Hardware Acceleration for UVM Based CLTs

Mohamed Saheel
Rohith M. S.
Andrew Tan
Agenda

• Need for simulation Acceleration
• Simulation accelerator requirements
• CLT preparation to port over to accelerator
• Simulation accelerator Runtime
• Results and Benefits
Need for Simulation acceleration

- Simulation of CLTs going up exponentially
- Integration on new design features which need thorough validation
- Project requirements to add multiple debug features
- Aggressive project execution time and need to hit validation goals
- Need to run real work testcases
- Drive to achieve better performance and throughput in validation environment.
Simulation Accelerator Requirements

- Requirement to support UVM on hardware emulation platform.
- Ability to port over CLT form pure simulation to accelerator with minor tweaks to achieve functionality.
- Ability to support both signal and transaction based modeling
- Provide minimal acceleration out of box
- Support additional debug features like assertions, coverage, etc
- Support interactive debug environment for RTL/UVM debug
- User friendly performance profiler
Porting CLT to Accelerator

1. CLT on pure simulation
2. Accelerator CLT on pure simulation
3. Code modification for functionality
4. Debug/Code modification
5. Run on Accelerator/Debug
6. CLT compiled for accelerator
7. Performance Profiling
8. Code modification
9. Deploy for production usage
CLT Setup on Accelerator

- Signal based modeling was chosen to make porting over CLT easier
- The below setup provided sizeable performance gains while requiring minimal code changes.
- User chooses the hardware top and compiler automatically partitions the design.
Runtime Acceleration Challenges

• Out of box performance is not achievable always given that design might have lot of unfriendly code for acceleration

• Some of the common bottlenecks
  – Too many HW-SW Port’s
  – Clock generation in Testbench which requires frequent synchronization
  – Accessing DUT clocks/signals from testbench frequently
  – Too many force/release from TB to DUT
  – Runtime access to design registers

• Performance bottlenecks needs to be identified through a runtime profiler
Case Study 1

- First CLT identified was a small portion of the design just to measure the potential benefit
- Out of box performance seen on initial bring up was 5-10x

<table>
<thead>
<tr>
<th>Mode</th>
<th>Test1</th>
<th>Test2</th>
<th>Test3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMULATION</td>
<td>11 Min</td>
<td>80 Min</td>
<td>2 Days?</td>
</tr>
<tr>
<td>SIMULATION (Without Assertions)</td>
<td>1 Min</td>
<td>8 Min</td>
<td>8Hrs</td>
</tr>
<tr>
<td>Acceleration</td>
<td>10 Sec</td>
<td>1 Min 6 Sec</td>
<td>~40 minutes</td>
</tr>
</tbody>
</table>

- Profiler showed lot of communication overhead which slows down runtime

---

**HW-SW Communication**

- **SW(Simulator)**
- **HW (Emulator)**
Case Study 2

- A bigger CLT was chosen
- Dut bring up was easy as we choose to preserve signal based modeling.
- Due to presence of sequencer and checker, a lot of transaction are seen on every clock edge. Which is leading to sizable slowdown.
- Optimization being added to reduce transaction between HW/SW boundary
Case Study 2 - Cont

- Profiler helps to identify the bottlenecks
- Explored ways to reduce the time spent in TB and HW-SW boundary

<table>
<thead>
<tr>
<th>The breakup of Communication Time indicates % of Communication Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>==============================================================</td>
</tr>
<tr>
<td>Elapsed Time</td>
</tr>
<tr>
<td>1102.690 s</td>
</tr>
<tr>
<td>Testbench Time</td>
</tr>
<tr>
<td>wait time</td>
</tr>
<tr>
<td>Communication Time</td>
</tr>
<tr>
<td>Synchronization</td>
</tr>
</tbody>
</table>
Runtime Acceleration

- Identified unwanted port’s across HW/SW which are not needed for functionality
- Moved the clock generation to HW side – Clock generation on HW is much faster and reduces overhead
- Optimized the clock access in TB side – access only when needed

<table>
<thead>
<tr>
<th>ORIG</th>
<th>MODIFIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>forever @(posedge clk)</td>
<td>if (ack) @(posedge clk)</td>
</tr>
<tr>
<td>... do something</td>
<td>... do something</td>
</tr>
</tbody>
</table>
Runtime Acceleration - Cont

• Moved some of the Trackers to DUT
  – Trackers record transactions and sends it to C side
  – These can be synthesized – Dependency on simulator can be reduced
Simulation Acceleration Results

• Set up simulation acceleration on a large UVM cluster level test bench with long test runtimes and compared performance.

• Significant reduction of runtime:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Small test</th>
<th>Large test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>103 seconds</td>
<td>870 minutes</td>
</tr>
<tr>
<td>Accelerator</td>
<td>27 seconds</td>
<td>41 minutes</td>
</tr>
</tbody>
</table>

• Performance gain of about 20x after performance adjustments.
• Still able to use waveform debugger tools with simulation acceleration.
Benefits and Applications

• Straight forward porting of code between simulation and acceleration
  – Minimal changes to TB and DUT.
• Ease of integration of inhouse compile and run tools
• Significant reduction of test runtimes.
• Can run more tests and collect results much more quickly.
  – More randomized test cases can be run, better coverage.
  – More data for machine learning applications.
    • Requires bulk amounts of varied data.
• Easier performance analysis
  – Accelerator profiler reports time spent on individual partitions and modules.
Benefits and Applications – Cont

• Seamless debug methodologies help to debug the issues exposed in HW
• Incremental compile feature helps when there is only TB changes are needed – No need to wait for long as in initial compile
• Accelerated VIP’s can be added here seamlessly which gives additional Performance Benefit
• Assertion and Coverage can be enabled in the platform which gives additional verification
Limitations and Improvements

• Emulator Compile times are more than Simulation – Reduces overall throughput if we consider from start to end

• TB will need modifications to avoid the bottlenecks
  – A proper Guidelines needs to be shared to Verification folks on writing it in emulation friendly way
  – Unnecessary usage of clock events can be avoided in TB side
  – Transactor based acceleration is more suitable when the gains are less – Need investment for the existing verification flow

• A dynamic profiler which tells us where exactly or at what time the runtime slows down