Ultra-Fast Boot Approach Using Suspend to RAM on Linux based IVI System

2017/6/1
Panasonic Corporation
Kazuomi KATO
1. Introduction
2. Approaches for optimizing startup time
3. Improvement the time at the COLD boot
4. Improvement the time using the WARM boot
5. Conclusions
1. Introduction
### Background

#### Increasing the amount of IVI system software.

<table>
<thead>
<tr>
<th></th>
<th>Year</th>
<th>1996</th>
<th>2006</th>
<th>2017</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Radio / CD / DVD</strong></td>
<td></td>
<td>30MHz</td>
<td>400MHz</td>
<td>1.5GHz</td>
</tr>
<tr>
<td><strong>Navigation</strong></td>
<td></td>
<td>QVGA</td>
<td>WVGA</td>
<td>1920x720</td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Due to the functionality such as smart phone link function, more and more functions and large-scale software are being required.**

**Data size is being increased due to the expansion of the display resolution and the realization of multi-screen being equipped.**

**At the same time, start-up time and quick response of the system are required by the driver much more.**
What benefits for users by reducing startup time?

Obviously necessary to start up the IVI system at high speed.

1. Users wants to start driving quickly after getting into a car and turning ignition ON.

2. Before driving a car, most users expect the navigation system to guide the route for their destination.

3. When going back to the car, users hope that the last display and music are immediately resumed.
Start-up time issues of IVI system:

1. How to realize the quick start-up time suitable for users who are used to smart phone. IVI system needs to be activated as soon as the user gets into the car.

2. How to improve the start up time of turning back the functions of music player, car navigation for user operation.

3. To improve the UX and display resolution, the amount size of data are increased. As a results, reading the data from storage costs much longer. So the start-up time is increased too.

4. In the application of vehicle, although large battery is equipped, the issues of dark current has to be considered, when the car not being used for several weeks.
2. Approaches for optimizing startup time
Methods of optimizing startup time

<table>
<thead>
<tr>
<th>Approach</th>
<th>Typical method</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Eliminate the bottleneck and improve processing</td>
<td><strong>Profiling and Analysis</strong>&lt;br&gt;Improving method based on the profiling and analysis.</td>
<td>Fundamentally improvable.</td>
<td>Know-how is required to specify the bottleneck and improve.</td>
</tr>
<tr>
<td>2. Startup is quickly shown as a user's view</td>
<td><strong>CAN wakeup</strong>&lt;br&gt;Method of starting the system in advance by CAN signal such as unlocking the door.</td>
<td>It seems to start up fast when getting into a car.</td>
<td>It depends on OEM requirements.</td>
</tr>
<tr>
<td>3. Fast return without completely stopping the system</td>
<td><strong>Suspend / Resume</strong>&lt;br&gt;Method of saving the current state to DRAM, turning peripheral device power off, waiting and restarting from the state saved at that time.</td>
<td>Fast resume is possible at turning power ON.</td>
<td>The electric power for maintaining DRAM data is consumed.</td>
</tr>
<tr>
<td>4. Carry out power OFF adding to No.3 approach</td>
<td><strong>Snapshot boot</strong>&lt;br&gt;Method of saving the kernel image of a certain point into the storage and booting up from that point when resuming the system.</td>
<td>Electric power is not consumed as compared with No.3.</td>
<td>If boot image size becomes large, the load time from storage will become a neck.</td>
</tr>
</tbody>
</table>
Our Strategy

At ALS 2016

We introduced the fast boot technique using the PRAMFS* and DRAM backup and improved mainly boot process till application startup.

* Persistent RAM File System

Solution for the next IVI product at this time

- Development on the next generation SoC.
- Improvement of startup including applications.
- Eliminating the bottleneck of the base system is significant as a baseline.
- Realization of the user experience of fast startup like smartphones.
### Methods of optimizing startup time

<table>
<thead>
<tr>
<th>Approach</th>
<th>Typical method</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Eliminate the bottleneck and improve processing</td>
<td><strong>Profiling and Analysis</strong></td>
<td>Fundamentally improvable.</td>
<td>Know-how is required to specify the bottleneck and improve.</td>
</tr>
<tr>
<td></td>
<td>Improvement including applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Startup is quickly shown as a user's view</td>
<td><strong>CAN wakeup</strong></td>
<td>It seems to start up fast when getting into a car.</td>
<td>It depends on OEM requirements.</td>
</tr>
<tr>
<td></td>
<td>Method of starting the system in advance by CAN signal such as unlocking the door.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Fast return without completely stopping the system</td>
<td><strong>Suspend / Resume</strong></td>
<td>Fast resume is possible at turning power ON.</td>
<td>The electric power for maintaining DRAM data is consumed.</td>
</tr>
<tr>
<td></td>
<td>Method of saving the kernel image of a certain point into the storage and booting up from that point when resuming the system.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Carry out power OFF adding to No.3 approach</td>
<td><strong>Snapshot boot</strong></td>
<td>Electric power is not consumed as compared with No.3.</td>
<td>If boot image size becomes large, the load time from storage will become a neck.</td>
</tr>
</tbody>
</table>

- **Profiling and Analysis**: Improvement including applications
- **CAN wakeup**: Method of starting the system in advance by CAN signal such as unlocking the door.
- **Suspend / Resume**: Method of saving the kernel image of a certain point into the storage and booting up from that point when resuming the system.
- **Snapshot boot**: Method of saving the kernel image of a certain point into the storage and booting up from that point when resuming the system.
## Target of the startup time

<table>
<thead>
<tr>
<th>Value</th>
<th>Measurement Case [From a ACC-ON]</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>The startup time</td>
<td>Displaying a map and playing a music on USB-memory</td>
<td>Under 1 sec</td>
</tr>
<tr>
<td></td>
<td>Displaying rear camera image</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Displaying a map and turning the radio on</td>
<td></td>
</tr>
</tbody>
</table>

In addition to the above, we try to reduce these values:
- the dark current at the Suspend state
3. Improvement a startup time at the **COLD** boot
In order to evaluate the startup time, used the following system which is based on Renesas R-Car M3 SoC (ARM Cortex-\textbf{A57@1.5GHz} x 2)

cf) Main changes from ALS 2016
   - CPU: ARM A15(32bit) -> A57(64bit)
   - MEM: DDR3 -> LPDDR4
   - eMMC: HS200 -> HS400
Software Environment for Evaluation

<table>
<thead>
<tr>
<th>Layers</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Navigation, Rear View, FM-Radio, USB-Audio (using QtQuick2)</td>
</tr>
<tr>
<td>Application PF</td>
<td>Qt 5.7; provided from The Qt company</td>
</tr>
<tr>
<td>OS</td>
<td>Linux kernel 4.6.x based on the BSP for R-car M3</td>
</tr>
</tbody>
</table>

Layers Diagram:
- Launch FW
- Power Manager
- App Manager
- DRM/KMS
- OpenGL ES 3.1
- VIN
- VSP-D/DU
- GPU(G6250)
- VIN
- AAC dec
- Control
- GStreamer V1.1
- Launch from init
- Control position /Window
- Own compositor
- Qt 5.7 (supported QtQuick2)
- Qt Apps (launch max 2 apps from main menu)
- Launch FW
- Evaluation board with R-car M3 SoC
- ARM trusted firmware
- U-boot
- U-boot
- Launch from init
- Control position /Window
- Own compositor
- Qt 5.7 (supported QtQuick2)
- Qt Apps (launch max 2 apps from main menu)
- Launch FW
- Evaluation board with R-car M3 SoC
Analysis points

• Divide the system startup sequence into several phases.
• Since the reasons which cause the bottleneck in each phase are different, considering the approach respectively is necessary.

Overview of each startup phase

- Loading time from eMMC device.
- Initialize a few devices.

- Mount partitions
- Loading device driver modules.

Startup flow

- Starting Linux kernel
  - MMU, page, ... initializing
  - Device driver initializing

GUI application startup is dependent on its application PF like toolkit and window system.

Bottleneck mainly caused by I/O
Result: The startup time of bootloader is reduced **120ms** by the followings.

**Bootloader**
- **Load arm-trusted-firmware** 50ms
- **Initialize some devices**
  - loaded env: 80ms
  - init PCI, etc: 50ms
  - others: 166ms

**Kernel**
- **436 ms**
  - **Load dtb, uImage from eMMC** 94ms
  - **Load dtb, uImage and initramfs from eMMC** 130ms

**System init**
- **Service/Apps**
  - **Loaded to memory and switch to kernel boot** 116ms

Before:
- 556ms

After:
- 296ms

- **Deleted ethernet settings (-50ms)**
- **Omitted initialization of unused devices at this phase (PCI, etc) (-50ms)**
- **Reduced redundant logs/delays (-6ms)**
- **Implemented HS400 driver (-49ms)**
- **Adopted initramfs to make faster in the later phase (+85ms)**
- **In dtb, moved the definition value used by bootloader to the beginning (-50ms)**
Optimize kernel boot

Result: The startup time of kernel was reduced 672ms by the followings

Bootloader → Kernel → System init → Service/Apps

Initialize CPU: 1040 ms

- Initialize devices exclude the lower: 300 ms
- Initialize USB: 300 ms
- Initialize SDHI: 100 ms

1040 ms before

368 ms after

- Change to loadable modules because not necessary until running services (-350ms)
- Clock up of CPU frequency (-40ms)
- Tuning defconfig to eliminate error/warning messages (-80ms)
- Deleted redundant logs (-57ms)

- Parallel initialization of PCIe, SDHI using kthread (-145ms)

Panasonic AUTOMOTIVE

Automotive & Industrial Systems
We use original init called “system init”. (Not System V init)

Why originally developed?
- Need the minimum init functionality.
- Standard init process like systemd is not necessary for our product requirements.

Own “system init” for running service process and making device environment for user land.
- Making device files and symbolic links
- Mount filesystem
- Activating a service process

Improvement result:
- Skip initializing eMMC because already done in bootloader. (-200msec)
- Reduce filesystem access and parallel execution (-191msec)
- Result: 900msec -> 509msec
Optimize Service/Apps processing

Result: The startup time is reduced about 3000 msec by the followings

- Modified makefile because implicitly linked libraries were not do prelink. (-0.9 s)
- Fixed to be load plugins (-0.5 s)
- Avoid butting of init/mount and other processing (-0.5 s)
- Delay load of loadable modules not used in this use case
  - Refactoring codes, etc

Ex) USB-Audio Apps
We faced some troubles. The causes were as follows.

- Apps launch is slow.
  - No wait for setting a network before launching apps (-3 sec)

- Application display is slow.
  - Omitted a connection verify processing for interfaces because the display unit’s specific is fixed. (-400 ms)

- Navigation display is slow.
  - There was some error on prelink process. Updated version of the tool for build was fixed. (-1 sec)

- USB-Audio playback is slow.
  - It was influenced by the size and format of album arts.
Result of the startup time at COLD boot

Achieved 4.5 seconds startup of car navigation and USB audio playing.

1. Before (Original boot)

   - Boot: 565 ms
   - Kernel: 1040 ms
   - init: > 900 ms
   - Service: About 6000 ms

2. After

   - Boot: 436 ms
   - Kernel: 368 ms
   - init: 509 ms
   - Service: About 3000 msec

Various tunings

Demonstration on this conference!

- Display the car navigation map.
- Restart the USB audio playing.

However, in order to achieve ultra fast startup (1 second), another approach is required.
4. Improvement a startup time using the WARM boot
Description of Power States (1)

For ultra fast boot, we added two states before the ACC OFF state.

For faster boot, reduced loading images from program storage by partial DRAM backup.

Remark) “SLEEP” state is order to prepare for cases where IGN-ON is made immediately after stops, for example, because of forgetting close the window.
Description of Power States (2)

The transition timing between each power state is as the follows.

**Suspend/Resume System**

- ACC ON
  - RUN
  - SLEEP
  - SUSPEND
  - ACC OFF
- ACC OFF
- Time out several seconds
- Time out 2-4 Week
- +B OFF
- +B ON
- CAN Wakeup
  - ACC ON / IG ON / BLE / TCU / etc...
- CAN Wakeup
  - ACC ON / IG ON etc.

**Trigger of switch state**

**Description of Power States**

<table>
<thead>
<tr>
<th>Power States</th>
<th>Transition Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC ON</td>
<td>RUN</td>
</tr>
<tr>
<td>ACC OFF</td>
<td>SLEEP</td>
</tr>
<tr>
<td>Time out</td>
<td>SUSPEND</td>
</tr>
<tr>
<td>2-4 Week</td>
<td>ACC OFF</td>
</tr>
<tr>
<td>+B OFF</td>
<td>+B ON</td>
</tr>
</tbody>
</table>

**Recognized as ACC OFF**

- +B OFF
- +B ON
- CAN Wakeup
  - ACC ON / IG ON / BLE / TCU / etc...
Power supply block at suspend state

- **Active component**
- **Powered down**
- **DRAM back up**

**IVI Unit**
- Self refresh
- ROM
- DRAM
- NAND

**IVI SoC**
- Sys CPU
- CAN
- I2S
- UART
- LPDDR4
- SPI
- CAN BUS
- Wake-up Trigger output

**Center Display**
- Serializer
- FPD-Link III

**Power supply block**
- CAN Tx/Rx
- CAN
- DCM
- BLE
- Haptic
-AMP
Software block for WARM boot

- ATF jumps to the kernel's resume point by setting a program counter.
- Skip launch & manager process because of already running condition.
- Kernel PM framework and power manager cooperate for restart.
- Apps sus&res methods is called with the notice from the manager.
Sequence for power management (suspend/resume)

- **ACC OFF**
  - SYS (System Reset/Power Control)
  - Specified sleep period
  - SUSPEND/Power OFF

- **ACC ON**
  - SUSPENDON
  - Power ON
  - SUSPEND Mode
  - Jump to Resume function
  - Resume
  - Return from sysfs

**Sleep Mode**
- **ARM Trusted FW**
- **Linux**
- **PowerManager**
- **Service/Applications**
  - Sound/Display OFF
  - Pause Applications

**Suspended**
- **Read/Check Power Status (Suspend?)**
- **Change DDR to Normal mode**
  - Save Registers
  - Restore Registers

**Restart**
- **ACC ON**
- **Sound/Display ON**
- **Continue Application**
Development for the WARM boot.

• Implementation of Suspend/Resume processing for device driver, application.
  – In drivers:
    • Implemented pm_suspend/pm_resume function of custom device drivers.
  – In Apps:
    • Implemented suspend and resume process for applications such as car navigation, USB audio and radio.
Effect of the startup time (COLD Boot)

The startup time is reduced to less than 5 sec.

Elapse times (ms) from ACC-ON
- USB-Audio: 1430
- Navigation: 1620
- Radio: 3863
- Display ON: 4255 msec

Await for HW stabilized
- bootloader
- Boot kernel
- init

Wait for connect Apps

Initialize App: 2280 ms
- Load gst-plugins: 500 ms
- Start Instance: 1400 ms
- Send/Recv events: 100 ms

Load: 500 ms~800 ms
- Load data
- Access music files: 170 ms
- Load music data: 300~600 ms
- Search position: 30 ms

Display ON

Play music

Delay load some modules
Effect of the startup time (WARM boot)

The system can skip almost steps of startup with applying WARM boot.

Wakeup CPUs & Resume devices

Elapse times (ms) from ACC-ON

Service/Apps

Required steps

Skipped block

450 ms

Display ON

Play music

Wait for HW stabilized

bootloader

Boot kernel

init

Application Manager

Radio

RCamera

USB-Audio

Navigation

Soc/Boot Loader

Recall the bootloader

init

Wait for HW stabilized

Play music

Display ON

Elapse times (ms) from ACC-ON

Wait for HW stabilized

bootloader

Boot kernel

init

Application Manager

Radio

RCamera

USB-Audio

Navigation

Soc/Boot Loader

Recall the bootloader

init

Required steps

Skipped block

Wakeup CPUs & Resume devices

Elapse times (ms) from ACC-ON

Wait for HW stabilized

bootloader

Boot kernel

init

Application Manager

Radio

RCamera

USB-Audio

Navigation

Soc/Boot Loader

Recall the bootloader

init

Required steps

Skipped block

Wakeup CPUs & Resume devices

Elapse times (ms) from ACC-ON

Wait for HW stabilized

bootloader

Boot kernel

init

Application Manager

Radio

RCamera

USB-Audio

Navigation

Soc/Boot Loader

Recall the bootloader

init

Required steps

Skipped block

Wakeup CPUs & Resume devices

Elapse times (ms) from ACC-ON

Wait for HW stabilized

bootloader

Bo
Dark Current during ACC OFF

The target dark current: Under 2mA

Current System
R-Car Gen2 + DDR3

Using Next Generation
R-Car Gen3 + LPDDR4
w/ improvement system

6.73 mA

DDR3 1GB
Backup
2.78 mA

DDR3 1GB
Backup
2.78 mA

R-CarH2 I/O
0.41 mA

System Backup
0.76 mA

1.51 mA

LPDDR4 2GB Backup
0.54 mA

LPDDR4 2GB Backup
0.54 mA

R-CarM3 I/O 0.23 mA

System Backup 0.20 mA

- R-Car H2 → R-Car H3
- DDR3 → LPDDR4
5.56 mA → 0.54 mA (@2GB)

- Delete unused circuit block
- Less standby current PMIC

Panasonic AUTOMOTIVE

Automotive & Industrial Systems 31
Result of the startup time

Several use case results of startup time are as follows.

<table>
<thead>
<tr>
<th>Use case</th>
<th>[Target]</th>
<th>COLD boot</th>
<th>WARM boot [1 sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case1</td>
<td>Display a map &amp; Play the music which user listened recently.</td>
<td>4.50 sec</td>
<td>0.45 sec</td>
</tr>
<tr>
<td>Map + USB-audio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use case2</td>
<td>Display a camera image / with predicted course line.</td>
<td>0.63 sec/4.36 sec</td>
<td>0.43 sec</td>
</tr>
<tr>
<td>Rear view Camera</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use case3</td>
<td>Display a map &amp; Play the fm-radio</td>
<td>4.34 sec</td>
<td>0.45 sec</td>
</tr>
<tr>
<td>Map + FM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main Menu</td>
<td>Display the menu &amp; Operation possible</td>
<td>4.23 sec</td>
<td>0.40 sec</td>
</tr>
</tbody>
</table>

(Remarks) Each Qt Apps is intermediate code. If these are machine cords made by QtQuick compiler, maybe more shorter.
5. Conclusions
Summary

1. Achieved 4.50 seconds for the **COLD** boot time includes a Map & USB-Audio.
2. Achieved 0.45 seconds for the **WARM** boot time includes a Map & USB-Audio.

And we could improve a dark current is less than 2mA in suspended status.

→ These techniques can also use in AGL based PF.

![Diagram showing boot times and process]

Before (Original boot)

- Boot: 565 ms
- Kernel: 1040 ms
- init: > 900 ms
- Service: About 6000 ms

1. COLD boot (After)

- Boot: 436 ms
- Kernel: 368 ms
- init: 509 ms
- Service: About 3000 ms

2. WARM boot

- Service: Total 450 ms

Demonstration on this conference!
Future plan

- **Further restoration according to the outer information after resume.**
  - Need to adjust time because from the application viewpoint, it appears the time is leap.
  - Need a quick reconnection with outer devices.
- **More reduce of dark current in suspend mode in order to save the battery**
  - Adoption of next-generation low-power devices (e.g. LPDDR4X / LPDDR5)

<table>
<thead>
<tr>
<th></th>
<th>Current</th>
<th>Next target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup time</td>
<td>4.5 sec / 0.45sec with sample apps</td>
<td>Same level with product apps</td>
</tr>
<tr>
<td>Dark current (12V) 4GB</td>
<td>1.54 mA</td>
<td>Under 1.0mA</td>
</tr>
<tr>
<td>Drivers (on R-car gen3)</td>
<td>Several Device Drivers</td>
<td>All Drivers using on the product PF</td>
</tr>
<tr>
<td></td>
<td>(cf. exclude Ether, CAN,…)</td>
<td></td>
</tr>
<tr>
<td>Apps ready of Suspend/Resume</td>
<td>Sample Apps only</td>
<td>Almost product Apps</td>
</tr>
</tbody>
</table>
Thank you!