Techniques for Reducing the Connected-Standby Energy Consumption of Mobile Devices

Jawad Haj-Yahya[1] Yanos Sazeides[2] Mohammed Alser[1] Efraim Rotem[3] Onur Mutlu[1]

[1]*ETH Zürich* [2]*University of Cyprus* [3]*Intel Corporation*

ABSTRACT

Modern mobile devices, such as smartphones, tablets, and laptops, are idle most of the time but they remain connected to communication channels even when idle. This operation mode is called connected-standby. To increase battery life in the connected-standby mode, a mobile device enters the *deepest-runtime-idle-power* state (DRIPS), which minimizes power consumption and retains fast wake-up capability. In this work, we identify three sources of energy inefficiency in modern DRIPS designs and introduce three techniques to reduce the power consumption of mobile devices in connected-standby. To our knowledge, this is the first work to explicitly focus on and improve the connected-standby power management of high-performance mobile devices, with evaluations on a real system.

We propose the *optimized-deepest-runtime-idle-power state* (ODRIPS), a mechanism that dynamically: 1) offloads the monitoring of wake-up events to low-power off-chip circuitry, which enables turning off all of the processor's clock sources, 2) offloads all of the processor's input/output functionality off-chip and power-gates the corresponding on-chip input/output functions, and 3) transfers the processor's context to a secure memory region inside DRAM, which eliminates the need to store the context using high-leakage on-chip SRAMs, thereby reducing leakage power.

We implement ODRIPS in Intel's Skylake client processor and its associated Sunrise-Point chipset. Our analysis of ODRIPS on a real system reveals that it reduces the platform average power consumption in connected-standby mode by 22%. We also identify an opportunity to further reduce platform power in ODRIPS by using emerging low-power non-volatile memory (instead of DRAM) to store the processor context.

Keywords

Power Management; Energy Efficiency; Connected Standby; Mobile Systems

1. INTRODUCTION

Mobile devices such as phones, tablets, and laptops, are idle the majority of the time [15, 9, 94, 2, 58, 20]. During idle periods, modern mobile devices operate at low power to increase battery life while remaining connected to a communication network for usability (e.g., for email notifications and phone calls). This operation mode is called *connected-standby* [32, 78, 31] and it is supported by modern operating systems (OS), such as in Microsoft's *Modern Standby* [59] and Apple's *Power Nap* [4] modes. In addition to the OS-supported modes for power savings, many modern processors support multiple idle power states known as C-states [31, 81, 92, 21, 37, 34, 82, 26]. C-states are numbered from 0 to n. $C0$ is referred to as the *Active state*, whereas the others are reduced-power states that are designated as Ci, where the larger the i, the deeper the power state. The processor consumes lower power in deeper power states, at the cost of increasingly higher entry/exit latencies to/from a deeper power state. Cn state (known as the deepest-runtime-idle-power state (DRIPS) [59]) provides the *lowest* runtime power, but it also requires the *highest* entry and exit latencies compared to all other states.

In the connected-standby mode, where the display panel is off, a modern mobile system (processor+chipset) spends most (e.g., $> 99\%$) of time in DRIPS, while the system performs kernel maintenance [57] for the rest of the time (e.g., $< 1\%$). The system exits DRIPS and enters the Active state, or simply "wakes up", upon receiving a wake-up event from either an internal timer or an external trigger through one of the inputs/outputs (IOs). Before entering DRIPS, the context (such as the configuration/status registers, firmware persistent data, and firmware patches) of various components is stored into always powered on (AON) SRAMs on the processor chip. These SRAMs are powered using an AON voltage supply that is distributed inside the processor and chipset dies.

Based on our evaluation of DRIPS implemented in a real Intel Haswell-ULT [86] mobile system, we identify three major sources of energy inefficiency. 1) The wake-up hardware (timer and crystal oscillator) consumes about 5% of the total system power consumption in DRIPS. The processor is normally optimized for high performance and hence clock sources for the wake-up hardware also operate at relatively high frequency ($24MHz$ or even $100MHz$ in some architectures [71, 33, 63, 21, 27]) to provide low exit latency from DRIPS. 2) Several IOs are AON IOs that, along with their clock circuitry, consume about 7% of the total system power consumption in DRIPS. 3) For a modern processor, the

context that needs to be saved in DRIPS can be on the order of tens to a few hundreds of kilobytes. Maintaining the context in on-chip SRAMs in DRIPS consumes significant leakage power that constitutes about 9% of the total system power consumption in DRIPS.

Our **goal** in this work is to significantly reduce the energy consumption of high-performance mobile devices in connected-standby mode by re-designing DRIPS. **To this end**, we introduce a new mechanism called *optimized-deepest-runtime-idle-power state* (*ODRIPS*) that includes three new techniques, each of which tackles one of the three sources of energy inefficiency we identify. Our new mechanism is based on three **key ideas**: 1) To save the power consumption due to the monitoring of wake-up hardware (timer and crystal oscillator), ODRIPS dynamically migrates the handling of the wake-up events to low-leakage always-on circuitry inside the chipset. This also enables turning off all clock sources on the processor side. 2) To turn-off all always-on IOs, ODRIPS dynamically hands over the responsibility of these IOs to low-power always-on circuitry inside the chipset. 3) To save the high leakage power of the on-chip SRAMs that store processor context, ODRIPS dynamically transfers the context into off-chip low-leakage memory (DRAM). Our techniques are micro-architectural. They *do not* require architectural or OS support and thus can be directly implemented in hardware and firmware.

We implement ODRIPS in Intel's Skylake [21] client processor family and its associated Sunrise-Point chipset [88]. Skylake-based products scale from a thermal design point (TDP) of $3.5W$ [40], for handheld devices, up to a TDP of $95W$ [39], for high-performance desktops. We find that our proposal is more critical for lower TDPs (e.g., $3.5W$ to $25W$), which target mobile devices, tablets, notebooks, and laptops, such as Microsoft Surface [87] and Apple MacBook Air [85]. Our evaluation on a real system reveals that ODRIPS reduces the platform average power consumption in connected-standby mode by 22%.

Our proposed techniques are general and hence applicable to most modern mobile system architectures. Mobile systems that do not have conventional chipsets can still exploit the power management integrated circuit (PMIC [75]) chip that exists in modern mobile SoCs [81, 50] to implement our proposed ideas.

The main **contributions** of this work are as follows:

- We introduce ODRIPS, the optimized-deepest-runtime-idle-power state, to improve the energy consumption of modern mobile devices in the connected-standby mode. We provide a detailed description of our three major ideas, hardware micro-architecture, power measurements, power-modeling, and complete system flow. To our knowledge, this is the *first* work to explicitly focus on and improve the connected-standby power management of high-performance mobile devices.
- Using real hardware measurements, we experimentally break down the total power consumption of the Intel Skylake client processor [21] and its chipset [88], to quantify the energy benefits of ODRIPS. Overall, our proposed three-pronged strategy reduces platform average power in connected-standby mode by 22%.
- We explore the effect of using Phase Change Memory (PCM) [90] (instead of DRAM) to store the processor context in ODRIPS. We find that doing so reduces the platform average power by an additional 14.5% (Sec. 8.3).

2. BACKGROUND

In this section, we 1) show how DRIPS is implemented in a modern mobile platform, 2) describe how the processor enters and exits DRIPS, and 3) explain the sources of power consumption in connected-standby.

2.1 Intel Skylake Mobile System Architecture

We present the high-level architecture of an Intel Skylake [21] mobile system that includes the processor and chipset in Fig. 1(a). The system also includes external voltage regulators, crystal oscillators (XTAL), and DRAM memory. When the system is in DRIPS, the power management unit (PMU ⑤) maintains a subset of the components (highlighted in green in Fig. 1(a)) powered using an AON supply. These components are required to remain fully functional even if the platform resides in DRIPS. This is essential for a reliable connected-standby mode that enables communication (e.g., email notifications and phone calls). The rest of the system components are completely turned off.

Our evaluation of a real mobile system in DRIPS reveals that the platform still consumes several tens of (e.g., ∼60) milliwatts with the processor consuming 18% of the total power (Fig. 1(b)). We describe our evaluation methodology in detail in Sec. 7.

Next, we discuss the steps required to enter and exit DRIPS.

2.2 DRIPS Entry and Exit Flows

DRIPS entry flow. Typically, after a few milliseconds of idleness, the compute-domains (i.e., cores and graphics) enter their deepest idle power-state, during which compute-domain context is saved into Cores/GFX Save/Restore SRAMs ⑧. After that, the PMU determines the next platform idle power-state based on status information reported by *latency tolerance reporting* (LTR) [1, 49, 31, 26] and *time to next timer event* (TNTE) [38]. LTR indicates how much latency the device can tolerate when accessing memory. TNTE reports whether a device is planning a wake-up event in the near future based on the internal timer.

When DRIPS is selected as the target state, the PMU orchestrates six key actions, in the following order: (1) flushing the last level cache (LLC) into DRAM, (2) turning off the compute-domain voltage-regulators, (3) storing the system agent[1] (SA) context into SA Save/Restore (S/R) SRAMs ⑦, (4) placing DRAM into self-refresh mode with the help of the CKE signal ⑥ to

[1]The system agent houses the traditional Northbridge. It contains several functionalities, such as the memory controller and the IO controllers [88].

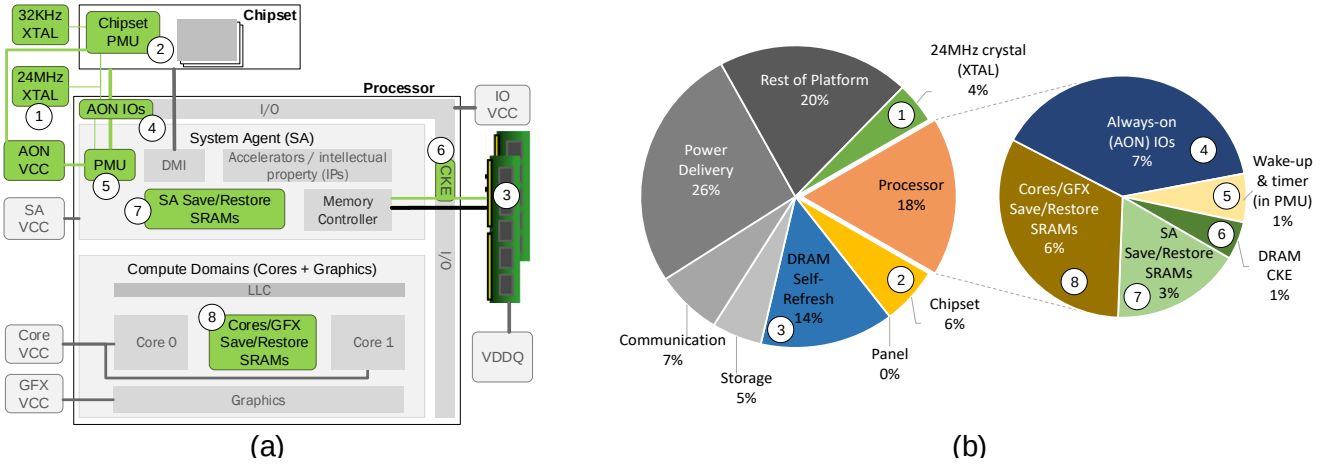

Figure 1: (a) High level architecture of an Intel Skylake [21] mobile system (processor + chipset). Powered-on system components in DRIPS are highlighted green (numbered) (b) Breakdown of platform power consumption in DRIPS. The total platform power consumption in DRIPS is ∼60 milliWatts measured at 30°C with 8GB DDR3L-1.6GHz.

avoid data loss due to excessive charge leakage from the capacitors (storage cells) of DRAM [17, 54], (5) turning off clock sources (except the clock sources for the internal timer and the AON IOs) and (6) turning off the voltage-regulator and then partially power-gating the PMU.

At this point, the platform is fully in DRIPS. As illustrated in Fig 1(a), in DRIPS the AON supply powers the AON IOs ④, the un-gated part of the PMU ⑤, the save/restore SRAMs (⑦ and ⑧), the CKE signals ⑥ (that drive DRAM self-refresh), and the always-on domains of the chipset ②. The wake-up timer and AON IOs are toggled and monitored periodically using the 24MHz clock ①.

DRIPS exit flow. The exit flow is simply the reverse process of the entry flow. The flow starts when the PMU detects a wake-up event. Next, the power is restored to the system by turning-on the voltage regulators, the context of the SA is restored from the S/R SRAMs, and the DRAM is allowed to exit the self-refresh mode. If the wake-up interrupt requires core handling, then the cores exit the low-power mode, during which their context is also restored from on-chip SRAMs. The graphics engine context is restored if the relevant driver requests so.

2.3 Average Power Consumption in the Connected-Standby Mode

Fig. 2 illustrates the connected-standby operation mode where the system periodically transitions between the Active state ($C0$) (for performing OS kernel maintenance tasks [58]) and *the Idle* state (DRIPS). On-demand transitions to the Active state occur in response to external triggers, such as user inputs, interrupts from networking devices, and other hardware events. The energy savings at a platform level are higher when the platform is idle most of the time (i.e., when it has higher DRIPS residency and lower $C0$ residency).

The average power consumption of the connected-standby mode is the sum of the power consumption times the weighted residency time of each state, including the *Entry* and *Exit* transition states, as shown in Equation 1.

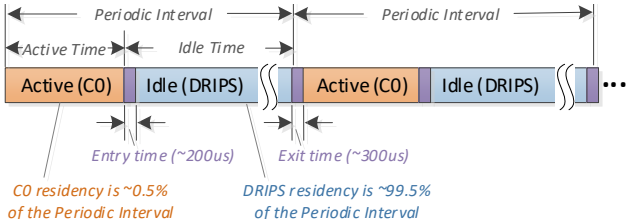

Figure 2: Average power of an Intel Skylake mobile system in the connected-standby mode, based on our evaluation. 99.5% of the time, the platform is idle (in DRIPS) and consumes tens of milliwatts (∼60mW). In the remaining 0.5% of the time, the platform is active (in C0 with display off) and consumes a few watts (∼3W).

$$\begin{aligned} Average\_Power = {} & C0\_power \times C0\_residency + \\ & DRIPS\_power \times DRIPS\_residency + \\ & Entry\_power \times Entry\_residency + \\ & Exit\_power \times Exit\_residency \end{aligned} \quad (1)$$

3. MOTIVATION: DRIPS ENERGY INEFFICIENCIES

This section makes three key observations about the energy inefficiencies of DRIPS in an Intel Skylake mobile system [21] through detailed timing and energy analyses. We explain our experimental methodology in detail in Sec. 7. We expect similar energy inefficiency issues to be present in other modern high-performance systems that employ DRIPS.

Observation 1. We show in Fig. 1(b) that the processor's internal timer and its wake-up event handling consume about 5% of the total power consumption in DRIPS. This power includes two key components: 1) wake-up monitoring and timer toggling ⑤ on the processor side and 2) the $24MHz$ crystal oscillator ① on the board. This non-negligible power contribution in DRIPS is due to two reasons. 1) The process technology of the processor is optimized for high performance rather than low power operation [14, 24]. 2) The internal timer is monitored/toggled at relatively high speed clocks ($24MHz$ or even $100MHz$ in other architectures [71, 33, 63, 21, 27]). The relatively high speed clock is mainly used to optimize the system for fast response within tens of *nanoseconds* when exiting shallow idle power states (not DRIPS).

However, very fast DRIPS exit/entry is an overkill since we observe that DRIPS can potentially afford *milliseconds* of exit latency. This is enabled mainly due to the relatively large buffering capabilities (e.g., buffering in the peripheral devices and buffering in the interconnects) that modern SoCs have to reduce platform average power consumption. For example, a modern SoC aggregates multiple interrupts and handles them together at the same time to reduce the number of wake-ups from the Idle state [34, 49]. This capability requires large buffers to store the data generated by peripheral devices, such as camera, network, or microphone.

This observation is consistent with those reported in prior work [18, 30, 65, 19, 31, 26]. In fact, Intel Haswell-ULT [86], a predecessor of Skylake that supports the connected-standby mode, has an exit latency of 3 milliseconds from DRIPS (called $C10$) [49]. The long exit latency of Intel Haswell-ULT guarantees enough time to re-initialize the voltage regulator. The voltage regulator re-initialization latency was optimized in the Skylake platform and reduced to few hundreds of microseconds.

Our observation provides the ability to bring energy savings by increasing the entry and exit latencies, by few tens of microseconds, without degrading user experience in the connected-standby mode. This can be realized by dynamically migrating the wake-up events (i.e., the internal timer, and wake-up IO) to a different chip that is optimized for low-power operation (i.e., with much lower leakage and clock frequency). Doing so can help to reduce the average power consumption of the platform in connected-standby mode.

Observation 2. We show in Fig. 1(b) that the processor's AON IOs ④ consume 7% of the total platform power in DRIPS. These IOs include 1) differential clock ($24MHz$) buffers, 2) two power management links (PML) between the processor and the chipset (one in each direction), 3) thermal reporting interface from the board, and 4) interfaces to the voltage regulator, reset, and debug circuitry. We observe that most of the functionality of these IOs is not required in DRIPS, especially if we migrate the internal and external wake-up events (i.e., timer ⑤ and IOs ④) to a different chip (as discussed in Observation 1).

The power-management interface between the chipset and the processor is required even in DRIPS to ensure that when the chipset detects a wake-up event, it can wake up the processor. However, the functionality of the other IOs can be potentially offloaded to the chipset and removed from the processor.

This observation provides us with the motivation to explore the opportunity to make the chipset the "hub" for hosting the wake-up events in DRIPS. Doing so enables turning-off (power-gating) the processor's AON IOs in DRIPS by using a control signal from chipset. When the chipset detects a a wake-up event, it turns on the AON IOs and starts the normal exit flow out of DRIPS. Doing so can help to reduce the platform average power consumption in connected-standby mode.

Observation 3. We show in Fig. 1 that the memories to save and restore (i.e., S/R SRAMs) processor context in the system-agent ⑦ and compute domains ⑧ consume a significant portion (9%) of the total power consumption of the platform in DRIPS. A high-end processor is normally fabricated with a process technology that is optimized for performance [14, 24], not power consumption. For example, we experimentally find that the leakage power of the SRAM in the processor is nearly *five* times that of equal-capacity SRAM[2] in the chipset. Our analysis shows that the processor S/R SRAMs draw high leakage power in DRIPS, even when their operating voltage is reduced to the minimum required for data retention.

We also observe that the DRAM power efficiency in storing data (i.e., power consumed per stored bit, in Watts/bit) is more than three orders of magnitude better than that of SRAM [12, 13, 23]. This provides us with the motivation to explore the use of off-chip DRAM (as opposed to on-chip SRAM) to save processor context in DRIPS.

One of the major concerns with storing processor context off-chip is security. In particular, processor context can be compromised, when it is transferred to DRAM, using one of the known attacks on the DRAM system [91, 84, 41, 61, 7, 73, 74, 35, 72, 47, 62]. Memory protection mechanisms, such as Intel SGX [55, 21, 92], can address this security concern by providing a protected memory region. Intel SGX implements memory encryption and authentication at the hardware level, which makes it more difficult to perform attacks on external memory. One of the main protection mechanism in SGX is a memory encryption engine (MEE) [55, 28], which encrypts and authenticates the contents of DRAM. SGX technology operates under the assumption that the security perimeter includes only the internals of the processor package, and in particular, leaves the DRAM untrusted [28, 55, 16]. Therefore, SGX's MEE provides confidentiality, integrity, and freshness guarantees to the processor context while it is stored in DRAM.

The above motivates us to explore the feasibility of dynamically transferring the processor context to a dedicated memory region within the SGX-protected DRAM memory during the DRIPS entry flow and restoring the processor context back from this region during the

[2]Both SRAMs operate at the minimum possible voltage (V_{min}) of the associated process technology.

DRIPS exit flow. Although our work focuses on using Intel SGX, we note that other architectures use similar memory protection mechanisms (e.g., IBM's SecureBlue++ [6] and AMD SEV [44]). Our general approach can be adopted by these architectures.

Based on these three observations, we propose *optimized DRIPS*, *ODRIPS*, a state that includes our three new techniques that improve the energy consumption of a modern system in the connected-standby mode. In the next three sections, we discuss in detail a possible way to implement each of our three techniques. Our description reflects, to a great extent, the actual implementation of ODRIPS in the Intel Skylake mobile processor [21] and the corresponding Sunrise-Point chipset [88].

4. TIMER WAKE-UP EVENT HANDLING

This section explains how we address the energy inefficiency in handling the timer wake-up events in DRIPS. The key idea is to 1) dynamically migrate the timer event handling to the chipset and 2) toggle the timer with a drastically slower (i.e., by 730×) clock. Compared to baseline DRIPS, our technique results in lower power but longer entry and exit latencies. Our technique 1) saves the wake-up timer power ⑤ (inside the processor's PMU) and 2) enables turning off the $24MHz$ crystal oscillator ① in DRIPS. This reduces the baseline DRIPS power by 5%.

4.1 Timer-Event Migration to the Chipset

A timer event is an interrupt that occurs when the time-stamp-counter (TSC) of the system reaches a prescheduled target time. The timer event is normally scheduled by the OS, applications, or the PMU to perform a future task. During entry into DRIPS, the PMU firmware determines the closest timer-event, among all scheduled timer-events, and wakes up the system when the TSC reaches the event's target time.

4.1.1 Design Alternatives

In our baseline architecture, DRIPS operates with two clocks: 1) a real time clock (RTC) with a frequency of $32.768KHz$ (for simplicity we refer to it as the $32KHz$ clock) and 2) another clock with a frequency of $24MHz$ (which toggles the timer). Both clocks are generated using two off-chip crystal (XTAL) oscillators, as shown in Fig. 1(a). To reduce DRIPS power consumption, we propose to shut down the $24MHz$ XTAL oscillator and use the drastically slower $32KHz$ clock to toggle the system.

To achieve this goal, we have two main design alternatives. 1) Bringing the $32KHz$ XTAL oscillator into the processor (in addition to the chipset) and toggling the timer with it. This solution enables turning off the $24MHz$ XTAL oscillator, but it requires more IO pins to the processor such that the $32KHz$ clock signal is also delivered to the processor. Increasing the IO pins of the processor chip increases both chip cost[3] [36] and processor power consumption. 2) The second design alternative is to enable the chipset to perform timer event handling at lower clock frequency. The second design alternative does not require additional IO pins and does not increase the power consumption of the processor. We also find that the second design alternative facilitates the power-gating of the remaining always-on (AON) IOs in ODRIPS (as we discuss in Sec. 5). Due to these advantages, we select the second design alternative. Next, we explain its design in detail.

[3]Interfaces (pins) that connect the chip to other chips or board components are relatively expensive as also observed by the International Technology Roadmap for Semiconductors (ITRS) [36]. Please also see [53].

4.1.2 Microarchitectural Changes

We present the microarchitectural changes needed to realize our second design alternative in Fig. 3(a). We add two timers to the chipset: a fast-timer that works with a $24MHz$ clock and a slow-timer that toggles at $32KHz$.

When entering ODRIPS, the system copies the main timer value from the processor to the chipset's fast-timer. Subsequently, the fast-timer starts toggling using the $24MHz$ clock, incremented by *one* every cycle (i.e., $Fast\text{-}Timer{+}{=}1$). A few cycles later, the PMU switches to the $32KHz$ clock (asserts the Switch_to_32KHz signal). The flow waits until the rising edge of the $32KHz$ clock before carrying out the actual switching, as we show in Fig. 3(b). At the edge of the slow clock, the fast-timer value is copied into the slow-timer, and slow-timer starts toggling with the $32KHz$ clock (i.e., incremented by a predefined *Step* value every slow cycle; $Slow\text{-}Timer{+}{=}Step$). At this point, the $24MHz$ clock can be gated and the crystal oscillator can be turned-off. The *Step* value is determined via a calibration process discussed in Sec. 4.1.3.

When exiting from DRIPS, the platform switches back to the fast-timer. The chipset's PMU turns on the $24MHz$ oscillator and de-asserts the Switch_to_32KHz signal as we show in Fig. 3(b). The process waits for the rising edge of the $32KHz$ clock, and copies the timer value (upper 64 bits) into the fast-timer. The fast-timer then continues toggling at $24MHz$. Subsequently, the value of the fast-timer is copied to the main timer inside the processor via the power management link (PML) channel as shown in Fig. 3(a) and the processor continues with the flow of exiting ODRIPS.

The PML is used for power management communication between the processor and the chipset. The PML has two physical master-slave interfaces (clocked with the $24MHz$ clock). The processor is the master for the interface from the processor to the chipset and the chipset is the master for the interface from the chipset to the processor. Consequently, the PML is a *deterministic* channel. For correct timing, whenever we transfer the timer value from/to the processor over the PML, we add a fixed constant to the transferred timer value to compensate for the time it takes to transfer the timer value on the communication channel.

4.1.3 Timer Precision and Step Calibration

The key challenge in switching the timer counting from the fast timer to the slow timer is to maintain the counting correctness of the fast timer before the system enters ODRIPS, during ODRIPS, and after the system

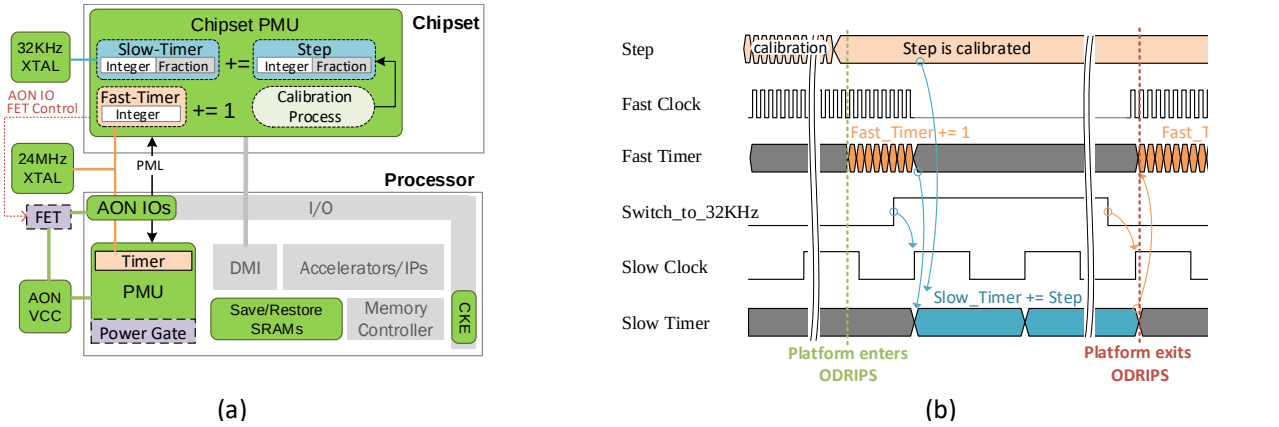

Figure 3: (a) Wake-up event handling and AON IO power-gating. New components are highlighted with dashed lines: Slow-Timer, Fast-Timer, on-board power-gate (i.e., FET) for AON IOs, processor's PMU power-gate. (b) Timer counting switches from Fast-Timer to Slow-Timer during ODRIPS entry to enable turning off the fast clock crystal oscillator (i.e., 24MHz-XTAL). During ODRIPS exit, timer counting switches back from Slow-Timer to Fast-Timer.

exits ODRIPS. While the system is not in ODRIPS, the fast timer is incremented by *one* every fast clock cycle (i.e., $Fast\_Timer$+=1). Since the fast clock signal is turned off in ODRIPS, the slow timer needs to keep the counting value of the fast timer updated, using the slow clock. To do so, at every slow clock cycle, the slow timer needs to be incremented by a *Step* value that represents the ratio between the fast and slow clock frequencies (i.e., $Slow\_Timer$+=$Step$). For example, if the slow clock frequency is *three* times slower than the fast frequency (i.e., $Fast\_Freq = 3 \times Slow\_Freq$) and the two clocks are aligned (e.g., their positive edges are aligned to each other), then at every slow clock cycle the slow timer should be incremented by *three*.

However, this counting mechanism has two main challenges. 1) As the slow and fast timers are clocked by two different clock sources (i.e., 32kHz-XTAL and 24MHz-XTAL, as shown in Fig. 3(a)), the edges of the clock signals are not necessarily aligned to each other. 2) The ratio between the frequencies of the fast clock and slow clock signals is not necessarily an integer value. Rounding the *Step* value to the nearest integer value can cause a *counting drift* (i.e., the discrepancy between the timers) over time. To overcome these two challenges, we need to 1) represent both the *Step* and the slow timer as a fixed-point numbers (i.e., integer and fractional parts) and 2) calibrate the fixed-point *Step* value at run time using the fast clock and slow clock signals.

***Step* Calibration.** The purpose of the calibration process is to calculate the value of the integer part and the fractional part of the fixed-point *Step* value. We assume that the *Step* value is represented as a fixed-point number with m-bit integer and f-bit fractional parts. We first describe the calibration step and then show how to calculate the number of bits (m and f) for each part. The calibration process includes two key steps. First, we count the number of clock cycles of the fast clock signal (N_fast) encountered within some clock cycles (N_slow) of the slow clock signal. Second, we divide N_fast by N_slow to calculate the average ratio between the fast and slow clock frequencies. These two steps give the value of the fixed-point *Step* value. The larger the N_slow value, the more accurate the average ratio between the two clock signals. To simplify the division implementation, we choose $N\_slow = 2^f$. In this way, we can divide N_fast by N_slow by simply placing the fixed point after the first f least significant bits of the *Step* number.

This calibration process lasts for several seconds. Its duration depends on the value of f and the slow clock frequency. However, this process needs to be carried out *only once* after each reset of the system.

We now calculate the values of m and f. The number of bits for the integer part of *Step*, m, should be enough to accommodate the ratio between the frequencies of the fast clock and slow clock signals. We calculate m as follows:

$$m = \lfloor \log_2 \frac{Fast\_Freq}{Slow\_Freq} \rfloor + 1 \quad (2)$$

The number of bits for the fractional part of *Step*, f, is determined by the desired precision level of the fast timer. A computing system typically requires a certain level of precision from clocks and timers. For example, a *one part per billion* (1 ppb) implies a maximum counting drift of three seconds over a century. For our system we maintain a 1 ppb precision.

To determine f, we calculate the counting drift, ε, i.e., the discrepancy between the actual value of the fast timer (N_fast) and the estimated value of the fast timer when using the slow clock (i.e., $\frac{Fast\_Freq}{Slow\_Freq} \cdot N\_slow$), as follows:

$$\varepsilon = N\_fast - \frac{Fast\_Freq}{Slow\_Freq} \cdot N\_slow \quad (3)$$

To achieve a 1 ppb precision, we should keep the value of ε *less than 1* within one billion (10^9) cycles of the fast clock signal (N_fast). Therefore, the number of cycles over which we should calibrate the *Step* is given by:

$$N\_slow = 2^f = \frac{(N\_fast - \varepsilon)}{\frac{Fast\_Freq}{Slow\_Freq}} > \frac{10^9 - 1}{\frac{24MHz}{32.768KHz}} \quad (4)$$

Thus, for this particular implementation with a 1 ppb precision, the slow timer needs to be incremented by a *Step* value that has $m = 10$ integer bits and $f = 21$ fractional bits.

4.2 Hardware and Power Cost of Wake-up Event Handling

Area overhead. Our microarchitectural changes require two new timers and a register added to the chipset: 1) 64-bit fast timer, 2) $(64 + 21)$-bit slow timer, and 3) $(10 + 21)$-bit register for storing the *Step* value. The area overhead of this additional hardware is negligible compared to the total chipset area (less than 0.01%).
Power overhead. The power consumption of the timers and the register is insignificant, less than 0.001% of the chipset power in DRIPS.

5. ALWAYS POWERED ON (AON) IO POWER GATING

We address the energy inefficiency due to the processor's AON IOs by power-gating (disabling) them in DRIPS. This helps us to save 7% of the platform power in DRIPS (portion ④ in Fig. 1(b)). As the process technology used for processor design is optimized for high performance rather than low power, most of the peripheral IOs (e.g., SPI, USB, SATA, communication, part of the PCI) are placed in the chipset rather than in the processor. In the previous section, we have explained how to migrate the timer wake-up event handling from the processor to the chipset in DRIPS. Such migration also facilitates the turning-off of the processor's AON IOs after offloading all the AON IO functionality to the chipset, as shown in Fig. 3(a).

5.1 Design Alternatives for IO Power-gating

Dynamically power-gating the processor's AON IOs requires 1) a power gate for the IOs and 2) controlling the power gate while the system enters or exits DRIPS. The controller should be in the chipset, as we want the chipset to be the "hub" for handling all wake-up events. There are two main design options to realize AON IO power-gating: 1) using an embedded power-gate (EPG) in the silicon die [93] to power gate the AON IOs or 2) using an external field-effect transistor (FET) [76] on the board to gate the power-rail of the AON IOs. The first alternative, EPG, is area-efficient and it does not require additional components on the board. The second alternative, FET, however, offers three key benefits: it 1) has less leakage compared to EPG, 2) does not require additional IO pins in the processor (which are an expensive resource), and 3) requires less design effort for the processor team. To obtain these benefits, we choose the second design alternative to implement the power-gating of the AON IOs.

5.2 DRIPS Flow Changes to Support AON IO Power-gating

After migrating the timer to the chipset (as we explain in Section 4.1), the processor hands over the responsibility of the AON IOs to the chipset. Consequently, the chipset disconnects the AON IOs by controlling the FET, as shown in Fig. 3(a). Doing so places the platform in ODRIPS while processor's IOs are power-gated.

The main processor's AON IOs include thermal reporting, power-management link (PML), reset, voltage regulator control serial interface, debug interface, and $24MHz$ clock buffers. The thermal reporting wake-up event arriving from an embedded controller (EC) is used to report thermal events. We offload this wake-up IO using the general purpose IOs (GPIOs) in the chipset to monitor the EC interface with the $32KHz$ clock signal inside the chipset's PMU. In DRIPS, PML IO is not in use. Once the chipset detects an event, the chipset can turn-on the AON IOs and send the event through the PML to the processor. The rest of the IOs are not used in DRIPS. The voltage-regulator control-serial-interface is used to control the voltage regulators of the compute units. Since compute domains are powered off in DRIPS, this interface is not required and it is used only when the system enters or exits DRIPS. The debug interface is also not used in DRIPS. After offloading all AON IO functionality to the chipset, the $24MHz$ clock source can be safely turned off.

5.3 Hardware and Power Cost of AON IO Power-gating

Power gating of the AON IOs requires a FET and two GPIOs, one for offloading the thermal event and another for controlling the FET. The FET imposes a minor area overhead to the board (less than 0.1% of the chipset area). The FET provides high isolation when it is turned off as its leakage power is less than 0.3% of the gated load's power. The chipset has a number of spare (unused) GPIOs. We use two of these spare GPIOs to facilitate IO power-gating. Hence, tackling the second source of energy inefficiency using our second proposed technique incurs insignificant area and power overheads.

6. PROCESSOR CONTEXT HANDLING

We tackle the third source of energy inefficiency due to the use of high-leakage SRAMs to store the processor context. In DRIPS, the processor context is retained inside on-chip SRAMs (⑦ and ⑧ in Fig. 1(a)), which account for 9% of the baseline platform's power consumption in DRIPS. The context SRAMs are powered with *retention voltage*, which is the lowest possible power supply voltage at which the data can be retained inside SRAM [31]. Thus, it is not feasible to further reduce SRAM voltage to save power in DRIPS.

6.1 Design Alternatives for Context Power Reduction

To save SRAM power in DRIPS, we consider three design alternatives. 1) Replacing SRAMs with embedded *non-volatile* memory (eNVM), such as eMRAM [77, 25]. This enables retaining the context inside eNVM while the voltage source is turned off. However, many emerging eNVMs still suffer from low endurance and high latency when compared to SRAM or DRAM [77, 25]. In addition, eNVMs currently introduce engineering complexities at design and fabrication times.
2) Storing the processor context in the SRAMs of the chipset instead of SRAMs of the processor. An SRAM in the chipset consumes approximately five times less leakage power compared to SRAM in the processor, due to different process technology optimizations in the chipset vs. the processor (i.e., power vs. performance [14, 24]). However, this alternative has three drawbacks: (i) we need to dedicate storage area (about $200KB$) for the processor context inside the chipset's SRAM, (ii) design complexity is non-negligible as we need to implement special flows in both the chipset and the processor to support the context save/restore operations, and (iii) SRAMs in the chipset still consume some power.
3) Storing the processor context in DRAM. This has two major advantages compared to the other two design alternatives: i) storing the context in DRAM theoretically consumes "zero" additional power, because in DRIPS the whole DRAM is anyway self-refreshed to retain all data, and 2) DRAM capacity is huge compared to the size of the processor context (GBs vs KBs). Therefore, reserving a small area of the DRAM for the context is feasible and inexpensive. However, processor context includes sensitive information and configuration state. This raises a security concern as several attacks [91, 84, 41, 61, 7, 73, 74, 35, 72, 62, 47], can be carried out on DRAM to reveal or change its contents. For this reason, memory protection architectures, such as Intel SGX [55, 21, 92], can be used to secure regions in DRAM systems.

Considering the benefits provided by the third alternative, we choose to store the processor context in off-chip DRAM. To our knowledge, this is the *first* work to propose to save the processor's sensitive context (including CSR, patches, fuses, etc.) outside the processor's chip, in DRAM. This is possible with the help of Intel SGX that helps to securely store the processor context in DRAM.

6.2 Context Transfer to Protected DRAM Area

We develop a new flow to dynamically transfer the processor context into a protected DRAM area, as shown in Fig. 4. At DRIPS entry, the PMU triggers two finite state machines (FSMs) that read data from the on-chip SRAMs and flush the read data into DRAM. The first FSM is located at the system-agent (SA FSM) and it handles the system agent context. The second FSM is located near the Last-Level-Cache (LLC FSM) and it is responsible for flushing the context of the cores and graphics into DRAM. Both context-flushing FSMs use a mechanism similar to the one that is already implemented in the baseline architecture for flushing the LLC into DRAM before turning off the LLC in a low-power state. The PMU firmware configures each FSM with the protected-memory *base-address* ($BaseAddr$) where the processor context is saved when the system enters DRIPS. In addition, a protected memory *range register* (Context/SGX RR) inside the memory-controller that determines if the memory access is to a protected memory region or to the rest of the memory. An access to a protected memory region is redirected to the memory encryption-engine (MEE) [28] before accessing the DRAM. MEE encrypts the data for writes (or decrypts for reads) and carries out the desired authentication [28]. The authentication process involves multiple accesses to the authentication tree metadata inside the DRAM. To alleviate performance overheads, the MEE is equipped with an internal "MEE cache" [28] that stores the metadata of the authentication tree.

At DRIPS exit, the context is restored from DRAM by following in reverse the steps of the entry flow. However, the PMU, memory-controller, and MEE need to be restored and activated first before the exit flow can access the DRAM and read the context from the protected memory region. Therefore, approximately $1KB$ of the processor context (only 0.5% of the entire processor context) is still required to be stored on-chip, in a dedicated small SRAM (Boot_SRAM) using a special FSM (Boot_FSM). When the Boot_FSM is triggered by the chipset, it restores the state of the PMU, memory-controller, and MEE from the Boot_SRAM. Hence, at DRIPS exit, five main steps are carried out: 1) the PMU is restored by the Boot_FSM (triggered by the chipset), 2) memory-controller and MEE are restored by the Boot_FSM, 3) DRAM is instructed to exit self-refresh mode, 4) the PMU triggers the *SA FSM* to restore the system agent components, and 5) the PMU triggers the LLC FSM to restore the context of the cores and graphics from the DRAM into the relevant on-chip SRAMs.

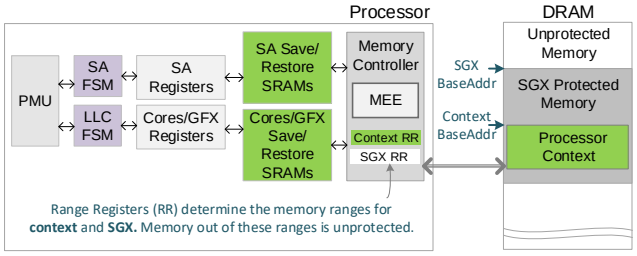

Figure 4: Flow for transferring the processor context into an SGX-protected DRAM area.

6.3 Latency and Hardware Cost

Save/restore latency analysis. We estimate the latency of transferring the processor context from the SRAMs into the protected DRAM region by building an emulation environment for the Skylake processor equipped with DDR3 that operates at $1.6GHz$. The emulation environment implements the processor's RTL code on an FPGA. The environment is used for pre-silicon RTL verification and runs at a slower frequency relative to the target processor chip. We measure the

number of cycles on the emulation environment and extrapolate them to the actual processor cycles. The latency for writing the context to the protected DRAM region is about $18us$, while that of reading the processor context from DRAM is about $13us$. This is a tolerable latency since much longer latency (in the millisecond scale) can be afforded while entering and exiting DRIPS (Sec. 3). These results have been post-silicon validated on a real system using a Skylake client processor. Our experimental validation shows an accuracy of 95% in our latency estimation.

Hardware cost analysis. The SGX-protected memory region in DRAM has a limited capacity of typically $64MB$ or $128MB$ [55]. This is more than enough space as our proposal requires at most $200KB$ of this storage space to save the processor context. This corresponds to less than 0.3% of the SGX protected memory capacity (i.e., less than 0.002% of the capacity of an $8GB$ DRAM).

7. EVALUATION METHODOLOGY

Power Model. We evaluate ODRIPS using our in-house *power model* to 1) estimate the average platform power when we apply each one of the three techniques and 2) calculate the energy break-even point of these techniques.

Our power model estimates platform average power consumption of the connected-standby periodic interval using Equation 1 as explained in Section 2.3. The periodic interval includes four main power states: 1) *Entry* state that includes the steps needed for preparing for entering DRIPS, 2) DRIPS, 3) *Exit* state that includes the steps needed for preparing for exiting DRIPS, and 4) Active state, $C0$, as we show in Fig. 2.

The *power-level* at each state is estimated using four main steps: 1) measuring the state's power consumption on a previous generation of our target processor, 2) scaling the power numbers to the new processor's process technology, 3) estimating power breakdown using design characteristics, and 4) projecting the expected power reduction when applying the power reduction techniques. This method is widely used in industry for power modeling [3, 69, 80]. Next, we provide more detail on this method.

First, we carry out power-measurements on a real system that includes Haswell-ULT [86], a predecessor of Skylake. Haswell-ULT is fabricated at $22nm$ process tecnology, while Skylake is fabricated at $14nm$. We provide the specifications of Haswell-ULT in Table 1. Haswell-ULT supports the connected standby mode. Our measurement tools (explained in this section) can accurately measure the power consumption at each of the four states of the connected standby mode. Second, to estimate the power consumption of our processor, Skylake, we scale the measured power consumption of Haswell-ULT ($22nm$) to that of Skylake ($14nm$). We use a similar method to estimate the chipset power. We perform the scaling using the characteristics of the new process that determines the scaling factor. More detail on this scaling method can be found in [8, 79]. Third, to further break down the measured (and the scaled) power of DRIPS into the power consumption of each sub-component, as shown in Fig. 1(b), we use power consumption estimation techniques [29]. These techniques exploit design characteristics (such as capacitance, leakage, and operational frequency and voltage) of architectural sub-components to estimate the relative power consumption of each sub-component. We estimate the absolute power consumption of each sub-component using the measured (and scaled) power consumption of the processor in DRIPS. Fourth, we estimate the power-level at each state when applying each one of the power reduction techniques using the power breakdown data and the expected effect of each technique on the platform power. For instance, turning off the crystal oscillators (XTAL) in ODRIPS saves XTAL power consumption when the processor is in ODRIPS.

Table 1: Baseline and target system parameters

Processor	Baseline: Intel i5-4300U Haswell, $22nm$ Target: Intel i5-6300U Skylake, $14nm$ Frequencies: $800 - 2400MHz$. L3 cache (LLC): $3MB$. Thermal Design Point (TDP): $15W$
Chipset	Lynx Point-LP (Haswell) [86], Sunrise Point-LP (Skylake) [88]
Memory	DDR3L-1.6GHz [64], non-ECC, dual-channel Capacity: $8GB$

We measure the residency of each state (i.e., the percentage of time the processor spends in a given power state) for the baseline Haswell system that supports the connected-standby mode using the performance counter monitor [89].

Power-model Validation. Before carrying out an actual implementation of ODRIPS, we use our power-model to estimate the potential power savings of our three proposed techniques. We validate our power model and power estimations by performing power measurements on a real system that includes the Skylake processor chip and chipset after ODRIPS fabrication (post-silicon). Intel Skylake and associated chipset (Sunrise Point-LP) implement both DRIPS (baseline from previous generation) and ODRIPS (our proposal) with a debug switch to disable the newly-implemented ODRIPS optimization. This debug switch allows us to accurately assess and compare the power consumption and latency of the two approaches, DRIPS and ODRIPS. We found that the accuracy of our power-model is approximately 95%.

Power Measurements. For the platform *power measurements*, we use a Keysight N6705B DC power analyzer [45] equipped with an N6781A source measurement unit (SMU) [46], as we show in Fig. 5. The N6705B (equipped with N6781A) is normally used to measure lower power devices, such as smartphones and tablets, with high accuracy (around 99.975% [46]). The power analyzer measures and logs the instantaneous power consumption of different device components. The power analyzer can measure in a single experiment the power consumption of the four power states: Entry into DRIPS, DRIPS, Exit from DRIPS and Active state. The power measurement values range from

Figure 5: Platform power measurement infrastructure using a Keysight N6705B power analyzer equipped with a N6781A source measurement unit (SMU).

tens of micro-watts (DRIPS) up to approximately three watts in Active ($C0$) state. We carry out multiple measurements for different platform components, including DRAM, storage (e.g., SSD), chipset, crystal oscillators, and the processor. Each measurement uses four analog channels with a 50-microsecond sampling interval.

Workloads. Our main workload is an idle platform workload that places the platform into the connected-standby mode. Our platform enters into connected-standby mode when it is idle (with display-off) for few tens of milliseconds. As shown in Section 2.3, in the connected-standby mode, the platform periodically transitions from the Active state ($C0$) to the Idle state (DRIPS). We measure the residency of each state in connected-standby mode on our baseline platform. We observe that (1) the residency at $C0$ and during state transitions is 0.5% of the total time and (2) the residency in DRIPS is 99.5% of the total time. More specifically, the system 1) spends about 30 seconds at idle, 2) wakes up with an exit latency of about 300 microseconds to the Active state to perform OS kernel maintenance tasks within $100-300$ milliseconds, 3) transitions back with an entry latency of about 200 microseconds to the Idle state.

To determine the energy break-even point of the three power reduction techniques used in ODRIPS, we perform two key steps for each one of our proposed techniques and the baseline DRIPS: 1) measuring the platform power consumption of connected-standby as we *sweep over* the residency in DRIPS, starting from $0.6ms$ up to 1 second with a granularity of $0.1ms$ and 2) determining the break-even point of each one of the three techniques by comparing the connected-standby average power measurement at each residency of DRIPS. For each technique, we select the DRIPS residency at which the platform average power of the proposed technique is lower than the one with the baseline DRIPS.

8. RESULTS

We evaluate the effect of our three proposed techniques on the energy consumption of a modern mobile device in the connected-standby mode (as depicted in Fig. 2). To demonstrate the benefits of our three new techniques, we perform three main evaluations. We evaluate 1) the average power consumption of the baseline platform in the connected-standby mode, 2) the average-power savings provided by each of our three techniques (wake-up event handling, denoted by `WAKE-UP-OFF`, AON IO power-gating, denoted by `AON-IO-GATE`, and moving processor context to SGX protected DRAM, denoted by `CTX-SGX-DRAM`) when added to the baseline platform, and 3) the average-power savings provided by implementing all three proposed techniques together as a new power state, ODRIPS. We present the results of these three evaluations in Fig. 6(a).

Based on these results, we make five key observations: 1) Our first technique, `WAKE-UP-OFF`, reduces the average-power consumption of the baseline platform by 6%. 2) Our second technique (together with our first technique[4]), `AON-IO-GATE`, reduces the platform average power consumption by 13%. 3) Our third technique, `CTX-SGX-DRAM`, by itself reduces the total average power consumption of the baseline platform by 8%. 4) Implementing all our three techniques together, `ODRIPS`, provides a significant platform average power reduction of 22%. The reduction comes from the following components: 1% (Wake-up & timer) + 5% (AON IO) + 4% (24MHz crystal) + 7% (S/R SRAMs) + 5% (power-delivery[5]). 5) The average power consumption during both the Active state (i.e., $C0$) and the transitions between the Active state and the Idle state (i.e., entering/exiting DRIPS), denoted by `Active&Transitions`, is more than 18% of the total platform average power in connected-standby mode.

In addition, Fig. 6(a), on its right y-axis using the blue line graph, plots the DRIPS residency break-even point as compared to the baseline, which represents the minimum amount of time that the workload needs to spend in the new ODRIPS to have lower energy consumption than the baseline DRIPS. We observe that each of our three techniques for ODRIPS has nearly a similar break-even point to that of the three techniques combined together (`ODRIPS`). The exact break-even point values for `WAKE-UP-OFF`, `AON-IO-GATE`, `CTX-SGX-DRAM`, and `ODRIPS` are $6.6ms$, $6.3ms$, $7.4ms$, and $6.5ms$, respectively. Since the connected-standby mode in our system is expected to spend 30 seconds in DRIPS (i.e., much longer than the empirically determined break-even points), we conclude that our new techniques and the new proposed state `ODRIPS` provide superior power efficiency over the baseline DRIPS.

8.1 Effect of Increasing Core Frequency

Motivated by the results presented in Fig. 6(a) that reveal significant platform average power consumption in the connected-standby mode for the Active state and entering/exiting DRIPS (`Active&Transitions`), we examine the effect of increasing the core frequency on average platform power in Fig. 6(b). The increase in core frequency can potentially reduce the Active state (i.e., $C0$) residency and enable going back to the ODRIPS

[4]The power gating of AON IOs should be applied along with wake-up event handling as the latter facilitates the power-gating of AON IOs by migrating the timer to the chipset.

[5]The measured power delivery efficiency in our system is 74% in DRIPS. Power delivery losses are added to the nominal power of each component. For example if the nominal power of SRAM is 10mW, then, with power delivery "tax", overall SRAM power is $10/0.74 = 13.51$mW.

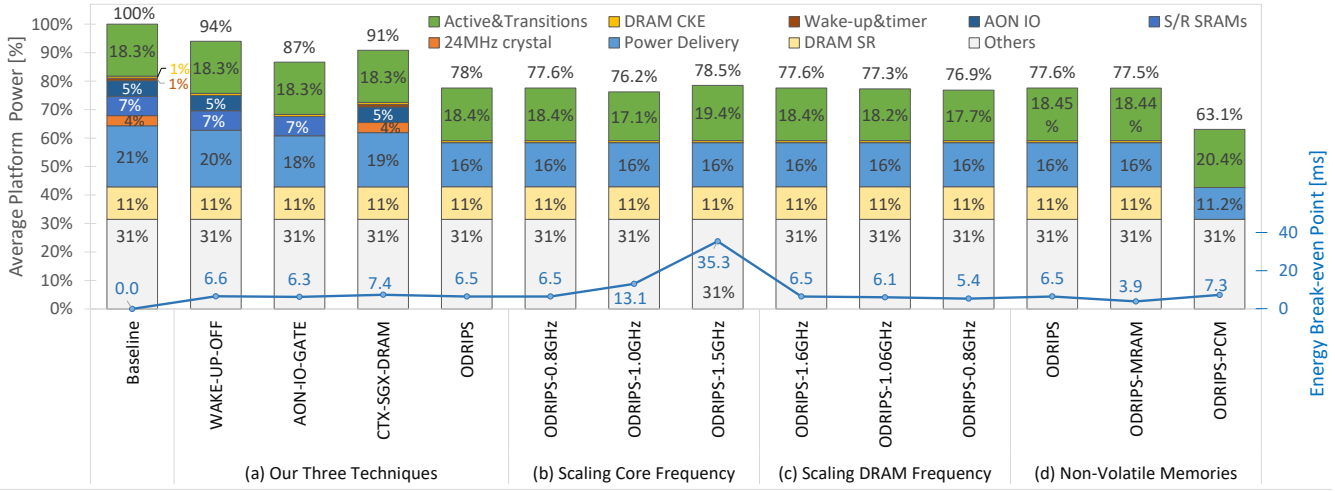

Figure 6: Platform average power consumption and energy break-even point for the baseline and (a) our three power reduction techniques and ODRIPS. (b) ODRIPS while scaling core frequency from 0.8GHz (baseline) to 1GHz and to 1.5GHz, and (c) ODRIPS while scaling DRAM frequency from 1.6GHz (baseline) to 1.06GHz and to 0.8GHz. (d) ODRIPS which uses embedded MRAM (`ODRIPS-MRAM`) and PCM (`ODRIPS-PCM`) memory technologies to store the processor context.

faster, which is also known as *race-to-sleep* [94, 22].

We make a key observation. The baseline frequency ($0.8GHz$) does *not* provide the minimum power consumption. Increasing the core frequency from the baseline frequency of $0.8GHz$ to $1.0GHz$ provides a small average power savings (1.4%). However, increasing the core frequency to $1.5GHz$ increases the average power consumption of the baseline platform by about 1%. This is due to an increase in the average power consumption of the platform during both the Active state and the transition from the Active state, even though the time spent in the Active state is reduced at $1.5GHz$.

We conclude that the best core clock frequency for low power consumption is at some point between $0.8GHz$ and $1.5GHz$ for our evaluation platform.

8.2 Effect of Reducing DRAM Frequency

We evaluate the effect of reducing the DRAM frequency on the average power consumption of the platform in Fig. 6(c). Theoretically, reducing the DRAM operating frequency reduces both the platform power consumption and memory bandwidth, but it also increases both memory latency [17] and the utilization of DRAM interface. That is, workloads that are sensitive to bandwidth or latency may spend more time in the Active state. Based on Fig. 6(c), we make two key observations. 1) Reducing the DRAM frequency below $1.6GHz$ shows a small effect (a reduction of about 0.3% and 0.7% for DRAM frequencies of $1.067GHz$ and $0.8GHz$, respectively) on the platform average-power consumption for our connected-standby workload.

The reduction in platform average power comes mainly from reducing the power in the Active state and transitioning states (`Active&Transitions`). 2) Memory bandwidth reduction increases the entry and exit latencies to ODRIPS, since a longer time is needed to save/restore the context to/from DRAM.

Such a latency increase is negligible for connected-standby mode due to the high residency (e.g., 30 seconds in our case [56]) at low power mode, ODRIPS.

We conclude that ODRIPS can be slightly more efficient with the lowest DRAM frequency (i.e., $0.8GHz$) for the connected-standby mode. While reducing the frequency of the DRAM is beneficial for the connected-standby workload, doing so might degrade performance of other workloads and therefore even result in an increase in the overall platform energy consumption [94]. Therefore, statically reducing the DRAM frequency is likely not a good strategy. Alternatively, it might be more efficient to apply dynamic voltage and frequency scaling to main memory, similar to [17, 10, 18].

8.3 Effect of Using Emerging Memory Technologies

We evaluate the potential energy savings from storing the processor context in two emerging memory technologies instead of DRAM. First, we evaluate eMRAM [77, 25], an embedded non-volatile memory. We assume an *optimistic* eMRAM design that has comparable 1) endurance, 2) power consumption, and 3) performance to SRAM. Second, we evaluate phase change memory (PCM) [90, 51, 52, 67, 52] as main memory (instead of DRAM), which eliminates the need for self-refresh and driving CKE signals. We include the results of these two experiments in Fig. 6(d). We refer to the combination of our first two techniques and the third technique that uses eMRAM instead of DRAM for storing the processor context as `ODRIPS-MRAM`. We similarly refer to ODRIPS that uses PCM instead of DRAM as `ODRIPS-PCM`.

We make two key observations. 1) `ODRIPS-MRAM` provides a slightly lower average-power consumption compared to ODRIPS. This is mainly due to the ability to simply turn-off the voltage source of eMRAM in

ODRIPS while the context remains saved inside the eM-RAM, which reduces the energy spent in sending the context outside the chip. This technique provides the lowest energy break-even point compared to the other techniques. 2) `ODRIPS-PCM` provides a significant reduction in the average power of the baseline platform (by 37%), which is an average power reduction of 15% compared to ODRIPS that uses DRAM. These large average power savings are mainly due to the non-volatility of the PCM, which obviates the need for DRAM self-refresh and the need to drive the CKE interface from the processor.

We conclude that using PCM as a storage medium for the processor context in ODRIPS provides significant additional platform average power savings.

9. RELATED WORK

To our knowledge, this is the first work to explicitly focus on and improve the connected-standby power management of high-performance mobile devices. We group prior works into three major areas.

Power state optimizations. There exist many prior works, from both academia and industry, on idle system power management. System hardware vendors introduced the deep idle state feature [81, 92, 21, 37, 34, 82, 26] to reduce energy consumption of devices. In addition, industry standards, such as ACPI [83], define power management policies to apply when the platform is idle. From academia, Hypnos [43] proposes a new sleep mode for micro-controllers that combines the state retention of shallow sleep modes with the low power of deep sleep modes. It achieves this by scaling the micro-controller's supply voltage to slightly above the SRAM's data retention voltage [66, 48] while the micro-controller is in sleep mode. This technique is efficient for low-leakage processors, but for high performance processors, as in our case, leakage is high even when using SRAM data retention voltage.

Utilizing NVM in low power states. Several efforts address the issue of efficiently saving the processor context before power down. Mementos [68] is one such solution that saves system state to on-chip flash memory. Advances in memory technologies have led to the emergence of non-volatile memories (NVMs) such as ferroelectric RAM (FeRAM), magnetic RAM (MRAM), resistive RAM (ReRAM) [11], which attempt to combine the speed and endurance of SRAM or DRAM with the non-volatility of flash memory. Quickrecall [42] and Hibernus [5, 70] use microcontrollers integrated with FeRAM, similar to [95], to enable efficient state saving.

Many emerging NVMs still have various limitations (e.g., fabrication cost) when compared to SRAM or DRAM [60, 11]. Therefore, these techniques are likely to have a longer-term impact on the high volume manufacturing of modern processors [21, 86, 63].

Using DRAM to save state. DRAM (or even hard disk) is used in prior works to save data from the OS. For example, at ACPI [83] suspend states S3 (suspend to RAM) and S4 (suspend to disk) [31, 26, 82], the OS context is saved to DRAM or to disk (HDD/SSD). This mechanism is initiated by the user and it involves the OS kernel in the save and restore process. The context that is saved is *not* the processor's (as in our case), it is the OS context whose security and authenticity is under the responsibility of the OS. In our approach, around $200KB$ of only processor context, which includes configuration registers, patching data, and fuse values, is saved and restored transparently to the OS. Thus, the target system has to fulfill a strict security requirement: the processor context must be protected when moving back and forth between the processor and the DRAM.

10. CONCLUSION

We present a coordinated three-pronged approach for architecting a new optimized deepest-runtime-idle-power state (ODRIPS) for improving the power consumption of high performance mobile platforms for the connected-standby mode. Our proposed ODRIPS design is based on three key techniques that together address the energy inefficiencies in current DRIPS designs. 1) ODRIPS offloads the monitoring of wake-up events to off-chip circuitry (in the chipset) that is optimized for low power operation to enable turning off all of the processor's clock sources. 2) ODRIPS offloads processor's always powered on (AON) IO functions off-chip and then power-gates the AON IOs. 3) ODRIPS utilizes the secure (e.g., SGX protected) memory region inside DRAM to save the processor context instead of storing the context in high-leakage SRAMs. Overall, our three techniques combined together save 22% of a real modern platform's average power consumption in the connected-standby mode. We demonstrate that the potential benefits gained from our proposed ODRIPS design can be even higher using emerging embedded and off-chip non-volatile memory (NVM) technologies. Such NVMs can further reduce the platform energy consumption of the mobile devices during their Idle state and provide a promising direction for further research.

ACKNOWLEDGMENTS

We thank the anonymous reviewers. We thank the SAFARI Research Group members for feedback and the stimulating intellectual environment they provide.

REFERENCES

[1] Jasmin Ajanovic. PCI express 3.0 overview. In *Hot Chips*, 2009.

[2] Farhan Azmat Ali, Pieter Simoens, Tim Verbelen, Piet Demeester, and Bart Dhoedt. Mobile Device Power Models for Energy Efficient Dynamic Offloading at Runtime. *Journal of Systems and Software*, 2016.

[3] Kumar Anshumali, Terry Chappell, Wilfred Gomes, Jeff Miller, Nasser Kurd, and Rajesh Kumar. Circuit and Process Innovations to Enable High-Performance, and Power and Area Efficiency on the Nehalem and Westmere Family of Intel Processors. *Intel Technology Journal*, 2010.

[4] Apple. Power nap. online, accessed February 2019. https://support.apple.com/en-us/HT204032.

[5] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. Hibernus: Sustaining computation During Intermittent Supply for Energy-Harvesting Systems. *ESL*, 2015.

[6] Rick Boivie and Peter Williams. SecureBlue++: CPU Support for Secure Execution. *IBM Technical Report*, 2012.

[7] Wayne Burleson, Onur Mutlu, and Mohit Tiwari. Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer. In *DAC*, 2016.

[8] J Adam Butts and Gurindar S Sohi. A Static Power Model for Architects. In *MICRO*, 2000.

[9] Aaron Carroll and Gernot Heiser. An Analysis of Power Consumption in a Smartphone. In *ATC*, 2010.

[10] Kevin K Chang, A Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, et al. Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. *SIGMETRICS*, 2017.

[11] An Chen. A Review of Emerging Non-Volatile Memory (NVM) Technologies and Applications. *SSE*, 2016.

[12] Chen Chen, David Barrera, and Adrian Perrig. Modeling Data-Plane Power Consumption of Future Internet Architectures. In *CIC*, 2016.

[13] Nakjung Choi, Kyle Guan, Daniel C Kilper, and Gary Atkinson. In-network caching effect on optimal energy consumption in content-centric networking. In *ICC*, 2012.

[14] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *MEJ*, 2016.

[15] ComScore. Cross Platform Future in Focus . online, accessed Feb 2019, 2017. https://bit.ly/2G391UB.

[16] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR*, 2016.

[17] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. Memory Power Management Via Dynamic Voltage/Frequency Scaling. In *ICAC*, 2011.

[18] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. MemScale: Active Low-power Modes for Main Memory. In *ASPLOS*, 2011.

[19] Anant Deval, Avinash Ananthakrishnan, and Craig Forbell. Power management on 14 nm Intel® Core- M processor. In *Cool Chips*, 2015.

[20] Ning Ding, Daniel Wagner, Xiaomeng Chen, Abhinav Pathak, Y Charlie Hu, and Andrew Rice. Characterizing and Modeling the Impact of Wireless Signal Strength on Smartphone Battery Drain. *SIGMETRICS*, 2013.

[21] Jack Doweck, Wen-Fu Kao, Allen Kuan-yu Lu, Julius Mandelblat, Anirudha Rahatekar, et al. Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. *IEEE Micro*, 2017.

[22] Rotem Efraim, Ran Ginosar, C Weiser, and Avi Mendelson. Energy Aware Race to Halt: A Down to EARtH Approach for Platform Energy Management. *CAL*, 2014.

[23] Chao Fang, F Richard Yu, Tao Huang, Jiang Liu, and Yunjie Liu. An Energy Efficient Distributed In-Network Caching Scheme for Green Content-Centric Networks. *Computer Networks*, 2015.

[24] Denis Foley, Pankaj Bansal, Don Cherepacha, Robert Wasmuth, Aswin Gunasekar, et al. A Low-Power Integrated x86–64 and Graphics Processor for Mobile Computing Devices. *JSSC*, 2012.

[25] O Golonzka, J-G Alzate, U Arslan, M Bohr, P Bai, J Brockman, B Buford, C Connor, N Das, B Doyle, et al. MRAM as Embedded Non-Volatile Memory Solution for 22FFL FinFET Technology. In *IEDM*, 2018.

[26] Corey Gough, Ian Steiner, and Winston Saunders. *Energy Efficient Servers: Blueprints for Data Center Optimization*. Apress, 2015.

[27] Aaron Grenat, Sriram Sundaram, Stephen Kosonocky, Ravinder Rachala, Sriram Sambamurthy, , et al. 4.2 Increasing the Performance of a 28nm x86-64 Microprocessor Through System Power Management. In *ISSCC*, 2016.

[28] Shay Gueron. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR*, 2016.

[29] Stephen H Gunther. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, 2001.

[30] Paul J Gyugyi, Roman Surgutchik, and Raymond A Lui. Network Interface Speed Adjustment to Accommodate High System Latency in Power Savings Mode, 2009. US Patent 7,603,574.

[31] Jawad Haj-Yahya, Avi Mendelson, Yosi Ben Asher, and Anupam Chattopadhyay. Power Management of Modern Processors. In *Energy Efficient High Performance Processors*. Springer, 2018. ISBN: 978-981-10-8553-6.

[32] Jawad Haj-Yihia. Connected Standby Sleep State, 2013. US Patent 8,458,503.

[33] Per Hammarlund, Alberto J Martinez, Atiq Bajwa, David L Hill, Erik Hallnor, et al. 4th Generation Intel Core Processor, Codenamed Haswell. In *Hot Chips*, 2013.

[34] Per Hammarlund, Alberto J Martinez, Atiq A Bajwa, David L Hill, Erik Hallnor, et al. Haswell: The Fourth-Generation Intel Core Processor. *IEEE Micro*, 2014.

[35] Zecheng He and Ruby B Lee. How Secure is Your Cache Against Side-Channel Attacks? In *MICRO*, 2017.

[36] Bernd Hoefflinger. ITRS: The International Technology Roadmap for Semiconductors. In *Chips 2020*. Springer, 2011.

[37] Song Huang, Michael Lang, Scott Pakin, and Song Fu. Measurement and Characterization of Haswell Power and Energy Consumption. In *E2SC*, 2015.

[38] Thomas Ilsche, Marcus Hähnel, Robert Schöne, Mario Bielert, and Daniel Hackenberg. Powernightmares: The Challenge of Efficiently Using Sleep States on Multi-Core Systems. In *Euro-Par*, 2017.

[39] Intel. Intel Core i7-6700K Processor. online, accessed June 2019. https://intel.ly/2udSI4A.

[40] Intel. Intel Core m7-6Y75 Processor. online, accessed June 2019. https://intel.ly/2MRGp4e.

[41] Suman Jana and Vitaly Shmatikov. Memento: Learning Secrets from Process Footprints. In *S&P*, 2012.

[42] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. QuickRecall: A HW/SW Approach for Computing Across Power Cycles in Transiently Powered Computers. *JETC*, 2015.

[43] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. Hypnos: An Ultra-Low Power Sleep Mode with SRAM Data Retention for Embedded Microcontrollers. In *CODES+ISSS*, 2014.

[44] David Kaplan, Jeremy Powell, and Tom Woller. AMD memory encryption. *White Paper*, 2016.

[45] Keysight. IntKeysight N6705B DC Power Analyzer. online, accessed June 2019. https://bit.ly/2MZ9Hhv.

[46] Keysight. IntKeysight Source Measure Units Power Modules. online, accessed June 2019. https://bit.ly/38kkStt.

[47] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ISCA*, 2014.

[48] Jaydeep P Kulkarni, Keejong Kim, Sang Phill Park, and Kaushik Roy. Process Variation Tolerant SRAM Array for Ultra Low Voltage Applications. In *DAC*, 2008.

[49] Nasser Kurd, Muntaquim Chowdhury, Edward Burton, Thomas P Thomas, Christopher Mozak, Brent Boswell, Praveen Mosalikanti, Mark Neidengard, Anant Deval, Ashish Khanna, et al. Haswell: A Family of IA 22 nm Processors. *JSSC*, 2015.

[50] John H Lau. Recent Advances and Trends in Heterogeneous Integrations. *JMEP*, 2019.

[51] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA*, 2009.

[52] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Phase Change Memory Architecture and the Quest for Scalability. *CACM*, 2010.

[53] Donghyuk Lee, Lavanya Subramanian, Rachata Ausavarungnirun, Jongmoo Choi, and Onur Mutlu. Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM. In *PACT*, 2015.

[54] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012.

[55] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel SGX Support for Dynamic Memory Management Inside an Enclave. In *HASP*, 2016.

[56] Microsoft. Behavior differences between S3 and Modern Standby. online, accessed January 2019. https://bit.ly/2sGQ3zW.

[57] Microsoft. Functional Overview of Modern Standby. online, accessed January 2019. https://bit.ly/35cEIF2.

[58] Microsoft. Modern Standby Wake Sources. online, accessed January 2019. https://bit.ly/2SPBrJc.

[59] Microsoft. Prepare Hardware for Modern Standby. online, accessed January 2019. https://bit.ly/2tkU78Z.

[60] Onur Mutlu. Memory Scaling: A Systems Architecture Perspective. In *IMW*, 2013.

[61] Onur Mutlu. The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. In *DATE*, 2017.

[62] Onur Mutlu and Jeremie S Kim. RowHammer: A Retrospective. *TCAD*, 2019.

[63] Ankireddy Nalamalpu, Nasser Kurd, Anant Deval, Chris Mozak, Jonathan Douglas, et al. Broadwell: A Family of IA 14nm Processors. In *VLSI Circuits*, 2015.

[64] Addendum No. JESD79-3-1.35 V DDR3L-800, DDR3L-1066, DDR3L-1333, DDR3L-1600, and DDR3L-1866. *JEDEC Std., JESD79-3-1A*, 1, 2013.

[65] Venkatesh Pallipadi, Shaohua Li, and Adam Belay. CPUidle: Do Nothing, Efficiently. In *Linux Symposium*, 2007.

[66] Huifang Qin. *Deep Sub-Micron SRAM Design for Ultra-Low Leakage Standby Operation*. University of California, Berkeley, 2007.

[67] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *ISCA*, 2009.

[68] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System Support for Long-Running Computation on RFID-scale devices. In *ASPLOS*, 2011.

[69] Thomas Rauber, Gudula Rünger, Michael Schwind, Haibin Xu, and Simon Melzner. Energy Measurement, Modeling, and Prediction for Processors with Frequency Scaling. *The Journal of Supercomputing*, 2014.

[70] Alberto Rodriguez Arreola, Domenico Balsamo, Anup K Das, Alex S Weddell, Davide Brunelli, Bashir M Al-Hashimi, and Geoff V Merrett. Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation. In *ENSsys*, 2015.

[71] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. Power Management Architecture of the Intel Microarchitecture Code-named Sandy Bridge. *IEEE Micro*, 2012.

[72] Gururaj Saileshwar, Prashant Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose Joao, and Moinuddin Qureshi. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *MICRO*, 2018.

[73] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin K Qureshi. Synergy: Rethinking Secure-Memory Design for Error-Correcting Memories. In *HPCA*, 2018.

[74] Ali Shafiee, Rajeev Balasubramonian, Mohit Tiwari, and Feifei Li. Secure DIMM: Moving ORAM primitives closer to memory. In *HPCA*, 2018.

[75] Chunlei Shi, Brett C Walker, Eric Zeisel, Brian Hu, and Gene H McAllister. A Highly Integrated Power Management IC (PMIC) for Advanced Mobile Applications. *JSSC*, 2007.

[76] William Shockley. A Unipolar Field-Effect Transistor. *IRE*, 1952.

[77] D Shum, D Houssameddine, ST Woo, YS You, J Wong, KW Wong, CC Wang, KH Lee, K Yamane, VB Naik, et al. CMOS-embedded STT-MRAM Arrays in 2x nm Nodes for GP-MCU Applications. In *VLSI Symposium*, 2017.

[78] Pat Stemen and Steven Berard. Understanding Connected Standby. *Microsoft Build*, 2011.

[79] Aaron Stillmaker and Bevan Baas. Scaling Equations for the Accurate Prediction of CMOS Device Performance from 180 nm to 7 nm. *Integration*, 2017.

[80] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L Greathouse, and Zhiying Wang. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *MICRO*, 2014.

[81] Qualcomm Technologies. Qualcomm Snapdragon 410E (APQ8016E) Processor Device Specification. online, 2018. https://bit.ly/37cBGT5.

[82] Steven Tu. Atom-x5/x7 Series Processor, Codenamed Cherry Trail. In *Hot Chips*, 2015.

[83] Vantage. Advanced Configuration and Power Interface (ACPI) Specification . online, accessed January 2019. http://www.acpi.info.

[84] Yao Wang, Andrew Ferraiuolo, and G Edward Suh. Timing Channel Protection for a Shared Memory Controller. In *HPCA*, 2014.

[85] Wikipedia. Apple macbook air. online, accessed June 2019. https://en.wikipedia.org/wiki/MacBook_Air.

[86] Wikipedia. Haswell Microarchitecture. online, accessed June 2019. https://en.wikipedia.org/wiki/Haswell_(microarchitecture).

[87] Wikipedia. Microsoft surface. online, accessed June 2019. https://en.wikipedia.org/wiki/Microsoft_Surface.

[88] Wikipedia. Skylake Microarchitecture. online, accessed June 2019. https://bit.ly/2NG8Hz2.

[89] Thomas Willhalm, Roman Dementiev, and Patrick Fay. Intel Performance Counter Monitor. https://software.intel.com/en-us/articles/intel-performance-counter-monitor.

[90] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. Phase Change Memory. *Proceedings of the IEEE*, 2010.

[91] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd Austin. Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors. In *HPCA*, 2017.

[92] Rumi Zahir, Mark Ewert, and Hari Seshadri. The Medfield Smartphone: Intel Architecture in a Handheld form Factor. *IEEE Micro*, 2013.

[93] Michael Zelikson and Alex Waizman. Embedded Power Gating, 2011. US Patent 7,880,284.

[94] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T Kandemir, Ravi Iyer, and Chita R Das. Race-to-sleep + Content Caching + Display Caching: a Recipe for Energy-Efficient Video Streaming on Handhelds. In *MICRO*, 2017.

[95] Michael Zwerg, Adolf Baumann, Rüdiger Kuhn, Matthias Arnold, Ronald Nerlich, et al. An 82μA/MHz microcontroller with embedded FeRAM for energy-harvesting applications. In *ISSCC*, 2011.