Interface Strategies for IoT

From the Lightweight to the Advanced

By Stephen Olsen, Product Line Manager, VxWorks

WHEN IT MATTERS, IT RUNS ON WIND RIVER
EXECUTIVE SUMMARY

The design requirements for graphical user interfaces (GUIs) used in embedded applications vary dramatically. With the proliferation of smart devices being ushered in by the Internet of Things (IoT), that variation has begun to increase to an unprecedented degree. Established platforms—such as VxWorks®, the world’s most widely deployed real-time operating system (RTOS)—embrace the full range of that complexity. Depending on the use case itself, as well as the design preferences of solution providers, very different approaches may be taken to create a simple yet robust way for users to interact with systems.

Some implementations are best served by simple GUIs that can be supported with minimal system resources, such as a small-footprint system-on-chip, using software rendering to a framebuffer. At the other end of the spectrum, some applications require feature-rich GUIs with advanced 3D graphics, shading, animation, and image transforms; these typically use hardware acceleration on dedicated graphics processors.

This paper examines both of these approaches to rendering GUIs for embedded applications, including in IoT.

TABLE OF CONTENTS

Executive Summary ................................................................. 2
System Considerations for GUIs in Embedded Applications .................. 3
Higher-End GUIs Rendered on the GPU ....................................... 4
  UI Acceleration on the GPU with Qt for VxWorks ......................... 4
  Software Tuning with Wind River System Viewer .......................... 5
  Representative Target Hardware for GPU-Rendered UIs: NXP i.MX 6 Series ..... 5
Sophisticated, Lightweight Interfaces Residing in the Framebuffer .......... 6
  Software Development with Wind River Tilcon Graphics Suite ................. 6
  Typical Framebuffer UI Target Hardware: NXP Vybrid VF6xx ............... 7
Capabilities for Safety-Critical Systems ...................................... 8
Conclusion .............................................................................. 8
UI COMPLEXITY MUST MATCH SITUATIONAL CONTEXT
Providing the most information possible in a user interface (UI) is not always the best course. In a human–machine interaction study with a defibrillator device, person-on-the-street users were intimidated by an overly complex interface. In fact, users would often hesitate to use the device, even in an emergency situation.

When the UI was stripped down to its most basic elements, people were more willing to use the device, potentially saving lives in the process.

SYSTEM CONSIDERATIONS FOR GUIS IN EMBEDDED APPLICATIONS
Compared to general-purpose computing environments, relatively simple GUIs may be indicated for use with embedded systems. This is particularly true in usages where significant constraints exist in factors such as processing and memory resources, energy consumption, heat, and cost. Some characteristics of general-purpose platforms, in comparison to embedded systems, are summarized in Table 1.

Table 1. Characteristics of General-Purpose Versus Embedded Systems

<table>
<thead>
<tr>
<th></th>
<th>General-Purpose Systems</th>
<th>Embedded Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>• 32-bit or 64-bit</td>
<td>• 32-bit</td>
</tr>
<tr>
<td></td>
<td>• 1–12 cores with Hyper-Threading</td>
<td>• 1–4 cores</td>
</tr>
<tr>
<td></td>
<td>• 1–3 GHz</td>
<td>• 100 MHz–1.5 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>• 8 GB or more</td>
<td>• Typically &lt;4 MB (potentially far more)</td>
</tr>
<tr>
<td>Storage</td>
<td>• Large hard drive(s)</td>
<td>• Possibly Flash SD card</td>
</tr>
<tr>
<td>Graphics</td>
<td>• Dedicated GPU</td>
<td>• Framebuffer or GPU</td>
</tr>
<tr>
<td></td>
<td>• OpenGL</td>
<td>• OpenGL ES</td>
</tr>
<tr>
<td>Implementation</td>
<td>• Large number of applications</td>
<td>• Dedicated applications</td>
</tr>
</tbody>
</table>

While embedded systems can have robust resources available, it is more common for them to be rather limited, compared to general-purpose systems. For example, a general-purpose system typically has more CPU resources available than an embedded system, with a substantial number of identical, robust cores that can be applied to computational work as needed. Embedded systems, by contrast, generally have more modest collections of cores, and different types of cores may be available for various roles. For instance, depending on its complexity, a GUI may be tied to execution on a specific CPU core, reserving the most robust core either for itself or for other work.

Embedded systems often have stringent memory constraints. It is common to see embedded systems with several orders of magnitude less memory than their general-purpose counterparts. Furthermore, while general-purpose systems can roll over to hard-drive storage, providing essentially unlimited memory space, embedded systems often have no local storage at all, or only a limited amount through add-in resources such as Flash SD cards.

While general-purpose systems tend to include specialized GPU hardware for graphics offload, supporting sophisticated graphical elements for GUIs, embedded systems may rely on simpler, framebuffer-generated graphical experiences. In other cases, embedded systems may be provisioned with GPU graphics hardware. The dedicated applications implemented by embedded systems may require very small or very large levels of resources, and as a reflection of that variation, the complexity of a given application’s GUI may also vary dramatically.
HIGHER-END GUIS RENDERED ON THE GPU

Complex GUIs may draw on a combination of framebuffer and GUI rendering, as illustrated in Figure 1. Here, software-based Tilcon functionality (shown on the left side of the figure and discussed in detail below) is combined with hardware-assisted graphics (shown on the right). Together, these approaches enable balance between lightweight system requirements and advanced user experiences.

The OpenGL/OpenGL ES driver on the right side of the figure adds capabilities for 3D graphics, and it is particularly well suited to 3D object manipulation in hardware. It interprets commands from the user application in both software and hardware to render both 2D and 3D objects as elements of the GUI display, using the GPU. The platform provides a shader language that allows for downloading different shaders for use with the GPU. However, because these shaders target and are optimized for the specific software driver, they are also GPU-specific, limiting portability.

OpenGL and OpenGL ES are open standards, which users can download from khronos.org. They can then benefit from participation in large user communities.

UI Acceleration on the GPU with Qt for VxWorks

The Qt Company and Wind River® have collaborated to integrate Qt’s UI framework into VxWorks. This co-engineering effort has resulted in the combination of robust, simplified UI capabilities that are essentially native to the embedded OS. User interactions at the application level are streamlined, using hardware acceleration to provide a smooth experience, even in demanding circumstances, including environments that require real-time, deterministic behavior, on a wide variety of hardware.

Using the combination of the Qt framework and VxWorks, applications can readily meet application needs for battle-tested predictability and reliability, while also delivering outstanding graphical user experiences.

The collaboration between Qt and Wind River provides even greater UI capability for applications running on VxWorks. As shown in Figure 2, the Qt framework can take the place of Tilcon in the GPU-accelerated architecture for UIs discussed above, providing a more robust environment. Qt also interfaces directly with evdev and OpenGL/OpenGL ES, and it is optimized for use with a variety of platforms, including Linux, which sets the stage for easy porting and incorporation of third-party tools if called for in the future. A particular advantage of the Qt solution is that it uses both the framebuffer and the GPU hardware, providing greater accuracy in rendering.

Figure 1. GPU-accelerated UI architecture

Figure 2. Qt solution for GPU-accelerated UIs
Joint development between Wind River and Qt leading up to the introduction of Qt 5.5 for VxWorks provides a smooth user experience and optimized workflow for VxWorks, with tools from the two companies working together. Qt provides a mature set of widgets to streamline development, augmented by its QML language for specifying and programming UIs. QML greatly simplifies the process for developers to specify the behavior of animations. This language is interpreted at run time, taking advantage of greater CPU power and allowing for dramatic improvements in the feature richness of UIs.

The Qt and VxWorks integration automatically implements hardware acceleration of the graphical elements of the UI, using the GPU. The visual user experience benefits through smoother transitions and the ability to handle complex visualizations without bogging down. At the same time, offloading graphics processing from the CPU helps make its processing resources available for other work.

Qt can also be run in the more constrained Tilcon environment while still delivering dividends in terms of UI richness. Qt for VxWorks is also flexible in terms of allowing customers to strip out unused elements, reducing the overall footprint.

The Qt engine is available in a limited version without some tools under the open source GPL license. A commercial (non-GPL) version is provided in conjunction with VxWorks.

**Software Tuning with Wind River System Viewer**

Wind River System Viewer is a run-time analysis tool for identifying and resolving application-execution problems, as well as improving system performance. It ships as part of Wind River Workbench, a collection of tools based on Eclipse for software design, development, debugging, test, and management. To reveal interactions among tasks, threads, interrupts, and system objects during application execution on a target, System Viewer offers detailed analysis and visualization, including graphical views as shown in Figure 3.

Based on the performance profiling provided by System Viewer, optimization of UIs and other application code is dramatically simplified and improved. Using the tool’s guidance about which routines are consuming the most processing and memory resources, developers can pinpoint inefficiencies and improve overall code quality. Taking better advantage of the available resources in constrained embedded systems allows for added features and functionality instead of wasted overhead.

**Representative Target Hardware for GPU-Rendered UIs: NXP i.MX 6 Series**

Higher-end GUIs for embedded applications require more hardware resources. The NXP i.MX 6 Series application processors offer significant enhancements in terms of CPU capability compared to the Vybrid VF6xx, as well as a sophisticated, built-in GPU, as summarized in Table 2. The platform is available in a range of SKUs, to meet varying needs. Single-, dual-, or quad-core versions are available, based on Cortex A9 or A7 cores, as well as options with a combination of Cortex A9 and M4 cores.

**Table 2. High-Level Specifications of the NXP i.MX 6 Series**

<table>
<thead>
<tr>
<th>CPU (max)</th>
<th>Memory (max)</th>
<th>Graphics (max)</th>
</tr>
</thead>
</table>
| Up to ARM Cortex A9 @ 1.2 GHz  
– Up to 1 MB L2 cache | 64-bit DDR3  
Two-channel 32-bit LPDDR2 | Vivante GC320 GPU  
– Built-in, hardware-accelerated graphics  
– 3D graphics with up to four shaders  
– Up to two 2D graphics engines |
Most of the offerings in the i.MX6 family feature capabilities for both 2D and 3D graphics. Particularly, using VxWorks on the higher end of this series of parts, customers can deliver sophisticated UIs that utilize features such as shaders, robust animations, and image transforms. Typical usages include automotive and in-flight infotainment, point-of-sale devices, and client devices such as tablets and smartbooks.

SOPHISTICATED, LIGHTWEIGHT INTERFACES RESIDING IN THE FRAMEBUFFER

A framebuffer is a simple graphics mechanism that stores a bitmap in a dedicated portion of RAM and uses it to create a UI display. The color values of each pixel can be stored using schemas that range from one bit each (monochrome) to 24 bits each (true color), plus an optional alpha channel for transparency information. All rendering is handled in software.

This model can be usefully extended using multiple framebuffers, with a concept known as double buffering. This approach allows setting an interrupt each time the page is redrawn, switching to a new framebuffer, and allowing the initial frame to be recalculated in the background. By switching back and forth in this manner (which can also allow more than two frames if desired), the visual effect of recalculating the frame is hidden, providing for smooth transitions between frames.

Moreover, modern hardware can support large enough numbers of framebuffers to support other capabilities—the NXP Vybrid VF6xx architecture discussed below, for example, can support as many as seven framebuffers overlaid on top of each other. This provides excellent support for elements such as alert or dialog boxes to be displayed within the display window but independent of the main UI.

Software Development with Wind River Tilcon Graphics Suite

For the development of rich UIs on embedded hardware that ranges from the simple to the complex, Wind River offers the Tilcon Graphics Suite, which is included with VxWorks. Developers can use the suite’s Wind River Tilcon Interface Development tool to create sophisticated UIs that use the framebuffer exclusively. Examples of such UIs are shown in Figure 4.

Tilcon is included with multiple profiles that come out-of-the-box for licensed VxWorks users to support the specific needs of various industry verticals. These include purpose-built capabilities for the following usages:

- **Aerospace Profile**: Boosts safety, security, connectivity, manageability, user interface, and graphics performance for aerospace and defense equipment makers
- **Industrial Profile**: Enhances security, connectivity, manageability, UI and graphics, and safety capabilities for intelligent, connected industrial devices
- **Medical Profile**: Includes a mapping guide to aid in creating and certifying medical devices, as well as middleware for security, connectivity, manageability, UI and graphics, and safety
- **Consumer Profile**: Provides differentiated, intelligent consumer devices with added capabilities for security, manageability, and UI and graphics
For each of these profiles, Tilcon offers a range of built-in, customizable widgets such as gauges, meters, and charts that can be incorporated into UIs and augmented as needed with custom artwork that can be imported from external locations. An important aspect of Tilcon’s low-overhead operation is that the UI-rendering pipeline does not require any code to be written. Instead, the Tilcon Interface Development Tool captures the design as a platform-independent resource file. That resource file is loaded on the Tilcon GUI Engine, as shown in Figure 5, which dynamically renders the UI on the target device, without recompilation or transcoding.

On the left side of Figure 5 is the Tilcon Interface Development Tool, which runs on the host and enables developers to actually create the UI itself. This what-you-see-is-what-you-get (WYSIWYG) environment, which can be used in either Linux or Microsoft Windows, allows sophisticated capabilities such as shading. These capabilities allow preprocessing to a bitmap, for a resource-efficient approach to providing attractive graphic UI elements. The tool also allows running within a simulated environment using Wind River VxWorks Simulator, enhancing the flexibility of the development process.

Everything on the right side of Figure 5 runs on the target. While this model can render relatively sophisticated UIs such as those shown in Figure 4, it is also somewhat limited in terms of the elements it uses, which include 2D graphics, fonts, and still images. A key advantage of UIs based on this model is that they use memory sparingly, making them suitable for usages on small-footprint devices while still being capable of rendering fairly sophisticated UIs.

**Output Through the Fbdev Handler**

The Tilcon Interface Development Tool packages all of the graphical and command elements needed for the GUI and supplies them to the Tilcon GUI Engine, which decodes that information and supplies the result to the framebuffer by way of the fbdev handler—which functions as an abstraction layer—either directly or through OpenVG. Alternatively, the user application can communicate directly with OpenVG. In either case, the framebuffer handles the actual rendering of the UI.

**Input Through the Evdev Handler**

On the input side, the evdev handler abstracts and manages connections to mouse, keyboard, touchscreen, and similar devices and passes those inputs to the Tilcon GUI Engine, where event responses are managed. Where needed, evdev has the ability to handle multi-touch as well as both capacitive and resistive touchscreens. Events based on specific keypresses can be interpreted in various ways as needed—a press of the “a” key can be interpreted as that letter in a text field, for example, or as an assigned command in a different context.

This model is highly extensible, with abstraction allowing a non-standard set of buttons or other physical controls, for example, to be assigned to whatever role is needed. It is also similar to native capabilities within Linux, which helps minimize porting effort to support the use of third-party UIs.

**Rendering with VG lib (OpenVG)**

VG lib is a software-based 2D rendering engine for vector graphics that abides by the OpenVG standard and is compatible with OpenVG APIs. Because of the nature of vector graphics, images rendered for UIs based on this approach can be viewed at any zoom level without loss of quality due to pixilation. That factor allows for high-resolution graphics to be handled with relatively low memory requirements.

**Typical Framebuffer UI Target Hardware: NXP Vybrid VF6xx**

As mentioned previously, the model of rendering the UI using the framebuffer is scalable for use on hardware that ranges from low-end systems to powerful ones with GPUs. However, a typical example of the hardware where the framebuffer-only approach may be implemented is the Vybrid VF6xx family, as described in Table 3. A few applications for which this device is often used with GUIs include kiosks with 2D displays, portable medical equipment such as infusion pumps and respirators, and monitors used in industrial applications.
The Vybrid VF6xx provides two heterogeneous processing cores: a Cortex A5 and a Cortex M4. Depending on how processing-intensive the GUI is, it can be run on either of those cores. The device also includes two separate and identical LCD display controllers, which allows the system to run two simultaneous instances of Tilcon, if desired. A video interface unit is also included that allows for image and vision capture.

Table 3. High-Level Specifications of the NXP Vybrid VF6xx

| CPU Cores | • ARM Cortex A5 up to 500 MHz  
| L1/L2 cache (L2 optional)  
| ARM Cortex M4 up to 167 MHz  
| L1 cache |
| Memory | • Up to 1.5 MB of on-chip SRAM |
| Display Controllers | • Support for up to two SVGA (1024x768) displays |

CAPABILITIES FOR SAFETY-CRITICAL SYSTEMS

Safety-critical applications—loosely defined as those for which failure could or would cause harm to people—are a traditional key area of focus for Wind River. In general, system operation is partitioned such that failures are isolated, preventing impacts on critical sections of the system. Wind River VxWorks Cert Platform, for example, is purpose-built to facilitate certification of applications to the stringent requirements of the international aerospace industry.

From a UI perspective, Wind River support for safety-critical usages draws on the core reliability of VxWorks, augmented by a number of industry partnerships. Core Avionics & Industrial Inc. (CoreAVI), for example, has developed a number of OpenGL drivers from the ground up for safety-critical implementations. This code lends itself to certification across industry requirements. The SCADE Suite of products from Esterel Technologies, now a part of ANSYS, generates graphics and code for safety-critical UIs.

Across hardware platforms, Wind River innovates with industry partners to support safety-critical applications in addition to more general-purpose embedded usages. For further information, contact Wind River at www.windriver.com/company/contact/request.

CONCLUSION

VxWorks is ideally suited for supporting UIs from the simple to the complex, on embedded systems that include simple framebuffers to complex GPUs. It supports usages that range from lightweight systems with rudimentary or sophisticated UI needs, to more robust UIs that include 3D animations and other full-featured capabilities. Moreover, VxWorks supports these needs without hindering its unrivaled deterministic performance. Multi-feature execution happens simultaneously, seamlessly, and with ease of use. This full-range offering optimally positions solution providers to differentiate their solutions in the increasingly crowded marketplace, including IoT.

Learn more about VxWorks at www.vxworks.com.
Existing VxWorks 7 customers can enjoy an evaluation of Qt for VxWorks today: www.qt.io/contact-us-vxworks