COMMUNICATION AND I/O ARCHITECTURES FOR HIGHLY INTEGRATED MPSoC PLATFORMS

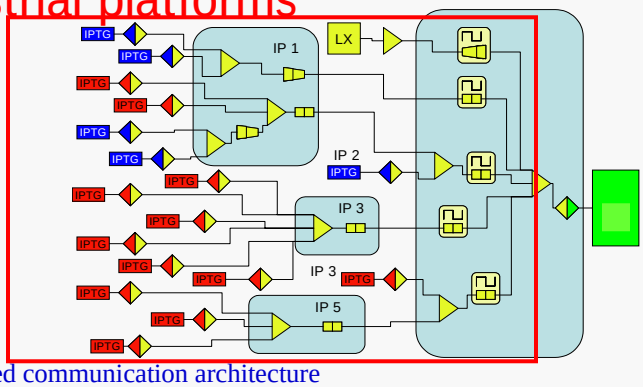
Martino Ruggiero
Luca Benini
University of Bologna
Simone Medardoni
Davide Bertozzi
University of Ferrara

In cooperation with STMicroelectronics

OUTLINE

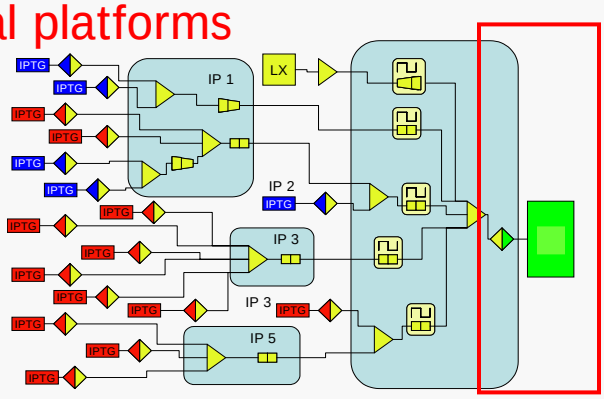
- Overview of industrial state-of-the-art set-top-box platforms
  - Segmented communication architecture
  - Off-chip SDRAM memory controller
- Crossbenchmarking of communication architectures
  - Single-layer architecture
    - ✓ Many-to-many traffic pattern
    - ✓ Many-to-one traffic pattern
  - Multi-layer architecture
    - ✓ Centralized high latency slave bottleneck
    - ✓ Faster on-chip shared memory
- Conclusions
- Hints for future work

State-of-the-art set-top-box industrial platforms



- Segmented communication architecture
  - ✓ Bridge performance is critical for the system
    - ✓Protocol conversion/adapter
    - ✓Frequency, size conversion
    - ✓Non-blocking behaviour for the injecting bus
    - ✓Ability to handle multiple outstanding transactions

State-of-the-art set-top-box industrial platforms

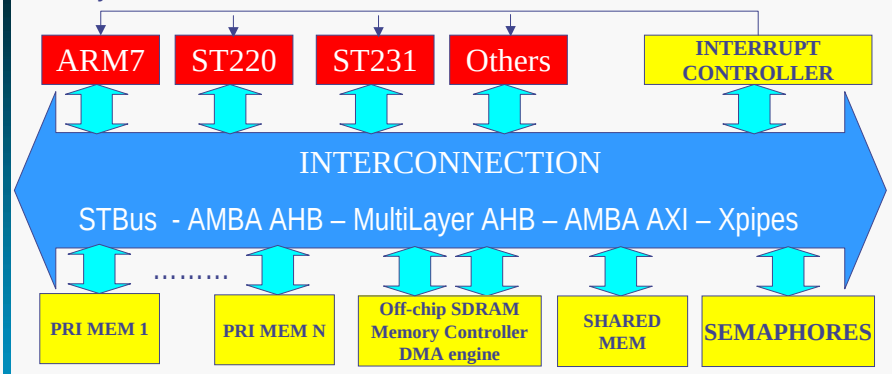


- Many platforms tend to have a global performance bottleneck:
  - ✓ memory controller for the off-chip SDRAM
    - DRAM integration is costly
    - Large processing data footprint requires large memories

Which relation between communication and memory architecture?

Virtual platform

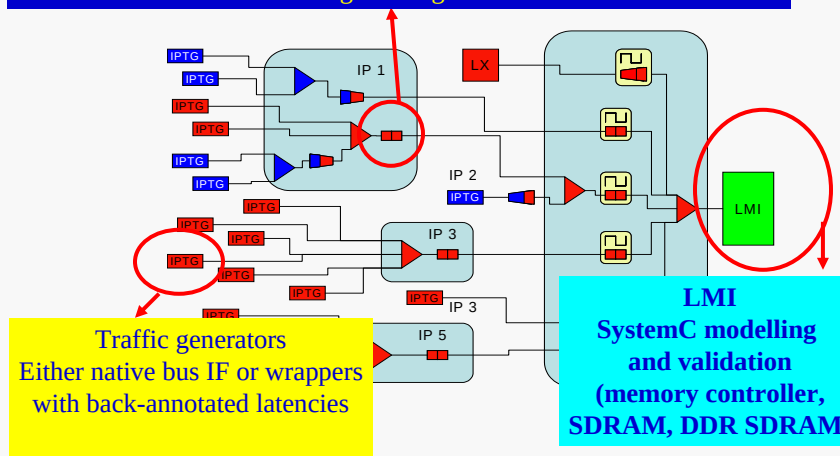
SystemC based environment for functional simulation



- Modelling accuracy emphasized
  - Cycle-accurate and bus signal-accurate
  - Processor cores modeled at the level of their IS
- Simulation speed: 60-150 kcycles/s (6 cores on P4 2.2 GHz)

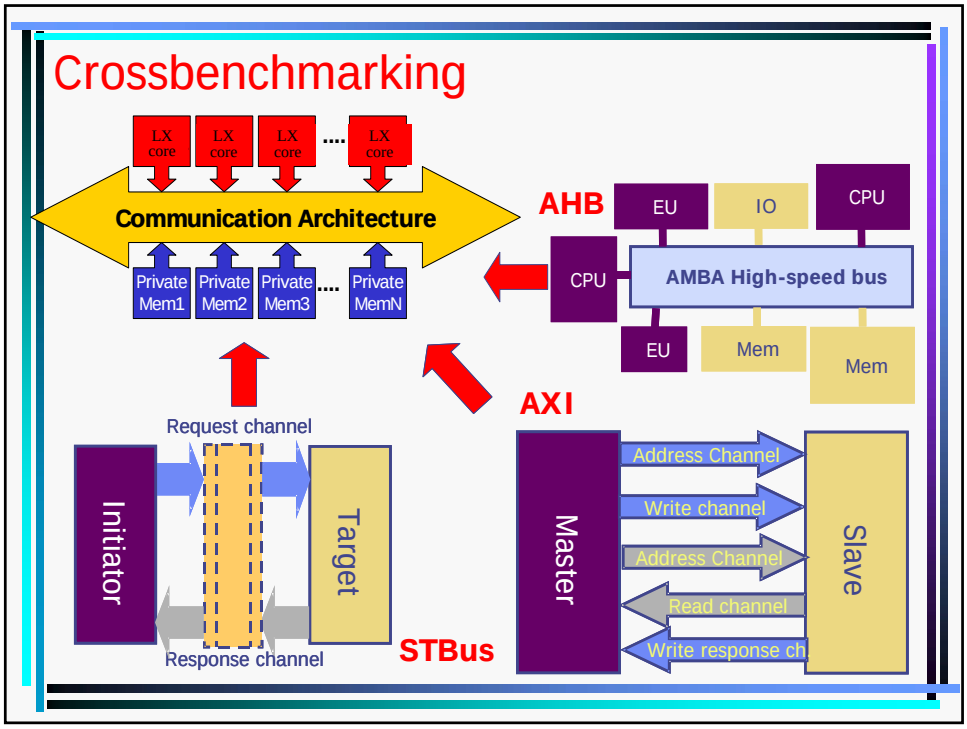
MPSIM extensions

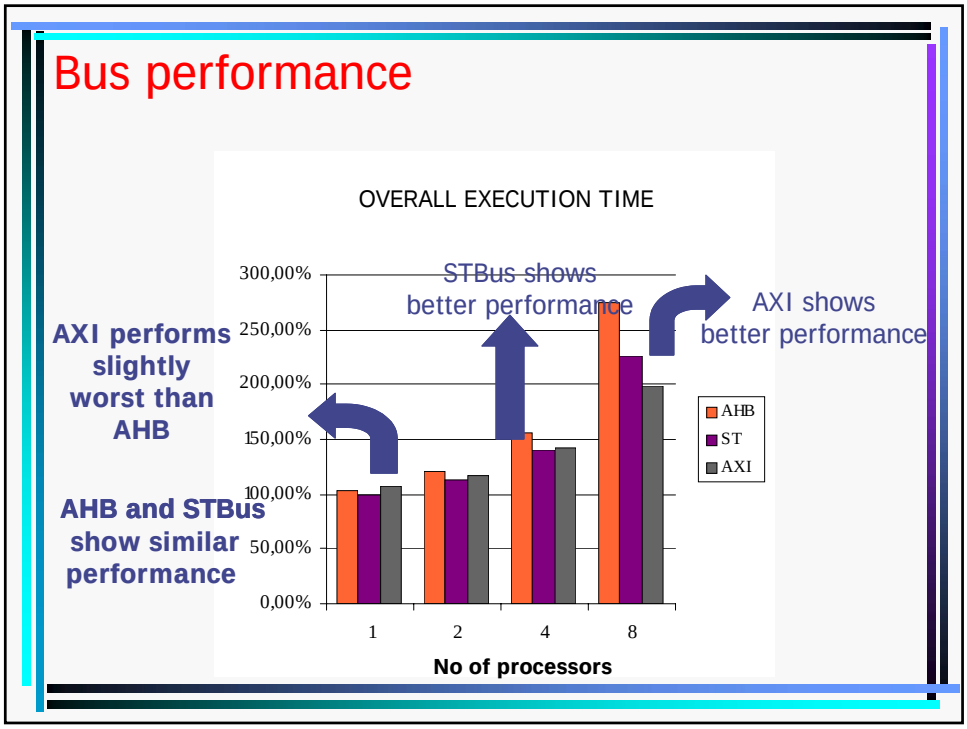
Buffer,size/freq converter for AHB-AHB and AXI-AXI, STBus-STBus
Protocol converters: AHB-AXI, AHB-STBus, AXI-STBus
Modelling of bridge latencies



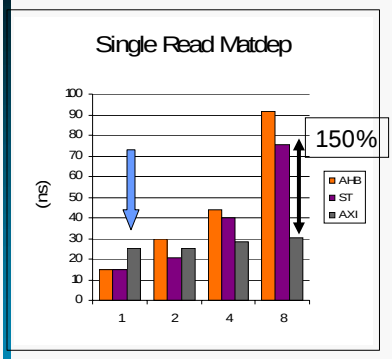
Traffic generators
Either native bus IF or wrappers with back-annotated latencies

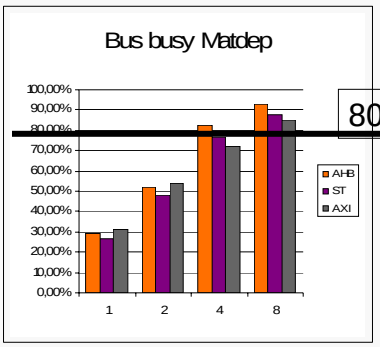
LMI
SystemC modelling and validation (memory controller, SDRAM, DDR SDRAM)


Crossbenchmarking
LX core
LX core
LX core
....
LX core
Communication Architecture
Private Mem1
Private Mem2
Private Mem3
....
Private MemN
AHB
EU
IO
CPU
CPU
AMBA High-speed bus
EU
Mem
Mem
AXI
Request channel
Initiator
Target
Response channel
STBus
Master
Address Channel
Write channel
Address Channel
Read channel
Write response ch
Slave


Bus performance
OVERALL EXECUTION TIME
300,00%
250,00%
200,00%
150,00%
100,00%
50,00%
0,00%
1
2
4
8
No of processors
AHB
ST
AXI
STBus shows better performance
AXI shows better performance
AXI performs slightly worst than AHB
AHB and STBus show similar performance

Transaction latency



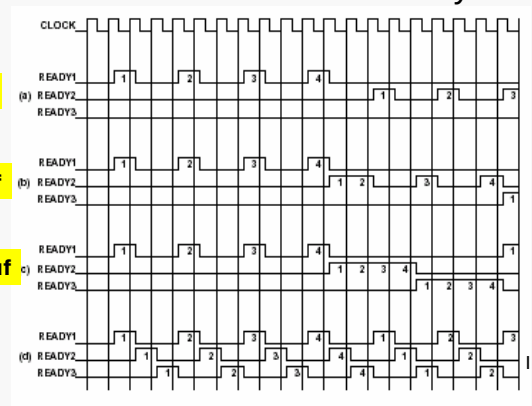


- AXI incurs higher transaction latency
  Poor performance with low bus traffic
- AXI scales better with increasing levels of bus congestion
  more complex arbiter and 5 independent channels
- 80% bus busy can be considered the performance crossing point of AXI

Fine-grain protocol analysis

allowed by protocol features

2 wait states memory



AHB

Cannot hide arbitration and slave response latency

STBUS low buf

One new request processed while a response is in progress

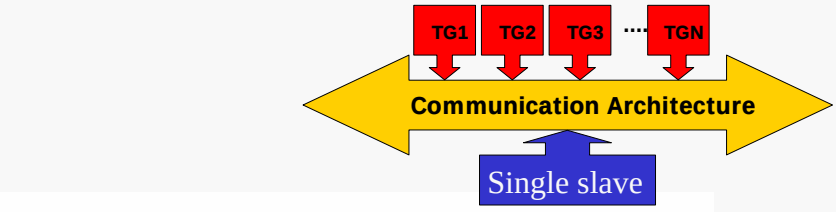
STBUS high buf

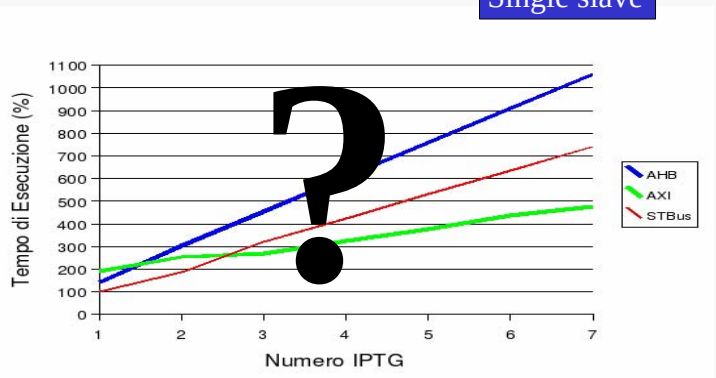
More requests processed while a response is in progress

AXI

Interleaving of transfers on the internal data lanes

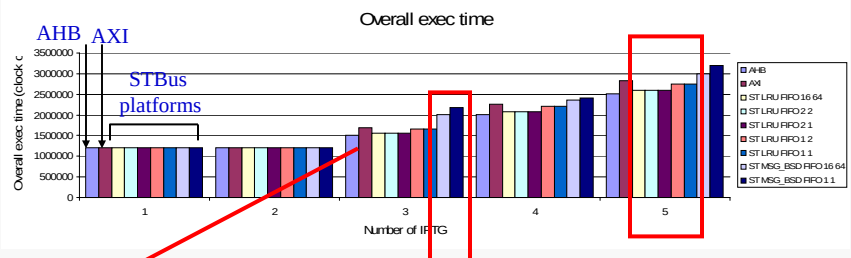
Single slave bottleneck





Execution time with single slave (on-chip shared memory)

1 wait state memory



AXI performs worst than AHB and STBus (LRU)

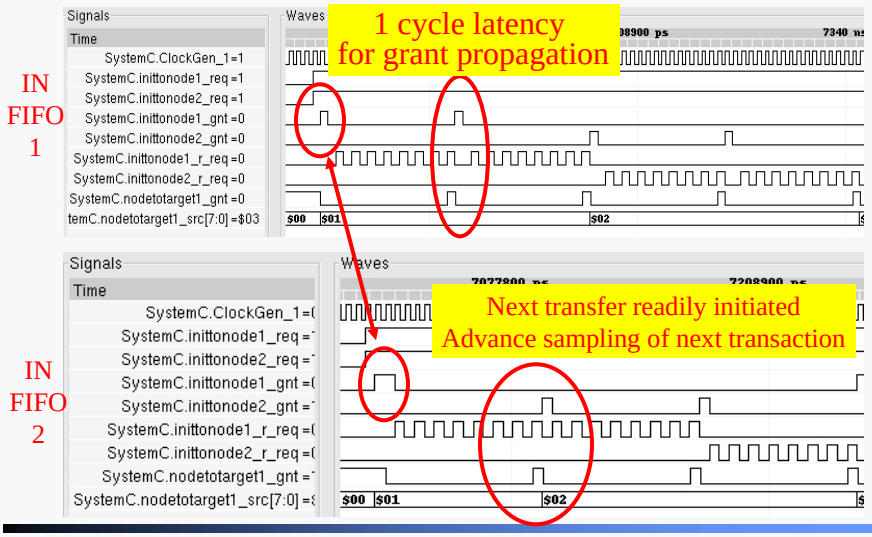
Message-based arbitration degrades performance

Performance Sensitive to Direct DataPath FIFO depth

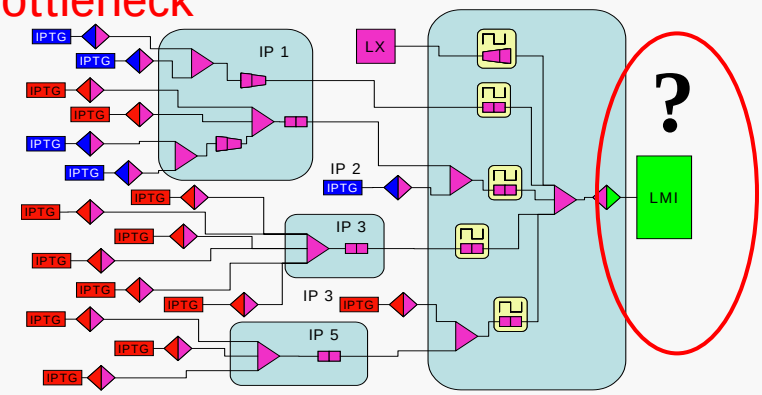
The maximum I can expect is the same performance for each bus

A centralized slave bottleneck is the best operating condition for AHB

FIFO-SIZE DEPENDENT STBus behaviour



Platform level centralized slave bottleneck

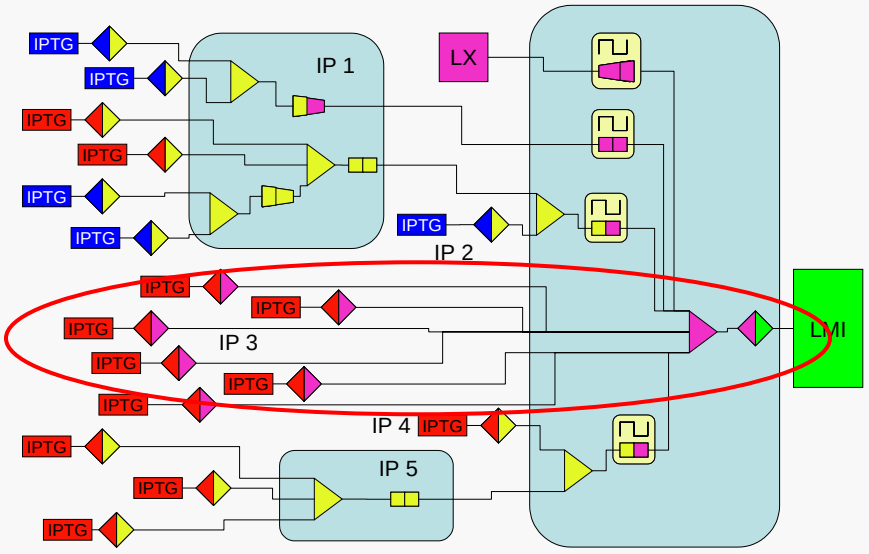


Full STBus, AHB and AXI platforms

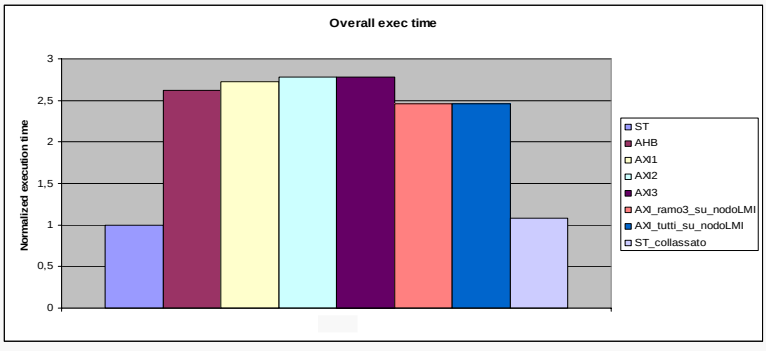
However, comparison not fair:

- AXI masters do not support multiple outstanding transactions
- Protocol converter AXI-STBus is blocking on read transactions
  - ✓Prevents memory controller optimizations

Collapsed AXI platforms

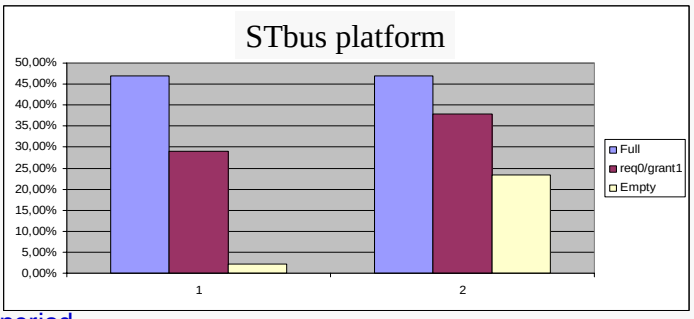


Overall execution time



- STBus leverages proprietary bridges
- AHB suffers from non-split architecture and single outstanding trans.
- AXI poor performance with centralized slave bottleneck
- AXI reduced platforms slightly improve performance
  - Now bridge performance not critical any more
  - Best scenario (heavy load) for AXI
  - However, LMI AXI-STBus conversion is still critical (blocking on reads)

LMI statistics - STBus



- First period
  - 47% full
  - 53% non-blocking (29% no requests, 24% accepting requests)
  - FIFO almost never empty (2% out of 29%)
  - Conclusion: Intensive memory traffic
- Second period
  - 47% full
  - 53% non-blocking (38% no requests, 15% accepting requests)
  - FIFO often completely empty (23% out of 38%)
  - Conclusion: bursty traffic, lower than period 1 on average

Removing AXI limitations

AMBA Platforms (AHB, Mixed AHB-AXI, AXI) → Protocol converter → LMI

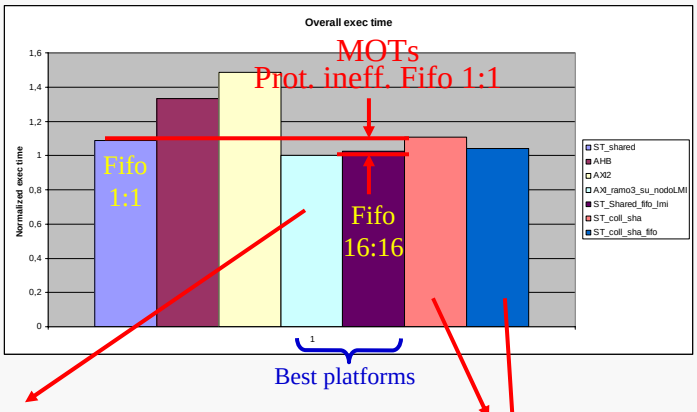
Flow bottleneck

Optimizations

Let us replace *ProtConv+LMI* with a *fast on-chip shared memory*

All Platforms (AHB, Mixed AHB-AXI, AXI, STBus) → FIFO, Shared Memory, Native bus IF

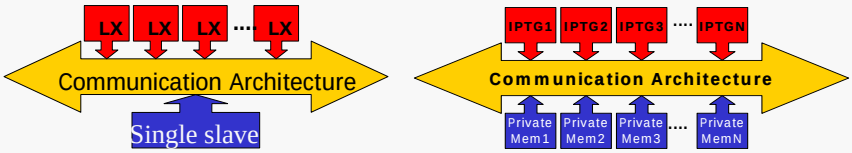
Platform performance



Collapsed AXI has no bridge/converter overhead and takes profit by the faster memory

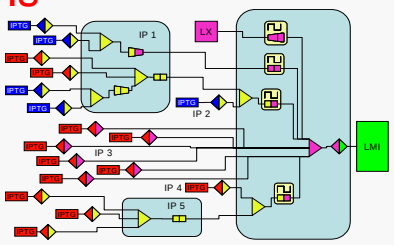
Message-based arbitration in the STBus central node.
Same improvement by adding slave FIFOs

Conclusions



- *Many-to-many* traffic pattern (*single layer architecture*):
  - AXI/STBus competition depends on % of bus utilization
  - AXI trades-off transaction latency with better scalability with heavy loads
  - AXI can allocate internal data lanes on a finer granularity than STBus
  - STBus under heavy loads can leverage crossbar instantiations
- *Many-to-one* traffic pattern (*single layer architecture*)
  - The maximum transfer efficiency is imposed by the slave
    - 1 ws SHA MEM – Max. efficiency 50%;
    - Mem. Controller with optimizations – need to keep IN FIFO full
  - Bus ability is to sustain that max efficiency
    - -AHB: pipelining control and data (OK for SHA,Not OK for LMI)
    - STBus: buffering =2 for SHA, >2 for LMI

Conclusions



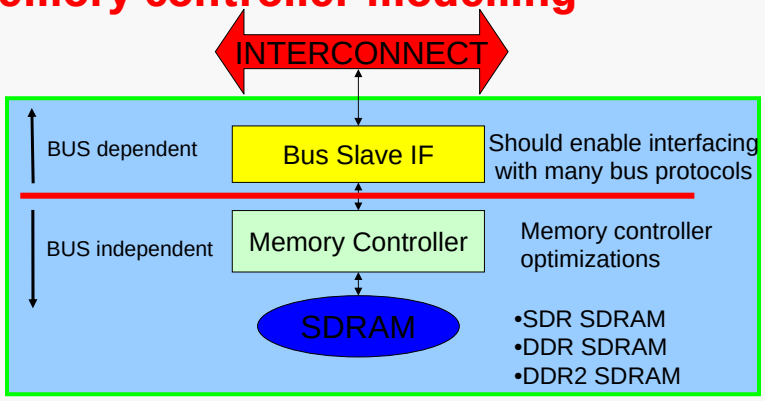
- Centralized *high latency slave bottleneck (multi-layer architecture):*
  - All you can require from a bus:
    distributed buffering & multiple outstanding transactions & split bus
    - larger initiator-perceived bandwidth
    - hides bus topology (and multi-layer latency)
- A *faster on-chip memory*
  - the buffer chain from initiator-to-target does not fill up
    - performance affected by multi-layer latency

Other bus features are less critical,
therefore bus differentiation is very difficult with this platform template

Hints for future work

- Bridges relief the lack of bus scalability..
- ..but introduce large complexity
- Why not using bridge-free multi-hop solutions (Networks-on-Chip) ?
- Optimize the I/O system so to take profit by the specific bus features
- higher bandwidth memory controller
- Multiple I/O ports
- On-chip shadowing shared memory(ies)

Memory controller modelling



Which interface architecture to the bus?
- Multi-port controller with arbitration on input ports
- DMA-capable controller

Which memory controller optimizations?
- transaction merging
- variable-depth lookahead