Reusable Processor Verification Methodology Based on UVM

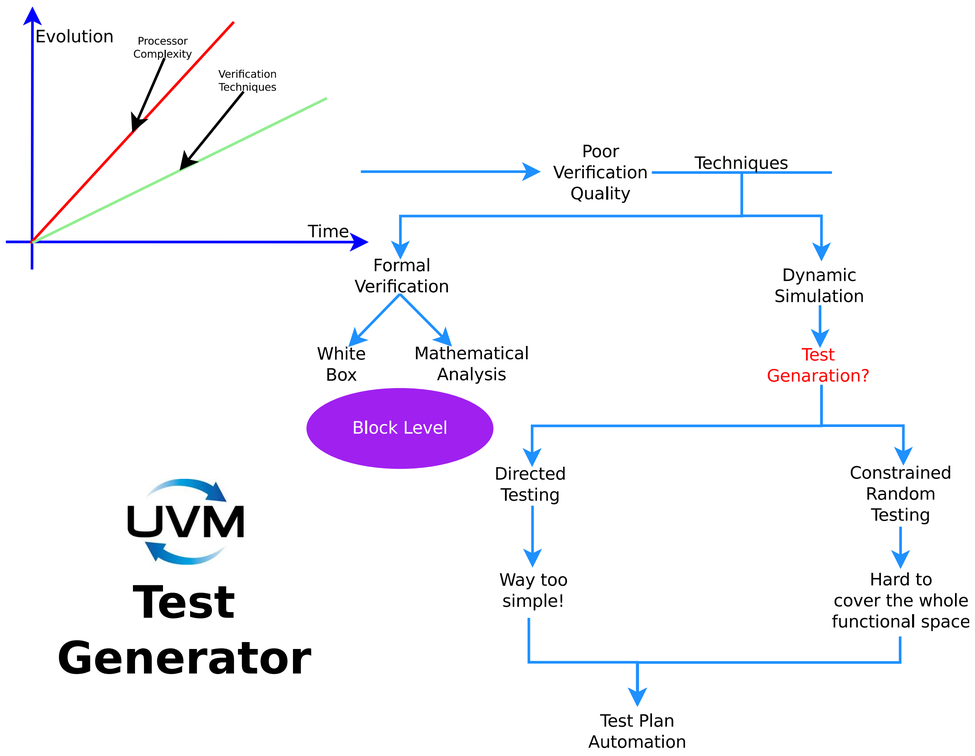
Mustafa Khairallah and Maged Ghoneima

Boost Valley for Engineering Services

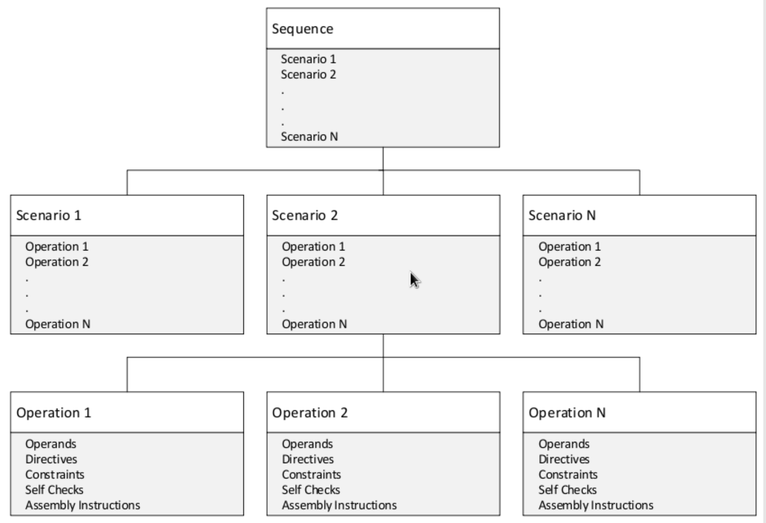
SYSTEMS INITIATIVE™

2014 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Introduction



Methodology



Sequence: High level interface of the stimulus generator.
- It combines one or more scenarios to generate the final test-case.

Scenario: Contains a group of operations along with constraints on the interactions between them.
- Scenarios simplify tests generated, as scenarios are classified according to what they are testing.

Operation: An actual micro-architectural operation that the processor can perform containing:
- Operands, randomization directives and constraints.
- Self-checks for the expected behavior of the operation.
- Actual assembly instructions needed to perform this operation.

TEST TEMPLATE DESIGN

The implementation of our methodology is done in a bottom up manner:

(1) Processor functional space is divided into families of possible execution scenarios, namely scenario families. Each of these families contains a part of the processor‟s functionality.

(2) For each family a group test templates is designed.

(3) The operations required to execute each scenario, and their relevant properties are defined.

(4) The test generator (sequence) is defined and used to generate scenarios within one or more scenario families.

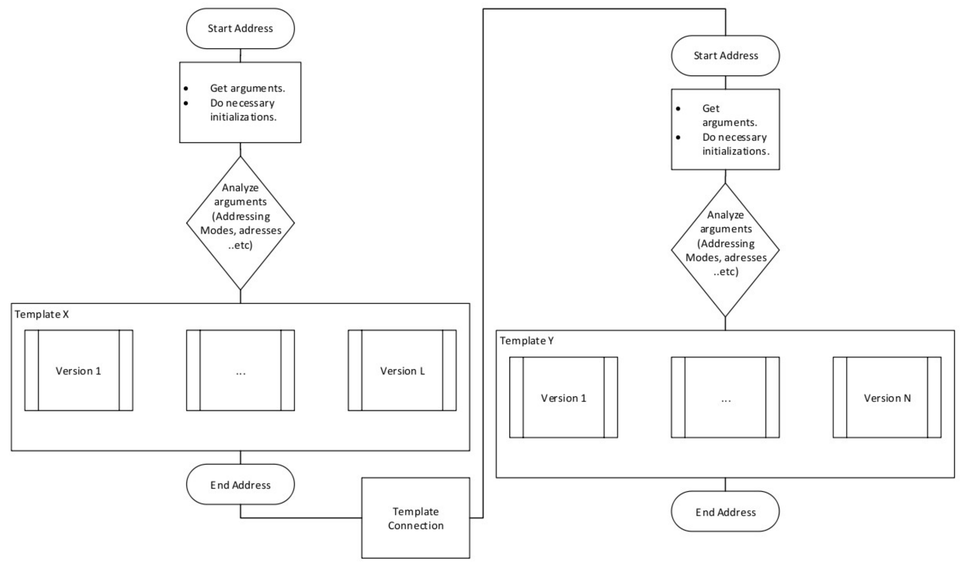
A scenario family runs a group of similar scenarios. Each scenario family is a UVM Sequence Item class; it has some properties like scenario name,memory map, input and output data, register name, addressing mode, operation name, etc.

```
function int jump_template (int start_adr);
int i;
i = start_adr;
.
mem_array[i+x]
= JP;
mem_array[i+x+1] = jp_address [7:0];
.
jump_template =
(i+LENGTH_OF_TEMPLATE);
endfunction // jump_template
```

After properties are defined, constraints are defined to describe the relationships between different properties. For example: a shift scenario should execute shift and rotate operations, data addresses should be outside the space of the program inside the memory map ...etc.

TEST TEMPLATE DESIGN

As shown below, these templates can be combined together to build complex scenarios, without the need to design these complex scenarios from the beginning. For example, a conditional branch in a program can have a low probability of being executed if the data it depends on are purely random. Such problems are handled using the constraints attached to each scenario family. The template connection block in the figure below checks where the previous template has ended, and calculates the correct address for the next template to start at.



Experimental Results: WISHBONE Z80 PROCESSOR CASE STUDY

Test/Scenario Family	No. of Created Templates	No. of Test Cases	No. of Cycles	No. of Events Covered	% of Reuse
Load/Store	7	2,912	38K	85	0%
Basic ALU	18	29,192	421K	332	20%
Advanced ALU	18	245	9.5K	73	70%
Jump/Branching	5	552	5K	14	20%

The verification environment implemented is a Coverage Driven Environment, so each test is terminated when some events are covered. An event is one possible occurrence of a property from the processor‟s specification.

For example, load from register A to register B, add two values with a carry generated, perform a Read-After-Write operation from Memory ...etc.

The required events are written inside a coverage collector. The percentage of reuse is the ratio between the reused code lines from older scenarios to the total code lines required to implement a certain scenario.

During our testing process several bugs were found. Among them some instructions, especially those with different addressing modes, or in Read-After-Write (RAW) hazards, were found to function incorrectly when issued in a row, and have to be separated by No-Operation instructions. Another major bug was found in branching instructions, where the program counter is not loaded correctly.

Conclusion

The ever-growing complexity of processor architectures and micro-architectures create a gap between the verification requirements and the test generation mechanisms available.

In this paper, we briefly discussed the latest directions and approaches in processor verification. In addition, we explored the capabilities of UVM to be used efficiently in processor functional verification.

We proposed a new testing methodology for processor verification that can be easily reused for different architectures or micro-architectures, which is based entirely on UVM and System Verilog. We provided a practical example for the adoption of this methodology in the verification of the Open Source Wishbone Bus Z80 Processor.

The results show how our methodology increased the amount of reuse in the later stages of verification and helped cover unseen events and detects several bugs that may not have been discovered using the processor.

Mustafa Khairallah
Design Verification Engineer

E-Mail: mustafa.khairallah@boostvalley.com
Cell Phone: +20-122-480-6242

BOOST VALLEY
Taking you that extra step