Power Minimization Derived from Architectural-Usage of VLIW Processors

C.Gebotys, R.Gebotys[1], S.Wiratunga
Department of Electrical and Computer Engineering,
University of Waterloo, Wilfrid Laurier University[1],
Waterloo, Ontario Canada N2L 3G1

Abstract

This paper presents an empirical approach to inferring low power code generation techniques for VLIW processors. Architectural usage variables are used to generate equations for power prediction which are in turn used to infer new code generation techniques for low power. Unlike previous techniques, the methodology empirically derives a power prediction equation and then based upon the coefficients of the architectural-usage variables identifies new VLIW code generation techniques for low power. The approach is illustrated using functional unit usage within a VLIW architecture and identifies a new operation rebinding technique for low power which improved power dissipation up to 18%. The approach is general and results are verified with real power measurements. This result is important for developing a general methodology for power minimization of embedded DSP software since low power is critical to complex DSP applications in many cost sensitive markets.

1 Introduction

Power is a growing problem especially as newer DSP processors continue to operate at higher frequencies (directly causing higher power dissipation). Even techniques which offer a small percent improvement in power are regarded as important. There is a growing need for power prediction models and for better understanding of how DSP embedded code can be modified to reduce power. As processors become more complex (greater parallelism, subword execution, pipelines, etc...) the power dissipation problem is also expected to become more complex.

Design of DSP embedded systems is a challenging process that deals with increasingly difficult applications (increasing functionality and changing standards), high performance constraints and low power dissipation requirements. Due to rapidly changing standards and applications a low risk programmable solution is typical using DSP programmable processor cores (combined with memory and special purpose functional units). Research has examined several techniques for meeting performance constraints[9] , however power has received very little attention in the research field until recently[1]. Embedded systems designers cannot use most existing low power techniques researched due to lack of detailed gate level representations of the processor being used (often due to proprietary or monetary constraints).

The energy dissipation of a processor running a program [2], E, is the product of the time required to execute the program (T), and the power dissipation (P). The average current (I) multiplied by the supply voltage (Vdd) gives the power (P) as in the equation below. The term T is equal to $N * \tau$, where N is the number of clock cycles and τ is the clock period, $E = P*T = I*Vdd*T = I*N*\tau*Vdd$. The measured current I is an important parameter for embedded systems design that needs to be studied in order to study how to generate low power software for DSP processors.

2 Related Research

In the area of low power, researchers have developed architecture-level models to be used in a simulation environment or higher level tools. Memory components, controllers [5], instruction registers of microprocessors [7,3], are examples of some components that are known to dissipate significant power in addition to datapath components. Researchers have tried to schedule operations[5], or swap operands [1] to reduce data bit switching. Researchers have also employed parallel instructions to improve performance which also reduced energy such as using parallel data transfer instructions[2]. Only a few of these researchers have verified these values as actual physical savings in energy[2,3].

Physical current measurement as a means of measuring power was used by researchers for analysis of processors in [1,2]. An instruction-level model of power was derived, consisting of a base power cost per instruction along with a overhead cost related to the next or nearby instruction. Their power prediction model achieved 10% error. Researchers in [3] formulated power prediction models for two different processors with error up to 4%. Many researchers have also shown that faster programs consume less energy [4] and have presented techniques that save power by producing faster or higher performance code generation techniques. Thus power prediction tools are necessary and important for embedded systems design.

Although the methodology presented in this paper is general, the TMS320C6201 processor[10] (C62) is used to obtain actual current measurements. The C62 is a complex VLIW-based (very long instruction word) DSP processor with eight parallel functional units (two multipliers, two address generation units, and four alu types of units). Code is generated from commercial TI C Compilers, in-house scheduling tools, and hand-generated assembly code. The methodology however is general and can be applied to any processor. The next section will outline the methodology and experimental setup to be used in the rest of the paper.

3 Experimental Approach

This section will briefly describe the methodology including the DSP processor, power measurement setup, the DSP programs, the variables, and the statistical analysis methods used in this paper.

Figure 1 shows the power minimization methodology driven by architectural-feature usage variables for embedded systems DSP design. Initially the DSP benchmark codes ($c \in \{1, ..., d\}$) are run on the DSP processor hardware using pseudorandom (and voice) type of input data and the currents are measured (see Current Measurement box). The average current (I_m^c) for each DSP code is recorded. Variables (u_i^c,$i \in \{1, ..., m\}$) are extracted from the DSP benchmark code (c). Variables represent architectural usage features (u_a^c, $a \in \{1, ..., n\}$). For example one might wish to identify how to bind operations to functional units for low power and define variables as the fraction of total instructions executed by each functional unit. These functional unit usage variables are combined with other basic variables (u_b^c, $b \in \{n+1, ..., m\}$) necessary to stabilize the power prediction equation. The power-prediction equation (

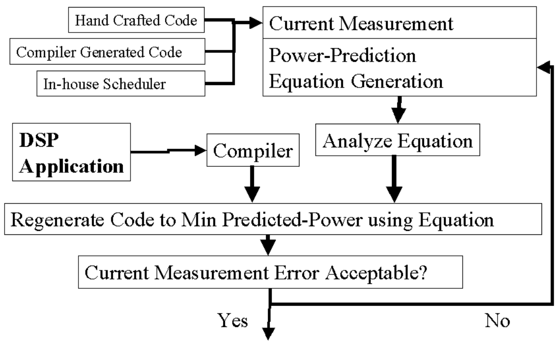
Hand Crafted Code
Compiler Generated Code
In-house Scheduler
Current Measurement
Power-Prediction Equation Generation
DSP Application
Compiler
Analyze Equation
Regenerate Code to Min Predicted-Power using Equation
Current Measurement Error Acceptable?
Yes
No

Figure 1: Methodology for minimizing power through DSP code generation.

$I_p^c = f(u_i^c) = \sum_{i=0}^{i=m} c_i u_i^c$, where $u_0^c = 1$) is obtained automatically using a linear regression technique using these variables (u_i^c,$i \in \{1, ..., m\}$) as predictors. The output is a set of coefficients c_i from equation for I_p^c which predicts current. The embedded systems designer then analyzes the model, looking at the smallest sized coefficients of the architectural-usage variables, to see if code can be regenerated to maximize usage of this architectural feature (which implies minimizing the usage of another architectural feature which has a large coefficient in the equation, since variables represent fraction of architectural usage). If the user can not rearrange the code to maximize usage of this functional unit, the next smallest coefficient identifies an alternative feature to maximize. Code is regenerated (using some technique such as rebinding operations to functional units, rescheduling as in figure 2 to allow for rebinding, or recompilation followed by rebinding) in an attempt to obtain code that minimizes the predicted-power defined by the equation. If the predicted current of new code is lower, it's real current is measured and if the error is not acceptable then it is added to the benchmark set and a new predictive power equation is generated. Otherwise if the error is acceptable a new technique for power minimization in embedded systems has been found.

The architectural-feature driven algorithm to explore software generation for low power is outlined below:

1) Define Variable Set: $[u_1^c, u_2^c, ..., u_m^c, I_m^c] \in S$, where I_m^c = measured current of code c, and u_i^c = variable i of code c.

$i \in F = \{1, 2, ..., n\}$, identifies architectural-feature variable set.

$i \in B = \{n+1, n+2, ..., m\}$, identifies basic variable set .

2) Perform Linear Regression, ie. find predictive-power equation:

Find $c_0, c_{1,...,}c_m \ni I_m^c - \{c_0 + c_1u_1^c + c_2u_2^c + ... + c_mu_m^c\}$

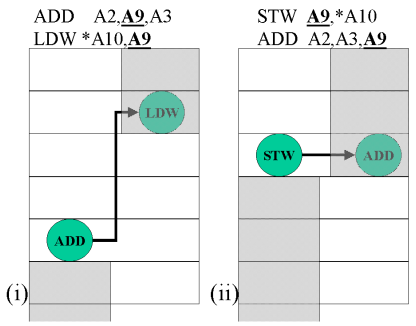

Figure 2: Rescheduling to support greater operation rebinding and parallelism using delay slots.

$u_i^c \setminus c$	*iir*	*loop*	*dct*
P	1	5.5	2.5
B	11	416	0
S_1	.076	.133	.259
M_2	.152	.108	.058
D_1	0	.141	.154
L_1	.152	.145	.124
L_2	.095	.109	.094
M_1	0	.035	.087
$\mathbf{I}_p^c$ (A)	.564	.736	.611
I_m^c **(A)**	.575	.730	.681
$\mathbf{I}_p^c - I_m^c$	-0.01	0.0054	-0.069

Table 1: Codes and their equation variables.

is minimum, $\forall c$.

Is error (minimum) low enough?, If not, add more basic variables into set S, and go to 1). otherwise go to 3)

3) Choose minimum coefficient of architectural-feature set $i| \min_{i \in F} c_i$.

If one cannot recode program to maximize this architectural feature set $F = F \backslash i$, $B = B \cup i$ and go to 3). otherwise go to 4).

4) Recode the user's application or DSP program (not a part of benchmark DSP set) , to create a new program (*new*) to maximize this architectural feature, i. Compute predictive power of new code, measure actual current of new code, I_m^{new}, If error$=I_m^{new} - \{c_0 + c_1 u_1^{new} + ... + c_m u_m^{new}\}$ is small enough for designer, you can stop, you've found a technique for reducing power. Otherwise , if error is too large, add this program into benchmark set and regenerate predictive power equation, ie. $S = S \cup [u_1^{new}, u_2^{new}, ..., u_m^{new}, I_m^{new}]$, and go to 2).

4 Experimental Results

The architectural-feature driven methodology to inferring methods of generation code that is low power for a particular DSP processor is illustrated with the C62 test evaluation board (along with an in-house designed interface board). An elaborate warmup and calibration was performed before and after each DSP program in order to verify power readings were correct and reproducable. An initial set of benchmark codes (hand-coded assembly, in-house scheduler and compiler generated versions of common DSP applications, such as the fast fourier transform, least means squares, high pass filter, discrete cosine transform, vector compare algorithms etc) were used. The objectives were to use this methodology to help determine if it were possible to exploit operation binding to functional units in the datapath in order to generate software with minimum power. The C62 power-prediction equation was a function of two basic variables (u_b^c) the parallelism (P, average number of non-NOP instructions performed per cycle), and the total number of branches (B) performed by the DSP code during one iteration of the program. The benchmark codes had parallelism ranging from 1 to 5.5 and branch variables ranging from 0 to over 400 (see table 1, as an example of some codes). The fraction of total non-NOP instructions performed by each functional unit (S_i, M_i, D_i, L_i, i=1, 2 [11]) during one iteration of the program formed the architectural-feature variable (u_a^c) set. Specifically, the predicted power measured in Amperes is I_p^c= $0.49 + 3.59 \times 10^{-3} \times P + 3.14 \times 10^{-4} \times B + 0.194 \times S_1 + 0.186 \times M_2 + .065 \times M_1 + 0.147 \times D_1 + 0.211 \times L_2 + .025 \times L_1$. The R^2value for this power prediction equation was 0.930 (indicating that the equation could account for 93% variation in current). The predicted current had minimum and maximum absolute current errors of $+0.1049$ and -0.1009 Amps and an average error of 4% overall. Table 1 provides an example of the variables used in the C62 power prediction model. The *loop* is a handcoded vector compare code (taken from [10]), the *dct* is the in-house scheduler version of a discrete cosine transform code example, and the *iir* is compiler generated code of an IIR filter. Table 1 illustrates how three very different codes can be used in same model to predict current with small errors (error) using the C62 power prediction equation. During the architectural-feature driven algorithm presented in this paper, the coefficients for each type of unit is summed (ie. L_{1+2} units have coefficient 0.236 and M_{1+2} units have coefficients 0.252). Thus the functional unit with the smallest coefficient is the D_{1+2} unit with coefficient of 0.147.

As an example of the functional usage-driven methodology for generating software for low power, an fft application (not a part of the initial benchmark set) was used. It required 0.5519A measured current and was predicted to have current of 0.5577A from the power-prediction equation. Since the D units had the smallest coefficients overall in the equation, the operations (*add*, *sub*, *mv*, *neg*[11]) were redistributed among functional units in this application. Specifically these operations which were assigned to L and S units in original code, where possible, were reassigned to D units. This resulted in fractional usage of D_{1+2} units changing from .57 to .69 and the fractional usage of L/S_{1+2} units combined changing from .307 to .211. The new code, had a predicted current of 0.5355A. The measured current for this new code was 0.5325A which verified the improvement in power of 3.6 %. The new code has the same performance as the original code, thus showing that it was possible to find ways of minimizing power for the same performance even with complex VLIW processors. A different dct application was rescheduled to increase the usage of D units by 0.455 over the combined usage of L and S units, and produced improvement in power from .6258 A to .53 A, an overall improvement of 18% (and predicted power had absolute error of only 0.033A).

5 Discussions and Conclusions

This new architectural-feature driven methodology for power minimization demonstrated power savings is possible even for complex DSP VLIW processors. The methodology was demonstrated for a functional unit usage exploration. It was found that by changing the fractional usage of different units in the VLIW, power could be saved. In particular the D units were identified as being more power efficient. This finding appears to make logical sense. Since the D units in the C62 support 2 to 3 times less functionality than the L and S units, it is highly likely that they would dissipate less power, since they would have less circuitry (or less power wastage). In fact experimentation through running single instruction codes on different functional units proved this to be the case. However the approach presented in this paper is general and can be applied to study other architectural features, which may prove useful to saving power. Using the C62 processor, parallelism and redistribution of functional unit usage (ie. performance, power) were limited by the differential capabilities of the functional units and their register file limitations (unlike newer architectures such as [8] which do not have these architecture constraints).

For the first time, an approach to identifying new ways of saving power through software has been developed. This approach is based upon real power measurements, and architectural-feature usage to infer new ways of decreasing power. This is important for embedded system designers who are designing with an existing processor or processor core. These designers are involved in the software design, where only access to the hardware is available and specifically detailed simulation models (from which register or bus switching data can be obtained) are typically not available. Results showed that predictive-power equations based upon variables derived from the code itself, could be used to infer new methods for lower power. A wide range of code, with parallelism varying from single functional unit usage to all 8 functional units usage in the VLIW processor, and branches varying from negligible to over 1000 branches, was used in the benchmarks to derive the useful predictive power equation.

Unlike previous research, this research addresses the difficult problem of determining how to write software code for a DSP processor to reduce power. These results show that it is possible to decrease power independently from performance and energy. Furthermore since power is an increasingly growing problem, the power experimentally lowered in this paper, up to 18 percent improvement is significant and important. This methodology is general (see figure 1) and can be applied to any processor. This research is important for industry since a methodology for developing power prediction for DSP code is critical as new processors or cores grow in complexity and become integrated into embedded DSP systems with stringent power, performance, and cost constraints. This research is supported by NSERC.

References

[1] V.Tiwari, S.Malik, A.Wolfe, "Power Analysis of Embedded Software ", IEEE Trans on VLSI, Dec 1994, p437-445.

[2]M.Lee, V.Tiwari, S.Malik, M.Fujita, "Power Analysis and Minimization Techniques for Embedded DSP Software", IEEE Trans on VLSI Design, March 1997, p123-135.

[3]C.Gebotys, R.Gebotys, "Statistically based prediction of power dissipation for complex embedded DSP processors" Microprocessors and Microsystems Journal, 23, p135-144, 1999.

[4]V.Tiwari, S.Malik, A.Wolfe, "Compilation techniques for low energy", ISLPED Oct 1994.

[5]C.L.Su, C.Y.Tsui, A.M.Despain, "Saving power in the control path of embedded procesors", IEEE Design and Test Comp., p24-30, Dec 1994.

[6]D.Shin, K.Choi, "Low power high level synthesis by increasing data correlation", ISLPED, 1997.

[7]H.Mehta, R.Owens, M.Irwin, D.Ghosh, "Techniques for low energy software", ISLPED , p72-75, 1997.

[8]StarCore 140 Specifications, Motorola and Lucent, 1999.

[10]Y-T.S.Li, S.Malik "Performance Analysis of Embedded Software Using Implicit Path Enumeration", IEEE Trans on CAD, Dec 1997, p1477-1487.

[11]TMS320C62 CPU and Instruction Set Reference Guide, Texas Instruments Inc., January 1997.