ARM TECHCON 2011

Performance Analysis of Cortex-A15 and AMBA4

ATC-120

Joe Tatham VP Engineering Carbon Design Systems

9/19/2011

Overview

The Cortex-A15 and related family of protocols and IP present both an opportunity and a challenge for the system architect. The opportunity presented is to create systems which are able to execute both single and multi-threaded workloads efficiently, and provide hardware-based support for virtualization and coherency. The challenge presented is to understand the protocols and IP enough that the system is not over or under designed, so that power and area constraints can be met. In order to meet the challenge and take advantage of the opportunity, virtual platforms can be utilized to quickly construct and analyze candidate system configurations.

Motivation

Along with the Cortex-A15, ARM is introducing a family of protocols and related IP. The protocol family is AMBA4, which extends AXI and APB, introduces the AXI4-lite and AXI4-stream protocols, and introduces ACE coherency extensions. The related IP includes components such as the CCI-400 coherent interconnect, the MMU-400 memory management unit, the GIC-400 interrupt controller, the NIC-400 network interconnect, along with others. As system architects start designing next generation products, they will be making decisions on which pieces of new IP to use, how to configure the IP, and how the pieces fit together into a system to achieve the desired product goals. The architects will need to experiment with many system configurations, running many different benchmark loads on the configurations, to determine the optimal configuration.

Figure 1 shows an abstracted system containing multiple coherent masters, a coherent memory, and non-coherent slaves. Even though this system contains only a few pieces of the new pieces of ARM IP being introduced with the Cortex-A15, it demonstrates how complex these new systems can be.

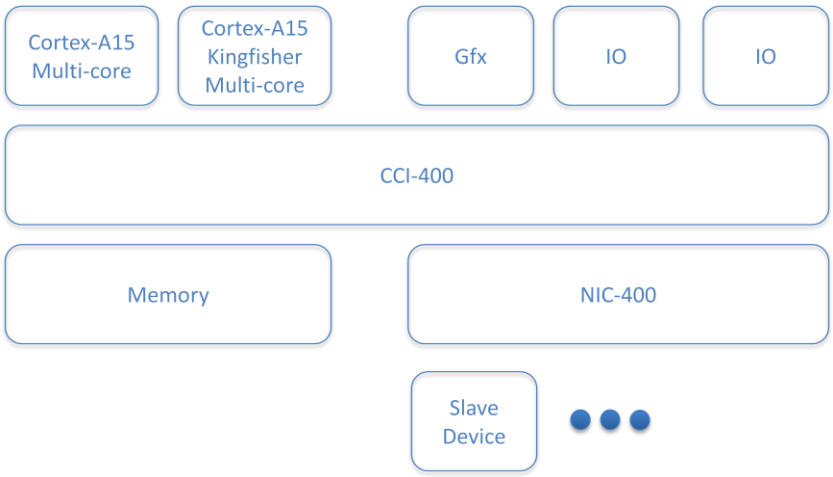

Figure 1 – Multiple coherent master system

AXI4 and ACE

The AXI4 protocol extends the AXI3 protocol by increasing the INCR burst length from 16 to 256 beats, and adds additional User, Region, and QoS signaling. The AXI4 protocol removes write interleaving and locked transactions.

ACE Transactions

The ACE protocol is a set of extensions for providing hardware-based coherency, and adds new transactions. The transactions apply to a domain, where the possible domains are:

- Non-shareable domain contains one master
- Inner-shareable domain contains a set of masters
- Outer-shareable domain contains the set of inner-shareable masters plus additional masters
- System domain contains all masters in the system

There are two types of barrier transactions: memory barrier, which ensures that transactions complete in-order, and synchronization barrier, which ensures that all outstanding transactions are complete. The cache maintenance transactions enable making cache operations visible to other masters. The distributed virtual memory (DVM) transactions provide support for sharing page tables. The snoop transactions support shareable read and write transactions.

ACE-Lite

The ACE-lite protocol is a subset of the ACE protocol, where a component implementing an ACE-lite port can snoop other masters, but cannot itself be snooped. So, the ACE protocol would be used for a processor, and the ACE-lite protocol would be used for a graphics processor or IO device, as examples.

Cortex-A15 and CCI-400

The Cortex-A15 contains up to four cores, an integrated L2 cache, and support for large physical addressing, among other features. The Cortex-A15 also supports coherent clustering of multiple Cortex-A15 masters through the use of the ACE protocol and a coherent interconnect, as shown in Figure 1.

The CCI-400 is the Cache Coherent Interconnect from ARM which supports hardware-based coherency by providing snooping for masters and slaves. The CCI-400 supports two ACE masters, three ACE-lite with DVM masters, and three ACE-lite slaves, see Figure 2.

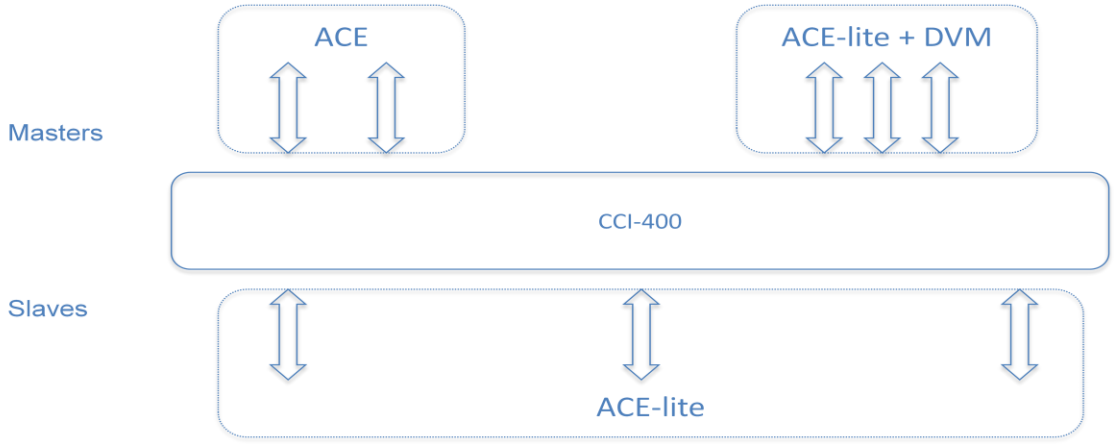

Figure 2 – CCI-400 connectivity

Cortex-A15 ACE Configurability

The Cortex-A15 ACE contains an ACE master port for the system bus. This port has two types of configurations: width and coherency mode. The width can either be 64 or 128 bit. The coherency modes supported are:

- AXI3 mode
- ACE non-coherent, with or without L3 cache
- ACE outer coherent
- ACE inner coherent

AXI3 Mode

Figure 3 shows a logical view of the original multiple coherent master system (from Figure 1) when the Cortex-A15 is in AXI3 mode. Since AXI3 does not have hardware coherency support, only one master is shown, although a multiple-master system could still be implemented using software coherency. The Cortex-A15 will only generate AXI3 traffic in this mode. Note that the interconnects have been removed for clarity.

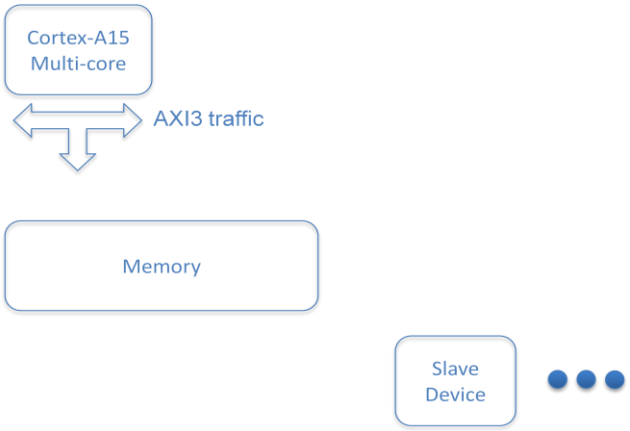

Figure 3 - AXI3 logical view

ACE Non-coherent Mode

Figure 4 shows the original system when the Cortex-A15 is in AXI4 non-coherent mode. In this mode, there are no other coherent masters on the ACE interface, and the Cortex-A15 can generate barrier transactions and also cache maintenance transactions.

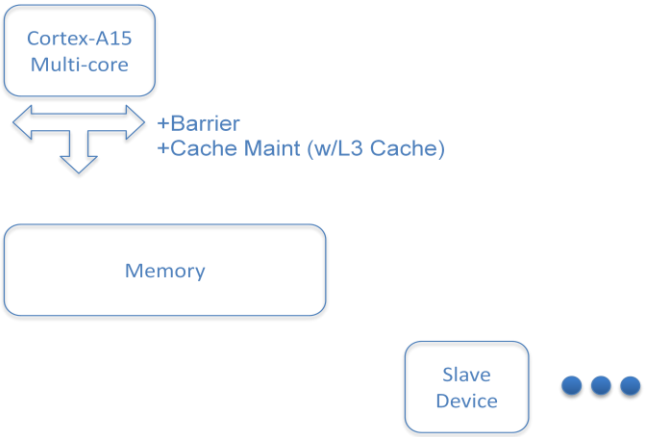

Figure 4 - ACE non-coherent logical view

ACE Outer Coherent Mode

Figure 5 shows the original system when the Cortex-A15 is in AXI4 outer coherent mode. In this mode, there are other coherent masters on the ACE interface, and the Cortex-A15 can generate snoop transactions in addition to barrier transactions and cache maintenance transactions. Shared memory in the outer shared domain is supported.

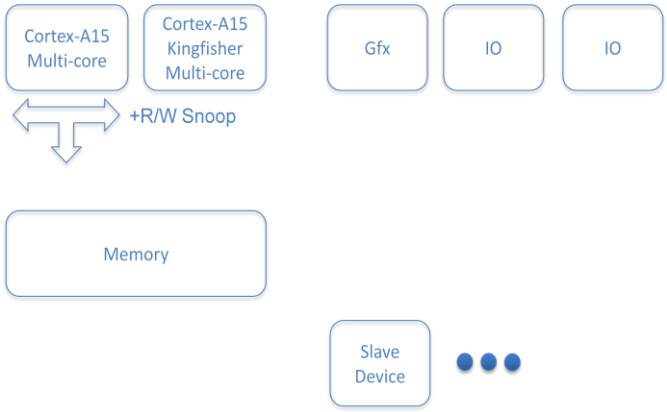

Figure 5 - ACE outer coherent logical view

ACE Inner Coherent Mode

Figure 6 shows the original system when the Cortex-A15 is in AXI4 inner coherent mode. This mode is similar to the outer coherent mode, but it adds support for shared memory in the inner as well as outer domain, and distributed virtual memory transactions can be generated.

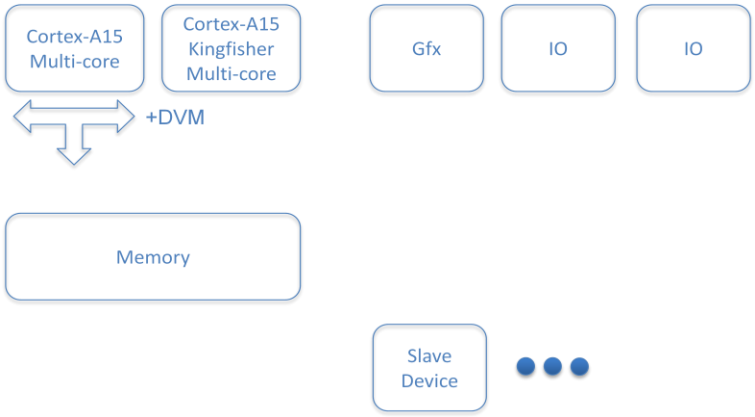

Figure 6 - ACE inner coherent logical view

Architectural Analysis

With the wide configurability of the Cortex-A15 and related IP, and the availability of new hardware coherency mechanisms, the architect has many decisions to make about which pieces of IP to use and in what configurations. Questions such as the following need to be answered to choose IP:

- What is the impact on single-thread performance with the use of hardware-based coherency?
- What is the impact on multi-thread performance with hardware vs. software coherency?
- In addition to performance tradeoffs, what are the power and area costs?

Once IP is chosen, the IP needs to be configured, which means choices such as:

- FIFO depths
- Interconnect widths

- Cache sizes

The best way to answer these selection and configuration questions is to implement the various systems under consideration, run real workloads through the systems, and analyze the results. This is possible through the use of virtual platforms.

Virtual Platforms

A virtual platform provides a simulation of a hardware system using models of the system components. The construction, simulation, and analysis features of a virtual platform provide the framework for making tradeoffs among different system implementations. A virtual platform also provides models of common IP and a mechanism to model proprietary IP. Figure 7 shows the usage flow of a virtual platform when performing architectural analysis.

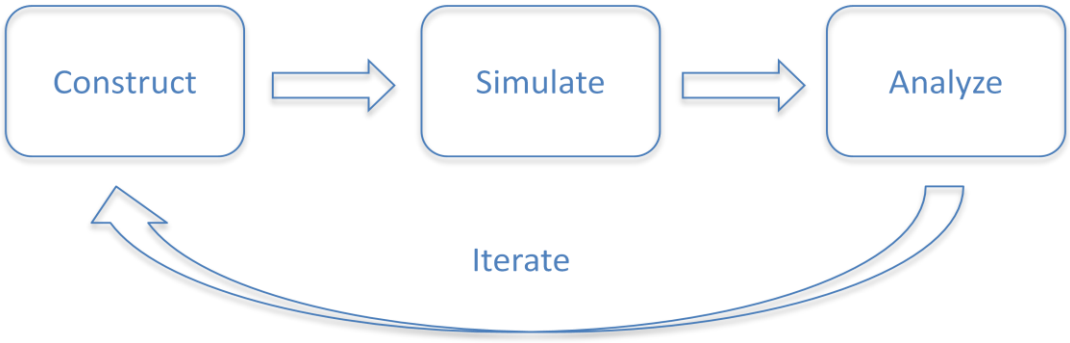

Figure 7 - virtual platform flow

Accuracy

Since a virtual platform is a simulation of hardware, the simulation can occur at different levels of abstraction with corresponding different degrees of timing accuracy. For some purposes, such as booting an operating system, an ISS (instruction set simulator) level of accuracy is sufficient. For other purposes, such as architectural analysis which requires accurate latency and throughput profiling, cycle accuracy is required. Figure 8 shows the levels of abstraction available.

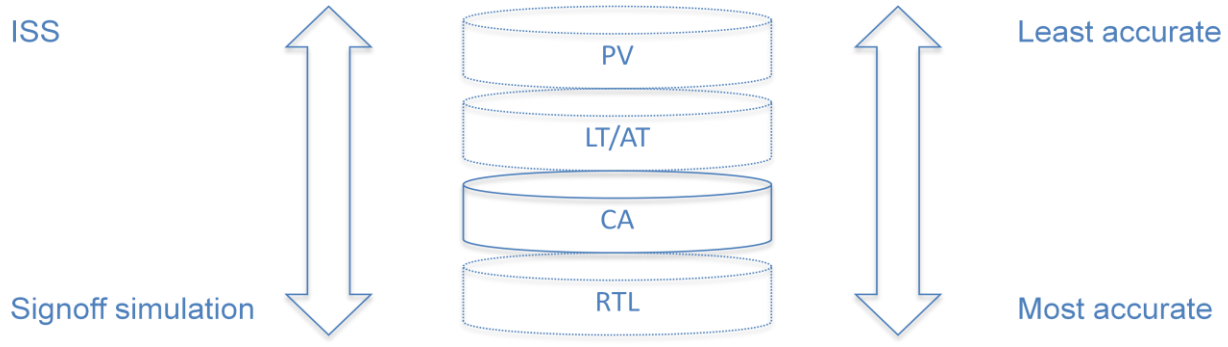

Figure 8 - virtual platform abstraction levels

PV

A model at the PV (programmer's view) level is behaviorally accurate with regard to transaction or instruction sequences, but is not intended to be accurate with time. An example is an ISS, where the model will execute instructions resulting in accurate register and system level behavior, but timing results due to pipeline effects are abstracted away; most ISS's will retire one instruction per cycle.

LT/AT

A model at the LT (loosely-timed) or AT (approximately-timed) level is also behaviorally accurate, and has some notion of being *x*% accurate with time. An example is a time-annotated ISS, where there may be some modeling of the latency of certain instructions (an ADD is one cycle, a MUL is four cycles), but detailed pipeline behavior is not modeled (two sequential ADD instructions without hazard may be retired in one cycle).

CA

A model at the CA (cycle-accurate) level is behaviorally accurate, and is also 100% cycle accurate. A cycle-accurate model of a CPU will retire instructions in the same cycle as the actual hardware, since the pipeline is fully modeled. This level of modeling is ideal for architectural analysis work, where the change to the depth of a FIFO, for example, will be accurately modeled.

RTL

An RTL model running under an event-based simulator is signoff accurate, and is appropriate for hardware validation teams to test an implementation. This level of accuracy is overkill for architectural analysis work.

Visibility

In order to analyze the performance of a system configuration with a given workload, the virtual platform must provide comprehensive visibility into all aspects of the system behavior. The platform must be able to correlate software instructions to bus transactions to signal waveform traces, as in Figure 9.

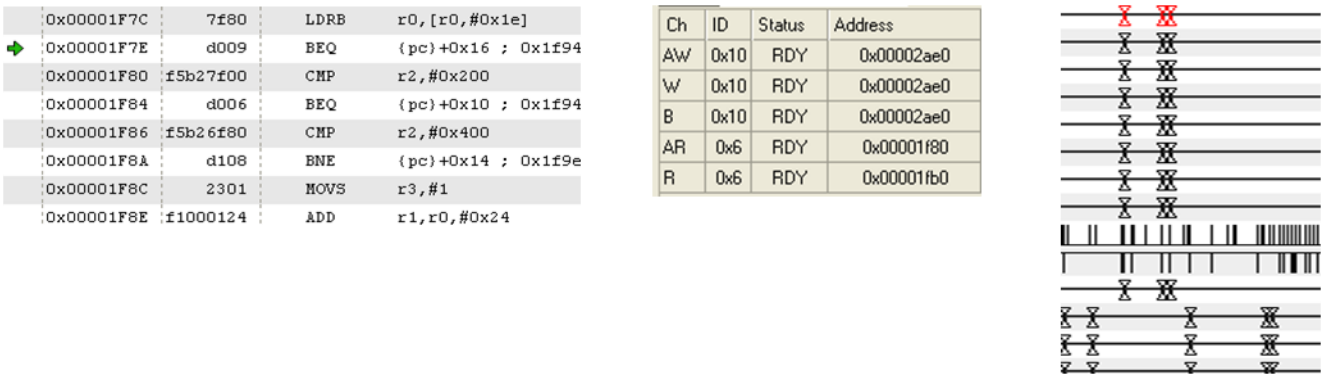

Figure 9 - virtual platform visibility

This correlation allows the user to understand the root cause of behavior. As an example, if a system delivers lower than expected throughput for a workload, what is the cause and what is the fix?

- FIFO depths need to be increased?
- Interconnect widths need to be increased?
- Software needs to be optimized for cache locality?

Iteration

After virtual platform execution results are analyzed, modifications can be made to the virtual system, and the simulation and analysis can be repeated to narrow in on an optimal system design. In addition to an iterative flow, a parallel flow can also be employed where many different system configurations are simulated at the same time. A parallel flow is especially useful to start the architectural analysis process, when there are a number of early candidate configurations that need to be weighed off against each other.

Summary

A system architect utilizing the Cortex-A15 and related IP from ARM has a large number of options available for IP selection and configuration. In order to design a product that meets the needs of the market place in terms of power, performance, and area, the architect must have the tools to run workloads across various different system realizations. A virtual platform is the tool that allows the architect to quickly assemble many configurations, simulate workloads, and analyze the results so that optimal design decisions can be made.