Hypervisor based approach for integrated cockpit solutions

Conference Paper · September 2018

DOI: 10.1109/ICCE-Berlin.2018.8576222

Shravan Karthik
Texas Instruments Inc.

Hypervisor based approach for integrated cockpit solutions

Shravan Karthik, Karthik Ramanan, Nikhil Devshatwar, Subhajit Paul
Texas Instruments (I) Pvt. Ltd
India

Vishal Mahaveer, Sheng Zhao, Manoj Vishwanathan
Texas Instruments Inc.
USA

Chetan Matad
PathPartner Technology Pvt. Ltd
India

***Abstract*—Rapid advances in SoC architectures have resulted in a shift in the automotive industry to integrate in-vehicle entertainment (infotainment) and selected ADAS functionality onto a single system, reducing system cost and payload. Standalone electronic control units (ECU) for surround-view (SRV), rear-view camera (RVC), heads-up display (HUD) and digitally instrument cluster are being substituted by integrated cockpit systems. Integrating the standalone ECUs into a single system represents a dichotomy of safety versus generality. Advanced driver assistance systems (ADAS) are expected to be fault-tolerant and are traditionally managed by a safety-certified RTOS. Infotainment applications have fewer constraints on safety but stress on interoperability. Thus, infotainment applications are driven by open-source libraries and are managed by a high-level operating systems (HLOS). To achieve an integrated cockpit system while adhering to each systems constraints, a hypervisor based solution is proposed. The hypervisor facilitates running multiple operating systems simultaneously, partitioning of system resources and providing virtualization of shared peripherals. The result of the proposed architecture is an integrated cockpit system with ADAS and infotainment components isolated from each other, and the ability to run multiple HLOS sharing the peripherals.**

***Index Terms*—Hypervisor, Automotive, Integrated Cockpit, ADAS, Infotainment**

I. INTRODUCTION

Electronics within the vehicle has been rapidly growing for the past two decades. The automotive electronics cost as a percentage of total car cost has increased from 20% in 2000 to 35% in 2017 [1]. The recent spike in electronics is attributed to integration of infotainment and ADAS systems to mainstream market segments, leading to a demand for complex and compute intensive applications. Today, infotainment systems support high-resolution displays, video-playback, and graphics applications such as navigation and digitally re-configurable clusters. These systems use Linux or QNX OS with customization's to suit OEM requirements. Emerging standards such as Automotive Grade Linux (AGL), Android for Automotive are gaining prominence due to access to a wide range of applications native to the respective eco-system.

Automotive applications have mixed criticality, i.e certain applications are more safety and mission critical than others [2]. Mission critical applications are operated in real-time environments where latency is minimal. They also need to be fault-tolerant and have deterministic execution cycle. These applications constitute ADAS functionality such as auto emergency braking (AEB), driver-monitoring system (DMS) and SRV. Applications with fewer safety constraints run on a HLOS such as Linux, leveraging available libraries for rapid application development. Apps in an infotainment system include media player, navigation. In most vehicles today there are stand-alone ECUs driving each system.

Advances in SoC architecture facilitates multiple processing cores within the same chip. These processing cores are diverse both in functionality and architecture. Certain processing cores within the SoC are optimized for specific tasks and serve as hardware accelerators, while others are generic purpose cores. The resulting SoC is one with a heterogeneous architecture capable of running multiple systems simultaneously.

Each ECU is powered using a SoC. Standalone ECUs reduce the overall complexity since the system is designed to perform a specific task. However, this increases the bill of materials (BOM) cost and the payload of the vehicle leading to inefficiencies. With SoCs now capable of handling complex systems, OEMs are integrating multiple ECUs into a single system.

In this paper, an architecture is proposed to integrate ADAS systems with infotainment systems driven by two or more HLOS using a hypervisor. To illustrate this, an integrated cockpit system running Linux, Android and TI-RTOS is implemented. Digital instrument cluster and tell-tale signs are driven by Linux, while applications such as navigation, radio and video-playback are managed from Android. The ADAS system managed by TI-RTOS runs a 3D SRV application.

The organization of the paper is as follows: Section III describes the requirements of each system and the proposed architecture, Section IV describes the implementation of the architecture using an integrated cockpit use-case, Section V presents the results and performance numbers for a reference implementation of the architecture and Section VI discusses impact of the architecture on existing systems and next steps.

II. RELATED WORK

There exist commercial hypervisor solution for embedded systems. Greehills Integrity, Freescale's Topaz, open source variants such as Xen, Jail-house are some examples.

Thiebaut et.al [3] present a secure hypervisor architecture to integrate into existing automotive systems. The paper describes methods to immure systems from DMA attacks and provides a

framework for secure inter-process communication. Reinhardt et.al [4] present a embedded hypervisor architecture for safety automotive systems and demonstrate this architecture using AUTOSAR and an RTOS. The paper describes in detail a micro-kernel approach to isolate safety systems using a hypervisor without a memory-management unit. To virtualize peripherals between virtual HLOS Joe et.al [5] describe a method to achieve GPU-sharing through API remoting.
Integrating infotianment and ADAS systems onto a single unit can also be done without a hypervisor. Omerovic et.al [7] describes an architecture to integrate these two systems through communication using a network stack. These systems don't provide isolation, and are susceptible to failure with little scope for recovery. Integrating infotainment and ADAS systems requires modifications to meet each systems constraints. Marathe et.al [6] describe a boot-flow to optimize the boot-up time required for bring up ADAS systems by loading the firmware from a boot-loader.
The work mentioned above demonstrates integrating hypervisor solutions into embedded systems. They also describe resource partitioning between systems and sharing of certain peripherals within the system. The hypervisors are used to integrate two or more HLOS or an RTOS and a HLOS. They are also responsible for partitioning of resources throughout the system. To the best of our knowledge there exists no solution to integrate safety critical ADAS functionality with multiple HLOS running inside a hypervisor. In our solution, the hypervisor only manages resources required by the HLOS, ensuring that the ADAS system operates with complete autonomy.

III. Design Goals and System Architecture

In this section, the design requirement of each subsystem is discussed. Subsequently, a generic system architecture is proposed, which accommodates these requirements.

A. Isolation of safety subsystem (ADAS system)

ADAS systems have stringent constraints on latency, response time and operate in an RTOS environment with deterministic execution cycles. The degree of robustness of system execution depends on the severity of the hazard posed in case of system failure. This is mapped to an Automotive Safety Integrity Level (ASIL) level defined by the functional safety for road vehicles standard – ISO 26262 [8]. To ensure ASIL compliance, resources required for the functioning of systems must be completely owned within the system. There shouldn't be any dependency on resources outside the system environment. This includes device-drivers, memory, CPU, hardware accelerators, data-control logic, and the end application. Requests to peripherals within this subsystem are initiated by master-core(s) running the RTOS. This is trivial in the case of a standalone ECU – where the entire system runs within the same environment, however, while integrating with quality management (QM) components, there could be a need for the latter to access peripherals within the former. In such scenarios, peripherals owned by the safety-critical subsystem must be isolated from the rest of the system, and any requests from outside must be proxy-ed through the safety subsystem. Facilitating requests from outside the safety subsystem requires a fail-safe inter process communication (IPC) mechanism. A fail-safe IPC monitors the state of the endpoints (target, initiator) and ceases information exchange either of the endpoints are in a non deterministic state. The IPC is also responsible for congestion control and precludes flooding of service requests. The natures of the requests are monitored using firewalls, which validate veracity and nature of the request. Upon validation, the master core in the safety subsystem receives the request and initiates actions to it. This ensures the master-core of the safety-subsystem continues to remain the sole initiator of peripheral access while ensuring entities outside this subsystem continue functioning. Fig. 1 describes a cluster application running on HLOS accessing CAN data from safety-subsystem.

B. Security through a closed system (Informational ADAS system)

Applications such as digitally re-configurable cluster serve as an Informational ADAS (Info-ADAS) system. While these systems have fewer constraints on safety, graphics performance and an interactive user-interface are key requirements. To reduce development time, Info-ADAS systems run from a HLOS such as Linux which provides ample support for graphics libraries. Leveraging its reach through open-source, Linux also has driver support for a wide range of peripherals and hardware-accelerators such as GPUs capable of accelerating rendering of 3D-content. However, the trade-off is Linux doesn't have deterministic execution cycles and being a monolithic kernel [9], it's susceptible to a crash in-case of a faulty driver or a rogue memory access. Running Info-ADAS in a closed system prevents exploitation of certain system vulnerabilities. The system thus supports limited functionality.

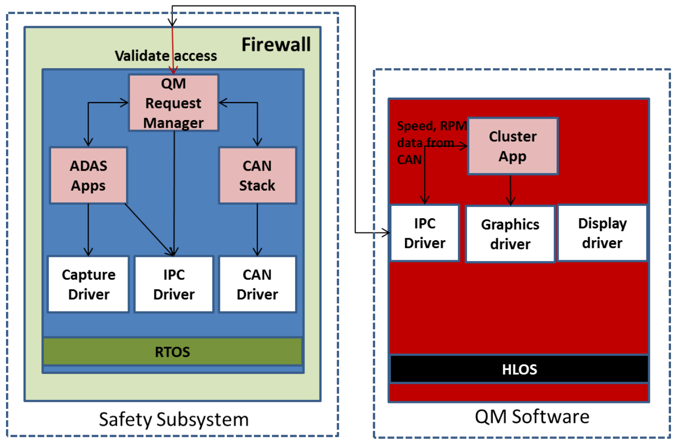

Fig. 1: Information exchange between safety subsystem and QM software

Unlike the safety-critical ADAS subsystem, resources necessary for functioning of this system aren't explicitly owned by the HLOS, but are distributed among the safety-subsystem, HLOS. To access resources owned by the safety-subsystem, a service request is sent, rendering the Info-ADAS system as client to the safety-subsystem. Resources owned by the safety subsystem are excluded from the Info-ADAS system. The service request accesses the information through a fail-safe IPC, but has no control on the resource. Alternatively, resources used exclusively by the Info-ADAS system are owned by the HLOS.

C. Feature rich infotainment system

Some of the key use-cases in an infotainment system are:

- Video playback for rear seat entertainment
- Radio and audio playback
- Internet connectivity
- Navigation

The infotainment system is designed to be feature rich, providing in-vehicle entertainment. The infotainment system needs access to processing blocks capable of video decode and radio acceleration. This system also serves as a gateway to external networks enabling applications to connect to cloud.
Infotainment systems operate in HLOS customized for automotive use-cases. Android for Automotive and AGL are some examples. These platforms also provide access to wide range of third party applications. Since this system is connected to external networks it is vulnerable to exploitation. Thus, applications which pose minimal hazard on failure operate within the infotainment system. Similar to the Info-ADAS system, all resources needn't reside with the HLOS.

D. Virtualization and partitioning of system resources

To accommodate multiple HLOS running on the same processing core a hypervisor is used. Each OS runs inside a virtual machine (VM). The hypervisor over-commits system resources through virtualization which can be achieved in two ways [10]:

- **Complete virtualization:** Each component along with its environment runs unmodified in a VM which interfaces with the hardware. No changes are required in the drivers responsible with interfacing the resource. Instructions to the shared peripheral are trapped by the hypervisor and stores it in a queue, while emulating a hardware device response that is sent back to the driver. The instruction execution is deferred until the resource becomes available.
- **Para-virtualization:** Each guest OS is aware of the other. Peripherals are shared through client-server architecture with changes in the driver for each guest OS. Para-virtualization requires the guest OS owning the resource to implement a back-end to service requests from other guest OS. Correspondingly other guest OS require modify the driver to provide a front-end which redirects request to the hypervisor. The hypervisor then facilities transmission of requests from the front-end to the back-end. In para-virtualization The entire system work as a single cohesive unit.

Para-virtualization differs from information access between Info-ADAS and safety subsystem. In para-virtualization a device entry and the corresponding driver exists for the shared peripheral. The front-end driver is modified to initiate a hypercall along with the necessary data. The hypervisor then maps the request to the back-end driver. However, resources owned by the safety subsystem are removed from the list of devices accessible by the Info-ADAS subsystem. A software stub exists on both systems to receive and transmit requests / information. Information is exchanged using a fail-safe IPC.
Resources which aren't used by both systems are partitioned based on requirement.

IV. IMPLEMENTATION

An integrated cockpit system comprising the following was developed to demonstrate the proposed architecture:

- Surround-view running on TI-RTOS (safety-critical ADAS system)
- Digitally re-configurable cluster running on Linux (Info-ADAS system)
- Services running on Android for Automotive (infotainment system)
- Xen hypervisor managing Android and Linux

The application was run on Texas Instruments Jacinto™ DRA76x chipset, catering to automotive market for infotainment and ADAS use-cases.
In this section an overview of the DRA76x SoC is presented following which the boot architecture is discussed. Isolation of the ADAS system and integration of Linux and Android through complete and para-virtualization is described.

A. Overview of DRA76x Architecture

The DRA76x is a heterogeneous SoC catering to high-end integrated cockpit use-cases [11]. The dual Arm® Cortex® - A15 processors serves as the MPU . The dual Arm® Cortex® -M4 subsystem (IPU) and the C66x Digital Signal Processing (DSP) subsystem serve as auxiliary cores. The SoC also houses accelerators for vision (Embedded Vision Engine), video encode and decode (Image and Video accelerator High Definition) and a 3D GPU (PowerVR® SGX544). There also exists an imaging, capture, radio and display subsystem.
A high speed interconnect interfaces all subsystems. RAM is shared across the system and stores the code and data sections for each core. Multi-level cache is present on each core to improve performance. Each core has registers which manages its internal state. The MPU is capable of running in hypervisor mode.

B. Applications running on the Integrated cockpit system

Surround-view: Surround-view (SRV) is an application where multiple camera feeds are stitched together to get a unified $360°$ view. Video is captured along the four cardinal directions and passed through a sequence of algorithms to generate views along multiple orientations. The application is run from the IPU and DSP, both running TI-RTOS. Drivers to control the capture subsystem run on the IPU. The algorithms

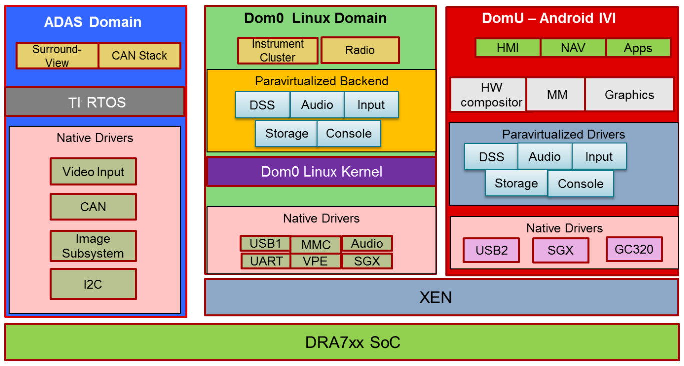

Fig. 2: Integrated Cockpit system – Resource partitioning

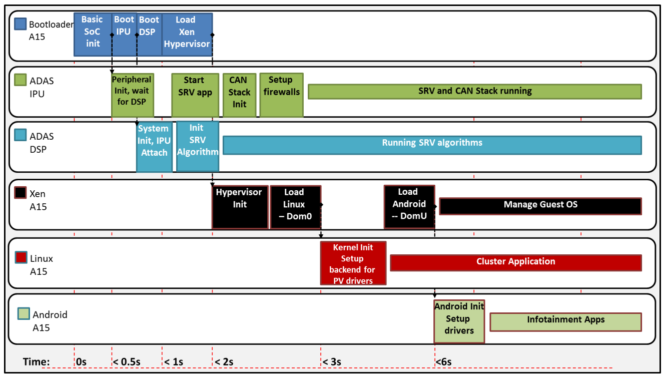

Fig. 3: Boot-flow of Integrated cockpit system

to generate a stitched image are optimized to run on the DSP [12]. In this implementation the firmware for the SRV algorithms is loaded on the C66x DSP-1.
Control Area Network Stack: Information from sensors is propagated to the SoC using a Control Area Network Stack (CAN). Conversely, messages from the SoC are sent over CAN to drive actuators. The CAN stack resides on the IPU.
Digitally re-configurable cluster: A digital instrument cluster is a set of instrumentation, displayed with a digital interface rather than traditional analog gauges. Based on information fed from the CAN stack a 3D dashboard is created using the GPU. The cluster application runs in the Linux domain. The cluster application requires GPU for 3D-graphics acceleration and display. The GPU is completely virtualized while the display is para-virtualized
Software Defined Radio (SDR): The SDR application supports AM/FM, HD Radio^{TM} and DAB radio standards. The radio app, which resides on the Linux domain, configures radio tuners and controls demodulators in the C66x DSP, a fixed and floating point accelerator suitable for audio and data decoding. In this implementation, the radio firmware is loaded to C66x DSP-2. The UI is an Android application that sends user commands to the radio application on the Linux domain via Ethernet.
Android for Automotive: Applications on Android are tailored for automotive use-cases. These included navigation and voice assistant. The applications run from the Android domain and require connectivity to the internet, due to which the network stack runs on Android. Navigation requires graphics to be offload to the GPU and a display driver to post content.
Video playback: For rear seat entertainment, video playback is integrated into the Android domain using third party applications. The video decode is off-loaded to the video decode subsystem – IVAHD.
Fig. 2 describes the distribution of peripherals used by the integrated cockpit system.

C. Boot-architecture

Fig. 3 describes the boot-flow of the system. The steps involved are:

1) **Basic SoC power management and loading of boot-loader to MPU :** Upon board power-up, basic power-management is performed following which the ROM is initialized. ROM further initializes the MPU and boot-media. The boot-loader is read from the boot media and is loaded on the MPU.
2) **Boot-loader:** The boot-loader initializes DDR and configures peripherals needed by the ADAS system. Pinmux and boardmux configurations are also performed at this stage.
3) **Loading of ADAS system firmware on auxiliary cores (IPU and DSP):** Since all peripherals needed by the ADAS system are owned by the IPU, the firmware can be loaded even before the Linux kernel.
4) **Boot-loader jumps to Xen hypervisor:** After the remote-core firmware is loaded the boot-loader jumps to the hypervisor. The hypervisor configures the second-stage address translation unit and proceeds to load 'Dom-0'.
5) **Xen loads Linux as 'Dom-0':** 'Dom-0' is setup and Linux kernel begins initialization. Peripherals owned by Linux are initialized and backend drivers for para-virtualized peripherals are loaded. Fail-safe IPC between Linux and the ADAS subsystem is also established.
6) **Xen loads Android as 'Dom-U':** Completion of kernel initialization ensures that back-end for para-virtualized peripherals are setup. The hypervisor then loads Android as 'Dom-U'.

D. Surround-view and CAN Stack integration

After the boot-loader takes the IPU out of reset the ADAS system begins its functionality. Resources required by the ADAS subsystem are identified and included as part of the firmware. The firmware sets up the necessary drivers and initializes required peripherals. After the CAN stack and surround view applications have been initialized, firewalls are set up. The firewall prevents external systems from accessing the peripherals owned by the IPU. The firewall also prevents indiscriminate overwrite of the IPU and DSP code and data-sections by external systems. After the firewalls are setup, a thread is spawned which keeps track of requests outside the

safety-subsystem.
Once the Linux system has completed boot-up, Linux requests the following information from the safety subsystem:

- Stitched SRV image to post onto display
- CAN information such as speed, RPM for rendering cluster

E. Para-virtualization of display subsystem

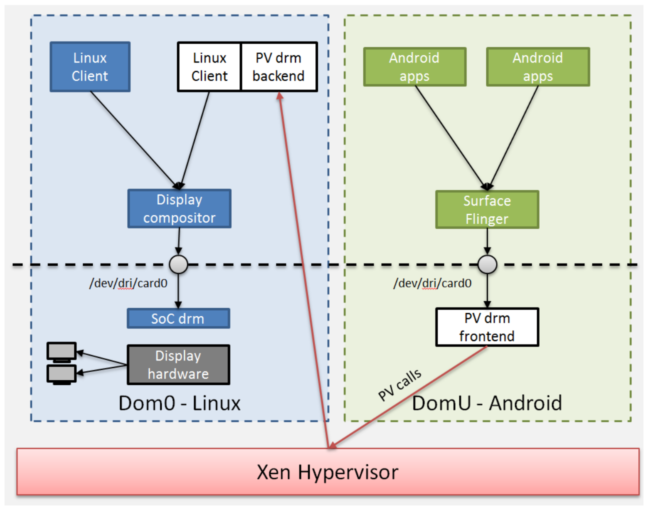

Fig. 4: Para virtualization of display in Xen

Both Linux and Android domains post content to display. To enable sharing of the display subsystem the drivers are para-virtualized. The Android driver is modified to serve as a front end. This driver sends buffers to Linux instead of posting content onto the display device. There is no change in the application as APIs called are the same. When an application wants to post content onto the display, the buffers are sent to the display driver. The display driver in-turn calls a Xen hyper-call.
On receiving the buffer from the display front-end, the hypervisor queues the buffer and raises an interrupt on Linux. When Linux domain is scheduled on the MPU, Linux receives an interrupt which is serviced by the display driver back-end. The display driver on Linux updates content onto the display. The display subsystem is para-virtualized since the hardware doesn't support virtualization. Only one system is capable of updating the hardware registers. Since the Linux system is isolated with limited application the updates to the display subsystem hardware are placed in Linux.

F. Complete virtualization of 3D GPU

To accelerate graphics rendering, both Linux and Android use the GPU. The GPU can be completely virtualized, ensuring no updates to existing drivers on both OS. Steps to achieve complete virtualization of the 3D-GPU are:

- The Memory-management-unit (MMU) for each VM is mapped to an intermediate address
- The intermediate address is translated using the second-stage MMU which is configured by the hypervisor

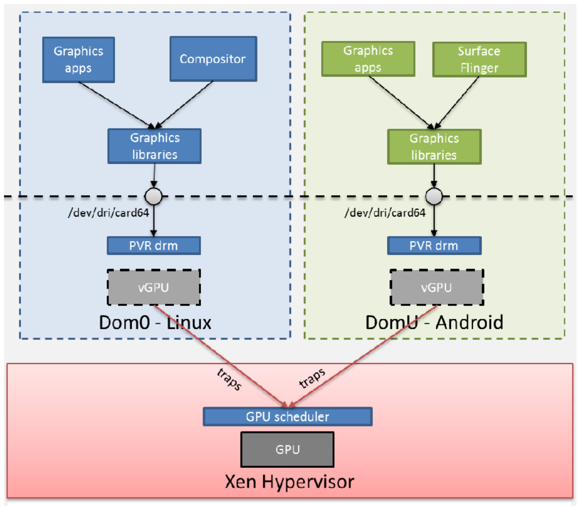

Fig. 5: Complete virtualization of GPU in Xen

- The intermediate address translates to the register space of the GPU and proceeds with the GPU address update
- If no entry exists for the intermediate address in the second-stage MMU, a fault is raised which is handled by the hypervisor
- If the access is intended to the GPU, the control returns to the guest OS mimicking memory access
- When the second-stage MMU is updated to include the intermediate address pending in the queue, the GPU registers are updated

V. RESULTS

The integrated cockpit system implements a safety-critical ADAS system, an Info-ADAS system and an infotainment system. The ADAS system is isolated, while the Info-ADAS is run within a closed system. The infotainment system is feature rich and is connected to external networks. The Info-ADAS system and Android share peripherals through virtualization. Peripherals supporting virtualization are completely virtualized while para-virtualization is adopted for peripherals that don't.

TABLE I: Boot-time and memory requirements

Application	**Boot-time (s)**	**Memory (MB)**
Surround-view	2	250
Digital Cluster	6	150
Android Automotive	30	400

Table I describes the boot-time measurements of the integrated cockpit system. The boot-time for the infotainment system is higher as Android begins initialization only after Linux kernel has finished loading. Para-virtualized drivers back-end reside on Linux which must be setup before front-end drivers are initialized on Android. Table II compares system load on each core. The MPU and GPU load is slightly higher than standalone use-cases due to the overheads associated with the hypervisor. This overhead can also be quantified using the

TABLE II: System load on DRA76x SoC

Use-case	IPU	DSP-1	DSP-2	GPU	MPU
Frequency (MHz)	212	800	800	665	1500
ADAS (%)	40	25	-	-	-
Info-ADAS(%)	40	-	-	22	15
Android (%)	-	-	38	65	38
Integrated Cockpit (%)	42	25	38	91	59

frames rendered by the GPU. Table III provides a comparison of the pixels rendered. In the integrated cockpit use-case, cluster is expected to run at 60-FPS, while the navigation application is expected to run at 30-FPS. When Android and Linux is running, the total pixels rendered decreases by 12.5%. In the event of an Android application crash the digital instrument cluster and surround-view application continue to work demonstrating the robustness of the safety-critical ADAS subsystem and achieving security of the Info-ADAS system through a closed system.

TABLE III: GPU performance across use-cases

Application	Linux (FPS)	Android (FPS)	Pixels rendered (million)
Cluster free run (1920x720)	115	-	158.9
Navigation free run (1920x1080)	-	76	157.6
Integrated cockpit	57	28	136.8

VI. DISCUSSION

A. Advantages

The proposed architecture helps achieve integration of three disparate systems into a single SoC. The key requirements pertaining to robustness, performance and safety are catered to while significantly reducing the overall cost of the system. The proposed architecture also allows for multiple HLOS to share peripherals that don't have support for virtualization at the hardware level.

B. Software complexity

Integrating the three systems into a single SoC has a software overhead. The software components modified or added to build the integrated cockpit system are:

- Firewalls to monitor unauthorized access
- Application stub on safety-critical ADAS system to service requests from QM systems
- Hypervisor integration to over-commit shared peripherals through virtualization
- Para-virtualization of shared peripherals without virtualization support in hardware

C. Limitations

- Most peripherals in the DRA76x SoC don't support virtualization. Thus, there's a software overhead to modify drivers for shared peripherals, inhibiting re-usability of code
- There's no policing of I/O access. A rogue DMA access from Android/Linux can potentially crash the safety-critical ADAS subsystem
- Re-design of board architecture may be needed due to conflicts in peripherals used such as I2C

D. Future work

An IOMMU can be integrated which manages address translation and monitors each transaction on the system, preventing un-authorized I/O accesses [14]. Guaranteeing QoS for VMs through shared policies is an enhancement to the existing architecture which will prevent starvation of resources for a VM [15]. Rebooting of the Android VM while maintaining the state of the Linux VM is another feature that can be explored. The boot-time of Android can also be reduced by suspending to RAM / disk.

ACKNOWLEDGMENT

We would like to thank GlobalLogic® for their help in integrating the Xen hypervisor to the DRA7x family of SoCs. We would also like to thank Socionext™ for providing a digital instrument cluster application.

REFERENCES

[1] PricewaterhouseCoopers, Spotlight on Automotive PwC Semiconductor Report, 2013
[2] Alan Burns and Robert I. Davis, Mixed Criticality Systems - A Review, January 2018
[3] Stefaan Sonck Thiebaut, Secure Embedded Hypervisor Based Systems for Automotive, Secure Embedded Hypervisor Based Systems for Automotive, IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, 2016
[4] Dominik Reinhardt and Gary Morgan, An Embedded Hypervisor for Safety-Relevant Automotive E/E-Systems, IEEE International Symposium on Industrial Embedded Systems, 2014
[5] Hyunwoo Joe, Dual display of virtual machines for automotive infotainment systems, IEEE Global Conference on Consumer Electronics, 2015
[6] Yogesh Marathe, Boot time optimization techniques for automotive rear view camera systems, IEEE International Conference on Consumer Ectronics, 2016
[7] Kristina Omerovic, Supporting sensor fusion in next generation android In-Vehicle infotainment units, IEEE International Conference on Consumer Electronics - Berlin, 2016
[8] International Organization for Standardization, Road vehicles - Functional safety, 2011
[9] Wolfgang Mauerer, Professional Linux Kernel Architecture, Wiley Publishing, Inc. 2008, ISBN: 978-0-470-34343-2, Chapter 1
[10] Hermann Hartig, L4 Virtualization and Beyond, 2017
[11] Cyril Clocher, Revolutionize the automotive cockpit, ti.com, 2017
[12] Vikram Appia, Surround view camera system for ADAS on TIs TDAx SoCs, ti.com, 2015
[13] S. P. Bingulac, On the compatibility of adaptive controllers (Published Conference Proceedings style), in Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory, New York, 1994, pp. 816.
[14] George Kornaros, I/O virtualization utilizing an efficient hardware system-level Memory Management Unit, International Symposium on System-on-Chip, 2014
[15] Parisa Heidari, QoS Assurance with Light Virtualization - A Survey, IEEE International Conference on Cloud Computing Technology and Science, 2016