ATC-104: Software Driven Power and Performance Characterization

Andy Ladd – VP Applications and Methodology

November 9th, 2010

Overview:

Architects and system engineers face extreme challenges in developing embedded SoC-based systems to meet the tight power budgets of today's mobile and handheld applications while still maintaining the performance and quality of service. Without fully understanding the performance and power, it is impossible to properly understand the Power, Performance, Area (PPA) trade-offs. These decisions need to be upfront, before the implementation phase has started. The problem is that current methods typically only provide rough estimates of power and performance and don't get refined until late in the product cycle when it is very hard to make changes. What is needed is a methodology to achieve accurate performance and power characterization in the early phases of a project.

Architects need to prove their design assumptions before a major commitment has been made to the design implementation. These include:

- IP selection
- Exploration of the performance impact of hardware/software trade-offs
- Measuring interconnect performance of complex bus architectures
- Understanding cache and memory subsystem performance and trade-offs.
- Analyzing HW accelerators/GPUs and their affect on the system.

In addition, architects need to consider two factors that come into play with respect to meeting power budgets for today's battery powered devices: 1) Is the hardware being architected and designed to efficiently conserve energy? 2) Is the software controlling the power states of the device operating as they should? Figuring out these two factors requires the ability to characterize the power early in the development cycle and provide the modeling and analysis capabilities to understand how they affect the power constraints of the system and how they relate to performance.

This discussion illustrates a methodology to understand both performance and power at the system-level running real software on a accurate virtual platform.

Background:

Three goals are discussed to help the architect and software engineer understand the power and performance trade-offs early in the project cycle:

1. Define a methodology to understand and characterize the power and performance early in the project cycle

2. Refine and debug the software that controls the power management of the system and understand it's affects on power and performance
3. Understand how process geometry and frequency affects power and performance on a SoC-based Design

Current methods suffer from various ailments. One solution is to use high level abstract models such as behavioral simulation or spreadsheets to characterize the performance and power of a system. While this can give the architect a rough understanding of the solution, the problem with this approach is that it does not provide the required level of accuracy to fully understand the behavior of the system. Another approach is to use RTL simulations to understand the performance and power. This can be augmented by gate and transistor level simulations to augment and refine the power analysis. But these techniques don't provide the simulation performance to run real software applications so the data suffers from unrealistic situations created by artificial verification testbenches and traffic generators. Finally, hardware solutions such as emulation or prototypes can provide the speed and accuracy but are available much too late in the design process to be an effective architecture analysis tool.

Approach:

The approach taken here to accomplish the goals outlined in the previous section primarily revolves around the creation of a virtual platform to represent the SoC-based system and to run the real software targeted for the system to understand the power and performance. Accuracy is required for the critical pieces of the analysis and for these components, cycle accurate models are used. Such a platform is ideal for understanding performance and power issues, as well as for the software that configures and manages the power of the system. The ideal system will have the following requirements:

- Speed: The platform must be fast enough to run real software and real traffic patterns
- Accuracy: Accurate component models to represent the key parts of the system in order to provide required performance and power measurements
- Visibility: Must provide the visibility to profile and monitor the interfaces and internals of the components
- Configurability: Ability to change and reconfigure platform to analyze a multitude of experiments

Basics:

Traditional flows and methodologies are very serial in nature whereby much of the true analysis and development is done late in the design cycle. The traditional flow suffers from requiring hardware prototypes, whether they be emulation, FPGA prototypes, or Silicon prototypes, to complete the software development, validate the hardware and software interfaces, and validate the architectural, performance, and power decisions. Providing a reference system for outside customers can come very late in the project cycle. Exacerbating

this backend loaded flow is the fact that bugs and issues discovered late in a project are very expensive to fix.

Virtual platforms (or ESL) provide a methodology that fixes many problems in the traditional flow by providing a platform to do very accurate performance and power modeling and analysis at the beginning of the architectural phase of a project. This provides a much better understanding of the performance and power of the system as well as the IP to be used. Starting with a better understanding provides a better launch for implementation and reduces the bugs and issues seen downstream, reducing the time it takes to implement and verify. In addition, the same platform can be used for pre-silicon software development long before silicon or FPGA prototypes are available. A virtual platform can also be used as a reference platform that can be delivered to outside users before actual silicon is available.

Understanding Performance:

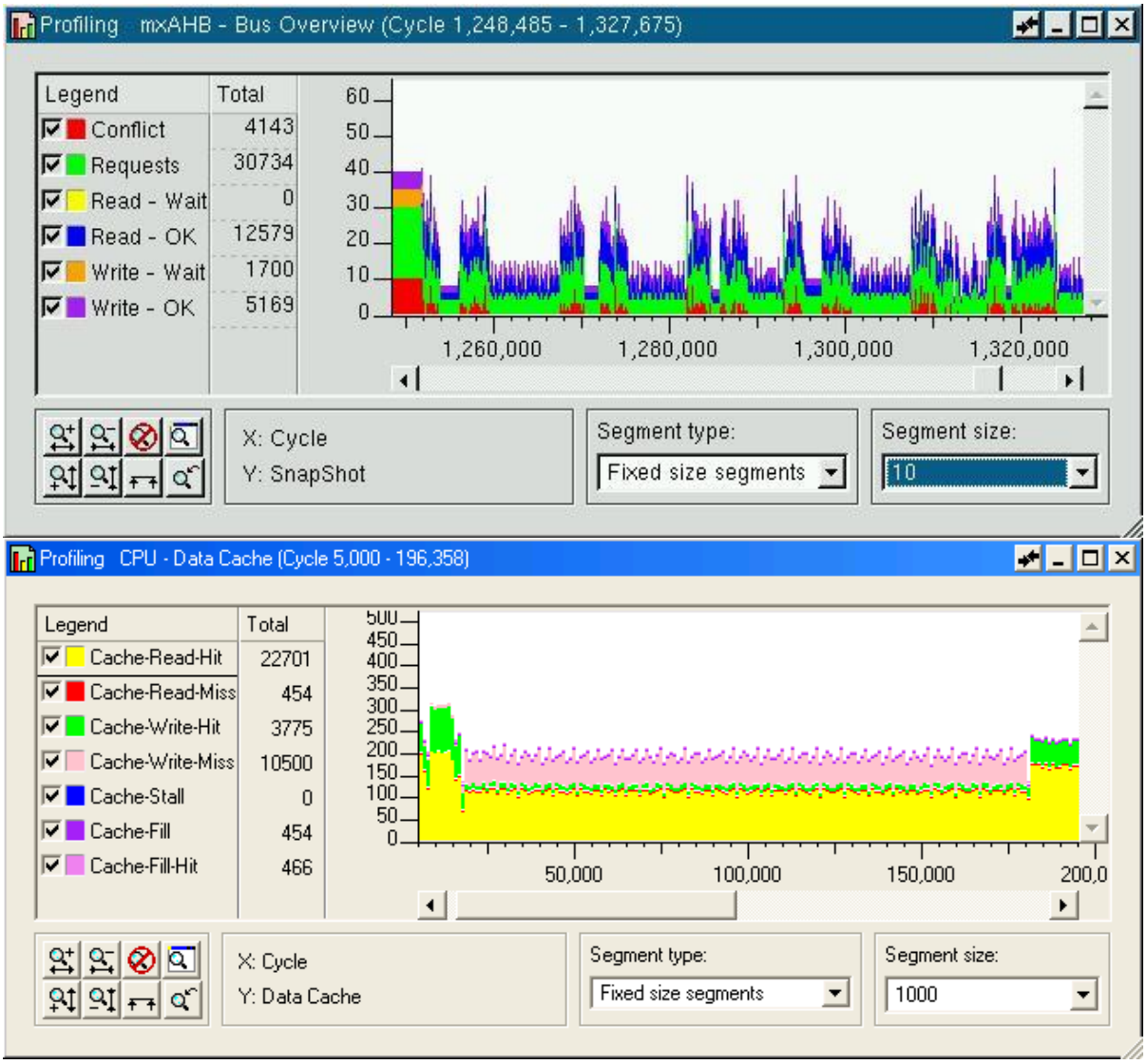
Part of goal 1 is to define a methodology to understand and characterize performance early in the project cycle. Achieving this goal enables architects to have a better understanding of the correct IP to use in their system and the utilization and performance of the hardware resources such as the cache/memory subsystem, bus fabric, and hardware accelerators and engines. It also allows them to understand and tune the software performance on the new hardware platform.

The methodology needed to achieve goal 1 contains several steps. First a virtual platform needs to be created that contains the vital and interesting components of the system. For fabric analysis this can be driven with traffic generators either algorithmically at the beginning of a project, or with traces as things are refined. Accurate models are needed of the bus fabric itself to understand the fine details of the latencies, throughput, arbitration, and utilization. The traffic generators should eventually be replaced with the rest of the system so that the bus fabric can be analyzed under real software loads. When analyzing the processor/cache/memory subsystem, the hardware accelerators, and peripherals, a virtual platform representing the entire system should be used along with accurate models in the critical paths. These platforms should be analyzed under real software loads in order to understand the performance in realistic situations.

The next step is to compile and load the representative software onto the virtual platform. This can be done through the typical software tool chain, for example with the Realview Development System. One advantage of this approach is that software developers use the same native compilers and debuggers that they are accustomed with to develop, debug and tune their code on real hardware.

Once the software (or traffic) is loaded onto the platform, the user can characterize the performance of the system. One key advantage of a virtual platform is the visibility and controllability the user has to take measurements and analyze behavior. With hardware based platforms (FPGA prototypes and silicon prototypes), visibility is limited and sometimes the user cannot inspect what they need to analyze. In addition, hardware based platforms can't be "stopped" such that the user can inspect or change things while time is frozen, without parts of

the system getting out of step. This is especially true in multi-core designs. Virtual platforms eliminate these problems because the platforms provide 100% visibility into each component and interface as well as letting the user "stop" the system and freeze time to inspect or change things without any part of the system getting out of step. These capabilities dramatically increase the user's productivity in profiling and characterizing the software as well as hardware aspects of the design. Examples of this include analyzing bus/fabric latencies, throughput, and utilization, cache profiling, frame rates, pixels rates, etc. Here are some examples of virtual platform profiling capabilities.



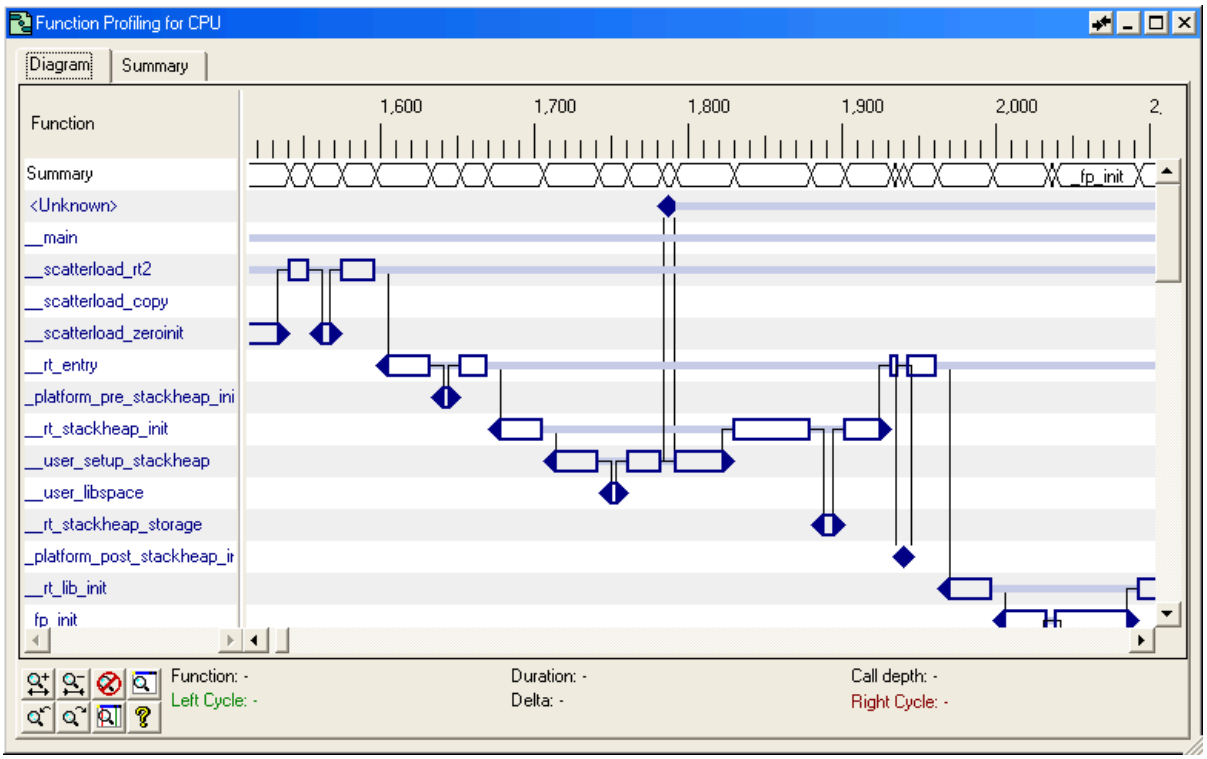


Figure 1. Hardware (Cache and Bus) and Software Profiling

Finally, the last step is to make changes to the hardware or software configuration based on the analysis conducted in the previous step and iterate until the performance goals are met.

Understanding Power:

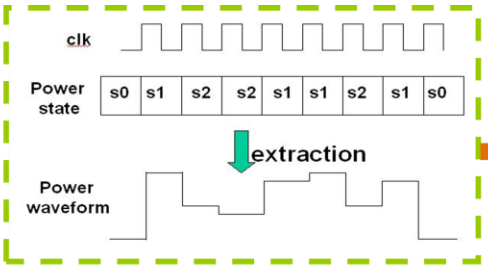
The second part of goal 1 is to define a methodology to understand and characterize the power early in the design cycle. Achieving this goal provides the architect with an accurate understanding of power consumption under a real software load enabling a correct understanding of clocking, clock gating, frequency and voltage scaling, and power shut-off. Contrast this to analyzing power with RTL verification environment or gate level simulations where the stimulus provided to the circuits is synthetic and artificial and often doesn't correlate well with what the circuits will see in a real system under a real software load. Often the RTL verification environment or gate level simulations indicate an overly pessimistic power consumption because they tend to over stimulate the design compared to the real software. This leads to over engineering the design to meet the power requirements and adding cost to the system.

Power analysis is part of a refinement process. Using virtual platforms with real software to understanding power early in the design cycle leads to better decision making early followed with power analysis at the RTL level and then at the gate and transistor level later. Again, issues found earlier in the design cycle are much easier to fix.

The methodology to attack power analysis can be broken down into two parts. First, a more brute force methodology based on an activity flow. In this flow activity data in the form of trace waveforms are collected from the interesting system components and fed into RTL power analysis tools. Basically, a virtual platform is created with the major and important components in the system much like was done for performance analysis described previously. Software that will be run in the real system is then loaded into the platform and run. Trace data is collected (for example VCD waveforms) for each component that the user wants to conduct power analysis (e.g. the processor, caches, fabric, memory subsystem, graphics unit, etc). This trace data is used as activity data to feed any of the available EDA RTL power analysis tools on the market. These tools use the RTL description of the design as well as the technology file of the process to calculate power but they rely on activity data to be supplied by a simulator. The virtual platform provides an ideal way to generate this activity data under a real software load.

One of the drawbacks of the above method is that dumping waveforms of activity data tends to slow down the virtual platform considerably. So the second part to this methodology involves a simple refinement from part one. We achieve this by just monitoring the power states of a design and doing tabular power look-ups to assign power to each power state dynamically. The way the power state flow works is that on the first run you collect the power data just as you did in the activity flow method above. But this time you also record what power state the design is in at the times of the simulation. The power states differ depending upon the IP but for example a GPU might have an IDLE power state, a power state for each of the pixel engines being enabled, and a power state for the cache/memory interface being enabled. Using the activity data to drive the RTL power estimator, the resulting output is averaged out for the time the device is in each power state. Then a look-up table is created that assigns the average power while the component is in that mode of operation for each state entry in the table.

For example:



Time	State	Power Value
1	S0	0.10
2	S1	0.33
3	S2	0.17
4	S2	0.15
5	S1	0.32
6	S1	0.34
7	S2	0.16
8	S1	0.32
9	S0	0.09

Example of average power data

Process	Frequency	S0	S1	S2
65nm	200 MHz	0.00095	0.00328	.00160
40nm	200 MHz	0.00072	0.00162	.00073

Once the table is generated, there is no longer a need to dump waveforms. Therefore subsequent runs only need to monitor the signals or registers that identify what power state the device is in and uses the table look-up to estimate the power while running in that state. Without the drain of the waveform dumping overhead, a great deal of software can be run and analyzed to understand the power of the system.

Power Management Software:

Goal two is to refine and debug the software that controls the power management of the system and understand its affects on power and performance. This includes the software that controls Dynamic Frequency and Voltage Scaling (DFVS). Dynamic Power Management (DPM), and Power Shut-off (PSO)/Power Gating. With respect to DVFS software and algorithms, the architect needs to understanding the trade-off between scaling the voltage and frequency versus the performance achieved. Making sure the algorithms are correctly calculating the workloads and deadlines to insure optimal energy operation while still meeting the performance needs of the system are critical. For example, miscalculation can cause unwanted video effects or glitches in the audio when deadlines are not being met. Miscalculation can also drain too much energy out of the batteries if the frequency and voltage is not scaled aggressively enough. In addition, how does the system engineer know if the latencies involved with switching the voltages and frequencies are being handled correctly in the hardware and software?

Another technique that adds its own twist is Power Shut-off and Power Gating. Similar to DFVS above the software that controls this needs to be debugged and optimized. An understanding is needed of how this software affects the hardware's performance and power.

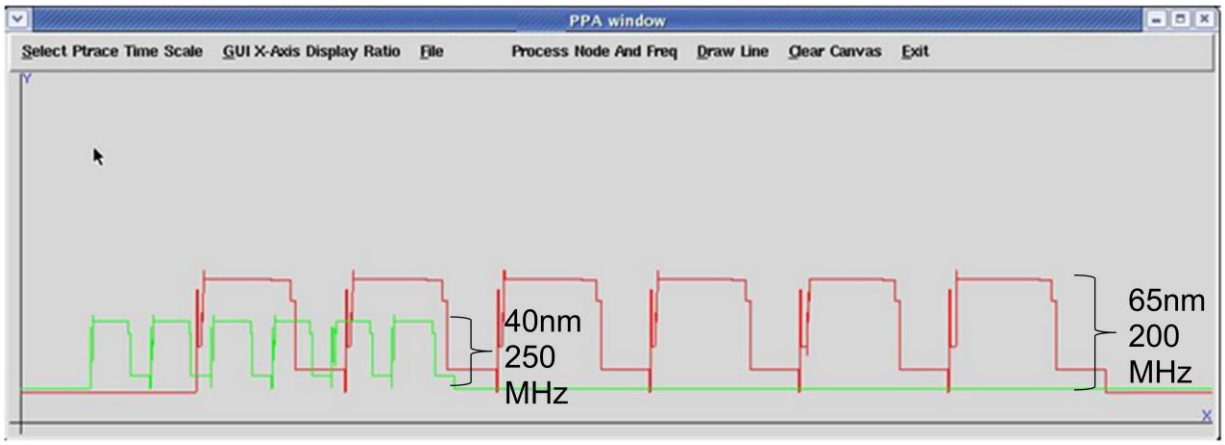
PSO wake-up and sleep times need to be factored in with the software as well as their affects with the hardware performance. In addition, how does the system engineer confirm that the proper state is being saved before power is shut-off and restored when the components are repowered?

Using software driven power analysis on a virtual platform of the system provides an ideal solution to debug the algorithms, power management, and scaling software and how they impact the power and performance of the system. The methodology to achieve this starts with creating a virtual platform of the important pieces of the system (processor, caches, fabric, memory subsystem, peripherals, accelerators, etc). Isolate the clock and voltage domains such that the system can control the frequency of the power domains and report the operating voltages. The frequency domains will determine the performance of the components while the frequency and voltage domains will feed the power analysis tools. The next step is to compile and load the representative power/frequency management software on the platform. After the user has debugged any issues of the software running on the system they can then measure the energy consumption to see if it meets the target goals. In addition they can measure the true performance of the system to make sure all the workloads are meeting their deadlines. Any issue with the energy consumption or performance trade-offs can easily be analyzed, fixed and refined within the software (and hardware). Adjustment can be made and the process reiterated until the correct balance between energy efficiency and performance is achieved.

Process Nodes and Frequency:

The final goal is to provide a method to understand how process geometry and frequency affects power and performance on a SoC Design. This is basically an application and example of using the techniques that have been discussed previously.

The methodology of this technique is to start with a virtual platform with associated software as described in the preceding sections. Then use the power analysis with power states flow described earlier. This allows the user to run multiple software applications efficiently by only tracing the power states of the components. Table looks-ups are created for each process node, and frequency (and voltage) making it easy to analyze the affects of changing process nodes and frequencies after the data has been collected. This can all be done post-simulation giving the project manager and architect an easy way to understand the value and impacts of migrating to different process nodes, frequency and voltages.



Summary:

We have illustrated a methodology for software driven performance and power analysis and characterization that is achievable very early in the design flow. This methodology is enabled through running real software on virtual platforms. We also illustrated two important use cases. The first involves the debug and refinement of power management software and it's affects on power and performance of the system. The second involves understanding and characterizing the power/performance trade-offs of between process geometry, frequency, and voltage targeted to the manufacturing process and how the project manager can select the correct one early in the design phase. The techniques shown here demonstrate how the user can get accurate performance and power data using accurate models in a virtual platform and, just as importantly, realistic performance and power data by running real software on that same virtual platform. Virtual platforms provide these capabilities early in the project phase when the architects are making their decisions and trade-offs providing a much more robust design flow and reducing the amount of engineering effort downstream in the design and implementation cycle.