
esa
Tools for fast simulation of very complex System-on-Chip
Sven Alexander Horsinka
TU Braunschweig
ESA UNCLASSIFIED - For Official Use
European Space Agency

Research Motivation

- Simulation of hardware architectures is a crucial tool in hardware design, validation and increasingly embedded software development
- Customized architectures prohibit the use of general simulators more and more

➢ **Virtual prototype platforms** address this issue by enabling fine grained customization

➢Executable prototypes are constructed from IP libraries

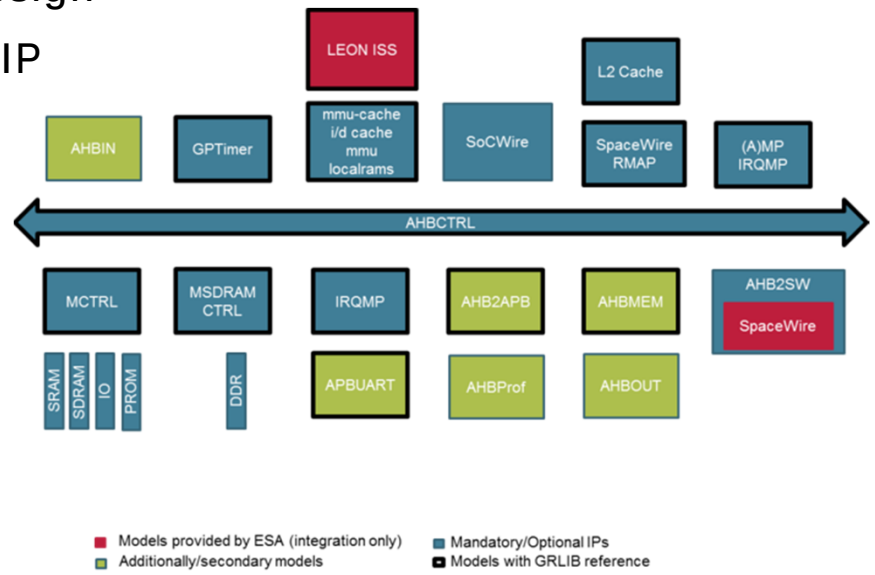
➢Standardization as IEEE1666 (SystemC) lead to high interoperability and design re-use

➢ ESA's **SoCRocket** is a virtual platform applying standardized coding styles and techniques for embedded SoC design exploration

Research Motivation

SoCRocket Virtual Platform:

- Targeted at design space exploration
- Enables flexible hardware software co-design
- TLM2.0 library of Cobham Gaisler GRLib IP
- IP developed by TU Braunschweig and Terma GmbH
- Ongoing research platform at TU Braunschweig



http://www.esa.int/spaceinimages/Images/2012/12/SoCRocket_SystemC_IP-Cores

Research Motivation

This research activity utilized and extended the SoCRocket virtual platform towards:

- Exploration of **complex many-core SoC**
  - Allowing the integration of previously discrete systems into a single design
  - **Mixed criticality** and **virtualization** features guarantee suitable separation and seamless software porting
- Research into a **parallel simulation model** for SoCRocket
  - Single threaded simulation delivers insufficient performance for high core counts
  - A **multi-process simulation model** on top of SoCRocket was developed and evaluated

Outline

- **Intro Transaction-level modelling**
  - SystemC/TLM2.0 and coding styles
  - Translating coding styles to packed based communication
- Mixed-criticality many-core architecture
- SoCRocket Many-core evaluation
- Distributed simulation
- Summary & Conclusion

Intro Transaction-Level Modelling

IEEE 1666–2005:

- Initial standardization of the SystemC language, defining class libraries for system and hardware design
- TLM 1.0 presented first techniques to defined hardware modeling on the transaction level

IEEE 1666-2011:

- TLM Version 2.0 presents more refined approach to model hardware systems towards different accuracy and simulation performance targets
- Included coding styles steer developers and clarify how to treat timing on the previously weakly defined transaction level.

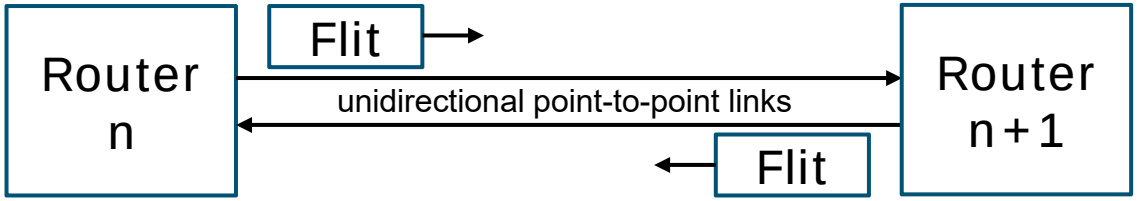
Intro Transaction-Level Modelling

- **Goals of the Loosely-Timed and Approximately-Timed coding styles:**
  - Accuracy vs. simulation performance tradeoffs (in absence of a clock signal)
- **Loosely-Timed coding**
  - Increased simulation performance by reducing synchronization frequency
  - Bus access represented by a single blocking function call
  - Additional optimization: **temporal decoupling** and **direct memory interface**
- **Approximately-Timed coding**
  - Dividing bus accesses into multiple phases
  - Typically implemented as two threads communicating via non-blocking function calls
  - Allows accurate reproduction of pipelined/out-of-order bus protocols

Intro Transaction-Level Modelling

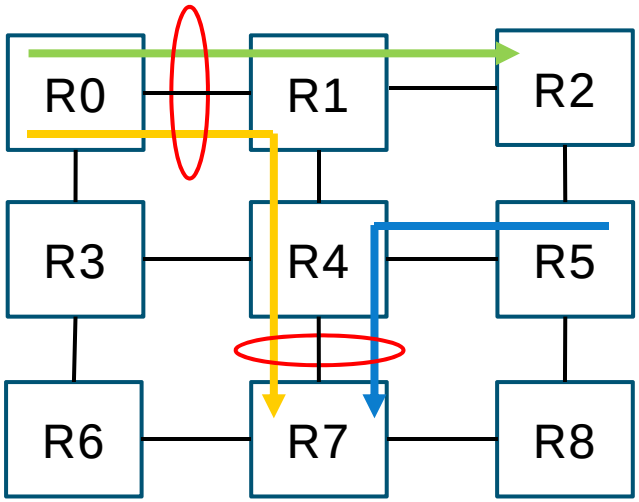
- A **shared bus** defines a singular point of synchronization between competing actors
- Increasing the resolution of access behavior yields higher overall accuracy for bus protocols utilizing outstanding requests, out-of-order responses

- **This does not typically hold true for predominantly packet-based network protocols**
  - At a certain point in the protocol hierarchy, neighboring routers exchange data in **fixed sized chunks** (often called Flow Control Unit, **Flit**)
  - Increasing the synchronization frequency on a point-to-point link does not yield a accuracy benefit



Translation of coding styles for network communication

➢ **Trade-off between accuracy and simulation performance shifts from individual links to a set of independent point to point links**



- All Routers need to stay in sync to reproduce **contention caused delays**
- Correct Flit arbitration is required
- Local hot spots can dramatically increase the communication delay of affected data streams while decreasing the delay for unaffected ones
- Timing error can accumulate very fast if contention is not reproduced accurately

Translation of coding styles for network communication

The deliberation between accurate and fast simulation can be made based on contention awareness

Contention aware:

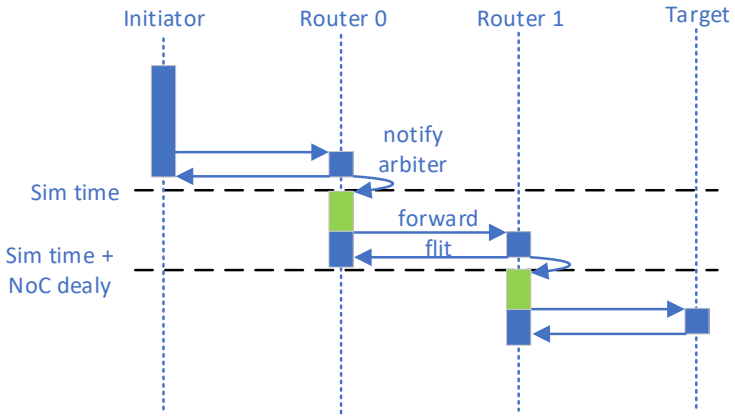
- High accuracy down to Flit arbitration
- Increased modelling and simulation effort:
  - Router model needs to explicitly model flow-control
  - Synchronization interval is defined by Flit-Router traversal time

Contention unaware:

- No synchronization between competing data streams
- Using static or estimated hop delay
- Allows use of **temporal decoupling** and direct **memory interfacing**

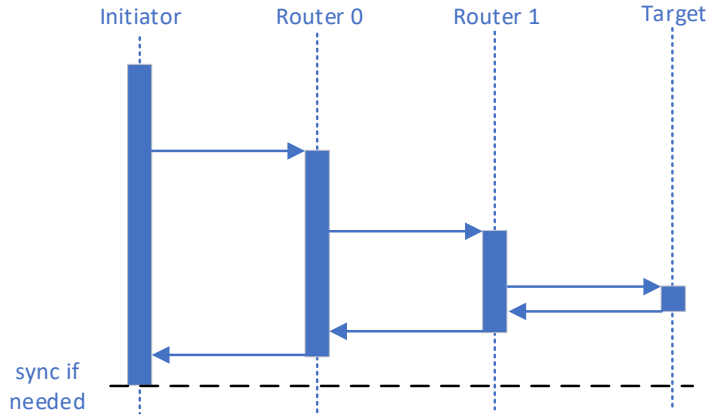
Translation of coding styles for network communication

Contention aware



- Synchronization with the simulator kernel after every hop
- Allows the correct annotation of contention based delays

Contention unaware



- Complete network traversal is performed in a single simulation step
- Synchronization to the global simulation time is performed by the actor if needed

Outline

- Intro Transaction-level modelling
- **Mixed-criticality many-core architecture**
  - Separation
  - Example architecture
  - Reference NoC-router and NoC-interface
- Transaction-level NoC implementation
- SoCRocket Many-core evaluation
- Distributed simulation
- Summary

Explored mixed-criticality architecture

Need for Many-Core architectures

- Feature size shrinks do not allow the previously increases in operating frequency
- Limits of shared bus architectures prohibit higher core counts beyond “multi-core”

Continuing the trend of SoCs, by integrating more and more compute components into a single design new communication architecture is required

➢ **Scalable on-chip networks** are becoming increasingly popular

However, designers need to prevent erroneous interference between tasks performed on a single chip:

➢ **Mixed-criticality support** enable side-by-side execution of versatile tasks

Explored mixed-criticality architecture

- Most mixed criticality systems differentiate applications into:
  - **Highly critical:** require strict performance and throughput guarantees
  - **Low criticality:** interference is not desirable but not operation critical

- How can this be achieved?

➢ **Separation!**

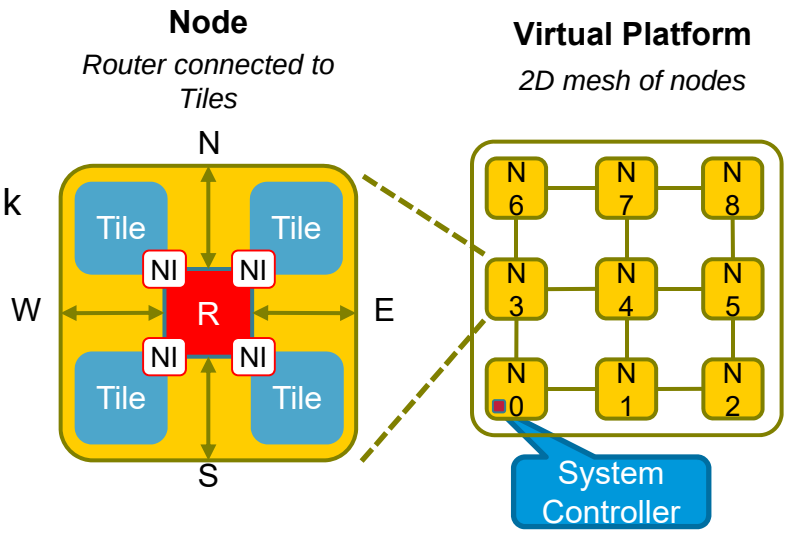
➢ **Spatial separation**: Access to critical memory/resources is only allowed by critical tasks

➢ **Temporal separation**: Low criticality tasks must not impact the throughput/performance of a critical tasks

Explored mixed-criticality architecture

This research is based on a reference architecture developed at TU Braunschweig called **IDAMC**

- Tasks are mapped to individual subsystems (tiles)
- Shared resources are accessed via an on-chip network
- Network resources are used by high and low criticality tasks and implement spatial and temporal separation features



Explored mixed-criticality architecture

Network Interface:

- Interface is local bus master and slave as well as interrupt source and target
- **Virtualization:** local address ranges and interrupt lines can be mapped to remote resources
- **Monitoring:** Erroneous outbound transactions are detected and invalidated before entering the NoC
- **Tile control:** Monitoring events may lead to tile recovery techniques

Network router:

- **2D mesh network** with up to four tile uplinks per router
- Flit based **wormhole switching**
- Credit based flow control
- **QoS** scheme handling data streams of mixed criticality

Outline

- Intro Transaction-level modelling
- Mixed-criticality many-core architecture
- **SoCRocket Many-core evaluation**
  - SoCRocket many-core architecture
  - Simulation performance evaluation
- Distributed simulation
- Summary & Conclusion

SoCRocket many-core architecture

- **Accuracy:**
  - The contention-aware router model was validated against the RTL model
  - Traces of packet injection and ejection times as well as arrival order w[TS14 TS15]aptured (RTL and TLM) for different traffic patterns and router configurations

- **Performance measurements:**
  - Standalone NoC with synthetic traffic compared to RTL
  - Simulation of video processing hardware cores distributed in the NoC (low computation, **high communication**)
  - Running software benchmarks in each tile using a shared memory tile (**high computation**, low communication)

Slide 18

TS14 Welche Bedeutung hat die Mixed-Criticality in diesem Zusammenhang. Meinst du die Contention Awareness eine Vorraussetzung zur Modellierung derartiger Systeme ist. Wenn ja, dann solltest du das schon am Anfang herausstellen (in der Problembeschreibung / Motivation).
Thomas Schuster, 04/12/2017

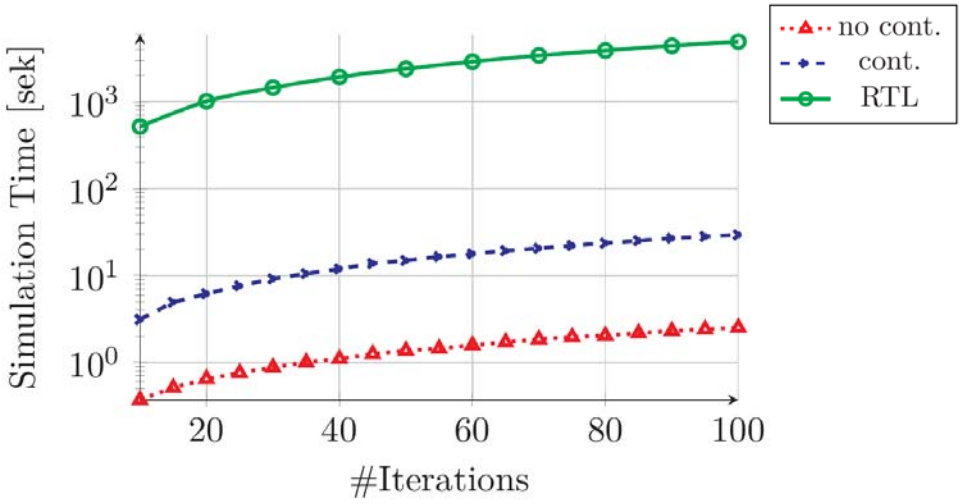
TS15 Es würde sich dann auch Anbieten 'Metriken' zur Messung von Genauigkeit und Verlässlichkeit derartiger Systeme vorzustellen. Was ist wichtig? Latenz, Durchsatz, Granularität der Daten (Resolution)
Thomas Schuster, 04/12/2017

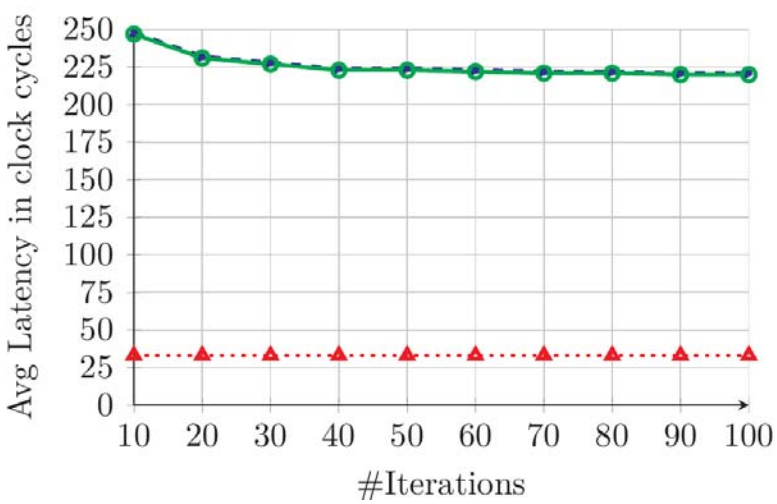
TLM2.0 NoC Router implementation - evaluation

Comparison RTL vs. TLM NoC Router

- Fixed 8x8 network dimensions with 64 traffic generators/receivers (custom cores)
- All-to-all traffic pattern
  - Each generator injects a packets of 5 flits sequentially addressed to all other nodes (injection rate = 1)
  - 10 – 100 iterations cumulating in 33.65 MB of data moved
  - Very high network load to get high delays for the contention aware model
- **Measurements**
  - Accuracy in clock cycles (minimum flit-router traversal takes 4 clk cycles)
  - Performance in seconds execution time on the host-system

TLM2.0 NoC Router implementation - evaluation





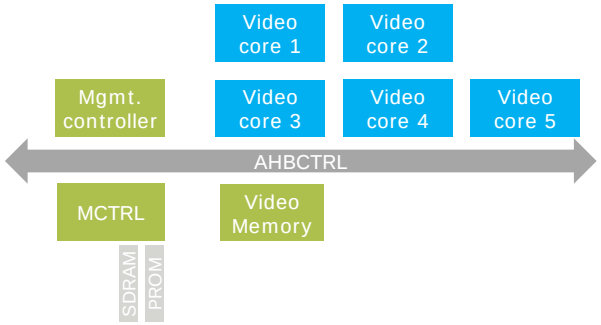
- **>160x** speedup between RTL and the contention-aware TL Model
- **1400x - 2000x** speedup between RTL and the contention-unaware TL Model

- **Static flit latency** in the contention-unaware TL Model
- Individual latency error always **below 3 clock cycles** (contention-aware model)

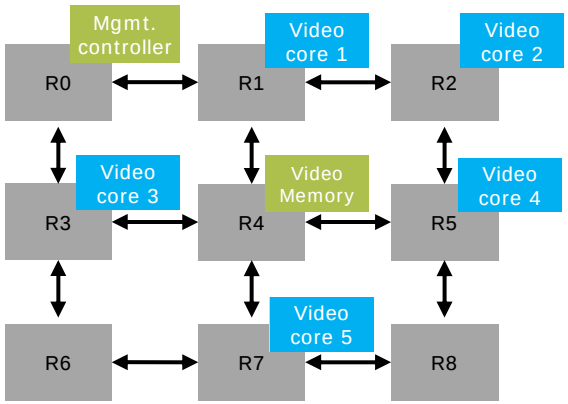
SoCRocket many-core architecture - Evaluation

- Video processing on shared memory without an ISS

Base Virtual Platform

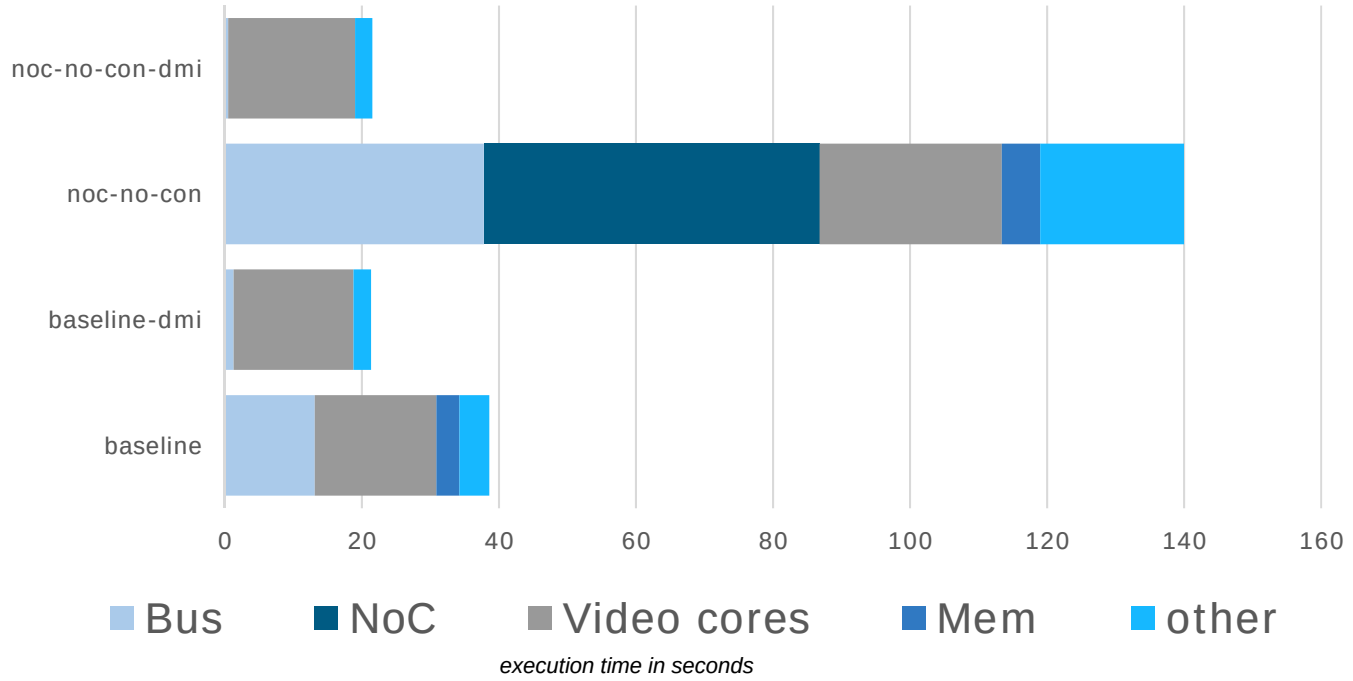


Tiled NoC Virtual Platform



- Collection of video accelerators working on a shared memory
- Video Memory Tile is reached via address translation

SoCRocket many-core architecture - Evaluation



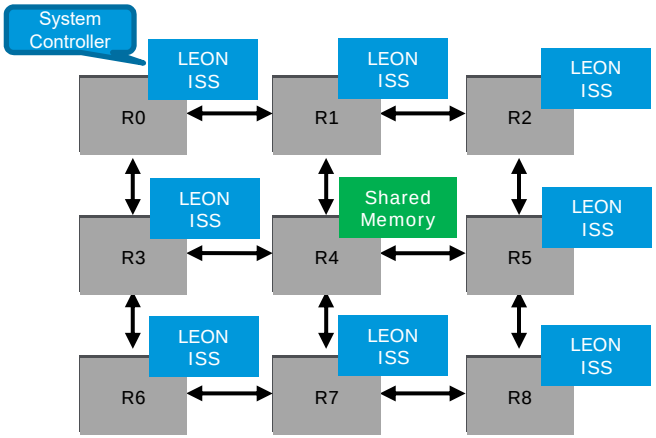
➢ Overall simulation timing error ~23% between coding styles

TS25
TS26
TS27

oCRocket many-core architecture - Evaluation

Software workload

- Each node executes software (FFT) using a shared memory tile



- Seven worker tiles are configured by the system controller
- Each worker reads samples from a shared memory, computes the FFT, and writes back the results to the shared memory

- High software workload with moderate communication overhead

Slide 23

TS25 Hier beginnt ein neues Experiment.
Thomas Schuster, 04/12/2017

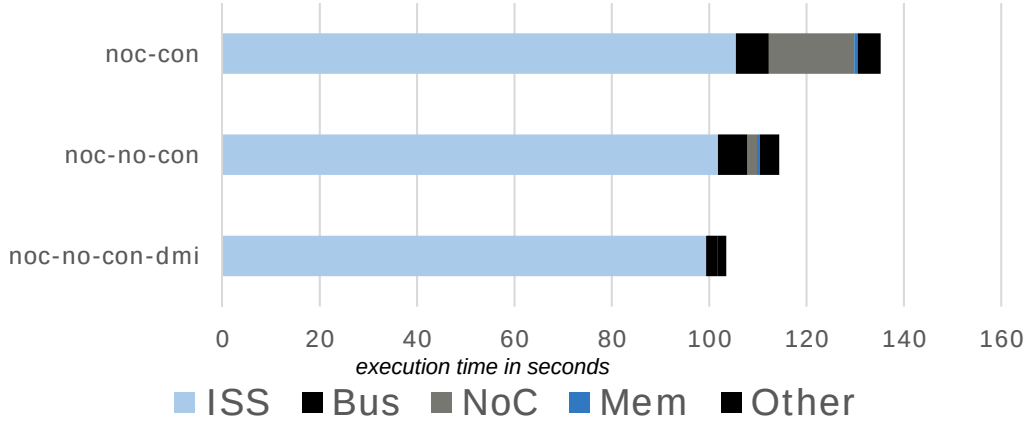
TS26 Du könntest die Headline um eine Unterschrift erweitern.
Performance eval ...
Experiment 2: Real-world workload
Thomas Schuster, 04/12/2017

TS27 für den Abschnitt zuvor:
Experiment 1: Synthetic workload (???)
Thomas Schuster, 04/12/2017

SoCRocket many-core architecture - Evaluation

Real-world workload

- Software FFT on shared memory



- Simulation performance is defined by the ISS
- Only a overall 18% slowdown when modeling network contention accurately
- Overall simulation timing error ~12% between coding styles

Outline

- Intro Transaction-level modelling
- Mixed-criticality many-core architecture
- SoCRocket Many-core evaluation
- **Distributed simulation**
  - Background and research focus
  - Parallel SoCRocket simulation model
  - Performance and accuracy evaluation
- Summary & Conclusion

Distributed simulation

Distributed discrete-event simulation has been a topic of interest for several decades with the classification into:

- **Conservative synchronization**:
  - Avoidance of situations leading to possible causality errors
  - Synchronization frequency defined by designs behavior
- **Optimistic synchronization**:
  - Detection and recovery of causality errors
  - Reduced synchronization frequency at the price of state save and recovery overhead

➢ Both classifications avoid causality error

➢ Not feasible for SystemC/TLM2.0 parallel simulation:

  ➢ Conservative approach requires a too high synchronization frequency

  ➢ SystemC simulation state too complex for an frequent duplication and rollbacks

Distributed simulation

This research targeted to following specialization:

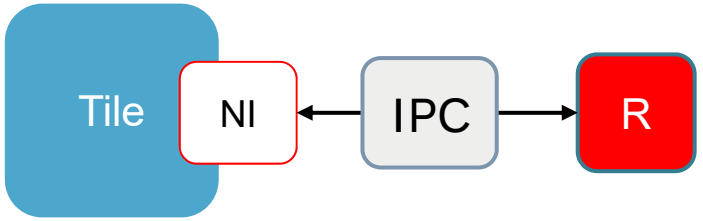
- **Target environment:** Modern workstation CPUs of eight cores or more
  - Parallel exaction on a single system allows low latency communication between simulation peers compared to network connected clusters.

- **Target partitioning:** Simulation model partitioning on low frequency communication links reduces inter process communication overhead
  - Separation based on tile boundaries keeps local bus communication within a single process
  - Only inter tile communication has to be passed between simulation processes
  - Handling packed based communication between tiles observes reduced complexity compared to multi phase bus communication

Distributed simulation

Low IPC overhead on a single machine allows separate handling of communication and synchronization:

- Overall time progression is managed by a central manager with knowledge of the whole simulation
- Communication between simulation peers can be passed directly between simulation peers without explicit synchronization on arrival



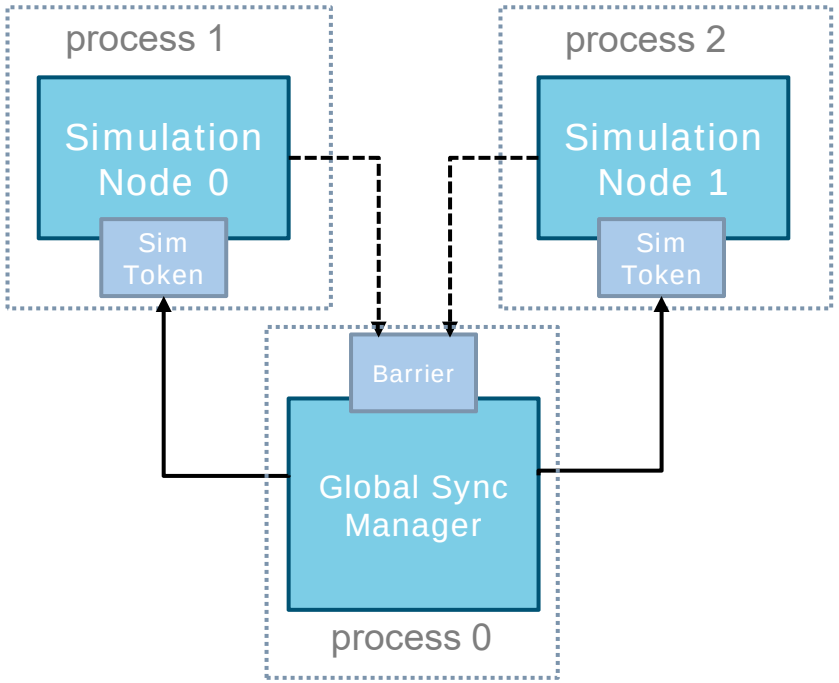
Inter process communication for the synchronization as well as communication was implemented using the Boost IPC library

- Memory mapped massage queues transport serialized data structures between simulation nodes

Distributed simulation

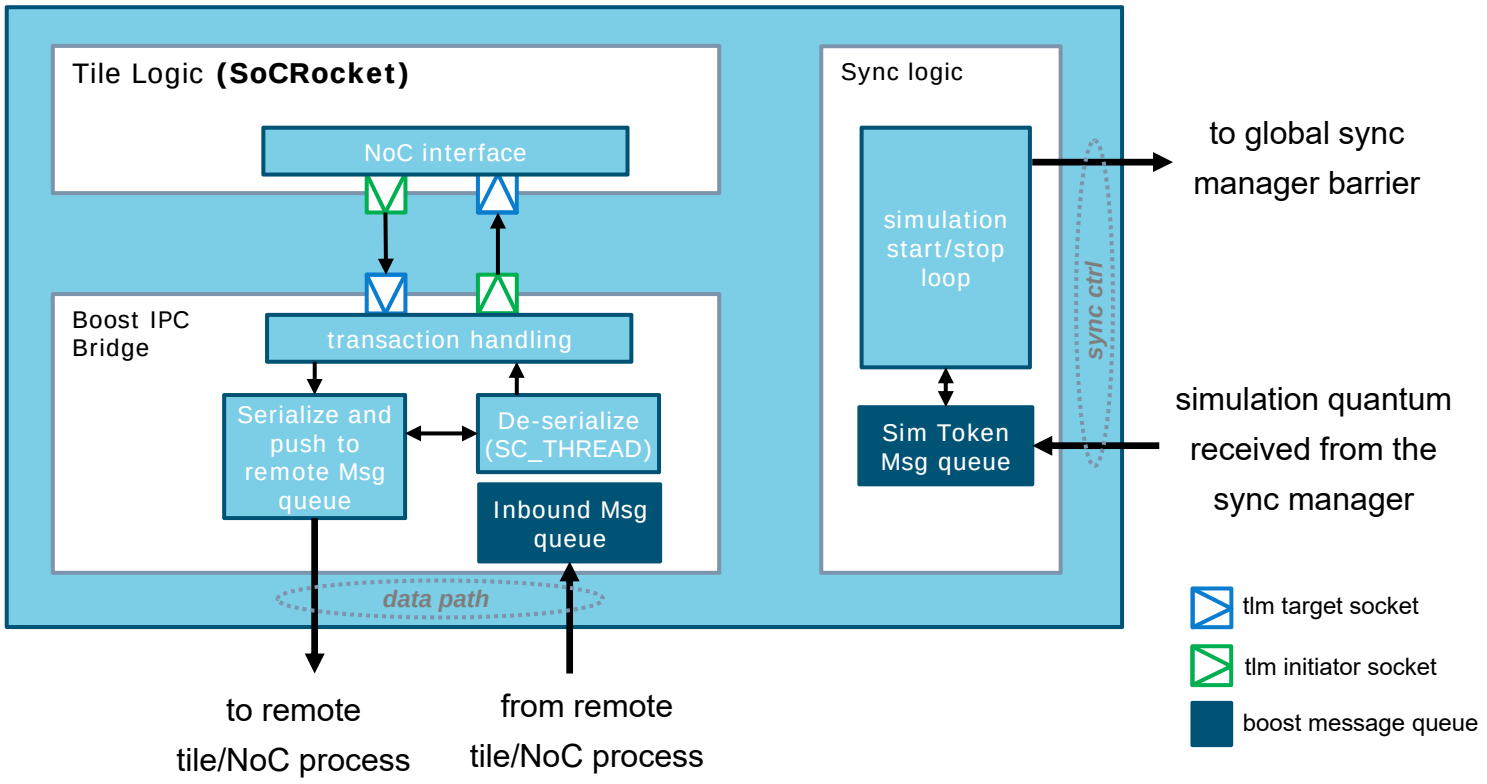
Synchronization manager:

- Nodes request simulation time by entering the sync manager barrier
- Sync manager sends sync tokens to the nodes after all entered the barrier
- Nodes run asynchronously until the granted simulation time is reached



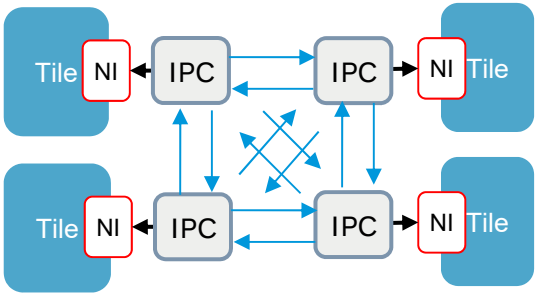
Distributed simulation

Simulation node structure



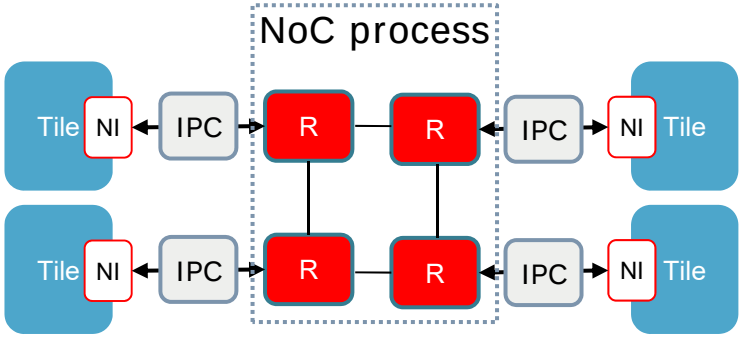
Distributed Simulation

Direct mode



- Communication is directly passed between simulation peers
- Reducing the overhead to serialize/deserialize communication twice

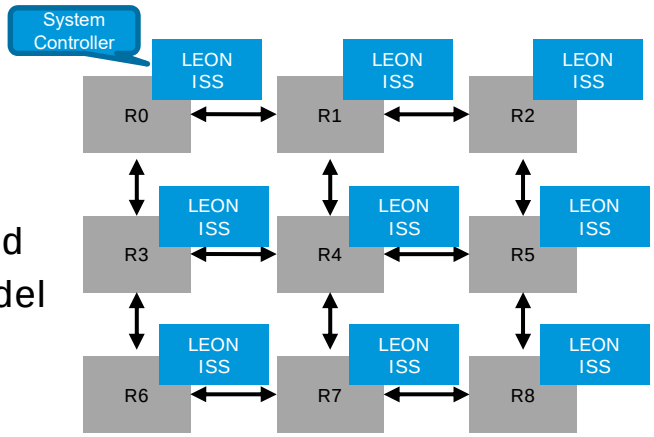
Explicit NoC



- All communication is passed to a separate NoC process
- Transparent to the single process simulation (software)

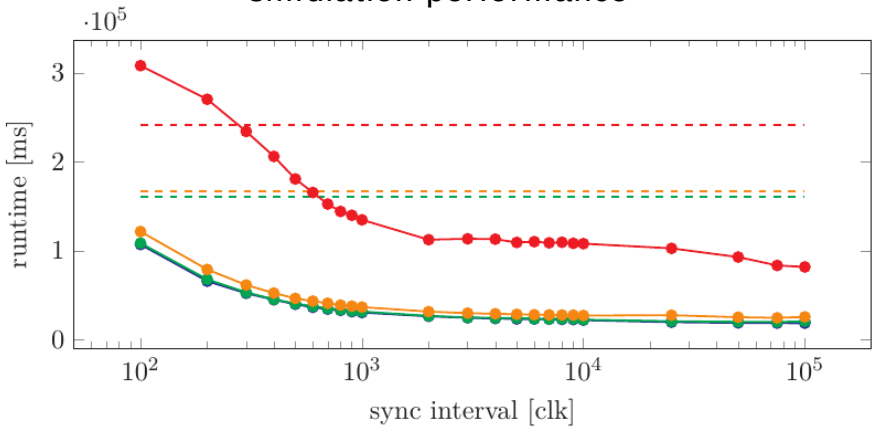
Distributed Simulation – Test Scenario

- 3 dimensional mesh with 8 worker nodes and one system controller
- Each worker performs iterations of remote tile communication and local computation
- Communication is modeled as data access to the system controller memory space
- Different communication to local computation ratios are investigated (between 1000-1 and 1-1000)
- Simulation performance and timing accuracy is compared to the same configuration in a sequential simulation model
- All simulations were run on a 8 core 16 thread CPU with 32 GB of RAM



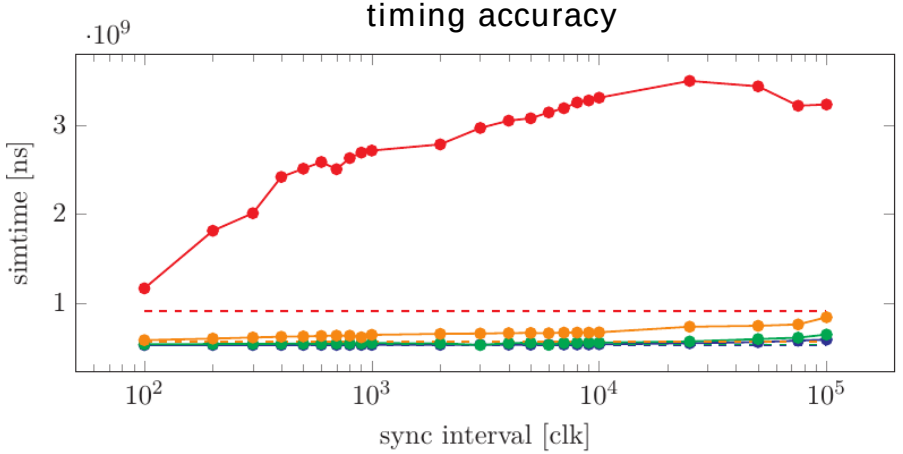
Distributed Simulation – Direct Mode

simulation performance

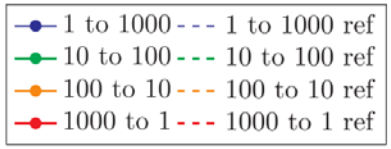


- Desirable speed-up achieved above ~1000 clock cycles
- Low communication scenario reaches up to ~8x
- High communication load only achieves ~3x

timing accuracy



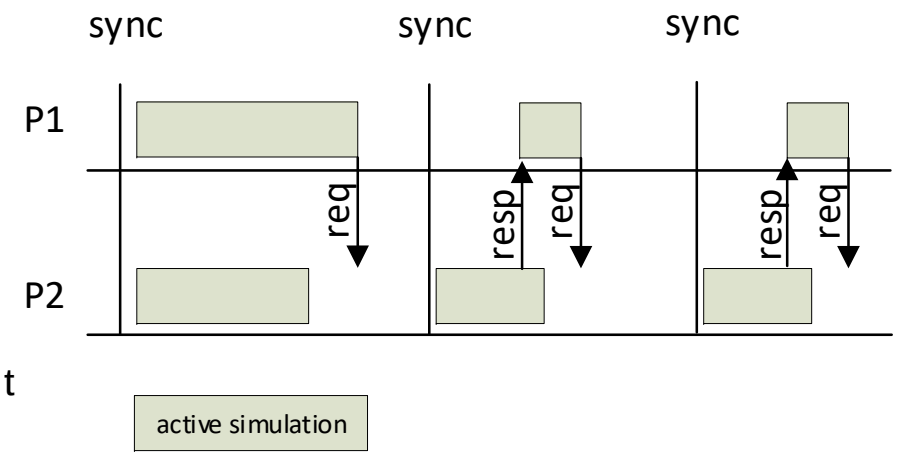
- Poor accuracy for high communication load (up to 287% timing error)
- Low to medium communication load scenarios only observe higher timing errors for long periods of async execution (up to 47%)



Distributed Simulation – Direct Mode

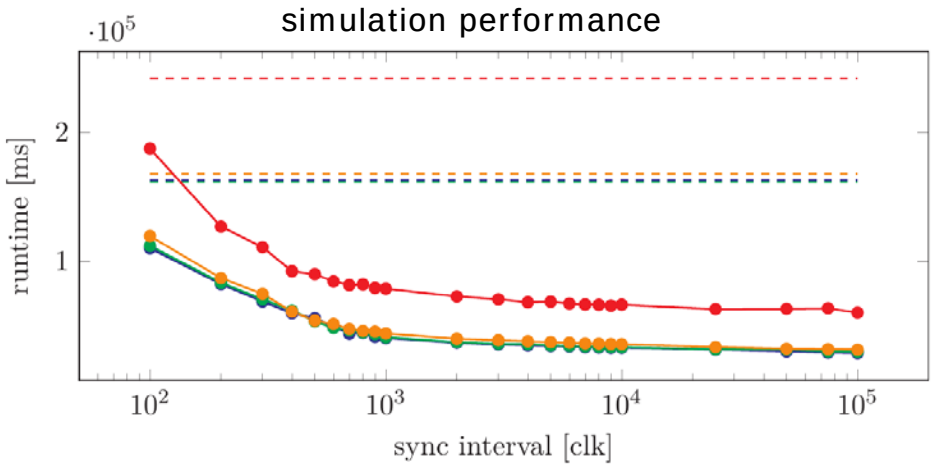
What causes the high timing error?

- Communication requests may only be acted upon after the next synchronization
- Frequent request response cycles between processes may cause significant timing errors
- Communication is directly bound to individual simulation time progression

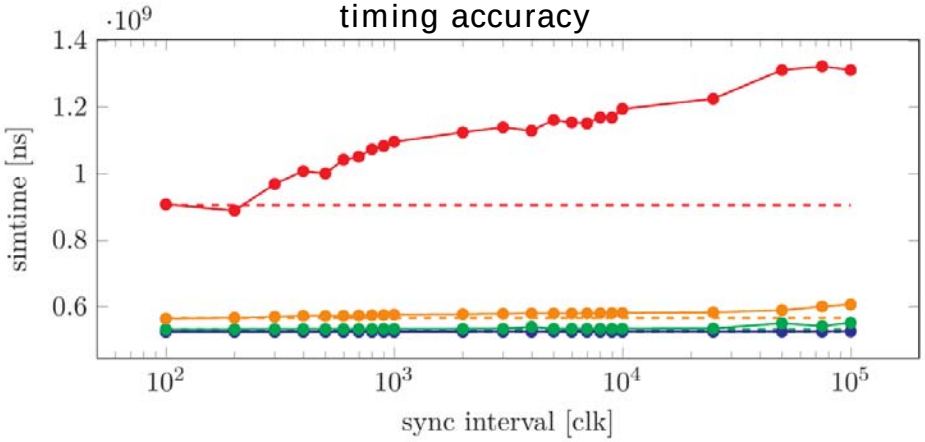


Adding an explicit communication process will act as a buffer between simulation processes

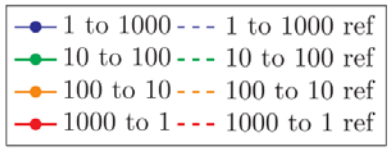
Distributed Simulation – Explicit NoC



- Ideal speed-up achieved between 1000 and 5000 clock cycles
- Low communication scenario reaches up to ~5.7x
- High communication load only achieves ~4x



- Significantly improved accuracy for high communication load (up to 45% timing error)
- Slight improvements for low to medium communication load (up to 7%)



Summary

- The SoCRocket library was extended by additional IP to investigate many-core architectures executing mixed criticality tasks
- A coding style was defined and analyzed for the on-chip network communication on the transaction-level
- Different many-core configurations were implemented, analyzed and optimized with regards to execution characteristics
- Based on this knowledge, a parallel simulation model was implemented to further optimize the simulation performance
- Using different operational modes, high performance or high accuracy can be achieved

Thank you for your attention

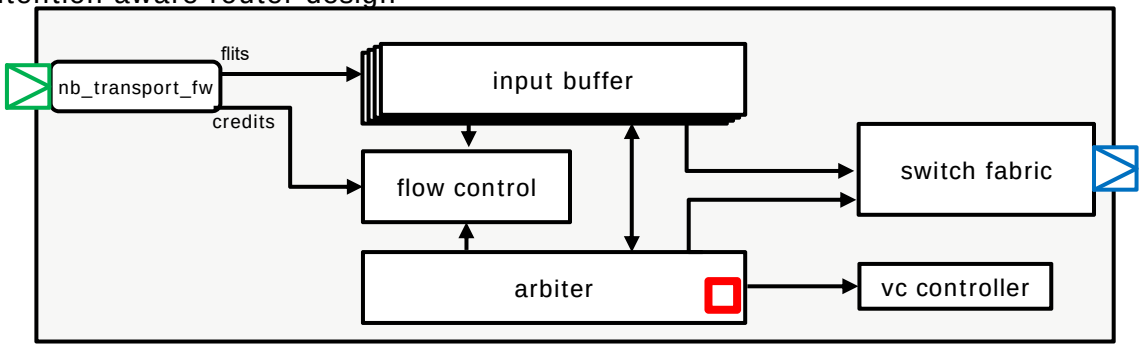
Outline

- Intro Transaction-level modelling
- Mixed-criticality many-core architecture
- **Transaction-level NoC implementation**
  - TLM NoC router
  - Payload representation
  - Standalone evaluation
- SoCRocket Many-core evaluation
- Distributed simulation
- Summary & Conclusion

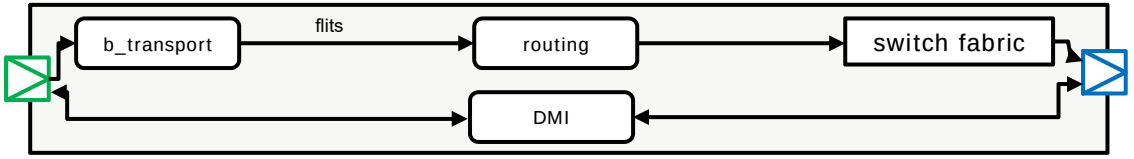
TLM2.0 NoC Router implementation

Generic router structure for contention based coding styles

Contention aware router design



Contention un-aware router design



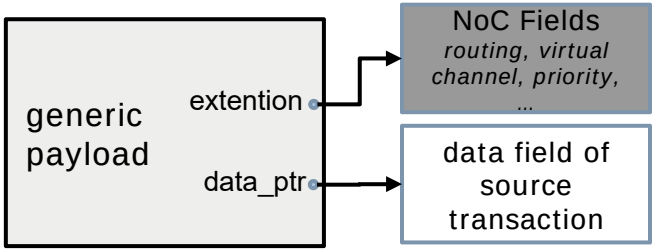
SC_THREAD multi_passthrough_target_socket multi_passthrough_initiator_socket

TLM2.0 NoC Router implementation

TLM Flit data representation (for both coding styles)

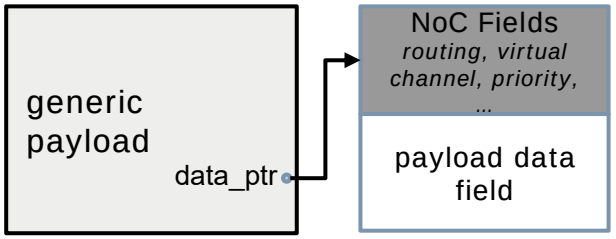
- Standard TLM2.0 generic payload objects are used for compatibility and maintainability reasons
  - Not applying generic payload fields are ignored
- Proposed options to model network specific transaction data fields:

Extensions



- NoC protocol specifics are separated from data
- Data field stays unchanged (no copy)
- Payload extension is non-ignorable

Serialization



- NoC protocol fields and data are serialized into a new bit-true data field
- True to the original protocol, simpler interfacing to RTL transactor
- Higher simulation overhead to serialize/de-serialize NoC Fields