Design Faster using HOLs

MAPLD 2008 Conference

Annapolis, MD

Tim Gallagher
Space Systems Company
Special Programs Strategic Development
Reconfigurable Technologies

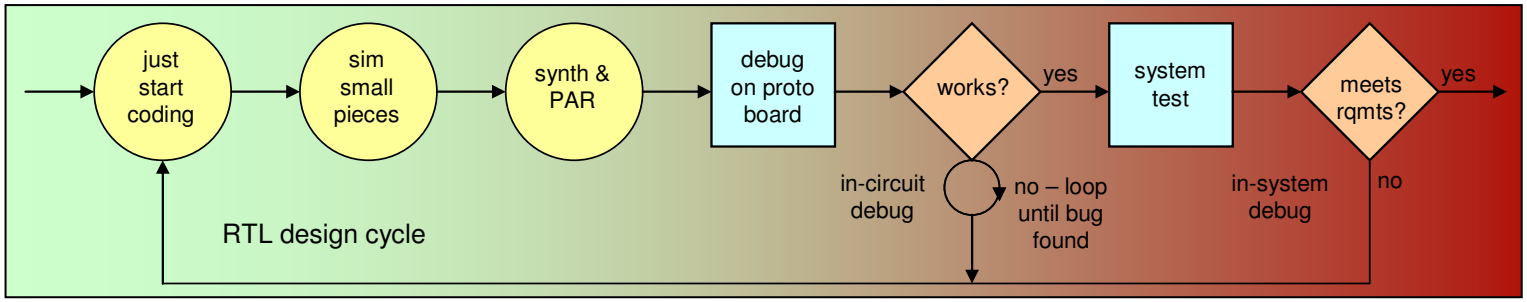
Design Faster using HOLs

- Why and how we use HOLs
- Case Study #1: Space Detection
- Case Study #2: Digital Channelizer
- Comfort vs Change
- Where we go from here

Why we use HOLs

Why

- The biggest advantage of FPGAs are now their biggest disadvantage
  - Reprogrammability has made designers lazy
  - Design as quickly as possible, debug in-circuit
    - Spend little time on requirement analysis and design documentation
    - Iterate many times to get glitches out
      - PAR and debug times have major schedule impact
      - Hope it all works correctly in system



Why

- Methods for 50K gate designs scale poorly
  - Using RTL you are designing state machines and process modules
    - Low level, bottom up circuits viewpoint
      - Just start coding
  - Using HOLs you are optimizing and verifying
    - High level, top down systems viewpoint
      - Thinking about the problem

VS

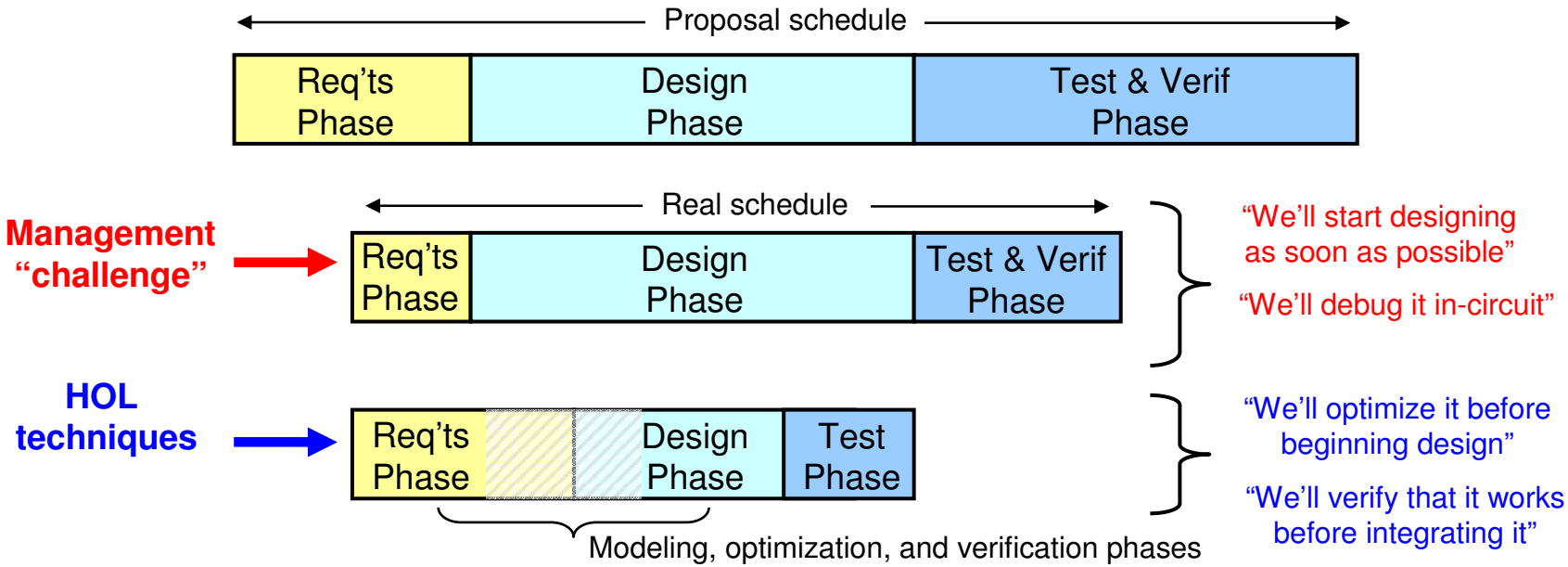
Optimized design = smaller, faster design

Up front verification = << PAR iterations

Up front verification = << System debug time

Why for Space

- **Schedule Compression**
  - **Affects Requirements & Test/Verification Phases**

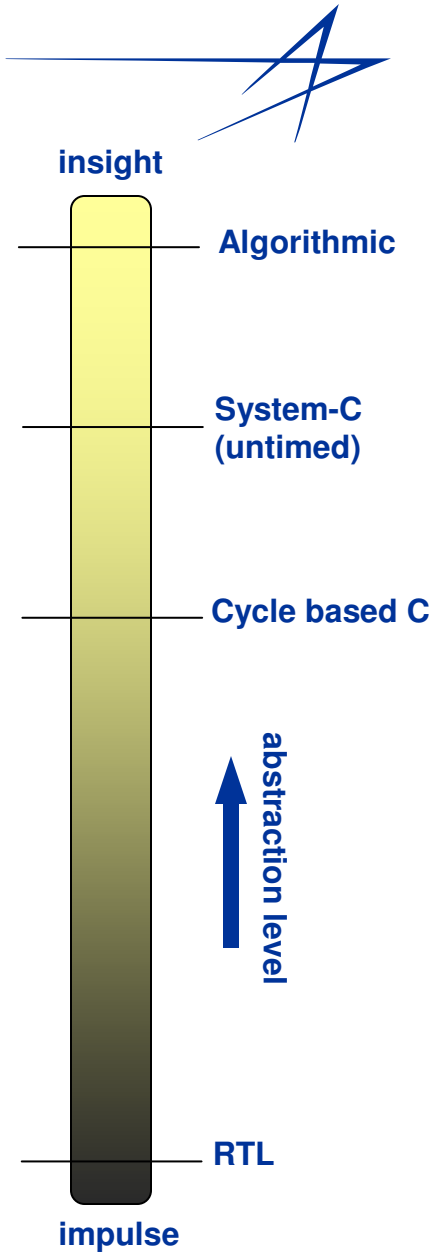


- **Marching Army**
  - **A launch deadline awaits, very painful to miss**
  - **May require first pass success to meet schedule**

How we use HOLs

How

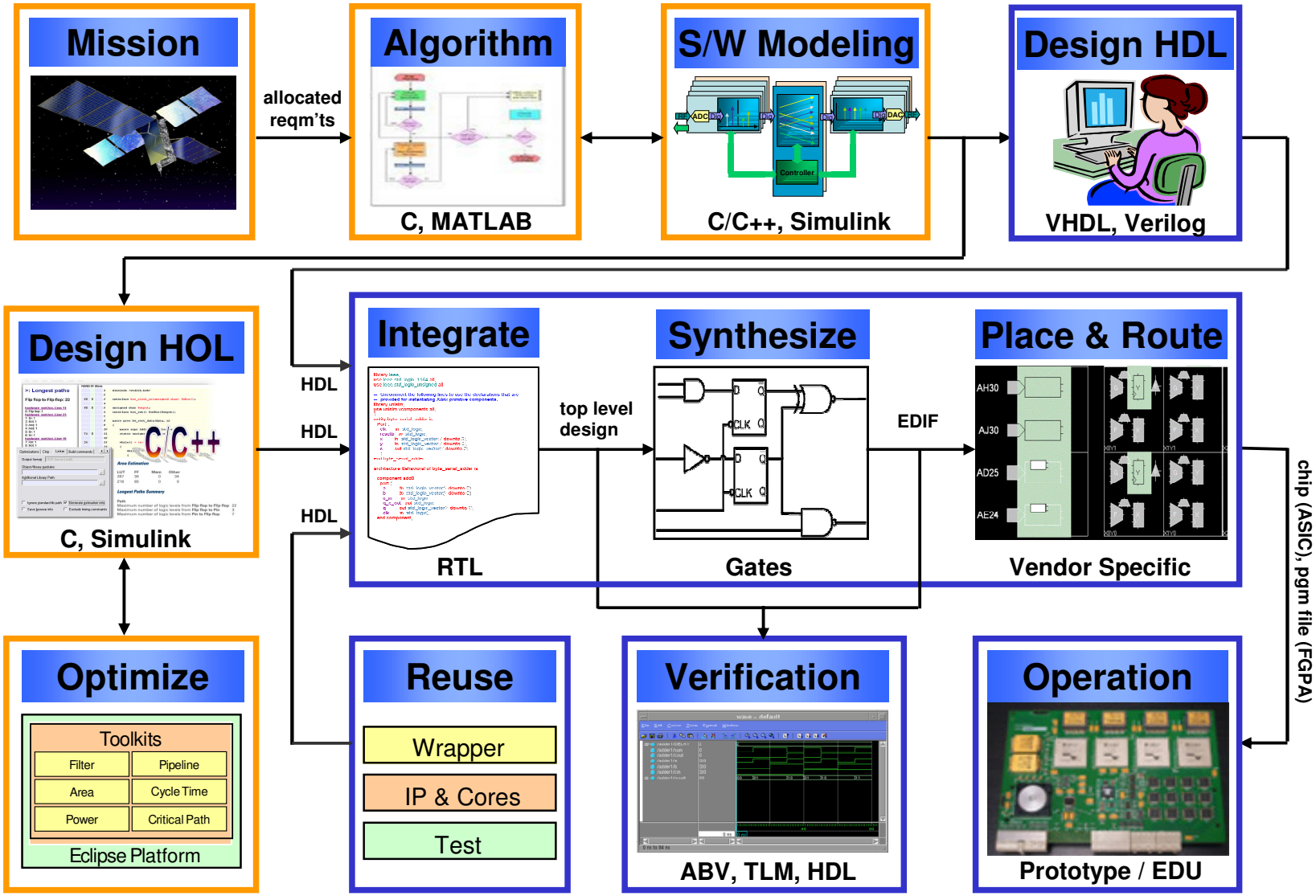
- Abstraction level is key
  - Is Clocked C the best compromise between productivity, performance, and maturity?
  - Higher abstraction level means higher productivity
  - Autocoded state machines
    - The conundrum of RTL
- S/W Engineers work at higher abstraction levels
  - System modeling and verification
  - Algorithm development



How

- Use HOL in conjunction with RTL coding
  - i.e. "Best of Both Worlds" methodology
    - Common sense approach given current tools
  - Treat HOL output as autocoded IP core
    - Provide RTL, interface and core descriptions
    - Integrate all RTL within an IDE
  - Use multiple HOL tools
    - Synplify DSP® (Simulink® / MATLAB® based)
    - AgilityDS (f. Celoxica) (Handel C™ based)
  - Added tool enhancements
    - Visibility add-ons (data/control flow graphs)
    - Area, speed, power analysis (I/F synthesis?)

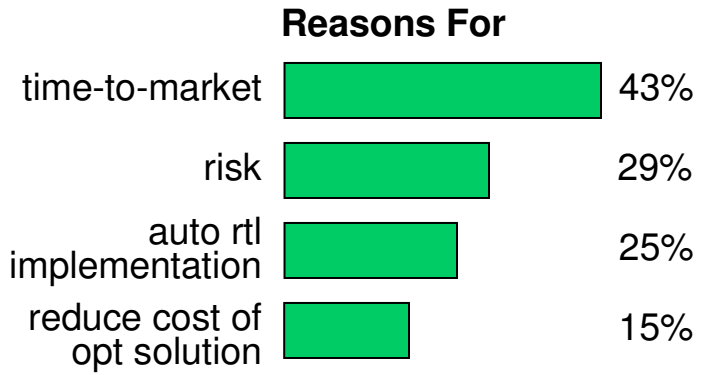
How

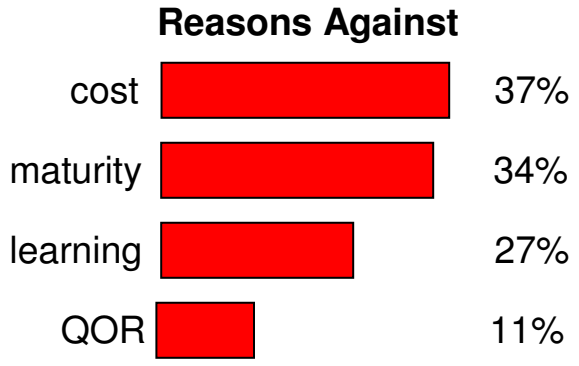

Mission
allocated reqm'ts
Algorithm
C, MATLAB
S/W Modeling
C/C++, Simulink
Design HDL
VHDL, Verilog
Design HOL
C, Simulink
HDL
HDL
HDL
Integrate
RTL
top level design
Synthesize
Gates
EDIF
Place & Route
Vendor Specific
chip (ASIC), pgm file (FGPA)
Optimize
Toolkits
Filter
Pipeline
Area
Cycle Time
Power
Critical Path
Eclipse Platform
Reuse
Wrapper
IP & Cores
Test
Verification
ABV, TLM, HDL
Operation
Prototype / EDU

How we choose HOLs

Tool Selection

- Tool Costs
- Learning curve
- Not on “Approved CAD List”
- Usability
  - Performance, IDE
  - Risk, stability and support
- User/Abstraction
  - H/W, S/W, Scientist
  - C, C++, Matlab, Simulink
- Design
  - Control, data flow, algorithmic
  - Image or DSP processing





Mentor Graphics Corps. survey, Dec ‘05

Case Study #1
Detection (of Satellites) in Space

Detection

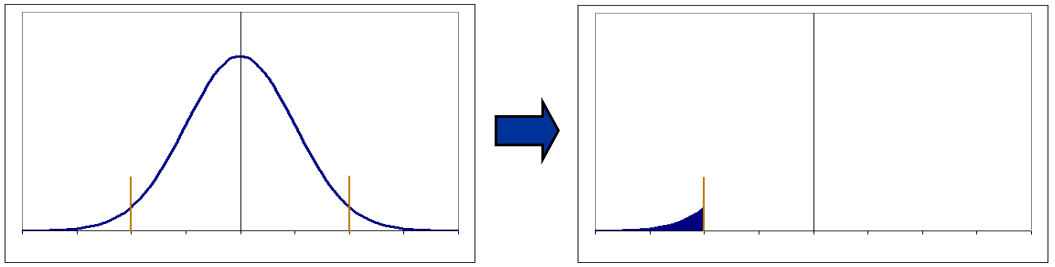
- Variance Filter on 20 image frames (N)
  - 1K x 1K pixels (*x*)

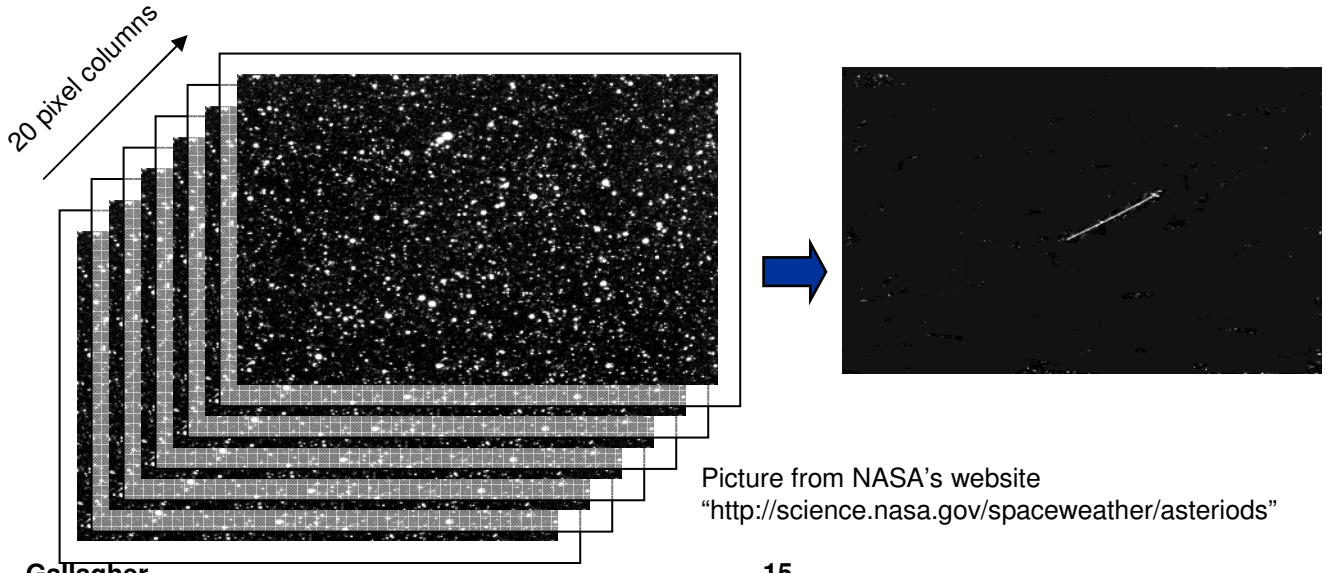
$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \quad where \quad \bar{x} = \frac{x_1 + x_2 + \bullet\bullet\bullet + x_N}{N} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- Real-time performance
  - Process a pixel element every clock cycle
- Consultant said it wouldn't work in FPGA
  - Size and performance issues
- Very quick demo schedule for proposal
  - Demo in new (under development) RCC system

Detection

1. For each *xi* calculate its deviation () from the mean then form the squares of those deviations
2. Find the mean of the squared deviations, variance σ^2
3. Take the square root of the variance and compare to end point value
4. If $\sigma >$ end point set all *xi* to zeroes else leave untouched





Picture from NASA's website
"http://science.nasa.gov/spaceweather/asteriods"

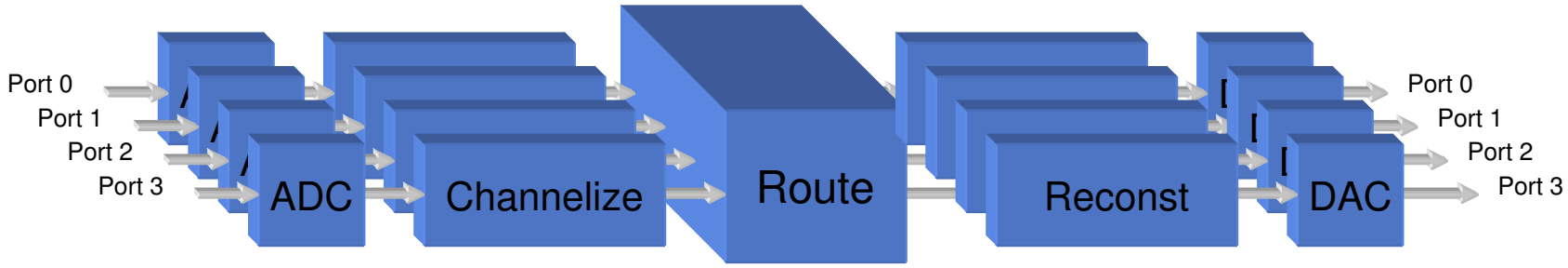
Detection

- Variance Filter
  - 2 ½ weeks to convert to Handel-C
  - 1 week top-level and board integration
    - Cycle accurate, bit accurate 1st time in-circuit
  - Took 1 minute to simulate in Handel-C
    - 4+ hours in ModelSim®
    - Handel-C allowed many design iterations/day
  - Produce faster and smaller circuits then hand-coded RTL
    - Optimized algorithm provides optimize circuits

Case Study #2
Digital Channelization for Space

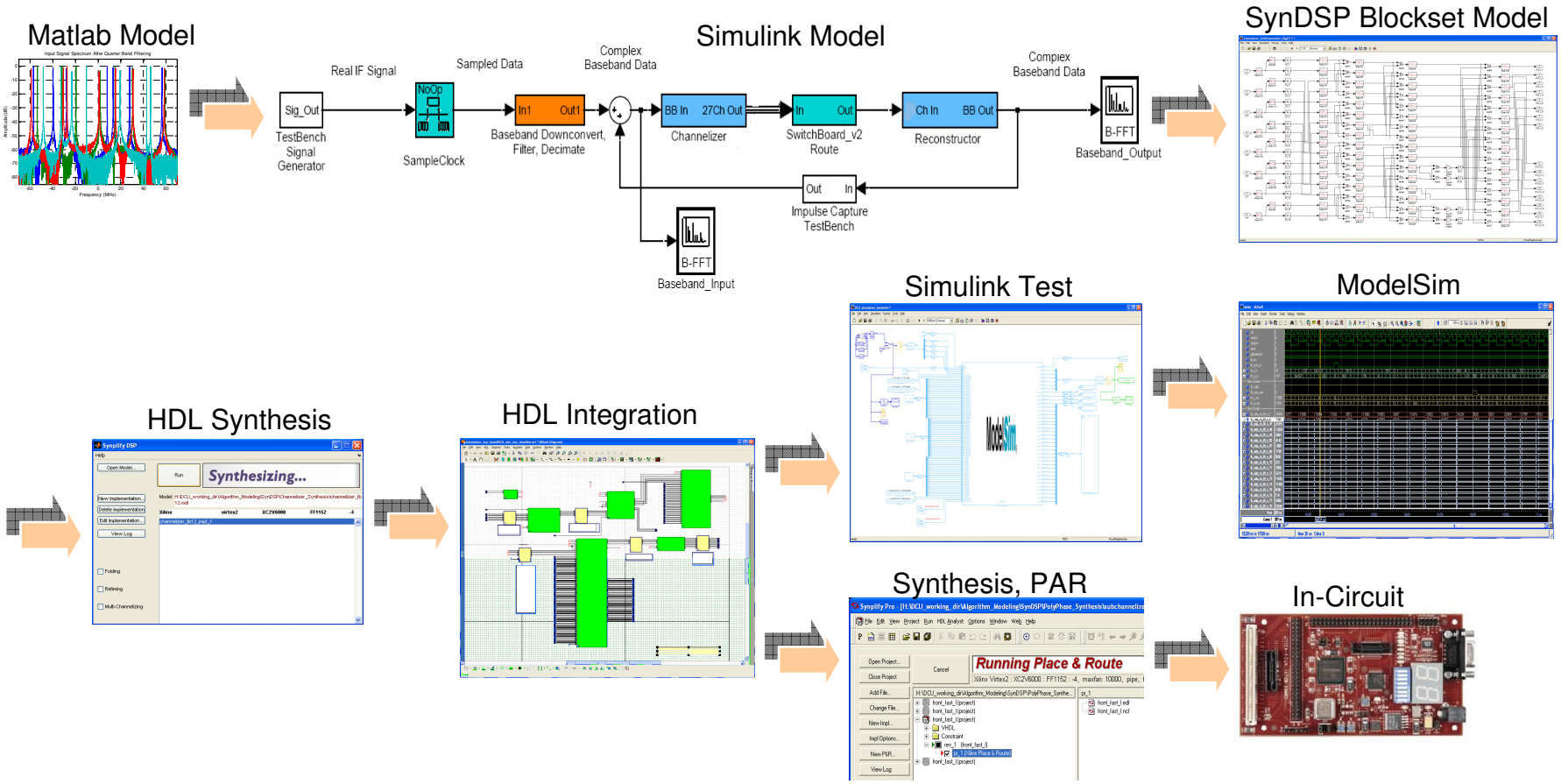
Channelizer

- Channelizer/Reconstructor
  - 500 MHz digitized frequency bandwidth
  - 108 narrowband subchannels
  - Alter the subchannel gains
  - Measure the average and peak signal power levels
  - Reconstruct the subchannels into a composite bandwidth
  - Transform the recombined spectrum for analog conversion



Channelizer

- **Channelizer/Reconstructor Design**
  - **Matlab ⇨Simulink ⇨Synplify DSP**



Channelizer

- DSP MATLAB design
  - Started as hand-coded VHDL (but not finished)
  - Simulink designer ported to Synplify DSP
    - >> Productivity over RTL code development
      - Estimated by scaling time to finish VHDL
- Network design
  - 32 Gbps per FPGA (16 Gbps RX & TX)
  - 108 subchannels routed to any other
    - In any order (sorting) with multicast
  - 7 weeks design and simulation in C code
  - 1 week to convert to Handel-C and then RTL
  - Cycle accurate, bit accurate 1st time in-circuit

Channelizer

	ADC Actel	DAC Actel	SER Actel	SWT Actel	CTL Actel	DSP Xilinx
Function	DSP PreProc Xilinx Cfg Cntrl Memory Cntrl	DSP PostProc Xilinx Voter	SERDES Cntrl Xilinx Voter	Network Switch Router	uProc I/F Mem Cntrl 1553B & RIU	Channelizer Reconstuctor Pwr & Gain
Inputs	280 MHz x 16-bits DDR	140 MHz x 3 x 22-bits	100 MHz x 16-bits x (9 + 6) (240 pins)	100 MHz x 5 SERDES (2.0 Gbps x 10)	Many dissimilar IOs	140 MHz x 32-bits + 100 MHz x 16-bits x 3
Outputs	140 MHz x 32-bits x 3	210 MHz x 10-bits x 2 x 4 (210 MHz x 80)	100 MHz x 16-bits x 9 (144 pins)	100 MHz x 5 SERDES (2.0 Gbps x 10)	Many dissimilar IOs	140 MHz x 22-bits + 100 MHz x 16-bits x 3
Usage	92% R 34% C 9% RAM	95% R 33% C 19% RAM	81% R 34% C 9% RAM	94% R 41% C 85% RAM	51% R 35% C 6% RAM	47% FF 44% LUTS 13% RAM 18% MULT
Tool	VHDL & Handel-C	VHDL & Handel-C	VHDL	Handel-C & VHDL	Handel-C	Synplify DSP, VHDL, Handel-C

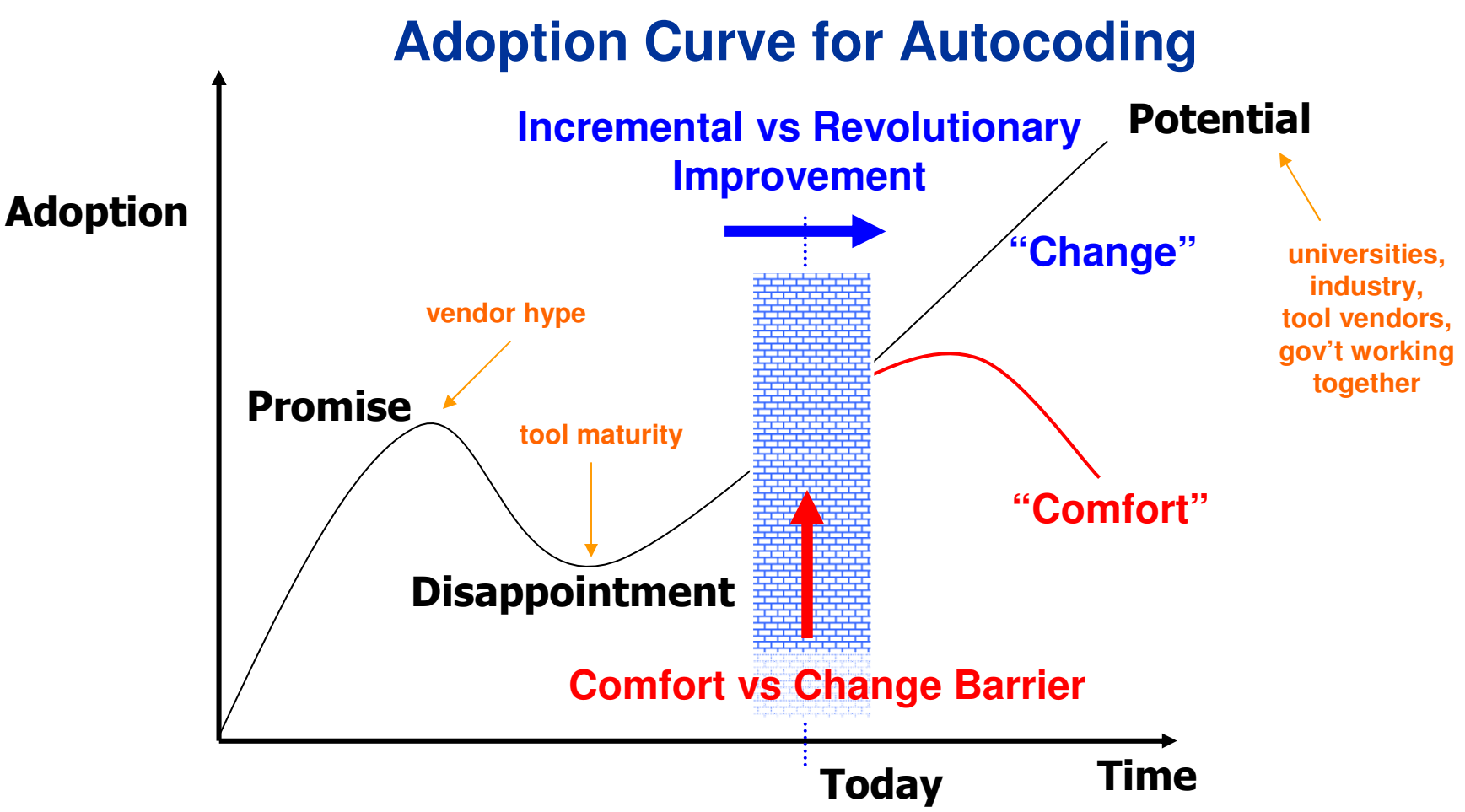
Actel® = RTAX2000
Xilinx® = XQR2V6000

Fast EDAC core - 210 MHz DAC I/F in Actel RTAX2000, VHDL FIFO wrapper encasing Handel-C EDAC wrapped around a Block RAM instantiation

Best of both worlds approach → joint HOL & RTL for rapid, high-speed, high-density designs

Comfort vs Change

Comfort vs Change



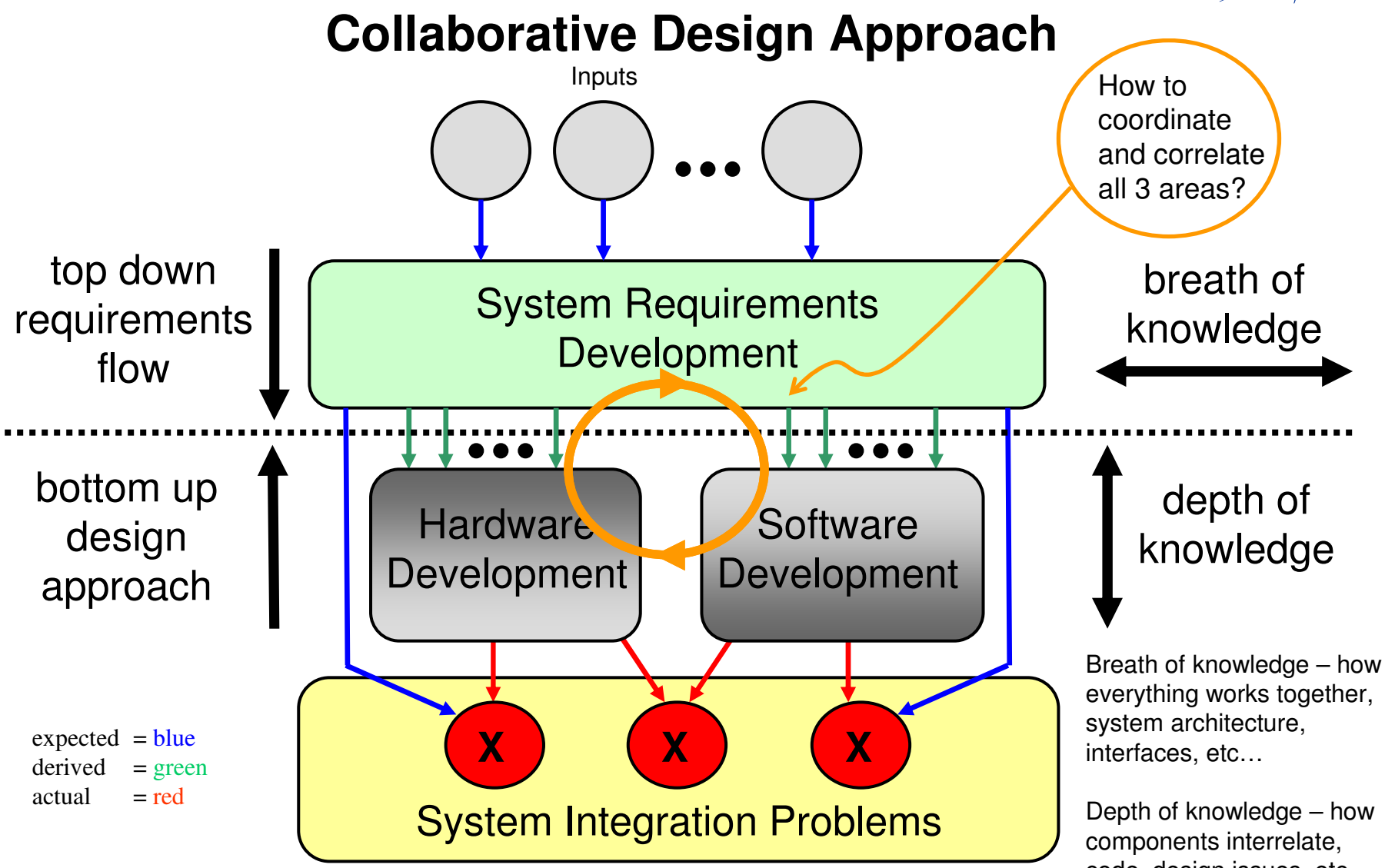
"Why doesn't ESL/HOL for RTL generation have more acceptance in the H/W community and what will make that happen?"

Comfort vs Change

- Experience vs Willingness to Learn
  - New grads have openness to using both software and hardware techniques
  - Pushing the envelope vs justifying any changes
  - Long-term investment in low-level RTL coding
- Systems-Level Thinking vs Schematic Based Thinking
  - OOD/OOA vs Gates
- Mandating HOL techniques for design doesn't work well
    - "Willful ignorance" (Urban Dictionary)

Where we go from here

Where

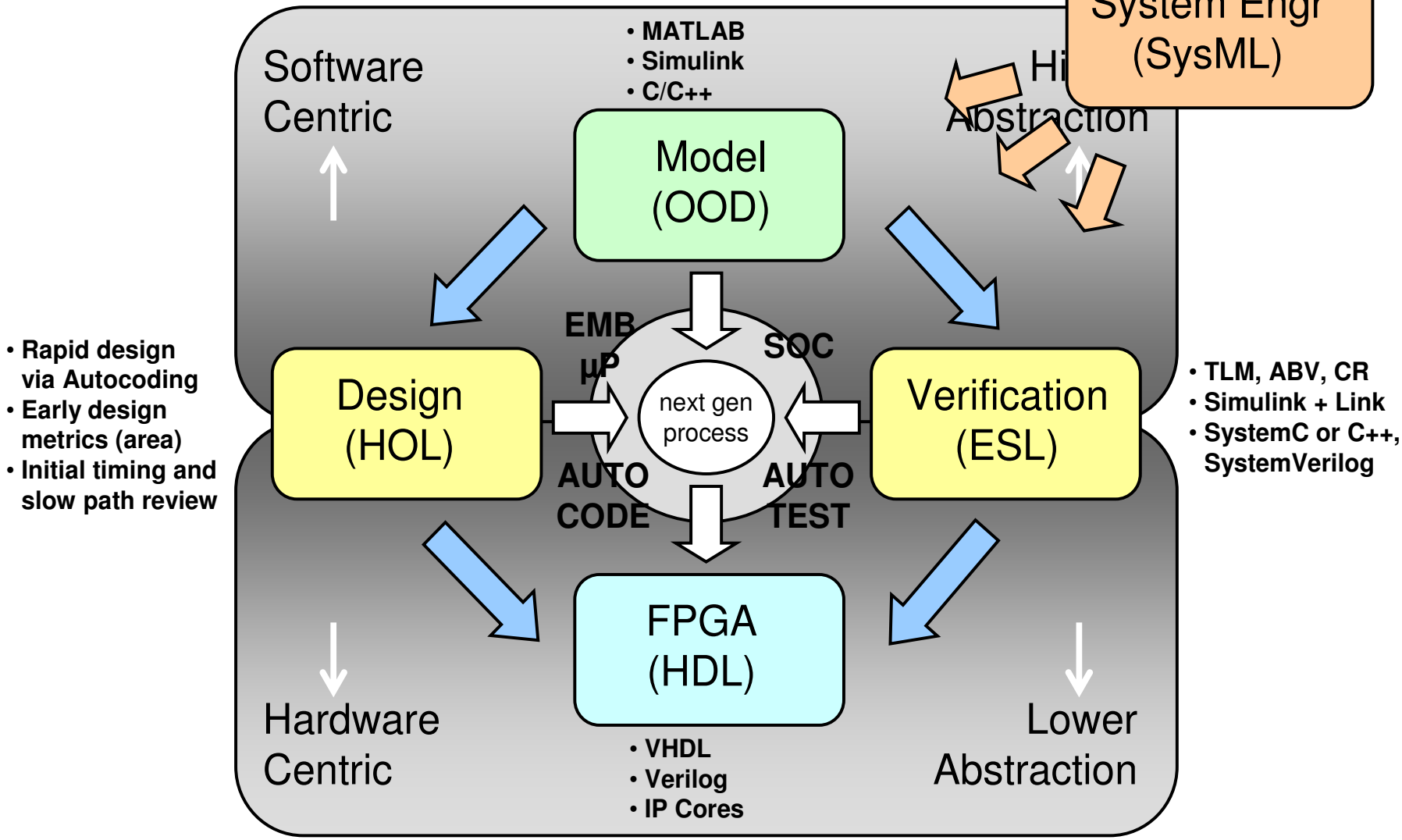

Collaborative Design Approach
Inputs
How to coordinate and correlate all 3 areas?
top down requirements flow
System Requirements Development
breath of knowledge
bottom up design approach
Hardware Development
Software Development
depth of knowledge
X
X
X
System Integration Problems
expected = blue
derived = green
actual = red
Breath of knowledge – how everything works together, system architecture, interfaces, etc…
Depth of knowledge – how components interrelate, code, design issues, etc…

Where

- Take a “higher” level viewpoint
  - ESL for modeling and verification then work for more HOL autocoding acceptance
  - MBSD for model based system design
- Work towards a Totally Integrated Approach
  - Requirements to Models to Gates to Verification
  - SysML, SystemC, SystemVerilog, C/C++
  - Quantifiable Metrics, Lessons Learned

Where

Integrated Design Approach



- Rapid design via Autocoding
- Early design metrics (area)
- Initial timing and slow path review

- TLM, ABV, CR
- Simulink + Link
- SystemC or C++, SystemVerilog