RC23637 (W0506-090) June 22, 2005
Computer Science

IBM Research Report

Analysis of Transition Energy and Latency of the PowerDown State in Advanced System-on-a-Chip Processors: 0.13 um Process Update

C. Michael Olsen
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Malcolm Ware
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Analysis of Transition Energy and Latency of the PowerDown State in Advanced System-On-a-Chip Processors : 0.13 um Process Update

C. Michael Olsen, IBM Research Division, Hawthorne, NY
Malcolm Ware, IBM Research Division, Austin, TX

Abstract *- We present a qualitative analysis of the transition energy and latency associated with dynamically exploiting the two most efficient low power states in advanced System-On-a-Chip (SOC) processors. In particular we present an equation for calculation of the transition energy and latency of the PowerDown (PD) low power state. We show that the average power consumption in the PD state is significantly influenced by the transition energy, and that the transition energy may be so large that it becomes more efficient to exploit the "lesser" ClockSuspend state in systems which use an operating system that employ a periodic timer interrupt mechanism. In particularly this is true for small form factor mobile devices. In this respect the SOC should remain powered off during OSC stabilization to save the potentially large energy contribution from SOC leakage power. Fiinally, we argue that operating systems that employ a work dependent timing (WDT) scheme can effectively eliminate the transition periods and extend the time spent in the PD state. In turn this can significantly reduce the average power consumption in the PD state in well designed systems. We expect the PowerDown state to be of most use in small form factor WDT based mobile systems that have very low idling power levels.*

1. Introduction

Power management has risen to become a very important design aspect in mobile devices. Many mobile devices are operated predominantly in the idling mode. In this mode, users are not using the device but they still expect the device to provide full functionality and to have instant-on response. In mostly idling devices, the low power states of the hardware components are of particular importance. This is due to the extended time these device spend idling and during which, in an accumulated sense, overall more energy may be dissipated than during active periods, i.e., while executing code. Thus reducing the power consumption of the low power states is crucial to extending battery life.

The main cause of power consumption in the low-power states of System-On-a-Chip (SOC) processors is the CMOS leakage current which keeps increasing in every new technology release. The reasons for this is the constantly shrinking feature size and the lowering of threshold voltage to accommodate smaller supply voltages [1,2]. Additionally, the increasing complexity of SOCs causes the device count to increase which further adds to the leakage current. In particular, mobile processors fabricated in the most recent 0.13 um process are anticipated to have leakage powers that can no longer be disregarded in view of the total system power budget. The overall effect of this is that the power consumption is increasing to an unacceptable level in processor idle periods. Even though there are means for controlling the leakage current in the inactive parts of the processor (see [2] for a brief overview), these methods are mostly in the experimental stage.

In more recently announced SOCs, a new low-power state has been introduced, namely the *PowerDown* (*PD*) state [3,4,6,8]. In this state the power to most of the SOC is removed, thus eliminating the leakage power experienced in the less efficient low-power states, such as the *ClockSuspend* (*CS*) state, where the SOC clock is globally stopped but the SOC is still powered. However, since it takes additional energy and time to transition into and out of the *PD* state as compared to the *CS* state, it is not always economical to exploit the *PD* state.

This paper presents a qualitative analysis of how much energy and latency is associated with dynamically exploiting the *PD* state (whenever the system is idling between system timer interrupts) and determines when it becomes economical to exploit the *PD* state over the *CS* state. From a systems usage perspective, the analysis represents a mostly idling system where most external devices to the SOC are shut down, or disabled. These devices only need to be woken up and restored when the user or the system explicitly needs them. So if a LAN interface is down and needs to be reinitialized, this will take extra time, and is not included in our analysis. However, this will be a rare event in the overall scheme of things (for an idling system), and the same can be said about the audio device and the display.

The paper uses power values from processors fabricated in 0.13 um process technology. In a previous paper [19], we presented a similar analysis, except for processors manufactured in 0.18 um process technology. This paper also reflects an improvement in the power consumption of DRAM technology that has taken place since our previous paper.

The paper is organized as follows. In Chapter 2 we

will shed some more light on the low-power states of advanced SOCs. Then, in Chapter 3, we present the analysis of the energy and latency associated with exploiting the *PD* state. Chapter 4 shows the results of the analysis, and finally, in Chapter 5, we discuss when it becomes more economical to exploit the *PD* state over the *CS* state.

2. Low Power States In Advanced System-On-a-Chip Processors

Advanced mobile System-On-a-Chip (SOC) processors [3-13] have multiple Power Management states. For a brief survey of some of these processors see [14]. The high-level configuration of advanced SOCs is shown in Figure 1.

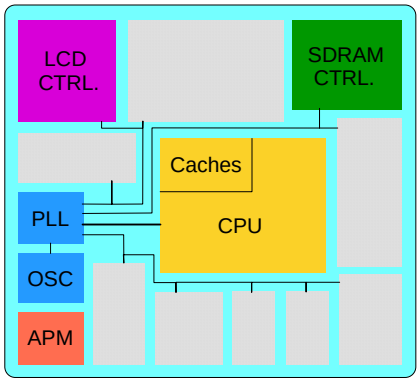

Figure 1. Typical high-level architecture of advanced SOC processors. The lines branching out from the PLL represents the clock tree. Buses not shown. Colored areas show some of the more important cores, while grayed areas represent other cores (UART, USB, touchscreen, etc.) APM is the advanced power management core; which is always powered. Every component in this figure has only illustrative and qualitative meaning.

An SOC is composed of several cores which each carries out a specific function. Some of the key cores we shall reference in the following are the Advanced Power Management (APM) core, the on-chip system oscillator (OSC) core, the Phase-Locked Loop (PLL) core, and the CPU core. The remaining cores are collectively referred to as "peripheral" cores. We shall also refer to the D- and I-caches (denoted as Caches in the figure) and which are part of the CPU core. Finally, we will refer to various parts of the clock distribution tree (also shown in the figure). The main issue with the clock tree is that the clock frequency may be different from core to core, as illustrated in Figure 1. Most importantly, the CPU usually runs at a faster clock frequency than any of the peripheral cores. The PLL generates all of the high-speed clocks to the CPU core, to the peripheral cores and to the local SOC bus. The APM employs its own low-speed high-precision oscillator and typically runs at 32 kHz. The oscillator is used for maintaining the real-time clock (RTC), the local RTC timer and to keep the Interrupt Controller Unit operational. The latter enables devices such as the RTC timer and external devices to wake-up the SOC.

Advanced SOCs may have all or some of the power states listed in Table 1. The way these low power states are implemented, and what they actually do when they are toggled, differs somewhat from processor to processor.

In the *Idle* state the clock to the CPU core is stopped, but other peripheral cores remain clocked. Basically all processors have this state, which is also sometimes referred to as the *halt* state. Most advanced SOCs also have a global *ClockSuspend* (*CS*) state in which the clock is globally stopped to all cores. The only core that remains active is the APM unit. The logical state in the cores is preserved. Some processors optionally allow the programmer to keep the OSC running. The benefit of this is that only the PLL lock-in time delay is experienced upon exiting this state (which may range from 50-200 us.) If the OSC is stopped, it takes significantly longer to exit this state, since it takes somewhere between 1-10 ms for today's high-speed OSC to stabilize, assuming the OSC uses a crystal for reference. An older processor such as EP7312 [13] may take up to 250 ms to exit out of the *CS* state.

Power State	Clock CPU/Periph	Power [mW]	Transition time & energy [ms], [uJ]
Idle	Off/On	>5	0, 0
ClockSuspend	Off/Off	0.25-5	>0.1, >1
PowerDown	Off/Off	0.05-0.2	>2, >40

Table 1. Definition of processor low power states and characteristics associated with each state. The "Clock" column indicates whether the clock is On or Off in the CPU core and in the Peripheral cores. The "Power" column indicates minimum SOC power level in the low power state. The "Transition" column indicates minimum (time,energy) required to enter and exit the low power state. The values are representative of advanced 32-bit SOC mobile processors and assumes the system crystal oscillator is turned off in the *PD* state (but is on in the *CS* state.) Data from [3-14] were used to populate this table.

Finally, some processors [3,4,6,8] have a *PowerDown* (*PD*) state in which power is removed from all the cores, including the OSC and PLL. Only the APM unit remains active which is achieved by putting this core on a so-called voltage-island on the SOC and powering it off a separate voltage supply (see for example [3]). One of the drawbacks with exploiting this state is that all SOC logical state and processor cache are lost. Therefore SOC state and D-cache would have to be saved before entering, and SOC state restored upon exiting, the *PD* state. Saving and restoring this SOC state requires additional energy and latency. In addition, as it was the case for the *CS* state, the

OSC stabilization time will add to the exit latency out of the *PD* state. An older processor like SA1110 [6] may take up to 160 ms to exit this state.

3. Analysis of Exploiting the PowerDown State: A Qualitative System Perspective

In the following we shall assume that the global SOC logical state is saved before entering the *PowerDown* (*PD*) state, and then restored on exiting the *PD* state. This will enable the system to be fully responsive in a matter of milliseconds upon detection of a wake-up event. The procedure for transitioning into and out of the *PD* state is as follows:

- Transitioning into *PD* state:
  1. Set CPU frequency, f_{cpu}, to maximum frequency at minimum supply voltage, $V_{dd,\min}$, or to maximum SDRAM clock, f_{mem}, whichever is the smallest.
  2. Save SOC peripheral state to memory.
  3. Stop unnecessary peripheral clocks.
  4. Save CPU state to memory.
  5. Write out D-cache.
  6. Enter *PD* state.

- Transitioning out of *PD* state:
  7. Wait for OSC to stabilize.
  8. Enable PLL.
  9. Restore CPU and SDRAM clocks as above.
  10. Do not restore D-cache.
  11. Restore CPU and peripherals states.

Secondly, the following will be assumed:

- It takes NC_{state} CPU clock cycles to read each state register and save it in cache. Equally it takes NC_{state} cycles to restore each state register.
- $t_{state<->mem,soc} = NC_{state} \bullet S_{state}/4/f_{cpu}$ is the time it takes to save the state to memory, or to restore the state from memory, and during which time we assume the SOC is running all the time, i.e., it doesn't exploit any low power states, not even the CPU halt state. S_{state} is the size of the processor state register set (in bytes).
- SDRAM modules are clock gated when there's no pending requests. Unused modules are in self-refresh state. This will significantly reduce memory power consumption during the transition.
- DRAM controller can manage 4 memory modules individually.
- In step 2 and 11, DRAM is accessed at a rate of 1 while loading the power management code used for the saving and restoring steps, and which is independent of the size of the SOC state, S_{state}, to be saved and restored and which is further independent of the SOC state addresses. $t_{pmcode<-mem} = S_{pmcode}/4/f_{mem}$ is the time it takes to load the power management code. This time also represents the time during which SOC is in active mode, and it represents the accumulated time the memory is accessed.
- In steps 2, 4 and 11, we shall assume that it takes three memory accesses to save, or restore, each state register, i.e., two accesses to retrieve address locations and one access to save, or retrieve, the state register data. $t_{state<->mem,mem} = 3 \bullet S_{state}/4/f_{mem}$ is the accumulated time the memory is accessed in order to either save the state to memory, or to restore the state from memory.
- In step 5, DRAM is accessed at a rate of 1 while writing out the D-cache and TLB. $t_{cache->mem} = S_{cache}/4/f_{mem}$ is the time it takes to write out the D-cache to memory and where S_{cache} is the size of the D-cache (in bytes). This time also represents the time during which SOC is in active mode, and it represents the accumulated time the memory is accessed.
- DRAM is accessed predominantly in burst mode due to cache line fills and flushes. Cache lines are typically 8 or 16 32-bit words long.
- We shall ignore the time it takes to enter and exit the DRAM clock gate state.
- Transition power levels are the same in the both the entry and exit transition periods.
- 32-bit architecture with separate data and addresses buses.
- A bulk decoupling capacitor of 10 uF is positioned in close proximity to the SOC chip to filter out any high-frequency noise on the power supply. A total of $E_{cap} = \frac{1}{2} \bullet 10\text{uF} \bullet V_{dd}^2 = 5$ uJ is stored in this capacitor assuming $V_{dd} = 1$ V.

Using the above assumptions and procedures, the total *PD* transition energy may be expressed as shown in Equation 1 and the total transition time may be expressed as shown in Equation 2. In Equation 1, $P_{soc,active}$ is the total SOC power @ f_{cpu} in the active state, including APM, OSC, PLL, leakage and I/O powers. $P_{mem,burst}$ is the memory power at the burst frequency f_{mem} and which includes the power required to drive the memory bus. $P_{soc,pm,CS}$ is the SOC power consumption in the *ClockSuspend* (*CS*) state (including leakage power and I/O power but excluding OSC and PLL active powers). $P_{mem,CG}$ is the memory power consumption during non-accesses when memory is clock gated by the DRAM controller. b_{osc} is a boolean variable which equals zero if the OSC is off while in the *PD* state, and which equals one if the OSC keeps running. b_{soc} is a boolean variable which

equals zero if the SOC is powered off during OSC stabilization (i.e., OSC either sits on its own voltage-island or it is part of the APM core). b_{soc} equals one if SOC and OSC are powered up simultaneously. In the case of $b_{soc} = 0$ (SOC off), we should really instead include the power consumption in the *PD* state, $P_{soc,pm,PD}$. But we ignore it since it will be very small as shown in Table 1.

Explanation of Equations 1 and 2

The eight lines of expressions in Equation 1 account for the following energies and actions. Note that the terms "processor" and "SOC" are equivalent.

Line 1: Energy associated with loading the transition code and saving all processor state to memory. Both processor and memory are in their *active* states.
Line 2: Energy associated with saving all processor state but accounting for the idling memory energy, i.e., the target memory module is clock gated while the processor is retrieving individual state registers since it takes many more CPU cycles to fetch a state register than it does saving it to memory.
Line 3: Energy associated with writing out the D-cache to memory. Both processor and memory are in their *active* states.
Line 4: Energy associated with the discharging of the 10 uF SOC decoupling capacitor every time the *PD* state is exploited.
Line 5: Energy associated with stabilizing OSC and PLL on transitioning out of the *PD* state. During the OSC stabilization period the relevant processor power is the sum of the OSC power and the leakage+I/O power (which is the power in the *CS* state.) The PLL is not turned on until after the OSC has stabilized which is why the PLL power only appears in the second term which accounts for the power consumption during PLL stabilization. Note that in the case where $b_{osc} = 0$ (OSC off), we ignore the SOC power consumption in the *PD* state since it is an insignificant contribution. Memory power is dominated by the self-refresh power.
Line 6: Energy associated with restoring all processor state from memory. Both processor and memory are in their *active* states.
Line 7: Energy associated with restoring all processor state but accounting for the idling memory energy, i.e., the target memory module is clock gated while the processor is restoring individual state registers since it takes many more CPU cycles to restore a state register than it does fetching it from memory.
Line 8: Energy associated with all other system energy contributions during the saving and restoring process which are not related to the processor and memory. More specifically, the background "offset" power consumption, $P_{sys,offset}$, includes display power, audio power, the power loss in the power supply, power loss in board bias circuits, and other system components.

In Equation 2, the first line in the last expression accounts for the time it takes to enter the state while the second line accounts for the time it takes to exit the state.

4. Results

For calculation of $E_{trans,PD}$ and $t_{trans,PD}$ we picked the following system parameter values:

- Minimum SOC voltage: $V_{dd,\min} = 1.0$ V.
- Processor process technology: 0.13 um.
- Temperature: 25C.
- Max CPU frequency at $V_{dd,\min}$: $f_{cpu} = 100$ MHz.
- Processor power: $P_{soc,active} = 25$ mW@ f_{cpu}.
- Size of transition code: $S_{pmcode} = 2$ KB.
- Size of data cache: $S_{cache} = 16$ KB.

$$
\begin{aligned}
E_{trans,PD} = {} & P_{soc,active} \bullet (t_{pmcode<-mem} + t_{state<->mem,soc}) + P_{mem,burst} \bullet (t_{pmcode<-mem} + t_{state<->mem,mem}) + \\
& P_{mem,CG} \bullet (t_{state<->mem,soc} - t_{pmcode<-mem} - t_{state<->mem,mem}) + \\
& P_{soc,active} \bullet t_{cache->mem} + P_{mem,burst} \bullet t_{cache->mem} + \\
& E_{cap} + \\
& (1 - b_{osc}) \bullet (P_{osc} + b_{soc} \bullet P_{soc,pm,CS} + P_{mem,SR}) \bullet t_{osc} + (P_{osc} + P_{pll} + P_{soc,pm,CS} + P_{mem,SR}) \bullet t_{pll} + \\
& P_{soc,active} \bullet (t_{pmcode<-mem} + t_{state<->mem,soc}) + P_{mem,burst} \bullet (t_{pmcode<-mem} + t_{state<->mem,mem}) + \\
& P_{mem,CG} \bullet (t_{state<->mem,soc} - t_{pmcode<-mem} - t_{state<->mem,mem}) + \\
& (P_{sys,offset} + 3 \bullet P_{mem,SR}) \bullet t_{trans,PD}. \qquad \text{(Eq. 1)}
\end{aligned}
$$

$$
\begin{aligned}
t_{trans,PD} = {} & t_{trans,PD,enter} + t_{trans,PD,exit} \\
= {} & t_{pmcode<-mem} + t_{state<->mem,soc} + t_{cache->mem} + \\
& (1 - b_{osc}) \bullet t_{osc} + t_{pll} + t_{pmcode<-mem} + t_{state<->mem,soc}. \qquad \text{(Eq. 2)}
\end{aligned}
$$

- Size of state: $S_{state} \in [0; 4000]$ Bytes.
- Number of CPU cycles per state: $NC_{state} \in [5; 25]$.
- Memory configuration: Four 16MB modules, 1.8V, 32-bit, 100MHz SDRAM [16].
- Memory bus frequency: $f_{mem} = 100$ MHz.
- Memory power per module in burst mode: $P_{mem,burst} = 162$ mW @ f_{mem}.
- Memory power per module when clock gated: $P_{mem,CG} = 0.6$ mW (includes 100 ns auto-refresh every 15.6 us.) Note, that the clock gated state is denoted PowerDown in the data sheet (though the DRAM is not powered down.)
- Memory power per module in self-refresh: $P_{mem,SR} = 0.3$ mW.
- OSC power and stabilization time: $P_{osc} = 0.5$ mW and $t_{osc} = 3.0$ ms @. $f_{osc} = 12$ MHz.
- PLL power and lock-in time: $P_{pll} = 4.0$ mW and $t_{pll} = 100$ us.
- $P_{soc,pm,CS} = 3.0$ mW.
- System offset power: $P_{sys,offset} = 5.0$ mW.

Figure 2 shows the transition time, $t_{trans,PD}$, as function of the size of the processor state, S_{state}, and with NC_{state} as parameter. The figure excludes the contribution from the OSC stabilization. For small state sizes, the main contributor to the total transition time is the PLL lock-in time of 100 us. The saving of the 16 KB D-cache adds another constant contribution of 41 us. The loading of the power management code takes 10 us. At very large state sizes and very large number of cycles per state sizes, the saving and restoring of the state registers become the main transition time contributors.

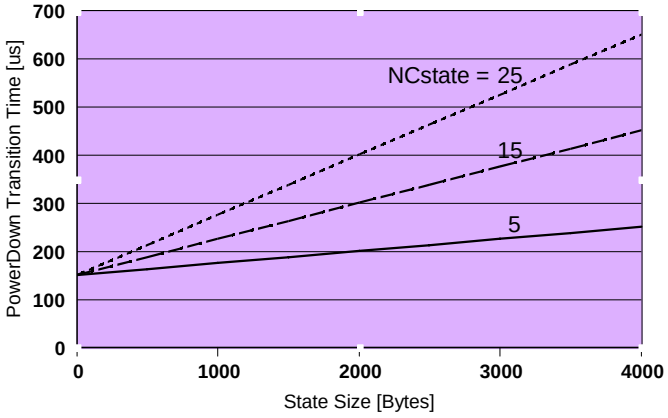

Figure 2. Transition time, $t_{trans,PD}$, as function of the size of the SOC state register set, S_{state}, and with the number of cycles per state register, NC_{state}, as parameter. The OSC is assumed to be running ($b_{osc} = 1$) while in the *PD* state. If the OSC is turned off, the OSC stabilization time would add a latency of 3,000 us.

Figure 3 shows the transition energy, $E_{trans,PD}$, as function of the size of the processor state, S_{state}, and with NC_{state} as parameter. Three cases are shown. The bottom set of curves (case 1) account for OSC running while in the *PD* state. For small state sizes, the main contributor to the total transition energy is the memory energy associated with saving the 16 KB cache into memory. As the state size grows and/or the number of cycles per state grows, so does the energy; which is true for the other two cases as well. At very large state sizes and very large number of cycles per state, the saving and restoring of the state registers become a very significant energy contributor and which is roughly equally split between processor and memory contributions.

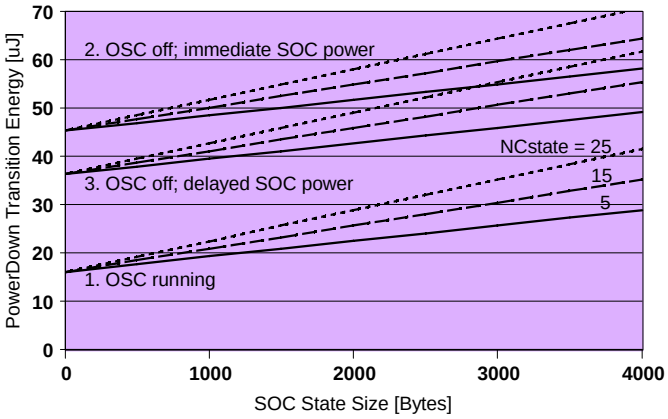

Figure 3. Transition energy, $E_{trans,PD}$, as function of the size of the SOC state register set, S_{state}, and with the number of cycles per state register, NC_{state}, as parameter. The OSC is assumed to be running ($b_{osc} = 1$) while in the *PD* state. If the OSC is turned off, an additional 34.5 uJ is incurred while waiting for OSC to stabilize.

The top set of curves (case 2) account for the more realistic case where OSC is turned off and power is applied immediately to the SOC on exiting out of the *PD* state. The exit transition delay, however, is now increased by the 3.0 ms OSC stabilization time. And as a result the transition energy increases by 29 uJ with respect to OSC running due to the energy consumed by background power, SOC leakage and memory self-refresh during the stabilization period. If possible, by delaying applying power to the SOC core until OSC has stabilized, we can at least eliminate the energy contribution from the leakage current during stabilization. This case is shown in the middle set of curves (case 3) where we gain about 9 uJ. Note that at higher temperatures and next generation SOCs, where the leakage will be much larger, the gain will be even more pronounced. For example, at 55C the SOC leakage power will increase to 12 mW and cause the curves in case 2 (immediate SOC power) to increase by about 28 uJ while the other two cases only experience a 1 uJ increase. Obviously, it would be desirable if power is applied to the SOC after the OSC has

stabilized, which would eliminate the 3ms•12mW = 36 uJ contribution from the leakage current (at 55C.)

It should be noted that the case with the lowest transition energy (i.e. case 1, OSC running) does not represent the best configuration for saving power, as one have to consider the energy consumption "inside" the *PD* state as well. In fact, the most efficient configuration is case 3 since this is the case where the OSC is turned off for the longest possible time. All other power contributions remain the same. The main reason for the discrepancy between case 1 and 3 is due to the contribution from the background power during stabilization. However, the background power is always present, so in case 1 we experience the background for a correpondingly longer time "inside" the *PD* state.

The transition energy is smaller than 71 uJ for the range of parameters shown in the figure. Even if we assume a system equipped with a CPU that is twice as inefficient (50 mW), and which has OSC and PLL powers that were twice as big (1 mW and 8 mW), and a twice as big background system offset power (10 mW), the worst case transition energy would remain smaller than 100 uJ in all considered cases. Further, $E_{trans,PD}$ @ $S_{state} = 0$ would only grow by 3 uJ. Transition times are unaffected by any of these changes.

Going back to case 1 (OSC running), other less significant contributions to the transition energy stem from the "constant" power consumers OSC, PLL and SOC leakage and which amount to 0.1-0.3 nJ, 0.6-2.6 uJ and 0.5-2.0 uJ, respectively, across the range of the parameters shown in Figure 3. Furthermore, the background system offset energy range is 0.8-3.3 uJ and the memory self-refresh energy range is 0.1-0.6 uJ. All these energies, however, would increase significantly if the CPU and memory were operated at for example 10 MHz instead of 100 MHz. Using Equation A.1 from Appendix A for calculating $P_{soc,active}$ at 10 MHz, we find that the transition energy increases to 51 uJ @ S_{state} = 0, to 77 uJ @ (S_{state}, NC_{state}) = (4000, 5) and to 140 uJ @ (S_{state}, NC_{state}) = (4000, 25) while transition times increase to 612 ns, 1612 ns and 5612 ns, respectively (see Figure 2 for comparison.) The only reason for the increase in the transition energy is due to the accumulated power consumption from the "constant" power consumers mentioned above since these consumers are now burning power over a considerably longer period. So the optimal CPU frequency should be as high as possible at $V_{dd,\min}$, but not greater than the maximum memory clock, while the memory clock should equal the CPU clock.

So how big is the transition energy compared to the energy consumed in the *PD* state? To determine this we will assume that the *PD* state is toggled between system timer interrupts, that the timer interrupt interval is 10 ms, and that the duration of the timer interrupt service routine can be ignored. Further, consider the case where the OSC is stopped, NC_{state} = 15, S_{state} = 2000 and that $P_{soc,pm,PD}$ can be ignored. From Figure 2 we get $t_{trans,PD}$ = 3.3 ms and from Figure 3 we get $E_{trans,PD}$ = 55 uJ. The later means that only 6.7 ms is spent in the *PD* state wherefore $E_{pm,PD}$ = $(4 \bullet P_{mem,SR} + P_{sys,offset}) \bullet$ 6.7ms = 60 uJ. Thus, the transition energy accounts for 48% of the total energy spent by toggling the *PD* state. For even more efficient systems, such as small form factor systems with less memory and $P_{sys,offset}$ < 5 mW, the transition energy will dominate the average power consumption even more. As we shall see in the next section, the sheer size of the transition energy can actually render the *CS* state are more efficient state to use.

5. ClockSuspend versus PowerDown

From a system perspective, whenever the system is idling, i.e., when there is no computational work to be done, the OS has to decide which of the two low power states, *CS* or *PD*, to exploit. In this section we shall discuss how this decision may be made by the OS.

Given the choice of the two low power states, *PD* should be exploited over *CS* if the total energy consumption associated with exploiting *PD* is smaller than the total energy consumption associated with exploiting CS. This condition may be expressed as

$$E_{trans,PD} + E_{pm,PD} < E_{trans,CS} + E_{pm,CS} \qquad \text{(Eq. 3)}$$

where

$$E_{trans,CS} = (P_{osc} + P_{soc,pm,CS} + 4 \bullet P_{mem,SR} + P_{sys,offset}) \bullet t_{trans,CS} + P_{pll} \bullet t_{pll} \qquad \text{(Eq. 4)}$$

is the system transition energy associated with transitioning out of the *CS* state. Transitioning into *CS* only takes a few clock cycles, and may be ignored. As in the *PD* case, we again take into consideration the OSC stabilization time. To carry over the notation from the previous chapter, we shall refer to the total memory power in self-refresh mode as $4 \bullet P_{mem,SR}$. $E_{pm,PD}$ and $E_{pm,CS}$ are the energies consumed while in the *PD* and *CS* low power states, respectively, i.e., in between the entry and exit transition periods. These values may be calculated as follows

$$E_{pm,PD} = (P_{soc,pm,PD} + b_{osc} \bullet P_{osc} + 4 \bullet P_{mem,SR} + P_{sys,offset}) \bullet (t_{idle} - t_{trans,PD}) \qquad \text{(Eq. 5)}$$

$$E_{pm,CS} = (P_{soc,pm,CS} + P_{osc} + 4 \bullet P_{mem,SR} + P_{sys,offset}) \bullet (t_{idle} - t_{trans,CS}) \qquad \text{(Eq. 6)}$$

where t_{idle} is the time the processor will spend idling and thus can be put into a low power state. Basically, this period

will equal the system timer interrupt interval. Equation 5 can represent the case where the OSC keeps running all the time ($b_{osc} = 1$) as well as the case where OSC is turned off ($b_{osc} = 0$) while in the *PD* state. This is to account for the fact that in some processors there may be no other choice but to turn off the OSC, while in other processors it may be an option. In Equation 6 we have assumed that the OSC is always running while in the *CS* state. We make this assumption for two reasons. Firstly, the OSC would not add significantly to the overall system power consumption in this state. Secondly, the transition time is significantly reduced if the system doesn't have to wait for the OSC to stabilize. Thus, the *CS* transition time is simply equal to the PLL lock-in time, i.e., $t_{trans,CS} = t_{pll}$. Note that both of the above reasons may hold true for the *PD* state as well, though the first reason is less likely to be true due to the much lower power consumption in the *PD* state than in the *CS* state.

Combining now Equations 3-6, we can calculate the idle times which satisfy Equation 3 as

$$t_{idle} > (E_{trans,PD} - E_{trans,CS} - (4 \bullet P_{mem,SR} + P_{sys,offset}) \bullet (t_{trans,PD} - t_{trans,CS}) - (P_{soc,pm,PD} + b_{osc} \bullet P_{osc}) \bullet t_{trans,PD} + (P_{soc,pm,CS} + P_{osc}) \bullet t_{trans,CS}) / (P_{soc,pm,CS} - P_{soc,pm,PD} + (1 - b_{osc}) \bullet P_{osc}) \quad \text{(Eq. 7)}$$

Figure 4 shows the idle time contours versus the SOC *CS* power. The contours represent the minimum idle time at which it become more economical to exploit the *PD* state than the *CS* state. The figure shows that the smaller the *CS* power level is, the longer a time must be spend in the *PD* state to offset the *PD* transition energy. The figure also shows that the most efficient operating point is to turn the OSC off and to apply power to the SOC after the OSC has stabilized ($b_{soc} = 0$). This is because the energy consumed in the *PD* state will more than compensate for the increase in the *PD* transition energy as compared to keeping the OSC running. In systems that have small *CS* power level, it is particularly beneficial to be able to turn the OSC off.

In operating systems such as Windows and Linux, the system timer interrupt frequency is 100 Hz, or 10 ms. In such systems, it will only be economical to exploit the *PD* state if the *CS* power level is greater than about 4 mW. This is a very interesting result as it shows how significant the *PD* transition energy really is, namely that it may be significant enough to make the *PD* state useless from the perspective of toggling the *PD* state between timer ticks. It should be noted that even if taking into consideration other values of NC_{state} (e.g., =5 and =25), that this only slightly changes the intersection point with the 10 ms idle time. The value of S_{state} has a somewhat stronger effect on the intersection point, as one would also expect from Figure 3 due to the stronger influence on the transition energy from S_{state}. Also notice that the intersection point with the 10 ms idle time is not significantly dependent on whether OSC keeps running, or whether OSC is shut down and power is applied simultaneously to OSC and SOC ($b_{soc} = 1$) on exiting the *PD* state or SOC power is applied after OSC stabilization ($b_{soc} = 0$).

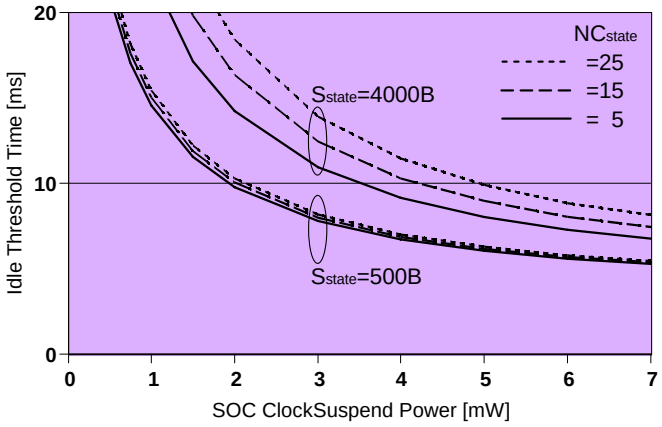

Figure 4. Minimum idle time at which it is equally efficient to exploit the *PD* state or *CS* state as a function of the power level in the *CS* state. The values from Chapter 4 are used, and NC_{state} = {5,15,25}, S_{state} = {500,4000} and $P_{soc,pm,PD}$ = 100 uW. Case shown for OSC off while in the *PD* state and simultaneous power applied to OSC and SOC during stabilization (i.e., b_{osc} = 0 and b_{soc} = 1).

6. Summary and Discussion

We have presented a qualitative analysis of the transition energy and latency associated with dynamically exploiting low power states between OS timer interrupts. Specifically, the analysis has centered around exploiting the two most power efficient low power states of advanced SOCs, namely the *PD* and the *CS* low power states. The analysis assumes that the whole system is already in a optimized low power state (e.g., display is off, audio device is off, etc) so that only the SOC state needs to be saved and restored in order to exploit the *PD* state.

One of most important results of the analysis is that the average power consumption associated with the *PD* state is significantly influenced by the *PD* transition energy, and may even dominate the power consumption This is particularly true for small form factor systems with small amounts of memory (< 64 MB) and which have small background power offsets, i.e., $P_{sys,offset}$ < 5 mW. In fact, it was found that the transition energy may be so large that it could render it more efficient to use the *CS* state than to use the *PD* state.

Eliminating transition energy by skipping OS timer ticks.

Another important result of the analysis is that which one of the low power states, *PD* or *CS*, that is the most energy efficient mainly depends on the SOC power consumption in the *CS* state, $P_{soc,pm,CS}$, and on the OS timer interrupt interval. For a given system, $P_{soc,pm,CS}$ is a constant. But the timer interrupt interval is not necessarily fixed in the sense that it can be configured to a longer interval at boot time. Though at the expense of reduced responsiveness with respect to handling multiple tasks. Figure 4 shows that the interval between OS timer interrupts may indeed render it inefficient to use the *PD* state between timer interrupts as compared to the *CS* state. This is because the interrupt interval limits how much time can be spent in the *PD* state. In other words, the energy in the *PD* state is not sufficiently smaller than the energy in the *CS* state to compensate for the *PD* transition energy. In an OS that only generates timer interrupts when there is real work to be done, such as in the Windows CE OS [17] and in the IBM Linux watch [18], the interval between adjacent timer interrupts will generally be much larger than the regular periodic timer interrupt interval of 10 ms. We denote the timing scheme in these systems as a Work Dependent Timing (WDT) scheme. WDT systems eliminate, or skip, workless timer interrupts. Thus, since the *PD* transition energy only adds a contribution to the total energy consumption every time a timer interrupt occurs, the overall energy contribution from the *PD* transition energy can be almost eliminated. So not only does a WDT based system make it obvious to exploit the *PD* state over the *CS* state, it also turns the *PD* state into an even more efficient low power state where the average power consumption is dominated by the power consumption in the *PD* state, $P_{pm,PD}$. This is in contrast to the conventional periodic timing mechanism where the transition energy may dominate the average power consumption.

It should also be noted that a WDT scheme would also enable turning off other system components between timer interrupts, such as the power supply, as the extended period between work related timer interrupts can probably absorb the larger latency associated with turning on these other system components. Of course, there may be other negative side effects of large turn on latencies, such as impact on user experience. Also, certain hardware inputs, such as a quick tap on a touchscreen, may be shorter than the wake-up time. Thus, the tap may wake up the system, but the system won't have a chance to determine the (x,y) coordinates (assuming edge-triggered interrupt detection and integrated touchscreen controller.)

Power supply decoupling capacitor.

In the analysis above we made the assumption of a 10 uF decoupling capacitor positioned next to the SOC. Generally, the total size of the power supply decoupling capacitor is much larger, perhaps 100 uF or more, to filter out a broader frequency band. However, we believe it's possible to split the capacitor into two components, namely, a 90 uF or more, main capacitor positioned at the output of the power supply, and a 10 uF capacitor positioned as close to the SOC as possible to filter out any additional high-frequency noise coupling into the power supply line between the two capacitors. The advantage of such a design is that the the main capacitor, in principle, can be gated off, at the board level, with a good low-resistance power FET (this would be done upon entering the *PD* state). In this fashion, only the energy stored in the 10 uF capacitor is lost while in the *PD* state. If a single (and ungated) 100 uF capacitor is used instead, a total of 50 uJ would be lost every time the *PD* state is exploited (assuming enough time is spent in the state to fully discharge the capacitor, e.g., about 33 ms for a leakage current of 3 mA.). As may be seen from Figure 3, a 50 uJ discharge energy would dominate the total *PD* transition energy. The drawbacks to this approach is additional wiring to enable the SOC to control the FET, additional board space to hold the FET and an extra capacitor, and additional cost of the FET. A more radical solution to this power loss is to design the SOC for total on-chip power gating in the *PD* state.

To keep OSC running or not.

It may be possible to separately control the state of the OSC, for example by incorporating it into the APM core as in [8], or by putting it on a voltage-island. In either case, it becomes possible to either keep it running while in the *PD* state to enable extra fast transition out the state, and it becomes possible to delay applying power to the SOC core until OSC has stabilized to eliminate the potentially very large leakage power contribution. As long as the OSC power consumption remains small, this may not be an unreasonable thing to do especially in view of the benefits of eliminating the OSC stabilization time. The same argument can not be made for the PLL, since keeping the PLL running while powered down would probably increase the power consumption in the *PD* state by too much. Besides, the PLL lock-in time is nearly insignificant. So there is no incentive in keeping the PLL running while powered down.

Appendix: Calculating SOC Power at Other Frequencies.

What if we want to calculate the transition energy at some other frequency, f_{cpu}, but we only know how much power the SOC consumes at some reference frequency, $f_{cpu,ref}$? Here's how to calculate the active SOC power at other frequencies (but at the same voltage). First note that $P_{soc,active}$ in Equation 1 is the total SOC power at some frequency f_{cpu} in the active state. Aside from accounting

for the digital switching power, $P_{soc,active}$ also includes power contributions from OSC, PLL, APM, leakage and I/O bias powers. Let's assume APM, leakage and I/O bias powers can be included into the power consumption of the *CS* state. We can now express the total active SOC power as

$$P_{soc,active}(f_{cpu}) = P_{osc} + P_{pll} + P_{soc,pm,CS} + \frac{f_{cpu}}{f_{cpu,ref}} \bullet (P_{soc,active}(f_{cpu,ref}) - P_{osc} - P_{pll} - P_{soc,pm,CS})$$

Eq. (A.1)

Using this formula and the values from section 4, the SOC would consume 9.3 mW @ 10 MHz assuming 25 mW SOC power @ 100 MHz. As seen from Equation A.1, the SOC will always consume at least 7.5 mW due to OSC, PLL and leakage powers.

References

[1] S.M. Sze, "Semiconductor Devices: Physics and Technology," J. Wiley & Sons, 1985.

[2] N.S. Kim et al, "Leakage Current: Moore's Law Meets Static Power," IEEE Computer, Dec. 2003.

[3] K.J. Nowka et al, "A 32-Bit PowerPC System-on-a-Chip with Support for Dynanmic Voltage Scaling and Dynamic Frequency Scaling," IEEE J. Solid State Circuits, Vol.37, No.11, 2002.

[4] A. Artieri et al, "NOMADIK: Open Multimedia Platform for Next-Generation Mobile Devices," STMicroelectronics, TA305, Feb. 2003.

[5] NeoMagic, "MiMagic5: Multimedia-Enhanced Applications Processor: Product Brief," Aug. 2002.

[6] Intel, "Intel StrongARM SA-11100 Microprocessor for Portable Applications: Brief Datasheet," April 2000.

[7] Intel, "The Intel PXA250 Applications Processor: White Paper," Feb 2002.

[8] Intel, "Intel PXA250 and PXA210 Applications Processors: Developer's Manual," Feb 2002.

[9] Motorola, "DragonBall MC9328MXL Integrated Portable System Processor: Reference Manual," Feb. 2003.

[10] Hitachi, "SH7750 series: Hardware Manual," July 2002.

[11] NEC, "Vr4131: Preliminary User's Manual," March. 2002.

[12] Cirrus Logic, "EP7211: Data sheet," May 1999.

[13] Cirrus Logic, "EP7312: Data sheet," May 2002.

[14] M. Baron, "Cool Performance for Handhelds," Microprocessor Report, Aug. 2002.

[15] J. Blyler, "Understanding Low-Power Designs," Wireless Systems Design, June 2001.

[16] Micron, "256Mb: x32 Mobile SDRAM," MT48H8M32LF Advance Datasheet, 2003.

[17] http://msdn.microsoft.com/library. Search for "WinCE power management".

[18] N. Kamijoh et al, "Energy Trade-Offs In the IBM Wristwatch Computer," Proc. 5th ISWC-01, Oct. 2001.

[19] C.M. Olsen et al, "Analysis of Transition Energy and Latency of the PowerDown State in Advanced System-On-a-Chip Processors," IBM Research Report #RC22970, Nov. 2003.