2014 ARM® TechCon™

UBM Tech

Performance Optimization for an ARM Cortex-A53 System Using Software Workloads and Cycle Accurate Models

Jason Andrews

carbon design systems
The Trusted Path to *Accuracy*

ARM® 2014 TechCon™

Agenda

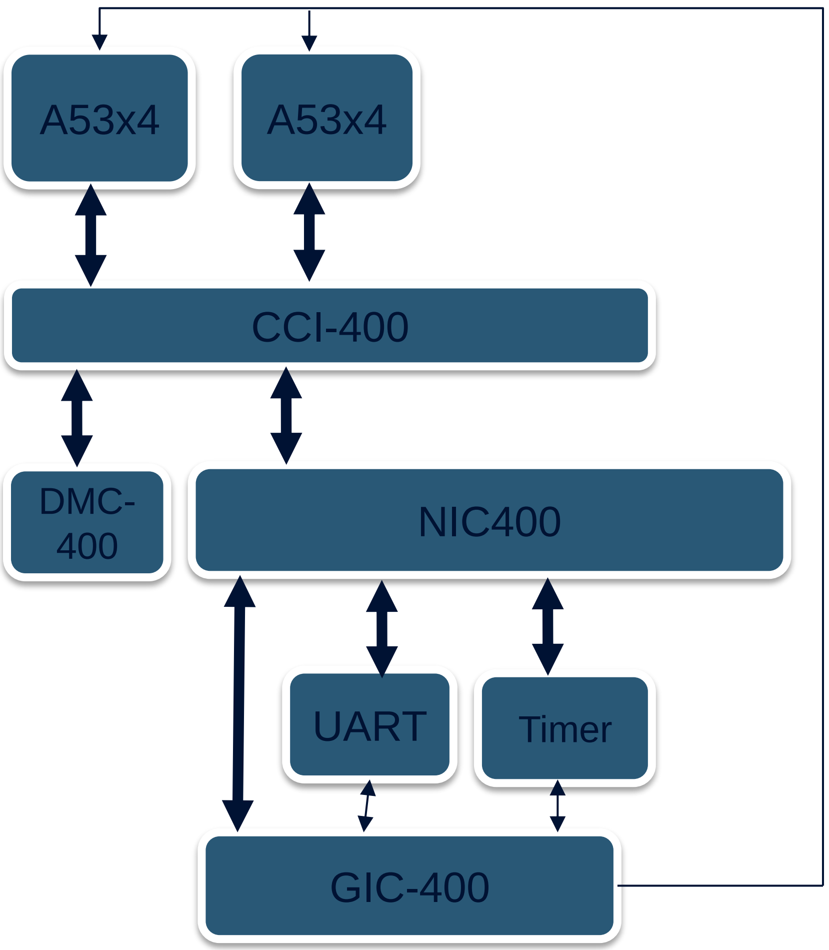
- System Performance Analysis
- IP Configuration
- System Creation
- Methodology: Create, Validate, Analyze
- System Level Optimization
  - Bare Metal Software
  - Linux Application Benchmarks

System Performance Analysis

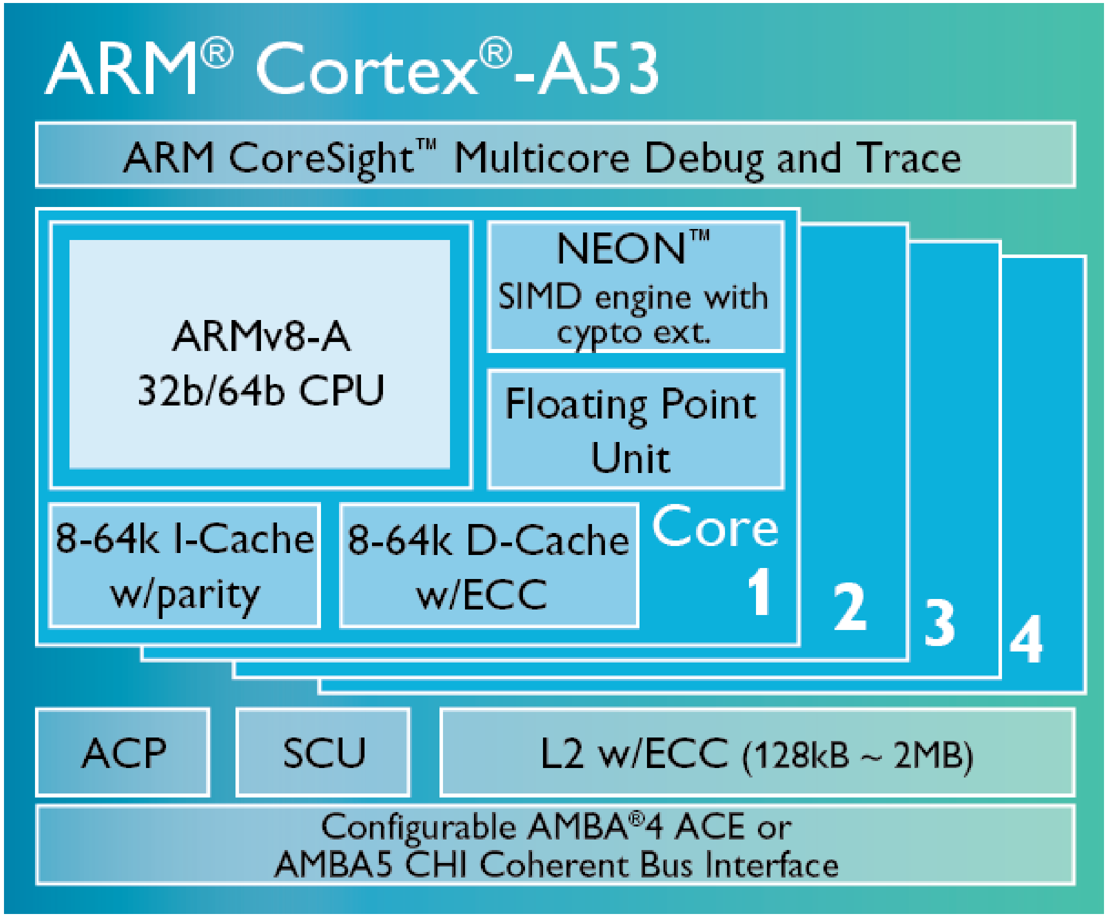
- Selecting and configuring IP for use in systems is difficult
- ***System Performance Analysis:*** the ability to create, validate, and analyze the combination of hardware and software
- Requirements
  - Cycle accurate simulation
  - Access to models of candidate IP
  - Easy way to create multiple designs and quickly change IP configurations
  - Capacity to run realistic software workloads
  - Analysis tools to make optimization decisions based on simulation results

Example System Components

- **Platform Components**
  - Multi-Cluster ARM Cortex-A53
  - Coherent Interconnect
  - Interrupt Controller
  - Timer & UART
- High performance DMC-400 DDR3

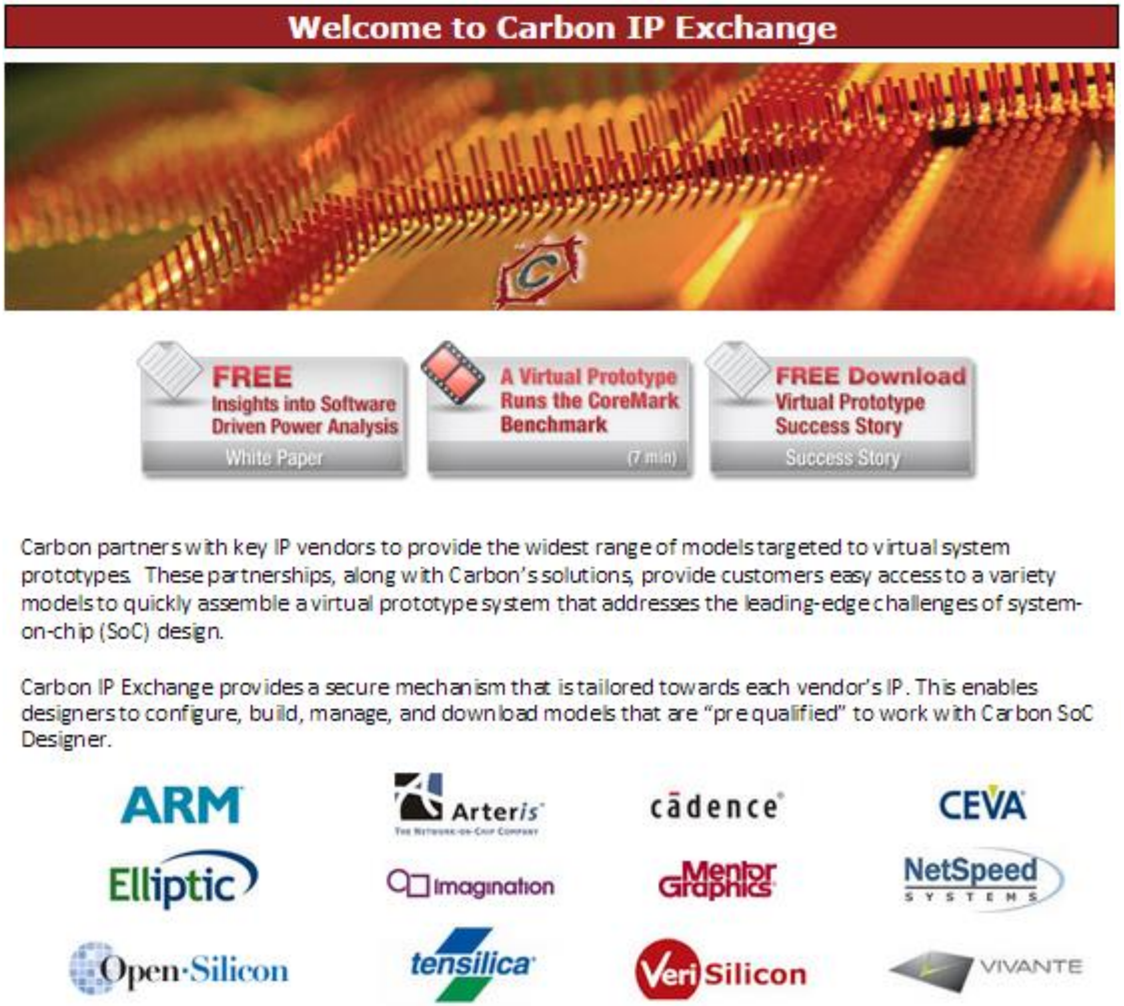


Cortex-A53



- Power Efficient ARMv8 processor
- Supports 32-bit and 64-bit code
- 1-4 SMP within processor cluster
- NEON™ Advanced SMD
- VFPv4 Floating Point

IP Model Creation


Welcome to Carbon IP Exchange

Carbon partners with key IP vendors to provide the widest range of models targeted to virtual system prototypes. These partnerships, along with Carbon's solutions, provide customers easy access to a variety models to quickly assemble a virtual prototype system that addresses the leading-edge challenges of system-on-chip (SoC) design.

Carbon IP Exchange provides a secure mechanism that is tailored towards each vendor's IP. This enables designers to configure, build, manage, and download models that are "pre qualified" to work with Carbon SoC Designer.

- Accurate models from leading IP providers
- Compile, manage and download 100% accurate models
- Only source for 100% accurate virtual models of ARM IP

Cortex-A53 Configuration

IP Configuration

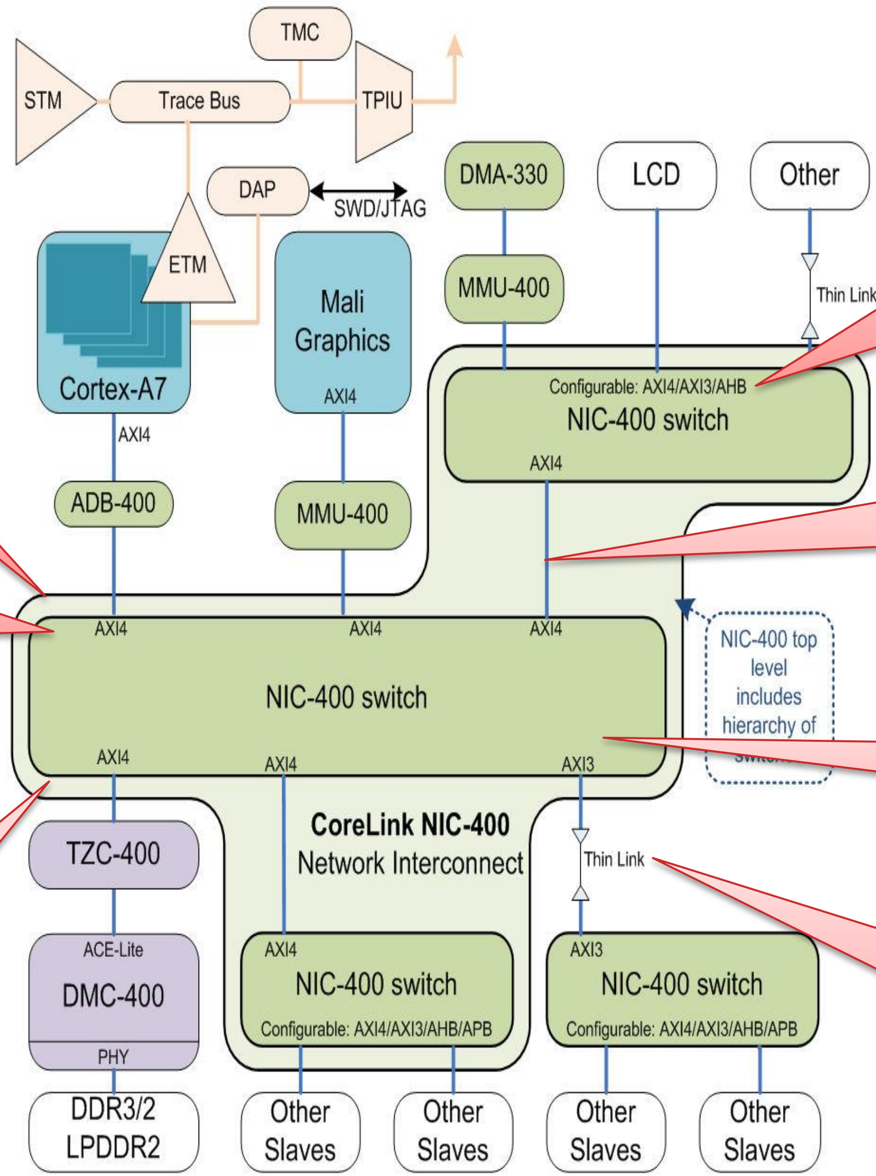
SELECT NUMBER OF CPUS	4	*Number of logical CPUs to include*
NEON FP	TRUE	*Include the NEON and Floating-Point unit in each CPU*
CRYPTOGRAPHY EXTENSION	TRUE	*Include the Crypto extensions in the NEON and Floating-Point unit in each CPU*
EXTERNAL MEMORY INTERFACE SUPPORT.	ACE	*Select AXI3 or ACE for the main bus interface*
L1 INSTRUCTION CACHE SIZE	64kB	*Select an L1 Instruction Cache Size (8kB | 16kB | 32kB | 64kB)*
L1 DATA CACHE SIZE	64kB	*Select an L1 Data Cache Size (8kB | 16kB | 32kB | 64kB)*
L2_CACHE	TRUE	*Include L2 Cache*
ACP	TRUE	*Include an ACP interface on the SCU*
L2 SIZE	2048kB	*Select an L2 Cache Size (128kB | 256kB | 512kB | 1024kB | 2048kB)*
L2_INPUT_LATENCY	1	*L2 Data RAMs Input Latency*

CoreLink NIC-400 Network Interconnect

Graphical UI for configuration in AMBA Designer

Choose topology to minimize CPU latency and routing congestion

Set multiple clock domains for best performance and power saving



Select protocol for each master/slave. Bridges inserted automatically

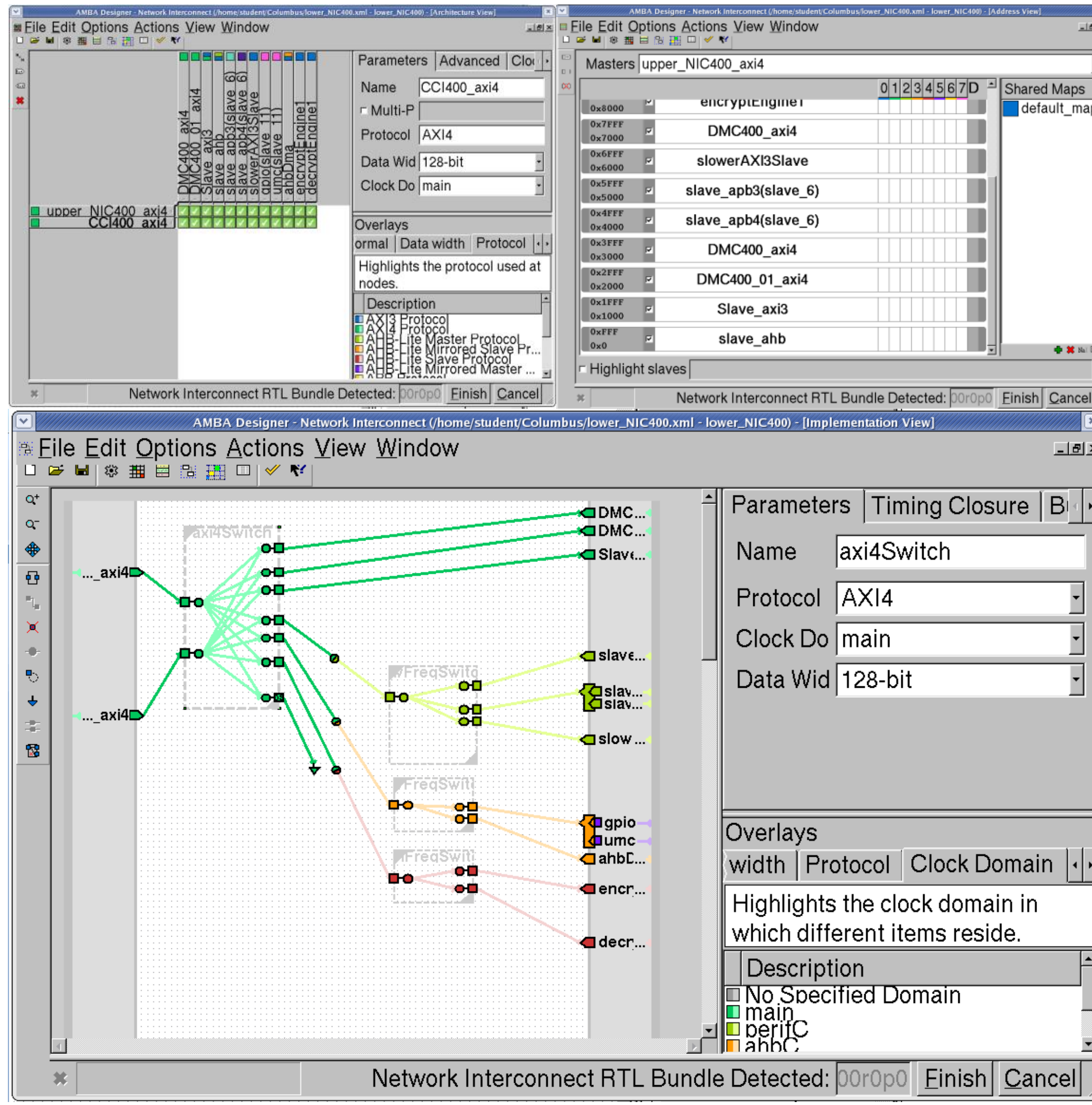
Select data widths 32 to 256-bit and buffer depths for required bandwidth

Registering options for fast timing closure

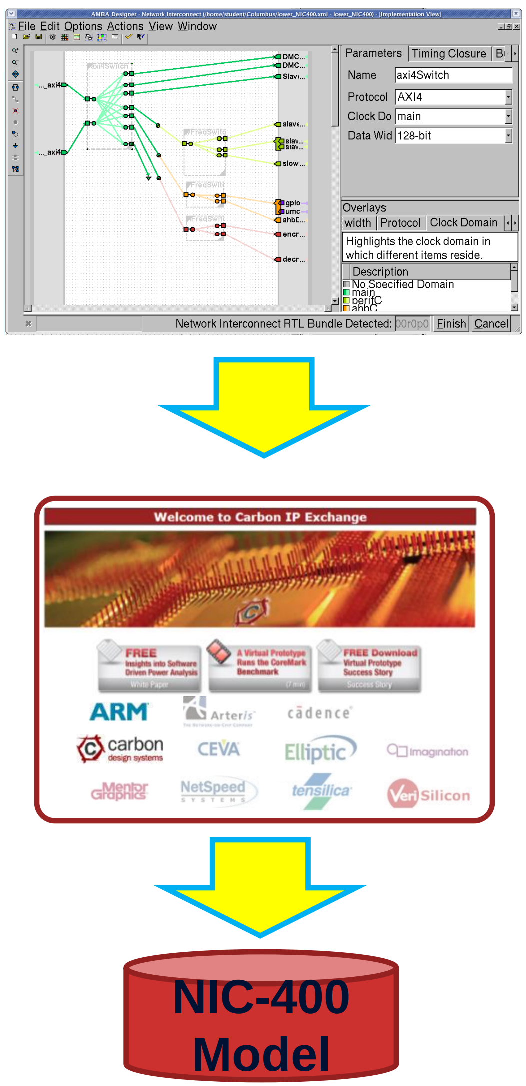
Configure Thin Links between NIC-400s to reduce routing and ease timing closure

AMBA Designer Configures the Interconnect

- Architecture View
  - Define masters slaves and connectivity
- Address Map View
  - Set multiple memory maps
- Architectural View
  - Design interconnect structure and features
  - Switch hierarchy
  - Widths
  - Clock domains
  - Buffer depths
  - Registering options

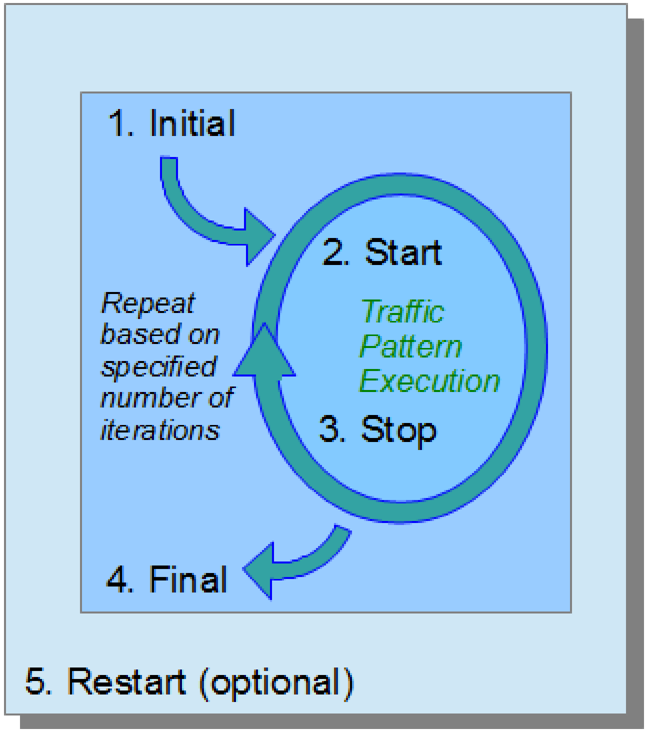


Creating the Accurate Model



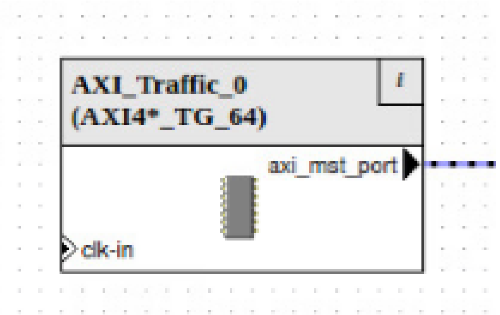
- Upload CoreLink AMBA Designer IP-XACT file to IP Exchange web portal
- 100% accurate model created automatically from ARM RTL
- Download link provided via email

AXI4 and ACE Traffic Generation

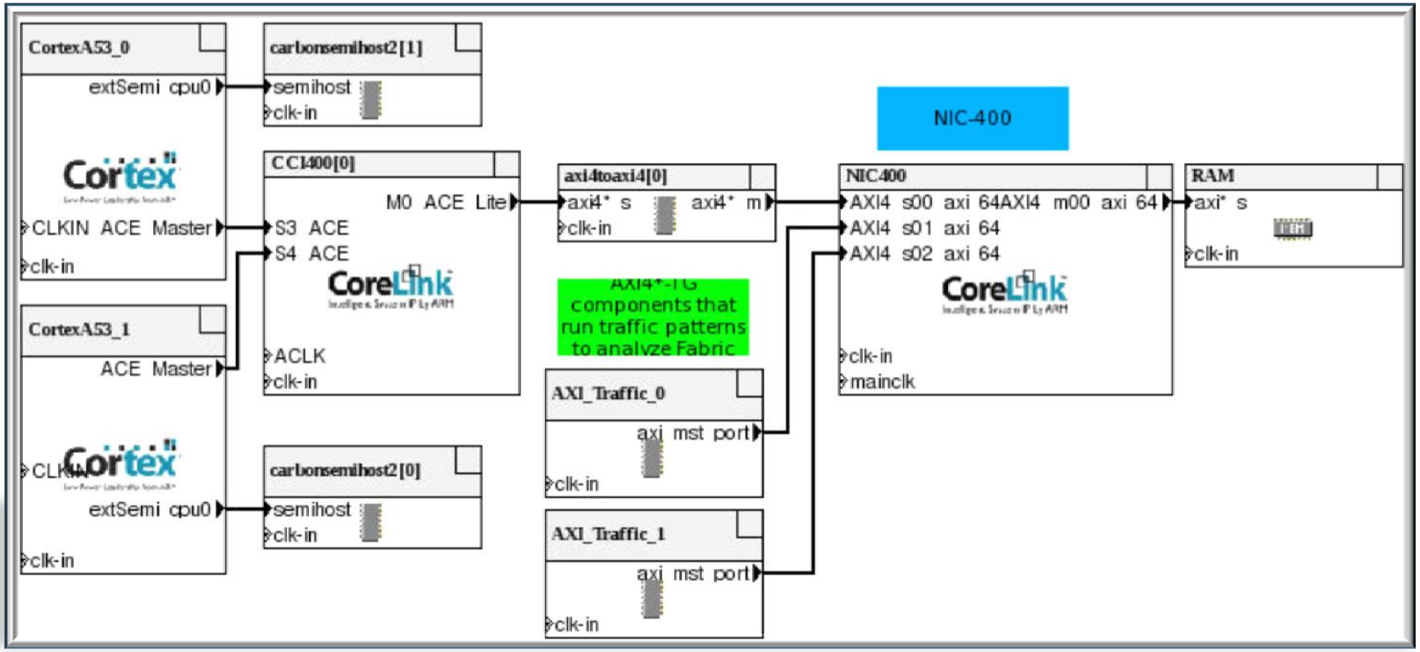


- Initial (wait/send events)
  - Start (wait/send events)
    - Iterations of execution of traffic pattern
    - Duration can be time or quantity of traffic
  - Stop (wait/send events)
- Final (wait/send events) (restart optional)

Used to Model Additional Bus Agents

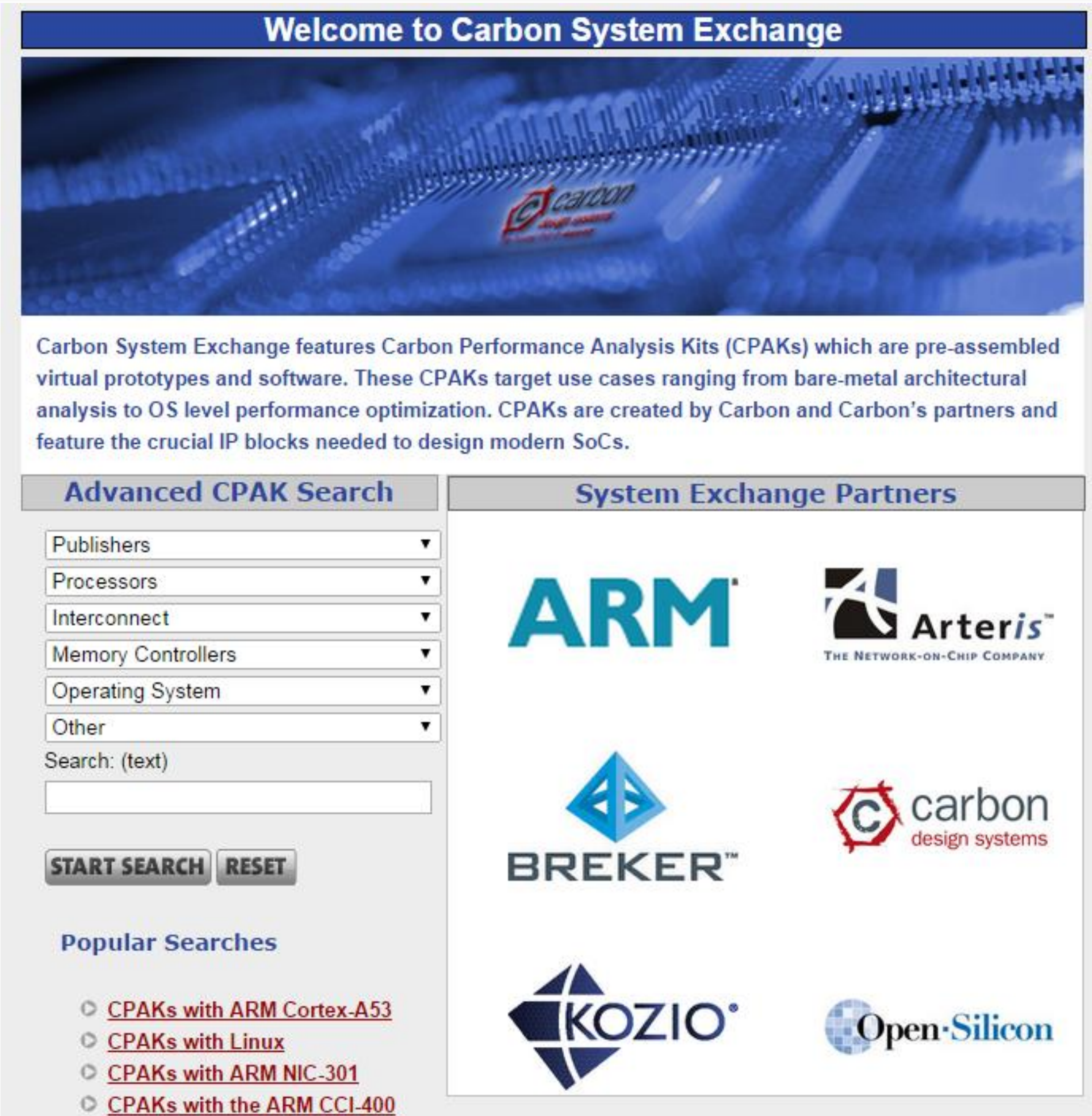


Carbon Performance Analysis Kits



- Pre-built, extensible virtual prototypes
  - ARM® Cortex™-A57, Cortex-A53, Cortex-A15, Cortex-A9, Cortex-A7
- Reconfigurable memory and fabric
  - NIC-400, NIC-301, CCI-400, PL310
- Pre-built software
- Swap & Play enabled
  - Execute at 10s to 100s of MIPS
  - Debug with 100% accuracy
- Source code for all software
- Downloadable 24/7 from Carbon System Exchange

Carbon System Exchange



- Portal dedicated to CPAK access
- Search by IP, OS or benchmark software
- Over 100 CPAKs featuring advanced ARM IP
- New CPAKs constantly being added

carbonsystemexchange.com

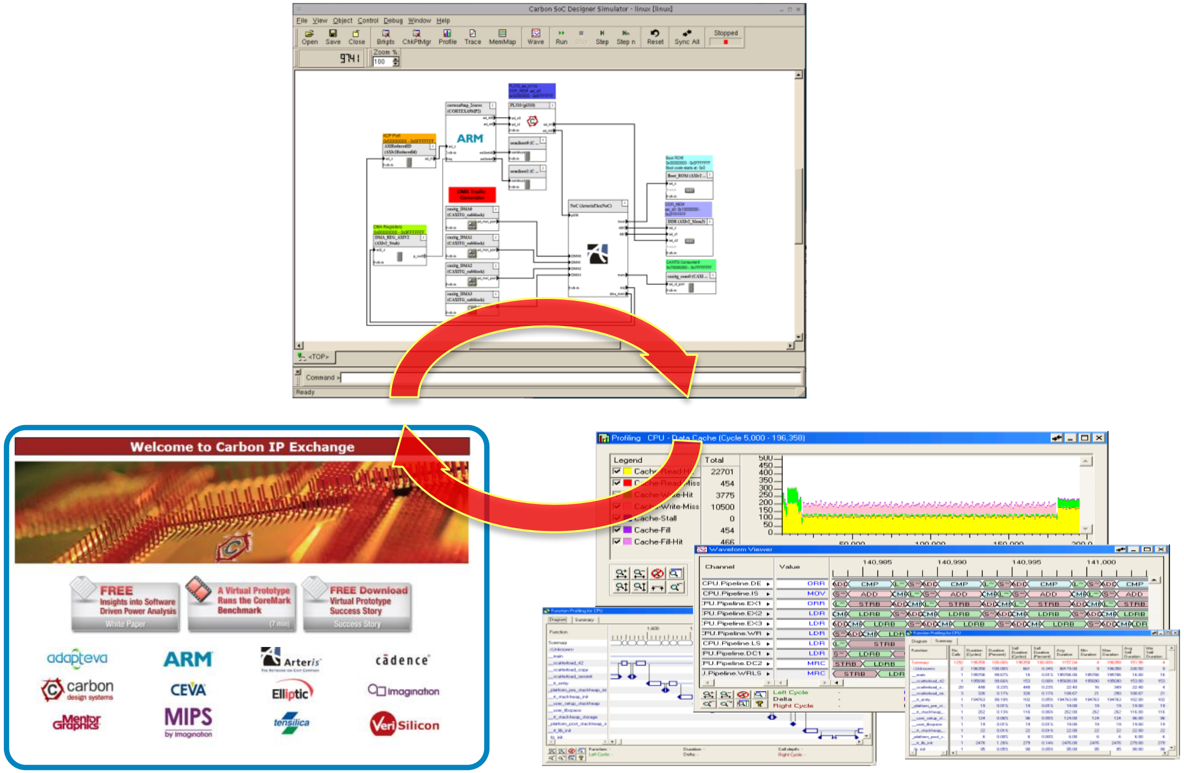
System Performance Analysis Methodology

Accurate System Virtual Prototyping

- Other methods insufficient to optimize price/performance/area tradeoffs
  - Spreadsheets are inaccurate
  - Approximately timed models miss details
  - Traffic generators and VIP lack crucial system traffic
- Only 100% accurate models for entire system can deliver 100% accurate results
- Best way to run real software on processors with real coherency, interconnect, interrupts and memory controllers

Extend Architecture Analysis beyond Interconnect and Memory Controllers

System Performance Analysis Methodology



- Create
  - Model Compilation
  - Fast IP configuration changes
  - System Assembly
- Validate
  - Bus and pipeline performance assumptions
  - IP blocks interfaces
  - Software Operation
- Analyze
  - Cache statistics
  - Memory Subsystems
  - Throughput & latency
  - Arbitration & synchronization

Two Primary Types of Software

Bare Metal Software Applications	Linux Applications
Compiled with ARM DS-5 compiler	Cross-compiled with Linux gcc and added to RAM-based Linux files system
Use semi-hosting for output	Use UART for output
Bring-up on Cycle-Accurate Models	Bring up on ARM Fast Models
Benchmarks ported to reusable startup code	Benchmarks use standard C Linux development environment

ARM A53 Performance Monitoring Unit (PMU)

- CPU Implements PMUv3 architecture
- Gather statistics on the processor and memory system
- Implements 6 counters which can count any of the available events
- Carbon A53 model instruments all PMU events
- Statistics can be gathered without any software programming
- Non-intrusive performance monitoring

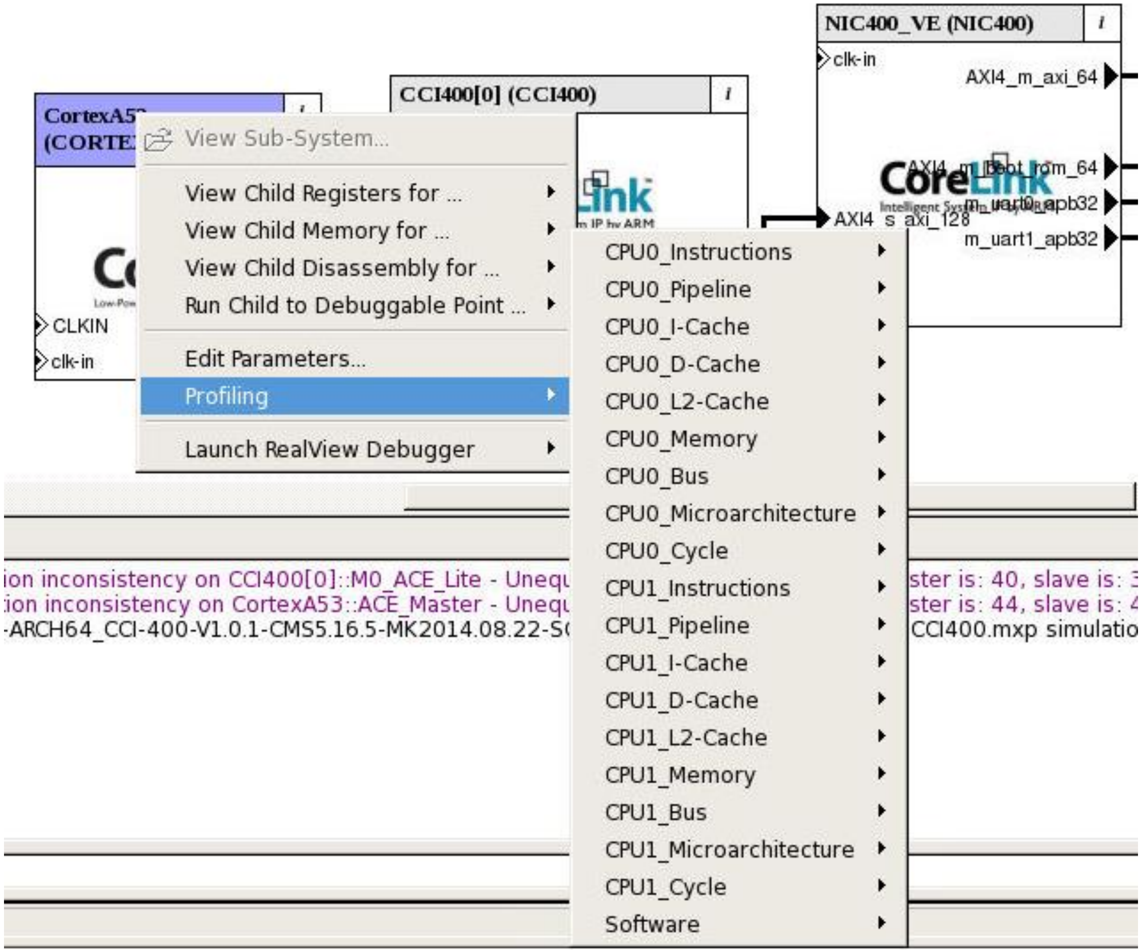
Automatically Instrumented Models Provide Visibility

Partial PMU Events

Table 12-28 PMU events

Event number	Event mnemonic	PMU event bus (to external)	PMU event bus (to trace)	Event name
0x00	SW_INCR	-	-	Software increment. The register is incremented only on writes to the Software Increment Register.
0x01	L1I_CACHE_REFILL	[0]	[0]	L1 Instruction cache refill.
0x02	L1I_TLB_REFILL	[1]	[1]	L1 Instruction TLB refill.
0x03	L1D_CACHE_REFILL	[2]	[2]	L1 Data cache refill.
0x04	L1D_CACHE	[3]	[3]	L1 Data cache access.
0x05	L1D_TLB_REFILL	[4]	[4]	L1 Data TLB refill.
0x06	LD_RETIRED	[5]	[5]	Instruction architecturally executed, condition check pass - load.
0x07	ST_RETIRED	[6]	[6]	Instruction architecturally executed, condition check pass - store.
0x08	INST_RETIRED	[7]	[7]	Instruction architecturally executed.
0x09	EXC_TAKEN	[9]	[9]	Exception taken.
0x0A	EXC_RETURN	[10]	[10]	Exception return.

Enable Profiling During Simulation



- Enable profiling events on each component: CPU, CCI
- Generates database during simulation

Example Software: LMbench

- Set of micro-benchmarks which measures important aspects of system performance
- Timing harness to reliably measure time
- Numerous benchmarks related to bandwidth and latency
- Example program: bw_mem

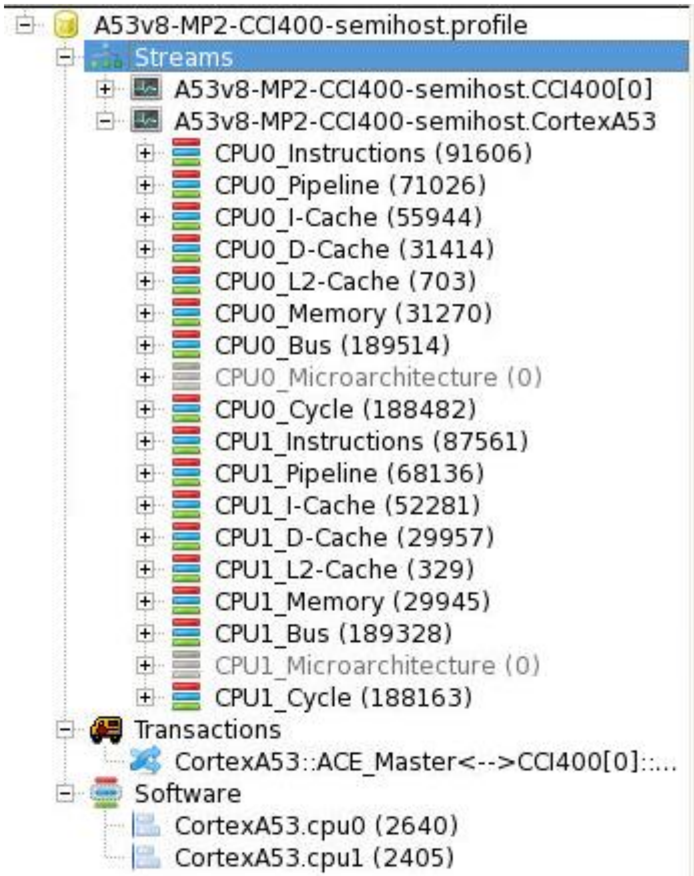
```
DESCRIPTION
       bw_mem allocates twice the specified amount of memory,  zeros  it,  and
       then  times  the copying of the first half to the second half.  Results
       are reported in megabytes moved per second.

       The size specification may end with ``k'' or ``m'' to mean kilobytes (*
       1024) or megabytes (* 1024 * 1024).
```

Multicore Scaling Effects

LMbench Benchmark:

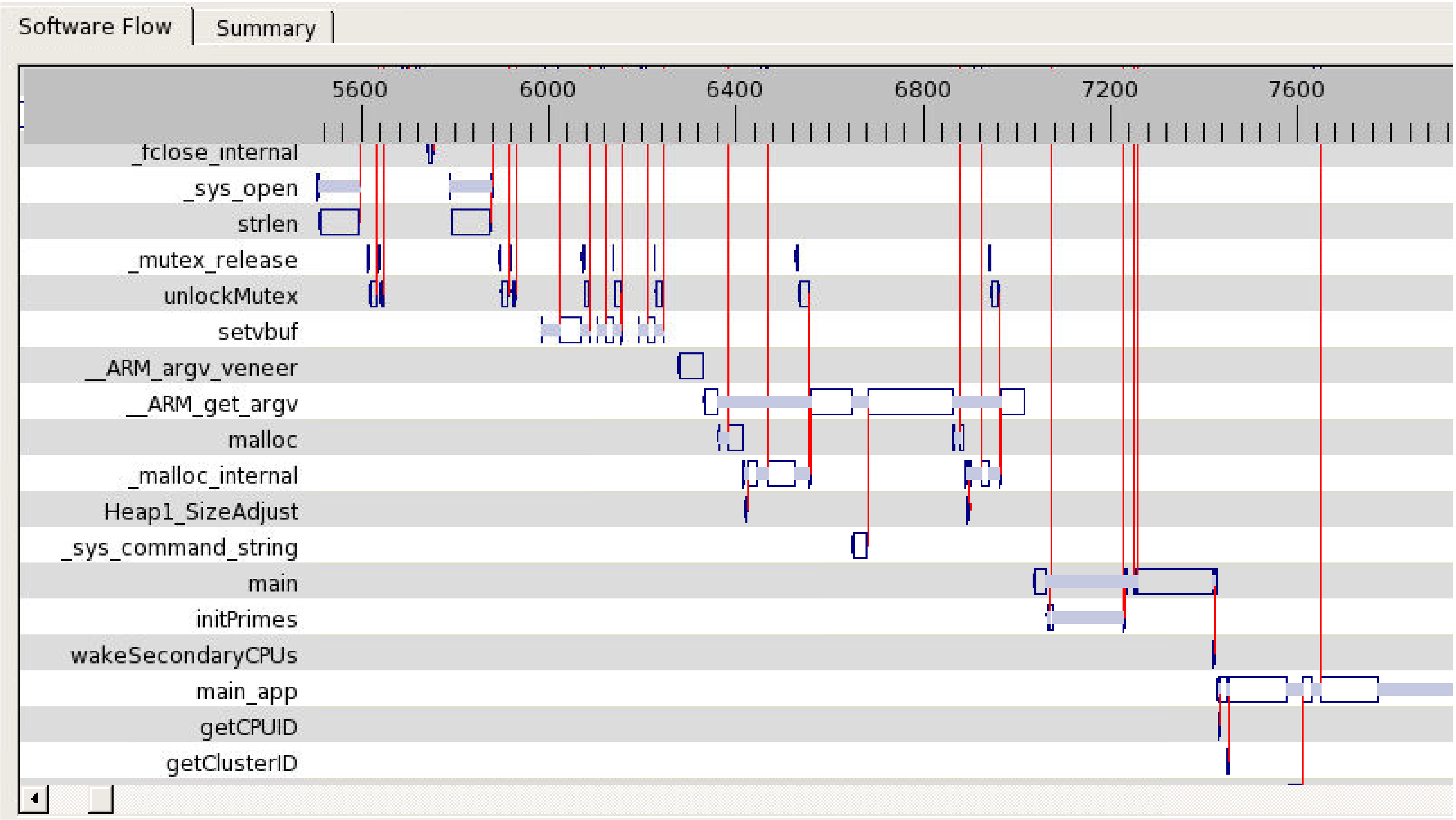
- Block Read Transfer Results
- How does the Transfer Size effect bandwidth?
- What is the bandwidth impact of accessing L2 or DDR?
- Multicore Scaling Effects
  - Linear scaling
  - Increased effective memory bandwidth
    - Cache bandwidth – doubles
    - DDR3 memory bandwidth - doubles

Analyzer Data from Multiple Sources

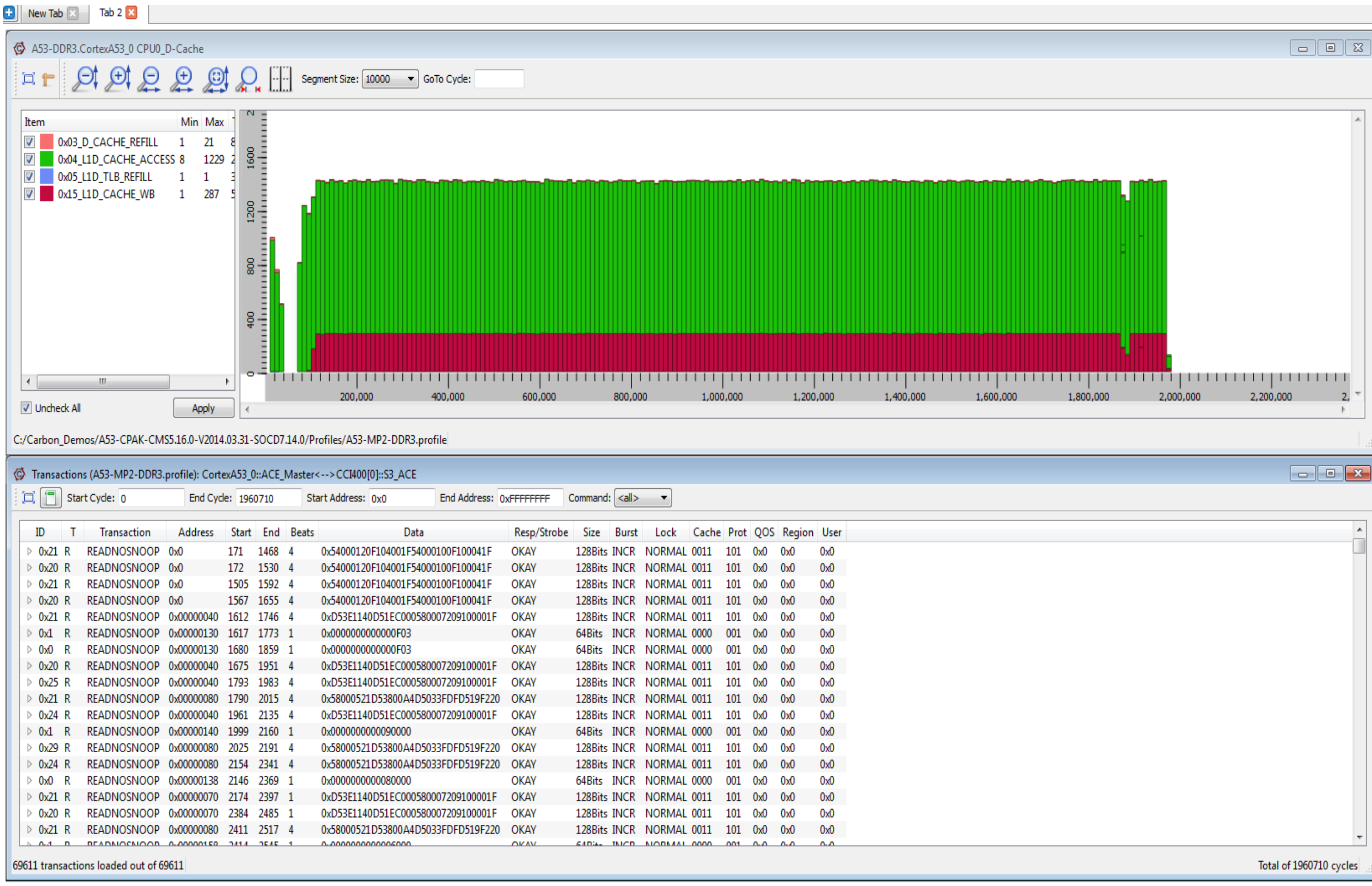


- PMU Information from A53 cores
- ACE Transaction streams between components
- Software Execution Trace

Software Execution Trace



Analyze System Performance Metrics



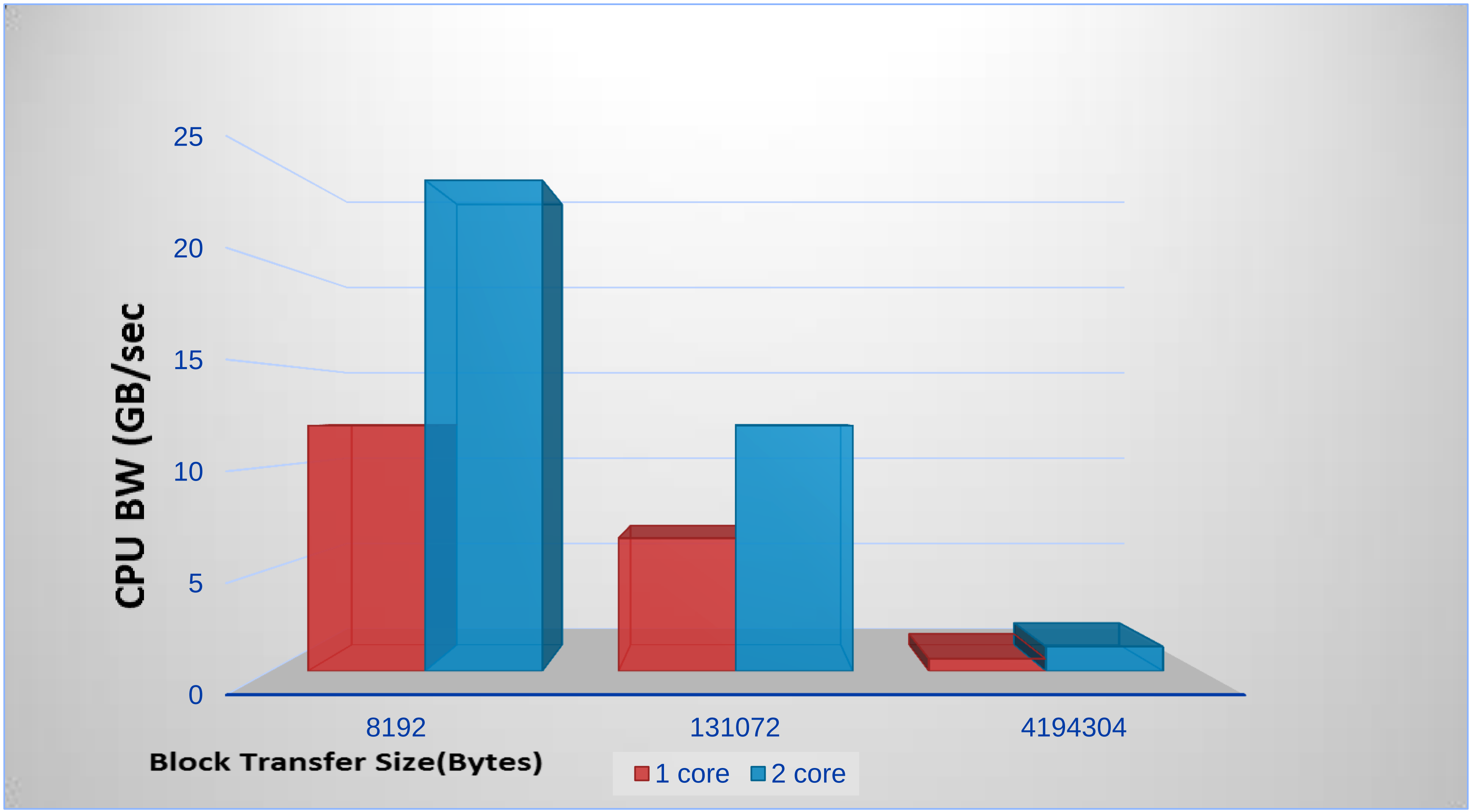
System Metrics Generated from Profiling Data

		Min	Max	Average
LATENCY	(latency/transaction cycles)	Min	Max	Average
	AXI4ACE Read-Trans (Addr) Latency	9	44	17.1891
	AXI4ACE Write-Trans (Addr) Latency	7	16	9.4286
	AXI4ACE Initial Read Latency	9	38	12.5023
	AXI4ACE Initial Write Latency	1	1	1.0000
	AXI4ACE Subsequent Read Latency	1	8	2.0032
	AXI4ACE Subsequent Write Latency	1	1	1.0000
	AXI4ACE Read Burst Latency	9	44	17.1891
	AXI4ACE Write Burst Latency	1	4	1.4286
	AXI4ACE Read Transactions Latency	9	44	17.1891
	AXI4ACE Write Transactions Latency	7	16	9.4286
EFFICIENCY				
	Read Channel Efficiency			0.1679
	Write Channel Efficiency			0.5000

Profile for A53v8-MP2-CCI400-semihost.CortexA53.CPU0

Calculated from Transaction Streams

Cortex-A53 LMbench Block Read Transfer Results



Additional A53 LMbench Suite Results

Test	# CPU	Size	Iterations	CPU BW (GB/sec)
Read/Write	2	L1+L2+DDR	2	11
Mem copy	2	L1+L2+DDR	2	12
bzero	2	L1+L2+DDR	2	8

LMbench Latency Results

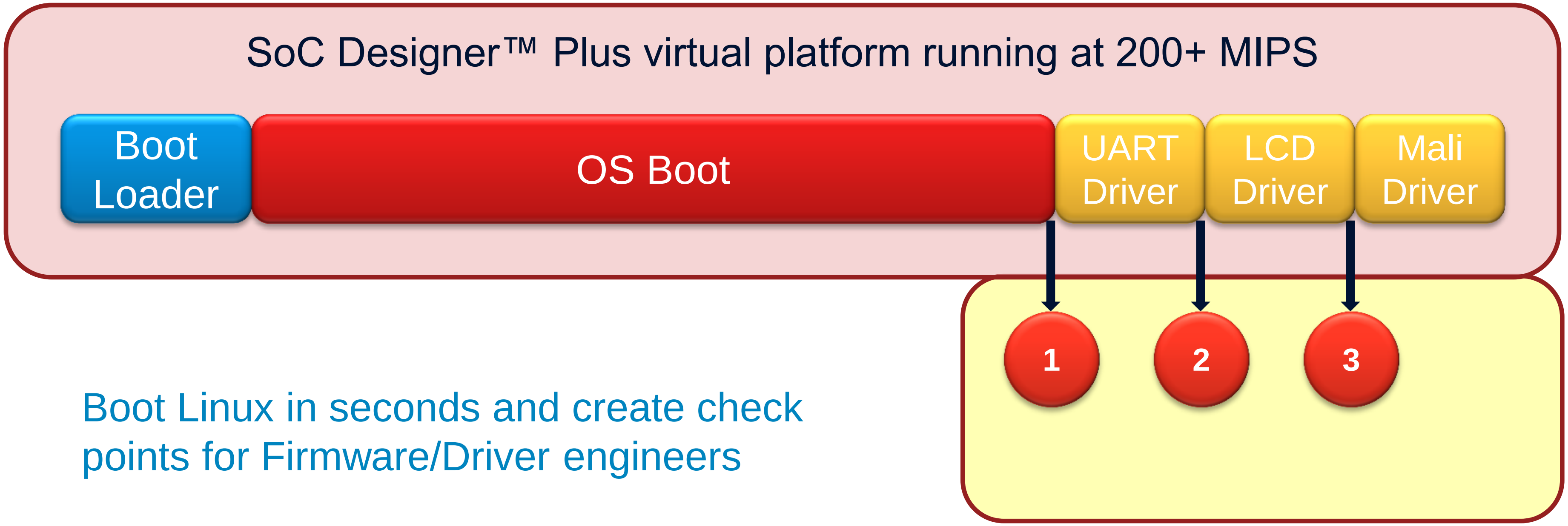
Test	# CPU	Iterations	Size	Stride	Latency (core cycles)
Read	2	8	L1	32	4
Read	2	16	L1+L2	64	8
Read	2	32	L1+L2+DDR	64	183

Observations

- Latency increase when accessing L2 with larger increase when going to DDR

Running real software on A53 increases confidence in metrics

Swap & Play



Boot Linux in seconds and create check points for Firmware/Driver engineers

- Driver developers can debug/validate driver code against an accurate system
- Cycle accuracy without having to spend time booting Linux in CA model
- Each driver developer can independently debug their own driver code

Linux Benchmark Development Flow

Create ARM Fast Model

Confirm system models and configuration

Develop software images and confirm they work

Create Cycle Accurate Model

Compile CA models and configure them to match previous step

Generate FM from CA

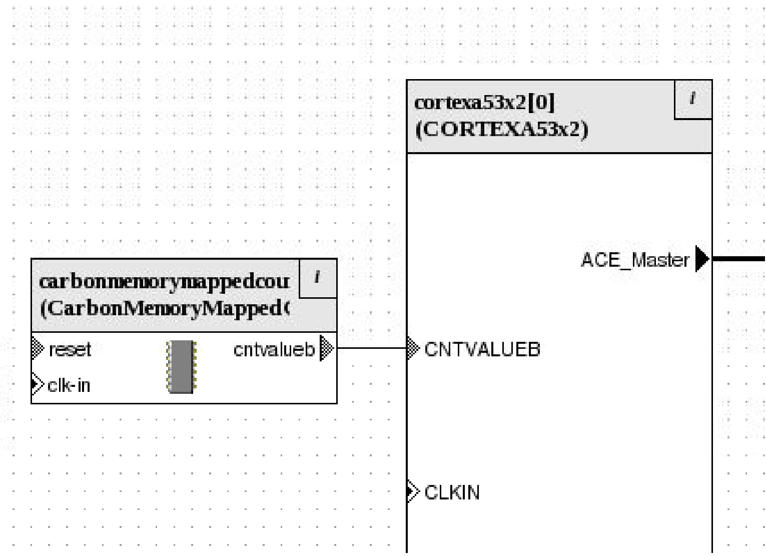
Wizard to convert CA to FM to check the CA configuration is correct and software functions properly

Run Swap & Play

Use checkpoints to run targeted segments of CA simulation

Timing Linux Benchmarks

- Notion of time comes from Linux timer
  - Use Internal CPU Generic Timers
  - Driven by Global System Counter, CNTVALUEB CPU input
  - Each increment of System Counter indicates the passage of time at some frequency
- Linux scheduler is based on concept of HZ which has a value of 100
  - Kernel tries to schedule about every 10 ms using provided timer



Cycle Based Simulation has almost no notion of time

Linux Device Tree for A53 Generic Timer: also called Architected Timer

```
timer {
        compatible = "arm,armv8-timer";
        interrupts = <1 13 0xff01>,
                     <1 14 0xff01>,
                     <1 11 0xff01>,
                     <1 10 0xff01>;
        clock-frequency = <100000000>;
};
```

Tells Linux the frequency of the timer, 100 MHz in this case. Changing frequency has 2 visible effects

1. Time reported to run a software benchmark will change
2. Kernel will re-schedule tasks more or less frequently

Running Linux Benchmarks

- Link everything into single AXF file for ease of use
  - Boot Loader
  - Kernel Image
  - RAM-based File System
  - Device Tree
- Kernel need not change as systems change
- Launch as initial process using kernel command line using Linux Device Tree

Technique to Launch Benchmarks on Boot

```
/dts-v1/;

/ {
        model = "RTSM_VE_Cortex_A15x4";
        compatible = "arm,rtsm_ve,cortex_a15x4", "arm,vexpress";
        interrupt-parent = <&gic>;
        #address-cells = <2>;
        #size-cells = <2>;

        /include/ "../../../../Applications/dts-tests/bw_pipe.dtsi"
```

```
chosen {
  bootargs = "root=/dev/ram0 rw console=ttyAMA0 earlyprintk rdinit=/root/bw_pipe.sh";

};
```

Automatically launch test script on boot
Include above file in Device Tree source to launch test

Detecting Start of Application

- Linux process launch
- Breakpoint to take initial checkpoint
  - Detect the process we want to track is launched
  - Places in Linux kernel where process creation takes place
  - Access the name of the new function to run in arguments
- Load checkpoint and start profiling

OS Level Performance Analysis
Using Fast Models, 100% accurate models, Swap & Play

- System benchmarks can execute for many billions of cycles
- Executing in cycle accurate system could take days
- Swap & Play enables accurate simulation of benchmark areas which it may take too long to reach in a single simulation
- Can execute multiple checkpoints in parallel to deliver days worth of results in a few hours
- Enables fast, accurate performance analysis of OS level benchmarks

Summary

- System Performance Analysis using Create, Validate, Analyze methodology
- Models of ARM’s advanced IP and CPAK reference systems with software enable decisions early in the design process
- Accurate IP models are easy to generate, easy to work with, and fully instrumented for analysis
- Ability to run software, including Linux benchmarks is a must for System Performance Analysis

Jason Andrews

Director of Product Engineering

jasona@carbondesignsystems.com

carbon design systems
The Trusted Path to *Accuracy*

2014 ARM TechCon

2014 ARM® TechCon™

UBM Tech