ON THE REUSE OF RTL ASSERTIONS IN SYSTEMC TLM VERIFICATION

Nicola Bombieri\textsuperscript{1,2}
Franco Fummi\textsuperscript{1,2}, Graziano Pravadelli\textsuperscript{1,2}, Valerio Garnieri\textsuperscript{1}, Francesco Stefanni\textsuperscript{1}. Tara Ghasempouri\textsuperscript{2}, Michele Lora\textsuperscript{2}, Giovanni Auditore\textsuperscript{3}, Mirella Negro Marcigaglia\textsuperscript{3}

\textsuperscript{1} EDALab s.r.l.
\textsuperscript{2} Dip. Informatica, Università di Verona
\textsuperscript{3} STMicroelectronics s.r.l.
Motivations
RTL IP reuse in SystemC TLM platforms

• SystemC and TLM
  – SystemC is the de-facto reference standard language for design and verification of Embedded Systems at system level
  – Transaction-level Modeling is the key paradigm for design and verification at high abstraction levels
• RTL IP reuse
  – Libraries of RTL IPs are available today, already stressed and verified
  – RTL IPs are mainly implemented in Hardware Description Language (HDL), such as VHDL or Verilog
• Design teams cannot often maintain double and equivalent implementations of IPs (RTL and TLM)
  – IP models often undergo manual interventions for optimization
  – Today, actually, optimizations are done at RTL
  – Optimizations over existing and already verified IPs are still expensive

Automatic RTL-TLM abstraction
Motivations (contd.)
IP verification after abstraction

- Existing tools for automatic RTL-TLM abstraction!
- Existing methodologies for verifying:
  - The abstracted TLM IP model
  - The TLM IP correct integration
- Assertion-based Verification (ABV)

What about existing RTL IP assertions?
**Goal:**
RTL assertion reuse in SystemC TLM platforms

With the aim of:
1. Avoid error-prone, time-consuming assertion re-definition
2. Reuse verification effort spent at RTL (for RTL assertion definition)

What happens to the TLM performance?
**Limits of related work:**

**Assertion-based Verification in SystemC TLM**

- **ABV in SystemC TLM:**
  - [Habibi-IEEE Trans.VLSI’06] First contribution for cycle-accurate TLM
  - [Ecker-IEEE ICCD’06, MEMOCODE’06, DATE’07] Proposal of specific assertion language for SystemC TLM
  - [Lahbib-IEEE DTIS’06] IBM FoCs assertion synthesis into SystemC TLM
- **Automatic generation of checkers for ABV in SystemC TLM:**
  - [Ferro-IEEE IDTL’08, FDL’09]
- **Formal tools for ABV in SystemC TLM:**
  - [Grosse-IEEE MEMOCODE’10]
- **TLM assertion reuse at RTL**
  - [Bombieri-IEEE DATE’07, Kasuya-DAC’07, Pierre-CODES’13]

No work for reusing RTL assertions in SystemC TLM
Methodology: generation of checkers from assertions and integration in the TLM model

- Two ways:
  1. Generation of HDL checkers, integration, and abstraction
  2. Generation of C++ checkers, abstraction, and integration
2: Automatic RTL-to-TLM abstraction of IPs

How the HDL scheduling works

How the TLM scheduling works

Synchronous process

Asynchronous process

“Synchronous” function

“Asynchronous” function

Clock Cycle $i$

Clock Cycle $i+1$

Scheduling: $ps_1$, $ps_2$, $ps_3$, $ps_4$

Execution: $ps_1$, $ps_2$, $ps_3$, $ps_4$

Scheduling: $pa_1$

Execution: $pa_1$

Rising edge ($\delta$-cycle 0)

Falling edge ($\delta$-cycle 0)

Scheduling: $ps_1$, $ps_2$, $ps_3$, $ps_4$

Execution: $ps_1$, $ps_2$, $ps_3$, $ps_4$

Scheduling: $pa_1$

Execution: $pa_1$

Simulation time

TLM model

scheduler(
    rising_edge();
    while(events_triggered) {
        delta_cycle();
    }
    falling_edge();
    while(events_triggered) {
        delta_cycle();
    }

fa1()
fs4()
fs3()
fs2()
fs1()
1. Generation of C++ checkers

- Example of RTL assertion: An input A or B high is always followed by output C high

```plaintext
-- psl P1: assert always A or B -> next C@(clk'event and clk='1');
```

A

B

C

clock

P1() {
  ...
}

Checker invocations

Generally clocked assertions
3: C++ checkers integration in the TLM model

- Generation of C++ checkers
- Integration of checkers
- Less time-consuming than HDL checkers integration
- Less overhead introduced than HDL checkers

C++ checkers

TLM model

PSL assertions
Some experimental results:

<table>
<thead>
<tr>
<th>Design</th>
<th>Processes (#)</th>
<th>RTL loc</th>
<th>Pipeline stages (#)</th>
<th>Latency (cc)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asynch.</td>
<td>Synch.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART</td>
<td>416</td>
<td>77</td>
<td>5866</td>
<td>–</td>
</tr>
<tr>
<td>Root</td>
<td>2</td>
<td>0</td>
<td>343</td>
<td>–</td>
</tr>
<tr>
<td>Div</td>
<td>1</td>
<td>5</td>
<td>1283</td>
<td>–</td>
</tr>
<tr>
<td>QNR</td>
<td>7</td>
<td>17</td>
<td>518</td>
<td>16</td>
</tr>
<tr>
<td>RLE</td>
<td>14</td>
<td>17</td>
<td>678</td>
<td>9</td>
</tr>
<tr>
<td>FDCT</td>
<td>259</td>
<td>196</td>
<td>5935</td>
<td>388</td>
</tr>
<tr>
<td>JPEG</td>
<td>281</td>
<td>231</td>
<td>18381</td>
<td>80</td>
</tr>
<tr>
<td>Error Correction</td>
<td>6</td>
<td>11</td>
<td>1666</td>
<td>–</td>
</tr>
<tr>
<td>Lambda Root</td>
<td>0</td>
<td>5</td>
<td>1092</td>
<td>–</td>
</tr>
<tr>
<td>Omega Phy</td>
<td>17</td>
<td>4</td>
<td>1595</td>
<td>–</td>
</tr>
</tbody>
</table>
### Some experimental results

<table>
<thead>
<tr>
<th>IP model</th>
<th>Checkers (#)</th>
<th>RTL (s)</th>
<th>Overhead (%)</th>
<th>TLM (s)</th>
<th>Overhead (%)</th>
<th>Speedup (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>24,19</td>
<td>-</td>
<td>11,61</td>
<td>-</td>
<td>2,08</td>
</tr>
<tr>
<td>UART</td>
<td>2</td>
<td>54,69</td>
<td>126,13</td>
<td>23,71</td>
<td>104,22</td>
<td>2,31</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>588,71</td>
<td>976,42</td>
<td>458,94</td>
<td>1.835,64</td>
<td>1,28</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>22,75</td>
<td>-</td>
<td>19,94</td>
<td>-</td>
<td>1,14</td>
</tr>
<tr>
<td>Root</td>
<td>2</td>
<td>97,44</td>
<td>328,32</td>
<td>37,00</td>
<td>85,55</td>
<td>2,63</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1.422,16</td>
<td>1.359,55</td>
<td>1.203,10</td>
<td>3.151,79</td>
<td>1,18</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>46,03</td>
<td>-</td>
<td>20,79</td>
<td>-</td>
<td>2,21</td>
</tr>
<tr>
<td>Div</td>
<td>2</td>
<td>125,43</td>
<td>172,51</td>
<td>23,03</td>
<td>10,87</td>
<td>5,44</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1.528,48</td>
<td>1.118,61</td>
<td>665,41</td>
<td>2.787,44</td>
<td>2,30</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>105,59</td>
<td>-</td>
<td>18,57</td>
<td>-</td>
<td>5,69</td>
</tr>
<tr>
<td>FDCT</td>
<td>2</td>
<td>209,65</td>
<td>98,55</td>
<td>34,58</td>
<td>86,24</td>
<td>6,06</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>2.250,88</td>
<td>973,65</td>
<td>1.054,86</td>
<td>2.950,75</td>
<td>2,13</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>94,54</td>
<td>-</td>
<td>12,09</td>
<td>-</td>
<td>7,82</td>
</tr>
<tr>
<td>QNR</td>
<td>2</td>
<td>202,46</td>
<td>114,15</td>
<td>25,45</td>
<td>110,57</td>
<td>7,95</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>2.110,55</td>
<td>942,43</td>
<td>950,84</td>
<td>3.635,83</td>
<td>2,22</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>96,12</td>
<td>-</td>
<td>12,99</td>
<td>-</td>
<td>7,40</td>
</tr>
<tr>
<td>RLE</td>
<td>2</td>
<td>219,80</td>
<td>128,66</td>
<td>28,19</td>
<td>117,12</td>
<td>7,80</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>2.207,52</td>
<td>904,35</td>
<td>985,70</td>
<td>3.396,25</td>
<td>2,24</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>307,83</td>
<td>-</td>
<td>42,02</td>
<td>-</td>
<td>7,33</td>
</tr>
<tr>
<td>JPEG</td>
<td>2</td>
<td>622,94</td>
<td>102,37</td>
<td>82,96</td>
<td>97,42</td>
<td>7,51</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>6.257,01</td>
<td>904,43</td>
<td>3.084,88</td>
<td>3.618,47</td>
<td>2,03</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>197,56</td>
<td>-</td>
<td>34,77</td>
<td>-</td>
<td>5,68</td>
</tr>
<tr>
<td>Error_corr.</td>
<td>2</td>
<td>386,73</td>
<td>95,76</td>
<td>70,02</td>
<td>101,38</td>
<td>5,52</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>3.971,00</td>
<td>926,80</td>
<td>2.603,49</td>
<td>3.618,47</td>
<td>1,53</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>487,54</td>
<td>-</td>
<td>69,13</td>
<td>-</td>
<td>7,05</td>
</tr>
<tr>
<td>Lambda_err.</td>
<td>2</td>
<td>791,15</td>
<td>62,27</td>
<td>121,79</td>
<td>76,18</td>
<td>6,50</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>7.406,19</td>
<td>836,13</td>
<td>3.568,10</td>
<td>2.829,69</td>
<td>2,08</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>487,54</td>
<td>-</td>
<td>80,94</td>
<td>-</td>
<td>6,02</td>
</tr>
<tr>
<td>Omega-phy</td>
<td>2</td>
<td>935,10</td>
<td>91,80</td>
<td>144,80</td>
<td>78,91</td>
<td>6,46</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>8.945,88</td>
<td>856,67</td>
<td>3.978,45</td>
<td>2.647,53</td>
<td>2,25</td>
</tr>
</tbody>
</table>
Conclusions

- Key idea of the proposed method:
  - to *recover* RTL IP assertions and make them suitable for ABV in SystemC TLM platforms

- Main contributions: *a two steps methodology*
  - A checker generator is adopted to automatically generate run-time checkers from existing RTL assertions
  - Checkers are integrated in the TLM IP models

- Observed results:
  - The overhead introduced by assertions (checkers) automatically generated through the proposed approach is comparable to the overhead introduced by assertions (checkers) manually defined.
  - The best results have been obtained with a limited number of assertions checked at TLM (10-15 per IP).

*More details and results offline. Thank you!*