Agile Formal Verification with Symbolic Quick Error Detection by Semantically Equivalent Program Execution

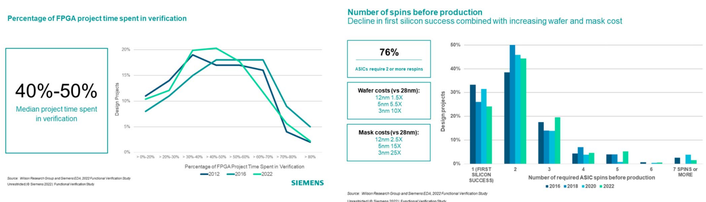
[1]Yufeng Li, [2]Qiusong Yang, [2]Yiwei Ci, [2,3]Enyuan Tian, [1]Yungang Bao, [1]Kan Shi crazybinary494@gmail.com, {qiusong, yiwei}@iscas.ac.cn, tianenyuan@nfs.iscas.ac.cn, {baoyg, shikan}@ict.ac.cn

[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; [2]Institute of Software, Chinese Academy of Sciences, Beijing, China; [3]University of Chinese Academy of Sciences, Beijing, China

1. Introduction

Processor verification is hard

- Highly complex microarchitectural optimizations
- Simulation inadequate
- Formal verification
  - Requires manual specify all desired properties
  - Requires in-depth microarchitectural knowledge
  - Requires high verification expertise
  - Time-consuming and error-prone
- Consumes $\geq$ 50% of design effort, yet only 24% of projects were able to achieve first silicon success
  - Modern processor designs have become substantially more complex, while supply chain challenges, including global competition and geopolitical risks, have shortened development timelines, thereby creating an urgent demand for innovative verification methodologies



Percentage of verification effort in development — Required number of spins

Figure 1: Wilson Research Group Study Results

2. Formal Verification Transformation: Symbolic Quick Error Detection (SQED)

QED (Quick Error Detection) + BMC (Bounded Model Checking):

- QED is a technique that converts original tests into QED family tests:

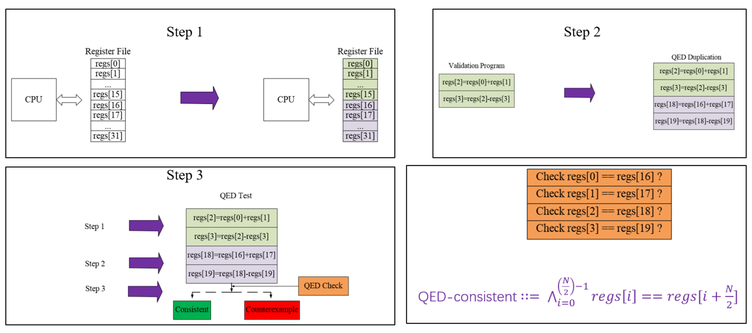


Figure 2: QED

  - Partition processors' "locations" (memory, registers) into original and duplicate spaces, with a one-to-one correspondence between them
  - Each original instruction is paired with a duplicate, operating on separate halves of the space
  - During testing, a functionally correct processor transitions between QED-consistent states, where values in the original and duplicate spaces are identical
- BMC tool automatically explores all valid symbolic original instructions
- The QED module in the pipeline's fetch stage transforms the original instruction into a duplicate and dispatches them into the pipeline
- self-consistency universal property: $QED\text{-}ready \Rightarrow QED\text{-}consistent$

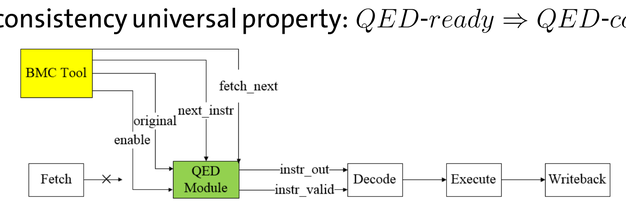


Figure 3: SQED-based formal verification

Superior:

- Automatic
  - Only a universal property to check - self-consistency
  - Independent of microarchitectural details
  - Requires low manual effort and low formal expertise
- Robust
  - Symbolically enumerates all programs up to n instructions
- Quick
  - Finds the shortest possible bug trace from an initial state
- Agile
  - 60X productivity
- Effective (vs. Traditional methods)
  - 100X reduction in time
  - 10^6X reduction in bug trace length

3. Limitations of Current Research

Logic bugs in a processor can be categorized as:

- Single-instruction bugs
  - It depends on the specific opcode and operands, independent of program context
  - Activating the bug and propagating it to program-visible states occur within a single instruction
- Multiple-instruction bugs
  - Program context is relevant
  - A sequence activates the bug, and then an instruction propagates it to program-visible states

Single-instruction bug: ADD opcode and zero operands

```
ADDI regs[1], regs[0], 0x4      // regs[1]=0x4
ADDI regs[2], regs[0], 0x2      // regs[2]=0x2
ADDI regs[3], regs[0], 0x0      // regs[3]=0x0
ADDI regs[4], regs[0], 0x0      // regs[4]=0x0
ADD regs[5], regs[1], regs[2]   // regs[5]=0x6
ADD regs[6], regs[3], regs[4]   // regs[6]=0xffff
```

Multiple-instruction bug: two consecutive multiplication instructions

```
MULH regs[3], regs[1], regs[2]      // MULH executed correctly
NOP                                 // Bug is not activated
MULH regs[19], regs[17], regs[18]   // MULH executed correctly
NOP                                 // Bug is not activated
MULH regs[3], regs[1], regs[2]      // Bug is activated
MULH regs[19], regs[17], regs[18]   // Bug occurs: the source operand is corrupt
```

Figure 4: Logic bug type

SQED cannot detect single-instruction bugs!!!

- Single-instruction bugs affect original and duplicate instructions in a QED test in the same way
  - Original and duplicate registers always hold the same value, thus maintaining self-consistency

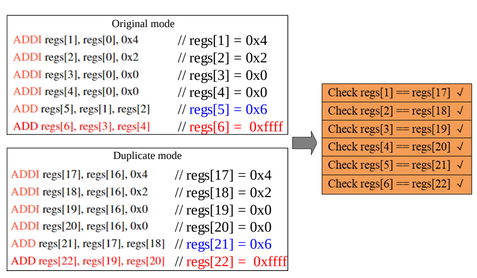


Figure 5: False positive of self-consistency verification

  - Insufficient detection of all types of logic bug
    - Broad operand range impairs simulation
    - Non-universal formal properties are dependent on microarchitectural details

4. Contribution

- Generalized self-consistency universal property
  - Single-instruction bugs unevenly impact original and semantically equivalent instructions

```
// Initialize:  regs[1] == regs[17],
//              regs[2] == regs[18],
//              regs[3] == regs[19],
//              regs[1] ≠ 0x0,
//              regs[2] ≠ 0x0

// Original instruction
ADD regs[3], regs[1], regs[2] // regs[3] = regs[1] + regs[2]

// Semantically equivalent sequence
SUB regs[20], regs[17], regs[17]  // regs[20] = 0
SUB regs[21], regs[20], regs[18]  // regs[21] = -regs[18]
SUB regs[19], regs[17], regs[21]  // regs[19] = regs[17] + regs[18]

Check regs[3]  == regs[19]
```

```
// Initialize:  regs[1] == regs[17],
//              regs[2] == regs[18],
//              regs[3] == regs[19],
//              regs[1] == 0x0,
//              regs[2] == 0x0

// Original instruction
ADD regs[3], regs[1], regs[2]          // regs[3] = regs[1] + regs[2]

// Semantically equivalent sequence
SUB regs[20], regs[17], regs[17]       // regs[20] = 0
SUB regs[21], regs[20], regs[18]       // regs[21] = -regs[18]
SUB regs[19], regs[17], regs[21]       // regs[19] = regs[17] + regs[18]

Check regs[3] (0xffff) ≠ regs[19] (0x0)
```

Figure 6: Insight

  - Correct processor execution of both original and semantically equivalent instruction sequences should result in consistent architectural states

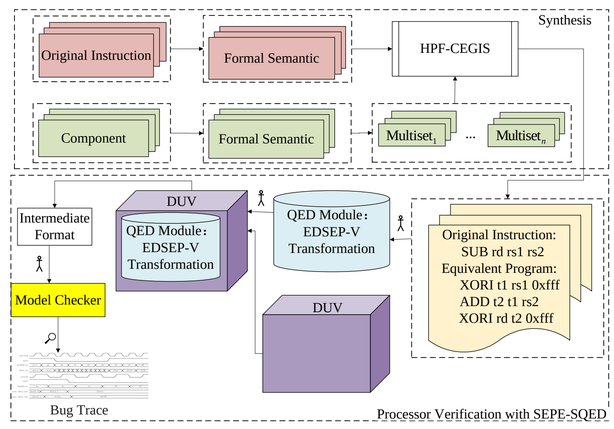


Figure 7: SEPE-SQED

- Counterexample-Guided Inductive Synthesis based on the Highest Priority First (HPF-CEGIS)
  - Using heuristic synthesis to find instruction sequences semantically equivalent to the original instructions
    - Prioritize selecting instructions that are semantically close to the original instruction as synthesis elements
    - Dynamically adjust priorities based on feedback mechanisms

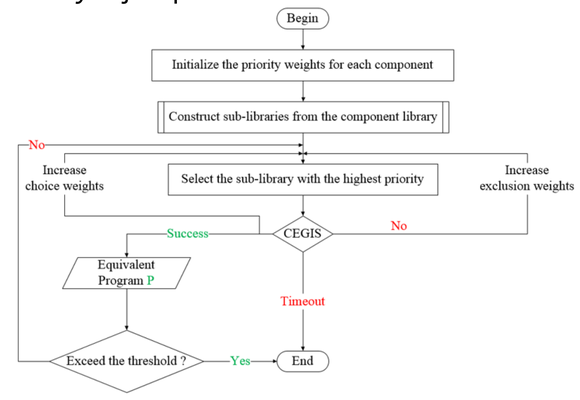
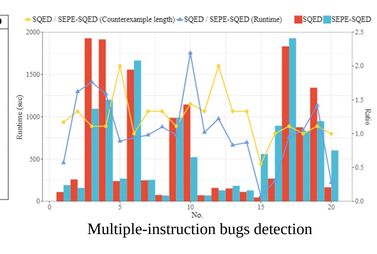


Figure 8: HPF-CEGIS

5. Evaluation

- SEPE-SQED detected all injected logic bugs
- Exhibiting shorter bug detection time and bug trace length for some bugs

Type	Function	SEPE-SQED	SQED
ADD	Addition of two register types	3410.93s	-
SUB	Subtraction of two register types	1436.46s	-
XOR	Exclusive-OR	430.47s	-
OR	Bitwise OR of two register types	1765.66s	-
AND	Bitwise AND of two register types	777.79s	-
SLT	Set if Less Than	3306.27s	-
SLTU	Set if Less Than, Unsigned	2437.10s	-
SRA	Shift Right Arithmetic	76.50s	-
MULH	Multiply High	2837.13s	-
XORI	Exclusive-OR Immediate	627.45s	-
SLLI	Shift Left Logical Immediate	1837.11s	-
SRAI	Shift Right Arithmetic Immediate	85.44s	-
SW	Store Word	288.62s	-



Single-instruction bugs detection — Multiple-instruction bugs detection

Figure 9: Experimental results on the RISC-V out-of-order superscalar processor RIDECORE

6. Conclusion

- The first proposition of a universal property to cover all types of logic bug in a processor
  - Eliminating the need for manually writing complex properties speeds up the verification process
  - Greatly reducing the barrier to formal verification