Design Flows for IP Integration: A Tutorial

Grant Martin
Fellow, Cadence Berkeley Labs

Medea+ Design Automation Conference and European EDA Week:
5 November 2003, 10:15

Abstract

• The last few years have seen a considerable evolution in the use and reuse of IP in system and SOC design. We have seen the emergence of a number of different design flows and methodologies, depending on the characteristics both of the IP and the end product. IP may be integrated at many different levels ranging from hard layouts of digital and analogue/mixed-signal cores, through re-synthesis of RTL designs, generation of parameterised implementations or just the reuse of algorithmic IP at the system level. How the IP gets integrated depends on the nature of the overall SoC design process - a single pass ASIC design, block-based integration of IP into an ad-hoc or fixed integration architecture, or perhaps application-oriented platform-based design. This tutorial will give an overview of various approaches for IP integration, and the issues associated with them, not neglecting the importance of the verification flows which are the necessary adjunct to design integration.

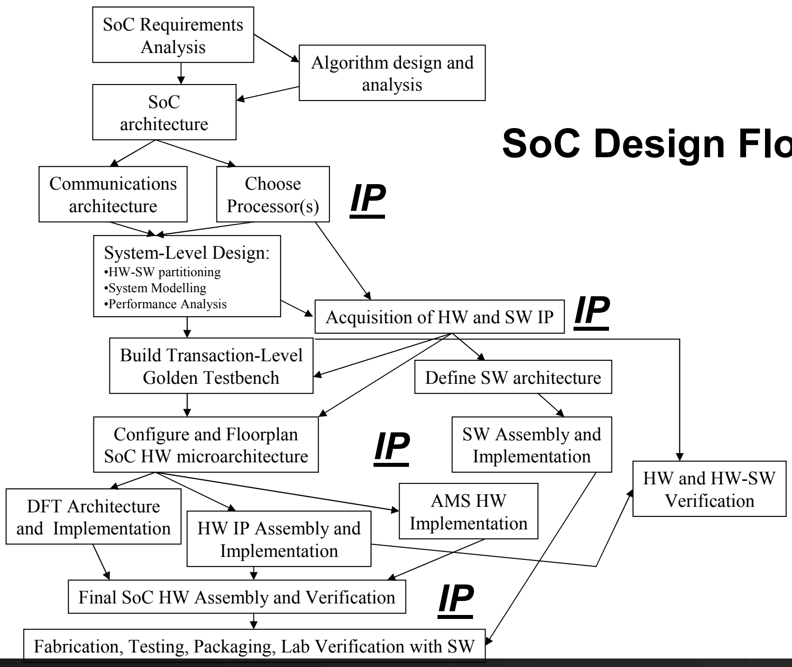
Outline

- The IP Integration Problem
- Integration Architectures
- Platform-Based Design IP Integration Design Flows
- Verification Integration Design Flows
- Physical Integration Design Flows
- The Business of IP Integration

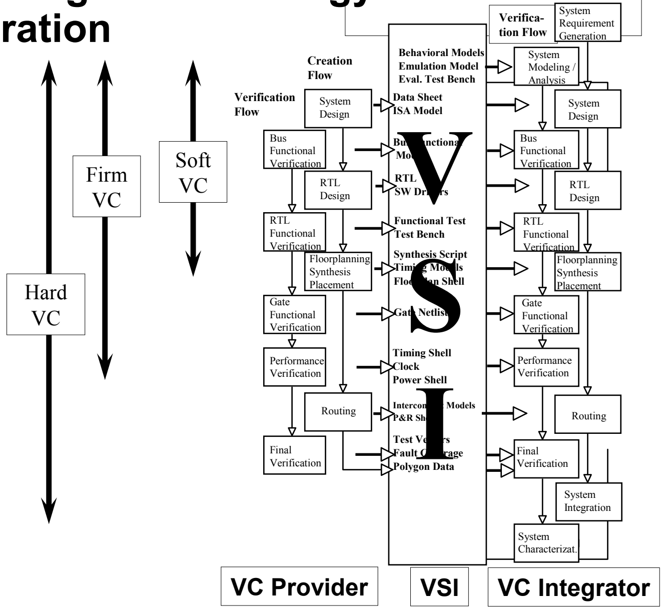
IP Integration is a Question of Style(s)

IP Duality	IP Creator	Socket	IP Integrator
Reuse Style	Ad-hoc Block by Block	Planned Block by Block	SoC Platform
Design Style	Single-pass ASIC, ASSP or Custom	Block Based Design	Platform Based Design
Level	Soft	Firm	Hard
Issues	Control	Time and Space	Economics
AMS	None	AMS Dominant: A/d	Digital Dominant: D/a

SoC Design Flow


SoC Requirements Analysis
Algorithm design and analysis
SoC architecture
Communications architecture
Choose Processor(s)
IP
System-Level Design:
•HW-SW partitioning
•System Modelling
•Performance Analysis
Acquisition of HW and SW IP
IP
Build Transaction-Level Golden Testbench
Define SW architecture
Configure and Floorplan SoC HW microarchitecture
IP
SW Assembly and Implementation
HW and HW-SW Verification
DFT Architecture and Implementation
HW IP Assembly and Implementation
AMS HW Implementation
Final SoC HW Assembly and Verification
IP
Fabrication, Testing, Packaging, Lab Verification with SW

VSI Design Methodology: IP Creation and Integration



Integration Architectures

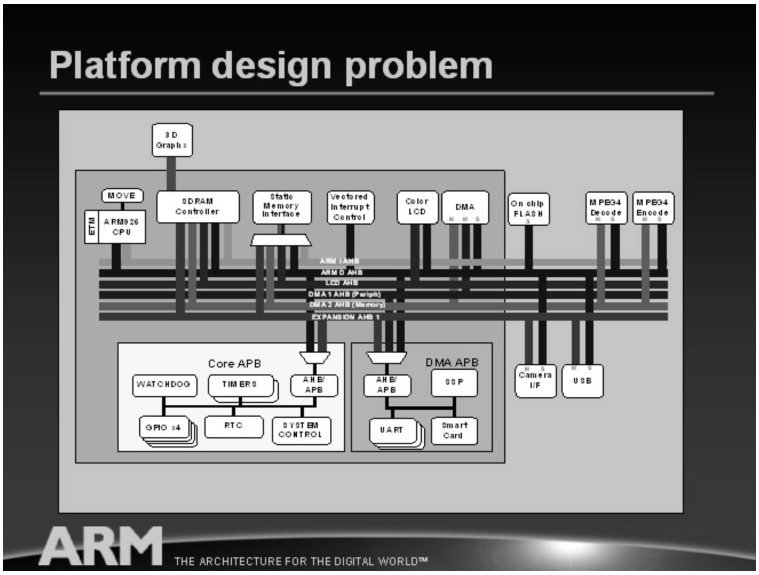
- You'll need
  - an SoC Infrastructure
    - Functional IP
    - Verification IP
    - Interconnect (Bus System) IP
    - ……
  - Global concepts
    - Interrupt System
    - Clocking System
    - Design For Test
    - On Chip Debug System
    - ……
  - Helpful:
    - One company-wide design system that allows reuse of
      - EDA scripts (synthesis,….)
      - Tool specific view libraries
  - Management Tools
    - Bug Tracking System
    - Clear Versioning Process
    - …..

Acknowledgements to Michael Payer, Infineon Technologies AG

Integration Architectures: Levels and Approaches

- By System Model
- By Verification Model
- By Physical Architecture Planning
- By Hard Block
- By Configuration

By System Modelling (SystemC)

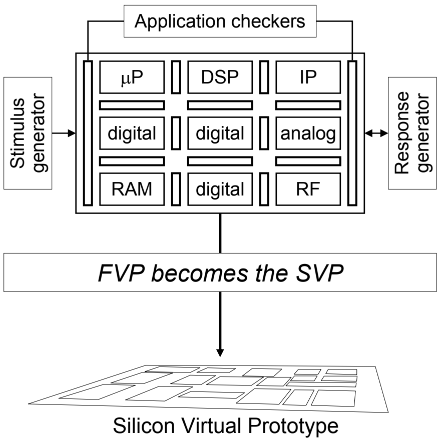


Platform design problem

Source: Jon Connell, ARM: DAC 2002 Open System C Meeting: "Platform Modelling for System Design Using SystemC"

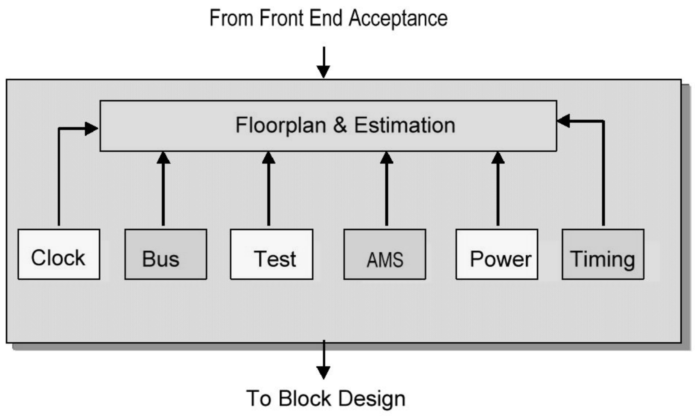
By Verification Modelling (The Functional Virtual Prototype (FVP))

Functional Virtual Prototype



- Executable specification
  - Transaction-level: 100x RTL speed
  - Architectural performance analysis
- Golden verification environment
  - Transaction coverage
  - Block-level reference models
  - Integration vehicle
- Early handoff vehicle
  - Embedded sw development
  - System design-in

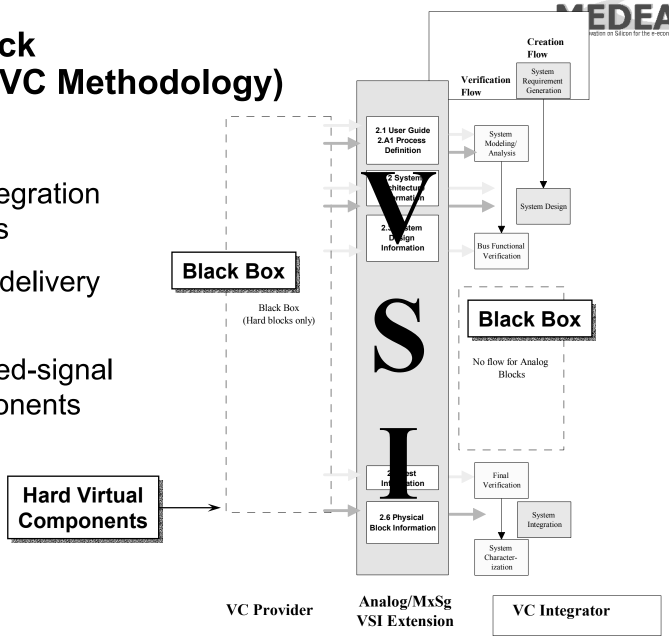
By Physical Architecture Planning (Block-Based Design)



Tasks designed and sequenced to minimize interaction/iteration

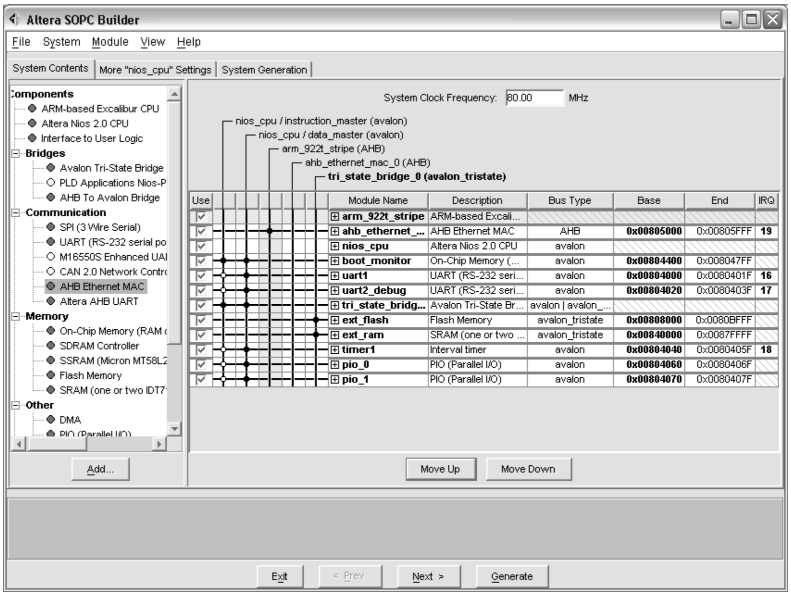
By Hard Block (VSI "Hard" VC Methodology)

- Focus on integration of "hard" VCs
- Standardize delivery mechanism
- Enables mixed-signal virtual components


Creation Flow
Verification Flow
System Requirement Generation
2.1 User Guide 2.A1 Process Definition
2.2 System Architecture Information
2.3 System Design Information
System Modeling/ Analysis
System Design
Bus Functional Verification
V
S
I
Black Box
Black Box (Hard blocks only)
Black Box
No flow for Analog Blocks
2.5 Test Information
2.6 Physical Block Information
Final Verification
System Integration
System Character-ization
Hard Virtual Components
VC Provider
Analog/MxSg VSI Extension
VC Integrator

Acknowledgements to Henry Chang, Cadence

By Configuration: (Altera SOPC Builder)



Components
- ARM-based Excalibur CPU
- Altera Nios 2.0 CPU
- Interface to User Logic

Bridges
- Avalon Tri-State Bridge
- PLD Applications Nios-P
- AHB To Avalon Bridge

Communication
- SPI (3 Wire Serial)
- UART (RS-232 serial po
- M16550S Enhanced UAI
- CAN 2.0 Network Contro
- AHB Ethernet MAC
- Altera AHB UART

Memory
- On-Chip Memory (RAM o
- SDRAM Controller
- SSRAM (Micron MT58L2
- Flash Memory
- SRAM (one or two IDT7

Other
- DMA
- PIO (Parallel I/O)

System Clock Frequency: 80.00 MHz

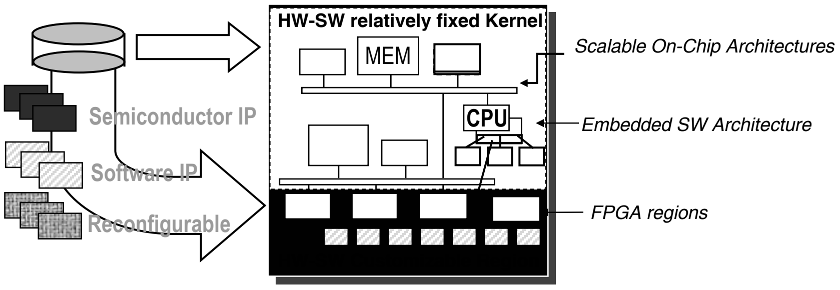
nios_cpu / instruction_master (avalon)
nios_cpu / data_master (avalon)
arm_922t_stripe (AHB)
ahb_ethernet_mac_0 (AHB)
tri_state_bridge_0 (avalon_tristate)

Use	Module Name	Description	Bus Type	Base	End	IRQ
✓	**arm_922t_stripe**	ARM-based Excali...				
✓	**ahb_ethernet_...**	AHB Ethernet MAC	AHB	**0x00805000**	0x00805FFF	**19**
✓	**nios_cpu**	Altera Nios 2.0 CPU	avalon			
✓	**boot_monitor**	On-Chip Memory (...	avalon	**0x00804400**	0x008047FF	
✓	**uart1**	UART (RS-232 seri...	avalon	**0x00804000**	0x0080401F	**16**
✓	**uart2_debug**	UART (RS-232 seri...	avalon	**0x00804020**	0x0080403F	**17**
✓	**tri_state_bridg...**	Avalon Tri-State Br...	avalon | avalon_...			
✓	**ext_flash**	Flash Memory	avalon_tristate	**0x00808000**	0x0080BFFF	
✓	**ext_ram**	SRAM (one or two ...	avalon_tristate	**0x00840000**	0x0087FFFF	
✓	**timer1**	Interval timer	avalon	**0x00804040**	0x0080405F	**18**
✓	**pio_0**	PIO (Parallel I/O)	avalon	**0x00804060**	0x0080406F	
✓	**pio_1**	PIO (Parallel I/O)	avalon	**0x00804070**	0x0080407F	

Source: Altera web site www.altera.com

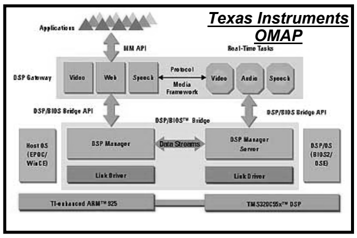
Platform-Based Design Integration Design Flows

- **Platform Based Design** is an organized method to reduce the time required and risk involved in designing and verifying a complex SoC, by heavy reuse of combinations of hardware and software IP. Rather than looking at IP reuse in a block by block manner, platform-based design aggregates groups of components into a reusable platform architecture.

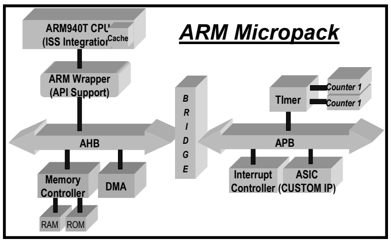


Semiconductor IP can be hard, soft, or firm; analog or digital
Software IP can be source or object

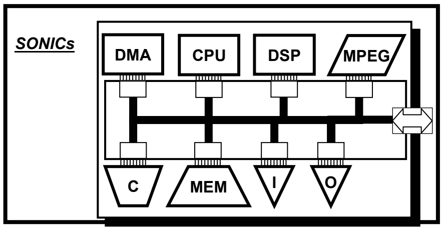
Platform Alternatives



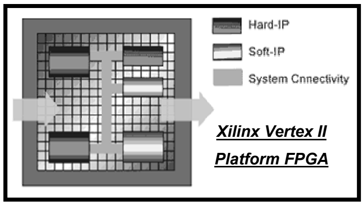
Full Application



Processor-Centric

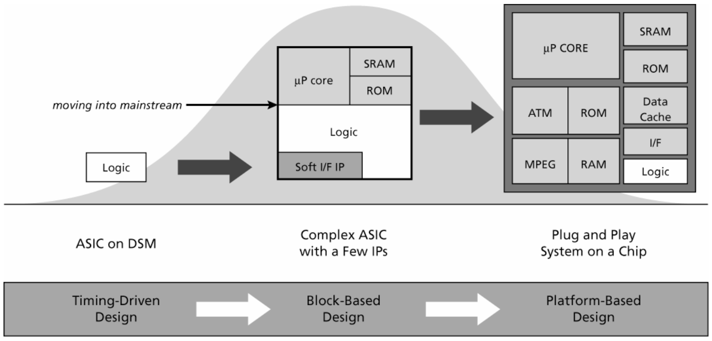


Communications-Centric



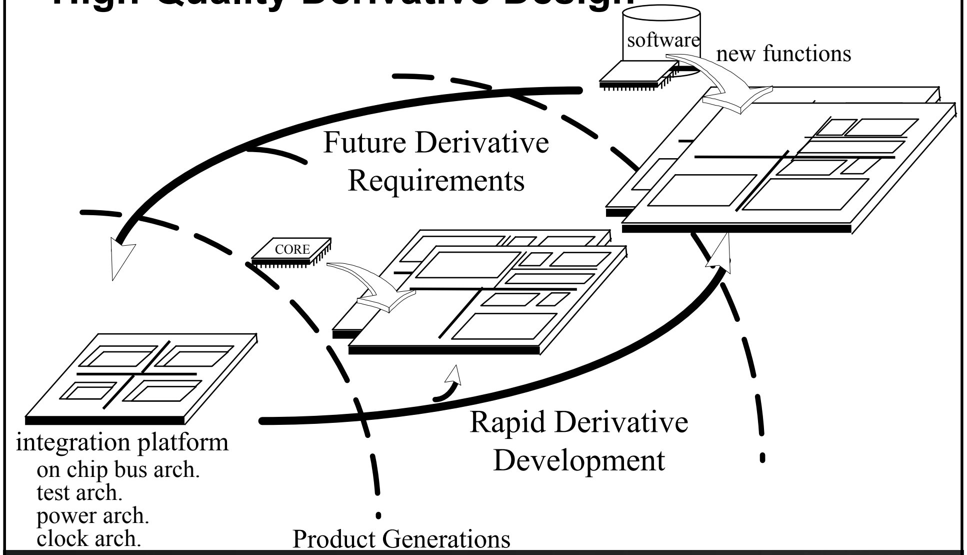
Highly-Programmable

Where Did Platform-Based Design Come From?



- For SoC's, Platform-Based Design is the ***next logical evolution*** in Design Reuse.
- In TDD, Reuse in ASIC design is of ***Cell-level Libraries***
- In BBD, Reuse in hierarchical design is of ***major IP Blocks*** (e.g., digital blocks built out of standard cells)
- In SOC, Reuse is of ***Collections of IP blocks*** organised into HW-SW architectures: also known as ***Integration Platforms***

Motivation: Rapid, Low-Risk, High-Quality Derivative Design



THE PLATFORM DESIGN CHAIN

- The platform creator and user can be different depending on the composition of the platform

Platform Creator	Platform User
ARM PrimeXsys (general purpose)	Sanyo, STMicroelectronics,
TI OMAP (portable multimedia)	Acer, Ericsson, Nokia, Sony, TI, Handspring
Philips Nexperia (multimedia)	Philips Electronics, Acer
Infineon Wireless	eAnywhere
Motorola Wireless i.250	RTX Telecom, Solomon Group, Giga Telecom, Benq, Eastcom, Compal Communication
Intel Xscale (general purpose)	Philips Electronics, Viewsonic Corporation, Microsoft PocketPC
Xilinx Vertex II	unknown

IP Creator owns platform
Platform User is IDM & System

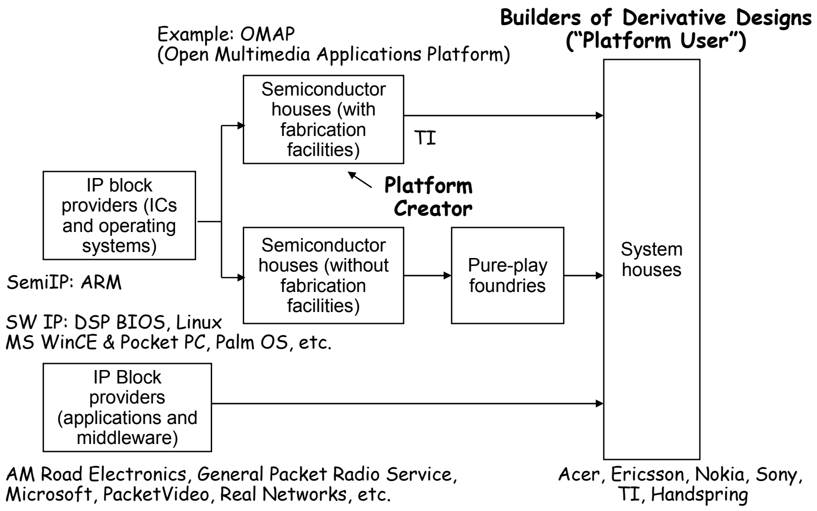
IDM owns platform
Platform User is themselves & System

IDM owns platform
Platform User is System

IDM owns platform,
Platform user is anyone

There could also be a software-level platform, e.g. Palm

DESIGN CHAIN AND PLATFORM EXAMPLE


Example: OMAP
(Open Multimedia Applications Platform)
Builders of Derivative Designs
("Platform User")
Semiconductor houses (with fabrication facilities)
TI
Platform Creator
IP block providers (ICs and operating systems)
Semiconductor houses (without fabrication facilities)
Pure-play foundries
System houses
SemiIP: ARM
SW IP: DSP BIOS, Linux
MS WinCE & Pocket PC, Palm OS, etc.
IP Block providers (applications and middleware)
AM Road Electronics, General Packet Radio Service, Microsoft, PacketVideo, Real Networks, etc.
Acer, Ericsson, Nokia, Sony, TI, Handspring

Source: Martin, G.; Schirrmeister, F., "A design chain for embedded systems," *IEEE Computer magazine*, Volume: 35 Issue: 3, 3/2002

Platform User Types – Impact on IP-Based Design Flows

"Power User"

– differentiates at all levels – software and hardware

– Develops additional custom hardware and software components

"Platform Differentiator"

– differentiates at the application level

– develops processor Application Software

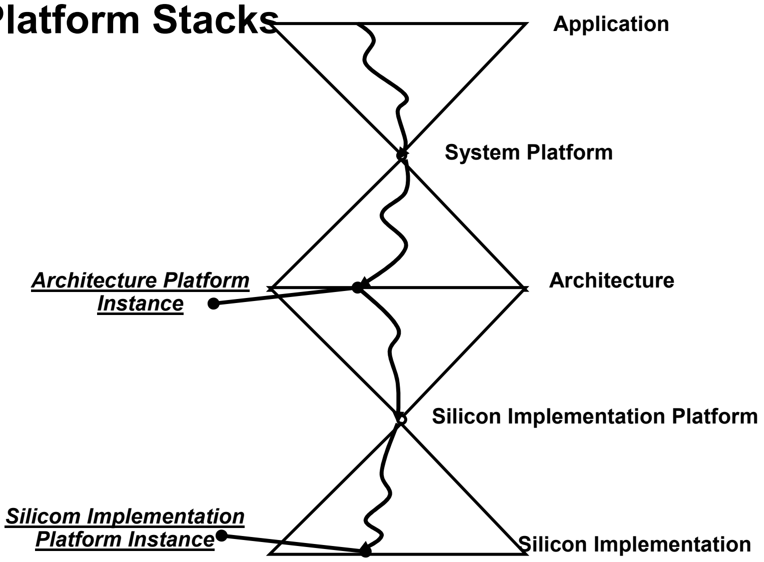
– Uses existing libraries as hardware accelerators

"Complete Package User"

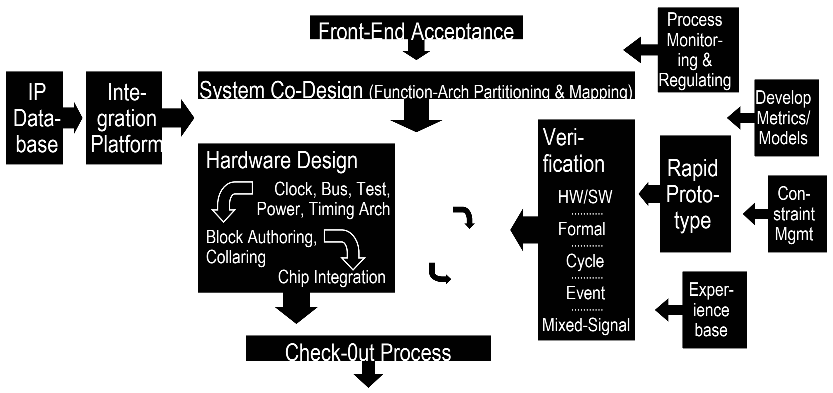
– expects complete solution (hardware and software)

– limited additional development and differentiations

Platform Design Methodologies: Platform Stacks

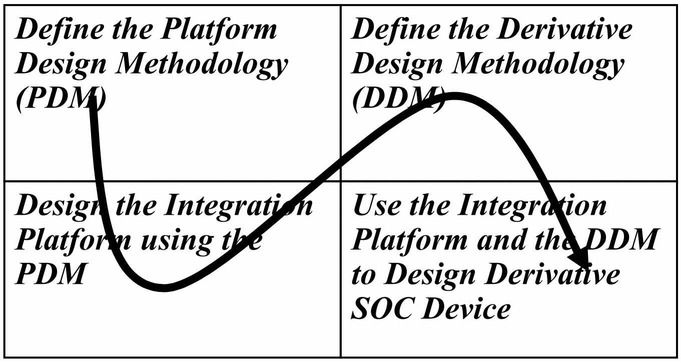

Application
System Platform
Architecture Platform Instance
Architecture
Silicon Implementation Platform
Silicom Implementation Platform Instance
Silicon Implementation

Platform-Based Design Extends an SOC Design Methodology



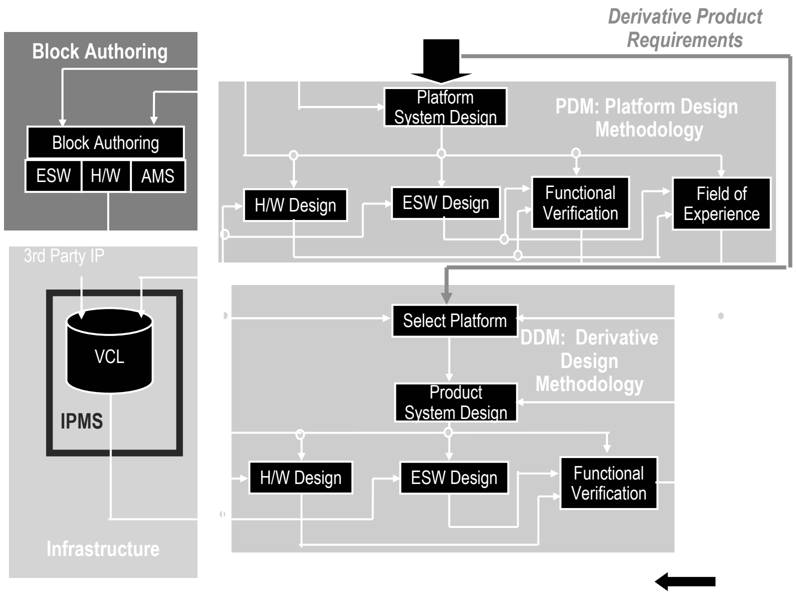
- Additional, incremental IP Design and Integration Issues

Platform-based Methodology for SoC Design

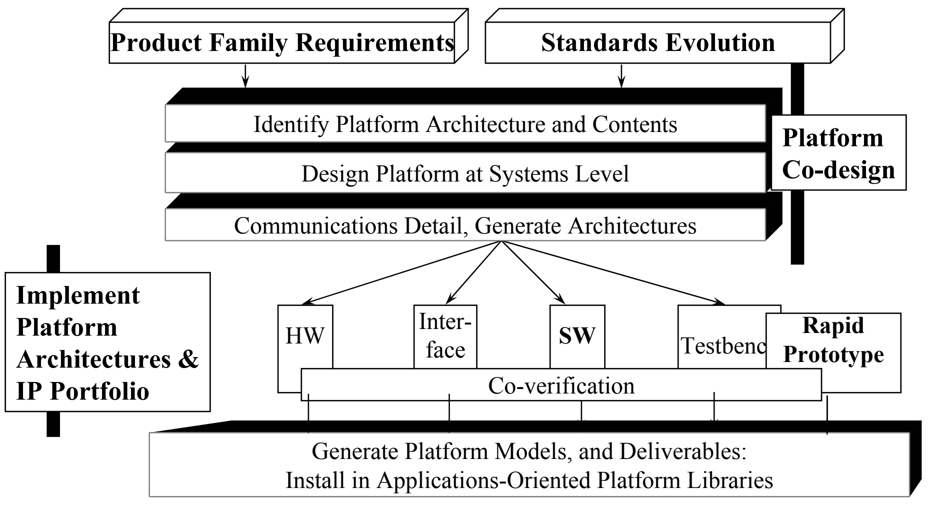


Define the Platform Design Methodology (PDM)	*Define the Derivative Design Methodology (DDM)*
Design the Integration Platform using the PDM	*Use the Integration Platform and the DDM to Design Derivative SOC Device*

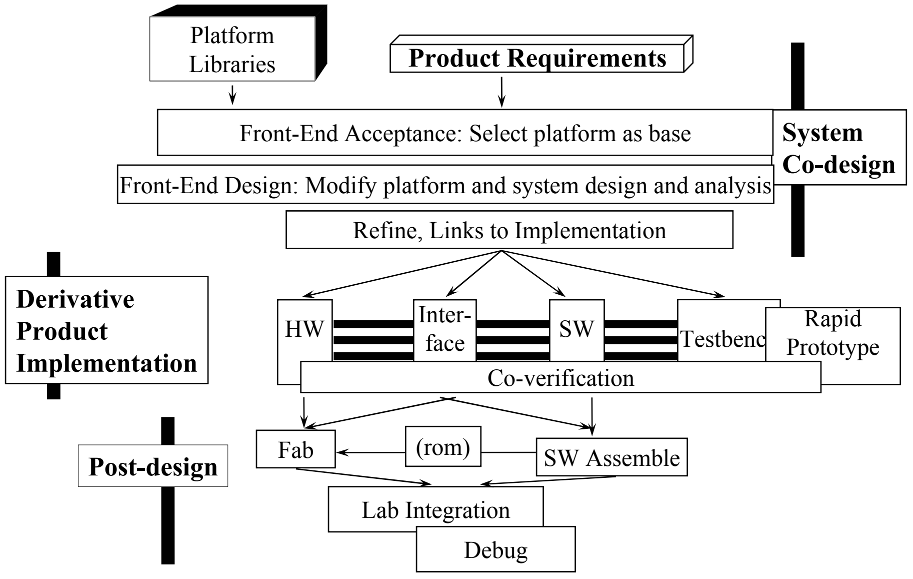
Platform-Based SOC Methodology



Platform Design Methodology (PDM)


Product Family Requirements
Standards Evolution
Identify Platform Architecture and Contents
Design Platform at Systems Level
Communications Detail, Generate Architectures
Platform Co-design
Implement Platform Architectures & IP Portfolio
HW
Inter-face
SW
Testbenc
Rapid Prototype
Co-verification
Generate Platform Models, and Deliverables: Install in Applications-Oriented Platform Libraries

Derivative Design Methodology (DDM)


Platform Libraries
Product Requirements
Front-End Acceptance: Select platform as base
Front-End Design: Modify platform and system design and analysis
Refine, Links to Implementation
System Co-design
Derivative Product Implementation
HW
Inter-face
SW
Testbenc
Rapid Prototype
Co-verification
Post-design
Fab
(rom)
SW Assemble
Lab Integration
Debug

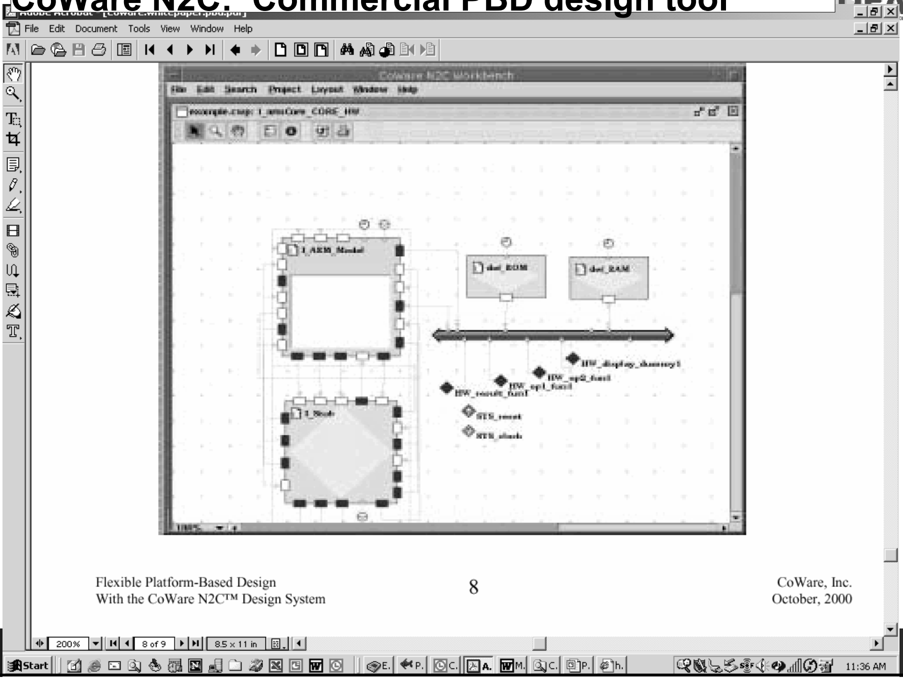
What is Needed to Support Platform-Based Design?

To be usable, a platform at any level exists as a black box, with:

- a real implementation;
- a definable, complete architectural description (AD) (at least derivable from the implementation);
- a complete and accurate set of models describing its actual behavior (this may be redundant with the AD, or the AD may call for more models than yet exist);
- a set of tools to permit integration of the platform model into the model of a higher level system;
- a set of tools to permit integration of the real platform implementation into the implementation of a higher level system.

Source: VSIA Platform-Based Design Study Group, January 2002

CoWare N2C: Commercial PBD design tool

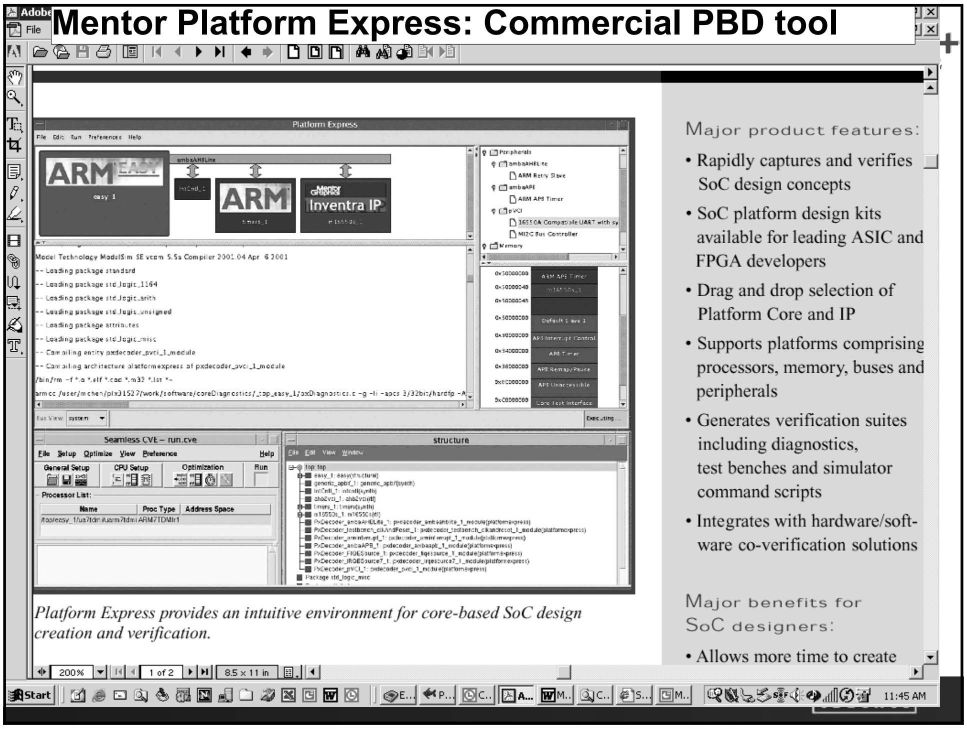


Flexible Platform-Based Design
With the CoWare N2C™ Design System

8

CoWare, Inc.
October, 2000

Mentor Platform Express: Commercial PBD tool



```
Model Technology ModelSim SE vcom 5.5a Compiler 2001.04 Apr  6 2001
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package std_logic_arith
-- Loading package std_logic_unsigned
-- Loading package attributes
-- Loading package std_logic_misc
-- Compiling entity pxdecoder_pvci_1_module
-- Compiling architecture platformexpress of pxdecoder_pvci_1_module
/bin/rm -f *.o *.elf *.cod *.m32 *.list *~
armcc /user/mchen/plx31527/work/software/coreDiagnostics/_top_easy_1/pxDiagnostics.c -g -li -apcs 3/32bit/hardfp -A
```

Platform Express provides an intuitive environment for core-based SoC design creation and verification.

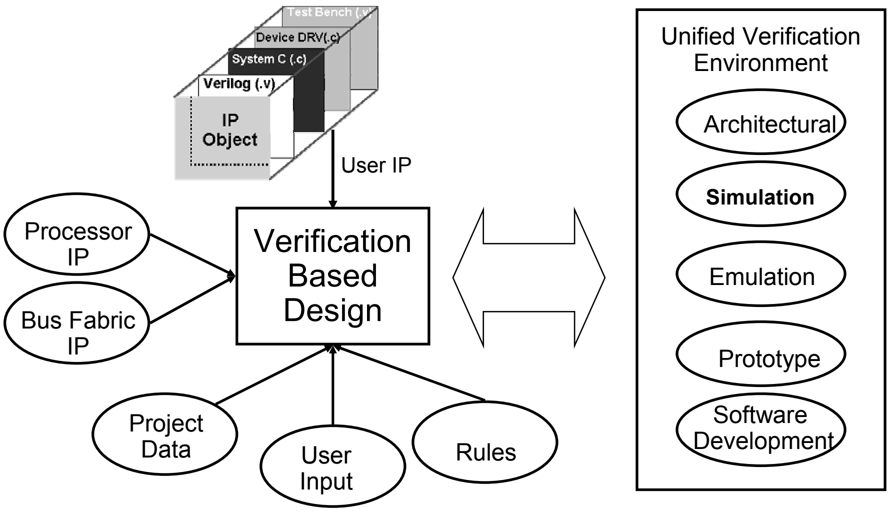
Major product features:

- Rapidly captures and verifies SoC design concepts
- SoC platform design kits available for leading ASIC and FPGA developers
- Drag and drop selection of Platform Core and IP
- Supports platforms comprising processors, memory, buses and peripherals
- Generates verification suites including diagnostics, test benches and simulator command scripts
- Integrates with hardware/software co-verification solutions

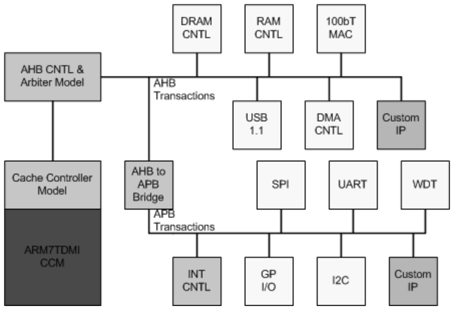
Major benefits for SoC designers:

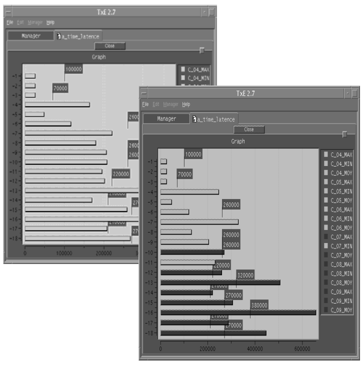
- Allows more time to create

Verification IP Integration Based Design



FVP Features: *Design Space Exploration*



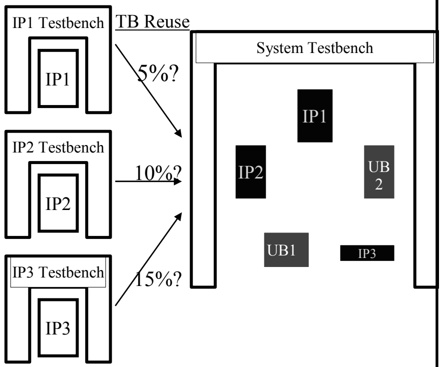


Programmable Sweep of Parameters
- Cache Memory Size
- Number of DMA Channels
- FIFO/Buffer depths for custom blocks
- Power Estimation
- Die size
- Etc.

Parameter Sweep Results
Increase Memory Size – See Results

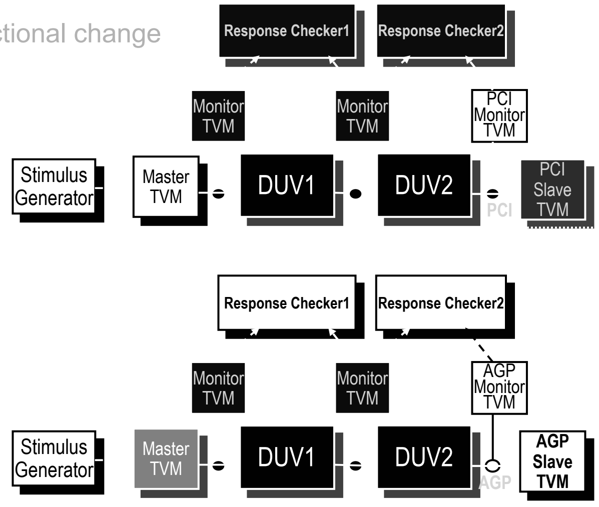
Functional Verification

- A verification methodology optimised for verification-based that:
  - Increases reuse of functional testbench components
    - From block through sub-system to chip and full system
    - From one design to derivative designs
  - Establishes functional coverage criteria
    - using transaction level coverage metrics
  - Improves debug time
    - through transaction level debug

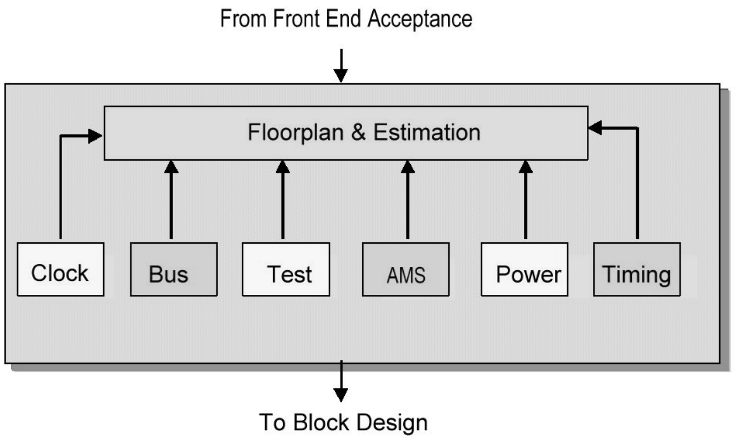


Derivative Design Verification

- An interface change
  - No functional change

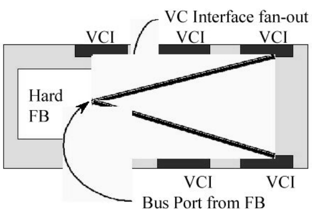


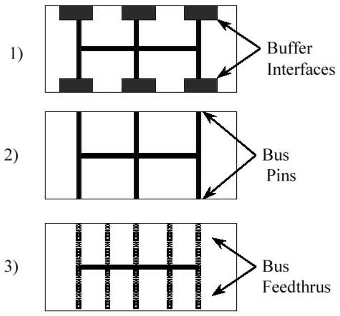
Physical Integration Design Flows: a harmonious variety of implementation architectures



Bus (On-Chip Communications Network) Planning

- Possible Hierarchy of On-Chip Buses:
  - System
  - Processor Sub-system
  - Peripheral
- Use of standardised bus architectures
  - Interface Wrappers: e.g. VSIA Virtual Component Interface (VCI)
  - Physical implementation of bus architecture has performance impacts – e.g. invariant timing using fixed buffer interfaces
- Separation of Kernel (FB) from buses with bridges and interfaces reduces implementation and verification effort
- Bus hierarchy matches bandwidths and latency requirements to IP block needs





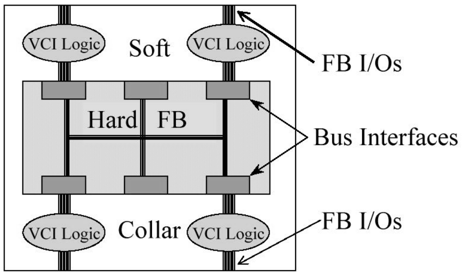
Timing and Clocking Architecture

- Three types of clock domains are typical
  - System clock domain - fastest requirements
  - Processor clock domains for each processor subsystem
  - Peripheral clock domain (standard bus)
  - Others might include asynchronous clock domain for peripherals and additional bus domains
- Clock Gating
  - Power reduction – either by slow-down of processing where possible or power-down of whole sub-systems when idle (dynamic or statically scheduled)
  - Under system or software control
- Compatibility of clock domains
  - A variety of methods to ensure synchronisation of clock domains
  - For example, “13” is a magic number in GSM systems

One SoC Clocking architecture concept

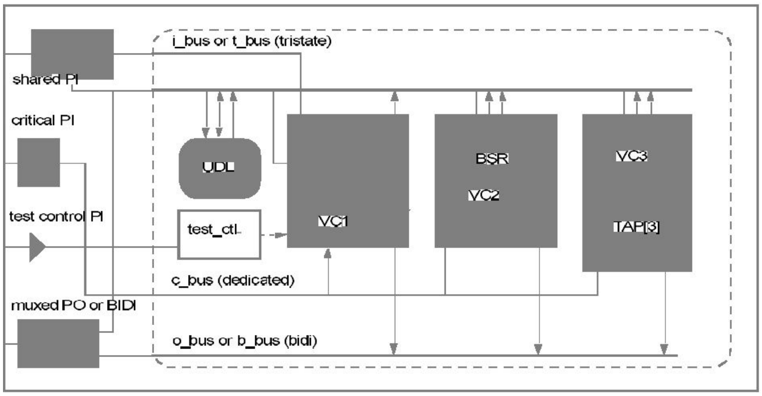
- Allow all components of an SoC to run with an individual speed in a purely synchronous design
- Implementation via a decentralized clock gating concept and a single central clock source
  - Decentralized concept offers greater flexibility than a purely central approach
- Basic to this concept
  - 1 unique clock running with the highest frequency (system clock) used inside the SoC
  - Routed as a balanced clock tree all over the chip
  - System clock is assumed to be used for synthesis of all blocks
- Every component of the SoC (CPU, bus, peripherals) derives its clocks from this system clock
  - by pulse swallowing
  - using clock gating cells

Physical Layout Architecture Using a "Foundation Block Structure"



- Bus interface buffers in hard portion of foundation blocks (fixed IP kernel)
- Foundation block collar contains assigned Virtual Component interface logic
- VC Interface pins must be relocate in collar

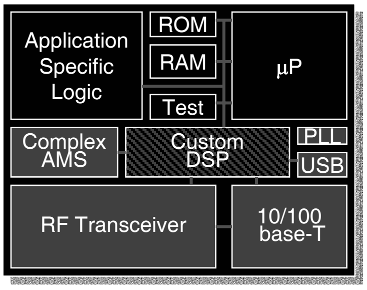
Hybrid DFT Architecture



- Scan
- BIST
- ATPG
- Functional
- Legacy

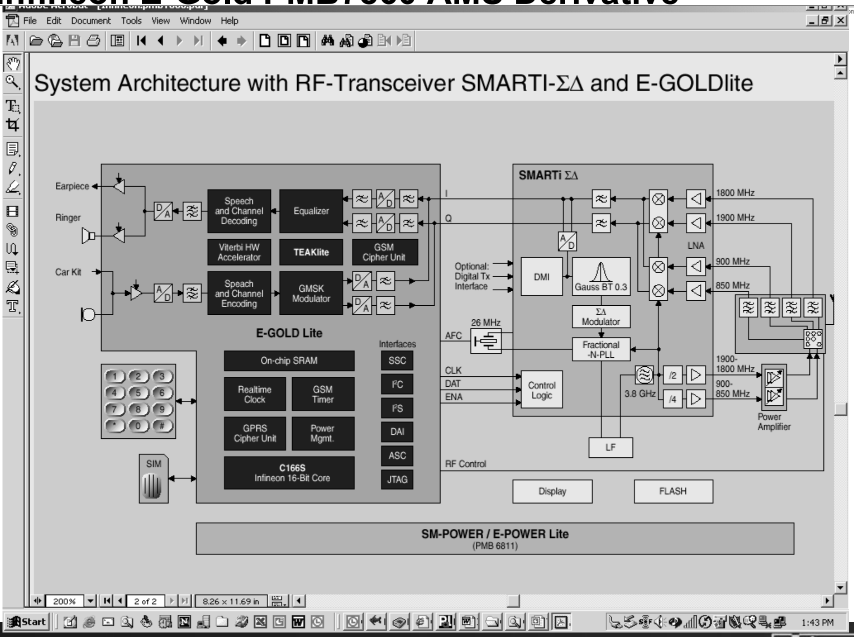
- Interconnected Using JTAG 1149.1 interfaces into a Hybrid Test Architecture
- IP Blocks may use individual test methodologies but they are all interconnected into the standard SoC test architecture using the common interface

The AMS SoC Architecture



- New kinds of SoCs
  - AMS blocks *cannot* be treated as black boxes
  - Large AMS content
  - Constraints imposed from AMS design are strong
  - IC Design controlled by the *analog* designer, who “owns” the chip and its integration
- Requires digital (SP&R) technologies- treated as a black box
- Can call this “A/d” SoC as opposed to “D” or “D/a”

Infineon E-Gold PMB7860 AMS Derivative



Chip Planning: Basic Guidelines for incorporating AMS IP

- Controlling substrate noise
- Controlling noise around the periphery of an analog block
- Avoid routing over the analog block
- Controlling noise in the power rails
- Placing analog block far away from the noisy digital block
- Placing metal shielding completely around and over analog block
- Controlling cross talk noise within analog buses
- Limiting the length of the wire can deter the signal buses from attracting noise
- Controlling cross talk noise in the I/O rings
- Use of standards in AMS IP Creation and Integration

VSI Alliance: I/V and Mixed-Signal Standards

- Implementation / Verification
  - Phase 1: Hard VCs
  - Phase 2: Soft VCs
  - Phase 3: Firm VCs
- "Hard is easy, soft is hard"

Aristo Technology

- Mixed-Signal
  - Extend work of other DWGs for AMS VCs
  - Phase 1: Hard AMS VCs
  - Phase 2: System-Level Design w/AMS VCs

Portability

VSIA AMS Extensions

VSI Arch & I/V

Section	Deliverable	Currently Used Formats	VSI Format(s)	Hard	Comments
2.6	**Physical Block Implementation**				
2.6.1*	Block description	GDSII, LEF	GDSII	M	
2.6.2*	Pin list/placement	LEF	VC LEF	M	Required if Hard is netlist based
2.6.3*	Porosity/blockage file	LEF	VC LEF	M	
2.6.4*	Footprint	LEF	VC LEF	M	
2.6.5*	Power/ground	LEF/document	VC LEF	M	
2.6.7*	Physical Netlist	Spice3 netlist format, Verilog-A Emerging: VHDL-AMS	VC Hspice	CM	

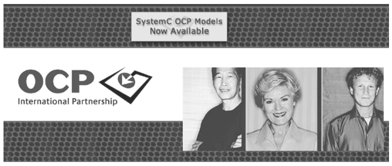
Extend for AMS

Section	Deliverable	Commonly Used Formats	VSI Format(s)	Hard	Comments
2.6.A22	**Interconnect Specifications**				
2.6.A22.1	Special Hookup Guidelines	document	document	M	
2.6.A22.2	Routing Constraints	document	document	M	
2.6.A22.3	Special Pin Requirements	document	document	M	
2.6.A22.4	Additional Power, Ground, and Substrate Interconnect Constraints	document	document	M	

MEDEA+
Systems Innovation on Silicon for the e-economy

The Business of IP Integration

VSI Alliance



VIRTUAL COMPONENT EXCHANGE
VCX™

SPIRIT

Design And Reuse
The Catalyst of Collaborative
SoC Design through SIP Exchange

accellera

MEDEA+
Systems Innovation on Silicon for the e-economy

IP Qualification

- Some industry standards – MORE, OpenMORE, VSIA Quality DWG (Quality IP Metric)
- Self-applied: publicity
- Lack of 3rd party certification
- Many organisations certify incoming IP quality themselves
- 3rd party providers rely more on reputation than facts – their customers must provide the facts:
  - "Measuring IP quality costs time and effort. Many of the large system and semiconductor companies have spent the last seven years creating in-house IP quality procedures, and a number of them claim it costs as much as 3 man-months to verify the quality of one single piece of IP."
    - Larry Cooke, "Why we don't have IP quality yet", EEDesign (online), July 24, 2003
- Conclusion – there is no current substitute for inspecting, QA'ing and certifying incoming 3rd party IP yourself

Conclusion

- IP reuse remains one of the big design challenges
- Design Flows for IP Integration depend on:
  - Reuse style
  - Design style
  - Level of Integration
- Platform-based design is one approach to integration that promotes high levels of reuse
  - Software as well as hardware architectures
- The business and standards aspects of IP Integration have a big impact on the design flows