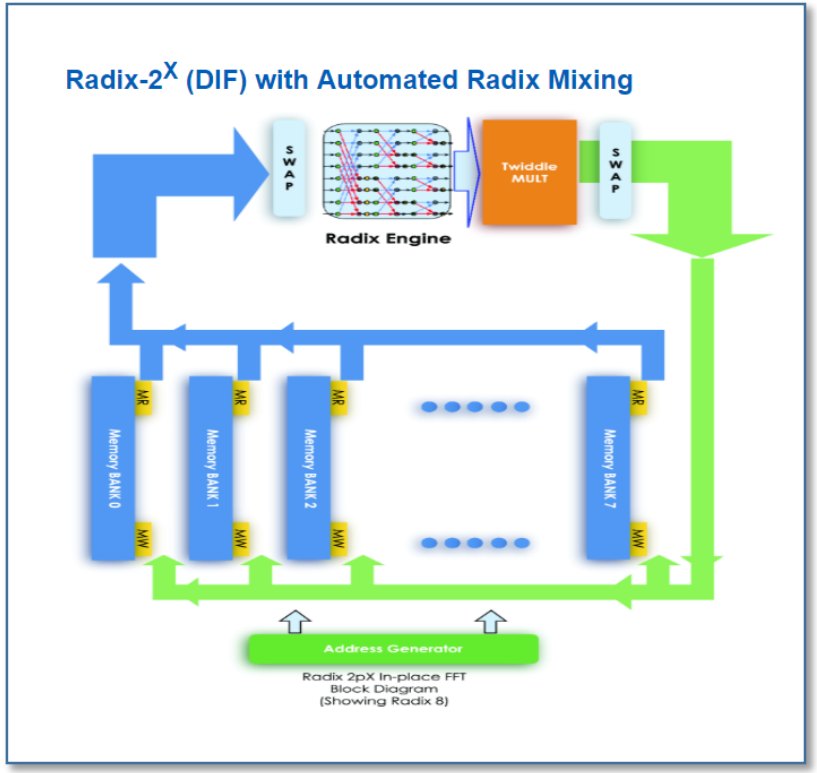
Efficient Verification of High-Level Synthesis IP

David Aerne Gagandeep Singh Deepak Mehta

Mentor®
A Siemens Business

Motivation

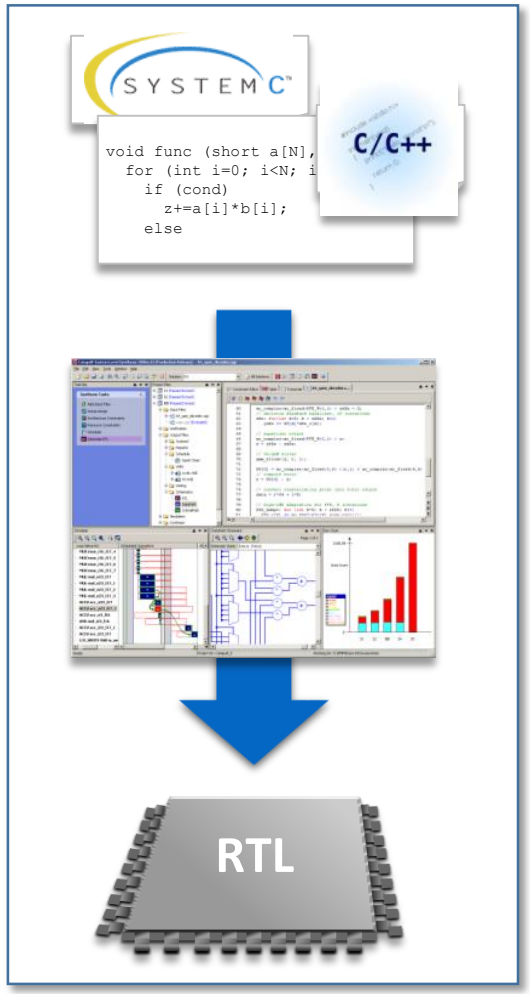
- Adoption of High-Level Synthesis (HLS) continues to grow
- Yet with continued need for “off-the-shelf” IP
- Libraries of HLS-ready C++/SystemC IP building blocks, e.g.
  - Math, linear algebra and trig functions
  - CORDIC algorithms
  - DSP operations including FIR filters and FFTs
  - Windowing classes to efficiently implement 1-d and 2-d algorithms
- Meeting the following requirements:
  - Optimized for Performance/Power/Area
  - Easy to use/configure/customize
  - High-Level Synthesis-ready
  - Proven



Need to efficiently verify these highly-configurable HLS IP blocks

High-Level Synthesis:
Design, Optimize and Verify at a Higher Level of Abstraction

- Generate high quality RTL from C++/SystemC level descriptions
  - Micro-architecture exploration is accelerated
    - Parallelism, Throughput, Latency, Area (loop unrolling and pipelining)
    - Memories vs. Registers (resource allocation)
    - Integrated Power estimation
  - Library of IP components, e.g. math, DSP, video algorithms
  - ASIC and FPGA target support
- Verify at a higher level of abstraction (HLS-ready C++ source)
  - Perform Formal checks prior to synthesis
  - Simulate 50-500x faster
  - Achieve Code and Functional coverage goals
- Post RTL generation verification and optimization
  - SLEC HLS to formally prove C++ model and RTL equivalency
  - Integrated Power Optimization



Verify C++ Design IP at higher level of abstraction

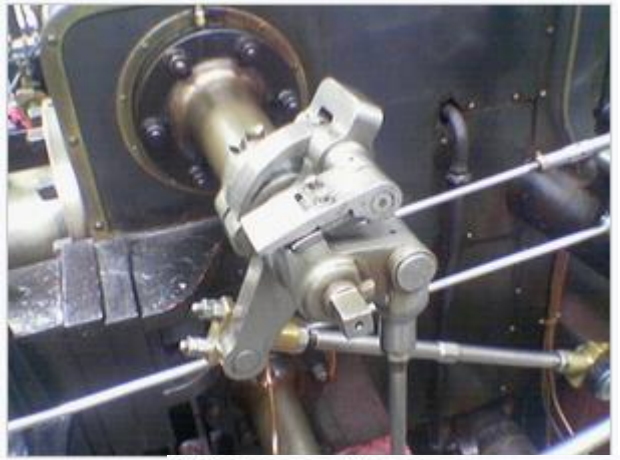
- Verify the HLS-ready C++ design IP
  - Prior to generation of RTL
- Within C++ testbench
  - C++ sim 50-500x faster than RTL simulation
- Using trusted RTL verification methodologies
  - PSS for re-targetable efficient CR stimulus
  - Assertion Based Verification
  - HLS-aware Code Coverage
  - Black-box Functional Coverage
  - Coverage database for viewing, merging, exclusions, etc.
  - Integration with Test/Verification Plan

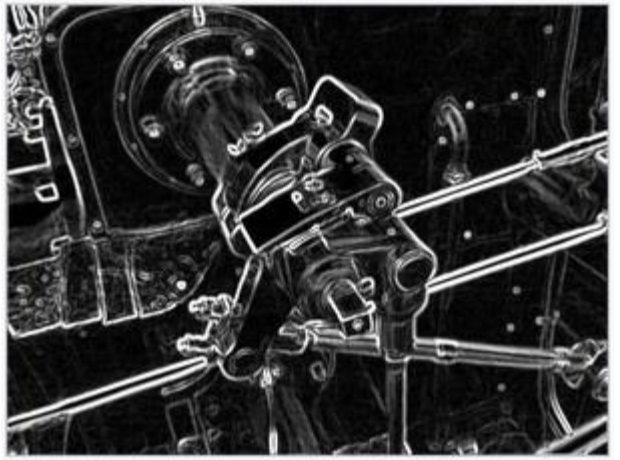
```
17 void sobel(int din[GRID][GRID], ac_channel<int> &dou
18
19 #ifdef CLIP
20   static ac_window_2d_stream<int,WN_SZ,WN_SZ,GRID,GRI
21 #elif defined MIRROR
22   static ac_window_2d_stream<int,WN_SZ,WN_SZ,GRID,GRI
23 #elif defined WIN
24   static ac_window_2d_stream<int,WN_SZ,WN_SZ,GRID,GRI
25 #elif defined BOUNDARY
26   static ac_window_2d_stream<int,WN_SZ,WN_SZ,GRID,GRI
27 #else
28 #endif
29   int Gx,Gy;
30   int tmpx,tmpy;
31
32   w.write(&din[0][0],row_sz,col_sz);    //write all
33
34   for(int i=0;i<GRID;i++){
35     for(int j=0; j<GRID;j++){
36       w++;
37       tmpx = tmpy = 0;
38       if(w.valid()){
39         for(int r=0;r<WN_SZ;r++){
40           for(int c=0;c<WN_SZ;c++){
41             tmpx += w(r-WN_SZ/2,c-WN_SZ/2)*cx[r][c];
42             tmpy += w(r-WN_SZ/2,c-WN_SZ/2)*cy[r][c];
43           }
44         }
```

2-D Convolution Windowing IP: Sobel Filter

```
ac_window_2d_stream<dType,K_SZ,K_SZ,NCOL,NROW,BC_MODE> window;
ac_flags_gen<NCOL,NROW> flags;

FRAME:do{
  if(window.canRead())
    data_in = input.read();

  flags.generate(data_in.TUSER, data_in.TLAST,sof,eof,sol,eol);

  window.slide_window(data_in,sof,eof,sol,eol);

  if(window.isValid()){
    KERNEL_Y:for(int i=0;i<K_SZ;i++){
      KERNEL_X:for(int j=0;j<K_SZ;j++){
        acc += window[i][j] * kernel[i][j];
    data_out.write(acc);
  }
}while(!window.eof)
```

Templated window IP class





Verify the Windowing IP over the various template parameters:
Data Type, Kernel Size, Image Size and Boundary Condition

* *Pictures courtesy of Simpsons contributor, Creative Commons Attribution*

Measure and Close Coverage on C++ Design IP

- Test/Verification plan
  - Integration with coverage data
  - Measure progress towards coverage goals
- HLS-Aware Code Coverage
  - Statement, Branch, FEC, Toggle
  - Synthesis aware including implications of function inlining and loop unrolling on resultant RTL
- Black-box Functional Coverage
  - Define desired Covergroups, Coverpoints, Bins and Crosses
  - Specify when to sample
- Collect data in coverage database
  - Provides for viewing, merging, ranking, exclusions

```
typedef enum ac_window_mode { AC_CLIP       = 1<<0,
                              AC_MIRROR     = 1<<1,
                              AC_WIN        = 1<<2,
                              AC_BOUNDARY   = 1<<3
                            } MyBCType;

CCOV_COVERGROUP_BEGIN( MyCCoverGroup_3, MyBCType, var1,
{
  // Coverpoint on ac_window_mode enum describing desir
  CCoverPoint<MyBCType>* cp1 = AddCCoverPoint( "cp_Boun
  cp1->AddFullRangeBin("FullRangeBin", (MyBCType)1, (My

  // Coverpoint on data values having mix of PointBins
  CCoverPoint< dType >* cp2 = AddCCoverPoint("cp_data",
  cp2->AddPointBin("PointBin_min", 0 );
  cp2->AddRangeBin("RangeBin_low", 1, 16383 );
  cp2->AddPointBin("PointBin_max", 16384 );

  // Coverpoint on image size using PointBins
  CCoverPoint< int >* cp3 = AddCCoverPoint("cp_ImageSiz
  cp3->AddPointBin("PointBin_min", 4 );
  cp3->AddPointBin("PointBin_mid", 100 );
  cp3->AddPointBin("PointBin_max", 480 );

  // define desired crosses based on above coverpoints
  AddCCoverCross("cross_cp1_cp2_cp3", cp1, cp2, cp3);
}
CCOV_COVERGROUP_END
```

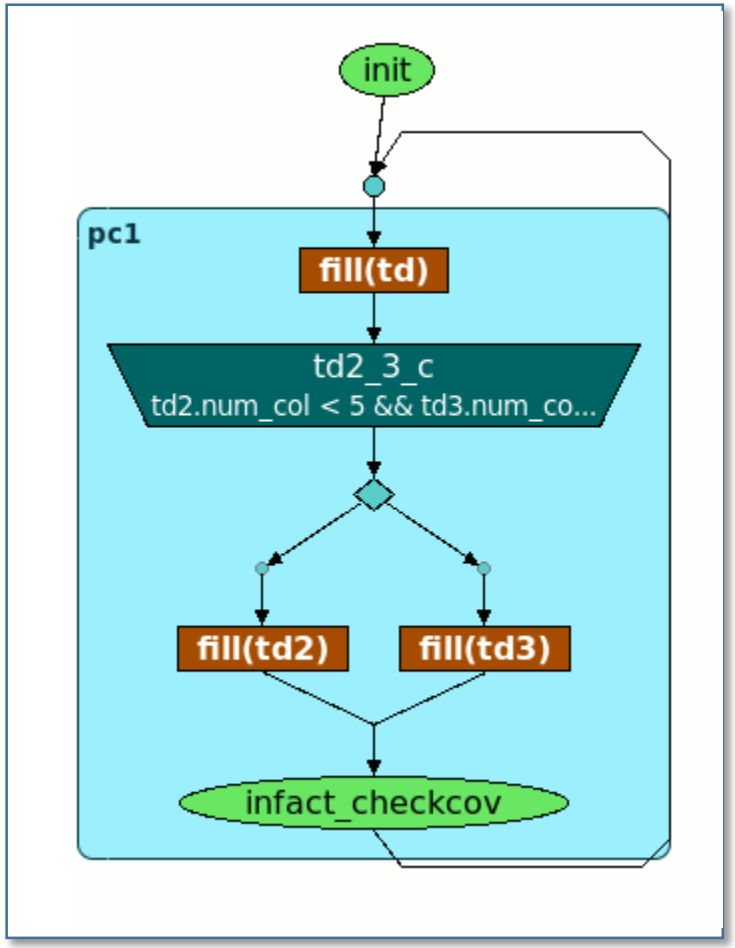
Develop Test Plan for 2-D Windowing IP

- Coverage Closure flow
  - Excel Add-In aids in the creation of xml-based Test (or Verification) Plan
  - Sections and Entries derived from Requirements or Product Specification
  - Links to desired cover items: CoverGroups, CoverPoints (bins), Crosses, etc.
  - Export XML to UCDB for merging with coverage results

Section	Title	Description	Link	Type	Weight	Goal
1	covergroup	MyCCoverGroup_3_inst	MyCCoverGroup_3,MyCCoverGroup_3_inst	CoverGroup	0	100
2	Image Size				1	100
2.1	cp1	covers Image Size	MyCCoverGroup_3,MyCCoverGroup_3_inst:cp_ImageSize	CoverPoint	1	100
3	Boundary Condition				1	100
3.1	cp2	covers Boundary Cond	MyCCoverGroup_3,MyCCoverGroup_3_inst:cp_BoundaryCondition	CoverPoint	1	100
4	data				1	100
4.1	cp3	covers data range	MyCCoverGroup_3,MyCCoverGroup_3_inst::cp_data	CoverPoint	1	100
5	crosses				1	100
5.1	cp1Xcp2Xcp3	cross of cp1 cp2 cp3	MyCCoverGroup_3,MyCCoverGroup_3_inst:cross_cp1_cp2_cp3	Cross	1	100

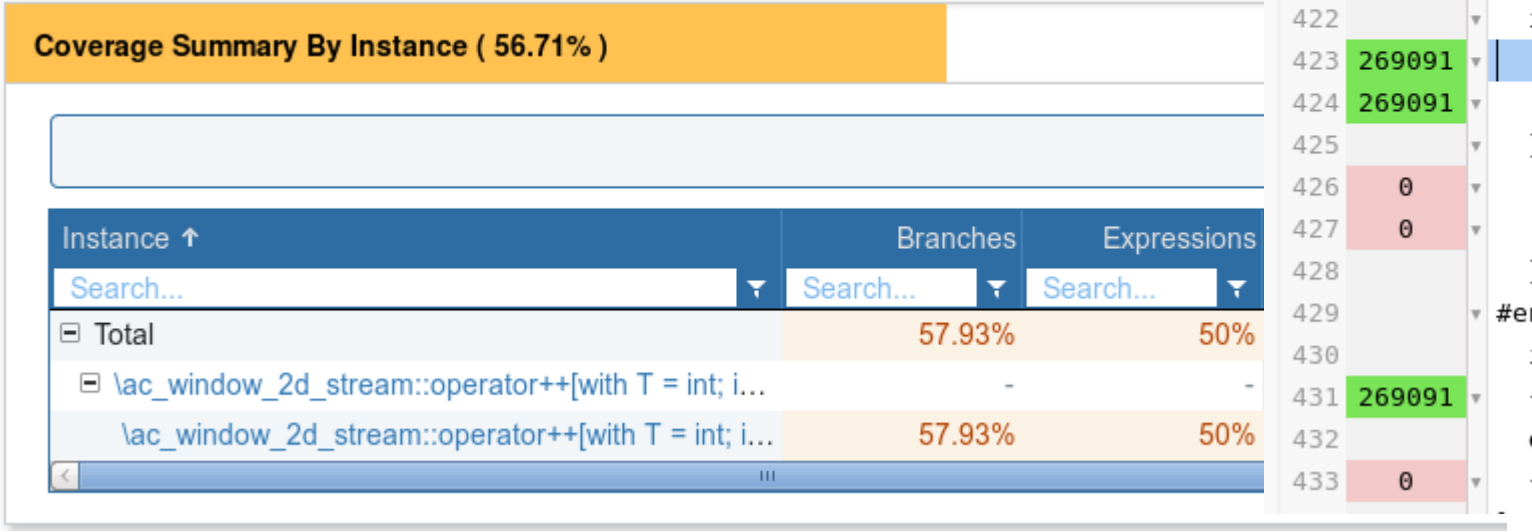
Portable Stimulus

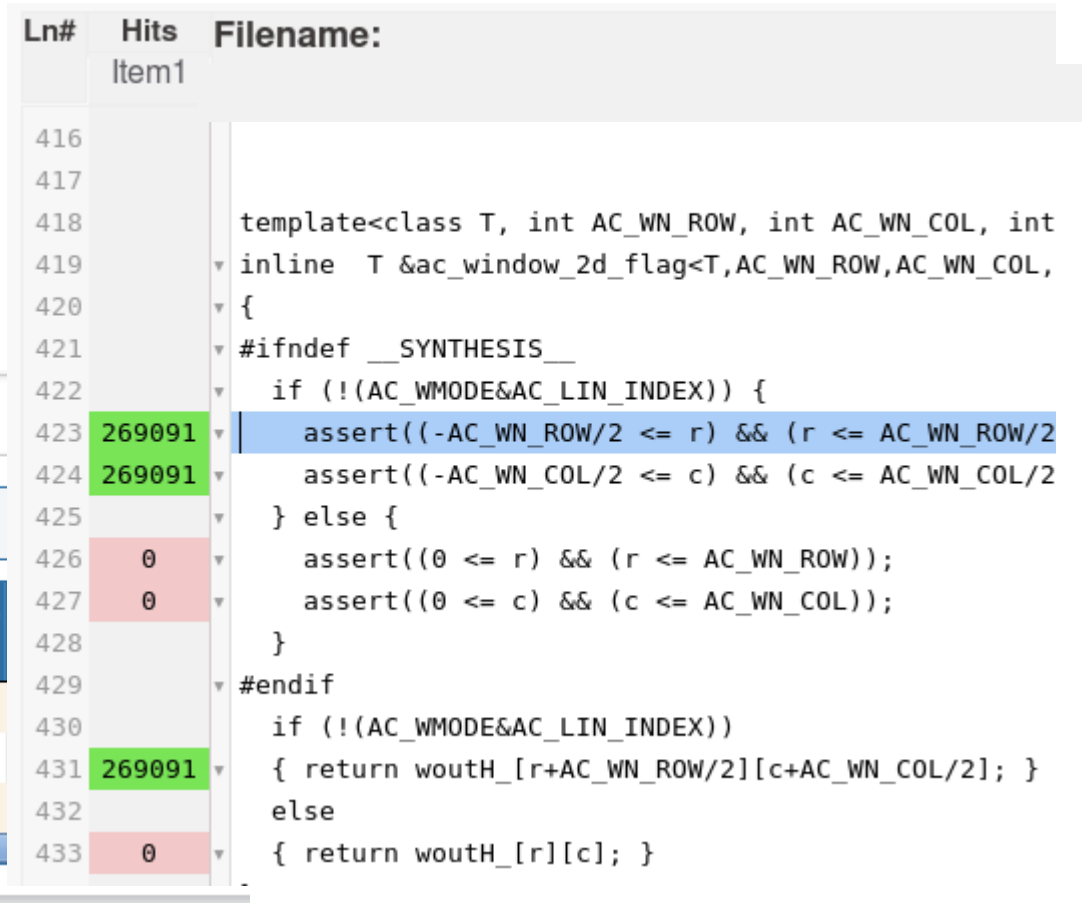
- Rule/graph based stimulus model
  - Variables, constraints, sequences
- Coverage strategy prioritizes goals
  - Coverpoints, crosses, scenarios
  - Improved coverage vs. directed or random
- Model is independent of environment
  - Can be wrapped in C, C++, SystemC, SV, etc.
  - Maintain random stability between environments



Measure and close coverage on 2-D Windowing IP

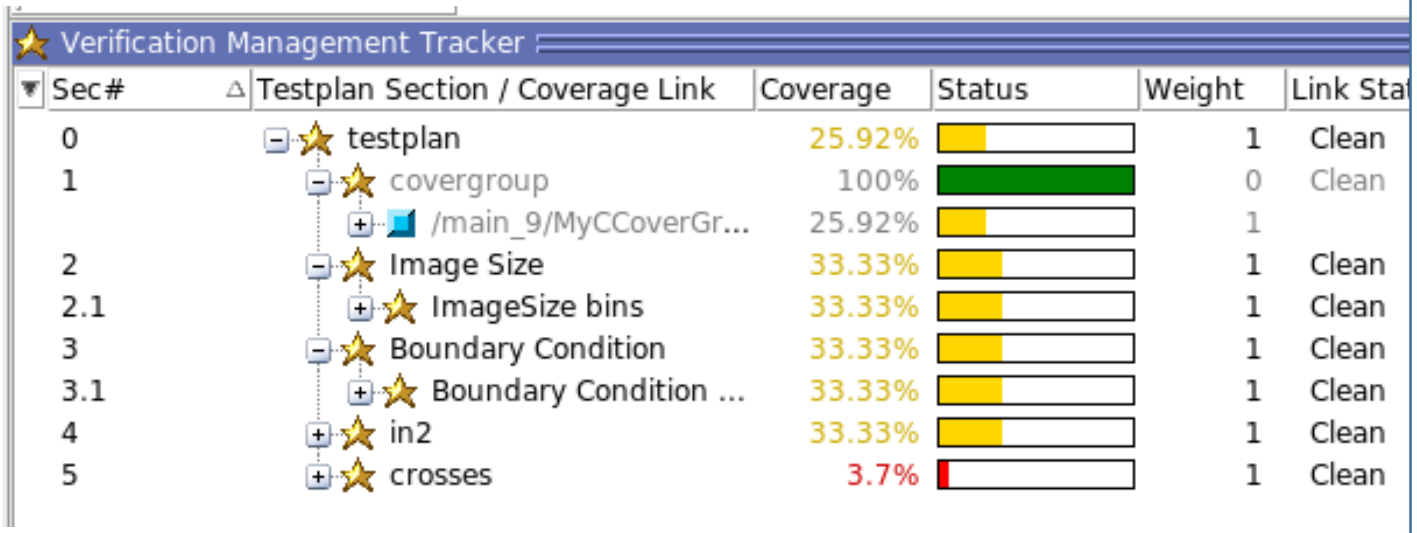
- Coverage Closure flow
  - Measuring code coverage C++ design source
    - Statement, branch, FEC, toggle
  - Adding tests as necessary
  - Merging coverage results
  - Applying desired exclusions





Measure and close coverage on 2-D Windowing IP

- Coverage Closure flow
  - Defining and Measuring functional coverage
  - Specifying when to sample data
  - Merging functional coverage results with Test Plan
  - Add more tests as required

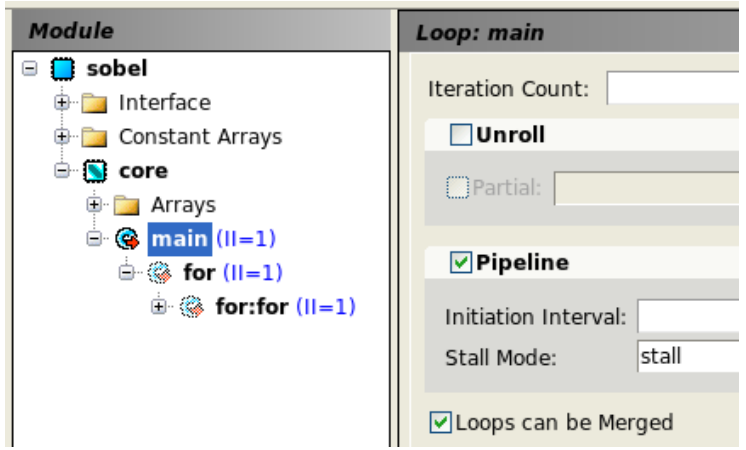



```
typedef enum ac_window_mode { AC_CLIP       = 1<<0,
                              AC_MIRROR     = 1<<1,
                              AC_WIN        = 1<<2,
                              AC_BOUNDARY   = 1<<3
                            } MyBCType;

CCOV_COVERGROUP_BEGIN( MyCCoverGroup_3, MyBCType, var1,
{
  // Coverpoint on ac_window_mode enum describing desir
  CCoverPoint<MyBCType>* cp1 = AddCCoverPoint( "cp_Boun
  cp1->AddFullRangeBin("FullRangeBin", (MyBCType)1, (My

  // Coverpoint on data values having mix of PointBins
  CCoverPoint< dType >* cp2 = AddCCoverPoint("cp_data",
  cp2->AddPointBin("PointBin_min", 0 );
  cp2->AddRangeBin("RangeBin_low", 1, 16383 );
  cp2->AddPointBin("PointBin_max", 16384 );

  // Coverpoint on image size using PointBins
  CCoverPoint< int >* cp3 = AddCCoverPoint("cp_ImageSiz
  cp3->AddPointBin("PointBin_min", 4 );
  cp3->AddPointBin("PointBin_mid", 100 );
  cp3->AddPointBin("PointBin_max", 480 );

  // define desired crosses based on above coverpoints
  AddCCoverCross("cross_cp1_cp2_cp3", cp1, cp2, cp3);
}
CCOV_COVERGROUP_END
```

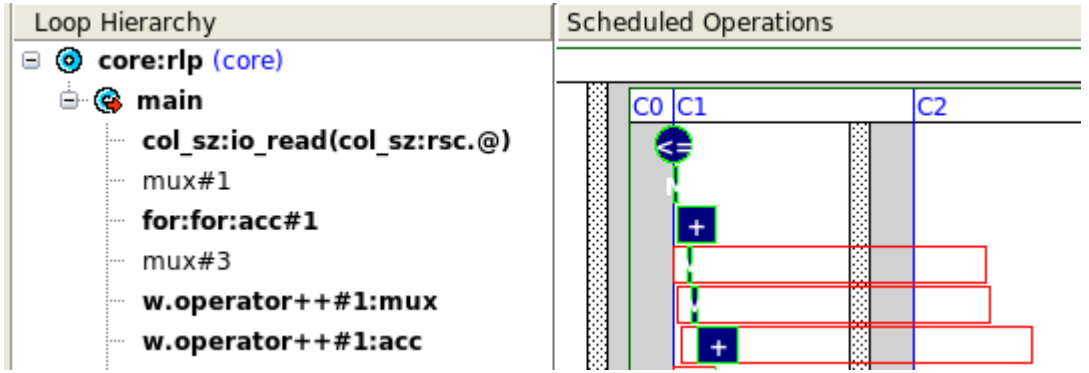
Synthesize the HLS C++

- Specify IP template parameters
  - Pixel Data Type
  - Image Size
  - Window Size
  - Boundary Condition
- Specify Synthesis constraints/directives
  - Latency and Throughput
  - Interface Synthesis
  - Memory Interfaces
  - Target Technology
- HLS to create the RTL
  - Sanity check using existing C++ tb

Mapping Constraints



Scheduled Operations

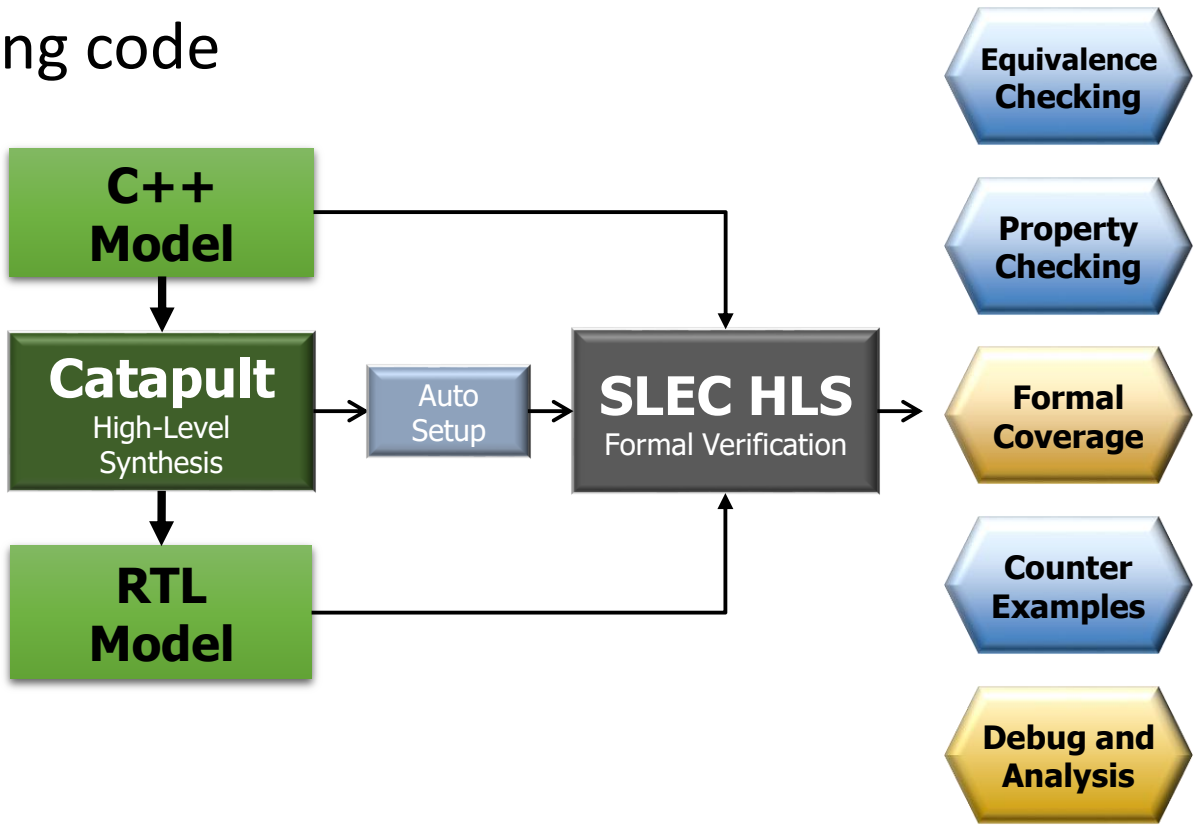


SLEC HLS for Formal C++ to RTL

- Sequential Logical Equivalence Checking and proof coverage technology

- SLEC "Advisor" points to non-converging code
  - Addresses the "all or nothing" problem in formal flows
  - Full proof vs. bounded proof

- Property Checking
  - Un-initialized memory reads
  - Out of bounds reads and writes
  - Accumulator of native C type



C++ Design IP efficiently verified and ready for (re-)use!

- IP creation and re-use important component of HLS
  - Implemented in C++/SystemC
  - Easily and efficiently incorporate desired functionality/algorithm into design
- Verify these configurable IP blocks using known and trusted RTL methodologies
- Yet realize productivity gains by using High-Level Verification (HLV) techniques
- Providing for efficient verification at the C++ level