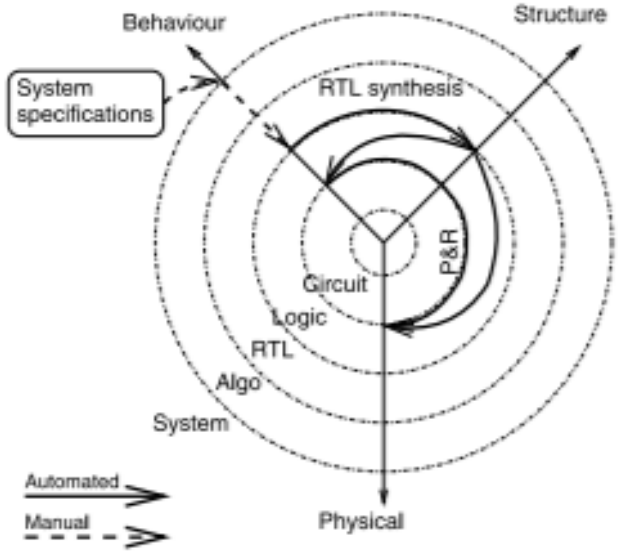
Internship:
High Level Synthesis

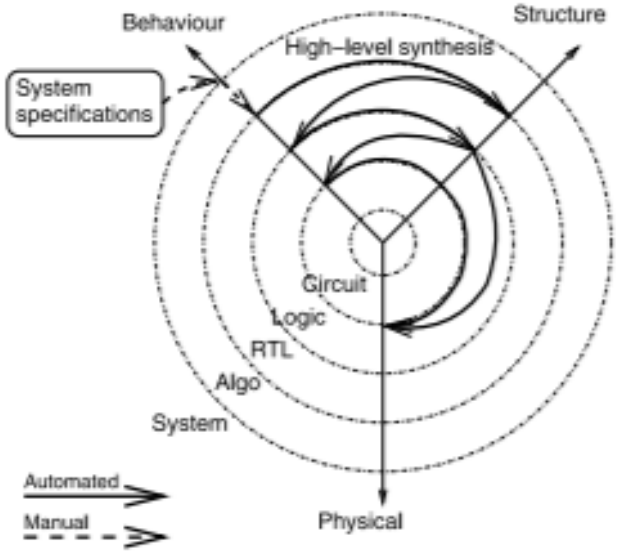
By Wouter Van Gansbeke

Supervisors: Wim Meeus & Bertrand Dujardin

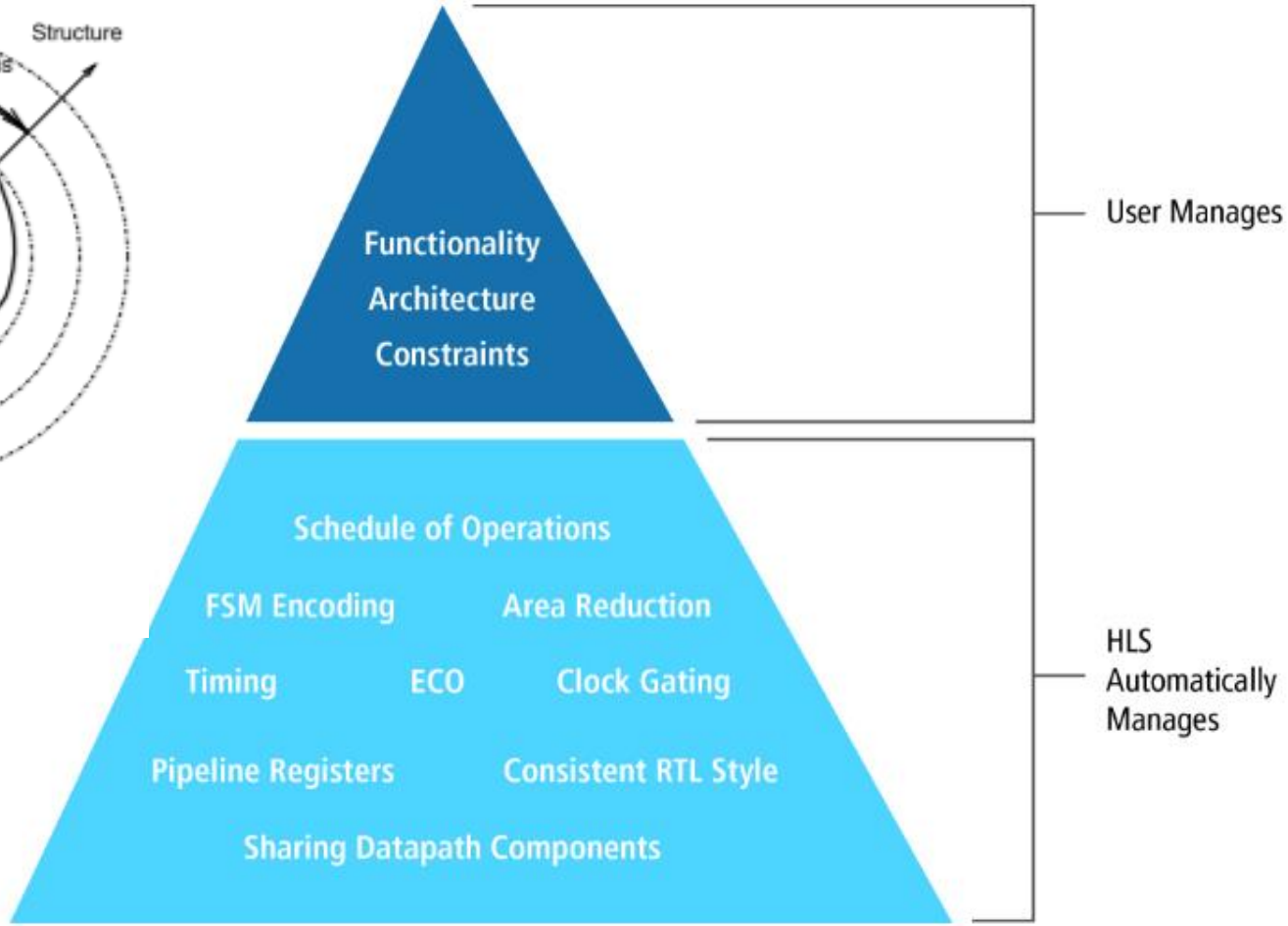
High Level Synthesis: Introduction



(a) RTL design flow



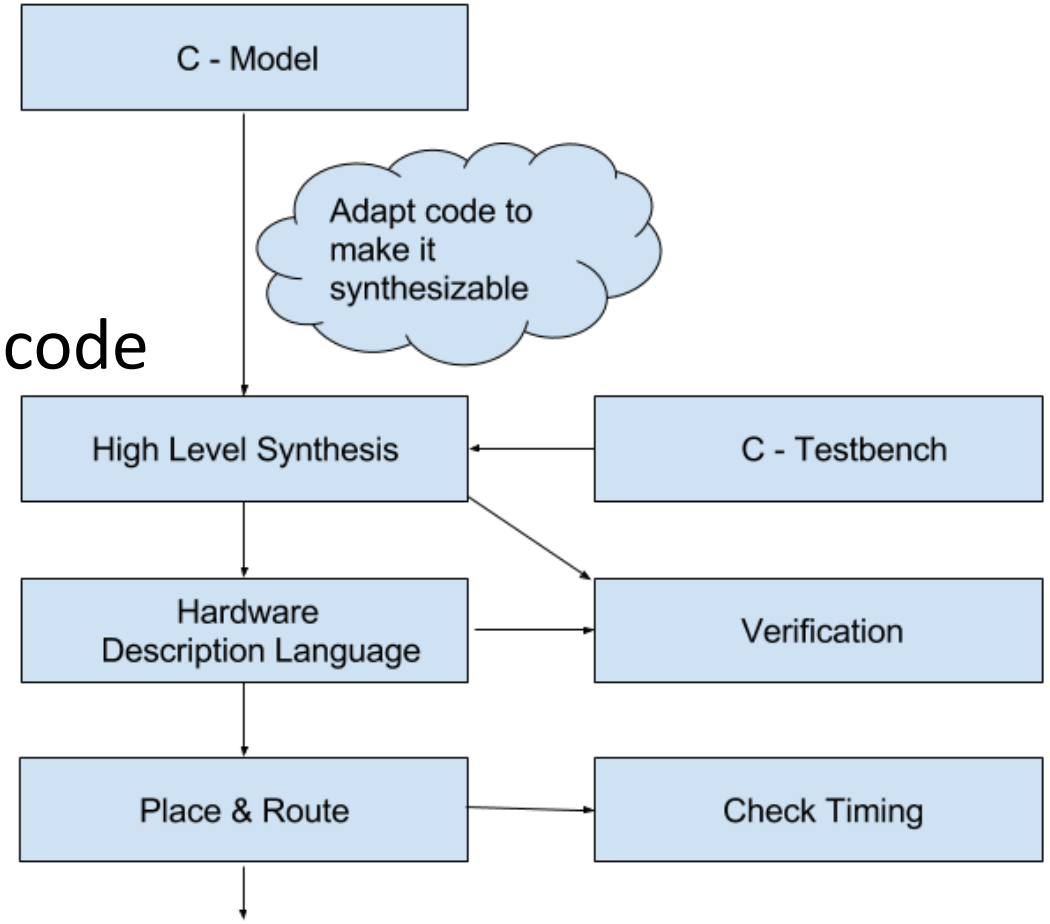
(b) HLS design flow



High Level Synthesis: Workflow

Main steps:

- Write C / C++ code and test bench
- Verify functional behavior of the C / C++ code
- Specify top level function, platform, clock, constraints and directives
- Synthesize C / C++ to RTL
- Verify RTL using the C test bench



High Level Synthesis: Benefits

Benefits:

- Accelerate design time
- Reduce verification / simulation effort and time
- Easy adaptation to other design requirements (reuse)
- Closer to algorithm, less code and by extend less errors
- Detect errors early in the design process
- Locate bottlenecks fast
- => Productivity increase and time to market decrease!

High Level Synthesis: Pitfalls

Pitfalls:

- No clock accurate designs possible
- Some software difficult / impossible to convert to hardware
- It is not possible to use existing RTL code as an input for HLS software
- Hard to read generated RTL code
- Cause of errors not always clear
- Maturity of the tool

From C model to synthesizable C (I)

Code changes:

- Top level function cannot be a class member function
- C constructs must be of bounded size
  - 'unsupported memory access on variable'
- Pointer to pointer in top level not possible:
  - Matrices
  - Pointer from within a class to other object
- Pointer casting
- No recursive functions (tail recursion)
- Arbitrary Precision data types to optimize word length
- No dynamic memory allocation
- No memset, memcopy

From C model to synthesizable C (II)

- Encoutered cosimulation problems:
  - 'Stuck at TB testing' or 'Segmentation Fault Error' caused by:
    - Large chunks of data
    - Calling top level functions too much
    - Not using volatile pointers when reused
      - Amount of rereads/rewrites need to be specified using INTERFACE directive
- Solution:
  - Array sizing
  - Call top-level function a small number of times
  - Use *Limit diskspace* configuration

Mapping on FPGA

- Pointers:
  - Can conceal relations
  - Limit optimizations
  - Monolithic vs Distributed memory on FPGA
  - Dynamic memory allocation: stack, heap
- Variables in registers
- Arrays in block ram
- No shared memory but hardware constructs exist
- Memory based => stream based on FPGA

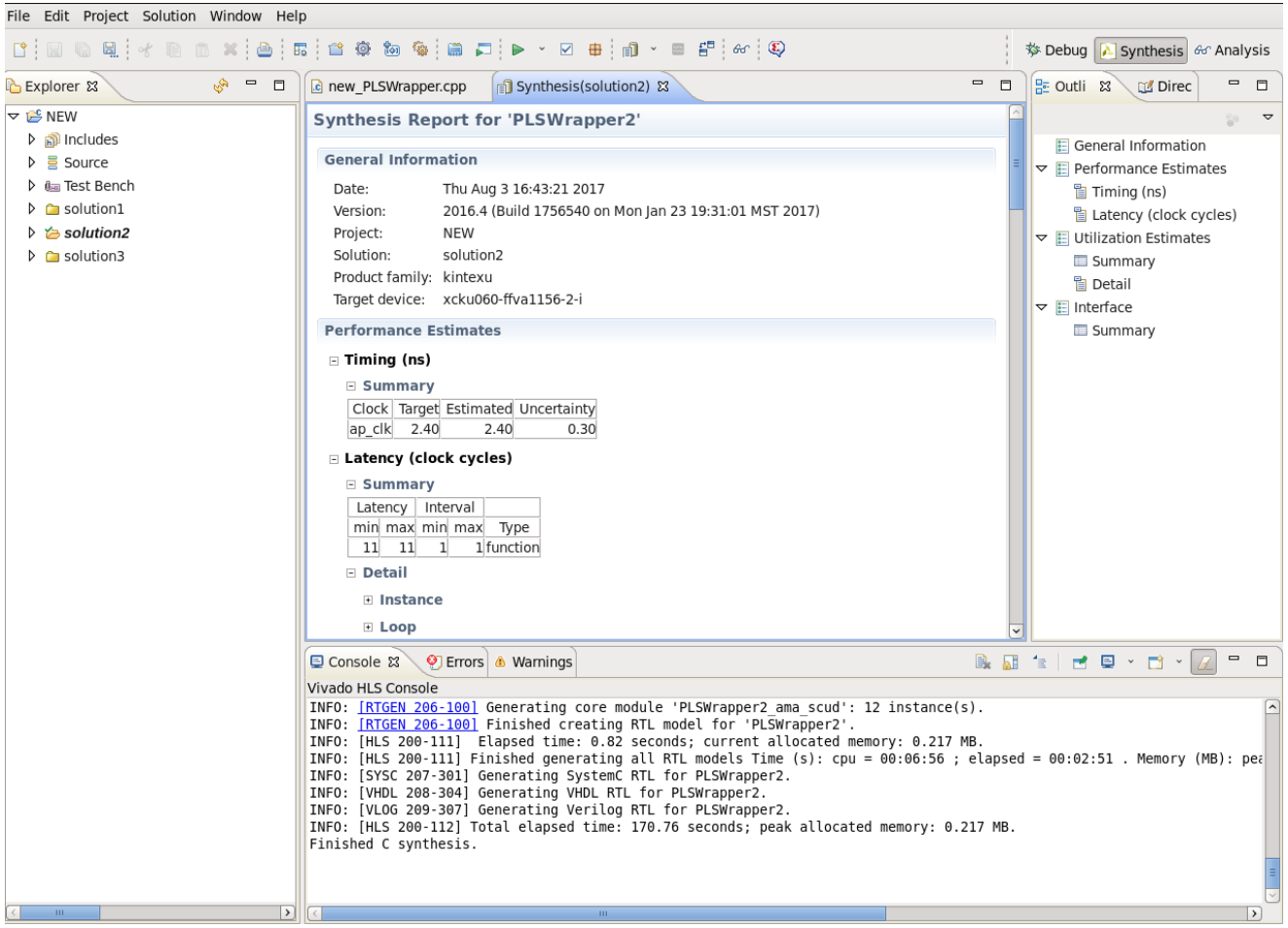
HLS Software: Xilinx Vivado HLS

Positive

- Easy & nice Eclipse based GUI
- Good tutorial and manual
- Online forum

Negative

- Bad estimates
- File name length limited by default
- User responsible for correct cosimulation result



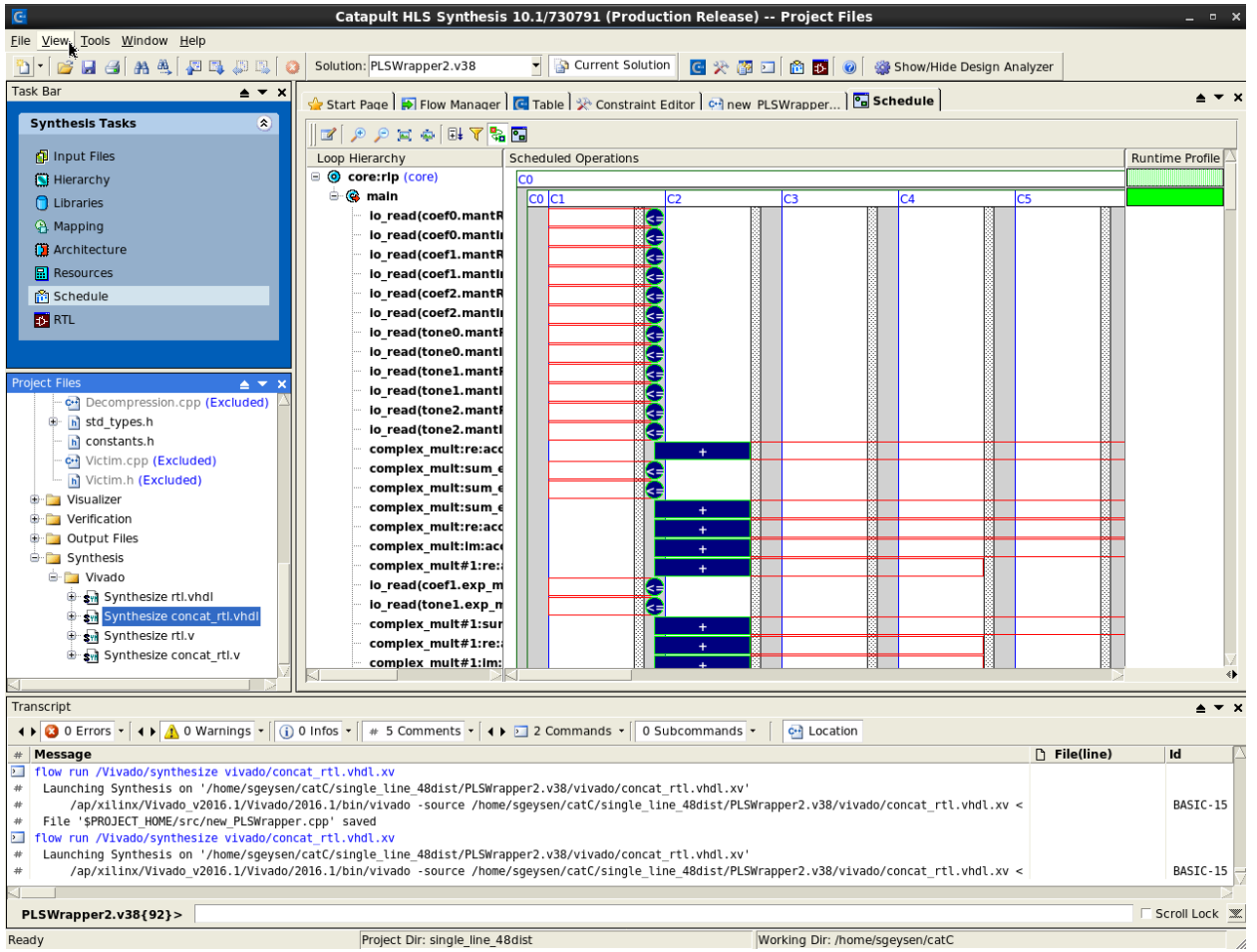
HLS Software: Mentor Graphics Catapult

Positive

- Not limited to Xilinx FPGA's
- Good overview of the synthesis steps
- Good manuals
- Cosimulation: Compares top level I/O of C and RTL code

Negative

- Limited amount of shortcuts
- Not easy to tweak the result
- Too optimistic about critical path
- Limited amount of online information



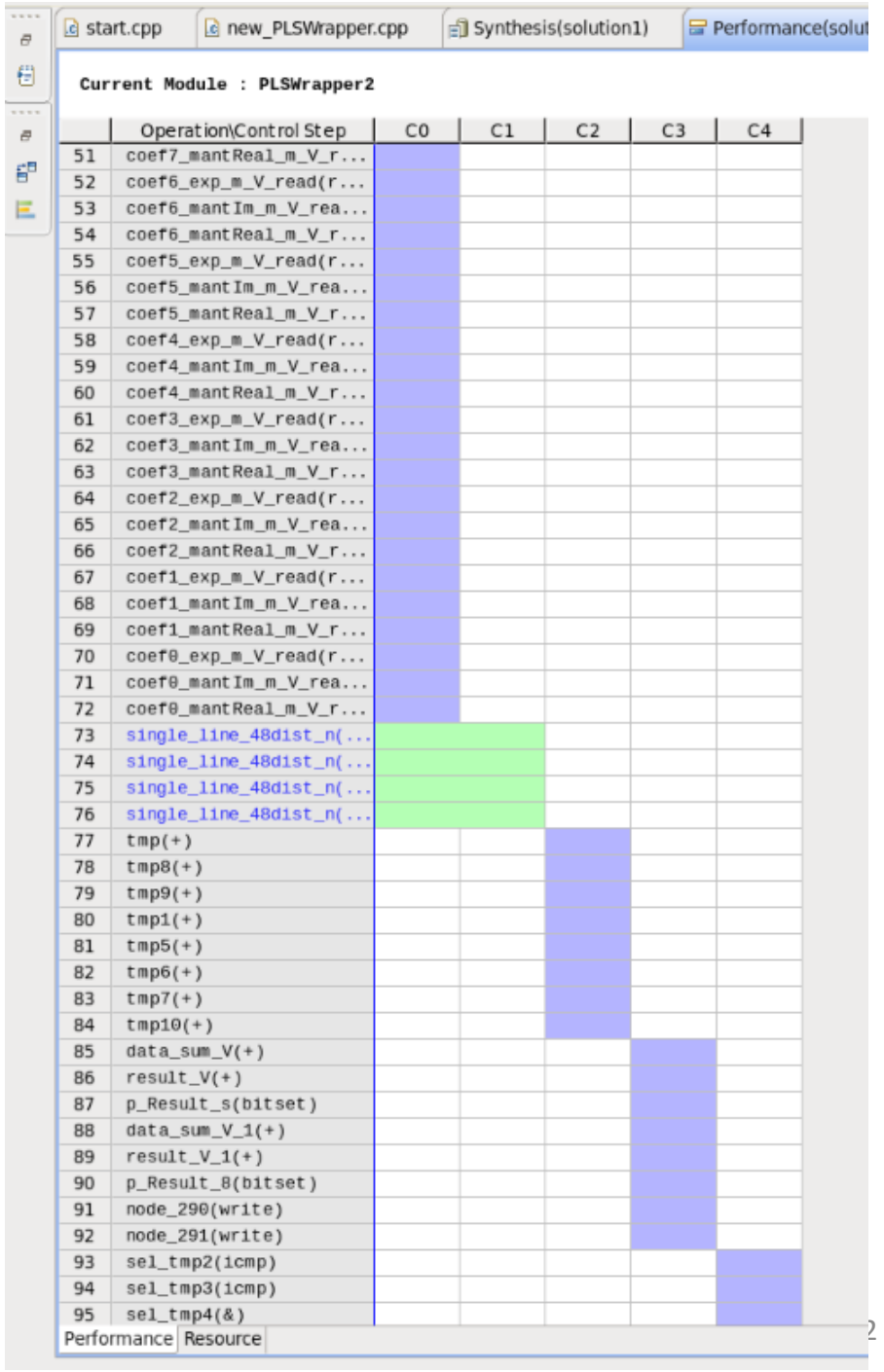
Arbitrary precision data types

- Avoid unnecessary resource utilization
- Each vendor different version:
  - Vivado HLS : AP (AutoPilot)
  - Catapult HLS: AC (Algorithmic C)
- Similarly but different:
  - Bit padding:

```
ac_int<3,true> ac_x = -1;
ap_int<3> ap_x = -1;
int ac_y = ac_x; // ac_y = -1 (sign bit padded)
int ap_y = ap_x; // ap_y = 7 (sign bit not padded)
```

Optimizations workflow (I)

- Directives/ pragma's & config used
- Strategy
  - Throughput: decrease initiation interval
  - Latency
  - Area



Optimizations workflow (II)

1. Initial changes using:
    - Interface
        - ap_none, ap_memory, ap_vld, ap_ovld
    - Datapack
    - Loop trip count
    - Re-reading
2. Increase concurrency
    - Dependencies:
        - Dataflow = Read/write
        - Anti-dependence = write/read
        - Output dependence = write
    - Pipeline by decreasing interval (loops)
        - Caution: memory accesses when using arrays
        - Block RAM has limited ports (2)
        - *Array partition* /*array reshape* pragma
        - Flush pipeline
    - Dataflow (functions)

Optimizations workflow (III)

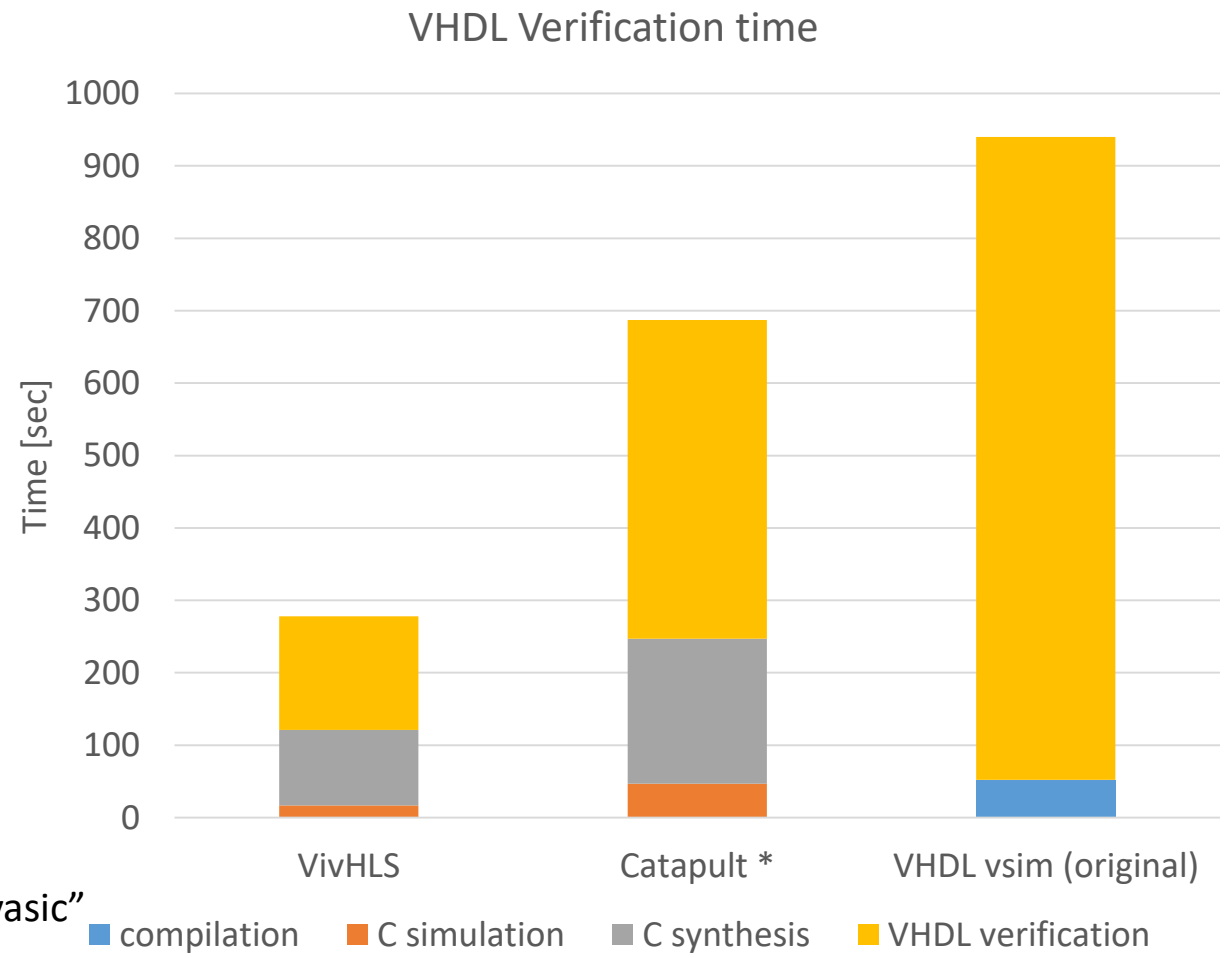
3. Latency
    - Inline recursively: reduce function call overhead
    - Latency directive: watch critical path
    - For loop incrementation
    - Question mark in latency report
4. Area
    - Inline
    - Adapted datatypes
    - Resource directive
    - Array mapping

Report

Version	Latency [cycles]	Interval [cycles]	FF	LUT	DSP48E
Arrays	47	48	1466	1835	9
Datapack	47	48	1589	1803	9
Pipeline	18	6	1740	1811	9
Partition arrays	11	1	4158	5670	36
No array/ ap_none	12	13	2815	5595	36
Pipeline wrapper	11	1	4276	5642	36
Recursive inline	11	1	4158	5670	36
Datapack	11	1	4158	5670	36
Flush enabled	11	1	4489	5600	36

Comparison: Design verification time

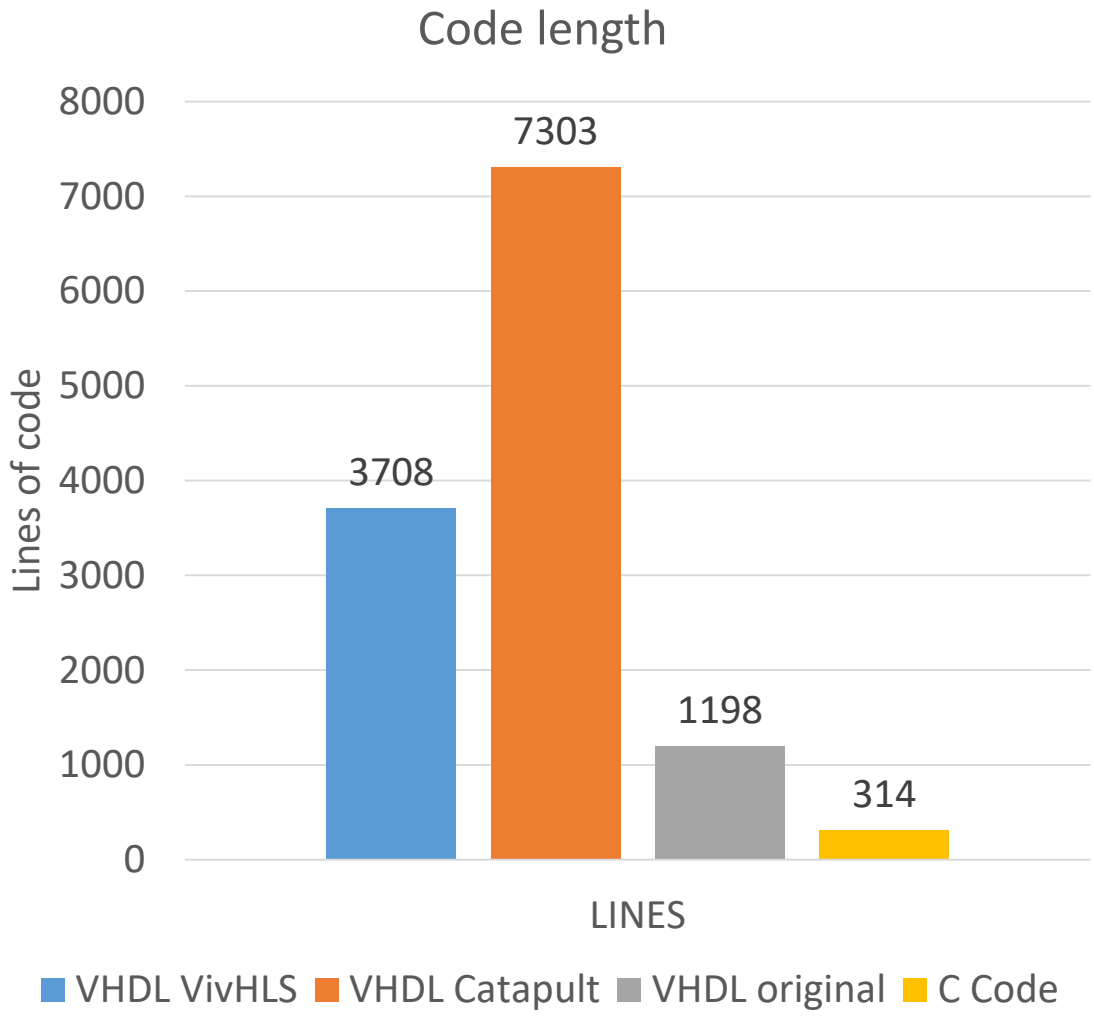
- Original RTL design:
  - Compilation: 52 sec
  - Running test: 14 min 58 sec
  - Total: 15 min 50 sec
- Vivado HLS:
  - C simulation: 17 sec
  - C synthesis: 1 min 44 sec
  - C / RTL cosimulation: 2 min 56 sec
  - Total: 4 min 57 sec
- Catapult C*:
  - C simulation: 47 sec
  - C synthesis: 3 min 20 sec
  - ModelSim simulation: 7 min 20 sec
  - Total: 11 min 27 sec



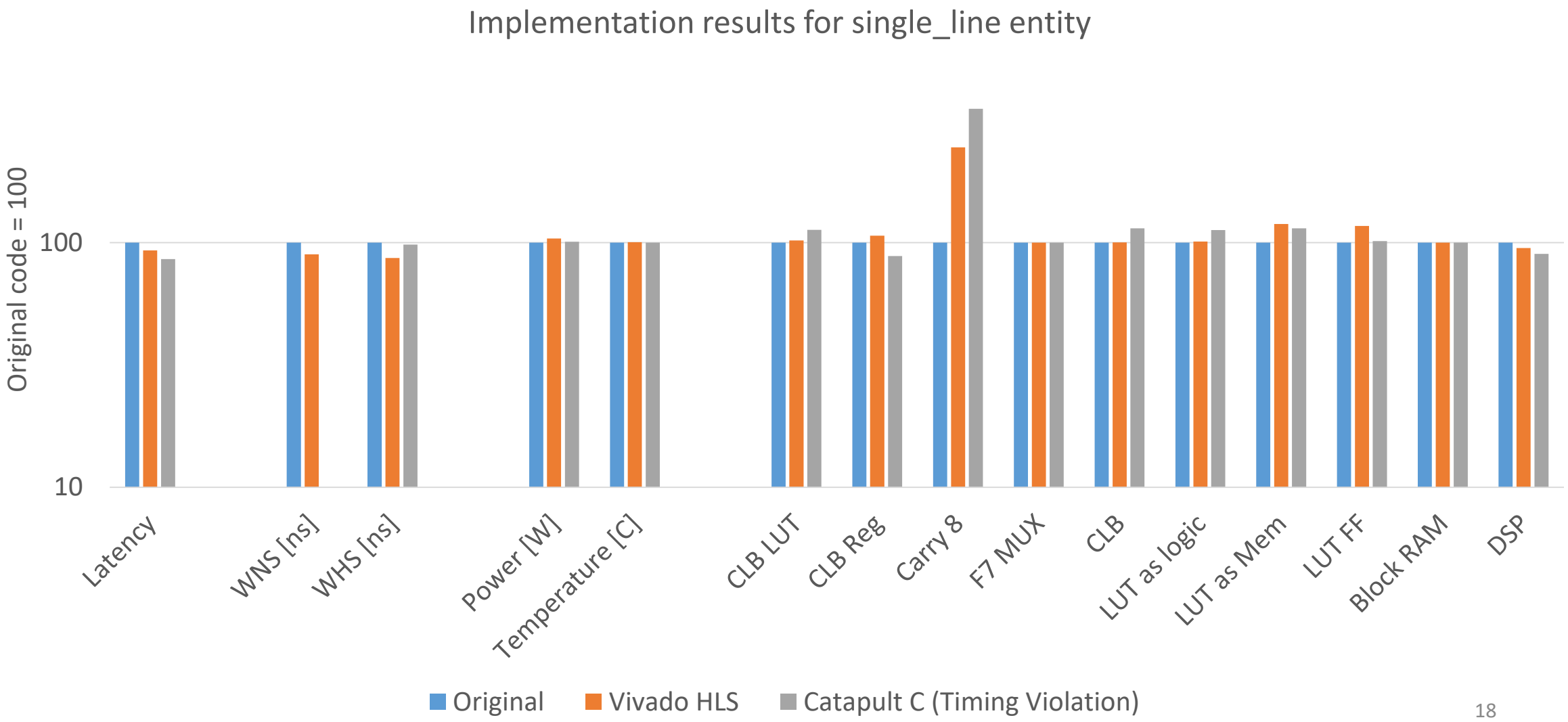
* Catapult running on “beantsu0212” server instead of “devasic”

Comparison: Code length

- Original RTL design:
  - 1198 lines in 10 files
  - + common files
- Vivado HLS generated VHDL:
  - 3708 lines total in 4 files
- Catapult HLS generated VHDL:
  - 7303 lines in 1 file
- C code:
  - 314 lines main logic
  - + common files



Implementation results



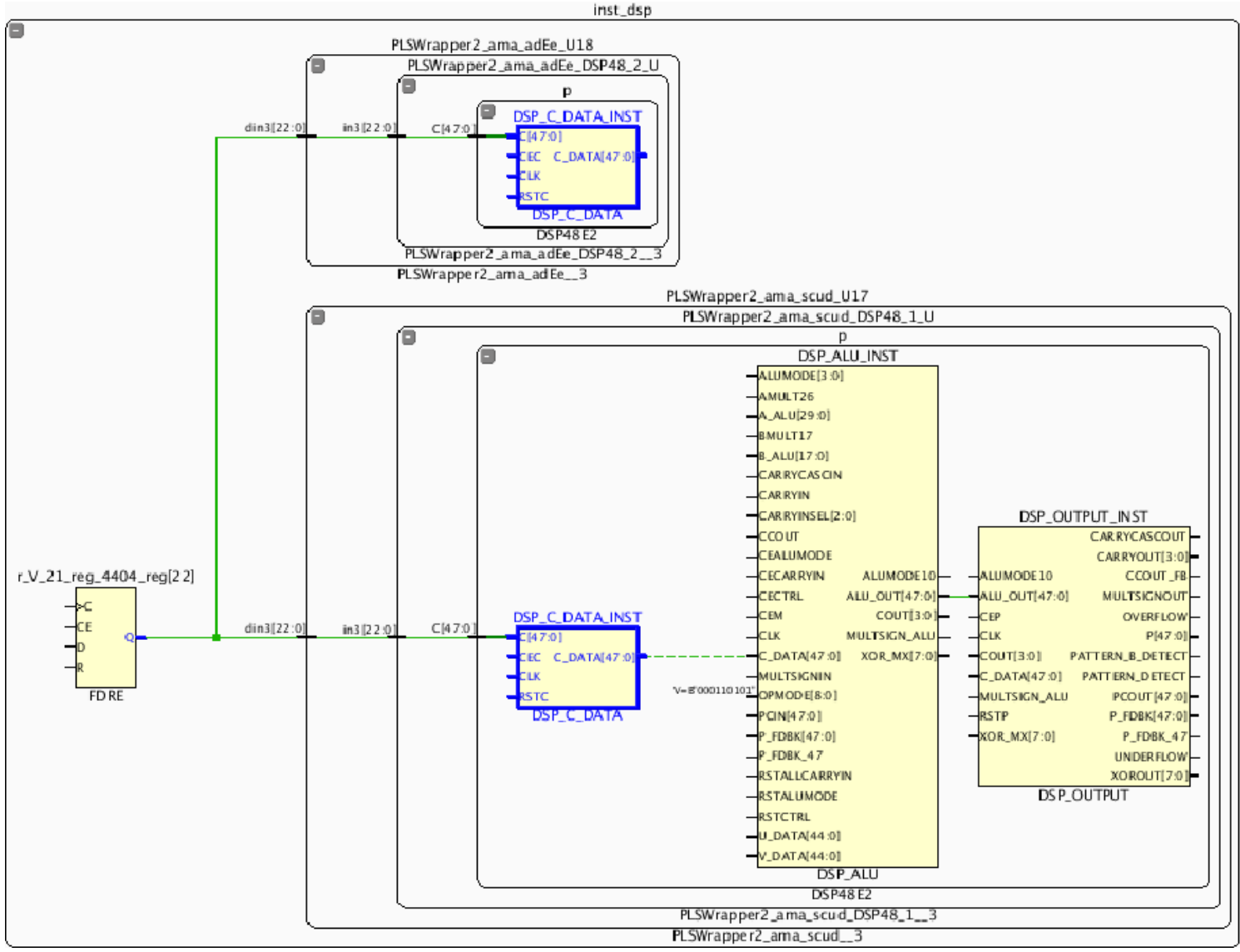
Problems

No timing closure

- Location:
  - Prevent crossing IO gap multiple times
- Fanout:
  - Signbit drives 52 ports

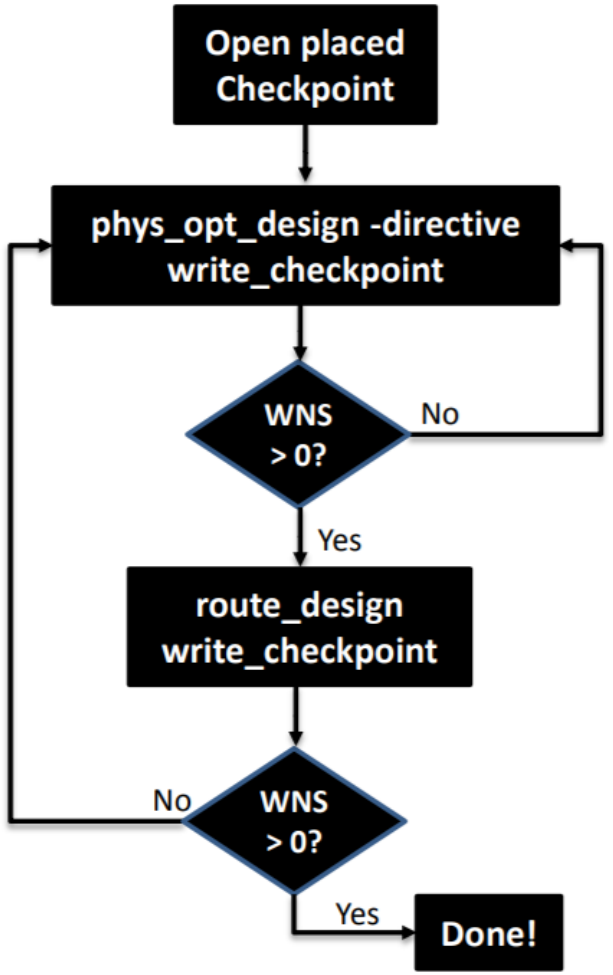
  = 2*(48-22)

- Paths
- Stages in mapping on FPGA
  - Synthesize
  - Place
  - Phys opt
  - Route
  - Post phys opt



Solutions: Optimize routing methodology

- Solve fanout with register replication
  - Load is halved, aggressiveFanout directive
- Solve location
  - Pblock creation
  - Ignore DSP in column next to IO block
    - DSP48E2_X8Y, DSP48E2_X17Y
- Shorten critical path with
  - Latency directive
  - Resource directive
  - Max_fanout directive (only in Catapult)



Timing Results for complete Bini

- WNS : -0.084 ns
- TNS : -0.354 ns
- 12 Failing endpoints
- WHS : 0.016 ns

Critical path in bini Ethernet subsystem (single_1g).

No failing endpoints in or connected to the HLS design

Conclusions

- HLS can accelerate RTL design of arithmetic parts a lot
- 3x faster verification
- Reuse of C testbench
- 4x smaller code to maintain
- Easy design changes (i.e. less area, different clock, pipelining ...)
- Easy to compare different solutions

- Critical path estimates of HLS tools not reliable
- Timing critical parts and connecting HLS blocks still needs to be done in RTL
- Tools will become better:
  - More accurate simulation libraries
  - More C code possibilities

Where to start

- Getting started:
- This presentation
  - Vivado HLS tutorial: chapter 1 & 2
  - Mentor HLS Bluebook: chapter 4 – Fundamentals of HLS
- Basic reference:
  - Vivado HLS reference guide ug902: chapter 3
  - Catapult user and reference manual: chapter 8

Questions?