Metropolis: An Environment for System-Level Design

The Metropolis Team
Presenter: Abhijit Davare

CHESS Review - May 10, 2004

Outline

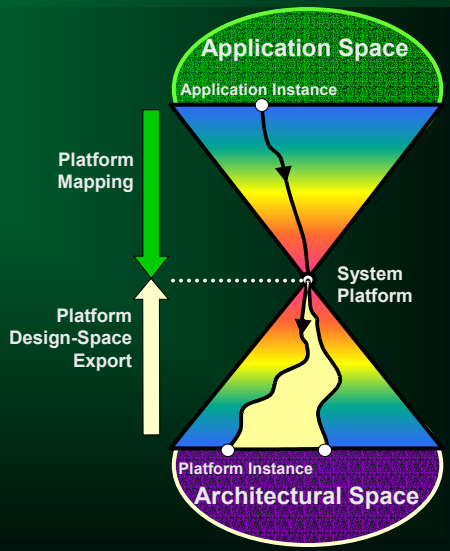
- Motivation
- Platform-Based Design Methodology
- Metropolis Framework
- Picture-in-Picture System
  - Functional Modeling
  - Architectural Modeling
  - Mapping
- Design Space Exploration
- Future Work

Motivation

- Challenges in Embedded Design:
  - Increasing complexity and heterogeneity
  - Time-to-market pressure
  - Verification
- Solution:
  - Design with formal semantics
  - Support different models of computation within a common semantic framework
  - Promote re-use by orthogonalizing concerns
    - Behavior vs. Architecture
    - Capability vs. Cost
    - Computation vs. Communication

Platform-Based Design Methodology

- Layers of abstractions are precisely defined to allow only relevant information to pass through
- Designs built on top of these layers are then isolated from unnecessary subsystem details but provided with enough information to fully explore their design space
- These layers of abstraction are called platforms
- The system can be presented as the combination of a top-level view, a bottom level view, and a set of tools and methods that map between abstraction layers

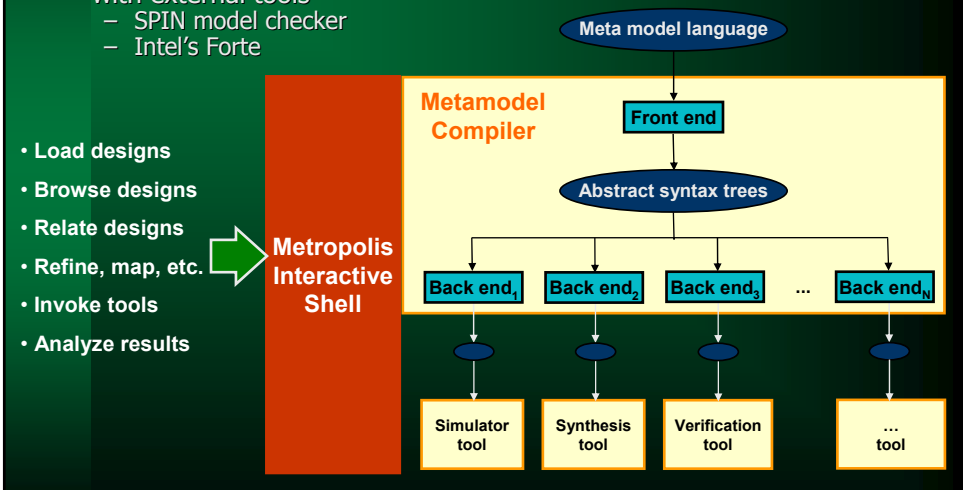


Metropolis Metamodel (MMM) Language

- Captures both architectural and functional aspects
- Syntax based on Java
  - Object-oriented
  - Interfaces
- Supports the importing of legacy code through "blackbox" statements
- Allows imperative and declarative constructs
- Basic constructs:
  1. Process – A thread of control
  2. Media – Connects processes with each other
  3. Quantity Managers – Decide allocation of scarce resources (e.g. time, power, access to media)

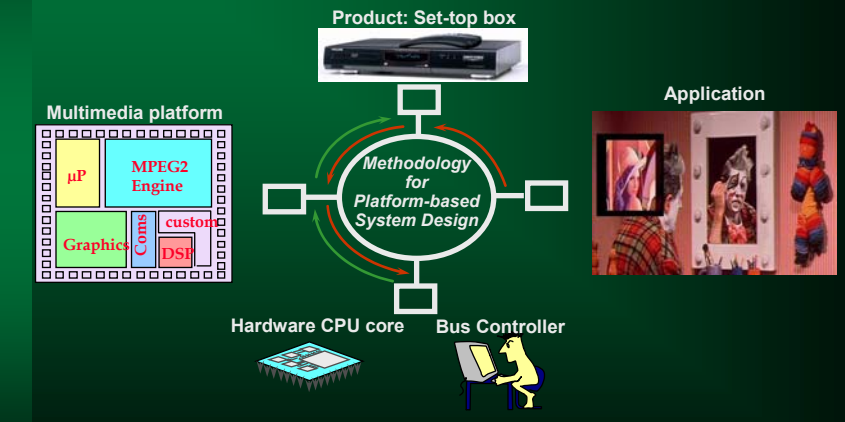
Metropolis Framework

- Modular structure facilitates addition of new backends and integration with external tools
  - SPIN model checker
  - Intel's Forte



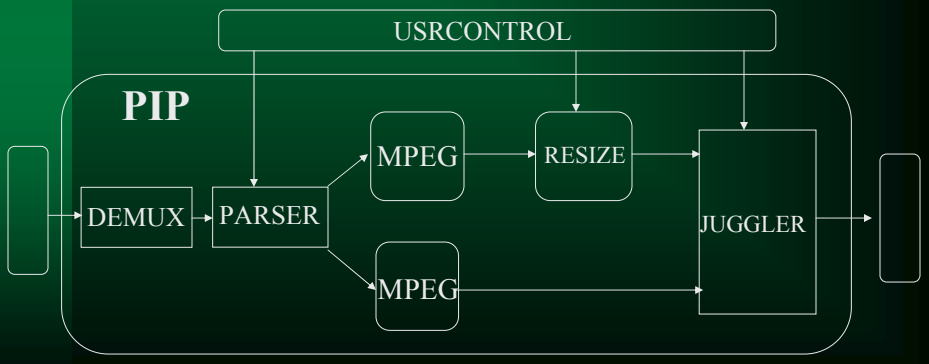
Picture-in-Picture (PiP) System Case Study

- Data dominated application, with some control



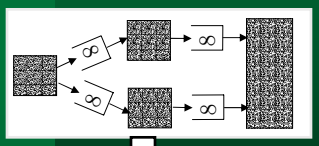
PiP Application

- Functional model contains 60 processes and 200 communication channels
- Some MPEG decoding code imported from C

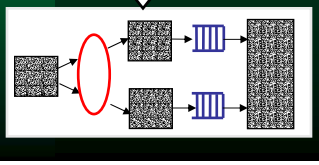


Functional Modeling

- Initial partitioning of functionality into processes chosen by designer
- Two levels of abstraction



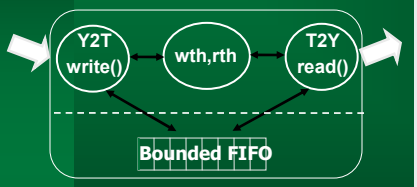
- Network of processes with a sequential program for each process
- Unbounded FIFOs with multi-rate read and write (known as YAPI API)
- Refinement relationship between communication media, properties to be preserved specified declaratively



- Communication refined to bounded FIFOs or shared memories with finer primitives (known as TTL API): allocate/release space, move data, probe space/data

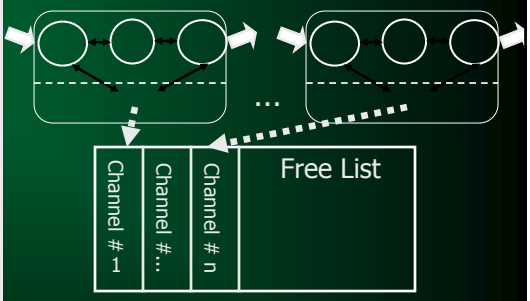
TTL: Bounded Resource Communication

- Independent FIFO:



- Netlist of media refines the original YAPI channel
- Bounded FIFO media is a circular buffer

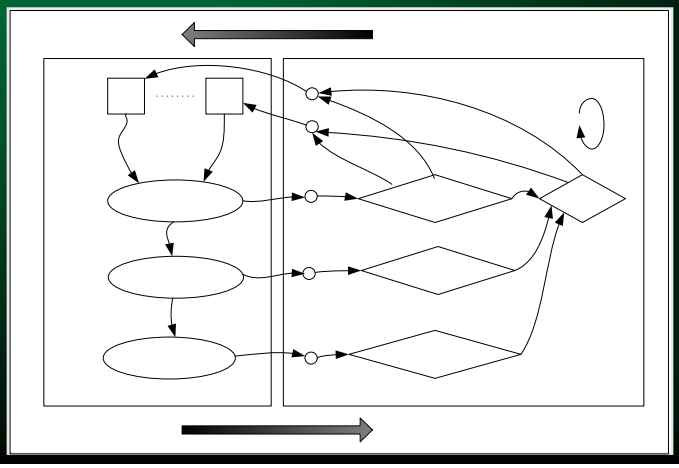
- Shared FIFO:



- Allocation and de-allocation schemes chosen

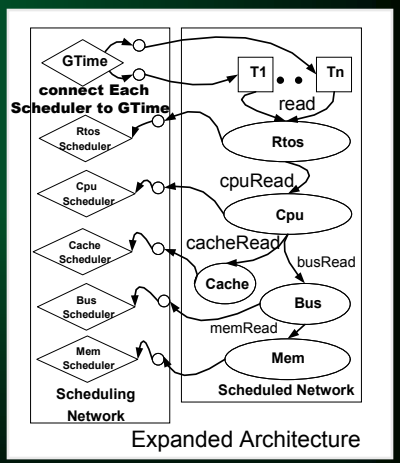
Architectural Modeling

- Phase 1: Scheduled Network makes requests
- Phase 2: Scheduling Network decides which requests to grant with resolve() and annotates events with quantity values



Creating new architectural models

- To model different architectural platforms, we can leverage interfaces and connect existing components in different ways
- To create more detailed architectural models, we can add new services at a finer granularity
- Different evaluation criteria for architectural performance can be supported by adding quantity managers (e.g. power)



Expanded Architecture

Modeling Scheduling Policies

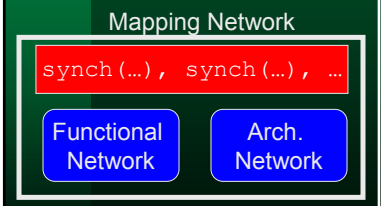
- Changes to scheduling policies are confined to quantity managers
- Example: First-Come-First-Serve (FCFS) and Time-Sliced Scheduling of tasks on a CPU
- FCFS:
  - Every time resolve() is called, the first request in the pending queue is permitted to proceed, all others remain blocked
- Time-Sliced Scheduling:
  - Resolve() internally simulates each time slice, first request that is completely satisfied is permitted to proceed
  - If requested times are larger than a time slice, intermediate states not taken into account

Performance Estimation

- Each request asks for a certain amount of a quantity
  - E.g. a Bus read may take 5 cycles and 10 µJ of Energy
- This amount represents the cost of utilizing a service
- Different physical implementation platforms can be modeled by changing these costs
  - Separation of concerns between behavior and cost
- For PiP case study, numbers taken from a particular configuration of the Xilinx VIrtex II FPGA
  - PowerPC core
  - CoreConnect Bus
  - SelectRAM+ memory
- As refined models are created, numbers are at finer levels of granularity
  - Fewer assumptions

Mapping

- Aim: To associate functional and architectural models explicitly and formally
- Add declarative constraints that associate events – usually service functions – in both models
- Accomplished with the "*synch*" keyword in MMM
- One of the key features that differentiates Metropolis from related approaches




```
e1 = beg(func_process, bf.release_data);
e2 = beg(arch_task, arch_task.release_data);
synch(e1, e2: n_bytes@e1 == n_bytes@e2,
          addr@e1 == addr@e2);
e3 = end(func_process, bf.release_data);
e4 = end(arch_task, arch_task.release_data);
synch(e3, e4);
```

Mapping (cont'd)

- Different design choices can be concentrated in the mapping network
  - Which version of architecture used? (abstract, refined, # CPUs, #tasks)
  - Which processes are mapped to which tasks? (priority)
  - Which communication scheme is used? (TTL with shared memory or independent FIFOs)
  - Where are the communication channels mapped in memory? (Which memory)

Design Space Exploration for PiP

- Questions to answer:
  - Number of CPUs in implementation platform?
  - Use cache?
  - Share memory for communication channels or keep independent FIFOs?
- Four critical channels chosen from Horizontal Resize block in PiP
- Axes of exploration
  - Number of CPUs: 1, 2, and 4
  - Abstract architecture and expanded architecture
  - Independent FIFOs and Shared memory
- Concentrate on read and write services

Design Space Exploration for PiP (2)

Arch. Configuration	Independent FIFO	Shared FIFO
Abstract with 1 CPU	0.219	0.156
Abstract with 2 CPUs	0.142	0.147
Abstract with 4 CPUs	0.103	0.129
Expanded	0.112	0.185

- Numbers represent normalized memory access times (μs) to process a fixed amount of MPEG data
- Shared FIFO scheme uses half of the total memory of the Independent FIFO scheme
- Increasing the number of CPUs is beneficial for both schemes
  - Fewer context switches since fewer tasks per CPU
- Tradeoff between communication schemes is more complex

Future Work

- Automated design space exploration
  - Automation based on:
    - Structural information
    - Trace-based information
    - Reasoning on formal model
  - Implement using Metropolis shell and customized backends
- Refined architectural models
  - Target Xilinx applications and Virtex II FPGA configurations