RISC-V プロセッサに対するフォールトインジェクション実験の結果分析

田上 凱斗[1,a)]　橋本 昌宜[2,b)]

概要：
オープンソースの命令セットアーキテクチャである RISC-V が注目を集めており，今後高信頼システムにおいても利用の拡大が予想される．一方で，市販されているチップの実装では，ソフトエラーと呼ばれる一過性のビット反転に対する対策が取られておらず，その信頼性が低下する恐れがある．そこで本研究では市販の RISC-V プロセッサチップに対して，デバッガを用いてレジスタとデータメモリを対象にフォールトインジェクションを実施し，プロセッサのエラー率を調査した．また発生したエラーの分析及びその原因推定を行った．レジスタに対するフォールトインジェクション結果より，プログラムにおけるレジスタの使用回数によってエラー率が変化することが分かった．データメモリに対するフォールトインジェクションでは，プログラムの実行時にアクセスするデータメモリのサイズが小さいとエラー率も小さくなることを確認した．

キーワード：RISC-V，フォールトインジェクション（FI），レジスタ，データメモリ

1. はじめに

近年，RISC-V と呼ばれる命令セットアーキテクチャが注目を集めている．このアーキテクチャはオープンソースであり，公式サイト上に仕様書が公開されている [1]．RISC-V はライセンス料がフリーとなっているため誰でも自由に使用可能である．加えて，Rocket Chip [2] のような HDL モデルや実デバイスが多く存在しており，RISC-V プロセッサの更なる利用拡大が期待されている．

しかし，一般に販売されているチップはソフトエラーと呼ばれるビット反転の対策がなされていない．ソフトエラーとは宇宙線が原因となって発生する一過性のビット反転のことである．このソフトエラーが発生するとシステムが誤動作する危険性があり，システムの信頼性を低下させる要因となり得る．市販されているプロセッサは高性能，低消費電力，かつ安価ということもあり様々なシステムで利用されているが，高信頼システムにおいては特にソフトエラー対策が重要となってくる．

そこで本稿では，高信頼システムに今後利用される可能性のある RISC-V プロセッサを対象に，フォールトインジェクションを行い，ビット反転の影響を調べた．三つのベンチマークプログラムにおいてレジスタを対象にフォールトインジェクションを実施した．そのうちの一つのベンチマークについては比較のためにデータメモリに対しても実験を行った．ベンチマークプログラムの違い，特にレジスタの利用状況とエラー率の関係について考察を行った．

2. 関連研究

ソフトエラーの発生率や影響の評価には主に 2 つの手法が用いられている．1 つ目はプロセッサに放射線を照射してソフトエラーを実際に発生させる手法である．2 つ目は人為的にビット反転を起こしてソフトエラーを再現する手法である．後者の手法は一般的にフォールトインジェクションと呼ばれている（これ以降，フォールトインジェクションを FI と表記することがある）．

FI ではシミュレータ/エミュレータやデバッガによってビット反転を発生させるため，ビット反転可能な場所に制限が生じることがある．市販のチップとデバッガを用いた FI の場合，フォールトの注入可能な箇所はユーザからアクセスできるレジスタやメモリだけである．RTL 記述とシミュレータ/エミュレータを用いた FI の場合はフリップフロップに注入することが可能であるが，トランジスタと

[1] 大阪大学 大学院情報科学研究科 情報システム工学専攻
[2] 京都大学 大学院情報学研究科 通信情報システム専攻
[a)] y-tagami@ist.osaka-u.ac.jp
[b)] hashimoto@i.kyoto-u.ac.jp

ⓒ 2021 Information Processing Society of Japan

いった回路素子レベルでは注入できない．

一方で，照射実験ではユーザからのアクセスが不可能な部分で発生するビット反転の影響も調査可能である．しかし，照射実験は加速器施設を利用して行うため時間の制約や経済的負担が高く，十分にデータを取得できない場合がある．反対に，FI ではソフトウェアやデバイスを用意するだけで実施可能なため，費用をあまりかけずにデータを得ることができる．このような理由から，FI を用いたソフトエラーの研究が広く実施されている．

[3] では FPGA に RISC-V プロセッサを実装し，重イオンの照射実験及び FI 実験を行っている．加えて，ビット反転の影響を緩和する手法の効果も分析している．[4] では FI 実験を用いて，RISC-V プロセッサの実装方式による違いがソフトエラーにどのような影響を及ぼすかについて調べている．[5] の論文ではマルチコア/メニーコアプロセッサを対象に中性子照射実験と FI 実験を行っている．さらに，それらの結果を利用してアプリケーションのエラー率予測モデルを評価している．[6] では MSP430 プロセッサの実デバイス版と FPGA に実装された HDL モデル版に対して FI 実験を実施している．それらの結果を比較することで，実デバイスの代わりとしての HDL モデルの有用性を評価している．

RISC-V プロセッサを対象とした論文では FPGA に実装したソフトコアが実験対象であったが，本稿では市販されている RISC-V プロセッサチップを対象に FI 実験を行った．中身がブラックボックスの市販プロセッサチップに対する評価技術の確立を目指し，第一ステップとして行った FI 実験結果を本稿では報告する．

3. 実験準備

本稿の実験の目的は，RISC-V プロセッサの実デバイスを対象に FI を行った際のエラー率を調べることである．RISC-V プロセッサを搭載したボードを対象とし，FI はデバッガを用いて実施する．実験のセットアップを以下で説明する．

3.1 使用するデバイス

本稿では，FI 実験の対象として SiFive 社の HiFive1 Rev B ボード [7] を使用する．このボードは 32 ビット RISC-V プロセッサを搭載しており，その動作周波数は最大で 320 MHz である．基本的な整数命令に加えて乗除算命令，原子命令，さらに圧縮命令をサポートしている．また，16 k バイトのデータメモリと 16 k バイトの L1 命令キャッシュと 4 M バイトの SPI フラッシュメモリを備えている．ホスト PC との接続には USB を利用し，フラッシュメモリへのプログラムの書き込みも USB 経由で行う．

3.2 使用するツール

FI の実施に二つのソフトウェアを使用した．一つ目は SiFive 社が提供する RISC-V GNU Toolchain の riscv64-unknown-elf-gdb である．バージョンは 8.3.0-2020.04.0 である．二つ目は SEGGER 社の提供する J-Link OB ソフトウェアの JLinkGDBServer である．バージョンは 6.90 である．riscv64-unknown-elf-gdb は通常のデバッガ GDB と同じであるが，単体では HiFive1 Rev B ボードを制御することができない．そのため，JLinkGDBServer を使用する．

3.3 ベンチマークプログラム

今回の FI 実験では以下の三つのベンチマークプログラムを使用した．

- SHA-1（最適化なし）
- SHA-1（最適化あり）
- クイックソート

SHA-1 はハッシュ値を計算するアルゴリズムであり，1 ビットでも値が異なると計算結果が大きく変化する．そのため，ビット反転の影響が結果に表れやすいと考えられる．クイックソートは FI 実験を行っている他の論文において比較的利用されているベンチマークであったため本稿でも使用した．これらのプログラムは C 言語を用いて実装しており，SHA-1 は [8] を参考に実装した．コンパイラは RISC-V GNU Toolchain の riscv64-unknown-elf-gcc を使用しており，SHA-1 の最適化では-O2 を指定した．

SHA-1 プログラムの入力はすべての実行で固定しており，最適化なしと最適化ありの両方で同じ値をプログラム内で与えている．クイックソートの入力は-100 から 100 までの 32 個の整数としている．これらの整数はプログラムの作成段階でランダムに決定したものである．ソート対象の 32 個の値はプログラム中でソート開始前にデータメモリにある配列に格納する．

SHA-1 プログラムではハッシュ値計算や結果の出力といった処理を全て main 関数内に記述している．一方，クイックソートプログラムではソートを行う部分を関数として独立させている．main 関数ではソートを行う関数の最初の呼び出しと結果の出力だけを実行する．これらのプログラムでは標準ライブラリ stdio と SiFive 社が提供する Freedom Metal ライブラリをリンクしている．

なお，今回の FI 実験ではベンチマークプログラムをベアメタルアプリケーションとして実行した．つまり，HiFive1 Rev B ボード上で OS を動かすことなくプログラムを直接実行している．

3.4 フォールトインジェクションの実施方法

FI の実施手順を説明する．まず，FI を実施するタイミングをプログラムカウンタの値によって制御する．プログラムカウンタの値は命令メモリのアドレスであり，デバッ

ⓒ 2021 Information Processing Society of Japan

ガを用いてブレイクポイントを設定することが可能である．このブレイクポイントに初めて到達した際に FI を行う．レジスタやデータメモリの値の読み出しにはデバッガのコマンドを使用し，対象ビットを反転させた値を計算してレジスタやメモリに書き戻す．その後，プログラムの実行を再開し計算結果をファイルに出力して一回分の FI が終了する．

FI の対象となるレジスタは 32 個であり，各レジスタに役割が存在する．ra はリターンアドレスを保持するレジスタである．sp，gp，tp はそれぞれスタックポインタ，グローバルポインタ，スレッドポインタである．t0–6 は一時レジスタ 0–6 である．s0–11 は保存レジスタ 0–11 である．なお，s0 は fp（フレームポインタ）として利用されることもある．a0–7 は関数の引数として使用されるレジスタである．a0 と a1 は戻り値を保持するレジスタとしても使われる．最後に，pc はプログラムカウンタであり，命令メモリのアドレスを保持する．

データメモリは大きさが 16 k バイトであり，メモリ全体を FI の対象としている．レジスタもデータメモリもデータ幅が 32 ビットで，それら全てのビットが FI の対象である．今回の実験ではフォールト注入箇所の偏りをなくすため，注入箇所の間隔をあらかじめ決めておき，その間隔に従って pc やレジスタ/メモリアドレス，ビット位置を決めることにした．

4. 実験結果

4.1 結果の分類方法

FI 実験で得られた実行結果は**表 1** に従って分類する．プログラムの実行は正常終了したが計算結果が間違っている場合は SDC (Silent Data Corruption) に分類する．プログラムの動作や出力に異常が見られる場合は DUE (Detectable Uncorrectable Error) に分類する．プログラムの動作と実行結果の両方が正常な場合は Mask に分類する．Mask はビット反転されたデータが使用されなかったときなどに発生する．

上記の分類の中で最も厄介なものは SDC である．上述した通り，SDC はプログラムは正常終了したにも関わらず計算結果が誤っていた場合を指しており，プロセッサの外部からはエラーの存在を検知できない．今回の FI 実験では各ベンチマークにおいて入力を固定しているため，計算結果が正しいかどうか容易に判断できるが，入力が変化する一般のシステムの場合は結果が正しいか判断するのは難しい．特に，SHA-1 のようなハッシュ値計算では，出力に規則性がなく正誤を判断することはできない．一般的な計算結果を事前に知ることができないシステムでは SDC がシステムの信頼性を低下させる要因となる．

一方，DUE では動作や出力に異常が見られるためエラーの検知は比較的容易である．しかし，エラーの修正はできないためプログラムを再実行するなど何らかの対処が必要になる．再実行が許容できないシステムでは，DUE もシステムの信頼性を低下させる要因となる．

表 1: 結果の分類方法

分類	内容
SDC	計算結果の誤り
DUE	誤動作または異常な出力
Mask	正常実行

次節以降，各ベンチマークプログラムに対する FI 実験の結果を本節の分類に基づいて示していく．SHA-1，最適化した SHA-1，クイックソートの三つにおいてレジスタを対象に FI 実験を行った．加えて，SHA-1 ではデータメモリに対する FI も実施した．それぞれの FI 実験において取得したデータは 12,000 個以上である．

4.2 SHA-1

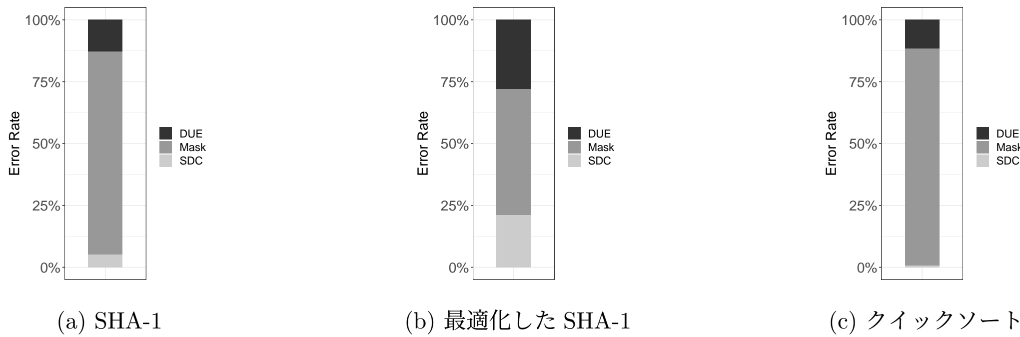
図 1a にレジスタを対象とした FI の結果を示す．ただし，今回の実験では main 関数内でのみ FI を実施した．正確に述べると，main 関数内の機械語を実行している pc の値の範囲内で FI を実行するタイミングを指定した．

図 1a より，レジスタを対象とした FI では DUE や SDC エラーの発生割合が高くないことが分かる．実際，Mask は 82%であり SDC は 5%のみである．SHA-1 はハッシュ値を計算するプログラムであるためビット反転によって計算を誤ると思われるが，レジスタへの FI では計算結果に与える影響は限定的であった．

更なる分析のためレジスタ毎のエラー率を**図 2** に示す．同時に，main 関数に対応するアセンブリコード内での，各レジスタのオペランドとしての静的な使用回数を**図 3** に示す．ここではソースレジスタ及びデスティネーションレジスタの両方をカウントしている．なお，プログラムカウンタは図 3 に含まれない．

レジスタ毎のエラー率を見てみると一部のレジスタにエラーが偏っていることが分かる．特に，sp，gp，s0，pc において DUE が多く発生している．gp はグローバルポインタであり，グローバル変数領域のアドレスを保持している．s0 はローカル変数領域のアドレスを保存するために使用されている．このレジスタはローカル変数の読み書きに用いられており，図 3 より多用されていることが分かる．これら四つのレジスタはメモリアドレスを扱っておりメモリアクセスの際に利用されることが多い．FI によってアドレスが変化すると，メモリアクセス例外や，pc の場合コントロールフローの乱れが発生する．その結果，DUE の割合が増大したと考えられる．

一方，SDC の割合が大きいレジスタは a4 と a5 である．これらのレジスタは関数の引数に使われるものであるが，図 3 で示されているように，main 関数に対応するアセン



(a) SHA-1　(b) 最適化した SHA-1　(c) クイックソート

図 1: 各プログラムにおけるレジスタ FI の結果

ブリコードにおいて a4 と a5 が頻繁に使用されていたことが読み取れる．特に，a5 の使用回数は圧倒的である．この二つのレジスタを主に利用してハッシュ値の計算が行われた．このような理由から a4 と a5 では SDC の割合が大きくなったと考えられる．

使用回数が 0 となっているレジスタについてはエラーが全く発生していない．しかし，sp と gp は図 3 を見る限りほとんど使用されていないにも関わらず，エラーが多く発生している．この結果より，sp と gp は main 関数以外（標準ライブラリやハードウェアを制御するライブラリなど）において多用されているのではないかと推測できる．sp と gp に関しては更なる調査が必要である．

4.3　最適化した SHA-1

SHA-1 プログラムのコンパイル時に最適化を行った場合の FI 結果について説明する．前述の通常の SHA-1 と同じく main 関数内でしか FI を行っていない．

まず，レジスタ全体でのエラー率を**図 1b** に示す．SDC は 21%であり DUE は 28%であった．最適化をしていない SHA-1 と比較すると明らかにエラーの割合が増加した．最適化をしていない場合，プログラムの処理にメインで使われるレジスタが一部に偏っていたためエラーの割合が全体として低くなっていた．しかし，最適化を行ったことで全体のエラーの割合が増えており，プログラムの処理に多くのレジスタが使用されたのではないかと推測できる．これを確かめるため，**図 4** に示したレジスタ毎のエラー率と**図 5** に示した main 関数に対応するアセンブリコード内の各レジスタの使用回数を分析する．なお，使用回数の計測方法は通常の SHA-1 の場合と同じである．

DUE は，最適化をしなかったときと同様に sp，gp，s0，pc において多く発生している．それらに加えて，ra や t0，t2，t4–6 においても比較的多く発生している．しかし，図 5 によるとこれらのレジスタはあまり使用されていないことが分かる．これらのレジスタは main 関数以外で主に利用されている可能性がある．

次に，SDC は t1，a0–7，s2–7，t3 において多く発生していることが分かる．a0–7 に関しては使用回数が多いため SDC の割合が増加したと考えられる一方，それ以外のレジスタについては使用回数が多いとは言えない．しかし，それらのレジスタは a0–7 よりも SDC の割合が大きくなっており，main 関数以外で主に用いられている可能性がある．逆に，a5 は最も使用回数が多いにも関わらず SDC の割合が 32%とそれほど高くない．今回のレジスタの使用回数はあくまでアセンブリコード上での静的な出現回数であり，実際のプログラム実行時の使用回数ではない．そのため，実際の動作では図 5 の通りに使用されていない可能性も考えられる．更なる調査が必要である．

以上の通常の SHA-1 との比較により，より多くの種類のレジスタが計算に用いられるようになると，その分エラーの割合も増加することが判明した．一方で，最適化によってプログラムの実行効率は大幅に上昇し，実行速度は 14 倍向上した．エラー率と実行時間の両方を考慮したエラー率に基づく信頼性の評価が必要である．

4.4　クイックソート

本節ではクイックソートにおけるレジスタを対象とした FI の結果を分析する．クイックソートでは main 関数とクイックソートを行う関数内で FI を実施した．なお，クイックソートにおける SDC とは，プログラムは正常終了したがソート結果が誤っている場合を指す．

始めに，レジスタ全体のエラーの割合に注目する．**図 1c** に結果を示す．結果としては SDC が 1%もなく DUE が 12%であった．エラーの割合の傾向としては通常の SHA-1 に似ている．したがって，レジスタ毎のエラーの割合や使用回数も同じような傾向となっている可能性がある．そこで，レジスタ毎のエラー率を示した**図 6** と使用回数を示した**図 7** を基に分析する．ただし，クイックソートにおけるレジスタの使用回数では，main 関数だけでなくクイックソートを実行する関数に対応するアセンブリコードでの出現回数も含める．

まず，DUE の割合について見てみると，通常の SHA-1 と同じく sp，gp，s0，pc において多く発生している．加えて，ra や a0–5 でも DUE が発生している．ra はリターンアドレスを保持するためのレジスタであり，呼び出した

ⓒ 2021 Information Processing Society of Japan

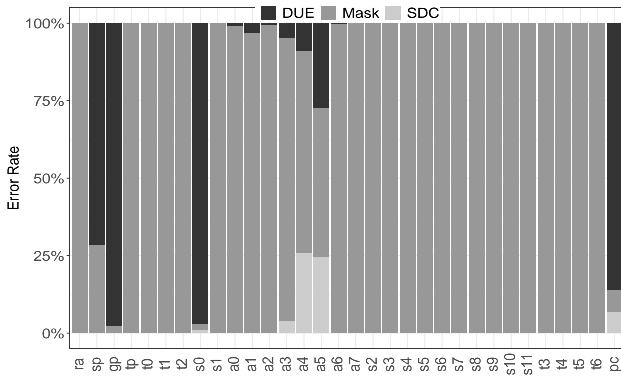


図 2: SHA-1 におけるレジスタ毎のエラー率

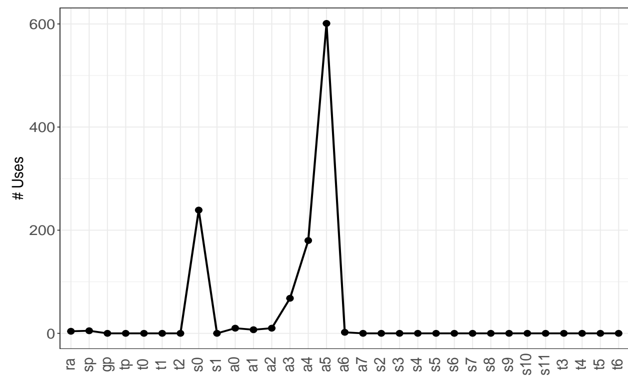


図 3: SHA-1 における各レジスタの使用回数

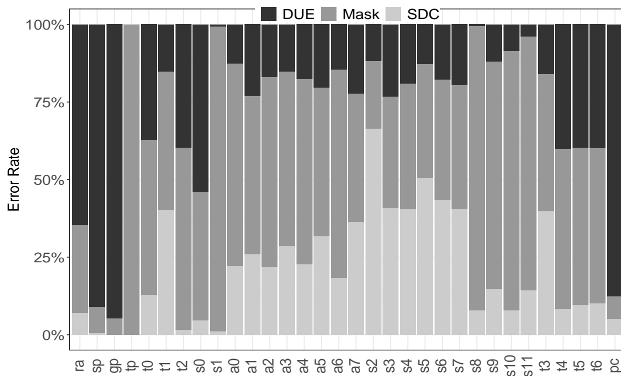


図 4: 最適化した SHA-1 におけるレジスタ毎のエラー率

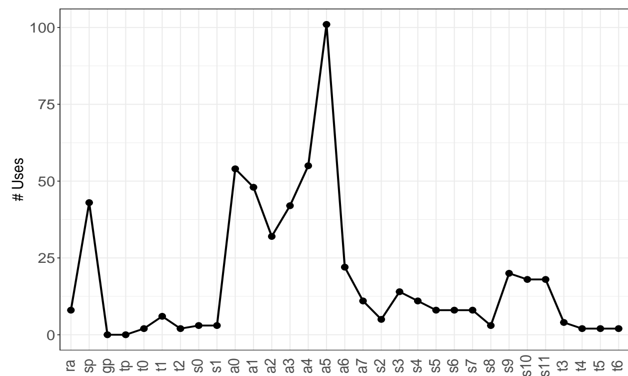


図 5: 最適化した SHA-1 における各レジスタの使用回数

関数から戻ってくる際に使用される．ra に FI を行うと関数から戻ってくるときのアドレスが変わってしまうため DUE が発生すると考えられる．しかし，使用回数が少ないため今回の実験ではほとんどエラーが発生しなかった．a0–5 は関数の引数を渡すときに使用されるレジスタである．クイックソートを実行する関数への引数には，ソート対象のデータが格納されている配列へのポインタとソート範囲を示すインデックスが渡される．アセンブリコードでは，配列へのポインタは a0，ソート範囲の下限を a1，上限を a2 に格納して関数へ渡している．しかし，クイックソートの実際の計算には a4 と a5 が主に用いられており，メモリアクセスにも使用されている．そのため FI によって DUE が発生したと考えられる．

SDC の割合は他二つのベンチマークと比較して非常に低い．比較的 SDC 割合の多いレジスタは s0，a1，a5 の三つである．s0 はローカル変数領域のアドレスを保持しており，変数値の読み書きにおいて利用される．s0 の値が FI によって変化した場合，意図していた変数とは異なる変数の値を読み書きする可能性がある．その結果ソートが正しく行えずに SDC となってしまったと考えられる．a1 については，先述の通りソート範囲の下限を関数に渡す際に使用されている．FI によって下限が変化することにより，ソート範囲が変わってしまいソートが正しく行われなかった結果 SDC が発生したと思われる．a5 は DUE の部分で述べたように，クイックソートの計算にメインで用いられている．この計算にはループ処理も含まれており，ループの制御変数を扱っていることがある．そのため，a5 に FI を行うとソートを正しく実行できなくなり SDC が起きると推察される．

以上の通常の SHA-1 との比較により，レジスタ毎のエラー率や使用回数が似ていることが判明した．レジスタに対する FI では，プログラムの構造よりもレジスタの使用回数や使用方法がエラー率に影響を与える傾向がみられた．したがって，コンパイラがどのようにプログラムをコンパイルするかによってエラー率が変化すると考えられる．

4.5 SHA-1 におけるデータメモリ

次に，データメモリに対する FI の結果を分析する．データメモリへの FI は通常の SHA-1 プログラムにおいて実施した．データメモリに関しても main 関数内でのみ FI を実施した．

図 8 に実験結果を示す．データメモリへの FI では SDC と DUE がほとんど発生しなかった．その割合はどちらも 1%だけである．レジスタに FI したときとは異なり，データメモリへの FI ではプログラムの動作に影響を与える割合が低かった．

そこで，main 関数の実行時にアクセスするデータメモリの大きさを調査した．この調査にはデバッガ GDB の

ⓒ 2021 Information Processing Society of Japan

図 6: クイックソートにおけるレジスタ毎のエラー率

図 7: クイックソートにおける各レジスタの使用回数

図 8: SHA-1 におけるデータメモリ FI の結果

`rwatch` コマンドを利用した．結果としては，実行時に読み出しアクセスするメモリは全体のわずか 3%であった．アクセス先のアドレスは一部の領域に集まっており，それ以外の多くのアドレスにはアクセスしていない．そのため，データメモリに対して FI を行ってもエラーがほとんど発生しなかった．逆に，実行時にアクセスするメモリの範囲が大きいとエラーの割合も増加する可能性がある．引き続き検証を進める必要がある．

5. おわりに

本稿では，RISC-V プロセッサを搭載した HiFive1 Rev B ボードに対して FI 実験を行った．ベンチマークプログラムとして SHA-1 の最適化あり/なしとクイックソートの三つを利用した．FI の対象は 32 個のレジスタであり，通常の SHA-1 においてはデータメモリに対しても FI 実験を行った．実験結果より，レジスタの使用頻度の偏りによってエラー率が変化することが分かった．最適化していない SHA-1 とクイックソートの結果の比較からは，エラー率のプログラム構造に対する依存性は見受けられなかった．データメモリに対する FI では，main 関数実行時にアクセスするメモリのサイズが小さいとエラー率も低くなることを確認した．

今後の課題としては，最適化した SHA-1 とクイックソートにおいてもデータメモリに対する FI を実施することが挙げられる．それらのプログラムの実行時にアクセスするデータメモリの大きさを調査し，通常の SHA-1 におけるデータメモリへの FI 結果と比較しながら分析を行う．一方，今回の FI では実施するタイミングを main 関数内に制限していたが，その制限を無くした場合の結果を調査することも挙げられる．

謝辞

本研究は，科研費基盤 (S) JP19H05664 ならびに JST OPERA JPMJOP1721 の助成による．

参考文献

[1] RISC-V International: Specifications, (online), available from ⟨https://riscv.org/technical/specifications/⟩ (accessed 2021-07-30).

[2] CHIPS Alliance: Rocket Chip Generator, (online), available from ⟨https://github.com/chipsalliance/rocket-chip⟩ (accessed 2021-07-30).

[3] de Oliveira, Á. B., Tambara, L. A., Benevenuti, F., Benites, L. A. C., Added, N., Aguiar, V. A. P., Medina, N. H., Silveira, M. A. G. and Kastensmidt, F. L.: Evaluating Soft Core RISC-V Processor in SRAM-Based FPGA Under Radiation Effects, *IEEE Trans. Nucl. Sci.*, Vol. 67, No. 7, pp. 1503–1510 (2020).

[4] Cho, H.: Impact of Microarchitectural Differences of RISC-V Processor Cores on Soft Error Effects, *IEEE Access*, Vol. 6, pp. 41302–41313 (2018).

[5] Ramos, P., Vargas, V., Baylac, M., Zergainoh, N.-E. and Velazco, R.: SEE Error-Rate Evaluation of an Application Implemented in COTS Multicore/Many-Core Processors, *IEEE Trans. Nucl. Sci.*, Vol. 65, No. 8, pp. 1879–1886 (2018).

[6] Isaza-González, J., Serrano-Cases, A., Martínez-Álvarez, A., Cuenca-Asensi, S., Guzmán-Miranda, H. and Aguirre, M. A.: Contrast of a HDL model and COTS version of a microprocessor for soft-error testing, *2017 18th IEEE Latin American Test Symposium (LATS)*, pp. 1–6 (2017).

[7] SiFive: HiFive1 Rev B, (online), available from ⟨https://www.sifive.com/boards/hifive1-rev-b⟩ (accessed 2021-07-30).

[8] The Internet Society: US Secure Hash Algorithm 1 (SHA1), (online), available from ⟨https://www.ipa.go.jp/security/rfc/RFC3174JA.html⟩ (accessed 2021-07-30).

ⓒ 2021 Information Processing Society of Japan