QUO VADIS, SLD? REASONING ABOUT THE TRENDS AND CHALLENGES OF SYSTEM LEVEL DESIGN

Alberto Sangiovanni-Vincentelli

Presentation by Michael Zimmer

September 21st, 2010

Current Problems

- Exponentially rising complexity in circuits and systems
  - Functionality
  - Verification
  - Time-to-market
  - Productivity
  - Safety and Reliability
- Can traditional design flows (i.e. RTL) continue to meet these demands?
- Embedded Systems more intricate

Possible Solutions

- Raise level of abstraction
  - For chips, this means going above RTL
    - 60% productivity increase? (International Technology Roadmap for Semiconductors)
- New levels of design reuse
- Need new “design science” for embedded system design
  - System Level Design

Challenges

- Heterogeneity and Complexity of the Hardware Platform
  - Exponential complexity growth
    - Transistors on a chip
    - Expanding use of embedded systems
    - More networking
  - Custom hardware implementations costly
    - Design reuse?
      - Looks more like a system (integrating predesigned components)

Challenges

- Embedded Software Complexity
  - Reconfigurable and programmable hardware platforms increase reliance on software
    - 1+ million lines of code in cell phone
    - 100+ million lines of code in automobiles
  - Embedded software requirements stricter
    - Continuously react with environment
    - Safety and reliability
  - How to verify?
    - Tens of lines per day

Challenges

- Integration Complexity
  - Top-down approach?
    - Requires knowledge of entire system for efficient partitioning
  - Integration of predesigned or independently designing components?
    - Need some way to standardize integration of components (often from different suppliers)

Challenges

- Industrial Supply Chain
  - Health and efficiency essential
  - System design needs to be supported across entire development
    - Integration of tools and frameworks from separate domains
    - Information flow between companies
    - Can more efficiently meets demands (safety, cost, etc)
  - Who benefits?

Example: Mobile Communications Design Chain

- Application Developers
  - Sell software directly to customer or come bundled with service provider
- Service Providers
  - Access to network infrastructure
- Device Makers
  - Manufacture cell phones with significant software content and hardware integration
- IP Providers
  - Provide components to design chain
- Outsourcing Companies
  - Manufacturing, design, etc

Example:
Mobile Communications Design Chain

- Boundaries under stress
  - SIM cards
    - Cell phone locked to service provider, but cell phone can still operate with different providers
  - Standards
    - Not locked to one IC provider, IC provider can provide to multiple device makers
- Unified methodology and framework favors balance that maximizes welfare of the system

Example: Automotive Design Chain

- Car Manufactures (OEMs)
  - GM, Ford, Toyota
  - Provide final product
- Tier 1 Suppliers
  - Bosch, Contiteves, Siemens
  - Provide subsystems
- Tier 2 Suppliers
  - Chip manufacturers, IP providers
- Manufacturing Suppliers
  - Not as common for safety and liability reasons

Example: Automotive Design Chain

- Sharing IP and standards could improve time-to-market, development, and maintenance costs
  - AUTOSAR, world-wide consortium, has this goal in mind
- Hard real time software hard to share
  - Can’t just add tasks and not affect behavior
  - New, strong methodology needed that can guarantee functionality and timing
- Would cause restructuring of industry
  - Plug and play environment results in better solutions
  - Tier 1 suppliers?

Needs of Supply Chain

- Design chains should connect seamlessly
- Boundaries between companies are often not clean
  - Misinterpretations result in design errors
- Optimization hard beyond one boundary

Platform Based Design

- Current approaches address either software or hardware but not both
  - Software approaches miss time and concurrency
  - Hardware approaches too specific for software
  - Don’t address all challenges

Desired Methodology

- Hardware and embedded software design as two faces of the same coin
- High levels of abstraction for initial design
- Effective architectural design exploration
- Detailed implementation by synthesis or manual refinement
- Platform
  - Reuse and facilitating adaptation of a common design to various applications

Conventional Use of Platform Concept

- IC Domain
  - Flexible IC where customization is achieved by programming components of the chip
    - FPGA, DSP, MPU, etc
    - Can’t always fully optimize
    - Xilinx Virtex II
      - FPGA with software programming IPs
  - Converging?
    - Semiconductor companies adding FPGA-like blocks
    - FPGA companies adding hard components

Conventional Use of Platform Concept

- PC Domain
  - Standard platforms have enabled quick and efficient development
    - X86 Instruction Set Architecture
    - Fully specified set of busses (USB, PCI, etc)
    - Full specification of I/O devices
  - Allows hardware/software codesign

Conventional Use of Platform Concept

- Systems Domain
  - Platform allow quick development of new applications
  - Sharing subsystems
    - Common mechanical features on automobiles like engines, chassis, powertrains, etc

Platform-Based Design Methodology

- Main Principles
  - Start at highest level of abstraction
  - Hide unnecessary details of implementation
  - Summarize important parameters of implementation in abstract model
  - Limit design space exploration to available components
  - Carry out design as sequence of refinements from initial specification to final implementation using platforms at various levels of abstraction

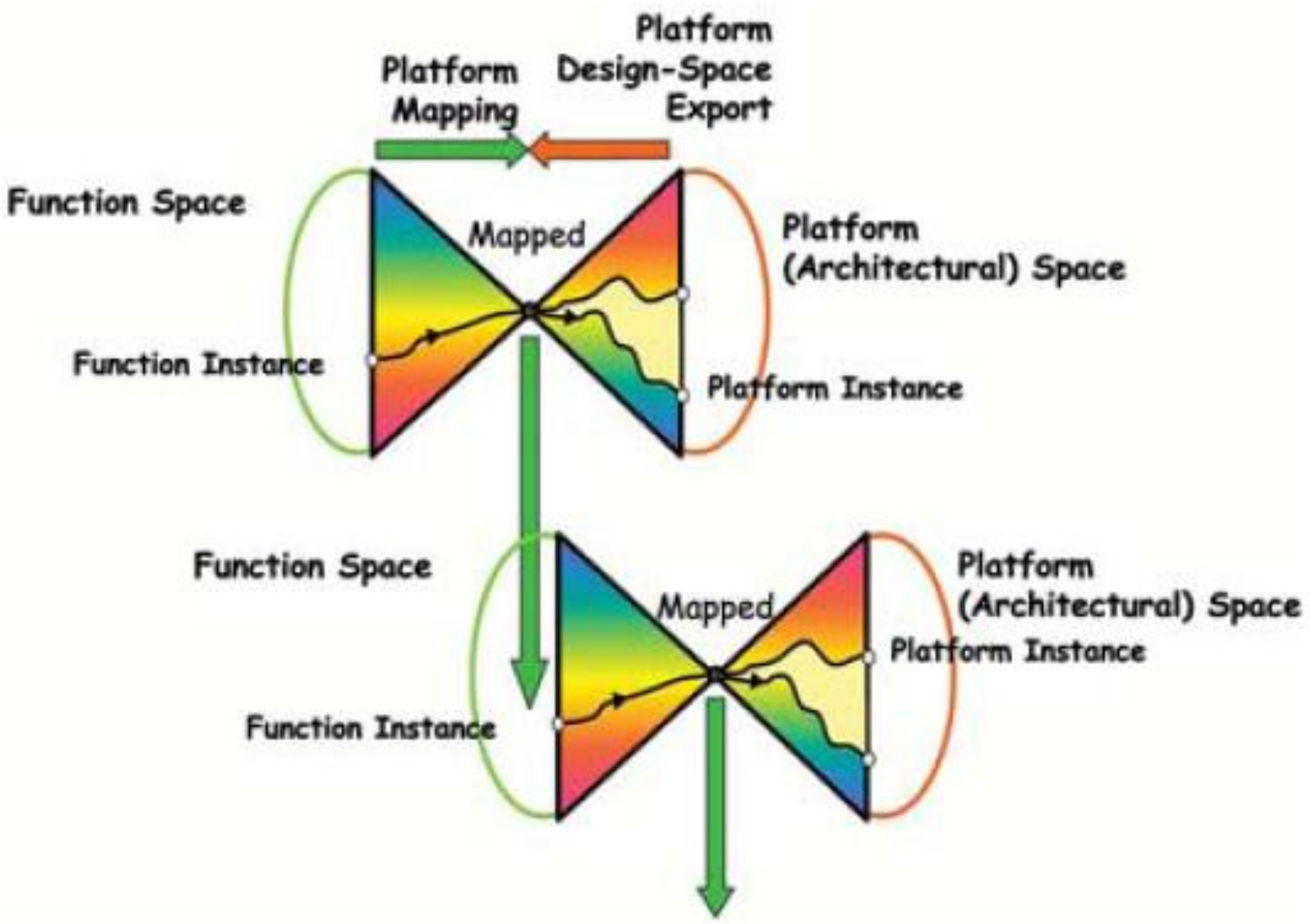
Platform-Based Design Methodology

- Platform
  - Library of components usable at current level of abstraction
    - Computational and communication blocks
    - Characterized by performance and functionality
    - Can have virtual components
- Platform Instance
  - Set of components selected with set parameters
- Mapping functionality to architecture
  - Important to keep separate

Platform-Based Design Process

- Meet-in-the-middle process
  - Top-down: Map functionality into instance of platform and propagate constants
  - Bottom-up: Build a platform by choosing components of the library
- Mapping becomes new functionality

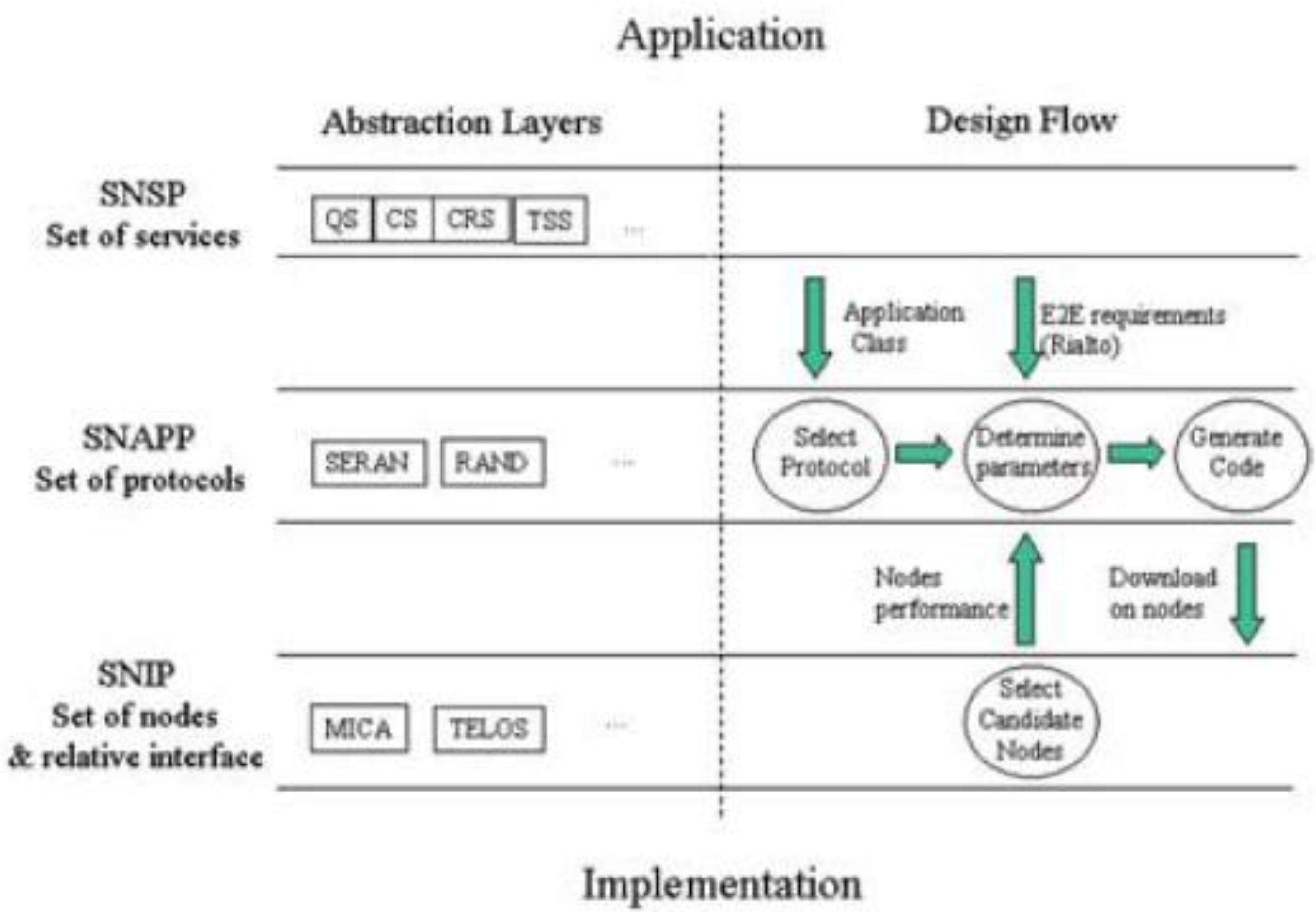
Fractal Nature of Design



Platform-Based Design

- Partitioning of software and hardware is the consequence of decisions at higher levels of abstraction
- Platforms should restrict design space
- Establishing number, location, and components of intermediate platforms is the essence of PBD
- Precisely defined layers
  - Better reuse

Example Application of PBD: Wireless Sensor Network Design



Model-Driven (Software) Development

- Closely resembles Platform-Based Design
- Model-Driven Architecture
  - Platform-Independent Model
  - Platform-Specific Models
  - Interface definitions
- Separation of function and platform

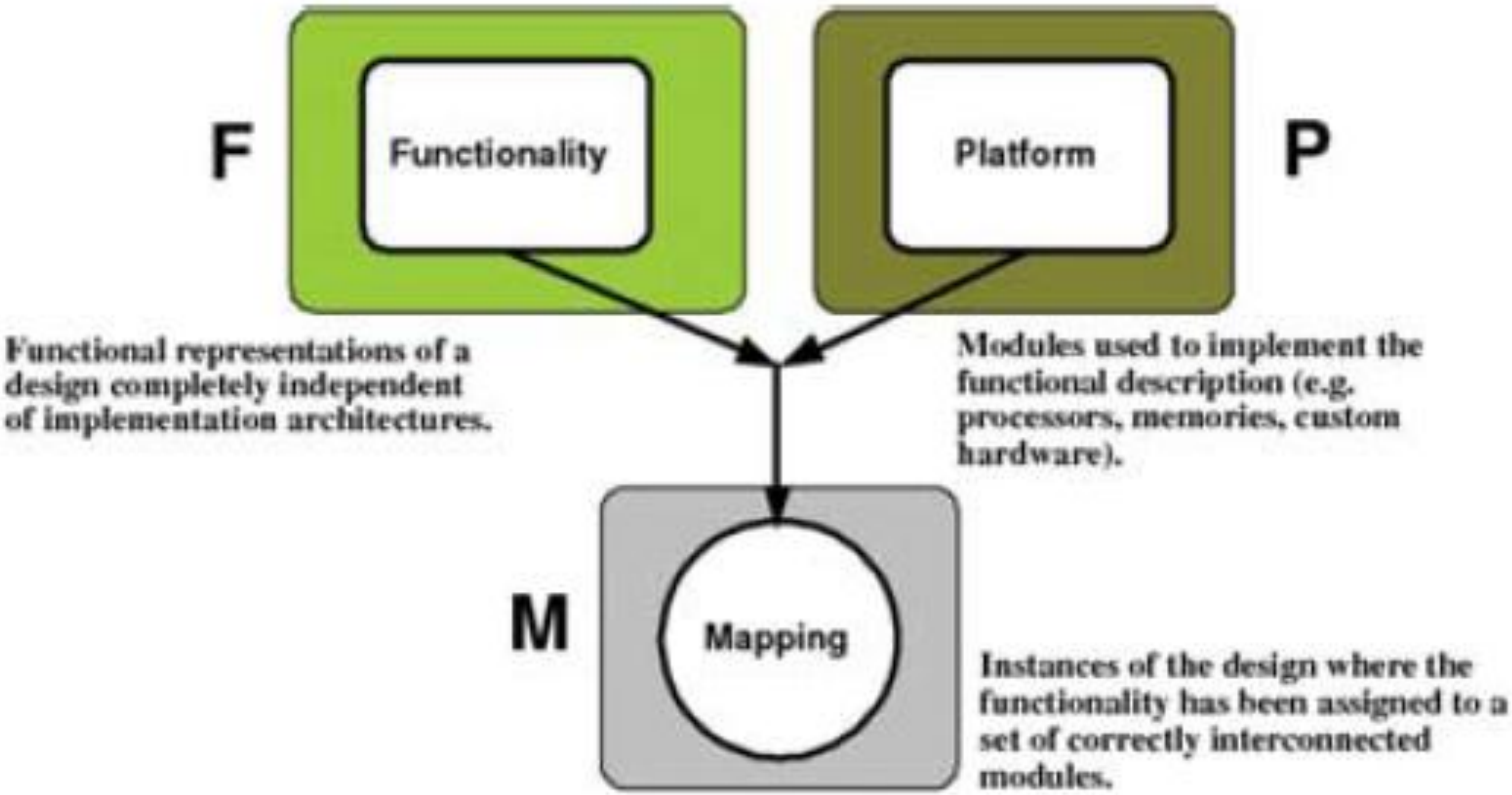
Domain-Specific Languages

- Vanderbilt University group evolved MDD for embedded software design
- Because a single modeling language not suitable for all domains
- But how to define and integrate various models?
  - Interaction must be mathematically well characterized
  - This allows model transformations

Remarks on Platform-Based Design

- Is being adopted
- Well-defined layers of abstraction help supply chain where performance and cost are the contract between companies
- Designers do need to be trained in PBD and have supporting tools

Overview


F
Functionality
Platform
P
Functional representations of a design completely independent of implementation architectures.
Modules used to implement the functional description (e.g. processors, memories, custom hardware).
M
Mapping
Instances of the design where the functionality has been assigned to a set of correctly interconnected modules.

Representing Functionality

- Need to capture at high level of abstraction without assumptions about implementation
- Languages for Hardware Design
  - Attempts to raise abstraction levels
  - SystemC
    - C lacks concurrency and notion of time
    - Capture particular aspects of hardware
    - Used for simulation (not directly synthesizable or verifiable)
  - SystemVerilog
    - Extend Verilog (RTL) to higher abstraction level

Representing Functionality

- Languages for Embedded System Design
  - Want higher productivity and correctness guarantees
  - Synchronous Languages
    - Strong formal semantics to make verification and code generation possible
    - Esterel, Lustre, Signal
    - Safety-critical domain

Representing Functionality

- Models of Computation
  - In traditional approaches, assumptions about architecture embedded in formulation
  - Want maximum flexibility while capturing design
  - Mathematically sound representations
  - Discrete Time
    - Flexible model
  - Finite State Machines
    - Less flexible, but easier to analyze and synthesize

Representing Functionality

- Heterogeneous Models of Computation
  - Mixing models is not trivial
  - Numerous approaches
    - LSV Model, Interface Automata
- Environments for capturing designs
  - Ptolemy II
  - ForSyDe and SML-Sys
  - Behavior-Interaction Priority Framework
  - Signal Processing Worksystem
  - Simulink
  - LabVIEW

Representing Architecture

- Needs to be represented to enable mapping of functionality
- Netlist that establishes how a set of components is connected
- Capabilities should be included
- "Cost" needs to be computed
  - Time, Power, etc.

Representing Architecture

- Software Architecture Description
  - Unified Modeling Language (UML)
    - Stresses successive refinement
    - Graphical nature
    - Too general? (difficult to express common programming constructs)
    - Profiles allow redefining for specific applications
      - SysML, Rational, Rhapsody, Tau
  - Eclipse
    - Integrated Development Environment

Representing Architecture

- Hardware Architecture Description
  - Useful when providing model for performance and property analysis
  - Transaction Level Modeling
    - Levels of abstraction above RTL, can it do better?
  - Assembly Tools
    - CoWare, Synopses, Mentor, and ARM all exploring model creation, integration, simulation, and analysis
  - Communication Based Design
    - Design of interconnect infrastructure and IP interfaces
    - Network-on-Chip
    - Global Interconnect becoming dominant

Representing Architecture

- Hardware Architecture Description (cont)
  - Microprocessor Modeling
    - Embedded systems normally contain software programmable processors
    - Tradeoff between speed and accuracy when modeling
    - Examples
      - Virtual Processor Model
      - C-Source Back Annotation
      - Interpreted Instruction-Set Simulator
      - Compiled Code Instruction-Set Simulator
      - Worst Case Execution Time Estimation

Mapping

- Mapping functional description to hardware instance
- Mismatch of models of computation
  - Asynchronous and synchronous
  - If forced to be the same, restricts design space
- Scheduling
  - For example, concurrent processes onto processor
  - Static vs. dynamic

Mapping

- Correct-by-Construction Mapping – Giotto
  - Solve scheduling problem by forcing models of computation to match
  - Time-triggered architecture
  - Separates platform independent functionality and timing from platform dependent scheduling

Mapping

- Automatic Mapping with Heterogeneous Domains
  - Needs to be a way to automate mapping process
    - Like logic synthesis
  - Need common mathematical language between functionality and platform
  - Tradeoffs in mapping
    - Granularity vs. Optimality

Metropolis Framework

- Unified framework for platform-based design
- Allows for different levels of abstraction and models of computation
- Metropolis Meta-Model
  - Most models of computation and formal languages can be translated into it
  - Can be used to capture and analyze functionality, and describe architectures and mapping

Metropolis Framework

- Functional Model
  - Functional netlist of a network of processes
- Architectural Model
  - Architectural netlist is an interconnection of computational and communication components
- Mapping
  - Mapping netlist instantiates both functional and architectural netlist with synchronization constraints

Metropolis Framework

- Tool Support
  - Allows for back-end tools for analysis
  - Simulator
    - translates to SystemC
  - Verification
  - Synthesis
  - Easy to incorporate external tools

Metropolis Framework

- Related Work
  - None support all the requirements of PBD
  - Polis System
    - Co-Design Finite State Machines
    - Limitations of target architecture and model of computation
  - VCC
  - Artemis Workbench
  - Mescal
  - CoFluent Studio
  - Simulink-Based Flows

Metropolis Design Example: JPEG Encoder Design

- Goal: Map algorithm efficiently onto a heterogeneous architecture
- Modeling and Design Space Exploration
  - Architecture-independent model of JPEG Encoder in Metropolis
  - Processor modeled in Metropolis
- Design Space Exploration and Results
  - Tried different mapping scenarios
  - Simulation close to actual implementation

Conclusions

- Platform-Based Design is a unifying design methodology for system design
- Promising achievements so far, but work still to be done
  - Better understand relationships in heterogeneous environment
  - More efficient algorithms and tools
  - More models must be developed
  - Industry must embrace new paradigms
  - Academia must develop new curricula