Design and Implementation of Automotive Virtualization Based on Xen

Sung-Min Lee
sung.min.lee@samsung.com

June 21, 2018
Acknowledgement

**Major Contributors**

Bokdeuk Jeong, Hanjung Park, Kiwoong Ha, Hyeongseop Shim, Jinsub Sim, Byungchul So, Min Kang, Junbong Yu, Hyeonkwon Cho, Sungjin Kim, Hakyong Kim, Sangbok Han, Vasily Leonenko, Dmitrii Martynov, Andrew Gazizov, Sergey Grekhov, Ildar Ismagilov, Imran Navruzbekov, and colleagues from S.LSI
Contents

Ⅰ  Why Virtualization
Ⅱ  Key Requirements
Ⅲ  Our Design and Implementation
Ⅳ  Evaluation
Ⅴ  Demo (Video Clip)
Why Virtualization in Automotive Industry?

- Easy Firmware Update for Automotive System
- Enhanced In-Vehicle Security
- Cost/Complexity Reduction
Key Requirements for Automotive Virtualization

- Fast Startup
- Fault Recovery for Reliable In-Vehicle Services
- High Performance Graphics Support for Guest OS
Overall Architecture

**Dom0 (Cluster OS)**
- Cluster OS UI
- Virtual I/O Backend Drivers (Storage, N/W, Serial, Input, Audio, Camera)
- PV Backend Drivers for DomU Passthrough (clocksource,regulator,pinctrl)
- Native Device Drivers

**DomU (IVI OS)**
- IVI Platform (e.g., GDP)
- Virtual I/O Frontend Drivers (Storage, N/W, Serial, Input, Audio, Camera)
- Virtual Graphics Frontend

**Xen Tools**
- Xen Tools

**Fault Recovery**
- Fault Recovery

**PV Frontend Driver for Passthrough**
- PV Frontend Driver for Passthrough

**Fast Startup**
- Fast Startup

**Fault Recovery**
- Fault Recovery

**Xen**
Fast Startup (1/3)

Reducing Memory Initialization Time

Xen Heap Initialization
- Open source Xen is based on per-page xen heap insertion
- Reduce the number of the *init* function invocations and remove spinlocks by implementing a new initialization function
- Skip `scrub_heap_pages()`

Static VM Memory Assignment
- Assign specific addresses and size for VMs in device tree and exclude them for Xen Heap insertion
- The number of pages to initialize is reduced and double definition overhead (*init + VM assigning*) is disappeared

Call `init` function by the number of total pages times

Don’t have to call `init` function many times

Allocate Dom0 memory

Don’t have to call `init` function many times

Double assignment

Original ver

Dominant chunk

Order of 2

Free heap pages

Domain direct assign pages

Double assignment disappeared

Xen Heap initialization range

Double assignment

Original ver

Xen Heap

Initializes total pages as heap

0xFFFFxxxx

Init

0x8000xxxx

Initialize total pages as heap

Dom0 memory

Dom0 Memory

Dom1 memory

Dom1 Memory

Double assignment

Samsung ver

Double assignment disappeared

0x8000xxxx

Xen Heap initialization range

Original ver

Samsung ver
Reducing DTB Lookup and Creation Time

**Relocate DTB Nodes**
- Scanning all nodes in device tree takes a lot when its size is large.
- Group scattered “chosen” and “memory” nodes into one node and relocate it at the head of device tree to search faster.

**Flattened Device Tree Description**
- Visit all nodes to find information nodes.
- Doesn’t have to visit all nodes.
- Original ver.
- Samsung ver.
- Special node “xen-early-scan-end”.

**Improve DTB Creation Algorithm**
- Creating device tree for control OS takes most of time.
- FDT library uses the most naïve algorithm when comparing strings.
- Replace it with better one, Boyer-Moore algorithm with time complexity $O(m+n)$. 

---
Fast Startup (3/3)

Assign Big Core and Concurrent VM Creation During the Xen Startup

**Assign Big Core for Dom0 Creation**
- Exynos8890 uses little core for bootstrap processor (CPU0)
- Assign tasklet to sleeping big core to create control OS
- CPU0 waits until domain creation done

**Concurrent VM Creation**
- To shorten the time spent on XL for DomU creation, Samsung moves its data structure initialization to xen startup
- Assign a guest OS creation tasklet to another sleeping big core and create simultaneously with control OS creation
Fault Recovery Scenario

1. **Reset Request**
   - `void panic (const char *fmt, ...)`
   - `HYPERVISOR_sched_op(SCHEDOP_shutdown, &r);`

2. **Execute reset VM**
   - `void panic (const char *fmt, ...)`
   - `machine_restart`

3. **Set timeout/notification period**

4. **Crash**
   - `void panic (const char *fmt, ...)`
   - `machine_restart`

5. **Kick**
   - `Xen Shutdown Manager`

6. **XEN**
   - `Samsung VM Reset Manager (Relaunch VMs)`
   - `HW Reset`

7. **HW Watchdog Timer**
   - `Timeout`

8. **Exynos8**
   - `Kick`
## Fault Recovery (2/3)

### Conventional VM Restart vs Our VM Reset

<table>
<thead>
<tr>
<th>VM Restart</th>
<th>VM Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>✷ Destroy a domain</td>
<td>✷ Destroy devices only</td>
</tr>
<tr>
<td>✷ Create a new domain</td>
<td>✷ Just reuse almost all resources like vcpu structure, shared_info, p2m table, etc</td>
</tr>
<tr>
<td>✷ Free/Allocate all resources</td>
<td>✷ Reset the existing domain</td>
</tr>
<tr>
<td>✷ Use hypercalls for destroy domain and create domain in xen</td>
<td>✷ Add a new hypercall for reset domain in xen</td>
</tr>
<tr>
<td>- XEN_DOMCTL_destroydomain</td>
<td>- XEN_DOMCTL_samsung_reset</td>
</tr>
<tr>
<td>- XEN_DOMCTL_createdomain</td>
<td></td>
</tr>
</tbody>
</table>
Fault Recovery (3/3)

VM Reset Procedure

Dom0 (Control domain)

- initiate_domain_create
  - Reload kernel binary
  - Reload device tree from memory
  - Reset CPU registers
  - xs_introduce_domain

XL

- libxl_domain_reset
  1. xs_release_domain
  2. Destroy devices

- devices_destroy_cb
  - Request domain reset

XS

- Reset flags setting

- Release Domain

DomU

- DomU unpause

- Hypercall: XEN_DOMCTL_unpausedomain

- Hypercall: XEN_DOMCTL_samsung_reset

- send_global_virq(VIRQ_DOM_EXC);

- domain_shutdown (Xen Shutdown Manager)
  - Set reason flags for shutdown
  - Pause vcpus

- Hypercall: HYPERVISOR_sched_op(SCHEDOP_shut_down, &r);

- Xen Watchdog Timeout

- Relaunching complete

Crash

VM Reset Procedure

- Hypercall: HYPERVISOR_sched_op(SCHEDOP_shut_down, &r);

- Xen Watchdog Timeout

- Relaunching complete
I/O Virtualization

**PV I/O Drivers**
- vNet, vStorage, vConsole: utilized existing open source codes
- vInput: provides touch
- vCamera, vAudio: developed from scratch
- Performance enhancement with direct cache operations on foreign pages

**Passthrough I/O Drivers**
- virtual clocksources, regulators and pinctrollers are provided to DomU for device passthrough

---

**Dom0 (Cluster OS)**
- vCamera BE
- vAudio BE
- vNet BE

**DomU (IVI OS)**
- vCamera FE (as a V4L2 driver)
- vAudio FE (as an ALSA driver)
- vNet FE

**I/O device**

---

**Dom0 (Cluster OS)**
- Pass-thru Device Drivers
- Native Device Drivers (Pinctrl, Regulator, Clocksource)

**DomU (IVI OS)**
- vCamera FE
- vAudio FE
- vNet FE

**Xen**
Graphics Virtualization (1/3)

Overall Architecture of Graphics Virtualization

Dom0 (Cluster OS)
- Samsung Virtual Display Manager (SVDM)
- Samsung Virtual Graphics Library (SVGL) BE
- GL Render
- Native Graphics Library
- Native Graphics Driver
- vGPU BE

DomU (IVI OS)
- Application
- Display Server
- Graphics Control Manager (GCM) FE
- Virtual DRI (GBM, Wayland Buf)
- Virtual EGL FE
- GLES APIs
- Codecs
- Virtual Display
- Native Window
- Native Graphics Library
- GL Render
- RenderThread
- Virtual OpenGLES ES BE
- Virtual EGL BE

Xen
Key Features and Performance Optimization

- Latest Graphics APIs Support: OpenGL ES v3.2/EGL v1.5 Support
- Shared Memory Between BE and FE With Simple Protocol
- Batch Processing for Transferring APIs
- Selective APIs Transfer to BE
- Adaptive Interrupt/Poll Processing in FE
- Efficient Graphics Buffer Management

Adaptive Interrupt/Poll Processing

- Dynamically Choosing Either Interrupt or Poll Based on Request Rate Between SVGL FE and vGPU FE
- Recalculate “the rate” at Every 250 ms
  \[ \text{Rate} = \frac{\# \text{of Buffer Flush OPs}}{\text{Time}} \]
- Performance Improvement Over Interrupt Method with the Threshold “0.3” in Policy (75 cmd buf flush OPs/250ms)

Policy

\[ \text{Threshold: } 0.3 \leq \# \text{if } 0.3 \leq \text{the calculated rate then Polling else Interrupt} \]
Efficient Graphics Command Buffer Management

**Graphics Command Buffer Transfer**
- Accumulate each Opcode of API whose return type is "void" and Data to the Buffer
- Flush Buffer for "glFlush", "glFinish" and "non-void return type APIs".
- Flush Buffer in the Case of "Buffer Full"

**Variable Size Buffer Allocation**
- Dynamically Resizable Buffer Management: Initial Buffer Size (30MB)
- Expand Buffer Pool Based on Available Buffer Size and Requested Size from SVGL FE
- Shrink Buffer When App is Terminated
Evaluation: Startup Performance

Time Measurement Unit: msec

Xen Startup Time Changes

Measured Time: start_xen() ~ init_done()
[xen/arch/setup.c]

<table>
<thead>
<tr>
<th>Method</th>
<th>Original ver</th>
<th>Samsung ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Line</td>
<td>1,479 msec</td>
<td>69 msec</td>
</tr>
<tr>
<td>Reduce Memory Initialization</td>
<td>144</td>
<td>10</td>
</tr>
<tr>
<td>Improve DTB creation</td>
<td>103</td>
<td>10</td>
</tr>
<tr>
<td>Bigcore Assigning</td>
<td>69</td>
<td>1</td>
</tr>
</tbody>
</table>

Time Measurement for Sections

- Page Scrubbing: 471 msec
- Domain creation: 59 msec
- Heap & Hardware Initialization: 685 msec
- Scanning DTB: 313 msec
Evaluation: VM Fault Recovery Time

- VM Restart Time vs. VM Reset Time (sec)

<table>
<thead>
<tr>
<th>Time Measurement</th>
<th>libxl_domain_unpause function [in xl_cmdimpl.c]</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM Restart</td>
<td>7.1</td>
</tr>
<tr>
<td>VM Reset</td>
<td>0.97</td>
</tr>
</tbody>
</table>

**Function Call:**
- `libxl_domain_unpause` (in `xl_cmdimpl.c`) for VM Restart
- `xl reset` and `xl restart`
### Evaluation: I/O Performance

**Storage and Network Throughput**

<table>
<thead>
<tr>
<th></th>
<th>Storage Throughput</th>
<th>Network Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq. Read</td>
<td>Seq. Write</td>
</tr>
<tr>
<td>Dom0</td>
<td>228MB/s</td>
<td>132MB/s (98.5%)</td>
</tr>
<tr>
<td>DomU</td>
<td>216MB/s (94.7%)</td>
<td>134MB/s</td>
</tr>
<tr>
<td></td>
<td>Dom0</td>
<td>DomU</td>
</tr>
<tr>
<td></td>
<td>129Mbits/s (92.8%)</td>
<td>129Mbits/s</td>
</tr>
<tr>
<td></td>
<td>Dom0</td>
<td>DomU</td>
</tr>
</tbody>
</table>
Evaluation: Khronos CTS

Number of Passed Test Case by Khronos CTS

<table>
<thead>
<tr>
<th>OpenGL ES Version</th>
<th>Dom0</th>
<th>DomU</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL ES 2.0</td>
<td>1228</td>
<td>1228</td>
</tr>
<tr>
<td>OpenGL ES 3.0</td>
<td>2991</td>
<td>2991</td>
</tr>
<tr>
<td>OpenGL ES 3.1</td>
<td>1392</td>
<td>1392</td>
</tr>
<tr>
<td>OpenGL ES 3.2</td>
<td>2511</td>
<td>2511</td>
</tr>
<tr>
<td>OpenGL ES EXT</td>
<td>261</td>
<td>261</td>
</tr>
</tbody>
</table>

Total # of Passed TC: 8,383
Evaluation: Graphics Performance

Glmark2-es-wayland 1920x1080 Off-screen

<table>
<thead>
<tr>
<th>Score</th>
<th>Dom0</th>
<th>DomU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>660</td>
<td>639</td>
</tr>
<tr>
<td>2nd</td>
<td>662</td>
<td>638</td>
</tr>
<tr>
<td>3rd</td>
<td>663</td>
<td>636</td>
</tr>
<tr>
<td>4th</td>
<td>662</td>
<td>640</td>
</tr>
<tr>
<td>5th</td>
<td>662</td>
<td>639</td>
</tr>
<tr>
<td>Average</td>
<td>661.8</td>
<td>638.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ratio</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>96.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td>96.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td></td>
<td>96.7%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td></td>
<td></td>
<td></td>
<td>96.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5th</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96.5%</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96.5%</td>
</tr>
</tbody>
</table>
**Demo**

**System Environment**
- Exynos 8890 Octacore (Big core: 2.28GHz, Little: 1.58GHz)
- 6 GB DRAM
- 32 GB MMC
- Mali T880 MP12
Thank you