MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Computing During Supply Voltage Switching in DVS Enabled Real-time Processors

Chunjie Duan, Sunil Khatri

TR2006-033 May 2006

Abstract

In recent times, much attention has been devoted to power optimization for real-time systems, while guaranteeing that such systems meet their hard (or soft) scheduling deadlines. To reduce power, different tasks in such systems may be run at different power supply voltages, in order to maximally utilize slack in the schedule. However, prior approaches have ignored the practical aspects of switching the power supply. In a typical IC, the VDD net is highly capacitive, and as a result, its vltage cannot be changed instantaneously. In traditional approaches, the assumption is that this net switches instaneously, which in effect makes it essential to include the VDD switching time in the worst-case execution time (WCET) or a process (adding pessimism to the WCET value). In our approach, we precisely model the switching of the VDD net, and allow the system to continue computations while VDD is being switched. the effect on the delay of tasks during this transition is precisely modeled. This allows a designer to obtain more realistic estimates of the WCET of a process, reducing the psssimism inherent in real-time system scheduling. Our approach can be implemented as a simple look-up table in a real-time scheduler. Our experimental results show that our model is highly accurate, with an error of less-than 0.2% compared to SPICE simulations.

IEEE International Symposium on Circuits and Systems

Computing During Supply Voltage Switching in DVS Enabled Real-time Processors

Chunjie Duan[†] Sunil P Khatri[*]
[†] Mitsubishi Electric Research Laboratories, Cambridge, MA 02139
[*] Department of EE, Texas A&M University, College Station TX 77843.

Abstract

In recent times, much attention has been devoted to power optimization for real-time systems, while guaranteeing that such systems meet their hard (or soft) scheduling deadlines. To reduce power, different tasks in such systems may be run at different power supply voltages, in order to maximally utilize slack in the schedule. However, prior approaches have ignored the practical aspects of switching the power supply. In a typical IC, the VDD net is highly capacitive, and as a result, its voltage cannot be changed instantaneously. In traditional approaches, the assumption is that this net switches instantaneously, which in effect makes it essential to include the VDD switching time in the worst-case execution time (WCET) of a process (adding pessimism to the WCET value). In our approach, we precisely model the switching of the VDD net, and *allow the system to continue computations while VDD is being switched*. The effect on the delay of tasks during this transition is precisely modeled. This allows a designer to obtain more realistic estimates of the WCET of a process, reducing the pessimism inherent in real-time system scheduling. Our approach can be implemented as a simple look-up table in a real-time scheduler. Our experimental results show that our model is highly accurate, with an error of $< 0.2\%$ compared to SPICE simulations.

1. Introduction

The area of embedded system scheduling arguably started with the seminal work of Liu and Layland [1] in 1973. In this paper, the authors assumed a single-processor system with n independent periodic tasks, and given worst case execution times (WCETs). Liu and Layland showed that if tasks were scheduled statically using a priority which was inversely proportional to their periodicity, the resulting schedule was optimum among all fixed priority schedules.

In recent times, the work of Liu and Layland has been extended in several ways. There have been several approaches to devise static power-conscious scheduling algorithms [2, 3, 4, 5]. Other dynamic schedulers were also reported [6, 7, 8, 9, 10] which utilized dynamic voltage scheduling (DVS). In a DVS processor, voltage may be modified dynamically, allowing the scheduler to trade off power for delay (by varying the VDD value of the processor). The independent task assumption of [1] was removed in [11]. Techniques to generate variable supply voltages were reported in [12, 13].

In the above scheduling algorithms, it is assumed that the delay overhead of VDD switching was negligible. However, this is not the case for realistic processors that are used to implement real time embedded systems. These processors have significantly capacitive VDD nets. For example, the capacitance on the VDD net in [14] was reported to be 160nF. In fact, designers make special efforts to *increase* this capacitance for signal integrity reasons. This clearly makes the zero VDD switching delay assumption weaker. As a result, the assumption that VDD switching has negligible delay overhead is unjustified in modern designs. If we were to use the zero VDD switching delay assumption, the worst-case VDD switching delay must be included in the WCETs of each task, resulting in more conservative WCETs. If the computation of a task is pre-empted n times by other tasks (operating at different voltages), we need to increment the WCET of the task by n times the worst case VDD switching delay.

For these reasons, it would be desirable to have a methodology which reduces the conservatism of WCETs by modeling the VDD switching delay and *accounting for it in the schedule*. In this work, we present such a technique. Suppose a task φ_i with scheduled voltage VDD_i was completed and the next task φ_j with scheduled voltage VDD_j is started. Let $VDD_i < VDD_j$. Assume that task φ_j had already been queued while φ_i was being processed. In this case, we begin the VDD switching from VDD_i to VDD_j *during* the time φ_i is being executed, which results in a condition where φ_i completes its work (defined henceforth as the number of cycles required in the computation of a task) earlier. Task φ_j now begins earlier than planned. We find the time T_1^* at which, if VDD switching from VDD_i to VDD_j is initiated, then the speed-up of φ_i is equal to the increased delay of φ_j. In other words, the work of both φ_i and φ_j completes before their respective deadlines. We formulate this problem and find an expression for the time T_1^*. We report the results of experiments to validate this expression, showing a close match between the mathematical model and the experimental delays.

Note that in the case that $VDD_i > VDD_j$, then the switching must be performed at the originally scheduled time (so that the work of φ_i can be guaranteed to complete). Since the average value of VDD is higher during the computation of φ_j, it completes earlier than scheduled. Once again, we ensure that the required work for φ_i and φ_j is completed before their respective deadlines.

The rest of this paper is organized as follows. In Section 2, we discuss some prior work in real-time scheduling. Section 3 describes our approach, while Section 4 reports the results of experiments we conducted to validate our approach. Section 5 summarizes our work.

2. Previous Work

In the seminal paper by Liu and Layland [1], the authors motivated the area of real-time systems, and provided a fixed priority scheduler which had an asymptotic upper bound for processor utilization of 69%. The focus of this work was schedulability, rather than power.

In recent times, with the growing interest in low-power real-time embedded systems, there have been several efforts to augment the work of [1] for low power applications. Most of these efforts attempt to reduce power by scaling the frequency of operation, the value of VDD[1], or by powering down the system during periods of inactivity. An excellent review of low power scheduling is found in [15].

In [2], the authors augment a fixed priority schedule in a power conscious manner. If there is dead-time in the schedule, such periods are filled in by reducing the clock frequency, VDD value or by system power-down. In [3], the authors devised an algorithm to find the optimal voltage for each task. They ignore the delay and power overhead of switching VDD. However, this is a problem in general since the VDD net on an IC can be significantly capacitive, especially for Systems-on-a-Chip (SOCs). For an large IC, this capacitance can be in the range of a a few 100 nF [14]. Later, in [4] an energy efficient fixed priority scheduling algorithm was reported, which could be used to find optimal voltages for each task or for entire task sets. Finally, in [5], a genetic DVS algorithm was presented.

Fixed priority dynamic voltage schedulers (DVS) have also been extended to dynamic schedulers. In [6], a DVS algorithm was reported for dynamic schedules, using slack analysis. In [7], the authors reported a static and dynamic algorithm for voltage and clock scaling of real-time embedded systems. In [8, 9], static as well as dynamic variable voltage schedulers were

[1] These techniques are classified as Dynamic Voltage Scaling (DVS) approaches. In these techniques, the VDD of the processor is reduced when there is slack in the schedule, thereby reducing the power. The VDD of a processor can be increased as well, resulting in a higher execution speed and power consumption.

reported for heterogeneous real-time distributed embedded systems.

The independent task assumption was removed in [11], where the authors reported a scheduling algorithm for periodic task graphs, with the additional ability to handle aperiodic tasks. These algorithms were generalized to the DVS scenario as well.

In [16], a battery aware static scheduler was presented, including an algorithm for voltage scalable processing elements (PEs). It was assumed that the PEs can perform voltage scaling. Such a capability is available in the Intel XScale processor [17] (in which voltage is continuously scalable) and the Transmeta Crusoe [18] (in which voltage is scalable in discrete steps).

In all the above efforts, the thrust was on scheduling algorithms. The time required to switch VDD was ignored, and implicitly included in the task WCET. This adds pessimism to the schedule, since the worst-case time taken by the VDD net to switch must be factored into the WCET of each process. In our work, we *allow task execution during VDD switching, allowing the WCET value to be more realistic for real applications*. This is a circuit oriented approach to DVS based scheduling, in which we are able to *eliminate the overheads of VDD switching*. Since our approach is *independent of the scheduling algorithms used*, it can be used along with any of the above scheduling algorithms. *The contribution of this paper is to describe a technique where the traditional pessimism in WCET is reduced. The results from this paper can be applied to any of the scheduling algorithms in practice today.*

Techniques to generate variable supply voltages were described in [12, 13]. These works describe solid state DC-DC regulators which could be used in VDD-scaled real-time systems. Our approach would work with either kind of regulator. Of course, the voltage regulator that is used can be off-chip as well.

3. Our Approach

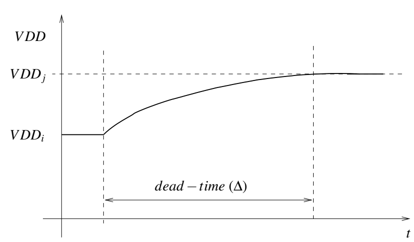

Figure 1: DVS timing diagrams

Figure 1 illustrates the problem being addressed by our approach. When a DVS enabled processor switches from VDD_i to VDD_j during operation, its switching waveform is a rising (or falling) exponential, since the VDD net on a modern IC is significantly capacitive, and the voltage regulation circuit has a finite series resistance. *In this section, we discuss both the cases, in which $VDD_i < VDD_j$ and $VDD_i > VDD_j$.* Traditional scheduling approaches consider the VDD transition to be an ideal step function, which means that the worst case *dead-time* (Δ) during VDD switching increases the WCET of each process. With typical values (VDD net capacitance $\sim$ 100nF, supply resistance $\sim 1\Omega$), we find that this dead-time can be in the 100ns range. This adds to the pessimism in the WCET value. Since the capacitance of the VDD net is quite high, the pessimism introduced can be quite high, especially when a process is repeatedly interrupted by other processes (which compute at different VDD values). If in fact the process is interrupted *n* times by other processes running at different VDDs, then its WCET must be increased by *n* times the worst case VDD switching delay.

Consider the VDD switching waveforms in Figure 2. This figure shows two tasks φ_i and φ_j that are scheduled contiguously. φ_i starts at $t = 0$ and both tasks must be completed before $t = T_2$. The WCETs of φ_i and φ_j are C_i and C_j respectively and their supply voltages are VDD_i and VDD_j respectively. Assume that $VDD_i < VDD_j$. The deadline for φ_i is T_1, and that of φ_2 is T_2. Assume that φ_j is queued already.

Subfigure a) illustrates the ideal case (i.e. the VDD switching waveform is an ideal step function). In other words, this assumes that the dead-time (Δ) is zero.

Subfigure b) illustrates the actual VDD switching waveform, which our method incorporates into the schedule. Our method must obviously meet

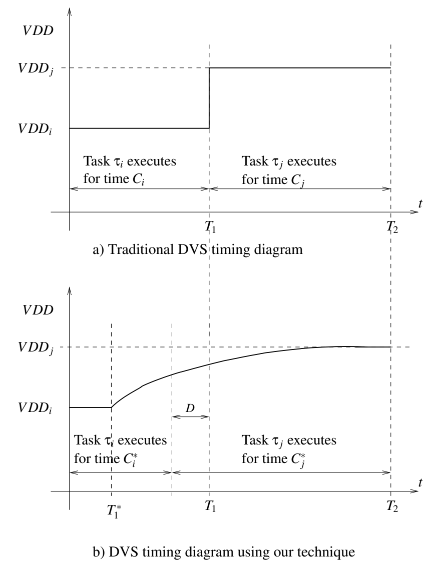

Figure 2: DVS timing diagrams

the deadlines T_1 and T_2. As we can see in the figure, because the rising time is significant, φ_j will not complete in time if the switching from VDD_i to VDD_j starts at the same time as was used in the ideal case of Figure 2a). In our approach, we start the VDD transition from VDD_i to VDD_j at time T_1^*, (*during* the execution of task φ_i). As a consequence, the average VDD value for task φ_i increases above VDD_i, so that it completes in time C_i^*, earlier than scheduled. Note that in this case, all the scheduled work for task φ_i is completed before the deadline T_1. Similarly, the average VDD value for task φ_j decreases below VDD_j, resulting in a situation where φ_j completes in time C_j^* (longer than its its original WCET). However, we must guarantee that all the work for task φ_j is completed before its deadline T_2. In other words, we need to find the time T_1^* to start the VDD transition, such that the speed-up of task φ_i is equal to the slow-down of task φ_j. Therefore, we need to find T_1^* such that:

$$C_i - C_i^* = C_j^* - C_j = D \tag{1}$$

In this manner, we can perform computation while VDD is being switched, allowing us to reduce the pessimism in the WCETs of the tasks. *The VDD switching is therefore performed on-the-fly, even while tasks φ_i and φ_j are being computed.*

If $VDD_i > VDD_j$, then by starting the transition at the same time T_1 as in the ideal case of Figure 2a), we can ensure that the work of φ_i is completed before its deadline T_1. For φ_2, the average VDD value is above VDD_j, and hence it completes earlier than scheduled, again guaranteeing that its work was completed before its deadline T_2.

The computation of T_1^* is performed as follows.

The total delay of computing tasks φ_i and φ_j using our on-the-fly VDD switching methodology is given by Equation 2.

$$\begin{aligned} N_{ours} = & \int_0^{T_1^*} F[VDD_i]dt + \\ & \int_{T_1^*}^{T_2} F[VDD_j + (VDD_i - VDD_j)e^{\frac{-(t-T_1^*)}{\tau}}]dt \end{aligned} \tag{2}$$

Here $F(x)$ is the frequency of operation of the embedded processor when the supply value is $VDD = x$ volts. N_{ours} is the total number of computational cycles that are performed in the interval [0, T_2] by our method. Also, τ is the RC time constant of the on-chip VDD network.

Similarly, the total number of computational cycles N_{ideal} (for the method of Figure 2a) which assumes that the VDD change is a step function) is given by Equation 3.

$$N_{ideal} = \int_0^{T_1} F[VDD_i]dt + \int_{T_1}^{T_2} F[VDD_j]dt \tag{3}$$

To compute $F[\cdot]$, we use the Alpha power law [19] MOSFET model. According to this model, the delay of a MOS circuit as a function of VDD is given by Equation 4. *Note that we assume that if the supply voltage is V_{DD}, then the frequency is inversely proportional to $D(V_{DD})$. In other words, frequency is not an independent variable, and is coupled to the value of V_{DD} utilized.*

$$D(V_{DD}) = K \cdot \frac{V_{DD}}{(V_{DD} - V_T)^{\alpha}} \tag{4}$$

The first task we performed was to find the value of α. We determined through experimentation that $\alpha = 2$. We used a simple 9-stage ring oscillator, along with two other circuits (apex7 and alu4) from the MCNC91 benchmark suite. We mapped these circuits using SIS [20] to a library of 21 gates using a predictive 0.1μm process technology [21]. The results are shown in Figure 3, which plots the value of $\frac{VDD}{(VDD-V_T)^2}$ against delay. This shows an accurate fit for $\alpha = 2$.

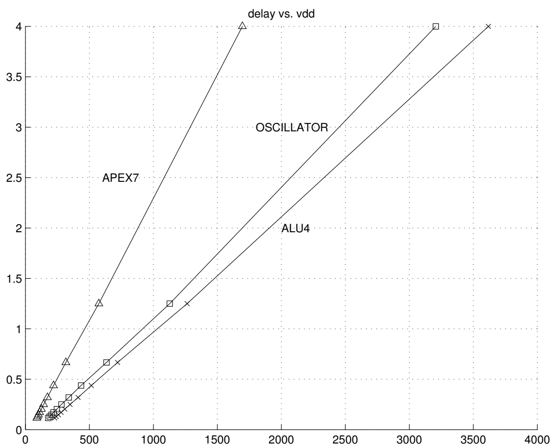

Figure 3: Finding the value of α in Equation 4

Therefore, the processor delay (and consequently its frequency) is:

$$D(V_{DD}) = K \cdot \frac{V_{DD}}{(V_{DD} - V_T)^2} \tag{5}$$

$$F(V_{DD}) = K' \cdot \frac{(V_{DD} - V_T)^2}{V_{DD}} \tag{6}$$

By simply plugging Equation 6 into Equations 3 and 2, we have the following two equations that compute the number of instruction executed in a time period, (in this case, [0, T_2]) with and without taking into account of the V_{DD} switching delay. We assume in the sequel that the VDD change is from a value V_a to V_b (where $V_a < V_b$).

If V_{DD} switching is a step function (the ideal case of Figure 2a)), we have:

$$\begin{aligned} N_{ideal} &= \int_0^{T_1} F(V_a)dt + \int_{T_1}^{T_2} F(V_b)dt \\ &= K' \cdot \{\int_0^{T_1} \frac{(V_a - V_T)^2}{V_a} dt + \int_{T_1}^{T_2} \frac{(V_b - V_T)^2}{V_b} dt\} \\ &= K' \cdot \{\frac{(V_a - V_T)^2}{V_a} \cdot T_1 + \frac{(V_b - V_T)^2}{V_b} \cdot (T_2 - T_1)\} \\ &= K' \cdot \{\frac{(V_b - V_T)^2}{V_b} \cdot T_2 + [\frac{(V_a - V_T)^2}{V_a} - \frac{(V_b - V_T)^2}{V_b}] \cdot T_1\} \end{aligned} \tag{7}$$

Now let us account for the VDD switching delay (the case of Figure 2b)). From T_1^* to T_2, V_{DD} is an exponential function of time:

$$\begin{aligned} V_{DD}(t) &= V_b + (V_a - V_b) \cdot e^{-\frac{t}{\tau}} \\ &= A + B \cdot e^{-\frac{t}{\tau}} \end{aligned} \tag{8}$$

where A = V_b and B = $V_a - V_b$. Substituting Equation 8 in Equation 2, and assuming that $T_2 \gg \tau$, we get:

$$\begin{aligned} N_{ours} &= \int_0^{T_2} F(V(t))dt \\ &= K' \cdot (\int_0^{T_1^*} \frac{(V_a - V_T)^2}{V_a} dt + \\ &\quad \int_{T_1^*}^{T_2} \frac{(A + B \cdot e^{-\frac{t}{\tau}} - V_T)^2}{A + B \cdot e^{-\frac{t}{\tau}}} dt) \end{aligned}$$

Without loss of generality, we let $T_1^* = 0$. In that case, T_1 becomes the *look-ahead* time, which we need to determine:

Let $t' = t/\tau$ and $T_2' = T_2/\tau$, and using the assumption that $T_2 \gg \tau$, we get:

$$\begin{aligned} N_{ours} &= \tau K' \int_0^{T_2'} (A + Be^{-t'} - 2V_T + \frac{V_T^2}{A + Be^{-t'}})dt' \\ &= \tau K' \{AT_2' - 2V_T T_2' + B(1 - e^{-T_2'}) \\ &\quad + \frac{V_T^2}{A}[T_2' + ln(A) - ln(A+B)]\} \\ &= \tau K' \{\frac{(A - V_T)^2}{A} T_2' \\ &\quad + [B + \frac{V_T^2}{A}(ln(A) - ln(A+B))]\} \\ &= K' \frac{(A - V_T)^2}{A} T_2 \\ &\quad + \tau K'[B + \frac{V_T^2}{A} \cdot (ln(A) - ln(A+B))] \\ &= K' \frac{(V_b - V_T)^2}{V_b} T_2 \\ &\quad + \tau K'[V_a - V_b + \frac{V_T^2 \cdot ln(V_b) - ln(V_a)}{V_b}] \end{aligned} \tag{9}$$

To find the time T_1 (look-ahead time) that makes $N_{ours} = N_{ideal}$, we combine Equation 9 and Equation 7 to get:

$$\begin{aligned} & K' \cdot \{\frac{(V_b - V_T)^2}{V_b} \cdot T_2 + [\frac{(V_a - V_T)^2}{V_a} - \frac{(V_b - V_T)^2}{V_b}] \cdot T_1\} \\ &= K' \frac{(V_b - V_T)^2}{V_b} \cdot T_2 + \tau K'(V_a - V_b + \frac{V_T^2(ln(V_b) - ln(V_a))}{V_b}) \end{aligned} \tag{10}$$

The first terms on both sides cancel each other, making the resulting equation independent of T_2, yielding:

$$T_1 = \frac{V_a - V_b + \frac{V_T^2}{V_b}(ln(V_b) - ln(V_a))}{\frac{(V_a - V_T)^2}{V_a} - \frac{(V_b - V_T)^2}{V_b}} \cdot \tau \tag{11}$$

As we can see, T_1 is independent of T_2, which is intuitively reasonable. The above expression for T_1 is true when we assume $T_2 \gg \tau$. *Note that if the $T_2 \gg \tau$ assumption does not hold, then a closed form expression for T_1 can still be computed. In this case, T_1 has a dependence on T_2 as well.*

T_1 is dependent on V_T and the V_{DD} of both tasks φ_i and φ_j. Also in reality, V_a and V_b are greater than V_T for circuits that operate above threshold voltage. Today's processors all utilize super-threshold conduction to perform their computations.

From Equation 11, the lower bound of $T_1 = \tau$ is achieved when $V_T = 0$. This bound is not very meaningful in practice, since $V_T = 0$ would result in extremely high leakage currents in the design.

When both V_a and V_b are much greater than V_T, T_1 is very close to τ.

Table 1 reports the calculated look-ahead delay T_1 (in multiples of τ) for several values of V_a and V_b. For this table, we used $V_T = 0.3V$.

Assuming the circuit is operating at or above threshold voltage V_T, we can determine from Equation 11 that the upper bound of T_1 is $1.494 \cdot \tau$. This upper bound is achieved when $V_a = V_T$. If V_a is no lower than 0.4V and $V_T = 0.3V$ (reasonable values for modern embedded processors) the upper bound drops to $1.12 \cdot \tau$. In practice, we would compute the look ahead time T_1 on-the-fly, or simply obtain it by performing a table look-up. This is

$V_a \backslash V_b$	0.35	0.4	0.6	0.8	1.0	1.2	1.4	1.6
0.35	-	1.175	1.184	1.169	1.150	1.134	1.121	1.110
0.4	0.877	-	1.113	1.120	1.113	1.104	1.096	1.088
0.6	0.780	0.870	-	1.032	1.041	1.044	1.044	1.043
0.8	0.778	0.849	0.965	-	1.014	1.020	1.022	1.023
1.0	0.787	0.847	0.951	0.985	-	1.007	1.011	1.013
1.2	0.798	0.851	0.945	0.978	0.992	-	1.004	1.007
1.4	0.809	0.856	0.942	0.973	0.988	0.996	-	1.003
1.6	0.819	0.861	0.941	0.971	0.985	0.993	0.997	-

Table 1: Look-ahead time T_1 as a function of V_a and V_b (multiples of τ)

feasible if V_{DD} has a limited number of discrete values. In practice, however, it may be more attractive to use a fixed upper bound value for simplicity of implementation.

It should be noted that our technique may only be applied under two conditions. First, adjacent tasks φ_i and φ_j should be such that the task φ_j is queued before time T_1^*. Secondly, the quantity C_j^* should be such as to allow the VDD_i to switch completely to VDD_j. Of course since the switching characteristic is an exponential waveform, it is sufficient in practice for the transition to be greater than 99% complete (which requires that $C_j^* > 4.6 \times \tau$). This is easy to check statically once a schedule is available.

If $V_a > V_b$, then the situation is simpler: If φ_i has a WCET less than the original switching time, then Table 1 is used to determine the look-ahead time (assuming the deadline of both tasks is T_2.) If the task φ_i has a WCET which is equal to T_1, then the switching of VDD must be performed at the originally scheduled time (so that the work of φ_i can be guaranteed to complete). Since the average value of VDD is higher during the computation of φ_j, it completes earlier than scheduled. Once again, we ensure that the required work for τ_i and τ_j is completed before their respective deadlines. We can use our method (analogous equations can be derived for the $V_a > V_b$ case) to determine how much earlier φ_j will complete. The remaining time before the deadline T_2 can be used to put the processor into a sleep state, further reducing power. Alternately, the supply voltage V_b can be recomputed, allowing for further power reduction.

Utilizing our method: Our approach can be utilized for static as well as dynamic schedules. All that is required to be known to find the look-ahead time is the value of V_a, V_b, τ and T_2. The discussions above indicate the constraints that must be satisfied when applying our methodology. The computation of look-ahead time is performed at least $1.494 \cdot \tau$ before a deadline. A simple table lookup (which is inexpensive in terms of computation time and power) yields the value of look-ahead time. In case $T_2 \gg \tau$, then the look-ahead time is independent of T_2.

4. Experimental Results

To verify the correctness of our analysis, we ran some SPICE [22] simulations using the computed value of the look ahead time T_1 (using Equation 11). We switched the VDD signal at this time instant, and computed the total number of cycles (N_{ours}) processed by the embedded processor in the interval [0,T_2]. We compared this with the number of cycles N_{ideal} processed in the same interval when VDD switches in an ideal manner. We ran the simulation for 3μs, which is about $6 \cdot \tau$. We again used the 9-inverter ring oscillator circuit as a reference design.

Table 2 lists the simulation results for some typical V_{DD} values. As indicated in this table, the absolute maximum error is less than 15 clock cycles. This translates to a worst-case error of less than 0.2% when our method is employed. In practice, a small guardband on the value of the look-ahead time would avoid this small error.

V_a (V)	V_b (V)	τ (ns)	T_1 (ns)	N_{ideal}	N_{ours}	Err(%)
0.4	0.8	500	525.5	4324	4317	0.2
0.4	1.0	500	556.5	6480	6495	0.1
0.4	1.2	500	548	6994	7010	0.2
0.6	0.8	500	516	4497	4490	0.2
0.6	1.0	500	520.5	7010	6994	0.2
0.6	1.2	500	522	8755	8747	0.1
1.0	1.4	500	505.5	11324	11310	0.1

Table 2: Validation of our Approach with SPICE Simulations

The close match between the simulation results and our computed value indicates that our analysis is accurate.

5. Conclusions

In recent times, the ubiquity of low-power real-time embedded systems has opened up several interesting research problems in scheduling algorithms for such systems in a power-aware context. To achieve this, several research papers utilize dynamic voltage scaling (DVS) of the power supply. However, the delay incurred during the switching of the VDD signal has not been addressed in these works. As a result, the worst case VDD switching delay is implicitly included in the WCET of each task, leading to an increased pessimism in the WCETs of tasks.

In this paper, we have described a technique to perform on-the-fly DVS. In our approach, computation continues while the VDD is being switched. This allows us to have more realistic WCET estimates for tasks, and maximally utilize the available computation time. In our approach, when there are two tasks φ_i and φ_j scheduled consecutively with supply values V_{DD_i} and V_{DDj}, in a manner that φ_j is already queued, then we switch V_{DD} on-the-fly during the execution of φ_i. We have derived the expression for the time instant that the V_{DD} switching must begin during the execution of φ_i. Our model was validated with SPICE simulations, and has a maximum error of 0.2% over a variety of switching conditions.

In the future, we plan to extend the application of our ideas to the determination of when to invoke sleep and idle modes for embedded processors.

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association of Computing Machinery*, vol. 20, pp. 46–61, Jan 1973.

[2] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proceedings, Design Automation Conference*, pp. 134–139, 1999.

[3] I. H. D. Kirovski, Q. Gang, M. Potkonjak, and M. Srivastava, "Power optimization of variable-voltage core-based systems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1702–1714, 1999.

[4] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proceedings, Design Automation Conference*, pp. 828–833, June 2001.

[5] M. Schmitz, B. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems," in *Proceedings, Design, Automation and Test in Europe Conference and Exhibition*, pp. 514–521, March 2002.

[6] W. Kim, J. Kim, and S. Min, "Dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *Proceedings, Design, Automation and Test in Europe Conference and Exhibition*, pp. 788–794, March 2002.

[7] Y.-H. Lee and C. Krishna, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," in *Proceedings. Sixth IEEE Real-Time Technology and Applications Symposium*, pp. 156–165, 2000.

[8] J. Luo and N. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," in *Proceedings, Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 719–726, Jan 2002.

[9] J. Luo and N. Jha, "Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems," in *Proceedings, International Conference on VLSI Design*, pp. 369–375, Jan 2003.

[10] V. Culver and S. Khatri, "Dynamic voltage scaling algorithm for energy reduction in hard real-time systems," in *Proceedings, Asia South Pacific Design Automation Conference (ASP-DAC)*, (Shanghai, China), January 2005.

[11] J. Luo and N. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," in *Proceedings, IEEE/ACM International Conference on Computer Aided Design*, pp. 357–364, Nov 2000.

[12] V. Gutnik and A. Chandrakasan, "An efficient controller for variable supply-voltage low power processing," in *Proceedings, Symposium on VLSI Circuits*, pp. 158–159, 1996.

[13] W. Namgoong, M. Yu, and T. Meng, "A high-efficiency variable-voltage CMOS dynamic DC-DC switching regulator," in *IEEE International Solid State Circuits Conference*, pp. 380–381, 1997.

[14] W. Bowhill, R. Allmon, S. Bell, E. Cooper, D. Donchin, J. Edmondson, T. Fischer, P. Gronowski, A. Jain, P. Kroesen, B. Loughlin, R. Preston, P. Rubinfeld, M. Smith, S. Thierauf, and G. Wolrich, "A 300 mhz 64 b quad-issue cmos risc microprocessor," in *Digest of Technical Papers, IEEE International Solid-State Circuits Conference*, pp. 182–183, Feb 1995.

[15] N. Jha, "Low power system scheduling and synthesis," in *Proceedings, IEEE/ACM International Conference on Computer Aided Design*, pp. 259–263, Nov 2001.

[16] J. Luo and N. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Proceedings, Design Automation Conference*, pp. 444–449, June 2001.

[17] "Intel XScale Technology." http://www.intel.com/design/intelxscale/.

[18] "Transmeta crusoe overview." http://www.transmeta.com/crusoe/.

[19] T. Sakurai and A. Newton, "Alpha-power law MOSFET model and its application to CMOS inverter delay and other forumals," in *IEEE Transactions on Solid State Circuits*, vol. 25, pp. 584–594, 1990.

[20] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[21] "BSIM3 Homepage." `http://www-device.eecs.berkeley.edu/~bsim3/intro.html`.

[22] L. Nagel, "Spice: A computer program to simulate computer circuits," in *University of California, Berkeley UCB/ERL Memo M520*, May 1995.