

A Low Power Hardware/Software Partitioning Approach for Core-based Embedded Systems

Jörg Henkel
C&C Research Laboratories, NEC USA
4 Independence Way, Princeton, NJ 08540
henkel@cctl.nec.com

Abstract We present a novel approach that minimizes the power consumption 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, ...) and thus minimizing power consumption. Our approach is comprehensive since it takes into consideration the power consumption 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 consumption 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.

1 Introduction

Minimizing power consumption of embedded systems is a crucial task. One reason is that a high power consumption 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 consumed between two re-charging phases. Hence, minimizing the power consumption of those systems means to increase the device's "mobility" – an important factor for a purchase decision of such device. Due to cost and power reduction, most of those systems are integrated onto one single chip (SOC: System-On-a-Chip). This is possible through today's feature sizes of 0.18μ that allow to integrate more than 100Mio. transistors on a single chip¹. In 2001 even larger systems of up to 400Mio. transistors may be integrated onto a single chip [2]. To cope with this complexity, state-of-the-art design methodology deployed is *core-based system design*[1]: the designer composes a system of cores i.e. system components like, for example, an MPEG encoder engine, a microprocessor core (short: μP core), peripherals etc. But still, the designer has a high degree of freedom to optimize her/his design according to the related design constraints since cores are available in different forms: as "hard", "firm" or "soft" versions (for a more detailed introduction of core-based design, please refer to [3]). Whereas in the 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 in form of a behavioral description, the designer may still decide whether to implement the core's functionality completely as a software program (running on a μP core) or as a hard-wired hardware (ASIC core). Or, the designer may partition the core's functionality between those (hardware and software) parts.

We present a novel approach that deploys a hardware/software partitioning methodology to minimize the power consumption of a whole system (μP core and instruction cache and data cache and main memory and application specific cores (ASIC cores) like, for

¹Due to the *design gap* [1] current SOCs hardly exceed 10Mio. trans. w/o mem. **Design Automation Conference** (®) Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

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 consumption of the other cores that are not subject to hw/sw partitioning (like the main memory, the 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

The rest of the paper is structured as follows: the following section gives an overview of related research while section 3 explains step by step the algorithms of our approach starting with a motivational example. Afterwards, in section 3.5 our whole design flow is introduced. Conducted experiments and obtained results are presented in section 4 while section 5 gives a conclusion.

2 Related Work

Hardware/software partitioning is a well-established design methodology with the goal to increase the performance of a system as described in approaches like [4, 5, 6, 7, 8, 9]. These and many other approaches' objective is to meet performance constraints while keeping the system cost (e.g. total chip area) as low as possible. But none of them provide power related optimization and estimation strategies.

Power optimization of software programs through analysis of energy consumption during the execution of single instructions has been conducted in [12]. In fact these basic investigations and results are one basis for our partitioning approach. In [13] the power consumption of high-performance microprocessors has been investigated and specific software synthesis algorithms have been derived to minimize for low power. The work reported in [18] deals with an architectural-oriented power minimization approach.

Algorithmic related power investigations and optimizations have been published by Ong et al. who showed that the power consumption may drastically depend on the algorithm deployed for a specific functionality. A power and performance simulation tool that can be used to conduct architecture-level optimizations has been introduced by Sato et al. [15].

In [11] a task-level co-design methodology is introduced that optimizes for power consumption and performance. The influence of caches is not taken into consideration. Furthermore, the procedure for task allocation is based on estimations for an *average* power consumption of an processing element rather than assuming, for example, data-dependent power consumption that may vary from clock cycle to clock cycle. The approach described in [10] uses a multiple-voltage power supply to minimize system-power consumption.

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 between 35% to 94%.

3 Low Power Partitioning Approach

The architecture of a system we apply our methodology to consists of a μP core, a set of standard cores (main memory, caches etc.) and a set of application specific cores. Our goal is to partition

²This is because — in case of the cache — the access pattern may change when a different hw/sw partition is used. Hence, power consumption is likely to differ.

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

3.1 Motivational Example and 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 resource *ALU* and *Register*. A *multiply* instruction uses 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 cycle⁴, those non actively used resources will still consume energy since the according circuits continue to switch. We denote to this situation as "the circuits are not actively used". Accordingly, "the circuits are actively used" when the *are* invoked at that time by an instruction. For each resource rs_i of all resources RS within a core, we define a utilization rate:

$$u_{rs} = \frac{N_{act_used}^{rs}}{N_{total}} \quad (1)$$

where $N_{act_used}^{rs}$ is the number of cycles resource rs 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 consumed by resources during times frames where those resources are not actively used, as

$$E_{non_act_used}^{core} = \sum_{rs_i \in RS} (1 - u_{rs_i}) \cdot P_{av}^{rs_i} \cdot T_{app} \quad (2)$$

where $P_{av}^{rs_i}$ is the average power that is consumed 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 consumption can be achieved by minimizing $E_{non_act_used}^{core}$. 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

$$\sum_{i=1}^{N_{core}} (E_{non_act_used}^{core^i} + E_{act_used}^{core^i}) \leq E^{initial_core} \quad (3)$$

Whenever one of the cores i, \dots, N_{core} is performing, all the other cores are shut down (as far as they are not used, of course), thus consuming *no* energy. Equation 3 is most likely fulfilled when the individual resource utilization rate

$$U_R^{core} = \frac{1}{N_R} \cdot \sum_{rs \in RS} u_{rs_i} \quad (4)$$

of each core is as high as possible (note: in the ideal case it would be 1). There, N_R gives the number of all resources that are part of that core. We use the values U_R^{core} of all participating cores (i.e. those that are subject to partitioning) to determine whether a partition of an application is advantageous in terms of power consumption or not.

At this point one could argue that we better shut down the individual resources *within* each core rather than deploying additional cores to minimize energy. This is because we suppose that a state-of-the-art core based design techniques are used as described in the introduction. This implies that the designer's task is to compose a system of cores they can buy from a vendor rather than modifying a complex core like a μP core.

Hence, our methodology allows to use core-based design techniques and minimizing energy consumption *without* modifying

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

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

complex standard cores. Whereas the previously described basic idea was formulated more general, the following implementation of our core/core partitioning algorithms⁵ is based on hardware/software partitioning between one μP core and an application specific core (ASIC core).

3.2 The Partitioning Process at a Glance

This section gives an overview of our low power partitioning approach in coarse steps. It 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_R^{core} 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 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. 1⁶. 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 *cluster*. A cluster in our definition is a set of operations which represents code segments like nested loops, if-then-else constructs, functions etc. The decomposition algorithm is not described here because it is not key to our approach. 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 consumption is given by the additional amount of (energy consuming) 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. Due to the importance, the separate section 3.3 is dedicated to that issue. 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_{max}^c 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, ...) they are 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. That step is one major part of the work presented here: 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. Due to the complexity of calculating U_R^{core} , a detailed description is given in a separate section 3.4. 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 consumption $P_{av}^{rs_i}$ is assumed⁷. $N_{cyc}^{rs_i}$ is the number of cycles resource rs_i is actively used whereas T_{cyc} gives the minimum cycle time the resource can run at. The energy consumed by the μP core is obtained by using our instruction set energy simulation tool (it will be explained in some more detail in section 3.5). The objective function OF of the partitioning process is defined as a superposition of the normalized total energy consumption and additional hardware effort we have to spend. Please note that E_{rest}

⁵Please note that we sometimes use the term *core/core partitioning* and sometimes the term *hardware/software partitioning*. Through our definition, both terms have the same meaning. But according to the specific context the one or the other term is actually used.

⁶Please 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.

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

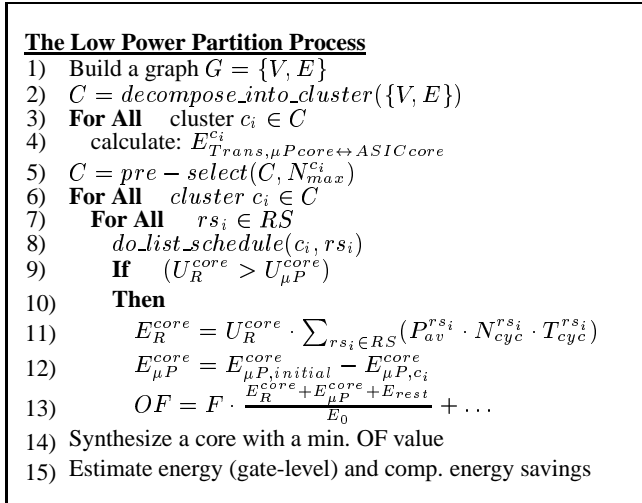


Figure 1: Pseudo code of our partitioning algorithm

gives the energy consumption of all other cores (instruction cache, data cache, main memory, bus). E_0 is provided for the purpose of normalization only. Finally, F is a factor given by the designer to balance the objective function between energy consumption and possible other design constraints. 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. These two steps are described during introduction of the whole design flow in section 3.5.

As already mentioned, the following two sections 3.3 and 3.4 are dedicated to a closer description of the pre-selection criteria for a cluster and U_R^{core} , respectively.

3.3 Determining the Pre-Selection Criteria of a Cluster

The pre-selection algorithm of clusters is based on an estimation for energy consumption of clusters due the additional traffic via the bus architecture. When a hardware/software partition of an application between a μP core and an ASIC core is deployed, the following additional bus traffic — based on the architecture shown in Fig. 2 a) where two cores communicate via a shared memory — is implied:

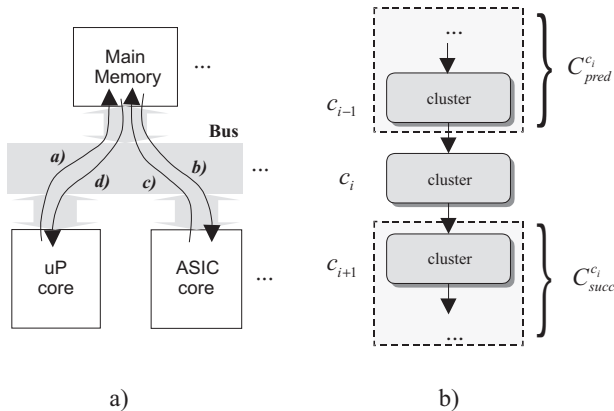


Figure 2: a) Bus transfer scheme b) nomenclature of algorithm to estimate those transfers; a cluster can either be mapped to “ μP Core” or to “ASIC Core”

a) When the μP core arrives at a point where it “calls” the

ASIC core, then it is depositing data or references to that data in the memory such that it can be accessed by the ASIC core for subsequent use.

- b) Once the ASIC core starts its operation it will access i.e. download the data or references to it from the memory.
- c) After the ASIC core has finished its job some data might be used by the μP core to continue execution. Therefore the ASIC core is depositing the according data or references to it in the main memory.
- d) Finally, the μP core reads data back from the memory.

The amount of transfers described in b) and c) occur in any case, no matter whether there is a μP core/ASIC core partitioning or not. Hence, we do not account for those in the following algorithm that is supposed to be the calculation of an *additional* (i.e. due to partitioning only) energy effort that would have to be spend. So, the pseudo code in Fig. 1 shows the algorithm that calculates the according necessary part of energy consumption due to core/core partitioning.

Computing the energy related to additional bus transfers

- 1) Number of bus transfers between μP core and memory

$$N_{Trans, \mu P core \rightarrow mem}^{c_i} = |gen[C_{pred}^{c_i}] \cap use[c_i]|$$

- 2) Take into consideration synergetic effects:

If $(implemented_in_ASIC_core(c_{i-1}))$

Then

$$N_{Trans, \mu P core \rightarrow mem}^{c_i} = N_{Trans, \mu P core \rightarrow mem}^{c_i} - |gen[c_{i-1}] \cap use[c_i]|$$

- 3) Number of bus transfers betw. ASIC core and memory

$$N_{Trans, ASIC core \rightarrow mem}^{c_i} = |gen[c_i] \cap use[C_{succ}^{c_i}]|$$

- 4) Take into consideration synergetic effects:

If $(implemented_in_ASIC_core(c_{i+1}))$

Then

$$N_{Trans, ASIC core \rightarrow mem}^{c_i} = N_{Trans, ASIC core \rightarrow mem}^{c_i} - |gen[c_i] \cap use[c_{i+1}]|$$

- 5) Total energy:

$$E_{Trans, \mu P core \leftrightarrow ASIC core}^{c_i} = (N_{Trans, \mu P core \rightarrow mem}^{c_i} + N_{Trans, ASIC core \rightarrow mem}^{c_i}) \times E_{bus\ read/write}$$

Figure 3: Pseudo code of the algorithm to calculate energy of bus-transfer in order to determine the pre-selection of cluster

The algorithm is based on the conventions shown in Fig. 2 b). There, each node represents a cluster. The arcs are indicating the direction of the control flow. The current cluster is denoted as c_i whereas the previous one is drawn as c_{i-1} and the succeeding one is given as c_{i+1} . Furthermore, we define $C_{pred}^{c_i}$ to represent *all* clusters preceding c_i . Similarly, $C_{succ}^{c_i}$ combines *all* clusters succeeding c_i . Step 1 computes the number of all transfers from the μP core to the memory. Apparently, only data has to be transferred that is *generated*⁸ in all clusters preceding the current one *and* that is *used* in the current one (i.e. that one that is supposed to be implemented on the ASIC core). Step 2 tests whether the preceding cluster might probably be already part of the ASIC core such that the estimation can take that into account accordingly. The estimation of commu-

⁸We use $gen[\dots]$ and $use[\dots]$ as it is defined in [16].

nication effort for the ASIC core (steps 3 and 4) follows the same principle as described steps 1 and 2. Finally, the total amount of energy due to core/core partitioning is calculated in 5) using an energy amount $E_{bus\ read/write}$ for a bus access⁹.

3.4 Determining the Utilization Rate

Now, since a scheduling has been performed, we can compute the resource utilization rate U_R^{core} of a core. The following definitions hold: CS is the set of all control steps (result of the list schedule) and cs_i is the denotation of one individual control step within CS . Furthermore, O_c is the set of all operations within a cluster c whereas $o_{i,c}$ is an operation within O_c that is scheduled into control step i . An operation can be mapped to one of the D resource types in $RS = \{rs_1, \dots, rs_D\}$.¹⁰ Please note that each type π of a resource rs — or short, rs_π — can have several instances. With these definitions we can discuss the algorithm in Fig. 4 that is given in pseudo code.

Computing U_R^{core} and GEQ_{RS}

- 1) Initialize $Glob_RS_List[][][]$
- 2) **For All** $cs_i \in CS$
- 3) Initialize $Loc_RS_List[][]$
- 4) **For All** $o_{i,c} \in O_c$
- 5) Build up $Sorted_RS_List[]$
- 6) $rs_\pi := Sorted_RS_List[0]$
- 7) **For All** $elements \in Sorted_RS_List[]$
- 8) $rs_\pi :=$ current resource type of $Sorted_RS_List[]$
- 9) **If** $\#(rs_\pi)_{rs_\pi \in Glob_RS_List[cs_i][[]]} >$
- 10) $\#(rs_\pi)_{rs_\pi \in Loc_RS_List[][]}$
- 11) **Then**
- 12) Update $\#(rs_\pi)$ in $Loc_RS_List[rs_\pi][[]]$
- 13) continue with 4)
- 14) Update $\#(rs_\pi)$ in $Loc_RS_List[rs_\pi][[]]$
- 15) Update $Glob_RS_List[cs_i][[]][[]]$ with $Loc_RS_List[][]$
- 16) Initialize GEQ_{RS}
- 17) **For All** $rs_\pi \in Glob_RS_List[][][]$
- 18) $GEQ_{RS} := GEQ_{RS} + \#(rs_\pi) \times GEQ(rs_\pi)$
- 19) **For All** $cs_i \in CS$
- 20) **For All** $rs_i \in RS$
- 21) **For All** instances is
- 22) **If** $(Glob_RS_List[cs_i][rs_i][is] == 1)$
- 23) **Then** $(util[rs_i][is] = util[rs_i][is] + \#ex_cycs)$
- 24) $U_R^{core} = \frac{1}{N_{cyc}^c} \cdot \sum_{rs_i \in RS} (\frac{1}{N_{is}} \cdot \sum_{is} u_r[rs_i][is])$

Figure 4: Pseudo code of our algorithm to compute the utilization rate U_R^{core} and the hardware effort GEQ_{RS} of a cluster

At the beginning a *global resource list* $Glob_RS_List[][][]$ is defined. The first index indicates the control step cs_i , the second stands for the resource type rs_π while the third is reserved for a specific instance of that resource type. An entry can be either a "1" or a "0". For example, $Glob_RS_List[34][5][2]=1$ means that during control step 34 instance "2" of resource type "5" is used. Accordingly, "0" means that is not used. The encoding of the existence of a module type is accomplished by providing or not providing an entry in $Glob_RS_List[][][]$.¹¹ Line 2 starts a loop for all control steps cs_i and in line 3 a local resource list $Loc_RS_List[][]$ is initialized. It has the same structure as the global resource list except that it is used within one control step only. Line 4 starts a loop for all operators within a control step. A sorted resource list is de-

⁹Please note that *read* and *write* operations imply different amounts of energy. Due to lack of space a detailed description of $E_{bus\ read/write}$ is not given here.

¹⁰Examples for a resource type are an *ALU*, a *shifter*, a *multiplier* etc.

¹¹This is possible since the implementation of $Glob_RS_List[][][]$ is a chained list.

finied in line 5. It contains all resources that could execute operator $o_{i,c}$. It is sorted according to the increasing size of a resource¹². An initial resource is selected in line 6. In the following lines 9 to 13 all possible resource types are tested whether they are instantiated in a previous control step. If this is true, that resource type is assigned to the current operator, an according entry is made in the local resource list and a new operator is chosen. In the other case the searching process through the local resource list continues until an already instantiated instance is found that is not used during the current control step. In case the search did not succeed, the first¹³ resource is assigned to the current operator and an according entry is made (line 14). When all operators within a control step have been taken care of, the global resource list is enhanced by that many instances of a resource as indicated by the local resource list (line 15).

As a result, the global resource list contains the assignment of all operators to resources for all control steps. We can use this information to compute the according hardware effort GEQ_{RS}^{core} in lines 16 to 18 where $\#(rs_\pi)$ gives the number of resources of type π and $GEQ(rs_\pi)$ is the hardware effort (i.e. *gate equivalents*) of an according resource type.

The final computation of the *utilization rate* is performed in line 24. Before, in lines 19 to 23 a list is created that gives information about how often each instance of each resource is used within all control steps. Note that $\#ex_cycs \cdot \#ex_times$ is the number of cycles it takes to execute an operation on that resource multiplied by the number of times the according control step is actually invoked.¹⁴ Finally, we can compute U_R^{core} in line 24. Please note that N_{cyc}^c is the number of cycles it takes to execute the whole cluster.

As a summary, in this section we have computed U_R^{core} that gives the average utilization rate of all resources deployed within a candidate core. As we have seen in section 3.2, U_R^{core} is actually used to determine whether this might lead to an advantageous implementation of a core in terms of energy consumption or not.

Also note that all resources contribute to U_R^{core} in the same way, no matter whether they are large or small (i.e. though they may actually consume more or less energy). This is because our experiments have shown that an according distinction does not result in better partitions though the individual values of U_R^{core} are different. Reason is that the *relative* values of U_R^{core} of different clusters are actually responsible for deciding on an energy efficient core/core partition.

3.5 Design Flow

The whole design flow of our low power partitioning methodology is introduced through Fig. 5. Please note that those parts that are surrounded by dashed lines are either a mixture of standard in-house and commercial tools (as it is the case for the block "HW Synthesis") or that it refers to work that has already been published elsewhere¹⁵ (as it is the case for the block we call in this paper "Core Energy Estimation"). All other parts are those that are new and subject to explanation in this paper.

The design flow starts with the box "Application" where an application in a behavioral description is given. This might be a self-coded application or an IP core¹⁶ purchased from a vendor. Then the application is divided into clusters as described section 3.2 after an internal graph representation has been build up. Preferred clusters are pre-selected by the criteria that is described in section 3.3. Next step is a list schedule that is performed for each remaining cluster.¹⁷ such that the utilization rate U_{core}^R using the algorithm in section 3.4 can be computed. Those cluster(s) that yield a higher utilization rate compared to the implementation of a μP core and that yield the highest core of the objective function, are provided to the hardware synthesis flow. This block starts with a behavioral

¹²This is for the computation of the hardware effort of the final core only.

¹³Note that the list is sorted. So, the first resource means the smallest and therefore the most energy efficient one.

¹⁴We obtain $\#ex_times$ through profiling and $\#ex_cycs$ through the CMOS6 technology library.

¹⁵Please note that we cannot give a reference here.

¹⁶IP stands for *Intellectual Property*.

¹⁷Please note that the flow in Fig. 5 is simplified i.e. it does not feature all arcs representing the loops in the according algorithms.

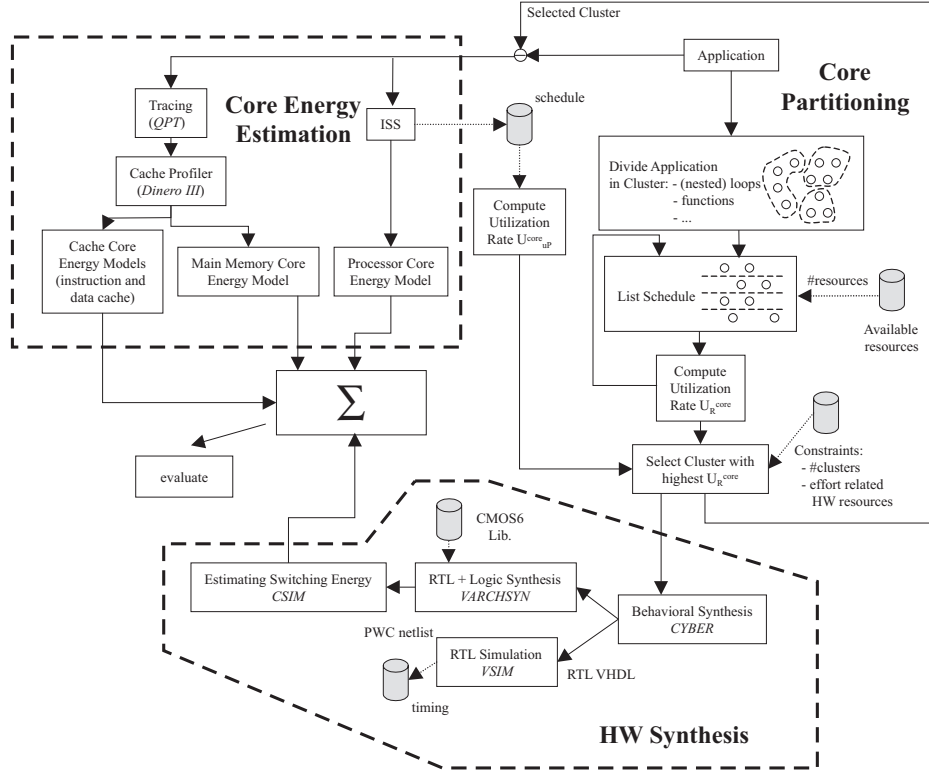


Figure 5: Design flow of our low-power hardware/software partitioning methodology

compilation tool, followed by an RTL simulator¹⁸ to retrieve the number of cycles it needs to execute the cluster, an RTL logic synthesis tool using a CMOS6 library and finally the gate-level simulation tool with attached switching energy calculation. Note that these steps, especially the last one, are the most time-consuming ones. Hence, our partitioning algorithm has to reduce the number of clusters to those that are most likely to gain an energy saving. The other application parts that are intended to run on the μP are fed into the "Core Energy Estimation" block. An instruction set simulator tool (ISS) is used in the next step. Attached to the ISS is the facility to calculate the energy consumption depending on the instruction executed at a point in time (the same methodology as in [12] is used). Analytical models for main memory energy consumption and caches are fed with the output of a cache profiler that itself is preceded by a trace tool (both: [17]). Finally, the total energy consumption is calculated and it is tested whether the total system energy consumption could be reduced or not. If "not" then the whole procedure can be repeated and the designer will make use of his/her interaction possibilities to provide the partitioning algorithms with different parameters. Please note that the designer does have manifold possibilities of interaction like defining several sets of resources, defining constraints like the total number of clusters to be selected or to modify the objective function according to the peculiarities of an application.

4 Conducted Experiments and Results

The experiments are based on our energy instruction simulation tool for a SPARCLite μP core, analytical models for main memory, instruction cache and data cache based on parameters (feature sizes, capacitances) of a 0.8μ CMOS process. We investigated the following DSP-oriented applications: an algorithm for computing 3D vectors of a motion picture ("3d"), an MPEGII encoder ("MPG"), a complex chroma-key algorithm ("ckey"), a smoothing algorithm for digital images ("digs"), an engine control algorithm

("engine") and a trick animation algorithm ("trick"). The applications range in size from about 5kB to 230kB of C code. Two rows are dedicated to each application: the initial (non-partitioned) "I" implementation and the partitioned "P" implementation. In each case the contribution of each involved core in terms of energy consumption is given. It is an important feature of our approach 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 consumptions of those cores (compare according rows of columns "i-cache", "d-cache" and "mem"). The sole energy estimation of the μP core and the ASIC core would not be sufficient since the energy consumption of the other cores in some cases drops dramatically as well (see for example the energy of the i-cache consumed by the "trick" application that drops from $5.58mJ$ to $12.59\mu J$). In one case ("ckey") which was in fact the less memory-intensive one, the contribution to total energy consumption could be neglected.

The rightmost four columns give the execution time before and after the partitioning. This is of paramount importance: we achieved high energy savings but *not* at the cost of performance (except for one case). Instead, energy savings are achieved at additional hardware costs for the ASIC core through our selective algorithms described in section 3. The largest (but still small) additional hardware effort accounted for slightly less than 16k cells. But in that case ("digs") a large energy saving of about 94% could be achieved. Due to today's design constraints in embedded high-performance applications, a loss in performance through energy savings is in the majority of cases not accepted by designers. On the other side a (low) additional hardware effort of 16k cells is not a real constraint since state-of-the-art systems on a chip have about 10Mio. transistors¹⁹.

Fig.6 visualizes the results by giving the percentage of energy sav-

¹⁸In order to keep the Fig. 5 of the design flow as clear as possible we did not draw the inputs of input stimuli pattern at various points in the design flow.

¹⁹Please note that due to the currently deployed technology of 0.18μ an even higher transistor count would be possible. But due to the current "design gap" (therefore see also [1], a maximum is currently about 10Mio. transistors on a chip (not including main memory).

App.		Energy						Sav%	Exec. Time [cycles]			
		i-cache	d-cache	mem	μP core	ASIC core	total		μP core	ASIC core	total	Chg%
3d	I	116.93uJ	14.26uJ	29.71uJ	566.78uJ	n/a	727.68uJ	-35.21	39,712	n/a	39,712	-17.29
	P	0.2627uJ	0.1227uJ	0.2626uJ	0.4705mJ	0.3078uJ	471.46uJ		32,689	154	32,843	
MPG	I	44.79mJ	17.98mJ	2.305mJ	74.32mJ	n/a	140.92mJ	-43.20	5,167,958	n/a	5,167,958	-52.90
	P	35.36mJ	13.14mJ	2.941mJ	27.14mJ	1.462mJ	80.04mJ		1,696,771	737,154	2,433,925	
ckey	I	0.0	0.0	0.0	329.99mJ	n/a	329.99mJ	-76.81	169,511,665	n/a	169,511,665	-74.98
	P	0.0	0.0	0.0	53.042mJ	23.468mJ	76.51mJ		30,258,256	12,144,420	42,402,676	
digs	I	11.69mJ	5.123mJ	0.493mJ	52.70mJ	n/a	70.00mJ	-94.12	3,706,291	n/a	3,706,291	-42.64
	P	14.27uJ	2.039uJ	20.43uJ	46.78uJ	4.03mJ	4.11mJ		6,347	2,119,750	2,126,097	
eng.	I	117.55uJ	33.50uJ	33.77uJ	270.10uJ	n/a	454.92uJ	-31.27	68,534	n/a	68,534	-24.26
	P	84.90uJ	26.62uJ	21.89uJ	167.86uJ	1.408uJ	312.68uJ		51,311	599	51,910	
trick	I	5.58mJ	410.40uJ	264.91uJ	18.54mJ	n/a	24.79mJ	-94.79	1,489,000	n/a	1,489,000	69.64
	P	12.59uJ	3.677uJ	71.70uJ	181.01uJ	1.025uJ	1291.98uJ		11,658	2,514,300	2,525,958	

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

ings and the related changes in execution time of a specific application. As seen, we achieved high energy savings between about 35% and 94% while the decrease in execution time (i.e. faster) ranges between about 17% and 75%. It shows that our approach is especially tailored for energy minimization and improvement of execution time is only a side effect. Note, that our algorithms could not find an appropriate cluster of the application "trick" yielding energy savings and a reduction of execution time. A closer investigation revealed that this application did not feature small clusters with a high U_{core}^R . But our algorithm rejects clusters that would result in an unacceptable high hardware effort (due to factor "F", line 13 in Fig. 1).

As a result of our approach we can summarize that our approach achieved tremendous energy savings for DSP-oriented application with a small hardware overhead and in most cases even reduced execution time (i.e. increased performance).

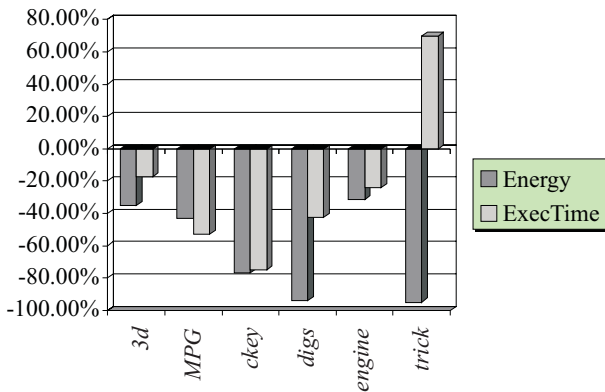


Figure 6: Results: achieved energy savings and change of total execution time

5 Conclusion

In this paper we have presented a novel low power partitioning approach for core-based systems that is very comprehensive since it takes into consideration a whole system consisting of a μP core, an ASIC core, caches and main memory. In addition, an important advantage against other low power system-level design approaches is that we can achieve tremendous energy savings of up to 94% at relatively low additional (hardware) costs. This has been possible since our methodology uses the idea of evaluation a utilization rate U_{core}^R at the operator-level, thus allowing to select efficient clusters. Furthermore, our methodology is tailored especially to computation and memory intensive applications like those found in economically interesting mobile devices like cell phones, digital cameras etc.

Further work will concentrate on deriving low-power methods for control-dominated systems.

References

- [1] M. Keaton, P. Bricaud, *Reuse Methodology Manual For System-On-A-Chip Designs*, Kluwer Academic Publishers, 1998.
- [2] *TI's 0.07 Micron CMOS Technology Ushers In Era of Gigahertz DSP and Analog Performance*, Texas Instruments, Published in the Internet, <http://www.ti.com/sc/docs/news/1998/98079.htm>, 1998.
- [3] R.K. Gupta, Y. Zorian, *Introducing Core-Based System Design*, IEEE Design & Test of Computers Magazine, Vol. 13, No. 4, pp. 15–25, 1997.
- [4] 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.
- [5] R.K. Gupta and G.D. Micheli, *System-Level Synthesis using Re-programmable Components*, IEEE/ACM Proc. of EDAC'92, IEEE Comp. Soc. Press, pp. 2–7, 1992.
- [6] Z. Peng, K. Kuchcinski, *An Algorithm for Partitioning of Application Specific System*, IEEE/ACM Proc. of The European Conference on Design Automation (EuroDAC) 1993, pp. 316–321, 1993.
- [7] J. Madsen, P. V. Knudsen, *LYCOS Tutorial*, Handouts from Eurochip course on Hardware/Software Codesign, Denmark, 14.–18. Aug. 1995.
- [8] T. Y. Yen, W. Wolf, *Multiple-Process Behavioral Synthesis for Mixed Hardware-Software Systems*, IEEE/ACM Proc. of 8th. International Symposium on System Synthesis, pp. 4–9, 1995.
- [9] A. Kalavade, E. Lee, *A Global Critically/Local Phase Driven Algorithm for the Constraint Hardware/Software Partitioning Problem*, Proc. of 3rd. IEEE Int. Workshop on Hardware/Software Codesign, pp. 42–48, 1994.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] Ch.Ta Hsieh, M. Pedram, G. Mehta, F.Rastgar, *Profile-Driven Program Synthesis for Evaluation of System Power Dissipation*, IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.576–581, 1997.
- [14] P.-W. Ong, R.-H. Ynn, *Power-Conscious Software Design – a framework for modeling software on hardware*, IEEE Proc. of Symposium on Low Power Electronics, pp. 36–37, 1994.
- [15] T. Sato, M. Nagamatsu, H. Tago, *Power and Performance Simulator: ESP and its Application for 100 MIPS/W Class RISC Design*, IEEE Proc. of Symposium on Low Power Electronics, pp. 46–47, 1994.
- [16] A.W. Aho, R. Sethi and J.D. Ullmann, *COMPILERS Principles, Techniques and Tools*, Bell Telephone Laboratories, 1987.
- [17] M. D. Hill, J. R. Laurus, A. R. Lebeck et al., *WARTS: Wisconsin Architectural Research Tool Set*, Computer Science Department University of Wisconsin.
- [18] P. Landman and J. Rabaey, *Architectural Power Analysis: The Dual Bit Type Method*, IEEE Transactions on VLSI Systems, Vol.3, No.2, June 1995.