Designing for Low Power in Complex Embedded DSP Systems

C.H. Gebotys, R.J. Gebotys
Department of Electrical and Computer Engineering,
University of Waterloo, Wilfrid Laurier University
Waterloo, Ontario, Canada N2L 3G1
cgebotys@optimal.vlsi.uwaterloo.ca

Abstract

This paper presents an empirical methodology for low power driven complex DSP embedded systems design. Unlike DSP design for high performance, research of low power DSP design has received little attention, yet power dissipation is an increasingly important and growing problem. Highly accurate power prediction models for DSP software are derived. Unlike previous techniques, the methodology derives software power prediction models using statistical optimization and it is verified with real power measurements. The approach is general enough to be applied to any embedded DSP processor. Results from two different DSP processors and over 180 power measurements of DSP code show that power can be predicted for embedded systems design with less than 4% error. This result is important for developing a general methodology for power characterization of embedded DSP software since low power is critical to complex DSP applications in many cost sensitive markets.

1. INTRODUCTION

Design of DSP embedded systems is a complex process that deals with increasingly challenging applications (increasing functionality and changing standards), high performance constraints and low power dissipation requirements. Low power dissipation is important for high reliability, increased portability and low cost production. Due to the large emphasis on flexibility (since standards and applications are rapidly changing), a low risk programmable solution is required. For this reason, embedded systems design typically involves using off the shelf DSP programmable processors or in-house DSP processor cores. The DSP core can be reused and combined with program/data memory, or special purpose functional units, and incorporated onto a large silicon chip. Even though some of these core based devices involve design of custom silicon (for example a special purpose functional unit for motion estimation), the majority of design effort involves generating code to meet performance and power constraints. Research has examined several techniques for meeting performance constraints[15], 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).

Power dissipation has a large impact on cost and reliability. It is an increasingly important problem in embedded systems designs not only for the portable electronics industry but in other areas including communications, consumer electronics, etc. Unfortunately few tools are available for designing embedded systems for low power. 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[14]. 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.

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 $N \ast \tau$, where $N$ is the number of clock cycles and $\tau$ is the clock period.

$$ E = P \ast T = I \ast Vdd \ast T = I \ast N \ast \tau \ast Vdd $$
Previous research has mainly concentrated on techniques to improve performance which in turn often improves energy (as long as \( p_i \leq \frac{p_{n}}{p_{n}} \)), where \( p_i \) = fraction of \( I \) that is increased and \( p_{n} \) = fraction of \( N \) that is reduced. However for fixed performance (or fixed throughput) which is common for DSP embedded systems it is not clear how to optimize code for power. Thus \( I \) is an important parameter for embedded systems design that needs to be studied and predicted for software energy prediction.

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 [7]. Memory components [12], controllers [9], instruction registers of microprocessors [11], are examples of some components that are known to dissipate significant power in addition to datapath components [7]. Researchers have tried to schedule operations [9], 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].

Physical current measurement as a means of measuring power was used by researchers for analysis of a general purpose processor in [1] and more recently for a DSP processor [2]. In both cases a current meter allowed power measurements on small programs (much less than 100ms) which are repeated several times in a loop until a stable current reading could be obtained. An instruction-level model of power was derived, consisting of a base power cost per instruction along with an overhead cost related to the next or nearby instruction. Their power prediction model achieved 10% error. It required characterization not only on a per instruction basis, but for pairs of instructions and beyond. Some tools were developed that provided performance improvement in code in addition to power improvements. In several cases and where data was available, their results showed that fewer instructions lead to faster code and lower energy. Many researchers have also shown that faster programs consume less energy [8] and have presented techniques that save power by producing faster or higher performance code generation techniques. Unfortunately fewer researchers have studied what can be done to optimize for power when the performance constraint (or throughput) is fixed as in many embedded DSP systems.

Embedded designers currently do not have access to sophisticated tools for power minimization or analysis. Typically only a compiler, instruction-level simulator, and the hardware itself are available. However low power is a very important criteria which they are being forced to design for. Thus power prediction tools are necessary and important for embedded systems design.

Although the methodology presented in this paper is general, the TI TMS320C5x processor [4] (C5x) and the TMS320C62xx processor [16] (C62xx) are used to obtain actual current measurements. This C5x processor has a single multiplier-accumulator datapath and is quoted by researchers as being one of the more challenging processors for code generation due to its highly heterogeneous instruction set architecture. The C62xx is a recently announced complex VLIW-based (very long instruction word) DSP processor with eight parallel functional units. Different types of data input (voice and pseudorandom data) along with variables, derived from the coded application only (for example switching in the instruction register, etc) are used to analyze the results. Code is generated from commercial TI C Compilers and researched optimization tools. 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 METHODOLOGY

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

Figure 1 shows the methodology for power prediction of embedded systems DSP design. The power prediction model is generated once (see power model generation in figure 1) from single instruction tests and DSP benchmark code. The single instruction tests involve repeating one instruction several times in a loop. This code is run on the DSP processor hardware using pseudorandom type of input data and the current is measured (see Current Measurement box). The average current \( (I_{avg}) \) for each type of instruction \((i)\) is recorded. Several DSP benchmark programs are also run with four different types of input data (two samples of voice data and two samples of pseudorandom data) and current measurements are recorded. Variables are extracted from the DSP benchmark code directly, such as the average bit switching in the instruction register, address busses, etc (available from the code directly) and a special variable, \( x_p \), where \( x_p = \frac{\sum i_i N_i}{\sum N_i} \) where \( N_i = number of instructions of
type in program. The power predictor model generation is based upon a linear regression model using these variables as predictors. The output is a model (or equation $y = ...$ etc) which predicts current. The embedded systems designer then generates code for their application and uses it with the power-prediction model to predict current. Code is regenerated (using some technique such as rescheduling or rewriting the application) in an attempt to obtain code that meets the performance constraint and minimizes the predicted power dissipation.

A. Energy Measurement Apparatus

Previous research [1], [2] has used a dual-slope integrating ammeter in series with the power supply of a microprocessor in order to obtain visual current readings that are proportional to the power consumed by the processor [1]. This method is limited by the fact that the program length has to be much shorter than the sampling rate of the meter in order to obtain a stable reading. This makes it difficult to empirically determine the power consumption of an entire program. The use of a more sophisticated meter, specifically the Fluke 607B GMM [6], allowed this limitation to be relaxed. This meter allows the user to select the period over which to average the current reading and also supports transmission of current values over an optically isolated RS232 port. This is a very useful feature that makes it possible to measure the energy consumption of an entire program.

All experiments were repeatable. After the board was powered up, a warm up period was allowed before any experiments were run. A series of NOP instruction were executed before and after each series of programs were run to calibrate any variation in power measurements due to temperature variation, etc.

B. DSP Programs

DSP code for embedded types of applications was generated using a commercial TIC Compiler, and existing optimization algorithms. A variety of common DSP applications, such as the fast fourier transform, least means squares, high pass filter, discrete cosine transform, etc were implemented. Different schedules, addressing arrangements and coding were used to study power effects. In many cases, different codes for the same filter were created with equivalent performance (for example using different schedules and address generation). One set of DSP benchmark programs was used to generate the power prediction model and a different (independent) set of DSP programs were used to verify the model.

For the DSP benchmark programs, the code sequence was a straight line basic block that was repeated several times and then placed within a loop. Each repeat of the program used a different part of the input data since it is well known that power depends upon not only the program but the signal statistics of the data [7]. For example if a DSP program used 40 words of speech data as input. The DSP program was repeated 100 times in a loop, performing computations on 4000 words of a continuous speech sample. This study was repeated with two different sets of pseudo-random generated data and two different samples of voice data.

C. DSP Processor and Instruction Power Observations

The TMS320C5x DSP processor is used to demonstrate the derivation of the power prediction model. The partial instruction set (or ISA) of this DSP processor is shown in table 1. For example instruction APAC performs $a = a + p$ and instruction MPY $m_i$ performs $p = m_i + t$, where $a, p$ and $t$ parameters are the accumulator register, product register (of the multiplier) and the input register to the multiplier respectively. The $m_i$ parameter is the memory operand, which can support direct or indirect memory addressing modes.
Current values, $I_i$ (mA), are given for individual instructions in this table. Note that in this case the parallel instructions have approximately 14% higher current, $I_i$ (mA), than the average of the two currents for the equivalent sequential instructions ($I_{MPY_{PA}}>I_{MPY_{AC}}$). For example APAC and MPY m perform the same function as MPYA m and similarly for LTA m (which performs $t=m$, and $a=a+p$).

<table>
<thead>
<tr>
<th>Instr</th>
<th>$I_i$ (mA)</th>
<th>Parallel</th>
<th>$I_i$ (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APAC</td>
<td>42.47</td>
<td>MPYA m</td>
<td>55.76</td>
</tr>
<tr>
<td>MPY m</td>
<td>53.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LTA m</td>
<td>42.47</td>
<td>LTA m</td>
<td>55.47</td>
</tr>
</tbody>
</table>

Table 1. Partial ISA for TMS320C5x.

This DSP processor has 8 address registers ($AR_i$) and the current one is pointed to by a 3 bit register ($ARP$). Most instructions can specify the next address register to be pointed to by ARP. The processor has address characteristics similar to many popular DSP processors such as M56000, TMS320C3x and the recently announced low power processor TMS320C54x DSP processor. Register based addressing ($*m$, memory address = $AR_i$), autoincrement ($*+m$, memory address = $AR_i$, $AR_i+=1$) or autodecrement ($*-m$, memory address = $AR_i$, $AR_i-=1$) and limited offset addressing ($0+, memory address=AR_i$, $AR_i+=AR0$) are supported. Addressing that requires increments to the address register that are not $0,-1,-1$ or $AR0$ incurs an extra instruction cost. An example of some variables obtained directly from analysis of the DSP benchmark code are shown in table 2. For example $IR$ in table 2 refers to the average switching of data stored in the instruction register (available from the DSP code), whereas $DABUS$ refers to the average switching of the data address bus. These variables were recorded for statistical analysis to be detailed in the next section.

<table>
<thead>
<tr>
<th>Variable(x’s)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$IR$</td>
<td>avg switching in instruction register</td>
</tr>
<tr>
<td>$DABUS$</td>
<td>avg switching in data addr bus</td>
</tr>
<tr>
<td>$PABUS$</td>
<td>avg switching in program addr bus</td>
</tr>
<tr>
<td>$MUL$</td>
<td>avg # of multiplies</td>
</tr>
<tr>
<td>$SUB$</td>
<td>avg # of subtracts</td>
</tr>
<tr>
<td>$LOAD$</td>
<td>avg # of loads</td>
</tr>
</tbody>
</table>

Table 2. Example of some model variables.

See the table below for an example which shows the current measurements for pairs of instructions with the same functionality, same addressing and same type of input data. This example also illustrates how instruction selection can affect power. Note again that taking the average of currents for single instructions in these cases can create 23% error in current values ($I_{LTA,MPY} > I_{LTA,MPY_{PA}}$).

<table>
<thead>
<tr>
<th>Instr Pairs</th>
<th>$I_i$ (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTA m1, MPY m2</td>
<td>69.7</td>
</tr>
<tr>
<td>LT m1, MPY m2</td>
<td>70.67</td>
</tr>
</tbody>
</table>

Table 3. Power variation with addressing.

D. Statistical Calculations

From the previous section one can see that trying to predict the current of a program from $x_p$ (using the average single instruction currents) can lead to a large error. As processors grow more complex it may not be possible to measure all pairs of sequential instructions to estimate this overhead as previously researched[2]. This approach becomes more difficult for multiple instruction issue, such as the case for the C62xx VLIW DSP processor[16]. A technique for automatically selecting important variables for power prediction is one method for handling this growing processor complexity. One such statistical technique will be described below to provide automatic model formulation.

A number of linear models were fit using the measured current as the dependent or $y$ variable. Several independent or predictor variables were considered, see table 2 for an example. The models were fit using a least squares algorithm to minimize the distance between the observed data and the predicted
data under the model. We assume that the $y$ data is some linear function of the $x$'s, $y = f(x)$. The least squares equation predicting average power from the $x$'s is given by the following linear equation $E(y|x) = b_0 + b_1x_1 + b_2x_2 + \ldots + b_dx_d$, where the $b_i$ represents the least squares estimates of the population parameters and $E(y|x)$ is the average or expected value of $y$ given $x$. A stepwise selection method was used to automatically find the best model for predicting current. The model reported is an excellent model statistically, model adequacy tests have $p - values < 0.001$, where $p - value$ is the observed level of significance[13].

The $R^2$ value is reported indicating the percent of variation in current accounted for by the model. The least squares equation is given predicting average current for the model (all coefficients are highly statistically significant, $p - values < 0.001$) and the standard error of prediction is given for the maximum residual as another measure of accuracy of the model. The normality assumption for statistical tests and confidence intervals was verified using normal probability plots and histograms. The statistical package SPSS[5] was used for all statistical calculations.

4. EXPERIMENTAL RESULTS

Initial observations from the DSP benchmark programs will be presented first followed by the power prediction model and its verification. Figure 2 illustrates the interdependencies of $I$ and $N$. The first bar shows the performance improvement of an optimized code which employs more parallel instructions due to improved scheduling, compared with compiler generated code (both employing direct memory addressing). The next 4 bars show the power improvement for the different types of data (R0, R1 are two different sets of pseudo random data and V0, V1 are two different voice samples). The fast fourier transform program is illustrated in figure 2. Typically the average current is higher for the faster code due to the implementation of parallel instructions, however in figure 2 the average current $I$ of the faster code decreased leading to an overall higher improvement in energy $I$.

In the tables below, different codes which implement the fast fourier transform were obtained from using different schedules ($a,b,c$) and different addressing techniques. In all cases the same input type of data was used (one sample of voice data). Note that the minimum energy solution ($I * N = 7694$) occurs at $N = 123$ clock cycles (not utilizing the fastest code, $N = 122$). Note that an energy difference of as much as 5% can occur for code generated for performance variation within 1% (ie. 123 cycles requiring 7694 = $I * N$,

![Figure 2 – FFT percent improvements in performance and power.](image)

versus 124 cycles requiring 8055 = $I * N$). As much as 4% difference in energy can occur for equivalent performance and type of input data. A previously researched methodology which chooses the fastest program for efficient energy implementation[2], for example using 122 cycles $I * N = 7949$, would in this example be dissipating unnecessarily higher energy of over 3%.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>122</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>I</td>
<td>63.17</td>
<td>63.14</td>
<td>64.92</td>
</tr>
<tr>
<td>$I * N$</td>
<td>7701</td>
<td>7949</td>
<td>7921</td>
</tr>
<tr>
<td>Schedule</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schedule</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>I</td>
<td>62.55</td>
<td>63.07</td>
</tr>
<tr>
<td>$I * N$</td>
<td>7694</td>
<td>7758</td>
</tr>
</tbody>
</table>

Now results from generation of the power prediction model will be presented. Specifically 168 benchmark DSP programs (each repeated several times in a loop) run with four different types of input data were executed on the DSP processor and the average current was read from the meter. Using the average current measurements (obtained from the single instruction tests) and the variables extracted directly from the DSP programs, along with variable $x_p$ were obtained. The variable $x_p$ together with the variables obtained directly from the DSP code (examples are given in table 2) were then used by the stepwise linear regression algorithm (see details of statistical procedure outlined in experimental section) in SPSS[5] to automatically form the equation for predicting current. The automatic power prediction model generation results will be presented here.
The first row provides a model applicable to the C5x processor with a excellent $R^2$ value and low percent error ($\%$err). More importantly it is very suitable for embedded systems design, since all inputs to the model can be obtained from the generated code itself along with a one time only model generation phase using single instruction and DSP benchmark tests together with statistical optimization. To further independently test out the validity or accuracy of our proposed model, variables from 84 other DSP programs (that were not used to derive the statistical model) were used in equation (1) to predict current. Results of this data were compared with actual current measurements. The predicted power or current value had an error less than 2% of the actual measured current for the different types of voice and pseudorandom data input.

![Figure 3 – Plot of predicted current (equation 1), versus actual current.](image)

For 168 cases (DSP benchmark programs), the stepwise selection procedure automatically produced the following model. The $x$ variables automatically chosen due to their significance by the statistical procedure are listed in order of their importance in predicting energy: LOAD, PABUS, SUB, IR, DABUS. The value of $R^2$ for this model is 0.89 or 89% of the variation in current is accounted for by these five variables. The equation for predicting current is

$$y = 35.99 + (29.74)\text{LOAD} + (5.88)\text{PABUS} + (14.27)\text{SUB} + (1.23)\text{IR} + (1.43)\text{DABUS}$$

(1)

The standard error of prediction for the largest residual is 0.1667. In other words we would be 95% confident that plus or minus 0.33 mA (= 2(0.1667)) of the predicted value of current would contain the average current. The confidence interpretation and tests of significance depend on the assumption of a normal distribution of residuals. The histogram of the residuals was normal thus verifying this assumption. The worst case residual was 1.29 mA, providing a maximum worst case error of 1.9% (calculated from the maximum residual divided by the predicted value of current for that case). The overall fit of the model can be seen in figure 3 where the predicted current, $y$ (from equation 1), is plotted against the actual measured current, $I$, both in mA's. The middle line graphs $x = y$, a perfect prediction line, with lines on each side 1 mA apart.

Table 4 below provides details of this statistically-derived model designed for the C5x. A comparison with other statistically-derived models derived using different amounts of detail can be found in [17].

<table>
<thead>
<tr>
<th>N</th>
<th>Proc</th>
<th>Model Variables</th>
<th>$%$err</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>168</td>
<td>C5x</td>
<td>LOAD, PABUS, SUB, IR, DABUS</td>
<td>1.91</td>
<td>.89</td>
</tr>
<tr>
<td>32</td>
<td>C62xx</td>
<td>mul2, readA, mul1</td>
<td>3.06</td>
<td>.91</td>
</tr>
</tbody>
</table>

Table 4. Power prediction models for C5x and C62xx.

The model in the last row of table 4 was obtained using the same statistical procedure, however it was applied to the more complex C62xx processor (using N=32 benchmark DSP codes). Power dissipation of this processor was typically 500 mA (an order of magnitude greater than the C5x processor). A more elaborate warmup and calibration procedure was required, however all results were repeatable. Power measurements ranged in value up to 14% for the different DSP codes. The statistical methodology provided a number of good models. The one presented above was a function of the average number of multiplies performed by the two multipliers in the datapath, mul1, mul2 and the average number of values read from register file A (readA) of the C62xx processor [16]. Specifically, the predicted power measured in Ampere is

$$y = 0.514 + (0.631)\text{mul2} + (0.0527)\text{readA} + (0.339)\text{mul1}$$

Although the TMS320C62xx DSP processor is a more complex processor than the C5x, a good model (with 3.06% worst case error) was obtained using the proposed statistical methodology as displayed in figure 1.

5. Discussions

In summary, current can vary significantly even when performance or $N$ is fixed (as is typical for many different embedded DSP codes where throughput is fixed).
For example results have shown that up to 4% changes in current (or power, energy) can be produced for different codes employing the same function, requiring the same number of clock cycles (N) and using the same type of input data. In embedded systems applications where N can vary (or even vary by a small percent) it has been shown that the fastest code does not necessarily produce the most energy efficient implementation. For example the code may have as much as 3% higher energy than other code implementations. Finally in cases where throughput is fixed or only minor variation in performance is allowed, accuracy in predicting current much greater than previously researched techniques (which achieved 10% accuracy using one DSP processor) is necessary. Once the current prediction model is generated from DSP benchmarks and single instruction tests, the model provides a fast technique for embedded systems designers to analyze different types of code for energy efficiency that is much faster than setting up each code to run in actual hardware and measuring current.

Although the statistically-derived model (see table 4) for predicting current presented in this paper is very accurate and was verified on independent DSP programs, one cannot infer causation. However our model does suggest some possibilities. This is evident from previous research which has found that instruction caches (including IR switching) contribute to energy dissipation of RISC processors [11]. A linear model was chosen originally for parsimony, however results indicate that it is applicable to power prediction. This research has also identified that accurate power models for software can be obtained from statistical methods using benchmark DSP programs. In cases where designers have good samples of data that their DSP processors will be using, one could use this data for input to the DSP benchmark and single instruction tests for generation of more accurate power prediction models. Results also indicate that prediction can be performed accurately without detailed switching information from processor internal registers and busses.

In contrast to previous research we have used statistical procedures that have verified a very accurate power model can be derived for DSP embedded software. Furthermore an independent assessment of the model, using data that did not derive the model, yielded excellent results. Although the results were presented using the C5x and C62xx DSP processors, the methodology is very general and can be applied to any processor (assuming appropriate variables can be extracted directly from the DSP code).

6. Conclusions

For the first time, physical current measurements on over 180 cases using two very different DSP processors, data types and DSP programs, have been used to show that power prediction model for software can be achieved and unlike previous research (with 10% error [2]) they are accurate (less than 4% error) and statistically-based. The statistical methodology (see figure 1) is general and can be applied to any processor. The model is derived from random data characterization of single instruction types and DSP benchmark programs. This is very useful for many embedded DSP systems where only access to the hardware and a functional simulator is available, specifically detailed simulation models (from which register or bus switching data can be obtained) are typically not possible.

In summary a fast accurate power prediction model for DSP embedded software was developed. Unlike previous research, our models with only a few variables, are very accurate; predicting current using statistical optimization with worst case errors ranging from 2% to 4% with $R^2$ values ranging from 0.89 to 0.91. High accuracy is necessary especially in embedded systems design where throughput constraints are employed and changes in current typically range from less than 10% to 14%. 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 which will continue to have stringent power, performance, and cost constraints. The authors would like to thank Alexander Bond for his excellent work. This research was supported with funding from NSERC.

References


