QT Instrument Cluster – Design & Implementation Document

---

1. Objective of the Project

The primary objective of this project is to **design, develop, and deploy a Digital Instrument Cluster using Qt Quick and QML** for an embedded ARM-based target.

The project focuses on:

- Creating a modular and scalable instrument cluster UI
- Implementing real-time gauges, indicators, and warning icons
- Deploying the application on an embedded ARM platform (OKT507)
- Demonstrating end-to-end Qt application development, cross-compilation, and deployment

This project aims to simulate a **production-style automotive instrument cluster UI** using Qt technologies.

> **Note:** We followed the instructions based on the forlinx

---

2. Problem Statement

Traditional embedded instrument clusters face multiple challenges such as:

- Limited graphical capabilities
- Tight coupling between UI logic and display hardware
- Difficulty in porting UI applications across architectures
- Performance and rendering issues on embedded platforms

There is a need for a **flexible, hardware-independent UI framework** that can:

- Support complex automotive graphics
- Run efficiently on embedded ARM systems
- Allow rapid UI development and iteration
- Be easily cross-compiled and deployed

Qt Quick provides a suitable solution to address these challenges.

3. Development Environment

Software Environment

- Qt Creator 5.12.5
- Qt Quick & QML
- Qt Multimedia
- Embedded Linux

Hardware Environment

- OKT507 ARM-based board
- TFT LCD display (1024 × 600 resolution)

---

4. Project Setup and Creation

The project was created as a **Qt Quick Application** using the `qmake` build system.

Project Name

`CarCluster_OKT507`

Core Project Files

- `main.cpp` – Application entry point
- `main.qml` – Main UI layout
- `.pro` file – Build configuration
- `qml.qrc` – Resource management

---

5. Project Structure

The project follows a **modular QML-based structure**:

File	Description
main.cpp	Initializes the Qt application engine
main.qml	Main instrument cluster layout
Speedometer.qml	Speed gauge UI component
Tachometer.qml	RPM gauge UI component
GearIndicator.qml	Gear status display
Indicators.qml	Turn and hazard indicators
WarningIcon.qml	Warning icon management
qml.qrc	Embedded resources (QML & SVG icons)

This modular design improves **readability, reuse, and maintainability**.

---

6. UI Design Approach

The UI is developed entirely using **QML and Qt Quick elements**.

Implemented Features

- Speedometer and Tachometer gauges
- Turn indicators and hazard lights
- Warning icons (battery, seatbelt, brake, temperature)
- Gear indicator display
- Camera view using Qt Multimedia

Design Characteristics

- Resolution-aware layout (1024×600)
- Animated gauge needles
- Icon-based warning representation
- Embedded-friendly color and font selection

Firebase Integration

- Firebase Realtime Database is used to fetch:
- - Indicator status
- - Battery temperature
- XMLHttpRequest is used for polling data every second.

---

7. Build and Testing (Desktop)

Initially, the project is built and tested using the **Desktop Qt Kit (x86)**.

Purpose of Desktop Build

- Validate UI layout
- Test animations and interactions
- Debug QML logic
- Verify performance before embedded deployment

This step ensures **early issue detection**.

---

8. Cross Compilation for Embedded Target

The OKT507 board uses an **ARM architecture**, which requires cross compilation.

Cross-Compilation Setup

- OKT507 SDK installed on host system
- ARM cross compiler configured in Qt Creator
- ARM Qt libraries integrated into the Kit

Important Note

- Desktop (x86) executables cannot run on OKT507
- Only ARM-compiled binaries are deployed to the board

---

9. Deployment on OKT507 Board

Deployment Steps

1. Build the project using ARM Kit
2. Copy the ARM executable to the OKT507 board
3. Set execution permissions
4. Configure Qt runtime environment variables
5. Execute the application on the target

Upon execution, the digital instrument cluster UI is displayed on the TFT LCD.

10.STEPS FOR PROJECT CREATION

Step 1: Install Qt Creator

1. Download and install Qt 5.12.5 from the Qt official installer.

2. During installation, select:

    ○ Qt 5.12.5 (Desktop)

    ○ Qt Creator

    ○ Required compiler (GCC)

3. Complete the installation

Step 2: Launch Qt Creator

1. Open Qt Creator.

2. Ensure Qt version 5.12.5 is detected under:

○ Tools → Options → Kits

Step 3: Create a New Project

1. Click File → New File or Project

2. Select Application

3. Choose Qt Quick Application

4. Click Next

Step 4: Project Naming and Location

1. Enter the project name:

CarCluster_OKT507

2. Choose the project directory.

3. Click Next

Step 5: Build System Selection

1. Select qmake as the build system.

2. Click Next

Step 6: Kit Selection

1. Select Desktop Qt 5.12.5 GCC 64bit (for initial development).

2. Click Next

3. Finish the project creation.

Note: This kit generates an x86 executable used only for desktop testing.

Step 7: Default Project Files Generated

Qt Creator automatically generates:

● main.cpp

● main.qml

● .pro project file

Step 8: Modify Project File (.pro)

1. Open the .pro file.

2. Add required Qt modules:

QT += quick qml gui

CONFIG += c++11

SOURCES += main.cpp

RESOURCES += qml.qrc

TARGET = car_cluster

Step 9: Create Resource File (qml.qrc)

1. Right-click project → Add New
2. Select Qt → Qt Resource File
3. Name it qml.qrc
4. Add:
    - QML files
    - SVG icons
5. This embeds all resources into the executable.

Step 10: Add QML Files

Create and add the following QML files:

- main.qml
- Speedometer.qml
- Tachometer.qml
- GearIndicator.qml
- Indicators.qml
- WarningIcon.qml

Each QML file represents a UI component of the cluster.

Step 11: Design UI Using QML

1. Use Qt Quick and Canvas elements.
2. Implement:
    - Speedometer
    - Tachometer
    - Indicators
    - Warning icons
    - Camera view
3. Use QML properties and timers for animations and updates.

Step 12: Desktop Build & Testing (x86)

1. Click Build → Build Project

2. Run the project on desktop.

3. Verify:

UI layout

- Animations
- Indicators
- Warning logic

Step 13: SDK Installation for OKT507

1. Install the OKT507 SDK on the host system.

2. SDK provides:

- ARM cross compiler
- Qt libraries for ARM

3. Configure SDK environment.

Step 14: Configure Cross Compilation Kit

1. Open Tools → Options → Kits

2. Add a new ARM Kit using:

- SDK compiler

ARM Qt version

3. Set this kit as active

Step 15: Build ARM Executable

1. Select OKT507 ARM Kit

2. Build the project.

3. Qt generates an ARM executable.

Important:

- x86 executable cannot run on OKT507
- ARM executable runs on OKT507

Step 16: Deploy to OKT507 Board

1. Copy the generated ARM executable file to the OKT507 board using SCP or USB.

2. Set execution permission for the executable:

- chmod +x car_cluster

3.Configure Qt runtime environment variables

- export QT_QPA_PLATFORM=linuxfb
- export QT_QPA_FB_DEVICE=/dev/fb0
- export QT_QPA_PLATFORM_PLUGIN_PATH=/usr/lib/qt/plugins
- export QT_QPA_FONTDIR=/usr/lib/fonts
- export XDG_RUNTIME_DIR=/tmp/runtime-root

4.Run the application:

- ./car_cluster

The digital instrument cluster UI appears on the OKT507 display.

---

11. Issues Faced During Deployment

During embedded deployment, the following challenges were observed:

- Canvas gauge rendering differences
- Needle alignment mismatch
- Low animation frame rate
- Color visibility issues
- DPI and scaling mismatch
- High CPU usage due to frequent UI updates
- Font rendering inconsistencies
- Canvas refresh issues

---

12. Result

The Qt-based digital instrument cluster successfully runs on the OKT507 board with:

- Functional gauges
- Real-time indicators
- Warning icons
- Embedded display compatibility

The project validates the **feasibility of Qt Quick for embedded automotive UI development**.

---

13. Future Scope

The project has significant scope for expansion and real-world automotive application:

Integration with Automotive Interfaces

- CAN bus data visualization
- ECU signal mapping to UI components
- Support for standard automotive protocols

Speedometer and Tachometer Readings in Real time.

---

14. Conclusion

This project demonstrates a complete workflow of:

- Qt Quick UI development
- Modular QML design
- Desktop testing
- ARM cross compilation
- Embedded deployment

The instrument cluster design is scalable, portable, and suitable for further enhancement, making it a strong foundation for embedded automotive HMI development.