<table>
<thead>
<tr>
<th>Title</th>
<th>Overview of Software Level Power Consumption Models and Power Saving Techniques for Embedded DSP Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors(s)</td>
<td>Casas-Sanchez, M.; Rizo-Morente, J.; Bleakley, Chris J.</td>
</tr>
<tr>
<td>Publication date</td>
<td>2005-11-25</td>
</tr>
<tr>
<td>Conference details</td>
<td>Conference on Design of Circuits and Integrated Systems (DCIS), Lisbon, Portugal, 23 - 25 November, 2005</td>
</tr>
<tr>
<td>Link to online version</td>
<td><a href="http://www.cs.ucd.ie/staff/cbleakley/home/papers/dcis05_defi_bf_review.pdf">http://www.cs.ucd.ie/staff/cbleakley/home/papers/dcis05_defi_bf_review.pdf</a>; <a href="http://www.dcis.org">http://www.dcis.org</a></td>
</tr>
<tr>
<td>Item record/more information</td>
<td><a href="http://hdl.handle.net/10197/7099">http://hdl.handle.net/10197/7099</a></td>
</tr>
</tbody>
</table>
Overview of Software Level Power Consumption Models and Power Saving Techniques for Embedded DSP Processors

Miguel Casas-Sanchez, Jose Rizo-Morente, C.J. Bleakley
Department of Computer Science, University College Dublin
Belfield Campus, Dublin 4, Ireland
{miguel.casassanchez, jose.rizo-morente, chris.bleakley}@ucd.ie

Abstract—Unlike DSP compilation for high performance, research for low power optimisation has received little attention, although power dissipation is a critical issue for mobile devices. This paper presents an overview of power consumption models and power saving techniques for embedded DSP processors applications and evaluates their application to the Texas Instruments TMS320VC5510 Digital Signal Processor. Software level power consumption models introduced in the literature are presented, along with their advantages and disadvantages. Several power saving techniques are presented, discussing their relevance for the VC5510 processor architecture. The significance of various instruction components with respect to consumption are considered in detail.

I. INTRODUCTION

The classic separation of hardware and software has led to the misconception that only hardware dissipates power, disregarding the power consumption effects of the code running on the machine. However, that would be analogous to stating that a car driver’s behaviour does not affect the engine’s fuel consumption. Different use of processor resources can dramatically change the power and energy consumption of embedded processors. With this idea in mind, over the past ten years modelling and reduction of power consumption from the software point of view, or equivalently from the compiler point of view, has emerged as an important issue in embedded VLSI system design, specially for portable and battery-powered systems such as digital cellular phones. Unfortunately, most of the literature in the field is scattered and deals with only a couple of isolated techniques.

In this study, current methods for modelling the power consumption of processors, from the software point of view, are presented. The processor under investigation is a popular, low-cost, medium-performance solution. Digital Signal Processors (DSP) have a highly irregular and specialised VLIW architecture, that leads to poor compiler performance [1], both in performance and energy terms. Power and energy saving techniques are reviewed, analyzing their impact for the present architecture and some typical signal processing applications.

This paper is organized as follows. First CMOS energy consumption formulae are presented in Section II. Then in Section III the instruction level power consumption models proposed in the literature are presented. Section IV describes the main features of the actual DSP and the measurement process. Section V presents a review of power optimizations, and shows the results of applying these techniques. Finally Section VI presents the conclusions.

II. BACKGROUND

Power consumption is given by $P = I V$, where $I$ is the average current and $V$ the core supply voltage. The energy $E$ consumed by a program is given by

$$E = P \tau = (I V) (N \tau)$$

where $\tau$ is the total duration of code execution, and $N$ is the number of cycles each of duration $\tau$. In the context of this paper $V$ and $\tau$ are fixed, unless otherwise noted, hence given $N$ for a program, there is only need to measure the average current $I$ to calculate the energy. Optimizations can be targeted either to reduce power ($I$) or energy consumption ($N$ and/or $I$).

Two components [2] determine the power consumption in a CMOS circuit: static and dynamic power consumption. Static power consumption, which is very low in this devices, can be obtained as $P_{static} = \sum I_{leakage} V$ and is determined by the manufacturing process. Dynamic power consumption arises from the transient current which is drawn when switching capacitive loads, and capacitive-load power consumption, expressed as

$$P_{dynamic} = C_L V^2 f N_{switching}$$

where $P_{dynamic}$ is the power consumed, $C_L$ the load capacitance, $V$ stands for the supply voltage, $f$ represents the operating frequency and finally $N_{switching}$ is the switching probability of the digital value. $C_L$ is manufacture-dependent. The dynamic power consumption can be improved either by reducing $f$, $V$ or $N_{switching}$.

Historically, compilers have been targeted to achieve either minimum code size or maximum speed. Energy consumption remained a total unknown with no tools to determine or even estimate it. Power models have been developed to tackle this problem.
III. POWER MODELS

Two types of power estimation models have been proposed, namely hardware level and instruction level. Hardware level power models calculate power and energy from detailed electrical descriptions, comprising circuit level, gate level, RT level or system models. Instruction level power models deal only with instructions and functional units from the software point of view and without electrical knowledge of the underlying architecture. In the present context, instruction stands for the combination of factors such as operation, addressing mode, operand formats and resources implied which determine its execution. Several Instruction power estimation models have been proposed. These can be classified into Instruction Level Power Analysis (ILPA) and Functional Level Power Analysis (FLPA).

The ILPA model, introduced by Tiwari et al. [3], [4], [5], states that the energy consumption of a program can be expressed in the form of (3).

\[ E_p = \sum_i (B_i N_i) + \sum_{i,j} (O_{i,j} N_{i,j}) + \sum_k E_k \]  

where \( B_i \) represents the base cost of instruction \( i \), \( N_i \) is the number of occurrences of each pair of instructions \( i, O_{i,j} \) stands for the circuit state overhead for the instruction pair \( i, j \) and \( E_k \) are the energy costs of other effects. In order to populate the model the current consumption of each instruction, and instruction combination, have to be experimentally measured for each processor. This method has a small margin of error, typically 2 to 4 percent for simple processors, but, the number of measurements is directly related to the complexity of the processor architecture (for example, the DSP 56K requires 1176 measurements [6]). Modelling functional unit activity, such as cache misses, idle domains, and pipeline stalls can require some effort. Tiwari’s methodology has been successfully applied to the Motorola DSP56K [6], ARM7 [7], M3DSP [8], Hitachi SH-4 [9], i960 [10] and Motorola 68HC11 [11], among others.

Improvements in the measuring process have been proposed. The NOP model [6] whereby instead of calculating the inter-instruction effect for each pair of instructions (proportional to \( N^2 \)), it is only calculated relative to the NOP instruction (proportional to \( N \)). Sinha and Chandrakasan [9] simplify the model to a first order, relating the current consumption only to the operating frequency and voltage, achieving around \( \pm 8\% \) error for their programs. Russell et al. [10], also used a first order model for the Intel i960 processor with \( \pm 8\% \) error. It is worth noting that this sort of simplification seems to be valid only for general purpose microprocessors, while for DSP’s like the TMS320VC5510 the error could be up to \( \pm 26.5\% \), and for the Motorola 56K [7] around \( \pm 20\% \). A second order model includes a logical partitioning of the instruction set called “instruction classes”, the error drops to \( \pm 4\% \) [9]. Using this nomenclature Tiwari’s model would be third order, a representative current consumption figure for each family of instructions, or fourth order, a consumption figure for each individual operation. Fifth order would be the current consumption for individual operations and address modes.

To avoid the large number of measurements required for ILPA descriptions and the complicated modelling of the functional unit activity, FLPA models have been proposed, relying on different sets of parameters describing the underlying activity. Some of these parameters have to be derived for every analysed program. Gebotys et al. [12] propose to use six variables, namely average switching in data, address buses and instruction register, together with the average number of loads, subtractions and multiplies, achieving an error within \( \pm 2\% \). Julien et al. [13] base the computation on parallelism rate, processing rate, program cache miss rate, external data memory access rate, DMA using rate, and some architectural parameters: clock frequency, memory mode, data mapping and data width. The error bound for this model applied to the Texas Instruments TMS320C6201 is around \( \pm 4\% \). Further similar FLPA models can be found in [14] and [15].

IV. EXPERIMENTAL FRAMEWORK

A. Target Processor

The target embedded DSP Processor used for the present study is the Texas Instruments TMS320VC5510, with variable core voltage and frequency up to 200MHz. This model belongs to a family specially focused on low cost and low power applications. It works with variable length instructions, thus achieving very compact code. It implements several special structural features relevant to the rest of the paper such as:

- Four functional units, namely Instruction, Program, Address and Data units, which form a four stage prefetch pipeline in the Instruction unit and a four stage pipeline, expanded up to eight stages in the case of memory operands. All the instructions are executed in one clock cycle, except the branches, and the CPU is able to execute two instructions simultaneously under some restrictions.
- Four 40 bit accumulator registers, eight 24 bit addressing registers and four 16 bit temporal registers, along with registers for circular addressing, hardware loops, stack and data page pointers, status registers and others.
- Several low power capabilities such as configurable clock frequency and core voltage. Independent idle domains off and on: CPU, DMA controller, Instruction Cache, Peripheral, Clock Generator and External Memory Interface (EMIF).
- Two independent 40 bit D-unit MAC units, one 40 bit D-unit ALU, one 16 bit A-unit ALU, one A-unit and one D-unit swap, and one D-unit barrel shifter. One A-unit address generation unit, featuring one extra ALU for indirect addressing modes.
- Twelve independent buses to memory, divided into two program buses (data and address), six data read buses and four data write buses, along with internal buses for constant passing between units.
• Instruction Buffer Queue (IBQ), that fetches four program bytes a cycle, up to 64, passes six bytes at a time to the instruction decoder, and is flushed each time the Program Counter jumps. One configurable 24Kbyte Instruction Cache, 64 Kbytes of dual RAM (DARAM) able to read and write in the same cycle, and 256 Kbytes of static RAM (SARAM) divided into 32 blocks that can be accessed independently. System ROM of 32 Kbytes and almost 16 Mbytes of external addressing. One DMA controller.

Measurements were taken using a TMS320VC5510DSK Development Software Kit [16], that provides 1.1 – 1.6V core voltage and 24MHz frequency reference connected to a PC running the TI Code Composer Studio (CCS). This tool was used to download and run the test programs. External software routines were used to trigger the measurements using a digital storage scope. The current drawn was measured with a non intrusive, 0.1mA resolution current probe. The probe bandwidth is around 50MHz providing enough resolution for the present purposes. The measurements were taken at 1.6V, 24MHz unless otherwise noted. The measurements are completely repeatable and automated as in Fig.1. The instructions were grouped according to the taxonomy in the Texas Instruments manuals [17], namely Arithmetic, Bit Manipulation, Logical, and finally Extended Registers and Move Operations.

Memory hierarchy optimization is a very important and broad issue, including maximizing the use of registers and cache memories instead of external memories [18], exploiting data locality and formats, memory alignment, allocation to multiple memories or positions, addressing multiple memory banks [19], choosing the right cache configuration, maximizing dual accesses to dual port memories [20], etc. Results for different memory code and data locations for a 128-dot product can be found in Table I. Use of memories near the CPU reduces consumed power and energy simultaneously.

<p>| Table I: Energy consumed by the 128-dot product program using different memories |
|------------------|------------------|------------------|</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Data</th>
<th>Core Energy (µJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>off-chip</td>
<td>off-chip</td>
<td>49.70 (100.00(%)</td>
</tr>
<tr>
<td>off-chip</td>
<td>on-chip SARAM</td>
<td>20.58 (41.41(%)</td>
</tr>
<tr>
<td>on-chip</td>
<td>off-chip</td>
<td>15.44 (31.06(%)</td>
</tr>
<tr>
<td>on-chip</td>
<td>on-chip SARAM</td>
<td>3.24 (6.52(%)</td>
</tr>
</tbody>
</table>

For the selected processor, varying the registers used in a single instruction within the same register bank or between banks, has no significant impact on power consumption. Results for the Logical, Arithmetic and Move operations vary by less than 3.6%. Different immediate values or operands lead to different bus switching activities. However variation in power due to data dependency is only significant for the D-unit when processing 40 bit arithmetic. Results for Logical operations show a 5.69% variation for 16 bit arithmetic and 13.69% for 40 bit arithmetic. Also in this line, register allocation compiler optimization, mapping variables into registers depending on their frequency of use, can lead to significant power and energy reduction. Results for the Pentium 4 [21] achieve 30% mean of energy saving while for the present architecture results for DSPStone [22] show savings of around 20% in mean. Register allocation is usually based on graph coloring[23]. However recent articles [24] propose the use of Partitioned Boolean Quadratic Programming, further improving energy consumption by 20%–40% through cycle count reduction.

Instruction alignment in variable instruction length processors is important. As in the present processor where operations involving 16 bit immediate data consume less power than those involving 8 bit immediates. Experiments for Logical instructions show 8.10% less consumption in the long case, and 5.63% in the long case for MAC operations. This appears to be due to the different instruction lengths - 3 bytes for 8 bit and 4 bytes for 16 bit. The latter being better aligned in memory and in the IBQ. Some 6-byte long instructions, including 23-bit wide immediates, need two memory cycles to be fetched, thus introducing wait cycles that increase the energy consumed. This phenomenon is called instruction misalignment, and is typical of variable instruction length processors, increasing energy consumption.

Memory accesses are usually made through indirect, register based, addressing modes. These modes imply the use of an Address Generation Unit (AGU) with the possibility of
register modification. Addressing modes with base register modification consume around 7% more power for Logical, Bit and Move operations, and 12% for Arithmetic operations. The power consumption of dual or single port RAM read only accesses is the same. Simultaneous DARAM read and write operations consume more power than their SARAM equivalents by 2.98% and 15.6%, respectively, however in most cases this increase is worth it, saving one access clock cycle.

Loop restructuring provides an important group of transformations, such as loop unrolling, loop-invariant code motion, and nested loop reordering. Tiwari et al. [25] propose energy optimization by loop unrolling, obtaining 10% saving for certain loop unrolling factors. Klass et al. [6] apply it to a FIR filter obtaining 20% mean power saving. Application of this technique to the present architecture leads to significant energy savings by means of cycle count reduction. The consumed power rises as the frequency of power demanding multiply-and-accumulate instruction increases due to the loop unrolling. Results can be found in Fig. 2.

Instruction Level Parallel Processing (ILPP) represents an important optimization. There are two forms: Very Large Instruction Word (VLIW) and Single Instruction Multiple Data (SIMD). Tiwari et al. [4] propose the extensive use of VLIW instructions for DSP’s. Lorentz et al. [26] explore the power savings of SIMD operations. Although in both cases the execution of a parallel operation increases the power consumption, energy saving is achieved through cycle count reduction.

Tiwari et al. [27] propose instruction and operands reordering to reduce the interinstruction effects, achieving 4% to 14% power saving. This is readily implemented by using a modified cost function in the compiler instruction selection; power cost instead of cycle count. They also introduced operand swapping in Booth multiplier architectures, in order to reduce the power consumed by switching activity. Results for the present processor show no variation at all.

Ishihara et al. [28] introduced Dynamic Voltage/Frequency Scaling (VFS) and developed the mathematically optimal frequency-voltage pair for a given task deadline, taking into account the power-delay tradeoff [2]. Much of the DVFS literature [29] is focused on systems with dispatcher and task priorities, which are out of the code transformations scope. It is worth noting that lowering frequency reduces power consumption, but the energy consumed by a program remains the same since it takes longer to complete. The current consumption dependence on frequency for the NOP instruction for the measured DSP, can be found in Fig. 3.

Kremer [30] introduced the idea of selected hibernation of components and remote task mapping. The power savings achieved for each of the configurable Idle Domains in the present architecture can be found in Fig. 4.

VI. CONCLUSIONS

In this article, actual methods to model the power consumption of processors from the software point of view were presented, along with their advantages and disadvantages, classifying them into Instruction Level Power Models and Functional Level Power Models. Their performance in terms of prediction accuracy, modelling difficulty and number of measurements was also discussed. The experimental framework was described, including the features impacting for
performance and power/energy consumption in the Texas Instruments TMS320VC5510 Digital Signal Processor, and the physical measurement methodology. Several power or energy saving techniques were on the other hand presented, making a distinction between power consumption reduction and energy saving. Their impact for the actual DSP architecture and some typical signal processing applications is assessed.

ACKNOWLEDGEMENTS

This work is sponsored by Enterprise Ireland (EI) under agreement number PC/2004/311. The authors would like to thank the Electrical and Electronical Engineering Department at University College Dublin, Ireland, for their support.

REFERENCES