Whitepaper

Hardware Convergence & Functional Safety: Optimal Design Methods in Today’s Automotive Digital Instrument Clusters

Version:
1.1

Released:
27-Jun-2016

Purpose:
This whitepaper discusses the convergence of digital instrument clusters, heads-up-displays, and in-vehicle infotainment systems into a single system solution and the impact developers need to be aware of to maintain functional safety.

For Support or Questions:
Name: Chris Giordano
Email: cgiordano@disti.com
Phone: +1 407 206 3390 ext 129

Name: Jim Carroll
Email: jim.carroll@mobica.com
Phone: +44 1625 446140
INTRODUCTION

The average modern high-end automobile contains about 100 million lines-of-code. That’s twice as many lines-of-code than exists in the Large Hadron Collider (McCandless, Pearl, & Miriam, 2015). From the vehicle MCU and the software controlling an array of systems, to the consumer that ranks technology features more important than car color (Autotrader, 2016), developing a solid architecture that meets safety guidelines is a significant challenge.

Consider the following statistics:

- 90% of all automotive innovations are driven by electronics and software (EETE Automotive, 2014)
- Up to 40% of a vehicle’s development costs are determined by electronics and software and that 50 - 70% of the development costs for an ECU (Electronic Control Unit) are related to software (Simon, 2010),

Coupling these two figures together shows that designing, developing, and maintaining a comprehensive software architecture is a leading cost driver.

Furthermore, as consumer electronics continue to propel automotive adoption of digital controls and touch screen user interfaces, consolidated hardware architectures become an attractive way to control costs. In addition, the effective bi-directional communication of these interfaces between driver and vehicle systems plays a critical role in safety.

This transfer of knowledge between the vehicle and the driver occurs through any of the following visual systems; in-vehicle infotainment (IVI), heads-up display (HUD), and the instrument cluster (IC). Passengers may also benefit from having access to a rear seat entertainment (RSE) system.

Most OEMs separate the management and development of these systems into separate departments. Each department develops an interface on a low-end system in order to meet their delivery deadlines. As a result, the overall vehicle solution becomes a complicated architecture of hardware and software silos (see Figure 1).
In the scope of the individual system, an architecture like this could be considered both cost efficient and manageable. However, in the scope of the entire vehicle this configuration creates a complex hardware and software architecture which is both costly to develop and maintain and extraordinarily difficult to guarantee the system works as expected.

In the past year, the industry has seen a move towards hardware consolidation which offers a unified systems architecture with one system-on-a-chip (SoC) managing multiple displays; the HUD and Cluster, the IVI and Cluster, or all three (see Figure 2).

This comes with its own set of challenges for the software architecture especially as Functional Safety (FuSA) requirements enter the picture. In this architecture, systems that usually carry FuSA requirements, such as the Cluster and HUD, may share the same SoC with systems that do not, such as the IVI.

This whitepaper discusses the needs, benefits, and recommendations for a unified systems architecture as it pertains to the system software while taking into account FuSA and the GPU load. This includes considerations for occupant safety and their demands for leading-edge technology.

**SINGLE GPU ARCHITECTURE OPTIONS OVERVIEW**

In a single GPU system, two architecture types prevail for managing the multiple display contexts; multi-layered and full display frame. This section provides a general overview on these two architectural choices.
**MULTI-LAYERED**

The multi-layered approach to display development typically consists of having several different visual software layers that become composited together in the final display. Some of these layers can be Safety Critical and some non-Safety Critical, more commonly known as mixed criticality (see Figure 3).

![Figure 3 – Depiction of a Composited Multi-layered Display](image)

The key to a successful architecture for this is software in a silo that is separately controlled either at the hardware level by separate GPU rendering or at the software level with a Compositor, such as the Weston/Wayland compositor or a reasonable facsimile. From a functional safety point of view, any piece of software added to the system should be pre-certified or at least certifiable, further complicating the software architecture of the system.

**FULL DISPLAY FRAME**

In a full display frame system, the entire display frame is rendered to a single layer. This significantly simplifies the overall architecture, but can have a limiting effect on the UI design. For example, if the UI Designer plans to add a moving map for navigation to an instrument cluster that contains safety critical content, the architecting engineer finds themselves needing to mix non-safety critical navigation content with safety critical cluster data. This inevitably leads back to a multi-layered display and a more complex architecture creating a design choice that must be taken into account at the requirements stage of the instrument cluster design.
ARCHITECTURAL CHALLENGES

Each of these architectural choices comes with a set of challenges that designers need to be aware of in order to determine the optimal user interface solution for the unified system. This section describes these challenges.

COMPOSITOR SPECIFIC CHALLENGES
When looking at a compositor-based solution, the two approaches available include software-based compositing and hardware-based compositing. Compositing takes non-safety critical and safety critical layers and composites them together in a single display frame. A great example of this would be to have an instrument cluster navigation application running on a single layer and the ASIL information and cluster visuals rendering on a separate layer.

Software compositors offer a flexible and independent solution from hardware and driver feature-based implementations. With a software compositor, developers do not need access to multiple hardware layers and do not need the ability to share driver and graphical contexts across multiple virtual spaces or guest operating systems.

In cases where hardware does not support multiple layers, or may not support enough layers to achieve the desired results, software compositing is a viable solution. However, as of the writing of this paper, no commercially available safety critical software compositors are available. Also, this may not be a certifiable solution since the SC and non-SC processes share the same address space.

The hardware would render the background layer first and the foreground layer on top of it. However, there are a limited number of layers, though some SoC types support five or more layers.

HYPERVISOR SPECIFIC CHALLENGES
A common solution currently deployed in IVI systems is based on the use of hypervisors. For each functional domain (e.g. infotainment, cluster, driver assist) the application software is running on a separate OS instance, but with all instances running on a single SoC and therefore GPU. Each OS runs in its own virtualized environment, which may be based on a hypervisor (see Figure 4) or a “container” framework such as Linux containers (LXC). This highlights the fundamental issue to be resolved; can SC and non-SC drivers run simultaneously on the same SoC?
Related solutions are also possible, such as Green Hills’ Multivisor. This allows concurrent OSs to run in their own virtual address space, allocated at boot time. This significantly simplifies the architecture of the graphics stack.

The Khronos Group specifies a safety critical version of the standard OpenGL graphics API, providing a well-understood environment for developers. This API has recently been updated to OpenGL SC 2.0, providing much richer APIs for developers by providing programmable pipeline functionality rather than the fixed pipeline available in the previous version. The SC APIs still lag the “parent” API, but the gap is rapidly diminishing. Currently the major SoC and GPU vendors and graphics driver companies are providing some level of SC driver support.

This combination of hypervisor and OpenGL SC graphics driver allows for high quality, leading edge graphics to be deployed for automotive SC subsystems. However, it is important to consider the functionality and stability of the entire system when SC and non-SC environments are combined on a single target SoC.

As a single GPU is available to accelerate graphics across all deployed OS instances, it is crucial that non-SC subsystems do not have unacceptable impact on the SC subsystems during operation. A typical example is GPU lock up. If the GPU locks up, SC content – such as driver instruments – may appear
static. Certain hardware vendors have solutions specifically for GPU lock up. SC compliant graphics drivers must be implemented in such a way that a GPU lock up would be detected and resolved within a short time period, ensuring that crucial GPU commands and responses are not lost. Such communication protocols must continue to be routed to the correct OS instance during such a restart, in order that the GPU failure has no discernible impact on the driver instruments. A positive by-product of such a GPU driver implementation is that all OS instances would benefit from the robustness of the GPU driver, whether the OS uses an SC driver or not.

SoC and GPU vendors currently typically allocate large amounts of software engineering effort to addressing such issues in pre-production for GPU drivers intended for use in non-SC contexts. GPU and SoC vendors already follow the engineering methodologies required to ensure SC compliance; the additional volume of engineering work required to develop and certify GPU drivers as SC compliant is less clear since there are a multitude of other components in the system that need to be taken into account. Pre-certified drivers are available, but it is likely that these will still require some “quality” issues to be addressed after integration. As a bare minimum, it is anticipated that support for SC APIs must be implemented or derived from existing API implementations, and that significant additional QA will be required.

The limitations of this approach derive from the management of limited hardware resources.

Multiple OSs running on a single host processor using a standard hypervisor architecture inevitably result in some performance impact; applications will run more slowly than in standalone environments, though the impact may only be significant when accessing peripheral hardware. While performance issues are eradicated using a multivisor architecture, GPU access is typically the most problematic: poorly written interrupt service routines (ISR) for other hardware device drivers (e.g. audio codecs) can have a significant impact on unrelated GPU hardware access.

Multiple OSs also imply that more software is required – this consumes more non-volatile memory (NVM). In operation, a higher RAM usage is also required. For hypervisor-based solutions, “fragmentation” of RAM results, as blocks of memory are not allocated to each OS at system boot. This is a well-understood issue, but does imply an increased bill of material (BOM) cost and careful SoC selection and configuration.

It is also important to note the low-end SoCs may not provide all of the necessary hardware support for deploying high performance virtualized OSs. However, since combining multiple systems onto one SoC will reduce overall development costs, higher performance SoCs can be used while still reducing BOM cost.

Resource sharing across OSs must be considered – if assets such as fonts, icons and images are to be made available across the OSs they must either be duplicated – which results in a waste of NVM – or stored in some shared file system – which presents an additional engineering system issue: the shared file system also needs to be SC compliant. However, if an SC-compliant RTOS is used it should already provide an SC-compliant file system.
POTENTIAL SOLUTION

Given these unified architectural challenges, this section offers a discussion of the potential solutions for user interface development. These include a cluster design where the full frame buffer is developed using a safety critical solution and compositing. The section concludes with cost factors to consider in these potential solutions.

FULL DISPLAY FRAME SAFETY CRITICAL CLUSTER

One such potential solution is to have a full display frame safety critical cluster. This option significantly simplifies the architecture by using a single operating system with a single GPU driver utilization. This option also easily allows for the sharing of UI design resources within the same software branch creating an environment to minimize or remove fragmentation. The key to this approach is to use tools that have been pre-certified in order to remove any potential risk for the program.

Such a solution has clear advantages relating to hardware contention, particularly for the GPU. GPU driver problems are notoriously complex to resolve, and as the GPU is fundamental to the operation of the entire UI, such issues are very visible and have a dramatic impact.

Building on a single OS simplifies the implementation of the GPU driver stack for the management of hardware contention, and reduces the risk of “edge case” defects being discovered in a production environment. With the reduced complexity, the majority of hardware related performance issues are also eliminated.

With the advent of SC compliant Khronos APIs and associated GPU drivers, the time to market for automotive Tier 1s and OEMs will also be reduced: it is simpler to implement an SC compliant driver for deployment in a single OS architecture than for a virtualized environment.

COMPOSITING

As discussed above, compositing is another option to consider. There are two potential compositing solutions to select from, hardware compositing and software compositing. While software compositing is a viable approach and proven in production, it also complicates the software architecture over the single display frame approach described above.

In using hardware compositors, the software architecture is not as complex and the compositing of the layers is handled in the hardware. Several hardware vendors have multiple drawing layers that can be supported and some for specific graphics languages. Hardware compositors and drivers offer a faster solution with the benefit of less software code to write, maintain, and certify.

In either case, the primary consideration is typically given to minimizing the Bill of Materials costs on the number and type of chips required on the board.
**Cost Factors**

Although the platform engineering effort for this solution is considerably simpler than for virtualized OSs, there may be additional effort required in the application domain. All applications must be available for the single target OS. Current solutions typically use multiple OSs, which implies application porting and QA effort. In some cases this can be trivial. For the majority of cases, porting between dependent components including kernel versions, APIs and OSs can have a significant impact.

The resultant implementation is likely to have improved performance, though this will not necessarily be discernable; a lower BOM cost is also likely.

**Conclusion**

When looking into system architectures for digital displays in automobiles, hardware costs must be balanced with how the interface content is built and displayed. Two leading architectures, compositing and full display frame are available to incorporate safety critical interface content into displays. The authors conclude that hardware consolidation between the HUD, Instrument Cluster, and Infotainment systems can significantly reduce the overall cost of the digital UI system development since these more powerful systems can now support redundant chips and compositing.

During this vital technological growth period, where vehicle software systems now control life critical capabilities such as braking and steering, a reliable software architecture is of equal or greater importance than cost. With autonomous systems, and the architecture the systems must communicate with, functional safety becomes significantly more important in the development of those systems. With proper architecting, and assistance from companies with expertise in safety critical development processes and systems, the automotive community can move forward with confidence while still maintaining the UI designer’s ability to handle brand awareness with an infinite toolbox of capabilities.
CITATIONS


NOTICE

ALL INFORMATION PROVIDED IN THIS WHITE PAPER, INCLUDING BUT NOT LIMITED TO COMMENTARY, OPINION, DISTI DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” THE AUTHORS MAKE NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, the authors assume no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of the authors. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. The DISTI Corporation’s products are authorized for use as critical components in life support devices or systems only with the use of the GL Studio Safety Critical runtime libraries and certification kit available from the DISTI Corporation.

Trademarks
DISTI, the DISTI logo, GL Studio, GL Studio SC are trademarks or registered trademarks of the DISTI Corporation in the United States and other countries. MOBICA and the MOBICA logo are trademarks or registered trademarks of MOBICA LTD in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright © 2016 The DISTI Corporation and MOBICA LTD. All rights reserved.