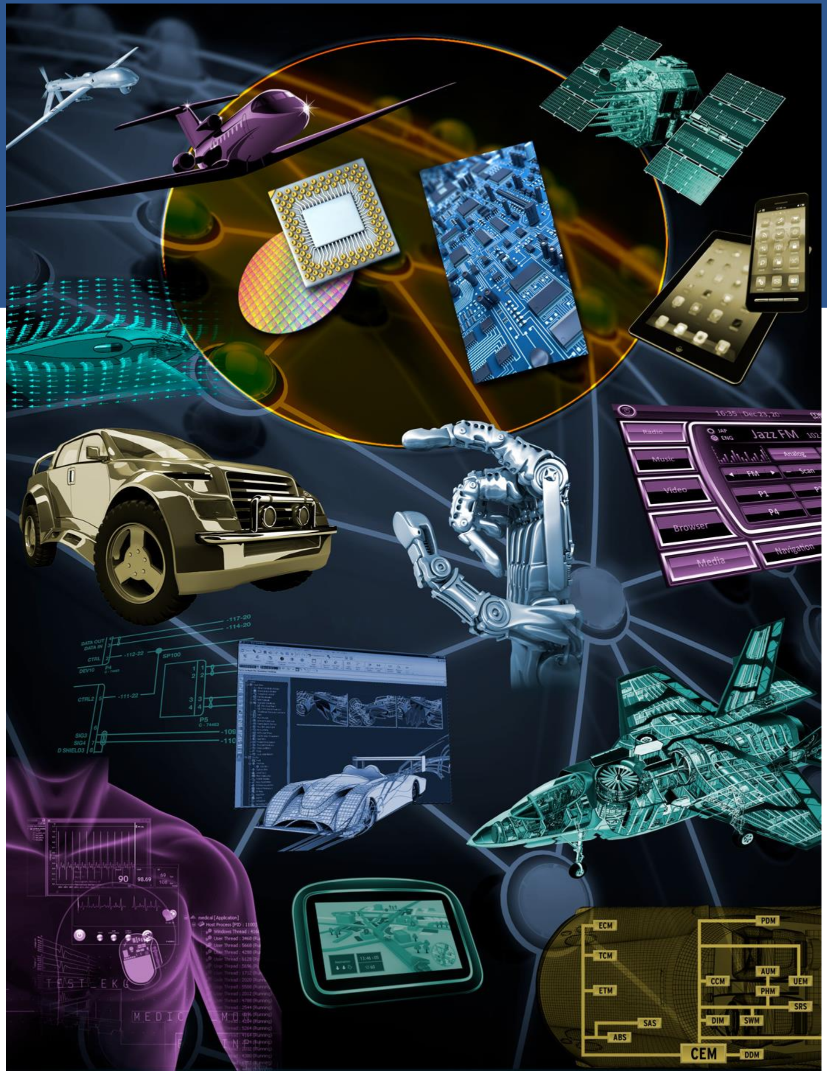



# High-level Synthesis: status and future trends

Andres Takach
Chief Scientist

March 2019

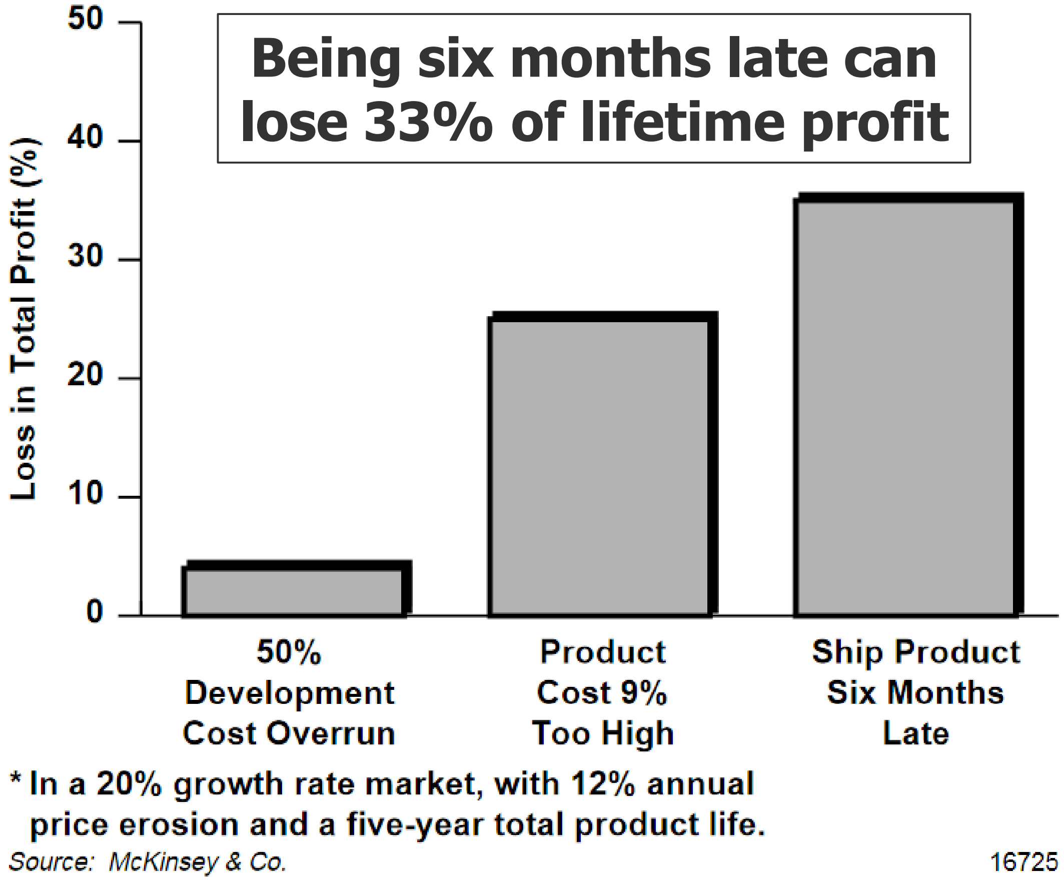
Mentor®
A Siemens Business

# Agenda

- Why HLS?
  - Impact of HLS on industry
- HLS for Computer Vision and Machine Learning
- Verification
- Low Power
- C++ and SystemC and Standards
- Open Source IP

# RTL Design is Stalling

- Algorithmic Complexity
  - Growing faster than the ability of RTL designers to code and verify
- RTL Verification Costs Increasing
  - RTL regressions involve server farms, electricity cost, licenses and time
  - Emulation delivers speed at a price
- RTL Reuse Penalty Costs
  - Moving to a different process and frequency can be a challenge in RTL
  - Reusing RTL can be inefficient for QoR
- Time To Market Kills Total Profit



# Impact of HLS on industry

- Qualcomm Designs 1.5-2x Faster
  - Used for new IP
  - Multi-standard codecs, Ultra HD resolution, Aggregated Wireless
- Google Designs Reusable VP9 IP In Half the Time
  - Technology Independent Video Decode IP
    - Tapeouts at 65, 28, 20
  - Verified 2x faster
- NVIDIA Achieves Cost Reduction of ~80% for Functional Verification with C++ HLS
  - C++ functional verification runtime ~500x less resources than RTL
  - Fast verification makes rapid product changes possible

# Impact of HLS on industry

- ST Imaging HLS Success for ISP (Automotive)
  - To date created 50+ Image Processing IPs using HLS Imaging Template
- Bosch delivered new designs ahead of schedule in 7 months with evolving specifications; improved quality over RTL
- SeeCubic Get To Market Faster & Slash Bug Rate
  - Ultra-D is 3D vision **without glasses!**
  - 95% of design done in C++ had 20% of total bugs
  - 5% done in hand RTL had 80% of total bugs
- FotoNation: Next-Gen Mobile Face Recognition With HLS
  - "3 weeks from Caffe to FPGA" "4x faster then hand coding"
  - "Verification is Easier - Bit exact between HW and C++ is native"
  - Instant retargeting to optimal ASIC RTL

# New Markets Bring New Competitive Pressures

- Key markets have significant requirements for new designs




**Reduced Time To Market with good QoR**

**Require FPGA Prototype & SoC/ASIC**

**Handle late changing specifications**

**Reduce Verification and Debug Cost/Time**

**Computer Vision & Neural Computing**

**High Bandwidth & Cellular Communication**

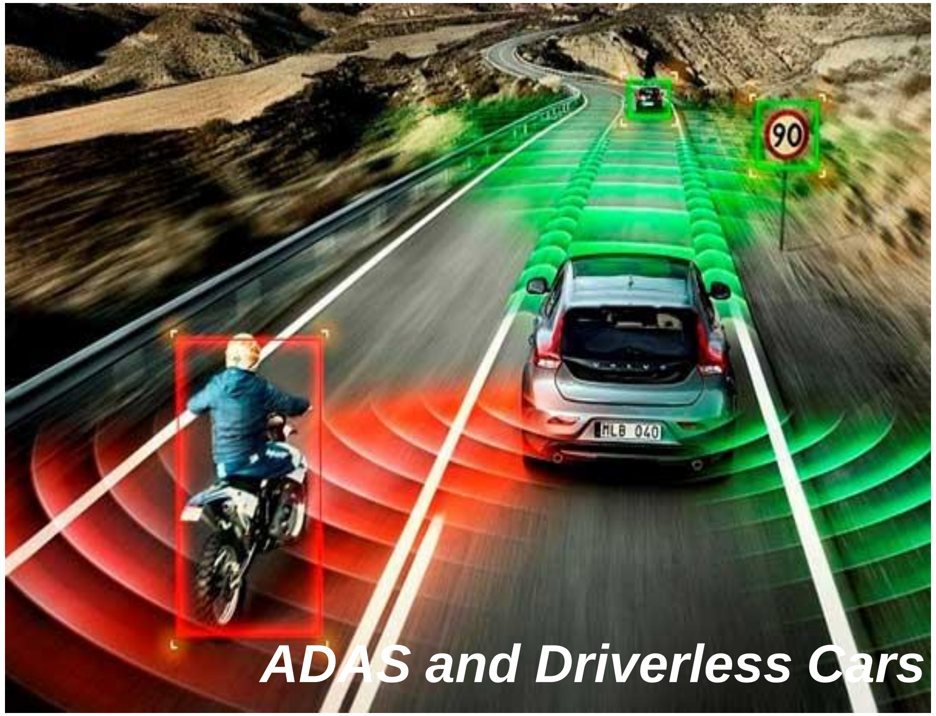
**Image Processing, Video & Compression**

**Complex Algorithms**

# Computer Vision Application Challenges

*Automotive and other "real-time" application especially challenging*

- Continually changing algorithms and sensors
- Computationally very expensive
  — Billions of operations/second
- High responsiveness required
  — High-bandwidth and low-latency
  — Real-time processing of data required
- Autonomous drive - Solution required to be < 100w
- Each provider wants to add their "secret sauce"



# What are the Choices for Hardware Platform?

*There is no clear winner today as this market is emerging*

Flexibility

- CPU
  - Not fast or efficient enough
- DSP
  - Good at image processing but not enough performance for Deep AI
- GPU
  - Good at training but too power hungry for long term inferencing solution
- FPGA
  - Low-power, mostly meets performance/latency, RTL flow not practical, not the lowest power, eventually cost for volume a problem
- ASIC
  - Lowest power, meets performance/latency, high NRE and no field modifications/upgrades, Algorithms still changing, RTL flow not practical, lowest volume cost
- Dedicated AI processors or accelerators in IP and ASIC
  - Popping up like weeds – high performance, locks customer in, many server target
- Some combination of the above

Power

# Numerous Possible Hardware/Memory NN Architectures for Inference Engines

- Machine learning architectures are still evolving
  - How to know which one is right for the application
  - Not enough time to do them all in RTL
- On-chip memory, memory bandwidth, power performance and area are all important

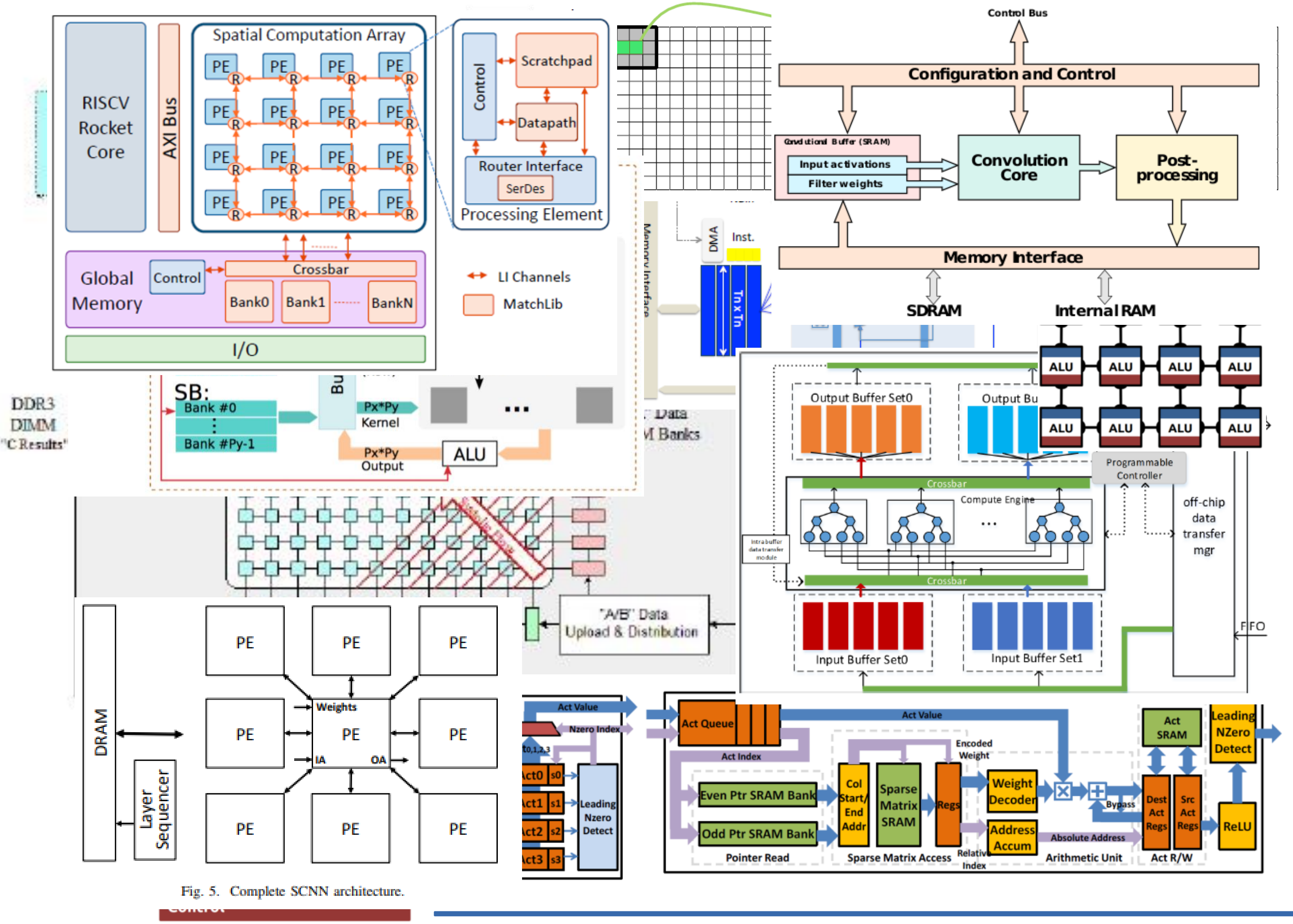



Fig. 5. Complete SCNN architecture.

# NVIDIA Research New Methodology with Catapult

*Machine Learning Accelerator SoC using an Object-Oriented HLS flow*

- NVIDIA Research with DARPA - New methodology for 10x faster chip design
- HLS to target 80% of future NVIDIA chips
  - Open-Source HLS IP:
    - https://github.com/NVlabs/matchlib
- 2 Tapouts - 20M+ gate Machine Learning accelerator SoC
- Foundation for NVDLA HW
  - NVIDIA Deep Learning Accelerators
- 2 DAC Papers; 2016,2018 available now
  - *Digital VLSI Flow for High-Productivity SoC Design*
  - *Hardware Accelerator for Mobile Computer Vision Applications*
- On-Demand Webinar
  - *Design and Verification of a Machine Learning Accelerator SoC Using an Object-Oriented HLS-Based Design Flow*

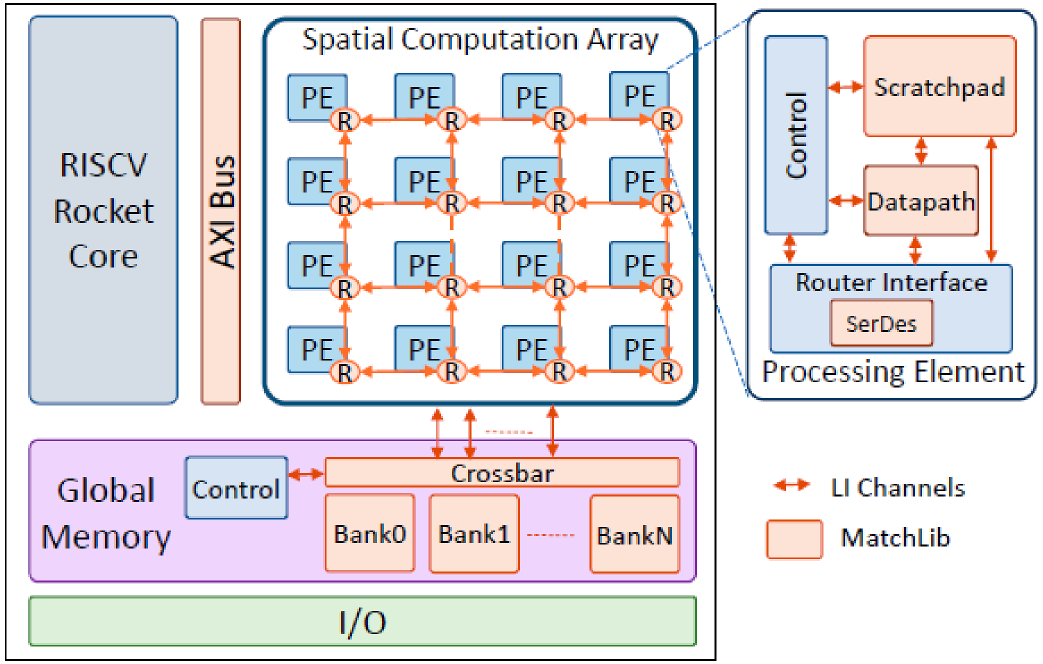
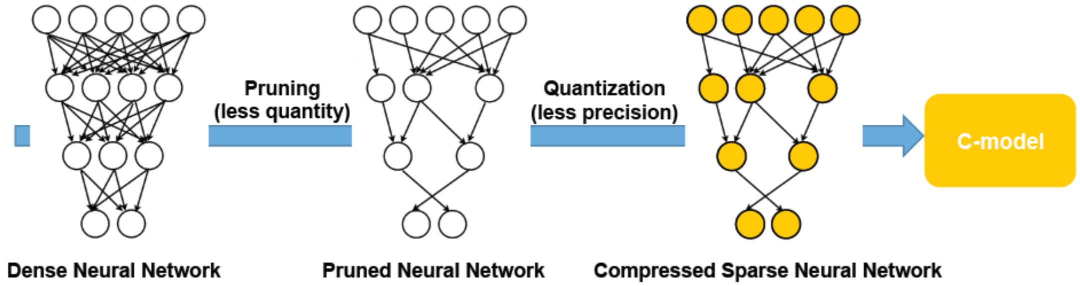


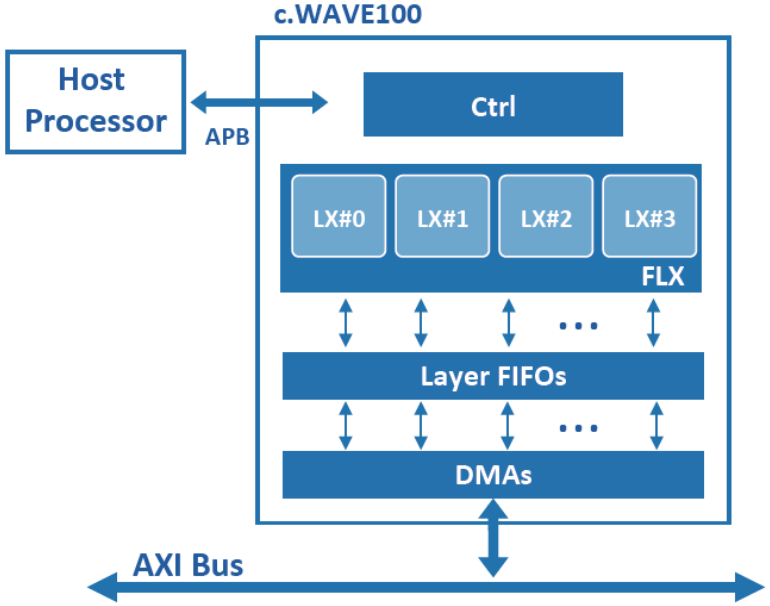

Figure 5: Prototype SoC

# Chips&Media Success for Deep Learning Object Detection IP

- Successfully delivered inference-targeted Deep Learning IP with move to HLS
  - RTL designers now plan to use HLS on all future new computer vision/deep learning IP
  - HLS is key to finding power optimized specific DNN



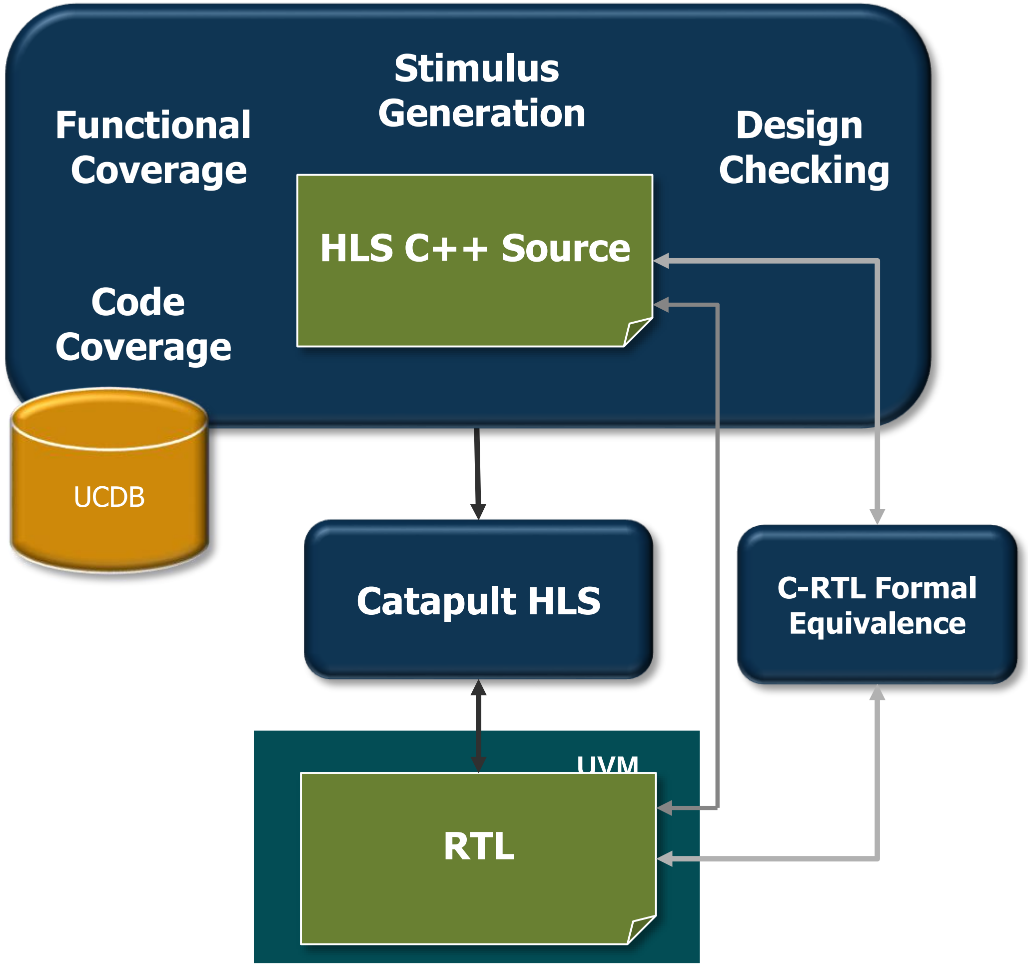

- Cut the block/IP design and verification time in half
  - New DNN architecture
  - Delivered critical FPGA customer demonstrator early
- HLS helped find optimal power/performance architecture that RTL "would not have had time"
- New detailed white paper
  [Design and Verification of Deep Learning Object Detection IP]



# VERIFICATION

# Complete High-Level Verification Solution

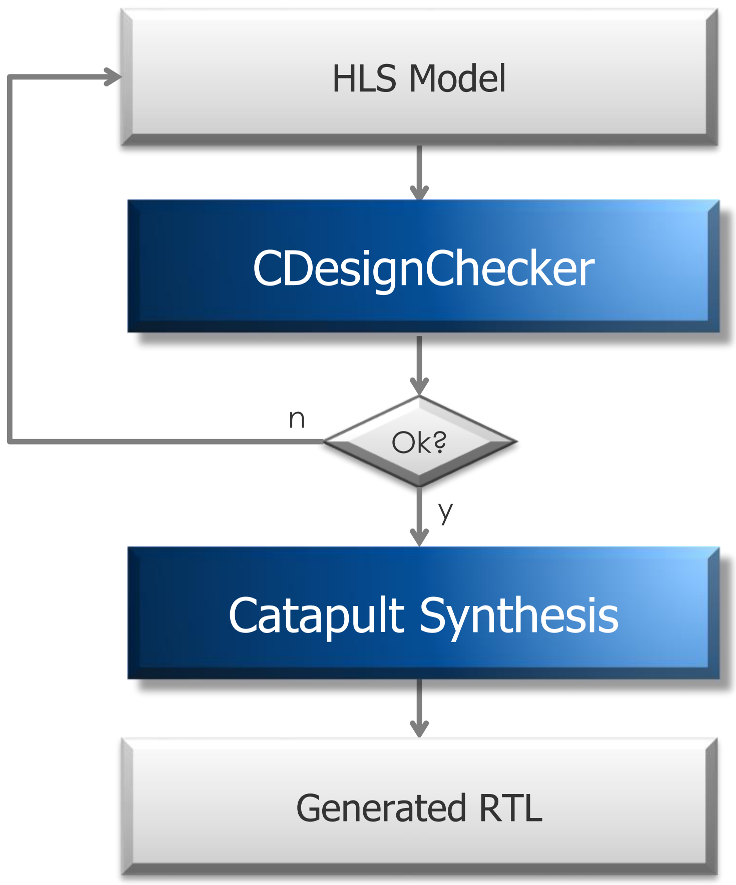
*Delivering on the Vision of C++ verification sign-off*




- Front-end C++/SystemC verification
  - Design Checker
  - Coverage
- Stimulus Generation
- Automatic C-to-RTL verification
  - SCVerify RTL Sanity Check
  - UVM Auto Generation for RTL
- SLEC HLS – C=>RTL=>Gate Formal Equivalence Checking

# Design Checking

- Quickly and easily find coding bugs and errors before synthesis or simulation
- Some C++ language behavior not well defined or ambiguous for hardware
  - Leads to mismatches between C++ and RTL simulation
  - Difficult to debug
  - Rarely caught in dynamic simulation
- Both static "lint" and formal checks
- Integrated in Catapult Prime and Ultra



# Coverage: Software Tools Don't Handle Functions for HLS

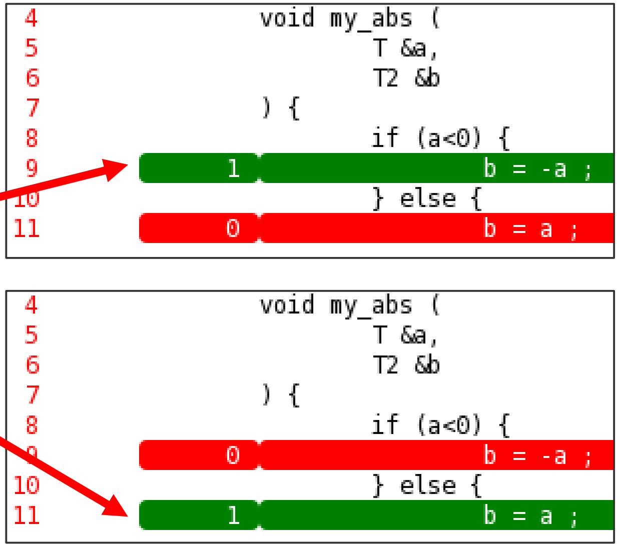
- C++ functions are "inlined" when synthesized
  - Software tools measure by function (2 calls) & line, not hardware instance
  - Catapult Coverage measures coverage metrics on every instance

```
      Branch data      Line data      Source code
                   :             : #include "abs.h"
                   :             :
                   :             : template <class T, class T2>
                   :           2 : void my_abs (
                   :             :         T &a,
                   :             :         T2 &b
                   :             : ) {
[ - + ][ + - ]     :           2 :         if (a<0) {
[ # # ][ + - ]     :           1 :                 b = -a ;
                   :             :         } else {
                   :           1 :                 b = a ;
                   :             :         }
                   :           2 : }
                   :             :
                   :           1 : void top_level (
                   :             :         data_in_a_t &a,
                   :             :         data_out_a_t &a_abs,
                   :             :         data_in_b_t &b,
                   :             :         data_out_b_t &b_abs
                   :             : {
                   :             :
                   :           1 :         my_abs(a,a_abs) ;
                   :           1 :         my_abs(b,b_abs) ;
                   :             :
[ + - ][ + - ]     :           4 : }
                   :             :
```

**Single Call Testbench**

```
data_in_a_t      a = -20 ;
data_in_b_t      b = 20 ;

top_level(a,a_abs,b,b_abs) ;
```



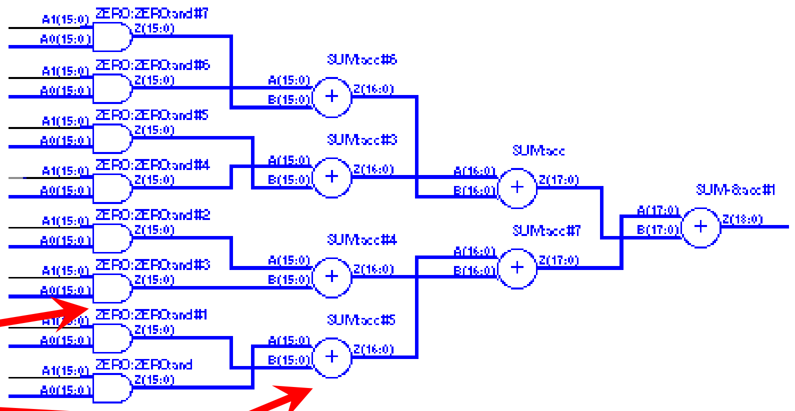

| Scope | TOTAL | Statement | Branch |
|---|---|---|---|
| TOTAL | 58.33 | 66.66 | 50.00 |
| top_level_15 | 100.00 | 100.00 | -- |
| my_abs[with T = ac_int<8, true>; T2 = ac_int<9, false>]_22_2 | 50.00 | 50.00 | 50.00 |
| my_abs[with T = ac_int<9, true>; T2 = ac_int<10, false>]_23_2 | 50.00 | 50.00 | 50.00 |

# Software Tools Don't Handle Loop Context

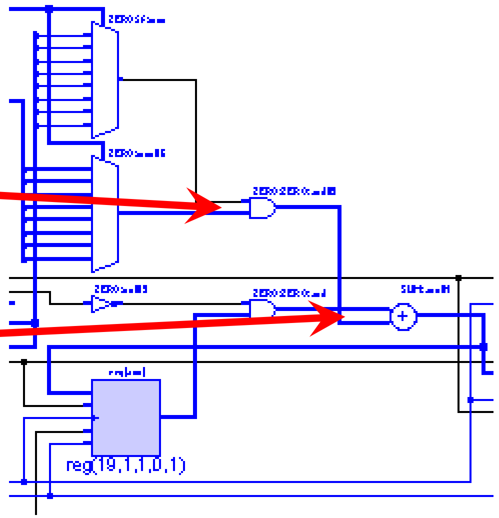
- Loop Unrolling affects hardware context

```
#include "unrolling.h"
void testcase (
  ac_int<8,false> &set_in,
  ac_int<16,false> a[8],
  ac_int<19,false> &result
) {
  ac_int<8,false> set = set_in ;
  ac_int<16,false> partial[8] ;
  ZERO:for(int i=0;i<8;i++) {
    partial[i] = a[i] ;
    if (set[i]==0) {
      partial[i] = 0 ;
    }
  }
  ac_int<19,false> acc = 0 ;
  SUM:for(int i=0;i<8;i++) {
    acc += partial[i] ;
  }
  result = acc ;
}
```

Fully Unrolled




Merged & Rolled



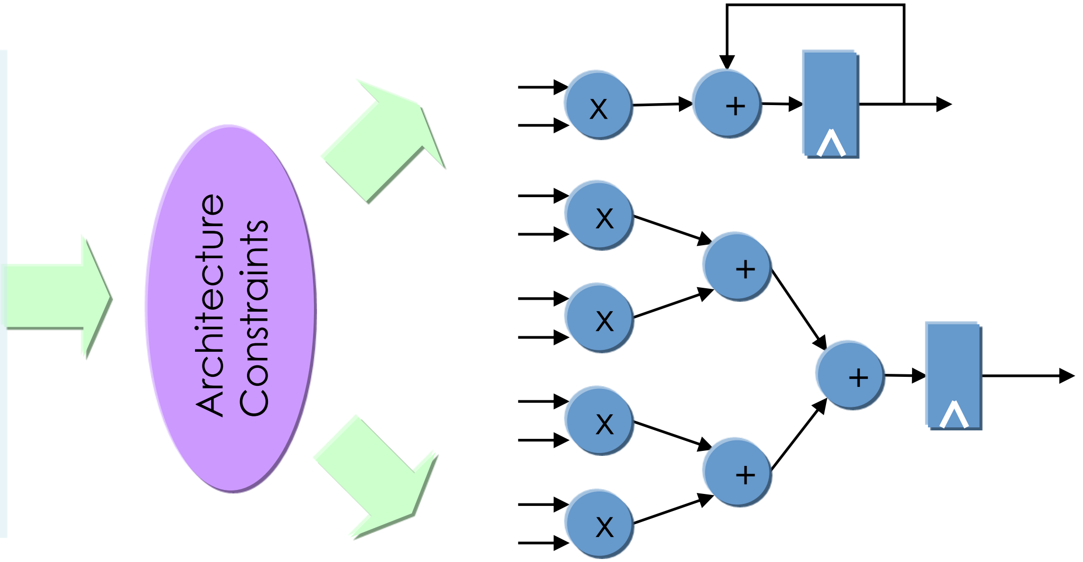
# LOW POWER

# Micro-Architecture Control

- User control over the micro-architecture implementation
  - Parallelism, Throughput, Area, Latency (loop unrolling & pipelining)
  - Memories (DPRAM/SPRAM/split/bank) vs Registers (Resource allocation)

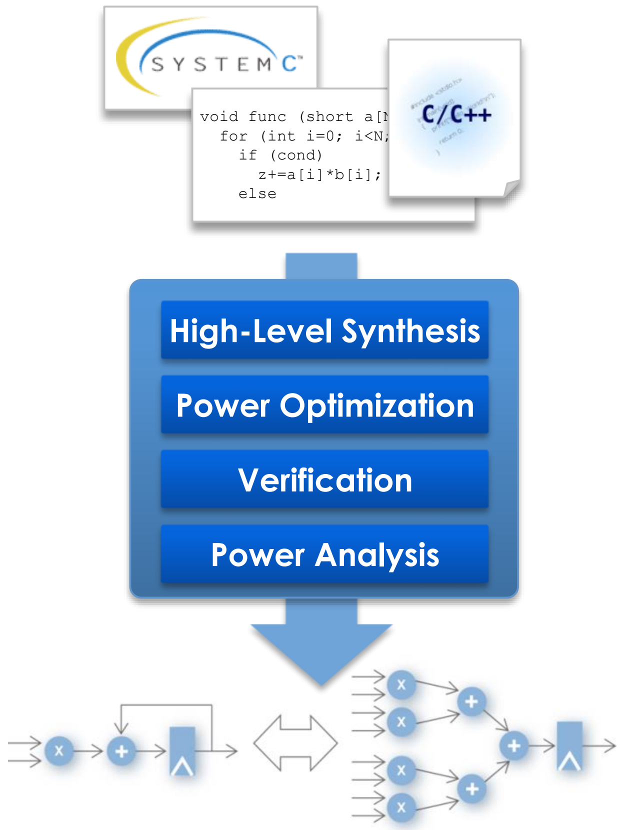
- Exploration is accomplished by applying constraints
  - Not by changing the source code

```
int mac(
  char data[N],
  char coef[N]
) {
  int accum=0;
  for (int i=0; i<N; i++)
    accum += data[i] * coef[i];
  return accum;
}
```



# Low Power HLS

- Integrated Early Power Estimation
  - Explore µArchitectures with constraints
  - Evaluate PPA alternatives for each design
  - Memory access minimization
    - Banking & interleaving




- RTL optimized for power
  - Strengthen register enables using sequential analysis to reduce power with clock gating
- Transformations to reduce loading new values into registers



Mentor A Siemens Business

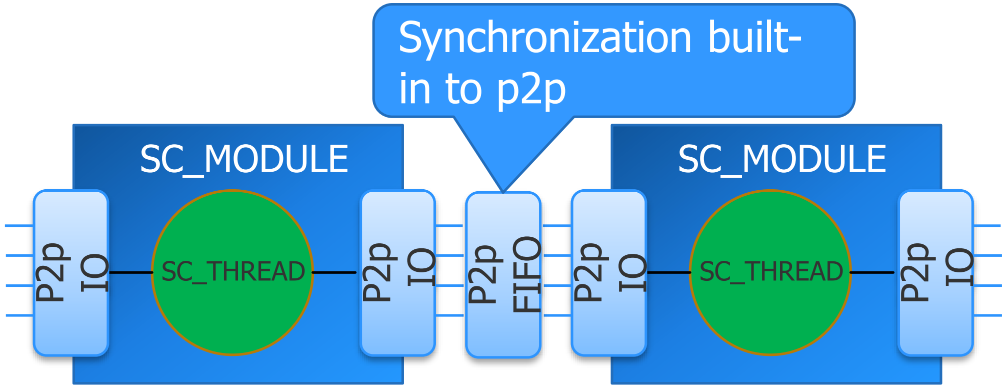
# C++ AND SYSTEMC AND STANDARDS

# Input Languages for HLS

- C++ and SystemC
- Popular SystemC abstractions
  - Untimed, Loosely-timed, Cycle-accurate
- Applicable to all use cases
  - Exploration and Implementation
  - Control Logic and Algorithms
- Flexibility to use the best language for a team, project or application
- Teams will typically select one language, but company may use both

# Multi-block Concurrency for Maximum Throughput

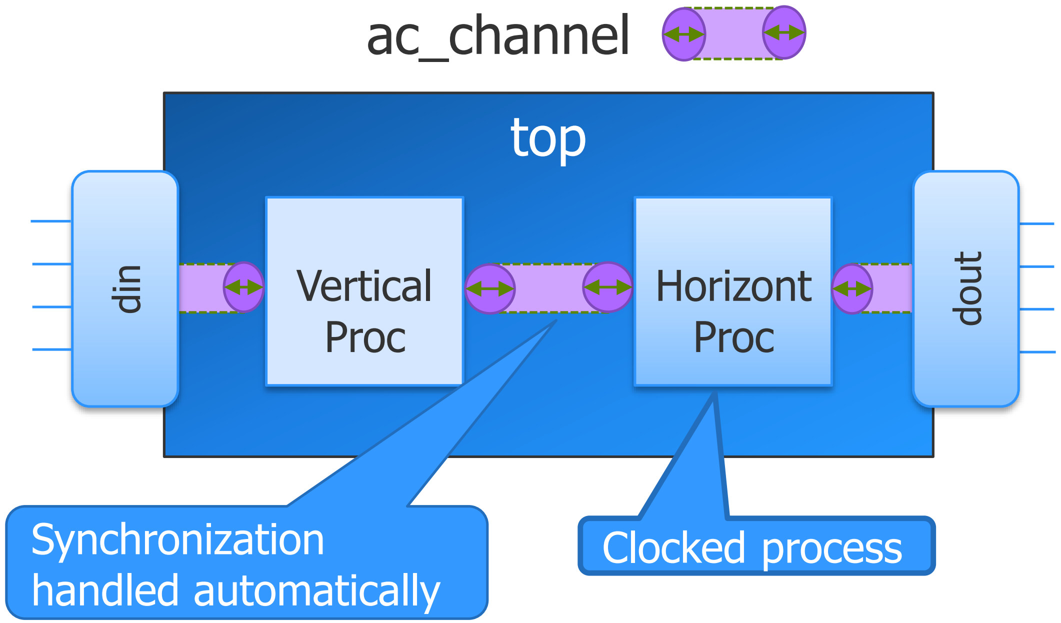
- HLS Builds Parallel Concurrent Processes from Sequential C++ Functions/Classes
  - No need for multi-threaded design and debug
- Design-blocks/processes run in parallel
  - High throughput
- SystemC concurrency is explicit using threads




```
void top(ac_channel<uint8 > &din,
         ac_channel<uint8 > &dout){

  static ac_channel<uint8> connect;

  vertical_proc(din,connect);
  horizontal_proc(connect,dout);
}
```



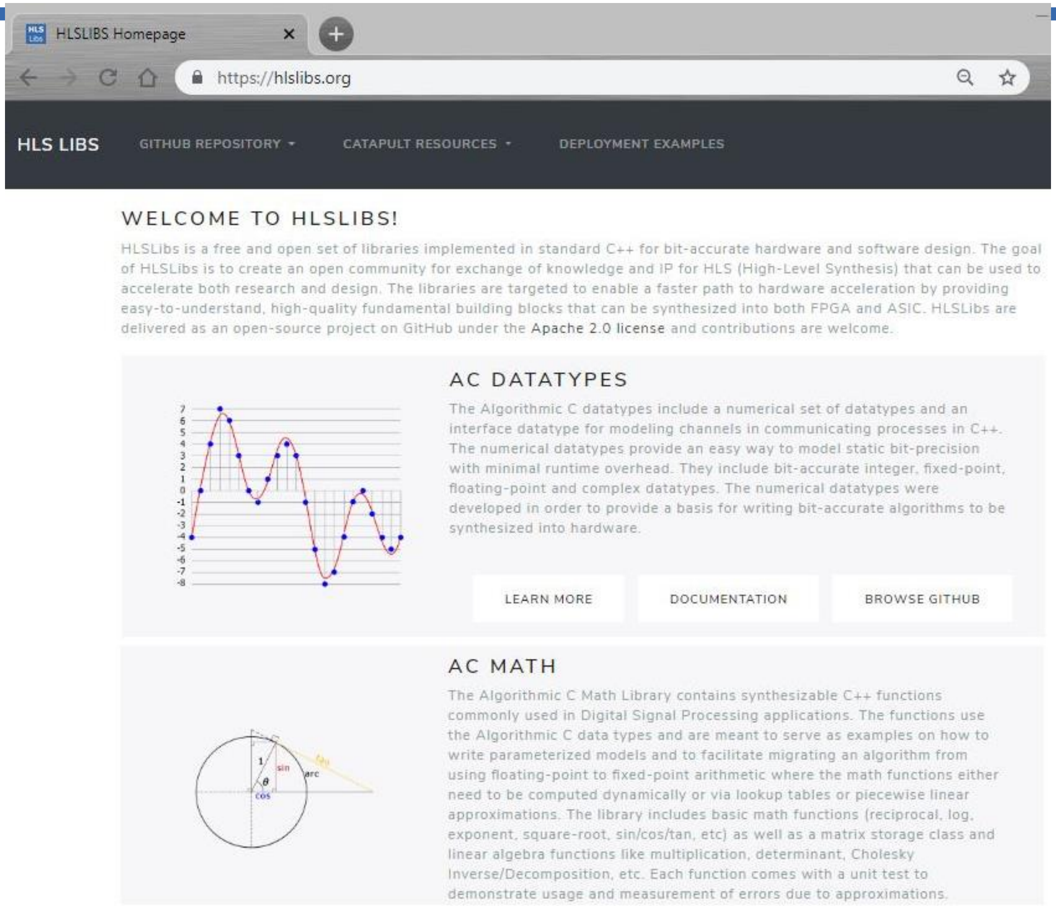
# C++ and SystemC

- Synthesis Standards (Accellera)
  - Proposed the inclusion of Algorithmic C Datatypes at SystemC Evolution Day 2017
    - Much faster and consistent semantics compared to SystemC datatypes
  - Presented a throughput accurate approach for modular IO in SystemC at SystemC Evolution Day 2017
    - A similar approach developed by NVIDIA for Matchlib.
  - Moving to C++11/14
- Language Standard for SystemC (Accellera, IEEE)
  - Move to C++11 or later
  - Proposals to use C++11 features to make it easier to write

# OPEN SOURCE IP

# HLS Open-Source IP – "HLS Libs"

- HLS Libs **Open-Source**
  Github Repository [http://hlslibs.org](http://hlslibs.org)
  - Deployment examples
  - Apache license
  - Options for high-perf or high-accuracy
- AC Datatypes
- AC Math
  - Basic math/trig functions
  - Matrix/Linear Algebra class/funcs
- AC DSP
  - 1-D Filter blocks
  - Various FFT architectures



# Conclusion

- Verification
  - Key motivator to move to HLS
  - Higher abstraction => much faster
    - Move most of the verification at this level
  - Set of tools to reduce bugs and help RTL verification
- Low Power
  - Main reason to build custom hardware
  - Most gains are achieved at higher abstraction
- High-level Synthesis allows designers to keep up with evolving architectures and changing specifications
- RTL integration and verification hand-off is greatly simplified

Mentor®

A Siemens Business

www.mentor.com