Linaro connect
Bangkok 2016

Suspend to Idle

Implementing on ARM platforms and investigating improvements

Presented by

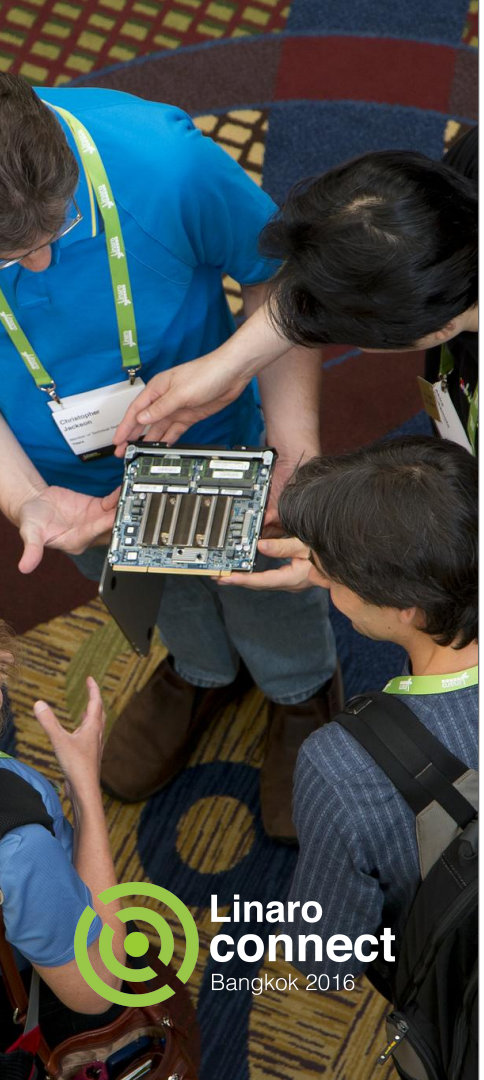
Andy Gross

Date

BKK16-502 March 11, 2016

Event

Linaro Connect BKK16



Agenda

- Introduction
- Why S2I?
- Implementation Requirements
- Hardware Setup
- Advantages
- Test results
- Improvements
- Questions

Why S2I?

Suspend to idle can place the system into a state where the cpus spend much more of their time in idle. This is accomplished by freezing the user space, and also by putting all peripheral devices into lower power states.

The end result is better power savings and reduced latency on resume compared to suspend to ram.

See the following for a good primer:
http://events.linuxfoundation.org/sites/events/files/slides/what-is-suspend-to-idle.pdf

Implementation Requirements

- Wake IRQ support in your IRQ controller.
- Wake interrupts must be configured to deliver interrupts during suspend to idle.
- Devices must support proper wake configurations.
- Wake events must be generated where appropriate.
- Implement the enter_freeze function only if special configuration is required to enter and/or quiesce the system.
- Timekeeping must be suspended during freeze.

Hardware Setup

All of my work was done on Qualcomm platforms. Initially, I started working on the db410c. However, I switched to the Qualcomm APQ8084 based Inforce IFC6540 due to lack of cpuidle support on 8916.

Development was done on the current mainline.

Qualcomm Freeze Implementation

Instead of the normal cpuidle processing, the deepest idle state that implements enter_freeze() is called.

The Qualcomm freeze function implementation does two things:

1. The first is disabling FIQs to get rid of the timer IRQs.
2. The second is placing the cpu in SPC.

Advantages of S2I

- In suspend, peripheral devices have already been quiesced. This leads to the CPUs being in idle state for longer periods of time. In turn, this means we can go into deeper power states to save more energy.
- Bringing CPUs out of idle state is less expensive than hotplugging them.

Test Results

On an APQ8084 w/ a fairly minimal set of devices (i2c, spi, serial, mmc):

- Device idled at 92mA
- After S2I, the device achieved 85mA
- On resume, device settled at 88mA, but after some mmc activity went back up to the expected 92mA.
- Suspend latency was 168ms, 165 of which was MMC related.

Improvements

- Reduce the cost of resume latency by deferring resume of some devices.
- Skip resume of devices that were previously runtime_suspended (provided no special suspend actions were required).
- Reduce cost of suspend latency by having devices place themselves in the lowest power state during idle.
- Find a more unified approach to dealing with timer interrupts and the lockup detector. This could result in not having to have a enter_freeze() function.

Questions?