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

Jason Andrews
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

ARM® Cortex®-A53

- ARMv8-A 32b/64b CPU
- ARM CoreSight™ Multicore Debug and Trace
- NEON™ SIMD engine with crypto ext.
- Floating Point Unit
- Core 1, 2, 3, 4
- 8-64k I-Cache w/parity
- 8-64k D-Cache w/ECC
- ACP, SCU
- L2 w/ECC (128kB ~ 2MB)

Configurable AMBA®4 ACE or AMBA5 CHI Coherent Bus Interface
IP Model Creation

- 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

<table>
<thead>
<tr>
<th>IP Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SELECT NUMBER OF CPUs</strong></td>
</tr>
<tr>
<td><strong>NEON FP</strong></td>
</tr>
<tr>
<td><strong>CRYPTOGRAPHY EXTENSION</strong></td>
</tr>
<tr>
<td><strong>EXTERNAL MEMORY INTERFACE SUPPORT</strong></td>
</tr>
<tr>
<td><strong>L1 INSTRUCTION CACHE SIZE</strong></td>
</tr>
<tr>
<td><strong>L1 DATA CACHE SIZE</strong></td>
</tr>
<tr>
<td><strong>L2_CACHE</strong></td>
</tr>
<tr>
<td><strong>ACP</strong></td>
</tr>
<tr>
<td><strong>L2 SIZE</strong></td>
</tr>
<tr>
<td><strong>L2_INPUT_LATENCY</strong></td>
</tr>
</tbody>
</table>
CoreLink NIC-400 Network Interconnect

- Set multiple clock domains for best performance and power saving
- Choose topology to minimize CPU latency and routing congestion
- Graphical UI for configuration in AMBA Designer
- 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

NIC-400 Model
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
**Portal dedicated to CPAK access**

**Search by IP, OS or benchmark software**

**Over 100 CPAKs featuring advanced ARM IP**

**New CPAKs constantly being added**

carbon-system-exchange.com
System Performance Analysis Methodology
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

<table>
<thead>
<tr>
<th>Bare Metal Software Applications</th>
<th>Linux Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiled with ARM DS-5 compiler</td>
<td>Cross-compiled with Linux gcc and added to RAM-based Linux files system</td>
</tr>
<tr>
<td>Use semi-hosting for output</td>
<td>Use UART for output</td>
</tr>
<tr>
<td>Bring-up on Cycle-Accurate Models</td>
<td>Bring up on ARM Fast Models</td>
</tr>
<tr>
<td>Benchmarks ported to reusable startup code</td>
<td>Benchmarks use standard C Linux development environment</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Event number</th>
<th>Event mnemonic</th>
<th>PMU event bus (to external)</th>
<th>PMU event bus (to trace)</th>
<th>Event name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>SW_INCR</td>
<td>-</td>
<td>-</td>
<td>Software increment. The register is incremented only on writes to the Software Increment Register.</td>
</tr>
<tr>
<td>0x01</td>
<td>L1I_CACHE_REFILL</td>
<td>[0]</td>
<td>[0]</td>
<td>L1 Instruction cache refill.</td>
</tr>
<tr>
<td>0x02</td>
<td>L1I_TLB_REFILL</td>
<td>[1]</td>
<td>[1]</td>
<td>L1 Instruction TLB refill.</td>
</tr>
<tr>
<td>0x03</td>
<td>L1D_CACHE_REFILL</td>
<td>[2]</td>
<td>[2]</td>
<td>L1 Data cache refill.</td>
</tr>
<tr>
<td>0x04</td>
<td>L1D_CACHE</td>
<td>[3]</td>
<td>[3]</td>
<td>L1 Data cache access.</td>
</tr>
<tr>
<td>0x05</td>
<td>L1D_TLB_REFILL</td>
<td>[4]</td>
<td>[4]</td>
<td>L1 Data TLB refill.</td>
</tr>
<tr>
<td>0x06</td>
<td>LD RETIRED</td>
<td>[5]</td>
<td>[5]</td>
<td>Instruction architecturally executed, condition check pass - load.</td>
</tr>
<tr>
<td>0x07</td>
<td>ST RETIRED</td>
<td>[6]</td>
<td>[6]</td>
<td>Instruction architecturally executed, condition check pass - store.</td>
</tr>
<tr>
<td>0x08</td>
<td>INST RETIRED</td>
<td>[7]</td>
<td>[7]</td>
<td>Instruction architecturally executed.</td>
</tr>
<tr>
<td>0x09</td>
<td>EXC_TAKEN</td>
<td>[9]</td>
<td>[9]</td>
<td>Exception taken.</td>
</tr>
<tr>
<td>0x0A</td>
<td>EXC RETURN</td>
<td>[10]</td>
<td>[10]</td>
<td>Exception return.</td>
</tr>
</tbody>
</table>
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

Calculated from Transaction Streams

<table>
<thead>
<tr>
<th>LATENCY</th>
<th>Min</th>
<th>Max</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI4ACE Read-Trans (Addr) Latency</td>
<td>9</td>
<td>44</td>
<td>17.1891</td>
</tr>
<tr>
<td>AXI4ACE Write-Trans (Addr) Latency</td>
<td>7</td>
<td>16</td>
<td>9.4286</td>
</tr>
<tr>
<td>AXI4ACE Initial Read Latency</td>
<td>9</td>
<td>38</td>
<td>12.5023</td>
</tr>
<tr>
<td>AXI4ACE Initial Write Latency</td>
<td>1</td>
<td>1</td>
<td>1.0000</td>
</tr>
<tr>
<td>AXI4ACE Subsequent Read Latency</td>
<td>1</td>
<td>8</td>
<td>2.0032</td>
</tr>
<tr>
<td>AXI4ACE Subsequent Write Latency</td>
<td>1</td>
<td>1</td>
<td>1.0000</td>
</tr>
<tr>
<td>AXI4ACE Read Burst Latency</td>
<td>9</td>
<td>44</td>
<td>17.1891</td>
</tr>
<tr>
<td>AXI4ACE Write Burst Latency</td>
<td>1</td>
<td>4</td>
<td>1.4286</td>
</tr>
<tr>
<td>AXI4ACE Read Transactions Latency</td>
<td>9</td>
<td>44</td>
<td>17.1891</td>
</tr>
<tr>
<td>AXI4ACE Write Transactions Latency</td>
<td>7</td>
<td>16</td>
<td>9.4286</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EFFICIENCY</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Channel Efficiency</td>
<td>0.1679</td>
</tr>
<tr>
<td>Write Channel Efficiency</td>
<td>0.5000</td>
</tr>
</tbody>
</table>

Profile for A53v8-MP2-CCI400-semihost.CortexA53.CPU0
Cortex-A53 LMbench Block Read Transfer Results

The graph shows the CPU bandwidth (GB/sec) for different block transfer sizes (Bytes) and core configurations.

- **8192 Bytes**:
  - 1 core: ~25 GB/sec
  - 2 core: ~20 GB/sec

- **131072 Bytes**:
  - 1 core: ~15 GB/sec
  - 2 core: ~10 GB/sec

- **4194304 Bytes**:
  - 1 core: ~5 GB/sec
  - 2 core: ~2.5 GB/sec

The results indicate a significant impact of core configuration on bandwidth performance.
## Additional A53 LMbench Suite Results

<table>
<thead>
<tr>
<th>Test</th>
<th># CPU</th>
<th>Size</th>
<th>Iterations</th>
<th>CPU BW (GB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>2</td>
<td>L1+L2+DDR</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Mem copy</td>
<td>2</td>
<td>L1+L2+DDR</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>bzero</td>
<td>2</td>
<td>L1+L2+DDR</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>
## LMbench Latency Results

<table>
<thead>
<tr>
<th>Test</th>
<th># CPU</th>
<th>Iterations</th>
<th>Size</th>
<th>Stride</th>
<th>Latency (core cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>2</td>
<td>8</td>
<td>L1</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>Read</td>
<td>2</td>
<td>16</td>
<td>L1+L2</td>
<td>64</td>
<td>8</td>
</tr>
<tr>
<td>Read</td>
<td>2</td>
<td>32</td>
<td>L1+L2+DDR</td>
<td>64</td>
<td>183</td>
</tr>
</tbody>
</table>

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

Running real software on A53 increases confidence in metrics
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

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
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