Mixed Safety Criticality Levels for Graphics Rendering

Introduction

The use of multicore, hypervisor and multiple independent partitions in modern embedded systems allows a new generation of display technologies to use one multifunctional hardware system to perform multiple independent tasks. Running multiple tasks on the same multifunctional hardware saves the application size, weight, and power, as well as reduces overall project cost. However, depending on the functional requirements and safety classification needed, an application may require a variety of safety certification levels running on the same graphics system. For example, one avionics partition may need to adhere to FAA and EASA Design Assurance Levels (DAL) A, while another partition may be required to support DAL D. Similarly, an ISO 26262 automotive application may require Automotive Safety Integrity Level (ASIL) D on one partition, and QM on another. While this poses less of a challenge on CPUs where execution can be fully pre-empted, GPUs typically are non pre-emptive. That is, it is generally not feasible to halt or interrupt rendering operations once they have been submitted to the GPU. With this restriction, how can integrators ensure that the lower level partition does not impact the higher level partition, and that all applications run properly to their designated DAL/ASIL?

This white paper discusses six different mixed safety criticality scenarios for graphics rendering in embedded systems, their pros and cons, and use case considerations.

Scenario #1:

All graphics rendered using a software renderer (i.e. no GPU hardware acceleration)

A software renderer allows graphics to be rendered entirely by the CPU without requiring any graphics hardware. While the CPU is preemptable, it does not provide the same level of graphics performance and is therefore not ideal for heavy graphics workloads. Although GPU hardware acceleration (discussed in Scenario #6) provides better rendering performance and does not consume CPU resources, using a software renderer allows the lower DAL rendering process to be interrupted and allows the GPU software emulation engine to completely switch over to the higher DAL application. The software rendered is typically OpenGL® SC 1.0 which uses a fixed function pipeline, as emulating shaders in a programmable pipeline (such as OpenGL SC 2.0) would be costly.

Besides the lack of performance, one drawback is that software rendering in safety critical applications still requires a display controller, which is typically a GPU display controller or, alternately, a display controller IP block in the FPGA. The GPU adds to overhead in terms of size, cost, power (for example, leakage current). This still requires BIT to check that the register configuration is not affected by a single event upset, and that there is no resulting impact to the memory interface and display controller settings that are critical to proper operation.

Use case: Using multiple software renderers for mixed safety criticality levels is a solution well suited for rendering that does not need a high frame rate such as full text displays (e.g., multi-purpose control and display unit, maintainer data, and even waypoint lists) and simple rendering such as graphical engine data.

Scenario #2:

Highest level application uses GPU hardware acceleration and lower level applications use a software renderer

GPU hardware acceleration and software rendering may be used in tandem to produce a solution with combined benefits. This configuration is easily certifiable and is best used when the highest safety criticality application has a heavy graphics workload. The application(s) requiring the higher criticality level can use the GPU to its fullest extent because there are no less critical applications using it. For example, in this scenario, if a DAL A application is the only application accessing the GPU, and a lower DAL C application has data to display, the DAL A application can command the GPU to copy and display the image the DAL C application created. In this case there is only one transfer of data required from the software renderer to the GPU per frame. A compositor, which can render windows from multiple applications onto a display and is certifiable up to DAL A, can also be used to take in low level rendered images to the GPU. For more information on the compositor, please read our white paper "A Safety Critical Compositor for OpenGL SC 1.0.1 and OpenGL SC 2.0".

In a multi-core environment, a lower criticality level application can run on a separate core and have no interference with higher level applications. This scenario also supports multi-rate rendering – for example, when a lower criticality level application needs to produce a PDF page every second, and a higher level application might need to run at 60 Hz.

This scenario does not come without processing drawbacks. The more lower level applications that are needed in a system, the more CPU processing power is required to render everything. Software rendering is slower than GPU rendering, and so there is a processing trade off for the lower safety criticality level applications.

Use Case: This scenario can be used for an ARINC 661 server that needs to draw 2D graphics or video on the display and where certain pages must also be slowly updated with text display data. An ARINC 661 server qualified to DAL A can be used to draw the Primary Flight Display, ADI ball and navigation information. At the same time, a lower level PDF reader used to display airport approach procedures in the same system needs to read a PDF page and write it to the screen. Certifying the PDF reader can be avoided by using the software renderer to create an image of PDF page. The renderer can pass the page as a texture to the GPU, which then displays that page using the ARINC 661 server that controls where specifically on the screen the PDF should appear. The only item being passed from a low level to high level application is a bitmap, and to ensure proper operation, the higher level application is capable of shutting down the lower level application if it misbehaves. Not only that, the higher level application can detect if it's not properly receiving the data, and the operating system can shut down if need be.

Figure 1: Primary Flight Display of a Boeing 777

Scenario #3:

Highest level application uses a software renderer and lower level applications uses GPU hardware acceleration

In this scenario, which is opposite to the one discussed above, the highest criticality level applications use a software renderer while the lower criticality level applications use GPU hardware acceleration. This scenario allows for less certification efforts for newer technologies such as Synthetic Vision Systems (SVS) for Degraded Visual Environments (DVE) while still maintaining the high assurance levels needed for Primary Flight Displays. For example, the Primary Flight Display's SVS 3D layer is at a lower DAL level and has the ability to shut down the SVS in case it misbehaves. This allows for a mixed DAL situation without shutting down higher DAL applications.

Some considerations should be made for this scenario. When using a software renderer for the higher level applications, the CPU usage is more than if the system were using multiple GPUs to accomplish the same task. The same potential also exists for keeping the GPU busy as identified in scenario #6. However in this case only the lower level application is impacted. While display controllers generally have good separation from the GPU in most SoC and discrete graphics solutions, candidate solutions must be evaluated for any interference channels that may exist between the GPU and the display controller. This ensures that if a lower level application overloads or otherwise corrupts the GPU, the display controller is always accessible to the highest level application.

Use case: In this scenario, a Primary Flight Display can be rendered by the software renderer. The ADI ball would run at 20 Hz in the front, while the SVS 3D display (which is best rendered by the GPU due to its large amount of data) would run in the background. The GPU hands a text image every frame to the software renderer. The software renderer then takes that image, sets it to the background behind the ADI ball, and puts it into the frame buffer on the GPU.

Scenario #4:

Upgrade all lower level applications to be equivalent of the highest application

If lower level applications are not overly complex, this scenario may be a viable option. As all applications in this scenario have been certified to the equivalent criticality level of the highest application, all applications are now safer to use, they can all use a GPU without any negative impact on each other, and the system can operate properly with a GPU or a software renderer – it does not require both. No software library is needed *or* no GPU is needed. If there is a GPU, all applications are able to run at a faster rate and performance is increased.

The main downside to this scenario is the need to certify all applications to a higher level. This will prove time consuming and costly for the integrator, especially when a higher level of certification is not required for all applications.

Use case: This scenario can technically be used on any system; however, due to the added cost and effort, it's not typically implemented unless the lower safety applications are small.

Scenario #5:

Architect the system(s) to eliminate mixed safety criticality graphics

One way to effectively accomplish this scenario is to increase the number of GPUs. Each GPU can be assigned its own a safety criticality level (for example, one for DAL A, one for DAL B, etc.) and then applications are assigned to their GPUs as per their required levels. When mixed on a single display, the highest criticality level GPU takes responsibility for driving the displays and the other GPUs send their outputs to it for display. Each individual display is owned by the highest application(s) that needs to display on it. For example, if the highest criticality level you need on one particular display is ASIL C, then the level C application owns the particular display.

Benefits to this scenario include a high throughput rate and maximum ability to compute, as all applications can use a GPU to render graphics. The CPU workload is reduced due to GPU offload, and the integrator has the ability to segment different safety criticality levels onto specific GPUs. In addition, now that many CPUs have integrated graphics, adding an additional GPU externally provides extra capabilities on top of the integrated graphics already available, which boosts performance power, allows for easy separation, and reduces memory transfer. The integrated graphics drive the output, so copying data back into the GPU is no longer required. For example, an SVS can run on the GPU, while the higher DAL applications can run on the integrated graphics – this increases performance over using a software renderer and reduces CPU usage. Dissimilar GPUs (for example, Intel integrated graphics and an AMD GPU) may also be used together to validate and detect Hazardously Misleading Information (HMI) per the CAST-29 paper.

It's important to note that multiple GPUs in the same System on a Chip (SoC) can be used in this scenario (for example, the multiple GPUs inside NXP's i.MX 8). Since the GPUs are contained within one SoC, there are no additional SWaP concerns.

Scenario #5 also has several drawbacks. With multiple GPUs, more hardware is needed in the system, which increases size, weight and power (SWaP) and requires higher integration effort due to the need to move output video between GPUs. Combining integrated graphics with a discreet GPU (for example, Intel Core™ i7 and AMD's E8860 GPU) also requires some platform-specific architecting to arrange support from the driver and configure the discreet GPU's output to flow into the integrated graphics.

Use Case: In the case of an SVS, all applications required may use a GPU, which results in faster performance. This allows the SVS the ability to fully utilize the existing hardware and gives integrators the option of adding additional GPUs as needed.

Scenario #6

All graphics rendered using GPU hardware acceleration

Hardware acceleration offloads the CPU by assigning certain processes to other hardware such as the GPU. This allows the CPU to focus more effectively on other tasks. A GPU is optimized for graphics, has a dedicated memory, and can process data in parallel. Although mixed safety criticality levels using a GPU can only exist together within one designated level of each other (for example, mixed applications of the same DAL level), one GPU may be used for all hardware acceleration in the right circumstances, eliminating the need for multiple GPUs, and thus, extra hardware. GPU hardware acceleration is optimal for heavy graphics workloads needing high performance processing, as it results in a higher frame rate, which typically produces a seamless render.

In real-time embedded safety critical applications, when it comes to mixed DAL levels this type of rendering has some major drawbacks. For example, using the GPU for hardware acceleration runs the risk of a lower DAL application keeping the GPU busy by operating on large amounts of graphics data or simply overloading the GPU command buffer, which could prevent or delay the more critical (DAL A/ ASIL D) applications from having enough GPU resources to operate properly. This can lead to hazardously misleading information because the data is stale. Typical GPU hardware acceleration also does not provide a mechanism for getting control of the GPU back once it has been provided with commands (i.e. there is no pre-emption like on CPUs). GPU hardware acceleration is also not an easy condition to detect in a timely manner. However, CoreAVI has developed a solution to this problem. Contact CoreAVI for a copy a white paper describing the solution to full GPU acceleration of mixed DAL applications.

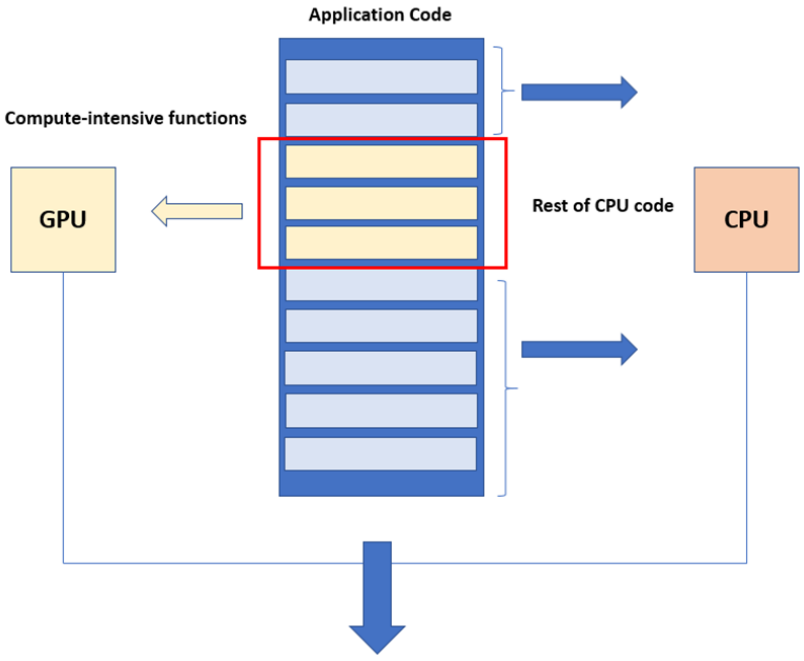

Figure 2: GPU Acceleration

Use case: GPU hardware acceleration can be used in a standard cockpit display system with a Primary Flight Display and navigational system where everything is at a higher DAL. This configuration allows the Primary Flight Display to be split up into multiple applications of multiple DAL levels, so long as the DAL levels are within one designated level of each other.

Find Out More

CoreAVI supports embedded graphics applications with GPUs supported by CoreAVI's fully featured safety certifiable OpenGL drivers.

More information about CoreAVI's ArgusCore™ OpenGL drivers, with extensions, along with a comparison of driver features can found here: CoreAVI safety certifiable graphics drivers.

Contact CoreAVI to find out what we are working on and to discuss your demonstration/evaluation requirements: Sales@CoreAVI.com.

Author

Mary Beth Barrans

Director of Marketing

Mary Beth Barrans joined CoreAVI in 2017. As the Director of Marketing, she is responsible for the product positioning and customer-focused power messaging for CoreAVI's safety certifiable graphics and hardware IP product lines, as well as strategic partnerships and events. She previously worked for Curtiss-Wright Defense Solutions as a Senior Marketing Product Specialist. Mary Beth has a Bachelor of Social Sciences, a Bachelor of Education, a Masters of Arts, and a Technical Writing designation.