OCP: Offload Co-Processor for Energy Efficiency in Embedded Mobile Systems

Jie Tang
Intel
Beijing, China
tangjie.bit@gmail.com

Chen Liu
Clarkson University
Potsdam, NY, USA
cliu@clarkson.edu

Yu-Liang Chou
University of California
Irvine, CA, USA
yulianc@uci.edu

Shaoshan Liu
Microsoft
Redmond, WA, USA
shaoliu@microsoft.com

***Abstract*— In current embedded mobile systems design, the application processor (AP) is often woken up to service interrupts and user requests. However, this kind of wakeups from sleep is very expensive in terms of battery usage. In the observation that the operating system/driver workloads are very light-weight, in this paper we propose the Offload Co-Processor (OCP) SoC architecture. In the OCP SoC design, when the device is idle, we offload the operating system workloads (mainly interrupt handling workloads) to an ultra-low-power co-processor. This way, the co-processor would be able to handle most wake-up requests without awakening the heavy-weight AP, thus avoiding the overhead of AP spin-up/down. Using GPS continuous sampling workload as a case study, we show that the proposed OCP SoC design would extend battery life by 3.5 folds.**

Keywords*— *Embedded Mobile System; Energy Aware Computing; Offload Co-processor; Application Processor; Interrupt Service Routines; GPS Continuous Sampling

I. Introduction

Battery is the most precious and limited resource on embedded mobile systems, and many studies have focused on increasing the battery life of such systems [1-3]. In current embedded mobile systems design, the application processor (AP) is the most power-consuming component, and is often woken up to service interrupts and user requests. For instance, the continuous GPS sampling workload would wake up the AP once per second to store the coordinates of the current location. Another example is that the cellular tower would ping the device, and thus wake up the AP once per minute to keep the connection alive. These wake-ups from idle scenarios are very energy-consuming, as we will show in later section.

When these wakeup calls to the AP happen, the AP would execute in kernel mode and most of the time, run the interrupt service routine (ISR) of the interrupting device. We have observed that the CPU usage of such operating systems (OS) workloads is actually very low, thus does not require a heavy-weight processor such as the AP. Based on this observation, we propose offloading operating system workloads to an ultra-low-power Offload Co-Processor (OCP) and keeping the AP in energy-saving deep-sleep state. This way, the co-processor would be able to handle most wake-up requests without awakening the heavy-weight application processor, thus leading to energy efficiency.

II. Operating Systems Workload Weight

The OS consists of basic kernel services such as memory management, thread scheduling, file and storage systems, as well as driver workloads such as interrupt service routines. In order to understand the CPU usage of the OS workloads, we run a set of CPU-intensive applications on a mobile device and sample the CPU usage by the OS kernel.

Figure 1 shows the CPU usage by the kernel in simple scenarios, including audio playback, video playback, streaming music, browsing static website, watching YouTube, browsing dynamic website, as well as using Skype. The results show that in the worst case, the kernel only uses 1.9% of the CPU.

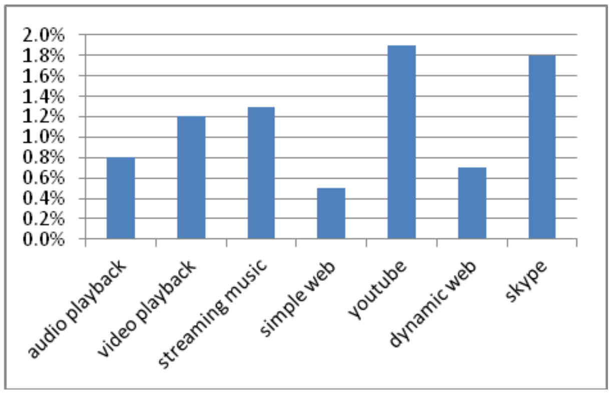

Fig. 1. CPU usage by the OS of simple scenarios

In the next step, we try to stress the CPU usage by combining these simple scenarios together, and the results are shown in Table I. Even though we run YouTube, browsing dynamic website, as well as Skype together, the CPU usage by the OS is only 4%. Drilling into the 4% CPU usage, it is consisted of running the first-level and second-level ISRs, as well as certain calls into the file and storage systems.

III. Energy Overhead of Wakeups

In current embedded mobile systems designs, when the system is not in use, the AP is put into low-power state to conserve energy. However, when there are requests coming from different components, such as the GPS, the AP is interrupted and woken up to service the interrupt. Usually these interrupts are very short: the AP spends a few milliseconds to service the interrupt and goes back to sleep mode again if there is no other requests.

In order to understand the power consumption of such scenarios, we collect energy trace from GPS sampling workload on a mobile device with an ARM 11 CPU as the AP [4], which can be found in many smart-phones today. Figure 2 shows the results, with the y-axis showing power consumption in milliWatts (mW), and the x-axis showing time elapsed in milliseconds (ms). Initially, the device is in sleep, consuming about 20 mW. Then the AP is interrupted and spinning up to full speed to execute the ISR. During this stage, it consumes about 1000 mW. Then after just 2 ms, the AP finishes processing. The power consumption, however, does not drop back to 20 mW directly. Instead, the CPU is stepping down to an idle state with a lower frequency, consuming about 650 mW. Then after 100 ms, it steps down again to a level consuming about 400 mW. Then it stays there for another 100 ms before going back to the deep-sleep mode completely. Therefore, even though we only need the AP to service the interrupt for 2 ms, the AP becomes active for about 200 ms, consuming 60 milliJoules (mJ) of energy.

TABLE I. CPU USAGE BY THE OS OF COMBINED SCENARIOS

Workload Combinations	CPU Usage
YouTube+dynamic web	3.30%
audio playback+dynamic web	3.10%
video playback+dynamic web	2.70%
YouTube+audio playback+dynamic web	3.20%
YouTube+dynamic web+Skype	4.00%

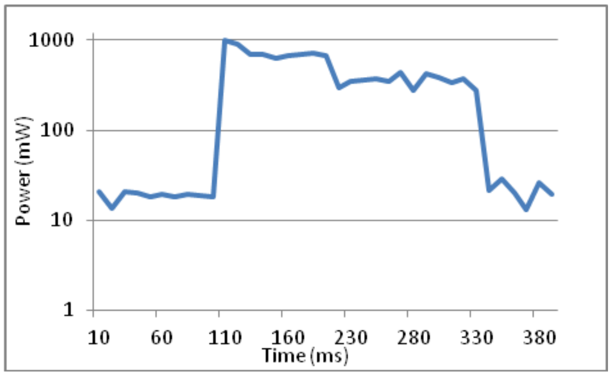

Fig. 2. Energy Consumption of a WakeupUnits

IV. OFFLOADING OS INTERRUPT WORKLOAD

The data from Section III shows that each time the AP is interrupted out of sleep mode; it introduces a very high energy overhead. The root cause of this is that the AP is spun up to full speed in an effort to provide good user experiences. Most of the time, however, the processing power provided by the AP is actually excessive for the interrupt routine it is handing. Associated with this, it also incurs significant energy overhead to spin-up/down the AP, leading to battery drain.

To address this problem, we propose the Offload Co-Processor (OCP) SoC architecture, as shown in Figure 3. In this architecture, the OCP is integrated with the AP and the interrupt controller on the same SoC. The OCP is a very light-weight ultra-low-power processor that is responsible for handling most of the interrupt services during device idle. In this way, we avoid triggering the AP to handle interrupts and leave it in deep-sleep state as long as possible. The OCP SoC works as follows:

1. The device switches into idle state, putting the AP and the OCP into sleep mode.
2. An interrupt comes in, triggering the OCP.
3. The OCP checks whether it needs to trigger the AP to service this interrupt.
4. If so, it interrupts the AP through inter-processor interrupt (IPI) and lets the AP to handle the interrupt.
5. Otherwise, it handles the interrupt and goes back to sleep mode upon completion.

Note that there could be several reasons that would lead to Step 4 above. First, the AP would be started if the intention of the interrupt is to wake up the AP. For example, if the user pushes the hardware button to bring up the device and turn on the screen; or if it is a timer interrupt to trigger the AP to perform certain task. Second, the AP would also be started if the OCP is not powerful enough to service the interrupt in a timely manner. For example, if the interrupt service routine (ISR) takes 1.5 seconds to run on the OCP but user wants it to finish within 1 second.

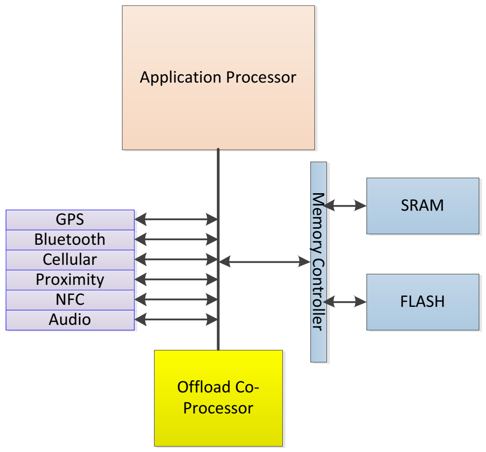

Fig. 3. SoC design with the Offload Co-Processor (OCP)

V. CASE STUDY: GPS CONTINUOUS SAMPLING

In this section, we use GPS continuous sampling application as a case study to analyze the energy consumption difference of the baseline design and the proposed OCP SoC design.

A. *OCP SoC Reference Design*

GPS sampling rate is usually 1 Hz, meaning that it interrupts the CPU once per second. It has been shown that GPS continuous sampling workload can take up to 95% of the total power consumption of the mobile devices [8], which also contributes to the reason why we choose it as the case study for our OCP SoC design. Actually [8] introduced a “LittleRock” design targeting the similar problem as we proposed here. Their approach, however, is a multi-processor design utilizing off-the-shelf components, while our design is from a micro-architecture point-of-view for next-generation many-core mobile devices.

In our proposed reference design of the OCP SoC architecture, we use an ultra-low-power MIPS32 4KS processor as the OCP [5]. MIPS32 4KS processor runs at 200 MHz and is able to execute 322 million instructions per second. When running at 200 MHz, it consumes 60 mW, comparing to the AP power consumption of around 1000 mW. This processor does not implement any dynamic voltage and frequency scaling and thus the spin-up/down of the processor is almost instantaneous.

B. *Energy Consumption Analysis*

In this subsection, we compare the battery life of baseline design versus the OCP SoC reference design using a simplified model as shown in Equation 1 below: *E* represents the amount of energy carried by a battery; *P* represents the power consumption (in Watts); *A* represents the percentage of time AP is in active state; and *T* represents the entire length of battery life. This equation estimates the battery life in ideal scenario with only GPS continuous sampling workload running in the background.

$$E = P_{AP-sleep}(1-A_1)T_1 + P_{AP-active}A_1T_1 \qquad (1)$$

We assume a battery of 1500 mAh at 3.7 V, which is usually used in mobile devices. And from the measurement in Section III, we obtained $P_{AP-sleep}$ as 20 mW, and $P_{AP-active}$ as 300 mW. This is because each time a GPS interrupt happens, it costs the AP 60 mJ to service the interrupt and last about 200ms ($A_1 = 20\%$). So $P_{AP-active}$ can be derived by 60 mJ/200 ms, which is 300 mW. Thus, in the baseline design, when running the GPS continuous sampling workload in the background, the battery would last about 73 hours.

$$E = P_{AP-sleep}T_2 + P_{OCP-sleep}(1-A_2)T_2 + P_{OCP-active}A_2T_2 \qquad (2)$$

In the reference OCP SoC design, $P_{AP-sleep}$ would remain at 20mW as the interrupt now is handled by OCP so the AP is kept in sleep. $P_{OCP-sleep}$ is 1 mW for the OCP as we specified the OCP is of ultra-low-power, especially in the sleep mode. $P_{OCP-active}$ would be 60 mW for the OCP. When interrupt happens every one second, the AP would be kept in sleep mode while the OCP would spin up to 200 MHz, perform computation for about 15 ms, and then go back to sleep. In this case A_2 would be 1.5%. During this period, it consumes 60 mW. Using Equation 2, we can derive that in this case, the battery would last about 254 hours, a 3.5 times battery life improvement compared to the baseline.

In general, based on Equations 1 and 2 and under the condition that the energy carried by the battery remains the same for both designs, we can derive the battery life extension *f* as a function of A_1 and A_2 :

$$f = \frac{T_2}{T_1} = \frac{P_{AP-sleep}(1-A_1) + P_{AP-active}A_1}{P_{AP-sleep} + P_{OCP-sleep}(1-A_2) + P_{OCP-active}A_2} \qquad (3)$$

With A_1 and A_2 both between the range of [0, 1] and plug in the power consumption numbers from this section for AP and OCP, we can get Figure 4. The maximum battery life extension is around 14 folds when A_1 is 1 and A_2 is 0, which means AP is kept active all the time while OCP is kept asleep. The minimum battery life extension is 0.25, which means a reduced battery life when A_1 is 0 and A_2 is 1, which means AP is kept sleep all the time while OCP is kept active all the time. Both of the two cases will not happen in real scenarios, nevertheless, they can show the trend. For the example we discussed earlier, *f* is 3.5 when A_1 is 20% and A_2 is 1.5%. Actually in this specific example, as long as AP is kept active for more than 21.4% of the time to handle the interrupt, then we can always achieve a battery life extension with the introduced OCP SoC design, even if the OCP is kept busy all the time, under the condition that OCP can finish the job within the deadline.

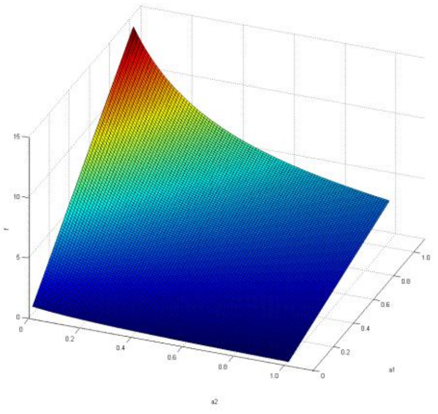

Fig. 4. Battery Life Extension for OCP Design

C. *Real-World Scenarios*

In this case study, we only consider a very simple scenario, such that the GPS is the only source of interrupt. However, in real-world scenarios, there are many interrupt sources when the phone is in idle state, these interrupt sources include:

- *Modem tower ping:* the device need to communicate with the radio tower periodically to make sure the connection persists.
- *Wi-Fi traffic:* when Wi-Fi is turned on, the Wi-Fi chip picks up signals in the air and sometimes brings the AP out of sleep.
- *Timer Interrupt:* the AP may have set up some timers before going into sleep.

- *Bluetooth/NFC:* Bluetooth and near-field communication may interrupt the AP to perform data transfer.

VI. Things to Consider in the OCP Design

In this section, we discuss issues associated with the proposed OCP SoC design.

A. *Different ISAs*

In the reference design, we propose integrating a MIPS OCP with an ARM AP. This means that the OCP and the AP are using different instruction sets. The implication of this problem is that we need two sets of ISRs, one set for the OCP, and the other set for the AP. The OCP ISR set needs to be a subset of the AP ISR set, such that if the OCP could not handle a particular interrupt or if the OCP does not have the processing power to handle an incoming interrupt, we could always fall back to the AP.

B. *Interrupt Overheads*

In the ARM architecture [9], in order to handle re-entrant interrupts, when an interrupt happens, the processor switches from *user* mode to *interrupt* mode, saves the *user* mode context and then switches to *system* mode to handle the interrupt. This mode switch may easily incur 20~30 cycles of overhead. In the OCP design, our goal is to minimize the interrupt overhead. There could be several options: first, we can trigger the AP when re-entrant interrupt happens, and in this way we can keep the OCP design simple. This technique would work well if the mobile devise is idle most of the time, and only one interrupt happens at a time. Second, we could have similar interrupt handling techniques as in the AP, but this would complicate the OCP design. Third, we could use hardware acceleration techniques, which will be discussed next.

C. *Interrupt Overflow*

We define interrupt overflow as the scenario that the OCP would not be able to handle the interrupt in time, and thus have to wake up the AP to handle the interrupts. It would happen if an interrupt would take longer time to service on the OCP than it is required, or if there are too many interrupts waiting to be serviced by the OCP. The goal of OCP design is to minimize interrupt overflow.

D. *Hardware Acceleration*

One natural way to minimize the interrupt overhead and the interrupt overflow is to apply hardware acceleration techniques [6, 7]. First, to minimize interrupt overhead, we could extend the OCP ISA to have a special instruction to save the context and switch mode at the same time. Second, to minimize interrupt overflow, we would need to study the specific ISR workloads and check whether we can apply instruction collapsing and parallelization techniques [6] to accelerate their execution.

VII. Conclusions

In this paper, we propose implementing an Offload Co-Processor (OCP) in embedded mobile systems to achieve energy efficiency. This co-processor is an ultra-low-power processor that executes the interrupt service routines (ISR) of different hardware components, and thus avoiding the expensive operation of waking up the heavy-weight application processor. We introduce a reference OCP design using the MIPS32 4KS low-power processor. Using GPS continuous sampling workload as a case study, we demonstrate that the OCP design would be able to extend the battery life by 3.5 folds during idle scenarios.

Since we have demonstrated that the OCP design is effective, moving forward, we plan to target more advanced issues as discussed in Section VI. Specifically, we plan to apply hardware acceleration techniques to minimize the interrupt overheads and the interrupt overflows, and then integrate the OCP with the AP (which have different ISAs) onto the same SoC. We plan to carry out the implementation on FPGA and execute it with real-world scenarios.

Acknowledgment

We want to thank Fang Hua for the Matlab support. This work is partly supported by the National Science Foundation under grant number ECCS-1301953. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, Energy consumption in mobile phones: a measurement study and implications for network applications, In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement, 2009.

[2] G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott, Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling, In Proceedings of the Eighth International Symposium on High-Performance Computer Architecture, 2002.

[3] J. Tang, S. Liu, Z. Gu, C. Liu, and J-L. Gaudiot, Prefetching in Embedded Mobile Systems Can Be Energy-Efficient, Computer Architecture Letters, DOI: 10.1109/L-CA.2011.2, February, 2011.

[4] ARM11 Processor family: http://www.arm.com/products/processors/classic/arm11/index.php

[5] MIPS32 4KS family: https://www.mips.com/products/processor-cores/mips32-4ks/#specifications

[6] J. Tang, S. Liu, Z. Gu, X-F. Li, and J-L. Gaudiot, Achieving Middleware Execution Efficiency: Hardware-Assisted Garbage Collection Operations, Journal of Supercomputing, DOI: 10.1007/s11227-010-0493-0, November, 2010.

[7] S. Liu, R.N. Pittman, A. Forin, and J-L. Gaudiot, On Energy Efficiency of Reconfigurable Systems with Run-Time Partial Reconfiguration, in Proceedings of the 21st IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2010).

[8] N. Priyantha, D. Lymberopoulos, J. Liu, EERS: Energy Efficient Responsive Sleeping on Mobile Phones, in Proceedsings of the International Workshop on Sensing for App Phones, November, 2010.

[9] ARM Architecture Reference Manual: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.architecture/index.html