CHAPTER 7

Video Application Power Consumption on Low-Power Platforms

Some mobile applications, particularly those on low-power devices, unjustifiably use a lot of energy, causing unnecessary strain on the device battery. Some applications start automatically as soon as the device is powered on or do not go to sleep when the user stops using them; in most cases these applications keep performing noncritical tasks. Other applications may devote only a small fraction of their energy budget to the core task, while spending a larger portion on user tracking, uploading user information, or downloading ads.

To conserve battery power, systems usually go to deep sleep at every opportunity, from which they can be awaken as needed; some applications abuse this feature by regularly waking the device for nonessential tasks, such as checking the server for updates, new content, or mail and messages, or for reporting back on user activity or location. For this reason, judicious use of available resources and capabilities by these applications can save significant energy and thereby improve battery life.

However, most mobile media applications have a different emphasis in their power-saving attempts. They usually don't have sufficient performance headroom for doing auxiliary tasks; so they need to consider the characteristically complex nature of the media usages, as well as the resource constraints of the device.

Viewed primarily from the media application point of view, this chapter begins by extending the power consumption, management, and optimization discussions that were begun in Chapter 6 to the domain of the low-power device. The chapter starts with a discussion of the power-consumption priorities of low-power devices. Then, it presents typical media usages on these low-power devices. Four such common usages—namely video playback, video recording, video conferencing, and video delivery over Miracast

Wireless Display—are analyzed and discussed. Examples of activities and a power profile for each of these usages are also offered. This is followed with a discussion of the challenges and opportunities present in system low-power states, which include finer-grained states than the ACPI power states presented in Chapter 6.

The next section discusses power-management considerations for these low-power platforms, particularly some display power-management techniques for saving power. Then, software and hardware architectural considerations for power optimization are surveyed and various power-optimization approaches are offered. In the final section, we briefly mention low-power measurement considerations and metrics.

The Priorities for Low-Power Devices

In the past several decades, we have experienced exponential growth in terms of computer speed and density, in accordance with the well-known Moore's law. Ultimately, this trend will come to an end, as the growth is restricted by the realism of power dissipation and the limits of device physics. But for now, the power consumption of individual computing elements has yet to level off. Mobile computing devices, such as smartphones, laptops, and tablets, as well as home entertainment electronics like set-top boxes, digital cameras, and broadband modems, essentially follow this same trend, and so there is still room for power optimization.

On the other hand, increased functionalities for the mobile devices are becoming common for many low-power embedded systems, ranging from electronic controllers of household appliances to home energy-management systems, and from in-vehicle infotainment to sophisticated medical equipment. Not only must such devices consume very little power, they must also be "always on, always available."

Thus, fueled by insatiable consumer demand and intense competition, manufacturers of mobile devices continue to offer progressively more functions and faster clocks, but these are required to conform to a lower power budget for extended battery life. Furthermore, the advent of wearable computing devices, such as smart earphones and smartwatches, require extremely low-power processors within amazingly tiny forms. For example, a smartwatch with a surface area as small as 1600 mm^2 generally requires more than a week's battery life, ushering in the heralded *Internet of Things* (IoT).

Designers aim to keep reducing the packaging, manufacturing, operational and reliability costs of these devices while simultaneously supporting increasingly complex designs. Such ambitious goals generally materialize with the introduction of new silicon process technologies of finer geometry roughly every two years. However, every generation of process shrinkage results in higher power leakage, owing to increased gate and junction diode leakage. Although the dynamic power consumption scales down with process geometry, growing wire density tempers such reductions. Therefore, a simple scaling down of dynamic power alone is often insufficient for next-generation applications. The increased insistence on performance obligations for evermore complex applications imposes aggressive battery-life requirements, so this calls for more aggressive management of leakage and active power.

In general, the priorities for low-power devices moving from one process geometry to the next, smaller geometry include:

- *Decreasing the dynamic power.* This is possible as the dynamic capacitance and the voltage are both generally reduced for a smaller geometry.
- *Maintaining the total static leakage power.* This is an active area of focus for hardware architects because, due to process technology, the leakage power tends to increase for a smaller geometry. So it is necessary to optimize in this area with a view to maintaining the leakage power.
- *Keep active leakage at a small percentage (e.g., approximately 10-15%) of the dynamic power.* Also, leakage power must not be allowed to dominate the power consumption equation.

Let us recall the power-frequency relationships presented in Chapter 6. Figure 7-1 shows the dynamic and leakage power with respect to frequency.

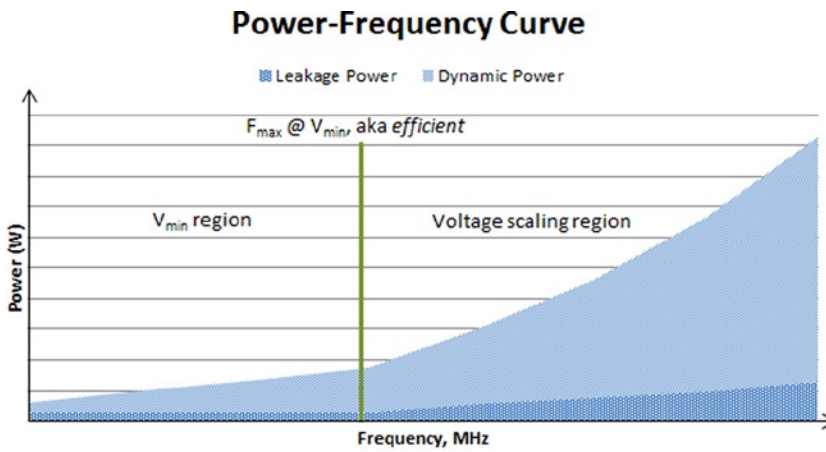

Figure 7-1. *Power-frequency relationship. For low-power design, the goal is to keep leakage power at ~10-15% of the dynamic power, even as the process shrinks to the next generation*

As is evident from Figure 7-1, there is an efficient frequency below which voltage and frequency scaling do not achieve good power reduction. In this V_{min} region, voltage only scales linearly with frequency, so leakage current, and consequently leakage power, becomes a determining factor for further power reduction. It is also necessary to point out that voltage cannot be arbitrarily reduced, as there is a minimum voltage needed to drive the circuit into the active state. Above the efficient frequency point, however, good voltage scaling can be achieved, as the voltage scales with frequency according to a cubic relationship. As a result, in this relatively higher-power region, power reduction can happen with an easy voltage-frequency tradeoff.

An example of mobile low-power platform architecture is shown in Figure 7-2. Power management and optimization are essential for each module of the architecture—namely the system-on-a-chip (SOC), storage module, input and output (I/O) module, sensors and cameras, controllers and communications module. In typical modern SoCs, there are dedicated circuits available for thermal control and power management—for example, the *power management controller* (PMC), which is discussed later.

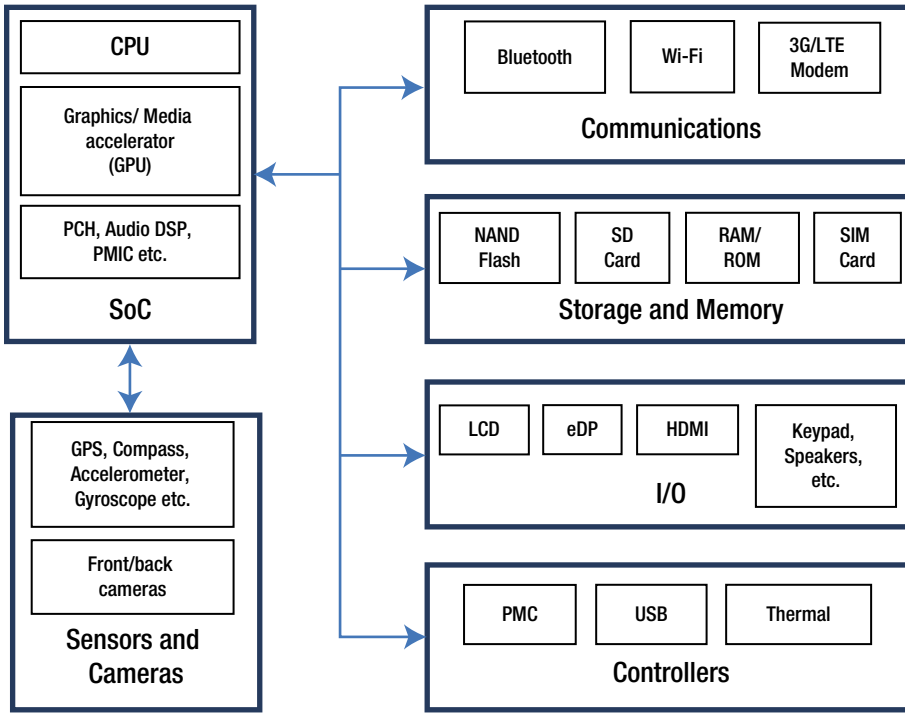

Figure 7-2. *An example of mobile platform architecture*

Typical Media Usage on Low-Power Platforms

Low-power computing platforms span a wide range of devices and applications, all of which need to save power while they are not in use, or only partially in use, regardless of whether media are involved. Table 7-1 lists some examples of low-power platforms.[1]

[1]B. Steigerwald, C. D. Lucero, C. Akella, and A. R. Agrawal, *Energy Aware Computing* (Hillsboro, OR : Intel Press, 2011).

Table 7-1. *Examples of Low-Power Platforms*

Category	Example	Special needs and requirements
Computing on the go	Smartphones, tablets, netbooks, wearable devices	Small form factor and limited storage capabilities; minimum heat dissipation and special cooling requirements; reduced memory bandwidth and footprint; multiple concurrent sessions; hardware acceleration for media and graphics; sensors; real-time performance; extended battery life
Medical equipment	Imaging systems, diagnostic devices, point of care terminals and kiosks, patient monitoring system	Amenable to sterilization; secure, stable and safe to health; resistant to vibration or shock; lightweight and portable; high-quality image capture and display capability; fanless cooling support, etc.
Industrial control systems	Operator-controlled centralized controller of field devices	Special I/O requirements, vibration and shock withstanding capabilities, variety of thermal and cooling requirements, ranging from fanless passive heat sink to forced air, etc.
Retail equipment	Point of sale terminals, self-checkout kiosks, automatic teller machines (ATMs)	Ability to withstand extreme ambient temperature, good air-flow design, security from virus attacks, non-susceptibility to environmental conditions (dust, rain etc.)
Home energy management systems	Centralized monitor of a home's usage of utilities such as electricity, gas and water; data mining of energy usage for reporting, analysis and customization (e.g., warning users when washing clothes is attempted during peak electrical rates or when a light is left on without anyone present, etc.)	Internet connectivity; various sensors; ability to control various smartphone apps; wireless push notification capability; fanless operation; ability to wake from a low-power or power-down state by sensors or signals from Internet; low power consumption while working as an energy usage monitor

(*continued*)

Table 7-1. *(continued)*

Category	Example	Special needs and requirements
In-vehicle infotainment systems	Standalone navigation systems, personal video game player, Google maps, real-time traffic reporting, web access, communication between car, home and office	Ability to withstand extreme ambient temperature; special cooling mechanisms in the proximity of heating and air-conditioning system; small form factor to fit behind dashboard; very low power level (below 5W); fast return from deep sleep (S3) state
Digital signage	Flight information display system, outdoor advertising, wayfinding, exhibitions, public installations	Rich multimedia content playback; display of a single static image in low-power state; auto display of selected contents upon sensor feedback (e.g., motion detection); intelligent data gathering; analysis of data such as video analytics; real-time performance
Special equipment for military and aerospace	Air traffic control, special devices for space stations and space missions, wearable military gears	Security; real-time performance; fast response time; extreme altitude and pressure; wearable rugged devices with Internet and wireless connectivity; auto backup and/or continuous operation
Embedded gaming	Video gaming devices, lottery, slot machines	High-end graphics and video playback; keeping attractive image on the screen in low-power state; various human interfaces and sensors; high security requirements; proper ventilation
Satellite and telecommunications	Scalable hardware and software in datacenters and base stations, throughput and power management and control	Compliance with guidelines requirements for environmental condition such as National Equipment Building Systems (NEBS); Continuous low-power monitoring
Internet of Things	Smart appliances, smart watches, smart earphones, smart bowl	Very low power for control and monitoring; Internet connectivity

Although the requirements are manifold from different low-power platforms, practical considerations include tradeoffs among processor area, power, performance, visual quality, and design complexity. It may be necessary to reduce nonessential functions or sacrifice visual quality while maintaining low power and keeping usability and elegance. This

approach would favor simplicity over performance, so as to support the energy-efficiency requirements for a specific device or application. Consequently, there are only a few media usages that are overwhelmingly commonplace in low-power platforms. Of particular importance are video playback and browsing over local and remote Wireless Display, video recording, and videoconferencing. Some details of these usages are discussed next.

Video Playback and Browsing

Video playback, either from local storage or streaming from the Internet, is one of the most prevalent and well-known media usage model for low-power devices. Video can be played back using a standalone application or via a browser, while the content is streamed from the Internet. As digital video is usually available in various compressed and coded formats, video playback involves a decompression operation during which the video is decoded before being played back on the display. If the resolution of the video is not the same as the display resolution, the decoded video is resized to match the display resolution.

Video decode and playback are quite complex and time-consuming operations when performed using the CPU only. Often, hardware acceleration is used to obtain higher performance and lower power consumption, as optimized hardware units are dedicated for the operations. In modern processors, hardware-accelerated playback is the norm, and it is essential for low-power platforms.

Figure 7-3 shows the software stack for a video playback usage model based on the Android operating system. The Media Extractor block demultiplexes the video and audio data. That data is fed to the corresponding Open MAX (OMX) decoder, which delivers the video bitstream to the LibMix library to begin decoding. The hardware decoder is used for the actual video decoding, which is driven by the media driver. The decoded video buffers are delivered to the Android audio-video synchronization (AVsync) software module. By comparing the video buffer's *presentation time stamp* (PTS) and the audio clock, AVsync queues the buffers to the Surfaceflinger to be rendered on the display at the appropriate time.

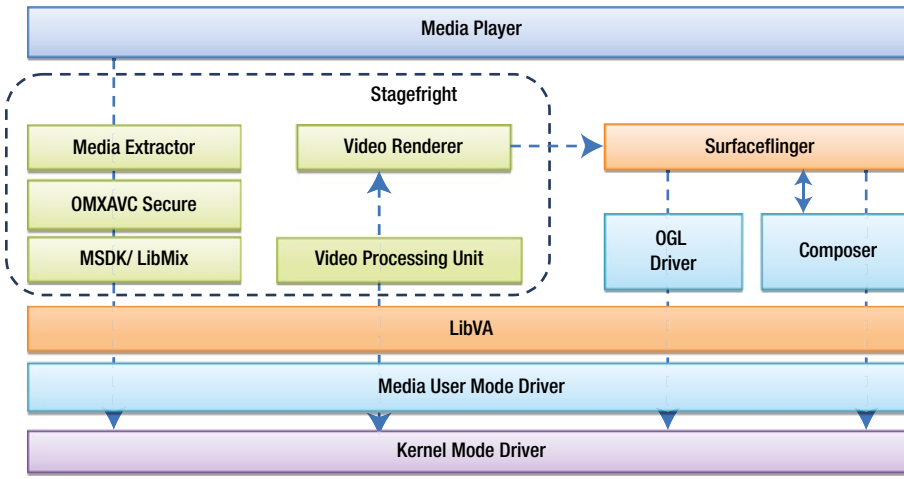

Figure 7-3. *Video playback software stack on Android*

Figure 7-4 shows a block diagram of the decode process in video playback. This process is typically defined by an international standards study group, and all compliant decoders must implement the specified definitions of the codec formats. Careful optimizations at various hardware and software levels are necessary to obtain a certain performance and power profile, especially on low-power platforms. Optimization approaches include implementation of repetitive operations on special-purpose hardware, optimized memory load/store/copy, cache coherency, scheduling, load balancing of tasks in various hardware functional units, optimized post-processing, opportunistically entering power-saving states, and so on.

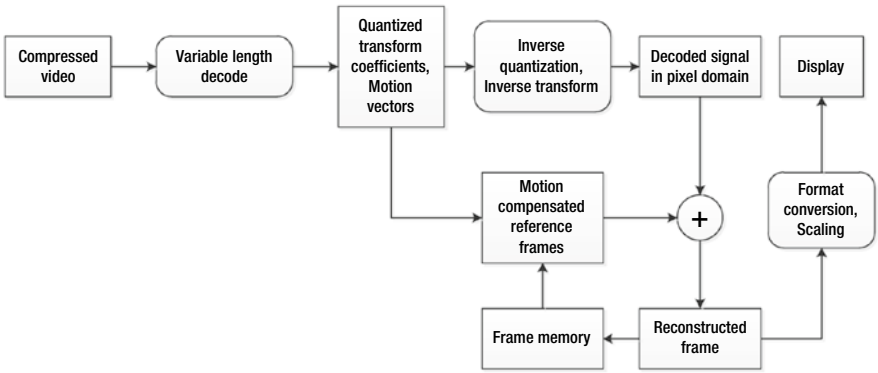

Figure 7-4. *Block diagram of video decode process*

In Figure 7-5, an example activity profile of the video playback usage is presented for AVC 1080p30 and 1080p60 resolutions. In this Intel Architecture-based low-power platform, a 1080p playback generally requires ~10-25% CPU and ~15-30% GPU activity depending on the frame rate. While the display unit consumes about half of the platform power, the SoC consumes ~20-30%. Notable power loss occurs at the voltage regulator (~15%), memory (~5%), and other parts of the platform.

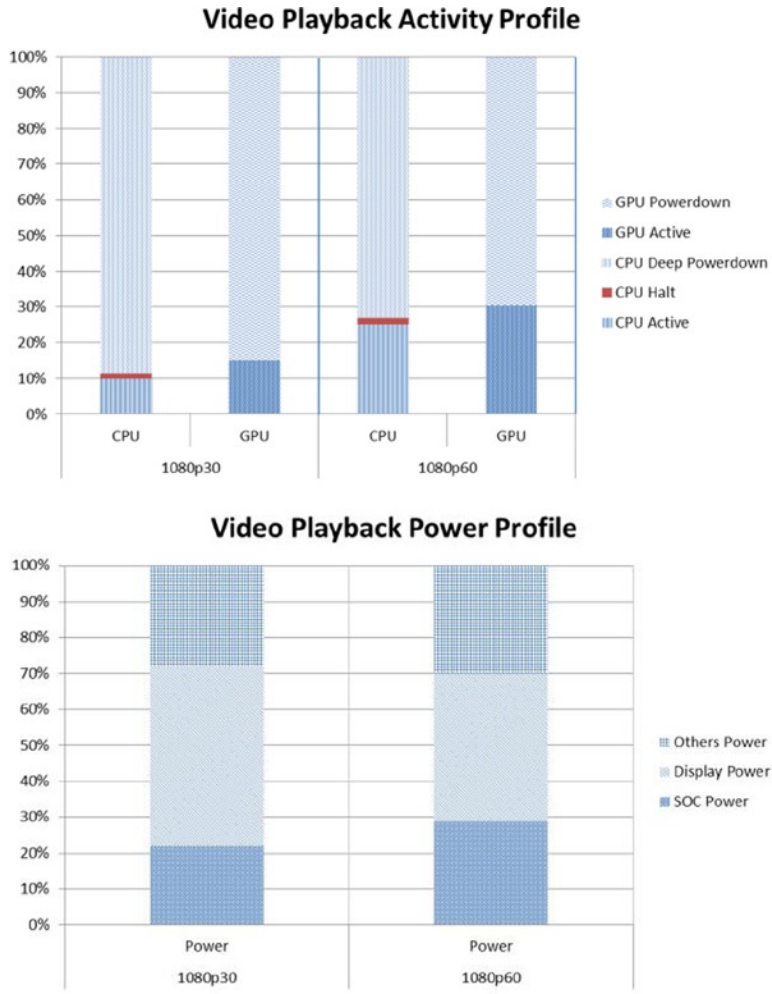

Figure 7-5. *Example of video playback activity and power profile*

Video Recording

In this usage model, the user captures a video of 1080p resolution with the device's main camera while the audio is captured with the integrated microphone. The output of the camera (usually compressed) is decoded and pre-processed with de-noising and scaling as needed. The resulting uncompressed source is encoded using hardware acceleration. Encoded and multiplexed video and audio streams are stored in local files. (While preview mode is fairly common for this usage model, for simplicity it is not considered in the following software stack, as shown in Figure 7-6.)

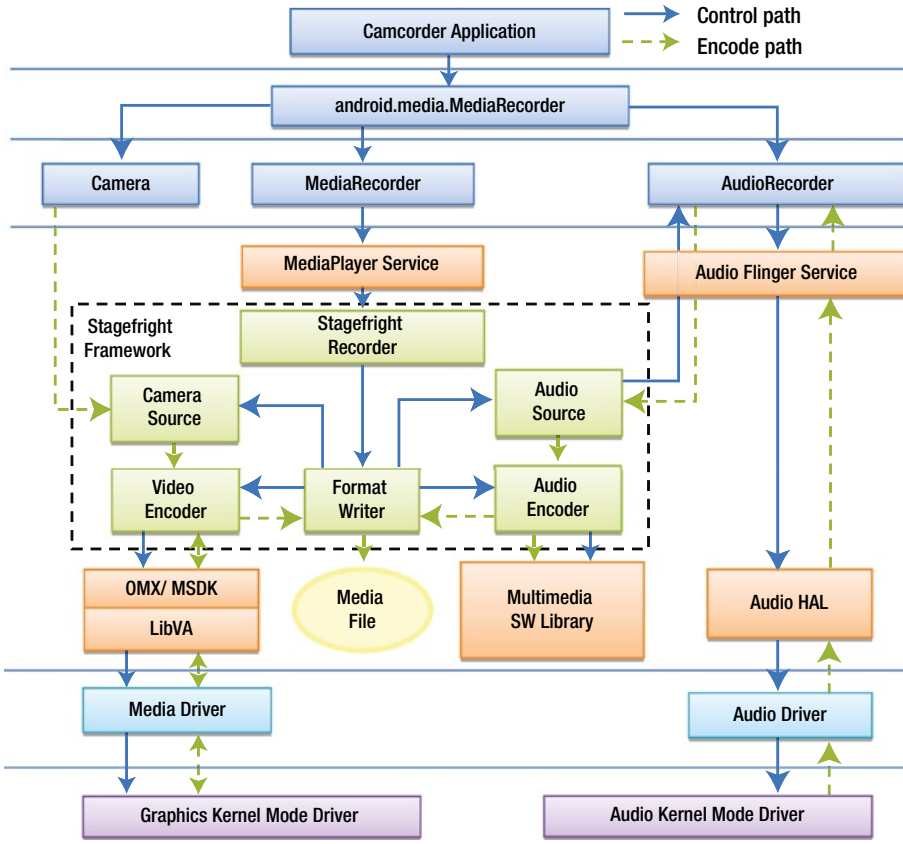

Figure 7-6. *Video recoding software stack on Android*

In Figure 7-7, an example video recording activity profile is presented for AVC 1080p30 and 1080p60 resolutions on an Intel Architecture platform. On this platform, recording a 1080p video generally requires ~40-60% of CPU activity, as well as ~30-60% of GPU activity depending on the frame rate. The SoC and the display unit both consume about a third of the platform power. Similar to the playback usage, significant power dissipation occurs in the voltage regulator (~15%), memory (~7%), and other parts of the platform. Note that compared to the playback usage, GPU activity is higher because of the heavier encode workload.

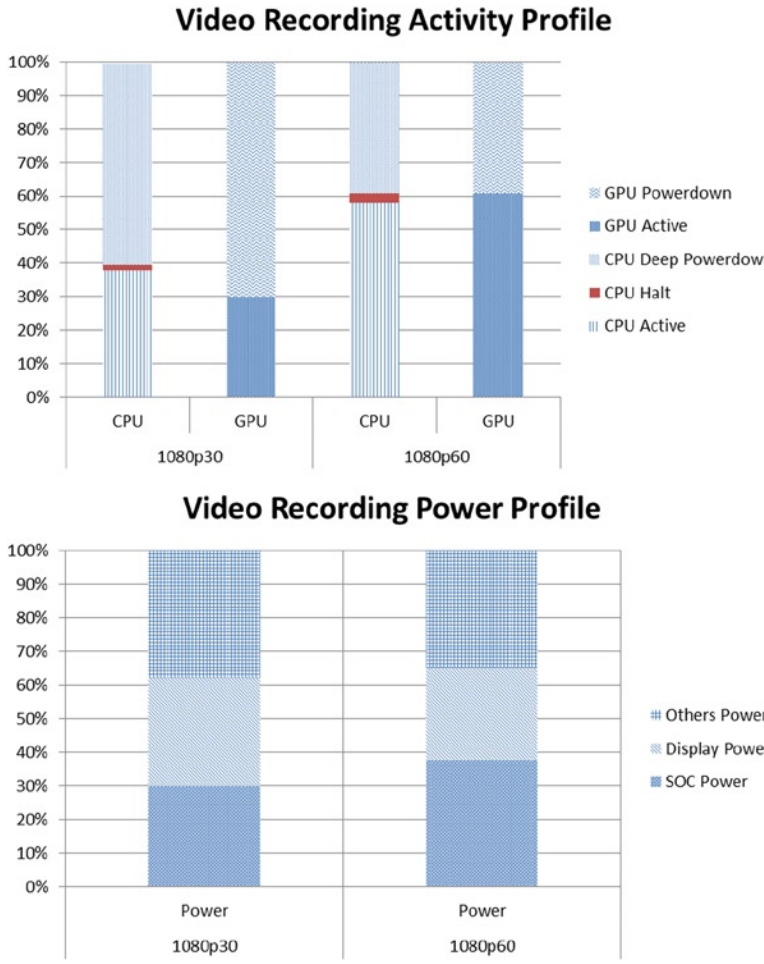

Figure 7-7. *Example video recording activity and power profile*

Video Delivery over Wireless Display and Miracast

In the video delivery over Wireless Display (WiDi) and Miracast usage, the device's Wi-Fi connection is used to stream encoded screen content captured from a local display to a remote HDTV or monitor. WiDi supports wireless transport of the associated audio content as well. In particular, the platform's video and audio encoding hardware acceleration capabilities are exploited to generate an audio-video stream encapsulated in Wi-Fi packets that are streamed over a peer-to-peer connection to a WiDi adaptor device connected to a remote display. Owing to the multi-role support of Wireless Local Area Network (WLAN), multiple connections may simultaneously be available at a wireless access point, serving the WiDi peer-to-peer connection as well as a dedicated connection to the Internet while sharing the same frequency. The multi-role allows, for example,

browsing of the Internet in clone or multi-tasking modes, as described below. Playback of video or browsing of the Internet is commonly protected by a digital rights management (DRM) protocol over the WiDi. An overview of a WiDi complete solution and the Miracast industry standard was presented in Chapter 6.

WiDi supports two main usage models:

- Clone mode, where the same content is presented on both a local and a remote display. The local display's resolution may be modified to match the remote display's maximum resolution. Also, if the WiDi performance is insufficient, the frame rate of the remote display may be downgraded.
- Extended mode, where there is remote streaming of a virtual display and the content is not shown on the device's embedded local display. There are two scenarios for the extended display:
  - Extended video mode shows the content on the remote display, while local display displays only the UI controls.
  - Multi-tasking is allowed, where a video is shown on the remote display while an independent application such as a browser may also run and show content on the local display.

Ideally, the platform should be designed so that there is no performance degradation when wireless display is activated.

Figure 7-8 shows the WiDi flow diagram for an Intel Architecture platform. One or more screen content surfaces are captured, composited, scaled, and converted to a format appropriate to the hardware-accelerated video encoder, while the audio is independently captured and encoded as well. The encoded video and audio bitstreams are then encrypted according to the HDCP2 protocol and the encrypted bitstreams are multiplexed and packetized to produce MPEG-2 transport stream packets that are ready to send over the Wi-Fi channel.

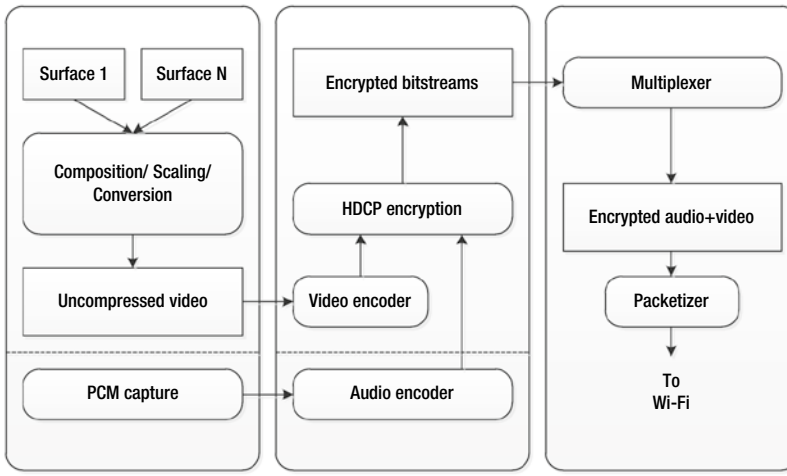

Figure 7-8. *Wireless Display flow diagram on an Intel Architecture platform*

Figure 7-9 shows the activity profile of three WiDi scenarios:

- Clone mode while the device is idle–that is, the local display shows a mostly static screen.
- Video playback in extended mode where the video is played back in the remote display.
- Multi-tasking in extended mode where a browser (or some other window) is independently presented on the local display while a video is played back in the remote display.

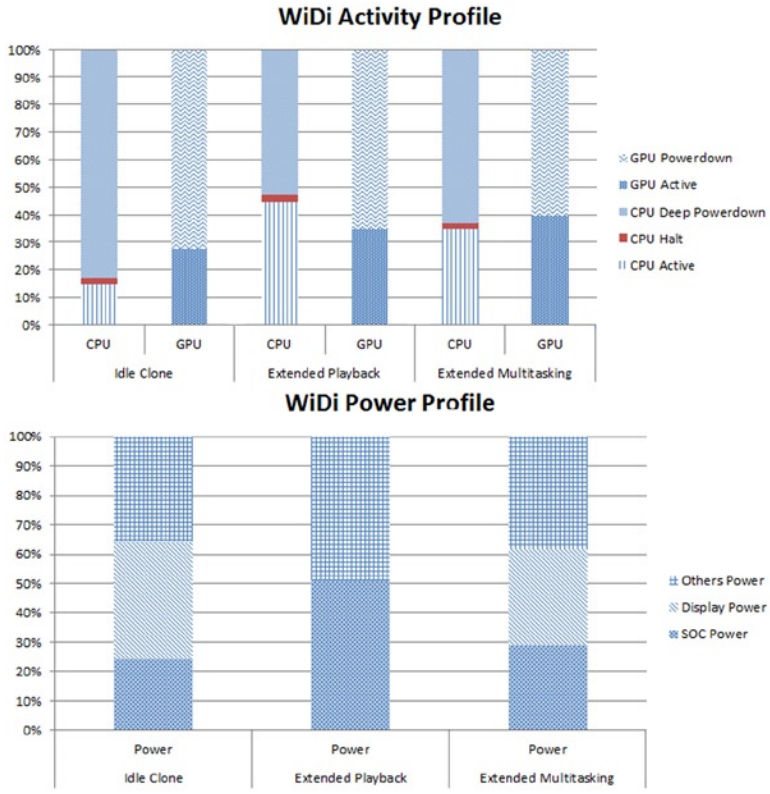

Figure 7-9. *Example WiDi activity profile*

For the idle clone scenario, depending on the display refresh rate, the screen content may be captured and encoded at 720p60 or 1080p30 resolution using hardware acceleration. The encoded video bitstream is typically encrypted, multiplexed with the audio bitstream, and divided into transport packets before transmission using the Wi-Fi protocol.Hardware-accelerated video —pre-processing may also be done before composition, scaling, and conversion to the uncompressed format that is fed to the encoder. When a video is played back in the clone mode, simultaneous decode, capture and encode operations happen using hardware-acceleration. For the extended mode scenarios, the local display does not show the video, the content is only shown in the extended wireless display; no composition, scaling, or format conversion is done and only decoding and encoding are performed, In all cases the audio is typically encoded in the CPU.

On the Intel Architecture platform, WiDi usages generally require ~15-45% of the CPU activity and ~30-40% of GPU activity, depending on the WiDi scenario. While the local display does not consume power in extended video mode, the SoC consumes about half of the platform power, the other half being distributed to the rest of the platform. For other WiDi modes, the platform power is more or less equally divided among the display, the SoC, and the rest of the platform.

Videophone or Videoconferencing

Multi-party videoconferencing is a generalized case of a two-party videophone or video chat, which uses a camera that is integrated with a device for capturing a video, usually in motion JPEG (MJPEG) or other compressed formats. Using the device's hardware-acceleration abilities, the camera output is then decompressed (if the camera output is compressed), processed, and re-encoded to a low-delay format at a dynamically varying bit rate with some error-recovery capabilities. This is basically video playback and video recording usage models concurrently applied, with the following stipulations:

- Both encode and decode operations must happen simultaneously in real time, usually accelerated by hardware.
- The camera output should be input to the encoding unit in real time as well. The camera output frame rate should be in concert with the encoding frame rate.
- The end-to-end delay from camera capture to packetized bitstream output should be constant.
- The video elementary streams for both incoming and outgoing bitstreams should be appropriately synchronized with the corresponding audio elementary streams.

For multi-party videoconferencing, usually in the local display there is one main video window and several thumbnail videos of the multiple parties involved. Figure 7-10 illustrates a flow diagram of typical videoconferencing. Assuming the camera capture format is a compressed one, the captured video is decoded, scaled, and/or pre-processed before encoding to a suitable format such as AVC with appropriate bit rate, frame rate, and other parameters to obtain good tradeoffs among video quality, delay, power consumption, and amount of compression. Also, the incoming video bitstreams are decoded and composed together before display. All these typically are done using hardware acceleration. Multiplexing-demultiplexing and packetization-depacketization are generally done in the CPU, while audio can be processed by special hardware units or audio drivers and kernels.

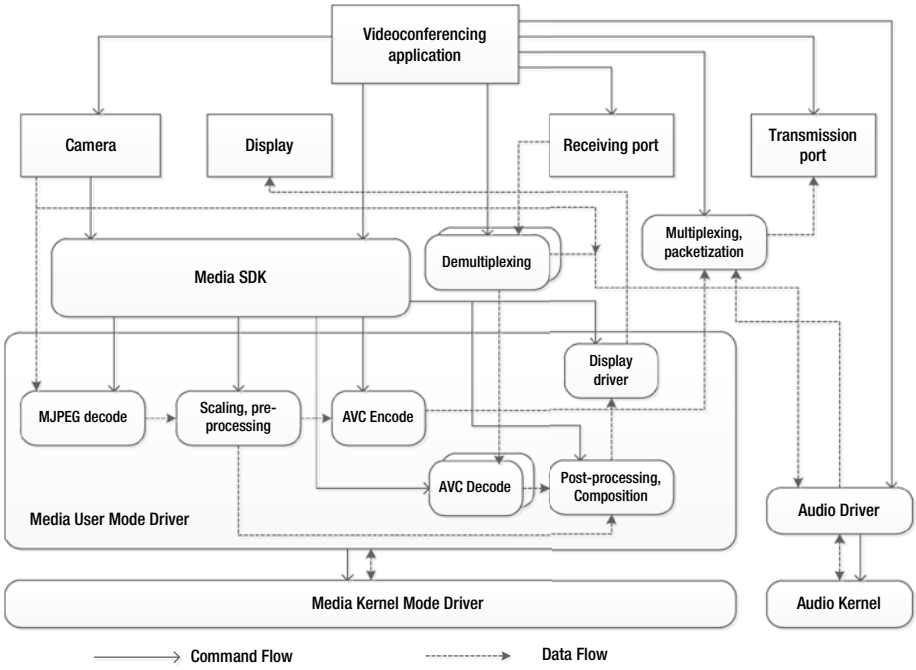

Figure 7-10. *Flow of a typical videoconferencing event*

The activity profile of the videoconferencing usage is similar to the WiDi usage. In both cases, simultaneous encode and multiple decodes are involved; encoding is done with low delay and variable bit rates amenable to changes in network bandwidth, and there are real-time requirements for both cases as well.

System Low-Power States

The goal of low-power platform design, from both hardware and software points of view, is to successfully meet the challenges posed by limited battery life, heat dissipation, clock speed, media quality, user experience, and so on while addressing the ever-increasing need to support complex multi-tasking applications. To this end, as was detailed in Chapter 6, the ACPI defines several processor states to take advantage of reduced power consumption when the processor is not fully active. However, the ACPI model of power management is not sufficient for modern mobile applications, especially with the requirement of "always on" connectivity. To address this issue, new low-power states have been defined, but they present some problems.

Drawbacks of the ACPI Simple Model

The ACPI simple model of power management has a few limitations:

- The model assumes that the operating system or the power manager will manage the power. However, this is not always the case. Some devices, such as disk drives, CPUs, and monitors, manage their own power and implement power policies beyond what is achievable by the ACPI model.
- The four device states are not sufficient. For example, D3 has two subsets. Also, a processor in D0 state has additional CPU (*Cx*) states and performance (*Px*) states, as described in the next section. In addition, a device may perform multiple independent functions and may have different power states for each function.
- Recent advances in the operating system impose new requirements on power management. For example, the concept of *connected standby* requires the system to turn off all activities except that of listening for an incoming event, such as a phone call.

Recognition of these drawbacks resulted in special S0iX states within the S0 state in Windows. These are intended for fine-tuning the standby states. A higher integer X represents higher latency but lower power consumption.

Connected Standby and Standby States

Connected Standby (CS) mimics the smartphone power model to provide an instant on/instant off user experience on the PC. Connected Standby is a low-power state in Windows 8 and later versions that uses extremely low power while maintaining connectivity to the Internet. With CS, the system is able to stay connected to an appropriate available network, allowing the applications to remain updated or to obtain notification without user intervention. While traditional Sleep state (S3) has wake latency of more than two seconds and Hibernate (S4) may take indefinitely longer, CS-capable ultra-mobile devices typically resume in less than 500 ms.

When the system goes into CS, the OS powers down most of the hardware, except the bare minimum required to preserve the contents of DRAM. However, the Network Interface Controller (NIC) still gets a trickle of power so it is able to scan incoming packets and match them with a special wake pattern. To preserve the connection, the NIC wakes the OS as needed. The OS also wakes up every few hours to renew its DHCP lease. Thus, the system maintains its layer-2 and layer-3 connectivity, even while the NIC is mostly powered down.

In addition, the system can be wakened in real time. Consider when a Skype call arrives and the system needs to quickly start the ring tone. This works through special wake-pattern packets. The Skype server sends a packet on a long-running TCP socket. The NIC hardware is programmed to match that packet and wake the rest of the OS. The OS receives and recognizes the Skype packet and starts playing the ring tone.

Connected Standby has the following four elements:

- Most of the hardware is in a low-power state.
- The NIC wakes the OS when the NIC needs OS intervention to maintain layer-2 connectivity.
- The OS periodically wakes the NIC to refresh its layer-3 connectivity.
- The NIC wakes the OS when there's a real-time event (e.g., an incoming Skype call).

Connected Standby platforms are usually designed with SoCs and DRAMs that have the following characteristics:

- *Less than 100 ms switch time between idle and active modes.* The active mode allows running code on the CPU(s) or the GPU, but may not allow accessing the storage device or other host controllers or peripherals. The idle mode may be a clock-gated or power-gated state, but should be the lowest power consumption state for the SoC and DRAM.
- *Support of self-refresh mode for the DRAM to minimize power consumption.* Typically mobile DRAM (LP-DDR) or low-voltage PC DRAM (PC-DDR3L, PC-DDR3L-RS) is used.
- *Support of a lightweight driver called Power Engine Plug-in (PEP) that abstracts SoC-specific power dependencies and coordinates device state and processor idle state dependencies.* All CS-capable platforms must include a PEP that minimally communicates to the operating system when the SoC is ready for the lowest power idle mode.

The process of preparing the hardware for low-power during Connected Standby can be visualized as an upside-down pyramid, as shown in Figure 7-11. The lowest power is achieved when the whole SoC is powered down, but this can occur only when each set of devices above it has been powered down. Therefore, the latency for this state is the highest within Connected Standby.

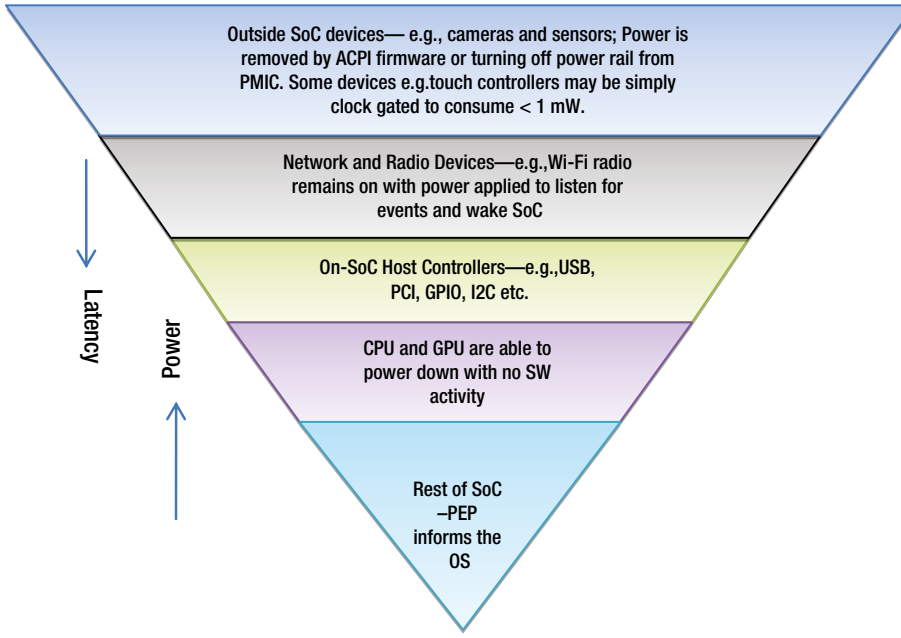

Figure 7-11. *Preparation for transitioning to low-power state during Connected Standby*

The S0iX states are the latest standby states, which include S0i1, S0i1-Low Power Audio, S0i2, and S0i3. These states are available on recent Intel platforms and are enabled for all operating systems. They are special standby states that allow the system to consume a trickle of power, and so they are crucial for Connected Standby to work.
A device enters CS when the off/on button is pushed or after an idle time-out; while in CS mode, it consumes very low power. For example, at CS mode, an Intel Z2760-based device consumes <100 mW, and can stay in this mode for more than 15 days without requiring a recharge. Figure 7-12 shows the flow of actions in CS mode.

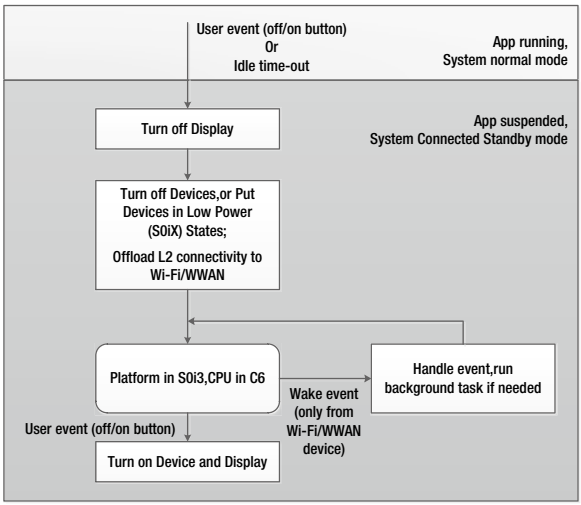

Figure 7-12. *Activity flow for Connected Standby*

Contrast this with ACPI S3 state, where all activities of the processor are paused when the system is sleeping and activities resume only upon signals from keyboard, mouse, touchpad, or other I/O devices. Connected Standby automatically pauses and resumes activity on the system in a tightly controlled manner, using the various S0iX states to help ensure low power and long battery life. A typical system with a 45 Watt-hour battery may achieve 100 hours of battery life in S3 state, while in S0i3 the battery life becomes three times as long.

Figure 7-13 shows a comparison of the S0i3 state with respect to the S0 and the S3 states. The S0i3 essentially gives the best of both power consumption and latency: it is less power consuming than the S0 (Active) state, consuming almost the same power as the S3 (Sleep) state while yielding wake latency in the order of milliseconds, similar to the S0 state.

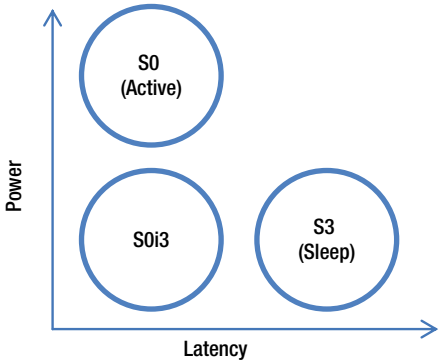

Figure 7-13. *Relative power-latency relationship for S0i3 compared to traditional ACPI Sleep (S3) state*

Table 7-2 provides estimates of the entry and exit latency for each of the low-power standby states on an Intel Architecture platform. Especially on Windows 8 and Windows 8.1 operating systems, these quick latencies translate to an exceptionally snappy wake performance from a user experience point of view, while delivering significant power savings.

Table 7-2. *Estimated Entry and Exit Latency of Standby States of an Intel Architecture Platform*

State	Entry Latency	Exit Latency
S0i1	~200 µsec	~400 µsec
S0i2	~200 µsec	~420 µsec
S0i3	~200 µsec	~3.4 msec

Combination of Low-Power States

Table 7-3 lists the combined system low-power states and ACPI power states for a typical low-power processor. The ACPI power states were described in Chapter 6, so only the effect of S0iX is added here for a clearer understanding.

Table 7-3. *Definitions of System Low-Power States*

State or Substate	Description
G0/S0/PC0	Full on. CPUs are active and are in package C0 state.
G0/S0/PC7	CPUs are in C7 state and are not executing with caches flushed; controllers can continue to access DRAM and generate interrupts; DDR can dynamically enter deep self-refresh with small wake-up latency.
G0/S0	Standby ready. CPU part of the SoC is not accessing DDR or generating interrupts, but is ready to go standby if the PMC wants to start the entry process to S0iX states.
G0/S0i1	Low-latency standby state. All DRAM and IOSF traffic is halted. PLLs are configured to be off.
G0/S0i1 with audio	Allows low-power audio playback using the low-power engine (LPE), but data transfer happens only through specific interfaces. Interrupt or DDR access request goes through the PMC. The micro-architectural state of the processor and the DRAM content are preserved.
G0/S0i2	Extended low-latency standby state. S0i2 is an extension on S0i1—it *parks* the last stages of the crystal oscillator and its clocks. The DRAM content is preserved.

(*continued*)

Table 7-3. *(continued)*

State or Substate	Description
G0/S0i3	Longer latency standby state. S0i3 is an extension on S0i2—it completely stops the crystal oscillator (which typically generates a 25 MHz clock). The micro-architectural state of the processor and the DRAM content are preserved.
G1/S3	Suspend-to-RAM (STR) state. System context is maintained on the system DRAM. All power is shut to the noncritical circuits. Memory is retained, and external clocks are shut off. However, internal clocks are operating.
G1/S4	Suspend-to-Disk (STD) state. System context is maintained on the disk. All power is shut off, except for the logic required to resume. Appears similar to S5, but may have different wake events.
G2/S5	Soft off. System context is not maintained. All power is shut off, except for the logic required to restart. Full boot is required to restart.
G3	Mechanical off.

Source: Data Sheet, Intel Corporation, April 2014. `www.intel.com/content/dam/www/public/us/en/documents/datasheets/atom-z36xxx-z37xxx-datasheet-vol-1.pdf`.

Power Management on Low-Power Platforms

The main power-management tasks are done by the operating system—for example, taking the idle CPU core offline; migrating and consolidating tasks, threads, and processes to a minimum number of the cores to allow other cores to go idle; limiting frequent back-and-forth switching of tasks between cores for variable workloads; managing tradeoffs between power gain and hardware or software latencies; load balancing between the active cores, and so on. In addition to the power-management features provided by the operating system according to the ACPI standard, there are several hardware and software approaches to power management, all with a singular goal of achieving the most power savings possible. We discuss some of these approaches next.

Special Hardware for Power Management

As power management is crucial for low-power platforms, in most contemporary devices there are special-purpose hardware units dedicated to these power-management tasks. Here are two such units.

Power Management Circuits

With the increase in system complexity comes the need for multiple sources of power at various voltage levels. Such need cannot be fulfilled by voltage regulators alone; power management ICs (PMICs) have been introduced for greater flexibility. Power Management ICs (PMICs) are special-purpose integrated circuits (or a system block in a SoC device) for managing the power requirements of the host system. A PMIC is often included in battery-operated devices such as mobile phones and portable media players. Typical PMIC efficiency is 80 to 85 percent, depending on idle or active states of applications. A PMIC may have one or more of the following functions:

- Battery management
- DC-to-DC conversion
- Voltage regulation
- Power-source selection
- Power sequencing
- Miscellaneous other functions

Power management is typically done with the following assumptions:

- All platform devices are ACPI compliant.
- SOC devices are PCI devices except GPIOs.
- The PCI device drivers directly write to PMCSR register to power down/power up the device; this register triggers an interrupt in the PMC.
- The device drivers use ACPI methods for D0i3/D3 entry/exit flows.

Power-Management Controller

The drivers and the OS alone are not adequate for power-management tasks such as save and restore context or handling special wake events. A special microcontroller, called the Power Management Controller (PMC), is typically used for powering up and powering down the power domains in the processor. It also supports the traditional power-management feature set along with several extended features. For example, in Intel fourth-generation Atom SoCs, the PMC performs various functions, including:

- Power up the processor and restore context.
- Power down the processor and save context.
- Handle wake events for various sleep states.
- Secure the boot.
- Perform low-power audio encode (using LPE) and general-purpose input-output (GPIO) control in order to be able to go to S0i1 when LPE is active.

The PMC is also used in ARM11 processors in power-saving mode,[2] where it determines (for instance, when instructed by the processor) that the processor should be put into dormant or shutdown mode; it asserts and holds the core reset pin; it removes power from the processor core while holding reset; and so on.

Display Power Management

The display is the major power-consuming component in most low-power platforms. Several power-management features have been implemented to manage and optimize the display power in various platforms. For example, in recent Intel platforms, the following are among the many display power-management features:

Panel Self-Refresh

The panel self-refresh (PSR) feature, typically used for *embedded display port* (eDP) displays, allows the SOC to go to lower standby states (S0iX) when the system is idle but the display is on. The PSR achieves this by completely eliminating the display refresh requests to the DDR memory as long as the frame buffer for that display is unchanged. In such cases, the PSR stops the display engine from fetching data from external memory and turns off the display engine.

Display Power-Saving Technology

The display power-saving technology (DPST) is an Intel backlight control technology. In recent versions, the host side display engine can reduce up to 27 percent of the panel backlight power, which linearly affects the energy footprint. The Intel DPST subsystem analyzes the frame to be displayed, and based on the analysis, changes the chroma value of pixels while simultaneously reducing the brightness of backlight such that there is minimum perceived visual degradation. If there is considerable difference between the current frame being displayed and the next frame to be displayed, new chroma values and brightness levels are calculated.

The DPST needs to be enabled by the display driver; the DPST cannot yet work in parallel with the PSR. If a MIPI[3] panel does not have a local frame buffer, then DPST should be enabled and the PSR can be disabled. In general, the DPST should be enabled for additional power savings for bridge chips that do not have local frame buffers.

[2]Details available at infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0143c/CHDGDJGJ.html.

[3]The MIPI (Mobile Industry Processor Interface) is a global industrial alliance that develops interface specifications, including signaling characteristics and protocols, for the mobile communication industry. Details are available at www.mipi.org.

Content-Adaptive Backlight Control

A liquid crystal display (LCD) consists of a backlight that shines through a filter. The filter modulates the backlight and is controlled by the pixel values that are to be displayed: the brighter the pixel, the more is the light that passes through. By controlling the array of pixels in an LCD filter, images of different brightness can be shown on the display. Such control is typically based on a power-brightness tradeoff.

Content-adaptive backlight control (CABC) takes advantage of the fact that the perceived brightness is due to the backlight being modulated by the filter. A dark frame appears dark because the filter does not allow much light to shine through. However, the same effect can be achieved by using a dimmer backlight and controlling the filter to allow more light to shine through. In other words, the backlight is dimmed and the brightness of the frame is boosted, resulting in the same perceived brightness.

However, reducing the brightness of the backlight decreases the available dynamic range. The CABC may clip pixels requiring a higher dynamic range, producing a washout effect. Also, it is possible that the target backlight brightness varies greatly from frame to frame; aggressively changing the backlight by such large amounts can result in flickering. Therefore, a tradeoff is necessary between CABC-based power savings and perceived visual quality. To get the best performance, typically separate histogram analysis is done for each color component to determine the desired brightness. Then the maximum of these is chosen as the target backlight. This ensures that all colors are accurately rendered.

The CABC engine resides inside either the display bridge or the panel. With the ability to dynamically control the backlight as the content changes, this feature looks for opportunities to save power by reducing the backlight power. The algorithm processes and analyzes the current frame in order to update the backlight for the next frame. This can work in parallel with the PSR. Power savings from CABC are bigger when playing higher FPS video.

Ambient Light Sensor

The Ambient Light Sensor (ALS) backlight power savings modulates the backlight based on surrounding light. More power savings (e.g., up to 30%) occur when there's less ambient light and there's less power savings (e.g., 20%) when there's more ambient light. The display power management typically uses ALS inside the panel or the bridge.

Low-Power Platform Considerations

It is easy to understand, from the power-frequency relationship (Figure 7-1), that there is an optimal frequency for a given power budget. With this fundamental concept, there are some software and architectural considerations to keep in mind, particularly from the point of view of limited power platforms.

Software Design

Regarding the resource-scarce wearable and ultra-mobile low-power platforms, there are several important ideas to evaluate from the application design point of view. It is a no-brainer that new applications take these considerations into account; however, existing applications can be modified to benefit from them as well.

Given that most low-power devices offer various power-saving opportunities, applications should exploit those prospects appropriately. For example, allowing networking devices to go to a lower power state for longer period may avoid unnecessary network traffic, as will using a buffering mechanism. But here are some additional suggestions for power savings.

Intelligent Power Awareness

Low-power designs typically trade performance for power savings, using techniques such as voltage scaling or frequency scaling, thereby reducing the voltage or the frequency, respectively, to reduce the dynamic power consumption. However, a minimum voltage level must be supplied for the circuit to be operational. Further, reducing the supply voltage increases circuit delay, which restricts the operating frequency of the processor; the delay cannot be too large to satisfy the internal timings at a particular operating frequency. Thus, a careful approach is necessary to determine the appropriate amount of voltage scaling, as demonstrated by the Intel (R) Speed Step (TM) voltage scaling technology.

Unfortunately, system performance requirements in modern processors are too high to make voltage scaling an attractive energy-saving solution. Also, processor frequency is not the only factor that affects the power-performance behavior of complex multimedia applications; memory access latency and memory bandwidth are also important factors. Besides, multimedia applications have real-time deadlines to meet, which further restrict the usefulness of voltage scaling. Therefore, in practical systems, the voltage and frequency need to be dynamically adjusted along the voltage-frequency curve.

Dynamic adjustment of voltage and frequency should normally be the job of the operating system. However, the operating system does not have knowledge of the expected workload, especially when bursty multimedia applications are involved. Also, such workloads are not easy to predict with reasonable accuracy and oftentimes interval-based schedulers are not sufficient. At best, the operating systems may use some heuristics to change the processor frequency and try to control power consumption. But this answer does not work well for bursty workloads such as video decoding, encoding, or processing.

If the application is power-aware, however, the actual workload can be predicted better in a dynamic fashion; and by doing so, a power reduction of over 30 percent is possible for a typical video decoding.[4] The power consumption of applications largely depends on the number of clock cycles needed for instructions and memory references. If applications can provide information about their future demands to the OS, the OS can then perfom more efficient scheduling of tasks and does not need to work with questionable predictions.

---

[4]J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low Power Microprocessor," *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking* (2001): 251–59. Association for Computing Machinery.

This application is particularly useful on resource-scarce mobile devices. Applications must be aware of their processing demands—in particular, the required number of clocks until the next deadline of the application's current task—and must inform the OS of those clock requirements. The OS can then take into account the cycle count along with the power-frequency curve to regulate and achieve the minimum frequency at which the application deadline can be met, thereby achieving optimal power consumption.

For example, for an H.263 decoding, the number of bits used in a coded frame is fairly proportional to the decoding time. Using a hint of the number of bits used in each frame, an application can determine the expected frequency requirement for a video and obtain a power saving of about 25 percent, as reported by Pouwelse et al.[5]

Often there is power-saving firmware logic to control the host system's power draw. However, the firmware does not have knowledge of the work that the application is trying to do, and threfore is unable to optimize the energy investments, as it must rely on measurements and look for opportunities to save the energy. An intelligent power-aware application, on the other hand, can generally use its knowledge of the task and the memory-utilization pattern to send hints to the firmware for further power saving.

In an example of such performance-aware power saving, Steigerwald et al.[6] mentions an online transaction-processing application, where the application monitors the performance counters of the transaction processing to determine the performance impact. This information is provided to the middleware, which learns and updates the optimal power-limiting policy on the system. At the lowest level, a power controller firmware component caps the power consumption of the system.

Quality Requirements

Although it's common to trade performance and visual quality for power saving, users of most modern games and media applications have a high quality expectation that must be met. It is possible to run the processor at a low frequency and lower the power consumption, but that's not acceptable for various games and video codec applications, as the frame-processing deadline would be missed. That would mean the frames would not be displayed in real time, resulting in dropped frames and a poor user experience. However, it is possible to meet the quality requirements while still saving power by carefully considerating both aspects together when designing and optimizing the software.

Hardware-Accelerated Blocks

Recent low-power processors of mobile handheld devices have come with various hardware blocks capable of specific tasks, such as camera image signal processing or video decoding. Often these hardware blocks are optimized for lower power consumption than if the task were performed on a general-purpose CPU. Applications should take advantage of these hardware blocks and offload as much processing as possible in order to lower the computational burden and save power consumption.

---

[5]Ibid.

[6]Steigerwald et al., *Energy Aware Computing*.

Energy-Efficient User Interfaces

From the various usage models described earlier, it is evident that the display is the most power-consuming unit of the platform. However, it is possible to design graphical user interfaces (GUI) in a energy-efficient manner, particularly for liquid crystal displays (LCD) or displays based on organic light-emitting diodes (OLED).[7] For these displays, different colors, color patterns, or color sequences consume different amounts of power, and thus, low-energy color schemes can be used to reduce power. For example, thin film transistor (TFT) LCDs, which are typically utilized in personal digital assistants (PDA), consume less power when black than when white.

Another way toward energy efficient GUI design is to improve the latency caused by interfaces with humans by using fewer and larger buttons for greater user convenience.

Code Density and Memory Footprint

With low-power mobile platforms such as smartphones, poor code density means that the code size of the firmware and driver required to run on the processor is too large. As a result, the device manufacturer has to use more RAM and flash memory to hold the firmware, compared to what optimized firmware would need. Thus, more memory is used resulting in higher cost and reduced battery life.

Since memory cost dominates the cost and power budget of the entire design, firmware optimization and code density can make a big difference. Optimization at the compiler level, and possibly instruction set architecture (ISA), can achieve much denser firmware binaries, thereby helping reduce power consumption.

Another way to shrink the memory footprint of binaries is to perform a lossless compression while writing to memory and the correspoding decompression when reading back from memory. However, the power savings would depend on the nature of the workload. As media workloads have good locality characteristics, it may be beneficial to use memory compression for certain media usages.

Optimization of Data Transfer

Modern mobile applications designed for low-power devices often benefit from available Internet connectivity by processing a large amount of data in the cloud and maintaining only a thin client presence on the device. While such an approach may boost performance, the large number of data transfers means the power consumption is usually high. But by exploiting the pattern of the data to be transferred, it is possible to reduce the data bandwidth usage, thereby also reducing power consumption. Furthermore, applications can take advantage of the new S0iX states for notification purposes, and allow the processor to go to lower power states more frequently.

[7] K. S. Vallerio, L. Zhong, and N. K. Jha, "Energy-efficient Graphical User Interface Design," *IEEE Transactions on Mobile Computing* 5, no. 7 (July 2006): 846–59.

Parallel and Batch Processing

Scheduling of tasks is a challenge in multi-tasking environments. Serialization of tasks over a long period of time does not provide the processor with an opportunity to go to lower power states. Parallelization, pipelining, and batch processing of tasks can grant the processor such opportunity and thereby help lower power consumption. If a task does not have specific processing rate, it may be useful to run the processor at the highest frequency to complete the task as fast as possible, and then allow the processor to idle for a long period of time.

Architectural Matters

The architectural considerations for low-power designs are quite similar, even for strikingly different hardware architectures, such as Qualcomm 45nm Serra SoC[8] and Intel 22nm fourth-generation Atom SoC. While it is important to design various system components for lower power consumption, it is also important to optimize the system design and manage it for lower power consumption. In particular, architectural ideas for power saving include the following.

Combined System Components

Integrating the processor and the platform controller hub (PCH) on the same chip can reduce power, footprint, and cost, and also can simplify the firmware interface. Utilizing run-time device power management, as initiated by the kernel and optimizing in the proper context, can significantly contribute to a low-power solution.

Optimized Hardware and Software Interaction

In the era of specialization, code sharing between drivers or hardware versions and operating systems has become less important, in favor of reduced processing and reduced power use. Emphasis has also shifted so that hardware commands are similar to API commands and less command transformation is used. The following ideas embody this concept.

Migrating the Workload from General-Purpose to Fixed-Purpose Hardware

Moving tasks from general-purpose to fixed-function hardware saves power, as a fewer number of gates can be used for a special purpose, thereby reducing the dynamic power consumption that depends on the number of gates that are switching. This technique should especially benefit non-programmable tasks such as setup, clip, coordinate calculation, some media processing, and so on.

---

[8]M. Severson, "Low Power SOC Design and Automation," retrieved July 27, 2009, from http://cseweb.ucsd.edu/classes/wi10/cse241a/slides/Matt.pdf.

Power Sharing Between the CPU and the GPU

To maximize performance within given platform capabilities, technologies such as Intel Burst Technology 2.0 automatically allow processor cores to run faster than the specified base frequency while operating below the specified limits of power, temperature, and current. Such turbo mode and power sharing are supported for both CPU and GPU. The high-performance burst operating points can be adjusted dynamically.

Using Low-Power Core, Uncore, and Graphics

All parts of the processor—the core, uncore, and graphics—should be optimized for low power consumption. The GPU being a significant power contributor, tasks such as composition of the navigation bar, status bar, application UI, video, and so on should go to the display controller as much as possible, so as to facilitate the lowest device activity and memory bandwidth.

Also, as power consumption is highly dependent on the use cases, processor power optimization should be done with practical use cases in mind. That is, a low-power solution is constrained by the worst use case; optimization for and management of both active and leakage power should be considered.

For reducing leakage power, know that although the PMIC can regulate power and can collapse the full chip power during sleep, such power collapsing requires data to be saved to memory and rebooting of all parts of the processor for restoration. The desirable impact, then, is when there is a net savings in power—that is, when the leakage power saved over a period of time is larger than the energy overhead for save and restoration.

Power reduction of RAM/ROM periphery and core array

All memories in the core array need to have leakage control. Power reduction should be considered for peripheral circuits as well, as significant power reduction is achievable from the RAM/ROM periphery.

Reduce V_{DD} during sleep mode to manage leakage

Minimizing V_{DD} of the entire die during sleep is beneficial, especially during short sleep cycles. This has the added advantage of small software overhead and fast restoration, as all memory and register states are retained. Voltage minimization is the most effective way to control leakage without complex management by the software. However, exceptions to this are the domains where voltage cannot be reduced due to the requirements of Connected Standby.

Independent memory bank collapse for large high-density memories

A common power-saving technique is to use power gating on some portions of memory banks that are not accessed by the application. Standby/active leakage reduction can be achieved by gating clock/data access to banks that are not needed. However, this requires

proper power management in software and firmware so that the appropriate, potentially idle memory banks are identified and gated correctly. Also, power gating is not possible for all blocks. Careful analysis will compare the savings achieved from dynamic and leakage powers.

Advanced low-power clocking and clock tree optimization

Clock tree power is a major contributor to total active power use. In the case of Qualcomm Serra, clock tree consumes 30 to 40 percent of the total active power. As clock architecture has a high impact on power use, its use and frequency should be carefully planned. In particular, the number of phase-locked loops (PLLs), independent clock domain control, frequency and gating, synchronicity, and so on need to be considered.

Clock domain partitioning

Partitioning the clocks across different clock domains allows better power management for various clocks, as the clocks can be independently turned off. For example, separate clock domains for I/O and the CPU can be controlled individually.

Independent frequency clock domains

Although asynchronous domains are more flexible, there is increased latency across clock domain boundaries. On the other hand, synchronous clock domains can save power in low-performance mode, but they are susceptible to higher power costs for the worst user case.

Fine-grained tuning of clock gating

One of the most effective ways to save active power is to use fine-grained automatic and manual tuning of clock gating. Either the hardware or the software can control the clock gating, but each has a cost. Analysis of clock gating percentage and efficiency of clock gating can reveal further optimization opportunities.

Using Power Domains or Power Islands

The idea of power islands can be illustrated with the example of a house. If there is only one power switch for all the light bulbs in the entire house, more energy must be spent when the switch is turned on, even when there is only one person in the house who needs only one light at any given time. Instead, if there are individual switches to control all the light bulbs, energy savings are achieved when only the needed light bulb is turned on. Similarly, in a typical computing system, 20 to 25 independent power islands are defined to achieve power savings. This allows for the following energy savings.

Independent voltage scaling for active and sleep modes

For different power islands, the voltage scaling can be done independently for both active and sleep modes. This implies both frequency and consequential power savings.

Power gating across power domains

The PMIC can control multiple power domains, yielding better power control and efficiency, as well as independent voltage scaling and power-collapsing abilities. However, the number of power domains is limited by the increased impedance of the power domain network (PDN), increased IR drop due to larger number of power switches, increased bill of materials (BOM), and the fact that level shifters and resynchronization are necessary at the domain boundaries. Dynamically controlling the power collapse of various domains using small, fast, and efficient chip regulators may be the best option.

When a power island is not power gated, all unused peripherals that are parts of the power island should be clock gated. On the other hand, when a power island is power gated, all relevant peripherals are in D0i3 (but the supply voltage may still be on).

Offering Power-Aware Simulation and Verification

To optimize the system for low power consumption, use power-aware simulation tools and verification methodology to check entry to and exit from low-power states, properly model the power collapse, verify the polarity of clamping, verify the power domain crossings, and so on. These methods include the following.

Tradeoffs among power, area, and timing

Custom design flows and circuits can produce more power-efficient results for some tasks; however, custom design requires more time and efforts. A balanced approach makes careful selection of areas for customization and optimization so as to obtain the greatest benefit. Good customization candidates are clock trees, clock dividers, memory and cell intellectual property (IP, a common term for innovative designs), and so on. It is better to move the customization to the IP block and to use automated methods to insert, verify, and optimize that IP block; that way, specific improvements and/or problems can easily be attributed to the appropriate block.

Comparative analysis of low-power solutions

Design decisions are generally complex, especially when many tradeoffs are involved. To judge how good a low-power solution will be, perform a comparative analysis using a similar solution. Usually this method exposes the strengths and weaknesses of each solution.

Power Optimization on Low-Power Platforms

On low-power platforms, several aspects of power optimization typically come into play simultaneously. Here, we use a video playback application to illustrate various power-optimization approaches. Each contributes a different amount to the overall power savings, depending on the application's requirements and the video parameters.[9]

Run Fast and Turn Off

In a hardware-accelerated video playback, the GPU does the heavy lifting by performing hardware-based decode, scaling, and post-processing, such as deblocking. Meanwhile, the CPU cores execute the OS processes, media frameworks, media application, audio processing, and content protection tasks. The I/O subsystem performs disk access and communication, while the uncore runs the memory interface. On a typical platform, these subsystems execute various tasks in sequential order. An obvious optimization is to exploit the overlap of tasks and to parallelize them so that the hardware resources can operate concurrently, making the most of the I/O bursts for a short time before becoming idle. In this case, the optimization focus is on reducing the active residencies and achieving power savings when the resources are idle.

Activity Scheduling

It is important to appropriately schedule the activity of the software and hardware, and make smart use of the memory bandwidth. In video playback, tasks are mainly handled by the CPU and the GPU, and are of three main categories: audio tasks, video tasks, and associated interrupts.

Audio activities are periodic with a 1 to 10 ms cycle, and are handled by a CPU thread that schedules audio decode and post-processing tasks, as well as related audio buffers for DMA operations. In addition to the regular periodic behavior of the audio DMAs, the DMA activity involves Wi-Fi and storage traffic from I/O devices going into memory, which are somewhat bursty in nature.

Video tasks are partially handled by the CPU, which performs the audio-video demultiplexing, while the GPU performs the decoding and post-processing tasks. Parallelizing the CPU and the GPU tasks are obvious scheduling choices to lower the overall power consumption.

As CPU power consumption is influenced by how many times the CPU needs to wake up from a low-power state to an operating state, and the energy required for such state transitions, significant power savings can be achieved with a power-aware application that avoids such transitions. It does this by using interrupts to the maintain execution sequences, instead of timer-based polling constructs, and by appropriately scheduling them. Further optimization can be achieved by scheduling regularly occurring audio DMAs further apart, or even offloading audio to a dedicated audio-processing hardware.

---

[9] A. Agrawal, T. Huff, S. Potluri, W. Cheung, A. Thakur, J. Holland, and V.Degalahal, "Power Efficient Multimedia Playback on Mobile Platform," *Intel Technology Journal* 15, no. 2 (2011): 82–100.

Reducing Wake-ups

Streaming the video playback usually requires communication devices to have smoothing buffers while accessing the memory due to the bursty nature of the incoming network data packets. Smart communication devices can reduce the number of CPU wake-ups by combining the interrupts and by using programmable flush threshold, and by treading the path to memory that is already active.

Burst Mode Processing

Video playback is done on a frame-by-frame basis, while frame processing time is between 15 and 30 ms, depending on the frame rate. The nature of video playback does not offer the CPU much opportunity to go to a low-power state and return to active state within this short time. To overcome this limitation, a pipeline of multiple frames can be created such that the associated audio is decoupled and synchronized properly, thereby allowing the CPU to aggressively utilize its low-power states. Although this approach requires substantial software changes, it should nonetheless be explored for very low-power devices.

Improving CPU and GPU Parallelism

As the decoding and processing tasks are done by the GPU, the main tasks that remain for the CPU are preparing the hardware acceleration requests for decoding, rendering, and presenting the video frames. These tasks are independent and, therefore, can be implemented in separate hardware threads, which can be scheduled and executed in parallel. By parallelizing the execution threads, the CPU package can stay in deep idle states for longer periods, achieving power savings.

GPU Memory Bandwidth Optimization

The GPU performs video decoding and post-processing. These processes require the highest amount of memory bandwidth in the entire video playback application. But memory bandwidth scales with the content and display resolution, as well as with the amount of video processing done on each frame. Displaying a frame with a *blit* method (i.e., drawing the frame directly onto a display window) requires copying the frame twice: once to a composition target surface and once to a render surface. These costly copies can be avoided by using the overlay method, by which the kernel directly renders the frame to the overlay surface.

Display Power Optimization

Display requires more than a third of the device's power for video playback applications. There are many ways to address this problem, most of which were covered earlier. In addition, a media playback application can benefit from frame buffer compression and reduction of refresh rate.

Frame buffer compression does not directly impact the video window; but for full-screen playback when update of the primary plane is not needed, only the overlay plane can be updated, thereby saving memory bandwidth and, consequently, power. Reducing the refresh rate, on the other hand, directly reduces memory bandwidth. For example, reducing from 60 Hz to 40 Hz results in a 33 percent reduction in memory bandwidth. Not only is memory power reduced, owing to less utilization of display circuitry, but the overall display power is also lowered. However, depending on the content, the quality of the video and the user experience may be poorer compared to a video with full refresh rate.

Storage Power Optimization

Frequent access to storage results in excessive power dissipation, while reduction of such access allows the storage device to go to lower power states sooner and contributes to overall power savings. However, unlike the CPU, storage devices typically need to be idle for more than a second before they are transitioned to a lower power state.

As the requirement for media playback storage access is on the order of tens of milliseconds, normally a storage device would not get a chance to sleep during media playback. A power-aware media playback application, however, can pre-buffer about 10 seconds' worth of video data, which will allow a solid-state storage device to enter a low-power state. Storage devices based on a hard drive are slower, however, and require multiple minutes of pre-buffering for any meaningful power savings.

The Measurement of Low Power

Measuring low power is generally done following the same shunt resistor method as described in Chapter 6. However, precise measurement and analysis of several power signals may be necessary to determine the impact of particular power-consuming hardware units.

Processor Signals for Power

In a typical low-power Intel Architecture platform, such as the Intel Atom Z2760,[10] the processor signals that are defined for power interface are as shown in Table 7-4. Power analysis is done by appropriately measuring these signals.

[10] "Intel Atom Processor Z2760: Data Sheet," Intel Corporation, October 2012, retrieved from www.intel.com/content/dam/www/public/us/en/documents/product-briefs/atom-z2760-datasheet.pdf.

Table 7-4. *Important Processor Signals for Power Interface*

Signal	Description
V_{CC}	Processor core supply voltage: power supply is required for processor cycles.
V_{NN}	North Complex logic and graphics supply voltage.
V_{CCP}	Supply voltage for CMOS Direct Media Interface (cDMI), CMOS Digital Video Output (cDVO), legacy interface, JTAG, resistor compensation, and power gating. This is needed for most bus accesses, and cannot be connected to $V_{CCPAOAC}$ during Standby or Self-Refresh states.
V_{CCPDDR}	Double data rate (DDR) DLL and logic supply voltage. This is required for memory bus accesses. It needs a separate rail with noise isolation.
$V_{CCPAOAC}$	JTAG, C6 SRAM supply voltage. The processor needs to be in Active or Standby mode to support always on, always connected (AOAC) state.
LVD_VBG	LVDS band gap supply voltage: needed for Low Voltage Differential Signal (LVDS) display.
V_{CCA}	Host Phase Lock Loop (HPLL), analog PLL, and thermal sensor supply voltage.
V_{CCA180}	LVDS analog supply voltage: needed for LVDS display. Requires a separate rail with noise isolation.
V_{CCD180}	LVDS I/O supply voltage: needed for LVDS display.
$V_{CC180SR}$	Second generation double data rate (DDR2) self-refresh supply voltage. Powered during Active, Standby, and Self-Refresh states.
V_{CC180}	DDR2 I/O supply voltage. This is required for memory bus accesses, and cannot be connected to $V_{CC180SR}$ during Standby or Self-Refresh states.
V_{MM}	I/O supply voltage.
V_{SS}	Ground pin.

Media Power Metrics

Many media applications share characteristics of real-time requirement, bursty data processing, data independency, and parallelizability. So, in media applications, it is important to measure and track several power-related metrics to understand the behavior of the system and to find optimization opportunities. Among these metrics are SoC, display, voltage regulator and memory power as percentages of full platform power, the CPU core and package C-state residencies, the GPU activities and render cache (RC)-state residencies, memory bandwidth, and so on.

As certain media applications may be compute-bound, memory-bound, I/O bound, or have other restrictions such as real-time deadlines, determining the impact of these factors on power consumption provides better awareness of the bottlenecks and tradeoffs.

It is also important to understand the variation of power with respect to video resolution, bit rate, frame rate, and other parameters, as well as with regard to system frequency, thermal design power, memory size, operating system scheduling and power policy, display resolution, and display interface. Analyzing and understanding the available choices may reveal optimization opportunities for power and performance.

Summary

The marriage of low-power devices with increased demand for application performance, and the various challenges for attaining such low-power use, has been described in this chapter. As downscaling of process technology, together with voltage and frequency scaling, provides reductions in power, these techniques fall short of achieving state-of-the-art low-power design targets. Analysis of common low-power scenarios from a media usage standpoint shows that more aggressive power-reduction approaches are necessary while taking the whole system into account.

To this end, various power-management and optimization approaches were discussed. Low-power measurement techinques were also presented. Together, Chapters 6 and 7 provide a good platform for understanding the tradeoffs between increased dynamic ranges for frequency tuning and greater static power consumption—elements that must be carefully balanced. In the next chapter, some of these tradeoffs between power and performance are viewed from the point of view of a media application.