SCHEDULING FOR POWER REDUCTION IN A REAL-TIME SYSTEM

Jason J. Brown[1] *Danny Z. Chen*[2] *Garrison W. Greenwood*[3]
Xiaobo (Sharon) Hu[2] *Richard W. Taylor*[1]

ABSTRACT

This paper describes how, through a combination of scheduling and buffer insertion, real-time systems may be optimized for power consumption while maintaining deadlines. Beginning with simple examples (components that have no internal pipelines and in which the only design freedoms are buffer insertion and scheduling), we illustrate the effect of adjusting the time at which data are processed on power consumption. Algorithms for optimizing the energy saving are proposed for several real-time system implementations including non-pipelined and pipelined. We also discuss extension to this preliminary work including selection of alternate processing units in order to reduce power consumption while maintaining deadlines.

1. INTRODUCTION

Power management [2, 9] is increasingly becoming a design factor in portable, hand-held computing systems. Since many such power-sensitive systems are also real-time, it makes sense to consider the effect that deadlines have on the ability of the designer to restructure and reschedule for power reduction. A combination of advances in hardware-software codesign and increased silicon capacity make it possible to consider sophisticated techniques for power management that rely on an ability to restructure software, hardware and schedules.

It is important to emphasize that optimizing a design for reduced energy consumption must be addressed at the system level. Embedded real-time systems are made up of a diverse set of components (*e.g.,* actuators, sensors, ASICs, microcontrollers, *etc.*). While it is possible to optimize individual components, this does not guarantee a minimal energy consumption at the system level because components do not operate in isolation. Indeed, changing the operation of one component may affect the operation (and hence, the energy consumption) of other components. It is therefore only at the system level that the interaction of all components can produce a global energy consumption estimate. Ultimately, one should aim at optimizing the power consumption of the complete system–actuators, sensors and computing hardware.

Re-scheduling and power-down techniques have been applied variously to minimize the power dissipation of a system. However, most of the these works focus on the techniques at the logic or behavioral levels of architectural abstraction. In particular, Monteiro et al. [7], describe a technique for rescheduling and pipelining of a control dataflow graph (CDFG) that enables the circuit in one part of a branch to be powered down once the value of branches conditional is known. This technique increases the control steps, however by using pipelined units they can overcome the throughput degradation. The somewhat related techniques of *precomputation* [1] and *guarded evaluation* [10] provide circuitry for powering down logic that is unused for the computation of the current clock cycle.

In this paper, we present some preliminary results of our work on system-level power management for real-time systems. We describe a technique based on rescheduling, buffer insertion and power down to reduced the energy consumption of a system of processing units. The technique draws on the vectorization idea from parallel processing. Vectorization (or *block-processing*) through rescheduling has been successfully used for improving the computation speed of DSP applications [5, 8]. Our goal, however, is to minimize system power consumption through rescheduling. Our main contribution is in identifying the effect of data execution rescheduling on the overall system power consumption. We also propose algorithms for optimizing the energy saving in several real-time system implementations including non-pipelined and pipelined.

The paper is organized as follows. Section 2. introduces some needed terminology and notation and then describes the problem through an example. Solutions to the rescheduling and buffer insertion problem for non-pipelined and pipelined systems are presented in Section 3. Finally Section 4. discusses how these techniques might be extended to allow multi-choice component selection, when competing components have different timing and energy properties.

2. PROBLEM DESCRIPTION

In this paper a set of real-time tasks is modeled as a directed acyclic graph (DAG). Each node represents a group of computations (or a task) and each edge represents a dependency between tasks. Further, we assume that each task is to be executed by a processing unit (PU). (Extensions to these assumptions will be discussed in the last section.) Interconnections between PUs are determined by the system architecture (e.g., global bus or local bus). Each PU has a number of parameters. In particular, we are interested in the power dissipated while (*i*) starting up, (*ii*) running (computing), (*iii*) idle (powered up but not computing), and (*iv*) powering down. Also, we are concerned with time taken to start up, compute and shutdown. Each PU has an associated energy dissipation graph, such as the one shown in Figure 1 where the parameters are defined as follows:

- $\tau_u(i)$: the time required to power up the i-th PU from off (note that "off" means the clock to the PU is gated)
- $\tau_d(i)$: the time required to power down the i-th PU
- $\tau_a(i)$: the time the i-th PU spends in active state (i.e., computing its function) which is dependent on the data parameters to the PU
- $\tau_{id}(i)$: the time the i-th PU spends in the idle state (i.e., computing nothing)
- $\tau_{ai}(i)$: the time required by the i-th PU to go from active state to idle state

[1] Hewlett-Packard Laboratories, Bristol, UK.
[2] Dept. of Computer Science & Engineering, Univ. of Notre Dame, Notre Dame, IN, USA. Chen's research was supported in part by NSF under Grant CCR-9623585. Hu's research was supported in part by HP Labs, Bristol, England under an external research program grant, and by NSF under grant MIP-9701416.
[3] Dept. of Electrical & Computer Engineering, Western Michigan Univ., Kalamazoo, MI, USA. The research was supported in part by HP Labs, Bristol, England under an external research program grant.

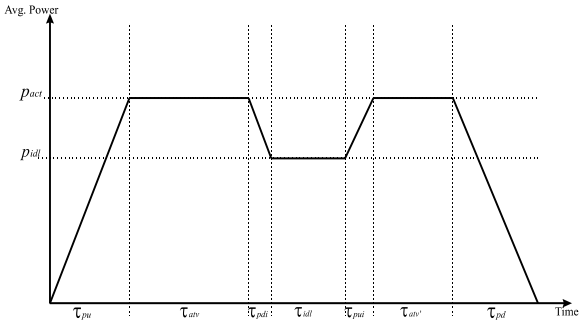

Figure 1. Example energy graph for a processing unit

$\tau_{ia}(i)$: the time required by the i-th PU to go from idle state to active state

$p_a(i)$: power dissipation of the i-th PU in the active state

$p_{id}(i)$: power dissipation of the i-th PU in the idle state

To simplify the formulation of the problem, we assume that τ_a is fixed for a PU on all possible inputs. Furthermore for each PU (denoted by U_i), the time to power up and power down is fixed. The energy dissipated when powering up and powering down is $\frac{1}{2}[\tau_u(i)p_a(i) + \tau_d(i)p_a(i)]$ and is denoted by $E_O(i)$. In our following deposition, we will ignore the energy due to the transitions between idle and active states in order to simplify the analysis.

Referring to Figure 1 it is easy to see that energy consumption could be reduced by powering down a PU while it is in idle state. However, the energy associated with powering down and up a unit may not be negligible. Repeated shutting down during short idle intervals may not be desirable. On the other hand, if one can adjust the time at which data is processed (i.e., rescheduling), it may be possible to minimize the number of power down and power up cycles, and hence achieve a greater energy saving.

To illustrate this idea, let us consider a simple example real-time system in which two tasks, A and B, are connected in series and the output of T_1 is the input of T_2. Two PUs, U_1 and U_2 are used to implement the two tasks, respectively. Figure 2 shows the basic schedule for this system in which

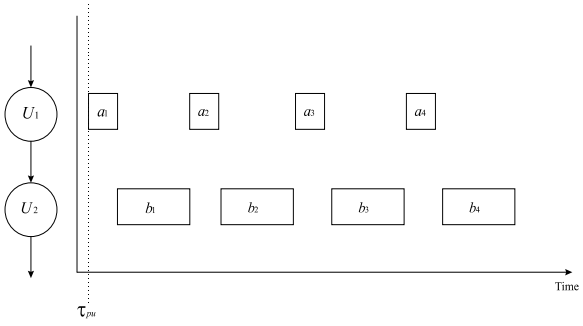

Figure 2. Example simple schedule

four data elements are to be processed. Observe that both units have idle time (indicated by the shaded areas) during which no active data processing occurs. In this case the total energy consumption can be calculated as

$$E_1 = 4 \cdot [\tau_a(1) \cdot p_a(1) + \tau_a(2) \cdot p_a(2)] + 3 \cdot [\tau_a(2) \cdot p_{id}(1) + \tau_a(1) \cdot p_{id}(2)] + E_O(1) + E_O(2) \quad (1)$$

For the same schedule, if we introduce one buffer to store the data produced by U_1, the energy consumption can be modified to

$$E_2 = 4 \cdot \sum_{i=1}^{2} \tau_a(i) \cdot p_a(i) + E_O(1) + E_O(2) + E_B \quad (2)$$

where E_B is the energy consumption by the buffer. (Energy consumption of a buffer is assumed to be a constant. This may be modified as pointed out in the last section.) Comparing Equations (1) and (2) a relationship can be derived between the parameters so that energy savings can be achieved. The energy savings differ depending on the number of inserted buffers. If there are four buffers available instead of one, a new schedule can be designed as shown in Figure 3. In this case, the energy consumption can be

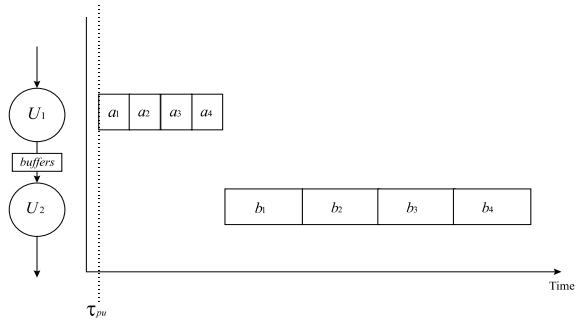

Figure 3. Example schedule for PUs and buffer

computed as

$$E_3 = 4 \cdot [\tau_a(1) \cdot p_a(1) + \tau_a(2) \cdot p_a(2)] + E_O(1) + E_O(2) + 4 \cdot E_B \quad (3)$$

Again, greater energy saving could be obtained for certain combinations of τ_a, p_{id}, E_O and E_B values.

In the rest of this paper we will discuss the scheduling and buffer insertion idea for several types of real-time system implementations. We first introduce some notation and terminology. Since we are focusing on real-time systems whose tasks can be represented as directed acyclic graph, we will represent the PUs that implement these tasks also as a DAG $G = (V, A)$ where a node $v_i \in V$ corresponds to the processing unit U_i. (Associated with each v_i are the timing and power parameters that were defined earlier in this section.) An arc $a_{i,j} \in A$ represents a dependency between processing units U_i and U_j. Finally, let $P_C \subset V$ be the nodes on the longest path from input to output.

As illustrated previously, inserting buffers into arcs can be exploited to reduce the number of power up and power down cycles. Let $\lambda_{i,j}$ denote the number of buffers needed at the arc $a_{i,j}$ when a single data element is processed at a time. If k input data elements are to be processed at a time, the number of buffers needed in $a_{i,j}$ increases by a factor of k. Specifically, the number of buffers required on $a_{i,j}$ becomes $b_{i,j}(k) = k \cdot \lambda_{i,j}$. The total number of buffers required in a system then becomes

$$B(k) = \sum_i \sum_j b_{i,j}(k) \quad (4)$$

For a given real-time application, a deadline $D(1)$ is defined such that the time it takes to process the first data element cannot exceed $D(1)$. $D(m)$ is defined similarly. If k data are processed at a time, the actual times it takes to process the first data element and m data elements are denoted as $T(1,k)$ and $T(m,k)$, respectively. Further, let $\delta_E(m,k)$ be the difference in energy consumption between

two schedules that either process k data at a time or simply one data at a time (assuming power-down is used). Then, the problem of scheduling and buffer insertion for reducing power can be described as follows.

Given a DAG representing the system under consideration, find k and $B(k)$ such that

(i) the total energy saving $\delta_E(m,k)$ is maximized,
(ii) the deadline constraints $D(1)$ and $D(m)$ are satisfied, and that
(iii) the data dependencies of the original DAG are preserved.

3. PROBLEM SOLUTIONS

In this section, we discuss our preliminary results for solving the scheduling and buffer insertion problem for several system implementations.

3.1. Non-pipelined systems

Consider a general non-pipelined system with m data elements to be processed. If k data elements are scheduled to be processed at a time, one method is to insert $b_{i,j}(k) = k \cdot \lambda_{i,j}$ buffers at each arc, $a_{i,j}$. In this treatment, the processing time $T(1,k)$ and $T(m,k)$ can be calculated as

$$T(1,k) = \sum_{v_i \in \{P_C - v_n\}} k \cdot \tau_a(i) + \tau_a(n) \quad (5)$$

$$T(m,k) = m \cdot \sum_{v_i \in P_C} \tau_a(i) \quad (6)$$

where v_n is the last node on the longest path. Note that rescheduling in this case does not increase the total processing time of m data. The total energy saving can be computed as

$$\delta_E(m,k) = \sum_{v_i \in V} [m(1-\frac{1}{k})E_O(i)] - (k-1) \cdot E_B \cdot \sum_{a_{i,j} \in A} \lambda_{i,j} \quad (7)$$

We give a straightforward procedure in the following to find the optimal schedule:

1. Find an upper bound on k based on the deadline constraints. That is,

$$k_{upper} = \frac{D(1) - \tau_a(n)}{\sum_{v_i \in \{P_C - v_n\}} \tau_a(i)}$$

2. Using Equation (7), set $\frac{\partial \delta_E(m,k)}{\partial k} = 0$ and solve the resulting quadratic equation for the optimum k value (k_{opt}).
3. Let $k = \min\{k_{opt}, k_{upper}\}$.

The total number of buffers, $B(k)$, can be reduced if the architecture of a non-pipelined system allows time sharing of the buffers. Here, we consider a somewhat simplified sharing scheme in which sharing is constrained only by data integrity and no other constraints are enforced. In this case, the minimum number of buffers for processing k data elements at a time, $B_{min}(k)$, can be expressed as $B_{min}(k) = \beta \cdot k$. Observe that finding β is equivalent to finding the maximum number of outputs that are *alive* at any instant of time. An output is alive if its data have not been consumed by the PU's that read the data. In the following we sketch an algorithm for solving this problem.

1. Use the topological sort algorithm to find the length of longest path, l_i, from input to each node v_i, where the path length represents the sum of $\tau_a(j)$'s for every node v_j on the path.
2. Use a sorting algorithm to assign each node an index I_i based on the ascending order of l_i. Nodes with equal path lengths get the same index value.
3. Set $\beta_0 = 0$.
4. For each index value I_i, compute β_{I_i} by subtracting the total number of inputs at every node with the same I_i value from β_{I_i-1}, and adding the nodes' number of outputs to the result.
5. $\beta = \max\{\beta_{I_i}\}$

Now, by replacing the last term in Equation (7) with $(k-1) \cdot \beta \cdot E_B$, the procedure used for finding the optimal k described above can be used similarly for this buffer sharing scheme.

3.2. Pipelined systems

If a real-time system requires certain throughput rate, a pipelined design may be desirable. The pipeline period is defined as $\mathcal{T} = \max_{1 \le i \le n}\{\tau_a(i)\}$. Then, $T(1,1) = N_C \cdot \mathcal{T}$ and $T(m,1) = (N_C + (m-1))\mathcal{T}$, where N_C is the total number of nodes along the longest path. A simple example in Figure 4 helps to see that pipelined systems could also benefit from buffer insertion. Here, two processing units are connected in series where the output of U_1 is the input of U_2. Figure 4(a) shows the basic schedule for this pipelined

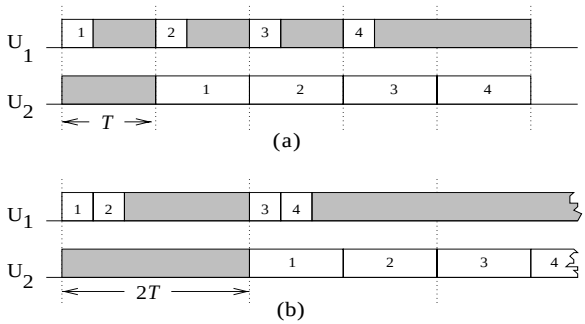

Figure 4. Feasible schedules for the two processing unit problem. Shaded areas represent idle periods for the respective processing units. Observe that the pipeline period in (b) is twice that of (a).

system and Figure 4(b) shows a schedule if two buffers are inserted to allow two data elements to be processes at a time. Comparing the two schedules both with the power-down option, we obtain an energy saving of

$$\delta_E(4,1) = 2E_O(1) - E_B \quad (8)$$

Of course, the additional buffer has increased the latency by $\mathcal{T}$. We have assumed that $\tau_a(1) < \tau_a(2)$ in the above example but it is easy to derive similar equations if $\tau_a(2) < \tau_a(1)$. For the case where $\tau_a(1) = \tau_a(2) = \mathcal{T}$, no individual powering up or down of processing units is required.

For a general pipelined system, the amount of energy saving can be computed by

$$\delta_E(m,k) = \sum_{v_i \in V} [m(1-\frac{1}{k})\tilde{E}_O(i)] - (k-1) \cdot E_B \cdot \sum_{a_{i,j} \in A} \lambda_{i,j} \quad (9)$$

where $\tilde{E}_O(i) = E_O(i)$ if $\tau_a(i) < \mathcal{T}$ and $\tilde{E}_O(i) = 0$ otherwise. The processing time can be calculated as follows.

$$T(1,k) = k \cdot (N_C - 1) \cdot \mathcal{T} + \tau_a(n) \quad (10)$$

$$T(m,k) = T(1,n) + (m-k) \cdot \mathcal{T} = k \cdot (N_C - 2) \cdot \mathcal{T} + \tau_a(n) + m \cdot \mathcal{T} \quad (11)$$

To preserve data synchronization in the original problem, the following requirement must be satisfied. Let $in_1(i)$ and $in_2(i)$ be two inputs to v_i, and let $N_{in_1(i)}$ and $N_{in_2(i)}$ be the number of processing units in a path from the primary system input to $in_1(i)$ and $in_2(i)$, respectively. Then, if $N_{in_1(i)} > N_{in_2(i)}$, $k(N_{in_1(i)} - N_{in_2(i)})$ buffers should be inserted along the path to $in_2(i)$. When there exist multiple such paths, the locations of these buffers can effect the final number of buffers needed. Note that k is simply a scaling factor. Hence, we can generalize this problem as follows. Let $\tilde{b}_{i,j} = 1 + \lambda_{i,j} \cdot b_{i,j}$. Then for each arc $a_{i,j}$ find $\tilde{b}_{i,j}$ such that

(*i*) The sum of $\tilde{b}_{i,j}$'s on every path from the primary input to a node is the same.

(*ii*) $\beta = \sum_{a_{i,j} \in A} \tilde{b}_{i,j}$ is minimized.

The techniques for retiming synchronous circuits [6] can be modified to solve this problem. Specifically, a retiming method similar to the one proposed in [3] can be applied to this problem. It follows then that the total number of buffers needed along the arcs can be calculated as $\beta \cdot k$.

Now, we give the following iterative procedure for finding k and number of buffers needed to design a pipelined system for processing k data at time.

1. Find β as outlined above.
2. Find an upper bound on k based on the deadline constraints. That is,

$$k_{upper} = \min \left\{ \frac{D(1) - \tau_a(n)}{(N_C - 1) \cdot \mathcal{T}} , \frac{D(m) - \tau_a(n) - m \cdot \mathcal{T}}{(N_C - 2) \cdot \mathcal{T}} \right\}$$

3. With

$$\delta_E(m,k) = \sum_{v_i \in V} [m(1 - \frac{1}{k}) \tilde{E}_O(i)] - (k-1) \cdot E_B \cdot \beta \quad (12)$$

set $\frac{\partial \delta_E(m,k)}{\partial k} = 0$ and solve the resulting quadratic equation for the optimum k value (k_{opt}).

4. Let $k = \min\{k_{opt}, k_{upper}\}$.

4. DISCUSSION AND FUTURE WORK

Power management in real-time embedded systems is both necessary and complex. There are significant opportunities for optimization in both power reduction and power smoothing. While these techniques are valuable when defining the silicon structure and hardware-software scheduling, they can also be applied to the complete system including sensors and actuators (components which tend to dominate the power budget).

In this paper, we presented our preliminary work on rescheduling and buffer insertion when there is slack time available in a real-time system. Based on our observations on several embedded systems under development, the approach seems very promising. Currently, we are collecting data from some applications and will present such data in the final version.

The problems and techniques presented here, while reasonably complex, do not reflect the full reality of optimizing real-time systems for energy efficiency and performance. First, real-time systems may contain cycles and complex dependency constraints. Also, tasks may share same processing units. The system architectures can also have many variations. More sophisticated rescheduling techniques will be necessary. Furthermore, the assumption in this paper is that PUs are not internally pipelined, hence more complex rescheduling is unnecessary but the techniques discussed result in a reduced throughput. Introducing pipelined PUs may enable this subsequent loss in throughput to be recovered or even increased, but at the cost of a more complex rescheduling algorithm. In addition pipelined PUs would enable our model to support more realistic behavior of components such as CPU RISC cores and DSP cores.

Currently a constant energy cost is associated with buffers to enable the rescheduling, which is clearly simplistic. A better cost model for buffers is also required, this should include the power dissipation of buffers in active and idle state and a time delay model for inserting and retrieving data. To determine number of buffers needed, fanouts of processing units and different requirements on buffer sharing should also be considered. Such extensions to the current model are relatively easy.

A more complex extension is to support the selection of PUs for particular tasks, where each PU would have an associated model including latency, throughput and energy graphs. Selection of actuators and sensors should also be included. These introduce a set of complex trade-offs between timing, performance and power utilization. In some previous work [4] we have shown that it is possible to find system architectures which have a good trade-off between conflicting attributes such as cost and speed. The software toolkit (called **EvoC**) will not only choose a partition of tasks between hardware and software implementations, but also select appropriate hardware devices. To study trade-offs including power consumption, we are investigating the integration of the techniques presented in this paper into **EvoC** for design space exploration of low-power, real-time embedded systems.

REFERENCES

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh and M. Papaefthymiou, "Precomputation-Based sequential logic optimization for low power", *IEEE Trans. on VLSI Systems*, **2**(4), 330-335, 1994

[2] A. Bellaouar and M.I. Elmasry, *Low Power Digital VLSI Design: Circuits and Systems*, Kluwer Academic Publishers, 1995.

[3] X. Hu, S. Bass and R. Harber, "Minimizing the number of delay buffers in the synchronization of pipelined systems", *IEEE Trans. on CAD of Integ. Cir. & Sys*, **13**(12), 1441-1449, 1994

[4] X. Hu, G. Greenwood and J. D'Ambrosio, "An evolutionary approach to hardware/software partitioning," *Parallel Problem Solving from Nature IV*, Lecture Notes in Computer Science 1141, H.M. Voigt, W. Ebeleing, I. Rechenberg and H.P. Schwefel (Eds.), Springer-Verlag, 900-909, 1996

[5] K. Lalgudi, M. Papaefthymiou, and M. Potkonjak, "Optimizing systems for effective block-processing : the k-delay problem", *IEEE 33rd Design Automation Conference*, pp.714-719, 1996.

[6] C. Leiserson and J. Saxe, "Retiming synchronous circuitry", *Algorithmica*, **6**(1), 5-35, 1991

[7] J. Monteiro, S. Devadas, P. Ashar and A. Mauskar, "Scheduling techniques to enable power management", *IEEE 33rd Design Automation Conference*, 1996

[8] M. Potkonjak and J. Rabaey, "Pipelining: just another transformation", *Int'l Conf. on Application Specific Array Processors*, 163-175, 1992

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

[10] V. Tiwari, P. Ashar and S. Malik, "Guarded evaluation: pushing power management to logic synthesis/design", *Int'l Symp. on Low Power Design*, 221-226, 1995