Towards Interfaces for Integrated Performance and Power Analysis and Simulation

Chris Bleakley[1], Tom Clerckx[2], Harald Devos[3], Matthias Grumer[4], Alex Janek[5], Ulrich Kremer[6], Christian W. Probst[7], Phillip Stanley-Marbell[8], Christian Steger[4], Vasanth Venkatachalam[9], Manuel Wendt[4]

[1] University College, Dublin, Ireland
[2] Vrije Universiteit Brussel, Belgium
[3] Ghent University, Belgium
[4] Graz University of Technology, Austria
[5] CISC, Klagenfurt, Germany
[6] Rutgers, The State University of New Jersey, U.S.A.
[7] Technical University of Denmark
[8] Technische Universiteit Eindhoven, The Netherlands
[9] University of California, Irvine, U.S.A.

Abstract. In the design and optimization of power-aware computing systems, it is often desired to estimate power consumption at various levels of abstraction, e.g., at the transistor, gate, RTL, behavioral or transaction levels. Tools for power estimation at these different levels of abstraction require specialized expertise, e.g., understanding of device physics for circuit-level power estimation, and as such are necessarily developed by different research communities.

In the optimization of complete platforms however, it is desired to be able to obtain aggregate power and performance estimates for the different components of a system, and this requires the ability to model the system at a mixture of levels of abstraction.

One approach to enabling such cross-abstraction modeling, is to define a mechanism for interchange of data between tools at different layers of abstraction, for both static analysis and simulation-based studies. This document presents preliminary discussions on the requirements of such an interface.

Keywords. Power Estimation Tools, Simulation, Tool Interfaces

1 Introduction

The requirements of computation, whether in the form of instruction execution in a general purpose processor, or in the form of a state machine responding to input stimuli, influences the physical implementation and runtime behavior of hardware. The energy consumed in a hardware platform is influenced by its carrying out of operations of interest in an application, whether due to the dynamic behavior of the application (e.g., and its associated dynamic power dissipation), or its hardware requirements (e.g., and its associated leakage power

Dagstuhl Seminar Proceedings 07041
Power-aware Computing Systems
http://drops.dagstuhl.de/opus/volltexte/2007/1107

dissipation). In design- and run-time estimation of power consumption and related issues (e.g., thermals, reliability), it is therefore often necessary to consider both the modeling of the "application", and that of hardware.

Tools for complete-system power estimation must necessarily be able to estimate power consumption of different components of the system, at different levels of abstraction, e.g., at the gate, RTL, or behavioral level. These different modeling abstraction requirements might arise either for different hardware structures (e.g., modeling for a single SRAM cell might be desired at the gate level, but an entire cache might be modeled at the RTL), or might be required for different stages of the design cycle (e.g., behavioral modeling in the early design stages, RTL and gate-level modeling later in the design cycle).

It is impractical to attempt to develop a single tool that on its own models every conceivable computing system, at every conceivable level of detail. Instead, it is desirable to be able to use existing tools for different abstraction levels, in a coherent integrated design system.

1.1 Proposal

One approach to enable the construction of such integrated power estimation toolchains is to *standardize the quantities / information of relevance to power and performance analysis, that can be extracted* from a design (hardware or executable benchmark) by a tool, along with a *measure of accuracy of these tool-reported estimates*. For example, in addition to, or including its standard parameters, a tool might take as input a standardized tuple of `(ambient_temperature, cycle_time)`, and might report as its output the tuple `(50E-3, 1%)`, corresponding, for example, to a power estimation report (averaged over some time window) of 50 mW and an associated accuracy of that value to be within 1% of hardware. This value could correspond to either, say, the reported leakage power of a temperature-aware leakage estimation tool, which ignores the supplied cycle time parameter, or to the reported average power consumption for some window of time on an instruction-set simulator, with the simulated processor's cycle time as given. These are simple illustrative examples of the potential uses of the interface. Such an interface, alongside the standardization of configuration parameters that can be provided to a tool (e.g., operating voltage, ambient temperature), enables the composition of tools conforming to the interface, into a system-level framework.

As a further example, consider a microarchitectural simulator that is attempting to perform accurate cache power estimation. Using the standard interfaces, it may interact with an RTL-level simulation of the cache, which might in turn query a gate-level simulator for detailed power estimates of sub-components of the cache.

2 Power Estimation Quantities

In order to enable the interchange and querying of quantities from tools, a canonical or reference set of quantities which may be reported by a tool must

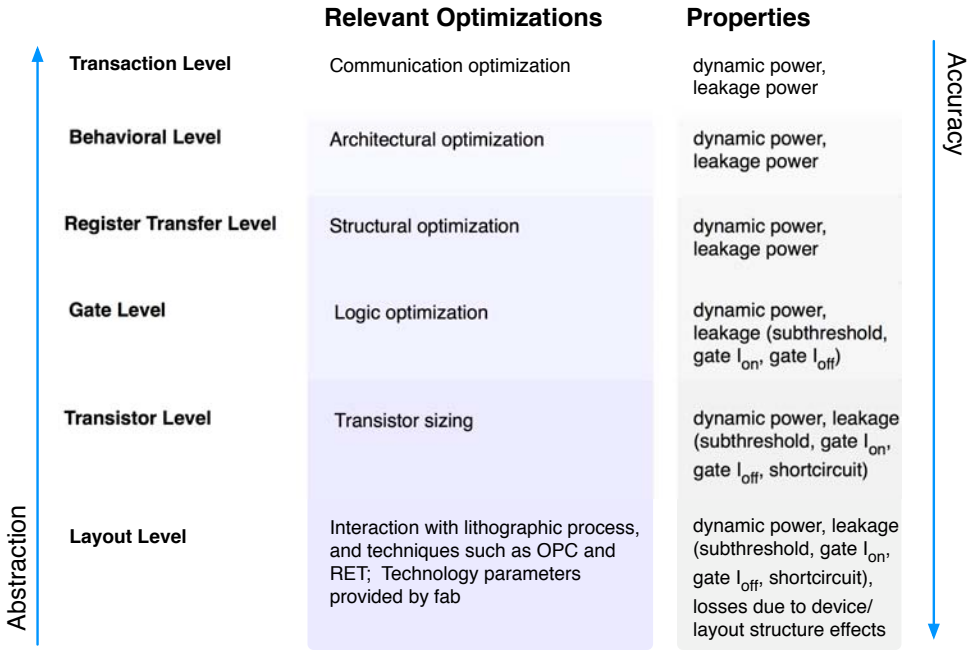

Fig. 1. Abstraction layers, and examples of the associated possible optimizations and relevant system properties.

be defined, for each of the layers of abstraction, as illustrated in Figure 1. The interpretation of the properties reported will be influenced by the parameters supplied to the query. Possible parameters to a query include:

- Operating voltage
- Ambient temperature
- Current time / clock cycle
- Gate-, RTL-, or transactional-level system description

The type of power estimates supplied will vary to some extent with the level of abstraction. For example, while it makes sense to consider short-circuit power for a gate-level description, this may not be so meaningful for a behavioral-level description. In what follows, examples of properties that may be taken as input to a tool, or output therefrom, for each of the abstraction layers in Figure 1, are presented. Such input properties, which are supplied as part of a request to a tool, may be based on a given abstraction level, may in practice be the result of a system designer's knowledge, or may be the output of a lower level estimator.

For each abstraction level, an example of the *basic unit of construction* and *granularity of time* at that layer is also provided. For example, at the gate-level abstraction layer, the basic units of construction are Boolean logic gates, and

the granularity of time is a clock cycle. Every property value, regardless of the layer of abstraction, has an associated *property accuracy*, representing the tool's perception of the accuracy of its (estimated) reported values, versus their actual values in hardware. Associating such perceived accuracy values to input and output properties will make it possible to reason about the quality of estimates obtained when combining estimation tools from different abstraction levels.

2.1 Gate-level abstraction

- **Unit:** Logic gate. Granularity of time is a clock cycle.
- **Output properties:** Temperature, dynamic power, sub-threshold gate leakage, gate I_{on} leakage, gate I_{off} leakage.
- **Input properties:** Clock frequency, voltage, temperature, current time, time window for averaging, request of average versus peak value for given time window.

It is likely that transistor-level estimates of properties will be supplied as input to the gate-level tools.

2.2 RTL abstraction

- **Unit:** Functional block. Time granularity is a clock cycle.
- **Output properties:** Temperature, dynamic power, lumped leakage power.
- **Input properties:** Clock frequency, voltage, temperature, current time, time window for averaging, request of average versus peak value for given time window.

It is likely that gate-level estimates of properties will be supplied as input to the RTL tools.

2.3 Behavioral-level abstraction

The properties and level of abstraction, specifically from the point of view of power estimation, are very similar to the RTL and transaction-level case.

- **Unit:** Functional block. Time granularity is the time for a state transition, usually a clock cycle.
- **Output properties:** Temperature, dynamic power, lumped leakage power.
- **Input properties:** Clock frequency, voltage, temperature, current time, time window for averaging, request of average versus peak value for given time window.

It is likely that RTL abstraction estimates of properties will be supplied as input to the behavioral-level tools.

2.4 Transaction-level abstraction

The properties and level of abstraction, specifically from the point of view of power estimation, are very similar to the RTL and behavioral-level case.

- **Unit:** Functional block or communicating entity. Time granularity is the time between transactions.
- **Output properties:** Temperature, dynamic power, lumped leakage power.
- **Input properties:** Transaction frequency, voltage, temperature, current time, time window for averaging, request of average versus peak value for given time window.

It is likely that RTL estimates of properties will be supplied as input to the transaction-level tools.

3 Abstraction and State Synchronization

Due to the different levels of abstraction at which tools operate, what may be considered "complete state" at one layer of abstraction will necessarily contain gaps in state at a lower layer. It is thus necessary to consider ways in which these gaps in state may be filled in.

One way to fill in the gaps in state is to pass down "bundled" machine state as one of the input properties to an estimation tool. Another mechanism would involve the lower-level estimation tool filling in the missing state, for example, by simulating from a known initial state, or from a simulated checkpoint. Such checkpoints or low-level "gap-filling" may be combined with the passing-in of state to an estimation tool. Thus, for example, an instruction-set simulator which models a pipelined microarchitecture (behavioral/RTL) may provide state such as the contents of the register file, memory read/write ports, bus state and the non-decoded portions of the pipeline latches to an RTL simulator, which will fill in the complete state of the pipeline latches and other control structures from an RTL checkpoint. This RTL tool may provide the state at the input to a functional unit to a gate-level tool for modeling that unit, and that gate-level tool may then re-create a valid state for the internals of the functional unit from scratch.

These interactions between estimation tools at the various levels of abstraction, the state they create, maintain, checkpoint and exchange, is illustrated in Figure 2.

4 Related Work

Several simulation environments currently provide interfaces for interconnection of components. Examples include the interfaces of the UNISIM simulation environment [1], and the plug-in application programming interface (API) for the Code Composer Studio (CCS) tools from Texas Instruments [2].

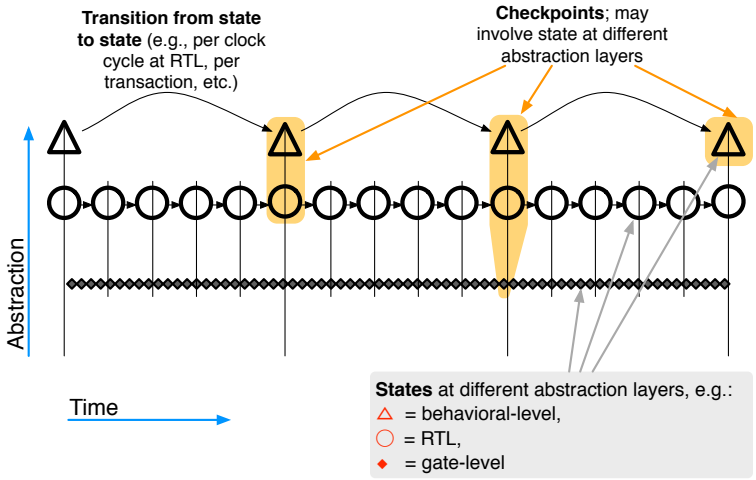

Fig. 2. Interactions between estimation tools at the various levels of abstraction, the state they create, maintain, checkpoint, and exchange.

The necessity of common formats for information interchange between hardware design tools has recently been recognized by several commercial vendors. In particular, several formats for exchange of power analysis information, such as the *Unified Power Format (UPF)* [3], shepherded by Accellera, and the *Common Power format (CPF)* [4] from Cadence Design Systems are in the process of being adopted or standardized. These interchange formats appear to be targeted primarily at computer-aided design methodologies for low power, focusing on issues such as consistent definition and semantics of power modes and the associated behavior of hardware under these modes of operation. While it is possible that they may prove to be the appropriate solutions for the challenges discussed in this paper, it is not yet clear whether this is indeed the case.

It will be necessary to maintain a uniform notion or representation of time across tools, especially if their interaction is going to be dynamic, e.g., during simulation, as opposed to static design-time interactions. Similar problems have been tackled in the area of parallel discrete event simulation [5], and the techniques employed therein might be of relevance.

References

1. UNISIM: UNIted SIMulation environment. (`http://www.unisim.org/`, accessed January 2007)
2. Texas Instruments Inc.: Software Developer's Kit Version 2.0 User's Guide. (2001)
3. Accellera: Unified Power Format (UPF). (`http://www.accellera.org/activities/upf`, accessed April 2007)
4. Si2: Common Power Format (CPF). (`http://www.si2.org/`, accessed April 2007)
5. Fujimoto, R.M.: Parallel discrete event simulation. Commun. ACM **33** (1990) 30–53