Software Methods meet Large-Scale System-on-a-Chip Design: the Arrival of Aspect-Oriented Design

Wolfgang Roesner, IBM Fellow, IBM Systems, HW Functional Verification  June 2015
Agenda

1. Context
   - Chip design principles
   - SoC example

2. SoC Logic Design
   - Constraints violating modularity

3. Aspect-Oriented
   - HW Design Strategies
Design Flow

Architecture Specification → Micro Architecture Spec → Logic Implementation (High-Level Lang) → Circuit Implementation

Mathematical Equiv Proof (Formal Verification) (100%) → Design Model

Design Verification Strategies

Goal: Find >98% Bugs pre-silicon
module p40s_icu_par_stg4tol1 (  
  clk_4to1,  
  act,  
  data_in,  
  toggle_4to1,  
  data_out  
);  
  // Interface I/O  
  // --------------  
  // Interface I/O  
  // --------------  
  input         clk_4to1;  
  input         act;  
  input  [0:7] data_in;  
  output        toggle_4to1;  
  output  [0:7] data_out;  
  reg           toggle_4to1_q;  
  reg    [0:7] data_q;  
  always @(posedge clk_4to1)  
  begin  
    toggle_4to1_q <= ~toggle_4to1_q;  
    if (act)  
      data_q <= data_in;  
  end  
  assign toggle_4to1 = toggle_4to1_q;  
  assign data_out    = data_q;  
endmodule

### Implementation

**Hardware Design Languages**

- **Structural**: modules, inputs, outputs, instantiation hierarchy
- **Behavioral**: functions, equations, sequential code
- **RTL - Register-Transfer Level**:
  - the function is specified as FSM-type definition
  - \((\text{outputs} , \text{regs}^{n+1}) = F (\text{inputs} , \text{regs}^n)\)
Design Verification

- Design has become the biggest effort in chip design (30-70% overall)
- Unit testing is the area where most bugs are found

The different engines have different strengths related to the verification tasks

- **Software simulation**
  - Slow, but low penalty for highly intrusive checking of model internals. Total model visibility

- **Hardware acceleration**
  - Faster, but need less intrusive driving/checking to not slow down hardware box

- **Formal Verification**
  - Exhaustive coverage, but higher skill needed to drive, doesn’t scale well with size

- **Hardware Bring-up**
  - Ideal speed, very limited visibility/controllability
Unsurprisingly, verification becomes more efficient with

- Clearly defined module interfaces
- Clearly defined module function

Filling the buffer at the module level…

…is easier than at the core level
**Power8 Processor SoC**

### Cores
- 12 cores (SMT8)
- 8 dispatch, 10 issue, 16 exec pipe
- 2X internal data flows/queues
- Enhanced pre-fetching
- 64K data cache, 32K instruction cache

### Accelerators
- Crypto & memory expansion
- Transactional Memory
- VMM assist
- Data Move / VM Mobility

### Energy Management
- On-chip Power Management Micro-controller
- Integrated Per-core VRM
- Critical Path Monitors

### Technology
- 22nm SOI, eDRAM, 15 ML 650mm2

### Caches
- 512 KB SRAM L2 / core
- 96 MB eDRAM shared L3
- Up to 128 MB eDRAM L4 (off-chip)

### Memory
- Up to 230 GB/s sustained bandwidth

### Bus Interfaces
- Durable open memory attach interface
- Integrated PCIe Gen3
- SMP Interconnect
- CAPI (Coherent Accelerator Processor Interface)
POWER Processor Technology Roadmap

- **POWER5/5+ 130/90 nm**
  - Dual Core
  - Enhanced Scaling
  - SMT
  - Distributed Switch +
  - Core Parallelism +
  - FP Performance +
  - Memory Bandwidth +
  - Virtualization

- **POWER6/6+ 65/65 nm**
  - Dual Core
  - High Frequencies
  - Virtualization +
  - Memory Subsystem +
  - Altivec
  - Instruction Retry
  - Dynamic Energy Mgmt
  - SMT +
  - Protection Keys

- **POWER7/7+ 45/32 nm**
  - Eight Cores
  - On-Chip eDRAM
  - Power-Optimized Cores
  - Memory Subsystem ++
  - SMT++
  - Reliability +
  - VSM & VSX
  - Protection Keys+

- **POWER8**
  - More Cores (up to 12)
  - SMT+++
  - Reliability ++
  - CAPI Support
  - Transactional Memory

- **POWER9**
  - Extreme Analytics Optimization
  - Extreme Big Data Optimization
  - On-chip accelerators

- **Future**
  - 1H 2014 Scale-Out Systems
  - 2H 2014 Scale-Up Systems
Building collaboration and innovation at all levels

Complete member list at www.openpowerfoundation.org
- Smart, simplified attach for accelerators: GPUs, flash memory, networking & FPGAs
- Connects directly to processor coherency fabric, sharing the same address space
- Accelerator is a peer to the processor core
- Improves performance, reduces latency, and provides more workload per dollar

- Open, architected interface for clients
- Power Service Layer provided by IBM
  - Client is not exposed to coherency rules and correctness concerns
POWER8 CAPI technology connections

- Proprietary hardware to enable coherent acceleration
  - Operating system enablement
    - Ubuntu LE
    - Libcxl function calls
  - Customer application and accelerator
    - Application sets up data and calls the accelerator functional unit (AFU)
    - AFU reads and writes coherent data across the PCIe and communicates with the application
      - PSL cache holds coherent data for quick AFU access
POWER8 CAPI application

Previously, updating these complex algorithms in IO + FPGA was:

- Time Consuming
- Complicated
  - Nine (9) step process that left room for error in coding
- Required advanced technical skills
  - System level programming skills (harder to find)

**Typical I/O Model Flow**

- **Device Driver Call**
- **Copy or Pin Source Data**
- **MMIO Notify Accelerator**
- **Acceleration**
- **Poll / Int Completion**
- **Copy or Unpin Result Data**
- **Ret. From DD Completion**

**New Flow with the Shared Memory Coherent Model (CAPI)**

- **Shared Mem. Notify Accelerator**
- **Acceleration**
- **Shared Memory Completion**

- Reduction in the number of steps required
- No longer need to go through the operating system (increases speed)
- CAPI reduces lines of C code by 40x compared to non-CAPI
POWER8 CAPI vs. I/O Device Driver: Data Prep

Typical I/O Model Flow:
- DD Call
- Copy or Pin Source Data
- MMIO Notify Accelerator
- Acceleration
- Poll / Interrupt Completion
- Copy or Unpin Result Data
- Ret. From DD Completion

Total ~13µs for data prep

Flow with a Coherent Model:
- Shared Mem. Notify Accelerator
- Acceleration
- Shared Memory Completion

Total 0.36µs
CAPI Usage Scenarios

- **FPGA Acceleration**
  - Accelerator behaves as a special-purpose processor running application at user level
  - Application passes pointers to accelerator with no copies, pinned pages, etc
  - Performance, Ease of Programming, Efficiency allow acceleration for workloads that would not have been accelerated without CAPI

- **Network Attach**
  - Elimination of device driver overhead
  - RDMA access to all of system memory without complexity and overhead of registration

- **Storage Attach**
  - Elimination of device driver overhead - ~40x reduction in software path length from device driver and file system for flash access
Agenda

1. Context
   - Chip design principles
   - SoC example

2. SoC Logic Design
   - Constraints violating modularity

3. Aspect-Oriented
   - HW Design Strategies
Chip Design Areas of Concern

- Functionality is not the only constraints for logic design
  - Mainline function & system performance

- Other areas of concern
  - DFT
    - logic to make design testable
  - Design for debug
    - trace/debug buses, error trigger hardware assertions
  - Design for security
    - logic to prevent intrusion
  - Design for reliability
    - for error checking, error repair/recovery (esp. soft error)
POWER Systems Approach

- Has features to handle many solid faults without outages
  - Extensive ECC use
    - To tolerate solid single bit errors
    - Without reduced performance
  - With spare capacity in many cases to repair without replacement
    - Spare bit lanes on busses, cache column repair, etc.
    - To avoid replacing parts

- Alternate Processor Recovery
  - To seamlessly move off workload from a processor with certain solid faults
  - Without taking any sort of outage

- Predictive Deconfiguration
  - Available but rarely needed due to other advanced features

- Mitigating the Extent of outage
  - Third line of defense when other features not effective
Chip Design Areas of Concern (2)

- **Area, Timing, Power**
  - Hard physical constraints (5+ GHz design!), which imposes modularity constraints
    - Forces for physical adjacency can contradict functional adjacency

- **DFT**
  - Intrusive logic structures to pinpoint manufacturing defects

- **Design for debug**
  - Intrusive logic structures to pinpoint functional defects quickly during bring-up (“instrumentation logic”)

- **Design for reliability**
  - Intrusive logic structures to pinpoint and repair functional defects in the field

→ Non-mainline concerns collide with a clean functional design and corrode functional modularity
State of the Art – “Modular pervasive logic”

- Structured approach with repeatable modules
  - Pervasive logic itself follows modular design & verification principles
  - BUT: it is still the very nature of pervasive logic to reach across & deep into functional modules

- State of the art for logic design:
  - RTL module source code contains a merge of all design concerns appropriately meshed together
There are multiple reasons that drive cross-cutting structures into RTL partitions:

- Physical proximity (timing!) has highest priority
- Global routing
- Distribution of centralized controls
  - Clock distribution, control, re-powering
  - Test control and DFT circuitry
- “Pervasive Logic”
  - Reset
  - Power management
  - Thermal management
  - Embedded instrumentation (e.g. debug, recovery)

→ **Hosting** of RTL structures in 3rd party RTL partitions
  - Iterative negotiations between design teams
  - Churn across partitions
  - Link otherwise concurrent development
  - De-focused ownership, loss of logical cohesion
  - → MORE BUGS
1
Context
Chip design principles
SoC example

2
SoC Logic Design
Constraints violating modularity

3
Aspect-Oriented
HW Design Strategies
New Vision: Aspect-Oriented Design

Subject Experts

requirements

Logic Designer

HDL Specification
Function, test, pervasive, power, etc.

Synthesis

Physical Netlist

Model Build

Verification
(all types)

Serialized design where all design aspects are implemented in a single piece of RTL code

Josh Friedrich – DAC keynote 2012

(Design Automation Conference)
Object-oriented software development ran into very similar issues
- The main modular construct is the *class*
- A well designed class hierarchy separates an application across well defined partitions of the *application problem domain* (“business logic”)

- Larger software systems also have to address cross-cutting aspects
  - E.g. persistence, transactions, security, logging

- Traditionally, such cross-cutting aspects are spread across the module (=class) hierarchy
  - Spreading cross-cutting code across modules violates the **modularity principle**
  - → **Tangled** code

Aspect-Oriented Programming supports the separation of aspects at the source-code level…
- → Cohesion of code in one place

… and the **definition** of how the separated aspects are woven in with each other to form a runnable program

* pic source: wikipedia
Candidate aspects that could be worthwhile to separate

- Logical partitioning vs. physical partitioning
- Mainline function vs. “plumbing”
- Mapping of generic state-holding inferred FFs to the appropriate state holding library blocks (e.g. latches)
- Clock control structures
- DFT logic
- Embedded instrumentation logic
- Power
- Back-annotation of any PD changes that verification needs to account for
  - (e.g. physical scan – order optimization)

UPF as a separated power specification is an example where the industry has already implemented “aspect-oriented” separation
Why not total automation with “Silicon Compilation”? 

- The traditional approach to weave-in many aspects has been synthesis (“silicon compilation”)
  - Examples:
    - DFT, FF/latch mapping, clocking
  - Biggest Issues:
    - Pushes all full-design analysis to gate-level (post-synthesis) -- expensive esp. for verification!
    - Takes control away completely from logic designer
      - “all or nothing” automation
      - Severely limits designer options in tight closure cases

- The goal for aspect-oriented RTL design is to define aspects and aspect-weaving at RTL level and stay in RT-level HDL
  - Keeps logic designer in control (where wanted)
  - Keeps verification at mostly RT-level despite aspect-separation
Aspect-Oriented RTL Design Process (I)

Mainline Function Design (RTL)

DFT Recipe File
DFT Designer Recipe File
Pervasive Recipe File
Power Recipe File (UPF)
Aspect-Oriented RTL Design Process (II)

Mainline Function Design (RTL)

Insertion & Weave

Pervasive Recipe File

Power Recipe File (UPF)

Faster Mainline Verif

Implementation Pervasive/DFT Simulation @ RT-Level
Aspect-Oriented Design is all about a useful separation of concerns to allow faster parallel design progress
  – Untangle parallel activities from each other to gain efficiency is design/verification

IBM EDA created a toolkit called “Morph” that supports
  – The specification of separate aspects separately
    • Mostly with RTL language
  – The weaving of aspects together to construct the complete design as RTL
    – Under the direction of “recipe files”

The resulting complete Morph - RTL output is input to logic synthesis / physical implementation

The Morph toolkit generally does RTL-to-RTL transformations
  – The correctness of each transformation is checked with equivalence proofs
  – Verification can be done efficiently at the appropriate aspect level
    • Even “pervasive logic” verification can be performed at RT-Level
Example: Morph Hierarchy

- Increasing need to allow a controlled separation of logical and physical design hierarchies
  - The reasons & constraints to group design elements along hierarchical boundaries is different between the two domains
    - Logic: functional cohesion
    - PD: geographic adjacency

- Today’s strict enforcement of a single logic/PD hierarchy makes logic/verification less efficient
  - Placement of logically adjacent blocks along PD geographies scatters adjacent logic into separated areas of the RTL hierarchy
  - RTL ownership conflicts

- Iterative back-annotation of PD artifacts & geographic information into a single logic/PD hierarchy makes implementation/integration less efficient
  - Refined and changing floor-planning information of physically adjacent blocks into various RTL hierarchies needs to loop through the logic team
  - RTL ownership conflicts

- The design and verification process is much more efficient if each domain can work along their optimal grouping and boundaries
Morph Hierarchy – Keeping Logical vs. Physical Hierarchy in Sync

- Logical Hierarchy: not entangled
  - RTL hierarchy that represents module structure optimized for functional cohesion
  - Standard Verilog / VHDL
  - Verification testbench focus on functional partitions

- Morph Hierarchy
  - Input 1: Logical RTL hierarchy
  - Input 2: “Recipe” to derive physical hierarchy
  - Output: Physical RTL hierarchy
    only delta changes from logical RTL

Recipe to Morph:
Morph Hierarchy – Equiv Checking

- All functional verification is performed on the logical hierarchy
- The physical hierarchy is mechanically derived from the logical hierarchy
  - Synthesis / PD starts here
- Equivalence Checking is required to verify the aspect weaving did not change the design functionally
Morph: Logical Hierarchy vs. Physical Hierarchy

Logical Hierarchy

Floorplan based on logical hierarchy

Physical Hierarchy

Implementation VHDL

Physical Floorplan
Conclusion

- The *aspect-oriented design* paradigm has started to gain traction
  - As an abstraction that captures many problematic symptoms driving us away from “RTL for verification”
  - As an approach to define solutions, which **substantially untangle** RTL and parts of the design process
  - As lever to optimize RTL for verification and thus…
    - ..avoid bugs and churn in RTL for non-functional changes
    - ..optimize verification models and testbenches, masking out non-verification aspects

- IBM is developing a family of aspect-oriented apps
  - Buy-in and benefits of the approach are growing across design/verification/integration team
  - Flexible but controlled de-coupling of aspects provides productivity

- The software world did converge quickly on a key vocabulary and a few base language constructs

- The RTL space needs more time and more players to mature the approach to a few, simple core constructs

- Aspect-oriented design approaches are growing around the industry
  - They might not call it that yet ;-)
  - Shout-out to: *Verific, Defacto Technologies, Invionics*