GUI Framework Selection for

# Constrained Embedded Systems

*A Comparative Benchmark Evaluation of TouchGFX, LVGL, and Qt for MCUs on STM32U5*

| | |
|---|---|
| **Published by** | Silicon Signals |
| **Document Type** | Technical White Paper |
| **Publication Date** | March 2026 |
| **Version** | 1.0 |
| **Classification** | Public Distribution |

## Executive Summary

Modern embedded systems increasingly require graphical user interfaces capable of real-time interaction, animation, and dynamic data visualization within the strict constraints of microcontroller-based platforms where CPU cycles, RAM, and bandwidth are finite resources.

Selecting an appropriate GUI framework is a critical system-level decision. The choice directly determines rendering performance, memory utilization, development velocity, and long-term scalability. This white paper presents a structured technical evaluation of three leading embedded GUI frameworks on TouchGFX, LVGL, and Qt for MCUs using identical UI scenarios executed on the same STM32U5 hardware platform. The approach enables direct, bias-free 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.

 

# Table of Contents

## Contents

Executive Summary ..... 2
Table of Contents ..... 3
1. Introduction ..... 4
2. Acronyms and Abbreviations ..... 4
3. Detailed Benchmark Demonstration Methodology ..... 5
3.1 Video Demo Methodology ..... 5
3.2 Static Demo Methodology ..... 5
3.3 2D Animation Demo Methodology ..... 5
3.4 SVG Demo Methodology ..... 5
3.5 Text Scroll Demo Methodology ..... 5
3.6 Motor Cluster Demo Methodology ..... 5
4. Licensing and Cost Comparison ..... 5
5. Open-Source Benchmark Demo Repositories ..... 6
6. Implementation-Level Workflow Details ..... 6
7. Problem Description: Increasing Demands on Embedded GUI Systems ..... 6
8. Framework Architecture Overview ..... 7
8.1 TouchGFX ..... 7
8.2 LVGL ..... 7
8.3 Qt for MCUs ..... 7
9. Benchmark Methodology ..... 8
9.1 Hardware Platform ..... 8
9.2 Benchmark Implementation Approach ..... 8
9.3 Test Scenarios ..... 8
9.4 Measured Metrics ..... 9
10. Benchmark Observations & Technical Analysis ..... 9
10.1 Rendering Stability ..... 9
10.2 CPU Utilization Analysis ..... 9
10.3 SVG Rendering Behavior ..... 9
10.4 Memory and Resource Trade-offs ..... 10
10.5 TouchGFX Performance Matrices ..... 10
10.6 LVGL Performance Matrices ..... 10
10.7 Qt for MCUs Performance Matrices ..... 11
11. Development Workflow & Engineering Effort ..... 12
11.1 TouchGFX ..... 12

 

11.2 LVGL ..... 12
11.3 Qt for MCUs ..... 12
12. Scalability & Portability Analysis ..... 13
13. Framework Selection Guidelines ..... 14
14. Practical Implications & Use Cases ..... 14
14.1 Industrial HMI: Process Control Panel ..... 14
14.2 Medical Monitor: Patient Bedside Display ..... 14
14.3 Consumer IoT: Premium Smart Home Device ..... 14
14.4 Cross-Platform Embedded Product Family ..... 15
15. Conclusion ..... 15
16. References ..... 16
17. Appendices ..... 16
Appendix A: Hardware Platform Specifications ..... 16
Appendix B: Benchmark Asset Specifications ..... 16
About Silicon Signals ..... 17

## 1. Introduction

Embedded GUI systems have transitioned from static display interfaces to sophisticated rendering pipelines that must handle complex concurrent workloads. Industrial HMIs, medical monitors, automotive infotainment clusters, and consumer IoT devices now routinely demand frame-level animation, real-time telemetry overlays, and vector graphics rendering all within the constraints of microcontroller-class hardware.

This white paper addresses a fundamental engineering decision faced by embedded product teams: which GUI framework is most suitable for a given application profile? The answer is non-trivial. Frameworks that appear similar in documentation differ substantially at runtime in how they utilize CPU cycles, how they allocate and reclaim memory, and how effectively they leverage hardware acceleration features such as DMA2D and Chrom-ART.

This evaluation is bounded to three commercially significant frameworks evaluated on a consistent hardware target (STM32U5, ARM Cortex-M33) using a unified benchmark methodology. The analysis is structured around runtime behavior, development workflow, and scalability providing a data-informed basis for framework selection decisions in production embedded system development.

## 2. Acronyms and Abbreviations

| Acronym | Description |
|---|---|
| MCU | Microcontroller Unit |
| MPU | Microprocessor Unit |
| DMA2D | Direct Memory Access 2D |
| FPS | Frames Per Second |
| QML | Qt Modeling Language |

| | |
|---|---|
| HAL | Hardware Abstraction Layer |
| RTOS | Real-Time Operating System |
| SVG | Scalable Vector Graphics |

# 3. Detailed Benchmark Demonstration Methodology


## 3.1 Video Demo Methodology

The video rendering benchmark uses sequential image-frame playback to emulate video streaming workloads. Each framework renders identical frame sequences at fixed timing intervals while CPU load, framebuffer throughput, and render latency are measured. This scenario evaluates memory bandwidth efficiency, framebuffer synchronization, and sustained rendering stability.

## 3.2 Static Demo Methodology

The static rendering benchmark evaluates rendering quality and idle-state efficiency using layered UI elements, alpha blending, gradients, and transparent objects. Measurements focus on idle CPU utilization, redraw behavior, and memory usage under low-animation conditions.

## 3.3 2D Animation Demo Methodology

The 2D benchmark includes moving sprites, rotations, scaling operations, and animated transitions running simultaneously. Metrics collected include animation smoothness, render time consistency, and hardware acceleration utilization.

## 3.4 SVG Demo Methodology

The SVG benchmark evaluates vector rendering efficiency using scalable assets with varying path complexity. The test measures tessellation overhead, anti-aliasing cost, and rendering stability during scaling and animated transformations.

## 3.5 Text Scroll Demo Methodology

The text scrolling benchmark measures glyph rendering performance, text layout processing, scrolling smoothness, and runtime update overhead under continuously changing textual content.

## 3.6 Motor Cluster Demo Methodology

The cluster benchmark simulates an automotive dashboard with real-time gauges, warning indicators, RPM meters, battery indicators, and synchronized animations. It evaluates multi-widget synchronization and periodic UI updates.

# 4. Licensing and Cost Comparison

| Framework | License Type | Hardware Restriction | Runtime Royalty | Commercial Cost |
|---|---|---|---|---|
| TouchGFX | ST SLA0048 | STM32 Only | No | Free for STM32 |
| LVGL | MIT Open Source | None | No | Free |
| Qt for MCUs | Commercial | Supported Platforms | Depends on License | Paid License |

## 5. Open-Source Benchmark Demo Repositories

TouchGFX Benchmark Repository: https://github.com/Silicon-Signals/stm32u5g9j-dk2-touchgfx-benchmark.git
LVGL Benchmark Repository: https://github.com/Silicon-Signals/stm32u5g9j-dk2-lvgl-benchmark.git
Qt for MCUs Benchmark Repository: https://github.com/Silicon-Signals/stm32u5g9j-dk2-qt-benchmark.git

## 6. Implementation-Level Workflow Details

TouchGFX workflow follows STM32CubeIDE integration with automatic code generation through TouchGFX Designer. LVGL requires manual display-driver integration, framebuffer management, and explicit optimization for DMA2D acceleration.
Qt for MCUs uses QML-based UI development where QML assets are compiled into optimized C++ code during build time.
Each framework was benchmarked using identical framebuffer configurations, synchronized timing intervals, identical graphical assets,
and FreeRTOS task scheduling to maintain fairness in runtime evaluation.

## 7. Problem Description: 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. Contemporary embedded UI requirements commonly include:

- Continuous frame updates video-like sustained throughput workloads
- Multiple concurrent animations rotation, translation, and composition
- Alpha blending and transparency layered compositing with real-time performance
- Vector graphics (SVG) rendering scalable content requiring computational geometry
- Real-time data updates and UI state transitions sensor feeds driving live display changes

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

| Challenge | Technical Impact | Business Risk |
|---|---|---|
| Limited framebuffer memory | Constrains resolution, color depth, and double-buffering | Display artifacts, screen tearing |
| CPU-bound rendering | Degrades background task execution, increases latency | System instability, missed real-time deadlines |
| Partial redraw complexity | Complex dirty-region tracking required | Increased development effort and defect risk |

| Challenge | Technical Impact | Business Risk |
|---|---|---|
| UI/system synchronization | Race conditions between UI updates and sensor tasks | Data inconsistency, display glitches |
| Framework portability | Hardware-specific code accumulates over product lifetime | High migration cost across hardware generations |

Table 1: Key embedded GUI engineering challenges and their system-level implications

## 8. Framework Architecture Overview

### 8.1 TouchGFX

TouchGFX is designed exclusively for STM32 microcontrollers. Its architecture leverages STM32-specific hardware acceleration features, particularly DMA2D (Direct Memory Access 2D) and Chrom-ART enabling efficient pixel operations with minimal CPU involvement. The rendering model is based on framebuffer strategies with partial update support, and memory allocation is entirely static, providing deterministic behavior suitable for safety-critical and real-time applications.

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

### 8.2 LVGL

LVGL (Light and Versatile Graphics Library) is a platform-independent graphics library with highly configurable rendering behavior. It provides high flexibility across hardware platforms and gives developers fine-grained control over memory usage and rendering pipeline configuration. This flexibility comes at a cost: optimal performance requires explicit developer-level optimization and deep embedded systems knowledge.

> Architecture Profile: LVGL
> Rendering: Configurable, single, double, or partial framebuffer modes
> GPU Integration: Software-based by default; optional hardware acceleration
> Memory Model: Manual control over rendering pipeline and buffer management
> Portability: Highly portable across MCU and MPU platforms

### 8.3 Qt for MCUs

Qt for MCUs applies a declarative UI model using QML, with higher-level abstractions that enable rapid UI development and iteration. QML-based UI definitions compile to C++, and the framework integrates an animation engine capable of complex motion sequences with relatively low developer effort. This abstraction comes at the

 

cost of higher memory footprint and more complex integration requirements compared to lightweight alternatives.

> Architecture Profile: Qt for MCUs
> Rendering: Event-driven model via QML engine
> GPU Integration: Integrated animation framework with optional hardware paths
> Memory Model: Higher footprint; less deterministic than static models
> Portability: Scalable toward full Qt desktop environments

# 9. Benchmark Methodology

## 9.1 Hardware Platform

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

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

## 9.2 Benchmark Implementation Approach

A unified benchmark application was implemented across all frameworks using identical UI structure, the same graphical assets, equivalent animation behavior, and consistent timing and execution flow. This methodology ensures that observed performance differences are attributable to framework behavior rather than implementation variance, a critical requirement for objective comparative analysis.

## 9.3 Test Scenarios

| Scenario | Description | Primary Metric |
|---|---|---|
| Video Rendering | Continuous frame updates via sequential image rendering. Measures sustained frame throughput and stability. | FPS, CPU % |
| 2D Animation | Rotation and translation of multiple images simultaneously. Evaluates transformation efficiency and motion smoothness. | FPS, Render Time |
| Static Rendering | Shape rendering with color transitions and alpha blending. Tests fill rate and blending pipeline efficiency. | CPU %, RAM |
| SVG Rendering | Vector graphic rendering and animation. Evaluates scalability and computational load of vector pipelines. | CPU %, Flash |

| Scenario | Description | Primary Metric |
|---|---|---|
| Text Scrolling | Rendering and smooth scrolling of large text blocks. Measures text engine efficiency and update handling. | FPS, CPU % |

Table 2: Benchmark test scenarios applied across all three frameworks

### 9.4 Measured Metrics

The following runtime metrics were collected for each framework across all test scenarios:

- Frames Per Second (FPS): rendering throughput and visual smoothness
- CPU Utilization (%): processing load imposed on the Cortex-M33 core
- RAM Usage: framebuffer and runtime object memory consumption
- Flash Usage: code and asset storage requirements
- Render Time per Frame: end-to-end time from draw call to display commit

## 10. Benchmark Observations & Technical Analysis

### 10.1 Rendering Stability

Frameworks with tight hardware acceleration integration demonstrated significantly more stable frame rates under sustained rendering workloads. TouchGFX, leveraging DMA2D offload for pixel fill and blending operations, maintained consistent frame timing across all test scenarios. LVGL, operating in software-rendering mode without explicit DMA configuration, exhibited frame-rate variance that increased proportionally with scene complexity. Qt for MCUs demonstrated stable performance in animation-heavy scenarios due to its integrated animation engine, though baseline CPU overhead was consistently higher.

**Click Here to Watch**

### 10.2 CPU Utilization Analysis

CPU utilization is the most critical metric for constrained embedded systems, as excessive UI rendering load directly reduces available cycles for application logic, communication stacks, and safety-critical tasks.

TouchGFX demonstrated the lowest CPU utilization across all test scenarios, consistently offloading pixel operations to DMA2D. LVGL's CPU load was moderate and highly sensitive to rendering configuration, developers who implemented partial framebuffer strategies and optional DMA hooks achieved performance approaching TouchGFX levels, but this required significant optimization effort. Qt for MCUs showed the highest baseline CPU utilization across all scenarios, attributable to QML engine overhead and object lifecycle management.

### 10.3 SVG Rendering Behavior

Vector rendering performance diverged substantially across frameworks. SVG operations increased CPU load significantly in software-driven pipelines, making hardware acceleration particularly valuable for vector-heavy UI designs. TouchGFX's GPU-accelerated path handled vector transformations with minimal CPU impact. LVGL's software vector engine required careful workload management to avoid CPU saturation during complex SVG

 

operations. Qt for MCUs demonstrated the smoothest SVG animation behavior due to its integrated vector engine, though at the cost of higher overall resource consumption.

### 10.4 Memory and Resource Trade-offs

Static memory allocation models (TouchGFX) provided fully predictable behavior, a requirement for safety-critical and hard real-time applications. Dynamic and configurable systems (LVGL, Qt for MCUs) offer greater flexibility but require careful tuning to avoid memory fragmentation and runtime allocation failures.

| Framework | RAM Profile | Flash Profile | Allocation Model |
|---|---|---|---|
| TouchGFX | Low - deterministic | Moderate | Static - no heap allocation |
| LVGL | Configurable - developer-controlled | Low | Configurable - heap or static |
| Qt for MCUs | Higher - QML engine overhead | Higher | Dynamic - QML object lifecycle |

Table 3: Memory profile comparison across evaluated frameworks

### 10.5 TouchGFX Performance Matrices

The following table presents measured performance metrics from the TouchGFX benchmark application executed on the STM32U5G9J-DK2 development board. Metrics cover CPU utilization and render time per demo scenario, alongside global memory figures for the complete application.

| Frameworks | TouchGFX | | | | | | |
|---|---|---|---|---|---|---|---|
| **Parameter** | **Static demo** | **2D demo** | **SVG demo** | **Video demo** | **Text Scroll demo** | **Cluster demo** | |
| **CPU (%)** | 6–8 | 7 | 23–26 | 4 | 59–62 | 10 | |
| **Render Time (ms)** | 0–1 | 3 | 7–9 | 4 | 20–22 | 1–2 | |
| **RAM usage (KB)** | Application | 95 KB | | | | | |
| | Frame buffers | 1500 KB | | | | | |
| | Stencil buffer | 380 KB | | | | | |
| | **Total** | **1975 KB** | | | | | |
| **Internal Flash (KB)** | 426 KB | | | | | | |
| **External Flash (MB)** | 44 MB | | | | | | |

Table 4: TouchGFX benchmark performance matrices (STM32U5G9J-DK2, March 2026)

### 10.6 LVGL Performance Matrices

The following table presents measured performance metrics from the LVGL benchmark application. TSC6 image compression (a lossy format supported by the NeoChrom GPU) was used for asset storage, which contributes to the reduced external flash footprint compared to other frameworks.

| Frameworks | LVGL | | | | | | |
|---|---|---|---|---|---|---|---|
| **Parameter** | **Static demo** | **2D demo** | **SVG demo** | **Video demo** | **Text Scroll demo** | **Cluster demo** | |
| **CPU (%)** | 2 | 30–33 | 15–16 | 5–6 | 4–5 | 12 | |
| **Render Time (ms)** | 1–3 | 3–4 | 9–12 | 5–6 | 23–28 | 8 | |
| **RAM usage (KB)** | Application | 289 KB | | | | | |
| | Frame buffers | 1500 KB | | | | | |
| | Stencil buffer | 375 KB | | | | | |
| | **Total** | **2164 KB** | | | | | |
| **Internal Flash (KB)** | 559 KB | | | | | | |
| **External Flash (MB)** | 24 MB | | | | | | |

Table 5: LVGL benchmark performance matrices (STM32U5G9J-DK2, March 2026). Note: TSC6 image compression used — lossy format supported by NeoChrom GPU.

## 10.7 Qt for MCUs Performance Matrices

The following table presents measured performance metrics from the Qt for MCUs benchmark application. Note that Qt for MCUs does not support runtime SVG rendering: SVG assets are pre-processed at compile time and converted to bitmap data embedded in the binary. SVG-scenario values are therefore marked with * to reflect this compile-time preprocessing behavior.

| Frameworks | Qt for MCUs | | | | | | |
|---|---|---|---|---|---|---|---|
| **Parameter** | **Static demo** | **2D demo** | **SVG demo *** | **Video demo** | **Text Scroll demo** | **Cluster demo** | |
| **CPU (%)** | < 1 | 5–6 | 14–15 * | < 1 | 2 | 2–3 | |
| **Render Time (ms)** | 3 | 8–9 | 6 * | 5 | 9–10 | 8–9 | |
| **RAM usage (KB)** | Application | 219 KB | | | | | |
| | Frame buffers | 1500 KB | | | | | |
| | Stencil buffer | 0 KB | | | | | |
| | **Total** | **1719 KB** | | | | | |
| **Internal Flash (KB)** | 519 KB | | | | | | |
| **External Flash (MB)** | 80 MB | | | | | | |

Table 6: Qt for MCUs benchmark performance matrices (STM32U5G9J-DK2, March 2026). * SVG scenario uses pre-rendered bitmap data compiled at build time; values reflect bitmap rendering, not runtime SVG parsing.

## 11. Development Workflow & Engineering Effort

### 11.1 TouchGFX

TouchGFX integrates tightly with STM32CubeIDE and the TouchGFX Designer tool. Auto-generated UI code and a structured drag-and-drop workflow reduce the manual integration effort significantly. Engineers with STM32 experience can reach functional UI prototypes quickly. However, the workflow is tightly coupled to the STM32 ecosystem porting to other hardware requires complete framework replacement.

### 11.2 LVGL

LVGL requires manual integration of display and input drivers, and the rendering pipeline must be explicitly configured. This demands deeper embedded systems expertise than TouchGFX's IDE-driven workflow. The payoff is complete control over system-level behavior, experienced developers can tune LVGL to achieve near-optimal performance on virtually any hardware target. The learning curve is steeper, and the initial integration effort is higher, but long-term flexibility is unmatched.

### 11.3 Qt for MCUs

Qt for MCUs enables UI development through QML and Qt Design Studio, providing a clean separation between UI layer and application logic. The declarative model allows frontend developers without deep embedded experience to implement complex UI behaviors efficiently. Integration overhead is higher than TouchGFX but the development velocity advantage is significant for complex, design-heavy applications. Commercial licensing costs must be factored into total cost of ownership.

| Dimension | TouchGFX | LVGL | Qt for MCUs |
|---|---|---|---|
| Primary IDE | STM32CubeIDE + Designer | Any editor / Eclipse / VS Code | Qt Design Studio |
| Code Generation | Auto-generated via Designer | Manual implementation | QML compiler to C++ |
| Driver Integration | Auto-configured for STM32 | Manual display + input drivers | Platform-specific HAL layer |
| Debug / Profiling | Integrated STM32 toolchain | Custom instrumentation | Qt tooling |
| Time to Prototype | Low | High | Low–Moderate |
| Required Expertise | STM32 + embedded C++ | Deep embedded systems | QML + embedded C++ |

Table 4: Development workflow comparison across frameworks

## 12. Scalability & Portability Analysis

Framework portability directly impacts long-term product evolution and the effort required to migrate across hardware generations. This is a frequently underweighted factor during initial framework selection but becomes a significant engineering liability as product lines expand.

| Framework | Portability | Scalability Path | Migration Effort |
|---|---|---|---|
| TouchGFX | Low - STM32 ecosystem only | Limited to ST hardware roadmap | High - complete framework replacement for non-STM32 |
| LVGL | High - any MCU or MPU | Broad cross-platform evolution | Low - HAL abstraction layer supports re-targeting |
| Qt for MCUs | Moderate - selected platforms | Scales toward full Qt desktop environments | Moderate - platform ports available for major MCU families |

Table 5: Portability and scalability comparison

# 13. Framework Selection Guidelines

Framework selection should be driven by system constraints, product goals, and long-term scalability requirements. No single framework is optimal for all use cases. The following matrix maps common engineering scenarios to the most suitable framework choice, based on the benchmark evaluation and architectural analysis presented in this paper.

| Evaluation Criterion | 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 |

Table 6: Framework Selection Matrix: Silicon Signals Benchmark Evaluation, March 2026

# 14. Practical Implications & Use Cases

## 14.1 Industrial HMI: Process Control Panel

An industrial controller HMI requires hard real-time response to sensor inputs, predictable memory behavior for safety certification, and sustained rendering of live telemetry data. CPU availability for PLC-style background tasks is critical. TouchGFX is the preferred choice: its DMA2D-accelerated rendering frees the CPU for process control logic, and its static memory model supports safety-critical certification requirements.

## 14.2 Medical Monitor: Patient Bedside Display

A patient monitoring device must display continuous waveform updates, alarm states, and trend graphs on a resource-limited MCU with strict regulatory constraints. LVGL provides the flexibility to optimize memory usage for certification compliance and supports cross-platform development if the hardware platform changes across product generations. Its open-source MIT license eliminates IP concerns in regulated environments.

## 14.3 Consumer IoT: Premium Smart Home Device

A premium smart home controller with animated UI, gesture-driven navigation, and design-forward aesthetics requires sophisticated visual behavior that is difficult to achieve with low-level frameworks. Qt for MCUs enables rapid iteration on complex UI designs through QML, with a clean separation between design and application logic, allowing parallel development tracks for UI and firmware teams. The commercial license cost is acceptable within a consumer device business model.

### 14.4 Cross-Platform Embedded Product Family

A product line spanning multiple MCU families (STM32, NXP i.MX RT, Raspberry Pi RP2040) requires a portable GUI solution that avoids hardware-specific lock-in. LVGL's hardware abstraction layer and MIT license make it the most practical choice, a single codebase can be retargeted across the product family with HAL-level driver changes, protecting the GUI investment across hardware generations.

## 15. Conclusion

GUI framework selection is a system-level decision with measurable consequences for rendering performance, memory utilization, development velocity, and long-term product scalability. This evaluation demonstrates that no single framework is universally optimal, the right choice depends on a structured analysis of hardware platform, UI complexity requirements, team expertise, licensing model, and portability needs.

TouchGFX delivers the best performance characteristics on STM32 hardware through tight hardware integration, making it the preferred choice for industrial and real-time applications where predictability is paramount. LVGL offers the greatest flexibility and portability for cost-sensitive or multi-platform products, but performance is contingent on developer-level optimization. Qt for MCUs enables the most sophisticated UI development workflows and supports complex visual designs with lower frontend development effort, at the cost of higher resource overhead and commercial licensing.

A benchmark-driven, hardware-specific evaluation methodology as applied in this paper provides the most reliable basis for framework selection decisions. Teams that defer this analysis to late-stage development often encounter performance or memory constraints that require costly architectural changes.

> Framework Recommendation Summary
>
> Use TouchGFX for STM32-based industrial and real-time products requiring predictable performance and static memory behavior.
>
> Choose LVGL for cost-sensitive, cross-platform products where developer control and portability are paramount.
>
> Select Qt for MCUs when UI sophistication and design quality are the primary product differentiators.

# 16. References


- STMicroelectronics. STM32U5 Series Product Reference Manual (RM0456). STMicroelectronics, 2023.
- ARM. Cortex-M33 Technical Reference Manual. ARM Limited, 2021.
- TouchGFX Framework Documentation. STMicroelectronics, 2024. Version 4.23.
- LVGL Documentation: Lightweight and Versatile Graphics Library. LVGL Project, 2024. Version 9.x.
- Qt for MCUs Documentation. The Qt Company, 2024. Version 2.7.
- FreeRTOS Reference Manual. Amazon Web Services, 2023. Version 10.x.
- Embedded Systems Architecture, 2nd Edition. Daniele Lacamera. Packt Publishing, 2023.
- ISO/IEC 9899:2018: Programming Languages: C. International Organization for Standardization, 2018.
- IEC 61508: Functional Safety of E/E/PE Safety-related Systems. IEC, 2010.


# 17. Appendices

## Appendix A: Hardware Platform Specifications

| Parameter | Specification |
|---|---|
| Microcontroller | STM32U595RIT6Q (ARM Cortex-M33, 160 MHz) |
| Flash Memory | 4 MB internal flash |
| SRAM | 786 KB internal SRAM |
| Display Interface | 16-bit parallel RGB + LTDC |
| Display Resolution | 800 × 480 pixels, 16-bit color |
| Touch Controller | Capacitive multitouch, I2C interface |
| GPU Acceleration | Chrom-ART Accelerator (DMA2D) |
| RTOS | FreeRTOS v10.5.1 |
| Toolchain | STM32CubeIDE 1.14 / GCC-ARM 12.2 |

Table A1: Benchmark hardware platform specifications

## Appendix B: Benchmark Asset Specifications

| Asset Type | Specification | Notes |
|---|---|---|
| Background Images | 800×480, RGB565 | Identical across all frameworks |
| Animation Sprites | 64×64 to 256×256, ARGB8888 | Pre-multiplied alpha where supported |
| SVG Assets | Scalable, path complexity: 50–200 nodes | Identical source files |
| Font Assets | Calibri 12/16/20/24pt | Pre-rasterized for TouchGFX/LVGL |

| Asset Type | Specification | Notes |
|---|---|---|
| Color Depth | 16-bit (RGB565) baseline | 32-bit where framework default |
| Framebuffer Config | Double buffer where available | Consistent across frameworks |

Table B1: Benchmark asset and configuration specifications

## About Silicon Signals

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

Areas of practice include embedded GUI development and optimization, real-time system architecture, hardware-accelerated image processing, RTOS integration and tuning, and cross-platform firmware engineering.


**Silicon Signals · Embedded Systems & Vision Solutions**