Technical White Paper

Choosing the Right Processor and Graphical User Interface (GUI)

Shriya Surti, Soobin Kim, Tobias Morell (KDAB)

Abstract

Selecting the right embedded user interface (UI) framework is both a technical and strategic decision. This white paper provides a platform-oriented selection guide mapping leading UI frameworks across the Texas Instruments Sitara™ processor family. The guide categorizes Sitara devices into multiple tiers, evaluates each framework, and addresses implementation best practices. By understanding how hardware constraints and UI frameworks interact, developers can make informed selections that optimize cost, performance, and time-to-market for their specific product requirements. Before diving in further, it is worth noting that there is no single "best" processor or GUI framework; the right choice depends on each project's requirements, team expertise, and licensing needs, and should be revisited as hardware capabilities and ecosystems evolve.

Table of Contents

1 Introduction 2
2 Understanding the Hardware Landscape and UI Implications 2
2.1 Hardware Platform Categories: The Baseline 2
2.2 Featured Platform Specifications & Architecture Context 3
3 Technology Deep Dive: The Frameworks 4
3.1 Qt: Cross-Platform Application Framework 4
3.2 Slint: The Modern Lightweight Alternative 4
3.3 LVGL: The Microcontroller Specialist 5
3.4 Flutter: The Mobile-First Approach 6
3.5 Other Notable Frameworks 7
4 The KDAB Decision Framework and Implementation Best Practices 7
4.1 Decision Criteria Beyond the Silicon 7
4.2 Implementation Best Practices: Optimizing Performance 8
4.3 Start-Up Time Considerations (The "Cold Boot" Problem) 8
4.4 Power Efficiency (Battery-Powered Devices) 8
4.5 Future Trends and Considerations 8
5 Summary: TI and KDAB's Perspective 9
5.1 About the KDAB Group 9
5.2 About the Texas Instruments Sitara MPU Group 9
6 References 9

Trademarks

Sitara™ is a trademark of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

Copyright © 2026 Texas Instruments Incorporated

1 Introduction

Selecting the right user interface (UI) technology for embedded systems is no longer a mere software engineering decision, but also a strategic business choice. End-users expect 60fps animations, high-resolution displays, and intuitive touch interactions regardless of the end application.

Achieving this level of performance requires careful consideration of hardware capabilities, performance goals, development resources, and long-term maintenance. Over-engineering the hardware to support a heavy UI framework inflates the Bill of Materials (BOM) budget. Conversely, selecting a rudimentary UI framework for capable hardware leaves GPU resources unutilized and extends time-to-market due to poor developer tooling.

This guide provides a platform-oriented overview of the leading embedded UI technologies and maps these options across the hardware spectrum of the TI Sitara family.

2 Understanding the Hardware Landscape and UI Implications

Before evaluating software frameworks, it is important to understand the hardware foundation. The embedded hardware spectrum is vast, and the boundaries between microcontrollers (MCUs) and microprocessors (MPUs) are increasingly blurring. Hardware platforms still fall into distinct categories, each imposing constraints that materially shape UI technology selection.

2.1 Hardware Platform Categories: The Baseline

Table 2-1. TI Sitara Processors HMI Tiers

TI Sitara HMI Tier	Devices	CPU	3D GPU / Video	NPU	Display	UI Constraints
High tier	AM67A	Up to Quad Cortex-A53	3D GPU up to 50 GFLOPS and video codec	Up to 4 TOPS	Triple display support	Capable of rich multi-display UIs with realistic bandwidth expectations; AI integration and a higher power budget can be available depending on device.
	AM67			N/A		
	AM62P			N/A		
Mid tier	AM62	Up to Quad Cortex-A53	3D GPU up to 8 GFLOPS	N/A	Dual display support	Good CPU/RAM headroom with entry-level 3D hardware acceleration. AI integration can be available depending on device.
	AM62A		N/A	Up to 2 TOPS	Single display support	
Low tier	AM62L	Up to Dual Cortex-A53	N/A	N/A	Single display support	Deployed in highly cost-sensitive products where software efficiency is essential to keep the BOM low.
	AM335	Single Cortex-A8	3D GPU up to 1.6 GFLOPS	N/A	Single display support	

Note

Suitability depends on the actual memory configuration, graphics driver, Linux/RTOS stack, display topology, application complexity and measured behavior on target hardware. This table is a selection guide, not a support or performance guarantee.

Copyright © 2026 Texas Instruments Incorporated

2.2 Featured Platform Specifications & Architecture Context

To contextualize how UI frameworks behave in the real world, it helps to examine the specific architectures of the most prevalent embedded silicon of today.

Note

Many of these SoCs are rarely deployed as custom chip-down designs from day one. Instead, they reach products through robust system-on-module (SoM) ecosystems from industry leaders. SoM partners accelerate industrial time-to-market and simplify hardware certification for modern Internet of Things (IoT) devices.

2.2.1 Texas Instruments AM335 Platform

- **Architecture:** Single-core 1GHz Cortex-A8 with NEON SIMD. All application code runs on the same core, so the UI and any real-time processing share resources; there is no dedicated real-time core.
- **Graphics and Display:** Imagination SGX530 3D graphics and a 24-bit LCD controller, supporting a single RGB display up to 1920 × 1080 at 30fps.

2.2.2 Texas Instruments AM62L Platform

- **Architecture:** Dual-core 64-bit Cortex-A53 up to 1.25GHz, providing enough processing power for a Linux UI while keeping the system simple with low-cost and low power capabilities.
- **Graphics and Display:** Single display support over DSI or DPI up to 1920 × 1080 at 60fps.

2.2.3 Texas Instruments AM62A Platform

- **Architecture:** Quad-core Cortex-A53 up to 1.4GHz, two Cortex-R5F cores at 800MHz for MCU-channel and device-management work, and a C7x deep-learning accelerator up to 2 TOPS. The R5F cores handle deterministic, low-latency tasks while the A53 cluster runs Linux and the graphical UI.
- **Graphics and Display:** Single display support over DPI up to 2048 × 1080 at 60fps.

2.2.4 Texas Instruments AM62 Platform

- **Architecture:** Quad Cortex-A53 up to 1.4GHz and single Cortex-M4F up to 400MHz, allowing customers to run a highly responsive and versatile GUIs across general purpose applications.
- **Graphics and Display:** Imagination AXE-1-16M GPU with 2D/3D acceleration; dual LVDS/DPI display support up to 1920 × 1080 at 60fps.

2.2.5 Texas Instruments AM62P and AM67 Platforms

- **Architecture:** Quad Cortex-A53 up to 1.4GHz plus two independent Cortex-R5F cores at 800MHz. The R5F cores can run real-time control or safety tasks independently of the Linux-driven UI.
- **Graphics and Display:** Imagination BXS-4-64 GPU with 2D/3D acceleration; triple display support over LVDS, DSI, or DPI up to 3840 × 1080 at 60fps. These devices also have H.264/H.265 4K UHD video capability.

2.2.6 Texas Instruments AM67A Platform

- **Architecture:** Quad Cortex-A53 up to 1.4GHz plus two independent Cortex-R5F cores at 800MHz. The device includes two deep-learning accelerators up to 4 TOPS to support AI capabilities.
- **Graphics & Display:** Imagination BXS-4-64 GPU with 2D/3D acceleration; triple display support over LVDS, DSI, or DPI up to 3840 × 1080 at 60fps. This device also has H.264/H.265 4K UHD video capability.

Note

KDAB's Perspective: The Total Cost of Ownership Equation

In architectural reviews we frequently see teams selecting a UI framework on developer familiarity alone (for example, web technologies or Flutter), only to discover late that the framework needs substantially more memory to run smoothly. Upgrading from a low-cost processor configuration to a higher-end configuration to absorb software overhead can destroy the margin of the product. Hardware and UI software must be selected symbiotically.

Copyright © 2026 Texas Instruments Incorporated

3 Technology Deep Dive: The Frameworks

3.1 Qt: Cross-Platform Application Framework

Qt is a complete, C++-based application platform — not just a UI renderer — with over 25 years of continuous development. Qt provides tooling, cross-platform deployment, and long-term commercial support (LTS) suited to products with 10-to-15-year lifecycles.

3.1.1 Technology Architecture

- **Qt Quick (QML):** Declarative UI on a hardware-accelerated scene graph.
- **Qt Widgets:** Traditional imperative C++ UI toolkit; relevant for data-heavy or CPU-rendered interfaces.
- **Qt Quick Compiler:** Ahead-of-time compilation of QML to reduce startup time and memory footprint.
- **Qt Quick 3D:** Native integration of 3D content inside 2D interfaces.
- **Qt for MCUs (Qt Quick Ultralite):** A custom QML subset and deterministic runtime targeting microcontrollers.

3.1.2 Platform Suitability & Real-World Scaling

- **Low-tier Sitara devices (AM335, AM62L):** Viable with rigorous optimization and an appropriately scoped UX. Minimize QML nesting depth, reduce overdraw and validate the graphics path early. Qt Widgets can be suitable for narrow data-oriented interfaces, but modern touch UX requires substantial product-specific design investment.
- **Mid-tier Sitara devices (AM62, AM62A):** A strong target for Qt Quick. These platforms offer a balance of CPU, RAM and graphics capability for moderately complex applications. Measure start-up, memory and sustained frame behavior on target hardware; Qt Quick Compiler can reduce launch-time work.
- **High-tier Sitara devices (AM62P, AM67, AM67A)**: Qt is an excellent choice, with hardware-accelerated graphics paths and scope for rich multi-display UI designs. Confirm the production graphics stack and validate composition and bandwidth for the intended display configuration.

3.1.3 The Business Case: Advantages and Challenges

- **Strengths:** Mature ecosystem; Qt Creator and Qt Design Studio (with a Figma bridge); single-codebase deployment across Linux, RTOS, Windows, iOS, and Android; certification support for IEC 62304 and ISO 26262 via the Qt Safe Renderer (a separate product; these certifications do not cover the full Qt framework); Squish test automation.
- **Challenges:** Large footprint versus minimal frameworks; learning curve around CMake/qmake; licensing complexity for closed-source products or static linking scenarios. Commercial Qt licensing can be a material total-cost-of-ownership consideration depending on deployment model, product volume, support needs, static-linking approach and additional Qt products. Teams must obtain current commercial terms and evaluate licensing alongside engineering and BOM cost.

Note

KDAB's Perspective: Licensing vs Development Velocity

Commercial Qt licensing is a direct and ongoing cost. The real comparison is between tool ecosystems; Qt Creator and Design Studio, Slint's live-preview tooling, and Flutter's hot reload each accelerate different workflows. Teams must evaluate tool fit against their product rather than assuming any one framework is uniquely productive.

3.2 Slint: The Modern Lightweight Alternative

Slint is a Rust-based declarative UI framework designed from the ground up for extreme efficiency, memory safety, and developer productivity. Slint reports a runtime RAM footprint under 300KB while still supporting animations, touch input, and hardware-accelerated rendering.

Copyright © 2026 Texas Instruments Incorporated

3.2.1 Technology Architecture

- **Declarative UI Language:** A custom, highly readable markup language with deep tooling support.
- **Multiple Rendering Backends:** Software renderer for MCUs, plus OpenGL, Metal, Vulkan, Direct3D, and Skia accelerated renderers for capable MPUs.
- **Language Bindings:** First-class Rust and C++ integrations, with additional bindings for Python and JavaScript/TypeScript.

3.2.2 Platform Suitability and Real-World Scaling

- **Low-tier Sitara devices (AM335, AM62L):** Slint is a strong candidate where low footprint, predictable memory behavior and a declarative workflow are priorities. The choice of software or accelerated rendering backend must follow the production graphics stack and measured workload.
- **Mid-tier Sitara devices (AM62, AM62A):** This tier aligns well with the strengths of Slint. GPU-accelerated backends can be used where supported; plan the renderer against the actual graphics API and driver availability.
- **High-tier Sitara devices (AM62P, AM67, AM67A):** Slint can support rich and responsive accelerated UIs, including multi-display designs. With more abundant resources, key advantages shift to Rust memory safety, licensing model, and cross-language bindings.

3.2.3 The Business Case: Advantages and Challenges

- **Strengths:** An approximately 300KB runtime, Rust-level memory safety, live preview and hot reload, and a Figma import that is useful if you already have Figma designs (though it is not a primary authoring workflow). The dual GPLv3 + Commercial license is straightforward. Low CPU wake events also make Slint attractive for battery-powered devices.
- **Challenges:** Slint is a younger framework. Third-party widgets and ecosystem depth trail Qt. The core design tools are excellent, but IDE integration and comprehensive documentation for complex edge cases are still maturing, and the certified-domains track record is still in its infancy.

Note

KDAB's Perspective: the Rust Revolution in Embedded

As a service partner for Slint, KDAB closely monitors adoption. We are seeing growing interest from EV charging and IoT industrial device teams. These projects share tight memory and CPU budgets, long lifespans, and a critical need for software security – areas where Rust shines. Slint allows these teams to ship fluid animated interfaces on hardware that traditionally supported only basic, low-resolution UI.

3.3 LVGL: The Microcontroller Specialist

The Light and Versatile Graphics Library (LVGL) is the leading open-source UI framework specifically optimized for microcontrollers. Written entirely in C with minimal external dependencies, the LVGL enables sophisticated graphical interfaces on hardware that historically supported only segment or monochrome bitmap displays.

3.3.1 Technology Architecture

- **Pure C Implementation:** Maximum portability across compilers and platforms.
- **Hardware Abstraction:** Pluggable display, input, and filesystem drivers.
- **Rendering Agility:** Highly optimized CPU software rendering with hooks for platform-specific GPU acceleration.
- **RTOS Integration:** Fully compatible with FreeRTOS, Zephyr, RT-Thread, or bare-metal.

3.3.2 Platform Suitability and Real-World Scaling

- **Low-tier Sitara devices (AM335, AM62L):** LVGL is highly viable for cost-sensitive applications. The small footprint of LVGL preserves resources for application logic and is an excellent choice for controlled appliance-style interfaces.
- **Mid-tier Sitara devices (AM62, AM62A):** LVGL remains suitable when portability, a small footprint or an existing investment matters. For graphics-rich HMI designs, evaluate whether a GPU-oriented declarative framework can use the platform more effectively and improve iteration speed.

Copyright © 2026 Texas Instruments Incorporated

- **High-tier Sitara devices (AM62P, AM67, AM67A):** LVGL is technically feasible, but its value can diminish when the product aims to make extensive use of graphics and multi-display capability. The choice must be deliberate rather than automatic.

3.3.3 The Business Case: Advantages and Challenges

- **Strengths:** True MCU optimization, a permissive MIT license with unrestricted commercial use, an active community, and LVGL Pro for visual design.
- **Challenges:** Manual C UI code is slower and more error-prone than declarative approaches. Pure C lacks the memory-safety guarantees of Rust, placing the burden of correct allocation and pointer handling on developers. The imperative API becomes hard to maintain as state complexity grows.

3.4 Flutter: The Mobile-First Approach

Flutter is Google's vision for cross-platform development. Flutter is widely adopted among mobile developers (consistently ranking among the top cross-platform mobile frameworks in developer surveys). The primary focus of Flutter remains mobile and web; embedded Linux support is capable but community-driven rather than a fully supported first-party effort.

3.4.1 Technology Architecture

- **Dart Language:** Garbage-collected, with JIT (development, hot reload) and AOT (production) compilation.
- **Custom Rendering:** All pixels drawn via the hardware-accelerated Skia engine (transitioning to Impeller), ensuring pixel-perfect consistency.
- **Widget Composition:** An "everything-is-a-widget" philosophy with highly reactive updates.

3.4.2 Platform Suitability and Real-World Scaling

- **Low-tier Sitara devices (AM335, AM62L):** Flutter is not a default recommendation. A production decision must follow a target-board proof of concept covering startup, memory use, sustained responsiveness, graphics driver behavior, and power budget.
- **Mid-tier Sitara devices (AM62, AM62A):** Flutter can be evaluated where the product has adequate RAM and a mature accelerated graphics path. Flutter can be justified if mobile-code sharing and available Flutter skills offset the additional runtime cost. Results vary materially by workload and hardware configuration.
- **High-tier Sitara devices (AM62P, AM67, AM67A):** Flutter is a credible option for visually rich applications when the system provides sufficient RAM and robust GPU acceleration. 60fps can be achievable for appropriate workloads, but the final application, graphics stack, resolution and display topology must be measured.

3.4.3 The Business Case: Advantages and Challenges

- **Strengths:** An unparalleled developer experience: hot reload, IDE support, rich built-in widgets (Material/Cupertino), and a massive package ecosystem (pub.dev). The ability to share a codebase between a mobile companion app and the embedded device reduces hiring bottlenecks.
- **Challenges:** Severe resource overhead can force hardware BOM upgrades. Embedded Linux support is largely community-driven: notable embedders include Toyota's ivi-homescreen (https://github.com/toyota-connected/ivi-homescreen), used in AGL-based automotive infotainment, and flutter-pi (https://github.com/ardera/flutter-pi) for Raspberry Pi and similar boards. Dart's garbage collection introduces unpredictable latency, making Flutter unsuitable for hard real-time requirements. In systems with dedicated real-time processors or software components, those tasks can be architecturally separated from the Flutter-driven UI.

Note

KDAB's Perspective: The Flutter "Hidden Cost" Trap

Because Flutter is BSD-licensed, organizations often assume Flutter is the cheapest route. But if your C/C++ team must learn Dart, and the hardware team has to double RAM and upgrade the GPU to handle Flutter's overhead, the TCO frequently exceeds the cost of commercial licenses for highly optimized frameworks like Qt or Slint.

Copyright © 2026 Texas Instruments Incorporated

3.5 Other Notable Frameworks

- **Altia**: Commercial embedded and automotive HMI development platform with a partner ecosystem that includes Texas Instruments demonstrations. It is worth evaluating for teams seeking a model-based UI workflow and vendor-supported deployment across embedded targets. Trade-offs include licensing, generated-code workflow, runtime footprint, graphics-stack integration and fit with the existing application architecture.
- **Avalonia UI**: Cross-platform .NET framework (C# / XAML). Can run on Linux via DRM/KMS. Trade-off: .NET runtime overhead; not suitable for MCUs.
- **Crank Software:** A commercial GUI toolchain spanning MPUs and MCUs, marketed for broad silicon support and rapid UI prototyping; worth evaluating for teams wanting a vendor-agnostic design/runtime pipeline.
- **Embedded Wizard**: Commercial MCU/MPU framework strong in automotive and white goods. Trade-off: Higher licensing cost.
- **Kanzi (Rightware)**: Premium framework dominant in automotive infotainment. Trade-off: Premium licensing.
- **.NET MAUI**: Microsoft's Xamarin successor. Strong for mobile/desktop; embedded Linux support is effectively community-only. Trade-off: Better for companion apps than the embedded device itself.
- **Unreal Engine UMG**: AAA game engine used in high-end signage and luxury EV dashboards. Trade-off: Heavy hardware needs, steep learning curve, large binaries.

4 The KDAB Decision Framework and Implementation Best Practices

The Texas Instruments Sitara HMI tier table in Section 2.1 is the primary platform-first selection guide for this edition. Table 2-1 provides primary starting points and alternative frameworks to evaluate.

4.1 Decision Criteria Beyond the Silicon

4.1.1 Licensing and Business Model

- **Open-source permissive:** LVGL (MIT), Flutter (BSD).
- **Open-source copyleft:** Qt (GPL/LGPL), Slint (GPL). Review static-linking and tivoization implications.
- **Commercial required:** Qt (closed-source/static), Slint (commercial), Qt for MCUs, Embedded Wizard, Kanzi.

4.1.2 Team Expertise and Development Velocity

- **C/C++ embedded teams:** Qt, Slint (C++ bindings), or LVGL.
- **Rust teams:** Slint.
- **C# / .NET teams:** Avalonia.
- **Mobile developers:** Flutter (hardware permitting).

Note

Note on Velocity: Declarative frameworks (QML, Slint, Flutter, XAML) generally yield significantly faster iteration than imperative manual coding (LVGL, Qt Widgets).

4.1.3 Long-Term Maintenance & Certification

- **Established track record:** Qt (25+ years), LVGL (10+ years), Embedded Wizard.
- **Emerging:** Slint (newer, commercially backed), Flutter embedded (community-driven).
- **Safety-critical (IEC 62304 / ISO 26262):** Qt (commercial) and Embedded Wizard are the conservative choices. Validating Slint is possible but requires investment. Flutter's garbage collection makes hard real-time certification exceedingly challenging.

Copyright © 2026 Texas Instruments Incorporated

4.2 Implementation Best Practices: Optimizing Performance

4.2.1 Memory and Graphics Management

- **Deterministic memory:** Static allocation where possible; profile worst-case RAM early; pool frequently created and destroyed objects.
- **Test on target hardware early:** A common and expensive pitfall is developing the entire UI on a workstation and only validating on the target shortly before launch. Run on target silicon from the first sprint, including thermal and boot-time measurements.
- **Asset optimization:** Compress and stream assets. On Qt projects, always use the Qt Quick Compiler.
- **Rendering efficiency:** Minimize overdraw through efficient layer composition. Use hardware acceleration using OpenGL, OpenGL ES, Vulkan, or platform-native APIs wherever available.

4.2.2 Power Efficiency

- Anything that can be suspended when not in use must be — GPU, display controller, sensors, and peripherals. Co-design the UI state machine with the system's power manager.

4.3 Start-Up Time Considerations (The "Cold Boot" Problem)

Application launch time critically impacts the perceived quality of embedded devices.

- **Qt:** The Qt Quick Compiler reduces cold boot by hundreds of milliseconds through AOT compilation.
- **Slint:** Minimal start-up overhead thanks to its tiny compiled binary size.
- **LVGL:** Fast initialization with minimal runtime setup.
- **Flutter:** Longer initialization due to Dart VM and framework loading.

Universal optimization: Trimmed dependencies and precompilation contribute to fast boots. Static linking can help but is not a silver bullet — apply it selectively rather than as a default performance lever.

4.4 Power Efficiency (Battery-Powered Devices)

- Implement adaptive frame rates based on UI activity (for example, 60fps active, 30fps idle, 1fps static).
- Anything that can be suspended when not in use must be — GPU, display controller, sensors, and peripherals. Co-design the UI state machine with the system's power manager.
- Batch display updates to reduce memory bandwidth and minimize CPU wake events.

4.5 Future Trends and Considerations

The embedded UI landscape continues to evolve rapidly. Periodic re-evaluation makes sure your technology choices remain optimal across long product life cycles.

- **WebAssembly and Web-Based UIs:** Growing interest in deploying UIs via WebAssembly for dashboards and remote monitoring. Both Qt for WebAssembly and Slint for WebAssembly are viable paths. The trade-off remains portability vs. runtime overhead.
- **Machine Learning Integration:** Modern platforms increasingly include AI accelerators, and UI frameworks are integrated with ML pipelines for gesture recognition and predictive interfaces. Qt already offers ML framework bindings.
- **The Rust Ecosystem Growth:** Memory safety without garbage collection is highly attractive for embedded systems. Slint leads as a Rust-native UI framework, and Qt is actively working on experimental Rust bindings.
- **Hardware Evolution:** New MCUs are approaching entry-level MPU performance, blurring the MCU/MPU boundary and enabling richer UIs on traditional MCU budgets. Vulkan adoption is growing, promising more consistent graphics APIs across vendors.

Copyright © 2026 Texas Instruments Incorporated

5 Summary: TI and KDAB's Perspective

Selecting an embedded UI technology is an exercise in compromise. There is no single "best" framework: only the best framework for a given combination of hardware capability, team expertise, licensing posture, and product lifecycle. Equally, there is no single "right" processor: the TI Sitara MPU portfolio is deliberately architected to span the full embedded HMI spectrum, from cost-optimized entry-level designs to feature-rich, multi-display, and AI-accelerated systems.

Across the entire Sitara MPU portfolio, TI provides a consistent software foundation including Linux SDK support, display drivers, and hardware-accelerated graphics stacks. This allows development teams to move between tiers as product requirements evolve without discarding their software investment. Frameworks that do not suit today's project constraints may become viable as hardware capabilities increase and ecosystems mature, so treat this selection as decisions to revisit — not a one-time choice.

5.1 About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++, and OpenGL applications across desktop, embedded, and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. Our experts build runtimes, mix native and web technologies, and solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling, and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market-leading, hands-on training for Qt, OpenGL, and modern C++ in multiple languages.

5.2 About the Texas Instruments Sitara MPU Group

Texas Instruments' Sitara portfolio is a family of processors designed for industrial automation, HMI, robotics, motor control, edge AI applications, etc. Built on the Arm® Cortex®-A architecture, the Sitara MPU family spans a wide range of performance tiers from entry-level, cost-optimized devices to multi-core and AI-accelerated devices. TI provides comprehensive development resources including the Processor SDK for Linux, real-time software, reference designs, evaluation modules, and a broad ecosystem of System-on-Module (SoM) and carrier-board partners to accelerate time-to-market. With decades of experience in embedded processing, TI supports product life cycles of ten years or more, giving industrial and commercial customers the confidence to build on Sitara for demanding, long-lived deployments.

6 References

- KDAB Group, www.kdab.com
- Texas Instruments, www.ti.com

Copyright © 2026 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025