Getting the Most Out of the ARM® CoreLink™ NIC-400

William Orme, ARM
Bill Neifert, Carbon Design Systems
ATC-300 **October 31, 2013**

Abstract

The ARM CoreLink NIC-400 Network Interconnect is an extremely versatile piece of IP capable of maximizing performance for high-throughput applications, minimizing power consumption for mobile devices, and guaranteeing a consistent quality of service. To achieve this versatility, the IP is highly configurable. To assist you finding the right configuration for your particular application we introduce the concept of virtual prototyping, modeling the NIC-400 and how to drive & analyze traffic through it. This paper highlights the key features of the NIC-400 as well as a methodology, based on real system traffic and software, for making the best design decisions.

ARM CoreLink NIC-400 Overview

The CoreLink NIC-400 Network Interconnect provides a highly configurable network of interconnect switches and bridges to connect up to 128 AXI or AHB-Lite masters to up to 64 AXI, AHB-Lite or APB slaves in a single NIC-400 instantiation.

The NIC-400 is the 4th generation AXI interconnect from ARM and is delivered as a base product of AMBA AXI3 and/or AXI4 interconnect with three optional, license-managed advanced features: QoS-400 Advanced Quality of Service to dynamically regulate traffic entering the network; QVN-400 QoS Virtual Networks to prevent blocking at arbitration points; and TLX-400 Thin Links to reduce routing congestion and ease timing closure for long paths.

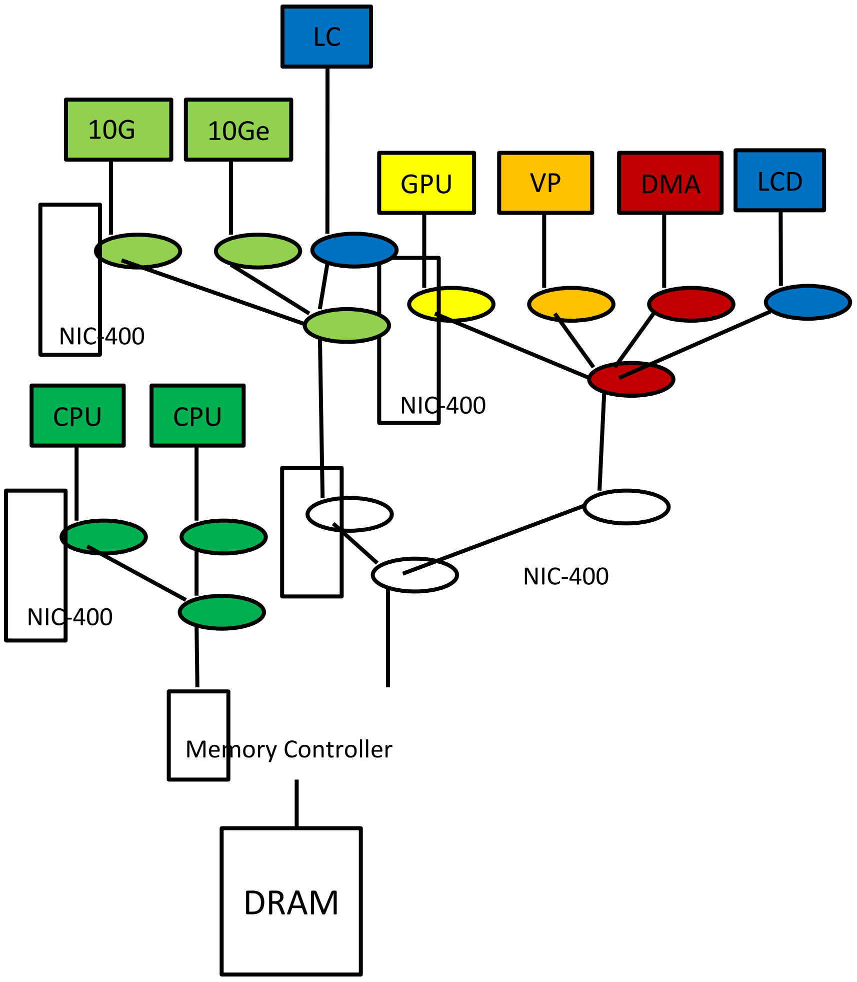
The design and configurability of the NIC-400 allows the user to implement the lowest latency highest performance interconnect for their set of master and slave requirements while minimizing silicon area and power.

The NIC-400 network of switches allows scaling up to very large numbers of masters and slaves while maintaining high maximum operating frequencies. As is common in most SoCs, the NIC-400 connects up IP blocks with a range of different AMBA interfaces with multiple data width and variable clock frequencies. Crossing all domains is optimized for low latency.

The NIC-400 provides a wide range of forward and reverse registering options on each of the AXI channels to assist in getting timing closure at the required frequency while not inserting any more gates than absolutely necessary to reduce latency, area and power.

To achieve the highest throughput while reducing stalls of the master to the bare minimum, the NIC-400 has fully configurable buffer depths, in order to keep gate count no higher than necessary.

The designer can select the topology of the network of switches to increase the efficiency of the interconnect in many ways. Traffic streams from multiple masters can be combined to increase wire utilization, reducing congestion. Grouping of masters by location can shorten paths between IP blocks and switches. The division of large switches in to multiple smaller switches can allow increased frequencies and provide low latencies for critical paths such as between CPU and DDR main memory.

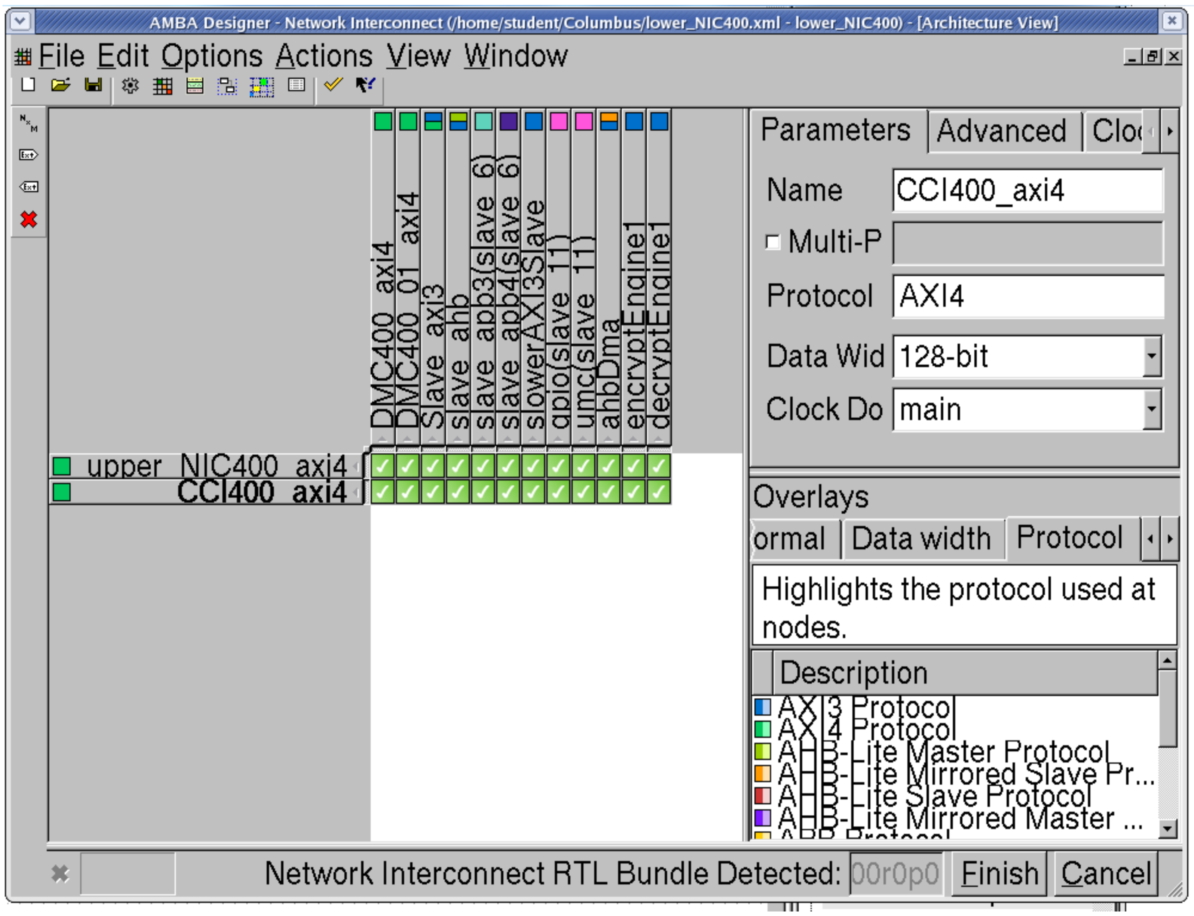


Each switch can select the appropriate data width and clock frequency to meet performance target while minimizing power and area. Different switches can be placed in different domains allowing hierarchical clock gating to reduce power whenever each domain is idle.

The NIC-400 switches and bridges support both AMBA AXI3 and the new AMBA 4 AXI4, with less wires (no WIDs) and enhanced streaming performance (longer burst support). All bridging between AXI4 and AXI3 is handled seamlessly by the NIC-400. APB support is extended to AMBA 4 APB4 with new write strobes and TrustZone signalling.

Configuring the NIC-400

Designers use the easy to drive AMBA Designer tool to configure the NIC-400. The user starts by defining the masters and slaves in his system, what bus protocol, bus width and clock each use and filling in a matrix of the required master-slave connectivity.


File Edit Options Actions View Window
Parameters Advanced Cloc
Overlays
ormal Data width Protocol
Finish Cancel

AMBA Designer - Network Interconnect (/home/student/Columbus/lower_NIC400.xml - lower_NIC400) - [Architecture View]

	DMC400_axi4	DMC400_01_axi4	Slave_axi3	slave_ahb	slave_apb3(slave_6)	slave_apb4(slave_6)	slowerAXI3Slave	gpio(slave_11)	umc(slave_11)	ahbDma	encryptEngine1	decryptEngine1
upper_NIC400_axi4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CCI400_axi4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

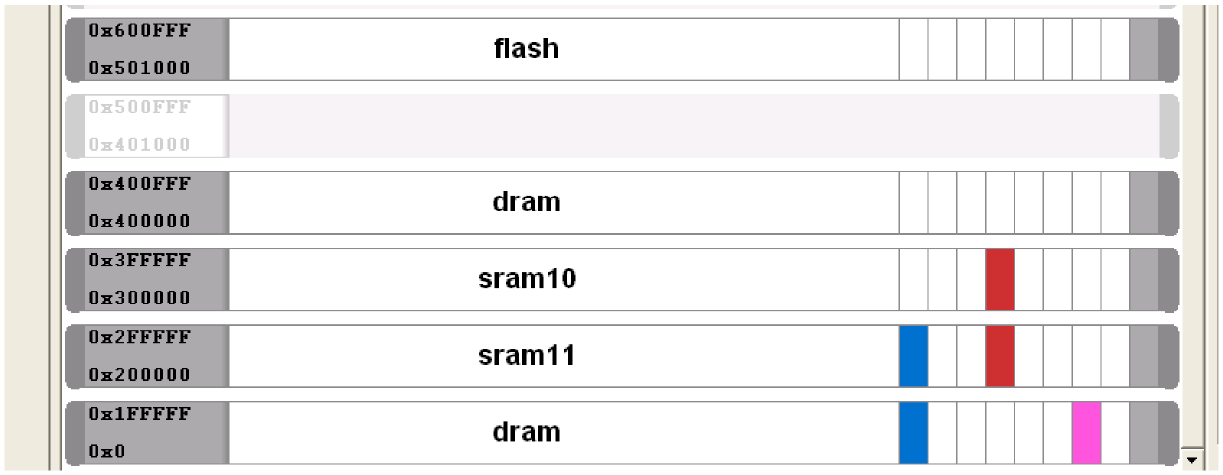
Name	CCI400_axi4
Multi-P	
Protocol	AXI4
Data Wid	128-bit
Clock Do	main

Highlights the protocol used at nodes.

Description
- AXI3 Protocol
- AXI4 Protocol
- AHB-Lite Master Protocol
- AHB-Lite Mirrored Slave Pr...
- AHB-Lite Slave Protocol
- AHB-Lite Mirrored Master ...
- APB Protocol

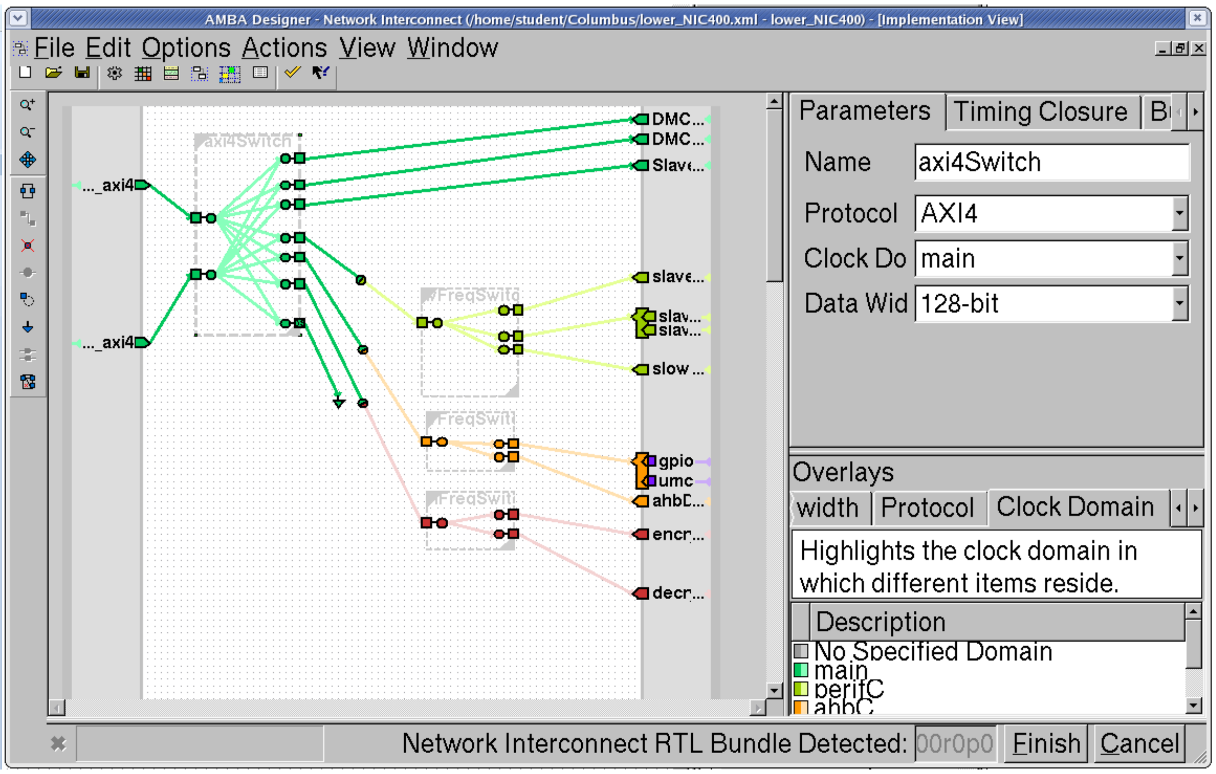
Network Interconnect RTL Bundle Detected: 00r0p0

The designer then sets up the address maps for each master and a global address map as required along with any remap options.



0x600FFF	0x501000	flash
0x500FFF	0x401000	
0x400FFF	0x400000	dram
0x3FFFFF	0x300000	sram10
0x2FFFFF	0x200000	sram11
0x1FFFFF	0x0	dram

The third step is to define the topology of the connectivity between all masters and all slaves in the implementation view. This allows the designer to minimize latency between critical masters and slaves, e.g. between the CPU and main memory, and to group components together to save wires and gates. The GUI then allows the designer to select configuration options for buffer depths, registering options, QoS regulators and other characteristics. The tool will then automatically insatiate the required master/slave interfaces, switch matrices and bridges in RTL and generate an accompanying IP-XACT description.



Name axi4Switch

Protocol AXI4

Clock Do main

Data Wid 128-bit

Highlights the clock domain in which different items reside.

	Description
▫	No Specified Domain
🟩	main
🟩	perifC
🟧	ahbC

Network Interconnect RTL Bundle Detected: 00r0p0

The designer can add Thin Link bridges to bridge between remote masters and slaves and NIC-400 instantiations with fully configurable wire count, bandwidths and latencies.

Using another overlay in the GUI, master-slave connections can be assigned to virtual networks to prevent blocking at arbitration points.

Optimizing the NIC-400

There are multiple ways to optimize an NIC-400 configuration. The one that first comes to mind of course is RTL simulation. This makes a lot of sense since it delivers the accuracy needed to make correct design decisions. There are multiple ways in which RTL simulation can be applied as well, including using UVM compliant verification IP at the various ports to stimulate the design and find corner cases. You can also place the NIC-400 into an RTL model of the system and use this as the stimulus. While each of these is a valid approach, they tend to execute slowly and offer less design flexibility.

Virtual prototypes have been getting increasing visibility in the design space as a means to raise the abstraction level for design. Even when running with 100% accurate models, virtual prototypes typically execute more quickly than their RTL simulation equivalents and are typically faster to assemble and iterate, making them a natural fit for design IP such as the NIC-400 which can have billions of possible configurations. A successful virtual prototype approach typically consists of two phases: optimization and validation.

In the optimization phase, the system is modeled abstractly and quickly to give a quick approximation of the actual system design. Validation then takes the design decisions made in the optimization phase and represents them within the context of an actual system design to ensure that they meet the actual product criteria.

Virtual Prototype Overview

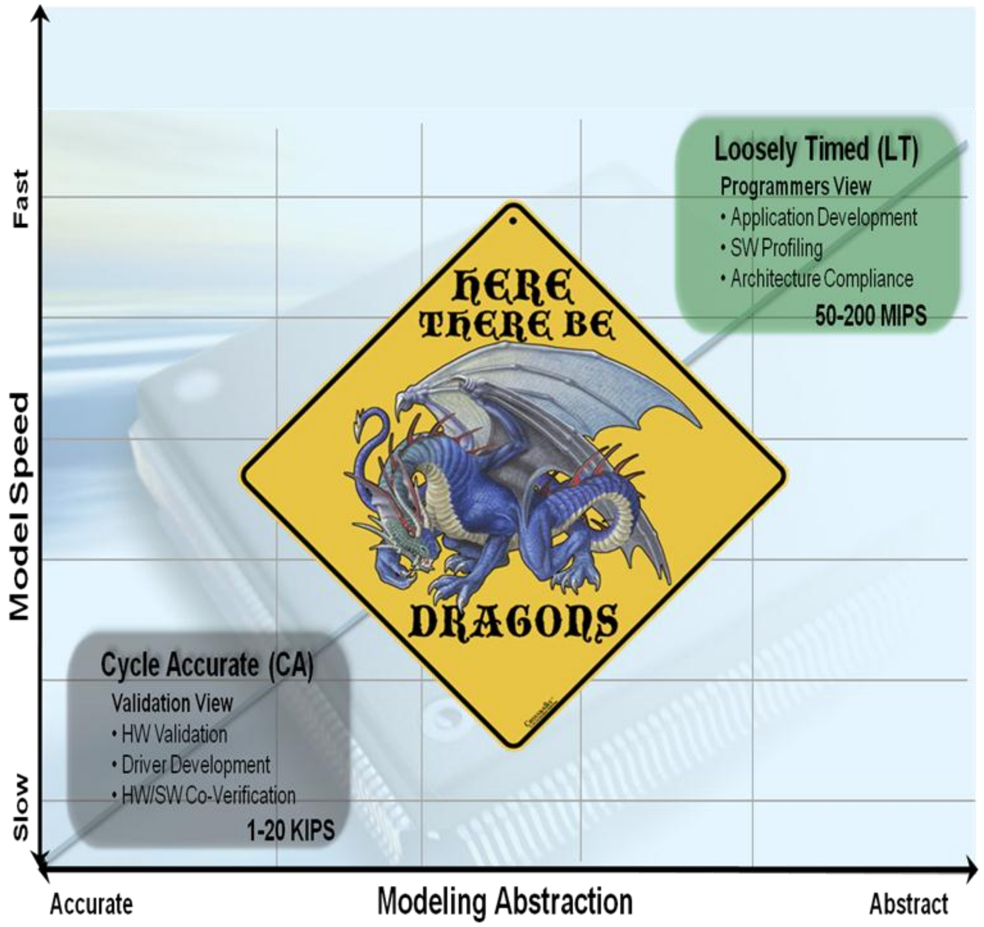
Before we get too in-depth on using a virtual prototype to optimize the NIC-400, let's first look at what a virtual prototype is. Virtual prototypes themselves are a very broad topic so this paper and presentation won't attempt to explain all the possible nuances but it's good to have a background to at least get everyone on the same page.

Virtual prototypes are, as the name would imply, a virtual representation of the system being designed. This representation can be in a variety of forms which change depending upon the problem they are targeted at solving. If we look at a typical SoC design cycle, virtual prototypes can be used at the very beginning of the cycle to make architectural tradeoffs such as hardware/software partitioning and algorithm refinement. As this process is being done, virtual prototypes can also be used to identify the optimal configuration of the IP being incorporated into the SoC design. Since IP typically comes from a variety of internal and external sources, this process can be substantially non-trivial as we'll see later on.

Once constructed, the virtual prototype can be a valuable tool to enable software and firmware design tasks. The internal visibility offered by the system model enables much more effective debugging since it is often possible to see what is happening in both the hardware and software domains simultaneously. How much visibility of course depends upon the accuracy of the models which comprise the system. As you might imagine from the large number of use cases, virtual prototypes can be constructed with varying levels of abstraction depending upon the design needs to be addressed.

Fundamentally, virtual prototype models are based upon the tradeoff between speed and accuracy. In the ideal world, virtual prototypes would execute with 100% accuracy, be available long before silicon and execute just as fast (or even faster than) the real design. Unfortunately however, the world is not an ideal place so tradeoffs are needed in order to meet our design goals and get a virtual prototype up and running quickly. Loosely timed (LT) models are at one end of the spectrum when it comes to tradeoffs. As their name suggests, LT models don't have all of the timing behavior of the actual underlying IP. In fact, they don't need to have any timing behavior at all. By removing the need for this timing information however, LT models can be developed much more quickly and more importantly, they can execute substantially faster than a cycle accurate model. This enables these models to be very valuable for software development tasks since many software developers don't care about timing level details and look far more for the underlying functionality instead. ARM Fast Models are probably the best known example of LT models in the marketplace and ARM makes these models available for most of their processor IP as well as certain peripherals. There is no Fast Model available for the NIC-400 but we'll talk more about how that situation gets handled later.

At the other end of the spectrum are cycle accurate models. These models represent all of the cycle behavior of the underlying IP, indeed for most commercially available cycle accurate models; they represent 100% of both the functional and cycle behavior of the underlying IP. This accuracy makes these models very valuable for architectural optimization and firmware development tasks but they're typically too slow to do much serious software development. You don't want to boot an OS on a completely cycle accurate system unless you have a lot of time on your hands. Carbon's 100% accurate models of ARM IP are probably the best known examples for cycle accurate models.



In the middle of the modeling spectrum between LT and CA models are models known as approximately timed (AT) models. These models are touted as taking a "best of both worlds" approach to modeling. They retain some of the speed of their LT counterparts as well as some of the accuracy of CA models. Typically marketed as coming in at around 80-90% average accuracy. These models can actually be a very valuable part of the model development process if incorporated as part of a high level synthesis (HLS) design flow.

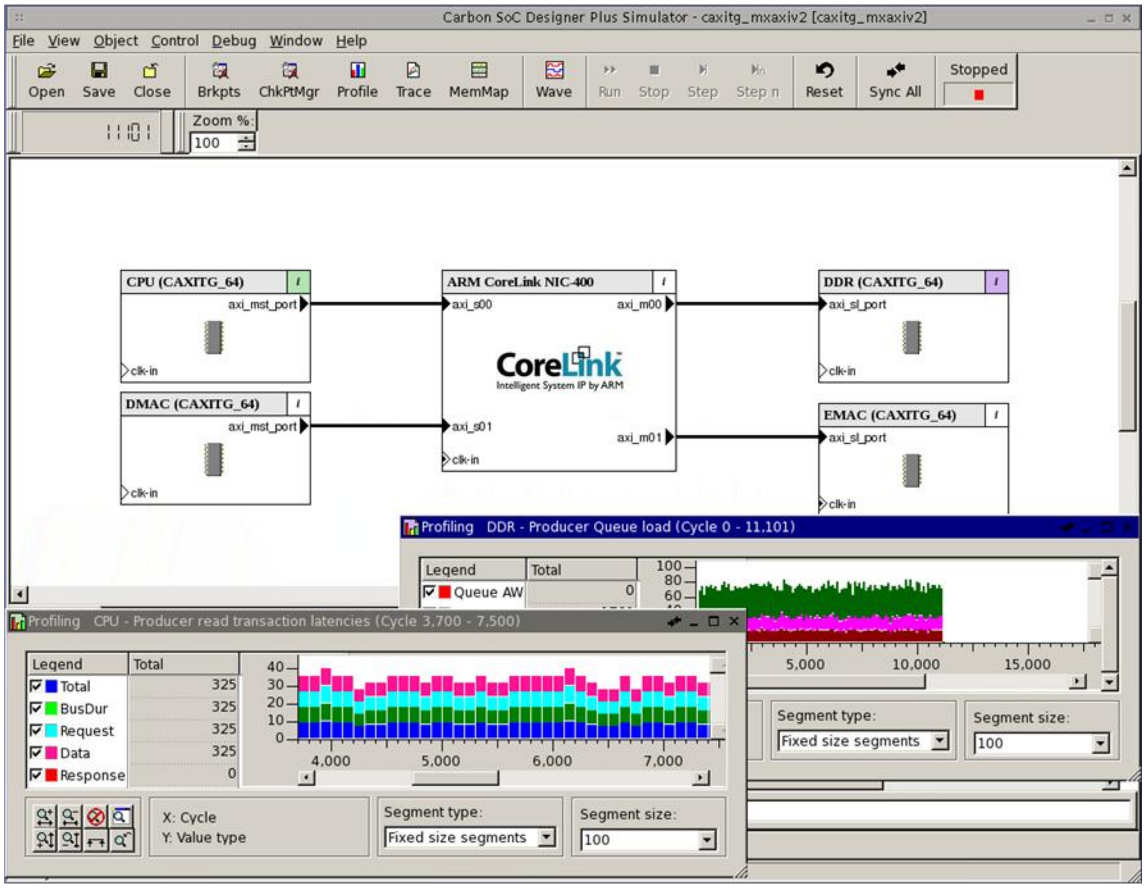
For existing IP, AT models are generally sub-optimal as they don't represent sufficient hardware detail to do low-level design tasks and don't execute quickly enough to enable software development. As the model is improved to better meet the needs of one faction (more accuracy, for example) it typically does so at the expense of the other faction (typically, speed) Combine this with the constant validation work needed to drive the underlying functionality of the model and you have a solution which is very difficult to develop and maintain. AT models used to be quite common for commercial processor IP but are now rarely delivered as IP vendors have gravitated towards a dual model approach with LT and CA models.

As was mentioned earlier, ARM offers Fast Models to meet the commercial demand for ARM LT models. ARM then partners with Carbon Design Systems to offer CA models for ARM IP. Enough of the background discussion though, let's talk about how this relates to the NIC-400.

Modeling the NIC-400

One of the challenges for creating an accurate model of the NIC-400 is managing the configurability of it. Given the possible matrix of connections, connection types, protocols, QoS, etc there are billions of possible unique model possibilities. ARM provides AMBA Designer to manage this configuration task and the model creation path bolts directly on to this flow. Once you've configured the interconnect using AMBA Designer, you can upload the IP-XACT file to Carbon IP Exchange which will then compile the model for you and make it available for download. (Note that authorization is required for this task both from ARM and Carbon to make sure that you have IP and tool access permissions) Once compiled, the model may be managed using the same portal and the IP-XACT file is even stored there in case you want to come back and modify the model later or assign it to a team member.

Once created, the NIC-400 model can easily be executed in tandem with traffic generators to quickly start gathering data. This approach, using traffic generators to both produce and consume traffic is not targeted at verifying the NIC-400 since that is assumed to be correct. Instead, this approach is targeted at measuring the performance characteristics of the model as it is exercised. Traffic can be parameterized to mimic the behavior of system IP and can be parameterized to sweep across a range of various options include burst length, priority, address, etc.


Carbon SoC Designer Plus Simulator - caxitg_mxaxiv2 [caxitg_mxaxiv2]
File View Object Control Debug Window Help
Open Save Close Brkpts ChkPtMgr Profile Trace MemMap Wave Run Stop Step Step n Reset Sync All Stopped
Zoom %
CPU (CAXITG_64)
axi_mst_port
clk-in
DMAC (CAXITG_64)
ARM CoreLink NIC-400
axi_s00
axi_m00
axi_s01
axi_m01
CoreLink
Intelligent System IP by ARM
DDR (CAXITG_64)
axi_sl_port
EMAC (CAXITG_64)
Profiling DDR - Producer Queue load (Cycle 0 - 11,101)
Profiling CPU - Producer read transaction latencies (Cycle 3,700 - 7,500)
X: Cycle
Y: Value type

Legend	Total
☑ 🟥 Queue AW	0

Legend	Total
☑ 🟦 Total	325
☑ 🟩 BusDur	325
☑ 🟦 Request	325
☑ 🟪 Data	325
☑ 🟥 Response	0

Zoom %: 100

Segment type: Fixed size segments

Segment size: 100

In the example shown here, a simple NIC-400 is configured with two masters and two slaves. The masters are set up to mimic the data loads from a CPU and DMA controller and the dummy targets are an Ethernet MAC and a DDR3 memory controller. Of course, since the traffic generators are quite configurable, it's possible to model any number of different sources or targets and we'll get more into that in a bit. Note though that we're analyzing traffic on any of the connections. The graphs shown here track the latency on the CPU interface and the queues in the DDR controller. The exact metrics for the system in question will of course vary based upon design targets however. It's also beneficial to correlate data across multiple analysis windows and indeed even across multiple runs.

The important thing we've done here is establish a framework to begin gathering quantitative data on the performance of the NIC-400 so we can track how well it meets the requirements. The results can be analyzed which will likely lead to reconfiguration, recompilation and re-simulation. It's not unheard to iterate through hundreds of various design possibilities with only slight changes in parameters. It's important to vary the traffic parameters as well as the NIC parameters however since, as we'll see later, the true performance of the NIC-400 and really, all interconnect IP, is how it impacts the behavioral characteristics of the entire system.

Driving and Analyzing Traffic

With the framework in place, we can start refining some of our decision making. The natural first target for this is the memory controller so in the block diagram above we've replaced the traffic consumer model for the DDR3 controller with an actual ARM CoreLink DMC-400 model which is also available from the Carbon IP Exchange website. While this model is not as configurable as the traffic consumer model it will model the actual interaction between the NIC-400 and the memory controller and thus deliver much more meaningful data. This use case, mixing a real interconnect with a real memory controller model is a very typical one for many architectural designs as these two components determine a substantial amount of the theoretical performance in the overall SoC. Through design iterations and changing configuration options in the NIC and DMC, it is typical to see performance improvements in the 25-40% range.

So far we've talked about traffic typically as a parameterizable random source. This gives you broad control over the type and frequency of data it may not closely mimic all of the desired IP characteristics such as responsiveness and transaction spacing. In these cases, it may be desirable to have other approaches for generating traffic. One example that is often used is vector playback. A waveform dump from a simulation of the target IP is used as a means for generating data to be played against the system. This can be a great way to generate representative traffic. Care must be taken when generating the waveforms of course since they must also be representative of how the device will behave in the real system and not just how it responds in an RTL testbench. A common way to get representative waveforms is to use a complete system virtual prototype such as the type which we'll discuss in the next section of the paper and use waveforms from that as the input to the traffic generator. The benefit would be that playing waveforms via a traffic generator may execute substantially faster than simulating a large IP block such as a GPU.

Finally, if more responsiveness and control are desired, traffic generators can be set up which generate very targeted traffic. The example here is from a Carbon traffic generator configured to simulate the behavior of a video processing block. Data patterns are modeled using XML and the controlled programmatically. This gives the use a lot more control over the data being played back but obviously there's a bit more assembly required if your desired traffic pattern isn't already modeled.

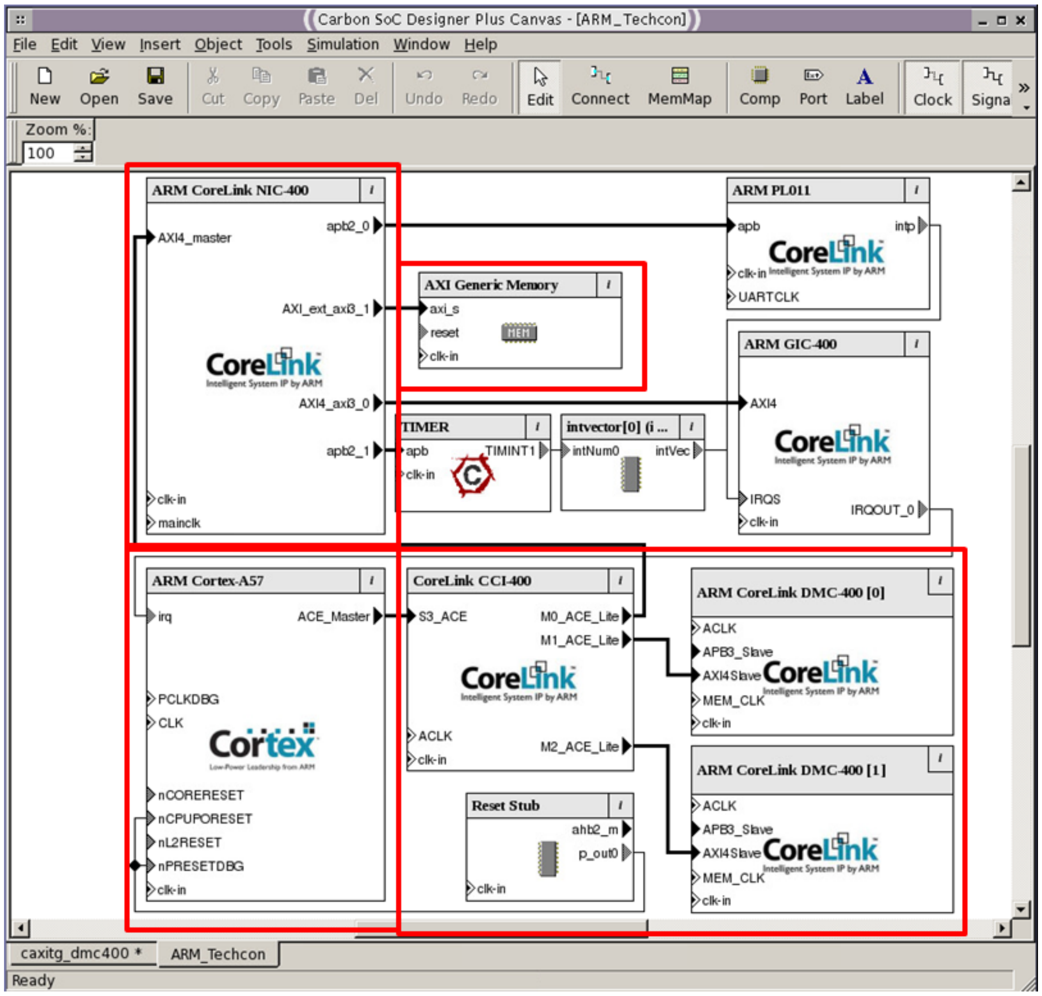
When you're examining hundreds, even thousands of potential design choices, the amount of data can prove to be overwhelming. Large number of potential systems can be eliminated using simple elimination algorithms (only look at design results where max latency on a particular connection is less than X for example). Once narrowed down, it's important to have a means to compare data sets against each other to identify the best configuration. In the example above, we're looking at the AXI latency between a processor and cache controller. The two simulations are identical except the number of DDR wait states is 1 in the top simulation and 3 in the other. This is for memory being accessed through the NIC-400. There are two interesting things to point out. As would be expected, the increase in wait states caused a greater average latency but, perhaps counter-intuitively it also consumed fewer simulation cycles to execute the overall program. This is a single datapoint but is shows how looking at IP configuration is a system level issue.

As we've seen, the traffic generator approach can do a great job at getting fast, accurate results even when there is nothing more than just the NIC-400 and a few sources and consumers. It can be set up quickly and enables easy testing of a wide range of system possibilities including corner cases which might be difficult to set up with real IP. It does have drawbacks however. Ultimately, no matter how much time you spend assembling your traffic generation schemes, they don’t reflect the actual behavior of the system running real software and this behavior can vary greatly depending upon the system software and IP configuration. Even a slight reordering of system software calls can have a big impact on overall system performance. Therefore, although it’s great to use traffic generators to get a good approximation of the system performance. The best way to validate that the system will actually meet your performance targets is to actually assemble the system. Thankfully, this is also possible using virtual prototypes.

Validating the NIC-400

A system level virtual prototype gives you a much more realistic view of what’s going on inside the system. This is obviously extremely important to handle cases which your traffic generator may not model as correctly as the real IP such as ordering, arbitration and number of outstanding transactions. Another item which is extremely important is coherency. While most of the coherent traffic in the system will be handled by the CCI/CCN IP it will still have an impact on system performance and a few software calls can greatly impact the system traffic in order to maintain this coherency.

Fundamentally, software can cause a dramatic impact on the performance of the overall system and getting this software up and running with the real hardware will enable both of these to be optimized in advance of actual silicon. A prime example of this is with system level benchmarks. These benchmarks are often used to market the IP once it has been finalized. Leading edge design teams will use these benchmarks during the design process of the SoC to drive traffic in the system but also to tweak the settings of the IP to maximize performance. This helps ensure that the actual silicon will meet the marketing specifications.



This is an example system level virtual prototype. In this case, the ARM Cortex-A57 is featured as the main processor in the system. There's a CCI-400 and multiple DMC-400s to handle most memory accesses and the NIC-400 hangs off the CCI to manage memory accesses to the rest of the system. This system is fully capable of booting Linux and then running a variety of system level benchmarks. Obviously we'd need to add a few more components, most notably a GPU, to model the capabilities of a leading edge SoC but even this simple system model is sufficient to optimize the performance of compute oriented benchmarks and maximize the performance of the processor/memory subsystem. We're using a parameterizable memory here on the NIC-400 to give some system level flexibility on additional components. Most systems will use multiple memories, one for each modeled component or, if desired, the actual IP models can be used. Since it can boot an OS however, this system level virtual prototype is valuable not only to validate the performance of the various system component but it can also be used to enable pre-silicon firmware development.

Simplifying the Task

Building a system level prototype such as this is a non-trivial task however. Each IP block needs to be configured, set up with the appropriate memory maps, tied together and placed into a valid system configuration. Then there is software which must be written to drive the whole system and configure each piece of IP. This is a substantial design task and all of these steps need to be done before any architectural value can be extracted. In order to simplify this process, ARM and Carbon have worked together to offer a variety of pre-built systems featuring ARM IP components. Called Carbon Performance Analysis Kits, each of these pre-built systems comes complete with IP models, a system to tie them all together and the software necessary to configure the IP and drive the system. While these sample system may not model 100% of the behavior of your actual system in their off-the-shelf form, they're provided with complete source code and are easily modifiable to quickly match your actual system configuration. IP blocks can be easily tweaked or replaced to more closely match the components in the SoC being designed.

Once assembled, the system level virtual prototype can of course give you all of the same architectural analysis characteristics which were available in the traffic generator driven system. You can still track the important throughput and latency numbers for the system. By tying in real software however you now have the capability to analyze the hardware capabilities of the system concurrently with the software which is driving it. In this diagram you'll see some of the profiling data available when executing software in the system level virtual prototype. This data is synchronized with the hardware analysis view and gives the designer the capability of quickly understanding the impact of software routines on the underlying hardware.

Fast or Accurate, Why Not Both?

As we discussed at the start of the optimization section, the fundamental weakness of an accurate virtual prototype is speed. The results may be accurate but, let's be honest; no software engineer is going to do any serious development on a prototype which runs too slowly, regardless of how accurate it is. At the 2012 ARM Techcon, ARM and Carbon did a joint presentation on a way to address this problem (ATC-100 High Performance or Cycle Accuracy? You Can Have Both!) It's basically a technology which enables a system level virtual prototype to execute using high speed ARM Fast Models and then swap over to a 100% accurate Carbon model representation at any breakpoint. This enables software development to be done at Fast Model speeds while still enabling accurate debug and analysis.

While the NIC-400 does not have a Fast Model available directly from ARM, a Fast Model can be generated directly from the Carbonized RTL model. This means that NIC-400 based systems can be used with this Swap & Play technology to boot an OS in seconds and then debug software issues or execute performance analysis tasks with 100% accuracy.

Conclusion

The ARM CoreLink NIC-400 raises the bar on system level performance while also providing the mechanisms to reduce area and power consumption. These new features, coupled with the vast configurability of any interconnect block, provide a vast array of design choices which can be made as the IP is incorporated into an SoC design. A two-step approach using accurate virtual prototypes can give designers confidence in their ability to make design decisions using approximated traffic models and then validate their results by running actual system software.

Authors

William Orme, Strategic Marketing Manager for System IP, Processor Division, ARM is responsible for the CoreLink NIC-400 and the next generation on-chip interconnect. At ARM since 1996 he has lead the introduction of many new products, including the ETM and subsequent CoreSight multi-core debug and trace products. Prior to joining ARM, William spent 12 years designing embedded systems from financial dealing rooms, through industrial automation to smartcard systems. William holds degrees in electronics and computer science as well as an MBA.

Bill Neifert, CTO and Founder, Carbon Design Systems has more than 20 years of EDA and design experience. After starting his career as an ASIC design and verification engineer at Bull HN/Zenith Data Systems, Bill held various applications engineering and AE management roles at Quickturn. He has a BSCE and an MSCE from Boston University.