An Energy-Aware Architectural Exploration Tool for ARM-Based SOCs

Dan Crisu, Sorin Cotofana, and Stamatis Vassiliadis
Computer Engineering Laboratory
Electrical Engineering Department
Delft University of Technology
2628 CD Delft, The Netherlands
Phone: +31 (0)15 2783 644   Fax: +31 (0)15 2784 898
Email: {dan, sorin, stamatis}@ce.et.tudelft.nl

Abstract— **In recent years, power consumption has become a critical concern for many VLSI systems. Whereas several case studies demonstrate that technology-, layout-, and gate-level techniques offer power savings of a factor of two or less, architecture and system-level optimization can often result in orders of magnitude lower power consumption. Therefore, the energy-efficient design of portable, battery-powered systems demands an early assessment, i.e., at the algorithmic and architectural levels, of the power consumption of the applications they target. Addressing this issue, we developed an energy-aware architectural design exploration and analysis tool for ARM based system-on-chip designs. The tool integrates the behavior and energy models of several user-defined, custom processing units as an extension to the cycle-accurate instruction-level simulator for the ARM low-power processor family, called the ARMulator. The models we implemented take into account the particular class, e.g., datapath, memory, control, or interconnect, as well as the architectural complexity of the hardware unit involved and the signal activity triggered by the specific algorithm executed on the ARM processor. Our tool can estimate at the architectural level of detail the overall energy consumption or can report the energy breakdown among different units. Preliminary experiments indicated that the estimation accuracy is within 25% of what can be accomplished after a circuit-level simulation on the laid-out chip.**

Keywords— **ARM CPU core; system-on-chip; ARMulator; energy-aware architectural exploration; battery-powered system.**

I. Introduction

With the advent of mobile platforms for computing and communications, system designers and integrators were confronted with a massive shortage of tools that enable early energy consumption estimation for such systems. CAD tool support for embedded system design is still limited and it addresses mainly functional verification and performance estimation.

The intricacy involved by these new electronic appliances imposed a new design paradigm to cope with the specific requirements, e.g., low cost with fast time to market, and restrictions they have. Also, energy consumption is a critical factor in system-level design of embedded portable appliances. A hardware-software co-design framework must be employed to proceed with the design from the software applications intended to run on these appliances to the final specifications of the hardware that implements the desired functionality given the above-mentioned constraints. Studies have demonstrated that circuit- and gate-level techniques have less than a 2x impact on power, while architecture- and algorithm-level strategies offer savings of 10–100x or more [1]. Hence, the greatest benefits are derived by trying to assess early in the design process the merits of the potential implementation. Architecture optimization corresponds to searching for the best design that optimize all objectives. Since the optimization problem involves multiple criteria (power consumption, throughput, and cost) to reach the global optimum a set of Pareto points [2] in the design space have to be found. A Pareto point corresponds to a global optimum in a mono-dimensional design evaluation space and, summing over the entire design space, the trade-off curves or surfaces are obtained. Ideally, when designing an embedded system, a designer would like to explore a number of architectural alternatives and test functionality, energy consumption, and performance without the need to build a prototype first.

Usually, typical portable systems are built of commodity components and have a microprocessor-based architecture. Full system evaluation is often done on prototype boards resulting in long design times. Power consumption estimation can be done only late in the design process, after the prototype board was built, resulting in slow power tuning turnarounds that doesn't meet the requirement of fast time to market. On the other hand, using field programmable gate array (FPGA) hardware emulators for functional debugging, with a fast prototyping time, can neither give accurate estimates of energy consumption nor of the performance.

Among the tools preferred for early performance assessment at the algorithmic and architectural level, in the last decade, were the cycle-accurate instruction-set simulators. Unfortunately, for power consumption estimation this approach was seldom easy to follow. There were only a few academic tools for power estimation (all based on or integrated in the SimpleScalar instruction set simulator toolset framework [3], [4], [5]) and almost no commercial products.

For several target general purpose processors a number of techniques emerged in the last few years. The processor energy consumption for an instruction trace was generally estimated by instruction-level power analysis [6], [7]. This technique estimates the energy consumed by a program by summing the energy consumed by the execution of each instruction. Instruction-by-instruction energy costs, together with non-ideal effects, are precharacterized once for each target processor. A few research prototype tools that estimate the energy consumption of processor core, caches, and main memory have been proposed [8], [9]. Memory energy consumption is estimated using cost-per-access models. Processor execution traces are used to drive memory models, thereby neglecting the non-negligible impact of a non ideal memory system on program execution. The main limitation of these approaches is that the interaction between memory system (or I/O peripherals) and processor is not modeled. Cycle-accurate register-transfer level energy estimation was proposed in [4]. The tool integrates RT level processor simulator with DineroIII cache simulator and memory model. It was shown to be within 15% of HSPICE simulations.

The drawback of all the above methods to estimate the power consumption is that they are based on certain architectural templates, i.e., general purpose processors and can be hardly adapted to model system-on-chip designs.

A new approach towards high-level power estimation is presented in this paper in the context of ARMulator [10], a cycle-accurate instruction-level simulator for the ARM low-power processor family. More in particular, we developed an energy-aware architectural design exploration and analysis tool for ARM based system-on-chip designs. The tool integrates the behavior and energy models of several user-defined, custom processing units as an extension to the ARMulator. These models take into account the impact of design complexity and signal activity on datapath, memory, control, and interconnect power consumption. So far we have implemented only the tool framework and the power calculators for the datapath part. Experiments carried on employing a toy coprocessor design indicated that the accuracy of the results obtained by behavioral simulation is within 25% of that obtained using circuit simulators.

The rest of the paper is organized as follows. We discuss for background purposes the power models in Section II. System model and the methodology for cycle-accurate simulation of energy dissipation are presented in Section III. In Section IV, to validate the employed methodology we design down to the physical layout a toy coprocessor for an ARM1020T CPU core in order to run a number of realistic experiments. Finally, Section V presents the conclusions and describes future work in the area.

II. Background

A comprehensive collection of techniques for analyzing the power consumption of chips at the architecture or RT level of abstraction is presented in [1]. The premise for the success of such methodology consists in the existence of a library of hardware cells (various operators for datapath part, gates for control logic, and bit-cells, decoders, sense amplifiers for memory cores) specified at the layout-level. Once such a library exists, it can be precharacterized resulting in a table of effective capacitive coefficients for every element in the library. Then using only this tables and the activity statistics derived during the architectural-level simulation the power consumption can be estimated easily. This precharacterization has to be done only once and only the effective capacitive coefficients table are needed for power estimation. The precharacterization results are valid only for a specific library of hardware cells and a given IC technology.

The power is analyzed separately for the four main classes of chip components: datapath, memory, control, and interconnect. For the first two classes, a model called the Dual Bit Type (or DBT) model is developed and it demonstrated good results, with power estimates typically within 10-15% of results from switch-level simulations. The DBT model achieves its high accuracy by carefully modeling both physical capacitance and circuit activity. The key concept behind the technique is to model the activity of the most significant (sign) bits and least significant bits separately. The DBT model applies only to parts of the chip that manipulate data. A separate model is introduced to handle power estimation for control logic and

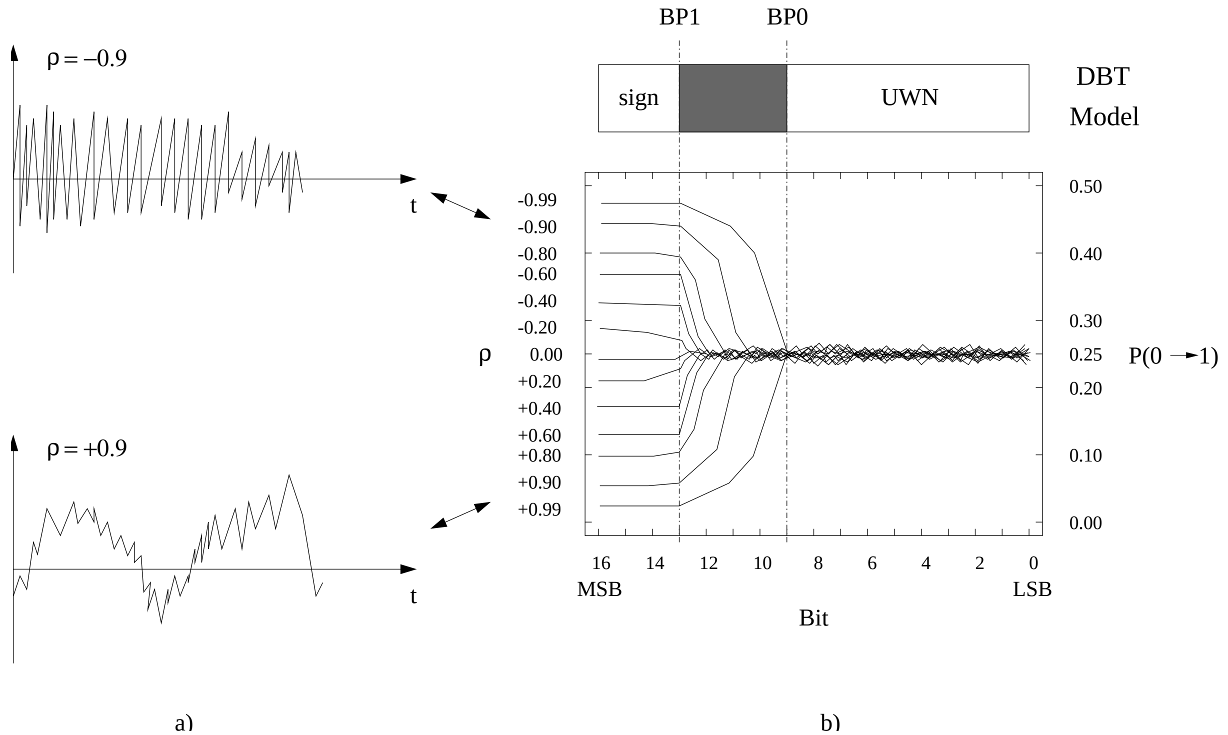

Fig. 1. a) Activity for positively and negatively correlated waveforms. b)Bit transition activity for data streams with varying temporal correlation.

signals. This model is called the Activity-Based Control (ABC) model. The method relies on the observation that although the implementation style of the controller (e.g., ROM, PLA, random logic, etc.) can heavily impact the power consumption, it is still possible to identify a number of fundamental parameters that influence the power consumption regardless of the implementation method. In a chip, datapath, memory, and control blocks are joined together by an interconnect network. The wires comprising the network have capacitance associated with them and, therefore, driving data and control signals across this network consumes power. The precise amount of power consumed depends on the activity of the signals being transferred, as well as the physical capacitance of the wires. The DBT and ABC models provide the activity information for control and data buses, but the physical capacitance depends on the average length of the wires in each part of the design hierarchy.

The DBT model accounts for two classes of input bits (this is the reason why it is called the Dual Bit Type model) and is presented in Figure 1. The DBT model reflects the distinct behaviors of both the LSB's and the MSB's of a data word. The LSB's are modeled as uniform white noise (UWN), while the MSB's account for the sign extension (if any) in the chosen data representation. In Figure 1.b the bit transition activity for several different two's complement data streams are shown. In particular, each curve corresponds to a Gaussian process with a different temporal correlation:

$$\rho = \frac{cov(X_{t-1}, X_t)}{\sigma^2} \tag{1}$$

where X_{t-1} and X_t are successive samples of the process and σ^2 is the variance. The position of the model breakpoints ($BP1$ and $BP0$) depends on how many bits are required to represent the numbers in the data stream; the unused bits are devoted to sign. The analytical expressions for the breakpoint positions of a stationary data stream as a function of word-level statistics such as mean (μ), variance (σ^2), and correlation (ρ) are:

$$BP1 = \log_2(|\mu| + 3\sigma) \tag{2}$$

$$BP0 = \log_2 \sigma + \Delta BP0 \tag{3}$$

$$\Delta BP0 = \log_2\left(\sqrt{1-\rho^2} + |\rho|/8\right) \tag{4}$$

Altering the distribution of the process merely introduces a negligible error in the model.

Since the sign bit activity is so different from the UWN bit activity, the capacitance model should account for both types of bits. Hence there should also

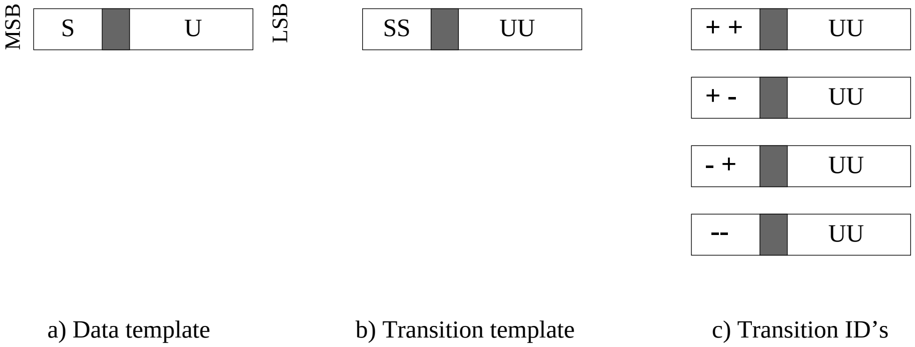

Fig. 2. Templates for identifying data bit types.

be capacitive coefficients for different sign bit transitions beside the capacitive coefficient for the UWN bits. Different templates can be associated with any data stream. The templates classifies bits in the data stream as either U or S for UWN and sign bits, respectively. The templates are presented in Figure 2.

For the transition template the **SS** indicates a transition from one sign value to another (possibly equal) value. Similarly, **UU** suggests that the UWN bits all transition from their current random values to some new random values. A transition template does not imply any particular sign value, however, a transition ID can be used to denote a specific sign transition.

These formal representations serve two purposes in our case. First, they help us to build the structure and the content of the effective capacitance coefficient table for every module existent in a library of hardware cells. This process actually needs to be done only once per library. Second, they will help us to estimate the power consumption of a module based on the transition activity seen on its input terminals given its associate effective capacitance coefficient table.

III. Proposed Design Exploration Framework

In this section we present our approach to estimate at the architectural level the power consumption of a coprocessor or peripheral unit coupled with an ARM CPU core on a system-on-chip.

A. System Model

Figure 3 gives an overview of the power analysis strategy that we propose.

We use this strategy for the design of peripheral units that augment or complement ARM CPU core functionality. The instruction set architecture of the ARM family of processors offers room for extensions to be added by providing the so called coprocessor instructions. Referring to the Figure 3, the inputs from the user are a description of a candidate architecture for the desired peripheral unit given in behavioral or structural VHDL and the set of data and the application program for which a power analysis is desired. The provided program is then compiled using the ARM native compiler. Usually, the code will embed, beside ARM native instructions, specific peripheral unit instructions. These specific instructions, when executed on the ARMulator (the ARM instruction set simulator), will be recognized as non-native or coprocessor instructions and they will trigger callback functions, installed using the ARMulator API (application programming interface), so specific actions (e.g., new data or commands are fed to the hardware description simulated in VHDL) can be taken. Moreover, every clock cycle, the ARMulator will sent signals to the VHDL simulator to advance the state of the simulated hardware description one more clock cycle. In this simple way the simulated hardware description will process its own data in lockstep with the ARM processor pipeline. Every clock cycle the activity on internal relevant signals is also collected and sent to the power analysis units. Rather than attempting to find a single power model for the entire chip, we take the approach of identifying four basic classes of components: datapath, memory, control, and interconnect. The total power consumption of the coprocessor or peripheral unit per program executed on the ARM processor is estimated.

The central elements of our architectural design exploration framework are:

- ARMulator, the cycle-accurate instruction set simulator for the ARM family of low-power processors;
- VHDL simulator, capable of saving the state of the simulated hardware description whenever it receives this command, also it is capable to reinitialize the hardware description with the previous saved state

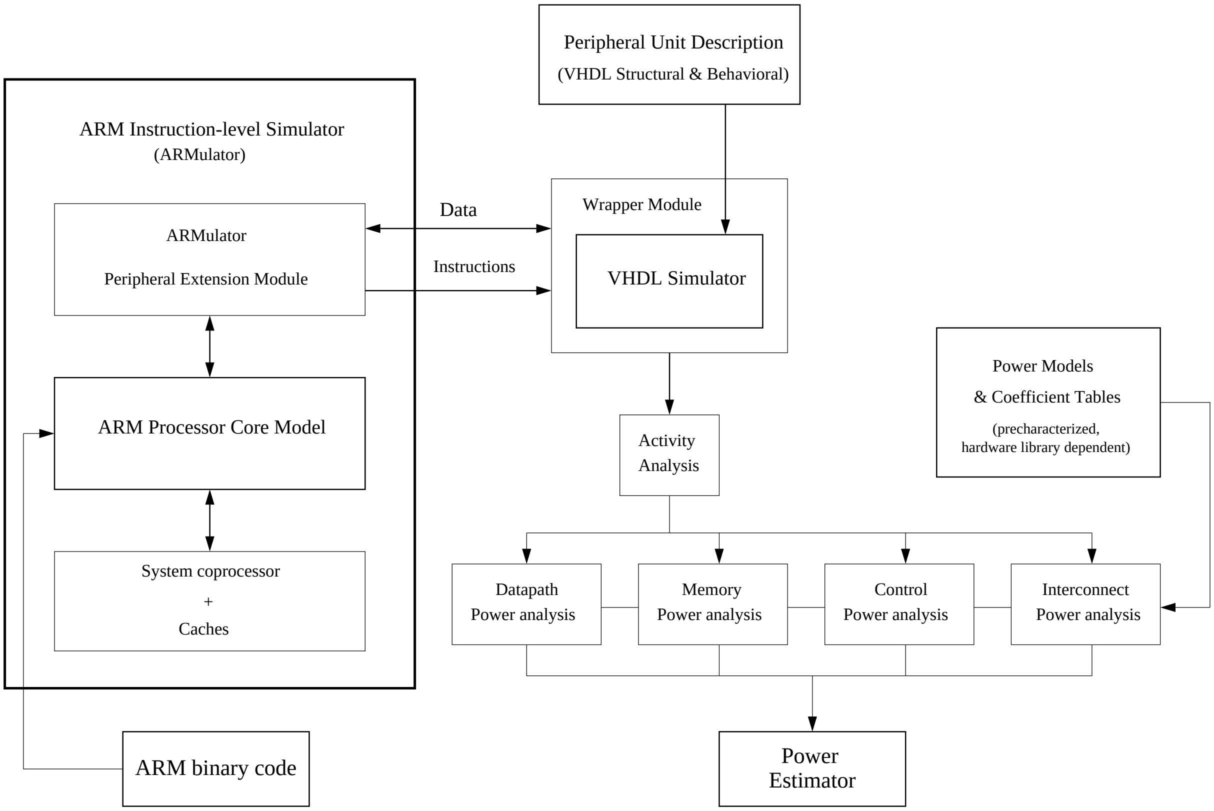

Fig. 3. System-on-chip simulator architecture.

before it processes the new stimuli sent by the ARM CPU core simulated on the ARMulator;
- Wrapper Module, that handles the communication between ARMulator (using ARMulator API) and the VHDL simulator; it is also responsible of passing correct formatted data between ARMulator, the VHDL simulator, and the activity analysis module;
- Precharacterized Power models and Effective Capacitance Coefficient Tables Module, that contain for a library of hardware cells all the technology dependent information required by the power analysis modules to compute the power consumption; the tables are derived only once for a given library of hardware cells (more detailed explanations are given in Subsection III-B);
- Activity Analysis Module, that feeds the Power Analysis modules (power calculators) with statistics about signal activity inside the simulated hardware description;
- Power Analysis Modules, that estimate the power consumption in the datapath, control, memory, and interconnect based on statistics received from the Activity Analysis Module and lookups in the effective capacitance coefficient tables;
- Power Estimator Module, that adds the estimates of power consumption of datapath, control, memory, and interconnect and offers the total figure of power consumption of the coprocessor or peripheral unit per program executed on the ARM processor;

The approach we have taken provide all the benefits of a co-design framework, moreover, it is also capable of power estimation:
- permits experimental partitioning schemes between features that must be provided by software and features that will be mapped in hardware;
- allows changing of the organization and order of execution of the algorithmic blocks to investigate and verify potential new architectures;
- provides methods of performance monitoring (in terms of throughput, and power consumption);
- accelerates the implementation of new algorithms and provides an environment in which to test them both individually and as part of an entire pipeline;
- allows tweaking the bit width precision and seeing potential impacts on result accuracy and performance factors.

B. Power Models of Datapath

In this subsection, we will describe the methodology of modeling power consumption of the datapath at

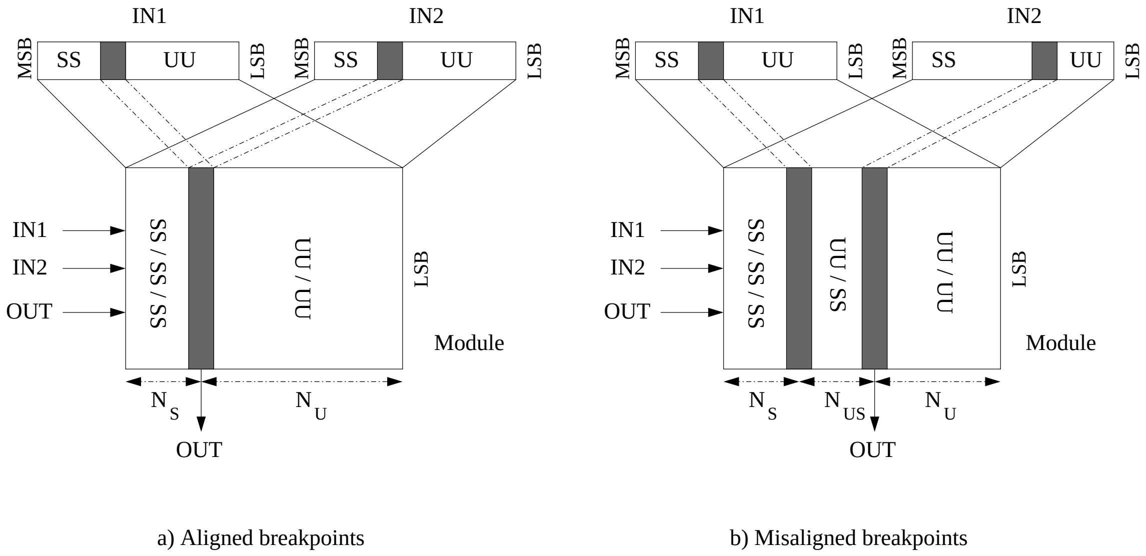

Fig. 4. Transition templates for two-input modules.

the architecture level. We followed the methodology presented in [1].

Having the library of hardware cells at the layout-level and a circuit-accurate simulator we will precharacterize first the library. During the precharacterization stage of the library of hardware cells a black-box model of the capacitance switched in each module for various types of inputs is produced. If desired, these capacitance estimates can be converted to an equivalent energy, $E = CV^2$, or power, $P = CV^2f$. The black-box capacitance models can be parameterized, i.e., taking into account the size or complexity of the module. The model accurately accounts for activity as well as physical capacitance. As a result, the effect of the input statistics on module power consumption is reflected in the estimates.

The DBT method accurately accounts for different types of input statistics by having a separate coefficient for each possible transition of both the UWN and the sign bits. For example, consider a bit-sliced subtractor as shown in Figure 4. The number of coefficients that are required will depend on how many inputs the module has. A two-input module, on the other hand, must be characterized for transitions on more than one data stream. For this case, the LSB region sees random (UWN) transitions on both inputs. The transition ID for this region of the module is written **UU/UU**. The effective capacitance of the module in this region will be described by the coefficient $C_{UU/UU}$. In the sign region, the module transition template has three components (**SS/SS/SS**) rather than two that might be expected. This is because the output sign can affect the capacitance switched in the module, and for some modules the sign of the inputs does not completely determine the output sign (in the case of the subtractor, subtracting two positive numbers could produce either a positive or a negative result). If the transition template for the module only includes inputs, then a particular transition might be classified as ++/++. This makes it appear as though the module undergoes no activity even though it is possible that the output does make a sign transition. Specifying the output sign (e.g. ++/++/+-) avoids this ambiguity. For this reason, the transition template for the sign region of a two-input module should contain an entry for the output, as well as the inputs. Considering all combinations for the transition ID's leads to the capacitance table presented in Table I. The table is valid only for a particular width of the datapath. The procedure for the generation of the effective capacitive coefficients table has two steps. During the pattern generation step, input patterns for various UWN, sign, and control transitions are generated for the module being characterized. In the next step, circuit simulation employing HSPICE circuit simulator is used to measure the capacitance switched for these input activity patterns. For more details on the generation process of the effective capacitive coefficients table the reader is referred to [1].

Intuitively, the total power consumed by a module should be a function of its complexity, i.e., size. This reflects the fact that larger modules contain more circuitry and, therefore, more physical capacitance. Under the DBT model, we can specify arbitrary capaci-

	Transition Templates	Capacitive Coefficients				
	UU/UU	$C_{UU/UU}$				1
	SS/SS/SS	$C_{++/++/++}$	$C_{++/++/+-}$	$C_{++/++/-+}$	$C_{++/++/--}$	64
		$C_{++/+-/++}$	$C_{++/+-/+-}$	$C_{++/+-/-+}$	$C_{++/+-/--}$	
		. . .	. . .	. . .	. . .	
		$C_{--/-+/++}$	$C_{--/-+/+-}$	$C_{--/-+/-+}$	$C_{--/-+/--}$	
		$C_{--/--/++}$	$C_{--/--/+-}$	$C_{--/--/-+}$	$C_{--/--/--}$	
Misaligned breakpoints only	UU/SS	$C_{UU/++}$	$C_{UU/+-}$	$C_{UU/-+}$	$C_{UU/--}$	8
	SS/UU	$C_{++/UU}$	$C_{+-/UU}$	$C_{-+/UU}$	$C_{--/UU}$	

TABLE I
Capacitive coefficients for two-input modules.

tance models for each module in the datapath library. For example, consider modeling the capacitance of a ripple-carry subtractor. The amount of circuitry and physical capacitance an instance of this subtractor will contain is determined by its word length, N. In particular, an N-bit subtractor can be realized by N one-bit full-subtractor cells. The total module effective capacitance function should, therefore, receive an argument proportional to the word length as shown here:

$$C_T = f\left(activity\_statistics, C_{eff}N\right) \qquad (5)$$

where C_{eff} is the average capacitive coefficients per bit table, f represents the total module effective capacitance function, and C_T represents the total module effective capacitance. The average capacitive coefficients per bit table is obtained after a process of data fitting (using least-squares approximation method) employing the effective capacitive coefficients tables generated for several sample width of the datapath (e.g., 4 bits, 8 bits, 16 bits, 32 bits). The average capacitive coefficients per bit table will be generated for the subtractor during the precharacterization stage and stored for the time when the estimate of the power consumption is needed. Many modules besides the subtractor also follow a simple linear model for the argument of the total module effective capacitance function. For example, ripple-carry adders, comparators, buffers, multiplexers, and Boolean logic elements all obey Equation (5). The DBT method is not restricted to linear capacitance models and can model non-linear modules like array multipliers and logarithmic shifters. The total module effective capacitance function f is actually the power model of the module under consideration and receives the activity statistics seen on the module terminals, the complexity parameters of that module (e.g., N), and a pointer to the average capacitive coefficients per bit table for that module. The reader is again referred to [1] for more details. The total module effective capacitance C_T represents the effective capacitance switched by that module every clock cycle during the execution of an application program on the ARM processor.

We have to mention that, up to date, the precharacterization of the library is done in a manual way. In the future, we intend to develop an automatic process of precharacterization. We believe that a command language with a dedicated grammar for the precharacterization process can be a possible solution. Within this approach every leaf cell in the library of hardware cells will be accompanied by a description file with specific commands for the precharacterization program.

IV. Experimental Results

To verify the power consumption prediction accuracy of the architectural space exploration tool we designed, we need to provide the following experimental setup: a precharacterized library of hardware cells, the description in VHDL of the peripheral or coprocessor to be simulated, and the binary code of the program to be simulated on the ARMulator containing calls to the coprocessor.

We precharacterized parts of a datapath library of cells (including a ripple-carry subtractor) designed in

Transition Templates	Capacitive Coefficients (fF/bit)															
UU / UU	35															
UU / SS	34	51	50	17.26												
SS / UU	0	28.3	41	29.5												
SS/SS	0	28.3	41	3.15	42	0	18.61	0	52	25.7	0	0	0.5	0	0	0
	0	35.4	0	52.1	16	38	36	16	0	56	0	0	49	27	0	0
	0	0	69	60	0	0	43	0	7.65	32.73	49	10	46.1	0	19.3	0
	0	0	0	2.4	0	0	54.9	72	0	0.15	0	47	0.75	25	22.8	2.8

TABLE II

Average capacitive coefficients per bit for the ripple-carry subtractor

UMC 0.18μm Logic 1.8V/3.3V 1P6M GENERICII CMOS technology. We extracted from the layout the circuit of the subtractor in three variants of the datapath width: for 4 bits, 8 bits, and 16 bits. After using the method presented in Subsection III-B we obtained the average capacitive coefficients per bit table presented in Table II.

We modeled in VHDL a toy coprocessor for an ARM1020T CPU core. It was designed starting from the datasheet of AMD's Am2901 four-bit bipolar microprocessor slice. The coprocessor has a datapath width of 8 bits. The organization of the toy coprocessor is presented in Figure 5. The coprocessor consists of a 16-word by 8-bit two-port register file, an ALU and the associated shifting, decoding and multiplexing circuitry. The 9-bit microinstruction word is organized in three groups of three bits each and selects the ALU source operands, the ALU function, and the ALU destination register. The ALU provides various status flag outputs. The ALU is capable of performing three binary arithmetic ($R+S$, $S-R$, $R-S$) and five logic functions (R OR S, R AND S, $\overline{R}$ AND S, R XOR S, R XNOR S).

To generate the application programs we analyzed real trace data for environmental control realized with well known microcontrollers (Intel 8051 and compatible). We extracted the recurrent patterns of control and data in these instruction flows and generated three instruction flows A, B, and C, along with the data using biased noise generators. We used biased noise generators because in the case of our toy coprocessor there is no already developed software available. We executed these instruction flows on the ARM processor family ISA and, using the framework described in Subsection III-A, we obtained power consumption estimates for the subtractor. They are presented in the second column of Table III.

In order to find the relative error of these estimations for the ripple-carry subtractor we have to compare the results obtained employing our design exploration tool with the power consumption estimated accurately with the HSPICE circuit simulator on exactly the same excitation patterns for the ripple-carry subtractor. For this purpose we designed down to the layout-level the toy coprocessor[1] using the library of hardware cells. The layout of the toy coprocessor is presented in Figure 6.

The simulation results on the extracted netlist of the ripple-carry subtractor are presented in the third column of Table III. We have to mention here that the circuit-level simulation of the subtractor took several hours for the three instruction traces executed on the ARM processor. This clearly indicate that a circuit-level simulation of the whole coprocessor, to obtain the power consumption directly, for an instruction trace executed on the ARM processor is computationally unfeasible. The relative error between the power estimated and the power consumption obtained by circuit-accurate simulation is presented in the last column of Table III. The power prediction accuracy is good, well within 25% of a direct circuit simulation with HSPICE.

V. Conclusions

A new approach towards high-level power estimation is presented in this paper in the context of ARMu-

[1] Actually for this simple case of the subtractor the layout for the whole coprocessor is not needed but we thought at the future extension of our power models for the memory, control, and interconnect part when we can reuse the design.

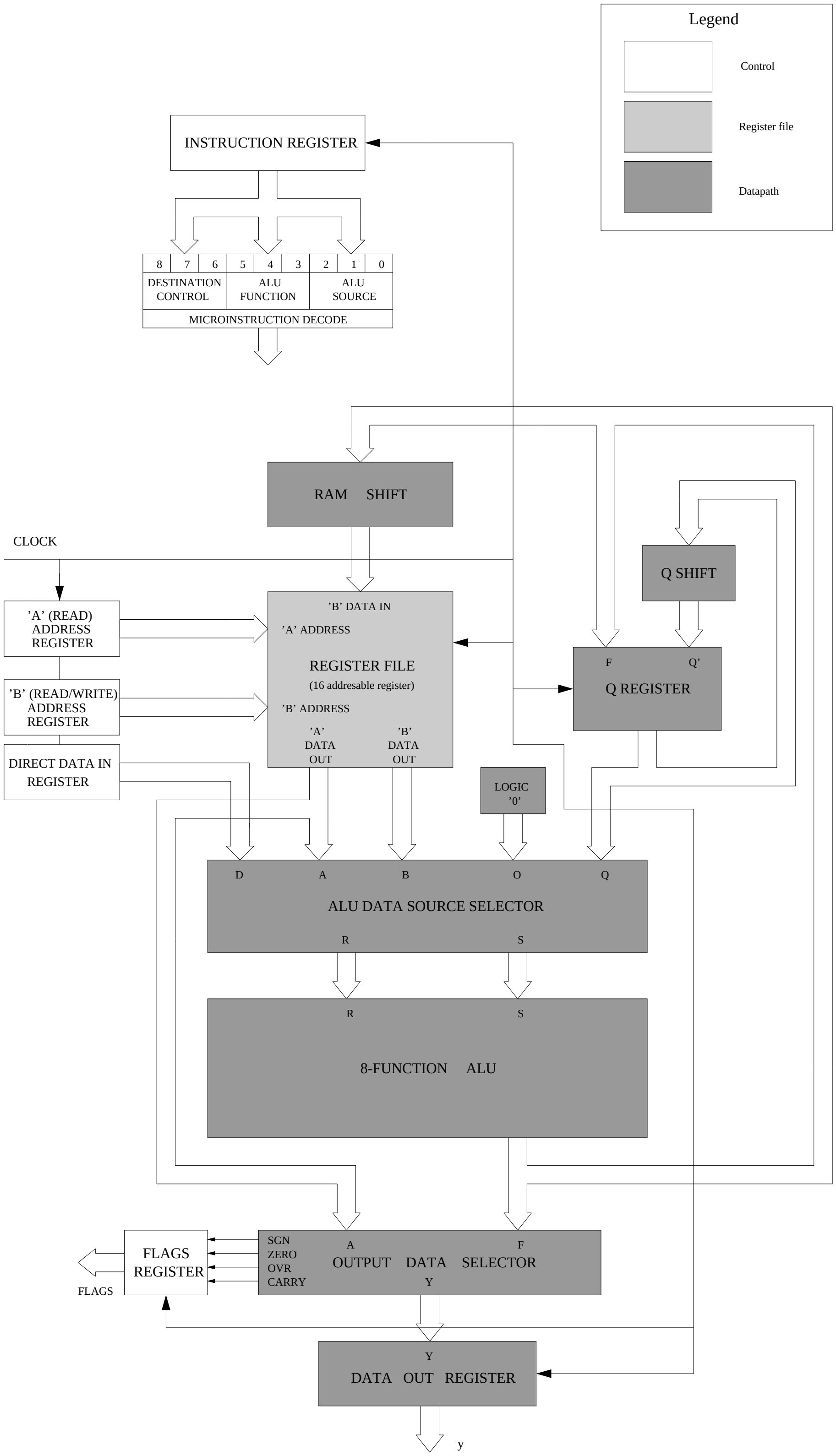

Fig. 5. Toy Coprocessor Block Diagram.

lator, a cycle-accurate instruction-level simulator for the ARM low-power processor family. More in particular, we developed an energy-aware architectural design exploration and analysis tool for ARM based system-on-chip designs. The tool integrates the behavior and energy models of several user-defined, cus-

Instruction Trace	Power Consumption (estimated)	Power Consumption (simulated)	Relative Error (%)
A	0.77 mW	0.91 mW	-15
B	1.02 mW	0.84 mW	21
C	0.63 mW	0.61 mW	3

TABLE III
Power consumption results for the ripple-carry subtractor.

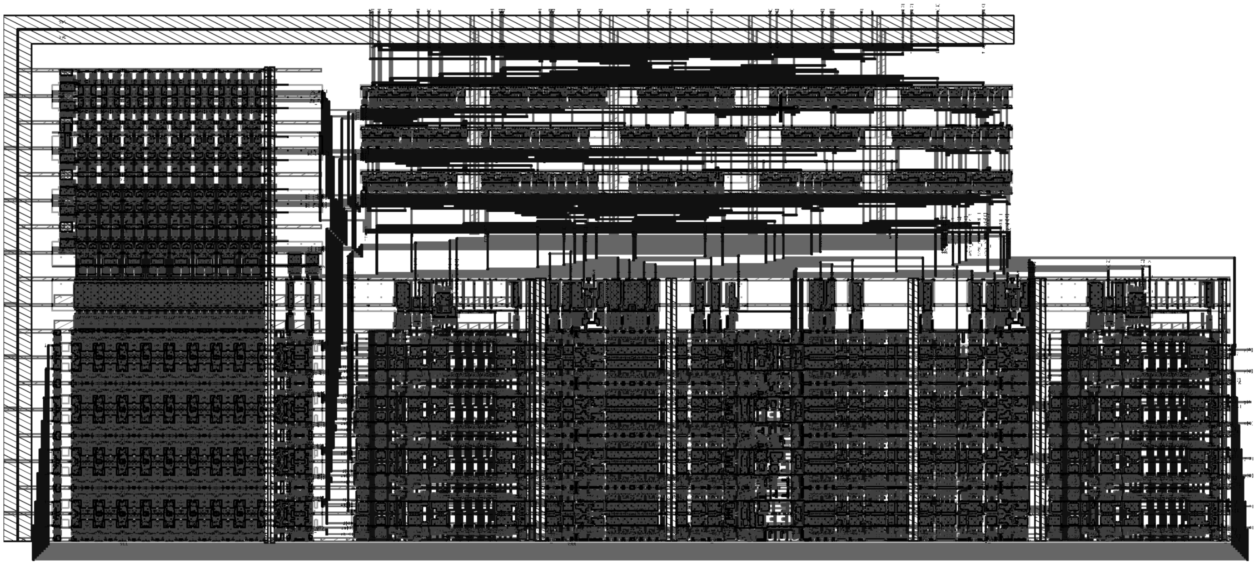

Fig. 6. Toy coprocessor layout. From left to right and up to down: register file, control, and datapath

tom processing units as an extension to the ARMulator. The models we implemented take into account the particular class, e.g., datapath, memory, control, or interconnect, as well as the architectural complexity of the hardware unit involved and the signal activity triggered by the specific algorithm executed on the ARM processor. Our tool can estimate at the architectural level of detail the overall energy consumption or can report the energy breakdown among different units. Preliminary experiments indicated that the estimation accuracy is within 25% of what can be accomplished after a circuit-level simulation on the laid-out chip.

Our endeavor to accurately predict power consumption within the ARM-based system-on-chip designs is an ongoing work. Our simulation framework has still have to be extended to take into account the power consumption in the control, memory and interconnect part of a hardware implementation. Also, an automatic process of hardware library precharacterization must to be developed to assure a fast transition from one technology to another, from one library of hardware cells to another. These issues will be addressed in future papers.

References

[1] Paul Landman, "High-Level Power Estimation," in *International Symposium on Low Power Electronics and Design*, Monterey CA, 1996, pp. 29–35.

[2] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[3] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Tech. Rep. Nr. 1342, University of Wisconsin-Madison Computer Sciences Department, June 1997.

[4] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. KIM, and W. Ye, "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower," *ISCA 2000*, 2000.

[5] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proceedings of the 27th International Symposium on Computer Architecture*, Vancouver, BC, June 2000, pp. 83–94.

[6] V. Tiwari, S. Malik, A. Wolfe, and M. Lee, "Instruction Level Power Analysis and Optimization of Software," *Journal of VLSI Signal Processing Systems*, vol. 13, no. 2–3, pp. 223–238, 1996.

[7] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 437–445, Dec. 1994.

[8] Y. Li and J. Henkel, "A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems," in *Proceedings of Design Automation Conference*, 1998, pp. 188–193.

[9] B. Kapoor, "Low Power Memory Architectures for Video Applications," in *Proceedings of 8th Great Lakes Symposium on VLSI*, 1998, pp. 2–7.

[10] ARM Limited, *ARM Developer Suite version 1.1*, 1999.