2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

A Video Entropy Coder Design and Verification Using HLS and HLV

Yuan-teng Chang, Siemens EDA

SIEMENS

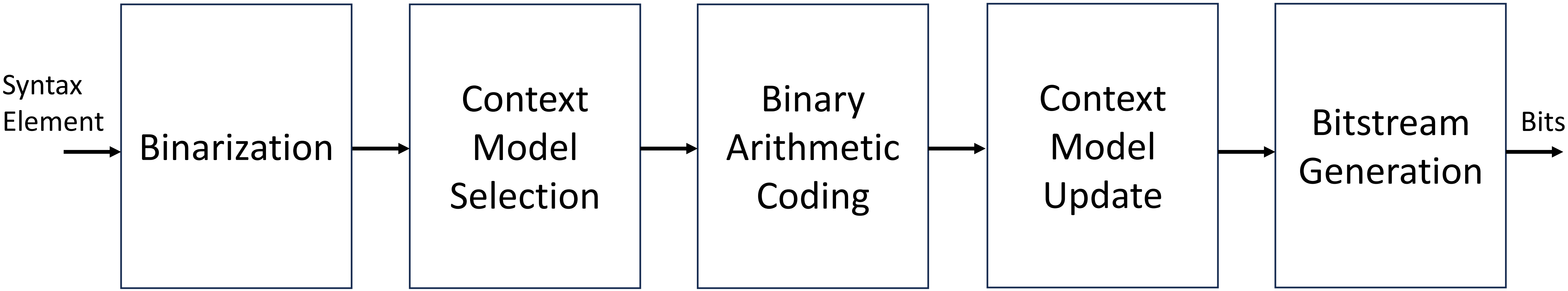
accellera SYSTEMS INITIATIVE

Introduction

- Modern video codecs deliver substantial compression efficiency, but this comes at the expense of increased algorithm complexity
  - ~10X bitrate saving but ~100X complexity from MPEG2 to H.266/VVC
- High-level synthesis (HLS) provides an opportunity of quickly exploring a variety of hardware architectures for such complicated algorithms, enabling analysis in terms of PPA
- Furthermore, high-level verification (HLV) flow can thoroughly verify HLS C++ model, allowing the generated RTL to be used directly
- In this paper, we propose a hardware architecture for AVS3 entropy coder, implement it using HLS C++/SystemC, and verify it through a HLV flow

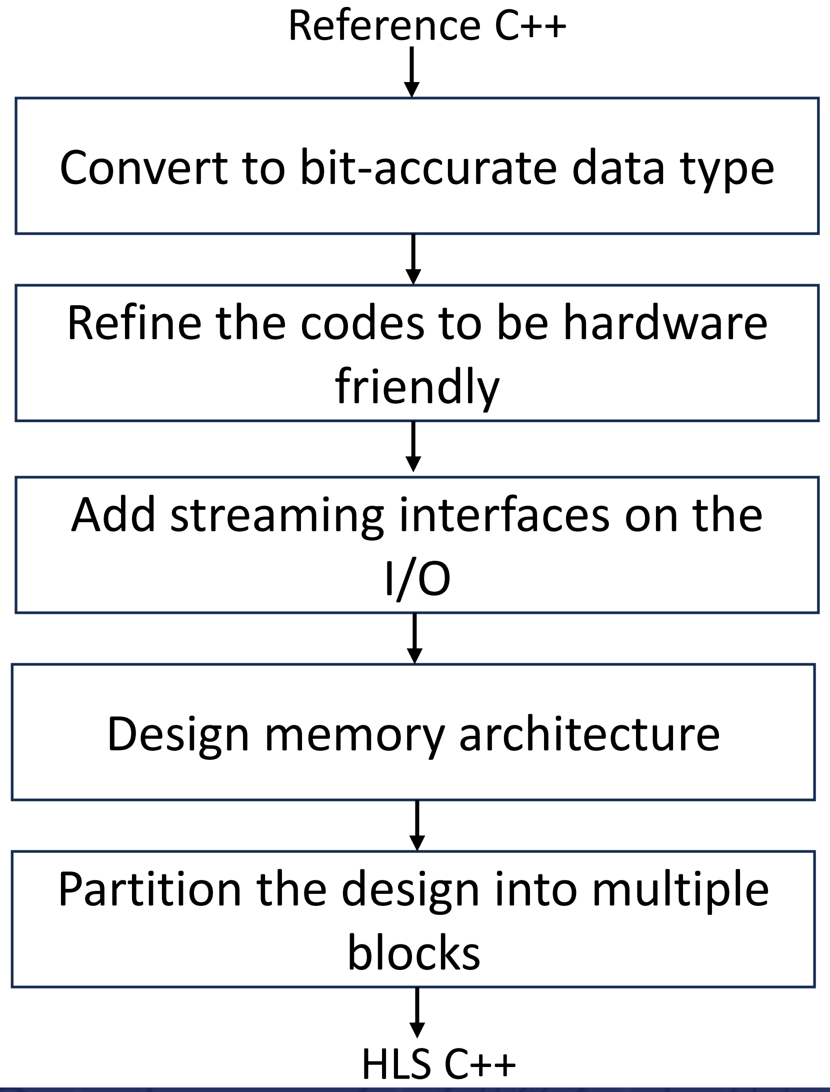
An Entropy Coder Algorithm

- CABAC-based entropy coding is employed in video coding standards H.264/AVC, H.265/HEVC, H.266/VVC, and AVS2/AVS3



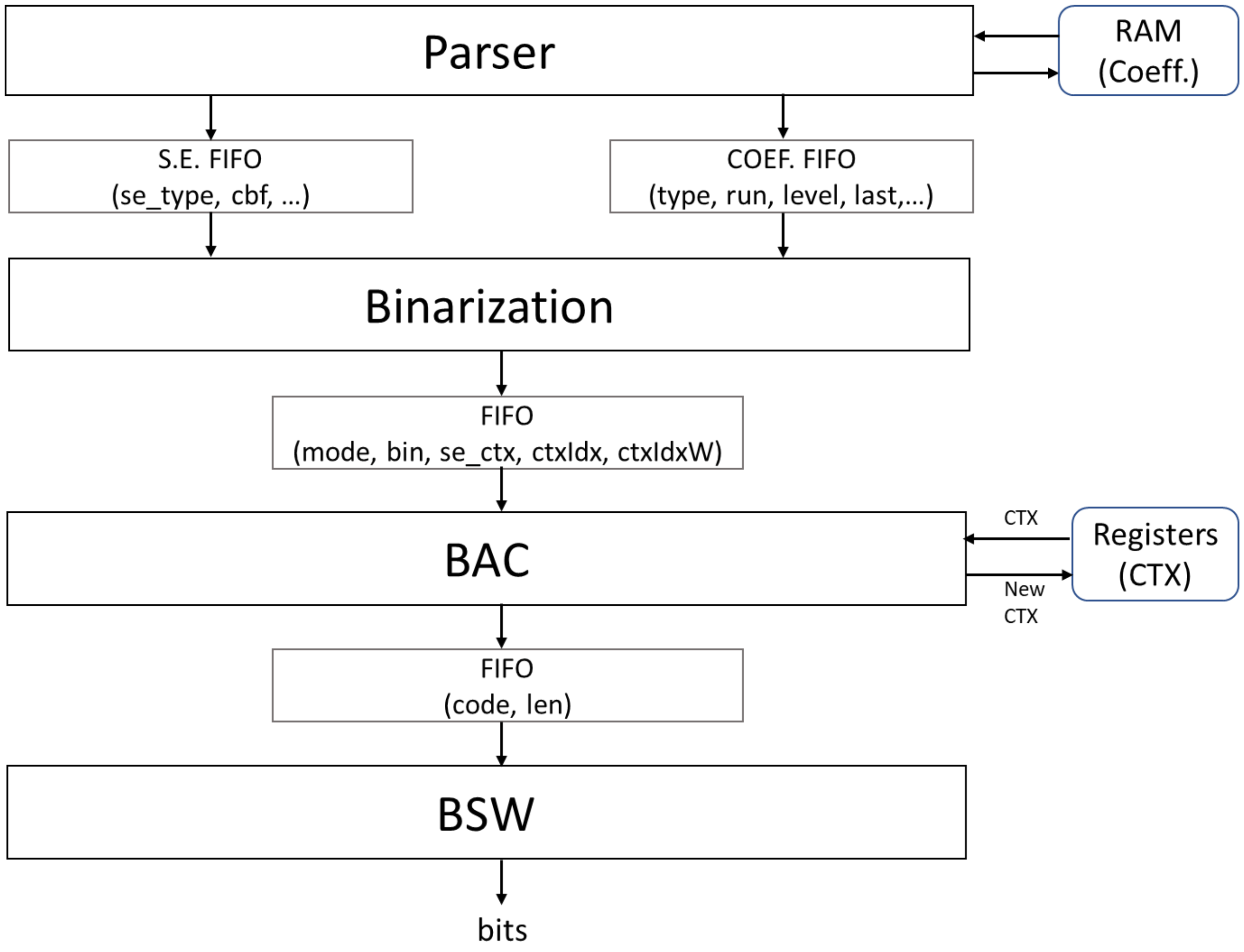
Converting Reference C++ into HLS C++

- This design starts from the AEC encode functions in HPM reference software
- Many functions have been refined for hardware implementation
  - EGk code, Scan order table, BAC, …
- Add memory architecture for coefficients reading and context model management
- Partition the design for function-level pipelining



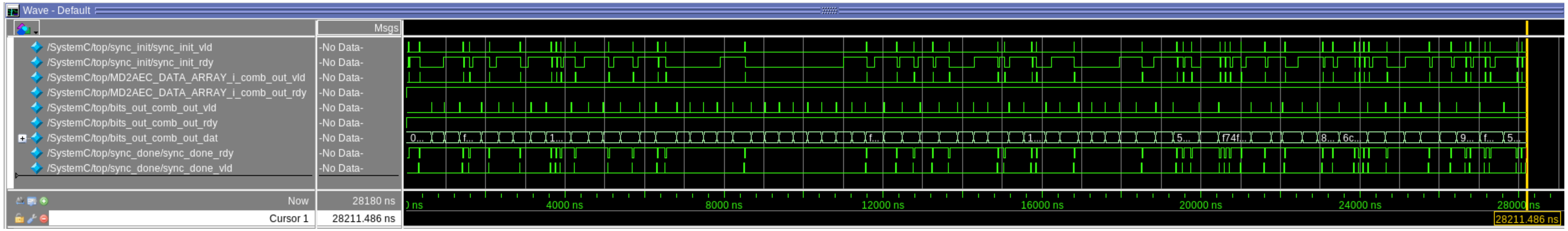
Hardware Architecture

- Fully pipelined with 1bin/cycle throughput
- Fundamental Architecture
  - Design with multiple blocks
  - Streaming interface with handshake protocol
  - Coefficients are read from an external SRAM
  - Context models are stored in local registers

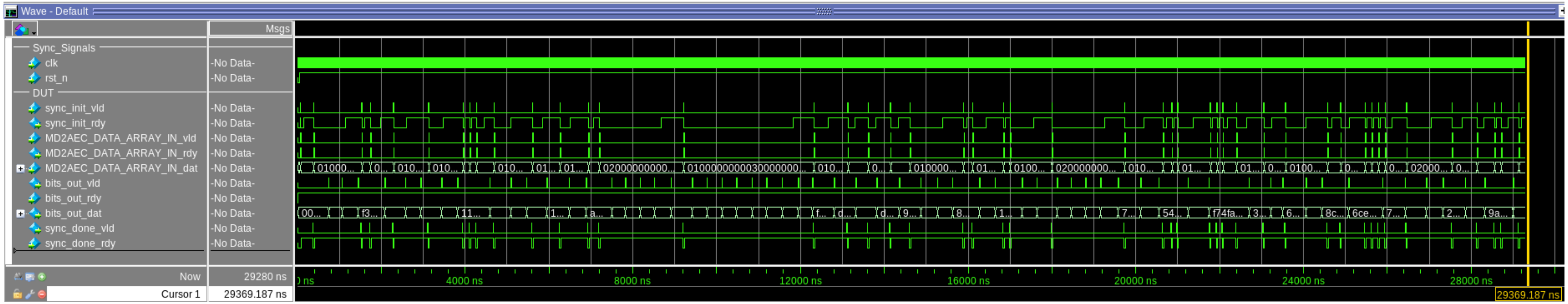


Early Performance Evaluation

- Pre-HLS (SystemC) Performance: 2821 cycles

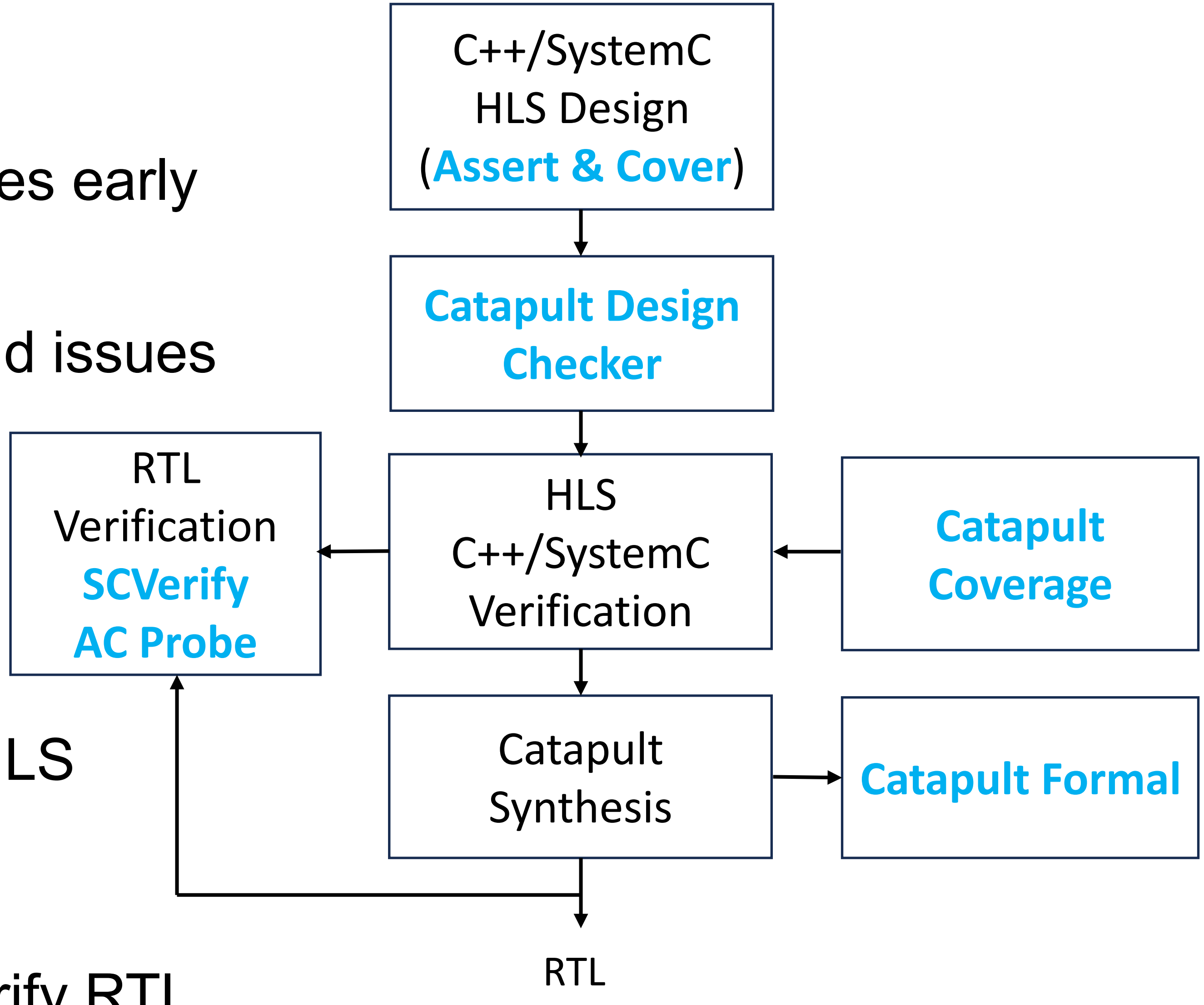


- Post-HLS (RTL) Performance: 2937 cycles



HLS & HLV Flow

- Assert & Cover
  - Deploy properties to catch issues early
- Catapult Design Checker
  - Static and formal analysis to find issues early
- SCVerify
  - Automatic RTL verification
- Catapult Coverage
  - Achieve coverage closure on HLS design source
- Catapult Formal
  - A suites of CFormal apps to verify RTL



Assert & Cover

- Assertions are a great way to catch bugs as early as possible in both HLS and RTL
- Catapult supports immediate assertions and cover properties in HLS C++ and SystemC
  - assert()
  - cover()
- All of these are automatically propagated from HLS C++ to RTL

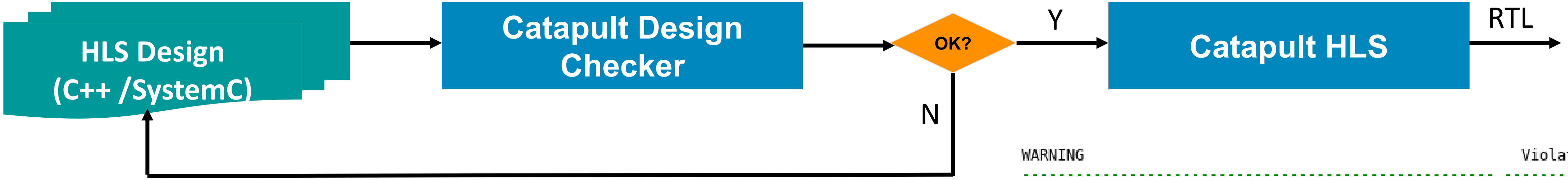
```
#include <ac_assert.h>
…
assert((bac_mode < 3));
cover((bac_mode==REGULAR));
cover((bac_mode==BYPASS));
cover((bac_mode==SPECIAL_REG));
…
```

RTL → SVA

```
//cover(bac_mode==REGULAR)
property aec_enc_bac_eq0_CP_p;
 @(posedge clk) disable iff (rst || !arst_n)
  (p_bac_mode_0_prb);
end property
aec_enc_bac_eq0_CP : cover property (aec_enc_bac_eq0_CP_p)
```

Catapult Design Checker

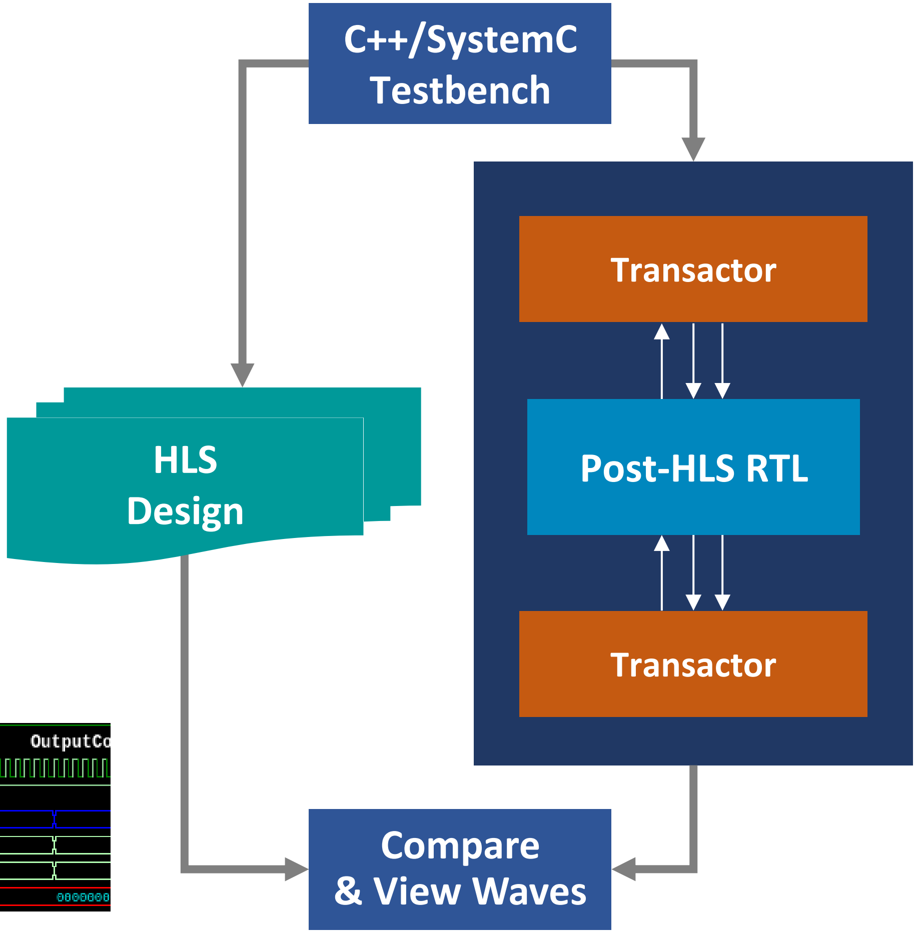
- Static Lint check before synthesis or simulation
  - Coding errors
  - QofR checks
  - Mismatches between C++ and RTL simulation

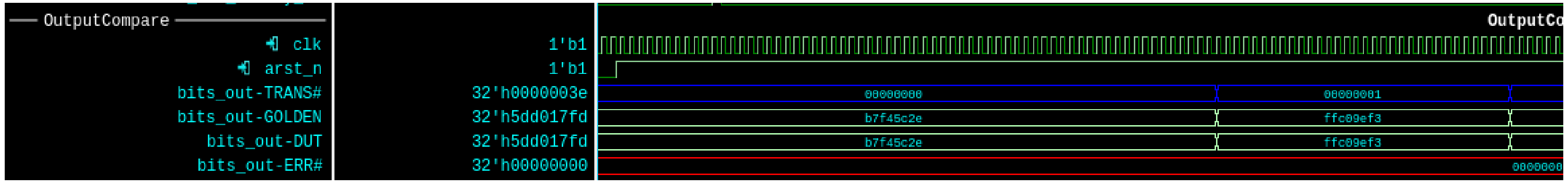



```
WARNING                                                                    Violated
-------------------------------------------------------------------------- ----------
ACC - Accumulator of native C type                                                0
ACS - Accumulator of saturated type                                               0
CBU - Conditional break in Unrolled Loop                                          1
CGR - Conditional Guard in Rolled Loop                                            0
DIU - Dynamic Index in Unrolled Loop                                              0
FVI - For Loop with Variable Iterations                                           6
OSA - Optimal Size Accumulator                                                    0
RIU - Rolled loop Inside Unrolled loop                                            0
SAT - Sub-optimal Adder Tree                                                      0
SUD - Suboptimal Use of Divide and Modulus Operator                               0
```

SCVerify Flow

- Dynamically validate RTL functionality
- Automated RTL smoke test for designers
  - Verify C++ against the post-HLS with cosimulation
  - Original C++ testbench reused to simulate the RTL
  - Transactors convert C++ function calls to/from pin-level

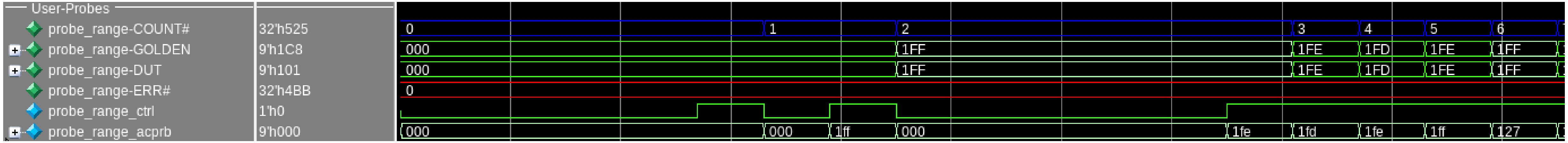




AC Probe

- Specifying Post-HLS probe points
- Very useful to track Pre-HLS (C++) values during RTL simulation

```
#include <ac_probe.h>
…
ac::probe((“probe_range”, range));
…
```



Catapult Coverage

- Bring RTL coverage metrics to the HLS world
  - Run 30x-500x faster than RTL simulation
  - Code coverage including Statement, Branch, FEC
  - Functional coverage including covergroups, coverpoints, bins and crosses
- HLS-aware code coverage vs software coverage tools
  - Function instantiation
  - Array indexing
  - Loop unrolling

Coverage Summary By Instance (91.1%)

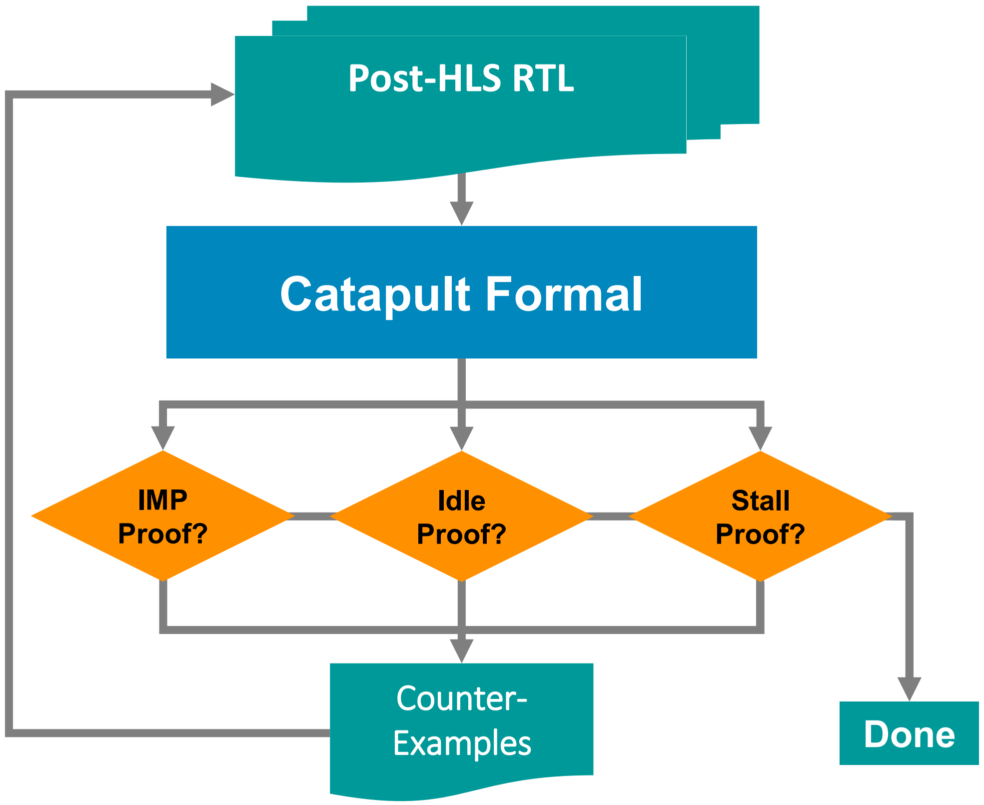
Instance	Branches	Expressions	Statements
Total	81.14%	100%	92.18%
accelerator_inst	-	-	100%
bus_if_inst	84%	-	91.97%
conv2d_inst	80.66%	100%	91.17%

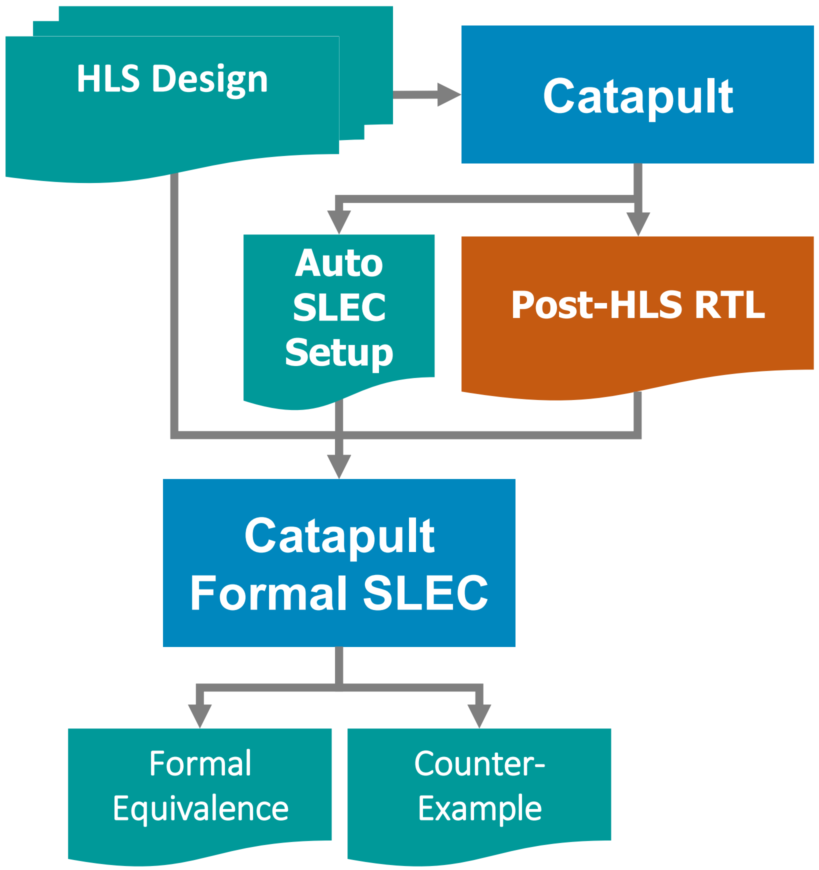
Covergroups Coverage (62.5%)

Covergroups	Bins	Hits	Goal	Coverage
main_23::MyCCoverGroup...	8	5	100	62.5%
MyCCoverGroup_1_inst	8	5	0	62.5%

Catapult Formal Apps & SLEC

- Formal equivalence between HLS C++ & Constraints vs RTL





Experimental Results

- This design has been synthesized using Catapult Ultra and Design Compiler and runs at 333MHz on TSMC 22nm technology
- The synthesis results show that area reported by Catapult is 28,236um2 slightly higher than DC area 24,383um2
- Power estimation is done by PowerPro under the hood of Catapult Ultra
- Achieve real-time processing of 8K video at 60fps with a target bitrate 80 to120 Mbps

Latency	Throughput	C2RTL Runtime	Power	Catapult Syn. Area	DC Syn. Area
19 cycles	1bin/cycle	406s	1,566um	28,236	24,383

Conclusion

- In this paper, we present a hardware architecture of an entropy coder, implemented in C++/SystemC and synthesized and verified using Catapult HLS and HLV flows

- HLS / HLV flows bring the benefits:
  - ~ 50% reduction in design and verification time compared with traditional hand-written RTL
  - Faster design space exploration
  - Efficient migration between FPGA prototypes and ASIC technologies
  - Easy to maintain C++ code and reuse