CLIPPER: Counter-based Low Impact Processor Power Estimation at Run-time

Jorgen Peddersen, Sri Parameswaran
School of Computer Science and Engineering, National ICT Australia
The University of New South Wales, Sydney, NSW 2052, Australia
{jorgenp,sridevan}@cse.unsw.edu.au

UNSW-CSE-TR-0618
September 2006

Abstract

Numerous dynamic power management techniques have been proposed which utilize the knowledge of processor power/energy consumption at run-time. So far, no efficient method to provide run-time power/energy data has been presented. Current measurement systems draw too much power to be used in small embedded designs and existing performance counters can not provide sufficient information for run-time optimization. This paper presents a novel methodology to solve the problem of run-time power optimization by designing a processor that estimates its own power/energy consumption. Estimation is performed by the addition of small counters that tally events which consume power. This methodology has been applied to an existing processor resulting in an average power error of 2% and energy estimation error of 1.5%. The system adds little impact to the design, with only a 4.9% increase in chip area and a 3% increase in average power consumption. A case study of an application that utilizes the processor showcases the benefits the methodology enables in dynamic power optimization.

1 Introduction

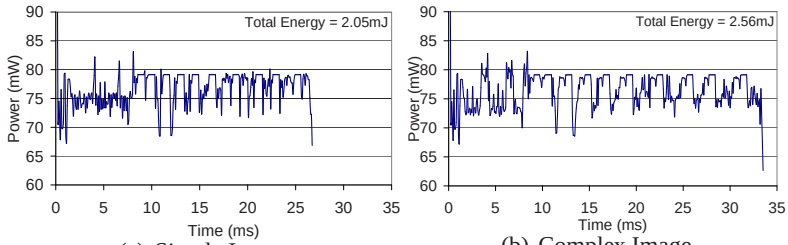
Power and energy consumption are becoming the most dominant bottlenecks constraining today's embedded systems. Reducing energy consumption, benefits the system by performing longer on limited battery supplies, reducing product weight by eliminating the need for heat sinks, and increasing reliability by dissipating less energy and running cooler. Batteries have a very limited amount of energy they can supply, and when power peaks over certain levels, the capacity of the battery drops more rapidly than usual[1]. Recently, many methods have been designed to manage power at all levels of design. These methods can be broadly split into static and dynamic methods.

Static power management involves predicting, simulating or profiling applications to record their performance and optimize power/ performance trade-offs to cater for the data set. All decisions are made before run-time, so worst case conditions are typically used to guarantee that constraints are met.

Dynamic power management for embedded systems allows tuning of hardware/software parameters to perform trade-offs between power consumption and performance of applications during execution. Dynamic techniques can make use of operational parameters of the system at run-time and make decisions that alter the future operation of the system.

However, dynamic power management is stymied by the lack of methods to efficiently obtain run-time measurements of power or energy consumption. Several techniques for dynamic power management have been proposed that require knowledge of how much energy has actually been consumed by the device to make run-time decisions[2, 3, 4]. Providing these values in a stand-alone system is difficult as measurement systems for power and energy usage draw too much power themselves, making potential savings useless.

Data dependent applications, e.g. multimedia, have no concept of the type of data they will be operating on until the data arrives at run-time. The type of data greatly affects the execution paths causing vastly different power and energy consumption. It is impossible to design static power algorithms to manage this class of applications due to this unpredictable behavior. Due to data dependence, dynamic power optimization methods are required to analyze recent operation and make decisions to manage future operation of the system. Other examples of data dependence include RF systems, where transmit and receive power changes for differing situations; and packet based unreliable communication systems, where selected packets can be dropped if power consumption becomes critical.



(a) Simple Image (b) Complex Image

Figure 1: JPEG Power Comparison

Motivation

The motivation for this work stems from two disparate, yet synergistic observations. The first observation is that there is significant correlation between the function a processor is performing and its power consumption. The second is that many applications operate quite differently depending on their data input.

The first observation leads to event based macro modeling[5], which allows us to count events in the system that contribute power to gain an adequate representation of power consumption. Over time, we only need to look at control signals to find these events; due to the randomness of data, the data signals tend to only provide random noise in the power trace. When combined with regression based modeling techniques[6], we can generate an accurate model to estimate power consumption of a device from its control signals.

The claims of the second observation are demonstrated by Figure 1. This shows the power waveform for two differing images encoded by a JPEG application. The execution time and average power for the two images differ, and energy use varies by 20%.

If the data dependent algorithms of the second observation could gain knowledge of their run-time power consumption, they could make trade-offs to optimize quality of service under power constraints. The first observation provides that knowledge via the addition of on-chip estimation of power using macro modeling. This estimation technique opens the door to a new breed of enhanced dynamic power management techniques for stand-alone systems with power constraints.

This work, for the first time, presents a methodology to modify a processor so that it can estimate its own power consumption at run-time by utilizing small counters attached to the control path of the design. We call this methodology CLIPPER, or Counter-based Low Impact Processor Power Estimation at Run-time. The counters have little impact upon the power consumption and area of the system. CLIPPER is a generic methodology which can be applied to a wide range of

processors, enabling and enhancing the use of many dynamic power optimization techniques. Application adaptation techniques greatly benefit from knowledge of energy consumption to handle run-time energy constraints (see Section 6). Additionally, hardware techniques such as Dynamic Voltage Scaling (DVS) can use power data to perform time/power trade-offs and manage heat consumption, although these DVS techniques are outside the scope of this paper.

Limitations of Performance Counters

Many modern high performance processors already have counters included in the processor to count system events for statistics related purposes[7, 8]. These counters often overlap with the events needed to model the power effectively at run-time. Although there are many events that can be detected and counted by these systems, there is a limitation to the number of simultaneous events that can be counted in a single execution[7]. Run-time power calculation is performed in multiple execution passes with different counters being used each time. Hence these processor types *cannot* be used to provide the run-time feedback of power consumption which motivates this paper. Additionally, performance counters are only included on high performance processors, so they may not be suitable for smaller embedded systems. The methodology described in this paper is applicable to any type or size of processor, and attempts to minimize the additional hardware required while allowing run-time feedback to the application so power optimization decisions can be made.

Organization

The rest of the paper is organized as follows. We review others' works related to this topic in Section 2. Section 3 discusses the CLIPPER methodology while Section 4 demonstrates an application of CLIPPER to a processor based on the SimpleScalar architecture. Results of experiments performed on this processor are given in Section 5. A case study of an application which CLIPPER enables is provided in Section 6, then Section 7 draws conclusions.

2 Related Work

Reduction of power/energy consumption of embedded systems has been a major field of research for the past few decades[9], covering both software and hardware approaches. Some of the proposed hardware techniques include Dynamic Voltage Scaling[9, 10] and Adaptive Body Biasing[11, 12, 13]. Proposed software methods

include Power-Aware Task Scheduling[14, 15], Power Macro Modeling[16] and Application Adaptation[2].

Powerscope[17] is a system designed to measure run-time power consumption and is used as a tool for power-aware computing. It was designed to be used with the tool Odyssey[2] to allow run-time application adaptation trade-offs to manage power consumption of a run-time system. However, this measurement procedure is power hungry and is not feasible for stand-alone embedded systems.

Many power estimation techniques have been proposed at all levels of design[16, 18, 19]. Macro modeling has emerged as one of the recurring themes of these works, as it provides simplified models for each of the devices in the system to estimate their power consumption. This is typically applied at the RTL, but may also be applied at other levels of design. Macro modeling can also be applied through instruction level analysis techniques[20, 21]. These techniques use software simulators to assign power levels to each type of instruction executed in the system. Bellosa showed in [5] that event based power macro modeling is an effective method for producing power models for various components of processors.

Wattch[22] is a tool that macro models the SimpleScalar simulator[23]. This system calculates power for a theoretical implementation of the SimpleScalar processor using power models of components from typical superscalar processors. The tool is useful for analyzing power and can be used to test dynamic power management models, however it is only a simulation tool and has no underlying hardware model to provide a true self estimation system.

Contreras and Martonosi, in [24], use existing Hardware Performance Counters (HPCs) in the Intel PXA255 processor to count five events in the system to calculate average power consumption with an error of 4%. However, the authors point out that the processor can only count two event types during any single execution, so multiple passes are needed to achieve the stated accuracy.

In [25], Haid et al. presents JouleDoc (an audio processing system), the only previous attempt we can find to estimate and feedback power/energy usage of a stand-alone embedded system at run-time. JouleDoc is a co-processor which utilizes macro modeling to estimate power of its single application. JouleDoc uses eight counters to estimate energy consumption with an error of 5%. The JouleDoc co-processor adds approximately 12,000 gates to the area of the system.

In contrast to [24] which requires existing performance counters to estimate power, CLIPPER shows how and where to insert counters into *any processor system* to estimate power accurately with minimal impact. In addition, our system allows run-time feedback of power/energy consumption rather than needing to execute the algorithm multiple times. In comparison to [25], our method allows for general systems to be characterized, as no method to apply the JouleDoc solution to systems apart from the selected application specific system is provided. Despite

its generality, CLIPPER also exhibits smaller impact upon the system due to lack of a co-processor and the use of smaller fixed increment counters.

Thus, the **contributions** of our paper are as follows:

- For the first time, we present a methodology to add dedicated fixed increment counters to a processor system so that it can estimate its own power/energy consumption at run-time.
- A method is proposed to show how and where counters should be placed to provide desired accuracy with minimal impact.
- We demonstrate via a benchmark application how feedback provided by run-time power estimation can be used to dynamically optimize performance under energy constraints.

3 Methodology

The CLIPPER methodology makes use of the techniques of *system decomposition theory*[26] and *regression analysis*[27]. This allows us to decompose the system into components that are analyzed separately to determine how their power consumption is related to events that occur in the component. The embedded system can hence be modeled by counting the events that occur and calculating the power. The mechanisms to find these events, compute the relationship of the events to power and provide run-time feedback of the power consumption to applications at run-time is detailed in this section.

3.1 Theory

System decomposition theory originated from the ontological model of an information system decomposition. The following basic definitions and theorems are obtained from [26], though a far more detailed description is given in the same paper.

- A system, σ, comprises a set of events.
- An event, c, is a boolean value representing the present state of one or more control signals.
- An event space, S, is a multi-dimensional hypercube, where each dimension corresponds to an event and each point is a state of the system

- A function, f, over a parameter space, S, is a function that corresponds to a power model of the component.
- A system $\acute{\sigma}$ is a subsystem of σ, and σ is a supersystem of $\acute{\sigma}$ if and only if the composition of $\acute{\sigma}$ is a subset of the composition of σ.
- A decomposition of a system σ is a set of subsystems D(σ) = $\{\sigma_i\}_{i \in I}$, such that each event in the system is included in at least one of the subsystems.

Let D(σ) = $\{\sigma_i\}_{i \in I}$ be a decomposition σ. The event space S(D) of the decomposition is:

$$S(D) \equiv S(D(\sigma)) = \otimes_{i \in I} S(\sigma_i)$$

Let c_i be an event in an ordered and finite set S_i. C is an n tuple of events, $C = \{c_1, c_2, ..., c_n\}$. Let C_1 be an x tuple of events $\{c_1, c_2, ..., c_x\}$ and C_2 be an $n - x$ tuple of events $\{c_{x+1}, c_{x+2}, ..., c_n\}$ such that n tuple C can be expressed as $\{C_1, C_2\}$. In addition, a function, $f(C_1, C_2)$, is independent of C_2 if $f(C_1, C_2) = f(C_1)$ which can be completely represented by events in C_1.

There are three requirements to ensure a valid system decomposition: i) the system must have a well-defined structure; ii) the system must only be represented by a known set of the events; iii) a change in an event, that belongs to a subsystem, must result in a significant change on the function of the subsystem. The reasons that system decomposition theory is applicable to modeling power of embedded systems : i) a system can be split into components that contribute power (e.g. dividers, caches, ALUs); and, ii) we can determine events in components that affect operation of the component or other components. These events typically represent the state of operation of the component (e.g. a divider in use, a cache miss, a cache hit etc.) and the more important events can be used to calculate power consumption of the system.

Regression analysis is an analysis method that expresses a model as a function of parameters. Each of the events in the system is modeled as consuming a particular energy amount. For example, a model of a system, M(σ), is expressed as a linear function of e_1, e_2, ..., e_n, where each e_i is a detected event of the system and can be represented as follows:

$$M(\sigma) = m_0 + m_1 e_1 + m_2 e_2 + ... + m_n e_n$$

where m_1, ... ,m_n are coefficients of the events e_1, ... ,e_n and m_0 is a base value. The function can take other forms of expression, such as quadratic or polynomial etc. The coefficients of the parameters, and the relationship of the model (i.e. linear, quadratic, polynomial etc) can be determined (if such a relationship exists) by commercial tools, when a sample dataset and parameters are given.

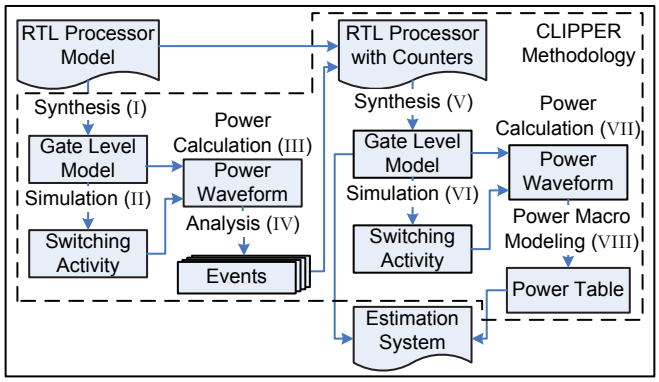


Figure 2: CLIPPER Methodology

3.2 CLIPPER

The CLIPPER methodology is outlined in further detail in Figure 2. The input is a synthesizable RTL description of a processor. Step I of CLIPPER synthesizes the processor to produce a gate-level model.

Step II simulates the gate level model of the processor to record its cycle-by-cycle switching activity. An application that tests many operating conditions of the processor is used when simulating to determine all possible power conditions. This application includes all of the instruction set, memory operations, I/O type instructions, different types of loops etc. and also makes good use of different states of the cache or caches utilized by the system.

In Step III, the switching activity is used by a power analysis tool such as Synopsys Primepower[28] to determine the power consumption of the processor. This power calculation provides a waveform for power consumption which is used to analyze the processor to determine which events in the system are responsible for most of the power consumption so a macro model can be made.

Event Analysis

Step IV of the CLIPPER methodology, expanded in Figure 3, discovers events used to create a macro model of the system. We first utilize the system decomposition theory to split the processor into it's modules for separate analysis. The power of each of the separate components can be analyzed to determine the events that contribute to its power. As the power of the system will be the sum of the power of the components, only those components that demonstrate large variation in their power consumption during run-time are required to be analyzed. Those models

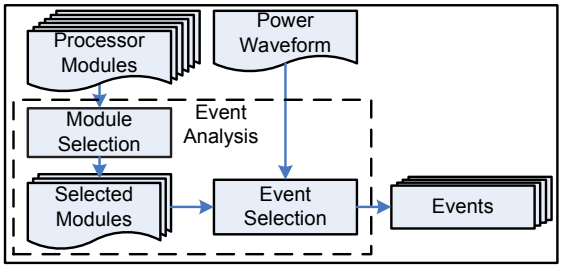


Figure 3: Event Analysis Flow

with minor variation can effectively be modeled with a constant value for power consumption. Module selection is performed by ranking all modules in the design by their power variation (max. power - min. power) and selecting the modules with the highest variation.

Changes in the power consumption of a system are often detectable by the control signals of the design. Therefore, the values or transitions of one or more control signals can be tested to detect these changes. The combination of these control signals is called an event, with each control signal being a condition for that event to occur.

Algorithm 1 is used to find events within modules. This algorithm creates a set of events, $events = \{e_1, ..., e_n\}$, for each module to characterize the system. An event e_i is represented as a set of control signals $\{c_1, ..., c_k\}$ which form the conditions of the event. The algorithm first discovers events and then simplifies them to reduce complexity.

Step 1 of the algorithm analyzes each power change in the system and creates events to attribute to the change. Control signals that change at the time of the change in power are correlated with power. Correlation is performed by looking at the power waveform whenever the control signal has the same value or transition of value. If power often has a similar change or similar level as it does at the current time, it is likely that the control signal contributes to the change and can be added to the list of control signals defining an event.

Step 2 attempts to simplify many of the events by reducing the number of controlling signals. This is due to groups of signals that switch simultaneously, e.g. if one signal is the logical NOT of another, or the logical AND of two others. If there exists a signal that switches at least whenever a second signal switches, the second signal is superfluous and is removed from the event.

Step 3 removes events that are duplicated due to the same changes in the system occurring. Events are then output to be added to the system.

Events selected by the algorithm are converted to logic circuits for detection.

Algorithm 1 Find Events Causing Power Consumption

```
for all m in selected modules do
  // Step 1: Find events for each change in power
  Order power changes of module m into P_1...P_n
  Where P_1 is largest and P_n is smallest viable size
  Let events = {e_1, ..., e_n} with all e_i empty
  for all P_i in P_1...P_n do
    Let c be a set containing all control signals that switch near time of P_i
    for all c_j in c do
      // Test if the control signals often contribute power
      if c_j correlates with power waveform then
        Add c_j to e_i
  // Step 2: Remove superfluous control signals from events
  for all e_i in events do
    for all c_j, c_k in e_i do
      if c_j occurs whenever c_k occurs in the simulation trace then
        Remove c_k from e_i
  // Step 3: Remove duplicate events
  for all e_i, e_j in events do
    if e_i contains same signals as e_j then
      Delete e_j
  // Output each event that remains
  for all e in events do
    print Control signals of e
```

Each event is formed by taking the AND of its conditions. Events that occur due to transitions of signals also require additional flip-flops to detect the change.

After event selection, a linear regression is performed to reduce the number of events in the system. Any events found to contribute little to the power in a regression are removed from the system. This helps to counter some of the redundancy of the provided algorithm.

Power contributions of external devices are also modeled. These devices include off-board memory, additional analog circuitry or user interfaces. Off-chip peripheral components are analyzed using the above method to provide additional events into the power estimation system. If power variance is small compared to average power of the system, the component is modeled as maintaining a constant average value for power consumption.

3.2.1 Counters

Counters are added to the design to tally occurrences of events. These counters are fixed increment counters to reduce their impact upon the area and power consump-

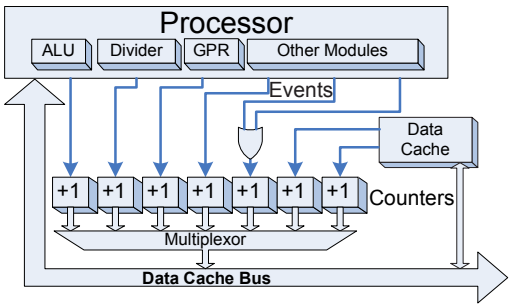


Figure 4: Counter Setup Example

tion of the system. Specific trigger events increment the counters. Additionally, a counter is automatically reset by hardware whenever the counter is read, so that power consumption will be valid between accesses of the counters.

Counters are interfaced to the processor so that they do not severely impact upon power consumption when read. Counters are interfaced via memory mapped I/O in a reserved memory space avoiding the data cache. Figure 4 depicts this method. An address decoding multiplexor is used to interface the counters to the data cache bus. Control signals are taken from various modules and include combinational logic to decode events.

For the system implementation in this paper we used equal sized 32-bit counters. This is the data bus width of the processor which allows the most amount of time to pass between reads. Despite their size, impact of these counters is still small, so we didn't further reduce the width of the counters.

3.2.2 Macro Modeling

The counters are added to the original RTL processor model. The newly created model, located at the top right of Figure 2, is passed through the same synthesis, simulation and power calculation flow as before to model the added impact of the counters to the system (Steps V, VI and VII of Figure 2).

Note that as the counters are small compared to the processor, the impact is not large enough to change the modules or events that were chosen by the earlier steps. The contributions of the events and idle power will increase due to the addition of the counters, but not by a significant amount (see Section 5).

Regression analysis is applied to the power calculations to find actual contributions to power of the specific events in Step VIII of Figure 2. Power is linearly related to the events in the system. Each event is associated with a power calculation which represents the contribution to power consumption each time the event

occurs. i.e. for every event e_i, there is an average additional power p_i which is caused by the event. In addition, a base power p_0 is provided by the regression to represent static power consumption and average power consumption of non-modeled components. These results are tabulated into an array of power values that will be utilized by the software loop to calculate the estimated run-time energy and power consumption.

3.2.3 Software

A software loop calculates total energy between counter readings by employing the following equation:

$$E_{tot} = \frac{CC \times p_{base} + \sum_{i=1}^{n} e_i \times p_i}{f}$$

where n is the number of events, f is the current clock frequency of the processor, CC is the number of clock cycles from the last energy reading, p_{base} is the base power used every cycle (due to static leakage power and average dynamic power of unselected modules), e_i is the counter value for event i and p_i is the power contribution of that event.

Average power for the period between reading of the counters is found by:

$$P_{ave} = \frac{E_{tot} \times f}{CC} = p_{base} + \sum_{i=1}^{n} \frac{e_i \times p_i}{CC}$$

This calculation is performed at recurring locations in an algorithm's task flow. For example, after each frame of a video application or after each line of processing in an image encoding application. This minimizes software impact.

Where the system has multiple states, such as if DVS were applied, a multi-dimensional version of the power array (one row per state) can be used. Analysis of this is outside of the scope of this paper.

4 Experimental Setup

We tested the CLIPPER methodology by applying it to a publicly available processor described in [29]. This processor is defined in RTL VHDL and implements all integer instructions of the SimpleScalar[23] PISA instruction set as an in-order, six-stage pipeline processor. Synopsys Design Compiler[28] was used to synthesize the processor in Steps I and V of Figure 2. Synthesis was performed with the 180nm TOWER library available from Synopsys with a clock frequency of 125MHz.

ModelSim[30] was used in Steps II and VI to perform gate level simulation of the synthesized processor. The processor was connected to simulated memories and an application was executed so that internal signal changes are stored in a Value Change Dump (VCD) file. Periodic readings of power counters were also taken during ModelSim simulation to allow us to determine estimation accuracy.

To perform Steps III and VII, the VCD produced by ModelSim was analyzed by Synopsys Primepower[28] to model power. By analyzing the design and switching activity from the VCD, Primepower produces an accurate prediction of how much power the synthesized chip would actually consume per clock cycle.

The module selection stage resulted in five components being chosen for power consumption. The chosen modules were the divider, multiplier, register file, instruction cache and data cache. Event analysis found nine events in the system that were used to model the selected modules. These events included cache memory being read or written and the divider or multiplier being utilized. An additional event was added to measure time.

The CLIPPER methodology allows the addition of events to model off-chip circuitry such as external memory. We have a DRAM power model derived from data sheets that can be modeled exactly by adding two events to the design. However, we have no other tools to more accurately estimate or measure power consumption of external memory chips. Therefore, we chose to ignore memory power contributions in our accuracy analysis as the actual model is equivalent to the estimation model, providing no benefit to proving the validity of the approach. Incidentally, the average power of the DRAMs is about equal to the average power of the processor.

Control of most counters was determined by an existing control signal, while other events required nominal logic gates or flip-flops to be added to test multiple signals. The counters were interfaced by memory mapped I/O, bypassing data cache as shown in Figure 4.

The modified version of the processor's HDL code with the added power counters will be made available to download at http://www.cse.unsw.edu.au/∼esl.

5 Results

This section lists some of the tests that were performed to discover the impact upon the experimental system and test the accuracy of the prediction mechanism.

Power contributions for each of the selected events were calculated using Step VIII of the CLIPPER flow (see Figure 2). The divider caused the greatest impact upon power, doubling the power consumption of the chip for three cycles each time it was used. The divider, multiplier and instruction cache each needed only one

Table 1: Metrics With and Without Counters

	Original	Modified	% Increase
Gate Area (NANDs)	127994	134238	4.9%
Total Power (mW)	77.19	79.48	3.0%

```
/* energy_arr contains the energy contributions   *
 * of each of the counters.  POW_NUM is the       *
 * number of counters including time count        */
unsigned int energy_arr[POW_NUM] = {19,31,10,...};

int calc_energy(void) {
  unsigned int i, energy = 0;
  for (i = 0; i < POW_NUM; i++) {
    /* Counters are mapped to the memory in int   *
     * sizes from address 0 to POW_NUM-1          */
    energy += energy_arr[i] * ( *((int *)(i * 4)) );
  }
  return energy;
}
```

Figure 5: Sample Energy Calculation Loop

event to accurately model their power contribution. The register file contributed to power when there was a register write or the registers selected for reading were changed between instructions. The data cache provides the remaining four events (miss, write-back, read hit and write hit).

Impact

Table 1 compares the area and power metrics of the original processor and the processor with added power counters. The first column names the metric. The second column gives values for the original processor, while the third gives the values for the modified processor with additional counters. The final column shows percentage increase of the metrics. Area values include interconnect area and were calculated after synthesis of the processor with Synopsys Design Compiler and were converted to the equivalent number of NAND gates. Total average power was provided after simulation of some test applications in Synopsys Primepower. The impact upon area of the processor is only 4.9% and average total power only increases by 3%. Note that our ten counter design of 6,224 gates uses about half the gates of JouleDoc's eight counter, 12,000 gate system.

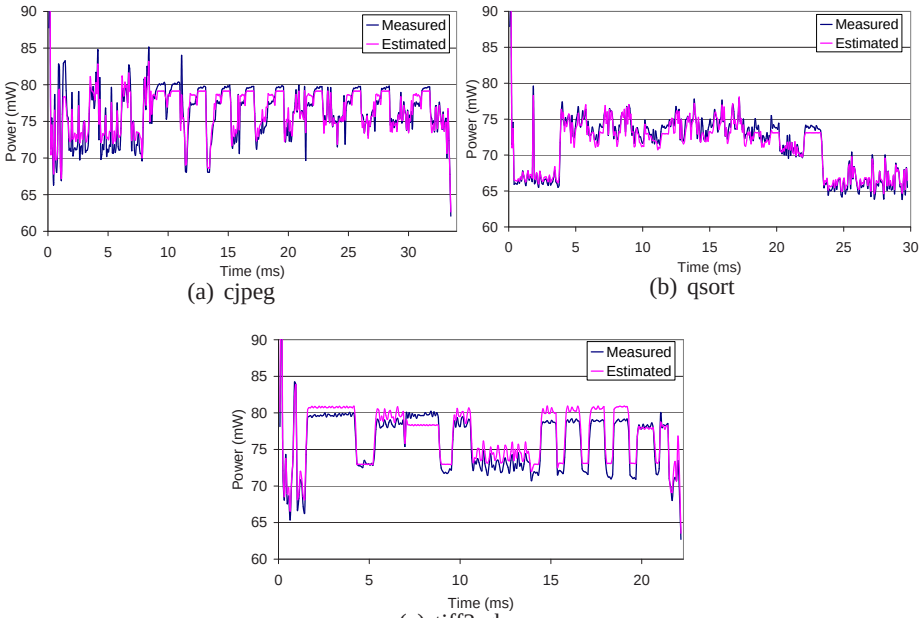
The function shown in Figure 5 was simulated in ModelSim to determine additional software impact. Note that this loop also accounts for energy used during its own execution; as order is preserved, any energy not seen in the current iteration will be seen in the next iteration. When applied to the SimpleScalar processor, the compiled loop executes 95 instructions taking 232 clock cycles with pipeline stalls and cache misses taken into account. Small applications require execution of the loop at most every 100,000 cycles representing only 0.23% of the execution time. Note that we do not wish to add hardware to the design to aid this calculation (such as a dedicated multiplier) as the infrequency of execution does not justify the persistent area and power costs of additional hardware.

Accuracy

Figure 6 shows power estimation results for cjpeg(6(a)), qsort(6(b)) and tiff2rgba(6(c)) from the MiBench testbench suite[31]. Each graph shows both measured and estimated values of power calculated by Synopsys Primepower and the processor power counters respectively. Power was sampled at a period of 10,000 clock cycles. The closeness of the waveforms in each graph demonstrates high accuracy.

Table 2 summarizes the estimation results for several testbench applications. The applications were cjpeg, qsort, tiff2bw and tiff2rgba from MiBench[31]; and rawcaudio, rawdaudio, g721e and g721d from Mediabench[32]. Column 1 states the name of the benchmark being executed while Column 2 lists the number of clock cycles taken to execute the benchmark. Columns 3 and 4 show the accuracy of power calculations as an average for all samples as well as the maximum per sample error respectively (sample period is 10,000 cycles). Columns 5 and 6 show the measured and estimated energy consumed over the run-time of the application. The final column shows error in energy estimation, with the sign indicating whether energy was over- or under-estimated. The table shows that the average absolute error for power calculation is less than 2%. The maximum sample power error is larger (e.g. 8% for rawcaudio), but this is only at a few points throughout each trace. Energy estimation has a total error less than 1.5% over all applications. These results show we are more accurate than JouleDoc which has an error of 5%, despite our generic approach.

In Figure 7 we show how choosing the number of modules and events that are selected affects the accuracy of the estimation procedure. We removed less significant events from the event list and created new macro models with the remaining counters. Figures 7(a), 7(b) and 7(c) show accuracy graphs utilizing 2, 5 and 8 counters respectively for the qsort benchmark. This demonstrates how adding more counters to the processor increases accuracy of the estimation. Note that Figure 7(c) bears resemblance to Figure 6(b), showing that qsort could have



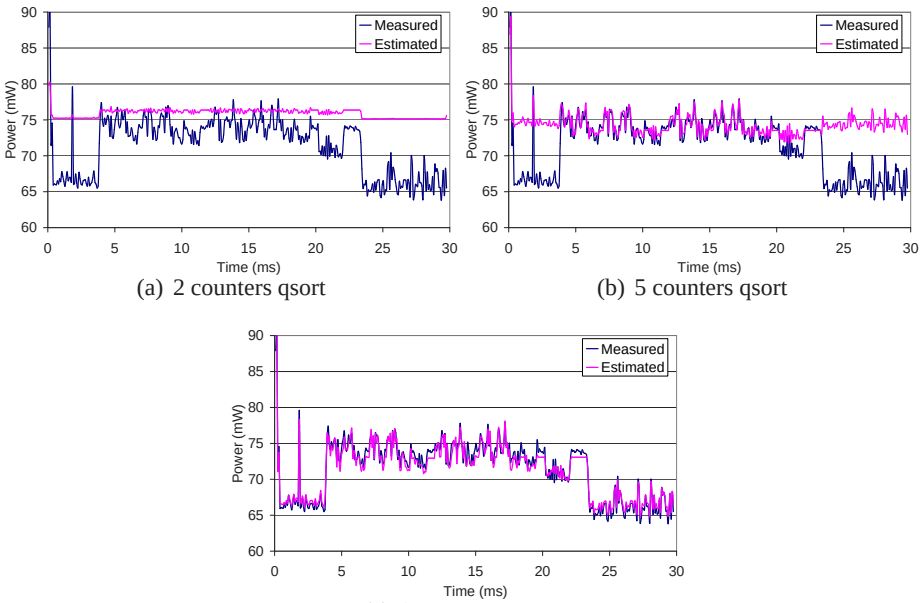
(a) cjpeg (b) qsort

(c) tiff2rgba

Figure 6: Measured vs. Estimated Power

Table 2: Power and Energy Errors

Bench-mark	Cycles (mils.)	Power Error		Energy (mJ)		Energy Error
		Avg	Max	Measured	Estimated	
g721e	2.9	0.77%	4.97%	1.566	1.577	0.74%
g721d	2.6	0.48%	5.05%	1.441	1.448	0.47%
cjpeg	4.2	1.27%	6.86%	2.559	2.564	0.21%
qsort	3.7	0.85%	4.80%	2.124	2.118	-0.24%
rawcaudio	3.2	1.45%	8.04%	1.988	1.961	-1.35%
rawdaudio	2.9	0.58%	2.30%	1.793	1.793	-0.03%
tiff2bw	3.1	0.89%	6.00%	1.924	1.925	0.03%
tiff2rgba	2.8	1.58%	6.24%	1.685	1.705	1.22%



(a) 2 counters qsort (b) 5 counters qsort

(c) 8 counters qsort

Figure 7: Results with Various Numbers of Counters

utilized fewer counters to achieve similar results, although this trend is not seen across all benchmarks.

6 Case Study

To demonstrate how a CLIPPER processor can be used for run-time power optimization, an application was modified to utilize the power estimation of the SimpleScalar processor described above. The JPEG benchmark was converted into a motion JPEG encoder. This application was then modified to make use of the processor to optimize it's quality under a given energy constraint. A stream of images from the flower garden and table tennis sequences[33] were encoded with energy calculated after each image. The modified application is able to encode images at various quality levels, for which approximate ratios of energy consumption were calculated. Twelve quality levels were created by choosing between DCT routines and varying compression rate. An additional level (level 0) was created that drops color information and encodes images in black and white to save computation time and energy.

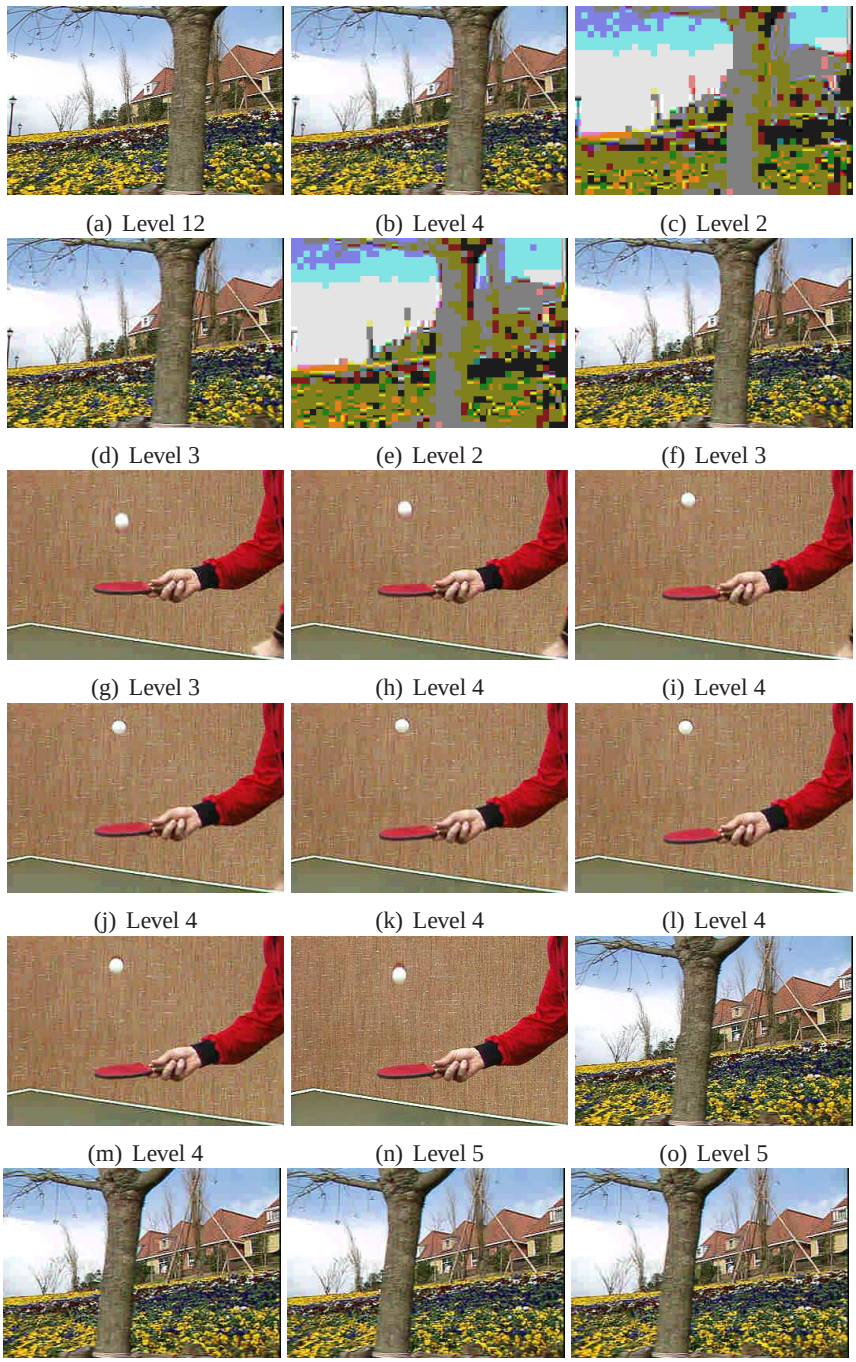
Algorithm 2 is a dynamic algorithm used to determine which quality level will

be used for future images. It assumes that future images will be similar to the current image when making decisions. The number of images that remain to be encoded is N, R is the remaining energy to encode those images, L is the current quality level with maximum MAX, and $levels$ is an array storing the relative energy consumption of each quality level. The values of this array were pre-computed by executing the algorithm on a range of images at the various quality levels and comparing the average energy consumption. Within an iteration of the main loop, E represents the energy of the most recently encoded image and l is used to calculate the quality level of the next image.

Algorithm 2 Application Decision Algorithm

$N \leftarrow$ Number of images to encode
$R \leftarrow$ Total available energy
$L \leftarrow MAX$
while $N > 0$ **do**
  Encode image at level L
  $E \leftarrow calc\_energy()$; // Get energy of encoded image
  $N \leftarrow N - 1$;
  $R \leftarrow R - E$;
  if $E > \frac{R}{N}$ **then**
    // Energy is too high, reduce level until it meets constraint
    if $L \neq 0$ **then**
      $l \leftarrow L - 1$; // Start form next level down
      while $l > 0$ **and** $E \times \frac{levels[l]}{levels[L]} > \frac{R}{N}$ **do**
        $l \leftarrow l - 1$;
  else
    // Energy is OK, increase level if it will stay under constraint
    while $l < MAX$ **and** $E \times \frac{levels[l+1]}{levels[L]} < \frac{R}{N}$ **do**
      $l \leftarrow l + 1$;
  $L \leftarrow l$; // Set level for further images

Figure 8 shows the output of a sample execution of the motion JPEG algorithm described above. Figure 9 shows both the energy usage of each image in the stream and the available energy per image for future images. Available energy per image is calculated by dividing total energy remaining by the number of images left to encode (R/N). The algorithm can encode images at quality levels from 0 to 12 with lower levels being very lossy to demonstrate extreme conditions. The first image is computed at maximum quality (level 12), and further images use Algorithm 2 to change the quality level. Figure 9 demonstrates how the algorithm finds the correct power level at which to encode images to maximize quality while attempting to stay under the available energy consumption level. Also note the garden images take more energy to encode than the table tennis images at equivalent levels (see



(a) Level 12 (b) Level 4 (c) Level 2

(d) Level 3 (e) Level 2 (f) Level 3

(g) Level 3 (h) Level 4 (i) Level 4

(j) Level 4 (k) Level 4 (l) Level 4

(m) Level 4 (n) Level 5 (o) Level 5

(p) Level 4 (q) Level 4 (r) Level 4

Figure 8: Output of Sample Application with Image Quality Levels

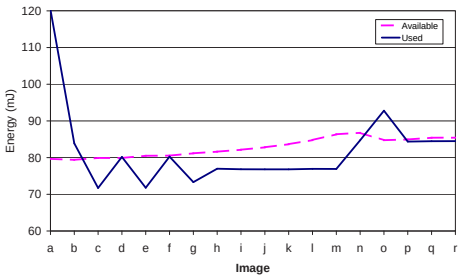


Figure 9: Energy Usage with Constraints

transitions from (f)-(g) and (n)-(o)). The algorithm automatically compensates for these different image types, altering the level for future images when it detects the change in energy. This case study demonstrates how CLIPPER enables new power optimization techniques.

7 Conclusion

This paper presented CLIPPER, a novel methodology to modify a processor design so that it can estimate its own power and energy consumption. Small counters added to the design allow event driven macro modeling to be economically performed in parallel with application execution. The discussed methodology can handle estimation of power consumption of external devices to the processor, and supports hardware power/energy reduction techniques unlike other run-time solutions. This is the first time a feasible, wide-ranging solution to the problem of run-time power estimation for dynamic power management of stand-alone systems has been proposed. Unlike performance counter methods, CLIPPER can be used to provide run-time feedback of data to achieve power optimizations.

We have also demonstrated an example of how a CLIPPER processor can be used with an application that dynamically modifies its QoS to meet energy constraints at run-time, a technique that is impossible without power estimation or prior knowledge of the input data.

The modified SimpleScalar processor produced by the CLIPPER methodology will be made available to download at http://www.cse.unsw.edu.au/∼esl so that further research can be performed.

References

[1] M Pedram and Q Wu. Design Considerations for Battery-powered Electronics. In *Proceedings of DAC*, 1999.

[2] J Flinn and M Satyanarayanan. Energy-Aware Adaptation for Mobile Applications. In *Proceedings of the ACM SOSP*, 1999.

[3] R McGowen. Adaptive Designs for Power and Thermal Optimization. In *Proceedings of the International Conference on Computer-Aided Design*, 2005.

[4] V Raghunathan, C Pereira, M Srivastava, and R Gupta. Energy-Aware Wireless Systems with Adaptive Power-Fidelity Tradeoffs. In *IEEE Transactions on Very Large Scale Integration Systems*, volume 13, pages 211–225. Feb 2005.

[5] F Bellosa. The Case for Event Driven Energy Accounting. Technical Report TR-I4-01-07, University of Erlangen, 2001.

[6] A Bogliolo, L Benini, and G De Micheli. Regression-Based RTL Power Modeling. In *ACM Transactions on Design Automation of Electronic Systems*, volume 18, pages 813–833. 1999.

[7] R Joseph and M Martonosi. Run-time Power Estimation in High Performance Microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2001.

[8] K Lee and K Skadron. Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors. In *Proceedings of the PDPS*, 2005.

[9] J Rabaey and M Pedram, editors. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.

[10] A Andrei, M Schmitz, P Eles, Z Peng, and B Al-Hashimi. Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints. In *Proceedings of DATE*, 2005.

[11] Y Le, J Luo, and N Jha. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-Time Embedded Systems. In *Proceedings of the International Conference on Computer-Aided Design*, 2003.

[12] Steven Martin, Krisztian Flautner, Trevor Mudge, and David Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power

Microprocessors Under Dynamic Workloads. In *Proceedings of the International Conference on Computer-Aided Design*, 2002.

[13] J Tschanz, S Narendra, R Nair, and V De. Effectiveness of Adaptive Supply Voltage and Body Bias for Reducing Impact of Parameter Variations in Low Power and High Performance Microprocessors. In *IEEE Journal of Solid-State Circuits*, volume 38. 2003.

[14] H Aydin, R Melhem, D Mosse, and P Mejia-Alvarez. Power-Aware Scheduling for Periodic Real-Time Tasks. In *IEEE Transcations on Computers*, volume 53, pages 584–600. May 2004.

[15] J Wang, B Ravindran, and T Martin. A Power-Aware, Best-Effort Real-Time Task Scheduling Algorithm. In *IEEE Workshop on Software Technologies for Future Embedded Systems*, 2003.

[16] A Raghunathan, S Dey, and N Jha. High-Level Macro-Modeling and Estimation Techniques for Switching Activity and Power Consumption. In *IEEE Transactions on Very Large Scale Integration Systems*, volume 11. Aug 2003.

[17] J Flinn and M Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[18] A Bergamaschi and Y Jiang. State-Based Power Analysis for Systems-on-Chip. In *Proceedings of the Design Automation Conference*, 2003.

[19] T Givargis, F Vahid, and J Henkel. A Hybrid Approach for Core-Based System-Level Power Modeling. In *Proceedings of the ASP-DAC*, 2000.

[20] S Nikolaidis and T Laopoulos. Instruction-Level Power Consumption Estimation Embedded Processors Low-Power Applications. In *International Workshop on Intelligent Data Acquisition and Advanced Computing Systems*, 2001.

[21] V Tiwari, S Malik, A Wolfe, and M Lee. Instruction Level Power Analysis and Optimization of Software. In *Proceedings of the International Conference on VLSI Design*, 1996.

[22] D Brooks, V Tiwari, and M Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the International Symposium on Computer Architecture*, 2000.

[23] D Berger and T Austin. *SimpleScalar Tool Set, Version 2.0*. http://www.simplescalar.com/docs/users_guide_v2.pdf, 1997.

[24] G Contreras and M Martonosi. Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005.

[25] J Haid, G Kaefer, C Steger, and R Weiss. Run-Time Energy Estimation in System-On-a-Chip Designs. In *Proceedings of the ASP-DAC*, 2003.

[26] Y Wand and R Weber. An ontological model of an information system. In *IEEE Transactions of Software Engineering*. 1990.

[27] Splus. http://www.insightful.com.

[28] Synopsys Tool Set. http://www.synopsys.com.

[29] J Peddersen, SL Shee, A Janapsatya, and S Parameswaran. Rapid Embedded Hardware/Software System Generation. In *Proceedings of the International Conference on VLSI Design*, 2005.

[30] ModelSim Simulator. http://www.model.com.

[31] M Guthaus, J Ringenburg, D Ernst, T Austin, T Mudge, and R Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *IEEE Annual Workshop on Workload Characterization*, 2001.

[32] C Lee, M Potkonjak, and W Mangione-Smith. MediaBench: a Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Proceedings of the ACM/IEEE International Symposium on Microarchitecture*, 1997.

[33] SAMPL Motion Imagery Database. http://sampl.ece.ohio-state.edu/data/motion.