The Qt Company

WHITEPAPER

Migrating Applications from MPUs to MCUs

based on a Qt Quick unified architecture

This white paper shares our experience of porting an existing Qt Quick application to a microcontroller (MCU) with Qt for MCUs.

We chose to port our existing “HomeChef” application, which comes in a desktop, mobile, and embedded deployment. We used a unified architecture for the different deployments, which have different use cases and operating systems. All the deployments, including the MCU port, was done with one code base and one toolchain to serve one brand identity.

We provide in-depth information about the overall process of shrinking a Qt Quick application down to 480kb with Qt for MCUs 1.0 and porting it to an STM 32F769. We explain how we could address the challenges in a surprisingly short timeframe while still achieving a performance typically associated with more powerful hardware.

Contents

HomeChef Eco-System: Qt Quick Unified Architecture

Development Challenges for Multi-Deployment Scenarios
When your users want to enjoy their favourite software on multiple devices, they expect a consistent experience across the board. At the same time, your challenge to create a cohesive, unified product architecture increases, the more platforms you port your software to. Additionally, there is a shift towards incremental and iterative development processes that adds additional fragmentation to product creation.

We believe approaches such as the Unified Architecture Method (UAM) support product delivery across platforms by taking advantage of discrete user interfaces (UIs) and reusable components in cross-platform environments.

In this fragmented context of product delivery, integration has become a key concern. To address this concern, we believe approaches such as the Unified Architecture Method (UAM) support product delivery across platforms by taking advantage of discrete user interfaces (UIs) and reusable components in cross-platform environments.

With increasing technical requirements, solutions that cater to new, innovative home equipment design become more complex and expensive. While easy-to-use MCUs underperform in these new sophisticated implementations, high-end industrial Microprocessing units (MPUs) come with higher hardware and software complexity, which can lead to high follow-on costs.

Proving the Qt Technology
To outline how to address all these challenges with Qt, we created a consumer electronics demo, "HomeChef", as a real-life use case of multiple deployments as one ecosystem. The idea was that optimizing cooking (faster, cheaper, less waste, healthier) will be a key aspect of future sustainability. HomeChef keeps people updated with what they have in their kitchen and what they need to buy in the supermarket. We believe comparable scenarios will apply to other industries and use cases. HomeChef comes with three different applications and devices:

A desktop deployment to prepare your dishes of the week: Choose recipes from recipe books, favourites, or automatic recommendations, and add the ingredients to a shopping list. HomeChef can also collect nutritional information and show you statistics of how healthy you live. Instead of creating a native desktop application, we decided to deploy this desktop application to WebAssembly so that users can run this application without any installation, which makes HomeChef a showcase of Software-as-a-service (SaaS) made with Qt.

The second deployment was the mobile app, which has the same functionality as the desktop version but optimized to smaller form factors and made as a native App for iOS and Android.

Last but not least, we created an **embedded cooking machine** that gives step-by-step cooking instructions as you prepare your meals. We utilized a Garz&Fricke Santaro, a i.MX6Dual-based human-machine interface (HMI) panel running Embedded Linux with Yocto as its build system. Finally, as technology highlights, we added both WebGL streaming for remote viewing of a camera output and WebEngine integration for viewing Youtube streams.

All the deployments communicate via an MQTT[1] broker in the cloud (or running locally for demonstration purposes).

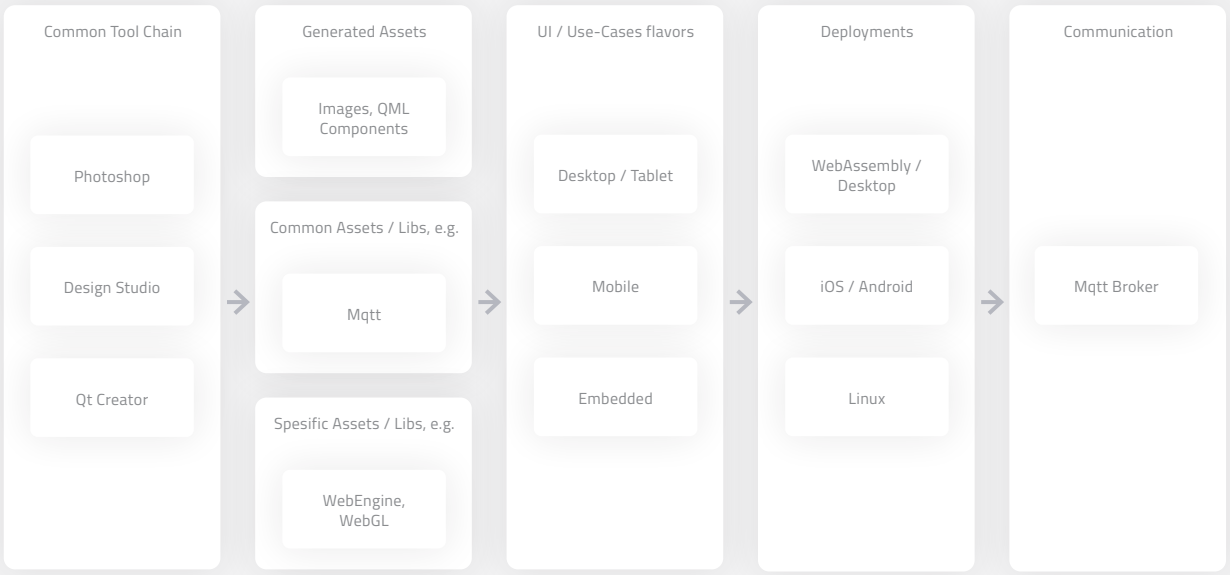


Figure 1: Unified Qt Quick Architecture

Our experience with Design Studio and WebAssembly in the making of the HomeChef

We used Qt Design studio[2] for the first time, and after initial training, it was easy to adopt in our development process. In the beginning, we used Qt Design Studio for prototyping design ideas, but soon it became an essential development tool.

1 MQTT is a machine-to-machine (M2M) protocol utilizing the publish-and-subscribe paradigm. Its purpose is to provide a channel with minimal communication overhead.

2 Qt Design Studio is a UI design and development environment for creating animated UIs and previewing them on the desktop or on Android and embedded Linux devices.

Extending to Qt UltraLite

Why did we want to develop HMIs on the MCU?
As an embedded HMI development company, Verolt was always interested in developing applications on embedded platforms like NXP and others. However, these applications almost always required Linux-based processors with GPUs and other hardware support.

We have always considered adding MCU-based HMI products to our portfolio. However, the vendor landscape so far has been quite scattered with incomplete offerings. When The Qt Company announced Qt for MCUs, a graphics framework and toolkit optimized for MCUs, it was an obvious choice for us to invest into this technology.

We will continue with an outline of our approach, the analysis we carried out for developing the HMI on Qt for MCUs, the gaps we identified, and the steps we took to overcome these gaps.

HomeChef Top-Down approach
Since the HomeChef application design and development on Linux were nearly complete, we decided on a top-down development approach for HomeChef on MCUs.

The advantage of the unified architecture is that it lets you effortlessly break up a project into smaller, more manageable pieces.

Divide and Conquer: The advantage of the unified architecture is that it lets you effortlessly break up a project into smaller, more manageable pieces.
We analysed HomeChef's features and their compatibility with MCUs, had brainstorming meetings on feasibility, and regular live-coding sessions with validation. This helped the team overcome the unknowns and achieve the feature completion goal with minimal design changes.

Thanks to this top-down approach, we could

- Focus on the features most relevant to the customer
- Test each feature as soon as we finished it. Integration tests and user tests frequently happened frequently throughout the project, which made the outcome more predictable
- Send continuous builds regularly to the customer with every newly-implemented feature

Gap Analysis (what was not available on Qt for MCUs)
A microprocessor typically runs an operating system that allows multiprocessing and multithreading. The fact that microcontrollers do not run such an operating system would impact the application design. The table below shows a feature comparison between Qt for Embedded Linux and Qt for MCUs.

Qt modules on Embedded Linux	Availability Qt for MCU
Qt_Network	Not available
Qt Quick Dialogs	Not available
Qt Quick Controls	With limitations
Qt Quick Layouts	Not available
Other modules (e.g. Qt MQTT, Qt Serialport etc.)	Not available

Developers need to be aware that Qt for MCUs does not support the Qt C++ modules. It does, however, let you use pre-existing code.

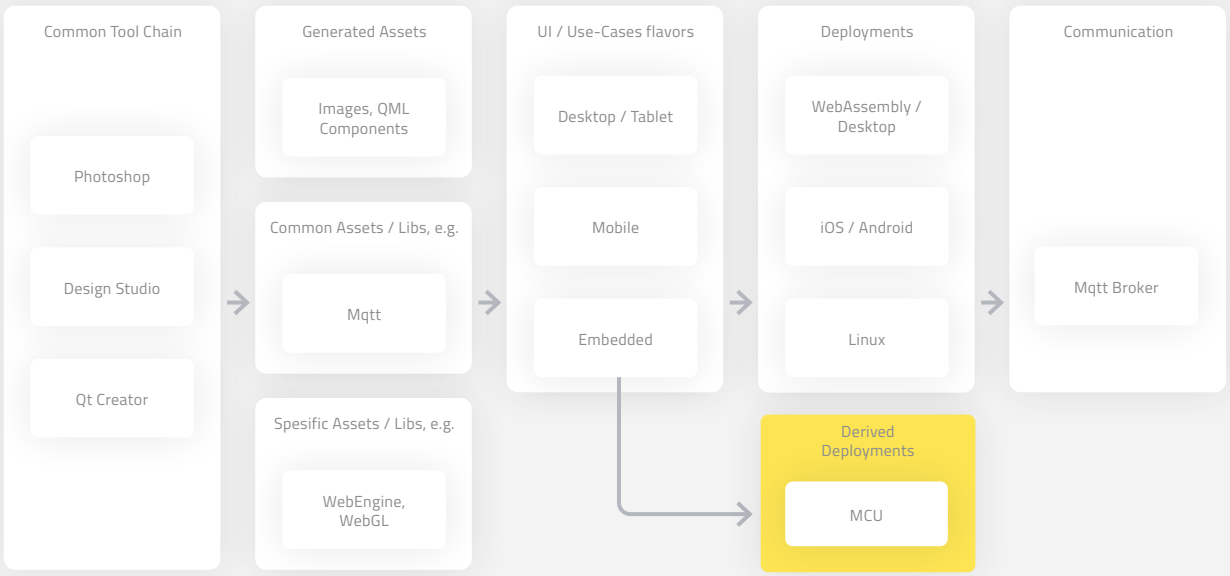


Figure 2: Deployment of HomeChef on an MCU

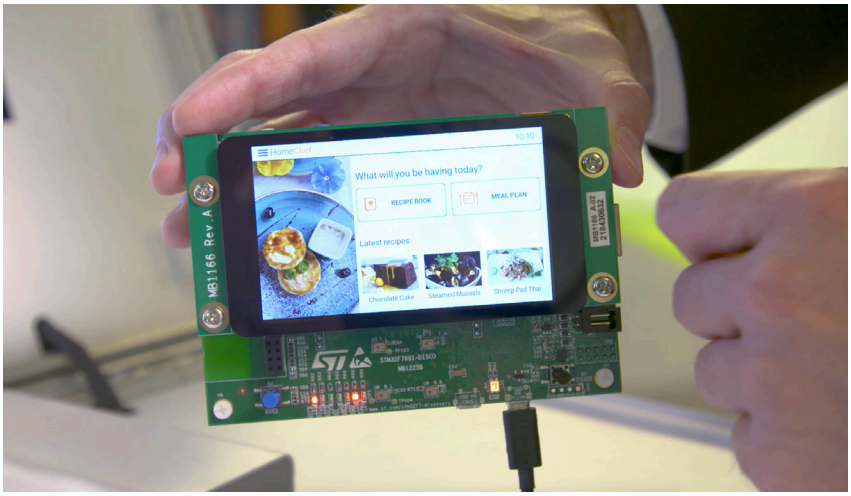
We decided to port the frontend, as the backend required more effort to integrate the MQTT and WIFI modules.

Qt for MCUs provides a subset of Qt Quick known as Qt Quick UltraLite[3] (QUL), which addresses the resource constraints in MCUs. The trade-off to the QUL's lower footprint comes in the shape of limited capabilities, compared to Qt Quick. To achieve the expected result, we had to simplify certain QML[4] design elements. Below are some of the learnings during this development phase.

Design Studio

We developed the frontend with the help of Design Studio. Because there was no support for QUL at the time of development, some components that were dependent on Design Studio components and Qt Quick required rework.

Design Studio component	Reworked for Qt for MCU	Reason for rework
Circular progress bar	QML needed to be updated with timer-based instead of timeline-based animations	Design studio component Timeline is not available for Qt for MCUs
Charts	Completely redesigned the component in QML	QChart module is not part of Qt for MCUs
Cooking Steps	Minor modification	QML type support was not available



The HomeChef app running on an MCU.

3 Graphics runtime optimised for high performance and low memory consumption on resource-constrained devices.

4 The Qt Modeling Language is a declarative Markup language for designing UI–centric applications.

Challenges migrating to Qt Quick UltraLite (QUL)

During the migration, we came across differences between QUL and Qt Quick. Here is a list of elements we needed to rework for HomeChef.

Qt Quick	QUL 1.0	Limitation	Workaround
QTimer	Available with limitations	Function Triggeronstart not available	Call explicitly start
Rectangle border	Not available		Overlapping Rectangle to show border
List View	Available with limitations	Horizontal scrolling not available.	Flickable property for horizontal scrolling.
Text Input	Not available		Custom type
Text Edit	Not available	-	Custom type
Property Animation	Available with limitations	Property animation to anchors is not available	Timer and Transform properties to Rotate the object. Prebake animation in photoshop
Java script functions	Not available		
Popup	Not available		Custom type with high Z-order
Font	Available with limitations	Font. Weight increases the memory footprint significantly. Font. Pixel size should be a fixed value	Dynamic resizing of font should be avoided.
Loader	Not available		Used visibility property for page loading
Scroll View	Not available		Flickable property for vertical scrolling
Drawer	Not available		Custom type with animation and visibility
Stack view	Not available		Custom type with visibility flag
Model	Available with limitations	Cannot declare in another file	Model and delegate should be declared in same file
Combobox	Not available		Custom types
Application window	Not available		No possible alternative, but planned for future version
ItemDelegate	Not available		No possible alternative, but planned for future version
Page	Not available		No possible alternative, but planned for future version

QUL Performance and MCU limitations

Individual animations were showing signs of performance degradation. To overcome this performance issue, we conducted several trials and finalised the solution by reducing computation and using a single timer for multiple micro-animations.

One such use case, a circular progress bar depicting the progress of kneading, was developed with design studio elements. In this particular case, we used a design studio element "Arc," which was animated by "Timeline," another Qt Design studio element. Since these elements were part of Qt Design Studio, we had to redesign the circular progress bar.

- The animation of the progress bar required more memory and power, so we had to take a timer-based rotation approach. We found that this approach significantly reduced the memory footprint and provided a smooth animation.
- At the end we ended up having an total application size of 480kB and being able to provide almost the same look and feel as the reference Embedded UI.

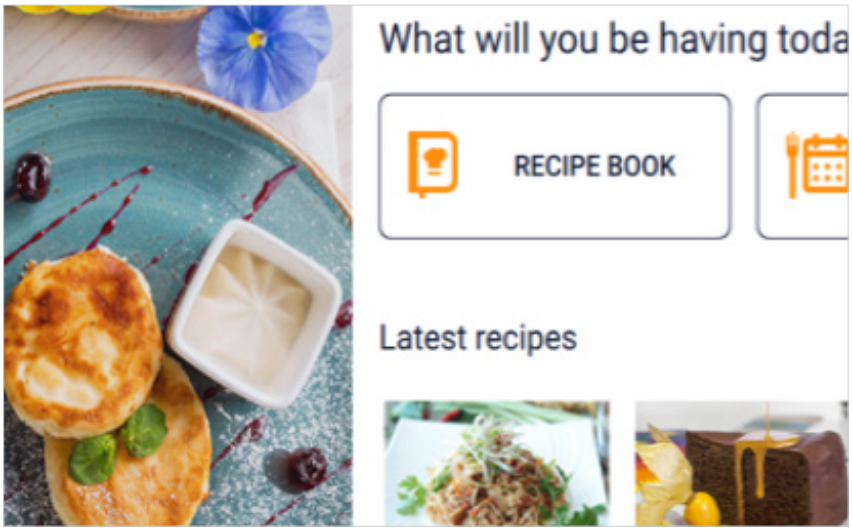
Figure 3 Animated elements (temperature, fan speed and progress bar) in the screen are optimised by reusing the same timer

Overall Effort and Timing

The key to migrating any software to a microcontroller to deliver an embedded system with a viable software architecture at reduced development costs and a quicker time to market.

In the case of HomeChef, the choice of unified architecture and advantage of Qt's offering on multiple platforms helped developers achieve a quick deployment on MCUs with a minimal learning curve. The lack of embedded software resources and development time required the reuse of software components. Here is an idea of the time and effort it took to port Qt Quick to QUL, based on our work on the four most complex screens in HomeChef for MCUs. We are basing our estimations on a developer of medium skill level with basic knowledge of QML.

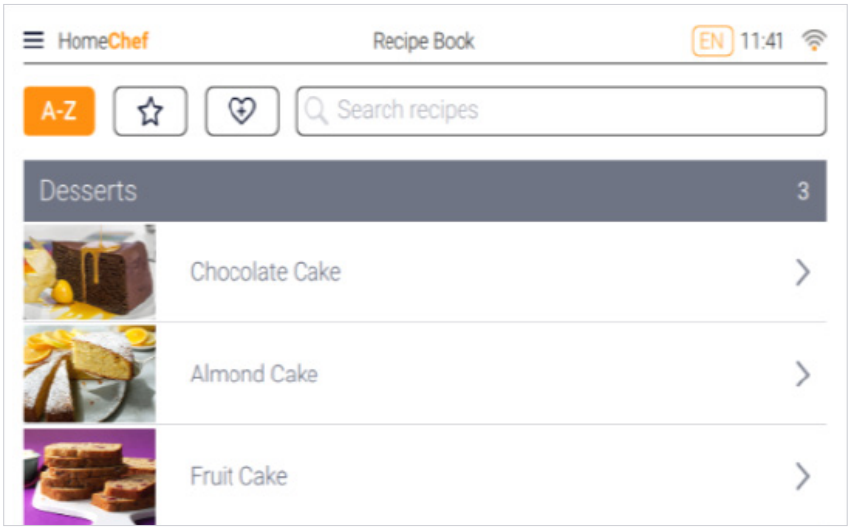
Screen



Complexity: Medium

- This screen had to undergo a medium level of changes involving a horizontal scrolling list and other GUI elements.
- **Time required**: One day

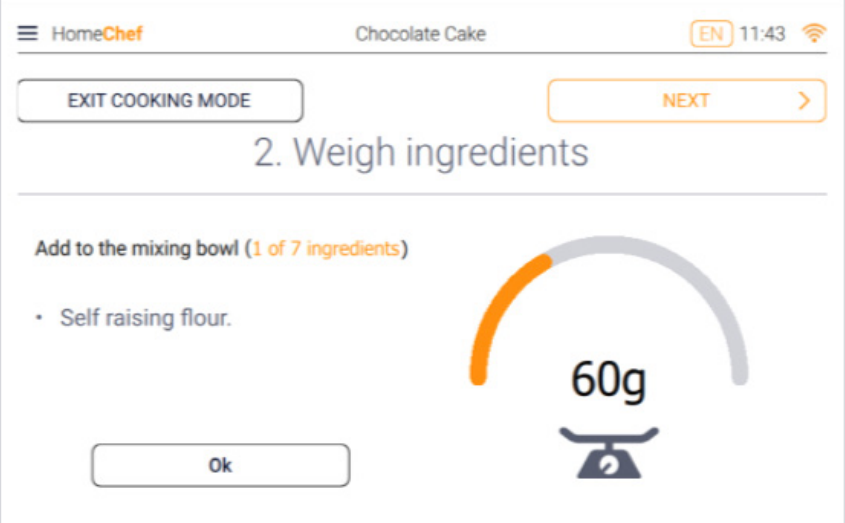
Screen



Complexity: High

- Nested collapsible list: This component had to undergo GUI changes to make it work on MCU.
- To accommodate the small screen size, the UI had been changed with enlarged fonts and mouse areas.
- **Time required:** Three days

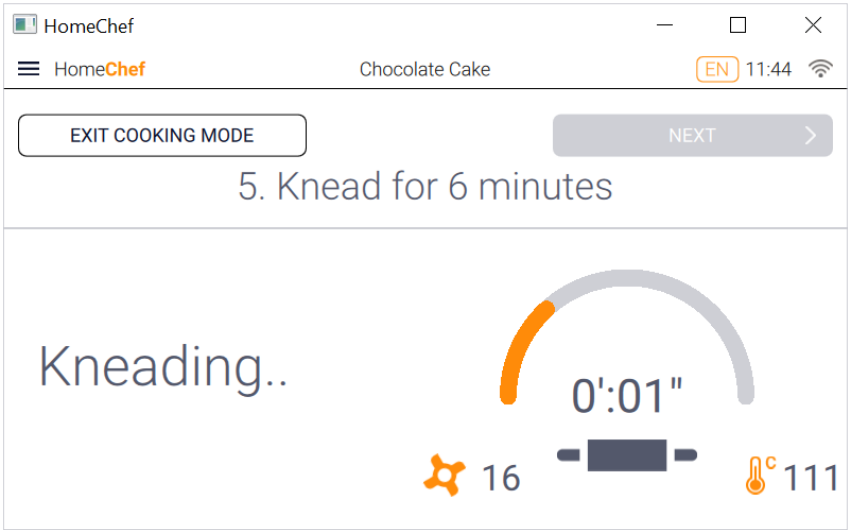
Screen



Complexity: Medium

- Animated circular progress bar
- Replacement of Qt Design Studio elements to timer-based animation
- Overlay of images has been used to display data and clip unwanted image data
- **Time required**: One day

Screen



Complexity: Medium

- Animated circular progress bar
- Rotation Animation
- Same timer used for multiple microanimations like fan or temperature.
- **Time required**: Two days

With Qt for MCUs, The Qt Company closes an essential gap in the low-cost-high-volume sector.

Conclusion/Learnings

We know from our customers that a common brand identity across all consumer/customer touchpoints is a crucial business driver. This applies not only to the consumer electronics industry like with our “HomeChef” demo but to many others.

With this demonstrator, we have been able to show that this is doable with one design and codebase across all deployment scenarios. As a result, fewer engineering skills were required, cooperation between design and development has been frictionless, the transition from prototyping to product development smooth, product iterations in sync, and components re-usable across the different UI flavours, versions, and platforms.

With Qt for MCUs, The Qt Company closes an essential gap in the low-cost-high-volume sector. As we have shown, the 1.0 version already comes with a comprehensive product-ready portfolio. Yes, things like improved QML type and SCXML[5], full Design Studio integration, or image processing (e.g. for 2.5D animations) would have been great to have, however, they haven’t been a show-stopper for us.

All-in-all, our proof of concept of the HomeChef ecosystem showed that it’s possible to reduce both the cost of engineering as well the bill of material for our customers with Qt for MCUs.

If not hampered by legacy code, a unified Architecture including MCUs is possible with Qt, and, done wisely, guarantees future innovation for engineering teams. As a service provider in this area, we are not aware of any better alternatives.

Need help? No Problem!

We can migrate to any thing. Please contact Qt to optimize your development projects.

Contact Qt at www.qt.io/contact-us/

Authors

Lars König
Director Software Services
Ulm, Germany
+49 151 612 575 36
Lars.Koenig@verolt.com
www.verolt.com

Ravi Dattatraya
Head of Engineering
Bangalore, India
+ 91 90359 87074
Ravi.Dattatraya@verolt.com
www.verolt.com

5 The Qt SCXML module provides functionality to create state machines from SCXML files.