Early Architecture Exploration leveraging TLM2.0

Challenges & Opportunity

Amit Garg
Product Architect
10th Sep. 2015
Agenda

Industry Design Trends

Architecture Exploration / Optimization – Requirements

TLM2.0 – AT: Solution, Challenges & Limitations

TLM2.0 – AT: Custom Protocol Modeling Framework

Conclusions & Next Steps

Q & A
Industry Trends
Today’s Megatrends

• Globally, there will be 19B networked devices in 2017, up from 12B in 2012

  –By 2017, there will be an average of 5 internet connected devices per person

• Globally, mobile data traffic will increase 13-fold between 2012 & 2017¹

• Every second, nearly a million minutes of video content will cross the network in 2017¹

¹ Source: Cisco VNI, 2011-2016; ²Source: Cisco VNI, 2013-2018
SoC Complexity

• More functions, more software, more resource sharing on chip

• Dynamic systems and performance are unpredictable

• Problems with system architecture design and specification causes significant delays

Source: VDC Research Group, 2012
Shift Left with Prototyping!

*Earlier SW Development, HW/SW Integration & System Validation*

**Standard Project Flow Without Prototyping**

- **Arch Design**
- SoC Hardware Development
- Manufacturing
- Software Development, HW/SW Integration & System Validation
- Spec Freeze
- Tape Out
- Silicon
- Project Finished

**Early Time-to-Market with Prototyping**

- **Arch Design**
- SoC Hardware Development
- Manufacturing
- Software Development, HW/SW Integration & System Validation
- Spec Freeze
- Tape Out
- Silicon
- Project Finished
- Shift Left

© 2015 Synopsys, Inc.
The Impact on Product Development
More Complex, More Software, Faster TTM

Must get the **architecture right**, no amount of downstream tools will compensate for the fundamentally wrong architecture.

Need to start **software development earlier**, in parallel with hardware design.

Must **accelerate HW-SW integration** and **system validation** – minimize waterfall development process.
Architecture Optimization – Challenges
• Optimal design is a multi-disciplined task
  – Engineers from system architecture, software, hardware, design IP, library IP and silicon process must participate to enable the design
  – Problem cannot be solved in any one area – need a complete system solution
System Architecture – Challenges

### Specification and Planning

**Application Use Cases**
- Which ones must drive architecture definition?
- Does SW exist yet?

**System Performance**
- What goals must the system achieve?

**System Power and Cost**
- What constraints must be met?

**IP Requirements**
- What decisions are already made?

### Architecture Simulation

**Multicore HW-SW Partitioning**
- How to explore:
  - # of processors
  - SW mapping to processors
  - System synchronization

**Interconnect and Memory Subsystems**
- How to optimize:
  - Global AXI interconnect
  - Global memory subsystem

**HW-SW Validation**
- How to measure with highest accuracy?
- Links with SW development
- Links with RTL based flows
3 Step Process for the Right Architecture

**Using Platform Architect MCO**

1. **Explore Architecture Alternatives**
   - Sensitivity analysis for performance & power

2. **Optimize Architecture**
   - Root cause analysis; remove bottlenecks

3. **Validate Architecture**
   - High accuracy checks on final architecture

Right Architecture
The Power Gap

- Moore’s Law: SoC complexity doubles every 18 months
- Battery energy density doubles every 10 years
- Power gap creates dark silicon and limits integration and complexity
How does architecture affect power?

**Hardware Architecture Questions**

- Power domain per core or entire CPU?
- Best cache size and organization?
- Use coherent interconnect?
- HW/SW Partitioning?

**Software Architecture and Configuration Questions**

- How to avoid page misses?
- Run fast and stop or DVFS?
- Run SW in parallel or power down idle cores?
- Turn off when idle?
- Timeout for moving to low power states?

**Components**

- Multicore CPU
- DDR Memory
- Cache Coherent Interconnect
- SRAM Memory
- DMA
- SATA
- PCIe
- GMAC
- Accelerators
SoC Performance Analysis

Architecture Prototyping Methodologies in Deployment

- Multicore HW-SW Partitioning
- Interconnect & Memory Subsystem
- HW-SW Performance Validation

Exploration
Large Scale Investigation of Alternatives

Optimization
Iterative Improvement
Based on a Starting Point

Validation
Checking with High Accuracy
SoC Performance Analysis
Architecture Prototyping Methodologies in Deployment

- Multicore HW-SW Partitioning
  - Task-driven Workloads
- Interconnect & Memory Subsystem
  - Trace-driven Workloads
- HW-SW Performance Validation
  - SW-driven Workloads

Task A: VPU
Task B: VPU
Task C
Task D: Task E
Traffic scenario
GFRBM
CSA CPU
RTL CPU
Software
Software
Interconnect & Memory Subsystem
Architecture Exploration – Key Requirements

- **Availability**
  - Define effective macro / micro architecture
  - Effective composition (models and systems)

- **Flexibility**
  - Enable effective what-if analysis for future use cases

- **Stimulus**
  - Effective and meaningful stimulus (else it is Garbage In Garbage Out)

- **Fidelity**
  - Fidelity of the models and accuracy

- **Analytics**
  - Input vectors (Frequency, Voltage.....)
  - Results (bandwidth, deadlines...)

© 2015 Synopsys, Inc.
System Architect’s are confronted with an disparate array of models & methodologies
TLM2.0 – AT : Solution, Challenges & Limitations
TLM2.0 – A Solution in the offing

• TLM2.0 standardizes the model interfaces
  – Addresses a big problem of model to model communication
  – Standard interfaces enable cross team, cross organization inter-operability

• Standard interfaces alleviate
  – Repetitive end user ramp up on modeling
  – Repetitive and expensive modeling
  – Enables reuse and fosters ecosystem

• But ....
  – TLM2.0 AT: Does not complete the specification for AT interfaces
  – Extension mechanisms are defined but multiple options
  – Options are non-inter-operable for AT use cases
TLM2.0 Recap - Generic Payload

Typical set of memory mapped bus attributes

- **command**: enum, \(\text{READ, WRITE, IGNORE}\)
- **address**: uint64, byte address
- **data**: unsigned char*, pointer to storage
- **length**: unsigned int, number of bytes in the data array
- **byte_enable**: unsigned char*, specifies sub-word accesses
- **byte_enable_length**: unsigned int, number of elements in byte_enable
- **streaming_width**: unsigned int, defines a streaming burst
- **response_status**: enum, \(\text{INCOMPLETE, OK, ERROR-code}\)

Extension mechanism

- Array of pointers to user defined payload extensions
- Defines rules for ignorable and mandatory extensions

Memory Management

- Reference counting mechanism
- Mandatory for AT, optional for LT

Helper functions for endianness conversion
TLM2.0 Recap - Interfaces, sockets, phases

- Sockets provide Forward and Backward paths and group interfaces

```cpp
template <typename TRANS = tlm_generic_payload,
          typename PHASE = tlm_phase>

class tlm_fw_nonblocking_transport_if : public virtual sc_core::sc_interface {

public:

  virtual tlm_sync_enum nb_transport(T TRANS& trans,
                                   PHASE& phase,
                                   sc_core::sc_time& t ) = 0;

};
```
TLM2.0 Recap - Approximately-Timed (AT)

TLM 2.0 Base Protocol

BEGIN_REQ

END_REQ

BEGIN_RESP

END_RESP

BEGIN_REQ must wait for previous END_REQ, BEGIN_RESP for END_RESP
TLM2.0 Recap - Timing Annotation

Phase

**BEGIN_REQ**
- Simulation time = 100ns
- Call
  - nb_transport(trans, BEGIN_REQ, 0ns)
- Return
  - TLM_UPDATED, END_REQ, 10ns
- wait(10ns)

**END_REQ**
- Simulation time = 110ns
- Call
  - nb_transport(trans, BEGIN_RESP, 0ns)
- Return
  - TLM_COMPLETED, END_RESP, 5ns
- Simulation time = 155ns

**BEGIN_RESP**
- Simulation time = 150ns
- Call
  - nb_transport(trans, BEGIN_REQ, 0ns)
- Return
  - TLM_UPDATED, END_REQ, 10ns
- Simulation time = 110ns
- wait(10ns)

**END_RESP**
- Simulation time = 155ns
- Call
  - nb_transport(trans, BEGIN_RESP, 0ns)
- Return
  - TLM_COMPLETED, END_RESP, 5ns
- Simulation time = 155ns
- wait (5ns)

Callee annotates delay to next transition, caller waits
TLM2.0 AT - Limitations

The ARM AXI Protocol (An example)

Address transfers: start of transaction
- Initiator to setup transaction attributes (except for data content and response)
- New transactions can be started independent of the other ‘transfers’ of a transaction

Write data transfers:
- Initiator to setup write data values
- Write data can happen in parallel to address (not before)

Write response transfer:
- Target to setup response data
- Write response always after last write data transfer

Read data transfers:
- Target to setup read data values and read response
- Read data should follow read address
The ARM AXI Protocol (An example)

- No beat timing for burst accesses.

- Address and data are always sent together for writes.

- Not possible to have concurrent read and write requests.

- Additional Protocol Specific Attributes like burst type, burst size (narrow transactions) etc. are absent.
There are three, and only three, recommended alternatives for the transaction template argument TRANS of the blocking and non-blocking transport interfaces and the template argument TYPES of the combined interfaces:

a) Use the generic payload directly, with ignorable extensions, and obey the rules of the base protocol. Such a model is said to be TLM-2.0 base-protocol-compliant (see 9.1).

b) Define a new protocol traits class containing a typedef for tlm_generic_payload. Such a model is said to be TLM-2.0 custom-protocol-compliant (see 9.1).

c) Define a new protocol traits class and a new transaction type. Such a model may use isolated features of the TLM-2.0 class library but is neither TLM-2.0 base-protocol-compliant nor TLM-2.0 custom-protocol-compliant (see 9.1).

These three alternatives are defined below in order of decreasing interoperability.
But ....

- Lack of examples for recommendation
- Lack of detail in the standard 4 phase AT protocol definition
- Extension mechanism for additional timing and functional fidelity subject to implementation
- Ad-hoc extensions using approach 3 most widely adopted
- Result:
  - No inter-operability
Custom Protocols – Inter-operable TLM2.0 Extension Framework
Objectives

• TLM2.0 based Infrastructure for Protocol Specific
  – Timing points per transaction, preserves interface timing accuracy
  – Protocol specific transaction attributes, for functional and temporal accuracy

• Defined rules and conveniences that
  – Allow Modeling with higher temporal and behavioral accuracy
  – Promote consistency in modeling
  – Enable accuracy/performance trade-offs

• Achieve interoperability with TLM2.0 Generic Payload – Base Protocol

• Achieve inter-operability across protocols and models
  – For functional and timing fidelity
Modeling Framework

Solution is based on TLM2 AT APIs

- Compliant to TLM2 AT GP and base protocol
- Extensions to support additional timing points for increased temporal accuracy

Defined set of modeling rules to ensure consistent modeling style

- Higher simulation performance (leverages temporal decoupling for architecture modeling)
- Consistent behavior (no non-determinism)
- Brings together cycle based and temporal de-coupling best practices

Scalability

- Optional extensions and timing points defined for different protocols
- Additional protocols can be supported with additional protocol engines
Modeling Interfaces

TLM2.0 Base Protocol with Ignorable extensions

- Implication: all models implement TLM2.0 Base Protocol

Benefits:

- One style for architecture exploration and SW development
  - Achieved by LT/AT inter-operability in TLM2.0

- Maximizes model re-use
  - One modeling style for all use cases
  - Offers high flexibility in order to choose trade-offs between speed and accuracy at run-time

- Eliminates the need for simulation interface glue
  - Always possible to fall back to TLM2.0 BP
Modeling Interfaces – (Ignorable) Payload Extensions

- **Extended-phase**
  - Placeholder for standard compliance
  - Transaction timing is embedded in the payload (protocol_state)

- **Protocol state**
  - Payload extension carries the protocol timing point (current stage in transaction life cycle)
  - Each protocol defines its own protocol state extension

- **Protocol specific attributes**
  - Payload extensions
  - Are in principle ignorable (ensures functional inter-operability)
  - Absence of an extension implies a default value
  - Default values should ensure temporal and functional accuracy
Modeling Interfaces

- Modeling is an extension to the standard TLM2.0 approach
  - Key new feature is support for automatic protocol conversion
    - Protocol conversion done (implicitly) on request for each individual attribute and phase
    - Timing conversion is captured in formal protocol specification
Protocol Conversion - How

• Socket identifies intended protocol (TLM2.0 Interface extension)
  – Additional API to select protocol; e.g. `port.set_protocol("AXI")`
  – Additional API to select clock interface; e.g. `port.set_clock(p_clock)`

• At simulation start initiators and target sockets are queried for their protocol
  – For incompatible protocols fall back to TLM2 Base Protocol
  – Formal protocol definition enables creation/generation of protocol pair converters

• A protocol conversion to TLM2 Base Protocol is always available
  – Part of protocol definition semantics

• Design rule checks static at design authoring
  – Additional support for warning and error messages at runtime for incompatible interfaces
Custom Protocol Definition (1/3)

ARM AXI – An Example

• Each logically grouped set of wires is defined as a channel. For example:
  
  DECLARE_PROTO_CHNL(AXI_WR_ADD_CHNL, AWVALID, AWREADY)

• Each channel identifies the start (busy) event and the end (free) event
  – AWVALID : Is the start / busy event
  – AWREADY: Is the end / free event
    – Free event can be specified in terms of clock cycles

• A protocol is defined by group of all the interface signals (channels) between two communication points

  DECLARE_PROTO_CHNL(AXI_WR_ADD_CHNL, AWVALID, AWREADY)
  DECLARE_PROTO_CHNL(AXI_WR_DATA_CHNL, WVALID, WREADY)
  DECLARE_PROTO_CHNL(AXI_WR_RESP_CHNL, BVALID, BREADY)
  DECLARE_PROTO_CHNL(AXI_RD_ADD_CHNL, ARVALID, ARREADY)
  DECLARE_PROTO_CHNL(AXI_RD_DATA_CHNL, RVALID, RREADY)
Custom Protocol Definition (2/3)

ARM AXI – An Example: Mapping to TLM2.0 GP BP

• A formal mapping to TLM2.0 GP BP is defined as follows

```plaintext
DECL_PROTO_CHNL_MAPPING(AXI_WR_ADD_CHNL, GP_REQ_CHNL)
DECL_PROTO_CHNL_MAPPING(AXI_WR_DATA_CHNL, GP_REQ_CHNL)
DECL_PROTO_CHNL_MAPPING(AXI_WR_RESP_CHNL, GP_RESP_CHNL)
DECL_PROTO_CHNL_MAPPING(AXI_RD_ADD_CHNL, GP_REQ_CHNL)
DECL_PROTO_CHNL_MAPPING(AXI_RD_DATA_CHNL, GP_RESP_CHNL)
```
Custom Protocol Definition (3/3)

ARM AXI – An Example

• Additional functional attributes are defined as (ignorable) payload extensions

• Examples of AXI payload extensions are as follows

```c
//Additional responses on top of TLM2 responses
DECLARE_RESP_EXTENSION(axi_response_extension, axi_rsp_enum, axiOK);

DECLARE_EXTENSION(axi_burst_type_extension, axi_burst_type_enum, axiINCR);
DECLARE_EXTENSION(axi_lock_type_extension, axi_lock_type_enum, axiNORMAL);
DECLARE_EXTENSION(axi_access_mode_extension, axi_access_mode_enum, axiNORM);
```
Heterogeneous System

ARM AXI – An Example

- ARM AXI Blocks
- Generic Re-configurable Memory Controller
- TLM2.0 GP BP Models
Heterogeneous System

ARM AXI – An Example

A Sample Trace of the System

TLM2.0 GP Transactions

CAMERA

Request phase (2 fibers)
Response phase

CPU_D

Overlapping Rd/Wr requests

AXI Transactions
Heterogeneous System
ARM AXI – Analysis & Optimization

AXI Read Data Channel Throughput

TLM2.0 GP for WriteTransaction

Detailed Per protocol per block analysis and statistics
TLM2.0 AT (Custom Protocols) – LT Interoperability

• Modeling framework enables
  – Interoperability between Loosely timed models and architecture models
  – Architecture analysis in context of real S/W workloads

• The proposed modeling is an enabler for use cases of
  – Complex S/W sequence is needed to initialize an architecture model / RTL
  – Partially timed processor cores to co-simulate in a detailed architecture simulation

• Enables system level study and optimization of S/W and H/W
Conclusions & Next Steps

- Standardization for custom protocol modeling is needed for TLM2.0 AT adoption

- Interoperability promise can be filled only with LT like unification of AT semantics

- Presented work is a step in the direction of unification of AT modeling style

- Proposed work retains all the flexibility and extensibility with which TLM2.0 as a standard was created
Q & A
Thank You