Virtual Development Platforms: What and How Much to Model

Bill Bunton
SystemC Evangelist
June 13 2010

LSI

Table of Contents

Virtual Prototype vs. Virtual Development Platform

- **Virtual Prototype (PV & CA** >80%**)**
  - **Internal Use Only**
  - **Predicts Hardware Performance**
  - Models new hardware and predicts performance
  - Used for Application and Software development
  - May serve as source for High-Level Synthesis

- **Virtual Development Platform (PV & CCA** >95%**)**
  - **Delivered to Customers**
  - **Models Hardware Performance**
  - Used for Performance Modeling
  - Used for Application and Software development
  - Probably not suitable for High-Level Synthesis

LSI Communication Processor
Virtual Development Platform Heritage

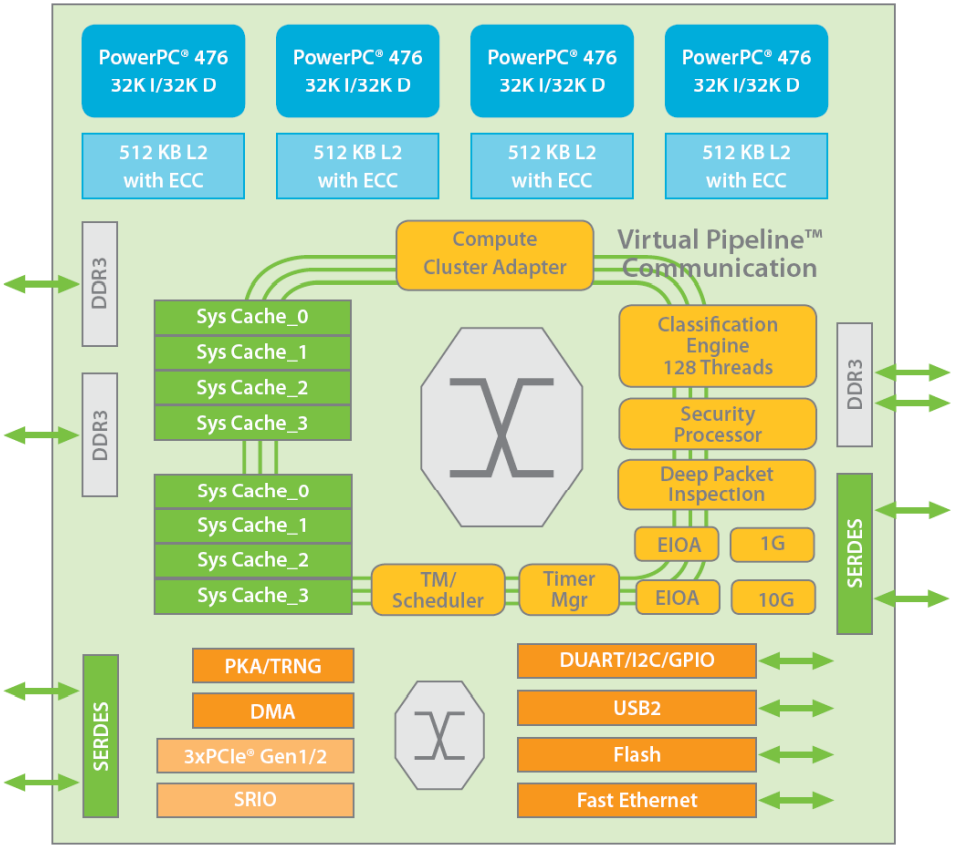
- **Each Generation of communication processor has been accompanied by an Integrated Development Environment (IDE)**:
  - Configuration interface
  - Custom compilation tools
  - Run-Time Environment
  - Hardware Simulator

- **The IDE is standards based**:
  - Eclipse
  - SystemC and TLM-2.0

- **Eclipse encapsulates**:
  - Configuration interface
  - Platform-Specific Software tools compilers debuggers
  - The SystemC Simulator

ACP3400 Multi-Core Communication Processor

- **Quad PowerPC 476 Processors**
  - 32K I cache, 32K D cache, 2MB L2 cache
- **Acceleration Engines**
  - Packet Processing Classification
  - Security Engine (Crypto)
  - Stream Editor
  - Deep Packet Inspection
  - Traffic Manager/Scheduler
- **Engine Communication Rings**
  - Virtual Pipeline
  - Memory Management
  - Timer Services
- **System Memory**
  - Eight System Caches 4MB
  - Dual DDR3 controllers



What and How Much to Model: Program Visible Registers

LSI

What and How Much to Model: Program Visible Registers

- **Complex SOCs may have 1000's of software-addressable registers**

- **A Virtual Development Platform (VDP) doesn't need to model all software-accessible registers**
  - It is typical to leave some registers undocumented

- **Organize and Model registers based on their intended use**:
  - Operationally Dynamic
  - Configuration Dynamic
  - Performance Metric
  - Hardware Error Reporting
  - Hardware Debug
  - Simulation-only Control

Program Visible Register Modeling

- **Operationally Dynamic Registers**
  - Model all registers in this group
  - Fully functional at bit level
  - Full timing accuracy
- **Configuration Dynamic Registers**
  - Model registers as needed to support system software
  - The function of some registers may be implemented by simulator configuration
  - Timing accuracy is unnecessary since the registers have little effect on dynamic system performance
- **Hardware Error Reporting Registers**
  - Could be modeled to support verification of Software Error handling

Program Visible Register Modeling

- **Hardware Debug Registers**
  - The logic behind these registers is probably not implemented there is no need to implement the registers
- **Performance and Trace Logic Control Registers**
  - A strict model of hardware may be sub-optimal
  - Consider simulation unique performance and trace implementations
  - The simulation model may omit performance and trace capability provided by the hardware
- **Simulation-only Control Registers**
  - **Provide a control interface for simulation unique features**
    - Force back-pressure
    - Resource starvation
    - Fault injection
    - Model fidelity control

Register Model Implementation

- **Use the TLM-2.0 socket and base protocol to implement the register-block interface**
  - If the register interconnect is proprietary, consider a custom payload
  - At a minimum, implement the non-blocking and debug interfaces
    - It may be sufficient to use the return path to complete transactions
    - More complex register behaviors may require a full handshake
  - Bus or Ring interconnect topologies can be implemented
- **Decode the Full Register Address Space**
- **Implemented Registers should provide full logical functionality**
- **Unimplemented registers**
  - Read only and unwritten should return a default value (0x0000 or 0xbad0)
  - Unimplemented and previously written should return written data
- **Unspecified registers should**:
  - When error reporting is defined -- Report error status
  - When error reporting is not defined -- Behave as an unimplemented register

What and How Much to Model: Functional Logic Blocks

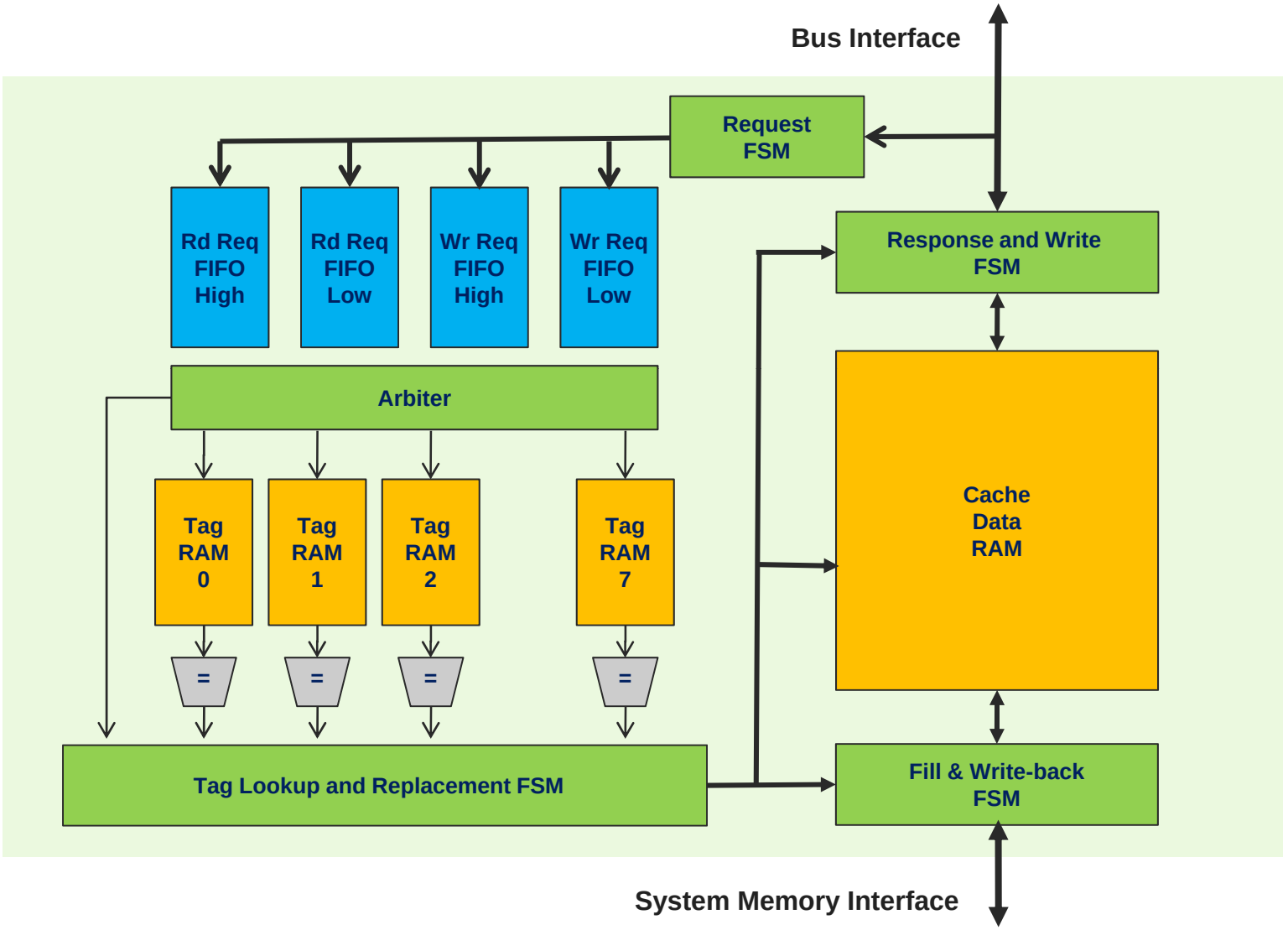
Cache Memory Reference Design

LSI

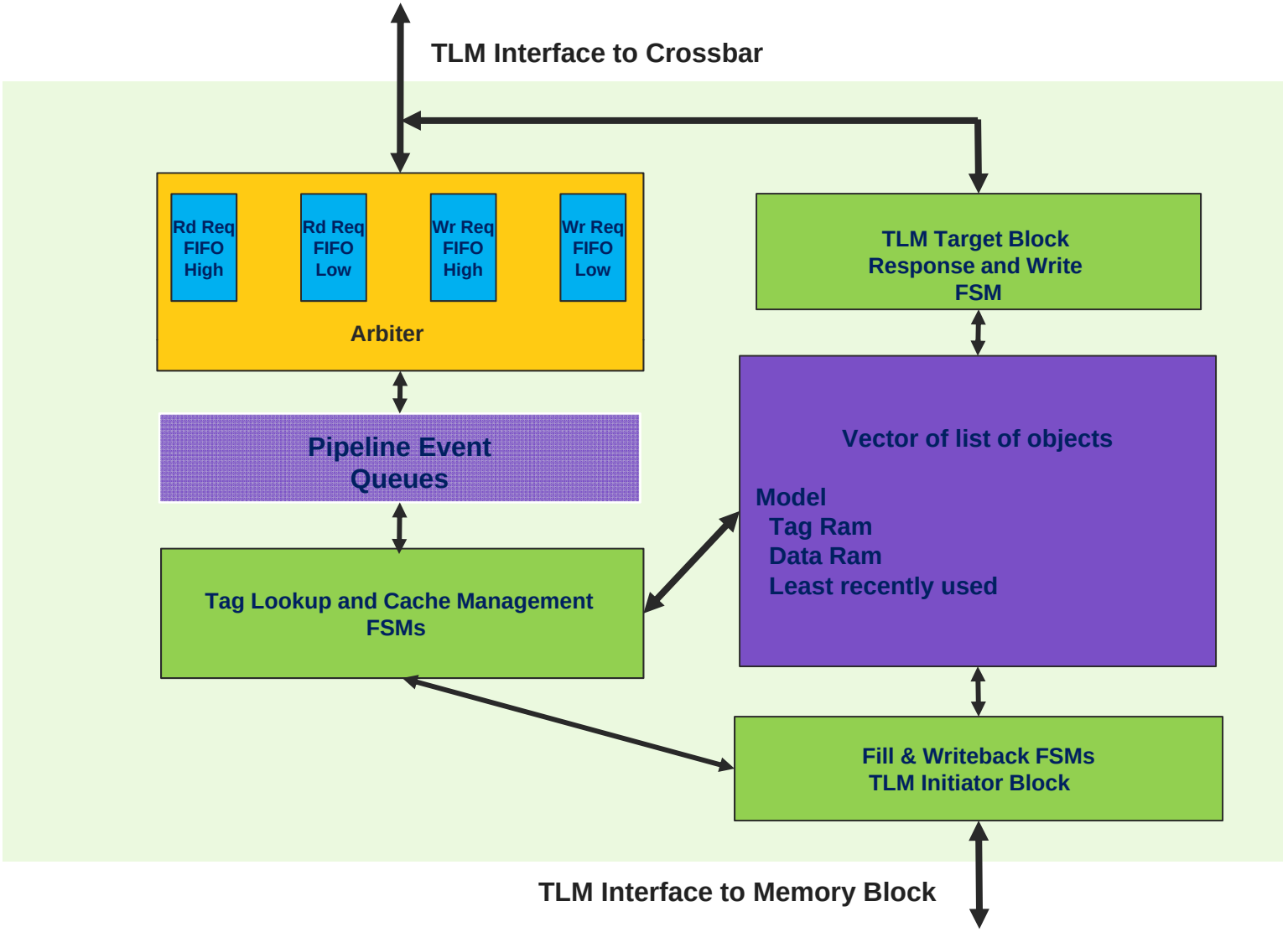
What and How Much to Model: Functional Logic Blocks

- **A Virtual Development Platform (VDP) should focus on:**
  - Logic functionality
  - Performance accuracy
  - Order of execution
- **Model the logical function, not the hardware implementations**
  - Model buses, FIFOs, arbiters and functional blocks
  - Model the state machine function not the hardware implementation
  - Model data storage, not RAM ECC
- **Performance accuracy**
  - Model clock periods not clocks
  - Prefer using delay periods greater than a single clock
- **Order of execution**
  - Model arbiters
  - Model flow control

Cache Memory Hardware Block Diagram



Cache Memory SystemC Model Block Diagram



Cache Memory SystemC Model Implementation

- **The Cache RAM and Tag RAMs are not directly modeled -- just their storage**
- **The request arbiter and input FIFO are modeled by a templated Arbiter class that uses STL queues for FIFOs**
- **Much of the cache logic and storage is modeled using C++ STL**
- `vector< list <cache_state_obj > >`
  - The Vector is indexed by cache index
  - The List is searched and maintained in LRU order
  - The object store cache state
    - Valid
    - Dirty state
    - Cache line data

Cache SystemC Model Implementation

- **Most state machine logic is implemented using sc_method**
  - Fundamentally a large switch statement (switch (FSM_state))
  - State changes should be event-driven or a multiple of the clock period
  - It may be convenient to use zero time states (SC_ZERO_TIME)

- **Pipeline logic uses an Ordered Payload Event Queue**
  - Based on the TLM-2.0 utility PEQ_with_get
  - The load side has a depth parameter and ok_to_load and ok_to_load_event
  - Objects in the PEQ are not reordered

- **Error Reporting Logic**
  - It is not practical to implement error detection and reporting logic in a Virtual Development Platform
  - Register logic can be used to model error reporting -- Out-of-band access to set and clear error status

Cache SystemC Model Implementation

- **Performance Metrics logic**
  - Model implementation should be designed to capture the same data at the same logical location as the system hardware
  - **Typical hardware performance metrics:**
    - Small number of large counters
    - Control provides selection of values or events for counter input
  - **Simulator performance metrics:**
    - Large number of metrics counters
    - Optional periodic metrics samples
    - Optional activity trace files (.csv)

Virtual Development Platform Validation

- **The primary attribute of a Virtual Development Platform is that it simulates real hardware with high fidelity**
- **Functional test and applications should be run on both the VDP simulator and the hardware**
- **When functional differences are detected, the simulation model must be corrected to track the hardware**
- **Performance metrics collected from hardware should be used to back-annotate the simulation model**
- **Back-annotation is the key to simulator fidelity**

Observations

- Quality Architectural Specifications are a must for Virtual Development Platform implementation
- Early Specification of Dynamic Control and Status Registers simplifies both hardware and simulator implementation
- Virtual Development Platform simulators model hardware behavior, not hardware implementation
- Plan simulator metrics for correlation with metrics implemented in hardware
- SystemC simulation models can have fidelity better than 10%
- Back-annotation from hardware is required for high fidelity

Thank You

If you would like to download a copy of the presentation slides in PDF format, click the Attachments link at the top of the presentation viewer.