High-Level Synthesis and Verification

Peter Frey, HLS Technologist
Problem Statement

- Designing your RTL is hard
  - Complex architectures
  - Specifications open to interpretation
  - Many constraints (Power, Linting, DFT, Synthesis)

- Fully debugging your RTL is impossible
  - Massive vector sets for HW and SW
  - Massive integrated SoCs
  - Design cycles under pressure

- Each year
  - Major advances in verification technology, but…
  - The problems still get worse
High-Level Synthesis
High-Level Synthesis

• Synthesizes “Accellera SystemC Synthesizable Subset” to production-quality RTL

• Arithmetic optimizations and bit-width trimming
• User control over the micro-architecture implementation
  – Parallelism, Throughput, Area, Latency (loop unrolling & pipelining)
  – Memories (DPRAM/SPRAM/split/bank) vs. Registers (Resource allocation)

• Multi-objective scheduling
  – Power, Performance, Area

• Hardware exploration is accomplished by applying different constraints
Properties of High-Level Synthesis?

1. Mapping from abstract transactions to pin-accurate protocols

2. Optimizing for performance & area in the target technology

\[ o = f(i,s) \]
Traditional Design Flow vs. HLS Flow

**Functional Specification**

**Architectural Specification**

**RTL Coding and Micro-architecture Optimization**

**RTL Verification**

**Power Analysis Manual Opt.**

**Logic Synthesis**

**SystemC Executable Design**

**High Level Synthesis**

**Power Analysis Automatic Opt.**

**RTL & Formal Verification**

**Logic Synthesis**
HLS Delivers QoF & Crushes RTL Design Time

- Examples of video, imaging and communication projects
- Generated RTL matches power, performance and area
- Projects complete in 10% to 50% of time needed for RTL

98% of RTL area

27% of RTL design & verification time
HLS-enabled Verification
Advances in Verification Technology

- Algorithm
- Specification Document
- TLM
- Testplan
- Assertions
- Coverage Points
- RTL
- Directed Testbench
- UVM
- Constrained Random

Export for SoC integration
Review of Hardware Abstractions

• Algorithmic Model
  – No timing or architecture

• Transaction-Level Model
  – Partitioned for hardware architecture

• RTL Implementation
  – Synthesizable to gates

\[ o = f(i, s) \]
Verification in ESL Platform

- Algorithmic Model can be used as a reference model
  - Can be embedded in SV/UVM environment
- Enables early software development
  - Software-driven testing
- <10 minutes simulation vs. 1 month simulation in RTL
Synthesizable TLM Verification

- Can be simulated effectively with UVM
  - Early start on UVM environment
- Leverage functional testing
- Based on Algorithmic Model, but partitioned for hardware
- Additional testing for internal control
- Limited performance testing
- Simulation ~100x faster than RTL
Coverage-Driven TLM Verification

- **Assertions and Cover Points**
  - Functional
  - SystemC

- **Testplan Coverage**
  - Based on cover assertions
  - Some tests require RTL

- **Code Coverage**
  - Function, Line, Condition/Decision
  - Many C++ based tools
  - Nothing specialized for hardware

```c
int18 alu(uint16 a, uint16 b, uint3 opcode) {
  int18 r;

  switch(opcode) {
    case ADD:
      r = a+b;  break;
    case SUB:
      r = a-b;  break;
    case MUL:
      r = (0x00ff & a)*(0x00ff & b);  break;
    case DIV:
      r = a/b;  break;
    case MOD:
      r = a%b;  break;
    default:
      r = 0;    break;
  }

  assert(opcode<5);
  cover((opcode==ADD));
  cover((opcode==SUB));
  cover((opcode==MUL));
  cover((opcode==DIV));
  cover((opcode==MOD));

  return r;
}
```
RTL Coverage

- RTL Generated from TLM model by HLS
- Reuse SystemC Vectors
  - Will give functional coverage
  - Some gaps in branch/FSM
- Add RTL tests to cover RTL
  - FSM reset transitions
  - Stall tests
- Gives nearly 100% coverage
  - Line, branch, condition
HLS Verification

Algorithm

Specifications

Synthesizable TLM

Assertion

Coverage Points

Directed Testbench

Testplan

UVM

Constrained Random

Formal Equivalence

HLS

RTL

Export for SoC integration
Summary

- Increasing design complexity & shorter design cycles
  - RTL simulation based debug & verification is the bottleneck
  - Faster simulation (or emulation) is not enough on its own

- Moving to higher levels of abstraction for design & debug
  - Focus on verifying functionality, not implementation details
  - Significant simulation performance & debug improvement

- Requiring automated generation of RTL from TLMs
  - Technology targeting
  - Power Performance Area analysis & optimization
  - Verifiably correct by construction

- Adopting HLS methodology shortens verification timescales
  - Majority of functional verification at algorithmic/TLM levels
  - Minimal RTL simulation and/or formal equivalence checks to prove RTL is correct
Thank You!