Agile Design Practices and High-Level Verification for Spacecraft Electronics Workshop

Moderator:
Tim Gallagher
Lockheed Martin

Workshop Agenda

15:50 – 16:00	Tim Gallagher	Intro
16:00 – 16:07	Tim Gallagher	Agile Design
16:07 – 16:14	Mike Wirthlin	Rapid Design
16:14 – 16:21	Mir Sayed Ali	HLS for DSP
16:21 – 16:28	Doug Krening	SystemVerilog
16:28 – 16:35	Mike Horn	OVM/UVM
16:35 – 16:42	Doug Johnson	Virtual Proto
16:42 – 16:49	JP Walters	FT Verification
16:49 – 17:09	Panel Session	
17:09 –	Audience Q&A	

Presenters and Panel Members

- Tim Gallagher on Agile Design
  - LM Fellow, Reconfigurable Computing Technologies Lockheed Martin Space Systems Company
  - Principal Investigator SSC SW Multi-Core Research
  - Agile Design Practices and Processes for FPGAs
  - Joint RTL and High-Order Language (HOL/ESL) Based Development Methodologies

Presenters and Panel Members

- Dr. Mike Wirthlin on Rapid Design
  - Associate Professor ECE, Brigham Young University
  - Associate Director of the BYU Configurable Computing Laboratory
  - Faculty Advisor in the NSF Center for High-Performance Reconfigurable Computing (CHREC)
  - Principle Investigator, DARPA Study on FPGA Design Productivity
  - Research Interests
    - Fault Tolerant FPGA design and reliable FPGA computing
    - FPGA Design Productivity

Presenters and Panel Members

- Mir Sayed Ali on HLS for DSP
  - o Sr. Staff Applications Engineer Microsemi Corp
  - o 11 years experience in the areas of FPGA design, verification and implementation
  - o Expertise on Microsemi IPs for space applications such as 1553, PCI and DSP
  - o Master's degree in Electrical Engineering from University of Oklahoma, Norman, USA
  - ✓ "Digital Signal Processing (DSP) Design Flow and Design Techniques Using RTAX-DSP FPGAs" Thur 8:50 MAPLD Session C

Presenters and Panel Members

- Doug Krening on SystemVerilog
  - Advanced Functional Verification utilizing SystemVerilog Consultant
  - Currently: Supporting Lockheed Martin / GOES-R
    - Verification Methodology Development
    - Verification Team Training
    - FPGA Verification
  - Previously:
    - President / Principal Engineer, FirstPass Inc
    - Director / Principal Engineer, Vitesse Semiconductor

Presenters and Panel Members

- Mike Horn on OVM/UVM
  - o Principal Verification Architect Mentor Graphics
  - o Primary responsibility to help organizations deploy UVM and OVM
  - o One of the authors of the UVM/OVM Online Methodology Cookbook
    - http://verificationacademy.com/cookbook/
  - o Used High-level Verification Languages (HVL) since 1999 including Specman E, Vera and SystemVerilog
  - ✓ "Applying OVM (UVM) to GOES-R C&DH Development" Wed 13:50 MAPLD Session B

Presenters and Panel Members

- Doug Johnson on Virtual Prototyping
  - Staff Applications Consultant at Synopsys, Inc.
  - 30+ years of industry experience in communication design engineering, electronic design automation (EDA) tools, applications engineering, digital signal processing (DSP), intellectual property (IP) licensing and account management
  - BSEE from the University of Illinois at Urbana-Champaign.

Presenters and Panel Members

- John Paul Walters on Verifying Fault Tolerance
  - USC/ISI Computer Scientist, Adaptive Parallel Execution Group
  - Research interests include fault tolerance and reliability, HPEC, multi-core processing
  - Co-developed Virtex-4 fault injector
  - Co-developed SpaceCube software fault tolerance layer
  - ✓ “Radiation Hardening of FPGA-embedded CPUs via Software, Validated with Fault Emulation” Wed 13:25 MAPLD Session B

Applying Agile Software Techniques to Hardware Design (FPGA)

Tim Gallagher
Space Systems Company

Agile Design for Hardware

- Why?
  - Multiple programs with “red” FPGA deliveries
    - Reprogrammability has made designers lazy

  - Design as quickly as possible, troubleshoot in-circuit
    - » Just throw together some code and hope it all works
    - » Spend little time on design architecture and analysis
  - Multiple Synthesis, Place & Route, Debug cycles
    - » Place & Route runs can take days on tough designs
    - » Difficult to debug in-circuit with today’s complexity

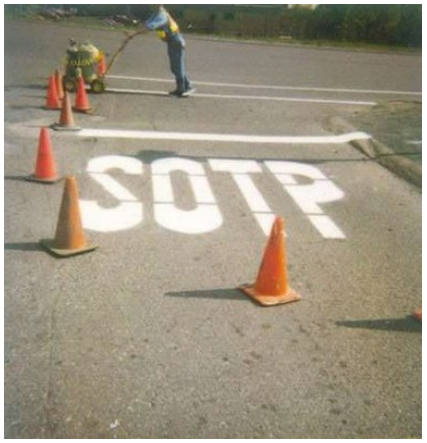
Cost, schedule, customer satisfaction issues!

  - Need a “one time to get it right” approach!
    - Agile sets the attitude for error free designs!

Agile Design for Hardware

- What?
  - Agile Development Methods
    - Rapid, short iteration design cycle
      - Concurrent development and rapid-turnaround between HW, SW, Systems, and Verification Teams
      - Delay decisions and deliver code whenever possible
    - Test Driven Design (TDD)
      - Verification team drives requirements, architecture, and design
    - High-Level Verification
      - Independent design and verification teams
      - Different tools, languages, and methods to enforce isolation of efforts

Agile Design for Hardware



- What?
  - Agile Development Methods
    - Static Analysis
      - Eliminate tedious line-by-line reviews
    - Design Patterns
      - Templates for common design issues such as CDC, interrupts, I/O links/buses
    - Just-In-Time Training (JIT^2)
      - Training delivered when needed
    - Metrics and Bug Tracking/Reporting
      - Includes real-time response to issues

Proven Agile Techniques for Error Free Design

Reuse, Reuse, and More Reuse

Mike Wirthlin

Brigham Young University

NSF Center for High Performance Reconfigurable Computing (CHREC)

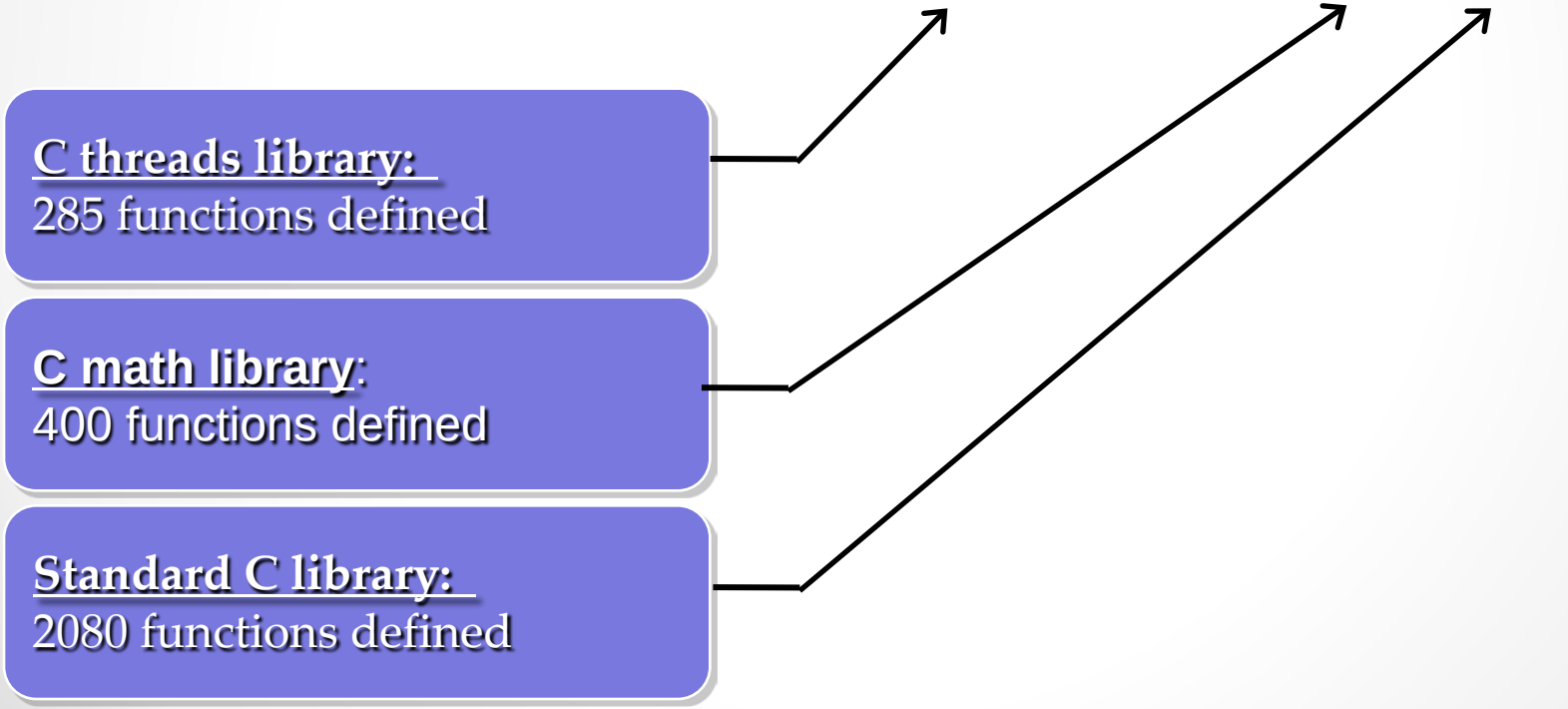
Presenters and Panel Members

- Dr. Mike Wirthlin,
  - Associate Professor, Brigham Young University (Department of Electrical and Computer Engineering)
  - Associate Director of the BYU Configurable Computing Laboratory
  - Faculty Advisor in the NSF Center for High-Performance Reconfigurable Computing (CHREC)
  - Principle Investigator, DARPA Study on FPGA Design Productivity
  - Research Interests
    - Fault Tolerant FPGA design and reliable FPGA computing
    - FPGA Design Productivity

DARPA Study on FPGA Design Productivity

REUSE

```
gcc –o netmon netmon.c –lpthread –lm –lc
```



DARPA Study on FPGA Design Productivity

- Reuse a key component of design productivity

Design Effort = Initial Design Effort × [(1-R) + (O × R)]

R: Fraction of design exploiting reuse
O: Overhead of reuse

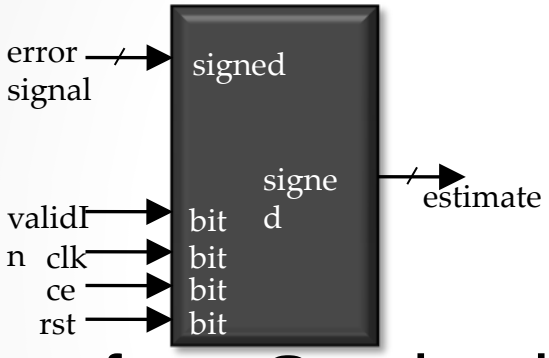
Example: R = .8 (reuse 80% of the code), O=.1 (10% overhead for reuse)
Design effort with reuse = ¼ the design effort without reuse

- It is difficult to reuse
  - Lack of documentation, test vectors, etc.
  - Too specialized
  - Not invented here (NIH)

"If the cost of reuse is more than 30% than the cost without reuse, reuse will seldom occur"

Reuse RTL Code with Meta-Data (B1-09)

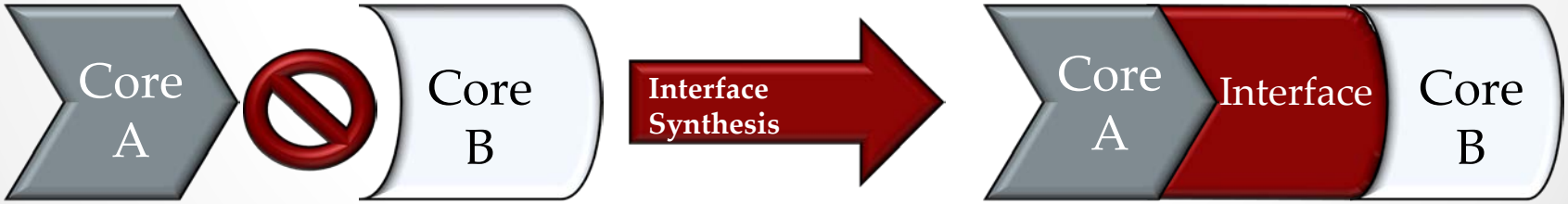
- Facilitate Automated Reuse



Loop Filter Parameterization	
Accumulation Width	32
Loop Bandwidth	0.01
Loop Damping Factor	1.0
Phase Detector Gain	6.0
DDS Gain	-1.0
Samples Per Symbol	2
K Precision	44
Order	2

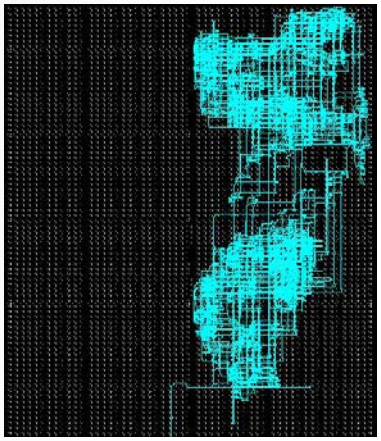
IP-XACT
IEEE P1735

- Interface Synthesis

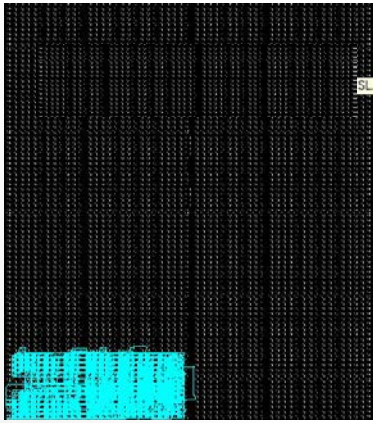


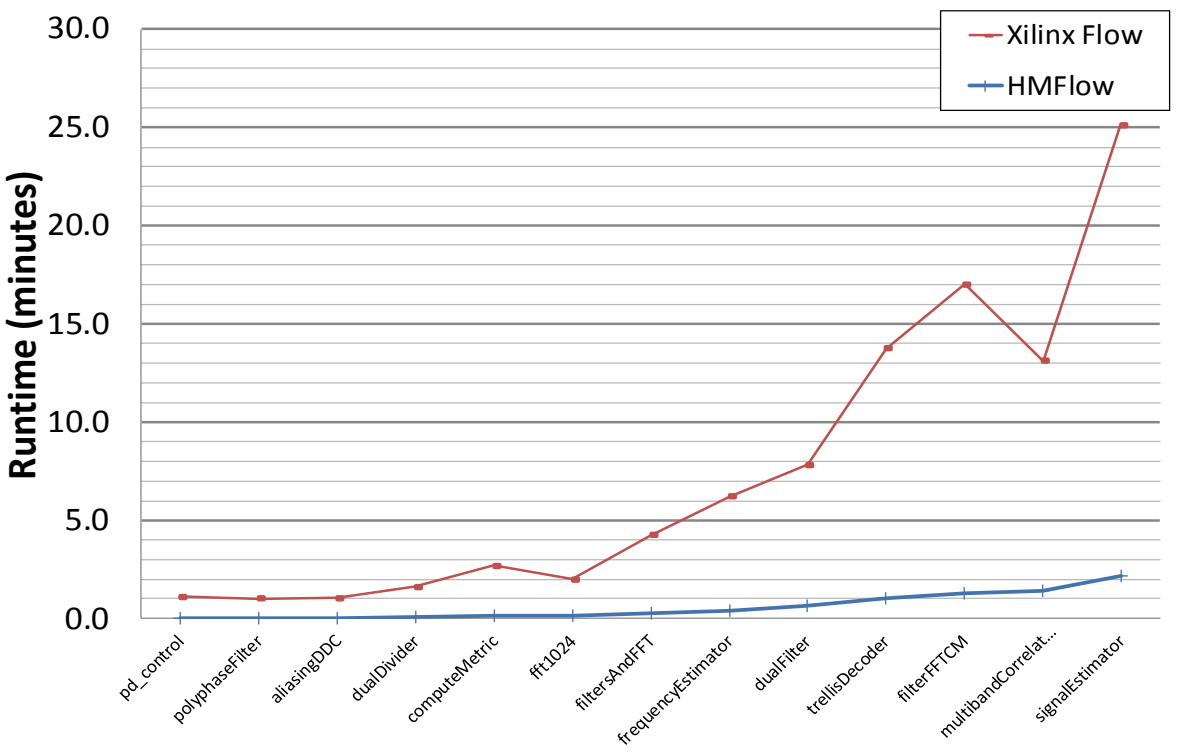
Reuse Module Placement and Routing (B1-11)

Regular Design



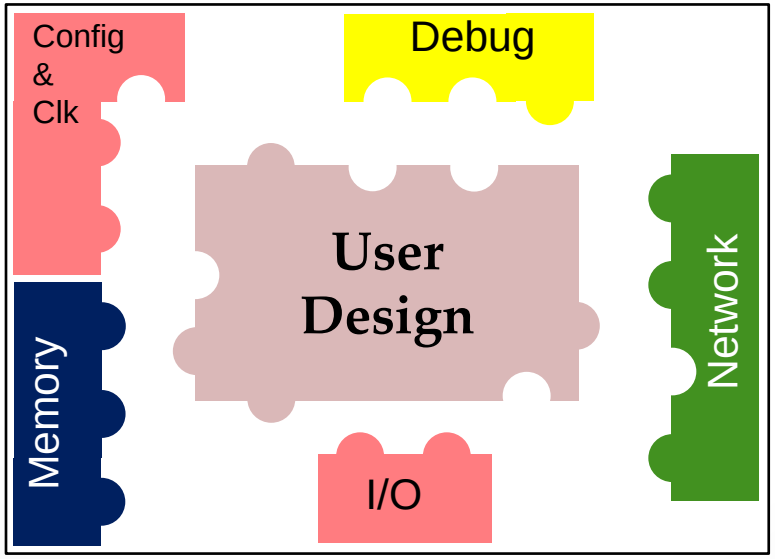
Hard Macro Design





Reuse Bitstreams

- View pre-verified and mapped hardware circuits as reuseable “chip”
- Compose systems through bitstream “plug and play”



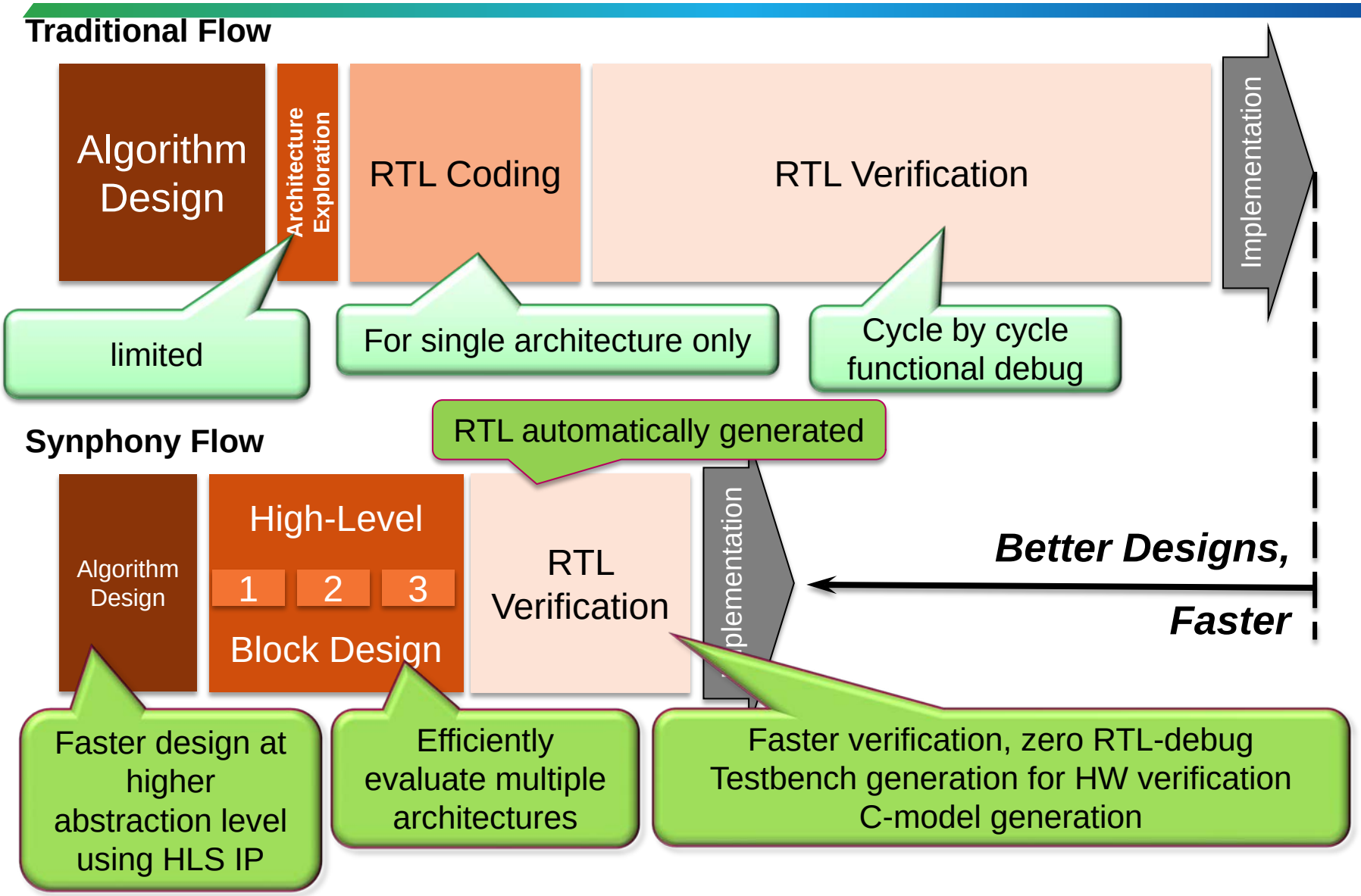
High Performance RTAX-DSP Design Using Synphony

Mir Sayed Ali
Application Engineering
August 2011

Topics

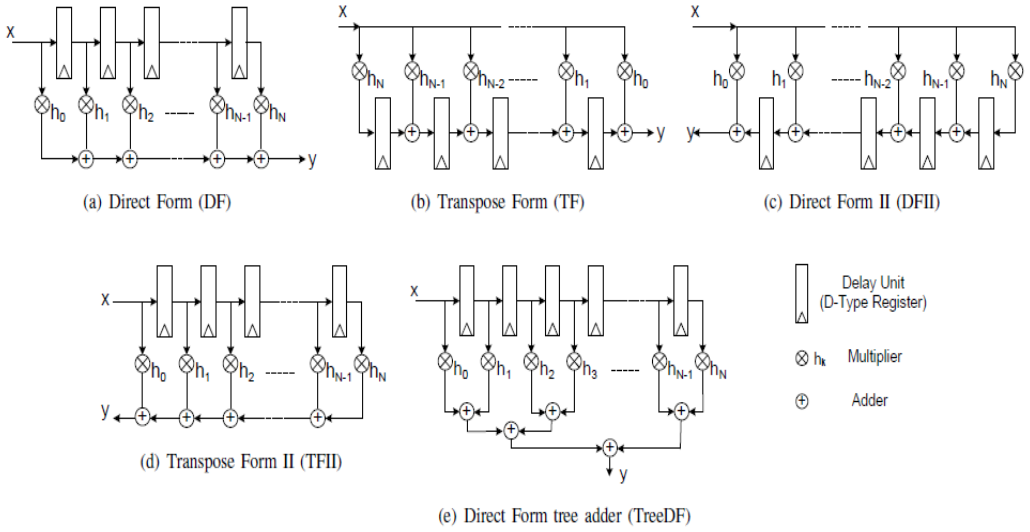
- RTAX-DSP Design Flow Overview
- Analyzing Architecture for Hand-Coded RTL
- Synphony Model Compiler Overview
- RTAX-DSP Design using Synphony AE
- Conclusion

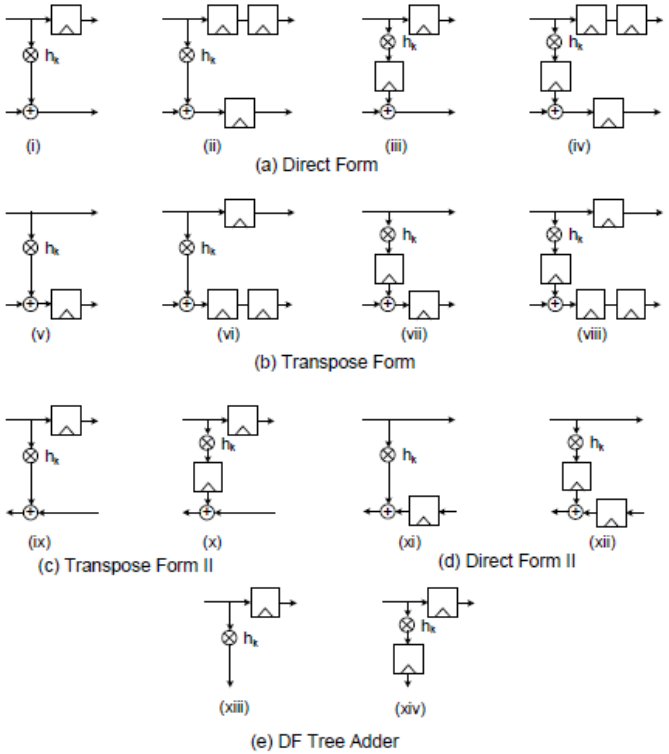
RTAX-DSP Design Flow Overview



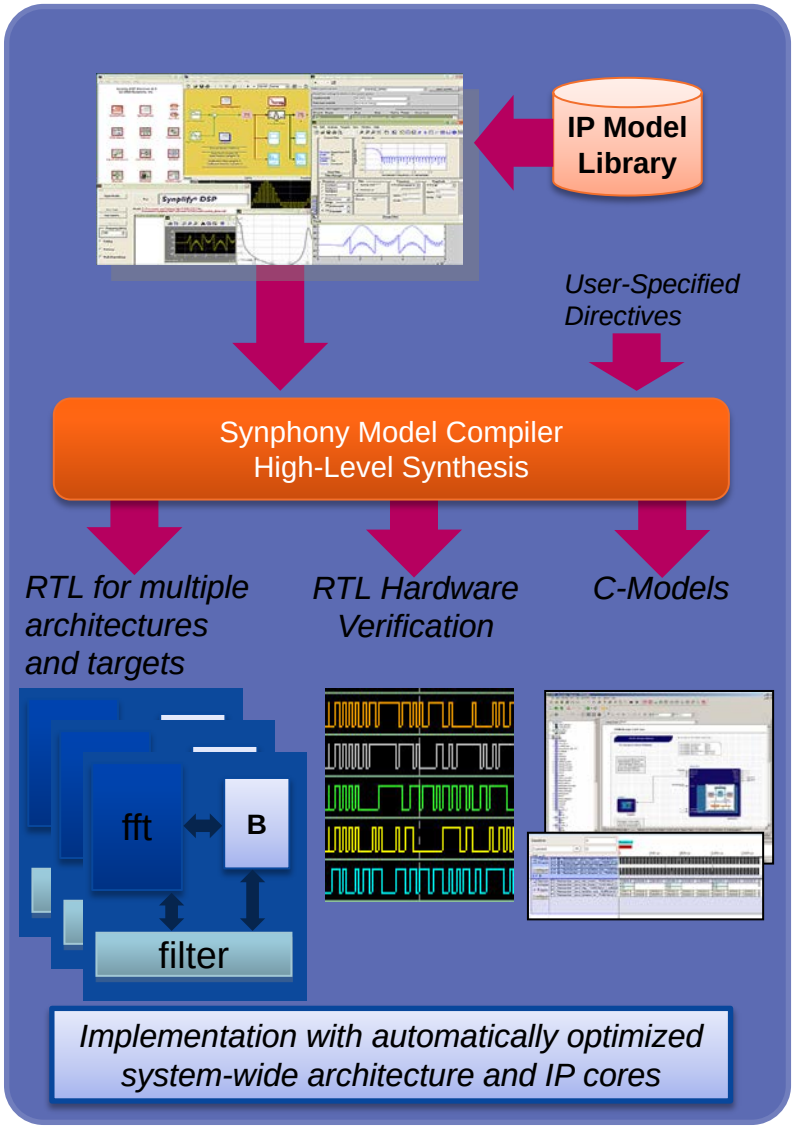
Analyzing Architecture for Hand-Coded RTL

- Hand-coding RTL is time consuming
- Limited ability for the designer to fully explore the design space.
  - Example: A FIR filter can be implemented in various ways and with various pipeline options (Ref: Ramsey Hourani, Ravi Jenkal, W. Rhett Davis, Winser Alexander “Automated Design Space Exploration for DSP Applications” Journal of Signal Processing Systems Volume 56, Numbers 2-3, 199-216, DOI: 10.1007/s11265-008-0226-2)





Synphony Model Compiler Overview



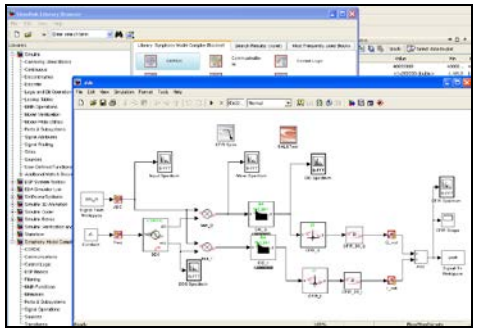
- Quickly create synthesizable multi-rate algorithms using optimized IP model library
- Verify & validate early using Simulink® simulation and debugging
- Globally optimize IP and system architectures using high-level synthesis
- Achieve superior QoR with high quality RTL optimized for ASIC and FPGA

RTAX-DSP Design using Synphony AE

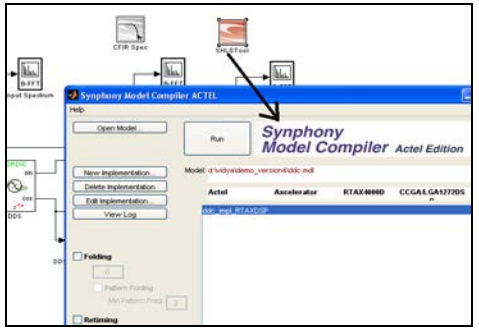
Simulink

Create Design in Simulink Using Microsemi/ Synphony Libraries

- Make sure that all design in-outs are defined with Port In and Port Out blocks from the Synphony blockset
- Simulate and verify the design in Simulink to ensure correct functionality



Add Signal Compiler to Model and create encrypted RTL Code and Testbench for simulation

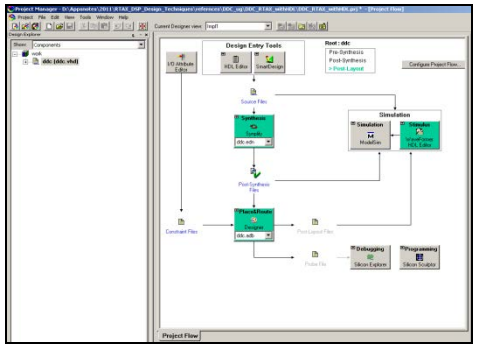


Convert the encrypted rtl to a regular RTL Netlist (Optional)

Libero

Create a Libero project and import Files and perform RTL simulation

- Add other logic/block in the Libero if needed



Synthesize HDL Code
Run Place & Route
Program Device

Conclusion

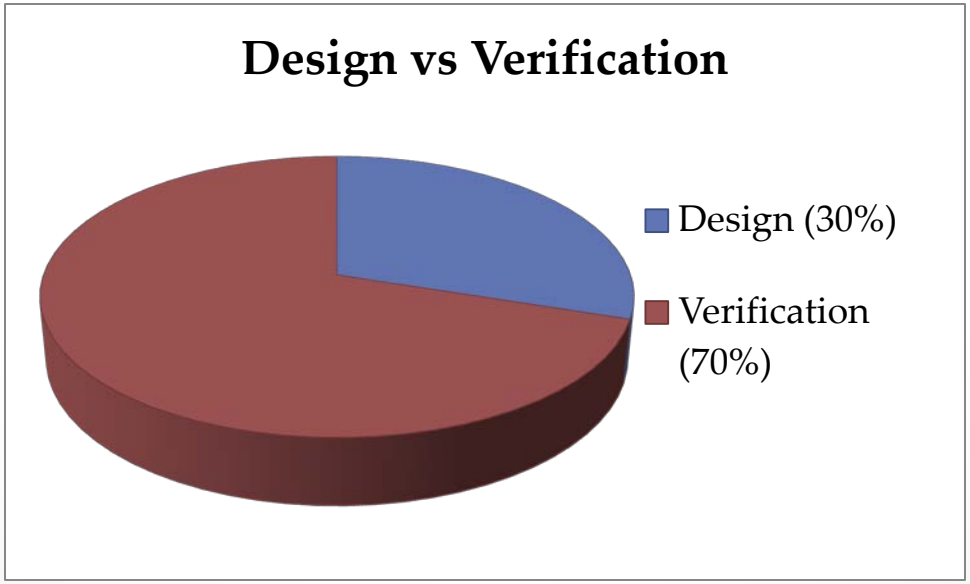
- Synphony allows Superior Simulink implementation flow
  - Quickly create synthesizable multi-rate algorithms
    - May need addition steps to convert encrypted RTL
    - May not always give higher performance compared to Hand_coded RTL
  - Higher capacity and superior optimization technologies for FPGA
  - Tight integration with Synplify (FPGA)
  - Best ease-of-use, portability and also re-use

Functional Verification with SystemVerilog

Panel Member:
Doug Krening
Advanced Verification Consultant

Motivation

- FPGA/ASIC Design and Verification Reality
  - Chip Complexity is Ever Increasing
    - Verification Complexity Growth Outpaces Design
  - High Quality, On Schedule, Within Budget ... or Fail



Project Management View

- Better, Faster, Cheaper – Pick Two?
  - New Technology helps Solve the Dilemma
    - Advanced Functional Verification Languages
    - e -> Vera -> SystemVerilog

The Engineer's View

- ***"Functional verification is a tedious, mind-numbing task."***

  Doug Krening, c2000

- ***"Functional verification is awesome. I love breaking a good design."***

  Doug Krening, c2010

- The Difference? Advanced Verification Languages
  - Old School == Lackluster Engineer
  - Advanced == Enthusiastic Engineer

Organizational Implications

- Requires a Dedicated Verification Organization
  - Engineers, Training, Methodology, etc.

Universal Verification Methodology (UVM) – Taking SystemVerilog to the Next Level

Panel Member:
Michael Horn
Mentor Graphics

Mentor Graphics

Why Use a Standard Methodology?

- SystemVerilog is a huge language
  - Data Types
  - RTL constructs
  - Classes/OOP
  - Assertions

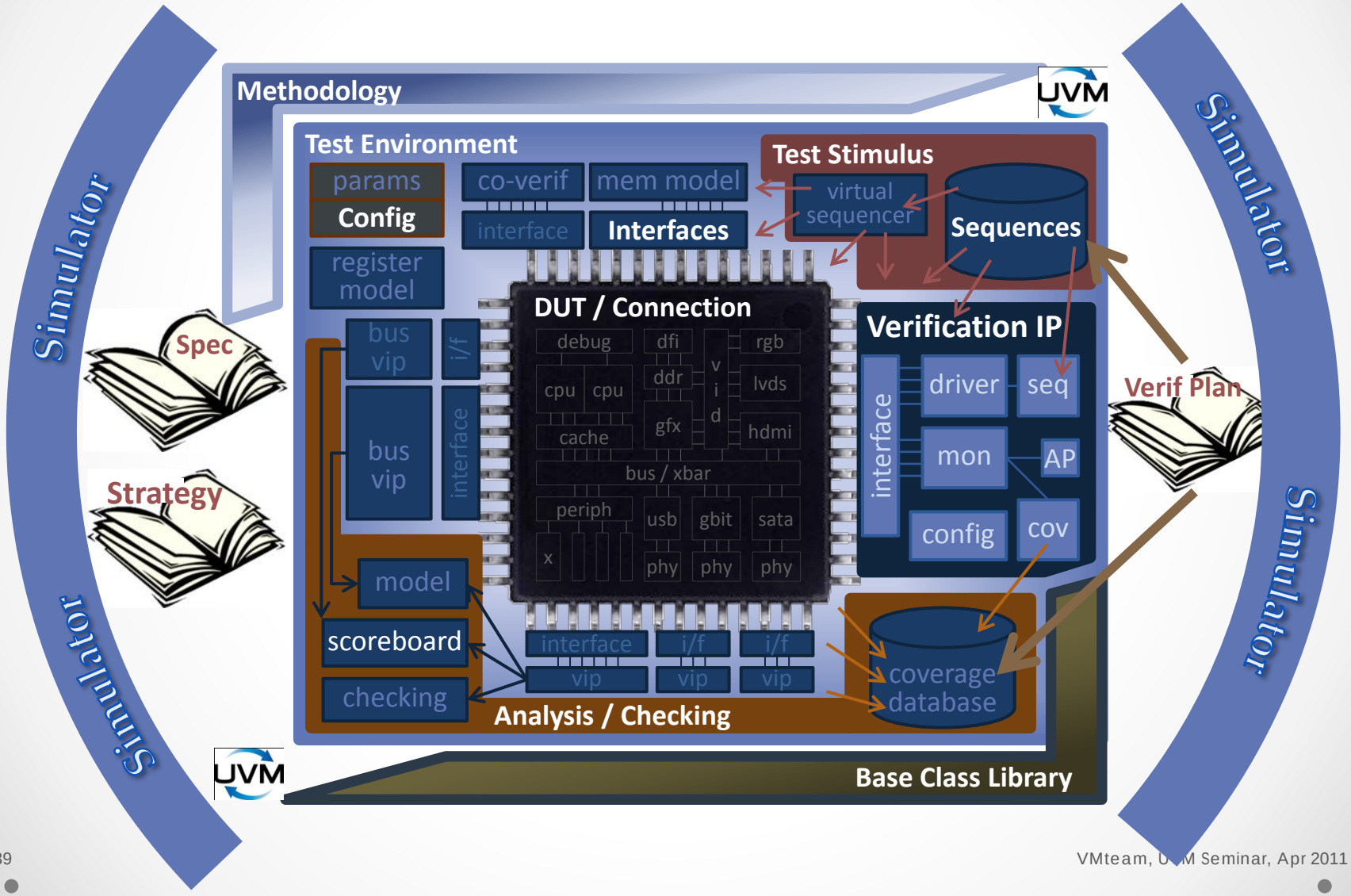
- Need to provide structure and guidance
  - Limit the choices to improve reuse/interoperability
  - Avoid chaos & repetition
  - Provide off the shelf training and support options
  - **Most Important** – Allow people to efficiently work together

What is the UVM?

- Universal Verification Methodology
- Accellera industry standard for verification methodology
  - ARM, Aldec, AMD, Atrenta, **Cadence**, Cisco, Cypress, Duolog, Freescale, IBM, Intel, Jasper, Magillem, **Mentor Graphics**, Nokia, NXP, Oracle, Paradigm Works, Qualcomm, Renesas, Semifore, SpringSoft, ST Microelectronics, **Synopsys**, Texas Instruments, Verilab, Xilinx
- Reference Implementation
  - SystemVerilog Base Classes
  - Based on OVM2.1.1

UVM Foundations

Objective	Justification
• Separation of stimulus generation from delivery	Several people can develop stimulus
• Raise the abstraction level of stimulus and checking	Increase productivity
• Test bench configuration	Avoid expensive recompilation
• Interoperability o Standard class library & API	Important for intra and inter-company development
• Reuse o VIP o Testbench components o Stimulus	Key to productivity


Simulator
Simulator
Simulator
Methodology
UVM
Test Environment
params
Config
co-verif
mem model
interface
Interfaces
register model
Test Stimulus
virtual sequencer
Sequences
DUT / Connection
debug
dfi
rgb
cpu
cpu
ddr
v i d
lvds
cache
gfx
hdmi
bus / xbar
periph
usb
gbit
sata
x
phy
phy
phy
Verification IP
interface
driver
seq
mon
AP
config
cov
Spec
Strategy
Verif Plan
bus vip
i/f
bus vip
interface
model
scoreboard
checking
interface
i/f
i/f
vip
vip
vip
coverage database
Analysis / Checking
UVM
Base Class Library

SYNOPSYS®
Predictable Success

Enabling Pre-Silicon Hardware/Software Validation With Virtual and FPGA Prototyping

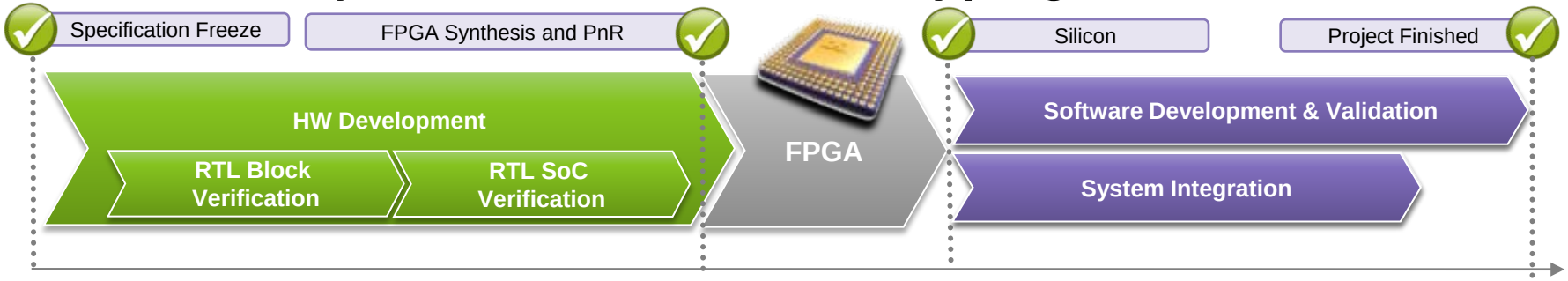
ReSpace/MAPLD
August, 2011

Doug Johnson
Staff Applications Consultant
Synopsys, Inc.

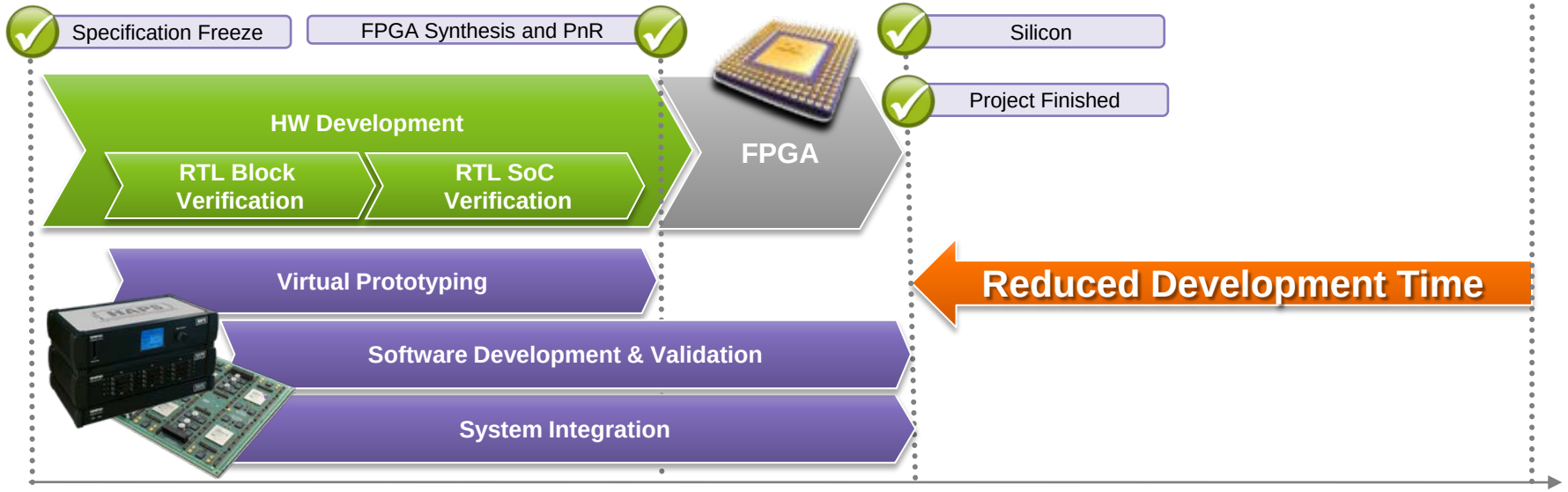
Why Prototype?

Faster HW/SW Integration & System Validation

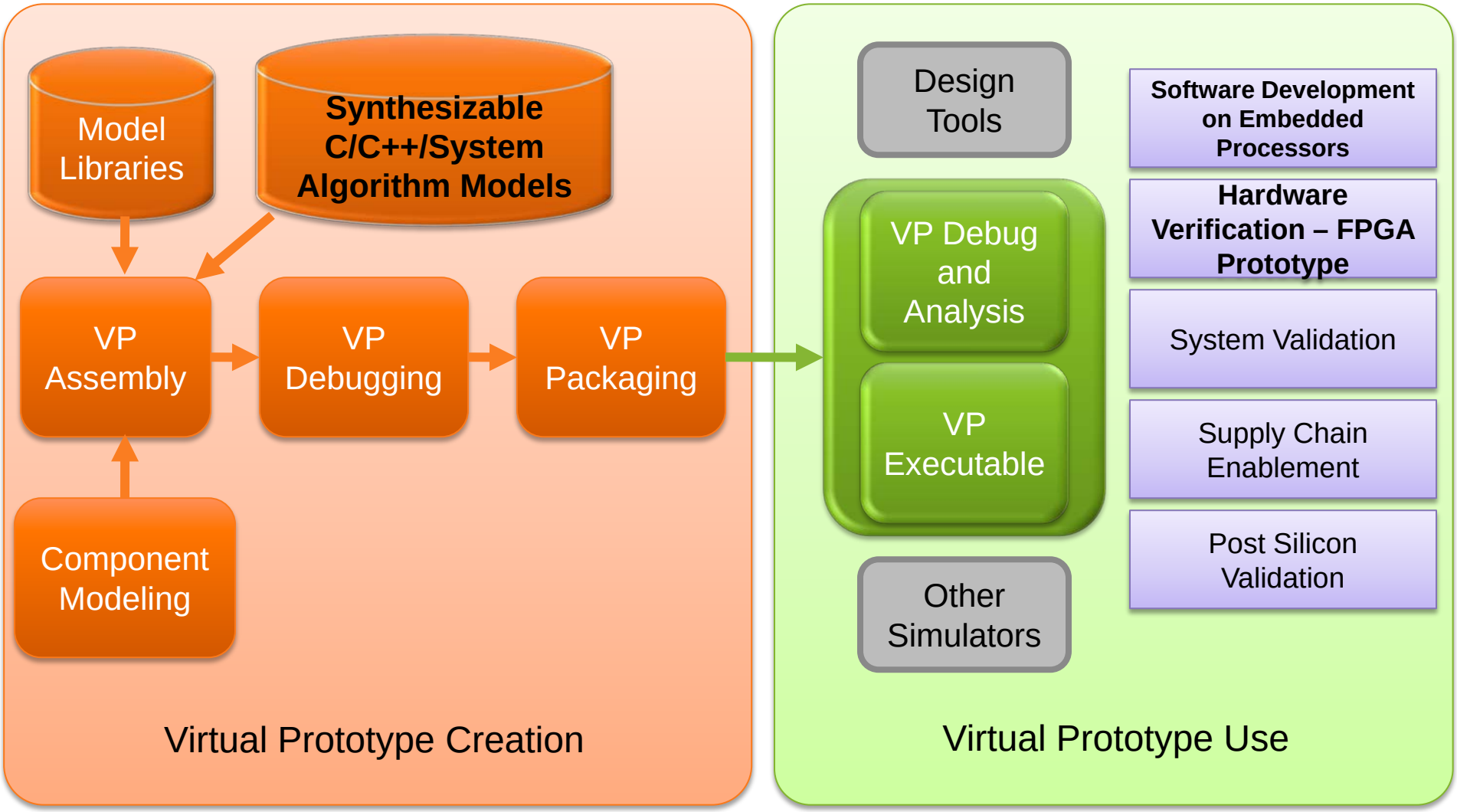
Standard Project Flow Without Prototyping



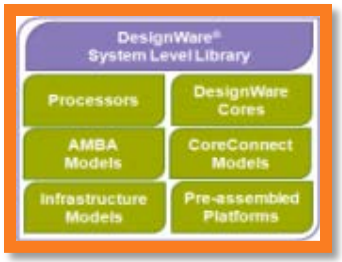
Reduced Development Time with Prototyping



Virtual Prototyping Flow

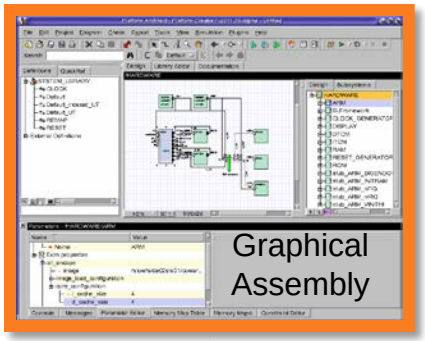


Virtual Prototype *Creation*

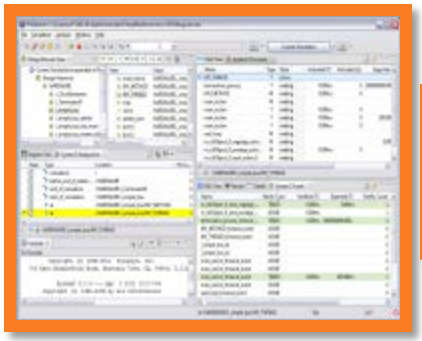


Model Libraries
3rd Party Model Integration

Debug: Source RTL Code and SystemC/TLM Aware



Graphical Assembly



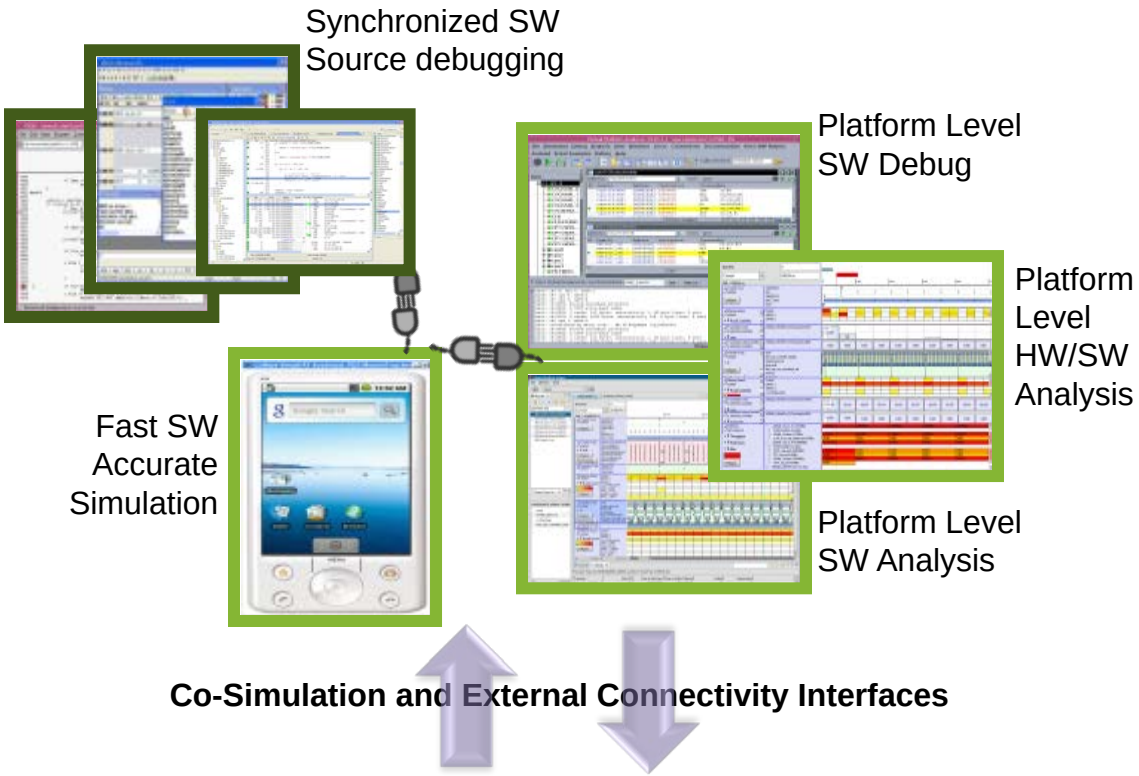
Automated Packaging

High-Level Synthesis - Component Creation

Algorithmic Component Modeling and Implementation Automation

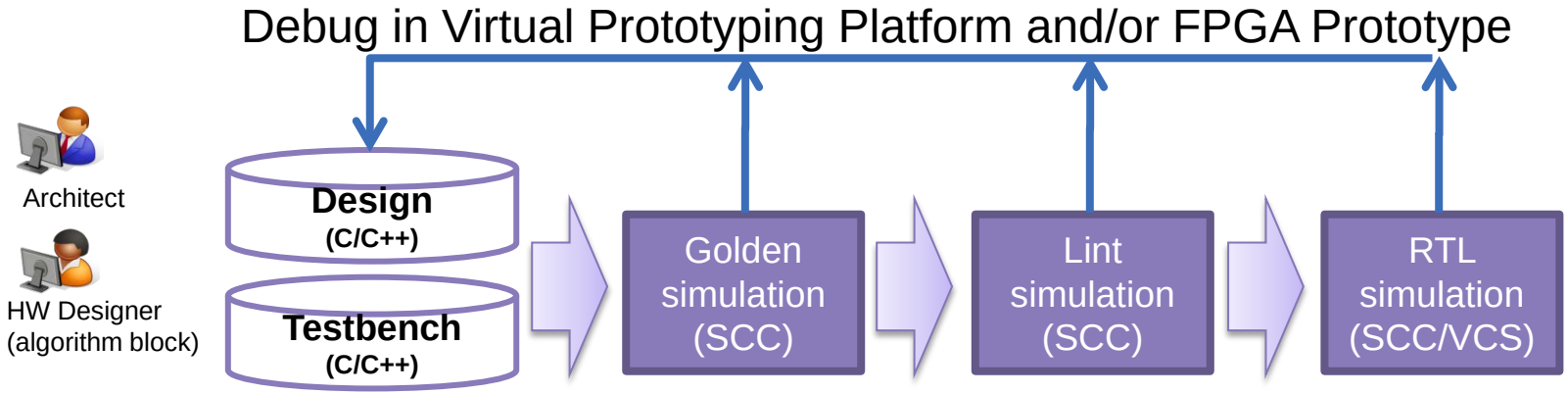
- **Exploit C-Synthesis** for algorithmic, DSP and communications modules
- **Exploit FPGA** processors, interconnect, peripherals and platform models
- Support for standard based technologies **SystemC/TLM 2.0**
- Efficient **graphical assembly** of virtual prototypes
- Support for **fast** simulation at **multiple abstraction levels**

Virtual Prototype *Use*



- Advanced software **debugging and analysis** tool for virtual prototypes
- **Synchronized integration** and execution with 3rd party SW debugging tools (ARM, GDB..)
- RTL Co-Simulation for **system level validation**
- **FPGA Prototyping** for real-world hardware/software validation

Synphony C Compiler Designer for Algorithmic Model Development



- SCC synthesizes untimed C/C++ code into timed RTL code for implementation
- Three verification levels within SCC and Virtual Prototyping tools
  - Golden: Simulates synthesizable C/C++ code and compares to the reference vectors
  - Lint: Checks for common coding errors such as overflow and out-of-bounds
  - RTL: Checks exact performance and verifies results match reference
- Designer can quickly verify system-level test vectors
  - Coarse-level vectors created to verify system-level functionality
  - Designer can quickly additional vectors to apply to RTL and FPGA prototype
- Virtual prototyping tools provide detailed graphical views and reports for debug if any simulation fails at any level of abstraction
  - High-level models
  - RTL
  - FPGA prototype with co-simulation interfaces

Advantages of FPGA and Virtual Prototyping

FPGA Prototyping for At-Speed Testing with Real-World Interfaces

Increased Level of Architectural Exploration

High-Level Synthesis

Synphony C Compiler High-Level Synthesis

Higher Performance & Earlier Validation

Links to Virtual Prototyping

Faster BringUp

Co-Simulation with VCS

Fault Tolerance Verification Through Software Fault Injection

John Paul Walters

USC/ISI

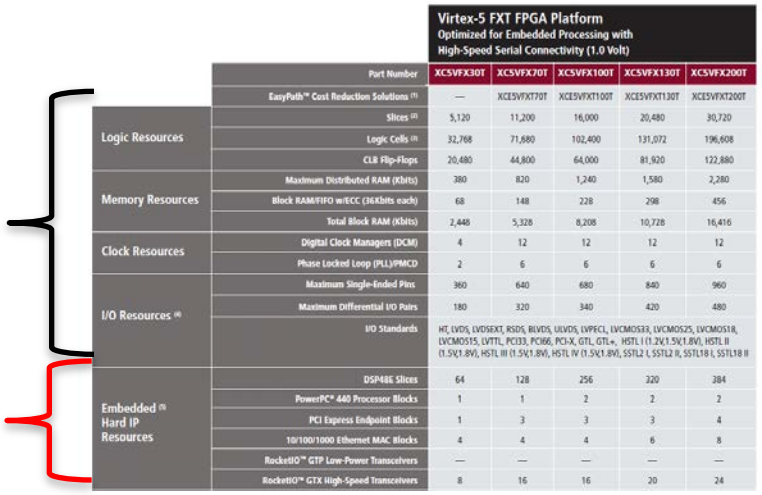
FPGAs Today

FPGAs have evolved, becoming heterogeneous

Hard core processors, Ethernet cores, Giga-bit transceivers

Legacy features (known mitigation techniques)

New features



Virtex-5 FXT FPGA Platform
Optimized for Embedded Processing with High-Speed Serial Connectivity (1.0 Volt)

	Part Number	XC5VFX30T	XC5VFX70T	XC5VFX100T	XC5VFX130T	XC5VFX200T
	EasyPath™ Cost Reduction Solutions (1)	—	XCE5VFXT70T	XCE5VFXT100T	XCE5VFXT130T	XCE5VFXT200T
Logic Resources	Slices (2)	5,120	11,200	16,000	20,480	30,720
	Logic Cells (3)	32,768	71,680	102,400	131,072	196,608
	CLB Flip-Flops	20,480	44,800	64,000	81,920	122,880
Memory Resources	Maximum Distributed RAM (Kbits)	380	820	1,240	1,580	2,280
	Block RAM/FIFO w/ECC (36Kbits each)	68	148	228	298	456
	Total Block RAM (Kbits)	2,448	5,328	8,208	10,728	16,416
Clock Resources	Digital Clock Managers (DCM)	4	12	12	12	12
	Phase Locked Loop (PLL)/PMCD	2	6	6	6	6
I/O Resources (4)	Maximum Single-Ended Pins	360	640	680	840	960
	Maximum Differential I/O Pairs	180	320	340	420	480
	I/O Standards	HT, LVDS, LVDSEXT, RSDS, BLVDS, ULVDS, LVPECL, LVCMOS33, LVCMOS25, LVCMOS18, LVCMOS15, LVTTL, PCI33, PCI66, PCI-X, GTL, GTL+, HSTL I (1.2V,1.5V,1.8V), HSTL II (1.5V,1.8V), HSTL III (1.5V,1.8V), HSTL IV (1.5V,1.8V), SSTL2 I, SSTL2 II, SSTL18 I, SSTL18 II				
Embedded (5) Hard IP Resources	DSP48E Slices	64	128	256	320	384
	PowerPC™ 440 Processor Blocks	1	1	2	2	2
	PCI Express Endpoint Blocks	1	3	3	3	4
	10/100/1000 Ethernet MAC Blocks	4	4	4	6	8
	RocketIO™ GTP Low-Power Transceivers	—	—	—	—	—
	RocketIO™ GTX High-Speed Transceivers	8	16	16	20	24

Xilinx V5FXT Datasheet

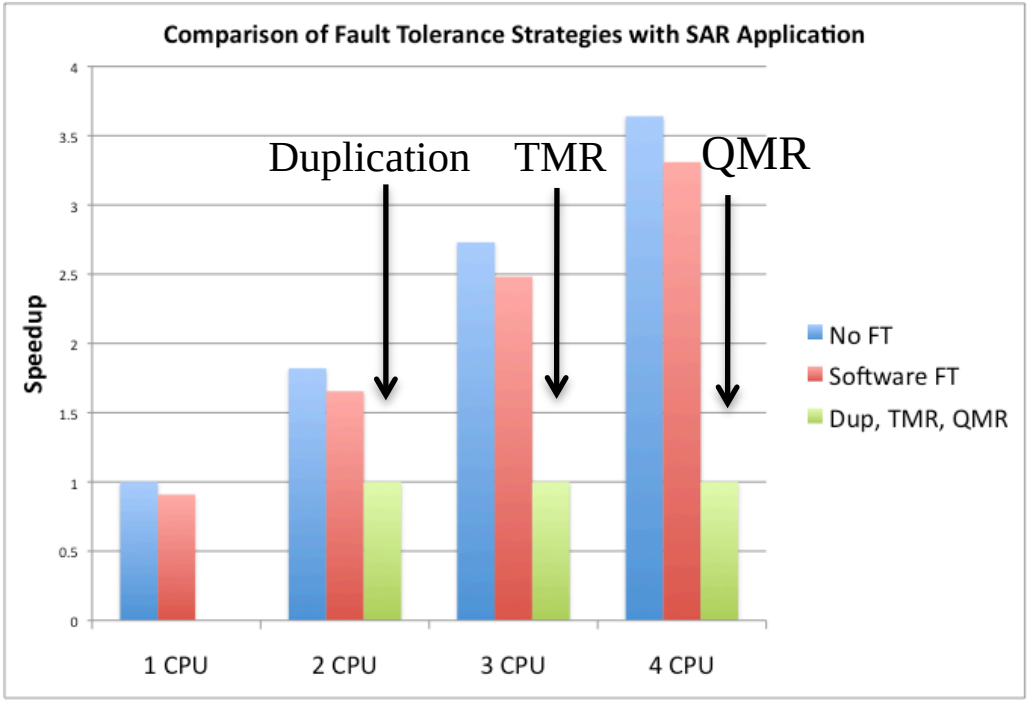
FPGA Embedded PowerPC outperforms radiation hardened RISC processors

Processor	Mongoose V	RAD6000	RAD750	Virtex4 PPC405	Virtex 5 PPC440
Dhrystone MIPS	8	35	260	900	2,200

How can we test the fault response of the embedded PowerPCs?

Software Fault Tolerance vs. Traditional Mitigation

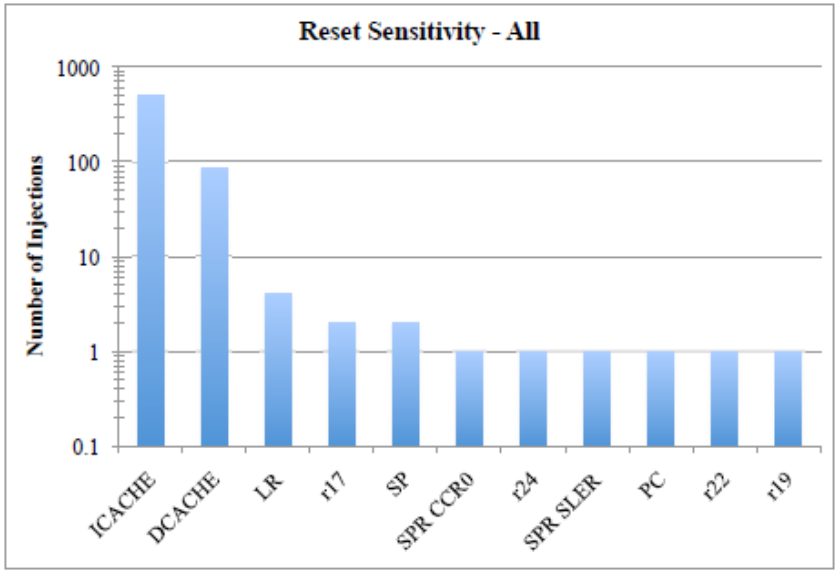
- Software-based approach leverages additional hardware for useful computation
- Heartbeats and assertions cause minimal overhead
- Checkpoints are taken according to the expected upset rate



Software fault tolerance allows more computation and fewer wasted cycles

Evaluating Fault Tolerance

- Several options: radiation testing, laser testing, software fault injection
- Software provides a low-cost way of evaluating fault tolerance at-speed
- We can now inject faults into registers, caches, memories through software



Summary and Open Questions

- Devices and software are becoming more complex
  - Current strategies don't scale
  - Start to push fault tolerance to the application-level
- We can help to provide some fault tolerance constructs
  - Checkpointing, heartbeats, control flow assertions, etc.
- Programmers must leverage application-specific details
  - Improve application efficiency
  - Improved detection
- Software fault injection is becoming more rigorous
  - Complements radiation and laser testing strategies
  - Level of fault detail much higher than radiation and laser testing
  - Inexpensive – can inject faults over days, weeks costing only board time
- Further work is needed to correlate software fault injection to radiation and laser results.