EURASIP Journal on Applied Signal Processing 2005:16, 2641–2654

SoftExplorer: Estimating and Optimizing the Power and Energy Consumption of a C Program for DSP Applications

Eric Senn
LESTER, University of South-Brittany, BP 92116, 56321 Lorient Cedex, France
Email: eric.senn@univ-ubs.fr

Johann Laurent
LESTER, University of South-Brittany, BP 92116, 56321 Lorient Cedex, France
Email: johann.laurent@univ-ubs.fr

Nathalie Julien
LESTER, University of South-Brittany, BP 92116, 56321 Lorient Cedex, France
Email: nathalie.julien@univ-ubs.fr

Eric Martin
LESTER, University of South-Brittany, BP 92116, 56321 Lorient Cedex, France
Email: eric.martin@univ-ubs.fr

Received 30 January 2004; Revised 20 October 2004

We present a method to estimate the power and energy consumption of an algorithm directly from the C program. Three models are involved: a model for the targeted processor (the power model), a model for the algorithm, and a model for the compiler (the prediction model). A functional-level power analysis is performed to obtain the power model. Five power models have been developed so far, for different architectures, from the simple RISC ARM7 to the very complex VLIW DSP TI C64. Important phenomena are taken into account, like cache misses, pipeline stalls, and internal/external memory accesses. The model for the algorithm expresses the algorithm's influence over the processor's activity. The prediction model represents the behavior of the compiler, and how it will allow the algorithm to use the processor's resources. The data mapping is considered at that stage. We have developed a tool, SoftExplorer, which performs estimation both at the C-level and the assembly level. Estimations are performed on real-life digital signal processing applications with average errors of 4.2% at the C-level and 1.8% at the assembly level. We present how SoftExplorer can be used to optimize the consumption of an application. We first show how to find the best data mapping for an algorithm. Then we demonstrate a method to choose the processor and its operating frequency in order to minimize the global energy consumption.

Keywords and phrases: power, energy, estimation, optimization, C program, DSP applications.

1. INTRODUCTION

Lowering the power consumption of today's electronic devices is more than ever a crucial challenge. Indeed, the market of mobile devices has exploded those last years: laptop computers, pocket PC, tablet PC, PDA, mobile phones, and so forth. It is remarkable that there is less and less difference between all these devices. With mobile phones, you can take pictures, record movies, or surf the internet as easily as with any laptop. A common point between all those new multimedia materials is the necessity to comply with the Universal Mobile Telecommunication Systems (UMTS) norm [1]. UMTS was adopted as the international norm for mobile telecommunication systems of the third generation (3G). It defines a set of high-speed services: voice, image, file transfer, fax, videoconference, as well as the capability to connect to any kind of network from any kind of place. Flexibility and reconfigurability will be necessary to achieve such a universal mobility. But powering becomes more and more difficult as the demand on processing power increases [2]. Battery life is now a very strategic feature for every mobile system.

There are many approaches for dealing with this power consumption problem. Basically, it is possible to distinguish

methods working on the hardware, or on the software, or methods trying to fit to both of them as well as possible. At the hardware level, power management IC or IP are now integrated in every system [3]. Improvements in the semiconductor industry are very promising. They are absolutely necessary to counterbalance the increasing static power consumption in the last VLSI chips. The two main approaches rely on the reduction of the power voltage, and on the reduction of the transistor's threshold voltage. In parallel, it is also proposed to dynamically control these parameters, together with the operating frequency, depending on the chip's current activity.

At the software level, a lot of optimizations can be conducted, with a very strong impact on the system's power consumption [4]. A lot of loop transformations were presented [5, 6, 7, 8], and the impact of the data organization on memory was studied [9, 10, 11]. Source code transformations were also proposed, in parallel with the data and memory organization [12], and the data transfer and storage exploration [13, 14]. In these works, responses to the following essential questions are sought : Which architecture to use for the system memory ? Where to place the data in this memory ? Which transformations to the program to perform to optimize the transfer of data in memory?

The fact is that the codesign step can lead to many solutions: there are many ways of partitioning a system and many ways of writing the software even once the hardware is chosen, and they will give very different consumptions in the end. To find the best solution is not obvious. Indeed, it is not enough to know whether the application's constraints are met or not; it is also necessary to be able to compare several solutions to seek the best one. So, the designer needs fast and accurate tools to evaluate a design, and to guide him through the design space. Without such a tool, the application's power consumption would only be known by physical measurement at the very last stage of the design process. That involves buying the targeted processor, the associated development tools and evaluation board, together with expansive devices to measure the supply current consumption, and that finally expands the time-to-market.

There are several methods to estimate a processor's power consumption, which we find at different levels in the modelling and analysis tool flow in microprocessor design. Lower-level power estimation tools work at the circuit level, like *QuickPower* (at the gate level) [15] and *PowerMill* (at the transistor level) [16]. They are accurate but actually not usable for making architectural decisions. Microarchitectural power estimators work mainly at the cycle level or at the instruction level. The tools *Wattch* [17] and *SimplePower* [**?**] are based on cycle-level simulations. They rely on analytical capacitance models which have to be developed for every block in the processor (ALUs, caches, buses, registers, RAMs, CAMs, etc.). Each model involves a large number of parameters sometimes difficult to determine for the microarchitect (related to the circuit or physical design) [19]. In the tool *PowerTimer* [20], the number and complexity of parameters were reduced. Microarchitecture-level power analysis tools are used with success in the design of microprocessors: for example *PowerTimer* to determine the depth of a pipeline [21], or *Wattch* to design a new issue queue for reusable instructions [22]. They are however not very useful for making decisions at the algorithmic level. Indeed, at the cycle level, the processor's behavior is simulated cycle by cycle. This is not a problem when only a small portion of the code (a few instructions) is simulated, but this may be very time consuming for large programs. Moreover, cycle-level simulations necessitate a low-level description of the architecture. This could be not too difficult to obtain for simple architectures, or for only a subpart of a more complex microarchitecture, but such a description is often unavailable for off-the-shelf processors. The difficulty to obtain a model increases with the ever-growing complexity of nowadays processors. For instance, the model proposed in *SimplePower* is limited to an in-order 5-stage pipelined datapath, with perfect cache—the energy consumed by the control unit and the clock generation and distribution is not considered.

Another approach is to evaluate the power consumption with an instruction-level power analysis (ILPA) [23]. This method relies on current measurements for each instruction and couple of successive instructions. Even if it proved accurate for simple processors, the number of measures needed to obtain a model for a complex architecture would become unrealistic [24]. Moreover, the instruction-level models should be improved to take into account pipelined architectures, large VLIW instruction sets, and internal data and instruction caches. Recent studies have introduced a functional approach [25, 26], but few works are considering VLIW processors and the possibility for pipeline stalls [27]. All these methods perform power estimation only at the assembly level with an accuracy from 4% for simple cases to 10% when both parallelism and pipeline stalls are effectively considered. As far as we know, only one unsuccessful attempt of algorithmic estimation has already been made [28].

To evaluate the impact of high-level transformations, we propose to estimate the application's consumption at the early stages in the design flow. We demonstrate that an accurate estimation of an algorithm's power consumption can be conducted directly from the C program without execution. Our estimation method relies on a *power model* of the targeted processor, elaborated during the *model definition* step. This model definition is based on the *functional-level power analysis* (FLPA) of the processor's architecture [29]. During this analysis, functional blocks are identified, and the consumption of each block is characterized by physical measurements. Once the power model is elaborated, the *estimation process* consists in extracting the values of a few parameters from the code; these values are injected in the power model to compute the power consumption. The estimation process is very fast since it relies on a static profiling of the code. Several targets can be evaluated as long as several power models are available in the library.

In this paper, we will neither focus on the model definition nor on the FLPA, which have been already extensively discussed in former publications [30, 31]. We will rather focus on the use of these power models to actually estimate and optimize the power and energy consumption of a code.

Our method and models are integrated in our power and energy estimation tool: SoftExplorer. There are currently 5 power models available, for the 5 following processors: the Texas Instrument C67, C64, C62, and C55, and the ARM7.

In this list, only the C55 is a low-power processor. This DSP is currently widely used in mobile devices, associated to the ARM7 or ARM9 in OMAP chips. The next generation of mobile devices however will inevitably require more powerful processing capabilities, which will imply more complex processors. Indeed, the C55 is already aided by video coprocessors for 2.5 G/3G applications. More complex DSPs will be used, like the C67/64/62, or low-power versions of these architectures (it is remarkable the C64 has already coprocessors included in it). In this paper, we are presenting a methodology that was successfully applied, not only on the C55 and ARM7, but also on the more complex architectures of the C67/64 and 62. We thereby demonstrate the applicability of our methodology upon a large class of architectures, and we also guarantee that it will still be usable with the future generations of processors. In accordance with this idea, we also present some interesting applications of the methodology not only on the C55, but also on the other processors for which we have developed a power model.

Provided that the power model of the processor is available, there is no need for owning the processor itself to estimate its consumption, nor for any specific development tool, since it is not even necessary to have the code compiled. Thus, given an algorithm, a fast and cheap comparison of different processors is possible. It is also possible to compare different algorithms, or different ways of writing an algorithm for a given application. The designer can locate which parts of the program are the most consuming, and focus his/her optimization effort on these parts. Our method takes into account another very important parameter for the power consumption: the memory mapping. Indeed, the designer can place the data in external or internal memory, and can choose the internal bank where the data is stored. The place of data has a very strong impact on the consumption, and the designer is able to try and compare different data mappings with the help of our tool.

2. MODEL DEFINITION

2.1. The functional-level power analysis

Our estimation is based on the functional-level power analysis (FLPA) of a processor. As stated before, the main advantage of this method compared to the ILPA is its simplicity, and the rapidity that it involves, both for building the model of a target, and for the estimation of a code. We recall here the FLPA general principle.

To perform a FLPA implies first to divide the processor in functional blocks. Functional blocks gather hardware resources that are activated together during a run. Obviously, relatively nonconsuming parts of the processor have to be discarded at that stage. Secondly, we have to find which features of the application impact the functional blocks' activity, and thus the power consumption. These features are formalized in two sets of parameters: the *algorithmic parameters* and the *architectural parameters*. Algorithmic parameters indicate the activity level between every functional block in the processor (parallelism rate, cache miss rate, etc.). Architectural parameters depend on the processor configuration which is explicitly defined by the programmer (clock frequency, data mapping, etc.). Then, it is necessary to find how the functional blocks' consumption varies with the parameters' values. We make the parameters vary with the help of elementary assembly programs elaborated to stimulate each block or subblock separately. The variations of the processor's supply current are measured on an evaluation board. A curve fitting of the graphical representation of these variations finally permits to determine the *consumption rules* by regression. The set of consumption rules for a given processor constitutes the so-called *power model* of this processor. The three main steps of the FLPA are summarized in Figure 1.

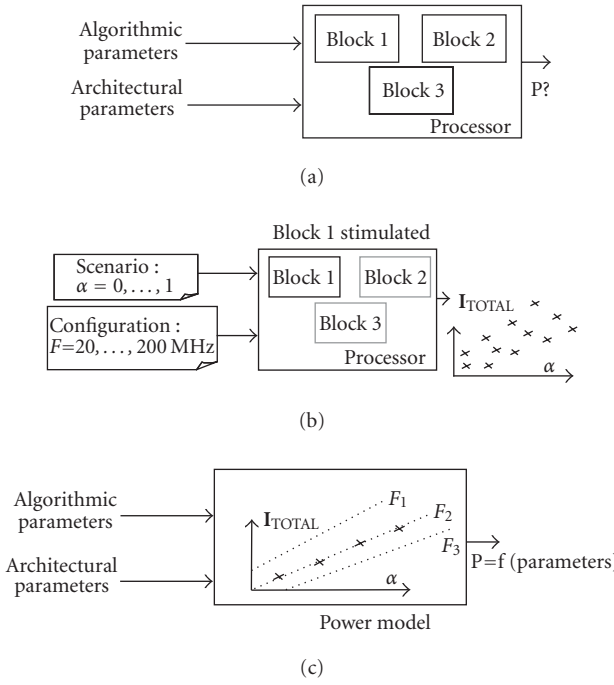

Figure 1: FLPA overview.

As stated before, we have developed a power model for five processors: the Texas Instrument C67, C64, C62, and C55, and the ARM7; and we have integrated these power models in our power and energy estimation tool SoftExplorer.

2.2. TI C62 and C67 power models

The TI C62 and C67 processors have complex architectures. Indeed, they both have a VLIW instructions set, a deep pipeline (up to 15 stages), and parallelism capabilities (up to 8 operations in parallel). Their internal program memory can be used like a cache in several modes, and an *external memory interface* (EMIF) is used to load and store data and program from the external memory [32]. Apart from the fact that the C62 operates in fixed point and the C67 in floating

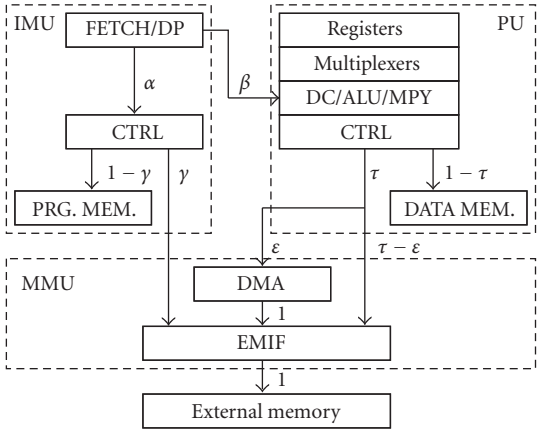

Figure 2: FLPA for the C62 and C67.

point, they have very similar architectures. As a result, the functional analysis is identical for both of them, and leads to the block diagram in Figure 2.

Three blocks and five algorithmic parameters are identified. The parallelism rate α assesses the flow between the FETCH stages and the internal program memory controller inside the IMU (instruction management unit). The processing rate β between the IMU and the PU (processing unit) represents the utilization rate of the processing units (ALU, MPY). The activity rate between the IMU and the MMU (memory management unit) is expressed by the program cache miss rate γ. The parameter τ corresponds to the external data memory access rate. The parameter ε stands for the activity rate between the data memory controller and the direct memory access (DMA). The DMA may be used for fast transfer of bulky data blocks from the external to the internal memory ($\varepsilon = 0$ if the DMA is not used).

To the former algorithmic parameters four *architectural parameters* are added, that also strongly impact the processor's consumption: the clock frequency (F), the memory mode (MM), the data mapping (DM), and the data width during DMA (W).

The influence of F is obvious. The C62 and C67 maximum frequencies are respectively 200 MHz and 167 MHz, but the designer can tweak this parameter to adjust consumption and performances.

The memory mode MM illustrates the way the internal program memory is used. Four modes are available. All the instructions are in the internal memory in the *mapped mode* (MM_M). They are in the external memory in the *bypass mode* (MM_B). In the *cache mode*, the internal memory is used like a direct mapped cache (MM_C), as well as in the *freeze mode* where no writing in the cache is allowed (MM_F). Internal logic components used to fetch instructions (for instance tag comparison in cache mode) actually depend on the memory mode, and so the consumption.

The data mapping impacts the processor's consumption for two reasons. First, the logic involved to access a data in internal or in external memory is different. Secondly, whenever a data has to be loaded or stored in the external memory, or whenever two data in the same internal memory bank are accessed at the same time, the pipeline is stalled and that really changes the power consumption. Pipeline stalls are counted in the *pipeline stall rate* (*PSR*). Like τ, DM is included in the *PSR* in the power model. Table 1 summarizes the set of parameters for the five considered processors.

Table 1: Sets of parameters.

Parameters	C67	C64	C62	C55	ARM7
α	X	X	X	—	—
β	X	X	X	X	—
γ	X	X	X	X	—
τ	X	—	X	—	—
ε	X	X	X	X	—
μ	—	X	—	—	—
σ	—	X	—	—	—
δ	—	X	—	—	—
PSR	X	X	X	—	—
W	X	X	X	X	—
F	X	X	X	X	X
MM	X	X	X	X	X
DM	X	X	X	—	—
PM	—	—	—	X	—

Even if the C62 and C67 share the same set of parameters, the consumption rules that link these parameters to their actual power consumption are different. Indeed, the C62 is built upon a 0.25 μm process, and the C67 upon a 0.18 μm one. The supply voltage is 2.5 V for the C62, and 1.8 V for the C67. They therefore do not have identical static and dynamic power consumptions, and ought to have different consumption rules and hence, power models.

For these processors, no significant difference in power consumption was observed between an addition and a multiplication, or a read and a write in the internal memory. Moreover, the effect of data correlation on the global power consumption appeared lower than 2%. More details on the consumption rules and their determination can be found in [31].

2.3. TI C64 power model

There are significant differences between the C64 and the C62/67 architectures. The internal program and data memories have been replaced by two level-1 caches—the former memory modes have disappeared. A single level-2 cache is used both for the data and program. SIMD instructions can be used. The number of registers is doubled ($2*32$ in place of $2*16$), as well as the number of DMA. Two coprocessors have been added (Viterbi + turbo decoder), and the C64 maximum frequency goes from 600 MHz to 1 GHz. As a result, the power model is slightly different. As shown in Table 1, three new parameters were added. The miss rate for the level-1 data cache is μ. Even if $\mu = 0$, the consumption may vary with the number of data reads or writes. In the C64, data are directly written into the level-2 cache and the power consumption is different for writes and reads. It is thus necessary to use two

different parameters: σ for the data read rate, and δ for the data write rate. The power consumption of the two coprocessors has not been included in the power model yet.

2.4. TI C55 power model

The TI C55 is a low-power processor with a fixed-point architecture that can only execute two instructions in parallel; however, it only fetches one instruction at each clock cycle, and pipeline stalls never occur. Another characteristic of this processor is the possibility to automatically idle some parts of its architecture if unused. This is integrated in the parameter PM (power management), which indicates the units in the sleep mode. The C55 internal program memory can be used in the same four modes as the C6x and it also contains a DMA and an EMIF. Because the C55's architecture is less complex than the C6x's, its power model has less parameters (see Table 1).

2.5. ARM7 power model

The ARM7TDMI is the simplest processor that we have modeled yet. It has a scalar architecture and its internal program memory can be used in three modes (mapped, cache, and bypass). Previous works on the StrongArm have established that the power consumption essentially depends on the clock frequency and the supply voltage [33]. Our own consumption measurements on the ARM7TDMI have fully validated this trend: the power consumption variations corresponding to various programs are under 8% of the global consumption. No algorithmic parameter is then required to model the ARM7, as represented in Table 1.

The time necessary to complete the model of the C62 was about 30 days, and it took 15 days for the ARM7. It went faster for the C67 and C55 for we took benefit from the previous study of the very similar C62 architecture. The use of an instruction-level method for such a complex architecture would have conducted to a prohibitive number of measurements. Indeed, with an ILPA approach, Bona et al. have characterized a simpler VLIW processor (the Lx) in 108 days [34].

3. ESTIMATION PROCESS

3.1. Prediction models

To compute the power consumption of an application, it is necessary to determine the parameters which are used in the target's power model. To get the energy consumption, we will see later how to evaluate the execution time of the algorithm on the target. The process of finding the values of an application's parameters gets difficult as the number of parameters in the power model increases. It is very simple for the ARM7 where only the frequency and memory mode are needed, but more difficult with the other processors. Determining the architectural parameters is straightforward since they are fixed by the programmer, or dependant on the architecture. To find the value of the algorithmic parameters implies a more precise knowledge of the processor's behavior. However, the parameters we have proposed are general enough to be used for any pipeline and/or superscalar architecture when necessary, and are extracted from the application according to the following principles. We will now explain these principles on the most complex architectures that we have studied yet, according to Table 1, the TI C62 and C67.

Before getting to the C-level, we will first observe what happens when an assembly program is executed on those processors. In the C6x, eight instructions are fetched at the same time. They form a *fetch packet*. In this fetch packet, operations are gathered in *execution packets* depending on the available resources and the parallelism capabilities. The parallelism rate α can be computed by dividing the number of fetch packets (NFP) by the number of execution packets (NEP) counted in the assembly code. However, the effective parallelism rate is drastically reduced whenever the pipeline stalls. Therefore, the final value for α must take the number of pipeline stalls into account. Hence, a *pipeline stall rate* (PSR) is defined, and α is computed as follows:

$$\alpha = \frac{NFP}{NEP} \times (1 - PSR). \tag{1}$$

Identically, the PSR is considered to compute the processing rate β, with NPU the average number of processing units used per cycle (counted in the code), and NPU_{MAX} the maximum number of processing units that can be used at the same time in the processor ($NPU_{MAX} = 8$ for the C6x):

$$\beta = \frac{1}{NPU_{MAX}} \times \frac{NPU}{NEP} \times (1 - PSR). \tag{2}$$

To determine α and β at the C-level, the three parameters NFP, NEP, and NPU must be predicted from the algorithm (instead of being counted in the assembly code). Indeed, even if our tool was initially designed to estimate the power consumption from an assembly code, the challenge here is to do it from the C program. It is clear that the prediction of NFP, NEP and NPU must rely on a model that anticipates the way the assembly code is executed on the target. This is actually related to the compiler behavior, and to the options chosen for the compilation. According to the processor's architecture and with a little knowledge of the compiler, four *prediction models* were defined.

(i) The *sequential model* (SEQ) is the simplest one since it assumes that all the operations are executed sequentially. This model is only realistic for nonparallel processors. However, it provides the absolute minimum bound of the algorithm's power consumption.

(ii) The *maximum model* (MAX) corresponds to the case where the compiler fully exploits all the architecture possibilities. With this model, we assume that the maximum number of operations executable in parallel in a superscalar processor are indeed executed in parallel. In the C6x, 8 operations can be done in parallel; for example 2 loads, 4 additions, and 2 multiplications, in one clock cycle. This model gives a maximum bound of the consumption. It will be also referred to as the "FULL_parallel" model.

(iii) The *minimum model* (MIN) is more restrictive than the previous model since it assumes that load instructions,

Table 2: Prediction models for the example.

Model	EP1	EP2	EP3	EP4	$\alpha = \beta$
MAX	2LD	2LD, 4OP	—	—	0.5
MIN	1LD	1LD	1LD	1LD, 4OP	0.25
DATA	2LD	1LD	1LD, 4OP	—	0.33

or store instructions, are never executed at the same time—indeed, it was noticed on the compiled code that all parallelism capabilities were not always fully exploited for these instructions, depending on the compilation options. This is especially the case when the compiler is settled to minimize the size of the assembly code. The data mapping is analyzed in this case only to assess the right value for the *PSR* (see Section 3.2). This model will be also referred to as the "SIZE_optimal" model. It will give a more realistic lower bound for the algorithm's power consumption than the sequential model.

(iv) At last, the *data model* (DATA) refines the prediction for load and store instructions. The only difference from the *MAX* model is that it allows parallel loads and stores only if they involve data from different memory banks. In the C6x, for instance, there are two banks in the internal data memory which can be accessed in one clock cycle. It is thus possible to load two data in one cycle if one data is in the first bank, and the other data in the second one. The place of a data in the memory is found in the data mapping file, which, as before, will be also used to determine the *PSR* (Section 3.2). Such a behavior is observed when the compiler is settled to minimize the execution time of the code. This model will be also referred to as the "TIME_optimal" model.

The prediction is performed by applying those models on all the program. As illustration, we present below a simple example:

```
For (i=0; i<512; i++)
{Y=X[i]*(H[i]+H[i+1]+H[i-1])+Y;}
```

In this loop nest, there are 4 loads (LD) and 4 other operations (OP): 1 multiplication and 3 additions. In our example, Y is stored in a register inside the processor. Here, our 8 operations will always be gathered in one single fetch packet, so $NFP = 1$. Because no NOP operation is involved, $NPU = 8$ and α and β parameters have the same value. In the SEQ model, instructions are assumed to be executed sequentially. Then $NEP = 8$, and $\alpha = \beta = 0.125$. Results for the other models are summarized in Table 2. X and H are supposed to be in distinct memory banks.

Of course, realistic cases are more elaborated: the parameters prediction is done for each part of the program (loops, subroutines, etc.) for which local values are obtained. The global parameters values, for the complete C source, are computed by a weighted averaging of all the local values. Along with the global consumption, we indicate in SoftExplorer the consumption of every loop in the code (Section 4). Such an approach permits to spot "hot points" in the program. In the case of data-dependent algorithms, a statistic analysis may be performed to get those values (see Section 3.4).

3.2. Pipeline stalls

As stated before, the pipeline stall rate *PSR* is needed to compute the values of the parameters α and β. To determine the *PSR*, we must evaluate the number of cycles where the pipeline is stalled (NPS) and divide it by the total number of cycles for the program to be executed (NTC):

$$PSR = \frac{NPS}{NTC}. \tag{3}$$

Pipeline stalls have several causes:

(i) a delayed data memory access: if the data is fetched in external memory (related to ε) or if two data are accessed in the same internal memory bank (related to the data mapping DM);

(ii) a delayed program memory access: in case of a cache miss for instance (related to the cache miss rate γ), or if the cache is bypassed or freezed (related to the memory mode MM);

(iii) a control hazard, due to branches in the code: we choose to neglect this contribution because only data-intensive applications are considered.

As a result, NPS is expressed as the sum of the number of cycles for stalls due to an external data access NPS_τ, for stalls due to an internal data bank conflict NPS_{BC}, and for stalls due to cache misses NPS_γ:

$$NPS = NPS_\gamma + NPS_\tau + NPS_{BC}. \tag{4}$$

Whenever a cache miss occurs, the cache controller, via the EMIF, fetches a full instruction frame (containing 8 instructions) from the external memory. The number of cycles needed depends on the memory access time T_{access}. As a result, where $NFRAME$ is the number of frames causing a cache miss,

$$NPS_\gamma = NFRAME \times T_{\text{access}}. \tag{5}$$

Similarly, the pipeline is stalled during T_{access} for each data access in the external memory. That gives, with $NEXT$ being the number of data accesses in external memory,

$$NPS_\tau = NEXT \times T_{\text{access}}. \tag{6}$$

A conflict in an internal data bank is resolved in only one clock cycle. So, NPS_{BC} is merely the number of bank conflicts $NCONFLICT$:

$$NPS_{BC} = NCONFLICT. \tag{7}$$

So, to calculate the *PSR*, we need the number of external data accesses $NEXT$, the number of internal data bank conflicts $NCONFLICT$, and the number of instruction frames that involve cache misses $NFRAME$ ((3), (4), (5), (6), and (7)). Those three numbers are directly extracted from the

Table 3: Relative error at the assembly level.

	C64	C62	C67	C55	ARM7
Maximal error	4.3%	4%	6%	2.5%	8%
Average error	2.6%	2.5%	2.4%	1.4%	—

assembly code when the estimation is performed at the assembly level. It is however remarkable that the two numbers $NEXT$ and $NCONFLICT$ can also be determined directly from the C program. Indeed, they are related to the data mapping which is actually fixed by the programmer by means of explicit compilation directives associated to the C sources, and only taken into account by the compiler during the linkage. The data mapping is integrated in the power model through the configuration parameter DM, which stands for the data mapping file.

External data accesses are fully taken into account through $NEXT$ which participates in the calculation of the PSR. This is why the external data access parameter τ is said to be "included in the PSR" in the sets of parameters given in Table 1.

The number of instruction frames that involve cache misses $NFRAME$, as well as the cache miss rate γ, can be determined statically if the memory mode MM is mapped, bypass, and freeze, or dynamically in cache mode. The assembly code size (with the total number of instruction frames) is needed for comparison with the cache size; a compilation is necessary. In this case, it is not possible to predict $NFRAME$ at the C-level. However, since the assembly code for digital signal processing applications generally fits in the program memory of the processors, γ is most of the time equal to zero (as well as $NFRAME$). Whenever $NFRAME$ and γ are not known in the early step of the design process, SoftExplorer will provide the designer with *consumption maps* to guide him through the code writing, as shown in Section 4.1 [35].

The number of DMA accesses can be determined from the assembly code or from the C program. Indeed, accesses to the DMA are explicitly programmed. The programmer knows exactly the number of DMA accesses; it is therefore easy to calculate the DMA access rate ε without compilation. It is computed by dividing the number of DMA accesses by the total number of data accesses in the program.

3.3. Estimation versus measures

The accuracy of our power and energy estimation at the assembly level has already been investigated [30]. For each of the five processors in our library of models, the power estimation was performed on a set of various algorithms—FIR filter, LMS filter, discrete wavelet transform (DWT) with different image sizes, fast Fourier transform (FFT) 1024 points, enhanced full-rate (EFR) Vocoder for GSM, and MPEG-1 decoder. The power consumption was also measured for all these algorithms and processors. The relative errors between measures and estimations are reported Table 3.

Our aim in this section is to demonstrate the precision of power estimation at the C-level. We first perform some comparisons with the values for α, β, and PSR that were extracted from the assembly code to obtain the former results at the assembly level, and the values that we can predict from the C program, according to our prediction models. In Table 4, the values of the power model parameters extracted from the assembly code, and from the C code assuming the DATA prediction model, are presented. We did not include the predictions for the other prediction models since they provide higher and lower bounds that are naturally farther from the extracted value. For these applications, $\gamma = 0$ since the whole code is contained in the internal program memory, and $\varepsilon = 0$ since the DMA is not used. The PSR measured value (PSR^m), obtained with the help of the TI development tool, is used for estimation at the assembly level (but the calculated value could be used as well). The average error between the predicted (PSR) and the measured (PSR^m) pipeline stall rates is 3.2%. It never exceeds 5.5% which indicates the PSR prediction accuracy.

The power consumption of the algorithm is then estimated from the parameters that we have predicted from the C program. The relative error between the estimation and the measured consumption for the TI C62 at $F = 200\,\text{MHz}$ is given in Table 5. Results are given for the four prediction models at the C-level. We recall in the ASM column the precision obtained at the assembly level.

Of course, the SEQ model gives the worst results since it does not take into account the architecture possibilities (parallelism, several memory banks, etc.). In fact, this model was developed to explore the estimation possibilities without any knowledge about the architecture of the targeted processor. It seems that such an approach cannot provide enough accuracy to be satisfying.

It is remarkable that, for the LMS in the bypass mode, every model overestimates the power consumption with close results. This exception can be explained by the fact that, in this marginal memory mode, every instruction is loaded from the external memory and thus pipeline stalls are dominant. As the SEQ model assumes sequential operations, it is the most accurate in this mode.

For all the other algorithms, the MAX and the MIN models always respectively overestimate and underestimate the application power consumption. Hence, the proposed models need a restricted knowledge of the processor's architecture; but they guaranty to bound the power consumption of a C algorithm with reasonable errors.

The DATA model is the most accurate since it provides a maximum error of 8% against measurements. After compilation, the estimation can be performed at the assembly level where the maximum error is decreased to 3.5%.

Eventually, the estimation possibilities at the C-level are summarized. According to the results obtained with the SEQ model, it seems unrealistic to determine precisely the power consumption without any knowledge about the targeted processor. A coarse grain prediction model, including only the architecture possibilities in terms of parallelism, number of processing units, and so forth, provides the maximum and minimum bounds of the algorithm's power consumption with an average error of 7.3% and 15.2%, respectively. The fine grain prediction model, with both elementary

Table 4: C-level parameters prediction versus ASM-level parameters extraction.

Application	Configuration		Assembly level			C-level		
	MM	DM	α	β	PSR^m	α	β	PSR
FIR	MM_M	INT	0.492	0.454	0	0.5	0.5	0
FFT	MM_M	INT	0.099	0.08	0.64	0.119	0.113	0.604
LMS-1	MM_B	INT	—	0.029	0.93	—	0.0312	0.95
LMS-2	MM_C	INT	0.625	0.483	0.25	0.76	0.475	0.24
DWT-1 ($64 * 64$)	MM_M	INT	0.362	0.287	0.027	0.365	0.324	0.0269
DWT-2 ($64 * 64$)	MM_M	EXT	0.0915	0.0723	0.755	0.105	0.0932	0.713
DWT-3 ($512 * 512$)	MM_M	EXT	0.088	0.0695	0.765	0.1	0.089	0.726
EFR	MM_M	INT	0.594	0.472	0.225	0.669	0.479	0.219
MPEG	MM_M	INT	0.706	0.715	0.108	0.682	0.568	0.09

Table 5: C-level power estimation versus measurements.

Application	Measurements	Estimation vs. measure. (%)				
	$P(W)$	ASM	SEQ	*MAX*	MIN	DATA
FIR	4.5	2.3	−38	5.5	−24.3	5.5
FFT	2.65	2.5	−10	28.5	−1	2.87
LMS-1	4.97	3.5	1.4	2.8	2	2.8
LMS-2	5.66	−1.8	−50	6.4	−15.2	6.4
DWT-1	3.75	1.9	−27	4.7	−13.2	4.7
DWT-2	2.55	−0.2	−10	3.4	−4.2	3.4
DWT-3	2.55	−1	−10.4	2.4	−4.7	2.4
EFR	5.07	−2.8	−50	11.1	−24	1.5
MPEG	5.83	0.7	−54	10	−33	−8
Average errors		1.8	27.8	8.3	13.5	4.2

information on the architecture and the data placement, offers a very accurate estimation with a maximum error of 8% against measurements.

3.4. Execution time prediction

A great part of the job for determining the execution time was already done for the *PSR*. In the previous section, we have indeed determined the number of pipeline stalls together with their duration. The only remaining thing to do is to add the number of cycles for executing the program to the number of cycles where the pipeline is stalled, and to divide by the processor's frequency. In fact, the algorithm is parsed loop by loop, and the data mapping is analyzed, to determine the number of memory conflicts which leads to the *PSR*.

We have estimated the execution time for the previously presented applications and compared it with the value given by the TI's development tool: CodeComposer. This value is exact, for CodeComposer, after compilation, traces the assembly code on the evaluation board. We have then computed the energy from the estimated power consumption and execution time, and compared it with the energy computed from the measured power consumption and execution time. Errors less than 1% are observed. For example, the error for the MPEG-1 decoder presented in the following section is 0.6%.

In the case of dynamic loops, the number of iterations is not known in advance. SoftExplorer takes into account the algorithm's dynamic behavior thanks to pragma directives added to the program: the user indicates a probability for the control structures (if, then, else, etc.) and the dynamic loops. A 50% probability is assumed whenever a pragma is missing. A dynamic profiling may be necessary; specific analysis tools are usable for this purpose. The user could also give the maximum limit for the number of iterations to get a maximum for the execution time and energy. At last, SoftExplorer considers the delays for division and function calls, to further increase the precision in estimating the execution time.

4. SOFTEXPLORER

4.1. Prediction types

All the previous estimations were performed with the help of SoftExplorer (v 5.0). SoftExplorer includes 17 000 lines of code and is written in C. When SoftExplorer is started, a configuration menu appears where it is possible to choose whether estimations will be performed at the C-level or at the assembly level. It is then necessary to choose the targeted processor's power model among those available. As stated before, there are currently 5 power models included (C67, C64, C62, C55, ARM7, though only the C62's power model is provided in the demo version). The input file (C code in our case, but could be also ASM code or PP-preprocessed file) is also indicated here. The data mapping file ought to be written in the same directory as the input file. Depending on the chosen power model, a configuration page for the C-level estimation appears. In the case of the C6x and the C55, the prediction model (SEQ, MIN, DATA, *MAX*) must be indicated, together with the processor's frequency. The memory mode (mapped, cache, freeze, bypass) is required for the C67, C62, and the C55. The prediction type has to be chosen as well. There are three prediction types.

Coarse prediction. There is no need for any further information from the user. The C code is parsed and the power consumption is computed with different values for γ and *PSR* from 0 to 100%. The result is a curve, or an area, displayed on the "C curves results" or the "C area results" page.

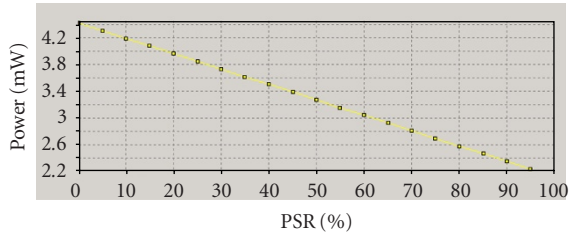

Figure 3: C curve: the power is a function of *PSR* (here) or γ.

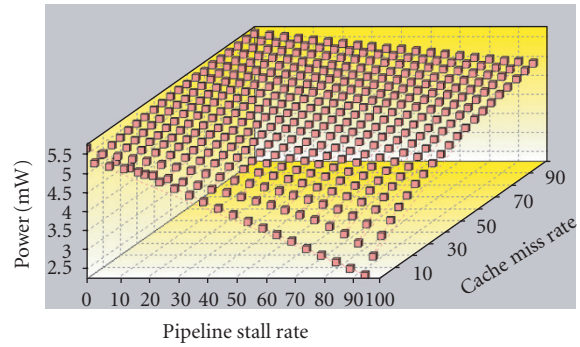

Figure 4: C area: the power is a function of *PSR* and γ.

Indeed, if the memory mode is *bypass* or *mapped*, then $\gamma = 0$, only the *PSR* varies, and a curve is computed. If the memory mode is *cache* or *freeze*, both the *PSR* and γ vary, and an area is drawn. These curves and areas are what we called *consumption maps*. The interest of the coarse prediction is to allow an estimation of an algorithm's power consumption very early in the design process. Indeed, even if the data mapping is not settled (the data mapping is not considered in this mode), the programmer can have an idea of the power consumption, choose the processor, compare different algorithms, and be guided in the optimizations to be conducted on the code.

Fine prediction. In this mode, the data mapping file is used to determine the data access conflicts (internal and external) for each loop in the code. If $\gamma = 0$, which is the case for a large number of DSP applications, these conflicts permit to determine, for every loop, the precise number of pipeline stalls (i.e., the *PSR*) and the execution time. This local knowledge of every parameter in the power model of the target permits to compute, for each loop, the power consumption, the execution time, and the energy consumption. An accurate optimization of each portion of the code can be conducted that way.

Exact prediction. The execution time, the cache miss rate, and the pipeline stall rate are no more predicted by SoftExplorer, but are instead provided by the user. They can be obtained with the help of the targeted processor's development tools (e.g., TI's CodeComposer for the C6x). C-level profilers can also be used for an estimation of the execution time.

We show, on an example, the results that can be obtained with the three prediction types above. The application is the MPEG-1 decoder from *MediaBench*. Estimation is performed first in the coarse mode for the C62. Figure 3 shows the evolution of the power consumption P with the *PSR* (memory mode is mapped). The maximal value for P is 4400 mW when $PSR = 0\%$, and its minimal value is 2200 mW for $PSR = 100\%$. Figure 4 shows the evolution of the power consumption with both the *PSR* and γ (memory mode is cache). This time, the max/min values for P are 5592/2353 mW. It can be observed that for the same *PSR*, the power consumption is always lower in the mapped mode. Table 6 indicates the max/min power consumption for the four memory modes.

A fine prediction takes into account the data mapping. In the mapped or bypass mode, the global power and energy consumptions are presented on the "results" page, with the execution time and the global values of parameters α and β.

Table 6: Power estimation in coarse prediction.

Pmax/Pmin (mW)	Mapped	Cache	Freeze	Bypass
	4400/2200	5592/2353	5690/2442	5882/5073

The power repartition in every functional part of the processor is also given (Figure 5). The DMA unit consumes no power for it is not used in this application. It is remarkable that a great part (47.4%) of the power consumption is due to the clock. The "loops" page display the power consumption, execution time, and α and β values, for each loop (Figure 6). These results are also presented in the form of a chronogram on the "graphics" page (Figure 7). The power consumption per functional part is also indicated on this graph. In the cache or freeze mode, the variations of the power consumption with γ are given on the "C curve" page.

An exact prediction is possible when the exact values for γ, *PSR*, and T_{exe} are known. In our example, the TI's development tool gives, after compilation, $\gamma = 0$, $PSR = 0.2$, and $T_{\text{exe}} = 40$ microseconds. The power and energy estimations are displayed on the "results" page. The "loops" and "graphics" pages display local values for α, β, and the power consumption, assuming that pipeline stalls and cache misses are equally scattered in the algorithm.

For the ARM7, only the memory mode, operating frequency, and execution time are needed. There is no more need for distinct prediction models, nor for the three prediction types. Indeed, we measured that the program itself has only a slight influence on the consumption (see Section 2.5). To model the code, as well as the compiler, is therefore pointless. The estimation is performed directly from the parameters provided by the user.

4.2. Estimation time and complexity reduction

The time for SoftExplorer to parse a code and to perform an estimation is smaller at the C-level than at the assembly level. At the C-level, indeed, there is no need for compilation or a dynamic profiling. The C program has much less lines than the assembly code, and the overall process is faster even if it involves a little more computation to evaluate the consumption against the prediction model. At the assembly level, the estimation time varies from 3 seconds for an FIR

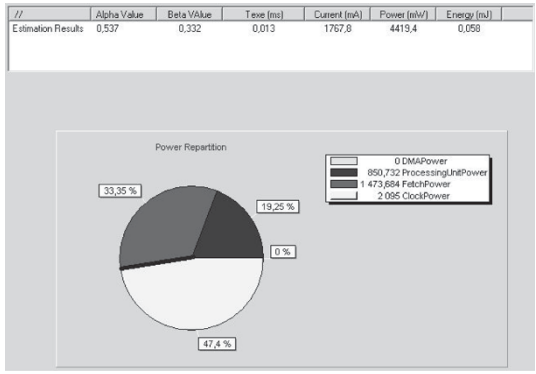

Figure 5: The "results" page also gives the power repartition.

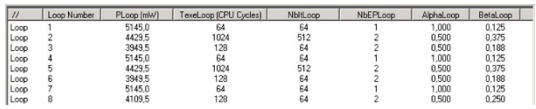

Figure 6: The "loops" page presents the results for each loop.

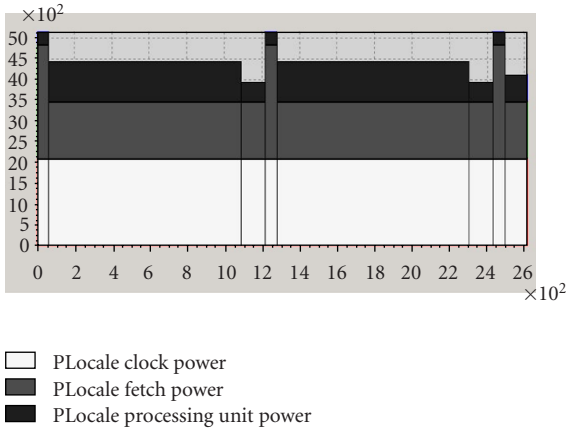

Figure 7: A graphical display of the consumption per loop.

1024 or an LMS 1024, to 10 seconds for a DWT 512 × 512. We have compared this time with the estimation time obtained with the tool SimplePower [**?**], which works cycle by cycle at the architectural level [29]. SoftExplorer appears to be thousands of times faster, since SimplePower treats the FIR 1024 in 4360 seconds, the LMS 1024 in 24 700 seconds, and the DWT 512 × 512 in 14 2000 s. SoftExplorer is even faster at the C-level: the estimation time is less than 1 second for every application that we have tested.

To work on the C code is much more convenient for the user. Firstly, the code is smaller and more readable. Secondly, the number of lines to be studied is drastically reduced once hot spots are located in the algorithm. Indeed, we have spotted, in the applications presented above, the consuming loops, and compared their length with the whole code [35]. Only 13% of the whole code, which represents only 10 lines, was significant for the FFT, 37% (17 lines) for the DWT, 31% (37 lines) for the EFR, 14% (4 lines) for the LMS, and 2% (30 lines) for the MPEG.

Table 7: Data mappings.

Mapping	a	b	c	e	f
1	EXT	EXT	EXT	EXT	EXT
2	B0	B0	B0	B0	B0
3	B0	B1	B0	B1	B0
4	B0	B1	B2	B3	B4

5. APPLICATIONS

5.1. Influence of the data mapping

We demonstrate on a simple example the influence of the data mapping on the power consumption and execution time. The algorithm that we use as a testbench manipulates at the same time 3 images of size 100 × 100 (a, b, c) and 2 vectors of size 10 (e, f) in 3 successively and differently imbricated loop nests. For the images and vectors are manipulated at the same time, their placement in memory has a strong influence on the number of access conflicts, and thus on the number of pipeline stalls. We show here that it is very quick and easy to try different placements, and to reach an optimal data mapping with the help of SoftExplorer. The four different mappings that we tried are presented in Table 7. The results provided by SoftExplorer are presented in Table 8.

In the first mapping, all the data structures are placed in the external memory (EXT). As a result, there are as much external accesses as accesses to the memory, and for every access, the pipeline is stalled (during 16 cycles for the C6x). The relation between parameters α and β and the power consumption is obvious. When the pipeline is stalled, the number of instructions that the processor executes in parallel (α) and the processing rate (β) decrease. As a result, the power consumption of the processor is reduced, but the execution time is lengthened and the energy consumption increases.

In the second mapping, all the structures are placed in the same bank in the internal memory (B0). There will be as much conflicts as before, but this time, the conflicts are internal. The C6x's pipeline is stalled during one cycle in case of an internal conflict. As a result, the time necessary to resolve all the conflicts, expressed in number of cycles, equals the number of conflicts itself.

The interest of the last mapping 4 is to give a minimum bound in term of number of conflicts since every structure in the algorithm is in a different bank. This solution will also give the higher power consumption and the smaller execution time. Indeed, since the pipeline is never stalled, the processor is used at its maximal capacity. The third mapping, which is achievable with a C6x, is as good as mapping 4 since it does not yield any conflict.

5.2. Choosing a processor and its operating frequency

Even if it is easy to obtain the power consumption and the execution time of an algorithm with SoftExplorer, to actually find the right processor and its operating frequency is not straightforward. Indeed, the global energy consumed by the application depends not only on the energy consumed when the algorithm is executed, but also on the energy consumed

Table 8: SoftExplorer results with different mappings.

Mapping	α	β	T_{exe} (ms)	Current (mA)	Power (mW)	Energy (mJ)	Conflicts	Tconflicts
1	0.0015	0.008	9.108	870	2174	19.81	27 601	441 616
2	0.333	0.167	0.414	1378	3444	1.426	27 601	27 601
3	0.5	0.25	0.276	1644	4109	1.134	0	0
4	0.5	0.25	0.276	1644	4109	1.134	0	0

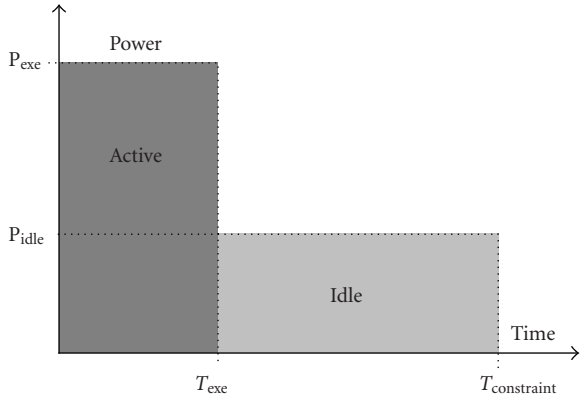

Figure 8: Power consumption and timing constraint.

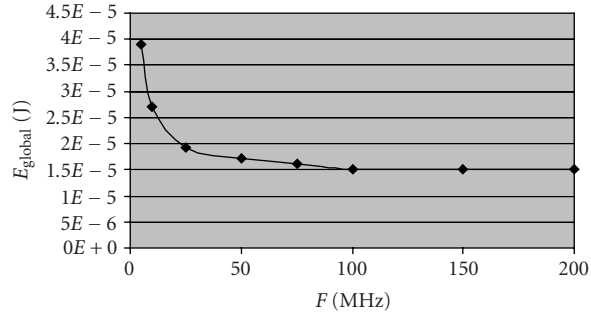

Figure 9: Energy versus frequency for the C55.

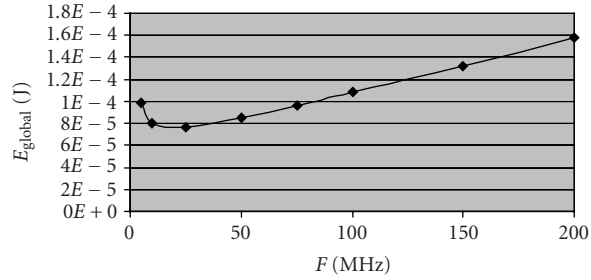

Figure 10: Energy versus Frequency for the C62.

when the processor is idle:

$$E_{\text{global}} = \text{P}_{\text{exe}} \times T_{\text{exe}} + \text{P}_{\text{idle}} \times (T_{\text{constraint}} - T_{\text{exe}}). \quad (8)$$

Figure 8 illustrates this equation. The timing constraint $T_{\text{constraint}}$ is the maximum bound for the execution time. Over this limit, the application's data rate is not respected. Basically, if the frequency is high, the execution time is small and the active power (P_{exe}) increases. The idle time also increases with the frequency. On the other hand, as long as the execution time is lower than the timing constraint, it seems possible to slow down the processor to decrease P_{exe}. So, is it better to operate with a high or a low frequency? In fact, it actually depends on the application.

We pursue a little farther the analysis with our preceding example, the MPEG-1 decoder (Section 4). This algorithm treats 4 macroblocs of a QCIF image in one iteration. A QCIF image (88×72) contains 396 macroblocs. Given a data rate of 10 images/s, the timing constraint is

$$T_{\text{constraint}} = \frac{1}{10} \times \frac{4}{396} = 1.01\,\text{ms}. \quad (9)$$

Then we use SoftExplorer to compute, at different frequencies, the execution time, power, and energy consumed by one iteration of the algorithm. Finally, we calculate with (8) the global energy consumed by the application at these frequencies. The results are presented in Figures 9, 10, and 11, respectively for the C55, C62, and the C67.

The two last curves (Figures 10 and 11) present a minimum that gives the optimal operating frequency for this application: about 20 MHz for the C62 and 40 MHz for the C67. This minimum is 0.076 mJ for the C62 and 0.021 mJ for the C67, hence, the best processor/frequency couple among those two would be the C67 at 40 MHz. The shape of these two global energy curves is predictable when (8) is rewritten as

$$E_{\text{global}} = (KF + C) \times \frac{N}{F} + K'F \times \left(T_{\text{constraint}} - \frac{N}{F}\right). \quad (10)$$

In this expression, P_{exe} is replaced with a dynamic term (KF), and a static one (C), while T_{exe} is trivially replaced by N/F, where N is the number of cycles for one iteration. P_{idle} is given by the constructor as a product of F. When (10) is developed, an F term and a $1/F$ term appear, and that explains the curves' shape.

Whenever P_{idle} does not vary with the frequency, it may be replaced with a constant term in (10), which, when developed, includes only a $1/F$ term. The resulting curve does not present a minimum anymore; this is the case for the C55. For this processor, the frequency that gives the lower global energy consumption is the higher possible (200 MHz). However, since the energy consumption is almost the same at 100 MHz, this last frequency will be preferred for it implies a

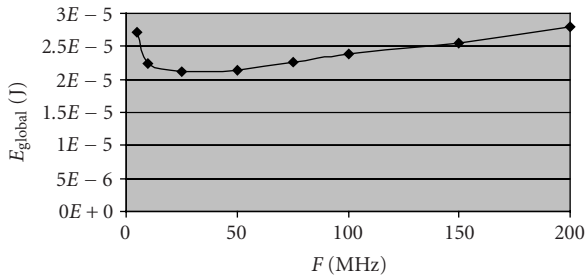

Figure 11: Energy versus frequency for the C67.

lower power consumption: cooling devices will be lighter in this case. In fact, the global energy consumption for the C55 at 100 MHz is 0.015 mJ. As a result, the C55/100 MHz couple is definitely better than the two preceding.

We did not represent the wake-up time on Figure 8. In fact, we measured that this wake-up time is very small before the execution time of the algorithm, and that the energy consumption involved is negligible. Moreover, to avoid waking-up at each iteration, it is preferable to process a whole image with no interruption, and then to idle the processor until the next image. This decreases again the energy contribution of the waking-up. Of course, this can only be done if the application can bare a latency of one image. In a situation where the wake-up time T_{wu} could not be neglected, it is still possible to evaluate the global energy. Assuming that T_{wu} is counted in processor's cycles and that the wake-up power P_{wu} is proportional to the frequency F, one can write (with A and B constants)

$$T_{\text{wu}} = A \times \frac{1}{F}, \qquad \text{P}_{\text{wu}} = B \times F, \tag{11}$$

and the wake-up energy E_{wu} is a constant too:

$$E_{\text{wu}} = \text{P}_{\text{wu}} \times T_{\text{wu}} = A \times B. \tag{12}$$

As a result, the curves that give the global energy in function of the frequency are shifted from the value of E_{wu}. The method to find the best processor and frequency remains the same.

6. CONCLUSION

We have introduced a new method for estimating the power and energy consumption of a DSP application directly from the C program. This method is build upon the functional-level power analysis (FLPA), which we designed initially to estimate the consumption from the assembly code. The advantages of FLPA against instruction-level methods are a shorter delay to obtain a model of a processor, smaller time to achieve the estimation, and the ability to deal easily with complex architectures. Indeed, we have demonstrated our methodology on a wide range of architectures from the very simple general-purpose RISC processor (ARM7) to the more and more complex DSPs: the low-power (C55), the fixed-point VLIW (C62), the fixed-point VLIW with L1 and L2 caches (C64), and the floating-point VLIW (C67). Moreover, very important phenomena like pipeline stalls, cache misses, and memory accesses are taken into account.

To be able to perform an estimation at the assembly level, it was necessary to define two models: a model for the processor and a model for the algorithm. The model for the processor represents the way the processor's consumption varies with its activity. The model for the algorithm links the algorithm with the activity it induces in the processor. Indeed, we showed that an algorithm has some intrinsic features like the parallelism rate or the processing rate.

At the C-level, a third model is necessary: a model for the compiler. Indeed, given an algorithm, the processor's activity actually depends on the compiler behavior. A programmer can set different options in the compiler to give the assembly code different features. Basically, the code can be optimized for performance (as fast as possible), or for size (as small as possible). Our model for the compiler is what we called the prediction model. We defined four prediction models. The DATA (TIME_optimal) and MIN (SIZE_optimal) models represent the former compiler behaviors: respectively optimization for performance and optimization for size. The *MAX* (FULL_parallel) model gives a maximum bound for the power consumption and represents a situation where all the processing resources of the processor would be used restlessly. The SEQ (SEQuential) model stands for a situation where operations are executed one after the other in the processor. That gives an absolute minimum bound for the power consumption. In the case of the two first prediction models, the data mapping was taken into account. Indeed, we have demonstrated that the number of external accesses and the number of memory conflicts are directly related to the processor's processing and parallelism rates, and to the pipeline stall rate (*PSR*), which have a great impact on the final power consumption and execution time.

We have developed a tool, SoftExplorer, which integrates our five power models, and can perform estimation both at the assembly and at the C-level. For DSP applications, and with elementary informations on both architecture and data placement, our C-level power estimation method provides accurate results together with the maximum and minimum bounds of the algorithm's power consumption. The precision of SoftExplorer was evaluated by comparing estimations with measures for several representative DSP applications. The precision varies slightly from a processor to the other. For the C62, the maximal/average errors are 4/2.5% at the assembly level, and 8/4.2% at the C-level. This definitely demonstrates the possibility of performing an accurate power estimation of a C-algorithm.

We exhibit the influence of the data mapping and show how to use SoftExplorer to optimize the consumption and to reduce the complexity by focusing only on the most consuming loops in the code. When the cache miss rate and/or pipeline stall rate are not defined, a consumption map allows to verify if the application constraints are respected. We present a methodology to determine the best processor/frequency couple, for a given application. SoftExplorer is used to estimate the algorithm's power consumption and

execution time on several targets and at different frequencies. We show that the global energy consumed by the application depends on the frequency. With the timing constraint, the processor's idle period of time is determined and the global energy computed. The frequency which yields the lowest consumption is adopted. The winning processor/frequency couple has the lowest energy consumption.

Future works include the development of new power models for the ARM9, the PowerPC, and the OMAP. A generic memory model will be added to include the external memory consumption in our power estimation. Power and energy estimation will be investigated at the system level.

REFERENCES

[1] T. Hauser, "Convergence of two different worlds," *Electronic Design Europe*, February 2003.

[2] H. Baaker, "Powering up the mobile phone," *Speech Technology Journal*, November/December 2001.

[3] S. Cordova, "Power management solutions for multimedia terminals," in *Proc. Batteries 2001 Conference*, Paris, France, April 2001.

[4] M. Valluri and L. John, "Is Compiling for performance == compiling for power?" in *Proc. the 5th Annual Workshop on Interaction between Compilers and Computer Architectures INTERACT-5*, Monterrey, Mexico, January 2001.

[5] A. Fraboulet and A. Mignotte, "Source code loop transformations for memory hierarchy optimizations," in *Proc. the Workshop on Memory Access Decoupled Architecture MEDEA, International Conference on Parallel Architectures and Compilation Techniques (PACT '01)*, pp. 8–12, Barcelona, Spain, September 2002.

[6] S. Singhai and K. S. McKinley, "A parameterized loop fusion algorithm for improving parallelism and cache locality," *The Computer Journal*, vol. 40, no. 6, pp. 340–355, 1997.

[7] J. Ramanujam, J. Hong, M. Kandemir, and A. Narayan, "Reducing memory requirements of nested loops for embedded systems," in *Proc. 38th IEEE Conference on Design Automation (DAC '01)*, pp. 359–364, Las Vegas, Nev, USA, June 2001.

[8] T. V. Achteren, G. Deconinck, F. Catthoor, and R. Lauwereins, "Data reuse exploration techniques for loop-dominated applications," in *Proc. IEEE Conference and Exhibition on Design, Automation and Test in Europe (DATE '02)*, pp. 428–435, Paris, France, March 2002.

[9] L. Benini and G. De Micheli, "System-level power optimization: techniques and tools," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 2, 2000.

[10] C. H. Gebotys, "Utilizing memory bandwidth in DSP embedded processors," in *Proc. 38th IEEE Conference on Design Automation (DAC '01)*, pp. 347–352, Las Vegas, Nev, USA, June 2001.

[11] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, and H. De Man, "Cache conscious data layout organization for embedded multimedia applications," in *Proc. IEEE Conference and Exhibition on Design, Automation and Test in Europe (DATE '01)*, pp. 686–691, Munich , Germany, March 2001.

[12] P. Panda, F. Catthoor, N. D. Dutt, et al., "Data and memory optimization techniques for embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 2, pp. 149–206, 2001.

[13] F. Catthoor, K. Danckaert, C. Kulkarni, and T. Omns, *Data Transfer and Storage (DTS) Architecture Issues and Exploration in Multimedia Processors*, Marcel Dekker, NewYork, NY, USA, 2000.

[14] W.-T. Shiue, S. Udayanarayanan, and C. Chakrabarti, "Data memory design and exploration for low-power embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 4, pp. 553–568, 2001.

[15] *Mentor Graphics Corporation: The EDA Technology Leader*, [Online]. Available: http://www.mentor.com/.

[16] *Synopsys Corporation: EDA Solutions and Services*, [Online]. Available: http://www.synopsys.com/.

[17] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proc. 27th International Symposium on Computer Architecture (ISCA '00)*, pp. 83–94, Vancouver, BC, Canada, June 2000.

[18] M. J. Irwin, M. Kandemir, N. Vijaykrishnan, and W. Ye, "The design and use of simple power: a cycle accurate energy estimation tool," in *Proc. 37th IEEE Conference on Design Automation (DAC '00)*, pp. 340–345, Los Angeles, Calif, USA, June 2000.

[19] D. Brooks, P. Bose, and M. Martonosi, "Power-performance simulation: design and validation strategies," in *ACM SIGMETRICS Performance Evaluation Review*, 2004.

[20] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM Journal of Research and Development*, vol. 47, no. 5/6, 2003.

[21] V. Zyuban, D. Brooks, V. Srinivasan, et al., " Integrated analysis of power and performance for pipelined microprocessors," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 1004–1016, 2004.

[22] J. S. Hu, N. Vijaykrishnan, S. Kim, M. Kandemir, and M. J. Irwin, "Scheduling reusable instructions for power reduction," in *Proc. IEEE Conference and Exhibition on Design, Automation and Test in Europe (DATE '04)*, vol. 1, pp. 148–153, Paris, France, February 2004.

[23] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 437–445, 1994.

[24] B. Klass, D. Thomas, H. Schmit, and D. Nagle, "Modeling inter-instruction energy effects in a digital signal processor," in *Proc. the Power Driven Microarchitecture Workshop in International Symposium on Computer Architecture (ISCA '98)*, Barcelona, Spain, 1998.

[25] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel, "An accurate and fine grain instruction-level energy model supporting software optimizations," in *Proc. IEEE International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS '01)*, pp. 3.2.1–3.2.10, Yverdon-Les-Bains, Switzerland, September 2001.

[26] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, "Function-level power estimation methodology for microprocessors," in *Proc. 37th IEEE Conference on Design Automation (DAC '00)*, pp. 810–813, Los Angeles, Calif, USA, June 2000.

[27] L. Benini, D. Bruni, M. Chinosi, C. Silvano, V. Zaccaria, and R. Zafalon, "A Power modeling and estimation framework for VLIW-based embedded systems," in *Proc. IEEE International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS '01)*, pp. 2.3.1–2.3.10, Yverdon-les-Bains, Switzerland, September 2001.

[28] C. H. Gebotys and R. J. Gebotys, "An empirical comparison of algorithmic, instruction, and architectural power prediction models for high performance embedded DSP processors," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED '98)*, pp. 121–123, Monterey, Calif, USA, 1998.

[29] J. Laurent, N. Julien, E. Senn, and E. Martin, "Functional level power analysis: an efficient approach for modeling the power

consumption of complex processors," in *Proc. IEEE Conference and Exhibition on Design, Automation and Test in Europe (DATE '04)*, vol. 1, pp. 666–667, Paris, France, February 2004.
[30] N. Julien, J. Laurent, E. Senn, and E. Martin, "Power consumption modeling and characterization of the TI C6201," *IEEE Micro*, vol. 23, no. 5, pp. 40–49, 2003, Special Issue on Power- and Complexity-Aware Design.
[31] J. Laurent, E. Senn, N. Julien, and E. Martin, "High level energy estimation for DSP systems," in *Proc. IEEE International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS '01)*, pp. 311–316, Yverdon-les-Bains, Switzerland, September 2001.
[32] TMS320C6x User's Guide, Texas Instruments Inc., 1999.
[33] A. Sinha and A. P. Chandrakasan, "JouleTrack - A web based tool for software energy profiling," in *Proc. 38th IEEE Conference on Design Automation (DAC '01)*, pp. 220–225, Las Vegas, Nev, USA, June 2001.
[34] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, "Energy estimation and optimization of embedded VLIW processors based on instruction scheduling," in *Proc. 39th IEEE Design Automation Conference (DAC '02)*, pp. 886–891, New Orleans, La, USA, June 2002.
[35] E. Senn, N. Julien, J. Laurent, and E. Martin, "Power estimation of a C algorithm on a VLIW processor," in *Proceedings of Workshop on Complexity-Effective Design (WCED '02) (in conjunction with the 29th Annual International Symposium on Computer Architecture (ISCA '02))*, Anchorage, Alaska, USA, May 2002.

Eric Senn is an Associate Professor at the University of South Brittany, France, and a member of the LESTER Laboratory since 1999. He was a Professor at the French Ministry of Defence in the GIP (Geography Image and Perception) Laboratory for the DGA (Déléguation Générale de l'Armement) from 1995 to 1999. His research interests include low-power design, architecture synthesis, and asynchronous circuits. He received a Ph.D. degree in electronics from the University of Paris XI, France, in 1998.

Johann Laurent is an Associate Professor at the University of South Brittany and works at the LESTER Laboratory. His research interests include software consumption estimation and power characterization for complex processors. He received a Ph.D. degree in electronics from the South Brittany University, France, in 2002.

Nathalie Julien is a Professor at the University of South Brittany in Lorient, France; she also works at the LESTER Laboratory in high-level design methods applied to low-power constraints for dedicated circuits, FPGAs, and DSPs. Her research interests include power estimation for complex processors and high-level synthesis that integrates power optimization and memory issues. She has a Ph.D. degree in electronics from the University of Limoges, France. She is a Member of the ACM, SIGDA, and SIGARCH.

Eric Martin is a Professor at the University of South Brittany in Lorient, and Director of the LESTER Laboratory. His research interests focus on advanced electronic design automation dedicated to real-time signal processing applications, including system specification, high-level synthesis, intellectual property reuse, low-power design, systems on a chip, and platform prototyping. He has a Ph.D. degree from the University of Paris XI, France. He is a Member of the IEEE, of the IEEE Computer Society, and of the IEEE Circuits and Systems Society.