Part 1: IP-based Design, System-on-Chip and Industrial Trends
Part 2: Design Methodology Progression
Part 3: Models of Computation
Part 4: The POLIS System
Part 5: Interface-based Design

Coping with Complexity

• Design Reuse - IP
• Separation of Concerns
  • logical vs. physical
  • logical vs. timing
  • function vs. communication
• Formalization - precise unambiguous semantics
• Abstraction - eliminate unnecessary details
• Decomposition - divide and conquer
• Incremental Refinement
System Level Design

Top-Down Bottom-Up Design Methodology:
- Abstraction: capture the desired system details
- Decomposition: partitioning the system behavior into simpler behaviors
- Successive Refinements: refine the abstraction level down to the implementation
- Meet in the middle with available IPs

Modeling & Design Re-use

- Re-use comes from abstraction
  - currently a problem because minor changes in pins invalidate composition
  - abstracting communication allows easier mix & match
- Exploration
  - compare IP by switching in different implementations
  - abstract communication enables the switch
- System level IP model
  - build “ideal” algorithmic model (like MPEG)
  - annotate “actual” architectural effects with resource contention and estimated or actual delays
**Problems with Past Design Method**

- Lack of unified hardware-software representation
- Partitions are defined *a priori*
  - Can't verify the entire system
  - Hard to find incompatibilities across HW-SW boundary
    (often found only when prototype is built)
- Lack of well-defined design flow
  - Time-to-market problems
  - Specification revision becomes difficult

**Forms of Design-Level Reuse**

- “The architecture of a component-based system is significantly more demanding than that of traditional monolithic integrated solutions.”
- Sharing:
  - Consistency: Programming and scripting languages
  - Concrete Solution Fragments: Libraries
  - Contracts: Interfaces
  - Interaction Architectures: Patterns
  - Subsystem Architectures: Frameworks
  - Overall Structure: System architectures
IP-Based Embedded System Design: Goals and Requirements

- Separation between function, timing and communication
- Formal specification model
  - facilitates system specification
  - implementation independent
    - eases HW-SW trade-off evaluation and partitioning
- Design/Implementation Verification
  (Formal Verification, simulation, rapid prototyping)
- Automatic Hardware, Software and Interface Synthesis
  - different hardware and software implementation styles
  - designer can concentrate on system level issues

High-Leverage Paradigms

If we face a problem that has become too complex to solve, eliminate the problem!

- Decompose
- Approximate
- Solve by construction
High Leverage Paradigms

◆ Orthogonalization of concerns: view designs along axes that can be dealt with independently
  - Timing and functionality
  - Computation and Communication

◆ Solve by Construction (circuit fabrics)

Communication Design

◆ Determine a protocol so that no matter what the communication topology and the nature of the IP’s the functionality of the overall system is guaranteed (TCP/IP like)

◆ Given the IP set and the interconnections, automatically synthesize protocols

◆ Given the IP set and a set of time-varying interconnections, automatically synthesize adaptive protocol that optimizes “performance” according to the current topology
The Methodology

- Orthogonalize computation and communication
- Plug-and-Play system design
- Chip assembled using IP cores exchanging data by means of a communication protocol
- Interface Logic Blocks (the shells) encapsulate and protect the IP cores (the pearls)
- Assume-Guarantee Reasoning is adopted to formally verify IP cores and communication protocols in separate steps

Work in collaboration with L. Carloni (UCB) K. McMillan, L. Lavagno and A. Saldanha (CBL)

Communication-based Design

Pearls (the IP Processes)
MicroShells (the IP Requirements)
MacroShells (the Protocol Interface)
Communication Channels
System Level Design Science

◆ Design Methodology:

   ▪ Top Down Aspect:
      ♦ Orthogonalization of Concerns:
         ♦ Separate Implementation from Conceptual Aspects
         ♦ Separate computation from communication
      ♦ Formalization: precise unambiguous semantics
      ♦ Abstraction: capture the desired system details (do not overspecify)
      ♦ Decomposition: partitioning the system behavior into simpler behaviors
      ♦ Successive Refinements: refine the abstraction level down to the implementation by filling in details and passing constraints

   ▪ Bottom Up Aspect:
      ♦ IP Re-use (even at the algorithmic and functional level)
      ♦ Components of architecture from pre-existing library

Separate Behavior from Micro-architecture

◆ System Behavior
   ♦ Functional Specification of System.
   ♦ No notion of hardware or software!

◆ Implementation Architecture
   ♦ Hardware and Software
   ♦ Optimized Computer

---

13

14
**IP-Based Design of Implementation**

Which Bus? PI? AMBA? Which One?

Which DSP Processor? C50? Can DSP be done on Micro-controller?

Which Micro-controller? ARM? HC11?

Can I Buy an MPEG Processor? Which One?

Do I need a dedicated Audio Decoder? Can decode be done on Micro-controller?

**Map Between Behavior from Architecture**

Transport Decode Implemented as Software Task Running on Microcontroller

Communication Over Bus

Audio Decoder Behavior Implemented on Dedicated Hardware
System Specifications

Two Basic Questions ...
Question I - IP Authoring

How to design a system block?

- Starting from the system level
- With a consistent test-bench
- Getting from the abstract, un-timed system model to the clocked HW or SW implementation model

Example

- Rake Receiver
  - Which are the optimal algorithms?
  - How does it work fixed point?
  - How is it best implemented?
  - Does the implementation work as specified in the system level
Two Basic Questions …
Question II – IP Integration

How to integrate system blocks?
- Starting from the system level
- With a consistent test-bench
- Getting from the abstract, un-timed system model to the clocked HW or SW implementation model
- Communication between blocks
- Addressing Platform Based design

Example
◆ 3G Cell phone
- Which are the optimal algorithms?
- Do they work together functionally?
- Is the architecture sufficient?
- Does the implementation integration work?

The new approach

◆ Not the typical stepwise top-down refinement: we rest on platforms!

◆ Explicit mapping of applications onto architecture components

◆ The higher the level of abstraction, the faster is the design time
The Essence of the Polis/Felix/VCC Approach

1988:

Traditional System Design

- System Behavior
- System Architecture
- System Implementation
- System Performance

VCC Separation and Mapping

1. System Behavior
2. System Architecture
3. Mapping
4. Implementation of System

Data Sheets on paper

Functional IP Integration

- Algorithm Integration
- SPW, C++, C, SDL, Matlab
- Does the functionally integrated design work?

Question: How does the functional integrated design work?

VCC allows

- to import functional IP from different sources
- to integrate functional IP from SPW, C++, C, SDL, Matlab etc.
- author C++, C or FSM based additional IP
- to assess the algorithmic integration aspects
- to create an unambiguous functional executable specification
Functional IP Integration

- Algorithm Integration: SPW, C++, C, SDL, Matlab
- Visualization
- Functional Integration and Simulation
- Does the functionally integrated design work?
- Executable Functional Specification

Architectural IP Integration

- Question: What does the system architecture look like?
- VCC allows
  - to model architectural IP at the system level
    - CPU, DSP, RTOS, Bus, Memory and dedicated HW/SW
  - to integrate the architectural models defining the platform
  - to present a system architecture to system customers
  - to create an unambiguous architectural specification
### Architectural IP Integration

- **Algorithm Integration**: SPW, C++, C, SDL, Matlab
- **Architecture Performance**: CPU, DSP, Bus, Memory, RTOS, HW, SW
- **Performance**: Does the functionally integrated design work?
- **Executable Functional Specification**: API for architectural components

---

### System Specifications

**Functional View for System Validation**

- **Driver-in-the-loop**
- **Engine RPM**
- **Vehicle Speed**
- **Vehicle Acceleration**
- **Transmission**
- **Accelerator**
- **System**
Closed loop vehicle model

Driver -> Vehicle

Key, Brake, Gas, Transm

force, speed, acceleration, jerk, rpm, fuel consumption...

emissions, external noise, temperature...

Engine & Driveline

spark advance, injection time, throttle angle

Controller

INPUTS:
- K - Key
- G - Gas Pedal
- T - Clutch Pedal & Gear Stick
- B - Brake Pedal
- C - Cruise Control

OUTPUT:
- n - Engine Speed
- F - Generated Force
- V - Vehicle Speed
- D - Comfort

Closed loop vehicle model

Stop
n=0
F=0

Start
n=n.
F=0

Idle
n=argmin(M_{fuel})
F=0

Rpm Tracking
n=m(G)
F=0

Force Tracking
min f(D, M_{fuel})

Fast Negative
max D

Force Transient

Fast Positive

Force Transient

Speed Tracking
V_{C}=V_{s}(.)

Idle & Trasm On
n=0(.)
**Engine Model**

- Air Management
- Mix Management & Injection
- Ignition (Spark) Timing
- Torque Generation
- Exhaust Gas Treatment

**Abstraction**

- The Plant consists of Engine+Drive-Line
- Torque Generation Model abstracted from very complex chemical-mechanical-thermo-dynamical process = FSM!
- Drive Line Dynamics represented as 3rd order linear dynamical system
- Control variables: spark + fuel injection
- Abstraction validated by theoretical devices (formal verification) + measurements on actual cars
**Hybrid Systems in Automotive**

- Driver (Reference) Model
- Engine + Drive-line (Plant) Model

**Cut-off Control: the Problem**

- When accelerator pedal is released, no torque is requested.
- Intuitive solution: reduce injection to zero immediately!
  - minimizes consumption and emissions
  - but, sharp torque variations can cause unpleasant power-train oscillations!
- Control Problem: Power-train oscillation reduction via injection signal control
- Present solution: open loop air/fuel modulation
  - engine speed transient during gear changes
  - power-train state not taken into account