SILICON SIGNALS

GUI library benchmark report on an ST platform

A Benchmark Evaluation of **Qt for MCUs, LVGL, and TouchGFX**

Silicon Signals · March 2026 · White Paper

This white paper presents a structured technical evaluation of Qt for MCUs, LVGL, and TouchGFX based on a common benchmarking implementation on the STM32U5 hardware platform. The objective is to provide engineers with practical, implementation-level insights to support framework selection based on measurable characteristics rather than assumptions.

Hardware Platform	Frameworks Evaluated	Publication
STM32U5 · ARM Cortex-M33 800×480 TFT · FreeRTOS DMA2D / Chrom-ART GPU	Qt for MCUs LVGL (MIT) TouchGFX	March 2026 Silicon Signals Technical White Paper

Executive Summary

Modern embedded systems increasingly require graphical user interfaces that support real-time interaction, animation, and dynamic data visualization — within the constraints of microcontroller-based platforms where CPU, memory, and bandwidth are limited.

Selecting an appropriate GUI framework is therefore a critical system-level decision. The choice directly impacts rendering performance, memory utilization, development complexity, and long-term scalability. This white paper presents a structured technical evaluation of Qt for MCUs, LVGL, and TouchGFX using identical UI scenarios on the same STM32U5 hardware platform, enabling direct comparison of runtime behavior, rendering efficiency, and development workflows.

Key Findings

- TouchGFX delivers the lowest CPU utilization and most predictable memory behavior on STM32 hardware through tight DMA2D integration.
- LVGL provides the greatest platform flexibility and cost efficiency, with performance highly dependent on developer-level optimization.
- Qt for MCUs enables the most sophisticated UI designs with the least frontend development effort, at the cost of higher resource overhead and licensing.

1. Increasing Demands on Embedded GUI Systems

Embedded GUI systems have evolved from static display interfaces to dynamic rendering systems that must handle complex real-world workloads simultaneously:

- Continuous frame updates (video-like workloads)
- Multiple concurrent animations
- Alpha blending and transparency
- Vector graphics (SVG) rendering
- Real-time data updates and UI state transitions

These requirements introduce several technical challenges that demand careful framework evaluation:

- Limited memory for framebuffer and assets
- CPU constraints for rendering and animation
- Need for efficient partial redraw strategies
- Synchronization between UI updates and system tasks

As GUI complexity increases, inefficiencies in rendering architecture or memory handling become immediately visible in system performance.

© 2026 Silicon Signals. Confidential.siliconsignals.io

2. Embedded GUI Framework Evaluation Matrix

Selecting a GUI framework requires evaluating multiple low-level and architectural parameters. A framework that performs well in one dimension may introduce trade-offs in another; therefore, evaluation must be based on realistic workloads.

2.1 Rendering Architecture

- Framebuffer strategy (single, double, partial)
- Region-based redraw vs full-screen updates
- GPU/DMA acceleration support

2.2 Resource Utilization

- RAM usage for framebuffers and UI objects
- Flash usage for assets
- CPU load during animation and updates

2.3 Development Model

- Declarative vs imperative UI design
- Code generation vs manual implementation
- Debugging and profiling capabilities

2.4 Portability and Platform Dependency

- Hardware-specific optimizations
- Effort required for porting across MCU/MPU platforms

3. Benchmark Methodology

3.1 Hardware Platform

All frameworks were evaluated on a consistent hardware platform to ensure comparable results:

- STM32U5 series microcontroller (ARM Cortex-M33)
- Integrated DMA2D (Chrom-ART) and GPU acceleration
- 800 × 480 TFT display with capacitive touch
- FreeRTOS-based runtime environment

3.2 Common Benchmark Implementation

A unified benchmark application was implemented across all frameworks with identical UI structure, same graphical assets, equivalent animation behavior, and consistent timing and execution flow. This ensures that performance differences are due to framework behavior rather than implementation variance.

© 2026 Silicon Signals. Confidential.siliconsignals.io

3.3 Test Scenarios

Each framework was evaluated using multiple UI workloads designed to reflect real-world embedded usage patterns:

Scenario	Description
Video Rendering	Continuous frame updates using sequential image rendering. Measures frame stability and sustained throughput.
2D Animation	Rotation and translation of multiple images. Evaluates transformation efficiency and animation smoothness.
Static Rendering	Rendering of shapes with color transitions and alpha blending. Tests fill rate and blending performance.
SVG Rendering	Vector graphic rendering and animation. Evaluates scalability and rendering complexity handling.
Text Scrolling	Rendering and smooth scrolling of large text blocks. Measures text rendering efficiency and update handling.

Table 1: Benchmark test scenarios applied across all frameworks

3.4 Measured Metrics

The following runtime metrics were collected across all test scenarios:

- Frames Per Second (FPS)
- CPU Utilization (%)
- RAM Usage
- Flash Usage
- Render Time per Frame

4. Framework Architecture Overview

4.1 TouchGFX

TouchGFX is designed specifically for STM32 microcontrollers and leverages hardware acceleration features. Its architecture delivers predictable memory behavior and efficient rendering on STM32 hardware with minimal abstraction overhead.

- Rendering: Framebuffer-based with partial updates
- GPU Integration: Tight coupling with DMA2D and Chrom-ART
- Memory Model: Static allocation — deterministic behavior
- Portability: Limited to STM32 platform ecosystem

4.2 LVGL

LVGL is a platform-independent graphics library with configurable rendering behavior. It provides high flexibility across hardware platforms and fine-grained control over memory and performance — but requires explicit optimization by developers.

- Rendering: Configurable — single/double/partial framebuffer
- GPU Integration: Software-based, optional hardware acceleration
- Memory Model: Manual control over rendering pipeline
- Portability: Highly portable across MCU and MPU platforms

© 2026 Silicon Signals. Confidential.siliconsignals.io

4.3 Qt for MCUs

Qt for MCUs uses a declarative UI model with a higher-level abstraction. QML-based UI definitions are compiled to C++, enabling faster UI development and iteration with advanced animation and styling capabilities — at the cost of higher resource overhead.

- Rendering: Event-driven model via QML engine
- GPU Integration: Integrated animation framework
- Memory Model: Higher footprint than lightweight alternatives
- Portability: Scalable toward full Qt desktop environments

5. Benchmark Observations

5.1 Rendering Stability

Frameworks leveraging hardware acceleration demonstrated more stable frame rates under continuous workloads. Software-driven rendering required careful optimization to maintain consistency across extended operation.

5.2 Animation Performance

Hardware-assisted animation pipelines significantly reduced CPU utilization during complex motion sequences. Timer-driven animation systems introduced variability depending on configuration and tuning quality.

5.3 SVG Rendering Behavior

Vector rendering performance varied significantly across frameworks. Complex SVG operations increased CPU load substantially in software-driven pipelines, making hardware acceleration particularly valuable for vector-heavy UIs.

5.4 Memory and CPU Trade-offs

Static memory allocation models provided predictable behavior suitable for safety-critical and real-time applications. Dynamic or configurable systems required careful tuning to avoid fragmentation and runtime inefficiency.

6. Development Workflow and Engineering Effort

TouchGFX

- Strong integration with STM32CubeIDE
- Auto-generated UI code via TouchGFX Designer
- Structured workflow optimized for STM32 — reduced manual integration effort

LVGL

- Manual integration of display and input drivers
- Flexible but requires deeper embedded systems expertise
- Greater control over system-level behavior — steeper learning curve

© 2026 Silicon Signals. Confidential.siliconsignals.io

Qt for MCUs

- QML-based UI development with Qt Design Studio
- Clean separation between UI layer and application logic
- Higher-level abstraction simplifies complex UI implementation

7. Scalability and Portability

Framework portability directly impacts long-term product evolution and the effort required to migrate between hardware generations:

Framework	Portability	Scalability Path
TouchGFX	Low — STM32 ecosystem only	Limited to ST hardware roadmap
LVGL	High — any MCU or MPU	Broad cross-platform evolution
Qt for MCUs	Moderate — selected platforms	Scales toward full Qt environments

8. Framework Selection Guidelines

Framework selection should be driven by system constraints, product goals, and long-term scalability requirements. The following matrix maps common engineering scenarios to the most suitable framework choice.

Evaluation Criteria	TouchGFX	LVGL	Qt for MCUs
Target Hardware	STM32 MCUs only	Any MCU / MPU	Selected MCU / MPU
Licensing Model	Free (STM32 only)	Open-source (MIT)	Commercial license
Rendering Approach	HW-accel (DMA2D/GPU)	SW-driven, opt. HW	QML engine layer
Memory Strategy	Static (deterministic)	Configurable	Higher footprint
CPU Utilization	Low	Medium	Higher
Animation Capability	HW-assisted	Flexible, manual	Advanced built-in
Developer Effort	Moderate	High	Low–Moderate
UI Complexity Support	Medium–High	Medium	High
Portability	Low	High	Moderate
Best Fit Use Case	STM32-based industrial / real-time	Cost-sensitive, cross-platform	Premium UI, design-heavy apps
Best Fit Use Case	**STM32-based industrial / real-time**	**Cost-sensitive, cross-platform**	**Premium UI, design-heavy apps**

Table 2: Framework Selection Matrix — Silicon Signals Benchmark Evaluation, March 2026

© 2026 Silicon Signals. Confidential.siliconsignals.io

9. Key Takeaways

- GUI framework selection directly affects system-level performance and resource usage — this decision should be made early in product definition.
- Rendering architecture and memory strategy are the most critical evaluation factors for constrained embedded environments.
- Hardware acceleration significantly impacts performance outcomes; frameworks that leverage DMA2D and GPU features outperform purely software-driven pipelines.
- No single framework is optimal for all use cases — the right choice depends on hardware platform, UI complexity, team expertise, and licensing constraints.
- A structured, benchmark-driven approach provides the most reliable basis for decision-making in production embedded system development.

Framework Recommendation Summary

Use TouchGFX for STM32-based industrial and real-time products requiring predictable performance.

Choose LVGL for cost-sensitive, cross-platform products where developer control is paramount.

Select Qt for MCUs when UI sophistication and design quality are the primary product differentiators.

10. About Silicon Signals

Silicon Signals specializes in embedded systems, vision solutions, and performance-driven product engineering. With expertise in low-level system design, hardware-software integration, and performance validation, Silicon Signals enables development of efficient and scalable embedded products.

Silicon Signals · Embedded Systems & Vision Solutions
White Paper · March 2026 · © 2026 Silicon Signals. All rights reserved.