法を利用したテスト手法

本項では，CPU エミュレータに加えた異常注入機構を利用したテスト手法を，ホワイトボックステストとブラックボックステストのそれぞれについて提案する．

まず，CPU エミュレータに加えた異常注入機構を利用したホワイトボックステストとしては，分岐網羅テストが挙げられる．このテストは，プログラム中の条件分岐全てが実行されるように異常な入力を注入するというものである．そうすることにより，入力に対する例外処理の不適切な実装による故障が高確率で発見できる．

また，CPU エミュレータに加えた異常注入機構を利用したブラックボックステストとしては，限界値分析によるテストケース作成が挙げられる．このテストは，外部デバイスの仕様を調べ，外部デバイスからソフトウェアが受け取る入力を有効な範囲と無効な範囲に分けた上で，その境界となる入力値を異常注入機構によって注入するというものである．そうすることにより，ソフトウェアの例外処理に外部デバイスの仕様が正しく反映されていない場合の故障が発見できる．

4.4.2 提案手法によって例外処理のテストが改善される点

本項では，提案手法によって例外処理のテストがどのように改善されるかを考察する．提案手法によって例外処理のテストが改善されると考える理由は以下の通りである．

自動的に異常な入力が注入されるため，他のテスト手法と併用して実行できる

提案手法では，異常注入設定ファイルを一度記述すれば，CPU エミュレータが起動する度に異常な入力がソフトウェアに注入される．そのため，他のテスト手法と同時に提案手法を実行することが可能である．

これは，例外処理をテストするに当たっては非常に有益だと言える．その理由は，例外処理は他の機能に比べて注意が払われ難い[8] が，本研究の提案手法を用いれば，他の機能をテストしながら例外処理を自動的にテストできるからである．

メモリから入力を読み取るソフトウェアに対する異常な入力の注入が容易に行える

提案手法により，既存の異常注入ツールでは困難な，メモリから入力を読み取るソフトウェアに対する異常な入力の注入が容易に行えるようになった．

既存の異常注入ツールの例としては，FIG[9] や Holodeck[1] が挙げられる．これらのツールでは，指定した関数の返り値を破棄し，その代わりに指定した値を新たな返り値として設定する．これにより，関数を用いて外部デバイスから入力を受け取るソフトウェアに異常な入力を注入することが可能である．しかし，組込みソフトウェアには，メモリから値を読むことで，外部デバイスから入力を受け取るものが存在する．このようなソフトウェアに対しては，FIG や Holodeck を用いて異常な入力を注入することが不可能である．

また，GDB[5] を用いれば，指定したメモリアドレスの値を変更することでソフトウェアに異常な入力を注入することが可能である．しかし GDB では，入力を注入する条件の指定に手間がかかるため，異常な入力をソフトウェアに注入して例外処理のテストを行うことは困難である．

しかし，本研究の異常注入機構では，CPU エミュレータのロード命令のオペランドを破棄し，その代わりに指定された値を新たなオペランドとして設定する．そのため，メモリから値を読むことで外部デバイスから入力を受け取るソフトウェアに対して，異常な入力を注入することが可能である．しかも，この注入は異常注入設定ファイルの 4 項目を指定するという簡単な操作で実現できる．これにより，組込みソフトウェアに実装された例外処理のテストが容易に行えると考えられる．

5. 関 連 研 究

異常注入機構を加えた CPU エミュレータとしては，Qinject[2),4)] が挙げられる．Qinject と本研究で大きく異なっている点は，異常を注入する条件の指定方法である．Qinject の異常注入機構では異常注入の条件としてプログラムカウンタの値を指定するのに対し，本研究の異常注入機構では異常注入の条件としてメモリアドレスや関数名，条件式を指定する．そのため，特定の命令を実行した結果に対するソフトウェアの対処をテストするなら Qinject が適切だが，特定の外部デバイスから受け取る入力に対する組込みソフトウェアの対処をテストするなら本研究の提案手法が適切だと考えられる．

6. まとめと今後の課題

本研究では，外部デバイスから受け取る入力に対する例外処理の効率的なテストを目的として，CPU エミュレータに異常注入機構を加えることで異常な入力を注入する手法を提案した．それを実装して実験を行ったところ，異常注入手法なしには実行が困難な例外処理を実行できることが確認された．

今後の課題としては，以下の案が挙げられる．

異常注入設定ファイルの書式を変更する

本研究では異常注入設定ファイルで外部デバイスより先に関数名を指定するが，この書式だと，特定の外部デバイスから受け取る入力に対して全ての関数が対処できるかを確認するといったテストが記述し難いと考えられる．そのため，関数名を指定せずに外部デバイスのみ指定する書式や，関数名より先に外部デバイスを指定する書式が必要だと考えられる．

異常注入設定ファイルの算出式，条件式でオペランドとして利用可能な値を追加する

現在の算術式や条件式のオペランドは，開発者がその値の内容をプログラムの実行前に理解して利用するものばかりである．そこで，それらだけではなく，算出式のオペランドに乱数を用いることにより，開発者が関心を払っていない値を検知する例外処理がテストされる可能性が高くなると考えられる．また，条件式のオペランドにも乱数を用いることにより，開発者が想定していない状況で異常な入力がソフトウェアに注入される可能性が高くなると考えられる．

他のソフトウェアに提案手法を適用する

本研究では改変した RockBox を実験対象として用いたが，それ以外のソフトウェアに対しても提案手法を適用する必要があると考えている．それによって，提案手法の妥当性をより正確に示すことができる．

適用対象として適切なソフトウェアの条件としては，ソフトウェアの改変なしに提案手法を適用できることや，提案手法によって実際に組込みソフトウェアの故障が発見できることが挙げられる．

物理 CPU に異常注入機構を加える

CPU エミュレータではなく物理 CPU に異常注入機構を加えてソフトウェアを動作させることにより，物理 CPU にも提案手法が適用できるかを確認する．また，物理 CPU 上の動作と CPU エミュレータ上の動作を比較し，それらの違いについて考察することで，提案手法の効果について考察する．

既存のテスト機構や統合開発環境から異常注入の指定を可能にする

注入する異常な入力の値や，それを注入する状況の指定を，既存のテスト機構や統合開発環境から行えるようにする．そうすることによって，既存のインターフェイスで容易に異常注入の指定を行ったり，既存のテスト機構や統合開発環境の機能と提案手法を組み合わせたテストを行ったりすることが可能になる．

参 考 文 献

1) Application Security,Secure Software Development, and software Security Training,
`http://www.securityinnovation.com/.`
2) Choices,
`http://choices.cs.uiuc.edu/.`
3) Dwarf Home,
`http://dwarfstd.org/.`
4) Francis, D., Ellick, C., Jeffrey, C. and Roy, C.: QInject: A Virtual Machine based Fault Injection Framework, *In International Conference on Architectural Support for Programming Languages and Operating Systems* (2008).
5) GDB: The GNU Project Debugger,
`http://www.gnu.org/software/gdb/.`
6) 重要なお知らせ：「FOMA D900i」をご愛用のお客様へ — お知らせ — NTT ドコモ,
`http://www.nttdocomo.co.jp/info/notice/page/040817_00.html.`
7) 重要なお知らせ：FOMA「P901i」　ご利用のお客様へ — お知らせ — NTT ドコモ,
`http://www.nttdocomo.co.jp/info/notice/page/050527_00.html.`
8) Martin, P. R. and Gail, C. M.: Regaining Control of Exception Handling, *Technical Report TR-99-14, Department of Computer Science*, University of British Columbia (1999).
9) Pete, B., Naveen, S., Jonathan T.: FIG: A Prototype Tool for Online Verification of Recovery Mechanisms, *In Workshop on Self-Healing, Adaptive and Self-Managed Systems*, (2002).
10) Rockbox - Open Source Jukebox Firmware
`http://www.rockbox.org/.`
11) SkyEye - Open Source Simulator,
`http://www.skyeye.org/.`
12) The Lex & Yacc Page,
`http://dinosaur.compilertools.net/.`