A Methodology for Minimizing Power Dissipation of Embedded Systems through Hardware/Software Partitioning

Jörg Henkel
C&C Research Laboratories
NEC USA, Princeton, NJ 08540

1 Abstract

We present a novel approach that minimizes the power dissipation of embedded core-based systems through hardware/software partitioning. Our approach is based on the idea of mapping clusters of operations/instructions to a core that yields a high utilization rate of the involved resources (ALUs, multipliers, shifters etc.) and thus minimizing power dissipation. Our approach is comprehensive since it takes into consideration the power dissipation of a whole embedded system comprising a microprocessor core, application specific (ASIC) core(s), cache cores and a memory core. We report high reductions of power dissipation between 35% and 94% at the cost of a relatively small additional hardware overhead of less than 16k cells while maintaining or even slightly increasing the performance compared to the initial design.

2 Introduction

Minimizing power dissipation of embedded systems is a crucial task. One reason is that a high power dissipation may destroy integrated circuits through overheating. Another reason is that mobile computing devices (like cell phones, PDAs, digital cameras etc.) draw their current from batteries, thus limiting the amount of energy that can be dissipated between two re-charging phases. Hence, minimizing the power dissipation of those systems means to increase the devices' "mobility" – an important factor for a purchase decision of such device. State-of-the-art design methodology is core-based system design: the designer composes a system of cores i.e. system components like, for example, an MPEG encoder engine, a μP core, peripherals etc. But still, the designer has a high degree of freedom to optimize their design according to the related design constraints since cores are available in different forms: "*hard*" or "*soft*"[1]. Whereas in case of a hard core all design steps down to layout and routing have already been completed, a soft core is highly flexible since it is a structural or even behavioral description of the core's functionality. Hence, after purchasing a soft core, the designer may decide whether to implement a core's functionality completely as a software program (running on a μP core) or as a hard-wired hardware (ASIC core) or he may partition the core between those (hardware and software) parts.

In this paper we present the first approach that deploys a HW/SW partitioning methodology to minimize the power dissipation of a *whole* system (μP core *and* instruction cache *and* data cache *and* main memory *and* application specific cores like, for example, MPEG, FFT etc. However, here we focus on the low power HW/SW partitioning method only but use our framework to estimate and optimize the power dissipation of the other cores that are not subject to HW/SW partitioning (like main memory, caches, etc.). But note that those other cores have to be adapted efficiently (e.g. size of memory, size of caches, cache policy etc.) according to the particular HW/SW partitioning chosen.[2]

3 Related Work

Hardware/software partitioning is a well-established design methodology in order to increase the performance of a system in approaches like [2, 3, 4]. These and many other approaches' objective is to meet performance constraints while keeping the system costs (e.g. total chip area) as low as possible. No power related optimization is provided. In [6] a task-level co-design methodology is introduced that optimizes for power dissipation performance. But the procedure for task allocation is based on estimations of an *average* power dissipation of a processing element rather than assuming, for example, data-dependent power dissipation that may vary from clock cycle to clock cycle. The approach [5] uses a multiple-voltage power supply to minimize system-power dissipation. Furthermore, prominent to our approach is [7] where a software power estimation/optimization method is presented.

Our approach is the first comprehensive system-level power optimization approach that deploys hardware/software partitioning based on a fine-grained (instruction/operation-level) power estimation analysis while yielding high energy savings (35% to 94%).

[1] For more details see [1].

[2] This is because — in case of the cache — the access pattern may change when a different HW/SW partition is used. Hence, power dissipation is likely to differ.

0-7695-0104-4/99 $10.00 © 1999 IEEE

The rest of the paper is structured as follows: the following Section 4 describes our partitioning method while Section 5 shows the design flow. Finally, Section 6 presents results and gives a conclusion.

4 Low Power Partitioning Approach

The architecture of a system we apply our methodology to, comprises: one processor core, a set of standard cores (main memory, caches etc.) and a set of application specific cores. Please note that the application specific cores are not yet defined. Also, the part of the system that is running as a program on the μP core is not yet defined.

Our goal is to partition a system (in the following we will simply talk of an *application*) between the μP core and the application specific core(s) in order to minimize the total power dissipation.

4.1 Basic Idea

During the execution of a program on a μP core different hardware resources within this core are invoked according to the instruction executed at a specific point in time. Assume, for example, an *add* instruction is executed that invokes the resources *ALU* and *Register*. A *multiply* instruction may use the resources *Multiplier* and *Register*. A *move* instruction might only use the resource *Register* etc. Conversely, we can argue: during the execution of the *add* instruction the multiplier is not used; during execution of the *move* instruction neither the *ALU* nor the *Multiplier* is used etc.[3] In case the processor does not feature the technique of gated clocks to shut down *all* non-used resources clock cycle per clock cycle[4], those non actively used resources will still dissipate energy since the according circuits continue to switch. We call the latter situation "*the circuits are not actively used*". Accordingly, "*the circuits are actively used*" in the previous situations described. For each resource r within a core we define a utilization rate as follows:

$$u_r = \frac{N^r_{act\_used}}{N_{total}} \tag{1}$$

where $N^r_{act\_used}$ is the number of cycles resource r is actively used and N_{total} is the number of all cycles it takes to execute the whole application. We define the "wasted energy" within a core i.e. the energy that is dissipated by resources during times frames where those resources are not actively used, as

$$E^{core}_{non\_act\_used} = \sum_{r=1}^{N_R} (1 - u_r) \cdot P^r_{av} \cdot T_{app} \tag{2}$$

where N_R is the number of resources within that specific core, P^r_{av} is the average power that is dissipated by the particular resource and T_{app} is the execution time of the whole application when executed entirely by this core. Minimizing the total energy dissipation can be achieved by minimizing $E^{core}_{non\_act\_used}$. Our solution is to deploy an additional core for that purpose i.e. to partition the functionality that was formerly solely performed by the original core, to a new (to be specified) application specific core and in parts to run it on the initial core such that

$$\sum_{i=1}^{N_{core}} (E^{core^i}_{non\_act\_used} + E^{core^i}_{act\_used}) \leq E^{initial\_core} \tag{3}$$

Whenever one of the cores $i, \cdots, N_{core}$ is performing, all the other cores are shut down (as far as this is possible), thus dissipating no energy. Equation 3 is most likely fulfilled when the individual resource utilization rate

$$U^{core}_R = \frac{1}{N_R} \cdot \sum_{r=1}^{N_R} u_r \tag{4}$$

of each core is as high as possible (note: in the ideal case it would be 1). We use the values U^{core}_R of all participating cores (i.e. those that are subject to partitioning) to determine whether a partition is advantageous in terms of power dissipation or not.

4.2 The Partitioning Process

Our approach is based on the idea that an application specific hardware (we call it in the following $ASIC\ core$) can — under specific circumstances — achieve a higher utilization rate U^{core}_R than a standard (programmable) processor core (in the following we refer to it as μP core) as demonstrated in the examples in the previous sections.

The Low Power Partition Process

1) Build a graph $G = \{V, E\}$
2) $C = decompose\_into\_cluster(\{V, E\})$
3) **For All** cluster $c_i \in C$
4) calculate: $E^{c_i}_{Trans,\mu Pcore \leftrightarrow ASICcore}$
5) $C = pre\_select(C, N^{c_i}_{max})$
6) **For All** $cluster\ c_i \in C$
7) **For All** $rs_i \in RS$
8) $do\_list\_schedule(c_i, rs_i)$
9) **If** $(U^{core}_R > U^{core}_{\mu P})$
10) **Then**
11) $E^{core}_R = U^{core}_R \cdot \sum_{rs_i \in RS} (P^{rs_i}_{av} \cdot N^{rs_i}_{cyc} \cdot T^{rs_i}_{cyc})$
12) $E^{core}_{\mu P} = E^{core}_{\mu P,initial} - E^{core}_{\mu P,c_i}$
13) $OF = F \cdot \frac{E^{core}_R + E^{core}_{\mu P} + E_{rest}}{E_0} + \frac{GEQ_{RS}}{GEQ_0}$
14) Synthesize a core with a min. OF value
15) Estimate energy (gate-level) and comp. energy savings

Figure 1: Pseudo code of our partitioning algorithm

[3] These are examples for demonstration purposes only. However, this might not apply in this simple form to a particular processor.

[4] This is actually the case for most today's processors deployed in embedded systems. An example is the LSI SPARCLite processor core.

Input to the partitioning process is a behavioral description of an application that is subject to a core/core partition between the ASIC core and the μP core. The following descriptions refer to the pseudo code in Fig. 1[5]. Step 1 derives a graph $G = \{V, E\}$ from that description. There, V is the set of all nodes (representing operations) and E is the set of all edges connecting them.

Using this graph representation, step 2 performs a decomposition of G in so-called *clusters*. We define a cluster as a set of operations that represents code segments like nested loops, if-then-else constructs, functions etc. Decomposition is done by structural information of the initial behavioral description solely. An important reason whether the implementation of a cluster on an ASIC core might lead to a reduction in energy dissipation is given by the additional amount of (energy dissipating) bus transfers. This is a very important issue for high-bus traffic, data-oriented applications we are focusing on. The according calculation is done in lines 3 and 4.[6] Line 5 performs a pre-selection of clusters i.e. it preserves only those clusters for a possible partitioning that are expected to yield high energy savings based on the bus traffic calculation. Here, the designer has a possibility of interaction by specifying different constraints like, for example, the total number of clusters N^c_{max} to be pre-selected. Please note that it is necessary to reduce the number of all clusters since the following steps 6 to 12 are performed for *all* remaining clusters.

In line 7 a loop is started for all *sets of resources* where the set of different resource sets RS is specified by the designer. The designer tells the partitioning algorithm how much hardware (#ALUs, #multipliers, #shifters etc.) he/she is willing to spend for the implementation of an ASIC core. The different sets specified are based on reference designs i.e. similar designs from past projects. Due to our design praxis 3 to 5 sets are given, depending on the complexity of an application. Afterwards, in line 8 a simple list schedule is performed on the current cluster in order to prepare the following step. The next is the computation of U_R^{core} (line 9). There it is tested whether a candidate cluster can yield a better utilization rate on an ASIC core or on a μP core. In case a better utilization rate is possible, a rough estimation on expected energy savings is performed (lines 11 and 12). Note that the energy estimate of the ASIC core is based on the utilization rate. For each resource rs_i of the whole sets of resources RS (as discussed above), an average power dissipation $P_{av}^{rs_i}$ is assumed[7]. $N_{cyc}^{rs_i}$ is the number of cycles resource rs_i is actively used whereas $T_{cyc}^{rs_i}$ gives the minimum cycle time the resource can run at. The energy dissipated by

[5]Please note that we do not use "{" and "}" to indicate the scope of validity of an *If* statement or a loop. Rather than that we indicate it by aligning to columns accordingly.

[6]Due to lack of space we do not provide the according algorithm here.

[7]The according data is derived by means of the CMOS6 library that is used later on for gate-level energy calculation as well.

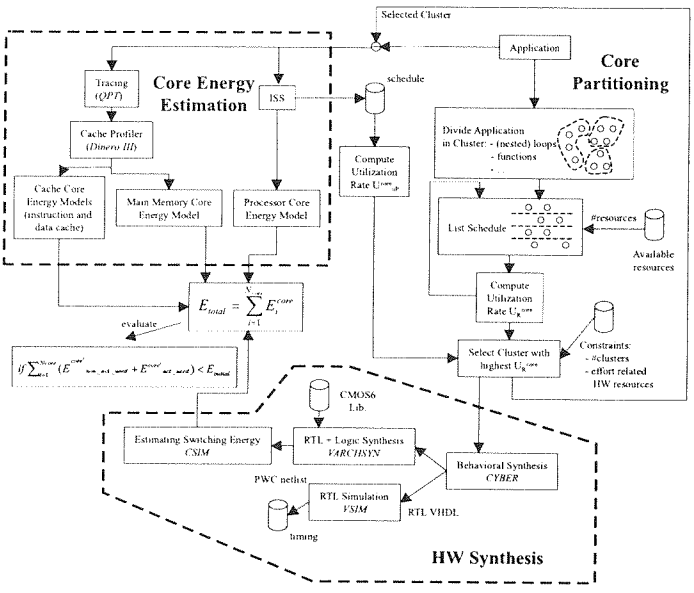

Figure 2: Design flow of our low–power hardware/software partitioning methodology

the μP core is obtained by using our instruction set energy simulation tool (see Section 5). The objective function of the partitioning process is defined as a superposition of the normalized total energy dissipation and additional hardware effort we have to spend. Please note that E_{rest} gives the energy dissipation of all remaining cores (instruction cache, data cache, main memory etc.). GEQ_{RS} is the hardware effort of a deployed resource set. E_0 and GEQ_0 are provided for the purpose of normalization only. Finally, F is a factor given by the designer to balance the objective function between energy dissipation and hardware effort. F is heavily dependent on the design constraints as well as on the application itself. For the partition that yields the best value of the objective function, the steps in lines 14 and 15 are executed: the synthesis and the following gate-level energy estimation.

5 Design Flow

Fig. 2 summarizes our methodology: an application is first divided into clusters (nested loops, functions). The clusters must be small enough to potentially yield a high U_R^{core} for a given but variable number of resources (ALUs, multipliers, shifters etc.). We use a structural-based approach that also minimizes data transfers between a potential cluster and the rest of the application. Then, for each cluster a list schedule is performed. The results of the schedule easily allow to determine the U_R^{core} value for each cluster. In the next step, one (or more clusters) are selected based on their U_R^{core} but also based on constraints (imposed costs like additional HW area, number of additional cores etc.). Once the cluster are

App.		Energy							Exec. Time [in 1000 cycles]			
		i-cache	d-cache	mem	μP core	ASIC	total	Sav%	μP core	ASIC	total	Chg%
3d	I	116.9uJ	14.26uJ	29.71uJ	566.8uJ	n/a	$727.7\mu J$	-35.21	39	n/a	39	-17.29
	P	$0.263\mu J$	$0.123\mu J$	$0.263\mu J$	$0.471mJ$	$0.308\mu J$	$471.5\mu J$		32	0.154	32	
MPG	I	$44.79mJ$	$17.98mJ$	$2.31mJ$	$74.32mJ$	n/a	$140.9mJ$	-43.20	5,167	n/a	5,167	-52.90
	P	$35.36mJ$	$13.14mJ$	$2.94mJ$	$27.14mJ$	$1.46mJ$	$80.04mJ$		1,696	737	2,433	
ckey	I	0.0	0.0	0.0	$329.0mJ$	n/a	$329.0mJ$	-76.81	169,512	n/a	169,512	-74.98
	P	0.0	0.0	0.0	$53.04mJ$	$23.47mJ$	$76.51mJ$		30	12	42	
digs	I	$11.69mJ$	$5.12mJ$	$0.49mJ$	$52.70mJ$	n/a	$70.00mJ$	-94.12	3,706	n/a	3,706	-42.64
	P	$14.27\mu J$	$2.04\mu J$	$20.43\mu J$	$46.78\mu J$	$4.03mJ$	$4.11mJ$		6	2	2	

Table 1: Results in terms of energy dissipation and execution time for both, initial (I) and partitioned (P) design.

selected, our standard in-house synthesis flow is deployed in order to determine the energy dissipated by a core through specific switching activities. The remaining part of the application (application minus selected clusters) is intended to be executed by a μP core. Therefore, it is fed into an ISS (instruction set simulator) that is attached to our core energy estimation framework[8]. Finally, the total system energy dissipation can be computed and compared to the initial (i.e. non-partitioned) design solution.

6 Experimental Results and Conclusion

We conducted experiments on four different applications using our methodology: an algorithm for computing 3D vectors of a moving image ("*3d*"), an MPEGII encoder ("*MPG*"), a complex chroma–key algorithm ("*ckey*") and a smoothing algorithm for digital images ("*digs*"). The applications range in size from about 5kB to 230kB of C code. As a μP we deploy a simulation-based model of a SPARCLite core. Analytical energy estimation models of other cores are based on parameters for a 0.8μ technology. Two rows are dedicated to each application: the initial (non-partitioned i.e. pure software solution) "I" implementation and the partitioned "P" implementation. As seen in column "Sav%", the achieved energy savings range from 35% to 94%. These large energy savings could be achieved by implementing system parts with a high utilization rates U_R^{core} as an application specific hardware rather than implementing as a software program running on an off–the–shelf processor. In addition, it is prominent to mention that those system parts that were finally implemented as an application specific hardware, accounted in all cases for the largest part of the execution time of the specific application. An important feature of our approach is that *all* system components are taken into consideration to estimate energy savings. This is because a differently partitioned system might have different access patterns to caches and main memory, thus resulting in different energy dissipations of those cores (compare according rows of columns "*i-cache*", "*d-cache*" and "*mem*"). Please also note that the total energy savings are not achieved at the cost of performance. Instead, the performance is even slightly increased (i.e. reduced number of clock cycles to execute). The additional hardware effort is in all cases reasonable (in every case less than 16,000 cells) — taken into consideration the high energy savings. In addition, we would like to stress that our core-partitioning approach yields the shown high energy savings especially in the case of DSP-oriented applications that can be found as part of economically interesting consumer products like cell phones, digital cameras etc.

[8]This framework features analytical energy estimation models for caches and main memories and an instruction-level, simulation based estimation model for a μP core. See [9].

References

[1] M. Keaton, P. Bricaud, *Reuse Methodology Manual For System–On–A–Chip Designs*, Kluwer Academic Publishers, 1998.

[2] F. Vahid, D.D. Gajski, J. Gong, *A Binary–Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning*, IEEE/ACM Proc. of The European Conference on Design Automation (EuroDAC) 1994, pp. 214–219, 1994.

[3] R.K. Gupta and G.D. Micheli, *System-level Synthesis using Reprogrammable Components*, IEEE/ACM Proc. of EDAC'92, IEEE Comp. Soc. Press, pp. 2–7, 1992.

[4] J. Henkel, R. Ernst, *A Hardware/Software Partitioner using a dynamically determined Granularity*, IEEE/ACM Proc. of 34th. Design Automation Conference (DAC) 1997, pp. 691-696, 1997.

[5] I. Hong, D. Kirovski et al., *Power Optimization of Variable Voltage Core-Based Systems*, IEEE Proc. of 35th. Design Automation Conference (DAC98), pp.176-181, 1998.

[6] B.P. Dave, G. Lakshminarayana, N.K. Jha, *COSYN: Hardware-Software Co-Synthesis of Embedded Systems*' IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.703-708, 1997.

[7] V. Tiwari, S. Malik, A. Wolfe, *Instruction Level Power Analysis and Optimization of Software*, Kluwer Academic Publishers, Journal of VLSI Signal Processing, pp. 1–18, 1996.

[8] A.W. Aho, R. Sethi and J.D. Ullmann,*COMPILERS Principles, Techniques and Tools*, Bell Telephone Laboratories, 1987.

[9] Y. Li, J. Henkel, *A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems*, IEEE/ACM Proc. of 35th. Design Automation Conference (DAC) 1998, pp. 188–193, 1998.