Reusable Processor Verification Methodology Based on UVM
Mustafa Khairallah and Maged Ghoneima
Boost Valley for Engineering Services

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 of 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;
mem_array[i+x] = JP;
mem_array[i+x+1] = jp_address [7:0];
endfunction

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.

Experimental Results: WISHBONE Z80

PROCESSOR CASE STUDY

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

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 microarchitectures 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 and microarchitectures, 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.