2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

High-Level Synthesis and Verification

Peter Frey, HLS Technologist

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

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

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

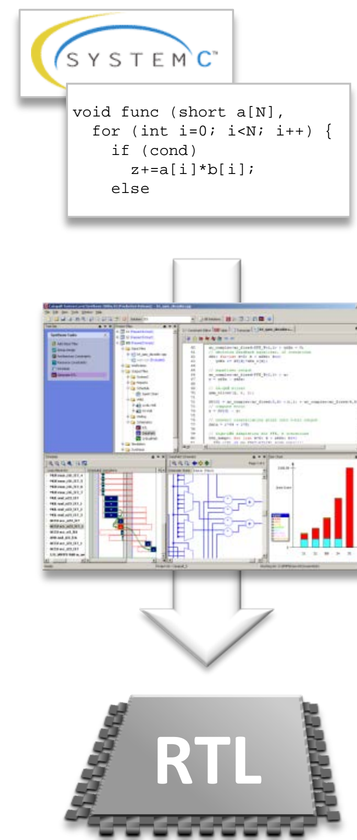
High-Level Synthesis

High-Level Synthesis

- Synthesizes "Accellera SystemC Synthesizable Subset" to production-quality RTL

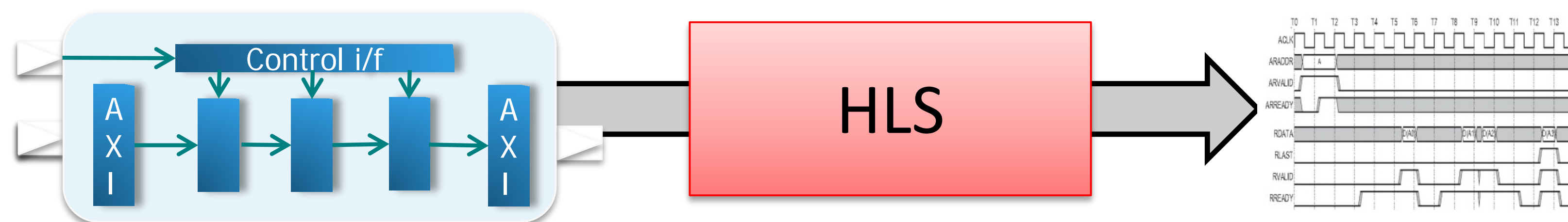
- Arithmetic optimizations and bit-width trimming
- User control over the micro-architecture implementation
  - Parallelism, Throughput, Area, Latency (loop unrolling & pipelining)
  - Memories (DPRAM/SPRAM/split/bank) vs. Registers (Resource allocation)
- Multi-objective scheduling
  - Power, Performance,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

- Hardware exploration is accomplished by applying different constraints

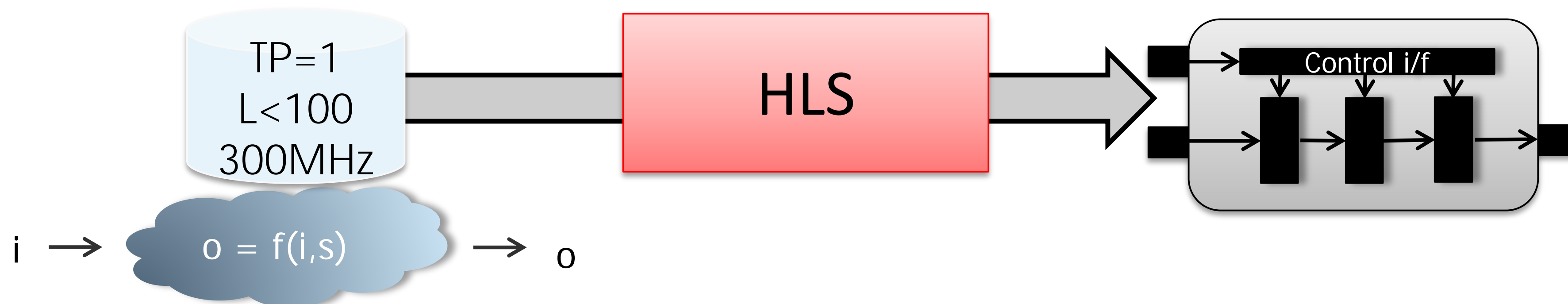


Properties of High-Level Synthesis?

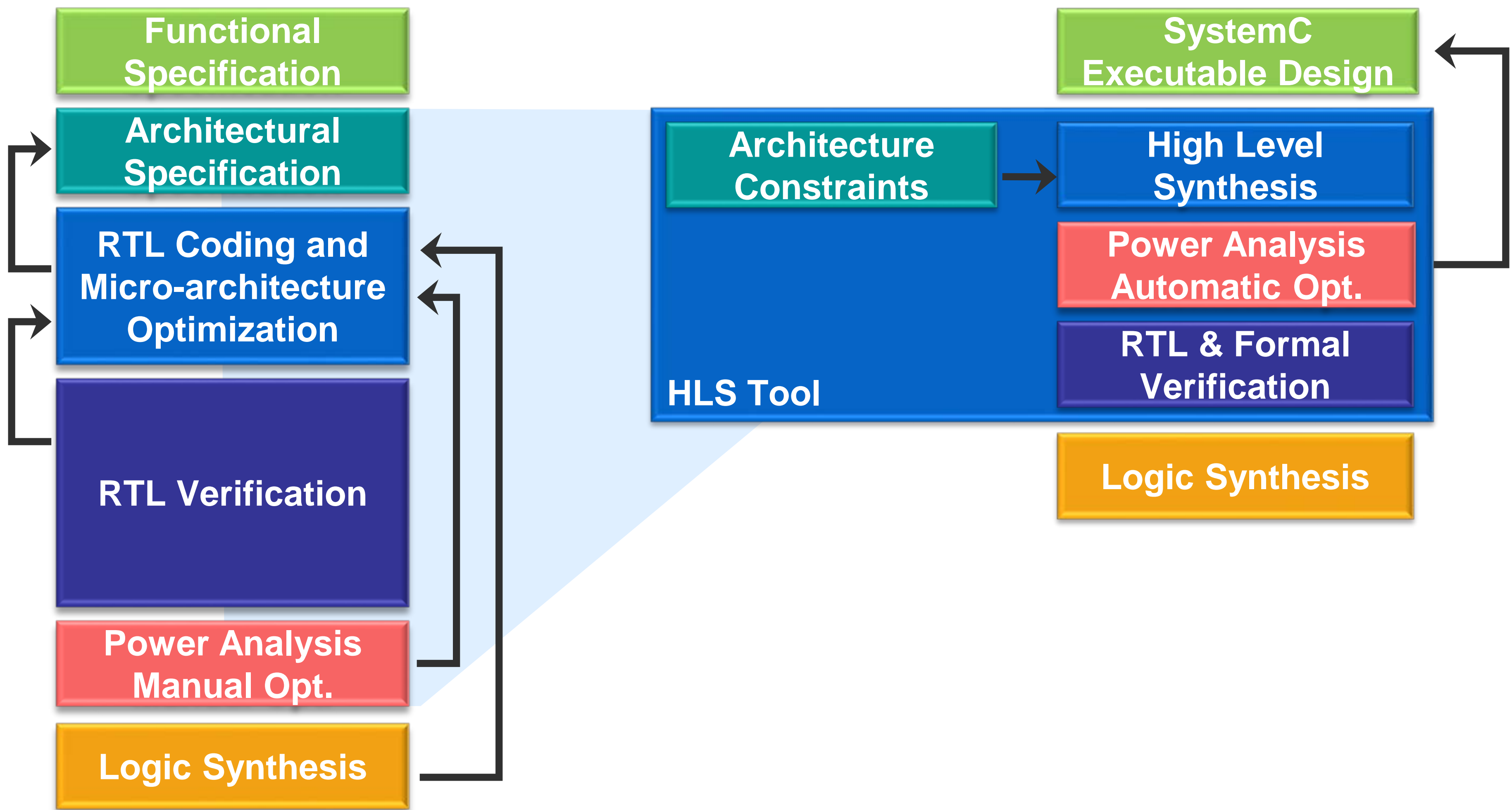
1. Mapping from abstract transactions to pin-accurate protocols



2. Optimizing for performance & area in the target technology

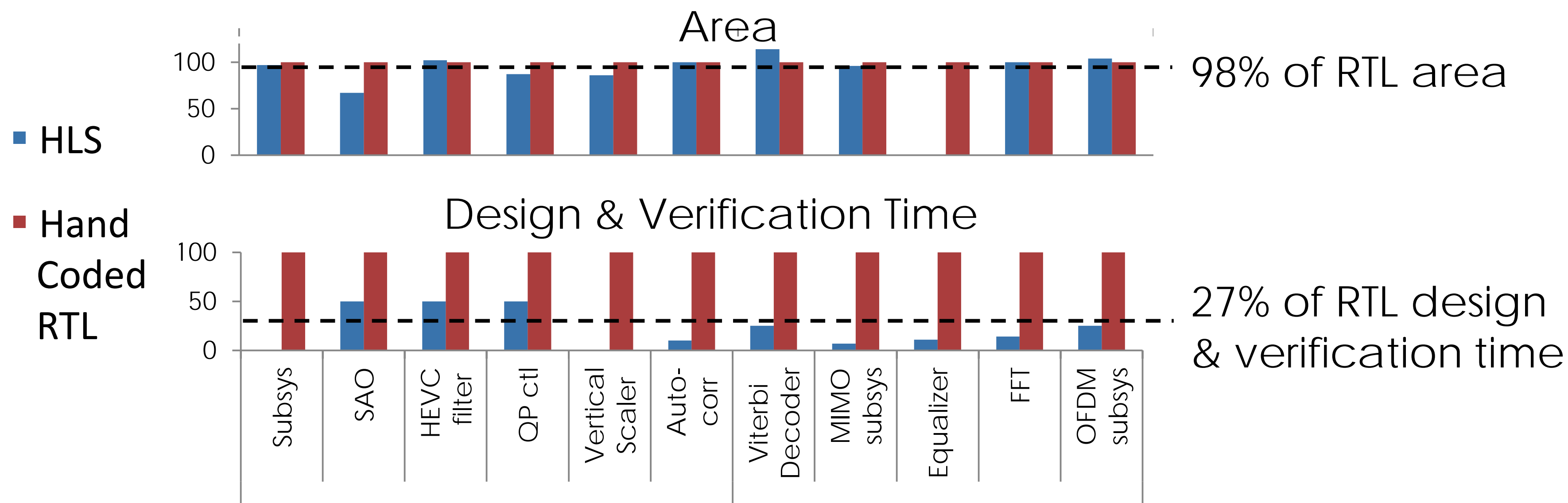


Traditional Design Flow vs. HLS Flow



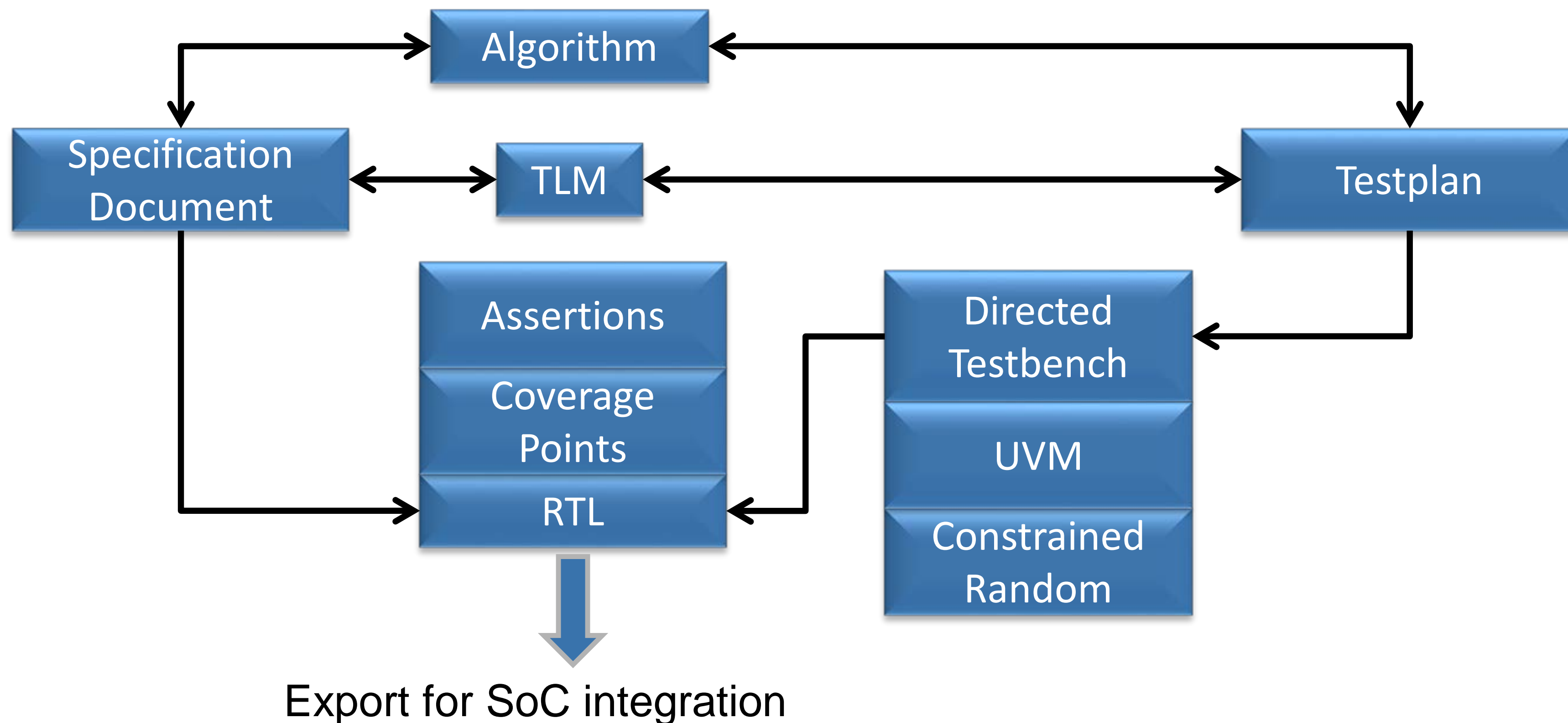
HLS Delivers QofR & Crushes RTL Design Time

- 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



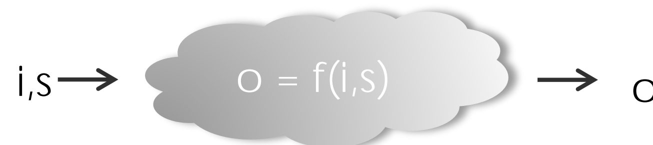
HLS-enabled Verification

Advances in Verification Technology

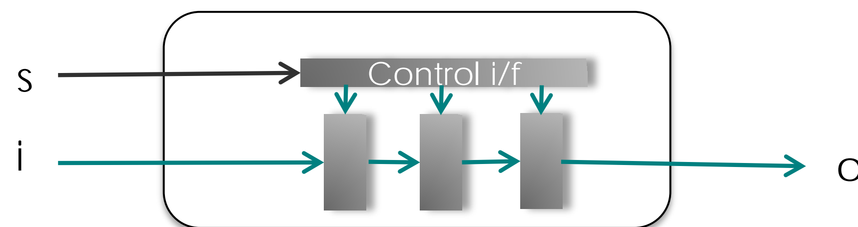


Review of Hardware Abstractions

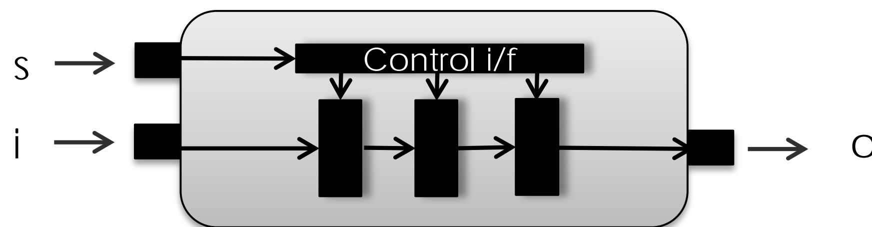
- Algorithmic Model
  - No timing or architecture



- Transaction-Level Model
  - Partitioned for hardware architecture



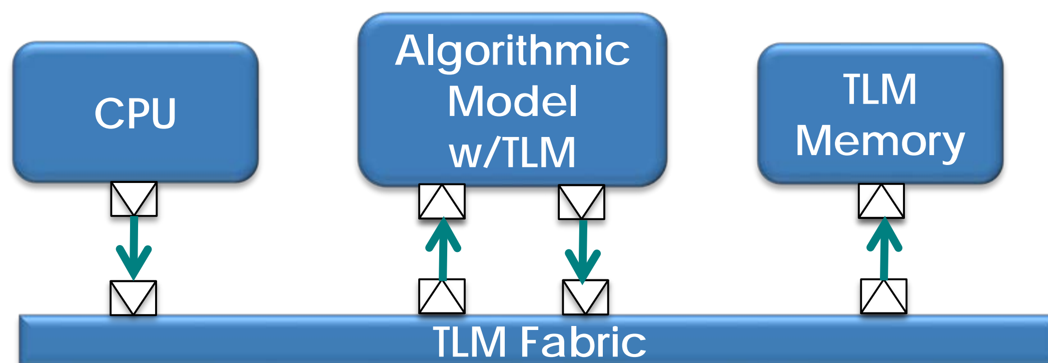
- RTL Implementation
  - Synthesizable to gates



Verification in ESL Platform

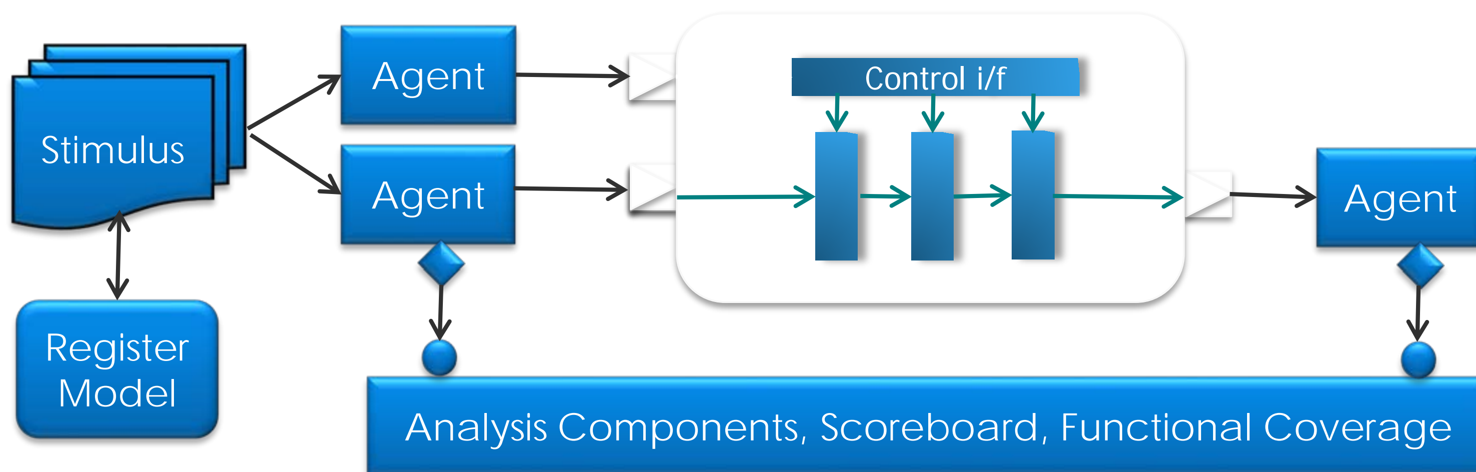
- Algorithmic Model can be used as a reference model
  - Can be embedded in SV/UVM environment
- Enables early software development
  - Software-driven testing
- <10 minutes simulation vs. 1 month simulation in RTL

ESL Platform



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

```
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;
}
```

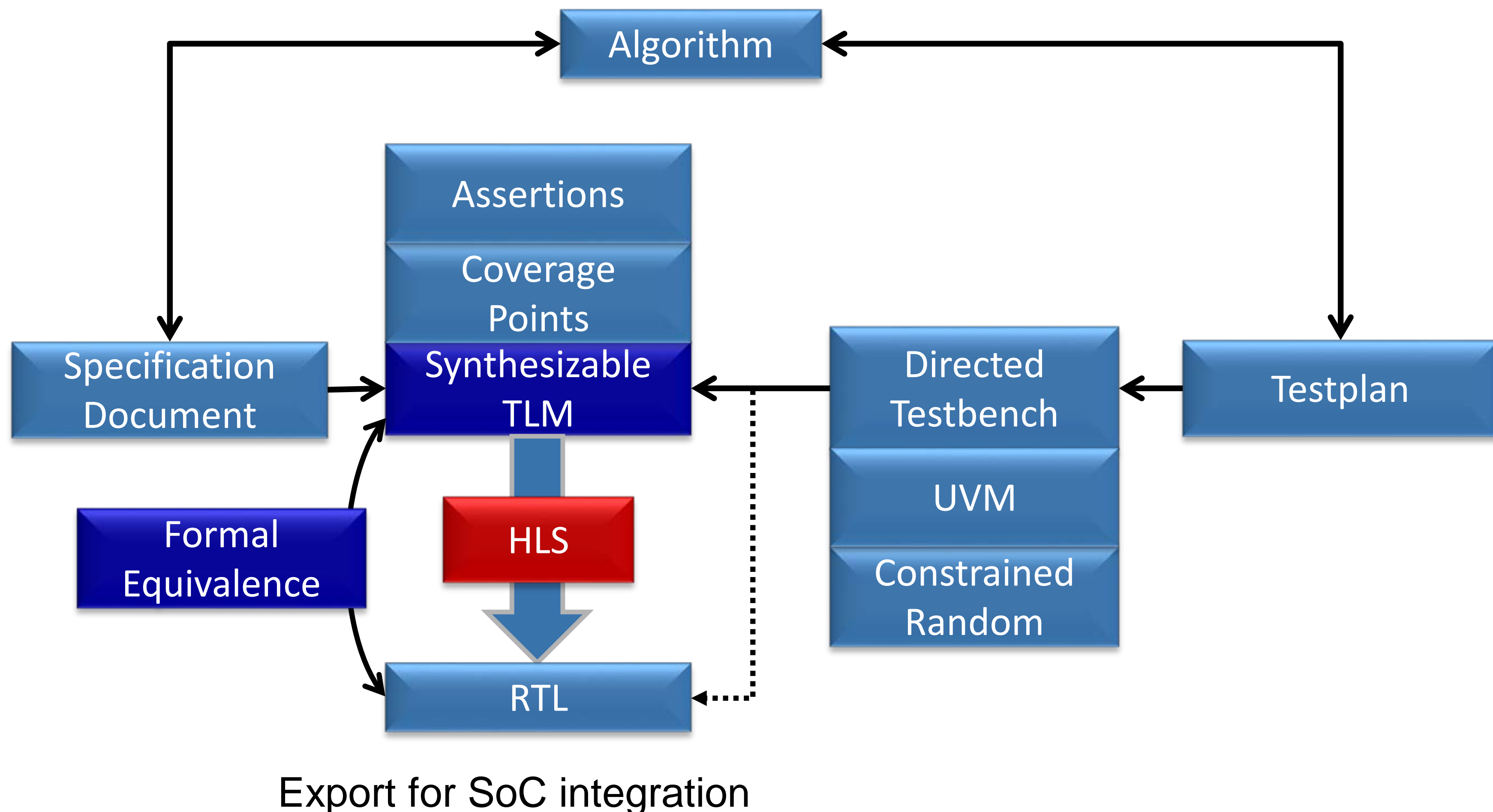
RTL Coverage

- 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

```
2103      input [0:0] sel;
2104      reg [9:0] result;
2105    begin
2106      case (sel)
2107        1'b0 : begin
2108          result = input_1;
2109        end
2110        1'b1 : begin
2111          result = input_0;
2112        end
2113        default : begin
2114          result = input_0;
2115        end
2116      endcase
2117      MUX_v_10_2_2 = result;
2118    end
2119    endfunction
2120
2121
2122    function [8:0] MUX_v_9_2_2;
2123      input [8:0] input_1;
2124      input [8:0] input_0;
2125      input [0:0] sel;
2126      reg [8:0] result;
2127    begin
2128      case (sel)
2129        1'b0 : begin
2130          result = input_1;
2131        end
2132        1'b1 : begin
2133          result = input_0;
2134        end
2135        default : begin
2136          result = input_0;
2137        end
2138      endcase
```

Instance	Design unit	Design un	Total cove	Stmt co	Stmts	Stmts	Stmt %	Stmt graph
scverify_top	scverify_top	ScModule						
clk2	sc_core::sc...	ScHierC...						
clk	sc_core::sc...	ScHierC...						
rtl	edge_detec...	Module	66.6%	742	621	121	83.7%	
k_cns_pipe	mgc_pipe_r...	Module	54.2%	391	277	114	70.8%	
linebuffer_inst	linebuffer(f...	Module	91.0%	264	260	4	98.5%	
sobel_inst	sobel(fast)	Module	74.3%	87	84	3	96.6%	
sobel_core_inst...	sobel_core(...	Module	74.3%	87	84	3	96.6%	
MUX_v_10_...	sobel_core(...	Function						

HLS Verification



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!