High-Level Synthesis

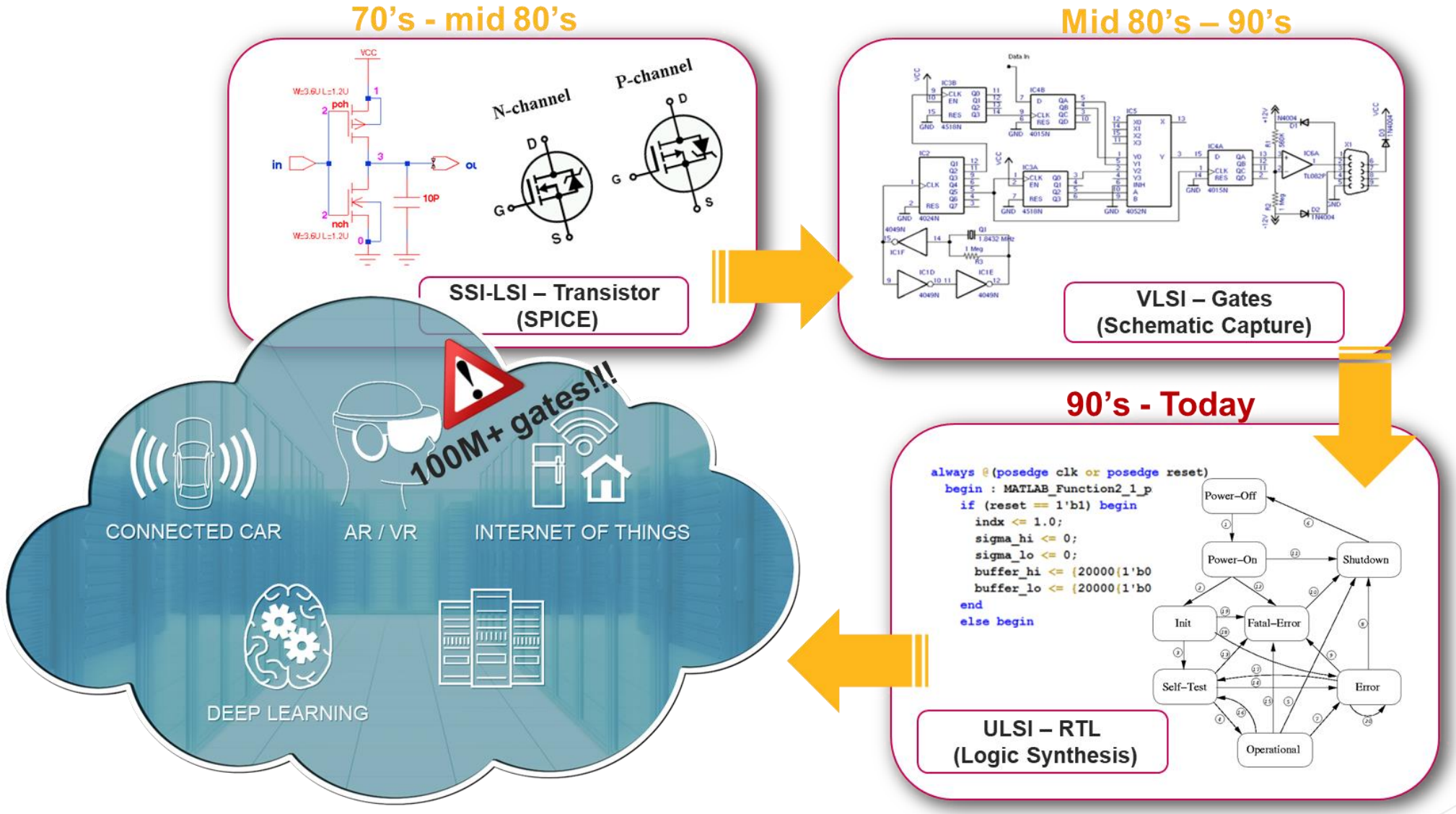
1) 高階合成(High-Level Synthesis)發展與業界採用近況簡介
2) 高階合成實務 以Cadence Stratus HLS為例

Edward Yeh (thyeh@cadence.com
Sr. AE Manager

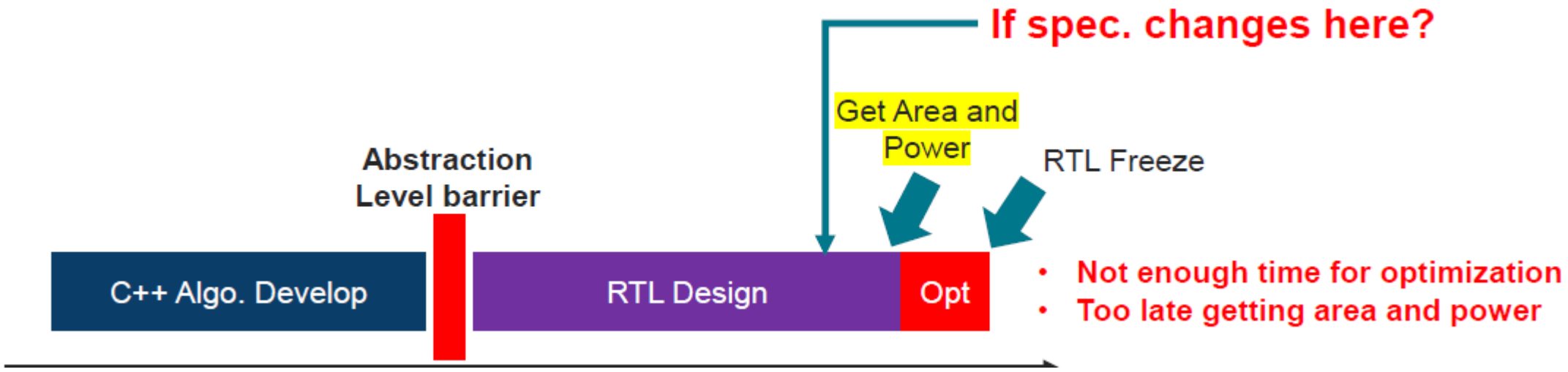
Dec. 2023

cadence®

Complexity drives design abstraction



RTL challenges



- Abstraction barrier
  - RTL design starts when algorithm completes
  - Need to understand C++ before Verilog coding
  - Long development time
- Changes in algorithm requires significant modification in RTL
- Need strong hardware expertise for good QoR
  - No time to attempt different micro-architecture implementations

What is High-Level Synthesis (HLS)

- **Describe higher abstraction than RTL, Output RTL (e.g. Verilog/VHDL)**
  - High level language (HLL) such as C++/SystemC/Python/Matlab
  - Domain specific language
  - More abstract HDL such as CHISEL, BSV (bluespec systemc verilog)

```
1   // See LICENSE for license details.
2
3   package example
4
5   import Chisel._
6
7   class GCD extends Module {
8     val io = new Bundle {
9       val a  = UInt(INPUT,  16)
10      val b  = UInt(INPUT,  16)
11      val e  = Bool(INPUT)
12      val z  = UInt(OUTPUT, 16)
13      val v  = Bool(OUTPUT)
14    }
15    val x  = Reg(UInt())
16    val y  = Reg(UInt())
17    when   (x > y) { x := x - y }
18    unless (x > y) { y := y - x }
19    when (io.e) { x := io.a; y := io.b }
20    io.z := x
21    io.v := y === UInt(0)
22  }
```

Source: https://github.com/ucb-bar/pwm-chisel-example/blob/master/src/main/scala/example/GCD.scala

```
1
2   package gcd;
3
4   interface IO;
5     method Action start(UInt#(32) a, UInt#(32) b);
6     method UInt#(32) result();
7   endinterface
8
9
10  (* synthesize *)
11  module mkGCD (IO);
12    Reg#(UInt#(32)) x <- mkReg(0);
13    Reg#(UInt#(32)) y <- mkReg(0);
14
15    rule r1 ((x > y) && (y != 0));
16      x <= x - y;
17    endrule
18
19    rule r2 ((x <= y) && (y != 0));
20      y <= y - x;
21    endrule
22
23    method Action start(
24      UInt#(32) a,
25      UInt#(32) b) if (y == 0);
26      x <= a; y <= b;
27    endmethod
28
29    method UInt#(32) result() if (y==0);
30      return x;
31    endmethod
```

Source: https://github.com/cfelton/alt.hdl/blob/master/examples/ex3_zpexgcd/bsv/gcd.bsv

Abstraction level of different language

图2比较了 Verilog, VHDL, Chisel, HLS, BSV 的抽象层次(Level of abstraction)和架构透明度(Architectural Transparent) 。架构透明度越高，语言对电路细节(微架构)的控制能力越强，生成的电路的行为也越容易被开发者掌控(可预测性高)。

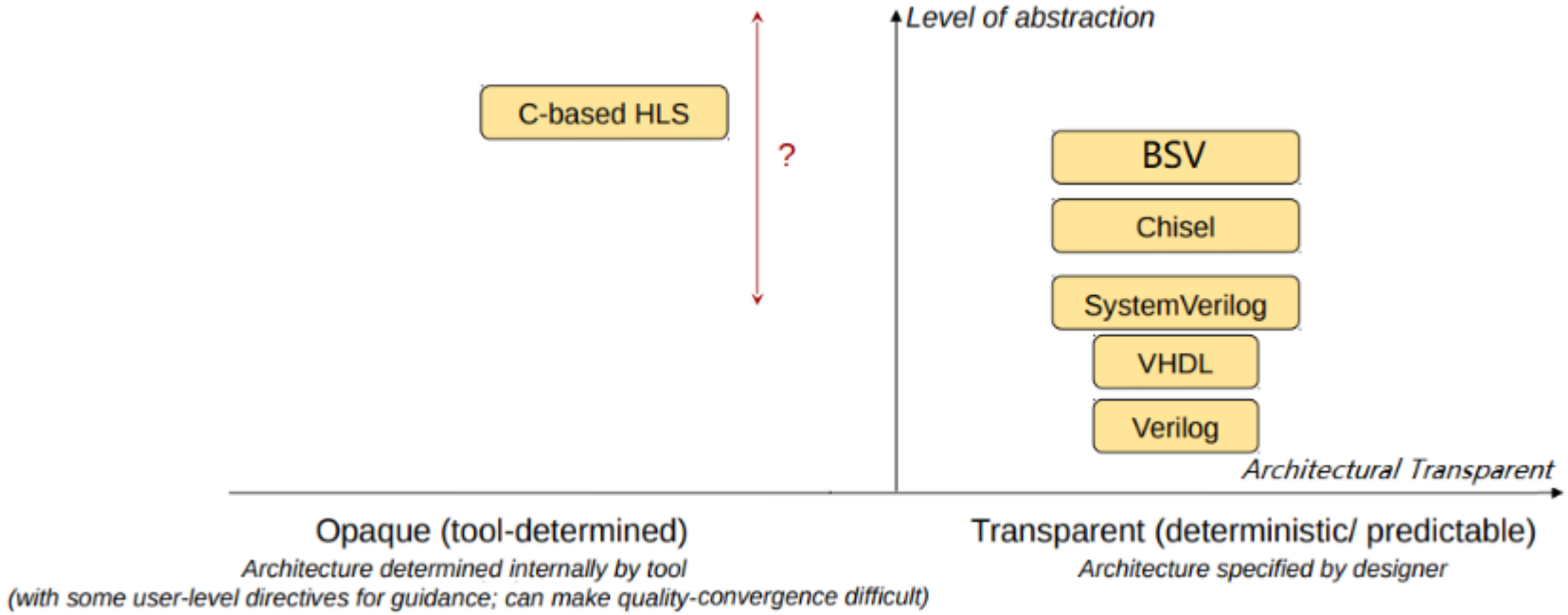


图2：比较 Verilog, VHDL, Chisel, HLS 与 BSV

Source: https://github.com/WangXuan95/BSV_Tutorial_cn#head8

High-Level Synthesis History

High-Level Synthesis History - 1

- Generation 1: 1980s – early 1990s
  - Research generation.
  - A number of seminal researches had strong impact on filed development.

- Generation 2: mid-1990 – early 2000s
  - EDA companies provided HLS tools (e.g. Behavioral Compiler from Synopsys)
  - Failed due to
    - No gains using generation 2 RTL (either better QoR or less effort with the same QoR)
    - Wrong input language: behavioral HDL
    - Poor QoR

Source: High-level Synthesis: Past, Present, and Future, IEEE Design and Test of Computers, 2009.

High-Level Synthesis History - 2

- Generation 3: from early 2000s
  - A lots of commercial HLS tools
    - e.g. Cadence C-to-Silicon complier, Forte Cynthesizer,
    - Metor CatapultC, AutoESL AutoPilot, Bluespec, NEC CyberWorkBench
  - Get some success in Industry (especially in Japan)
    - Right language: C/C++/SystemC-based synthesis
    - Improved QoR
    - Focus on domain of application (e.g. data-path)
    - Design domain shifted to signal and multimedia processing
    - Demand from algorithm-to-FPGA

Source: High-level Synthesis: Past, Present, and Future, IEEE Design and Test of Computers, 2009.

3rd Generation of HLS: Brief

Typical design steps of High-level synthesis

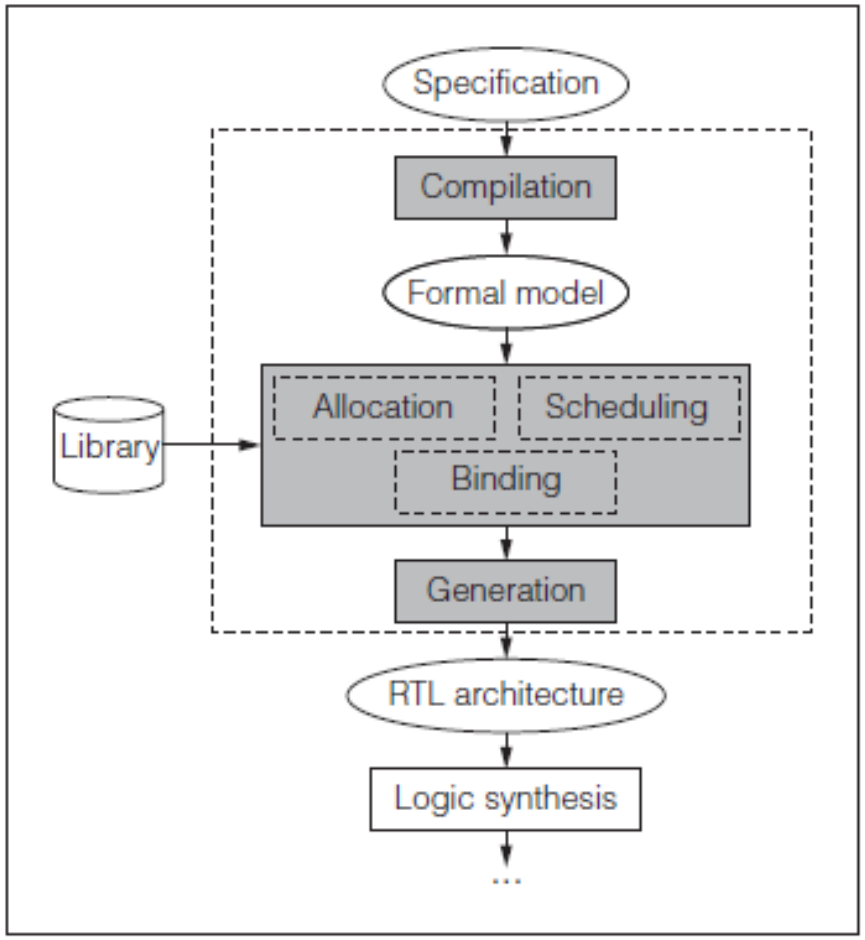


Figure 1. High-level synthesis (HLS) design steps.

Source: An Introduction to High-Level Synthesis, IEEE Design and Test of Computers, 2009.

Specification:

an HLL such as C/C++/SystemC/C-variant with **determined micro-architectures**

Compilation:

compiler-level optimization

Formal model:

CDFG, control data-flow graph

Library:

characterized resource/functional unit (FU). Pre-characterization or on-the-fly characterization

Allocation:

allocated required resources/FUs

Scheduling:

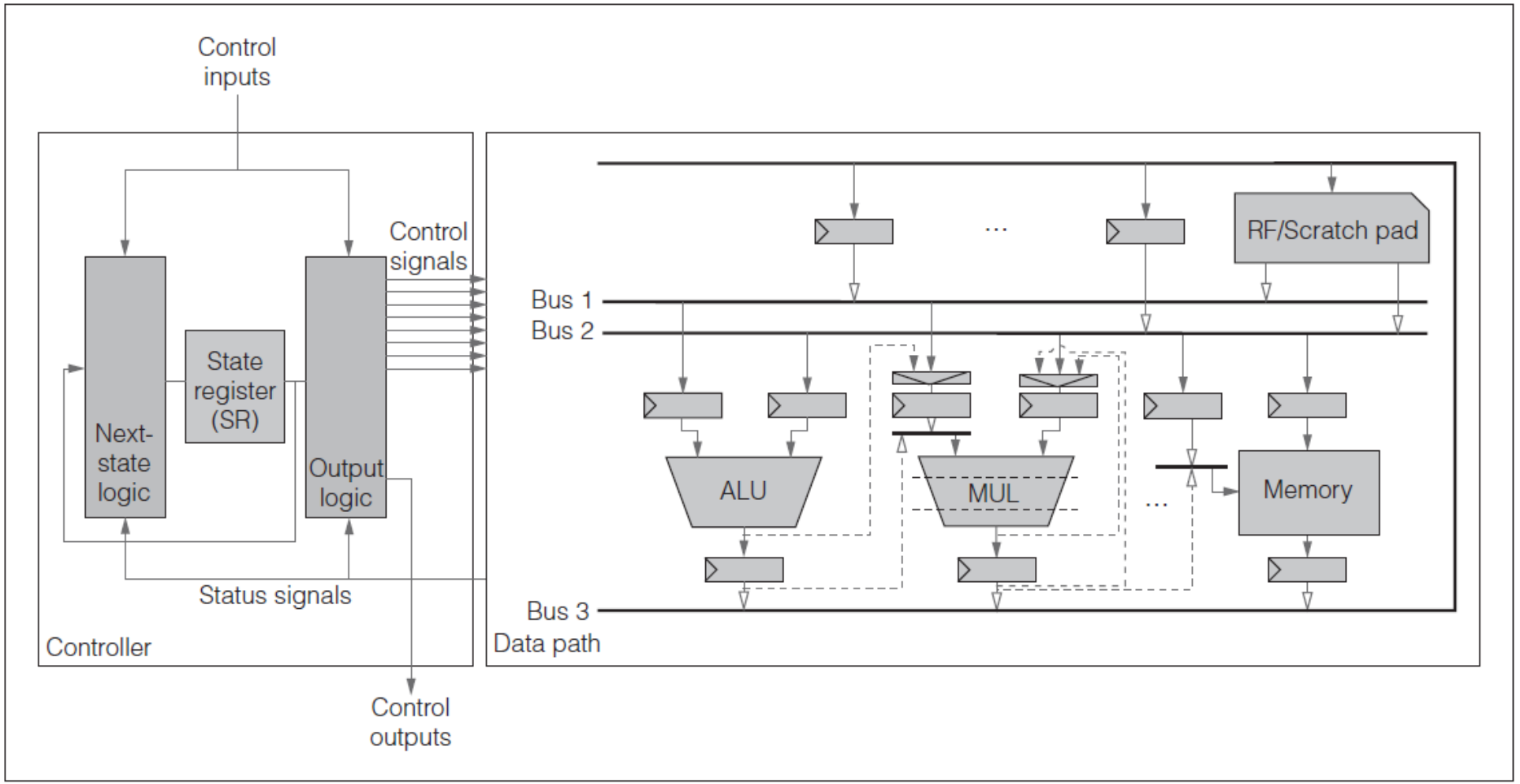
the cycle to which operations will be assigned according to given constraints

Binding:

the FUs to which operations will be assigned to

the registers to which variables will be assigned to

Typical RTL architecture

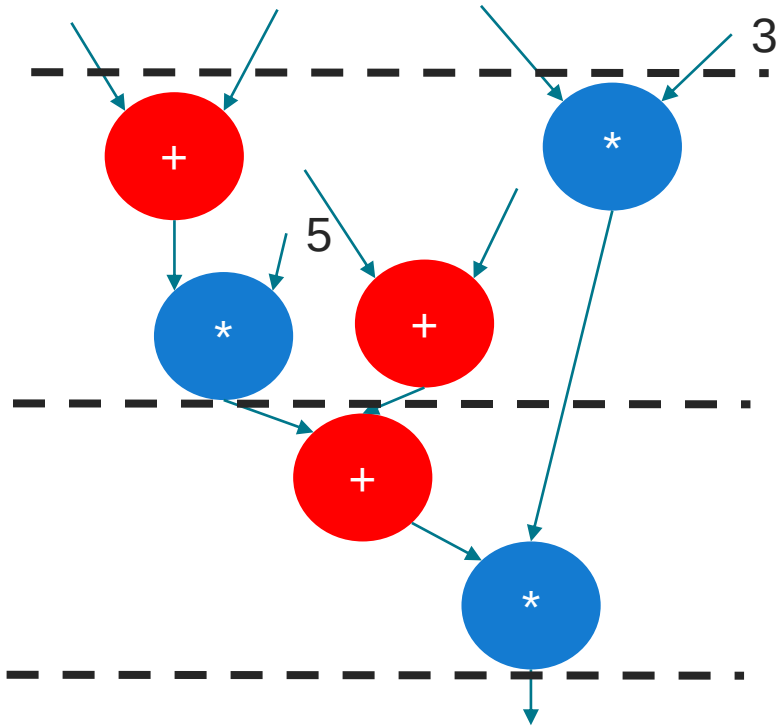


Source: An Introduction to High-Level Synthesis, IEEE Design and Test of Computers, 2009.

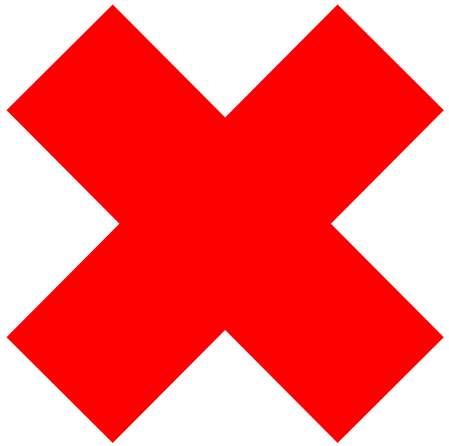
An example of scheduling

- Assume the delay of multiplier is 3ns, and the delay of adder is 2ns.

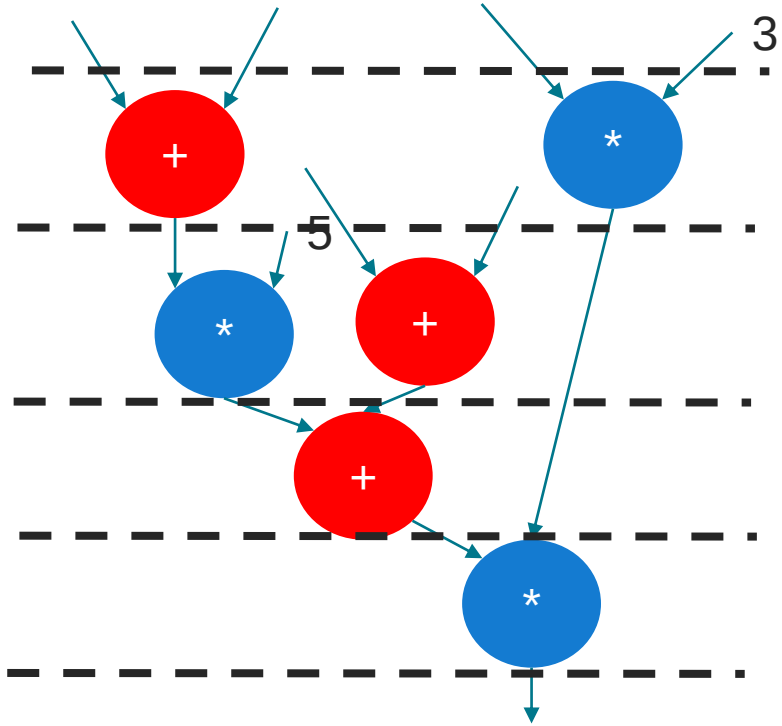
Clock period: 10ns, latency constraint:2



Clock period: 3ns latency constraint: 3



Clock period: 3ns latency constraint: 4



Micro-architectures

- **HLS tool does not know what micro-architecture a user want to implement.**
  - An array is a memory or a set of registers?
    - Sometimes code change is required if an array is mapped to memory
  - Pipelined design or non-pipelined design?
  - A for-loop is unrolled or not?
  - Handshaking with other modules?

- **The user should assign those by directives/pragmas to guide HLS tools**
  - HW guys can do this job better

```
while(1) {
    // pipeline directive
    out = mem[in] + mem[in+1];
}
```

Resource conflict

```
while(1) {
    // pipeline directive
    in = mem [raddr]
    out = func(in);
    mem[waddr] = out
}
```

Need to avoid WAR hazard
(the same address)

Commercial 3rd Generation of HLS

- ASIC flow
  - **Stratus HLS, Cadence (#1 market share)**
  - **Catapult HLS, Siemens**
  - CyberWorkBench, NEC
  - Synphony HLS, Synopsys
  - Bluespec compiler, Bluespec (open source code)
- FPGA
  - **Vivado/Vitis HLS, AMD**
  - HLS Compiler, Intel
  - C-to-Hardware, Altium
  - FPGA SDK for OpenCL, Intel

3rd HLS Benefits: Improves Productivity

- HLL models greatly reduce development time and verification time

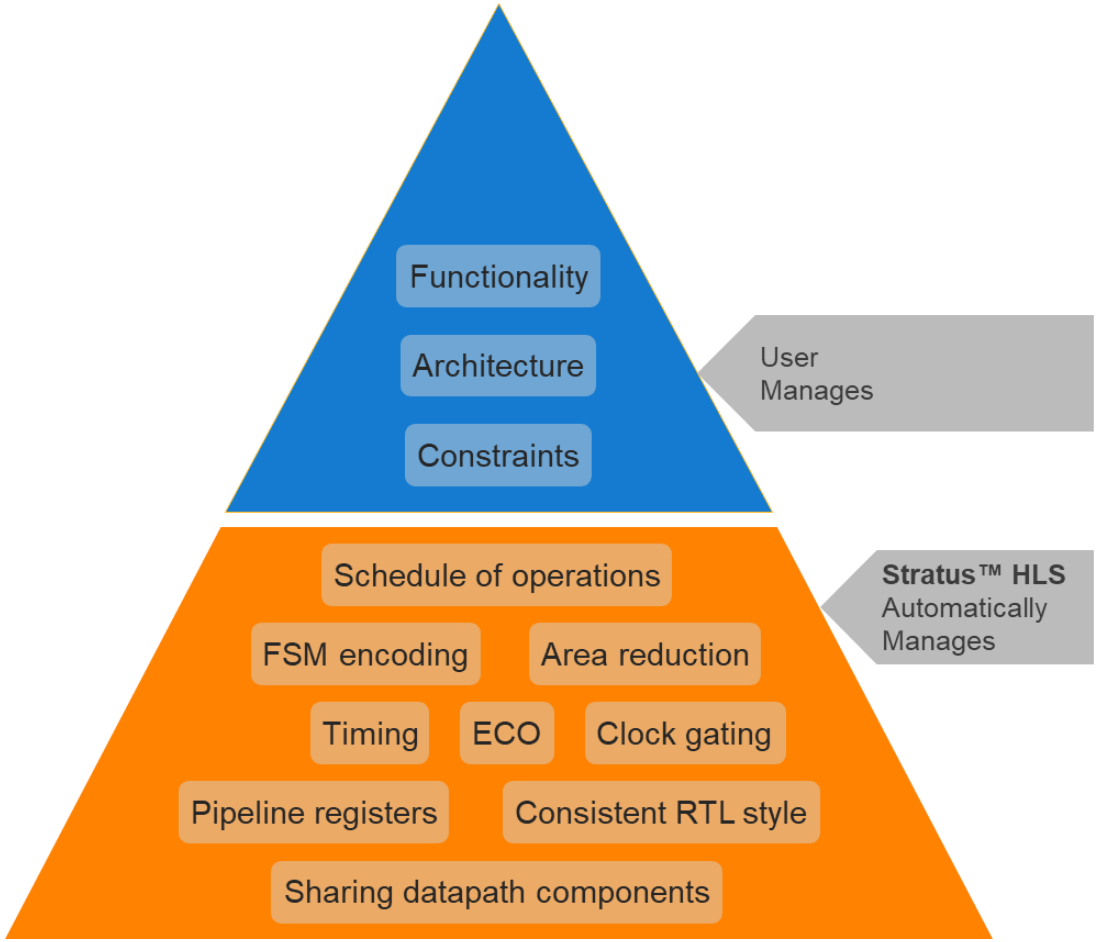


TABLE VIII
HLS AND RTL PRODUCTIVITY

	HLS		RTL		HLS/RTL
Person #	Hours	Quality*/Hours	Hours	Quality*/Hours	Quality Ratio
1	2	80	4	14	5.9×
2	4	44	9	11	4.1×
3	12	19	21	17	1.1×
4	9	47	18	6	7.6×
5	5	60	9	6	9.4×
6	20	15	26	2	9.0×
Avg.	9	44	14	9	6.2×

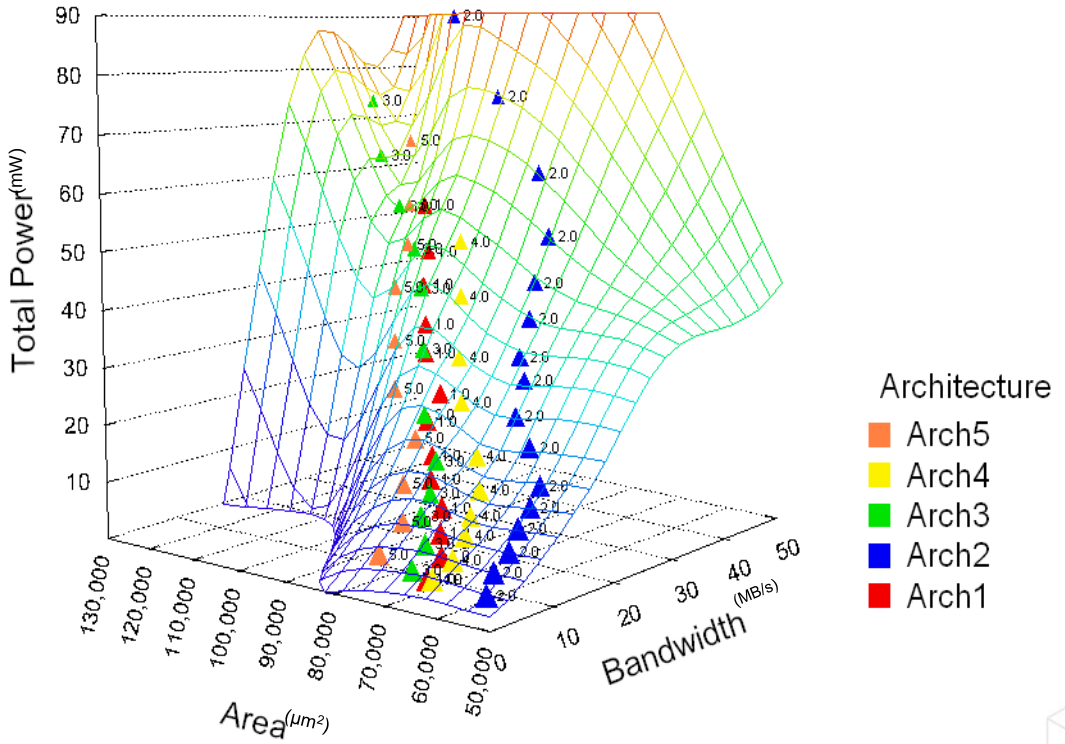
Source: Are We There Yet? A Study on the State of High-level Synthesis, IEEE Transaction on CAD, 2019.

3rd HLS Benefits: Improves Quality of Results

- Create multiple implementations from one behavioral IP description
  - Remove guesswork in picking “best” architecture
  - “Best” architecture varies for different applications
  - Explore to find optimal tradeoffs

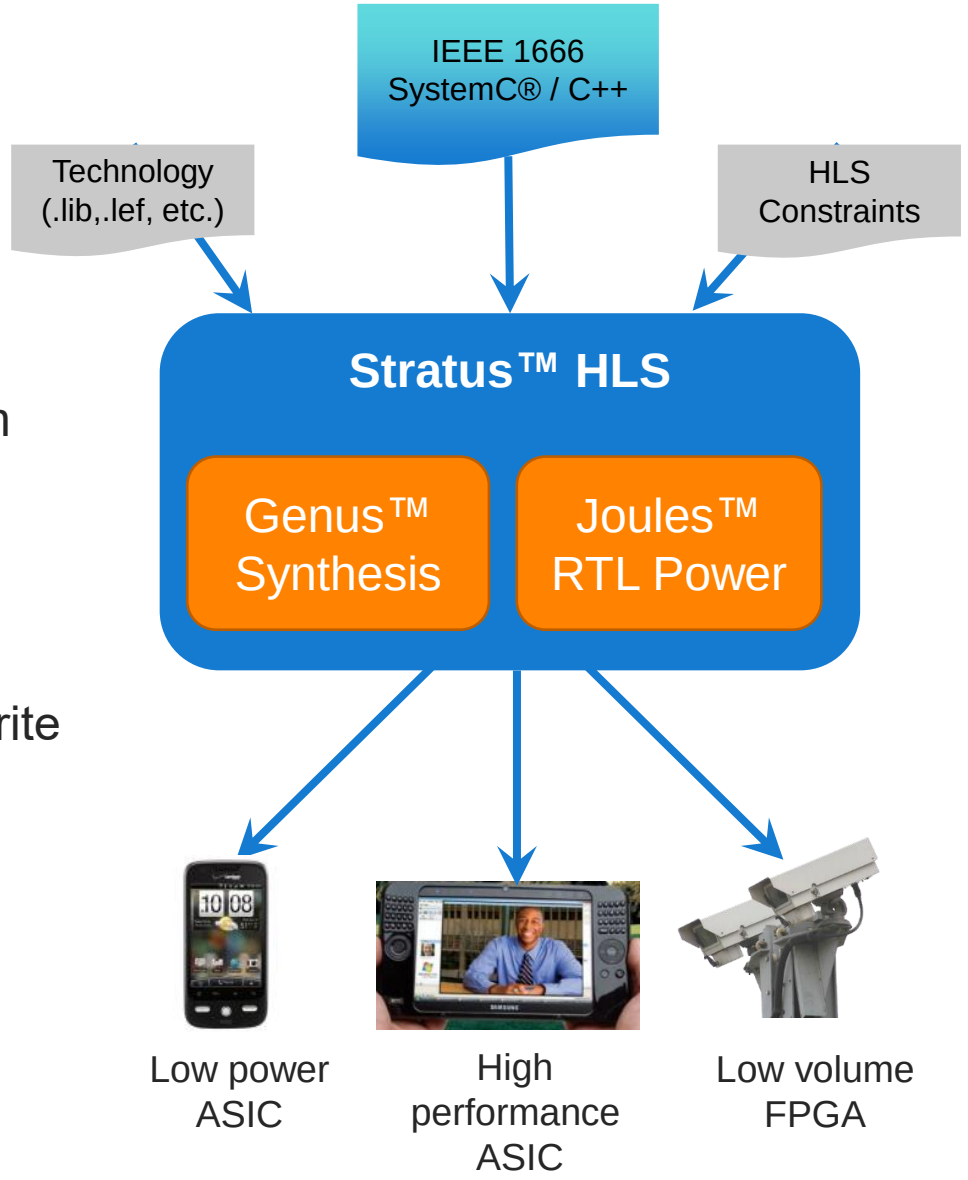
- Micro-architecture exploration
  - Automated by Stratus™ HLS
  - *Ex: Sharing vs. parallelism in the datapath*
  - *Ex: Changing perf. constraints or pipeline depth*
  - *Ex: Array storage as memory vs. register file vs. flops*

- Macro-architecture exploration
  - Assisted by Stratus HLS
  - *Ex: Changing I/O interfaces*
  - *Ex: Functional and algorithmic changes*



3rd HLS Benefits: Improves Reuse

- One IP model → different architectures and/or interfaces
  - C++ is golden source code
  - Directive files drive implementation for each application
- Behavioral IP is easy to change
  - “Minor” algorithmic changes require complete RTL rewrite
  - With Stratus™ HLS, reuse most of D&V environment
- Extend useful life of your IP



Stratus HLS improves ROI on your IP development

3rd HLS Benefits: Example

Before and After Stratus HLS flow

If spec. changes here?

Get Area and Power

RTL Freeze

Abstraction Level barrier

C++ Algo. Develop

RTL Design

Opt

- Not enough time for optimization
- Too late getting area and power

C++ Algo. Develop with Stratus HLS assist

Talk in same language

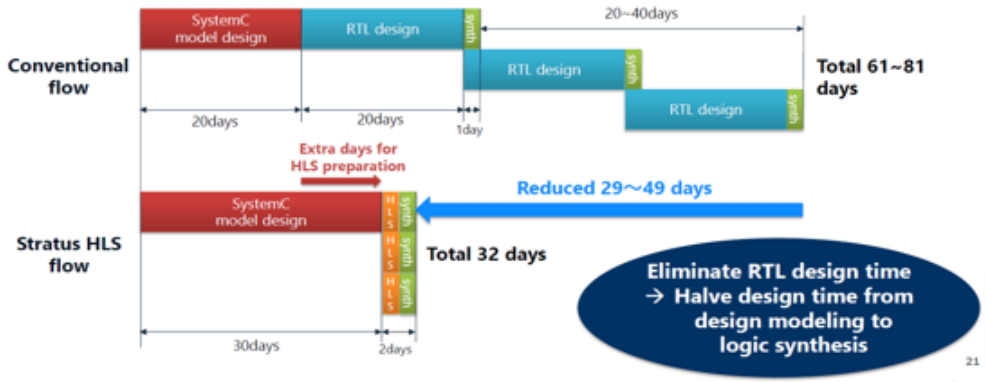
RTL Freeze

Stratus HLS + Opt

Early trade-off bitwidth and performance

Re-use code in design

You can start design when algo is 80-85% complete



- 2X productivity
- The same patterns verifies design, faster simulation
- Get power and area during design stage

3rd HLS: Weakness (IMPO)

- Not a one-step function from HLL-to-RTL
  - Need hardware expertise to guide HLS tools, and for better QoR
- Still a block/IP-based design tool
  - May not be attractive to Architects or Algorithm developers
- Not suitable for the designs needing fine-grained control

- Other well-known problems
  - HLL & RTL equivalent checking
    - No mature solution so far
  - ECO has room to improvement
  - 100% RTL coverage is impossible

HLS adoption in Industry

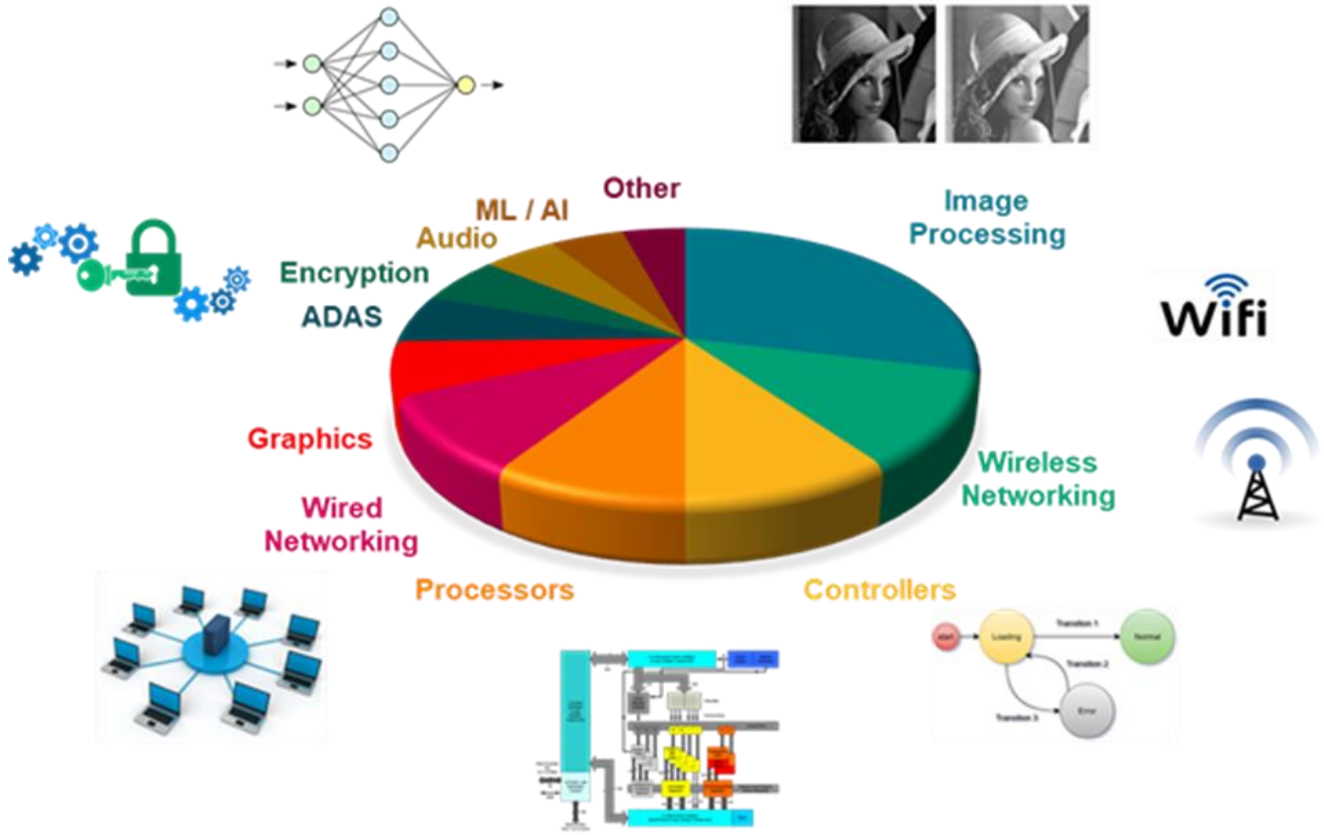
Who has adopted HLS?

Company
Qualcomm
Broadcom
Nvidia
AMD
MediaTek
Marvell
Realtek

Source: https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/resources/

- **6 of top-7 IC design houses employ Stratus HLS**
- **Intel, Samsung, lots of customers in Japan, China**

What types of designs are developed by HLS



Stratus™ HLS Production Applications
Source: Cadence HLS industry survey, 2018

- Stratus is used on a wide variety of design
- Key applications
  - Signal, audio, image, and video-processing
  - AI/ML

User experiences from Industry - 1

- Design: 802.11ah baseband
  - The same baseband functionality with different performance
    - **RTL will be different**

- HLS achieves
  - Different interface by #define due to Radio is typically specified by customer
  - Single code-base with different implementations for different performance requirements
    - C++ template
  - Ability to handle new requirement due to spec. change
    - 2 months to make change
    - Hand RTL flow is infeasible to achieve this

User experiences from Industry - 2

- From Top CPU company
  - The design teams who are happiest report :

    "The HLS flow got us to meet our milestone X times faster than we estimate with our hand-written RTL approach"

    (Where the average value of X is 2)

    "The design couldn't be done using conventional methods, HLS was the only path that gets us there".

  - Higher-level language (usually C++/SystemC) is used, instead of Verilog/RTL
    - Source code is focused on describing **functionality** (architecture)
    - The source is usually written by a designer or an architect.
      - In some cases it's possible to re-use existing C++/SystemC code.
    - The compact SystemC code (average reduction of ~70%) is verified with 10X-1000X reduction in simulation time.

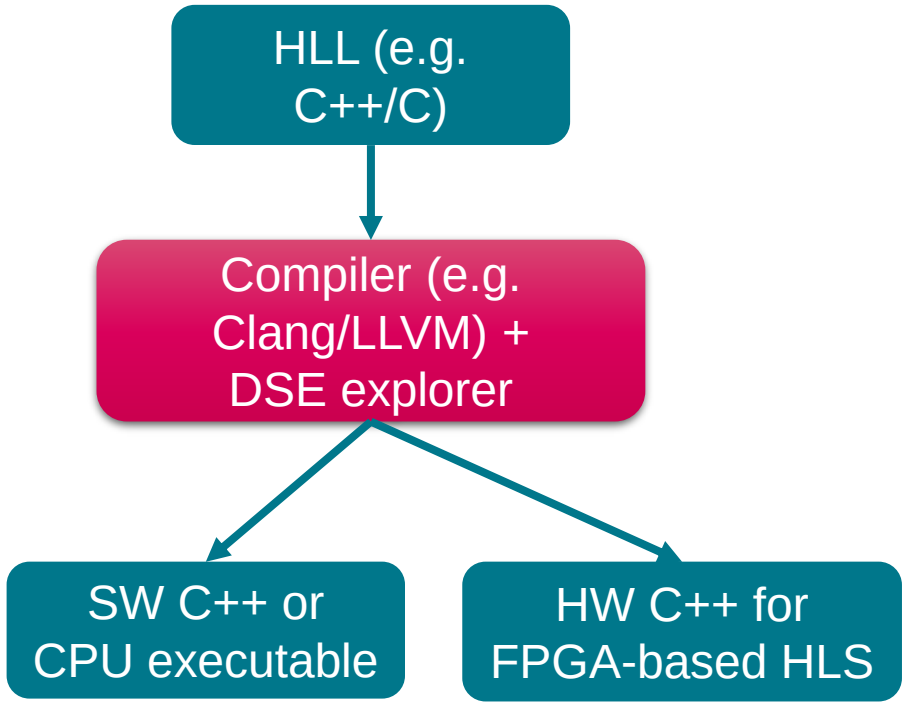
User experiences from Industry - 3

- From Top IC design house

- IP team's experience using HLS has been excellent
  - Small sized team able to meet **aggressive SoC deadlines** on multiple chips
  - Ability to accommodate **late enhancements** in the algorithm design and be code complete with verification ready in a matter of days.
  - No need for ECO with thorough verification in SystemC environment and RTL DV environment
  - **Design reuse** from moving target technology nodes achieved with 0 lines of code change
  - Bulk of the IP design is bug-free in RTL verification environment since the RTL is auto-generated, SystemC model is thoroughly tested in SystemC verification and 100% SystemC code coverage achieved. This results in **improved design and DV** team productivity.

FPGA-based HLS for Heterogenous Architecture

General Concept



Commercial:

- LegUp HLS/Smart HLS, MACROCHIP
- Merlin Compiler, Falcon Computing Solution (acquired by AMD/Xilinx in 2020)
- SLX-FPGA (now part of Vitis), Silexica (acquired by AMD/Xilinx in 2021)

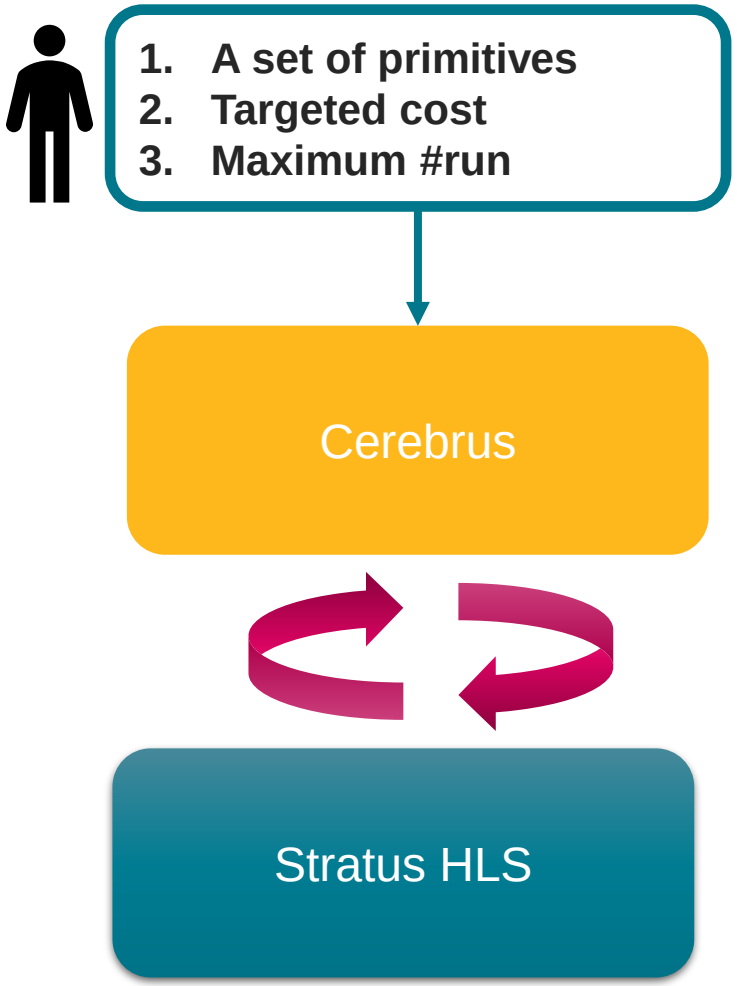
Academic:

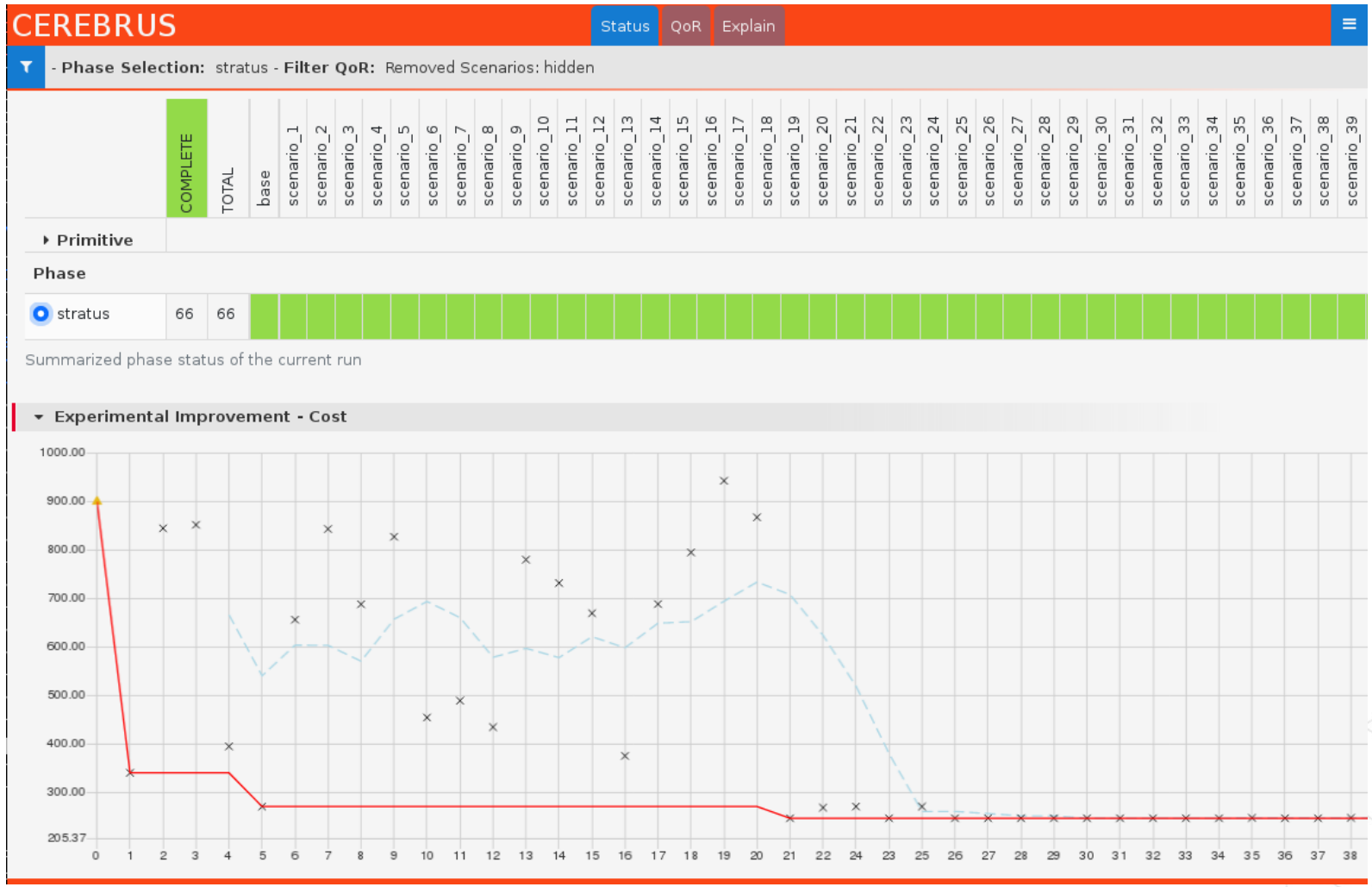
- DWARV (Delft Workbench Automated Reconfigurable VHDL) HLS Compiler
- FORST
- HOT & SPICY, University of Southern California

Source: Towards Automatic High-Level Code Deployment on Reconfigurable Platforms: A Survey of High-Level Synthesis Tools and Toolchains, IEEE Access, 2020

Machine Learning helps Design Space Exploration with HLS

Cerebrus + Stratus





LLM for Code Generation

ChatGPT to generate Verilog - 1

- Implement a 128-bit multiplier with two inputs and one output in Verilog

```
// stage1
always@(clk) begin
    stage1_result = in1 * in2;
end

// stage 2
always@(clk) begin
    stage2_result = stage1_result + 256'b0;
end

// stage 3
always@(clk) begin
    out = stage2_result;
end
```

ChatGPT to generate Verilog - 2

- Implement a 3-stage pipelined design in verilog. Two adders in stage1, two multipliers in stage2, and one right shifter in stage3.

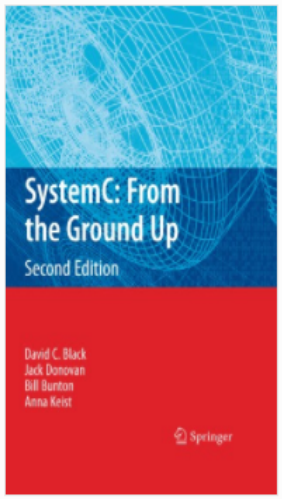
```
// stage1
always@(clk) begin
    stage1_result1 = in1 +  in2;
    stage1_result2 = in1 + in3;
end

// stage 2
always@(clk) begin
    stage2_result1 = stage1_result1 * in1;
    stage2_result2 = stage1_result2 * in2;
end

// stage 3
always@(clk) begin
    out = (stage2_result1 + stage2_result2)>>1;
end
```

Code generation in C/C++ is easier than one in Verilog. Will LLM be part of HLS to speed up HLS design?

Synthesizable SystemC: Brief

SystemC

- SystemC is a C++ library for HW descriptions
- IEEE 1666-2011
  - SystemC + TLM2.0 (Transaction-Level Modeling Standard, v2)



SystemC: From the Ground Up

Authors (view affiliations)

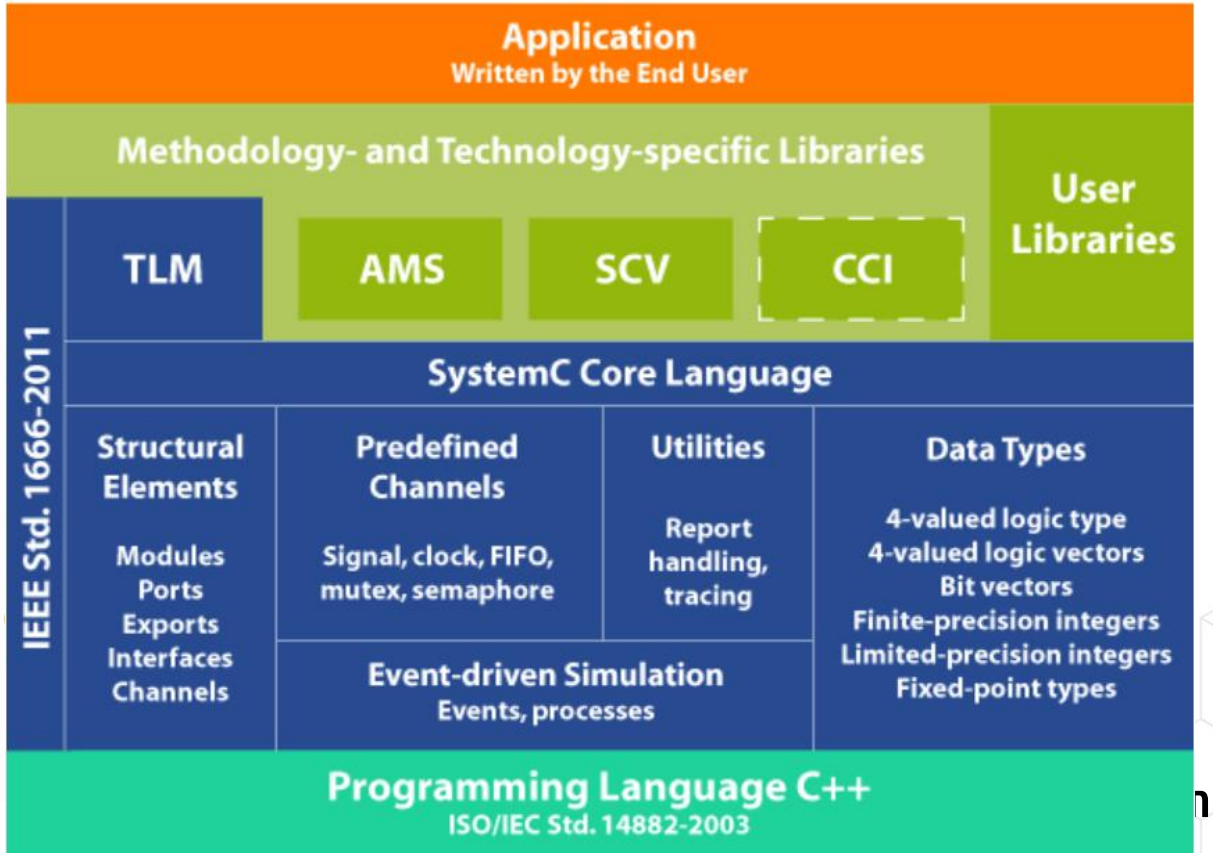
David C. Black, Jack Donovan, Bill Bunton, Anna Keist

Emphasizes TLM 2.0 and the recently adopted IEEE 1666

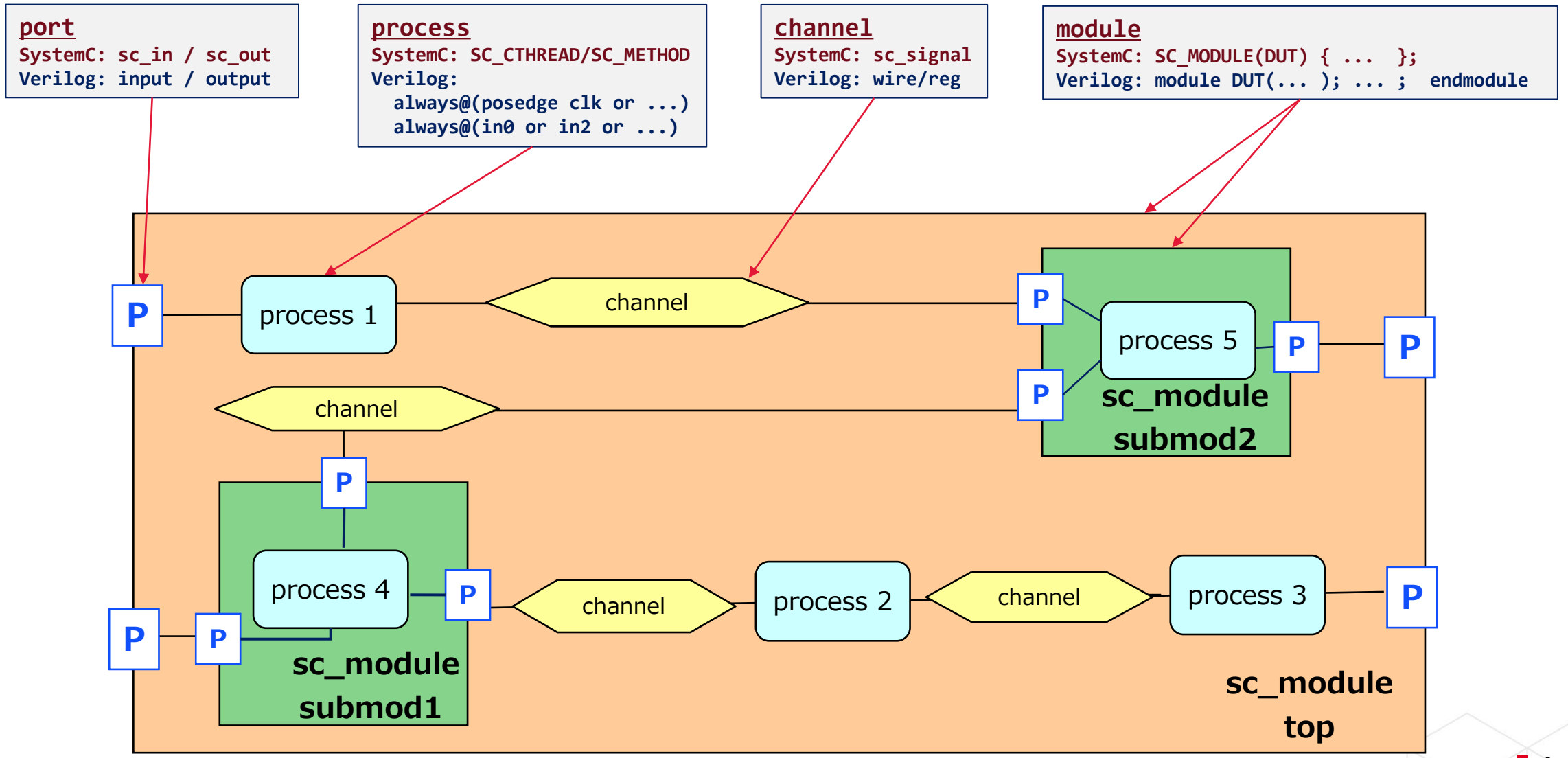
Includes SystemC Verification Library as well as an appendix that features a SystemC

Expands and updates material throughout the book

Includes supplementary material: sn.pub/extras



Main Components of Synthesizable SystemC



Typical Synthesizable SystemC Module

Header file : DUT.h

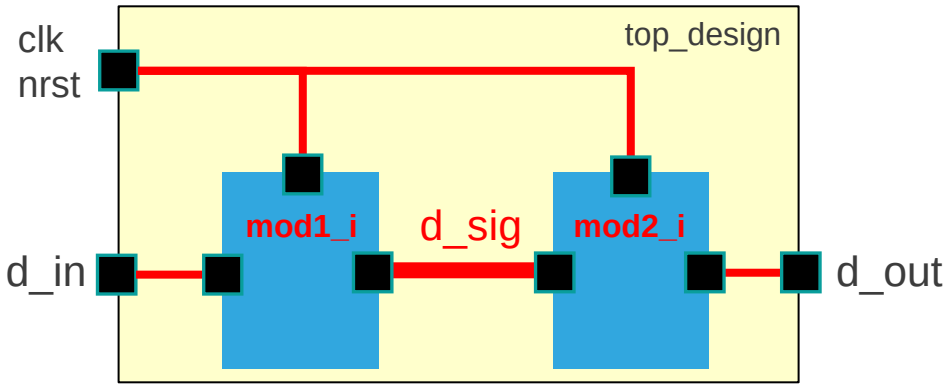
```
#ifndef DUT_H
#define DUT_H

#include <systemc.h>
#include <stratus_hls.h>

SC_MODULE(DUT) {
  sc_in <bool>         CLOCK;
  sc_in <bool>         RESET_n;
  sc_in <sc_uint<8> > IN_A;
  sc_in <sc_uint<8> > IN_B;
  sc_out<sc_uint<9> > OUT;

  void proc();

  SC_CTOR(DUT)
    : CLK("CLK")
    , RSTn("RSTn")
    , IN_A("IN_A")
    , IN_B("IN_B")
    , OUT("OUT")
    {
      SC_CTHREAD(proc, CLOCK.pos());
      async_reset_signal_is(RESET_n, false);
    }
};

#endif
```

Implementation file : DUT.cpp

```
#include "DUT.h"

void DUT::proc() {
  sc_uint<8> temp1;
  sc_uint<8> temp2;
  sc_uint<9> sum;
  {HLS_DEFINE_PROTOCOL("reset");
  OUT.write(0);
  wait();
  }
  while(true) {
    {HLS_DEFINE_PROTOCOL("input");
      temp1 = IN_A.read();
      temp2 = IN_B.read();
    }

    sum = temp1 + temp2;

    {HLS_DEFINE_PROTOCOL("output");
      OUT.write(sum);
      wait();
    }
  }
}
```

Hierarchical Modules

The sc_signal is used to connect the sub modules




```
SC_MODULE(top_design)
{
  sc_in < bool >                clk;
  sc_in < bool >                nrst;
  sc_in < sc_uint<8> >          d_in;
  sc_out< sc_uint<8> >          d_out;

  sc_signal< sc_uint<8> >       d_sig;

  mod1     mod1_i;
  mod2     mod2_i;

  SC_CTOR(top_design)
  : mod1_i("mod1_i")
  , mod2_i("mod2_i")
  {
    mod1_i.clk(clk);
    mod1_i.nrst(nrst);
    mod1_i.d_in(d_in);
    mod1_i.d_out(d_sig);

    mod2_i.clk(clk);
    mod2_i.rst(rst);
    mod2_i.d_in(d_sig);
    mod2_i.d_out(d_out);
  }
};
```

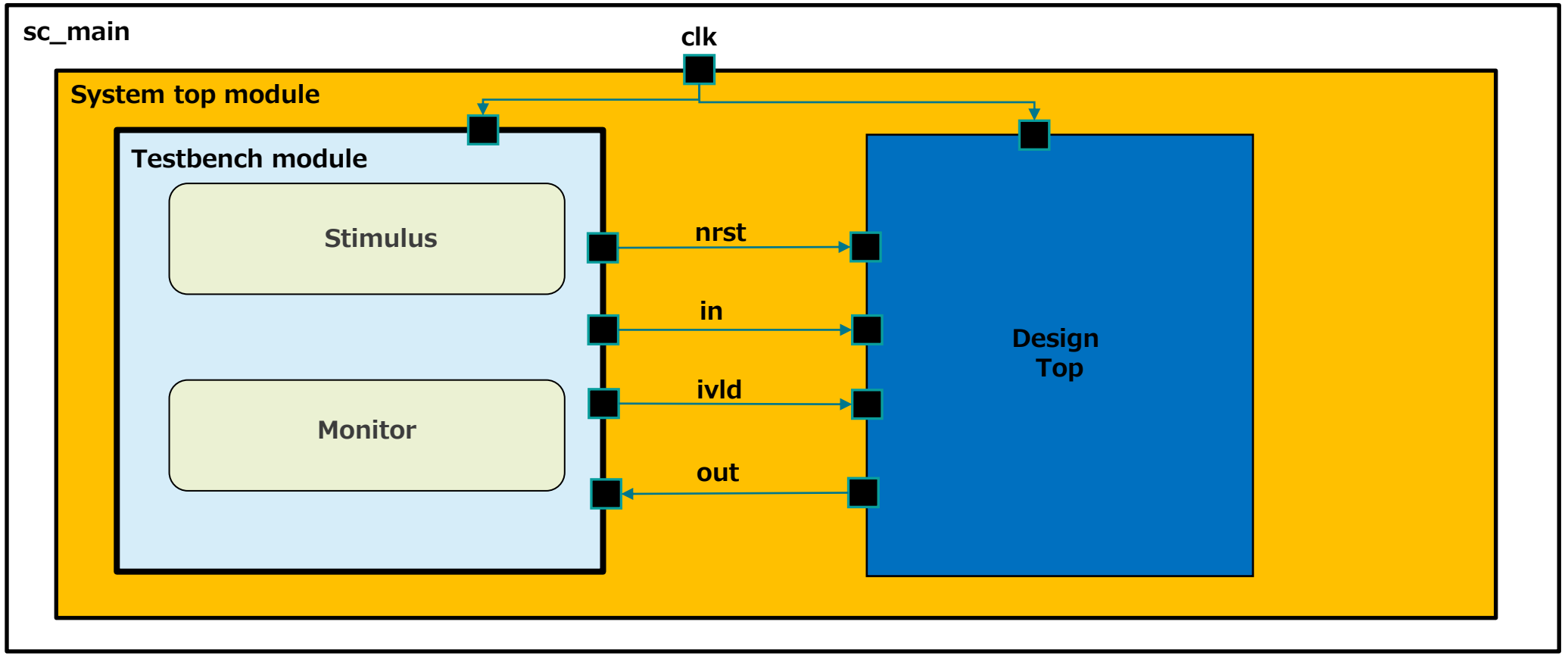
Supported Synthesizable SystemC data types

Data types	Descriptions	HLS synthesizable	Examples

sc_bit	1bit	Yes	int tmp;
			sc_uint<24> tmp24;
			sc_biguint<256> tmp256;
			
			tmp24.range(15, 0) = a + b;
			tmp256.range(47,32) = tmp24.range(31,15);
			
			sc_in< sc_uint<8> > din;
			
			sc_out< sc_biguint<256> > dout;

sc_logic	1bit(represents 0, 1, X, Z states)	Yes (only the 0,1 states)	
sc_bv<N>	N-bit	Yes	
sc_lv<N>	N-bit (represents 0, 1, X, Z states)	Yes (only the 0,1 states)	
sc_int<N> **sc_uint<N>**	N-bit signed integer. N<=64 N-bit unsigned integer. N<=64	Yes	
sc_bigint<N> **sc_biguint<N>**	N-bit signed integer. N>=1 N-bit unsigned integer. N>=1 Usually used to represent N>64 data.	Yes	
sc_[u]fixed<..>	User-defined precision	Yes	
sc_[u]fixed_fast<..>		yes	
sc_[u]fix[_fast]	**no**	Yes	

Typical Module Hierarchy of DUT and TB



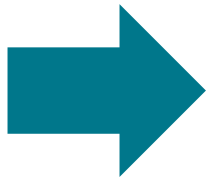
TB will be used in SystemC and Stratus-generated RTL verification
TB should be time-independent

NOTEs when using SystemC

SystemC Coding for better QoR - 1

- Use proper bitwidth for variables
- Function sharing manually usually results in better QoR

```
if (a)
   funcA(m, n, o);
else if (b)
   funcA(p, q, r);
else
   funcA(x, y, z);
```



```
if (a)
   t1=m; t2=n; t3=o;
else if (b)
   t1=p; t2=q; t3=r;
else
   t1=x; t2=y; t3=z;

funcA(t1, t2, t3);
```

SystemC Coding for better QoR - 2

- Reduce lifetime of variables (for register reduction)

```
void my_thread() {
      // declare permanent variable here
      ….
      ….
      while(1) {
          // declare intermediate variables here

          for (int i=0;i<10;i++) {
                ….
                ….
          }
      }
}
```

SystemC Coding for better QoR - 3

- Use switch-case with constant instead when full-case will not happen for shift operation, array access, and partial selection.

```
sc_uint<64> data;
sc_uint<6> shifter;
…
result = data << shifter;
```

Analyze algorithm

```
sc_uint<64> data;
sc_uint<6> shifter;
…
switch (shifter)
{
case 2: result = data << 2;break;
case 4: result = data << 4; break;
…
}
```

```
sc_uint<32> data;
sc_uint<5> idx
…
result.range(7,0) =
data.range(idx, idx-7);
```

Analyze algorithm

```
sc_uint<32> data;
sc_uint<5> idx;
…
switch (shifter)
{
case 7:   result.range(7, 0) = data.range(7, 0);break;
case 15: result.range(7, 0) = data.range(15,7); break;
case 23: ….
case 31: ….
}
```

SystemC Coding for better QoR - 4

- Explicit (constant) descriptions for shift, array accesses, partial selection (if necessary for QoR)

```
sc_uint<32> data[128];
sc_uint<7> idx;

a = data[idx];
b = data[idx+1];
```

Analyze algorithm

```
sc_uint<32> data[128];
sc_uint<7> idx;

switch (idx)
{
case 0: a = data[0]; b = data[1]; break;
case 2: a = data[2]; b = data[3]; break;
…..
case 126: a = data[126]; b = data[127];
```

Avoid memory access in deep branch

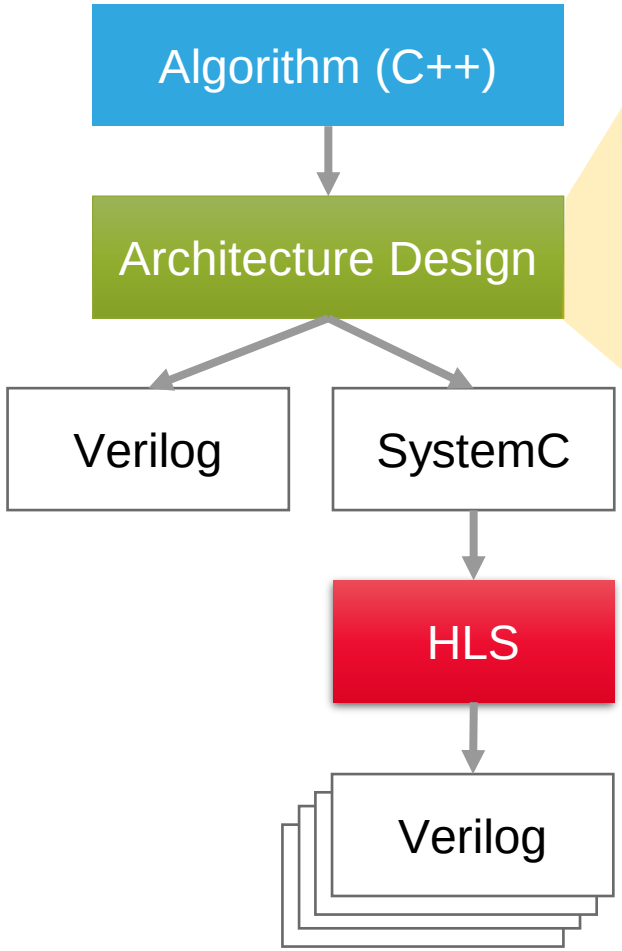
- Schedule becomes easy

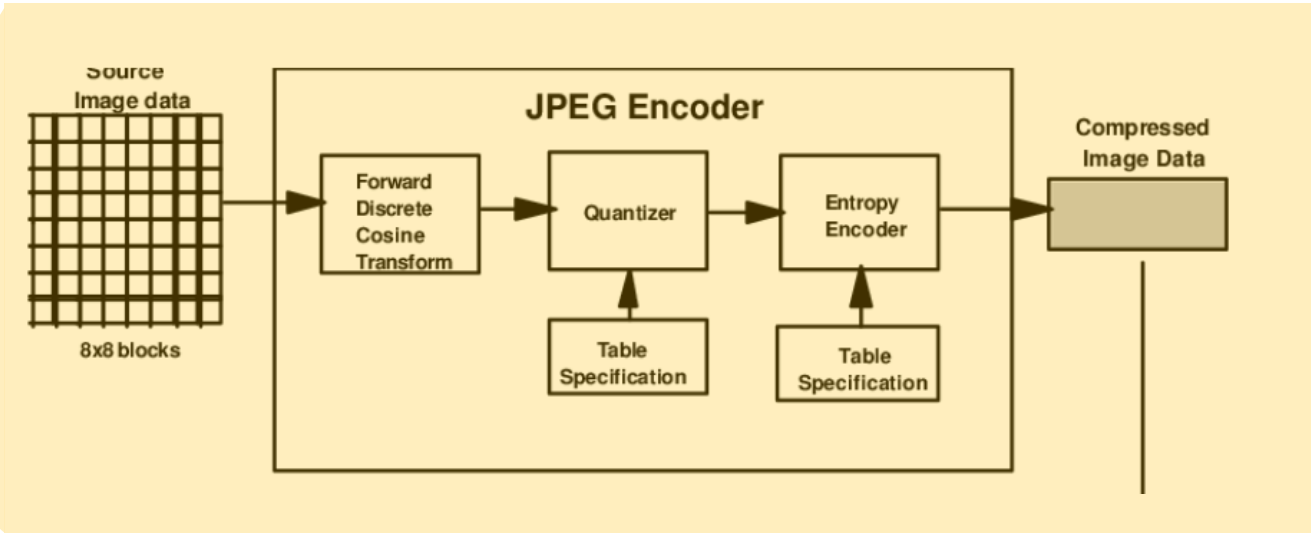
```
if (a) {
   if (b) {
      if (d) {
         if (e) {
            rdata = mem[addr];
         }
         else {
            rdata = mem[addr+2];
         }
      }
      else {
            rdata = mem[addr+8];
      }
   }
}
else {
   rdata = mem[addr+10];
}
```

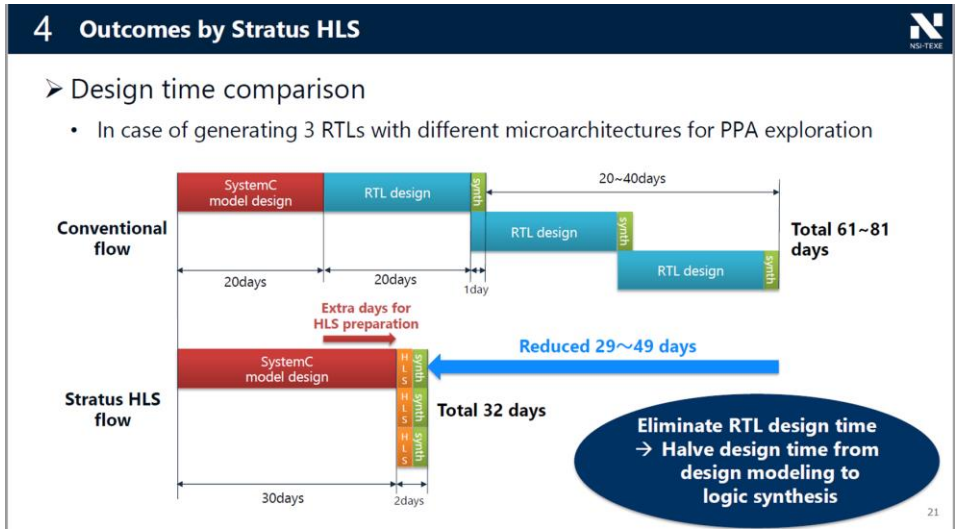
```
if (a) {
   if (b) {
      if (d) {
         if (e) {
            raddr = addr;
            ren = 1
         }
         else {
            raddr = addr+2;
            ren=1;
         }
      }
      else {
            raddr = addr + 8;
            ren=1;
      }
   }
}
else {
   raddr = addr+10;
   ren=1;
}

if (ren) {
   ren=0;
   rdata = mem[raddr];
}
```

NOTEs when using ANY HLS

HLS is not magic - 1



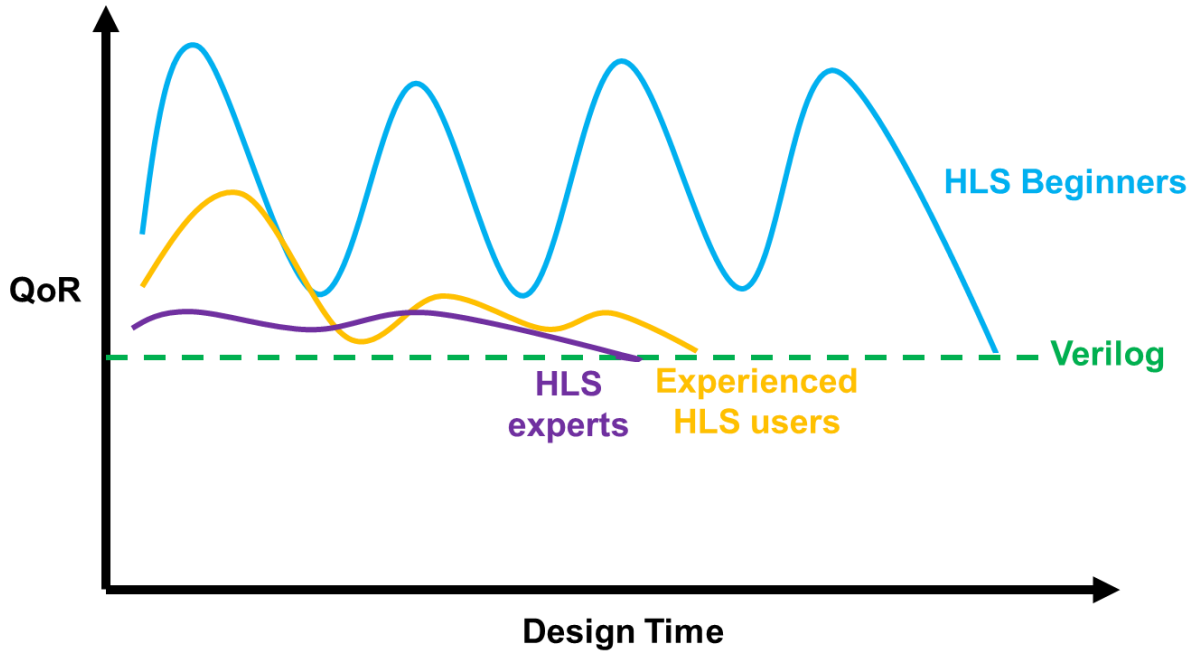




Architecture Design before HLS is a MUST.

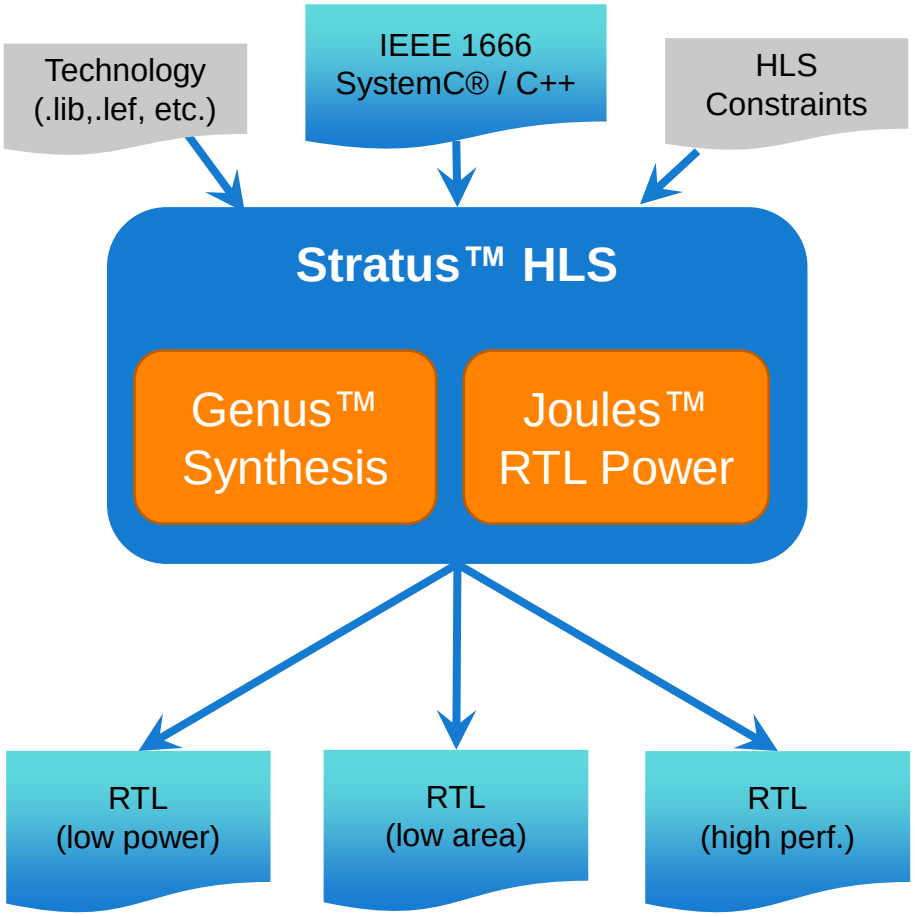
HLS is not magic - 2

- **Current HLS technology achieves competitive QoR to hand RTL**
  - **QoR-driven systemc coding**
  - **Well understand the u-architectures you want**
    - **Ability to map HLS constraints to your desired u-architectures**
  - **Understand you will have growing pain due to learning curve**



Stratus HLS Brief

Overview



Behavioral IP is
broadly reusable across
technologies and PPA targets

10x faster to verified production silicon

Writing RTL by hand

- Functionality
- Architecture
- Constraints
- Sharing
- Schedule of operations
- Multiplexers
- Pipeline registers
- Area reduction
- FSM encoding
- Timing closure
- Clock gating
- Consistent RTL style
- ECO

1-2 architectures
in 5-6 months

Creating RTL with HLS

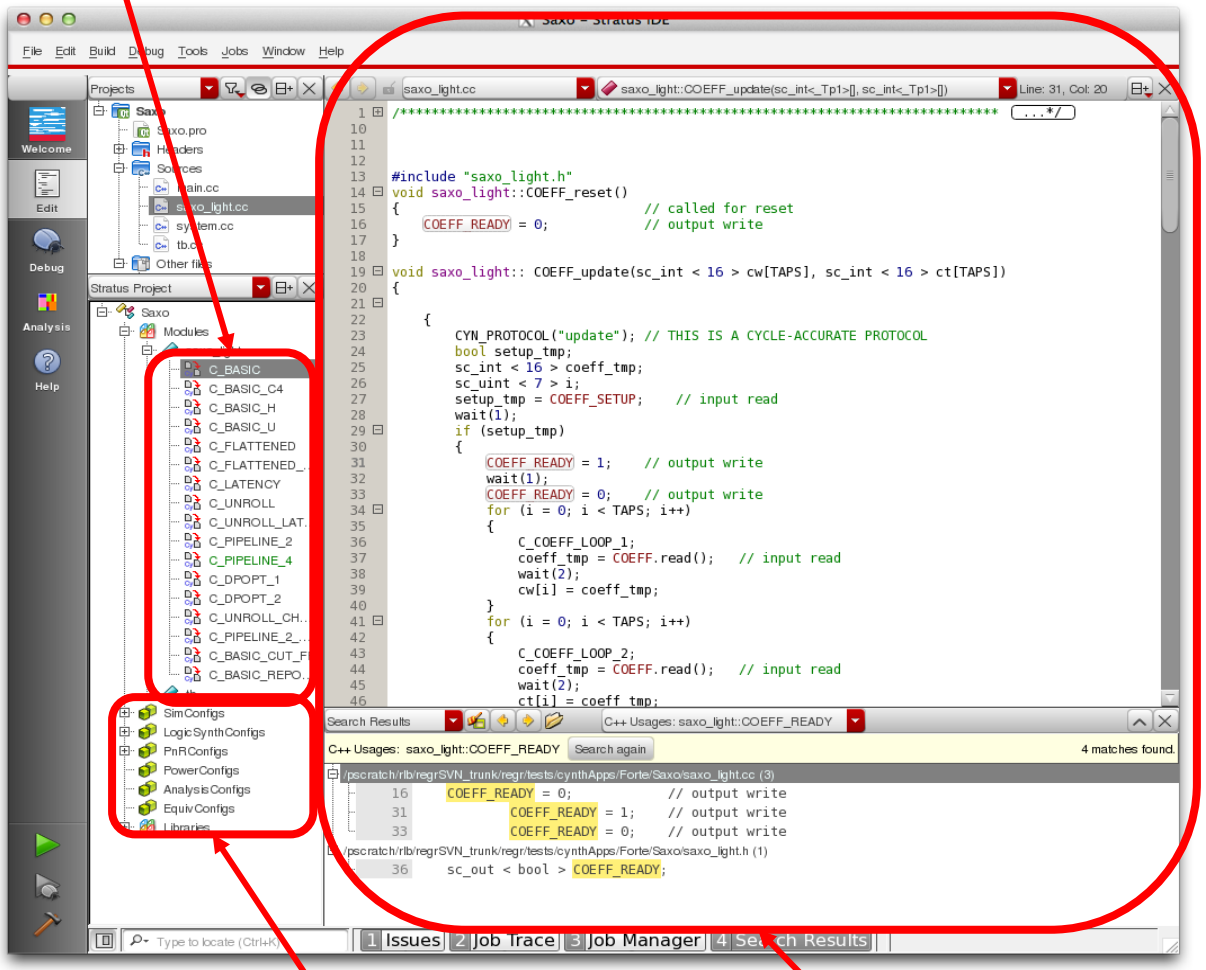
- Functionality
- Architecture
- Constraints
- **Automated by Stratus HLS**

100's of architectures
in 1-3 weeks

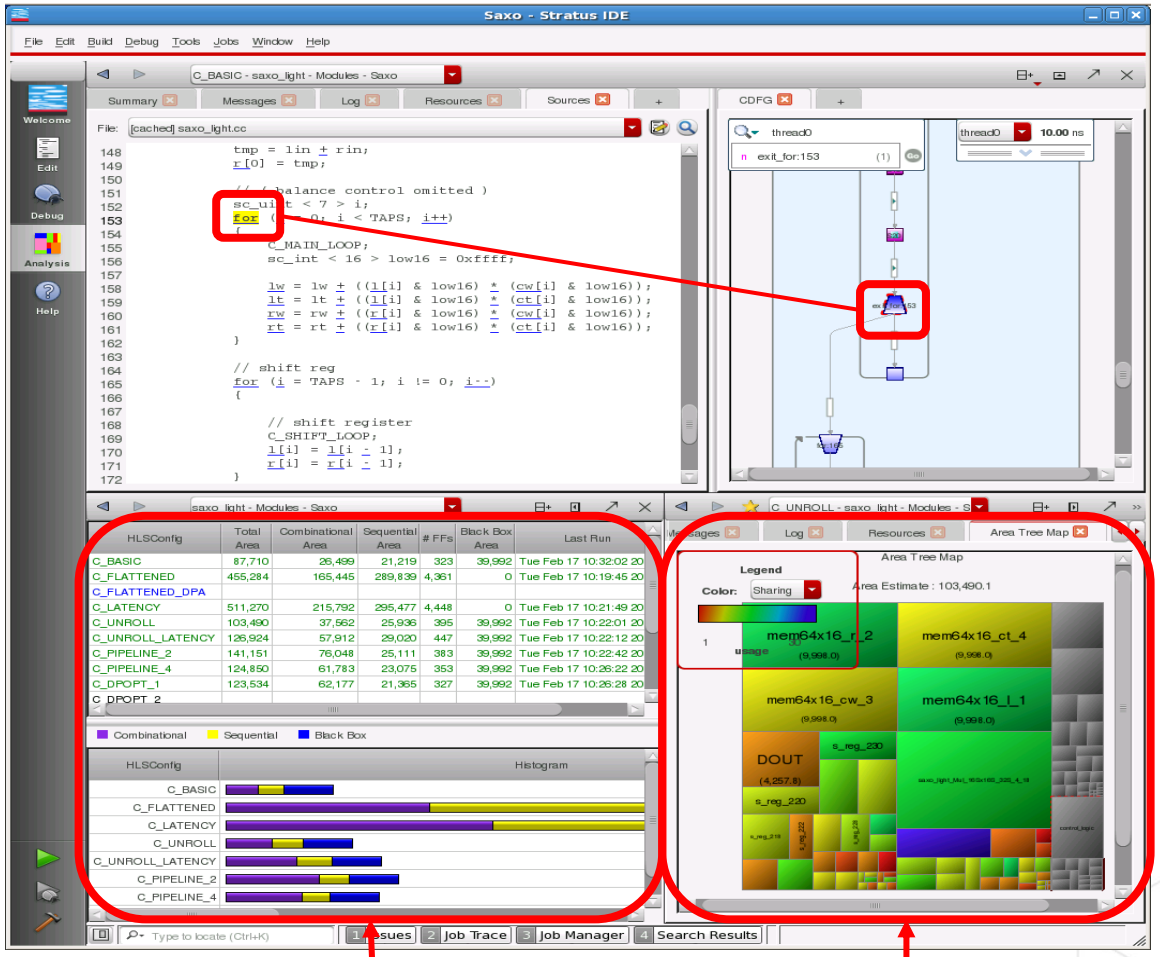
Choosing the right architecture can achieve
2x better Power, Performance & Area

Do everything in Stratus IDE

Automates multiple implementations from one source



Hot links back to IP source code



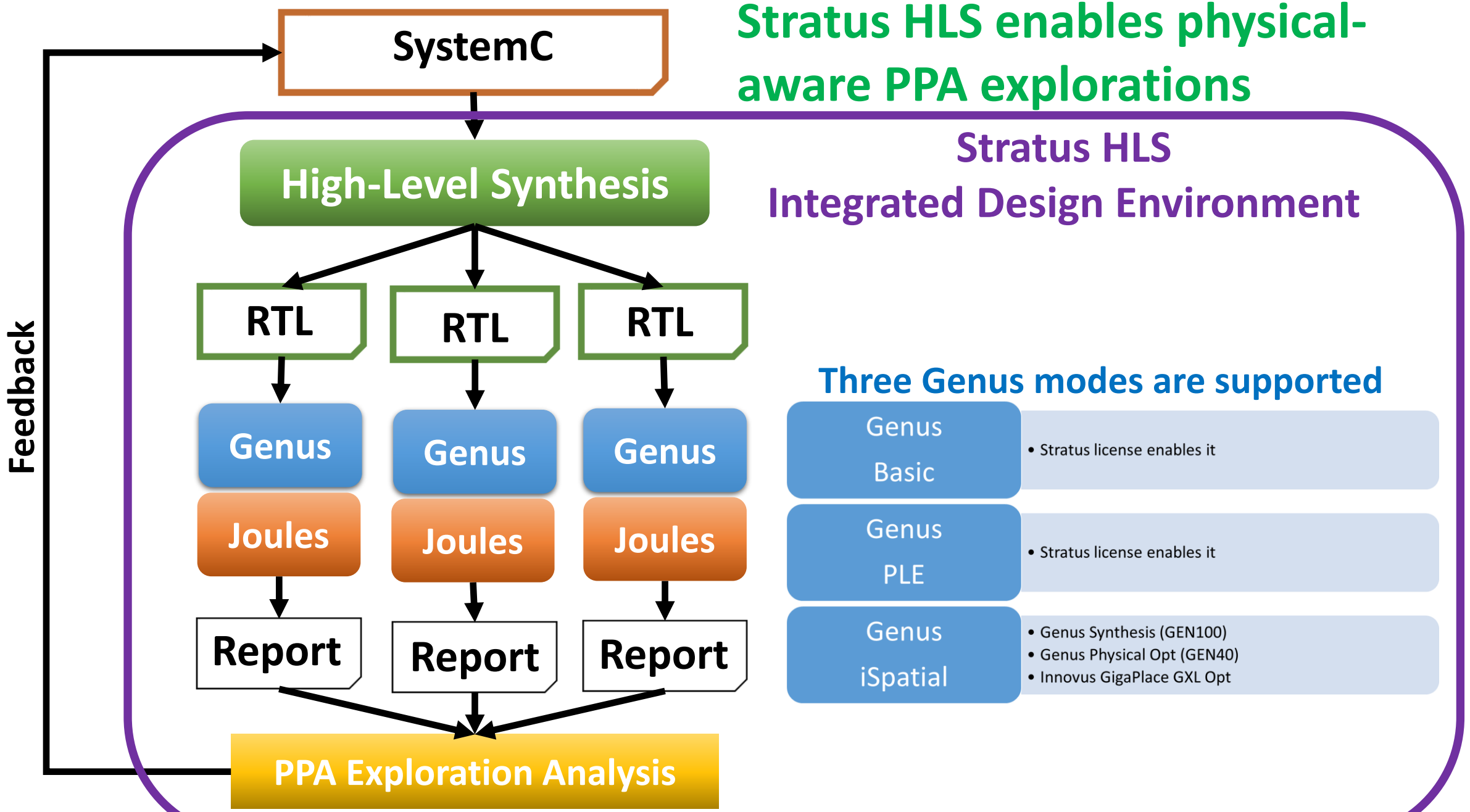
Automates integration with downstream tools

"Standard" IDE

Compare multiple configurations

Detailed analysis of each config.

Stratus HLS enables physical-aware PPA explorations

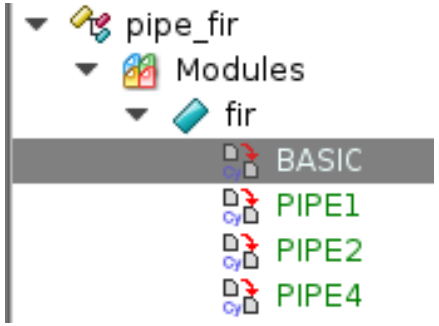


Three Genus modes are supported

Genus Basic	• Stratus license enables it
Genus PLE	• Stratus license enables it
Genus iSpatial	• Genus Synthesis (GEN100) • Genus Physical Opt (GEN40) • Innovus GigaPlace GXL Opt

Micro-architecture exploration by Stratus IDE

```
SC_MODULE(xxx)
{
   sc_in<bool> clk;
   sc_in<bool> nrst;
   cynw_p2p<my_data>::in din;
   cynw_p2p<my_data>::out dout;
   ….
   SC_CTOR() {
      ….
   }
};

….
while (1) {
#ifdef PIPE
    HLS_PIPELINE_LOOP(HARD_STALL, V_II, “pipe”);
#else
    HLS_CONSTRAINT_LATENCY(0, HLS_ACHIEVABLE, “lat”);
#endif
    idata = in.get();
    result = func(idata);
    out.put(result);
}
```

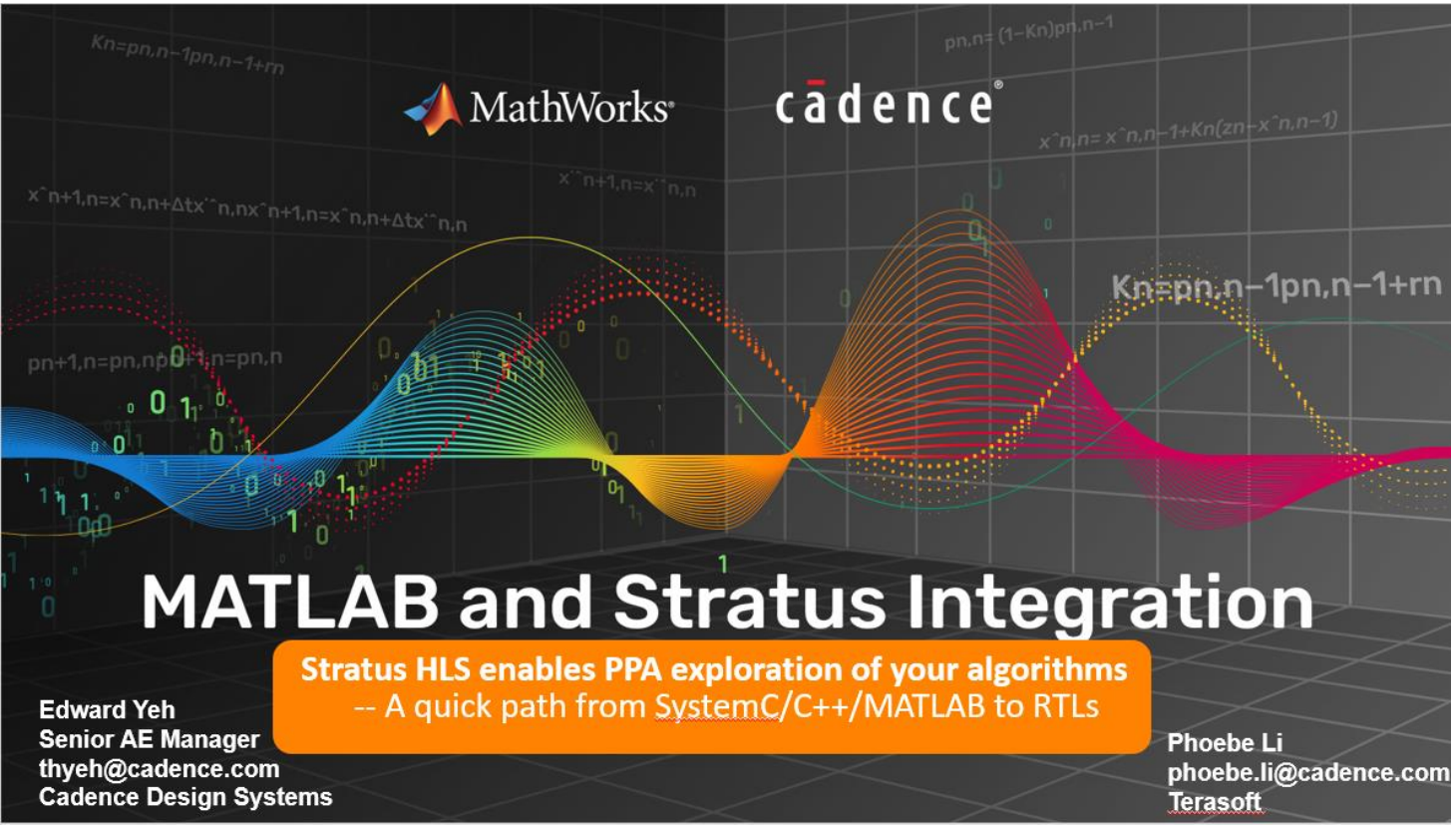



```
define_hls_config $design BASIC
define_hls_config $design PIPE1 -DPIPE -DV_II=1
define_hls_config $design PIPE2 -DPIPE -DV_II=2
define_hls_config $design PIPE4 -DPIPE -DV_II=4
```

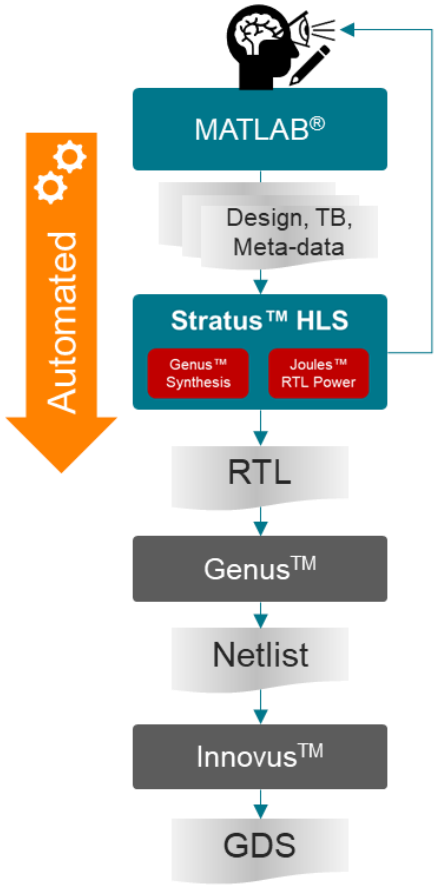
Architecture explorations

Minutes in Stratus vs. Days in RTL

Stratus HLS X MATLAB



New Flow



- Automated creation of synthesizable SystemC
- Automated import and Stratus project creation
  - Includes design and TB

Benefits

- Early PPA visibility
- Productivity force multiplier
  A single engineer produces results previously requiring a team

Stratus Synthesizable IP Functions (Highlights)

Pre-verified building blocks accelerate design and verification

Communication / Interfaces

- Simple bus
- AXI4, AXI4-Lite
- AXI3, APB
- Memories
- FIFOs
- CDCs
- Line buffer
- Stream buffer
- Circular buffer
- Point-to-point handshake
- Trig-done (with data)

Synthesizable Datatypes

- Floating point
- Fixed point
- Complex

Fixed-Point + Integer Functions

- barrelShift function
- barrelShifter class
- min / max(array)
- min_index / max_index(array)
- min / max(v1, v2)
- abs(value)
- swap(v1, v2)

Fixed-Point Functions

- Iterative fixed-point divider
- Sine / Cosine
- Normalized Sine / Cosine
- log2, exp
- atan2, tanh

Integer Functions

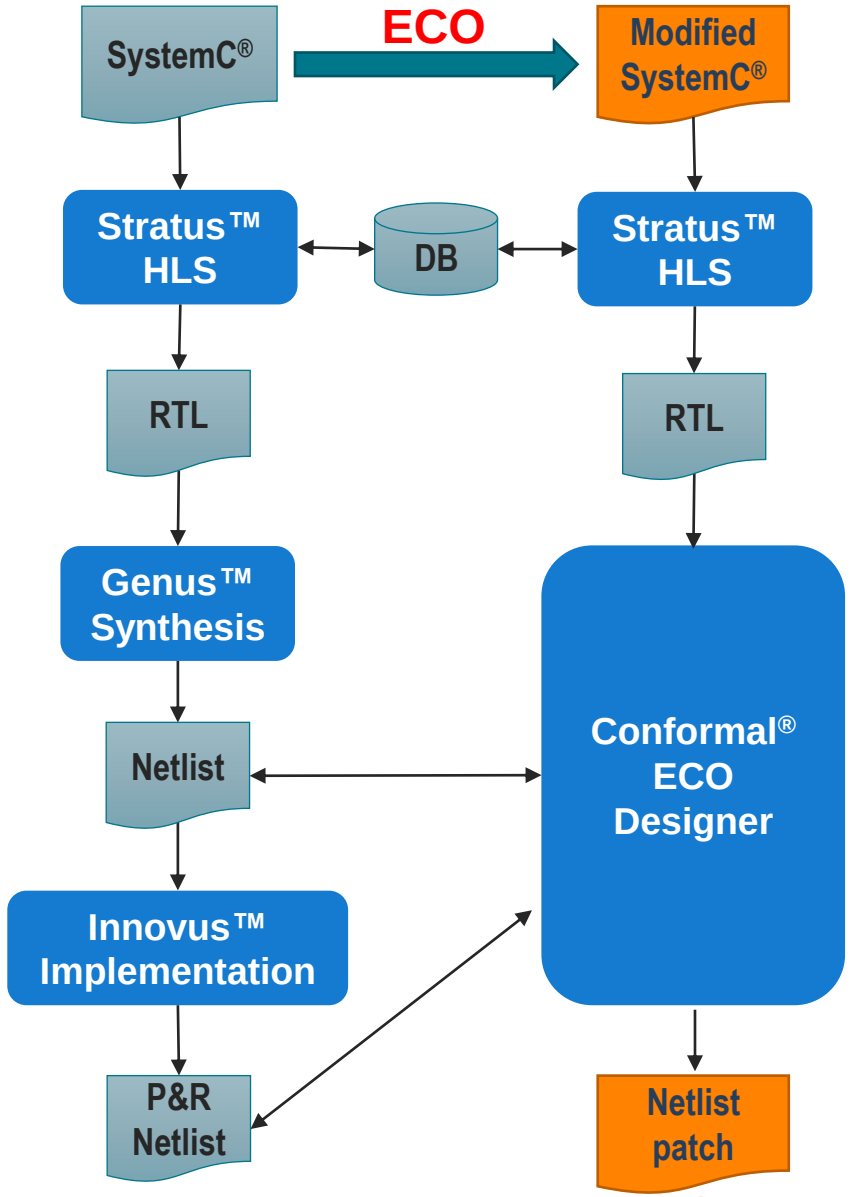
- leadingOne::fromMSB
- leadingOne::fromLSB
- iterative divider
- clip(value,min,max)
- clamp(value,min,max)

Stratus ECO Flows

- Stratus™ HLS supports two methods for ECOs

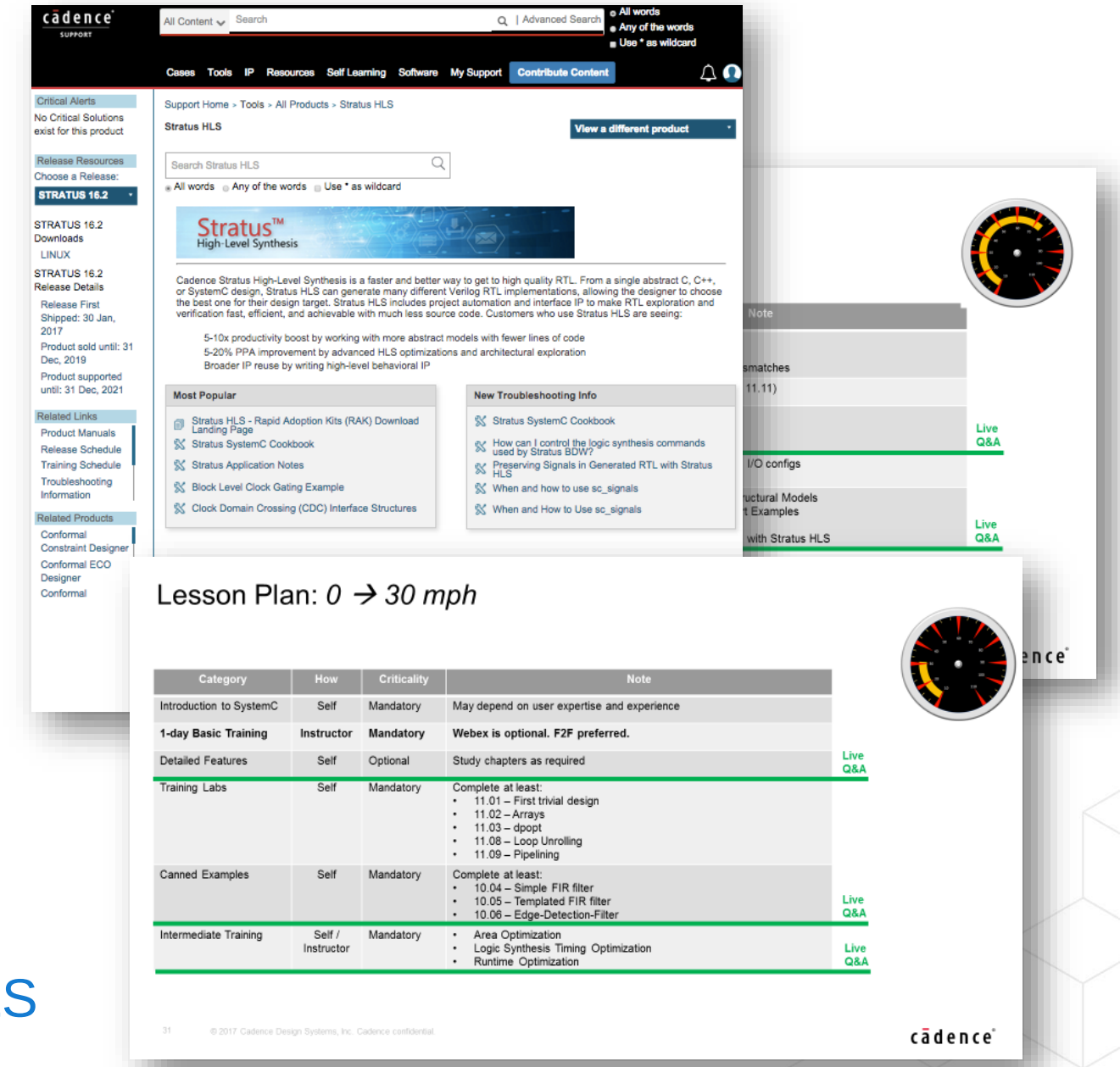
- Top-down ECO
  - Change is made in SystemC
  - Stratus ECO mode minimizes changes in RTL
  - Conformal® ECO creates netlist patch to minimize impact on implementation flow

- Bottom-up ECO
  - Change is made in netlist
  - Manual change is propagated to RTL and SystemC
  - No re-synthesis minimizes risk



Stratus HLS – Education

- Extensive web-based materials
  - Rapid Adoption Kits (RAKs)
  - Application notes
  - Tips, tricks, best practices
- Includes lesson plans
  - Based on experience and time available
- Examples and labs
  - Included in the Stratus IDE
  - Use as kick-start templates
- http://support.cadence.com/StratusHLS



Frequently-used Synthesis Constraints

Synthesis constraints & micro-architecture - 1

- Tell Stratus HLS HOW to implement your design.
  - I/O protocols
  - Pipelined or non-pipelined.
  - Latency constraint
  - Loop unrolling or not
  - Flatten array or register file, or memory

- Stratus HLS assigns synthesis constraints by DIRECTIVESs in codes or TCL commands in project.tcl
  - **Different synthesis constraints result in different RTL structures.**

Synthesis constraints & micro-architecture - 2

- HLS directive's effective scopes
  - Many HLS directives are effective within the '{ }'
  - Some directives are effective entire module when it is specified in a module constructor.

- Micro-architecture constraints
  - HLS_UNROLL_LOOP
    - Specify unrolling loops of for, while, do/while
  - **HLS_PIPELINE_LOOP**
    - Specify pipelining loops of for, while, do/while
  - **HLS_CONSTRAIN_LATENCY**
    - Constrain latency in a region within '{ }'
  - HLS_FLATTEN_ARRAY
    - Specify flattening arrays
  - **HLS_MAP_TO_MEMORY**
    - Specify mapping arrays to memories

- Timing constraints
  - **HLS_DEFINE_PROTOCOL**
    - Specify cycle-accurate regions (reset, protocol specific)
  - HLS_SET_INPUT_DELAY
    - Specify input delay for input ports or sc_signals
  - **HLS_ASSUME_STABLE**
    - Specify input ports or sc_signals to be treated as stable.

I/O Protocols

- Stratus HLS will inject cycles to meet performance requirements and minimize area
  - “free scheduling”
- However, all designs communicate with their surrounding modules in a cycle accurate manner
  - “fixed scheduling”
- Example: an array must be populated by 8 input port reads in 8 consecutive cycles
  - The wait() inside the ‘for’ loop will create this timing

Argument is a label used for reporting

Enclose the protocol code in braces

```
{
  HLS_DEFINE_PROTOCOL( “read_protocol” );
  for( int i = 0; i < 8; i++ ) {
    MEM[i] = inp.read();
    wait();
  }
}
```

HLS_DEFINE_PROTOCOL() tells Stratus HLS that the schedule for this block of code is fixed.

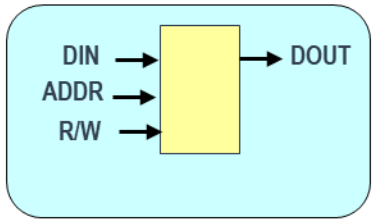
Any wait() inside the block will always become one clock cycle

Arrays, Storage, and Memories

- Stratus HLS can implement arrays in the following hardware styles

```
SC_MODULE( dut )
{
  …
  sc_uint<8>  mem[256];

  void thread0() {
       …
       a = mem[1] + mem[2];
    …
  }
}
```

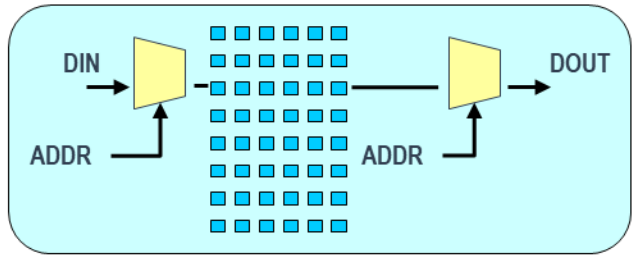
HLS_MAP_TO_MEMORY



This is the default

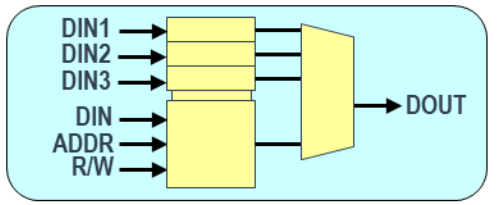
Memory

HLS_FLATTEN_ARRAY



Flattened into a set of individual variables

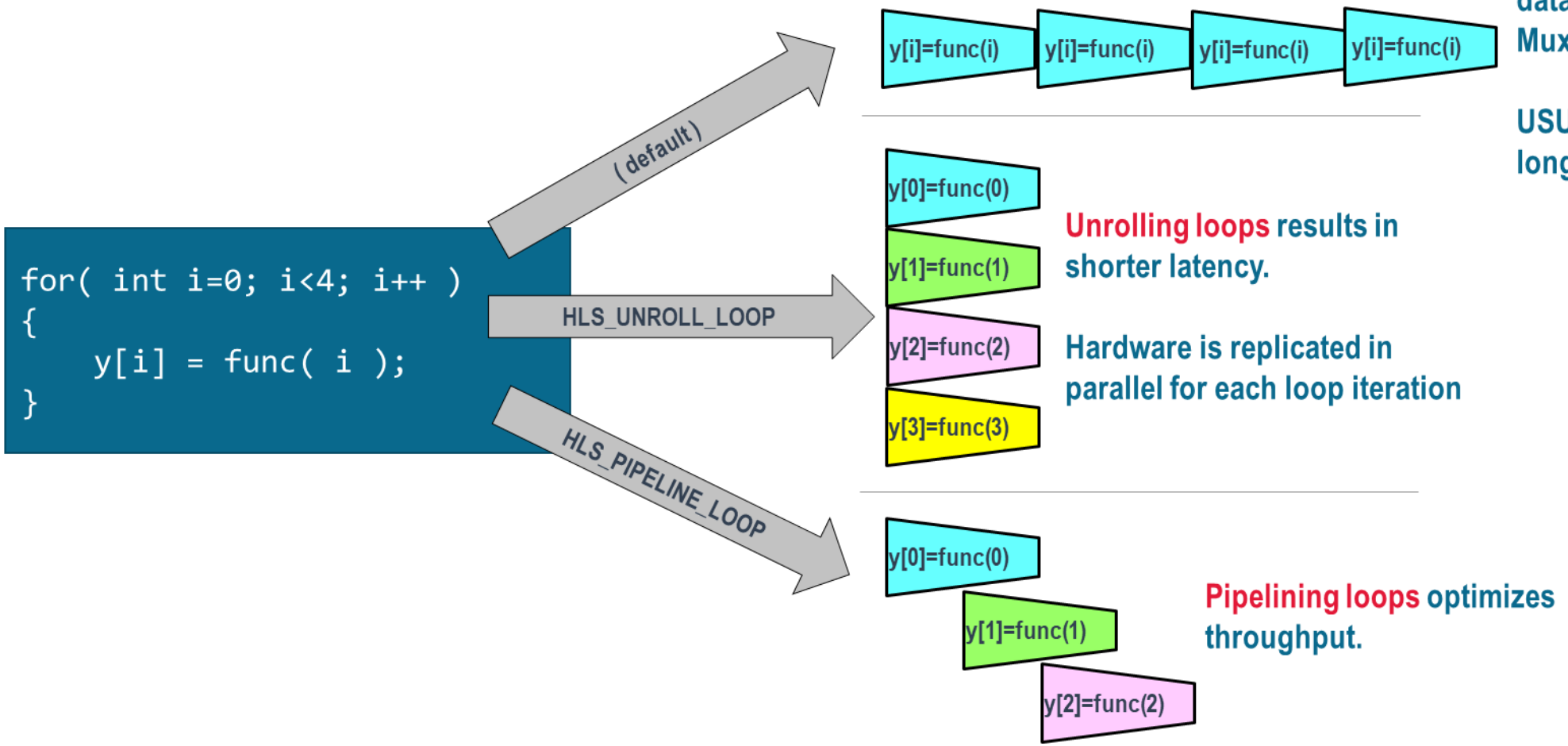
HLS_MAP_TO_REG_BANK



A register bank

Loops - 1

- Stratus HLS can implement loops with the following hardware



In rolled up loops datapath is one iteration of loop. Muxing shares hardware.

USUALLY, smallest hardware and longest latency

Unrolling loops results in shorter latency.

Hardware is replicated in parallel for each loop iteration

Pipelining loops optimizes throughput.

Loops - 2

- The loops can be specified with 3 types of the micro-architectures.
  - Non-unrolled and non-pipelined
  - Unrolled by HLS_UNROLL_LOOP

```
HLS_UNROLL_LOOP( ON, "unroll" );
```

User-specific label

type : ON, ALL, OFF, COMPLETE, AGGRESSIVE, CONSERVATIVE

  - Pipelined by HLS_PIPELINE_LOOP

User-specific label

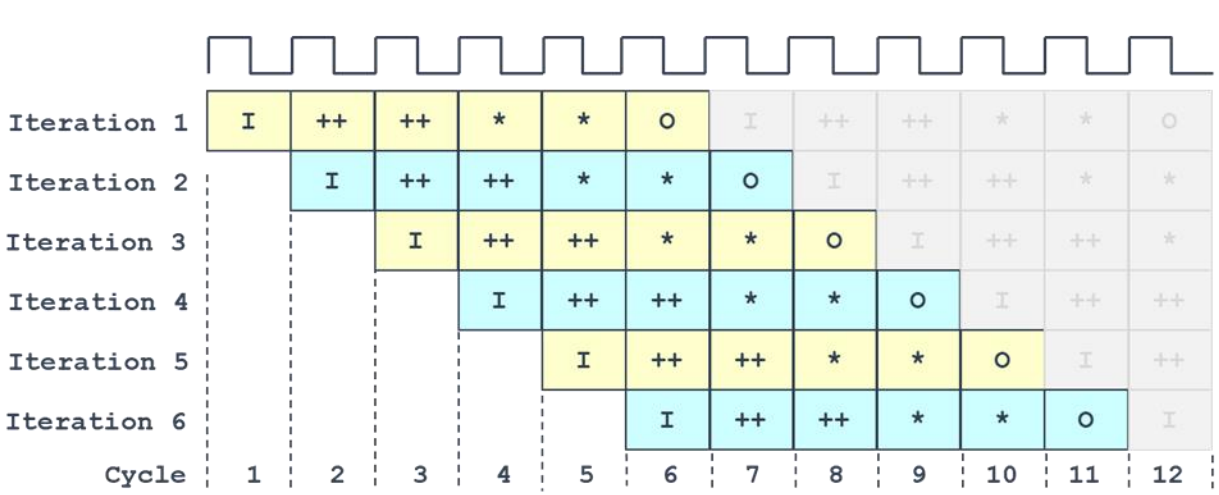
```
HLS_PIPELINE_LOOP( HARD_STALL, 1, "main_pipeline" );
```

Pipelined type: HARD_STALL or SOFT_STALL

Initiation Interval(II) for specifying the data throughput.

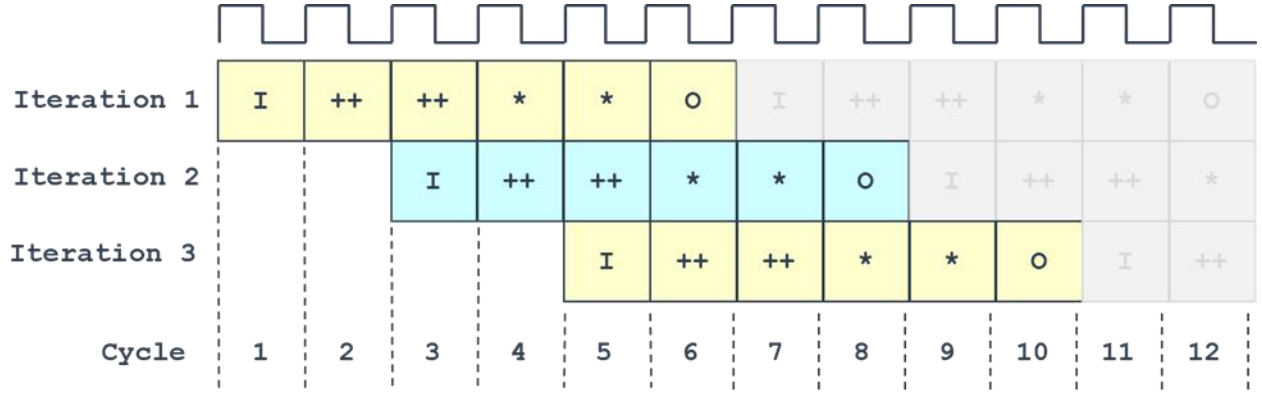
Loop Pipelining

- Different initiation interval (II) results in different pipeline architecture.



Resources	4 Adders, 2 Multipliers
Latency	6 clock cycles
Throughput	1 I/O per 1 clock cycles

Fully pipelined for a 6x improvement in throughput and only 2x increase in resources!



Resources	2 Adders, 1 Multiplier
Latency	6 clock cycles
Throughput	1 I/O per 2 clock cycles

Throughput tripled with no increase in resources!

Latency Constraint

- It can be used on any region bounded with {..}
- Syntax
  HLS_CONSTRAIN_LATENCY(min_lat, max_lat, “user-specific label”)
- It can also be used to specify the latency of the pipelined loop.

```
while (true) {
   HLS_CONSTRAIN_LATENCY(1, 10,"main_loop");
   d = din.read();
   dout.write( d+1 );
   wait();
}
```

The label can be used in the latency report.

Min latency

Max latency

```
while (true) {
  HLS_PIPELINE_LOOP(HARD_STALL, 1,"main_loop");
  HLS_CONSTRAIN_LATENCY(1, 10,"main_loop");
   d = din.read();
   dout.write( d+1 );
   wait();
}
```

The label can be used in the latency report.

Min latency

Max latency

Typical SC_MODULE with constraints

```
#include "DUT.h"

void DUT::proc() {
  {HLS_DEFINE_PROTOCOL("reset");
    OUT.write(0);
    wait();
  }
  while(true) {
    HLS_PIPELINE_LOOP(HARD_STALL,1,"my_pipe");
    HLS_CONSTRAIN_LATENCY(0,HLS_ACHIEVABLE,"my_lat");
    HLS_ASSUME_STABLE(IN_MODE,"IN_MODE");
    sc_uint<8> temp1, temp2;
    sc_uint<32> sum;
    sc_uint<2> mode;

    {HLS_DEFINE_PROTOCOL("input");
      temp1 = IN_A.read();
      temp2 = IN_B.read();
      mode =  IN_MODE.read();
    }

    if (mode==1)
       sum = temp1 + temp2 + mode * 11;
    else if (mode==2)
       sum = temp1 - temp2 + mode * 22;

    {HLS_DEFINE_PROTOCOL("output");
      OUT.write(sum);
      wait();
    }
  }
}
```

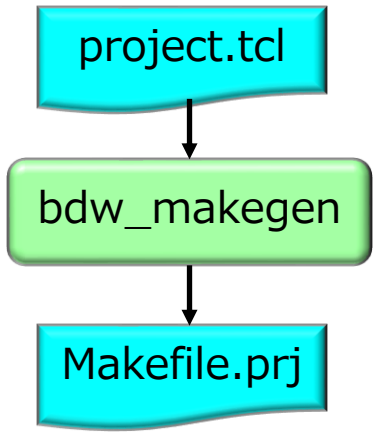
Project.tcl

Makefile

- Makefile should include a Makefile.prj which will be generated by Stratus' bdw_makegen command based on project.tcl
- The following is the minimum settings for running Stratus.

Makefile

```
-include Makefile.prj

Makefile.prj : project.tcl
        @bdw_makegen project.tcl
```



Typical project.tcl - 1

```
#*********************************************
# Copyright 2015 Cadence Design Systems, Inc.
# All Rights Reserved.
#*********************************************

#
# Libraries
#
set LIB_PATH     "[get_install_path]/share/stratus/techlibs/ \
                  GPDK045/gsclib045_svt_v4.4/gsclib045/timing"
set LIB_NAME     "slow_vdd1v2_basicCells.lib"
use_tech_lib     "$LIB_PATH/$LIB_NAME"

#
# C++ compiler options
#
set CLOCK_PERIOD "5.0"
set_attr cc_options       " -g -DCLOCK_PERIOD=$CLOCK_PERIOD"
```

Technology library: using a dummy 45nm technology library

Compilation options

Typical project.tcl - 2

```
#
# stratus_hls options
#
set_attr clock_period                   $CLOCK_PERIOD
set_attr balance_expr                   delay
set_attr default_input_delay            0.1
set_attr default_stable_input_delay     0.0
set_attr default_protocol               false
set_attr dpopt_auto                     op (or op,expr)
set_attr flatten_arrays                 all
#set_attr ignore_cells                   "XYZ* ABC*"
set_attr message_detail                 2
set_attr output_style_reset_all         yes
set_attr path_delay_limit               120
set_attr power                          on
set_attr unroll_loops                   on
set_attr wireload                       none
```

Typical project.tcl - 3

```
#
# Simulation Options
#
use_systemc_simulator           built-in/incisive/xcelium
use_verilog_simulator           incisive/xcelium/vcs
enable_waveform_logging         -vcd/-shm/-fsdb

#
# System Module Configurations
#
define_system_module    main      main.cc
define_system_module    system    system.cc
define_system_module    tb        tb.cc

#
# Synthesis Module Configurations
#
define_hls_module       dut       ./src/DUT.cpp
define_hls_config       dut       BASIC
```

Specify simulators

Specify testbench

Specify target HLS module with configurations

Typical project.tcl - 4

```
#
# Simulation Configurations
#
define_sim_config B            {dut BEH}
define_sim_config BASIC_V     {dut RTL_V BASIC}

#
# Logic Synthesis Configurations
#
set_logic_synthesis_options     [list BDW_LS_TECHLIB $LIB_PATH/$LIB_NAME]
{BDW_LS_DO_DISSOLVE 1} {BDW_LS_NOGATES 1}
define_logic_synthesis_config   L {dut -all} -command bdw_rungenus
```

Specify simulation configurations

Specify logic synthesis configurations