Zephyr™ Power Management

Ramesh Thomas
OTC, Intel
Agenda

- Why Power Management?
- The core concepts behind Zephyr RTOS PM
- Power Management Infrastructures
- Future direction
Think Possible…
Zephyr RTOS PM – Core Concepts

- Multi architecture/board/SOC
- Designed for IoT/embedded
- Customizable for different needs
- Flexibility and variety of options
- Scalable design
- Follow open source process
Zephyr RTOS components
(partial)

Application

API

HAL

Device Management

Power Management

Device Drivers

Kernel

Peripherals

UART

SPI

I2C

Timers

Boot ROM
PM high level layout

SOC Interface
- Boot ROM
- SOC PM Interface
- HAL
- SHIM

Device Management
- Device Drivers

Application
- Thread 1
- Thread 2
- Thread 3
- ...

PM Subsystem
- Kernel
Zephyr RTOS PM Deep Dive
Zephyr RTOS PM features

- Event based kernel idling
- System power management
- Device power management
First a quick intro to the scheduler…
Kernel scheduling and idling

- Priority based scheduling
- Threads wait on semaphore or yield
- Idle Thread scheduled when no other thread can run
- Idle Thread is lowest priority thread
- System Power Management happens in Idle Thread
Inside the Idle Thread

- Kernel scheduler gets invoked from ISR of timer or other event
- If no thread is ready to run, schedules Idle Thread again
Event Based Idling

- Turn off periodic mode timer
- Set one shot timer
- CPU Wait for event

Ordered list of thread wait/timeout periods:
2 secs, 5 secs, 10 secs, 15 secs

No ticks until a thread is ready to run

- Power saved by avoiding unnecessary wake events
- ISR turns periodic mode timer on again
System Power Management
Hooks into the Kernel Idle Thread

- `_sys_soc_suspend(idle time)`
  - Going to idle
- `_sys_soc_resume()`
  - Notify low power state exit or wake event
  - SOC implementation dependent
- Simple and intuitive
  - When idle - save power
  - When active - real-time performance
Triggered from Idle Thread

- Turn off periodic mode timer
- Set one shot timer
- Call hook function `_sys_soc_suspend(idle time)`

- CPU Wait for event
- Turn off periodic mode timer

- Handled?
  - No
  - Yes

- `_sys_soc_suspend()` sets wake event
- Wake -> ISR -> Periodic Mode On -> Scheduler
Inside _sys_soc_suspend
Quick look into HW PM features…
Categories based on HW PM features

- CPU Low Power State
  - CPU clock gated
  - Peripherals active
- SOC Deep Sleep
  - CPU power gated
  - Selective RAM retention
  - Most peripherals lose power
- Different power savings
- Different wake latencies
- Different resume paths
_sys_soc_suspend(<idle time>)

- Setup wake event
- If short idle time
  - Any PM operation that takes less time
  - Enter a CPU low power state
- If long idle time
  - Save states of devices that will lose power
  - Any PM operation that saves more power
  - Enter SOC Deep Sleep
_sys_soc_resume()

- Deep Sleep wake notification
  - Depends on SOC specific implementation
- Wake event notification
  - Optionally called from ISR of wake events
  - Before Kernel schedules other tasks or process nested interrupts
  - Call _sys_soc_disable_wake_event_notification() if not required
Device Power Management
Device Power States

- Classified based on device state retention
  - DEVICE_PM_ACTIVE_STATE
  - DEVICE_PM_LOW_POWER_STATE
  - DEVICE_PM_SUSPEND_STATE
  - DEVICE_PM_OFF_STATE
Device Power Management Overview

- Integrated with Device Management
- Drivers maintain per device power states
- Device APIs to set and get state
- Application, Driver or SOC interface can set states
- Multiple design options to manage device PM
  - Central – Only in _sys_soc_suspend()
  - Distributed – By Applications, Drivers, SOC Interface.
Device PM APIs

device_list_get(struct device **device_list, int *device_count)

device_get_power_state(struct device *device, uint32_t *device_power_state)

device_set_power_state(struct device *device, uint32_t device_power_state)

device_busy_set(), device_busy_clear(),

device_any_busy_check(), device_busy_check()
Device Driver PM Interface

- PM Control Function
- Control codes
  - DEVICE_PM_SET_POWER_STATE
  - DEVICE_PM_GET_POWER_STATE
- Part of Device Interface
- Access only through Device APIs

```c
int (*device_pm_control)(
    struct device *device,
    uint32_t command, void *context);

static int example_control_fn(...)
{
    switch (ctrl_command) {
    case DEVICE_PM_SET_POWER_STATE:
        set state code
        break;
    case DEVICE_PM_GET_POWER_STATE:
        get state code
        break;
    }
    return 0;
}
```
Power Management Examples
PM Example 1 (Distributed Device PM)

- App
  - set_state SUSPEND
  - Yield
  - set_state SUSPEND
  - Yield

- UART
  - set_state SUSPEND

- SPI
  - sys_soc_suspend

- SOC Interface
  - sys_soc_suspend

- Idle Thread
  - sys_soc_suspend
  - SOC Deep Sleep

- HW
  - Idle Thread
PM Example 2 (Central Device PM)

- **App**
  - **Yield**
  - **set_state**
    - SUSPEND
    - ACTIVE

- **UART**
  - **set_state**
    - SUSPEND

- **SPI**
  - **set_state**
    - ACTIVE

- **SOC Interface**
  - **_sys_soc_suspend**
  - **SOC Deep Sleep**
  - **Resume**

- **Idle Thread**

- **HW**
Adding PM Support

- Configure Board, SOC, CPU, Arch
  - (If not done already…)
- Enable/Disable PM feature configs
- `_sys_soc_suspend` / `_sys_soc_resume`
- PM support in device drivers
- PM support in application
Summary
Future direction

- New PM features derived from kernel updates
  - Tick-less kernel
  - Different time unit options
- Add ARC* and ARM* examples
- Distributed Device PM examples

*Other names and brands may be claimed as the property of others.
Summary

- Multi Arch, CPU, SOC, Board support
- Simple and Intuitive hook interface
- Versatile Device PM options
- Configurable, Scalable, Portable
- Open Source
Questions