# Interconnect Optimization with SoC Designer

The SoC interconnect must be designed and optimized to support a heterogeneous mix of data paths which may inherently have widely varying performance characteristics. The fabric must reliably deliver the required throughput and hide latency for performance critical paths while simultaneously managing the flow of traffic for slower paths and ports requiring lower bandwidth. Thus the system bus as a whole must strike the appropriate between latency and throughput for the collection data paths. Optimizing around this balance is essential to minimizing power, performance, area (PPA) costs and avoiding an inefficient, over-designed SoC.

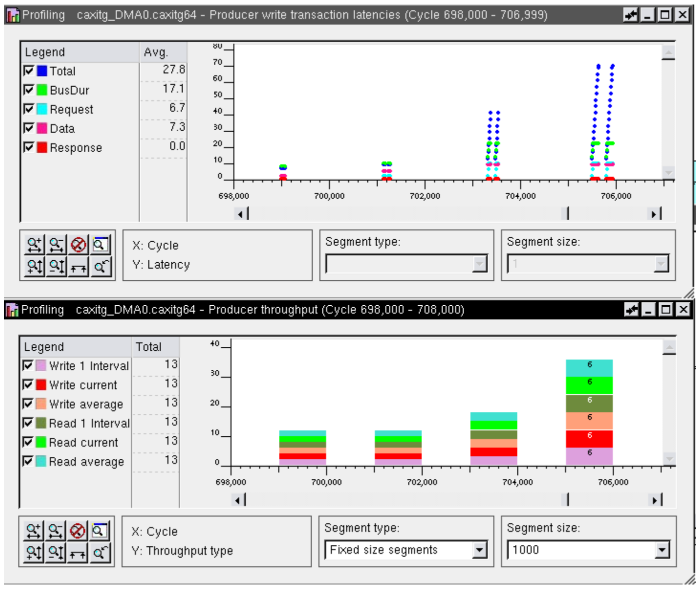
There are many other questions involved with optimization to allow for a balanced SoC: What is the best way to isolate and eliminate performance bottlenecks? How can load-balancing and quality of service (QoS) simultaneously be ensured? How will cache coherency impact the interconnect traffic – and system throughput?

## Architectural exploration Vs. Performance tuning: Chicken or Egg?

There is a circular dependency between the design phases related to interconnect optimization. Early optimization of the system architecture to provide robust performance across a wide range of I/O profiles through the bus can limit the ability to fine tune the critical performance paths of the final design. Conversely, the application traffic from the actual IP cores driving the bus as well as real-world variation in the behavior of the slave components handling bus traffic can compromise the intended architectural design of the interconnect.

Usually the CPU core(s) and possibly the GPU are selected prior to interconnect but other IP blocks including the main memory, DMAs, general purpose peripherals are yet to be selected. Thus, the interconnect is to be designed and optimized with an incomplete understanding of the actual workloads that it will need arbitrate and load balance, as well as uncertainty of the latency and throughput constraints that the actual slave devices will impose.

Unfortunately, key design decisions for interconnect optimization require an understanding of the final system. Consider the performance sensitive path between the CPU and main memory. The system bus must harmonize with the main memory controller configuration and programming such that the memory bandwidth is maximized while the latency through the bus is minimized. Other less performance sensitive paths may be slower or may require less bandwidth. This design tradeoff is fundamental and must be accurately profiled, characterized, and displayed for the SoC architect.



Profiling caxitg_DMA0.caxitg64 - Producer write transaction latencies (Cycle 698,000 - 706,999)

| Legend | Avg. |
|---|---|
| 🟦 Total | 27.8 |
| 🟩 BusDur | 17.1 |
| 🟦 Request | 6.7 |
| 🟪 Data | 7.3 |
| 🟥 Response | 0.0 |

X: Cycle
Y: Latency
Segment type:
Segment size:

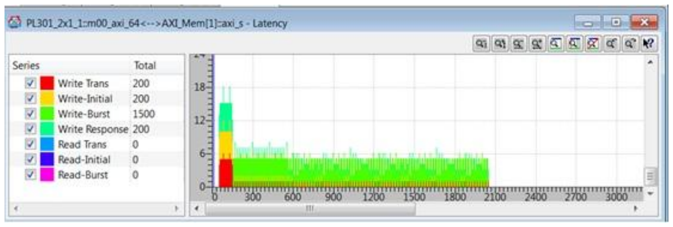
Profiling caxitg_DMA0.caxitg64 - Producer throughput (Cycle 698,000 - 708,000)

| Legend | Total |
|---|---|
| 🟪 Write 1 Interval | 13 |
| 🟥 Write current | 13 |
| 🟧 Write average | 13 |
| 🟩 Read 1 Interval | 13 |
| 🟩 Read current | 13 |
| 🟦 Read average | 13 |

X: Cycle
Y: Throughput type
Segment type: Fixed size segments
Segment size: 1000

**The relationship between latency & throughput visualized in SoC Designer**

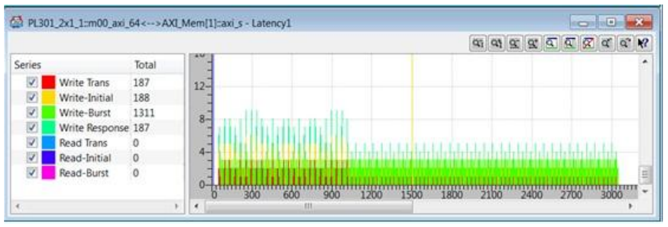
A comprehensive I/O prioritization scheme must be established such that the fabric can provide flow control and arbitration to deliver appropriate QoS for the application. Moreover, real world dynamics such as transient shifts in traffic, head-of-line blocking, or small latency deviations from slave devices must not disrupt the intended flow control of the interconnect as seen below:



PL301_2x1_1::m00_axi_64<-->AXI_Mem[1]::axi_s - Latency

| Series | Total |
|---|---|
| 🟥 Write Trans | 200 |
| 🟨 Write-Initial | 200 |
| 🟩 Write-Burst | 1500 |
| 🟩 Write Response | 200 |
| 🟦 Read Trans | 0 |
| 🟦 Read-Initial | 0 |
| 🟪 Read-Burst | 0 |

**Test Case Workload: High priority traffic completes in the first 150 cycles followed by low priority traffic**



PL301_2x1_1::m00_axi_64<-->AXI_Mem[1]::axi_s - Latency1

| Series | Total |
|---|---|
| 🟥 Write Trans | 187 |
| 🟨 Write-Initial | 188 |
| 🟩 Write-Burst | 1311 |
| 🟩 Write Response | 187 |
| 🟦 Read Trans | 0 |
| 🟦 Read-Initial | 0 |
| 🟪 Read-Burst | 0 |

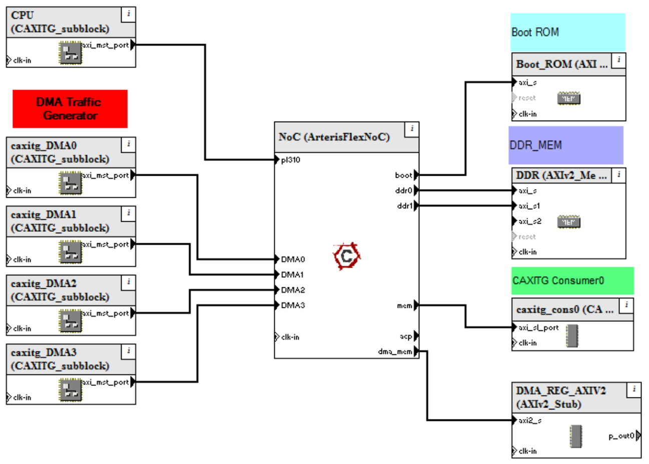
**Real World Workload: A single cycle delay on the target results in interleaving of high and low priority traffic and a 10x delay in completion of the high priority workload**

In this example, back-pressure on the fabric resulting from a single cycle delay at the memory controller led to a full queue and head-of-line blocking condition on the input FIFO for the high priority traffic. As a result, the fabric arbiter allowed low priority traffic to access the fabric and get issued to the memory controller ahead of blocked high priority traffic. These complex interactions typically only rear their ugly head with real world workloads.

This "chicken or egg" design dilemma of configuring interconnect to meet performance objectives while providing robust QoS early in the design process when the exact I/O workloads and target constraints are unknown **can be resolved**. The circular interconnect optimization process can be translated to an iterative approach that appropriately balances architectural optimization with 100% accurate performance analysis progressively as key SoC design decisions are made.

## A Two-Phased Approach: Phase 1 – Architectural Exploration

The first phase involves configuring and building a 100% cycle accurate model of interconnect available from Carbon IP Exchange and quickly and easily isolate performance bottlenecks with traffic generators and flexible memory sub-system models.



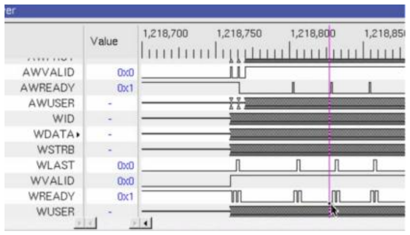
**CAXITG & Memory Models configured for architectural exploration with 100% cycle accurate Arteris FlexNoC interconnect**

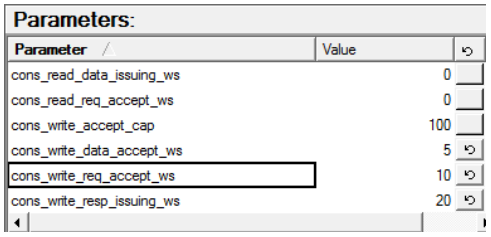
The Carbon AXI Traffic Generator (CAXITG) enables performance analysis of AMBA® AXI-based systems in Carbon SoC Designer. Workloads can be dialed in at simulation time to configure random data rates, read/write mix, and more subtle traffic features such as burstiness and issuing capability (pipelining). The CAXITG profiling capabilities also allow for:

- AXI throughput & latency profiling
- Transaction Tracing
- AXI back-pressure analysis
- CAXITG can also be configured as a traffic consumer

**Waveforms can be viewed within SoC Designer to analyze back-pressure to identify performance bottlenecks**

**CAXITG consumer wait states can be used to simulate lower cost, lower performing memory sub-systems**




Parameters:

| Parameter | Value |
|---|---|
| cons_read_data_issuing_ws | 0 |
| cons_read_req_accept_ws | 0 |
| cons_write_accept_cap | 100 |
| cons_write_data_accept_ws | 5 |
| cons_write_req_accept_ws | 10 |
| cons_write_resp_issuing_ws | 20 |

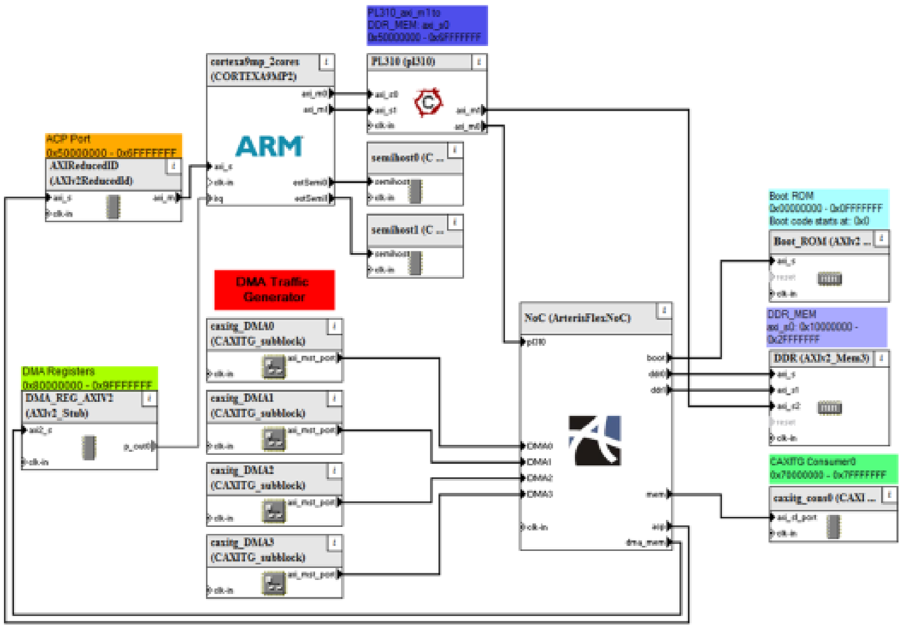
The system above allows for a robust sweep of wide variety of workloads through the interconnect, stress tests, to congest the fabric and isolate performance bottlenecks, and "What if" analysis to explore the tradeoff between memory subsystem cost and performance implications for the fabric.

Once again, it may be tempting to use a fully connected mesh to hook together all of your IP with wide, low-latency busses but you'll probably spend more power than you need and have unnecessarily fast speeds on some data transfers.

## Phase 2 – Optimizing for real world workloads

While the architectural exploration phase is essential for efficient high level interconnect optimization, performance must be validated with The IP used in the design and real world workloads. The reusability of virtual prototypes make them well suited for starting interconnect optimization with a Phase I prototype early in the design phase and then migrating to a Phase 2 real-world prototype by plugging in 100% implementation accurate IP blocks as they are selected later in the design process.

For example the system described earlier can be reused adding an ARM Cortex A9 processor in place of the traffic generator.


PL310_axi_m1 to DDR_MEM: axi_s0 0x50000000 - 0x6FFFFFFF
cortexa9mp_2cores (CORTEXA9MP2)
PL310 (pl310)
ARM
ACP Port 0x50000000 - 0x6FFFFFFF
AXIReducedID (AXIv2ReducedId)
semihost0 (C ...
semihost1 (C ...
DMA Traffic Generator
caxitg_DMA0 (CAXITG_subblock)
caxitg_DMA1 (CAXITG_subblock)
caxitg_DMA2 (CAXITG_subblock)
caxitg_DMA3 (CAXITG_subblock)
DMA Registers 0x80000000 - 0x9FFFFFFF
DMA_REG_AXIV2 (AXIv2_Stub)
NoC (ArterisFlexNoC)
Boot ROM 0x00000000 - 0x0FFFFFFF Boot code starts at: 0x0
Boot_ROM (AXIv2 ...
DDR_MEM axi_s0: 0x10000000 - 0x2FFFFFFF
DDR (AXIv2_Mem3)
CAXITG Consumer0 0x70000000 - 0x7FFFFFFF
caxitg_cons0 (CAXI ...
pl310
boot
ddr0
ddr1
DMA0
DMA1
DMA2
DMA3
mem
acp
dma_mem
clk-in

**Arteris FlexNoC platform with ARM Cortex A9 dual core CPU**

The design tradeoffs explored earlier, such as arbitration and QoS, can be revisited and validated against actual multicore CPU traffic. The platform can be further modified as DMA IP blocks and memory controllers are selected.

## Coherency & Accuracy

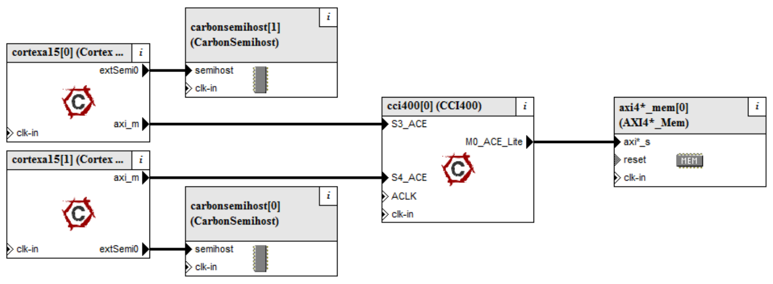
The migration to hardware based cache coherency in SoC's has introduced significant complexity to the interconnect optimization process. Artificial workloads with custom traffic generators and behavioral models of coherent IP add too much overhead to the development cycle and are too error prone to reliably characterize and validate cache coherency. Accuracy, that is, 100% implementation accurate models of IP with real world traffic is essential.

The AXI Coherency Extensions (ACE) to the AMBA bus protocols complicate both the transactional protocol and the design (and subsequent verification) of a coherent interconnect. The bulk of the complexity that follows from providing ACE coherency in the interconnect comes from:

1) Command handling for nearly 20 read and write commands defined in the ACE specification
2) Maintenance of cache state required to ensure coherency across all appropriate domains as well as enforcement of legal vs. illegal operation.
3) Correct execution of Barrier transactions across multiple cores issuing such transactions

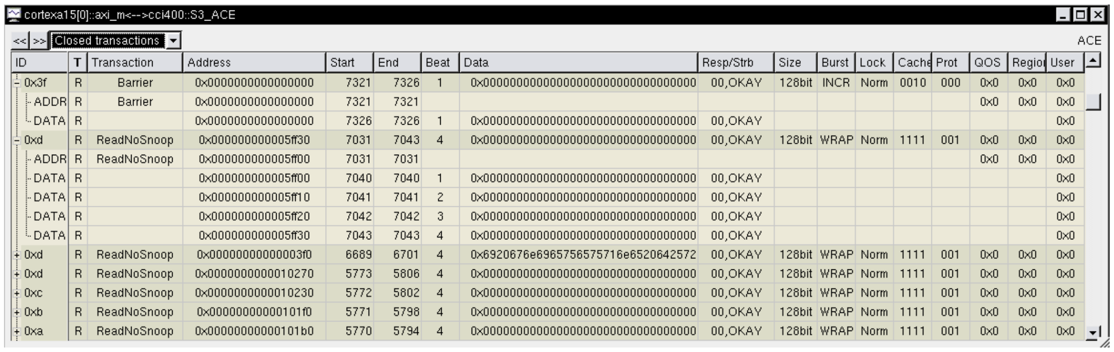
For the SoC architect, it is critical to understand the impact of ACE coherency traffic on the fabric and on the overall system. Unfortunately, due to the complexity of the multicore cache coherency and the ACE specification, this is not a task well suited for traffic generators and behavioral models. Only 100% implementation accurate models of IP masters and interconnect will suffice for generating real world coherency workloads, verifying coherency within the fabric, and analyzing its impact on performance.

The A15 Bare Metal CPAK provides a great staring point to immediately start analyzing cache coherency and its performance implications with an ARM Cortex A15 and ARM CCI-400 cache–coherent Interconnect.



**Carbon A15 bare metal CPAK multi-processor reference platform**

ACE monitoring capabilities within SoC Designer make it easy to analyze transactional behavior of coherency operations as well as the impact of barrier transactions.



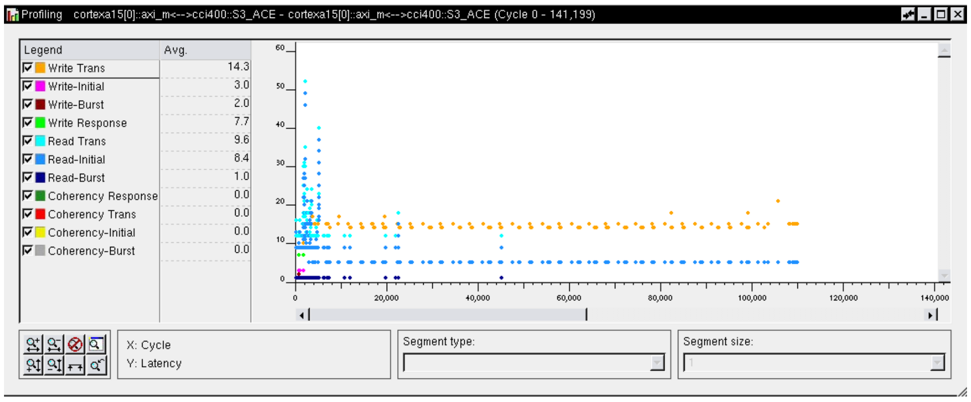
cortexa15[0]::axi_m<-->cci400::S3_ACE

<< >> Closed transactions — ACE

| ID | T | Transaction | Address | Start | End | Beat | Data | Resp/Strb | Size | Burst | Lock | Cache | Prot | QOS | Region | User |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x3f | R | Barrier | 0x0000000000000000 | 7321 | 7326 | 1 | 0x00000000000000000000000000000000 | 00,OKAY | 128bit | INCR | Norm | 0010 | 000 | 0x0 | 0x0 | 0x0 |
| ADDR | R | Barrier | 0x0000000000000000 | 7321 | 7321 | | | | | | | | | 0x0 | 0x0 | 0x0 |
| DATA | R | | 0x0000000000000000 | 7326 | 7326 | 1 | 0x00000000000000000000000000000000 | 00,OKAY | | | | | | | | 0x0 |
| 0xd | R | ReadNoSnoop | 0x000000000005ff30 | 7031 | 7043 | 4 | 0x00000000000000000000000000000000 | 00,OKAY | 128bit | WRAP | Norm | 1111 | 001 | 0x0 | 0x0 | 0x0 |
| ADDR | R | ReadNoSnoop | 0x000000000005ff00 | 7031 | 7031 | | | | | | | | | 0x0 | 0x0 | 0x0 |
| DATA | R | | 0x000000000005ff00 | 7040 | 7040 | 1 | 0x00000000000000000000000000000000 | 00,OKAY | | | | | | | | 0x0 |
| DATA | R | | 0x000000000005ff10 | 7041 | 7041 | 2 | 0x00000000000000000000000000000000 | 00,OKAY | | | | | | | | 0x0 |
| DATA | R | | 0x000000000005ff20 | 7042 | 7042 | 3 | 0x00000000000000000000000000000000 | 00,OKAY | | | | | | | | 0x0 |
| DATA | R | | 0x000000000005ff30 | 7043 | 7043 | 4 | 0x00000000000000000000000000000000 | 00,OKAY | | | | | | | | 0x0 |
| 0xd | R | ReadNoSnoop | 0x00000000000003f0 | 6689 | 6701 | 4 | 0x6920676e6965756575716e6520642572 | 00,OKAY | 128bit | WRAP | Norm | 1111 | 001 | 0x0 | 0x0 | 0x0 |
| 0xd | R | ReadNoSnoop | 0x0000000000010270 | 5773 | 5806 | 4 | 0x00000000000000000000000000000000 | 00,OKAY | 128bit | WRAP | Norm | 1111 | 001 | 0x0 | 0x0 | 0x0 |
| 0xc | R | ReadNoSnoop | 0x0000000000010230 | 5772 | 5802 | 4 | 0x00000000000000000000000000000000 | 00,OKAY | 128bit | WRAP | Norm | 1111 | 001 | 0x0 | 0x0 | 0x0 |
| 0xb | R | ReadNoSnoop | 0x00000000000101f0 | 5771 | 5798 | 4 | 0x00000000000000000000000000000000 | 00,OKAY | 128bit | WRAP | Norm | 1111 | 001 | 0x0 | 0x0 | 0x0 |
| 0xa | R | ReadNoSnoop | 0x00000000000101b0 | 5770 | 5794 | 4 | 0x00000000000000000000000000000000 | 00,OKAY | 128bit | WRAP | Norm | 1111 | 001 | 0x0 | 0x0 | 0x0 |

**Transactional monitoring of Barrier ACE transactions in context of application**

Additionally, the profiling capabilities of AXI 4 Transactional interfaces that are built into the SoC Designer simulator allow fast comparison of non-coherent application workloads against coherent workloads. The impact of coherency operations on average latency can be clearly visualized by profiling the ACE traffic over the SoC Designer transactional interface:

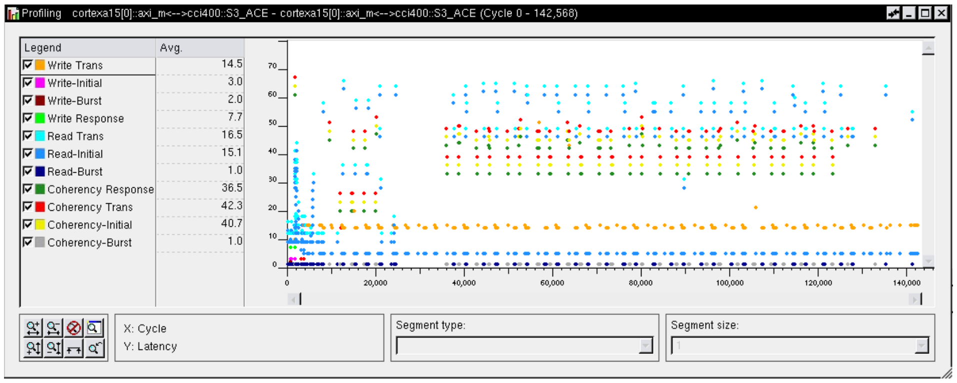


Profiling cortexa15[0]::axi_m<-->cci400::S3_ACE - cortexa15[0]::axi_m<-->cci400::S3_ACE (Cycle 0 - 141,199)

| Legend | Avg. |
|---|---|
| 🟧 Write Trans | 14.3 |
| 🟪 Write-Initial | 3.0 |
| 🟥 Write-Burst | 2.0 |
| 🟩 Write Response | 7.7 |
| 🟦 Read Trans | 9.6 |
| 🟦 Read-Initial | 8.4 |
| 🟦 Read-Burst | 1.0 |
| 🟩 Coherency Response | 0.0 |
| 🟥 Coherency Trans | 0.0 |
| 🟨 Coherency-Initial | 0.0 |
| ⬜ Coherency-Burst | 0.0 |

X: Cycle
Y: Latency
Segment type:
Segment size:

**Non-coherent A15 application workload with average read latencies of 9.6 cycles**



Profiling cortexa15[0]::axi_m<-->cci400::S3_ACE - cortexa15[0]::axi_m<-->cci400::S3_ACE (Cycle 0 - 142,568)

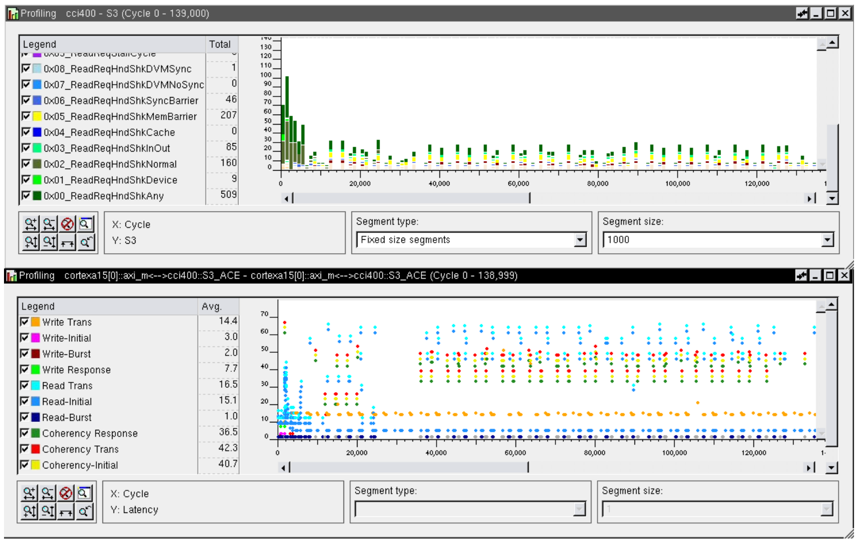
| Legend | Avg. |
|---|---|
| 🟧 Write Trans | 14.5 |
| 🟪 Write-Initial | 3.0 |
| 🟥 Write-Burst | 2.0 |
| 🟩 Write Response | 7.7 |
| 🟦 Read Trans | 16.5 |
| 🟦 Read-Initial | 15.1 |
| 🟦 Read-Burst | 1.0 |
| 🟩 Coherency Response | 36.5 |
| 🟥 Coherency Trans | 42.3 |
| 🟨 Coherency-Initial | 40.7 |
| ⬜ Coherency-Burst | 1.0 |

X: Cycle
Y: Latency
Segment type:
Segment size:

**Coherent A15 workload from the same application shows > 70% increase in average read latency**

The shape of the bus traffic is visualized with the coherency operations and the associated high latency reads displayed standing apart from the normal application traffic. In addition to the visual representation of the traffic, SoC Designer's profiler displays performance statistics such as average latency per transaction type so that the 9.6 cycle average Read latency of non-coherent traffic as well as the 70 % increase in latency when compared with coherent traffic (16.5 cycle average read latency) can be readily observed and calculated.  Cache coherency of shared memory access in a multiprocessor environment is essential to system performance and has clear benefits to the application developer and system architect.  The costs of this coherency are not usually as clear and often overlooked.  The SoC Designer profiling capabilities help the architect quantify this cost and interpret the cost in the broader context of overall system performance.

To get a deeper understanding of the nature of the coherency on application traffic, the internal handshaking events of the CCI-400 can be profiled, analyzed, and visually correlated to the bus traffic.


Profiling   cci400 - S3 (Cycle 0 - 139,000)

| Legend | Total |
|---|---|
| 🟪 0x09_ReadReqStallCycle | 0 |
| 🟦 0x08_ReadReqHndShkDVMSync | 1 |
| 🟦 0x07_ReadReqHndShkDVMNoSync | 0 |
| 🟦 0x06_ReadReqHndShkSyncBarrier | 46 |
| 🟨 0x05_ReadReqHndShkMemBarrier | 207 |
| 🟦 0x04_ReadReqHndShkCache | 0 |
| 🟩 0x03_ReadReqHndShkInOut | 85 |
| 🟩 0x02_ReadReqHndShkNormal | 160 |
| 🟩 0x01_ReadReqHndShkDevice | 9 |
| 🟩 0x00_ReadReqHndShkAny | 509 |

X: Cycle
Y: S3
Segment type: Fixed size segments
Segment size: 1000

Profiling   cortexa15[0]::axi_m<-->cci400::S3_ACE - cortexa15[0]::axi_m<-->cci400::S3_ACE (Cycle 0 - 138,999)

| Legend | Avg. |
|---|---|
| 🟧 Write Trans | 14.4 |
| 🟪 Write-Initial | 3.0 |
| 🟥 Write-Burst | 2.0 |
| 🟩 Write Response | 7.7 |
| 🟦 Read Trans | 16.5 |
| 🟦 Read-Initial | 15.1 |
| 🟦 Read-Burst | 1.0 |
| 🟩 Coherency Response | 36.5 |
| 🟥 Coherency Trans | 42.3 |
| 🟨 Coherency-Initial | 40.7 |

X: Cycle
Y: Latency
Segment type:
Segment size:

**CCI-400 Coherency event profiling in relation to A15 ACE interface transactions**

Understanding the impact of coherency operations within the interconnect and to the overall system performance can help architects partition their designs across multiple networks on chip.  Additionally, if performance penalties associated with a particular coherent IP block can be well understood with 100% accurate traffic analysis, the optimal appropriate coherency domain can be selected to mitigate the system performance costs.