Strobel *et al. EURASIP Journal on Embedded Systems* (2017) 2017:2
DOI 10.1186/s13639-016-0039-5

EURASIP Journal on Embedded Systems

RESEARCH Open Access

Low power memory allocation and mapping for area-constrained systems-on-chips

Manuel Strobel*, Marcus Eggenberger and Martin Radetzki

Abstract

Large fractions of today's embedded systems' power consumption can be attributed to the memory subsystem. In order to reduce this fraction, we propose a mathematical model to optimize on-chip memory configurations for minimal power. We exploit the power reduction effect of splitting memory into subunits with frequently accessed addresses mapped to small memories. The definition of an integer linear programming model enables us to solve the twofold problem of allocating an optimal set of memory instances with varying size on the one hand and finding an optimal mapping of application segments to allocated memories on the other hand. Experimental results yield power reductions of up to 82 % for instruction memory and 73 % for data memory. Area usage, at the same time, deteriorates by only 2.1 %, respectively, 1.2 % on average and even improves in some cases. Flexibility and performance of our model make it a valuable tool for low power system-on-chip design, either for efficient design space exploration or as part of a HW/SW codesign synthesis flow.

Keywords: Integer linear programming, ILP, Low power, On-chip memory, SRAM, System-on-chip, SoC

1 Introduction

The ubiquitous nature of embedded systems substantiates the need for design and development methods that yield the lowest possible power consumption. Fortunately, such specialized systems often perform only known tasks, which allows engineers to optimize specifically for those tasks without compromise. Since up to 60 % of an embedded system's power consumption is attributed to memory [1], optimizing the memory subsystems is an evident design goal. A commonly used method to reduce memory power consumption is splitting memory into several individual memory instances [1–6].

A break-down analysis of the energy consumption caused by reading from on-chip static random-access memory (SRAM) shows that less than 1 % is consumed by the actual memory cells and about 90 % by components such as precharge unit, sense amplifiers, and address transition detection [7]. The energy consumption of these components is heavily affected by the overall size of the SRAM instance being accessed. One can make use of this aspect to reduce the total energy consumption by splitting on-chip memory into multiple instances such that frequently accessed segments of the address space reside in separate small memory instances.

However, splitting memory into multiple units requires an interconnect, e.g., a bus or a custom fabric, that forwards read and write requests to the individual memory instances. Obviously, this interconnect consumes on-chip area and energy itself and may offset the benefits of the split memories. Furthermore, using multiple small memories instead of a single large one implies an increase of area requirements and thus can become prohibitive. To achieve the highest power reduction possible, it is also necessary to reorganize the logical address space such that the most frequently accessed segments are grouped together and can be mapped to the same physical memory instance. For example, if the most frequently accessed memory addresses are uniformly distributed over the application's address space, frequent and infrequent addresses will inevitably be mapped to the same memory instances voiding any benefit of a split memory architecture. Altogether, this makes the search for an optimal memory configuration a non-trivial task.

In this work, we propose an integer linear programming (ILP) model that solves the twofold problem of finding the optimal allocation of memory instances as well as the

*Correspondence: manuel.strobel@informatik.uni-stuttgart.de
Embedded Systems Department, Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Pfaffenwaldring 5b, 70569 Stuttgart, Germany

optimal mapping of address ranges to allocated memories. The model is parameterized to allow user constraints such as limiting the maximum number of memory instances or the available area. The model can be used to optimize the on-chip memory architecture for a single dedicated software, multiple but replaceable applications, or coexisting applications in a multitasking environment.

The rest of this work is organized as follows. Section 2 discusses existing research in the field of split memories. We declare our problem statement in Section 3 and set up the design space, which is used for the elaboration of our formal ILP model in Section 4. The integration with system synthesis is outlined in Section 5. Section 6 discusses evaluation results, and we conclude with Section 7.

2 Related work

Heuristics for optimized memory configurations have been investigated for different design goals. Mai et al. [1] enable manual algorithm execution by largely simplifying the low power optimization problem. The authors of [5, 8] target the combined optimization problem of memory and bus partitioning for multi-master, multi-memory systems. Zhuge et al. [9] distribute variables between different memory instances to increase digital signal processor (DSP) performance. However, we aim for an optimal solution.

Optimal algorithms have been employed within a confined problem scope or with reduced complexity. Such algorithms are either limited to a fixed address layout [2, 10]; only work with two memory instances [11]; coarsely quantize memory accesses over time [10]; or do not consider leakage or deselect power [2]. Furthermore, some allow only splitting into equally sized sub-banks [10, 12] or only estimate the segmentation overhead [2, 12, 13].

A closely related field of interest lies in the optimization of scratch pad memories (SPM), which are an efficient replacement for caches in embedded systems used in conjunction with external memory [14]. To identify the optimal selection of address ranges to be mapped to the SPM, ILP models have been developed [4, 6] and dynamic programming has been used [13]; all of which only consider a single fixed size SPM and cannot be used to optimize the SPM's sub-organization.

Another common problem in the SPM domain is the logical partitioning of address spaces to reduce the number of SPM fills [15–17]. While related, we solve a different problem: the partitioning of the physical on-chip memory organization to minimize power consumption.

On-chip memory configurations have also been optimized for application-specific purposes, e.g., for DSP [18] or video processing applications [19, 20]. Our work differs as we intend to provide a generic optimization methodology independent of the targeted application specifics.

The main contribution of this work is a mathematical model to identify a low power memory configuration for a set of applications. The model differentiates between read and write accesses and supports leakage and deselect power. From an almost arbitrary list of memory types, the model allocates the optimal number and types of memories and yields an optimal mapping of address space ranges to the selected memories to achieve the lowest possible power consumption. User-defined constraints provide the ability to efficiently explore the design space in the area and power domain. An automated optimization flow contributing HW and SW optimizations makes our model further a valuable tool for system synthesis.

3 Problem statement

The targeted hardware platforms are single-CPU systems-on-chips (SoC) with support for multiple on-chip memories of varying structure, i.e., sub-banking, and size (cf. Fig. 1).

We distinguish between the following three modes of operation:

1. *Single-App* —The system is dedicated to the execution of a single application.
2. *Combined* —Consideration of multiple applications that are exchangeable, e.g., via firmware update but executed by the system exclusively (only one application resides in on-chip memory at any point in time).
3. *Multitasking* —Multitasking with static or dynamic scheduling.

We assume a set $M = [1, m]$ of different memory types and a set of applications $A = [1, a]$. Each application a is characterized by a corresponding set of application profiles $P_a = [1, p]$, which will be detailed later in this section.

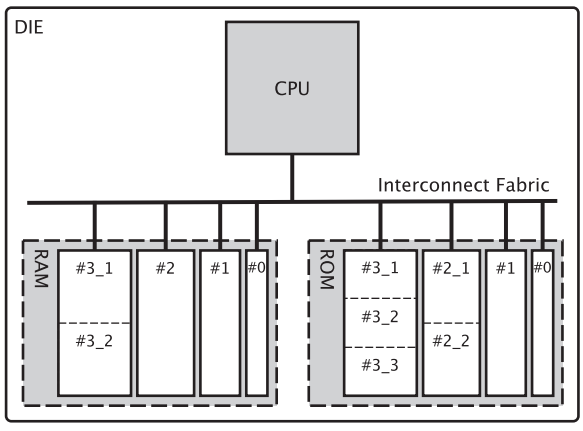

Fig. 1 Exemplary single-CPU SoC with heterogeneous on-chip memory structure

In case of *Multitasking*, each application is constantly executed by one and the same task.

The problem statement is defined as follows: Find an allocation α of memory instances and a mapping β that assigns each application profile to exactly one memory instance such that α and β yield the lowest power consumption of all possible allocations and mappings while satisfying area and user-defined constraints. For the sake of readability and clarity, we focus on instruction memory. However, the model can also be used for data memory with small modifications, which we point out whenever applicable. In order to emphasize whether memory is read only or can be written to, we refer to instruction memory as ROM and to data memory as RAM in the following.

3.1 Design space

The basic elements of the design space are the individual memory types m_i, whereas the actual design space is composed of all possible combinations of one or more memory types with multiple selections of individual memory types being possible. That is, all possible allocations $\alpha \in \mathbb{N}_0^m$ with α_i being the number of instances of memory type m_i. A memory type refers to a specific instance of on-chip SRAM according to a technology library. The individual memory types can differ in various aspects such as size, area, or sub-banking organization.

In our model, each memory type m_i is defined by a set of relevant physical parameters. For ROM, these parameters are:

- The size in kilobytes
- The area in technology size units (e.g. mm^2)
- The read current in μA/MHz, consumed when accessing the ROM
- The deselect current in μA/MHz, consumed when the memory is idle
- The leakage current in μA, which is permanently consumed

For RAM, the write current must be accounted as additional parameter, also given in μA/MHz. Note that read, write, and deselect current are given with respect to the operational clock frequency of the system, which we consider to be fixed. The interconnect fabric is not directly part of the design space. However, it contributes to overall power consumption depending on the total number of allocated memory instances and is therefore considered in our model as well.

This power model is in line with vendor-specific datasheets (e.g., [21]), making it suitable for the exploration of the design space, which we prune from any solution that is infeasible or does not meet the designer's needs using three constraints. One constraint ensures that enough memory is provided for the embedded software application. The other two constraints allow limiting the available area for the split memory organization and the total number of memory instances.

An actual memory configuration for a given set of application profiles can be identified by a pair of power consumption on the one hand and on-chip area requirement on the other hand. For each individual dimension, power and area, a single minimal solution exists in the design space, while both together consequently define the range of the solution space. Within these borders, further pareto-optimal solutions can be obtained through variation of the abovementioned constraints.

3.2 Application profiles

Applications are described in terms of a set P_a of application profiles, each representing the behaviors of a part of the application, which are possibly periodic. Our model does not presume a certain granularity for application profiles and thus can be chosen according to the individual needs. For example, ranging from fine to coarse grained, application profiles can represent individual instructions, basic blocks, functions, or groups thereof.

The relevant characteristics of an exemplary application profile are shown in Fig. 2a. Each profile assumes a fixed period, which can be divided into two parts: an active phase, the duty cycle, and an idle phase. While memory can only be accessed during the duty cycle, it is not necessarily accessed throughout the entire duty cycle as indicated by the individual peaks in Fig. 2a. However, the power consumption does not depend on the individual points in time when the memory is accessed but only on the duration of those accesses. This allows us to simplify the application profile by combining the individual short memory accesses and modeling them as a fraction of the duty cycle called access probability. Figure 2b shows the resulting, simplified application profile. For RAM, the access probability is replaced by a pair of read and write probabilities, representing the fraction of time for read and write operations during the duty cycle (cf. Fig. 2c, d).

Note that while application profiles are designed to support the periodic nature often found in embedded applications, they can easily be used to describe non-periodic behavior by setting the period to the total application runtime.

3.3 Schedule

In case of *Single-App* and *Combined* operation modes (cf. Section 3), the overall system runtime is attributed to one application only. In a *Multitasking* environment, however, we assume one task per application and a schedule that determines the share, each application consumes of total system runtime. This aspect is modeled in a flexible way

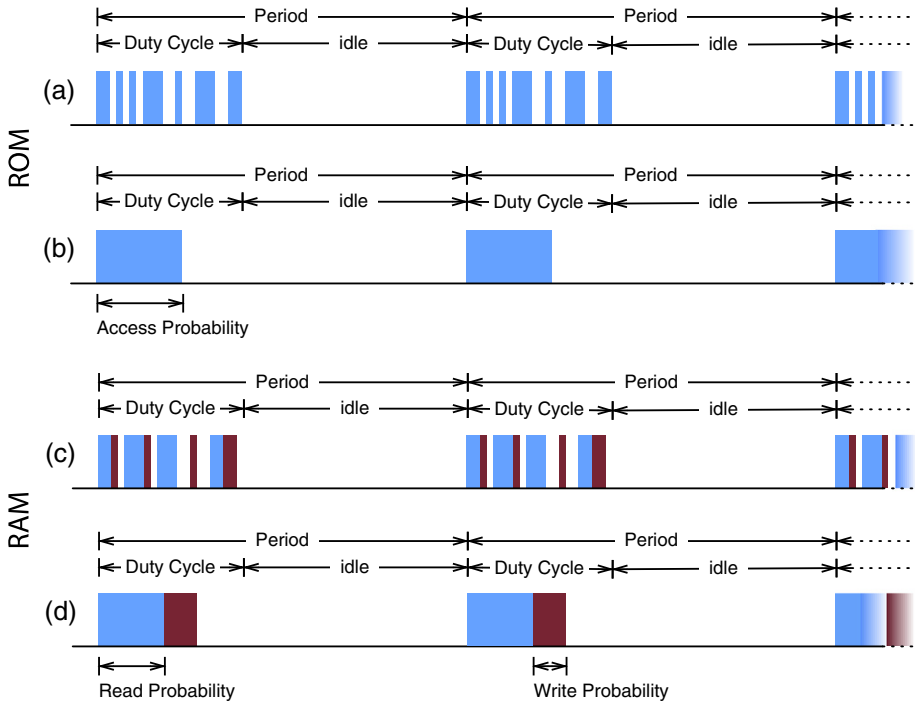

Fig. 2 a Exemplary application profile with period and duty cycle (ROM). *Blue bars* represent read memory accesses. **b** Simplified application profile using an access probability. **c** Exemplary application profile (RAM). *Blue bars* represent read, *red bars* write memory accesses. **d** Simplified application profile using read and write probabilities

by a set $S \in [0,1]^{|A|}$ with each element s_a representing the share of total execution time for each application and $\sum_{a=1}^{|A|} s_a = 1$. In a static schedule, application runtimes and the hypercycle suffice to determine this set. In case of dynamic scheduling, reasonable values for S can be obtained from system simulation.

3.4 Power model

In this work, we focus on reducing the average power consumption $P_{\text{avg}} = E_{\text{tot}}/T$ depending on the total energy consumed E_{tot} and the runtime T of the application(s). In this section, we establish a component wise definition of the required energy E_p for a single application profile. This model forms the basis of our ILP model, presented in Section 4.

Due to the periodic nature of application profiles, the total energy consumption of a single profile only depends on the energy consumed in one period. For a given memory instance, energy consumption depends on whether it is currently being accessed or not. When reading from memory, standby current plus read current is consumed, and otherwise, standby current plus deselect current is consumed. A single period of a profile p accordingly consumes the sum of read, deselect, and standby energy:

$$E_p = E_{\text{read}} + E_{\text{desel}} + E_{\text{stdby}} \tag{1}$$

The three components E_{read}, E_{desel}, and E_{stdby} are defined as follows:

$$E_{\text{read}} = d \cdot p_r \cdot I_r\left(f\right) \cdot V \cdot t_p \tag{2}$$

$$E_{\text{desel}} = (1 - d \cdot p_r) \cdot I_d\left(f\right) \cdot V \cdot t_p \tag{3}$$

$$E_{\text{stdby}} = I_s \cdot V \cdot t_p \tag{4}$$

with duty cycle d, access probability p_r, and period t_p of the profile; I_r, I_d, and I_s representing read, deselect, and standby current as defined by the technology library; and V as the supply voltage of the memory.

To determine the overall power consumption of an application a, all application profiles must be considered. Note that we use uppercase indices for the combined energies and lowercase indices for the energy consumption of individual profiles. According to Eq. 1, the total energy consumption of an application is the sum of overall read, deselect, and standby energy:

$$E_{\text{tot}} = E_{\text{READ}} + E_{\text{DESEL}} + E_{\text{STDBY}} \tag{5}$$

The read energy of an application is defined as the sum of all individual application profiles' read energies (Eq. 6). By substituting E_{read_i} with Eq. 2, we find in Eq. 7 that the

individual profile periods t_{p_i} can be eliminated to obtain Eq. 8.

$$E_{\text{READ}}(a) = \sum_{i=1}^{|P_a|} E_{\text{read}_i} \cdot \frac{T}{t_{p_i}} \tag{6}$$

$$= \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \cdot I_r(f) \cdot V \cdot t_{p_i} \cdot \frac{T}{t_{p_i}} \tag{7}$$

$$= T \cdot \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \cdot I_r(f) \cdot V \tag{8}$$

The combined deselect energy cannot be expressed in an additive form as deselect current only applies when no profile is accessing the memory (cf. Eq. 9). Again, the individual profile periods t_{p_i} can be eliminated to get Eq. 11.

$$E_{\text{DESEL}}(a) = \left(T - \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \cdot t_{p_i} \cdot \frac{T}{t_{p_i}} \right) \cdot I_d(f) \cdot V \tag{9}$$

$$= \left(T - T \cdot \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \right) \cdot I_d(f) \cdot V \tag{10}$$

$$= T \cdot \left(1 - \cdot \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \right) \cdot I_d(f) \cdot V \tag{11}$$

Standby energy is consumed independently of any application profile (Eq. 12).

$$E_{\text{STDBY}}(a) = T \cdot I_s \cdot V \tag{12}$$

Altogether, Eqs. 8, 11, and 12 prove that the total energy consumption E_{tot} does not depend on the individual profile periods but only on the corresponding duty cycles and access probabilities. With $P_{\text{avg}} = E_{\text{tot}}/T$, one can further eliminate the application runtime T, i.e.:

$$P_{\text{READ}}(a) = \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \cdot I_r(f) \cdot V \tag{13}$$

$$P_{\text{DESEL}}(a) = \left(1 - \cdot \sum_{i=1}^{|P_a|} d_i \cdot p_{r_i} \right) \cdot I_d(f) \cdot V \tag{14}$$

$$P_{\text{STDBY}}(a) = I_s \cdot V \tag{15}$$

For RAM, the average power increases by the write power P_{WRITE} and the formula for P_{DESEL} must be amended such that deselect current applies when no profile is reading or writing to memory.

$$P_{\text{WRITE}}(a) = \sum_{i=1}^{|P_a|} d_i \cdot p_{w_i} \cdot I_w(f) \cdot V \tag{16}$$

$$P_{\text{DESEL}}(a) = \left(1 - \sum_{i=1}^{|P_a|} d_i \cdot (p_{r_i} + p_{w_i}) \right) \cdot I_d(f) \cdot V \tag{17}$$

4 ILP model

For a complete model of the whole memory architecture, not only the individual memory instances but also the interconnect fabric must be accounted (cf. Section 3.1). As the fabric grows in complexity with the number of connected memories, its power consumption and area requirements grow. To capture this effect, we model both power and area requirements of the interconnect fabric as piecewise linear function of the number of connected memories. This allows using exact data points for desired configurations and relying on linear interpolation otherwise. Let $P_F : \mathbb{N}_0 \to \mathbb{R}$ and $A_F : \mathbb{N}_0 \to \mathbb{R}$ denote the functions for the interconnect fabric's power consumption and area requirements, respectively.

4.1 Memory allocation

One part of the ILP problem is finding an allocation $\alpha \in \mathbb{N}_0^{|M|}$ with M being the set of available memory types and α_i representing the number of allocated instances of memory type m_i. In the ILP model, α is constrained to be of type non-negative integer: $\forall i \in [1, |M|] : \alpha_i \geq 0$.

Using the allocation α and a user-defined parameter $\text{mems}_{\max}$, we limit the maximum number of memory instances:

$$\sum_{i=1}^{|M|} \alpha_i \leq \text{mems}_{\max} \tag{18}$$

With $A_M \in \mathbb{R}^{|M|}$ representing the area requirements of the individual memory types, parameter $\text{area}_{\max}$ constrains the total area available for all memory instances and the interconnect:

$$\sum_{i=1}^{|M|} \alpha_i \cdot A_{M,i} + A_F \left(\sum_{i=1}^{|M|} \alpha_i \right) \leq \text{area}_{\max} \tag{19}$$

4.2 Application mapping

For each application a, we represent the mapping as a binary matrix $\beta_a \in \{0,1\}^{|P_a| \times |M|}$, with the elements β_{aij} indicating whether application profile i of application a has been mapped to memory type j ($\beta_{aij} = 1$) or not

($\beta_{aij} = 0$). To ensure a correct solution, each application profile must be bound to exactly one memory type:

$$\forall a \in [\,1, |A|\,], \forall i \in [\,1, |P_a|\,] : \sum_{j=1}^{|M|} \beta_{aij} = 1 \tag{20}$$

Furthermore, enough instances of a given memory type must be provided fitting all application profiles mapped to that memory type. Let $\sigma^{P_a} \in \mathbb{N}_0^{|P_a|}$ and $\sigma^{M} \in \mathbb{N}_0^{|M|}$ be the vectors representing memory required by application profiles and memory provided by memory types, respectively. With this, we specify the memory requirements as follows.

$$\forall a \in [\,1, |A|\,], \forall j \in [\,1, |M|\,] : \sum_{i=1}^{|P_a|} \beta_{aij} \cdot \sigma_i^{P_a} \leq \alpha_j \cdot \sigma_j^{M} \tag{21}$$

Note that Eq. 21 only ensures that the total allocated memory is sufficient for *each* application individually and not for *all* applications at the same time. This reflects the *Single-App* and *Combined* operation mode (cf. Section 3) with a single but interchangeable software application such as a firmware. Thus, Eq. 21 allows each application exclusive access to the whole memory.

For the consideration of *Multitasking*, enough memory has to be allocated in order to satisfy the requirements of all applications at the same time. Equation 21 consequently has to be altered by replacing the universal quantifier $\forall a \in [\,1, |A|\,]$ with a summation $\sum_{a=1}^{|A|}$ to support concurrent applications that share the memory. Accordingly, we get Eq. 22 as follows:

$$\forall j \in [\,1, |M|\,] : \sum_{a=1}^{|A|} \sum_{i=1}^{|P_a|} \beta_{aij} \cdot \sigma_i^{P_a} \leq \alpha_j \cdot \sigma_j^{M} \tag{22}$$

It is worth mentioning that neither Eq. 20 nor Eq. 21/22 explicitly bind application profiles to memory *instances* but only to memory *types*. This significantly reduces the complexity of the ILP problem without sacrificing precision or correctness.

4.3 Optimization goal

From Section 3.4 and Eqs. 13 to 15, we already know that the average read power consumption neither depends on the application runtime nor on the period of the individual profiles and that the same holds for the deselect and standby power. Note that we assume the application runtime T to be independent of the memory selection and, thus, to be constant. We justified this assumption by only splitting a single memory into multiple instances of the same timing characteristics. Our utilized interconnect is further barely affecting the critical path of memory accesses. Consequently, we do not have to consider common periods of the individual profiles in our power model. Based thereon, we can now derive the individual power components for each memory j and application a as given in Eqs. 23 to 25.

$$P_{\text{read},j}(a) = \sum_{i=1}^{|P_a|} \beta_{aij} \cdot d_i \cdot p_{r_i} \cdot I_{r,j}\left(f\right) \cdot V \tag{23}$$

$$P_{\text{desel},j}(a) = \left(\alpha_j - \sum_{i=1}^{|P_a|} \beta_{aij} \cdot d_i \cdot p_{r_i}\right) \cdot I_{d,j}\left(f\right) \cdot V \tag{24}$$

$$P_{\text{stdby},j} = \alpha_j \cdot I_{s,j} \cdot V \tag{25}$$

Furthermore, combining Eqs. 23 to 25 and the interconnect fabric's power consumption $P_F(n)$ allows us to postulate the average power consumption P_{avg} for *Single-App* and *Combined* mode to be minimized by the ILP solver by choosing suitable variable assignments for allocation α and mapping β.

$$P_j(a) = P_{\text{read},j}(a) + P_{\text{desel},j}(a) + P_{\text{stdby},j} \tag{26}$$

$$P_{\text{avg}} = P_F\left(\sum_{i=1}^{|M|} \alpha_i\right) + \frac{1}{|A|} \sum_{a=1}^{|A|} \sum_{j=1}^{|M|} P_j(a) \tag{27}$$

In the *Single-App* case with $|A| = 1$, as well as in *Combined* operation mode, we assume all applications to be of equal importance and thus average their individual power contributions $\sum_{j=1}^{|M|} P_j(a)$ (cf. Eq. 27).

In the case of *Multitasking*, Eq. 27 is replaced by Eq. 28 in order to prioritize the power contribution of each application. To this end, we assign different weights according to the scheduled execution times as specified by $s_a \in S$ for each application (cf. Section 3.3).

$$P_{\text{avg}} = P_F\left(\sum_{i=1}^{|M|} \alpha_i\right) + \sum_{a=1}^{|A|} \sum_{j=1}^{|M|} s_a \cdot P_j(a) \tag{28}$$

5 System synthesis

In this section, we briefly illustrate how the ILP model can be incorporated into a low power system synthesis flow. Since the model not only yields the optimal allocation of memories but also provides an optimal mapping of profiles to memory instances, our process is ideal for hardware/software codesign. As illustrated in Fig. 3, the general synthesis flow can be divided into a sequence of four steps: cross compilation of the application (a), extraction of application profiles (b), ILP solving (c), and optimization of hardware and software domains (d, e).

In the first step (a), the application's source files are cross compiled for the target architecture, and the resulting object files are linked to an executable binary. The binary is then analyzed to set up the list of application profiles. While application profiles can be modeled at different levels of granularity, a reasonable starting point is creating application profiles on a basis of symbols, which

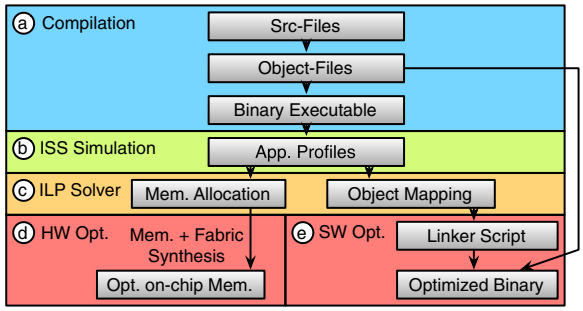

Fig. 3 Synthesis flow for low power HW/SW codesign

correspond to functions or objects. Objects not only represent global variables but also stack and heap, for which we only extract the initial sizes as their eventual sizes are determined at runtime. For each symbol in the binary, the name, starting address, and size are extracted.

In the second step (b), an instruction set simulator (ISS) is used to complete the application profiles. For functions, the simulator records the number of instruction fetches and time spent in the function. For all other symbols, reads and writes are recorded. Furthermore, stack and heap sizes are tracked to determine their maximum extent.

The resulting profiles are then fed to the ILP solver (c), which solves two independent ILP problems, one for instruction memory and one for data memory. The resulting allocation is then used to instantiate the required memory IP blocks and to synthesize the interconnect fabric (d). Based on the optimal mapping, a linker script is created to re-link the previously compiled object files with an optimal address space layout (e).

6 Evaluation

We evaluated the applicability of our approach using the following four applications from the Embedded Microprocessor Benchmark Consortium (EEMBC) MultiBench benchmark suite [22]: *IP reassembly*, *IP check*, *MD5*, and *Huffman*. IP reassembly reflects the work of a network router when reassembling fragmented packets, IP check performs IP header validation, MD5 performs checksum calculation, and Huffman implements the decoding algorithm commonly found in image and video compression standards. In *Single-App* operation mode, each application was evaluated individually, i.e., $|A| = 1$. In *Combined* operation mode, we considered the optimization of all four applications with $|A| = 4$. In case of *Multitasking*, the above applications were considered as parts of a simple network benchmark that processes incoming data from a network with one task per application. IP check is performed per incoming packet while IP reassembly, MD5, and Huffman are only executed for each fully received fragmented packet. As exemplary case, we assumed a mean packet fragmentation of 4 and the following schedule $S = [\frac{1}{7}, \frac{4}{7}, \frac{1}{7}, \frac{1}{7}]$ accordingly.

To provide the reader with a consistent terminology, we refer to the set of all applications and all operation modes, as described above, whenever using the term *evaluated scenarios*.

Profiling data, as basis for the application profiles P_a, was extracted using the ppc405 ISS from the SoCLib platform [23].

We used CACTI 6.5 [24] to generate a set of 79 different memory types M ranging from 512 bytes to 16 Mbytes in the 45 nm technology node. For each size, multiple versions with different number of sub-banks were created to allow the ILP solver choosing between low active power and low standby power memories. Note that the number of sub-banks is *not* an ILP variable as the CACTI-generated memory data is not parametric. Instead, two memories differing in their sub-banking organization correspond to two different memory types available to the ILP solver. From the set of information that CACTI provides per memory type, dynamic read/write energy per access and leakage power were utilized to derive the corresponding currents as required by our model.

The interconnect fabric was designed as a parameterized, multiplexer-based VHDL model and has been synthesized using the NanGate 45nm Open Cell Library [25]. Power simulations were performed using actual memory access traces for individually synthesized fabrics. While the interconnect prolongs the critical path for memory accesses, all investigated setups were still able to run at clock frequencies of up to 800 MHz.

For our experiments, we assumed an arbitrary but reasonable memory operation frequency of 100 MHz and a system operation voltage of 1.0 V. Since all equations in our model have linear character, these two parameters can easily be modified to values as dictated by a system design at hand. Together with this basic information, the application profiles, data about the interconnect, and the memory data from CACTI, all required input parameters for the optimization model can be provided. To get an idea about the impact of the central optimization model parameters, i.e., read, write, and standby current, Table 1 provides exemplary values for a subset of single-banked memories, as they are available to the ILP solver. Please note that the deselect current I_d is not listed in this table. Since our interconnect model was designed in a way that keeps the interfacing signals of the connected memory components stable during non-access phases, we are able to ignore the impact of I_d. However, the deselect current is partly present in vendor data sheets to keep track of signal line toggling during idle periods of a memory component and therefore supported by our model.

Table 1 Optimization model parameters for an exemplary set of single-banked memories with 32 bit bus width

Size (bytes)	l_r (mA)	l_w (mA)	l_s (mA)
512	0.309996	0.288318	0.000110234
4K	0.649919	0.597679	0.000932013
32K	2.27616	1.64435	0.00649379
256K	10.7093	2.63148	0.0453433
2M	46.6984	14.5283	0.368005
16M	111.438	45.8345	2.86068

6.1 Power optimality

For all evaluated scenarios, separate optimizations for instruction memory and data memory were performed. For both, the ILP model was solved with different limits for the allowed number of partitions ranging from 1 (unpartitioned) to 8. This allows investigating the effect of split memory configurations of different sizes. The resulting optimal power consumptions for each limit are plotted in Fig. 4. For instruction memory, a split memory configuration of only two instances already causes a drastic power reduction (80.5 % for IP reassembly, 67.6 % on average). Increasing the number of partitions shows slight improvements for up to four memories (82.9 % for IP reassembly), but no evaluated scenario can benefit from more than four instances. With a still very good power reduction of 60.2 %, the MD5 benchmark benefits the least from split memories.

Increasing the number of data memories reduces power consumption more gradually, and all applications can benefit from up to eight memory instances. The average power reduction of data memory is 60.8 %, with a minimum of 37.8 % for Huffman decoding and a maximum of 73.2 % for IP check. While still yielding good results, splitting data memory was not as beneficial as splitting instruction memory, which we attribute to the large heap requirements of the applications. It is worth noting that 6 or 7 partitions is not favorable for the IP reassembly benchmark, but allowing eight partitions eventually reduces the power consumption by another 5 %. This highlights that the optimal number of memory partitions cannot be known in advance and should be a free variable in the optimization process.

Table 2 shows a detailed power consumption break down for instruction and data memory of two exemplary applications comparing unpartitioned and optimal solutions. For a given type of memory, characterized by *Size* and number of sub-banks (*Banks*), *Num* represents the number of instances allocated by the ILP solver, and *Objs* is the number of functions or global variables mapped by the ILP to that memory type. The average power consumption P (mW) is given under consideration of the mapped objects and their memory access patterns. *Reads* and *Writes* state the relative amount of memory accesses caused by the mapped objects.

For instruction memory, the IP reassembly benchmark profits heavily from the split memory configuration as

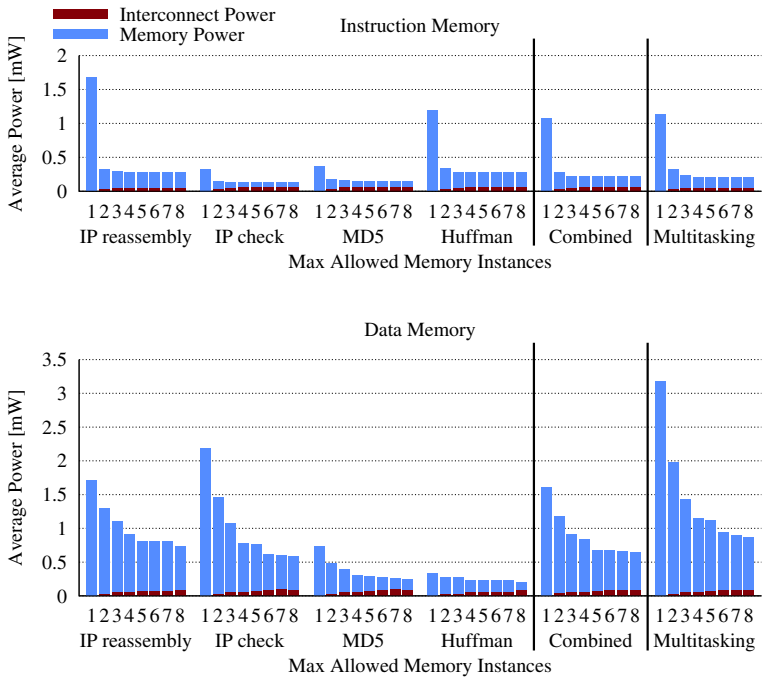

Fig. 4 Power consumptions of instruction memory (*top*) and data memory (*bottom*) with varying limits for memory instances

Table 2 Memory power consumption details for instruction memory (IP reassembly) and data memory (MD5)

Instruction memory (IP reassembly)

Mems	Num	Size	Banks	P (mW)	Reads (%)	Funcs
1	1	64K	16	1.6708	100	241
4	–	–	–	0.2856	–	–
	1	512	1	0.2057	95.1	9
	1	2K	2	0.0108	3.1	9
	1	4K	2	0.0081	1.7	8
	1	32K	16	0.0022	0.1	215
	Interconnect			0.0588		

Data memory (MD5)

Mems	Num	Size	Banks	P[mW]	Reads (%)	Writes (%)	Objs
1	1	2M	16	0.7350	100	100	37
8	–	–	–	0.2442	–	–	–
	1	2K	2	0.0081	33.4	30.9	33
	2	8K	8	0.0068	20.4	0	1
	5	256K	16	0.1412	46.2	69.1	3
	Interconnect			0.0882			

it features a highly non-uniform memory access pattern. Here, only 9 of 241 functions make up 95.1 % of all instruction fetches. Consequently, the major power reduction is achieved by moving these 9 functions into a separate, small 512-byte memory. This clearly shows that separating the most frequent functions into low read-power memories significantly reduces overall power consumption (in this case, 82.9 % considering interconnect power).

For data memory, the causes for power reduction are not as evident because a large number of reads and writes address the very large heap (up to almost 2 Mbytes). Thus, power reduction is mainly achieved by spreading the heap across 5 equally-sized, smaller memories. However, 33 fairly small objects still account for 32 % of all reads and writes and thus have been moved to a small, low power memory.

Also shown in Fig. 4 are the results for the *Combined* optimization of all applications. The savings for the average power consumption in this operation mode amount to 79.9 % in the mean for instruction memory. This large power reduction is attributed to the relatively inefficient reference case $\text{mems}_{\max} = 1$. Since the size of the single memory is determined by the largest application, the power consumption for applications with a small memory footprint increases. As a result, already using two memory instances significantly reduces power consumption as the most frequent functions no longer reside in the large memory with a high read power consumption. For data memory, the power consumption can be reduced to 59.7 %, which is even better than the average power reduction of 58.7 % when performing individual optimizations (*Single-App*). Again, this is attributed to all applications using a large heap dominating the memory organization and thus power consumption.

The investigation of the computed power figures for the *Multitasking* operation mode (cf. Fig. 4) reveals a maximum power consumption reduction of 81.5 % in case of instruction memory. As for the *Combined* setup, an increase of the $\text{mems}_{\max}$ constraint from 1 to 2 allowed memory instances already results in an improvement of over 70 %. Once more, this is due to a particularly inefficient reference case with a single unpartitioned memory that has to fit the sum of all memory footprints of all applications in case of *Multitasking*. This aspect also influences the optimization of data memory where a significant overall improvement of 72.5 % is obtained. This value is close to the best power reduction of all evaluated RAM scenarios as achieved for the IP check benchmark in *Single-App* operation mode with 73.2 %.

Altogether for the aspect of power consumption, an average improvement of 73.7 % for instruction memory and 61.2 % for data memory can be presented through all evaluated scenarios. Compared to results from literature, these values constitute a solid improvement. The authors of [1] provide average power savings of 17.8 % for instruction and 47.8 % for data memory in comparison with a single-banked memory configuration. Benini et al. [2] consider SRAM uniformly, i.e., without clearly distinguishing between code and data sections, and state an average improvement of 41.7% versus monolithic memory. With savings between 44.7 and 64.8 % [12], respectively, 59.2 and 73.9 % [10], there is also related work that yields comparable or even slightly better saving ratios than our approach. However, note that both methods take memory components with retention mode into account. Accordingly, idle memories are put into sleep state with negligible leakage power consumption, which is considered as beneficial factor. For that reason, a direct comparison of these approaches with our work is not possible.

6.2 Area requirement

In order to discuss the aspect of area requirement, Fig. 5 depicts the area footprints in mm^2 that correspond to the optimal power configurations as given in Fig. 4. Interestingly, we can observe that power consumption and area are not correlated. The assumption of an increasing area requirement with increasing number of utilized memory instances, as mentioned in the introduction (cf. Section 1), is consequently disproved. However, please note that only memories of size 2^N are available to the ILP solver. Depending on the evaluated scenario, we can observe that the area requirement either improves over the number of allowed memory instances (IP check, data

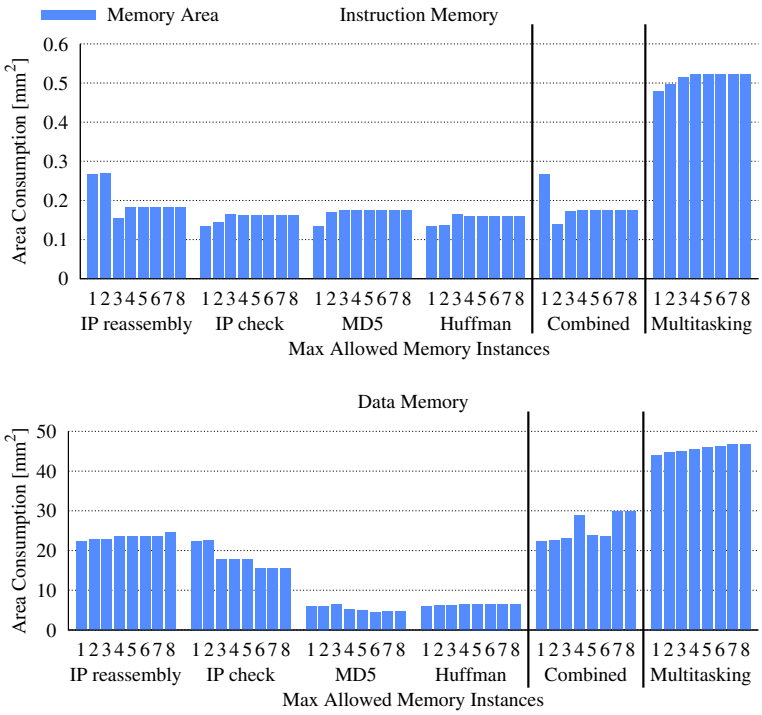

Fig. 5 Area requirement of instruction memory (*top*) and data memory (*bottom*) with varying limits for memory instances

memory), deteriorates (*Multitasking*), or does not follow any trend at all (*Combined* operation mode). Hence, area should also be a free variable in the optimization process.

The area impact of the interconnect fabric for up to eight connected subscribers amounts in any case to less than 0.002 mm^2 and is therefore considered as negligible and not illustrated in Fig. 5.

Averaging all experiments in the range of 1 to 8 memories yields a deterioration of area consumption by 2.1 % for instruction memory and 1.2 % in case of data memory. In comparison, the partitioning method of Mai et al. [1] introduces an area overhead of 80.33 % for instruction and 44.64 % in case of data memory. With respect to these values and the accompanied considerable improvements in power consumption (cf. Section 6.1), the presented small increase in on-chip area emphasizes the strength and relevance of our approach.

Altogether, finding the optimal power/area trade-off is not trivial. Therefore, especially in power and area critical designs, an efficient design space exploration is crucial, in order to reduce effort and costs.

6.3 Pareto optimality

Even though our model is basically designed for power optimization, we are also able to obtain area optimal results from it. To this end, the power cost function (cf. Eq. 27 respectively Eq. 28 in Section 4) is replaced by an area cost function that is derived from Eq. 19. The former power cost function is further incorporated as user constraint. In this setup, another experimental series was carried out together with the ILP solver minimizing area requirement. With the above-presented results, we get one solution with minimal power, and one solution with minimal area requirement per evaluated scenario. These two results span up a solution range in the two-dimensional power/area design space. All other obtained configurations represent a trade-off in one or the other dimension and therefore always reside between the borders of our solution range. Through variation of user constraints, i.e., maximum number of allowed memories ($\text{mems}_{\max}$), maximum power consumption ($P_{\max}$), respectively, area budget ($\text{area}_{\max}$), we are able to influence the ILP solver in order to obtain even more valid implementations. Eventually, the exploration of the resulting design space allows the identification of pareto-optimal solutions. All implementations that belong to this solution subset represent a trade-off that is not dominated by any other solution in the design space, i.e., a better value for one criterion is automatically bound to a worsening on the second criterion. The curve that connects all pareto-optimal points is referred to as pareto front.

As an example, design space and pareto front are illustrated for the IP check instruction memory in Fig. 6. Single marks represent all valid implementations as obtained from the optimization process; however, only the solutions on the pareto front are of actual relevance. The

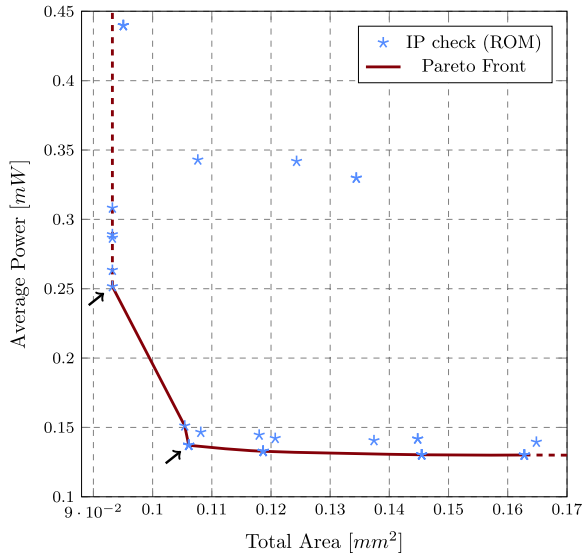

Fig. 6 Power/area design space exploration example for instruction memory (IP check)

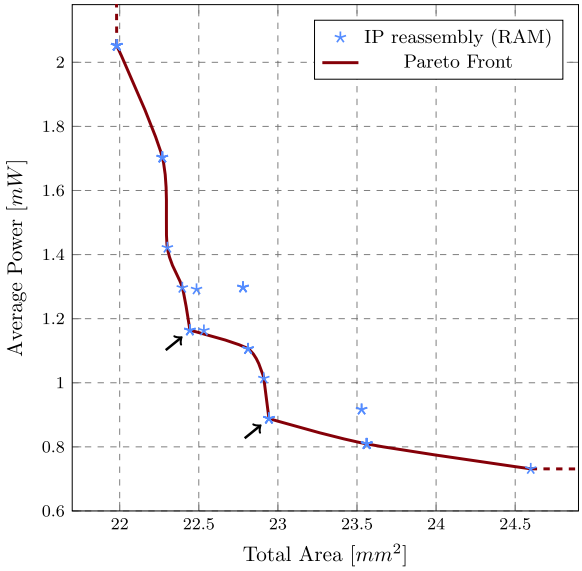

Fig. 7 Power/area design space exploration example for data memory (IP reassembly)

highlighted points on this curve are especially interesting as they identify local extrema and thus mark the most reasonable solutions to choose from. For the example illustrated in Fig. 6, two such points can be identified. One is more preferable in terms of area whereas the other is superior in terms of power consumption. Further observation reveals that, except for three solutions in the middle of the depicted range, all other points are either identical or close to the minimum on one criterion, which facilitates the selection process significantly.

In the second example, as given in Fig. 7, the individual memory configurations are more distributed and not as close to the solution range borders as in the previous example. Exploration of this design space for IP reassembly data memory nevertheless also reveals two local extrema that identify the most reasonable configurations on the pareto front.

From Sections 6.1 and 6.2, we already know that power and area do not correlate. The presented combined consideration of both criteria in a pareto investigation consequently closes the gap of finding a reasonable trade-off configuration.

6.4 ILP performance

To show that our model is fast enough to be used in an optimized synthesis flow, we measured the execution times of the ILP solver for the different problems. We used the IBM CPLEX ILP solver on an Intel^{TM} E5-2660 v2 Xeon System clocked at 2.2 GHz and having 128 GB RAM. The individual execution times of the most complex cases ($\text{mems}_{\max} = 8$) are listed in Table 3.

Optimizing individual applications had average execution times of 1.95 s for RAM models and 21.7 s for ROM models, which proves the efficiency of our approach. We attribute the high performance to the fact that we do not map application profiles to individual memory instances but only to memory types and the ILP solver chooses the ideal number of instances. The consideration of multiple applications in the other operation modes results in more ILP variables and thus in a slight increase of execution times, e.g., for the *Multitasking* operation mode to 6.71 s for the RAM model and 138.13 s in case of ROM. Strikingly, even more time is required for the *Combined* optimization processes. With 19.92 s for data memory, we are still in a reasonable range; however, 97.36 min for instruction memory appears to be disproportionately large, compared to the other values. The main explanation for this lengthy execution time is the depletion of the host machine's main memory. Even though 128 GB constitutes a considerable amount of RAM resources, some optimization problems go beyond this scope, which results in swap

Table 3 ILP execution times for eight allowed memory instances

Benchmark	Instr. Mem	Data Mem
IP reassembly	47.47 s	0.25 s
IP check	16.48 s	2.37 s
MD5	9.24 s	1.19 s
Huffman	13.55 s	3.97 s
Combined	97.36 min	19.92 s
Multitasking	138.13 s	6.71 s

operations to the file system. This severely slows down the whole optimization process, which results in execution times in the range of over an hour, as for the example above. Nevertheless, as our main focus lies on application specific and highly optimized systems-on-chips we still consider those times as reasonable commitment in exchange for a power-optimal memory configuration.

7 Conclusions

In this article, we have provided a mathematical model to determine an optimal on-chip memory organization with respect to low power. The model is highly flexible, allowing applications to be modeled with different degrees of precision, supports optimization for multiple applications at the same time, and works with a large list of memory types. A particular advantage of our approach is given by the ability to provide optimal allocation and mapping at once. Hence, number and type of memory instances as well as mapping of address space ranges to the allocated set of memories can be obtained from one and the same workflow. Achieved power savings of up to 82 % for instruction memory and 73 % for data memory in a set of industrial grade benchmarks prove the benefits of our model. Its value is further emphasized through a particularly small deterioration of area requirement that is bound to these power savings. On average, additional area of 2.1 % for instruction memory and 1.2 % for data memory is required only. Furthermore, we showed how our model can be used for efficient design space exploration in the power/area domain and, on top of that, how an incorporation into an automated synthesis flow makes it a valuable tool in low power HW/SW codesign.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
We would like to thank Nouman Naim Hasan for his valuable input on modeling the memory utilization of applications.

Received: 23 January 2016 Accepted: 20 June 2016
Published online: 07 July 2016

References

1. S Mai, C Zhang, Y Zhao, J Chao, Z Wang, in *Proc. International Conference on ASIC (ASICON)*. An application-specific memory partitioning method for low power (IEEE, Guilin, China, 2007)
2. L Benini, A Macii, M Poncino, in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*. A recursive algorithm for low power memory partitioning (IEEE, Rapallo, Italy, 2000)
3. S Krishnamoorthy, U Catalyurek, J Nieplocha, A Rountev, P Sadayappan, in *Proc. The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*. Hypergraph Partitioning for Automatic Memory Hierarchy Management (IEEE, Tampa, FL, USA, 2006)
4. F Menichelli, M Olivieri, Static minimization of total energy consumption in memory subsystem for scratchpad-based systems-on-chips. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **17**(2), 161–171 (2009)
5. S Pasricha, ND Dutt, A framework for cosynthesis of memory and communication architectures for MPSoC. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD). **26**(3), 408–420 (2007)
6. S Steinke, L Wehmeyer, B-S Lee, P Marwedel, in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Assigning program and data objects to scratchpad for energy reduction (IEEE, Paris, France, 2002), pp. 409–415
7. SL Coumeri, DE Thomas, in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*. Memory modeling for system synthesis (IEEE, Montery, CA, USA, 1998), pp. 179–184
8. S Srinivasan, F Angiolini, M Rugiero, L Benini, V Narayanan, in *Proc. SOC Conference*. Simultaneous memory and bus partitioning for SoC architectures (IEEE, Herndon, VA, USA, 2005)
9. Q Zhuge, EH-M Sha, B Xiao, C Chantrapornchai, Efficient variable partitioning and scheduling for DSP processors with multiple memory modules. IEEE Trans. Signal Process. (SP). **52**(4), 1090–1099 (2004)
10. M Loghi, O Golubeva, E Macii, M Poncino, Architectural leakage power minimization of scratchpad memories by application-driven subbanking. IEEE Trans. Comput. **59**(7), 891–904 (2010)
11. T Liu, Y Zhao, CJ Xue, M Li, Power-aware variable partitioning for DSPs with hybrid PRAM and DRAM main memory. IEEE Trans. Signal Process. **61**(14), 3509–3520 (2013)
12. L Steinfeld, M Ritt, F Silveira, L Carro, in *Proc. IFIP TC 10 Int'l Embedded Systems Symposium (IESS)*. Low power processors require effective memory partitioning (Springer, Paderborn, Germany, 2013)
13. F Angiolini, L Benini, A Caprara, An efficient profile-based algorithm for scratchpad memory partitioning. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD). **24**(11), 1660–1676 (2005)
14. H Takase, H Tomiyama, G Zeng, H Takada, in *Proc. Embedded Software and Systems (ICESS)*. Energy efficiency of scratch-pad memory at 65 nm and below: an empirical study (IEEE, Sichuan, 2008)
15. A Kannan, A Shrivastava, A Pabalkar, J-E Lee, in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*. A software solution for dynamic stack management on scratch pad memory (IEEE, Yokohama, 2009)
16. CJ Seung, A Shrivastava, K Bai, Dynamic code mapping for limited local memory systems. Proc. IEEE Int'l Conf. on Application-specific Systems Architectures and Processors (ASAP) (2010)
17. A Shrivastava, A Kannan, J Lee, A Software-Only Solution to Use Scratch Pads for Stack Data. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD). **28**, 1719–1727 (2009)
18. F Balasa, CV Gingu, II Luican, H Zhu, in *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*. Design space exploration for low-power memory systems in embedded signal processing applications (IEEE, Taipei, 2013)
19. F Sampaio, M Shafique, B Zatt, S Bampi, J Henkel, in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. dSVM: Energy-efficient distributed Scratchpad Video Memory Architecture for the next-generation High Efficiency Video Coding (IEEE, Dresden, Germany, 2014)
20. B Zatt, M Shafique, S Bampi, J Henkel, A low power memory architecture with application-aware power management for motion & disparity estimation in Multiview Video Coding. Proc. IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD) (2011)
21. STMicroelectronics, M48Z35 256 Kbit (32 Kbit x 8) SRAM Datasheet, 2011. http://www.st.com/web/en/resource/technical/document/datasheet/CD00000550.pdf. Last visited on 01/11/2016
22. EEMBC, EEMBC Multibench 1.0 Multicore Benchmark Software. http://www.eembc.org/benchmark/multi_sl.php. Last visited on 01/11/2016
23. SOCLIB Consortium, Projet SOCLIB: Plate-forme de modélisation et de simulation de systèmes integrés sur puce (SOCLIB project: An integrated system-on-chip modelling and simulation platform) Technical report, CNRS, 2003. http://www.soclib.fr/
24. N Muralimanohar, R Balasubramonian, NP Jouppi, CACTI 6.0: A Tool to Model Large Caches. HP Laboratories, HPL-2009-85 (2009). http://www.hpl.hp.com/techreports/2009/HPL-2009-85.pdf
25. NanGate Inc, NanGate FreePDK45 Open Cell Library. http://www.nangate.com/?page_id=2325. Last visited on 01/11/2016