Case study of creating test suite for RISC-V microprocessors using errata sheets

Nikita Garaev
Ivannikov Institute for System Programming of RAS
Higher School of Economics
Moscow, Russia na-
garaev@gmail.com

***Abstract*—This paper investigates the use of error descriptions from microprocessor errata sheets to generate test programs using MicroTESK for RISC-V ISG. The main objective is to develop a microprocessor error database and a test suite for RISC-V architecture based on the error descriptions from the obtained database. To evaluate the proposed approach, we apply the obtained test suite on RTL models of known RISC-V microprocessors: SCR1 from Syntacore and CVA6 from OpenHW Group. The work is in progress.**

Index Terms—RISC-V, testing, errata.

I. Introduction

With the increase in the number of placed transistors, achieving lower operating voltages, and the modernization of microprocessor designs, the growth in both performance and energy efficiency of the microprocessor is observed, as well as a manifold increase in the complexity of testing and verification. As a result, the potential number of emerging errors from various design oversights, inadequate test sets, or the non-standard and extreme complexity of predicting scenarios in which the microprocessor may behave erroneously also increases.

Detecting all errors in an upcoming microprocessor to be released is an impossible task. Therefore, a certain part of errors inevitably makes its way into the final product and can only be discovered after the processor has been in the market for some time. As a result, vendors regularly publish errata documents, which describe known errors of the product in detail. Such descriptions may contain valuable information for constructing rare and realistic test scenarios.

In this paper, we propose an approach to constructing test programs for microprocessors based on the use of code examples and error descriptions from errata documents. The approach is applied to the test program generator MicroTESK for RISC-V [1]. Currently, we have collected 3042 error descriptions from "errata sheets," some of which have become the basis for test program templates for the test generator.

To evaluate the proposed approach, we will apply the generated test programs to RTL models of RISC-V microprocessors: SCR1 from Syntacore [2] and CVA6 from OpenHW Group [3].

Previously, a similar approach was proposed for configuring the Csmith test generator using code fragments from Bugzilla's bug reports and tested on several versions of the GCC compiler by Md Rafiqul Islam Rabin and Mohammad Amin Alipour from the University of Houston [4]. The results they obtained suggest the prospect of implementing a similar approach in microprocessor testing.

Contributions. This paper aims to contribute the following:

- An approach to building test programs using code fragments and error descriptions from microprocessor errata sheets is proposed
- The proposed approach is applied during the testing RTL-models of RISC-V microprocessors.

II. Background

A. Used solutions for testing RISC-V microprocessors

RISC-V is a universal microprocessor architecture with open instruction set (ISA) and specification. It was developed at UC Berkley and is intended to be the fifth generation architecture for microprocessors built using the RISC methodology. The main feature of the architecture is its extensibility: the architecture is presented in the form of extensions, most of which are optional. This results in its advantage: the developer is free to choose a set of instructions for the task at hand. It is also possible to define one's own extension to extend the functionality, for example, to implement cryptographic functions.

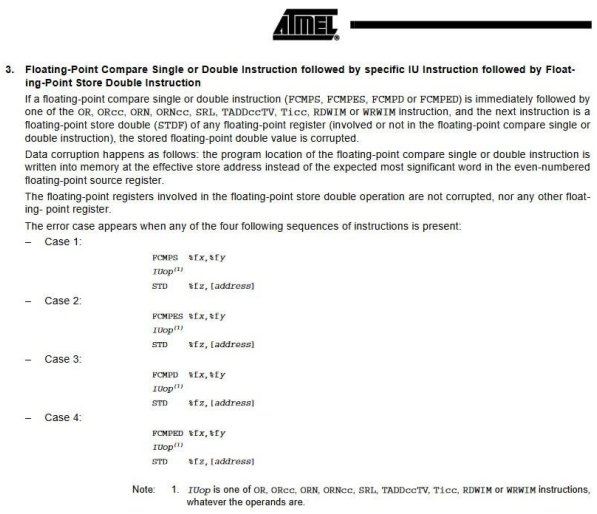
The standard solutions for testing this architecture are RISC-V ISA Tests [5], Compliance Tests [6], and Torture TG [7]. As the target architecture, they are developed at UC Berkley.

RISC-V ISA Tests and **Compliance Tests** are predefined test suites aimed at individual verifying each instruction in the ISA. They are designed to verify whether the behavior of the processor implementation correspond to the RISC-V specification.

Torture TG is a random test generator that builds programs from manually defined sequences of instructions. It is targeted at testing parallel execution of several instructions on the pipeline. Impossibility to manually define a test scenario is a lack of this solution.

In this paper, presented solutions are used as a reference for evaluating the tests constructed by the suggested approach. MicroTESK for RISC-V is used as the base tool for test generation.

MicoTESK for RISC-V is a configurable TPG based on the MicroTESK Toolkit. It is developed at ISP RAS and can be configured using a formal processor specification written in nML, and an additional one that describes the memory subsystem in mmuSL. It generates programs based on templates



3. **Floating-Point Compare Single or Double Instruction followed by specific IU Instruction followed by Floating-Point Store Double Instruction**

If a floating-point compare single or double instruction (FCMPS, FCMPES, FCMPD or FCMPED) is immediately followed by one of the OR, ORcc, ORN, ORNcc, SRL, TADDccTV, Ticc, RDWIM or WRWIM instruction, and the next instruction is a floating-point store double (STDF) of any floating-point register (involved or not in the floating-point compare single or double instruction), the stored floating-point double value is corrupted.

Data corruption happens as follows: the program location of the floating-point compare single or double instruction is written into memory at the effective store address instead of the expected most significant word in the even-numbered floating-point source register.

The floating-point registers involved in the floating-point store double operation are not corrupted, nor any other floating- point register.

The error case appears when any of the four following sequences of instructions is present:

– Case 1:

```
FCMPS  %fx,%fy
IUop(1)
STD    %fz,[address]
```

– Case 2:

```
FCMPES %fx,%fy
IUop(1)
STD    %fz,[address]
```

– Case 3:

```
FCMPD  %fx,%fy
IUop(1)
STD    %fz,[address]
```

– Case 4:

```
FCMPED %fx,%fy
IUop(1)
STD    %fz,[address]
```

Note: 1. *IUop* is one of OR, ORcc, ORN, ORNcc, SRL, TADDccTV, Ticc, RDWIM or WRWIM instructions, whatever the operands are.

Figure 1. An example of an error description from the TSC695 microprocessor error sheet

that are implemented in the Ruby language, which allows to use advantages of the high-level language to describe a test script. An example of a similar template will be given later in the text of the paper.

B. Errata sheet

As mentioned before, an errata document is a document or application that contains a detailed description of errors. As an example, let's consider the [8-10]. The structure of the document may vary from vendor to vendor. It may include:

- A brief description of the product and its revisions affected by the document;
- history of revisions to the document;
- a summary table of errors;
- a detailed description of each specific error.

The description of the error may consist of the following elements:

- definition of symptoms and conditions of erroneous behavior emergence;
- an example of a scenario causing the error (pseudocode or assembler);
- possible ways of erroneous behavior's workaround;
- error status.

Figure 1. It is an example of an error description from the SPARCv7 32-bit TSC695 microprocessor errata sheets from Atmel Corporation. It occurs when attempting to write a floating-point number to memory right after the combination of the comparison instruction for real numbers and a specific arithmetic instruction [8]. It is expected that such a description will be sufficient to create test programs that reflect real processor scenarios and can cover more critical paths and functional blocks compared to random tests and standard tests targeting the instruction set system (ISA).

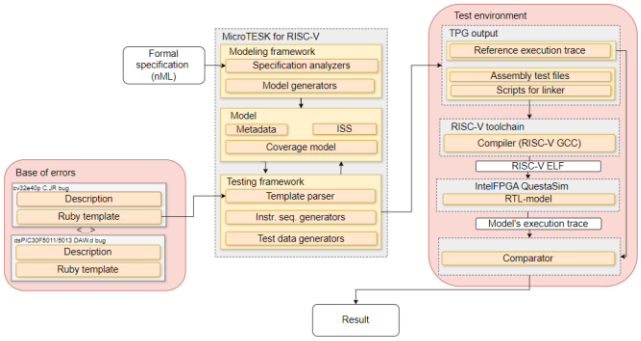


Figure 2. Testing architecture

III. Approach

Figure 2 shows the testing architecture that implements the approach. The key elements are: (1) a database of errors and test program templates for MicroTESK generator, (2) MicroTESK for RISC-V generator, (3) a testing environment consisting of a compiler, an RTL model simulation environment, and a comparator.

Base of Errors. It is a collection of microprocessors' error descriptions of different architectures that obtained as a result of document analysis and test patterns for MicroTESK for RISC-V generator.

The process of composing the database is described as follow steps:

- Analysis of the error document. At this stage, the presence of a detailed errors' description in the document is examined and the possibility of forming a test scenario's description of using the presented format is evaluated;
- Analysis of the error descriptions in the document. At this stage, we estimate the contained pseudocode or description of the error context, the possibility of implementing the test scenario on the target architecture, examine whether the description contains any contextual restrictions (which registers to use, the range of address values or values for arithmetic instructions, etc.);
- Composing test program templates for the selected test generator based on our previous analysis of the obtained descriptions.

Test Template. The error given in Figure 1 for a microprocessor on the RV32-IMF* microarchitecture is represented as a test template for MicroTESK for RISC-V as follows. A basic template that describes the basic elements of the generated test (epilogue, branching for passed/failed tests, etc.) is inherited. The generator allows to define a data section for test data and result record:

```
class ErrataMicrochip0001 < RiscVBaseTemplate
    @xreg_excl = nil
    @freg_excl = nil

    def TEST_DATA
        data {
            org 0x0
            align 4
            label :float_s_data
            float Random::rand(0.01..64.0)
            label :float_s_result
            float 0x0
            space 4
        }
end
```

The error stimulus can be defined as an iterative block of instructions. According to current example, basic arithmetic instructions from extension I and instructions from extension M will be used as a stimulus:

```
def err_stimuls
    iterate {
        Or x(_ FREE, :exclude => @xreg_excl), x(_
FREE, :exclude => @xreg_excl), x(_ FREE, :exclude =>
@xreg_excl)
        ori x(_ FREE, :exclude => @xreg_excl), x(_
FREE, :exclude => @xreg_excl), _
        xor x(_ FREE, :exclude => @xreg_excl), x(_
FREE, :exclude => @xreg_excl), x(_ FREE, :exclude =>
@xreg_excl)
        <...>
        And x(_ FREE, :exclude => @xreg_excl), x(_
FREE, :exclude => @xreg_excl), x(_ FREE, :exclude =>
@xreg_excl)
    }
end
```

To define the basic block of the test program, a block with the parameter "`:combinator =>'product'`" is used to enable the template processor to construct all possible combinations from the sequences returned by the iterators of the error stimulus block and the target instruction block. This basic block demonstrates an example for three single-precision number comparison target instructions and the case where the target instructions do not use affected by read and write operations registers:

```
def run
    block(:combinator => 'product') {
    la addr_store=x(_), :float_s_result
    la addr_val=x(_), :float_s_data
    flw freg_val=f(_), addr_val, 0

    @xreg_excl = [addr_store, addr_val]
    @freg_excl = [freg_val]

    iterate {
        feq_s x(_ FREE, :exclude => @xreg_excl), f(_
FREE, :exclude => @freg_excl), f(_ FREE, :exclude =>
@freg_excl)
        flt_s x(_ FREE, :exclude => @xreg_excl), f(_
FREE, :exclude => @freg_excl),    f(_ FREE, :exclude =>
@freg_excl)
        fle_s x(_ FREE, :exclude => @xreg_excl), f(_
FREE, :exclude => @freg_excl), f(_ FREE, :exclude =>
@freg_excl)
    }

    err_stimuls
    fsw freg_val, addr_store, 0
    }.run
end
 }
```

MicroTESK for RISC-V. Generator creates an assembler test file, a reference execution trace, an ld-script for linking the assembler test and an ELF test file based on the written templates. The default formal specifications and configuration are used for this paper.

Test Environment. Assembly tests and linker scripts generated by MicroTESK are compiled into RISC-V ELF using GCC [11] in this environment. The resulting executables files are fed to the input of the RTL model of the microprocessor under test, with IntelFPGA QuestaSim [12] responsible for simulation. Furtherly, the comparator compares the traces of test programs executed by the RTL model for deviation of the model's behavior from the reference one. Its result is the output of traces that do not match the reference behavior. To evaluate the tests, the vcover utility included in the Questasim package is used to obtain a detailed report on the code coverage of the tested RTL model.

Tested microprocessors. Testing is performed on RTL models of the following RISC-V microprocessors.

CVA6 — RV64-IMAC single-task microprocessor with an extraordinary 6-stage pipeline from OpenHW Group. It is capable of running Unix-like operating systems as it implements M, S, U privilege levels. It has separate TLB, hardware PTW and transition prediction module in its implementation (target branch buffer and branch history table).

SCR1 — RV32-IMC MCU-class microprocessor with 2-4 stage pipeline from Syntacore (M-level privileges).

IV. Status and Issues

Currently, active work is underway to develop test patterns for the MicroTESK for RISC-V generator. 41 sources of microprocessor error descriptions were analyzed, from which 3042 descriptions were obtained. According to the obtained descriptions the following conclusion can be drawn: not all error descriptions can be defined in the form of a test pattern. As an example, the error description [13] (the error index in the document is AAJ4) cannot be realized in the form of a template for RISC-V architecture due to the absence of processor operation mode separation in the target architecture.

It is expected to obtain an extensive test suite created on the basis of the performed analysis. The following iterations are required to obtain it:

- Expand the base of errors with descriptions and templates;
- Analyze the resulting set by applying it together with ISA Tests, Compliance Tests, Torture TG and MicroTESK for RISC-V base templates applied on RTL processor models.

References

[1] *MicroTESK for RISC-V* - http://www.microtesk.org/download/microtesk-riscv/

[2] *SCR1* - https://github.com/syntacore/scr1

[3] *CVA6* - https://github.com/openhwgroup/cva6 (accessed Feb. 08, 2024).

[4] Md Rafiqul Islam Rabin, Mohhamed Amin Alipour. *Configuring Test Generators using Bug Reports: A Case Study of GCC Compiler and Csmith*. SAC-SVT'21.

[5] *RISC-V ISA Tests* - https://github.com/riscv-software-src/riscv-tests

[6] *RISC-V Compliance Tests* - https://github.com/lowRISC/riscv-compliance

[7] *RISC-V Torture* - https://github.com/ucb-bar/riscv-torture

[8] *TSC695 Errata Sheet* - https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/Errata/doc4280.pdf

[9] *Chip Errata for the MPC7447A* - https://www.nxp.com/docs/en/errata/MPC7447ACE.pdf

[10] *ARM AT360 / AT370 Errata Notice* - https://documentation-service.arm.com/static/5ef0b6c5dbdee951c1ccd9cb?token=

[11] *RISC-V GNU Toolchain* - https://github.com/riscv-collab/riscv-gnu-toolchain

[12] *Questa*-Intel® FPGA Starter Edition Software* -

https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/questa-edition.html

[13] Document Number:320836-037US Intel® Core™ i7-900 Desktop Processor Extreme Edition Series and Intel ® Core™ i7-900 Desktop Processor Series - https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/core-i7-900-ee-and-desktop-processor-series-spec-update.pdf