Challenges of Formal Verification on Deep Learning Hardware accelerator

Satish, Yellinidi Dasarathanaidu
Overview

• About DELHA
• FV Challenges
• FV approach
• Formal Property Verification
• Transaction Equivalence Verification
• Sequential Equivalence Verification
• Verification Completeness
• Results
• Summary
DELHA – Deep Learning Hardware Accelerator IP
DELHA

- Highly programmable/configurable soft IP
- Supports INT, Fixed and Float point operations
- Master CSM co-ordinate overall data flow
- Compute Engine coordinated by many sub-blocks, each having multiple configurable FSM
- AXI/APB interfaces
- Applications areas of DELHA requires highest safety standard (no room for even minor bug)
- Exhaustive Verification needed
FV Challenges

• Tool Limitations
• Result convergence
• Setup time and initial flow bring-up
• Formal Tool and flow expertise
• Confidence on FV based results
FV approach

• Tool Limitations
  – Divide & conquer approach
  – Separate verification for control and data path

• Results convergence
  – Simplifying the properties/sequences
  – Constraint tuning
  – Bounded proof
FV approach

• Initial Setup time and flow bring-up
  – Writing system Verilog based design properties
  – ABV BFM
  – Automating results checks
  – Flow bring-up with help from CAD team
  – Subsequent derivatives was quicker

• Tool and flow expertise
  – Collaborating with support team and vendors
  – Trainings / discussions

ABV BFM – Assertion Based Verification Bus functional Model
CAD – Computer Aided design team providing tool/flow support
FV approach

• Formal property Verification
  – Control path verification
• Transaction Equivalence Verification
  – Data Path Verification
• Sequential Equivalence Verification
  – Old RTL vs New RTL verification
Formal Property Verification

- Executed on control path
- Divide and conquer on complex blocks
- Targeted sub-units
  - SRAM controllers
  - Control state machine
  - MAC blocks controllers inside Processing Element (PE)
  - Local FSM in PE
Formal Property Verification
Formal Property Verification

• Constraints are extracted from existing simulation based verification

• Run flow
  – Over constrained to wiggle
  – Fine tune assume and properties
  – Slowly relax the constraints till target requirements
  – Finally under-constraints

• End-to-end checks for smaller blocks

• Temporal assertions for medium size blocks
Formal Property Verification

• To gain confidence on assumptions
  – Individual verified blocks are connected with its masters/slaves
  – Proven Assertions converted into assumes
  – Enabling only some control path
  – Black boxing memories
  – Enabling only relevant properties
  – End-to-End checks at higher hier. with directed constraints
Transaction Equivalence Verification

• Data Path verification
• RTL vs C++ model
• Tool limitation
  – SystemC model converted to C++
• Adapting the Behavioural Model to the RTL
• Wide data path constrained to narrow path for faster convergence
Transaction Equivalence Verification

Transaction based equivalence checks

C++ Model  
<table>
<thead>
<tr>
<th>Specification</th>
<th>RTL</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto equivalence checking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verification Algorithms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differences</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transaction Equivalence Verification

• Challenges
  – Mapping boundary pins between RTL and C++ model
  – Correlating the data flow path
  – Choosing right place for cut points
  – Failure debug
    • Bit level to transaction level comparison
    • Visualization and understanding the root cause
Sequential Equivalence Verification

• RTL vs RTL for subsequent releases
  – Useful for matured RTL releases
  – Faster checks for incremental changes
    • Feature additions based on last minute req.
    • Re-partitioning
    • ECO

• No need of writing assertions/properties
  – Auto proof checks generated at ports

• Proof convergence are sometimes challenging
  – Design constraints
  – Checks on critical sub-blocks
Verification completeness

- IP Module Verif
- UVM/OVM verif
- FV
- TEV
- FPV
- SEC
- Match/error
- Results

Properties Assert/cover
Golden ref model
Previous release RTL
Match/error
Pass/Fail
Results

DUT
DUT
DUT
DUT

Coverage
Coverage

OVM/UVM infra setup
Design constraints
Tests/sequences
Scoreboard
Checkers
Results

<table>
<thead>
<tr>
<th>Block</th>
<th>Setup time in min</th>
<th>Run-time to find bug in min</th>
<th>Bugs</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Unit</td>
<td>20</td>
<td>5</td>
<td>7</td>
<td>Full proof</td>
</tr>
<tr>
<td>Control FSM</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>Full proof</td>
</tr>
<tr>
<td>MAC</td>
<td>10</td>
<td>1</td>
<td>11</td>
<td>Bounded</td>
</tr>
<tr>
<td>data path</td>
<td>20</td>
<td>5</td>
<td>8</td>
<td>Bounded</td>
</tr>
<tr>
<td>Interface</td>
<td>10</td>
<td>1</td>
<td>7</td>
<td>Full proof</td>
</tr>
</tbody>
</table>

Results of Formal Verification
Summary

- Verification QoR
- Functional Coverage
- TAT for iterations and fixing bugs
- Saved huge regression runs for SBV
- Less impact on Machines and resources
- Faster debug

TAT – Turn-Around-Time, SBV – Simulation Based Verification
Questions