TOSHIBA

Leading Innovation

Suspend-to-RAM implementation on Freescale 74xx without PMU

4/18/2007

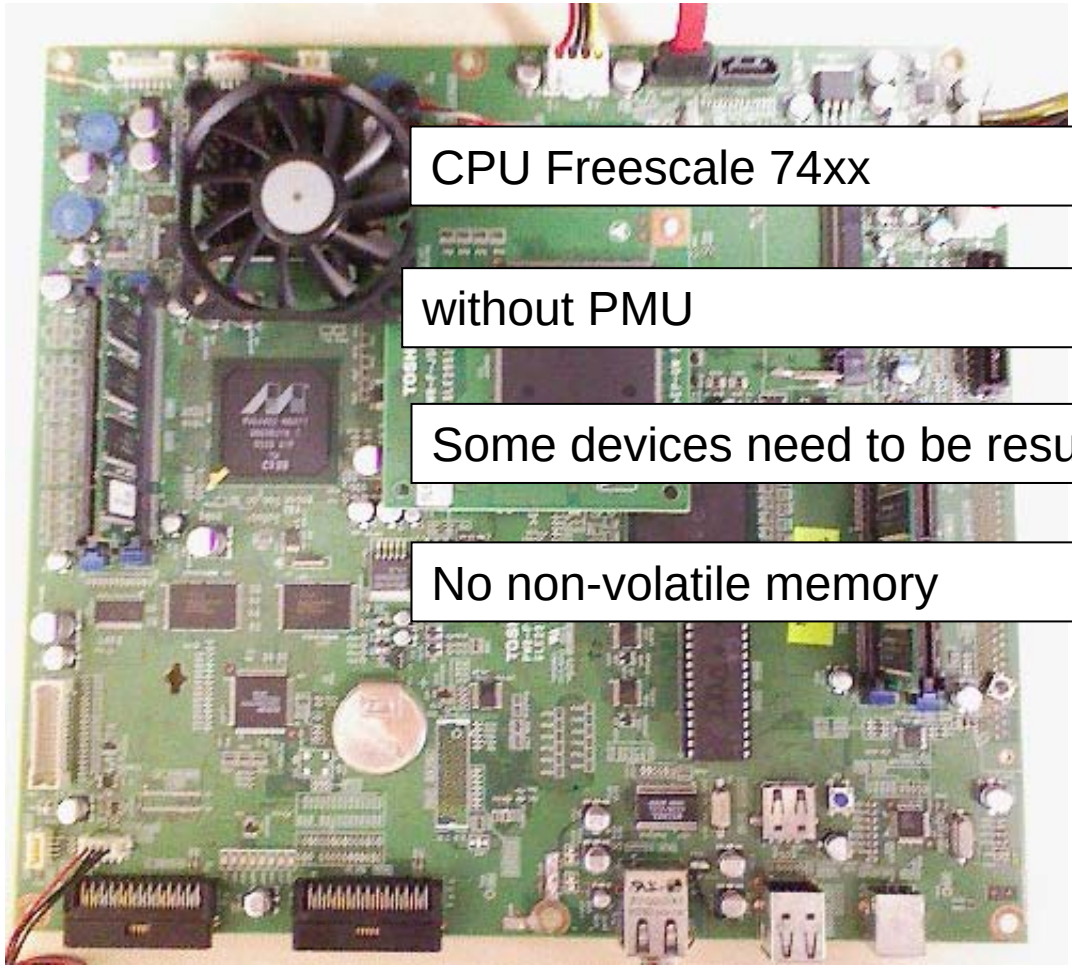
Fujihito Numano
Corporate Software Engineering Center

Agenda – Goals and Backgrounds

1. **Target Board**
2. **Goals**
3. **Backgrounds**
   - Freescale 74xx programmable power mode
   - PMU (Power Management Unit)
4. **Idle power management**
5. **Suspend to RAM**
   - Overview
   - enter_state()
   - Device Power Management
   - pm_ops functions
6. **Performance**
7. **Our next challenges**

1. Target Board

TOSHIBA TEC Corporation development board



- CPU Freescale 74xx
- without PMU
- Some devices need to be resume trigger
- No non-volatile memory

2. Goals

Requirement

The system automatically enters in the low-power “sleep” mode after a period of inactivity.

Status – Implemented a test version

- Go to suspend to RAM after a period of inactivity
- Reduce power consumption by more than 50%
- Resume triggered by the specific device access.

Points

- Suspend to RAM with 74xx power saving “sleep” mode
- Device control without “PMU”
- Simple & compact implementation

3. Backgrounds

- **The system enters in the low-power “sleep” mode automatically after a period of inactivity to reduce power consumption.**

- **The system returns to the normal “working” mode immediately if needed.**

⇒ The boot up time is so long that ,

power off and system shutdown cannot be used

System suspend & resume should be used.

3. Backgrounds

- **No non-volatile memory**

  ⇒ Hibernation function cannot be supported
  "Suspend to RAM", "Idle power management", and
  "Device power management (Device on and off control)" functions.
  are effective.

- **The target CPU is Freescale 74xx**

  ⇒ It needs to support programmable power mode
  and we plan how to implement suspend to RAM by using it.

- **No external H/W, such as PMU (Power Management Unit), supports to control power status**

  ⇒ The kernel itself should control power status
  and especially control resume trigger devices.

Freescale 74xx power saving mode

2 power saving modes are available to the system,

Nap mode and Sleep mode

Nap mode

Instruction fetching is halted.

The clocks for time base, decrementer remain running.

So the CPU is soon returned to the normal (RUN) mode by the “decrementer” timer interruption

⇒

The system cannot keep the “power saving mode (Nap)”.

Nap mode should not be applied to suspend to RAM but Idle power management.

Freescale 74xx power saving mode

Sleep mode

Power consumption is further reduced by disabling bus snooping.

All internal functional units are disabled.

Internal exception, such as timer interrupt ,does not occur.

So the system does not return to normal mode by itself.

⇒ The system can keep sleep “power saving” mode.

It is appropriate for suspend to RAM

Disabling bus snooping

⇒ The system needs to flush TLB and cache on resume.

PMU – The Power Management Unit

The Power Management Unit (PMU) is a microcontroller that governs power functions for Apple computers.

And is responsible for coordinating following power management functions.

- Monitoring power connections and battery charges
- Charging batteries when necessary
- Controlling power to other integrated circuits
- Shutting down unnecessary components when they are left idle
- Controlling sleep and power functions (on and off)

(from Wikipedia, the free encyclopedia)

Without PMU support

Without PMU , the kernel needs to do some power management functions by itself.

- Controlling power to other integrated circuits
- Shutting down unnecessary system components when they are left idle
- Controlling sleep and power functions (on and off)
  (In the target, the resume trigger devices are controlled in particular)

Agenda - Implementations and Performance

1. **Target**
2. **Goals**
3. **Backgrounds**
   - Freescale 74XX programmable power mode
   - PMU (Power Management Unit)
4. **Idle power management**
5. **Suspend to RAM**
   - Overview
   - enter_state()
   - Device Power Management
   - pm_ops functions
6. **Performance**
7. **Our next challenges**

4. Idle power management

- **The idle power management for 74xx has been already implemented in the power PC kernel**

- **When the kernel starts, the idle process is launched. It is set to the lowest priority by the process scheduler .**

- **If "/proc/sys/kernel/powersave-nap" is set to 1, the idle process tries to set the CPU to NAP mode.**

5. Suspend to RAM implementation - Overview

- **Implementations for 74xx sleep mode**
  - Save and restore the processor contexts
  - Timer , TLB and cache flush
  - The resume exception handler

- **Device control without “PMU”**
  - The resume trigger devices are controlled in particular.

- **Simple & compact implementation**
  - All functions dependent on 74xx are wrapped on pm_ops function table .
  - All functions independent on the CPU architecture can be used without any changes , such as enter_state() and device power management functions.

enter_state()

All suspend to RAM implementations are included in enter_state() function.

Applications and following suspend command also call it indirectly.

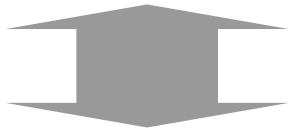
```
#echo –n mem > /sys/power/state
```

It is based on preprocessing, main and post processing functions.

- suspend_prepare()
- suspend_enter()
- suspend_finish()

enter_state()

The functions dependent on 74xx are wrapped on the pm_ops function table.



All implements independent on the CPU architecture can be applied to the target system without any change.

- Process control
- Console control
- Device power management

enter_state()

enter_state()

suspend_prepare()

pm_prepare_console()	← exchange the virtual console for the one of suspend only
disable_nonboot_cpus()	← Stop other CPUs (for multi processor)
freeze_processes()	← Freeze processes
nr_free_pages()	← check and allocate memories for suspend
pm_ops->prepare()	**← preprocessing for the suspend dependent on 74xx**
suspend_console()	← suspend the console output
device_suspend()	← suspend the devices

suspend_enter()

device_power_down()	← suspend devices which fail to suspend in device_suspend() function or others
pm_ops->enter()	**← main routines dependent on 74xx. The system suspend here**
device_power_up()	← resume devices

suspend_finish()

device_resume()	← resume devices entried in the dpm_off table
resume_console()	← resume the console output
thaw_processes()	← resume processes
enable_nonboot_cpus()	← restart other CPUs(for multi processor)
pm_ops->finish()	**← post-processing for the suspend dependent on 74xx**
pm_restore_console()	← restore the virtual console

Device Power Management

- The kernel can control device power state by “device_xx” functions.
- “device_register” function entries the device in the “dpm_active” list at the initialization of the driver. All devices in the list can be controlled by the kernel.
- On suspending the system, enter_state() calls device_suspend() and device_power_down().

  The devices are set to device power down or device off state.
- On resuming the system, enter_state() function calls device_power_up() and device_resume() function.

  The devices are restored to device power on state.

Device Power Management functions

enter_state()
    suspend_prepare()
        :
        pm_ops->prepare()
        :
        device_suspend() ← **suspend the devices**

    suspend_enter()
        device_power_down() ← **suspend devices which fail to suspend in device_suspend()**
            or devices not entried in the dpm_active table
        pm_ops->enter()

        device_power_up() ← **resume devices**

    suspend_finish()
        device_resume() ← **resume devices entried in the dpm_off table**
        :
        pm_ops->finish()
        :

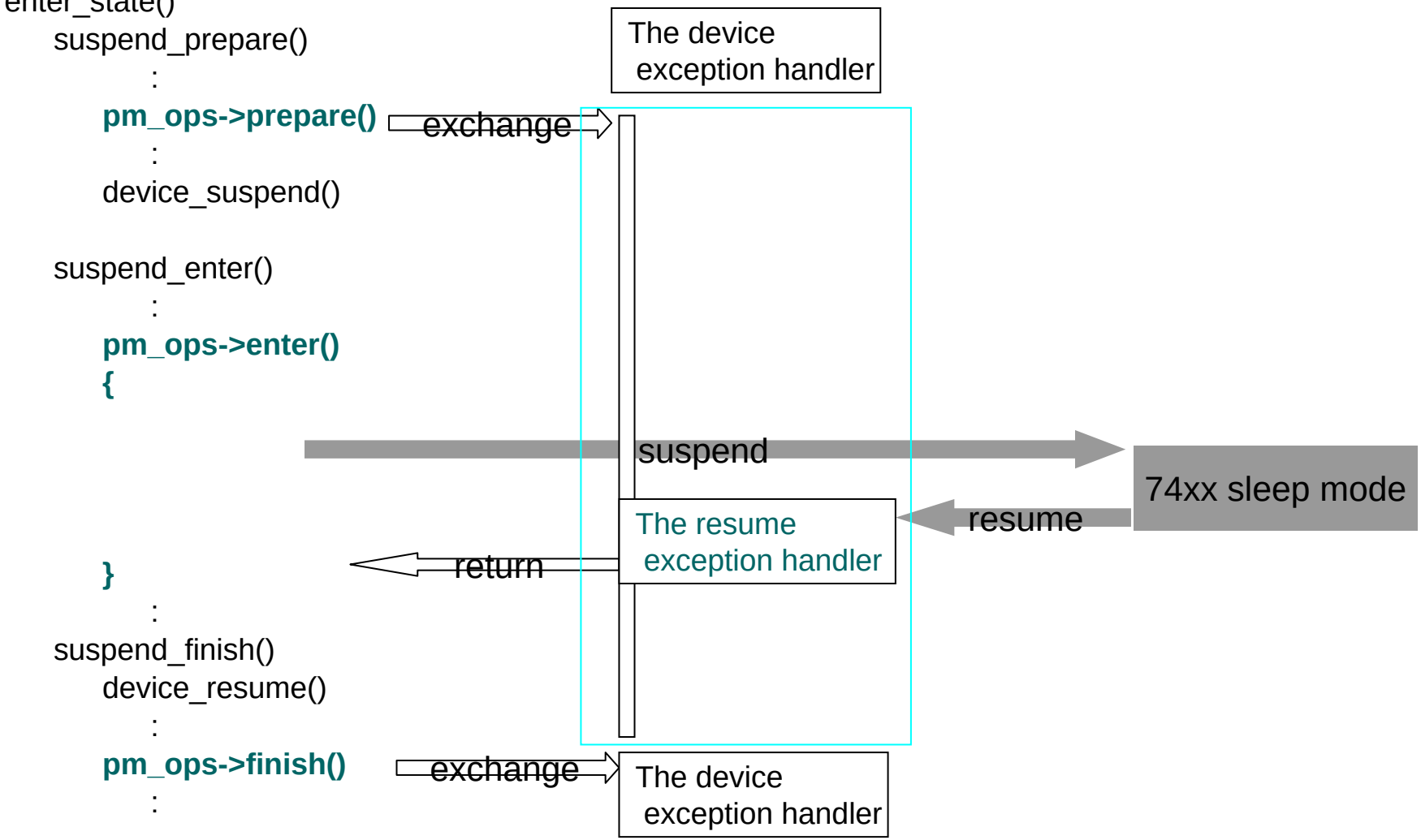
Device Power Management for resume trigger devices

- The resume trigger devices are controlled in particular.

- While the system suspends, their exception handlers are exchanged for the resume exception handlers (for 74xx sleep mode).

	Resume Trigger Device	**General Device**
Example	**LAN (supports wake-up on LAN)** **Power button** **LCD "rid" button** **Input device**	HDD LCD
Resume Trigger	Yes	No
Control function	pm_ops functions (dependent on 74xx)	device_xx() (independent on the CPU architecture)

Power Management for resume trigger device



pm_ops function table

- “pm_ops” function table can wrap functions dependent on 74xx.
  All functions independent on the CPU architecture can be used
  without any changes ,
  such as enter_state() and device power management functions.

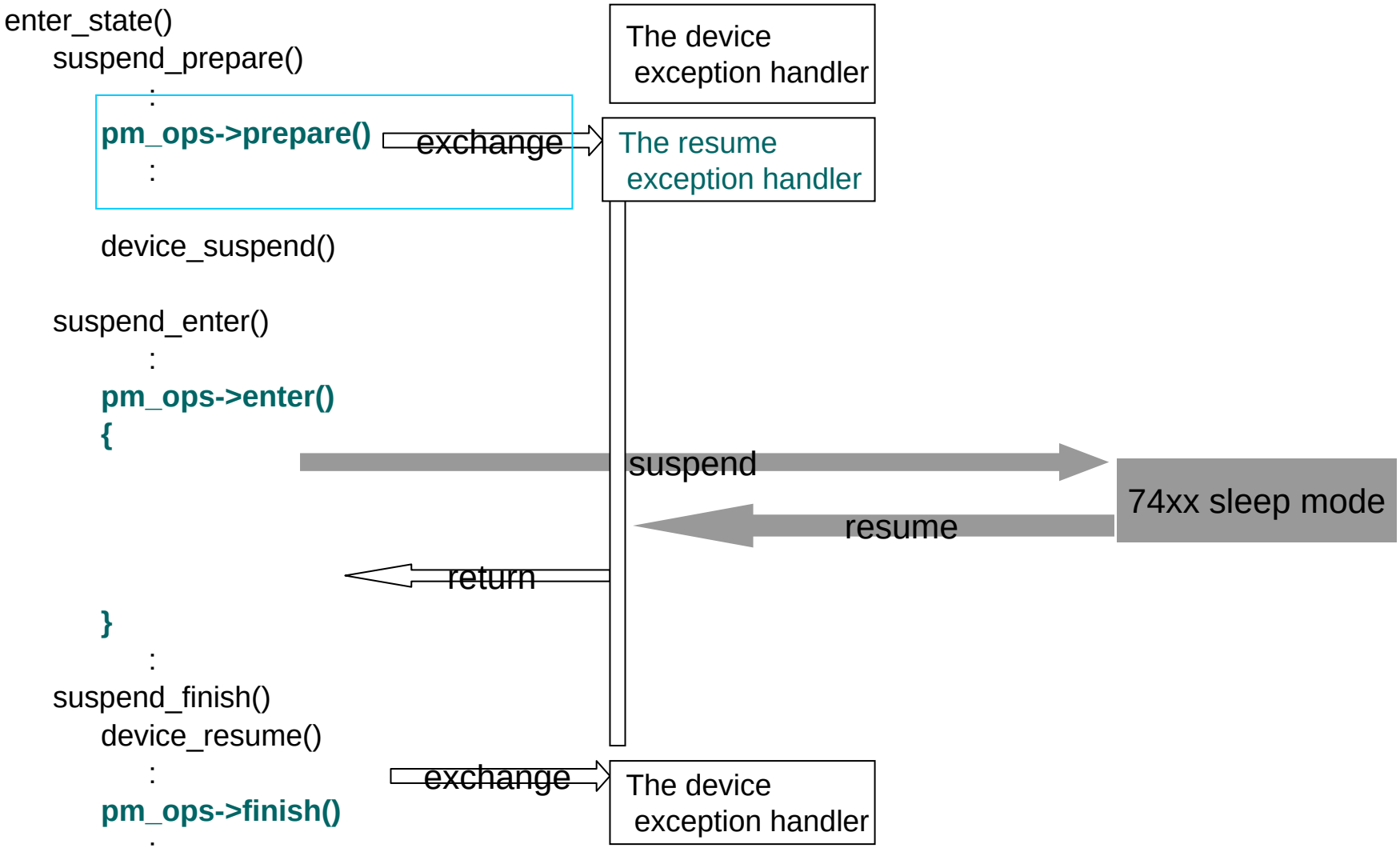
- Our reference comes from the implementations in the x86 kernel.
  implemented mainly ACPI BIOS processing.

- The codes, preprocessing , main routines and post-processing, are set in the table as follows. They are called from enter_state() function.

/drivers/power/main.c

```
static struct pm_ops acpi_pm_ops = {
    .prepare = acpi_pm_prepare,
    .enter = acpi_pm_enter,
    .finish = acpi_pm_finish,
};
```

pm_ops->prepare()



pm_ops->prepare()

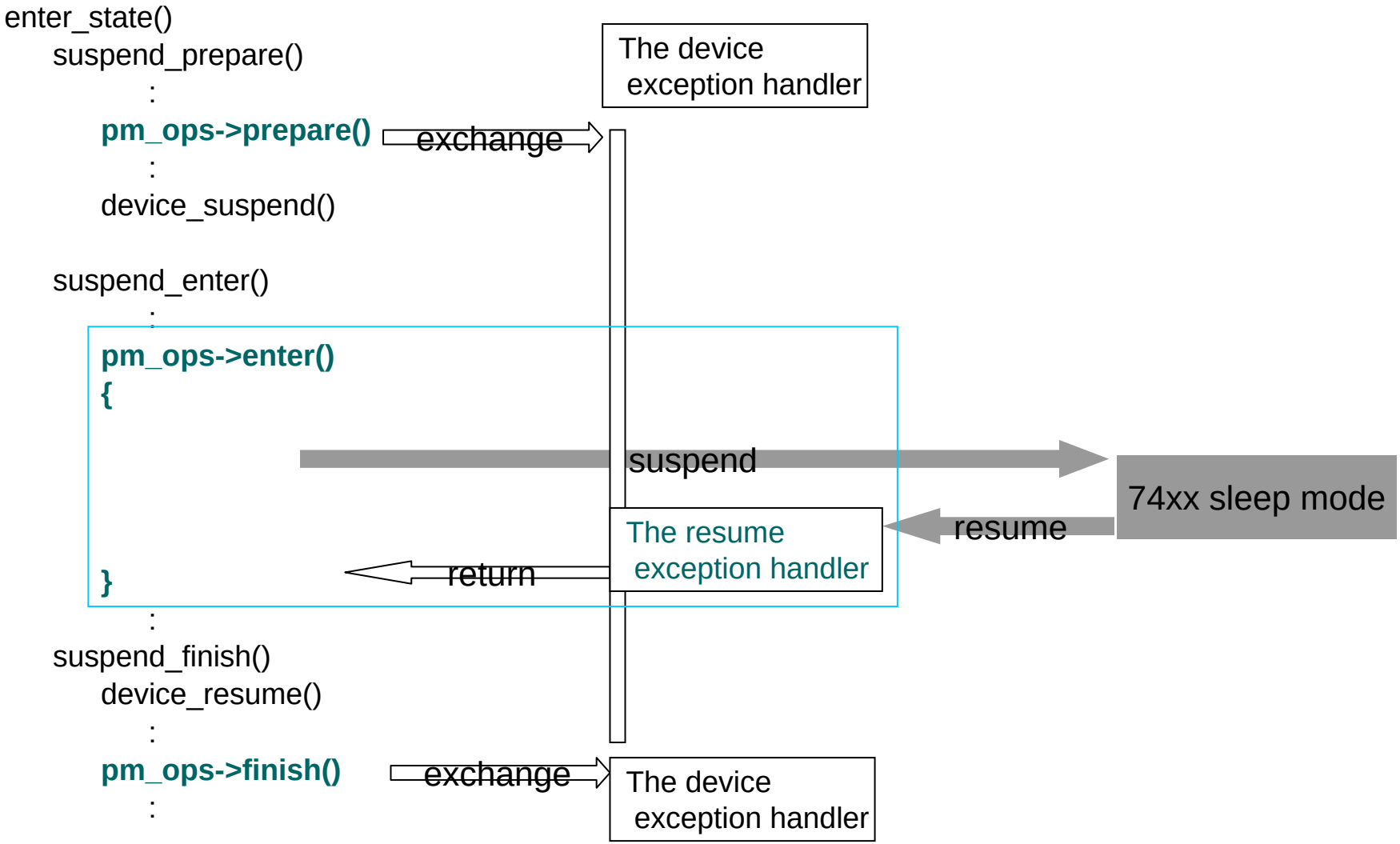
The preprocessing dependent on 74xx
is wrapped in pm_ops ->prepare() function.

1. Interrupt disabled by the exception of resume trigger devices.
   Usually the interrupts are used for their device controls.
   Disabled for the followings

2. Exchange the exception handlers for the exception handlers for resume only.
   The resume exception handler is the start point where the 74XX system resume from the sleep mode.

- Our reference comes from the following pm_ops->prepare() implementations in the x86 kernel.
  - Control resume trigger devices and their device state
  - Prepare resume vectors

pm_ops->enter(), the resume exception handler



pm_ops->enter()

The suspend main routine dependent on the CPU architecture.
is wrapped in pm_ops ->enter() function.

1. Interrupt enabled by the exception of resume trigger devices (resume exception handlers).

2. Save the processor contexts
3. Destroy timer and cache
4. Go to Freescale 74XX sleep mode
   (The system goes to suspend)

5. Returned from the resume exception handler.
6. Interrupt disabled by the exception of resume factor devices (resume exception handlers).

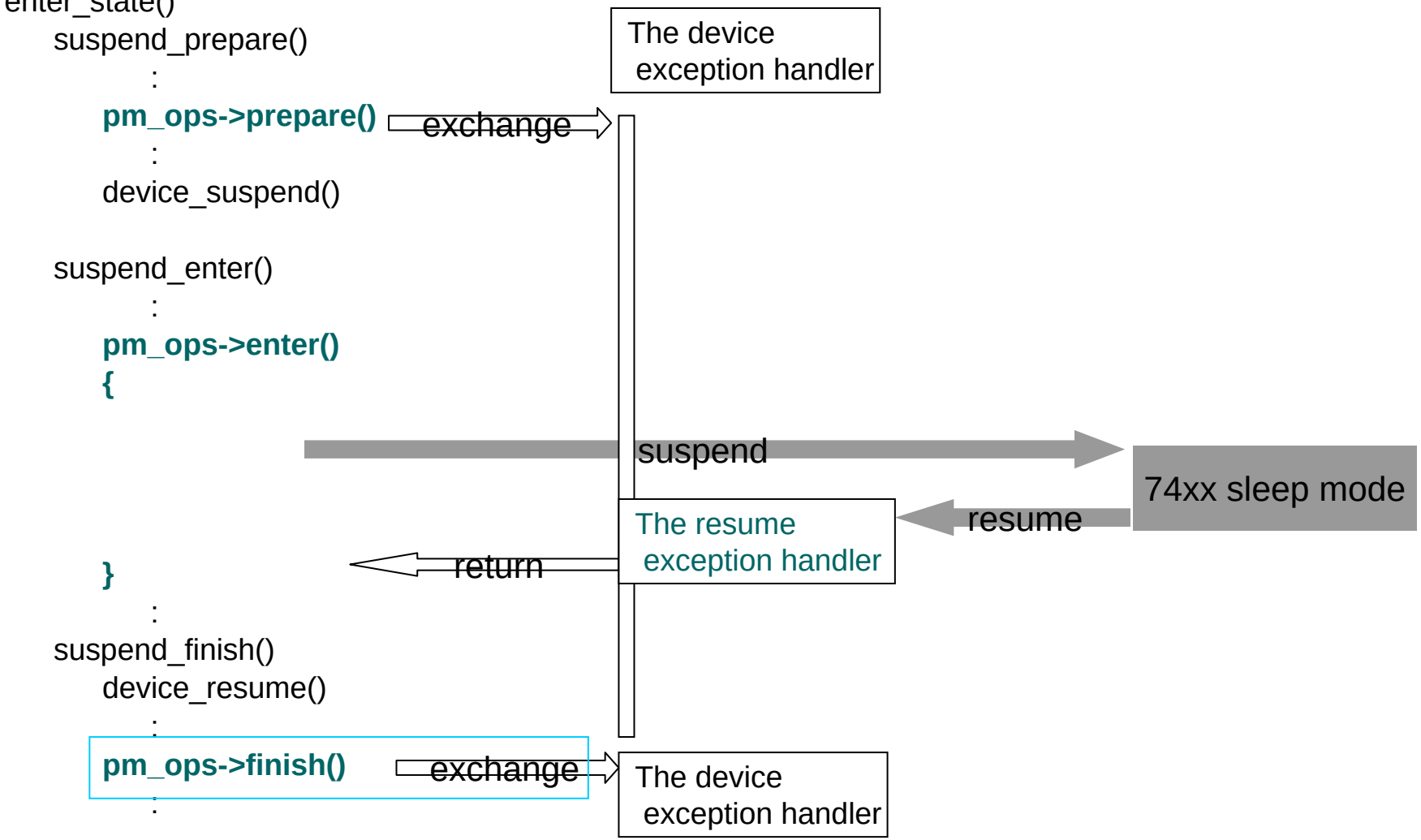
- Our reference comes from the following pm_ops->enter() implemented in the x86 kernel.
  - Save the processor contexts
  - Flush cache

The exception handler for resume only

- The resume starts from the resume exception handler.
  It is set in pm_ops->prepare(),
  and set off in pm_ops->finish().

- Followings is the flow of the handler.
  1. Restore the CPU contexts
  2. Validate timer and cache
  3. Flush cache and TLB
  4. Reassemble MMU
  5. Finish (and return to pm_ops->enter() function)

pm_ops->finish()



pm_ops->finish()

The postprocessing dependent on the CPU architecture is wrapped in pm_ops ->prepare() function.

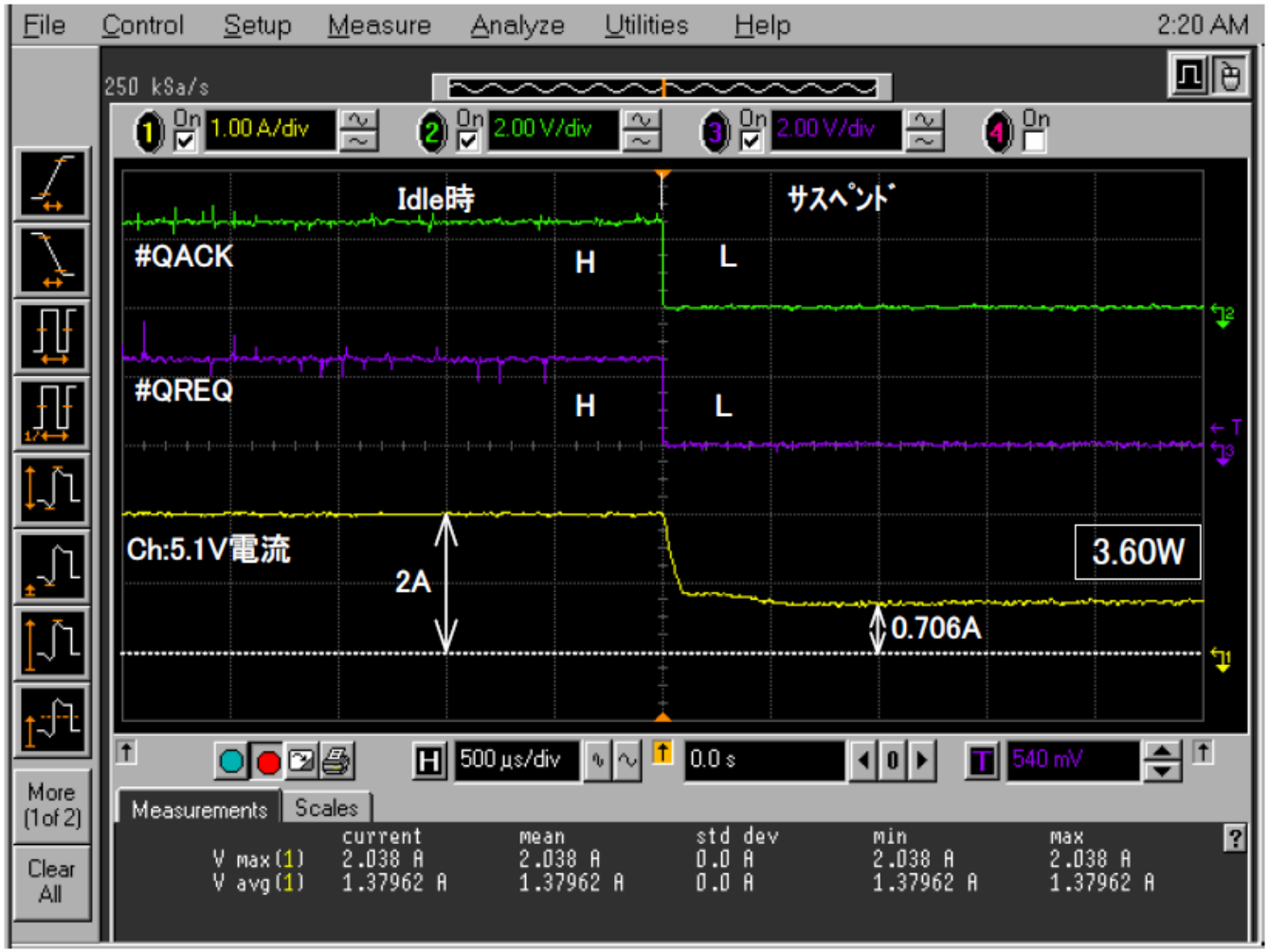
1. Restore the normal exception handler of the resume trigger devices from the resume exception handlers

2. Interrupt enabled

6. Performance

- **We implemented test version. And succeed to reduce power consumption**

  by more than 50%.



7. Our next challenges

- **Measureing the resume speed and optimizing the kernel if needed.**

- **Longer persistence of Idle (Nap mode) with tickless kernel function.**

- **Hibernation function supports for the next target with non-volatile memory**

Acknowledgement

TOSHIBA Corporate Software Engineering Center

Hiroshi Nozuwe

Tsutomu Owa

Masahiro Yamada

Fujihito Numano

TOSHIBA TEC Corporation members

TOSHIBA Information Systems members


TOSHIBA
Leading Innovation