V.Prasanna Kumar / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 6, November- December 2012, pp.312-325

Energy-Efficient Design of Battery-Powered Embedded Systems

V.Prasanna Kumar M.Tech,
Asst.Professor, Dept Of E.C.E, L.N.B.C.I.E.T, Satara

Abstract

Energy-efficient design of battery-powered systems demands optimizations in both hardware and software. We present a modular approach for enhancing instruction level simulators with cycle-accurate simulation of energy dissipation in embedded systems. Our methodology has tightly coupled component models thus making our approach more accurate. Performance and energy computed by our simulator are within a 5% tolerance of hardware measurements on the SmartBadge [2]. We show how the simulation methodology can be used for hardware design exploration aimed at enhancing the SmartBadge with real-time MPEG video feature. In addition, we present a profiler that relates energy consumption to the source code. Using the profiler we can quickly and easily redesign the MP3 audio decoder software to run in real time on the SmartBadge with low energy consumption. Performance increase of 92% and energy consumption decrease of 77% over the original executable specification have been achieved.

Index Terms—Low-power design, performance tradeoffs, power consumption model, system-level.

I. INTRODUCTION

ENERGY consumption is a critical factor in system-level design of embedded portable appliances. In addition, low costs with fast time to market are crucial. As a result, typical portable appliances are built of commodity components and have a microprocessor-based architecture. Full system evalua- tion is often done on prototype boards resulting in long design times. Field programmable gate array (FPGA) hardware emu- lators are sometimes used for functional debugging but cannot give accurate estimates of energy consumption or performance. Performance can be evaluated using instruction-set simulators (e.g., [1]), but there is limited or no support for energy consump- tion evaluation.

Ideally, when designing an embedded system built of com- modity components, a designer would like to explore a limited number of architectural alternatives and test functionality, en- ergy consumption, and performance without the need to build a prototype first. In addition, designers need to optimize software both during hardware development and once the prototype is built in order to get the best performance and energy consump- tion from the system. Embedded software optimization requires tools for estimating the impact of program transformations on energy consumption and performance.

This work presents a complete solution for all embedded system design issues discussed above. The distinctive features of our approach are the following: i) complete system-level and component energy consumption estimates as well as battery lifetime estimates; ii) ability to explore multiple architectural alternatives; and iii) easy estimation of the impact of software changes both during and after the architectural exploration. The tool set is integrated within the instruction set simulator provided by ARM Ltd. [1]. It consists of two components: a cycle-accurate system-level energy consumption simulator with battery lifetime estimation and a system profiler that correlates both energy consumption and performance with the code. Our tools have been tested on a real-life industrial application, and have proven to be both accurate (within 5% of hardware measurements) and highly effective in optimizing the energy consumption in embedded systems (energy consump- tion reduced by 77%). In addition, they are very flexible and easy to adopt to different systems. The tools contain general models for all typical embedded system components but the microprocessor. In order to adopt the tools to another processor, the ARM ISS needs to be replaced by the ISS for the processor of interest.

The rest of this manuscript is organized as follows. We discuss related work in Section II. System model and the methodology for cycle-accurate simulation of energy dissi- pation are presented in Section III. Section IV shows that the simulation results of timing and energy dissipation using the methodology presented are within 5% of the hardware mea- surements for the Dhrystone test case. Hardware architecture trade-offs for SmartBadge's real-time MPEG video decode design are explored using cycle-accurate energy simulation in Section V. The profiling support we have developed is presented in Section VI. A full software design example of MP3 audio decoder for the SmartBadge that uses our profiler is shown in Section VII.

II. RELATED WORK

As portable embedded systems have grown in importance in recent years, so has the need for tools that enable energy consumption estimation for such systems. CAE support for embedded system design is still limited. Commercial tools target mainly functional verification and performance estima- tion [3]–[6], but provide no support for energy-related cost metrics.

Processor energy consumption is generally estimated by *nstruction-level power analysis*, first proposed by Tiwari *et al.* [24], [25]. This technique estimates the energy consumed by a program by summing the energy consumed by the execution of each instruction. Instruction-by-instruction energy costs, together with nonideal effects, are precharacterized once for all for each target processor. An approach proposed recently in [12] attempts to evaluate the effects of different cache and bus configurations using linear equations to relate the main cache characteristics to system performance and energy consumption. This approach does not account for highly nonlinear behavior in cache accesses for different cache configurations that are both data and architecture dependent.

A few research prototype tools that estimate the energy con- sumption of processor core, caches, and main memory in SOC design have been proposed [7], [10]. Memory energy consump- tion is estimated using cost-per-access models. Processor ex- ecution traces are used to drive memory models, thereby ne- glecting the nonnegligible impact of a nonideal memory system on program execution. The final system energy is obtained by summing over the contribution of each component. The main limitation of the approaches presented in [7] and [10] is that the interaction between memory system (or I/O peripherals) and processor is not modeled.

A more recent approach presented in [11] combines multiple power estimators into one simulation engine thus enabling de- tailed simulation of some components, while using high-level models for others. This approach is able to account for interac- tion between memory, cache and processor at run time, but at the cost of potentially long run-times. Longer run-times are caused by different abstraction levels of various simulators and by the overhead in communication between different components. The techniques that enable significant simulation speedup are pre- sented, but at the cost of the loss of detail in software design and in the input data trace.

Cycle-accurate register-transfer level energy estimation is pre- sented in [8]. This tool integrates RT level processor simulator with DineroIII cache simulator and memory model. It is shown to be within 15% of HSPICE simulations. Unfortunately, this ap- proach is not practical for component-based designs such as the one presented in this paper, as it requires knowledge of the in- ternal design of system components. In addition, it is slower than our approach as it models at lower abstraction level.

An alternative approach for energy estimation using measure- ments as a basis for estimation is presented in PowerScope tool [9]. PowerScope requires two computers to collect the measure- ment statistics, some changes to the operating system source code, and a digital multimeter. Although this system enables accurate code profiling of an existing system, it would be very difficult to use it for both hardware and software architecture ex- ploration we present in this paper, as in the early design stages neither hardware nor operating systems or software are avail- able for measurements.

Finally, previous approaches do not focus on battery life op- timization, the ultimate goal of energy optimization for portable systems. In fact, when the battery subsystem is not considered in energy estimation significant errors can result [21]. Some an- alytical estimates of the tradeoff between battery capacity and delay in digital CMOS systems are presented in [18]. Battery capacity is strongly dependent on the discharge current as can be seen from any battery data sheet [22]. Hence, it is important to accurately model discharge current as a function of time in an embedded system.

In contrast to previous approaches, in this work memory models and processor instruction-level simulator are tightly integrated together with an accurate battery model into cycle- accurate simulation engine. Estimation results obtained with our simulator are shown to be within 5% of measured energy consumption in hardware. In addition, we accurately model battery discharge current. Since we develop only one simulation engine, there is no overhead in executing simulators at different levels of abstraction, or in the interface between them. Thus, our approach enables fast and accurate architecture exploration for both energy consumption and performance.

In an industrial environment, the degrees of freedom in hardware design for embedded portable appliances are often very limited, but for software a lot more freedom is available. As a result, a primary requirement for system-level design methodology is to effectively support code energy consumption optimization. Several techniques for code optimization have been presented in the past. A methodology that combines automated and manual software optimizations focused on optimizing memory accesses has been presented in [17]. Tiwari *et al.* [24], [25] uses instruction-level energy models to develop compiler-driven energy optimizations such as instruction reordering, reduction of memory operands, operand swapping in Booth multipliers, efficient usage of memory banks, and series of processor specific optimizations. Several other opti- mizations have been suggested, such as energy efficient register labeling during the compile

phase [19], procedure inlining and loop unrolling [7], as well as instruction scheduling [27]. Work presented in [20] applies a set of compiler optimizations concurrently and evaluates the resulting energy consumption via simulation.
All of the techniques discussed above focus on automated instruction-level optimizations driven by the compiler. Unfor- tunately, currently available commercial compilers have lim- ited capabilities. The improvements gained when using stan- dard compiler optimizations are marginal compared to writing energy efficient source code [16]. The largest energy savings were observed at the interprocedural level that compilers have not been able to exploit.

Code optimization requires extensive program execution analysis to identify energy-critical bottlenecks and to provide feedback on the impact of transformations. Profiling is typically used to relate performance to the source code for CPU and L1 cache [1]. Leveraging our estimation engine, we implemented a code profiling tool that gives percentages of time and energy spent in each procedure for every system component, not only CPU and L1 cache. Thanks to energy profiling, the programmer can easily identify the most energy-critical procedures, apply transformations, and estimate their impact not only on pro- cessor energy consumption, but also on memory hierarchy and system busses.

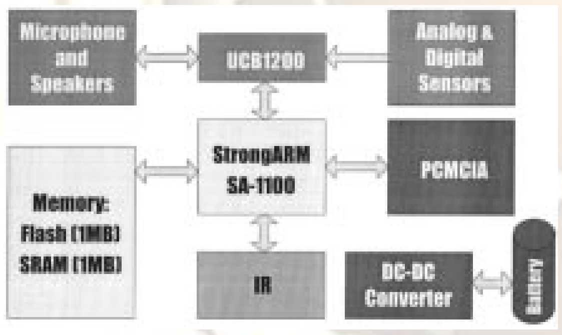


Fig. 1. SmartBadge.

Our approach enables complete system-level and component energy consumption estimates as well as battery lifetime es- timates. In addition, it provides an ability to quickly explore multiple architectural alternatives. Finally, it enables software optimization both during and after architectural exploration using our energy profiling tool. In the following section we present the cycle-accurate energy simulator architecture to- gether with energy consumption models for the components modeled.

III. SYSTEM MODEL

Typical portable embedded systems have processors, storage, and peripherals. We use SmartBadge [2] throughout this paper as a vehicle to illustrate our methodology and to obtain hard- ware measurements. The SmartBadge, shown in Fig. 1, is an em- bedded system consisting of the StrongARM-1100 processor, FLASH, SRAM, sensors, and modem/audio analog front-end on a PCB board powered by the batteries through a DC–DC converter. The initial goal in designing the SmartBadge was to allow a computer or a human user to provide location and envi- ronmental information to a location server through a heteroge- neous network. The SmartBadge could be used as a corporate ID card, attached (or built in) to devices such as PDAs and mobile telephones, or incorporated in computing systems. The design goal for the SmartBadge has since been extended to combine lo- cation awareness and authentication with audio and video sup- port. We will illustrate how our methodology has been used for architecture exploration of the new SmartBadge that needed to support real-time MPEG video decode feature. In addition, we will show how our profiler and code optimizations can be used to improve code for MP3 audio decoder.

The system we use in this work to illustrate our methodology, the SmartBadge, has an ARM processor. As a result, we im- plemented the energy models as extensions to the cycle-accu- rate instruction-level simulator for the ARM processor family, called the ARMmulator [1]. The ARMulator is normally used for functional and performance validation. Fig. 2 shows the sim- ulator architecture. The typical sequence of steps needed to set up system simulation can be summarized as follows: 1) The designer provides a simple functional model for each system component other than the processor; 2) The functional model is annotated with a cycle-accurate performance model; 3) Ap- plication software (written in C) is cross-compiled and loaded in specified locations of the system memory model; and 4) The simulator runs the code and the designer can analyze execution using a cross-debugger or collecting statistics. A designer inter- ested in using our methodology would only need to addition- ally provide cycle-accurate energy models for each component during step 2) of the simulation setup. Thus, the designer can obtain power estimates with little incremental effort.

We developed a methodology for enhancing cycle-accurate simulators with energy models of typical components used in embedded system design. Each component is characterized with equivalent capacitance for each of its power states. Energy spent per cycle is a function of equivalent capacitance, current voltage, and frequency. The equivalent capacitance allows us to easily scale energy consumed for each component as frequency or voltage of operation change. Equivalent capacitances are cal- culated given the information provided in data sheets.
Internal operation of our simulator proceeds as

follows. On each cycle of execution the ARMulator sends out the infor- mation about the state of the processor ("cycle type") and its address and data busses. Two main classes of processor cycle types are *processor active*, where active power is consumed, and *processor idle*, where idle power is consumed. The processor idle state represents an off-chip memory request. The number of cycles that the processor remains idle depends on L2 cache and memory model access times. L2 cache, when present, is always accessed before the main memory and so is active on every memory access request. On L2 cache miss, main memory is accessed. Memory model accounts for energy spent during the memory access. The interconnect energy model calculates energy consumed by the interconnect and pins based on the number of lines switched during the cycle on the data and ad- dress busses. The DC–DC converter energy model sums all the currents consumed each cycle by other system components, ac- counts for its efficiency loss, and gets the total energy consumed from the battery. The battery model accounts for battery effi- ciency losses due to the difference between the rated current and discharge current computed the current cycle.

The total energy consumed by the system per cycle is the sum of energies consumed by the processor and L1 cache (E_{CPU}), interconnect and pins (E_{Line}), memory (E_{Mem}), L2 cache (E_{L2}), the DC–DC converter (E_{DC}) and the efficiency losses in the battery (E_{Bat})

$$E_{\text{Cycle}} = E_{\text{CPU}} + E_{\text{Line}} + E_{\text{Mem}} + E_{\text{L2}} + E_{\text{DC}} + E_{\text{Bat}}. \quad (1)$$

The total energy consumed during the execution of the software on a given hardware architecture is the sum of the energies con- sumed during the each cycle. Models for energy consumption and performance estimation of each system component are de- scribed in the following sections.

A. Processor

The ARM simulator provides a cycle-accurate, instruction- level model for ARM processors and L1 on-chip cache. The model was enhanced with energy consumption estimates based on the information provided by the data sheets. Two power states are considered: active state in which processor is running with

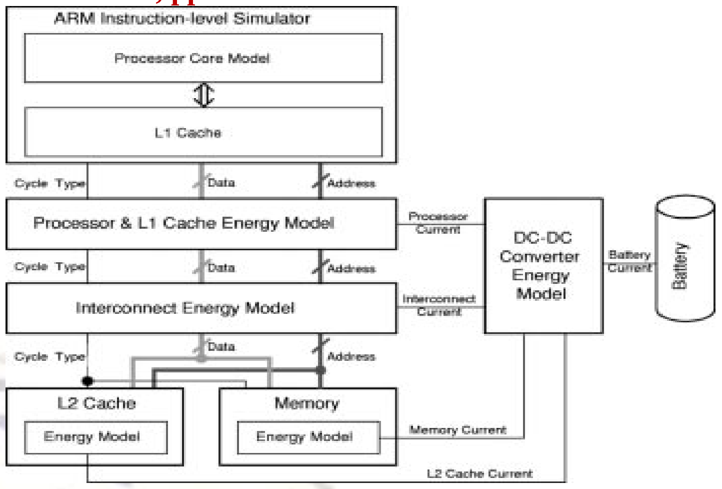


Fig. 2. Simulator architecture.

the on-chip cache, and the state in which the processor is exe- cuting NOPs while waiting to fill the cache.

Note that in the case of StrongARM processor used in this work, the data sheet values for current consumption correspond well to the measured values. Wan [26] extended the StrongARM processor model with base current costs for each instruction. The average power consumption for most of the instructions is 200 mW measured at 170 MHz. Load and store instructions re- quired 260 mW each. Because the difference in energy per in- struction is minimal, it can be expected that the average power consumption value from the data sheets is on the same level of accuracy as the instruction-level model. Thus we can use data sheet values to derive equivalent capacitances for the Stron- gARM. Note that for other processors data sheet values would need to be verified by measurement, as often data sheet values report the maximum power consumption, instead of typical.

When the processor is executing with the on-chip cache, it consumes the active power specified in the data sheet P_m mea- sured at given voltage V_m and frequency of operation . Total equivalent active capacitance within the processor is estimated as

$$C_{\text{CPU},a} = \frac{P_m}{V_m^2 f_m}. \quad (2)$$

The amount of energy consumed by processor and L1-cache at specified processor cycle time T_{cycle} and CPU core voltage V_{cc} is

$$E_{\text{CPU, active}} = P_{\text{CPU},a} T_{\text{cycle}} = C_{\text{CPU},a} V_{cc}^2. \quad (3)$$

When there is an on-chip cache miss, the processor stalls and ex-
ecutes NOP instructions which consume less power.

can be estimated from the power consumed during execution of
NOPs $P_{\text{CPU, NOP}}$ at voltage V_m and frequency f_m

$$C_{\text{CPU, NOP}} = \frac{P_{\text{CPU, NOP}}}{V_m^2 f_m}. \quad (4)$$

The energy consumed within processor core per cycle while ex- ecuting NOPs is

$$E_{\text{CPU, NOP}} = C_{\text{CPU, NOP}} V_{cc}^2. \quad (5)$$

B. Memory and L2 Cache

The processor issues an off-chip memory access when there is an L1 cache miss. The cache-fill request will either be ser- viced by the L2 cache if one is present in the design or directly from the main memory. On L2 cache miss, a request is issued to the processor to fetch data from the main memory. Data sheets specify the memory and L2 cache access times and energy con- sumed during active and idle states of operation.

Memory access time T_{mem} is scaled by the processor cycle time T_{cycle} to obtain the number of cycles the processor has to wait to serve a request N_{wait} (6). Wait cycles are defined for two different types of memory accesses: sequential and nonsequen- tial. Sequential access is at the address immediately following the address of the previous access. In burst type memory the se- quential access is normally a fraction of the first nonsequential access

$$N_{\text{wait}} = \frac{T_{\text{mem}}}{T_{\text{cycle}}}. \quad (6)$$

Two energy consumption states are defined for each type of memory: active and idle. Energy consumed per cycle while

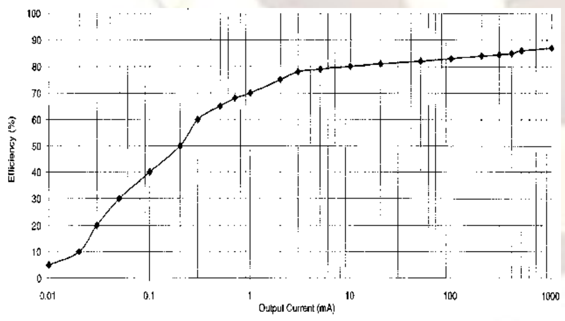


Fig. 3. DC–DC converter efficiency

memory is in active state operating at supply voltage V_{dd} is a function of equivalent active capacitance, voltage of operation and number of total access cycles ($N_{\text{wait}}+1$)

$$E_{\text{Mem, active}} = \frac{C_{\text{mem}} V_{dd}^2}{N_{\text{wait}} + 1}. \quad (7)$$

Active memory capacitance C_{mem} can be estimated from the ac- tive power specified in the data sheet P_{mem} measured at voltage V_m and frequency f_m

$$C_{\text{mem}} = \frac{P_{\text{mem}}}{V_m^2 f_m}. \quad (8)$$

Multibank memory can be represented as multiple one-bank memories

Idle state can be further subdivided into multiple states that describe modes of operation for different types of memories. For example, DRAM might have two idle states: refresh and sleep. ρ_i The designer specifies the percentage of the time memory spends in each idle state. Total idle energy per cycle for memory is

$$E_{\text{Mem, idle}} = T_{\text{cycle}} \sum_{i=0}^{n} P_i \rho_i \quad (9)$$

where P_i is power consumption in idle state . Both RAM and ROM are represented with the same memory model, but with different parameters.

The L2 cache access time and energy consumption are treated the same way as any other memory. L2 cache organization is de- termined from the number of banks, lines per bank, and words per line. Line replacement can follow any of the well-known replace- ment policies. Cache hit rate is strongly dependent on its organi- zation, which in turn affects the total memory access time and the energy consumption. Note that we are simulating details of the L2 cache access and thus know the exact L2 cache miss rate.

C. Interconnect and Pins

The interconnects on the PCB can contribute a large portion of the off-chip capacitance. Capacitance per unit length of the interconnect is a parameter in the energy model that can be ob- tained from the PCB manufacturer. The length of an intercon- nect can be estimated by the designer based on the approximate placement of the selected components on the PCB. Pin capaci- tance values are reported on the data sheets.

For each component the average length of the clock line, data, and address buses between the processor and the component are provided as one of the input simulation parameters. Hence, the designer is free to use any wire-length estimate [14] or mea- surement. The interconnect lengths used in our simulation of SmartBadge come from the prototype board layout.
The total capacitance switched during one cycle is shown in (10). It depends on the capacitance of one interconnect line and the pins attached to it C_{switch} and the number of lines switched during the cycle N_{switch}

$$C_{\text{line}} = N_{\text{switch}} C_{\text{switch}}. \quad (10)$$

The total energy consumed per cycle $E_{\text{Interconnect}}$ is a function of the voltage swing on the lines that switched V_{dd} total capac- itance switched C_{line} and the total time to access the memory $N_{\text{wait}}+1$

$$E_{\text{Line}} = \frac{C_{\text{line}} V_{dd}^2}{N_{\text{wait}}+1}. \qquad (11)$$

D. DC–DC Converter

DC–DC converter losses can account for a significant frac- tion of the total energy consumption. Fig. 3 from the datasheets shows the dependence of efficiency on the DC–DC converter output current. Total current drawn from the DC–DC converter by the system each cycle I_{out} is a sum of the currents drawn by each system component. A component current I_c is defined by

$$I_c = \frac{E_c}{V_c T_{\text{cycle}}} \qquad (12)$$

where E_c is the energy consumed by the component during cycle of length T_{cycle} at operating voltage V_c.
Total current drawn from the battery I_{bat} can be calculated as

$$I_{\text{bat}} = \frac{I_{\text{out}}}{\eta_{\text{DC}}}. \qquad (13)$$

η_{DC} Efficiency can be estimated using linear interpolation from the data points derived from the output current versus efficiency plot in the data sheet. From our experience, a table with 20 points derived from the data sheets gives enough information for accurate linear estimation of values not directly represented in the table.
Total energy DC–DC converter draws out of the battery each cycle is

$$E_{\text{DCbat}} = I_{\text{bat}} V_{\text{bat}} T_{\text{cycle}}. \qquad (14)$$

The energy consumed by the DC–DC converter E_{DC} is differ- ence between the energy provided by the battery E_{DCbat} and the energy consumed from the DC–DC converter by all other components, E_{out}

$$E_{\text{DC}} = E_{\text{DCbat}} \quad E_{\text{out}}. \qquad (15)$$

E. Battery Model

The main battery characteristic is its rated capacity measured in megawatt hours. Since total available battery capacity varies with the discharge rate, manufacturers specify plots in the datasheets with discharge rate versus battery efficiency similar to the one shown below.

The discharge rate (or discharge current ratio) is given by

$$R_I = \frac{I_{\text{ave}}}{I_{\text{rated}}} \qquad (16)$$

where I_{rated}, the rated discharge current, is derived from the battery specification and I_{ave} is the average current drawn by the DC–DC converter. As a battery cannot respond to instantaneous changes in current, a first order time constant is defined to determine the short-term average current drawn from the battery [23]. Given , and processor cycle time T_{cycle}, we can compute N_{bat}, the number of cycles over which average DC–DC current is calculated as

$$N_{\text{bat}} = \frac{\tau}{T_{\text{cycle}}}. \qquad (17)$$

Then, I_{ave} is computed as

$$I_{\text{ave}} = \frac{1}{N_{\text{bat}}} \sum_{\text{cycle}=1}^{N_{\text{bat}}} I_{\text{system}}(\text{cycle}) \qquad (18)$$

where I_{system} is the instantaneous current drawn from the bat- tery. With discharge current ratio, we estimate battery efficiency using battery efficiency plot such as the one shown in Fig. 4. The

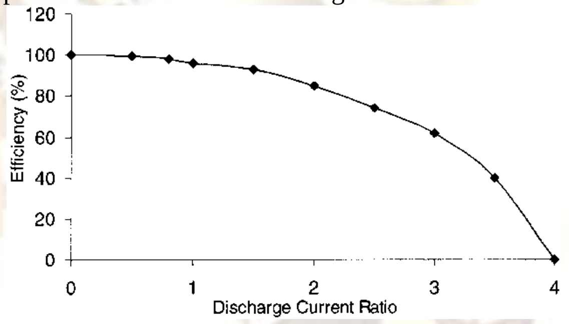


Fig. 4. Battery efficiency.

total energy loss of the battery per cycle E_{Bat} is the product of energy drained from the battery by the system with the effi- ciency loss $(1 \quad \eta_{\text{Bat}})$

$$E_{\text{Bat}} = (1 \quad \eta_{\text{Bat}}) I_{\text{ave}} V_{\text{Bat}} T_{\text{cycle}}. \qquad (19)$$

Given the battery capacity model described above, battery es-timation is performed as follows. First, the designer character- izes the battery with its rated capacity, the time constant, and the table of points describing the discharge plot similar to the one shown in Fig. 4. During each simulation cycle discharge cur- rent ratio is computed from the rated battery current and average DC–DC current calculated from the last N_{bat} cycles. Efficiency is calculated using linear interpolation between the points from the discharge plot. Total energy drawn from the battery during the cycle is obtained from (19). Lower efficiency means that less battery energy remains and thus the battery lifetime is propor- tionally lower. For example, if battery efficiency is 60% and its rated capacity is 100 mAhr at 1 V, then the battery would be drained in 12 min at average DC–DC current of 300 mA. With efficiency of 100% the battery would last 1 h.

IV. VALIDATION OF THE SIMULATION METHODOLOGY

We validated the cycle-accurate power simulator by com- paring the computed energy consumption with measurements on the SmartBadge prototype implementation. The SmartBadge prototype consists of the StrongARM-1100 processor, DC–DC Converter, FLASH, and SRAM on a PCB board. All the com- ponents except the CPU core are powered through the 3.3 V supply line. CPU core runs on 1.5 V supply. DC–DC converter is powered by the 3.5 V supply. DC–DC converter efficiency table contains 22 points derived from the plot shown in Fig. 3. Stripline interconnect model is used with 1.6 pF/cm capacitance calculated based on the PCB board characteristics [13]. Table I shows other system components. Average current consumed by the processor's power supply and the total current drawn from the battery are measured with digital multimeters. Execution time is measured using the processor timer.
Industry standard Dhrystone benchmark is used as a vehicle for methodology verification. Measurements and simulations have been done for ten different operating frequencies of SA-1100 and SA-110 processors. Dhrystone test case is run 10 million times, 445 instructions per loop. Simulations ran

Table 1
DHRYSTONE TEST CASE SYSTEM DESIGN

Component Units	Cycle T. (ns)	Active P (mW)	Idle P (mW)	Pin Cap. (pF)	Line L. (cm)
SA-1100	5-20	400	170	5	N/A
FLASH (1MB)	80	74	0.5	10	2
SRAM (1MB)	90	55	0.01	8	3

On HP Vectra PC with Pentium II MMX 300 MHz processor and 128 MB of memory. Hardware ran 450 times faster than the simulations without the energy models. Simulations with energy models were slightly slower (about 7%). Fig. 5 shows average processor core and battery currents. Average simulation current is obtained by dividing the total energy consumed by the processor core or the battery with their respective supply voltages and the total execution time.

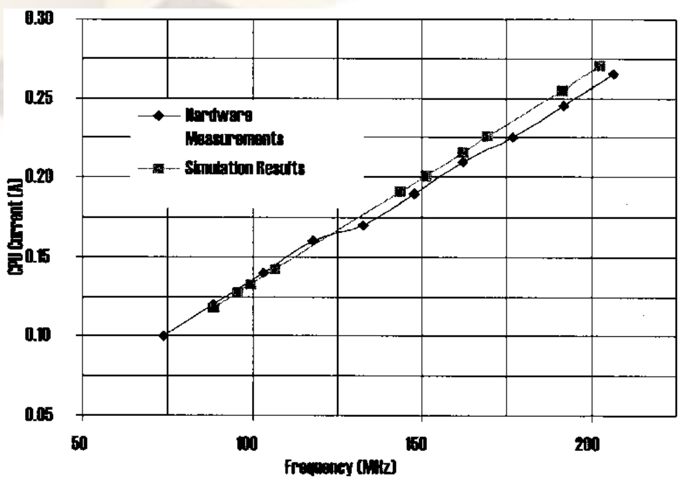
Simulation results are within 5% of the hardware measure- ments for the same frequency of operation. The methodology presented in this paper for cycle-accurate energy consumption simulation is very accurate and thus can be used for architecture design exploration in embedded system designs. An example of such exploration is presented next.

V. EMBEDDED MPEG DECODER SYSTEM DESIGN EXPLORATION

The primary motivation for the development of cycle-accu- rate energy consumption simulation methodology is to provide an easy way for embedded system designers to evaluate mul- tiple hardware and software architectures with respect to per- formance and energy consumption constraints. In this section we will present an application of the simulation methodology to embedded MPEG video decoder system design exploration. The MPEG decoder design consists of the processor, the off-chip memory, the DC–DC converter, output to the LCD display, and the interface to the source of the MPEG stream. The input and output portions of the MPEG decoder design will not be consid- ered at this point. We focus on selection of memory hierarchy that is most energy efficient.

The characteristics of memory components considered are shown in Table II. Two different instruction memories were evaluated—low-power FLASH and power-hungry burst FLASH. We looked at three different data memories—low- power SRAM, faster burst SRAM, and very power-hungry burst SDRAM. Both instruction and data memories are 1 MB in size. We considered using L2 cache in addition to L1 cache. Unified L2 cache is 256 Kb, four-way set associative. The hardware configurations simulated are shown in Table III. The MPEG video decode sequence we used has 12 frames running at 30 frames/second, with two I, three P, and seven B-frames. We found that the results we obtained with a shorter video sequence matched well the results obtained with the longer trace.

Fig. 6 shows the amount of time each system component is active during the MPEG decode and the amount of energy con- sumed. The original configuration is limited by the bandwidth of data memory. L2 cache is very fast but also consumes too much energy. Burst SDRAM design fully solves the memory bandwidth problem with least energy consumption. Instruction



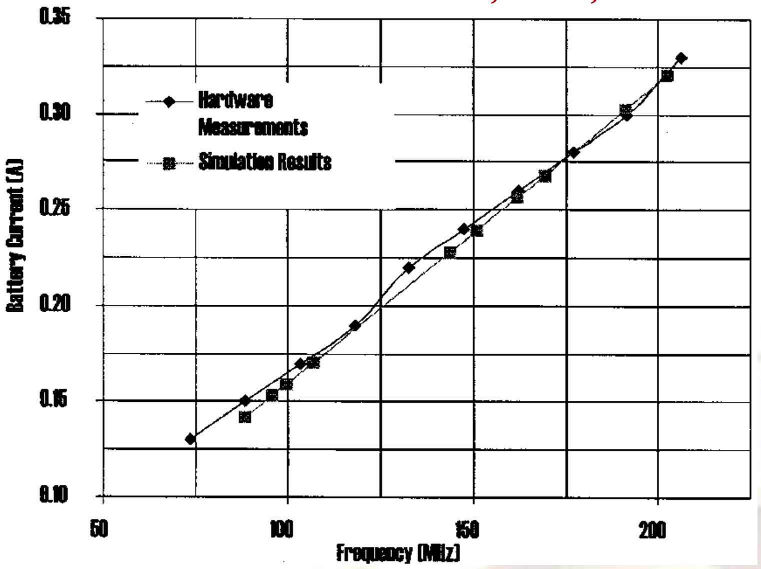


Fig. 5. Average processor core and battery currents.

TABLE II
MEMORY ARCHITECTURES FOR MPEG DESIGN

Name	First Acc. (ns)	Burst Acc. (ns)	Active Pwr (mW)	Idle Pwr (mW)	Line Cap. (pF)	Pin Cap. (pF)	Manuf.
FLASH	80	N/A	75	0.5	4.8	10	Intel
BFLASH	80	40	600	2.5	4.8	10	TI
SRAM	90	N/A	185	0.1	8	8	Toshiba
BSRAM	90	45	365	1.7	8	8	Micron
BSDRAM	30	15	430	10	8	8	Micron
L2 cache	20	10	1985	330	3.2	5	Motorola

TABLE III HARDWARE CONFIGURATIONS

Name	Instruction Memory	Data Memory	L2 cache Present
Original	FLASH	SRAM	no
L2 cache	FLASH	BSDRAM	yes
Burst SRAM	BFLASH	BSRAM	no
Burst SDRAM	BFLASH	BSDRAM	no

memory constitutes a very small portion of the total energy due to the relatively large L1 cache in comparison to the MPEG code size. The DC–DC converter consumes a significant amount of total energy and thus should be considered in system simula- tions. We conclude from this example that using faster and more power-hungry memory can be energy efficient.

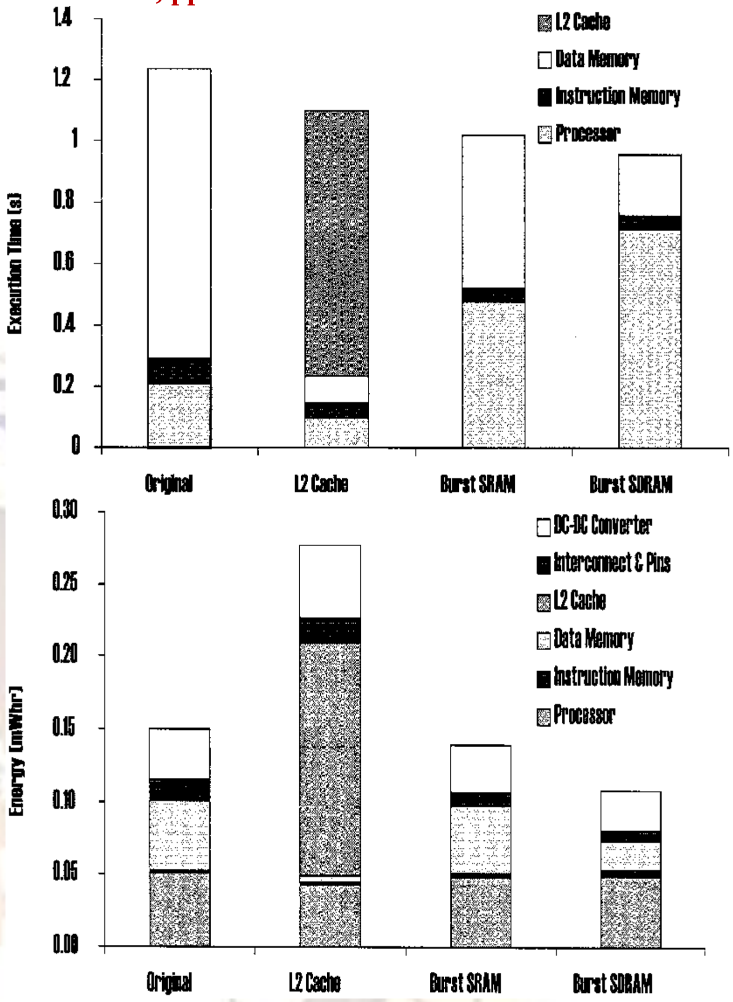


Fig. 6. Performance and energy consumption for hardware architectures

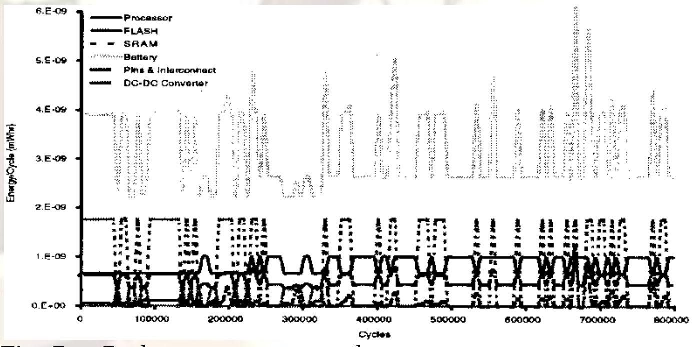


Fig. 7. Cycle-accurate energy plot

The analysis of peak energy consumption and the fine tuning of the architectures can be done by studying the energy con- sumption and the memory access patterns over a period of time. Fig. 7 shows the energy consumption over time of the processor with burst FLASH and SRAM. Peak energy consumption can reach twice the average consumption, so the thermal character- istics of the hardware design, the DC–DC converter, and the bat- tery have to be specified accordingly.

For best battery utilization, it is important to match the current consumption of the embedded system to the discharge charac- teristic of the battery. On the other hand, the more capacity bat- tery has, the heavier and more expensive it will be. Fig. 8 shows that the instantaneous battery efficiency varies greatly over time with MPEG decode running on the hardware described above.

Lower capacity batteries have larger efficiency losses. Fig. 9 shows that the total decrease in battery lifetime when contin- ually running MPEG algorithm on a battery with lower rated discharge current can be as high as 16%. The battery's time con- stant was set to ms.

The design exploration example presented in this section il- lustrates how the methodology for cycle-accurate energy con- sumption simulation can be used to select and fine-tune hard- ware configuration that gives the best tradeoff between perfor- mance and energy consumption.

The main limitation of the cycle-accurate energy simulator is that the impact of code optimizations is not easily evaluated. For example, in order to evaluate energy efficiency of two different implementations of a particular portion of software, the designer would need to obtain cycle-by-cycle plots and then manually relate cycles to the software portion of interest. The profiling methodology presented next addresses this limitation.

VI. PROFILING OF SOFTWARE ENERGY CONSUMPTION

The profiler architecture is shown in Fig. 10. The shaded por- tion represents the extension we made to the cycle-accurate en- ergy simulator to enable code profiling. Profiling for energy and performance enables designers to identify those portions of their source code that need to be further optimized in order to either decrease energy consumption, increase performance, or both. Our profiler enables designers to explore multiple dif- ferent hardware and software architectures, as well as to do sta- tistical analysis based on the input samples. In this way the de-

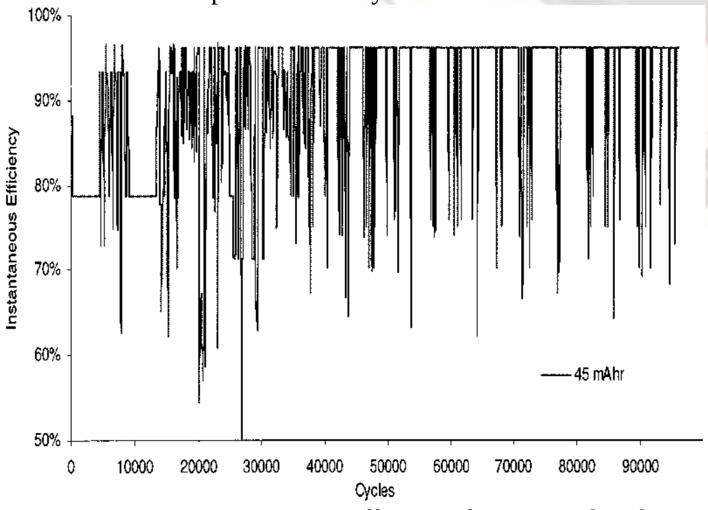


Fig. 8. Battery efficiency for MPEG decoder.

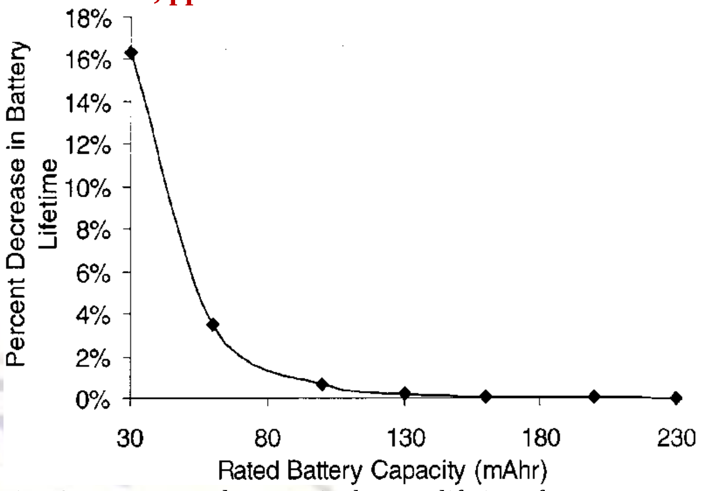


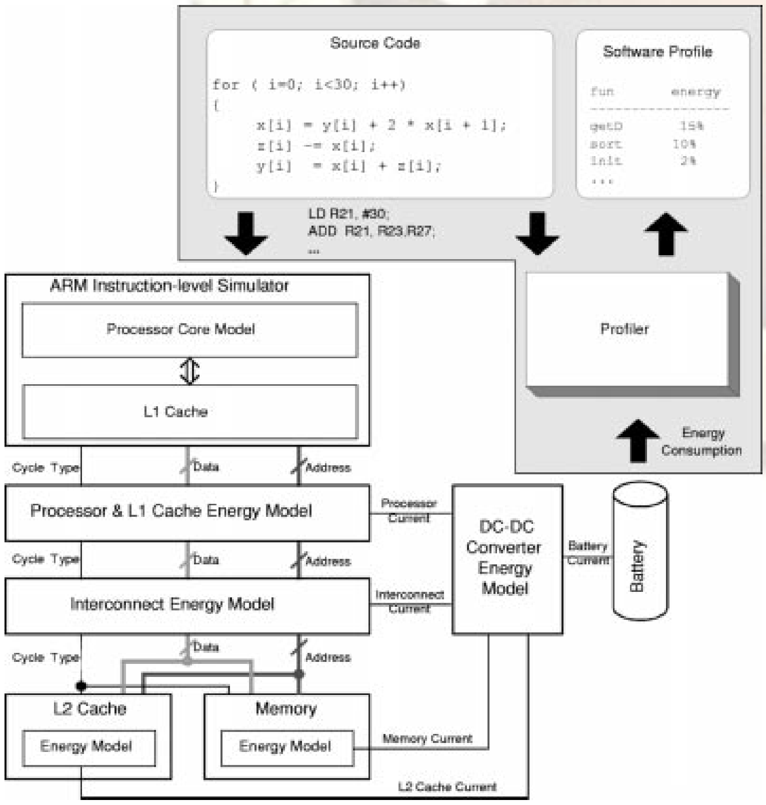
Fig. 9. Percent decrease in battery lifetime for MPEG decoder

sign can be optimized for both energy consumption and perfor- mance based on the expected input data set.

The profiler operates as follows. Source code is compiled using a compiler for a target processor (e.g., application or op- erating system code). The output of the compiler is the exe- cutable that the cycle-accurate simulator executes (represented in this figure as assembly code that is input into the simulator) and a map of locations of each procedure in the executable that a profiler uses to gather statistics (the map is correspondence of assembly code blocks to procedures in "C" source code). In order to increase the simulation speed, a user-defined profiling interval is set so that the profiler gathers statistics only at pre- determined time increments. Usually an interval of 1 s is suf- ficient. Note that longer intervals will give slightly faster ex- ecution time, with a loss of accuracy. Very short intervals (on the other of a few cycles) have larger calculation overhead. For example, energy consumption calculation gives approximately 10% overhead to standard cycle-accurate performance simula- tion. Profiling with an interval of 1 s gives negligible overhead over energy simulation (less then 1%), with still accurate results. During each cycle of operation, the cycle-accurate energy consumption simulator calculates the current total executiontime and energy consumption of all system components as shown in (1). The profiler works concurrently with the cycle-accurate simulator. It periodically samples the simulation results (using sample interval specified by the user) and maps the energy and performance to the function executed using information gathered at the compile time. Once the simulation is complete, the results of profiling can be printed out by the total energy or time spent in each function.

The main advantage of the profiler is that it allows designers to obtain energy consumption breakdown by procedures in their source code after running only one simulation. This information is of critical importance when designing an embedded system, as it enables designers to quickly identify and address the areas in the source code that will provide largest overall energy sav- ings. A good example of profiler usage is shown in Table IV. The table shows a portion of energy profile for MP3 audio de- code. The first column gives the name of the top procedure, fol- lowed by its children. The next column gives the total energy spent for that procedure. For example, the total energy spent running the program ( ) is 0.32 mWhr. The final column gives the amount of energy spent only in that particular proce- dure. For example, under it is clear that and its descendants spend the most energy, 0.0671 mWhr. Looking at the entry for , it is easy to see that the largest portion of energy is consumed by its child, . There- fore, the procedures to focus optimization on are and . Although in this example we showed source code profile of total battery energy consumption, the pro- filer can report energy consumption for any system component, such as SRAM or the interconnect.

The profiler allows for fast and accurate evaluation of software and hardware architectures. Most importantly, it gives good guidance to the designer during the design process without requiring manual intervention. In addition, the profiler accounts for all embedded system components, not just the processor and the L1 cache as most general-purpose profilers do. In the next section we present a real design example



```
for ( i=0; i<30; i++)
{
    x[i] = y[i] + 2 * x[i + 1];
    z[i] -= x[i];
    y[i]  = x[i] + z[i];
}
```

```
LD R21, #30;
ADD  R21, R23,R27;
...
```

```
fun          energy
--------------------
getD          15%
sort          10%
init           2%
...
```

Fig. 10. Profiler architecture.

uses the profiler to guide the implementation of the source code optimizations described earlier for the MP3 audio decoder running on the martBadge.

VII. OPTIMIZING MP3 AUDIO DECODER

The block diagram of the MPEG Layer III audio decoding algorithm (MP3) is shown in Fig. 11. It consists of three blocks: frame unpacking, reconstruction, and inverse mapping. The first step in decoding is synchronizing the incoming bitstream and the decoder. Huffman decoding of the subband coefficients is performed before requantization. Stereo processing, if applicable, occurs before the inverse mapping which consists of an inverse modified cosine transform (IMDCT) followed by a polyphase synthesis filterbank. We obtained the original MP3 audio decoder software from the International Organization for Standardization [28].
SubbandSynthesis
Our design goal was to obtain real-time performance with low energy consumption while keeping in full compliance with the MPEG standard.
Given the limited compiler support available [16], our ap- proach to code optimization is based on manual code rewriting and optimization guided by our profiler. Code transformations are applied in layers, starting from a high level of abstraction and moving down to very detailed and architecture-specific op- timization. In the next three subsections, we will describe in detail the three optimization layers, moving from high to low abstraction. The results of optimizations applied to the MP3 de- coder will be presented in the last subsection. Note that all the optimizations presented in the following subsections were per- formed manually.

A. Algorithmic Optimization

The top layer in the optimization hierarchy targets algorithms. The original specification is first profiled to identify all compu- tational kernels, i.e., the procedures where most time and power are spent. Each computational kernel is then analyzed from a functional viewpoint. Then, the alternative algorithms for im- plementing the same functionality are considered and compared

˘ IMUNIĆ *et al.*: ENERGY-EFFICIENT DESIGN OF BATTERY-POWERED EMBEDDED SYSTEMS

Table IV
SAMPLE ENERGY PROFILING

Name	Cumulative (mWhr)	Self (mWhr)
main	3.20E-01	2.52E-02
...		
III_hybrid		6.71E-02
SubBandSynthesis		3.72E-02
III_stereo		2.75E-02
III_reorder		2.02E-02
III_antialias		1.45E-02
III_dequantize_sample		1.40E-02
III_hufman_decode		3.74E-03
III_get_scale_factor		1.28E-04
decode_info		3.20E-05
...		
III_hybrid	6.71E-02	6.36E-03
inv_mdctL		6.07E-02
SubBandSynthesis	3.72E-02	1.95E-02
chendct32_scaled		1.77E-02
III_stereo	2.75E-02	2.75E-02
III_reorder	2.02E-02	2.02E-02
III_antialias	1.45E-02	1.45E-02
III_dequantize_sample	1.40E-02	1.40E-02
III_hufman_decode	3.74E-03	1.53E-03
huffman_decoder		2.17E-03
initialize_huffman		1.03E-05
hsstell		3.20E-05

with the original one. At this level of abstraction, we consider only high-level estimators of algorithmic efficiency (such as number of basic operations).

Our approach to algorithmic optimization in MP3 decoding has been conservative. First, we focused on just one computa- tional kernel where a large fraction of run time (and power) was spent, namely the *subband synthesis*. Second, we did not try to develop new original algorithms but we used previously pub- lished algorithmic enhancements [29], [30] that are still fully compliant to the MPEG standard. The new algorithm incorpo- rates an integer implementation of the scaled Chen discrete co- sine transform (DCT) instead of a generic DCT in the polyphase synthesis filterbank. The use of a scaled DCT reduces the DCT multiply count by 28%.

B. Data Optimization

At a lower level of abstraction than the algorithmic level, we optimize code by changing the representation of the data ma- nipulated by the algorithms. The main objective is to match the characteristics of the target architecture with the processed data. In our case, the executable specification of the MPEG de- coder performed most computations using doubles, while the processor SA-1100 has no hardware floating point support. As a result, a direct implementation of the decoding algorithm, even after algorithmic optimization, was unacceptably slow and power-consuming. Trying to reduce the precision of floating point computation, such as discussed in [31], would have helped only marginally as the processor would have to emulate in soft- ware all the floating point operations.

To overcome this problem, we developed a fixed-precision library and we implemented all computational kernels of the algorithm using fixed precision numbers. The number of dec- imal digits can be set at compile time. The ARM architecture is designed to support computation with 32-bits integers with maximum efficiency. Hence, little can be gained by reducing data size below 32 bits. On the other hand, when multiplyingtwo 32-bit numbers, the result is a 64-bit number and directly truncating the result of a multiplication to 32 digits frequently leads to incorrect results because of overflow. To increase ro- bustness, 64-bit numbers have been used for fixed-point compu- tation. This data type is supported by the ARM compiler through the definition of a `long long` integer type. Computing with `long long` integers is less efficient than using 32-bit integers, but results are accurate and the risk of overflow is minimized.

Data optimization produced significant energy savings and speedups for almost all computational kernels of MP3 without any perceivable degradation in quality. The fixed-point library developed for this purpose contains macros for conversion from fixed-point to floating point, accuracy adjustment, elementary function computation.

C. Instruction Flow Optimization

Moving further down in abstraction level, the third layer of optimizations targets low-level instruction flow. After extensive profiling, the most critical loops are identified and carefully analyzed. Source code is then rewritten to make computation more efficient. Well-known techniques such as loop merging, unrolling, software pipelining, loop invariant extraction, etc. [36], [35] have been applied. In the innermost loops, code can be written directly as inline assembly, to better exploit specialized instructions.

Instruction flow optimizations have been extensively applied in the MP3 decoder, obtaining significant speedup. We do not describe these optimizations in detail because they are common knowledge in the optimizing compilers literature [36], [35]. However, in our case most optimizations were performed manually due to lack of support by the ARM compiler.

A simple example of this class of transformation is the use of the multiply-accumulate instruction ( ) available in the ARM SA-1100 core. The inner loops of subband synthesis and inverse modified cosine transform (the two key computational kernels of MP3 decoder) contain matrix multiplications which can be implemented efficiently with multiply-accumulate. In this case, we forced the ARM compiler to use the instruc- tion by inlining it in assembly.

D. Results of MP3 Audio Decode Optimization

Table V shows the top three functions in energy consumption for each code revision we worked on. The original code has a very large overhead due to floating point emulation, about 80% of energy consumption. The next largest issue is the redesign of SubBandSynthesis function that implements the polyphase syn- thesis filterbank. The details of each optimization type, namely algorithmic, data, and instruction-level optimizations, have been presented above.

We will use the SubBandSynthesis function redesign as a ve- hicle to illustrate the use of our profiler. In the initial stage, we transferred all critical operations to fixed-point from floating point. The transfer resolved the issue with floating-point opera- tions but at the same time increased SubBandSynthesis fraction of total energy six times. Next we introduced a series of instruc- tion-level optimizations that resulted in a 30% decrease of Sub- BandSynthesis fraction of total energy, to 34.32% as shown in

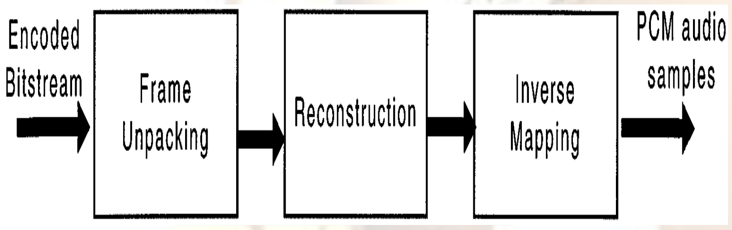


Fig. 11. MP3 audio decoder architecture.

Table V
PROFILING FOR MP3 IMPLEMENTATIONS

MP3 Code Rev.	1st	2nd	3rd
Original code	Floating Pt. 80.31%	SubBandSynthesis 10.31%	III_stereo 1.43%
Algorithmic Opts.	Floating Pt. 62.73%	III_stereo 6.12%	III_reorder 5.62%
Data & Instruction	SubBandSynthesis 34.32%	inv_mdctL 18.22%	III_stereo 7.32%
Combined Opts.	inv_mdctL 18.98%	III_stereo 8.61%	main 7.87%

Table VI ENERGY FOR MP3 IMPLEMENTATIONS

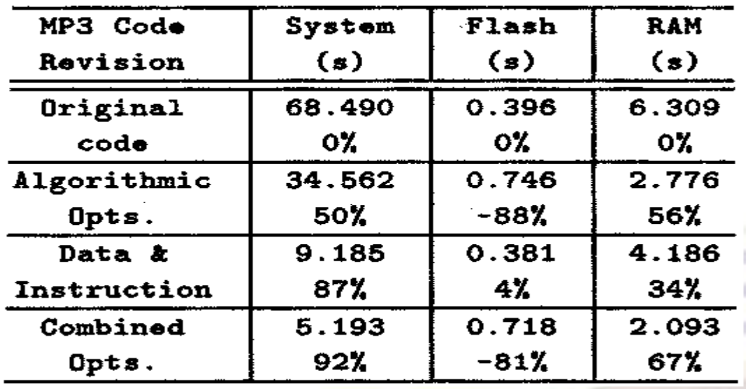
MP3 Code Revision	Battery (mWhr)	CPU (mWhr)	Flash (mWhr)	RAM (mWhr)	DC-DC (mWhr)	Lines (mWhr)
Original code	0.446 0%	0.089 0%	0.005 0%	0.178 0%	0.045 0%	0.129 0%
Algorithmic Opts.	0.107 76%	0.020 77%	0.007 -44%	0.040 77%	0.011 76%	0.029 77%
Data & Instruction	0.130 71%	0.025 71%	0.004 27%	0.051 71%	0.013 71%	0.037 71%
Combined Opts.	0.105 77%	0.019 78%	0.007 -41%	0.040 78%	0.010 77%	0.028 78%

the Table V. In parallel we had decided to try the algorithmic changes on the current code.

Profiling results in Table V show that the algorithmic opti- mizations considerably reduced the energy consumption of Sub- BandSynthesis function—it does not appear in the top three func- tions, and in fact it is only 3.2% of the total energy consump- tion. The final step is to combine the algorithmic changes with the data and instruction-level changes, resulting in decrease of Sub- BandSynthesis fraction of energy consumption to 6% of total.

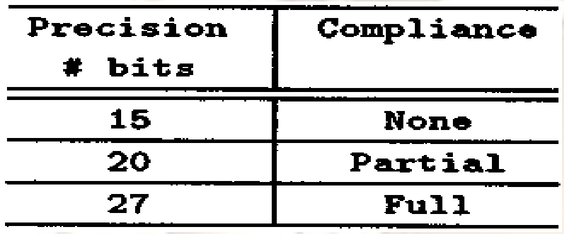
System and component energy consumptions are shown in Table VI for different revisions of source code optimization. Positive percentages show energy decrease with respect to the original code. Table VII shows the same results but for performance measurements. Positive percentages show perfor- mance increase. Although the energy savings of algorithmic versus data and instruction-level optimizations as compared to original code are comparable, the performance improvement of data and instruction-level optimizations is significant. Note that the increase in energy consumption and the decrease in performance of Flash is due to the increase in code size with the algorithmic change in SubBandSynthesis procedure. The total improvement in system performance and energy consumption more than makes up for the degradation of Flash performance and energy consumption. Combined optimiza- tions give real-time performance for MP3 audio decode which is a primary constraint for this project. In addition, lower energy consumption enables longer battery life. Note that faster implementation that is also more energy efficient might imply higher power consumption, which can be an issue for thermal design of the device. In the case presented in this paper, it

Table VII
PERFORMANCE FOR MP3 IMPLEMENTATIONS



MP3 Code Revision	System (s)	Flash (s)	RAM (s)
Original code	68.490 0%	0.396 0%	6.309 0%
Algorithmic Opts.	34.562 50%	0.746 -88%	2.776 56%
Data & Instruction	9.185 87%	0.381 4%	4.186 34%
Combined Opts.	5.193 92%	0.718 -81%	2.093 67%

Table VIII
FIXED-POINT PRECISION AND COMPLIANCE



Precision # bits	Compliance
15	None
20	Partial
27	Full

was critical to get real-time performance with longer battery lifetime. The average and peak power consumption constraints are met with our final design.

The final MP3 audio decoder compliance to the MPEG stan- dard has been tested as a function of precision for fixed-point computation. We used the compliance test provided by the MPEG standard [32], [33]. The range of RMS error between the samples defines the compliance level. Table VIII shows that results. Clearly, the larger number of precision bits results in better compliance. In our final MP3 audio decoder we used 27 bits precision. Using our design tools to guide the software optimization process, we have been able to increase performance by 92% while decreasing energy consumption by 77%, with full com- pliance to the MP3 audio decode standard.

VIII. CONCLUSION

I developed a methodology for cycle-accurate simulation of performance and energy consumption in embedded systems. Accuracy, modularity, and ease of integration with the instruc- tion-level simulators widely used in industry make this method- ology very applicable to the embedded system hardware and soft- ware design exploration. Simulation is found to be within 5% of the hardware measurements for Dhrystone benchmark. We presented MPEG video decoder embedded system design explo- ration as an example of how our methodology can be used in prac- tice to aid in the selection of the best hardware configuration.

I have also developed a tool for profiling energy consump- tion of software in embedded systems. Profiling results enabled us to quickly and easily target the redesign the MP3 audio de- coder software. Our final MP3 audio decoder is fully compliant with the MPEG standard and runs in real time with low energy consumption. Using our design tools we have been able to in- crease performance by 92% while decreasing energy consump- tion by 77%.

REFERENCES

[1] Advanced RISC Machines Ltd (ARM), *ARM Software Development Toolkit Version 2.11*, 1996.

[2] G. Q. Maguire, M. Smith, and H. W. P. Beadle, “SmartBadges: A wear- able computer and communication system,” in *Proc. 6th Int. Workshop Hardware/Software Codesign*, 1998, Invited talk.

[3] CoWare. *CoWareN2c* [Online]. Available: url:www.coware.com/n2c. html [4] Mentor Graphics. [Online]. Available: www.mentor.com/codesign

[5] Synopsys. [Online]. Available: www.synopsys.com/products/hwsw

[6] Cadence. [Online]. Available: www.cadence.com/alta/products

[7] Y. Li and J. Henkel, “A framework for estimating and minimizing energy dissipation of embedded HW/SW systems,” in *Proc. Design Automation Conf.*, 1998, pp. 188–193.

[8] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye, “Energy-driven integrated hardware–software optimizations using SimplePower,” in *Proc. 27th Int. Symp. Computer Architecture*, 2000, pp. 24–30.

[9] J. Flinn and M. Satyanarayanan, “PowerScope:A tool for profiling the energy usage of mobile applications,” in *Proc. 2nd IEEE Workshop Mo-bile Computing Systems Applications*, 1999, pp. 23–30.

[10] B. Kapoor, “Low power memory architecutres for video applications,” in *Proc. 8th Great Lakes Symp. VLSI*, 1998, pp. 2–7.

[11] M. Lajolo, A. Raghunathan, and S. Dey, “Efficient power co-estimation techniques for SOC design,” in *Proc. Design, Automation Test Europe Conf.*, 2000, pp. 27–34.

[12] T. Givargis, F. Vahid, and J. Henkel, “Fast cache and bus power estima- tion for parameterized SOC design,” in *Proc. Design, Automation Test Europe Conf.*,

2000, pp. 333–339.

[13] OZ Electronics Manufacturing. *PCB Modeling Tools* [Online]. Avail- able: url: www.oem.com.au/manu/pcbmodel.html

[14] A. El Gamal and Z. A. Syed, "A stochastic model for interconnections in custom integrated circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 888–894, Sept. 1981.

[15] T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," in *Proc. Design Automation Conf.*, 1999, pp. 867–872.

[16] T. Simunic, L. Benini, G. De Micheli, and M. Hans, "Energy-efficient design of battery-powered embedded systems," in *Int. Symp. Low-Power Electronics Design*, 1999, pp. 212–217.

[17] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vanduoppelle, *Custom Memory Management Methodology: Explo- ration of Memory Organization for Embedded Multimedia System De-sign.* New York: Kluwer, 1998.

[18] M. Pedram and Q. Wu, "Battery-powered digital CMOS design," in *Proc. Design, Automation Test Europe Conf.*, 1999, pp. 17–23.

[19] H. Mehta, R. M. Owens, M. J. Irvin, R. Chen, and D. Ghosh, "Tech- niques for low energy software," in *Proc. Int. Symp. Low Power Elec- tronics Design*, 1997, pp. 72–75.

[20] M. Kandemir, N. Vijaykrishnan, M. Irwin, and W. Ye, "Influence of compiler optimizations on system power," in *Proc. 27th Int. Symp. Com- puter Architecture*, 2000, pp. 35–41.