A ARTIFACT APPENDIX

A.1 Abstract

This artifact[8] uses RTL2μSPEC to produce a μSPEC model for the RISC-V multi-V-scale [29, 31], and COATCheck [25] to conduct formal MCM verification of the μSPEC model with respect to 56 litmus tests [31]. Overall, RTL2μSPEC requires two runtime environments:

- `rtl2uspecEnv`, where RTL2μSPEC runs as a C++ extension to Yosys
- `cadEnv`, where Jaspergold is installed and is able to evaluate RTL2μSPEC-generated SystemVerilog Assertions (SVAs).

A.2 Artifact check-list (meta-information)

- **Data set:**
  - *RISC-V multi-V-scale* SystemVerilog design for RTL2μSPEC to consume as input
  - *TCL and Python driver scripts* to support evaluation of RTL2μSPEC-generated SVAs on the multi-V-scale

  Data set components can be accessed here: https://github.com/yaohsiaopid/multicore_vscale_rtl2uspec_ae.git
- **Run-time environment:**
  Running RTL2μSPEC requires:
  - Symbiotic EDA Edition of Yosys
    Please contact edmund@symbioticeda.com and office@symbioticeda.com for academic license.
  - Cadence JasperGold

  Running the full end-to-end MCM verification case study featured in this artifact additionally requires:
  - COATCheck MCM verification tool (*included in container image*)

  To facilitate artifact evaluation, the compilation and execution environments for RTL2μSPEC, including RTL2μSPEC source code (https://github.com/yaohsiaopid/rtl2uspec), and COATCheck have been wrapped as a container image: yaohsiao/micro21:v0.2.3. The image requires the user to obtain Yosys access, as detailed below. *A runtime environment where JasperGold has been installed is required in addition to the container image.*
- **Output:**
  Given the multi-V-scale as input, RTL2μSPEC will produce a μSPEC model, called `vscale.uarch`, along with performance for various parts of the synthesis procedure. As a secondary output, COATCheck will produce qualitative and quantitative MCM verification results by indicating MCM compliance (or not) with sequential consistency (the multi-V-scale's MCM) and verification runtimes, respectively.

A.3 Installation

(1) **Setup the RTL2μSPEC execution environment.** (`rtl2uspecEnv`)
The below assumes that one has reached out to YOSYSHQ and obtained instructions on how to download their software wrapped in a `tar.gz` file and a corresponding license file ending with `.lic`. Our artifact submission features a Docker image that includes all software dependencies, with the exception of JasperGold and Yosys, and requires users to provide the software and license file paths as mentioned (replace <TARGZPATH> and <LICPATH>). Run the commands as follows. **The last line should be executed within the container.**

```
$ export TABBYCAD=<TARGZPATH>
$ export TABBY_LIC=<LICPATH>
$ docker run -itd --name microtest yaohsiao/micro21:v0.2.7
$ docker cp $TABBYCAD microtest:/home/tabbycad.tar.gz
$ docker cp $TABBY_LIC microtest:/home/tabbycad.lic
$ docker attach microtest
$ cd /home && . envsetup.sh
```

This step should end with the following result:

```
export PATH=/opt/symbiotic-20201202A-serp/bin:$PATH
export SYMBIOTIC_LICENSE=/home/symbiotic.lic
==================================
[success] yosys path is at /opt/symbiotic-20201202A-serp/
    bin/yosys
==================================
```

Path to multi-V-scale design: `/home/multicore_vscale_rtl2uspec`
Path to RTL2μSPEC: `/home/rtl2uspec`

(2) **Setup the JasperGold execution environment** (`cadEnv`):
- Confirm that JasperGold can be found in PATH
  `$ which jc`
- Install relevant python3 packages
  `$ yum install -y python3 && python3 -m pip install numpy pandas`
- Populate the multi-V-scale design

```
$ git clone https://github.com/yaohsiaopid/
      multicore_vscale_rtl2uspec_ae.git
    multicore_vscale_rtl2uspec &&
  mkdir multicore_vscale_rtl2uspec/gensva
```

A.4 Experiment workflow

(1) **Intra-instruction HBI synthesis.** In `rtl2uspecEnv`,
`$ cd /home/rtl2uspec && make init && make intra_hbi`
- `make init`: compiles RTL2μSPEC using source files located in `src_revised`. RTL2μSPEC's required user-provided design annotations are supplied as a header file, `src_revised/design.h`. For example, `src_revised/design.h` includes design information like the instruction fetch register (`IFR`) signal name, which is declared as a `string` type. The value of the `IFR` `string` is the hierarchical name in the RTL design of the state element that stores instructions when they are first fetched from instruction memory on a given core. For the multi-V-scale, the `IFR` is the `core_gen_block[0].vscale.pipeline.inst_DX` signal, and it is instantiated concretely in the multi-V-scale design files (`/home/multicore_vscale_rtl2uspec/src/main/verilog`). The `src_revised/design.h` header file is also used to specify which ISA instructions should have their behavior formalized and included in the final μSPEC model. This is done by enumerating (`opcodes_name, valid_exe_condition`) pairs, where `opcodes_name` is a `string` name for an instruction of interest, and `valid_exe_condition` describes the how to recognize the instruction of interest from its binary encoding. Given the focus of our paper is on extracting μSPEC models for conducting MCM verification, `src_revised/design.h` specifies two relevant ISA instructions for the multi-V-scale: `sw` (appears first, and thus will be referred to with ID 0 by RTL2μSPEC) and `lw` (appears second, and thus will be referred to with ID 1 by RTL2μSPEC).
- `make intra_hbi`: runs CDFG analysis over the Verilog design supplied in `script/multicore_yosys_verific.tcl`, namely the multi-V-scale located at `/home/multicore_vscale_rtl2uspec` in this artifact evaluation. CDFG analysis identifies the set of state elements that are reachable from the user-supplied `IFR` in the input design's netlist and generates corresponding intra-instruction HBI hypotheses in the form of SVAs. These SVAs are output into the folder `build/sva/intra_hbi/`. Metadata files `ever_update_[0-9]+.txt` for each instruction type list relevant state elements to be considered for inclusion in the instruction's execution path, pending the outcome of SVA evaluation. SVAs corresponding to an instruction metadata file can be found in a `ever_update_[0-9]+.sv` file with the same integer

[8] Official artifact can be found here: https://doi.org/10.5281/zenodo.5492990.

ID. These integer IDs match the order in which instructions were enumerated in the src_revised/design.h file. The result should be

```
build/sva/intra_hbi/
|-- ever_update_0.sv
|-- ever_update_1.sv
`-- .... several other files
```

(2) **Intra-instruction HBI hypothesis evaluation.**
- Copy the folder /home/rtl2uspec/build/sva/intra_hbi/ in rtl2uspecEnv to cadEnv under multicore_vscale_rtl2uspec/gensva/.
- Evaluate SVAs in cadEnv:
  $ python3 revised_script/intra_hbi.py
  The script invokes JasperGold to evaluate the SVA files in the folder and, based on the results (proven/cex), generates a modified version of metadata file ever_update_[0-9]+.txt, called ever_update_[0-9]+.txt.res.
  This file features a new field for each row (updated/fixed), which indicates whether the instruction of interest (denoted by the file ID) does/does not update the state element of interest (denoted by a row of the file).
  Upon termination of SVA evaluation, the script prints out total number of SVAs evaluated and the total runtime, **which should match the first two rows of the Intra-Instr. column in Fig. 5 in the paper.**

```
================================================
Total time on intra-instruction HBI (sec) : 271.063000
Total number of SVA evaluated: 105
================================================
```

- Copy the folder multicore_vscale_rtl2uspec/gensva/intra_hbi from cadEnv back to rtl2uspecEnv to replace original folder /home/rtl2uspec/build/sva/intra_hbi/ so that rtl2uspecEnv has the updated metadata files.

(3) **Inter-instruction HBI synthesis.** In rtl2uspecEnv,
$ cd /home/rtl2uspec && make inter_hbi
Based on the results from previous step (intra-instruction HBI evaluation), this step deduces per-instruction DFGs, and iterates over all pairs of per-instructions DFGs to generate all inter-instruction hypotheses. The result of inter-instruction HBI synthesis will be stored in build/sva/inter_hbi/ and be structured as follows:

```
gensva/
|-- inter_hbi
|   |-- 0.sv
|   |-- 1.sv
|-- |-- .... several other files
|   |-- hbi_meta.txt
|   `-- hbi_meta.txt.detail
`-- intra_hbi
    |-- .... several other files
```

hbi_meta.txt.detail listed all generated inter-instruction HBI hypotheses (one per row) that will be evaluated along with their corresponding SVA file (in the file_# field of the list). One of the rows in hbi_meta.txt.detail should look like the following to indicate this hypothesis is validated by the SVA contained in 0.sv.

```
file_#,hbi_type,samecore,i0_type,i1_type,i0_loc,i1_loc,...
0,0,1,0,0,core_gen_block[0].vscale.pipeline.ctrl....
```

hbi_meta.txt contains metadata pertaining to all unique SVAs that will be used to validate all inter-instruction HBI hypotheses.

(4) **Inter-instruction HBI hypothesis evaluation.**
- Copy the folder /home/rtl2uspec/build/sva/inter_hbi/ in rtl2uspecEnv to cadEnv under multicore_vscale_rtl2uspec/gensva/.
- Evaluate SVAs cadEnv:
  $ python3 revised_script/inter_hbi.py
  As in intra-instruction HBI evaluation, this script invokes JasperGold for each SVA files in the inter_hbi/. Based on the results (proven/cex) a modified version of hbi_meta.txt, called hbi_meta.txt.res, is generated, which includes a new field for each row (updated/fixed). As before, the script prints out total number of SVAs evaluated and the total runtime, **which should match to first two rows of the Inter-Instr. column of Fig. 5 in the paper.**

```
================================================
         (Spatial)|  (Temporal)|    Dataflow|
cnt              1|          12|           2|
time      5.347000|   31.632000|   15.801000|
================================================
```

- Copy the folder multicore_vscale_rtl2uspec/gensva/inter_hbi from cadEnv back to /home/rtl2uspec/build/sva/inter_hbi/ in rtl2uspecEnv. rtl2uspecEnv should now have new files, namely /home/rtl2uspec/build/sva/inter_hbi/hbi_meta.txt.res

(5) **μSPEC generation.** In rtl2uspecEnv,
$ cd /home/rtl2uspec && make uspec .
This pass aggregates the results from previous steps, merges state elements having the same ordering behaviors into "mega-nodes," and generates the final μSPEC model, named vscale.uarch. The mega-nodes will be instantiated as single nodes during instruction execution path enumeration in the μSPEC model. Part of this pass also includes a syntatic translation of the proven HBI hypotheses to the μSPEC DSL. An excerpt of the μSPEC model generated by our artifact evaluation is included below for reference.

```
StageName 0 "IF_".
StageName 1 "mgnode_2".
StageName 2 "mgnode_0".
StageName 3 "hasti_mem_mem".
StageName 4 "mgnode_3".
StageName 5 "mgnode_1".

% ProgramOrder
Axiom "PO_man": forall microop "i1",  forall microop "i2",
    SameCore i1 i2 => ProgramOrder i1 i2 =>
      AddEdge ((i1, IF_), (i2, IF_), "PO", "orange").
```

A.5 Evaluation and expected results

Our artifact evaluates the synthesized μSPEC model against a suite of litmus tests using the COATCheck MCM verification tool. In rtl2uspecEnv,
$ cd /home/rtl2uspec && make eval_uspec
This step obtains a suite of litmus tests [31] to evaluate compliance of a μSPEC model with Sequential Consistency (the MCM of the multi-V-scale). It then uses COATCheck to evaluate the RTL2μSPEC-generated μSPEC model against these same litmus tests. An example of the results that should be generated is shown below. Each row features the name of a litmus test and the runtime (ms). Runtimes correspond to **blue performance bars Fig. 6 of the paper. The final line of output should also indicate that none of the litmus tests fail to execute in a Sequentially Consistent manner**, demonstrating that COATCheck has proven the multi-V-scale to implement Sequential Consistency with respect to the litmus tests considered.

```
.....
safe027.test,29.083897
safe029.test,16.207506
safe030.test,22.950519
sb.test,11.006003
ssl.test,16.676122
wrc.test,23.565418
--- 1379.073456 ms ---
======= ALL TESTS PASSES =======
```