Windows power management basic knowledge

July 28, 2015 Dave Feng

Contents

Introduction

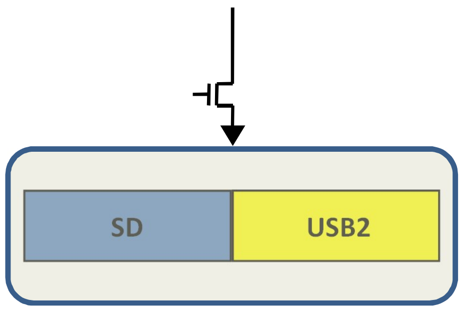
Windows OS support many features in power management aspects to save power consumption and extend mobile devices' battery life. Software power management is a big topic, many concepts are introduced. In a Windows driver, the power management related routines (functions) are distributed in many places. Some of the routines in a driver are used to control hardware registers, some co-work with OS to let the whole system work properly when state transitions, other routines may provide information to other software components(other drivers or OS) to make some policy decision.

To make the driver work properly, it is essential to understand the fundamentals in Windows power management. Without the basic knowledge, it is difficult for a driver developer to figure out a workable logic to enable hardware low power feature, also it is hard to communicate with HW engineers when debugging issues.

Power Domain (Power Rail)

SOC's controllers need power supply (Vcc) to work. The different power supply is different power domain. To save power, we need to power down the whole controller's power supply when the controller is not used. It is better to segregate all controllers with separate power rails to maximally save power consumption, and then every controller has individual power domain which can be powered on/off when it is not used. But power rail take big place, this means more power rails take more area in silicon, so SOC shares one power rail among several small controllers to save silicon area.

Consequently, when several controllers share one power domain, the power supply cannot be shut down even when only one controller is using power. Only after all controllers in the power domain are not used, the power domain can be shut down. The power leakage due to the power supply is called static power leakage.



1. An imaginary power domain has two controllers
2. Power domain actually is controlled by a switch
3. only SD and USB2 both are not used, the power switch can be off
4. a power domain has one voltage

A controller's sub components may belong to different power domains. Every controller consists of many digital. .e.g. a USB host controller has USB core logic and PHY. USB core logic's voltage is different from PHY's voltage supply, so a USB host controller belong two different power domains.

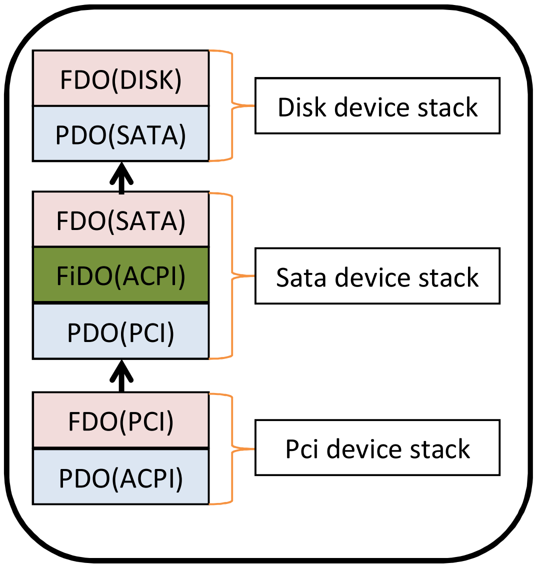
Generally controller driver cannot control power domain because one controller driver does not know if other controllers in the same power domain are working or not. Power domain is controlled by system level software like system BIOS or special driver like PEP which knows all drivers' power state.

Dynamic clock gating

A working controller in SOC or external device is driven by clock. Clock's flip-flop can consume much power (dynamic power leakage). When a controller is idling (not used), software can shut down the clock, this is called dynamic clock gating. After clock is gated, there may have static power leakage due to power supply

Device Stack

In Windows driver model, bus driver creates physical device object (PDO), during the device is enumerated, function driver (or class driver) creates functional device object (FDO). If there has a filter driver, upper FiDO or lower FiDO is created by the filter driver. We call the PDO/FDO/FiDO a device stack. Bus driver, function driver, along with filter driver (if has) work together to manage a device which can be a physical hardware or pure software device has no real hardware. Generally we call a driver stack which may cover more than one PDO from bottom up, but a device stack has only one PDO and one FDO. This is important when we explain power policy owner.



The driver stack is composed of ACPI driver, PCI bus driver, SATA controller driver (storport+miniport) and disk driver (classpnp+disk)

There has three device stacks in the driver stack

S State

S state is system power state. S states are defined in ACPI, From S0 to S5. Power consumption is decreased as number increases. S1 and S2 are not used in Windows.

S0 is fully ON state. In S0 state, display is not necessary ON

S3 is sleep state. DDR is in self refresh state. CPU is powered off. Some power domains are still powered on when system is in S3 to support wake, for example, USB's PHY is powered on to support USB wake up.

S4 is hibernation. DDR is powered off; OS's contents are saved in hard disk.

S5 is power off state.

S0ix state

Starting from Windows 8, Mobile device supports a new system low power mode state called Connected Standby state (later in Win 10, Modern Standby). Not all platforms support CS state, CS needs special BIOS, and not all SOCs are targeted for CS.

From software aspects, when display is off, system is in CS state. To save power, platform must enter a platform defined low power state like S0i1, S0i2…. S0ix's target is to consume very small power (approach to power consumption when system is in S3), but system can be woken up instantly.

Conceptually S0ix is still in S0 state.

D state

D state is device’s power state. From D0 to D3cold, D states are also defined in ACPI.

D0 state is fully on state. When a device is in D0, it is well configured and can transfer data. Generally, bigger x(Dx) means lower power consumption.

D state is a logical power state; ACPI does not define how a device should be configured in a specific Dx state.

For PCI device, PMCSR is used to control the device’s D state. When a PCI device is put into D3, PMCSR should be set D3.

For a pure software device, such as the device which is driven by a protocol driver, the D state is meaningless from power consumption perspective, but D state is used to describe the logical power state to communicate with OS or other drivers.

For other real hardware devices, it is the driver or platform’s responsibility to map the D state to a hardware state and define how to configure the hardware when the device power state transitions. For example, when hard disk is in D3 state, disk driver decides on how to configure the hard disk to save power. Maybe stop rotating, power off some power supply.

Not all drivers change hardware configuration when the device’s D state transitions. Some driver may change hardware’s power settings when the device is idle to save power even when the device is in D0 state. Keep this in mind.

D0 and D0 initialized

There has another state called D0 uninitialized. From power supply perspective, D0 is the same as D0 uninitialized. The difference between D0 and D0 uninitialized is: only after software properly configuring a D0 uninitialized device, then this device transitions to D0. A device in D0 un-initialized state cannot be used to transfer data. For example, a USB controller is in D0 un-initialized state cannot enumerate device and the USB core cannot work, only after its registers are configured (as USB controller defines), the controller can work properly.

D3cold to D0 process

When a controller is in D3cold state, generally its power supply is totally off. But to support wake up, a controller in D3 cold may also have some side band signals, so it may have very small power supply to maintain its state.

When a controller is in D3 cold, all the controllers share the same power domain with it are also in D3 cold. To transition the controller to D0, the power domain should power on, then all the controllers are in D0 un-initialized state, then driver can configure the controller to transition to D0.

Why D state is necessary?

As explained above, D state is a logical state; some devices’ real hardware power consumption may have no difference between when they are in D0 and in D3. Why it is necessary to use D states?

Using D states is to communicate between software components.

1. OS cannot know every device’s power state without a variable to track.

2. A bus can enter a lower power mode only if all its children are in a lower power state. Without a device power state variable, bus driver does not its children's power state.

3. System can enter S3 only after the whole system's devices are in lower power state. Without a device power state variable, system does not know when it can enter sleep state

Components in a device

Some complex devices may have several inter-dependent components, an audio device can have playback and recorder functions. To save power; power manager can manage the components separately. Most real hardware devices have only one component.

F state

F state is component's power state. Defining F state is to control different power consumption levels when a device is in D0. When a device is in D0, components in this device can have F0, F1, F2,…Fn states. The deeper F state is, the less power is consumed. A component can enter Fn power state (n!=0) from F0, and from Fn to F0, but cannot enter a lower power F state from another lower power F state. That is to say, a component cannot enter F2 from F1. When a component is in F1, and can enter F2 after a while, power manager will put the component into F0 firstly, and then put it to F2 state.

S state to D state map

Calculating the accurate D state when system is in Sx is really complex, it needs ACPI driver, parent driver and BIOS co-operation. By default, when system is in S3 or S4, all devices are in D3. If a device can support wakeup, but the wakeup signal can only be generated in D0 or D1 or D2, in such case, when system is in S3, device maybe not in D3.

CPU C/P state

CPU core (cache, PLL, clock…) consumes much power when it is running code, but for a computer, most time CPU is idle. Cx(x! =0) is CPU core power state when it is idle. The larger x, the deeper state the core is, and the more time is needed to come back to C0. For X86 architecture, core enters C1 by executing _HLT instruction, enters C2 by executing I/O read instruction, the I/O port address is designated in BIOS. Generally, deeper state than C2 is implemented in hardware; entering into these states is automatically done by hardware.

Even when CPU is in C0 state, it can run at different frequency levels, higher frequency level consumes more power. To dynamically change the frequency, P state is used. OS power manager can select different P sate based on the working load.

Power manager

Power manager is an OS kernel module (in ntoskrnl.exe). Power manger is responsible for all kinds of power management related jobs. CPU power state decision making, manages power settings, collects user input, tracks all devices' D state, components' F state tracking.

S IRP

Before system state transitions, power manager initiates an IRP and directs to device driver. This IRP tells driver that system is trying to change S state, and driver should handle this IRP. Only power manager can create and initiates S IRP.

D IRP

D IRP is related to device power state. D IRP is initiated by power manager or device driver. When power manger wants to transition a device's power state, it can initiate a D IRP and sends to the device driver. When a device receives an S IRP, to handle the S IRP, the device driver may need to change the device's D power state, in such case, device driver can initiate a D IRP.

D IRP Handling

When device driver receives D power IRP, generally it passes down because bus driver will do the real job to put the real hardware into lower power state

If a driver can control the hardware and put the hardware into lower power mode, generally it cannot put the device into lower device power state by setting the hardware register without initiate device power IRP. Driver must let the power manager know that it will transition to another device power state by requesting power IRP, or the power manager does not know the device's power state at all.

Power related IRP handling is done by WDF if the driver is WDF based

Power related ACPI methods

When a device is put in lower power mode like D3, its power resource (like Vcc) can be shut down, but such power resource generally cannot be controlled in Windows driver because the power resource is platform dependent. BIOS is the place to control the power resource.

In such case, a method is created in BIOS for the device, when device's power state transitions, ACPI driver will invoke the method in BIOS. How does ACPI driver know when the device power state transition? Because the device is declared in ACPI name space, ACPI driver is the device's filter driver.

Runtime power management

Runtime device power management means when system is S0, Device is not necessary always in D0, when device is not used(idle), device power state can transition to Dx(x!=0) or component power state can transition to Fx(x!=0)

Runtime idle detection mechanism

When system is in S0 and device is not used (no in progress I/O), we call the device is in idle state. Generally if the device is not used, driver can put the device in lower power state (Dx), but frequently transitioning the device into and out of a D state needs much time, so an idle timer is used. Only after the device is being idle some time, driver puts the device into a lower power state.

Power framework

Power framework or runtime power framework is a set of kernel APIs for driver development. A driver can register with power framework, and provide some callbacks, when system is running, kernel power manager will call the driver provided callbacks when the device is idle, or F state transition, or D state transition…

System power policy owner

System power policy owner is actually kernel power manager. It decides after how long the system is being idle, and then system can enter S3; power usage in AC/DC mode; CPU idle residency parameters; core parking parameters; what power scheme is used…

Device power policy owner

Device power policy owner is the device stack's function driver. This function driver decides how to define the device's idleness, when the device power D/F state can change, and the power state dependency on other device's power state…

Wake

System wake up

When the system transitions from S3/S0ix to S0, we call system wake up. Generally it is due to wakeup source (external wakeup interrupts or timer/RTC)

Device wake up

Device wake up means when system is in S0, but the device is in D3 (power is lost), and the device has the capability to detect some external signals, when the signal happens, the device can power on the device itself with driver's co-work.

Device wake up is different from device resume: For example, when a storage device is in D3, the storage driver can transition the storage device from D3 to D0 after receives incoming I/O, this is device resume not device wake up.

Wait/Wake IRP

Power state machine

Power state machine is a diagram OS or a device driver maintains, this diagram tracks the whole system power states or the device power states which are internally used by OS or the driver.

For a device, the power states are, but not limited to, Dx and Fx, because power states' transitions are very complex, to describe every specific state, power state machine generally records many

DEVICE CAPABILITY

Device capability is a device's PNP and power attributes, PNP gets a device's capability via sending PNP IRP to the device and device function driver pass down the IRP, after IRP reaches to bus driver, but driver fills the data structure based upon real hardware capability or ACPI setting. When the IRP is completing, function driver can change the capability's data member per real hardware attribute or real usage case, finally PNP manager gets the device's capability. Device's capability can decide if the device can wake up the system in Sx, if it is removable…

Power Engine Plug-in (PEP)

Why need PEP?

S/D/F states resolves many power management problems and improve power efficiency much, but not all. To resolve following questions, we need a new driver,

1. Some devices share clock, when one of the devices is not used, somehow it will be in D3 if the driver apply runtime power management, when all the devices are not used, the shared clock should be shut down to save power, but drivers share the same clock do not communicate, so even all devices are in D3, nobody will shut down the clock. we need a driver to track all the devices' state, when all the devices are in D3, this driver can shut down the clock
2. Even when all of the processor's cores are in the lowest power state and all devices are in D3cold, the system still consumes power in system level, we can power down the whole system (leave minimal power on for maintaining wake up logic). We need a driver to define a system low power mode (S0ix), this low power mode depends on core's C state and devices' D state.
3. Some CPU's low power mode depends on device's activity. For example, when USB has DMA transfer, CPU cannot enter C2. CPU's low power mode entry is offered by OS, if OS is not aware of the dependency, C2 entry can lead unpredictable result. In such case, we need a driver to tell OS the CPU's low power dependency on device.

PEP is such kind driver, whenever following state transitions, power manager will notify PEP driver, then PEP driver can decide what to do

1. a device D state
2. a component's F sate
3. CPU's C state