Ferhat Abbas University Setif 1
Faculty of Sciences
Department of Computer Science

Master Thesis

Presented for the award of the MASTER in Computer Science

OUIZ Ikram and BELHAOUI Fatima

Specialty: Computer Networks and Distributed Systems

MEMOIRE TITLE:

Development of a Prayer Times Display Control System Using the Qt Framework and Socket Communication

Defended on ***June 14, 2025*** *in front of the Committee composed of:*

President:	Habib AISSAOUA	Assistant Professor at UFAS1, Algeria
Examiner:	Lakhdar KANOUNI	Assistant Professor at UFAS1, Algeria
Supervisor:	Nabil GUELLATI	Associate Professor at UFAS1, Algeria

Academic Year: 2024 - 2025

Acknowledgments

We would like to express our sincere thanks and deep gratitude to everyone who contributed to the completion of this work, whether directly or indirectly.

We are especially grateful to our supervisor, **Dr. GUELLATI Nabil**, for his valuable guidance, constructive feedback, and continuous support, which had a significant impact on the success of this project.

We also extend our sincere thanks to all the professors of the **Computer Science Department** for the knowledge and skills they have imparted to us throughout our academic journey and for their constant encouragement and support.

We would also like to thank our families for their continuous support and encouragement, as well as our friends and colleagues who stood by us throughout the different stages of this project.

Finally, we thank everyone who contributed with a word, advice, or support in the completion of this work. We are truly grateful to everyone of them.

Dedication

ikram

In the Name of Allah, the Most Gracious, the Most Merciful, All praise is due to Allah, who granted me patience and strength to complete this journey. I still remember studying for the Baccalaureate with enthusiasm, dreaming of my chosen university field. And today, by the grace of Allah, I have my Master's degree in my hands.

To my beloved father, who devoted his efforts, time, and resources so that this day would become a reality in my life. To my dear mother, whose prayers never left my side, and whose love lit my path in my darkest moments.

To my siblings Sawsan, Youssef, Bouthayna, and Mahdi, thank you for always being by my side and for your constant encouragement. May you always remain close to my heart.

To my loyal and sincere friends, Thank you for the warmth and love with which you surrounded me.

To my supervising professor, without whom this work would not have seen the light of day, and to my project partner Fatima, with whom I shared the struggle, and who stood by me like a sister and an unforgettable support—

From the depths of my heart, thank you all.

Fatima

All praise and thanks are due to Allah, who granted me the strength, patience, and grace to complete this journey. I still remember the days I was preparing for the baccalaureate, full of excitement for the future— And today, by His will, I proudly hold a Master's degree in my hands.

To my beloved father, whose sacrifices knew no bounds— who gave his time, effort, and every ounce of support to see me reach this day. To my dearest mother, whose prayers accompanied me at every step, and whose love was a light in my darkest hours.

To my precious grandmother, who always dreamed of seeing me graduate— This is for you.

To my soul sisters and faithful companions, Yomna and Khalida, Thank you for being the warmth in my journey.

To my supervisor, whose guidance shaped this work, and without whom this project would never have come to light.

And to my project partner, Ikram, who stood beside me through every challenge— a true friend and unwavering support.

From the depths of my heart, thank you all.

Abstract

This project presents the development of a digital prayer times display and control system designed for mosque environments, integrating modern technologies to enhance the worship experience. The system consists of a central controller application and remote display units, enabling real-time and customizable presentation of daily prayer times. Built using the Qt framework and C++ programming language, the system leverages the capabilities of both TCP and UDP socket communication to ensure reliable and efficient data transmission. Raspberry Pi devices serve as display units due to their compact form, low power consumption, and continuous operational capabilities. The interface offers flexibility and adaptability to meet the specific requirements of various mosques, including features such as a countdown to the next prayer, automatic Adhan playback, and receiving and displaying images over the network. This work bridges academic learning with real-world application, demonstrating how embedded systems and network technologies can support religious practices and community organization in a modern and interactive manner.

ملخص

يقدّم هذا المشروع تطوير نظام رقمي لعرض والتحكم في مواقيت الصلاة، صُمم خصيصًا لبيئة المساجد، ويهدف إلى دمج التقنيات الحديثة لتعزيز التجربة الروحية. يتكوّن النظام من تطبيق مركزي للتحكم ووحدات عرض بعيدة، مما يتيح عرضًا لحظيًا وقابلًا للتخصيص لمواقيت الصلاة اليومية. تم تطوير هذا النظام باستخدام إطار العمل tQ ولغة البرمجة C++، ويستفيد من بروتوكولات الاتصال PCT وPDU لضمان نقل فعال وموثوق للبيانات. وقد تم اختيار أجهزة iP yrrebpsaR كوحدات عرض نظرًا لصغر حجمها، وانخفاض استهلاكها للطاقة، وقدرتها على العمل بشكل متواصل. يتميز واجهة النظام بالمرونة وإمكانية التكيّف لتلبية الاحتياجات الخاصة بمختلف المساجد، كما يتضمن وظائف مثل العد التنازلي للصلاة القادمة، وتشغيل الأذان تلقائيًا، بالإضافة إلى استقبال الصور وعرضها عبر الشبكة. يمثّل هذا العمل حلقة وصل بين المعارف الأكاديمية وتطبيقها العملي، ويُظهر كيف يمكن للتقنيات المدمجة وتقنيات الاتصال أن تساهم في دعم الممارسات الدينية وتنظيم المجتمع بطريقة حديثة وتفاعلية.

Résumé

Ce projet présente le développement d'un système numérique d'affichage et de contrôle des horaires de prière, conçu pour les environnements des mosquées et visant à intégrer les technologies modernes afin d'enrichir l'expérience spirituelle. Le système se compose d'une application centrale de contrôle et d'unités d'affichage distantes, permettant une présentation en temps réel et personnalisable des horaires de prière quotidiens. Développé en utilisant le framework Qt et le langage C++, le système exploite les protocoles de communication TCP et UDP pour assurer une transmission de données efficace et fiable. Des appareils Raspberry Pi sont utilisés comme unités d'affichage en raison de leur petite taille, de leur faible consommation d'énergie et de leur capacité à fonctionner en continu. L'interface du système est flexible et adaptable aux besoins spécifiques de différentes mosquées, et intègre des fonctionnalités telles que le compte à rebours jusqu'à la prochaine prière, la diffusion automatique de l'Adhan, ainsi que la réception et l'affichage d'images via le réseau. Ce travail constitue un lien entre les connaissances académiques et leur application pratique, démontrant comment les technologies embarquées et de communication peuvent soutenir les pratiques religieuses et l'organisation communautaire de manière moderne et interactive.

Contents

List of Figures

Listings

List of Tables

List of Abbreviations

Abbreviation	Definition
API	Application Programming Interface
AV	Audio-Visual
CPU	Central Processing Unit
FTP	File Transfer Protocol
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
MP3	MPEG Audio Layer 3
OOP	Object-Oriented Programming
PC	Personal Computer
QSS	Qt Style Sheets
RAM	Random Access Memory
RTL	Right-to-Left (text direction)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
USB	Universal Serial Bus
UTF-8	8-bit Unicode Transformation Format

Introduction

With the rapid advancement of modern technology, smart systems have become an integral part of everyday life, including in religious contexts. One of the most notable applications of this integration is the adoption of digital solutions to enhance the worship experience and improve the organization of mosque activities in a more efficient and adaptable manner.

Prayer is the cornerstone of Islam and one of its most significant acts of worship. Muslims have consistently placed great emphasis on performing prayers on time and establishing suitable environments within mosque settings to facilitate this obligation. Building on this principle, this project introduces a comprehensive digital solution for displaying prayer times inside mosques in an interactive and contemporary way, leveraging microcomputing and network communication technologies.

The system is built using a Raspberry Pi as the display unit, chosen for its compact size, low power consumption, and suitability for continuous operation. It uses UDP and TCP protocols are used to ensure efficient and reliable data transmission between the controller application and the display screens.

The project is developed using C++ and the Qt framework, which offers advanced tools for creating modern graphical user interfaces, as well as integrated support for multimedia and network functionalities. The interface is designed to be customizable, accommodating the specific needs of different mosques and highlighting the flexibility of the proposed system.

This work aims to bridge the gap between the theoretical knowledge acquired during academic studies and its practical implementation through a real-world application. It demonstrates how technology can serve as a valuable tool in supporting both spiritual values and community organization.

Chapter 1

Qt Library

1.1 Introduction

Every operating system provides at least one library for creating windows. However, these libraries are usually designed to function only within their respective operating systems, which limits their portability and flexibility.

To overcome this limitation, developers rely on cross-platform libraries that offer greater adaptability for multi-platform development. As stated by Blanchette and Summerfield, “cross-platform GUI frameworks enable developers to write applications once and deploy them on multiple operating systems with minimal changes” [2]. Among the most widely used GUI libraries, we can mention:

— **NET**: Primarily supports the Microsoft Win32 API. The C# language has grown rapidly while also supporting many other languages, including C++.

— **GTK+**: A widely used toolkit for Linux and macOS, providing a rich set of widgets for GUI development.

— **wxWidgets**: An object-oriented programming (OOP) library that provides similar functionality to Qt.

— **FLTK**: A lightweight library specifically designed for cross-platform GUI development.

— **Qt**: ”A powerful and flexible framework that has become the foundation for major projects, including the KDE desktop environment” [3].

Qt consists of several key modules that work together seamlessly to provide a flexible and robust solution for developers. These modules cover a wide range of functionalities, including graphical user interfaces, networking, databases, 3D graphics, and audio/video capabilities, such as those provided by Qt Company cite qtcompany.

— **QtCore**: core library offers fundamental functions including data structures, event management, and file handling.

— **QtGui**: The module that makes it possible to process images, create 2D graphics, and interact with GUI elements.

— **QtWidgets**: This component enables the building of typical user interfaces by providing a traditional set of GUI elements, including buttons, text boxes, and labels.

— **QtNetwork**: Network application development tools that facilitate communication over TCP/IP, HTTP, and FTP protocols. The construction of media players and other multimedia applications is made possible by QtMultimedia’s audio and video working features.

— **QtSql**: A module that supports well-known database management systems, including MySQL, PostgreSQL, and SQLite, and integrates database functions into applications.

— **QtQuick**: A framework for leveraging QML, a declarative language made for contemporary applications, to create dynamic and fluid user interfaces.

— **Qt3D**: A collection of tools for rendering 3D visuals and developing 3D apps.

Thanks to its cross-platform capabilities and integrated modules, Qt stands out as an ideal solution for developing powerful and versatile applications.

1.2 History of Qt

1.2.1 Qt development period (1991 - 1993)

Haavard Nord began writing the fundamental classes that would later constitute the core of the Qt library in 1991, while Eirik devoted himself to its design [2].

A year later (1992), Eirik developed the principle of ”Signals & Slots,” a robust and easy programming model for event handling, which was subsequently adopted by many other development environments. Harvard applied this idea manually cite blanchette2006c++.

In 1993, they created Qt’s first Graphics Kernel and successfully designed their graphical user interface (GUI) elements. Towards the end of the year, Harvard suggested launching a company to develop ”the best C++ GUI framework in the world.” [2].

1.2.2 Laborious start and introduction of Qt (1994 - 1995)

The two young developers started in 1994 in a complicated situation. They had ambitions to enter the graphical user interface (GUI) software market, an area already controlled by major rivals; however, they faced a lack of customers, a completed product, and funding. Fortunately, their spouses had jobs, which provided them with the opportunity to cover their private expenses for two years, a period they considered sufficient to complete product development and begin generating income [2].

The letter ’Q’ was chosen as the class prefix because the letter looked beautiful in Harvard’s Emacs font. The ’t’ was added to stand for ”toolkit,” inspired by Xt, the X Toolkit. The company was incorporated on March 4, 1994, originally as Quasar Technologies, then as Troll Tech, and is now known as Trolltech [2].

In April 1995, following the intervention of a university professor who had taught courses at Harvard Nord, they landed a software design contract with a Norwegian company called Metis to develop software based on Qt. That’s when Arnt Gulbrandsen joined the team. Subsequently, he developed a sophisticated documentation system and contributed to the writing of the code for Qt [2].

The first release of Qt (version 0.90) was made available at sunsite.unc.edu on May 20, 1995. Its release was announced on the computer. Os. Linux. Announce the newsgroup six days later. This was the first occasion where Qt was made available to the public.

1.2.3 Period of commercial success (1996 - 1997)

In March 1996, the European Space Agency (ESA) acquired 10 commercial licenses of Qt, becoming their second customer. Despite still modest sales, Eirik and Haavard continued to work on the library while recruiting a new developer [2].

- May 1996: Qt 0.97 debuts.

- Qt 1.0 was released on September 24, 1996.

- Late 1996: Qt is at version 1.1, and the company has 8 customers in 8 countries, having acquired a total of 18 commercial licenses [2].

That same year, Matthias Ettrich founded the KDE project, which then made Qt the technological basis of the KDE desktop environment [4].

The introduction of Qt 1.2 in April 1997, coupled with the adoption of Qt by KDE, was crucial to its triumph, cementing Qt as the unofficial standard for creating C++ GUIs on Linux [2].

1.2.4 Qt's Transition to Open Source (1999 and Beyond)

In June 1999, Qt 2.0 was released, introducing a new open-source license, called the "Q Public License (QPL)," making Qt compliant with the Open Source Definition [5].

Since then, Qt has continued to evolve, with the release of Qt 3, Qt 4, Qt 5, and most recently Qt 6. Today, Qt has become one of the most widely used libraries in the world for developing cross-platform applications [3].

1.2.5 Continued global success of Qt

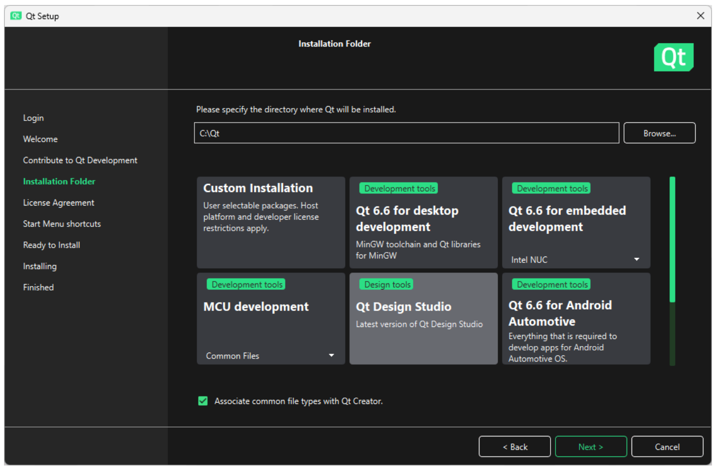
Since the establishment of Trolltech, Qt has seen a continued rise in popularity, transforming into one of the most adopted frameworks in the software development field. This triumph demonstrates the excellence of Qt and its ease of use, evolving from a product reserved for a few expert developers to a tool acclaimed by thousands of customers and tens of thousands of open-source developers worldwide [6].

1.3 Getting Started with Qt

1.3.1 Installing Qt

Qt, the cross-platform framework behind Qt Creator, has a rich and prestigious history. As a Digia project, it has its website at qt-project.org and offers both commercial and non-commercial licenses.

To get started with the non-commercial version for free, visit bit.ly/13G4Jfr[7].



Please specify the directory where Qt will be installed.

C:\Qt

Custom Installation
User selectable packages. Host platform and developer license restrictions apply.

Development tools
Qt 6.6 for desktop development
MinGW toolchain and Qt libraries for MinGW

Development tools
Qt 6.6 for embedded development
Intel NUC

Development tools
MCU development
Common Files

Design tools
Qt Design Studio
Latest version of Qt Design Studio

Development tools
Qt 6.6 for Android Automotive
Everything that is required to develop apps for Android Automotive OS.

✓ Associate common file types with Qt Creator.

Figure 1.1: Qt Setup – Installation Folder and Component Selection Screen

1.3.2 Setting Up a Qt Project

Creating a ’Hello World’ Application

Select ”New File or Project...” from the File menu in Qt Creator. Using the New Project Wizard, you can choose the type of project, name it, and configure additional details. To develop our initial application:

1_ Choose ”New File or Project...” after launching Qt Creator.

2_ Select ”Application” > ”Qt Console Application” and click ”Choose...”.

3_ Enter the project name HelloWorldConsole, choose the desired path, and click ”Next”.

4_ Select the Desktop Qt Kit, ensure both Release and Debug settings are checked, and click ”Next”.

5_ Skip the version control step by leaving it at ”None” and click ”Finish”.

6_ Let’s begin with a basic Qt application. Before compiling and running it, we will examine it line by line.

Listing 1.1: Hello World in Qt [1].

```
// helloWorld/main.cpp
#include <QApplication>
#include <QLabel>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QLabel label("Hello␣World");
    label.show();
    return a.exec();
}
```

The first two lines include the header files for the Qt classes we need: QApplication and QLabel.

Each Qt class in Qt 4 has exactly one header file, named without the common .h extension. Ensure that the header filenames are correctly capitalized when using the #include directive.

The main function of a Qt program typically starts by creating a QApplication object, which receives the command-line arguments. Without this object, no GUI application can function as it provides essential features like the event loop, which keeps the program running until the window is closed.

Next, a QLabel object is created to display the text ”Hello, world!”. Initially, this object is invisible, and we make it appear by calling its show() function.

Calling exec() starts the event loop, ensuring the application processes user interactions. The loop remains active until quit() is invoked, which happens automatically when the last main window (label) is closed and removed from memory.

Building and Compiling a Qt Program

When compiling this program, the process varies across platforms. The Qt vendor Trolltech addresses this with qmake, a tool that generates Makefiles from project files, allowing cross-platform development.

qmake generates a Makefile containing all necessary compilation instructions. On Windows, it can also generate Visual Studio project files [1].

Using qmake to Generate Project Files and Makefiles

To generate a project file for the ”Hello, world!” program, run qmake with the -project option. In a shell, navigate to the directory containing the source file and run:

```
user@linux:helloWorld$ qmake −project
```

This generates a file named helloWorld.pro with the following contents [1]:

```
#helloWorld/helloWorld.pro

Automatically generated by qmake

TEMPLATE = app
CONFIG −= moc
DEPENDPATH += .
INCLUDEPATH += .

Input

SOURCES += main.cpp
```

The SOURCES entry lists the source files in the project, while TEMPLATE defines whether the output is an application or a library (lib). To generate a Makefile, simply run:

```
user@linux:helloWorld$ qmake
```

On Windows, this command produces Makefile, Makefile. Debug and Makefile.Release. The main Makefile acts as a reference for the other two, which contain platform-specific build instructions [1].

On Unix systems, qmake generates a single Makefile unless debug versions of the Qt libraries are missing. Running make then compiles the program, producing an executable with debug output.

To ensure debug libraries are installed, look for files like libQtCore_debug.so 4 on Unix or qtcored4.dll on Windows. If needed, enable ”Build debug libraries” in the Qt setup options.

To make qmake generate both debug and release builds on Unix, add the following line to helloWorld.pro:

```
CONFIG += debug_and_release
```

Alternatively, you can set the environment variable QMAKEFLAGS to ”CONFIG+=debug_and_release” Before running qmake. However, adding this configuration to the .pro file is preferable for teams using version control systems like CVS, Subversion, or Visual Source Safe.

For a debug-only build, set CONFIG to debug. To generate only the release build, use release [1].

Running the Compilation Process

To compile the project, use the make command:

```
user@linux:helloWorld$ make release
```

This produces a release version. To build a debug version, run:

```
user@linux:helloWorld$ qmake -project
```

1.4 Layouts, Object Hierarchy, and Memory Management

1.4.1 The Architecture of Layout Classes

Widgets can be arranged in windows in a flexible and orderly manner using Qt's Layout classes. Making responsive user interfaces that adjust to various window widths requires these classes.

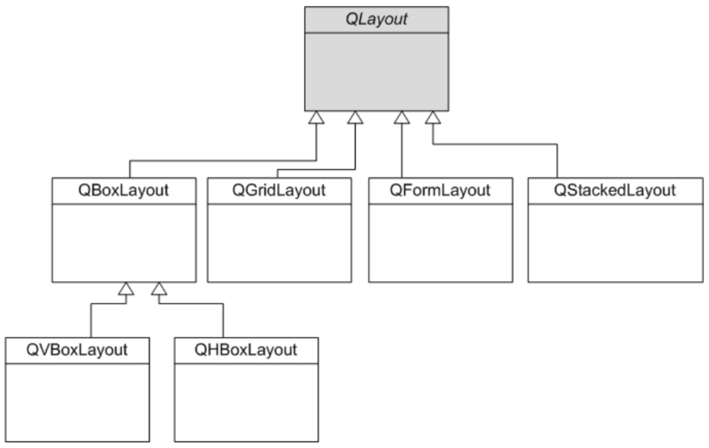


Figure 1.2: Hierarchy of Layout Classes in Qt

All classes inherit from the base class QLayout (abstract class). So we roughly count the classes:

- QBoxLayout
- QHBoxLayout
- QVBoxLayout

- QGridLayout
- QFormLayout
- QStackedLayout

Horizontal and vertical layouts

Elements can be arranged vertically or horizontally using the QHBoxLayout and QVBoxLayout classes, respectively. QBoxLayout has two subclasses: QHBoxLayout and QVBoxLayout. Sorting objects into rows or columns is made simpler by these classes.

Example of using QHBoxLayout:

Listing 1.2: Example of using `QHBoxLayout` to arrange three buttons horizontally in a Qt application

```
#include <QApplication>
#include <QPushButton>
#include <QHBoxLayout> // don't forget!

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget window;

    // creation of button widgets
    QPushButton *button1 = new QPushButton("One");
    QPushButton *button2 = new QPushButton("Two");
    QPushButton *button3 = new QPushButton("Three");

    // creation of a horizontal layout
    QHBoxLayout *layout = new QHBoxLayout;

    // placement of widgets in the horizontal layout
    layout->addWidget(button1);
    layout->addWidget(button2);
    layout->addWidget(button3);

    // placement of the layout in the window
    .setLayout(layout);

    window.show();
    return app.exec();
}
```

In this example, QHBoxLayout is used to build and arrange three buttons horizontally.

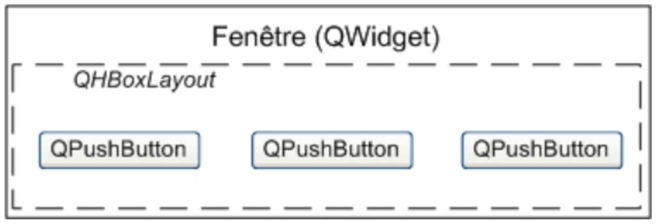


Figure 1.3: Example of QHBoxLayout with QPushButtons in a QWidget Window.

The Grid Layout

More organizational freedom is possible by using the QGridLayout class to organize elements in a grid of rows and columns.

Example of using QGridLayout:

Listing 1.3: Qt Application with QGridLayout

```
#include <QApplication>
#include <QPushButton>
#include <QGridLayout>
#include <QWidget>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWidget window;
    // Create buttons
    QPushButton *button1 = new QPushButton("One");
    QPushButton *button2 = new QPushButton("Two");
    QPushButton *button3 = new QPushButton("Three");
    QPushButton *button4 = new QPushButton("Four");
    QPushButton *button5 = new QPushButton("Five");
    // Create grid layout
    QGridLayout *layout = new QGridLayout;
    layout->addWidget(button1, 0, 0);

    layout->addWidget(button2, 0, 1);
    layout->addWidget(button3, 1, 0, 1, 2);
    layout->addWidget(button4, 2, 0);
    layout->addWidget(button5, 2, 1);

    // Set the layout for the window
    .setLayout(layout);
    window.show();

    return app.exec();
}
```

In this example, QGridLayout is used to organize the buttons in a grid, with each button's row and column defined.

The form layout

Data entry forms with elements grouped in rows with input fields and labels are made with the QFormLayout class.

Example of using QFormLayout:

Listing 1.4: Qt Application with QFormLayou

```
#include <QApplication>
#include <QLineEdit>
#include <QFormLayout>
```

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget window;

    QLineEdit *firstName = new QLineEdit;
    QLineEdit *lastName = new QLineEdit;
    QLineEdit *age = new QLineEdit;

    QFormLayout *layout = new QFormLayout;
    layout->addRow("First &Name:", firstName);
    layout->addRow("Last &Name:", lastName);
    layout->addRow("A&ge:", age);

    window.setLayout(layout);
    window.show();

    return app.exec();
}
```

Combine layouts

Complex user interfaces can be created by combining several layout types.

Example of combining QVBoxLayout and QFormLayout:

Listing 1.5: Example of using QFormLayout and QVBoxLayout to create a form with input fields and an Exit button

```
#include <QApplication>
#include <QPushButton>
#include <QLineEdit>
#include <QFormLayout>
#include <QVBoxLayout>
#include <QWidget>
```

```
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWidget window;

    // Create input fields
    QLineEdit *firstName = new QLineEdit;
    QLineEdit *lastName = new QLineEdit;
    QLineEdit *age = new QLineEdit;

    // Create the form layout
    QFormLayout *formLayout = new QFormLayout;
    formLayout->addRow("First Name:", firstName);
    formLayout->addRow("Last Name:", lastName);
    formLayout->addRow("Age:", age);

    // Create an exit button
    QPushButton *exitButton = new QPushButton("Exit");

    // Connect the exit button to close the application
    QObject::connect(exitButton, SIGNAL(clicked()), &app, SLOT(quit()));

    // Create the vertical layout
    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout->addLayout(formLayout);
    mainLayout->addWidget(exitButton);

    // Set the layout for the window
    window.setLayout(mainLayout);
    window.show();

    // Enter the application main loop
    return app.exec();
}
```

This example combines QFormLayout and QVBoxLayout to produce a user interface with an exit button and a data entry form.

"Layouts in Qt are crucial for creating flexible user interfaces that adjust to the available space and provide a clean, organized look.” [3].

For further information and examples, the official Qt documentation and tutorials can be consulted [3].

1.4.2 Memory Management in Object Hierarchies

Memory management is a crucial component of C++ that guarantees the effectiveness and stability of applications.

However, Qt can take over some of the work of memory management, as follows. Objects of classes that are derived from the QObject class can be arranged to form a tree structure: Objects may possess ’child’ objects. If such an object is deleted, then Qt automatically deletes all the child objects, and these ’children’ in turn delete their own ’offspring’, etc [1].

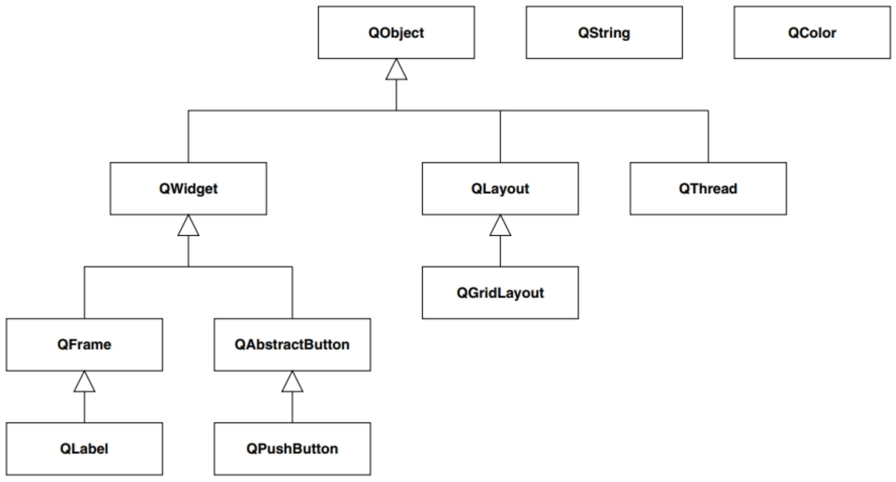


Figure 1.4: Qt Class Hierarchy Diagram.
[1]

Children must be created on the heap using the new keyword, which guarantees that child objects are a part of the object tree and are automatically deleted when the parent is deleted. This allows Qt to ensure proper memory management by automatically deleting all child objects associated with an object tree when the root object is deleted.

Not putting objects on the heap (i.e., not allocating them with new objects) is a common beginner's mistake: for example, tools that are created only on the stack, In a constructor class, it is deleted by the compiler after processing is finished. Although the app creates the widget briefly, it is never visible to eye [1].

For instance, a QVBoxLayout object is specified as a pointer and assigned using new when it is formed as a child of a QWidget object:

```
QVBoxLayout* mainLayout = new QVBoxLayout(this);
```

MainLayout in this example is a pointer to a QVBoxLayout object. The constructor receives this (the parent object of type QWidget) and adds the QVBoxLayout to the parent's child list. This guarantees that all QVBoxLayout objects involved are erased when all parents are removed.

Consequently, child objects should always be created on the heap using new and set as children of the parent object in order to guarantee appropriate memory management and prevent any issues.

1.5 Signals and Slots

The signals and slots mechanism is fundamental to Qt programming. It enables the application programmer to Bind objects together without the objects knowing anything about each other. We have already connected some signals and slots, declared our signals and slots, implemented our slots, and emitted our signals. Let's take a moment to look at the mechanism more closely [2].

Slots are nearly the same as regular member functions in C++. Like any other C++ member function, they can be directly invoked, be virtual, be overloaded, be public, protected, or private, and have any kind of parameter. The distinction is that a slot can be linked to a signal as well. In this scenario, it is automatically triggered whenever the signal is sent out.

The connect() statement looks like this:

connect(sender, SIGNAL(signal), receiver, SLOT(slot)); where signal and slot are function signatures and sender and receiver are pointers to QObjects.

Without names for the parameters. In essence, the textbf SIGNAL() and **SLOT()** macros turn their input into a string.

- **One signal can be connected to many slots: [8]**

Listing 1.6: One signal connected to multiple slot

```
connect(slider, SIGNAL(valueChanged(int)),
        spinBox, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)),
        this, SLOT(updateStatusBarIndicator(int)));
```

The slots are called in an arbitrary order, one after the other, when the signal is released.

- **Many signals can be connected to the same slot: [8]**

Listing 1.7: Many signals can be connected to the same slot

```
connect(lcd, SIGNAL(overflow()),
        this, SLOT(handleMathError()));
connect(calculator, SIGNAL(divisionByZero()),
        this, SLOT(handleMathError()));
```

The slot is called when either signal is released.

- **A signal can be connected to another signal:**[8]

Listing 1.8: A signal can be connected to another signal

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
        this, SIGNAL(updateRecord(const QString &)));
```

The second signal is released simultaneously with the first. Other than that, signal-slot connections and signal-signal connections are identical.

- **Connections can be removed:**

In Qt, connections between signals and slots can be broken by calling disconnect() to end a particular signal-slot connection[8].

Listing 1.9: Disconnecting a signal from a slot

```
disconnect(lcd, SIGNAL(overflow()),
           this, SLOT(handleMathError()));
```

This is rarely needed, because Qt automatically removes all connections involving an object when that object is deleted

A signal must have the same parameter types in the same order to be connected to a slot or another signal: [8]

Listing 1.10: Signal-slot connection with matching parameter types

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),
        this, SLOT(processReply(int, const QString &)));
```

In the rare event that a signal contains more parameters than the slot to which it is attached, the extra parameters are just disregarded: [8]

Listing 1.11: Signal with extra parameters ignored by slot

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),
        this, SLOT(checkErrorCode(int)));
```

If the application is created in debug mode,[caption=Signal with extra parameters ignored by slot] Qt will raise an error at runtime if the parameter types are incompatible, the signal is absent, or the slot is not there. Likewise, if parameter names appear in the signal or slot signatures, Qt will issue We have just utilized slots and signals with widgets thus far. However, QObject is where the mechanism itself is implemented.

And extends beyond GUI programming. Any subclass of QObject can use the mechanism: a warning [8].

Listing 1.12: Qt QObject class with signal and slot

```
class Employee: public QObject
{
 Q_OBJECT
public:
 Employee() { mySalary = 0; }
 int salary() const { return mySalary; }
Public slots:
 void setSalary(int newSalary);
Signals:
 void salaryChanged(int newSalary);
Private:
 int mySalary;
};
```

1.6 Core Network Tools in Qt

A comprehensive collection of tools for managing network operations is offered by the **Qt** framework. **QtNetwork** is one of the most potent modules, enabling developers to deal with **TCP** and **UDP** protocols, execute network operations like client-server communication, and interface with internet protocols like **HTTP** and **FTP**. The fundamental tools that Qt offers for network development will be covered in this part, along with several real-world applications.

1.6.1 QTcpSocket and QTcpServer

For handling **TCP** connections, the classes **QTcpSocket** and **QTcpServer** are essential in Qt.

- Using the **TCP** protocol, a client and a server can connect using textbfQTcp-Socket.
- To configure a server that waits for incoming connections from clients, use textbfQTcpServer.

These tools are essential for building networked applications that require reliable, persistent communication, such as chat applications or remote device control systems [9].

Example Using QTcpSocket

A connection between the client and the server can be made using **QTcpSocket**. The example that follows demonstrates how to transmit a message using **TCP** from the client to the server.

Listing 1.13: Basic example of connecting a QTcpSocket and sending data

```
QTcpSocket *socket = new QTcpSocket(this);
socket->connectToHost("localhost", 1234);
if (socket->waitForConnected(5000)) {
    socket->write("Hello, Server!");
    socket->flush();
    socket->waitForBytesWritten();
}
```

In this example, the client delivers a message to the server after connecting to it at `localhost` over port `1234`.

1.6.2 QNetworkAccessManager

If you want to interact with internet protocols like **HTTP** or **FTP**, **QNetworkAccessManager** is the tool you need. This class enables sending and receiving data over the internet, making it easy to interact with APIs or upload files [9].

Example Using QNetworkAccessManager

Here is a basic example of sending an HTTP request using **QNetworkAccessManager**:

Listing 1.14: Using QNetworkAccessManager to perform a simple HTTP GET request

```
QNetworkAccessManager *manager = new QNetworkAccessManager(this);
QNetworkRequest request;
request.setUrl(QUrl("http://example.com"));
QNetworkReply *reply = manager->get(request);
connect(reply, &QNetworkReply::finished, this, [reply]() {
    QByteArray response = reply->readAll();
    qDebug() << response;
});
```

This code example demonstrates how to send an HTTP GET request using **QNetworkAccessManager** [9].

In this example, the client reads and displays the response after sending a GET call to `http://example.com`.

1.7 File Management and Database Handling Using Qt

While **QtSql** gives an interface to interact with databases like **SQLite**, the **QtCore** library offers a variety of tools for handling local files in **Qt** applications. This part will cover working with **SQLite** through **QtSql** and storing data using **QFile**.

1.7.1 Saving Data with QFile

textbf QFile provides a mechanism for reading from and writing to local files. It can be used to handle both text and binary files. **QFile** gives you access to files through **QIODevice**, making it easy and flexible to work with data in local files [3].

Example of Using QFile to Write Data to a File

Listing 1.15: Writing text to a file using QFile and QTextStream

```
#include <QFile>
#include <QTextStream>
```

```
QFile file("data.txt");
if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
    QTextStream out(&file);
    out << "Hello, world!" << endl;
    file.close();
}
```

In this example, **QTextStream** is used to write the text ”Hello, world!” to the file after opening the file `data.txt` for writing. Following the operation, the file is closed.

Example of Using QFile to Read Data from a File

Listing 1.16: Reading text from a file using QFile and QTextStream

```
#include <QFile>
#include <QTextStream>
#include <QDebug>

QFile file("data.txt");
if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
    QTextStream in(&file);
    QString content = in.readAll();
    qDebug() << content;
    file.close();
}
```

In this example, **QTextStream** is used to read the complete contents of the file `data.txt` after it has been opened for re

1.7.2 Using SQLite with QtSql

If a database is required, **Qt** provides direct support for **SQLite** through the **QtSql** module. **SQLite** is particularly useful for applications that do not need a complex database system, such as desktop or mobile apps. For debugging or printing content, the `qDebug()` function can be used.

Example of Using QtSql with SQLite

First, ensure that your project contains the **QtSql** module:

Listing 1.17: Including Qt SQL module headers

```
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QDebug>
```

After that, you can run SQL queries in an open **SQLite** database:

Listing 1.18: Basic SQLite database operations using QSqlDatabase and QSqlQuery

```

QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName("example.db");

if (db.open()) {
QSqlQuery query;
query.exec(
    "CREATE TABLE IF NOT EXISTS users("
    "id INTEGER PRIMARY KEY, "
    "name TEXT)"
);



 query.exec("INSERT INTO users (name) VALUES ('Alice')");
 query.exec("INSERT INTO users (name) VALUES ('Bob')");

 // Read data
 query.exec("SELECT * FROM users");
 while (query.next()) {
 int id = query.value(0).toInt();
 QString name = query.value(1).toString();
 qDebug() << "ID:" << id << "Name:" << name;
    }
    db.close();
} else {
    qDebug() << "Database not open!";
}
```

This example involves creating a **SQLite** database called `example.db`.

If it doesn't exist, a table called `users` is made.

Data (person names) is entered into the table.

qDebug() is used to read data from the table and print it.

1.8 Improving GUI Design with Qt

In this section, we will examine **Qt Designer** and **Qt Style Sheets (QSS)**, two crucial tools that Qt offers to enhance GUI design.

1.8.1 Qt Designer

Qt Designer is a powerful graphical tool that enables developers to design and build user interfaces visually. It provides a drag-and-drop interface to place widgets (buttons, labels, input fields, etc.) onto a form, making the GUI design process much easier and faster. Qt Designer generates an XML-based '.ui' file, which can later be loaded and used within the application. This tool simplifies the creation of professional and responsive UIs without the need to manually write complex code for each widget's layout [10].

1.8.2 Using Qt Style Sheets (QSS)

Qt Style Sheets (QSS) are a powerful feature in Qt that allows developers to apply custom styles to the widgets in their application, similar to how CSS works in web development. With QSS, you can define the look and feel of your application by specifying properties such as color, fonts, borders, padding, and more. QSS provides flexibility and control over the visual appearance of the interface [10].

Example of QSS Usage

The example below demonstrates how to use QSS to decorate a button:

Listing 1.19: Styling QPushButton using Qt Style Sheets (QSS)

```
QPushButton {
    background-color: #4CAF50;
    color: white;
    font-size: 16px;
    border-radius: 5px;
}
QPushButton: hover {
    background-color: #45a049;
}
```

In this example, the white text on a green button background changes slightly when the button is hovered over. QSS facilitates the creation of unique themes and designs for your application by enabling you to apply comparable styles to additional widgets, including labels, text boxes, and menus.

1.9 Dynamic Dialogs

Dialogs are distinct windows in Qt that are used to display specific information or gather user input; dynamic dialogs are those that are created or altered dynamically at runtime rather than being pre-designed in Qt Designer.

1.9.1 Creating a Simple Dynamic Dialog

The `QDialog` class can be used to construct a dynamic dialog. Here's an example:[8]

Listing 1.20: Creating a dynamic dialog with QLabel and QPushButton in Qt

```
#include <QApplication>
#include <QDialog>
#include <QVBoxLayout>
#include <QLabel>
#include <QPushButton>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QDialog dialog;
    dialog.setWindowTitle("Dynamic Dialog");
    QVBoxLayout layout(&dialog);
    QLabel label("Hello! This is a dynamic dialog.", &dialog);
    QPushButton closeButton("Close", &dialog);
    QObject::connect(&closeButton, &QPushButton::clicked, &dialog, &QDialo
    layout.addWidget(&label);
    layout.addWidget(&closeButton);
    dialog.exec();
    return 0;
}
```

This code uses the **QDialog** class to generate a basic dynamic dialog. A **QDialog** object is initialized, its title is set, and a vertical layout (**QVBoxLayout**) is added. A **QPushButton** has been added to shut the dialog when it is clicked, and a **QLabel** is utilized to show a message. The dialog can close when the button is pushed because the **QObject::connect** function connects the button click event to the **accept()** method.

1.9.2 Using QInputDialog for User Input

You can use `QInputDialog` for basic user input collection:[8]

Listing 1.21: Getting user text input using QInputDialog

```
QString name = QInputDialog::getText(nullptr, "Enter Name", "Your name:");
```

1.9.3 Advanced Dynamic Dialogs

Dynamic dialogs can incorporate many input fields, validation, and real-time modifications in more intricate applications. An example of a more sophisticated dialog that gathers user information and dynamically adjusts labels is shown below: [8]

Listing 1.22: Creating a custom dialog with QLabel, QLineEdit, and QPushButton in Qt

```
#include <QApplication>
```

```
#include <QDialog>
#include <QVBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>

class CustomDialog: public QDialog {
public:
    CustomDialog() {
        setWindowTitle("Advanced Dynamic Dialog");
        QVBoxLayout *layout = new QVBoxLayout(this);
        QLabel *label = new QLabel("Enter your name:", this);
        QLineEdit *lineEdit = new QLineEdit(this);
        QPushButton *okButton = new QPushButton("OK", this);

        layout->addWidget(label);
        layout->addWidget(lineEdit);
        layout->addWidget(okButton);

        connect(okButton, &QPushButton::clicked, this, &QDialog::accept);
    }
};

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    CustomDialog dialog;
    dialog.exec();
    return 0;
}
```

The class **CustomDialog** defined by this code is an inheritor of **QDialog**. Within the constructor, the widgets are arranged vertically using a **QVBoxLayout**. The dialogue includes: The user is prompted to input their name by

- A **QLabel** prompting the user to enter their name.

- A **QLineEdit** where the user can type their name.

- A **QPushButton** labeled ”OK” to close the dialog.

The dialog is closed when the ”OK” button is hit since the **connect** function connects it to the **accept()** procedure.

1.9.4 Benefits of Dynamic Dialogs

Using dynamic dialogs in Qt provides several advantages:

- **Flexibility:** Depending on the runtime circumstances, they enable customisation.
- **Reusability: ** It is possible to reuse code-based dialogs in other application sections.
- **User Interaction:** They make it possible for interactive apps with dynamic dialogs.
- **Reduced UI Overhead:** There's no need to use Qt Designer to pre-design every conceivable dialog.

1.10 Conclusion

In this chapter, we went over some of the core and crucial elements provided by the Qt framework for creating user interfaces, communicating across networks, and controlling memory. We also looked at how to improve the user experience with programs like `Qt Designer` and `Qt Style Sheets (QSS)`. Along with learning how to handle signals and slots, which offer a reliable means for objects to interact without direct coupling, we also looked at useful techniques like `Layouts`, which aid in the variable organization and arrangement of items within the user interface. We also covered memory management in object hierarchies and how to use Qt's features to make this process easier.

Important networking tools like `QTcpSocket` and `QTcpServer` for building TCP-based networked applications and tools for connecting to the internet like `QNetworkAccessManager` were also addressed. We may create intricate and adaptable apps in networked contexts by employing these techniques.

In terms of data management, we discovered how to use `QFile` to read and write data to local files and `QtSql` to communicate with databases such as SQLite, which facilitates developers' ability to store and retrieve data effectively.

Chapter 2

Raspberry Pi

2.1 Introduction

Microcomputing has improved significantly in the last few decades, enabling the use of compact and reasonably priced equipment to carry out cutting-edge technological initiatives. Because of its many uses and user-friendliness, the Raspberry Pi is one of the most well-known (*Single Board Computers - SBC*) that has transformed the electronics and computer industries. Initially created to aid in the teaching of computer science, it swiftly evolved into a potent instrument utilized in a number of domains, including system automation, artificial intelligence, and the Internet of Things (*IoT*).

Raspberry Pi is a mini-computer characterized by its compact size, low cost, and flexibility in working with various software and hardware applications. It features an *ARM* processor, *RAM*, and multiple input/output ports such as *USB*, *HDMI*, and *GPIO*, which facilitate interaction with sensors, motors, and other systems. Due to these capabilities, *Raspberry Pi* has become an ideal choice for engineers, developers, and hobbyists alike [11].

Raspberry Pi is significant because it can provide a full computing environment with minimal power consumption, which makes it a perfect tool for creating smart systems and embedded applications. Users can choose the environment that best fits their needs thanks to its compatibility with many operating systems, including *Raspberry Pi OS, Ubuntu, and Windows IoT Core.*

The demand to create more effective and secure communication protocols is growing as wireless networking technologies improve at a rapid pace. With support for multiple communication technologies including *Wi-Fi, Bluetooth, and Zigbee*, the Raspberry Pi is an ideal platform for testing and refining *MAC* protocols in wireless networks. This device facilitates practical experiments to evaluate different protocol performances and analyze their behavior in various environments, contributing to the development of more reliable and efficient communication systems.

2.2 History of Raspberry Pi

The Raspberry Pi Foundation in the UK created the Raspberry Pi line of inexpensive single-board computers to advance computer science education and ensure that everyone has access to technology.

2.2.1 Origins and Development

Under the direction of Eben Upton, a group at the University of Cambridge came up with the concept for the Raspberry Pi in 2006. Making a computer that was inexpensive for students who wanted to study computer engineering and programming was the aim. The initial model, **Raspberry Pi 1 Model B**, was released in February 2012 and cost only about $35. It had an ARM11 processor that ran at 700 MHz and 512 MB of RAM. This model was an immediate success, with the initial batch of 10,000 units selling out rapidly [12].

2.2.2 Major Versions

Numerous enhanced Raspberry Pi models have been made available, such as:

- **Raspberry Pi Zero (2015)**: A smaller version priced at just $5 [13].
- **Raspberry Pi 2 (2015)**: 1 GB of RAM and a quad-core Cortex-A7 CPU.
- **Raspberry Pi 3 (2016)**:quad-core 64-bit processor with Bluetooth and Wi-Fi integrated.
- **Raspberry Pi 4 (2019)**: Support for 4K screens, USB 3.0 ports, and up to 8 GB of RAM.

2.3 Impact and Future

With more than 50 million sales as of 2023, the Raspberry Pi is among the best-selling computers globally. It is extensively utilized in engineering, education, and industrial applications such as home automation, robotics, and the Internet of Things. The Raspberry Pi Foundation continues to innovate by developing new hardware and software to support its mission of democratizing computing, as well as providing free educational resources to help individuals learn programming and electronics [14].

2.4 Components of Raspberry Pi

This section outlines the key components of the Raspberry Pi, including its processor, input/output ports, GPIO pins, storage, and networking capabilities [14].

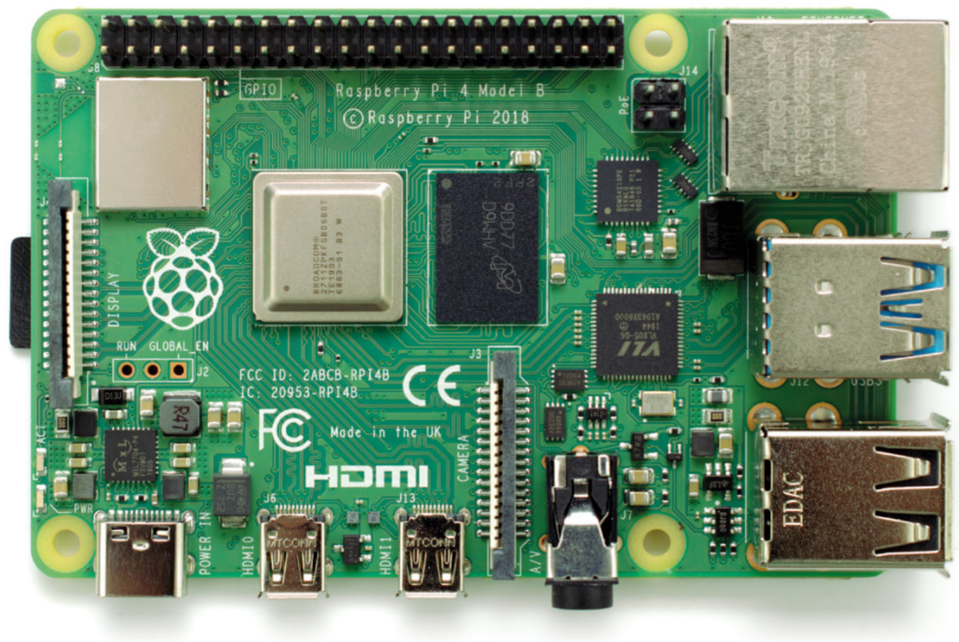


Figure 2.1: The Raspberry Pi 4 Model B
[15]

2.4.1 Processor (ARM) and RAM

An ARM-based processor, which is renowned for its performance and energy economy, powers the Raspberry Pi. Mobile devices and embedded systems frequently employ the ARM architecture. The Raspberry Pi has different amounts of RAM depending on the model; the most recent variants have 8GB of RAM, while older models have 512 MB. This combination allows the Raspberry Pi to handle tasks such as web browsing, media playback, and even lightweight gaming [13].

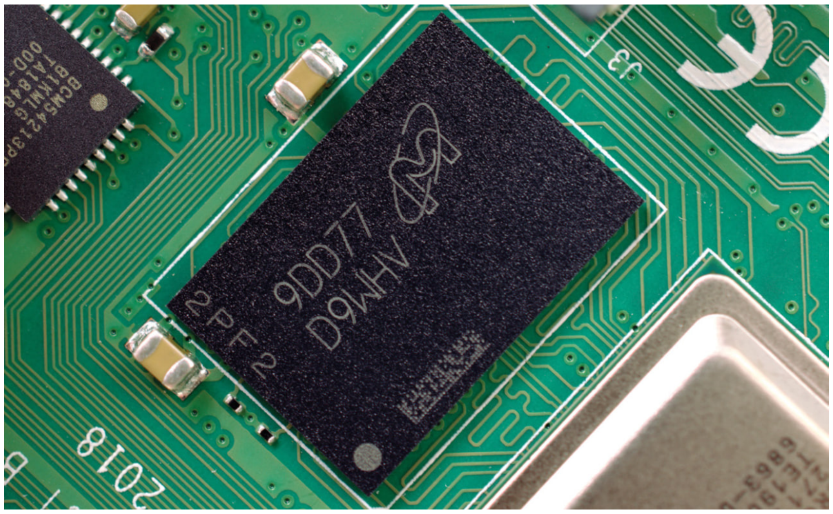


Figure 2.2: The Raspberry Pi's random access memory (RAM) [15]

2.4.2 Input/Output Ports (USB, HDMI, Ethernet, etc.)

Because of its many input/output connections, the Raspberry Pi can be used for a wide range of purposes. Among these ports are:

- **USB Ports**: used to connect external storage devices, keyboards, and mice, among other accessories.

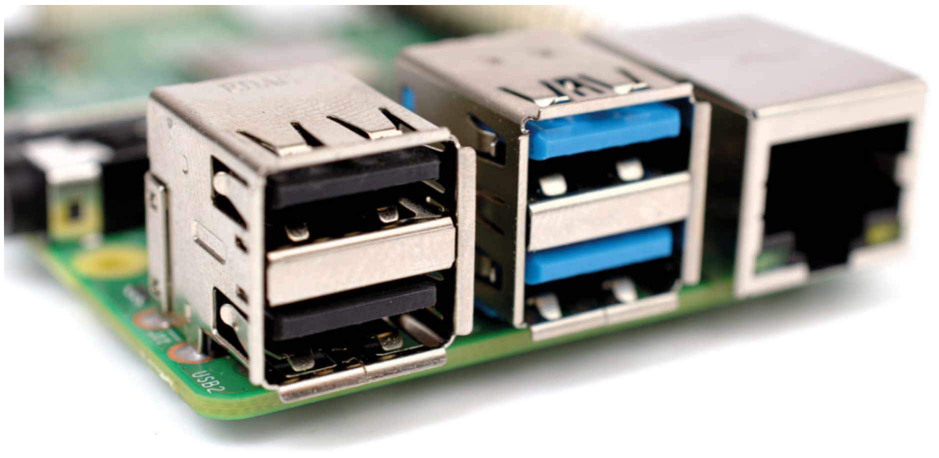


Figure 2.3: The Raspberry Pi's USB ports
[15]

- **HDMI Port**: enables the Raspberry Pi to be connected to TVs or displays by supporting audio and video output.

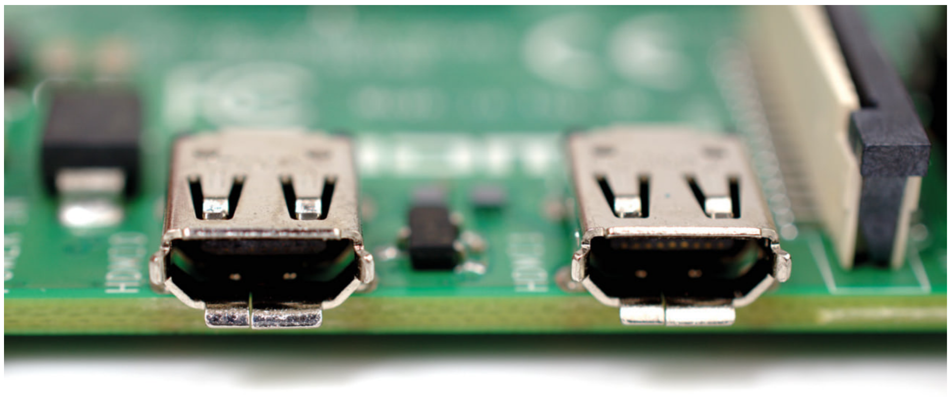


Figure 2.4: The Raspberry Pi's micro-HDMI ports
[15]

- **Ethernet Port**: provides dependable internet connection through wired network connectivity.

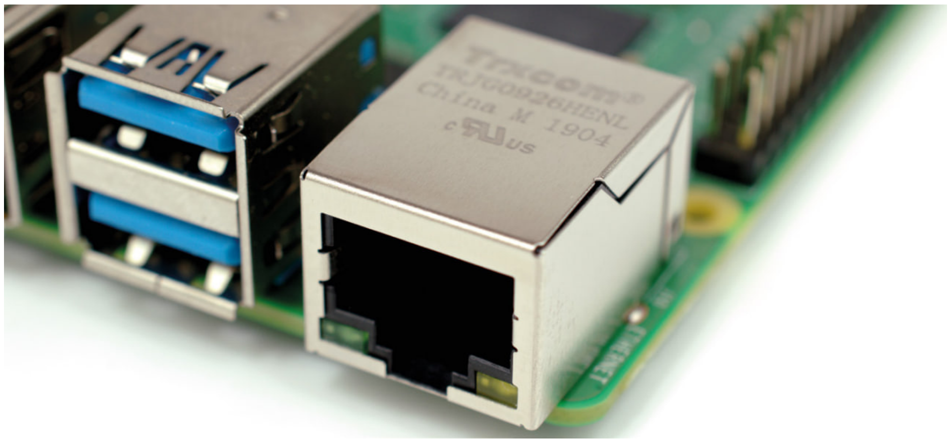


Figure 2.5: The Raspberry Pi's Ethernet port
[15]

- **Audio Jack**: permits audio output to external speakers or headphones.

These ports enhance the usability of the Raspberry Pi for various applications, from simple desktop computing to more advanced automation projects [16].

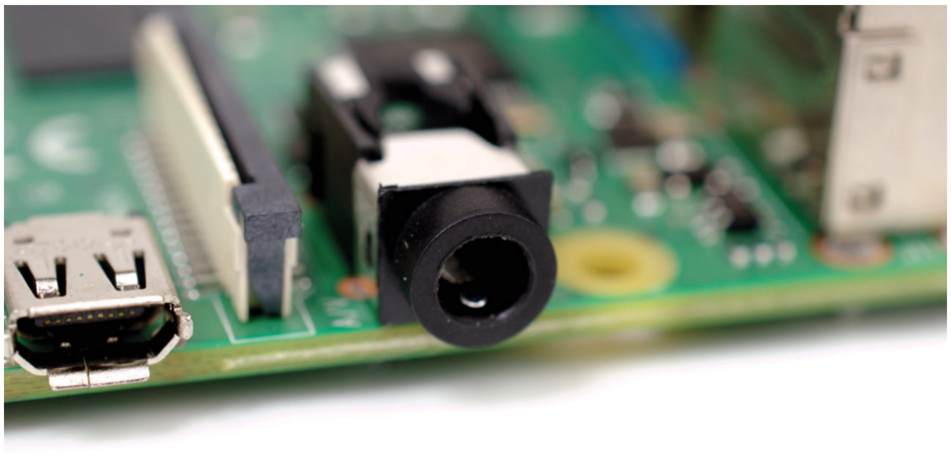


Figure 2.6: The Raspberry Pi's AV jack
[15]

2.4.3 GPIO Pins and Their Uses

The Raspberry Pi's General Purpose Input/Output (GPIO) pins are among its most potent features. The Raspberry Pi can communicate with external hardware elements like sensors, LEDs, and motors thanks to these ports. It is possible to program the GPIO pins to:

- Read input signals (e.g., from a button or sensor).
- Control output devices (e.g., turning an LED on or off).

This functionality makes the Raspberry Pi an excellent choice for electronics projects, robotics, and IoT applications [12].

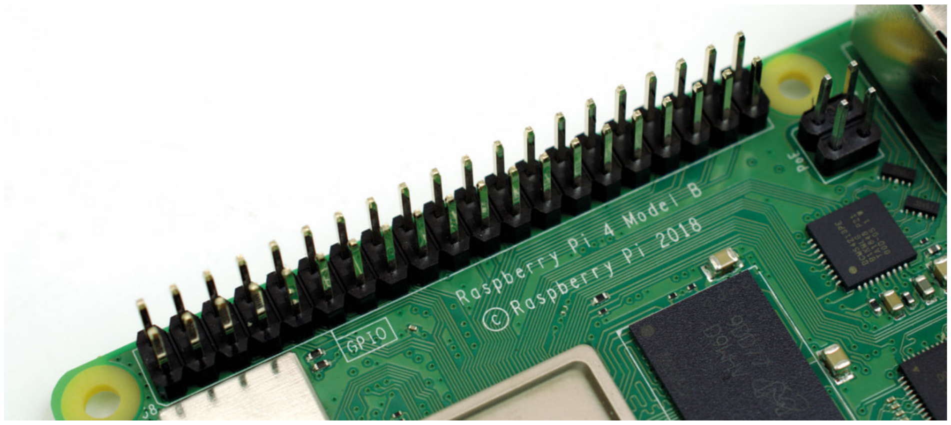


Figure 2.7: : The Raspberry Pi's GPIO header
[15]

2.4.4 Storage (microSD Card)

A microSD card serves as the main storage device for the Raspberry Pi. The operating system (such as Raspberry Pi OS) and all user files are stored on the microSD card.

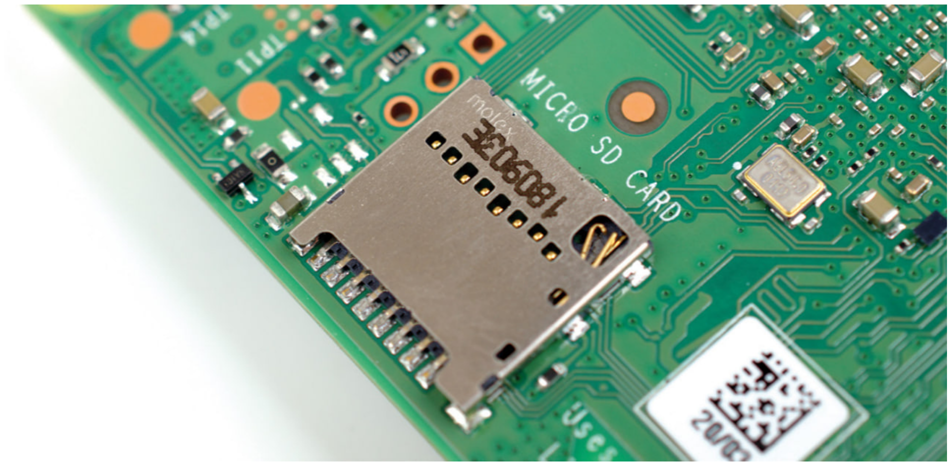


Figure 2.8: The Raspberry Pi's microSD card connector
[15]

2.4.5 Networking Support (Wi-Fi, Bluetooth)

Wireless connectivity is made possible by the built-in Bluetooth and Wi-Fi in the majority of contemporary Raspberry Pi models. Important characteristics include:

- **Wi-Fi**: allows for dependable and quick internet connectivity by supporting both the 2.4GHz and 5GHz bands.
- **Bluetooth**: permits interaction with Internet of Things devices and peripherals, including speakers, keyboards, and mice.

These networking features make the Raspberry Pi ideal for projects requiring wireless communication and internet connectivity [16].

2.5 Supported Operating Systems

This section explores various OS options, including Raspberry Pi OS and other alternatives.

2.5.1 Raspberry Pi OS (formerly Raspbian)

The official operating system created and tailored for the Raspberry Pi is called Raspberry Pi OS. Based on Debian Linux, it includes a user-friendly desktop environment and a set of pre-installed software tools for programming, education, and general computing [14]. There are three main versions of Raspberry Pi OS:

- **Raspberry Pi OS with Desktop**: offers all necessary apps in a comprehensive graphical user interface (GUI).
- **Raspberry Pi OS Lite**: A version for headless server applications that runs on a command line.
- **Raspberry Pi OS with Recommended Software**: Includes additional tools such as LibreOffice and Scratch for educational use [16].

2.5.2 Other Operating Systems

In addition to Raspberry Pi OS, the Raspberry Pi is compatible with many alternative operating systems that serve various purposes:

- **Ubuntu**: a complete Linux distribution for desktop and server use.
- **Windows IoT Core**: Microsoft OS intended for Internet of Things and embedded applications.
- **OSMC (Open Source Media Center)**: It turns the Raspberry Pi into a video streaming center and is based on Kodi.
- **RetroPie**: An operating system designed to simulate vintage video game systems.
- **DietPi**: A lightweight Linux-based OS optimized for performance and minimal resource consumption [13].

These operating systems increase the Raspberry Pi's usefulness and qualify it for use in industrial, educational, and recreational settings.

2.5.3 How to Install an Operating System on Raspberry Pi

The instructions below outline how to install an operating system on a Raspberry Pi:

1. **Download the OS Image**: From its official page, select the OS of your choice.
2. **Flash the OS to a microSD Card**: To write the operating system image onto a microSD card, use programs like **Raspberry Pi Imager** or **balenaEtcher**.
3. **Insert the microSD Card**: Insert the card into the microSD card port on the Raspberry Pi.
4. **Power On the Raspberry Pi**: Let the gadget boot up after connecting it to a power supply.
5. **Complete the Setup**: Update software, adjust network settings, and alter the environment as necessary. [14].

These instructions allow users to configure and personalize their Raspberry Pi for a variety of uses, such as embedded system development and general computing.

2.6 Comparison Between Raspberry Pi and Similar Devices

The Raspberry Pi has gained significant popularity due to its versatility. However, other alternatives such as **Arduino Uno**, **BeagleBone Black**, **NVIDIA Jetson Nano**, and **Odroid XU4** provide distinct features that cater to different use cases [17]. This section provides a detailed comparison to help users determine the best SBC for their needs.

2.6.1 Comparison of Devices

Table 2.1 presents a comparison of key features between Raspberry Pi and similar devices [18].

Table 2.1: Extended Comparison of Raspberry Pi and Similar Devices

Feature	**Raspberry Pi 4**	**Arduino Uno**	**BeagleBone Black**	**Jetson Nano**	**Odroid XU4**
Hardware Specifications					
Processor	Cortex-A72 Quad-core	AT-mega328P (8-bit)	Cortex-A8 (1 GHz)	Cortex-A57 Quad-core	Cortex-A15 Octa-core
Clock Speed	1.5 GHz	16 MHz	1.0 GHz	1.43 GHz	2.0 GHz
RAM	2/4/8 GB LPDDR4	2 KB SRAM	512 MB DDR3	4 GB LPDDR4	2 GB DDR3
Storage	microSD	Flash memory	microS-D/eMMC	microSD	microS-D/eMMC
GPIO Pins	40	14	92	40	30
Connectivity and Power					
Wi-Fi	Optional	No	No	No	No
Ethernet	Yes	No	Yes	Yes	Yes
USB Ports	4×USB 3.0	1×USB 2.0	1×USB 2.0	4×USB 3.0	2×USB 3.0
Power Input	5V USB-C	5V DC (Barrel)	5V DC	5V DC	5V DC
Power Usage	∼5W	∼0.5W	∼2.5W	∼10W	∼5W
Multimedia and Use Cases					
GPU	VideoCore VI	None	PowerVR SGX530	NVIDIA Maxwell 128 cores	Mali-T628 MP6
Video Output	HDMI	None	micro-HDMI	HDMI	HDMI
AI/ML Support	Basic	None	No	Yes	Moderate
Best Use Case	IoT, Multi-media	Prototyp-ing	Industrial Con-trol	AI/ML Projects	HPC

2.7 Device Analysis

2.7.1 Raspberry Pi 4

With a **quad-core processor, numerous RAM options, built-in WiFi connectivity**, and compatibility with **Linux-based OSs**, the Raspberry Pi 4 is one of the most popular SBCs. It is ideal for **IoT, automation, and educational projects** [16].

2.7.2 BeagleBone Black

BeagleBone Black is widely used in **industrial applications** and has **more GPIO pins** than Raspberry Pi. It also features **onboard eMMC storage**, which makes it more reliable for critical tasks.

2.7.3 NVIDIA Jetson Nano

Deep learning and artificial intelligence applications are the focus of Jetson Nano's design. It is perfect for **machine learning, image processing, and robotics** because it has **CUDA-enabled GPU cores**. However, it **consumes more power** than Raspberry Pi.

2.7.4 Odroid XU4

Designed for **high-performance computing**, the Odroid XU4's **octa-core processor and USB 3.0 support** are utilized for **gaming, media servers, and parallel processing tasks**. However, it has a **smaller community** compared to Raspberry Pi.

2.8 Applications of Raspberry Pi

2.8.1 Education: Teaching Programming and Electronics

The Raspberry Pi is widely used in educational settings to teach programming, electronics, and computer science. Its affordability and ease of use make it an excellent tool for students and educators [14]. Key applications include:

- **Programming**: instructing students in languages like Java, Python, and Scratch.
- **Electronics**: introducing GPIO (General Purpose Input/Output) pins, circuits, and sensors to kids.
- **STEM Projects**: promoting experiential learning via interactive projects, coding competitions, and robotics.

2.8.2 Internet of Things (IoT): Smart Home Systems

The Raspberry Pi is a popular choice for IoT projects due to its connectivity options and processing power. It is commonly used to build smart home systems, including [13]:

- **Home Automation**: remotely operating appliances, lights, and thermostats.
- **Security Systems**: use motion sensors and cameras to monitor residences.
- **Environmental Monitoring**: monitoring air quality, humidity, and temperature.

2.8.3 Robotics: Building Simple Robots

The Raspberry Pi is an excellent platform for robotics projects, offering both computational power and GPIO connectivity [16]. Applications include:

- **Educational Robots**: instructing students in robotics fundamentals.
- **Autonomous Robots**: creating autonomously navigating and task-performing robots.
- **Remote-Controlled Robots**:constructing robots with Bluetooth or Wi-Fi control.

2.8.4 Multimedia: Media Center

With the ability to broadcast and play multimedia content, the Raspberry Pi may be turned into a potent media center. Typical uses include:

- **Media Players**: Use programs such as Kodi or OSMC to play images, music, and movies.
- **Streaming Devices**: streaming media from websites such as Spotify, YouTube, and Netflix.
- **Home Theater Systems**: combining sound systems and TVs to create a full home theater setup.

2.9 Practical Projects Using Raspberry Pi

Raspberry Pi is a single-board computer that provides a cost-effective solution for many applications, including automation, IoT, and robotics. It supports various programming languages, particularly Python and C++, and integrates well with different hardware components such as sensors, cameras, and actuators [16]. This section discusses Seven practical projects that leverage Raspberry Pi's capabilities.

2.9.1 Project Implementations

Prayer Time Display Board

A prayer time display board is a simple yet useful project that fetches prayer timings from an online API, such as Aladhan API [19], and displays them on an LCD or LED matrix. This project requires:

- Raspberry Pi
- LCD screen or 7-segment display
- Internet connection for fetching real-time prayer times

Smart Surveillance System

Using OpenCV with Raspberry Pi, a motion detection system can be built that captures images or videos when movement is detected. The system can send real-time security alerts through a Telegram bot.

Home Weather Station

A weather station can be implemented using temperature and humidity sensors such as DHT11 or BME280. The gathered information can be shown on a web dashboard and entered into a database.

IoT Smart Home System

The goal of this project is to use the Raspberry Pi and MQTT to automate household appliances. Home Assistant can be configured to control lights, fans, and air conditioning remotely.

Attendance System Using RFID

By integrating an RFID module with a Raspberry Pi, an automated attendance system can be developed. It records check-ins and check-outs by scanning RFID cards.

Classic 2D Game Development

Using Pygame, developers can build classic 2D games with sprite animations and simple physics [20]. This project is excellent for teaching game development to beginners.

Fire and Gas Detection System

A safety monitoring system can be built using an MQ-2 gas sensor to detect harmful gases and fire hazards. The system can send email notifications or sound an alarm.

2.10 Challenges Facing Raspberry Pi

The Raspberry Pi has a number of practical and technological drawbacks despite its popularity. The main issues customers run into when utilizing the device are covered in this section.

2.10.1 Performance Limitations Compared to Traditional Computers

The Raspberry Pi cannot compete with traditional desktop or laptop computers, despite its amazing capabilities for a single-board computer. Some of its main drawbacks are as follows:

- **Processing Power**: The ARM-based processors that power the Raspberry Pi are far less powerful than the x86 processors that power the majority of personal computers. This limitation restricts its ability to handle computationally intensive tasks such as video editing, 3D rendering, or running multiple virtual machines.
- **RAM Constraints**: The latest Raspberry Pi models provide up to 8GB of RAM, which remains insufficient for memory-intensive applications like large-scale data analysis or high-end gaming [16].
- **Storage Speed**: The dependence on microSD cards as the primary storage medium results in considerably slower read/write speeds compared to SSDs or NVMe drives commonly used in modern computers [14].

Because of these limitations, the Raspberry Pi is less appropriate for applications requiring a lot of memory or computing power.

2.10.2 Cooling and Power Consumption Issues

Many thermal and power-related issues are brought on by the Raspberry Pi's small size and lack of active cooling solutions:

- **Thermal Throttling**: The Raspberry Pi may overheat when subjected to prolonged heavy workloads, which would cause **thermal throttling** to lower CPU performance to protect the hardware. This issue is particularly noticeable in demanding applications such as gaming or high-resolution media streaming.
- **External Cooling Solutions**: Users frequently turn to extra cooling devices, like fans or heatsinks, to reduce overheating. However, these accessories increase both the cost and complexity of the setup.
- **Power Consumption Considerations**: While the Raspberry Pi is more energy-efficient than traditional computers, its power demands may still pose challenges in **battery-powered projects** or applications requiring minimal energy consumption [21].

Stable functioning, especially in continuous-use scenarios, requires proper power planning and efficient heat management techniques.

2.10.3 Compatibility Challenges with Certain Software

Because of its **ARM architecture** and primary **Linux-based operating systems**, the Raspberry Pi may have problems with software compatibility.

- **Limited Support for Proprietary Software**: Many commercial software applications are designed exclusively for x86 processors and mainstream operating systems like **Windows and macOS**, making them incompatible with Raspberry Pi devices.
- **Peripheral Driver Limitations**: Some peripherals, such as specialized printers, scanners, or external hardware components, may not have adequate driver support for Raspberry Pi, limiting their usability in professional environments.
- **Performance Bottlenecks**: Even when software is compatible, the Raspberry Pi's modest hardware specifications may result in slow performance, instability, or **suboptimal execution of complex applications**.

The popularity of the Raspberry Pi in specialist sectors and enterprise-level applications may be limited by these compatibility issues.

2.11 Future and Expected Developments

The Raspberry Pi has revolutionized the field of low-cost computing. Several significant advancements in hardware, its function in AI and cloud computing, and its growth into new domains are expected as technology advances.

2.11.1 Future Developments in Raspberry Pi Processors

Significant improvements have been made to Raspberry Pi processors over the years. Future developments are anticipated to concentrate on:

- **Increased Performance**: Next-generation Raspberry Pi models will likely feature more powerful multi-core ARM processors with higher clock speeds and improved energy efficiency.

- **Integrated AI Accelerators**: To cater to AI and machine learning demands, future processors may incorporate **dedicated neural processing units (NPUs)** or **tensor processing units (TPUs)**, enabling faster execution of deep learning models.
- **Enhanced Graphics Capabilities**: Improved GPUs will support advanced **3D rendering**, better **gaming performance**, and enhanced **multimedia processing** [16].
- **Advanced Semiconductor Technologies**: The transition to **smaller fabrication processes** (e.g., 5nm or 3nm) will reduce power consumption while increasing processing power, making Raspberry Pi even more efficient [14].

The Raspberry Pi's standing as a competitive computer platform going forward will be strengthened by these improvements.

2.11.2 Role of Raspberry Pi in Artificial Intelligence and Cloud Computing

The Raspberry Pi is gradually being integrated into AI and cloud computing applications. Future advances in this arena are predicted to include:

- **Edge AI Applications**: Due to its low cost and small size, Raspberry Pi is ideal for **edge AI**, where data is processed locally instead of relying on cloud-based processing. This is particularly beneficial for **real-time IoT applications**.
- **Support for AI Frameworks**: Enhanced compatibility with frameworks like **TensorFlow Lite**, **PyTorch**, and **OpenCV** will allow developers to efficiently deploy AI models on Raspberry Pi devices.
- **Cloud Integration**: Improved **networking capabilities** and **software tools** will enable seamless integration with cloud platforms such as **AWS, Google Cloud, and Microsoft Azure**, facilitating hybrid cloud-edge computing solutions.
- **Distributed Computing and Clustering**: Future Raspberry Pi versions may support **more efficient clustering**, enabling users to build **low-cost supercomputers** for distributed computing, AI, and **big data processing**.

With these developments, Raspberry Pi will remain a vital instrument for cloud computing, **low-power intelligent systems**, and AI research.

2.11.3 Predictions for Its Use in New Fields

The Raspberry Pi is a promising contender for adoption in novel and creative industries due to its price and versatility. Potential future uses include:

- **Healthcare**: Raspberry Pi could revolutionize **telemedicine**, **health monitoring**, and **affordable diagnostic devices**, making healthcare more accessible.
- **Education**: As an affordable educational tool, Raspberry Pi will continue to play a vital role in **teaching programming, electronics, and AI** to students worldwide [22].
- **Smart Cities**: With its **low power consumption**, Raspberry Pi can be used in **traffic management systems, environmental monitoring, and energy-efficient smart lighting solutions** [21].
- **Space Exploration**: Due to its **compact size and energy efficiency**, Raspberry Pi is being considered for use in **small satellites and space research projects**.
- **Agriculture**: In **precision farming**, Raspberry Pi can be deployed for **soil monitoring, automated irrigation, and crop management**, improving agricultural efficiency and sustainability.

These patterns suggest that Raspberry Pi will keep spurring innovation in a variety of fields.

2.12 Socket Communication in Raspberry Pi: Concepts, Implementation, and Applications

Socket communication is a fundamental technology that enables networked devices to send and receive data [23]. Raspberry Pi, with its built-in networking capabilities, supports both wired and wireless connections, making it ideal for developing IoT applications and client-server models [16].

2.12.1 Concepts of Socket Communication

Over a network, sockets offer a way for processes to communicate with one another. The Raspberry Pi is compatible with a variety of socket types, including:

- **TCP Sockets**: utilized for dependable, relationship-focused communication.
- **UDP Sockets**:Ideal for quick, connectionless data transmission.
- **Unix Domain Sockets**: used on the same system to facilitate communication between processes.

These socket types enable Raspberry Pi to function as both a server and a client in networked applications [21].

2.12.2 Implementation of Socket Programming in Raspberry Pi

Python can be used for Raspberry Pi socket programming. A basic TCP server is shown in the example below:

Listing 2.1: Basic TCP server in Python listening and responding to clients

```
import socket

HOST = '0.0.0.0'
PORT = 65432

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.bind((HOST, PORT))
server_socket.listen(5)

print("Server is listening...")
```

```
While True:
    client_socket, client_address = server_socket.accept()
    print(f"Connected with {client_address}")
    data = client_socket.recv(1024).decode('utf-8')
    print(f"Received: {data}")
    client_socket.send("Hello from Raspberry Pi!".encode('utf-8'))
    client_socket.close()
```

A TCP server that receives data from clients, listens for incoming connections, and responds is set up using this script.

Likewise, the following is an implementation of a basic TCP client:

Listing 2.2: Basic TCP client in Python connecting and communicating with server

```
import socket

HOST = '192.168.1.100'  # Server IP
PORT = 65432

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

client_socket.send("Hello, Server!".encode('utf-8'))
response = client_socket.recv(1024).decode('utf-8')
print(f"Received: {response}")

client_socket.close()
```

After establishing a connection with the server, this client sends a message and watches for a reply.

2.12.3 Applications of Socket Communication in Raspberry Pi

The Raspberry Pi's versatility in socket connectivity makes it suitable for a wide range of uses:

- **IoT Devices**: Through connectors, the Raspberry Pi can gather and send sensor data.

- **Home Automation**: It can use network communication to remotely operate smart gadgets.
- **Remote Access**: The Raspberry Pi may be accessed and controlled from several locations.
- **Web Servers**: Web applications can be served by a Raspberry Pi with the help of lightweight frameworks.

These applications highlight the significance of socket programming in modern technology.

2.13 Conclusion

The Raspberry Pi has transformed the computing and electronics industries, increasing accessibility to technology and encouraging creativity in both individuals and companies. Since its launch in 2012, it has become a fundamental tool in education, allowing students worldwide to learn programming, electronics, and computer science in a hands-on and cost-effective manner [14]. Additionally, it has driven innovation in fields such as the Internet of Things (IoT), robotics, artificial intelligence (AI), and multimedia [12].

Despite its many successes, the Raspberry Pi faces challenges such as performance limitations, cooling issues, software compatibility, and power consumption, which may restrict its use in certain applications. However, its advantages, including affordability, flexibility, and ease of use, make it a preferred choice. Furthermore, ongoing advancements in hardware, such as more powerful processors and integrated AI accelerators, are enhancing its capabilities and addressing many of these constraints.

Looking ahead, the Raspberry Pi is expected to play a greater role in AI and cloud computing, particularly in edge computing. Its applications in healthcare, smart cities, space exploration, and smart agriculture further highlight its versatility [21].

In conclusion, the Raspberry Pi is a prime example of technology that is easily accessible, reasonably priced, and useful. It bridges the gap between innovation and education and offers developers and makers an open platform. It will continue to be a catalyst for technical development, creativity, and problem-solving as it develops further, enabling people and communities to create a more intelligent and sustainable future.

Chapter 3

Desing and implementation

3.1 Introduction

After reviewing the theoretical foundations related to the Qt framework and network communication protocols, this chapter presents the practical implementation phase, which forms the core of our project's applied work. The objective is to provide a comprehensive overview of the application development process, including development stages, technical decisions, and integrated system features.

The project involves a remotely managed system where a control computer transmits prayer times to a display screen connected to a Raspberry Pi. This setup enables the dynamic and efficient display of prayer times, delivered in real-time over a network using both UDP and TCP protocols.

The main features implemented in the application include:

Dynamic display of prayer times on a screen connected to a Raspberry Pi.

Daily automatic updates of prayer times using the most recent data.

Playback of the Adhan (call to prayer) at designated times.

Reliable and efficient data transmission was established between the control computer and the Raspberry Pi using UDP.

This chapter thoroughly explains each feature, the rationale behind technical and software design decisions, and the interactions between all components. Screenshots of the user interface and UML sequence diagrams are included to illustrate system behavior and structure.

3.2 Development Environment

Careful selection of both hardware and software tools was crucial to the successful implementation of the application. This section outlines the development environment, considering the project's objectives, available resources, and technical constraints.

3.2.1 Operating System

Two operating systems were employed during development:

- **Windows 10**: Utilized on the development machine for writing, compiling, and testing the application source code.
- **Raspberry Pi OS** (formerly Raspbian): A Linux-based operating system installed on the Raspberry Pi, used to run the prayer times display.

3.2.2 Programming Language

The project was developed entirely in **C++**. This language was chosen for its native compatibility with the Qt framework, high performance, and low-level access to system resources. Moreover, C++ facilitates efficient memory and resource management, which is crucial in embedded environments such as the Raspberry Pi.

3.2.3 Qt Library

We used **Qt 4**, a mature and widely adopted C++ framework renowned for its cross-platform capabilities and robust performance. It was chosen for several reasons:

Stability and Lightweight

Qt 4 is a stable and lightweight version of the framework, making it well-suited for resource-constrained devices such as the Raspberry Pi. It offers a responsive user experience while maintaining minimal system overhead.

Graphical User Interface (GUI) Development

The `QtGui` module was utilized to develop a responsive and visually appealing user interface. Key Qt classes employed include:

- QLabel: Displays the current time, date, and prayer times
- QTimer: Manages periodic updates, including real-time clocks and countdown timers
- QPalette and `QPixmap`: Facilitate dynamic background changes, for example, at the start of prayer times.
- QFont and `QColor`: Used to customize the appearance of text elements.

Network Communication

The `QtNetwork` module provided essential classes such as `QUdpSocket` and `QTcpSocket` to manage UDP and TCP communications, respectively. This enabled us to combine the reliability of TCP with the speed and low overhead of UDP, resulting in a responsive real-time data exchange system.

Signals and Slots Mechanism

A key feature of Qt—the signals and slots mechanism—was employed for inter-component communication. This mechanism proved especially useful for triggering prayer time events, updating clocks, and handling incoming data.

Multimedia Playback

Although Qt 4 lacks advanced multimedia features, it supports the Phonon framework, which was utilized to play the Adhan audio files at the designated prayer times.

Portability

Qt 4's cross-platform capabilities enable easy adaptation of the application to other environments with minimal code modifications.

Documentation and Support

Despite being an older version, Qt 4 remains well-documented and widely supported, which significantly facilitates development and troubleshooting processes.

3.2.4 Integrated Development Environment (IDE)

We used **Qt Creator**, the official integrated development environment (IDE) from the Qt Project. It provides a modern and unified interface for writing, managing, and debugging code, along with deep integration with build tools and project configuration options, thereby streamlining the development process.

3.2.5 Data Transmission

Communication between the control computer and the display unit (Raspberry Pi) utilized both the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). UDP was employed for fast and lightweight data transmission, making it ideal for sending short, time-sensitive updates such as prayer times. Conversely, TCP was used for transmitting more critical or larger data, such as images, where guaranteed delivery and correct ordering are essential.

3.2.6 Hardware Setup

The hardware used for development and testing consisted of the following components:

- A **computer** (PC) running the controller application to manage and transmit prayer times.
- A **Raspberry Pi**connected to an **HDMI display**, running the display application.
- A **local area network (LAN)** enabling communication between the PC and Raspberry Pi via UDP and TCP sockets.

3.3 System Architecture

The system is composed of two main units:

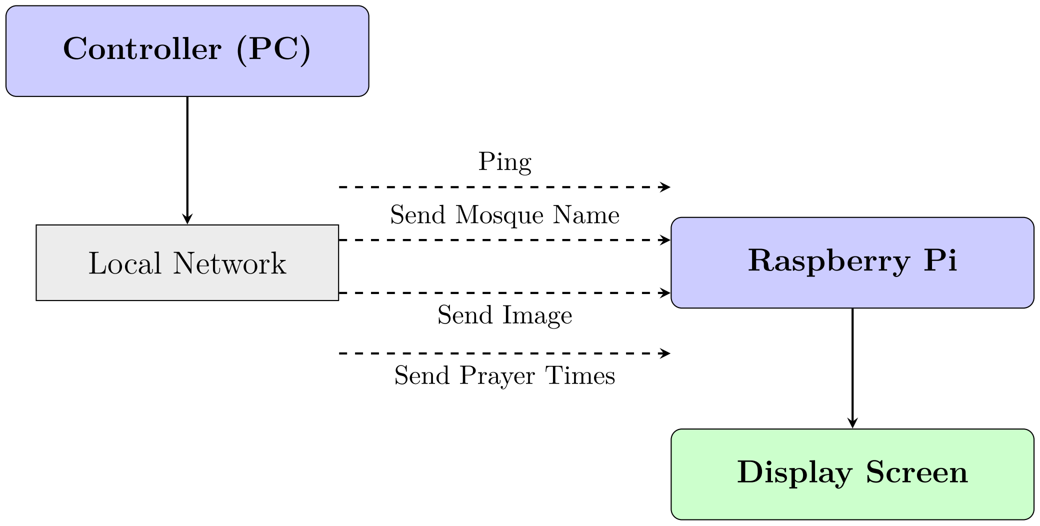


Figure 3.1: Complete communication diagram

The controller is responsible for sending various types of data, such as prayer times, mosque name, and images, to the Raspberry Pi. The Raspberry Pi then displays this information on the connected screen. The system ensures timely synchronization and efficient resource utilization.

3.4 Interface Design

The user interface was carefully designed to be clear, visually appealing, and responsive. It incorporates several key components:

3.4.1 Time Display

The current time is displayed in 24-hour format using a `QLabel`, which is updated every second via a `QTimer` to ensure real-time accuracy.

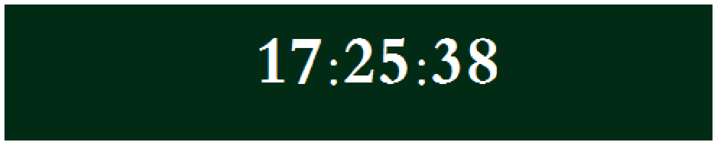


Figure 3.2: Custom Time displayed on the interface

3.4.2 Gregorian and Hijri Date Display

Both Gregorian and Hijri dates are displayed and refreshed every second. The Gregorian date is retrieved using the `QDate` class, while the Hijri date is computed through a Julian date conversion algorithm.

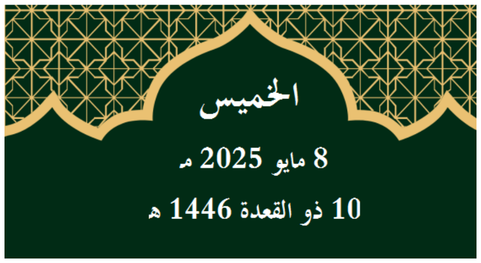


Figure 3.3: Custom Gregorian and Hijri Date displayed on the interface

3.4.3 Prayer Times

Labels are used to display the times for each of the five daily prayers, as well as sunrise:

- Fajr
- Shurooq

- Dhuhr
- Asr
- Maghrib
- Isha

Each prayer time is shown using a `QLabel` and is updated dynamically according to the received schedule.

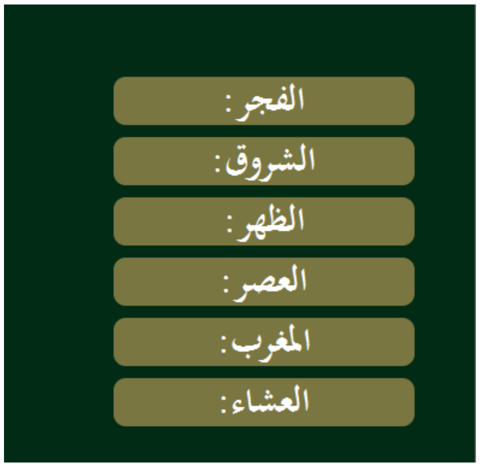
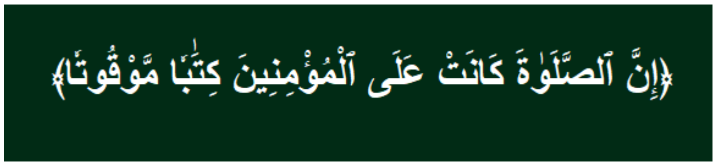


Figure 3.4: Custom Prayer times displayed on the interface

3.4.4 Quranic Verse Display

A Quranic verse is displayed permanently on the interface as a reminder of the importance of prayer. The verse: ”Indeed, prayer has been decreed upon the believers a decree of specified times” It is shown continuously. When a prayer time begins, this verse is temporarily replaced with another verse relevant to the specific prayer, and then the original verse is restored afterward.



﴿إِنَّ ٱلصَّلَوٰةَ كَانَتْ عَلَى ٱلْمُؤْمِنِينَ كِتَٰبًا مَّوْقُوتًا﴾

Figure 3.5: Custom Quranic Verse displayed on the interface

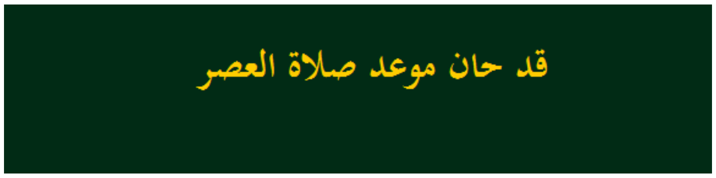
3.4.5 Mosque Logo Display

A custom logo for the mosque is displayed using `QPixmap` upon receiving data from the controller, along with the mosque's name. The logo can be dynamically modified by replacing specific colors (e.g., white) with alternative tones such as dark green. This enhances the visual identity of the interface and ensures consistency with the application's overall design.

Figure 3.6: Custom mosque logo displayed on the interface

3.4.6 Azan Audio Player

The Phonon library is employed to play the Azan audio at prayer times. This functionality is managed using `Phonon::AudioOutput` and `Phonon::MediaObject`, which load and play an MP3 file asynchronously.



قد حان موعد صلاة العصر

Figure 3.7: Custom Azan Audio Player displayed on the interface

3.4.7 Remaining Time for Next Prayer

The time remaining until the next prayer is continuously updated and displayed using a dedicated label in the following format:

"Next Prayer: in 00:53:28

This countdown text updates every second to reflect the precise time remaining until the upcoming prayer.

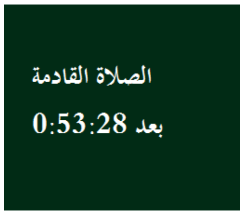


Figure 3.8: Illustrative image of the interface

3.4.8 Background Image Display

A background image is displayed using a `QLabel`. The image is resizable to cover the entire window, providing a visually pleasing backdrop.

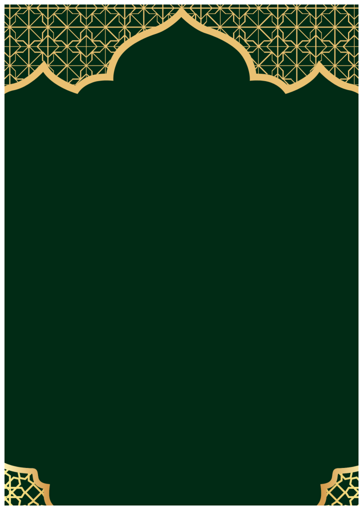


Figure 3.9: Illustrative image of the interface

3.5 Interface Operation

The interface operates according to the following workflow:

1. The prayer times are retrieved and shown when the application first launches.
2. Every second, the current time and date are updated and shown.
3. When the prayer time begins, the Quranic text changes and returns to its original form after the call to prayer ends.
4. The remaining time for each prayer is calculated based on the current time.
5. The Azan audio is played when it is time for prayer.

3.6 Remote Control Application

The controller application allows the user to manage the display unit remotely. It contains functional buttons that correspond to specific application operations.

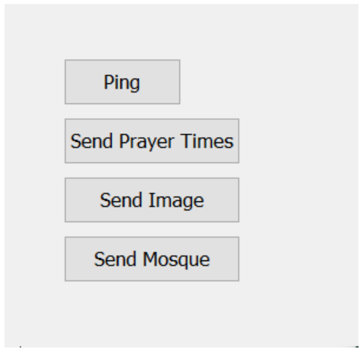
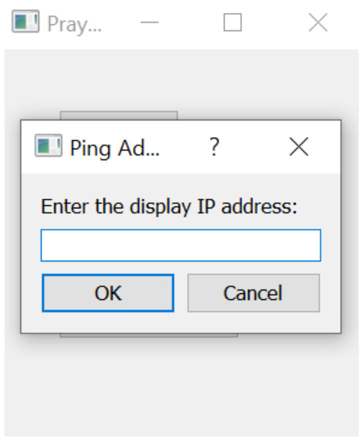


Figure 3.10: controller interface

3.6.1 Connection Verification (Ping)

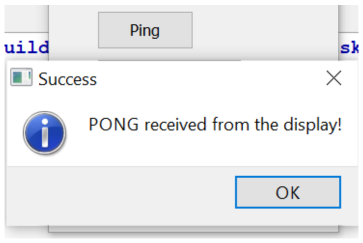
A *Ping* button is included in the interface to test the connection between the PC and the display. When pressed, the system sends a `PING` message via UDP to the target IP address.

If the display is online, it replies with a `PONG` message. Upon receiving this, the controller confirms the connection by showing a notification.



Enter the display IP address:

Figure 3.11: Entering IP Address in Controller



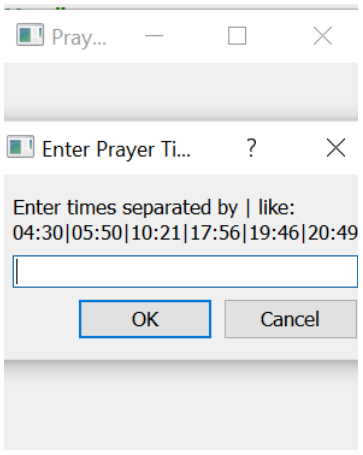
PONG received from the display!

Figure 3.12: PONG response confirmation

3.6.2 Sending Prayer Times

Prayer times (Fajr, Shurooq, Dhuhr, Asr, Maghrib, and Isha) are sent using the *Transmit Prayer Times* button in the following format:

Prayer_times:|04:30|05:50|12:05|17:56|19:46|20:49 This string is sent over UDP. Once received, the display extracts and updates the prayer times accordingly. An acknowledgment may also be sent to confirm successful reception.



Enter times separated by | like:
04:30|05:50|10:21|17:56|19:46|20:49

Figure 3.13: Entering Prayer time in Controller

3.6.3 Sending Images via TCP

The `sendImage()` function enables image transmission via TCP. After the user selects an image and inputs a display duration (in seconds), a header is created in the following format:

SEND_IMAGE|*image_name*|*duration*|*size*

The header is sent first, followed by the image data. Once transmission completes, the connection is closed.

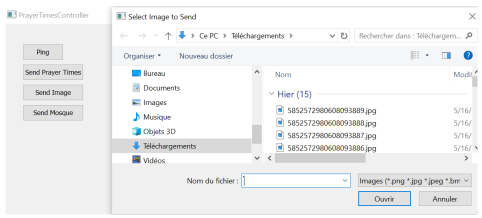
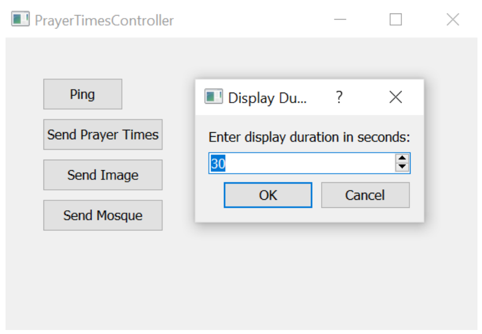


Figure 3.14: choice Image in Controller



Enter display duration in seconds: 30

Figure 3.15: choice duration in Controller

3.6.4 Sending Mosque Name

The `sendMosque()` function sends the mosque's name via UDP. After entering a valid name, the following message is generated and transmitted: **MOSQUE_NAME:YourMosqueName** Upon reception, the display updates the name accordingly.

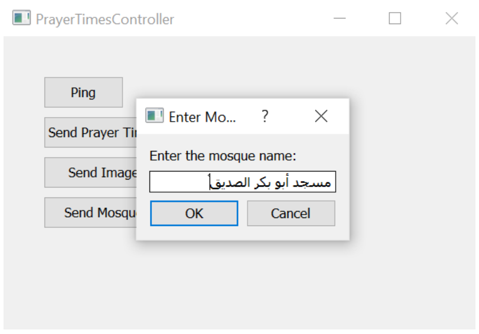


Figure 3.16: Entering Mosque Name in Controller

3.6.5 Feature Summary

- Test the connection with the display via Ping/Pong.
- Send prayer times.
- Send a custom image to be displayed.
- Send the mosque name.

3.7 The Unified Modeling Language (UML)

UML(Unified Modeling Language) is a standardized modeling language used for visualizing, specifying, constructing, and documenting the components of a software system. In this project, various UML diagrams were employed to illustrate the system's functionality, structure, and internal communication.

3.7.1 Use Case Diagram – Prayer Time Display System

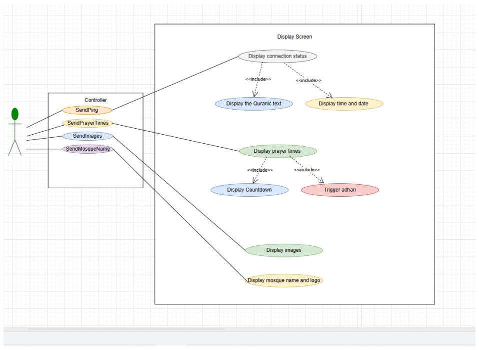


Figure 3.17: Use Case Diagram of the Prayer Time Display System

This Use Case Diagram represents the interaction between the main user (administrator) and the system components that manage and display prayer times on the mosque's digital screen.

Actor

- **User:** Represents the administrator or operator who manages the system using the controller interface.

System Components (Use Cases)

1. **SendPing**
   Function: Sends a connection test signal from the controller to the display screen.
   Includes:

   - *Display connection status:* Every ping operation includes an automatic update of the connection status.

2. **SendPrayerTimes**
   Function: Sends updated prayer times to the display.
   Includes:

   - *Display prayer times:* The times are shown on the screen.
   - *Trigger adhan:* The system will automatically call the adhan at the correct time.
   - *Display Countdown:* A countdown to the next prayer is displayed.

3. **SendImages**
   Function: Sends visual content to the display screen.
   Leads to:

   - *Display images:* Images are shown on the screen between or during prayer-related activities.

4. **SendMosqueName**
   Function: Sends the mosque name and logo for display.
   Leads to:

   - *Display mosque name and logo:* Branding appears on the main interface.

Display Screen Functionalities

- Display connection status: Shows if the connection with the controller is active.
- Display the Quranic text: May show verses or supplications.
- Display time and date: Constantly displays current time and Gregorian/Hijri date.
- Display prayer times: Main feature, shows today's prayer schedule.
- Trigger adhan: Automatically plays the call to prayer at the appropriate time.
- Display Countdown: Shows a real-time countdown to the next prayer.
- Display images: Rotates or presents visual media sent from the controller.
- Display mosque name and logo: Displays static mosque identity.

Include Relationships (`<<include>>`) Explanation

These indicate mandatory sub-functions that must always occur as part of a larger use case:

- **SendPing** always includes checking the connection.
- **SendPrayerTimes** always triggers adhan and countdown features.

These relationships ensure modular design and the reuse of functionalities.

Objective of the diagram:

This diagram clarifies the division of responsibilities between the controller and the display unit, illustrating which functions are remotely triggered and which operate autonomously.

3.7.2 Sequence Diagram

This diagram illustrates the interaction between key components involved in sending prayer times and playing the Adhan.

Participants:

- User: Initiates the transmission of prayer times.
- Controller: Acts as the processor responsible for communicating prayer times to the display via the network.
- Data transmission between the controller and the display is handled using UDP and TCP
- Display: The device that shows prayer times and plays the Adhan.
- The component responsible for playing the Adhan audio.

Sequence of Events:

- sendPrayerTimes Clicked(): Triggered when the user clicks the ”Send Prayer Times” button, initiating system interaction.
- sendPing(): Sends a ”ping” message via UDP to confirm connectivity.
- sendPrayerTimes(): The controller transmits prayer times data via a UDP socket to the display.
- sendMosqueName(): Sends the mosque name via UDP to customize the mosque name shown on the display interface. It also hides the displayed image when required (e.g., after the Adhan finishes).
- sendImage(): Sends an image (such as the mosque logo or photo) via TCP.
- receiveData(): The display receives data sent from the controller via UDP and TCP.
- update PrayerLabels() (internal message in Display): Updates the display components with the new prayer times.
- triggerAzan() (internal message in Display): Initiates playing the Adhan when prayer time begins.
- playAzan(): Calls the AzanPlayer to play the Adhan audio.
- hideImage(): Hides the displayed image when necessary (e.g., after the Adhan finishes).

- onAzanFinished() (internal message in AzanPlayer): Fires an event indicating the completion of the Adhan playback.

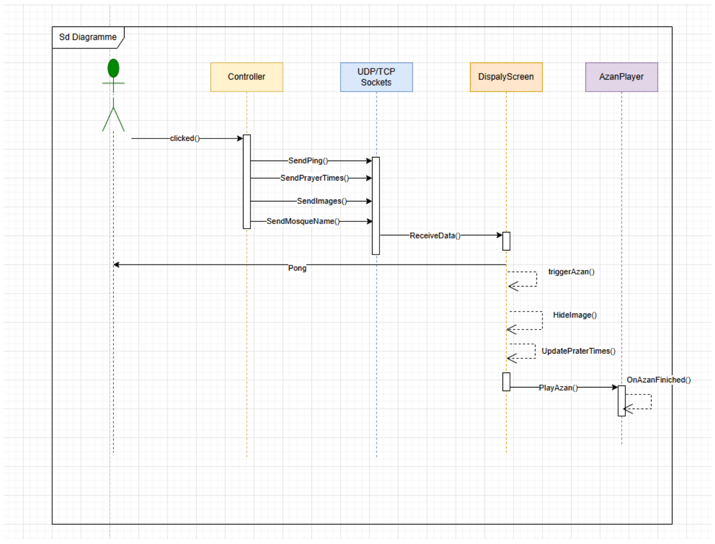


Figure 3.18: Sequence Diagram of the Prayer Time Display System

Objective of the diagram:

This diagram illustrates the interaction between components to synchronize events, ensuring that prayer times are updated accurately and the Adhan is played correctly and promptly.

3.8 Performance and Resource Usage

Due to the limited hardware capabilities of the Raspberry Pi, performance optimization and efficient resource management were critical considerations throughout both the design and testing phases of the application.

During development, the system's behavior was continuously monitored to ensure that it remained responsive and stable under prolonged operation. The following performance metrics were recorded:

- **CPU Usage:** The application maintained low CPU usage, typically remaining under 10 % during idle periods and reaching up to 25% during audio playback or image transitions.

- **Memory Usage:** The memory footprint averaged approximately 70 MB during runtime, accounting for the Qt GUI components, dynamic background rendering, and the Phonon audio playback module.
- **Storage Requirements:** The complete application package—including MP3 audio files and display images—requires less than 150 MB of disk space.
- **Monitoring Tools:** Tools such as `htop`, `top`, and `valgrind` were used during testing to monitor resource consumption and identify potential memory leaks or performance bottlenecks.

These Results confirm that the application is suitable for continuous deployment on Raspberry Pi devices without imposing a significant load on system resources.

3.9 Testing

This section presents the tests carried out to verify the correct functionality of the Prayer Times Display application developed using Qt 4.

3.9.1 Testing Environment

Testing was conducted on a Windows 10 machine with the Qt 4 libraries installed. Network-related tests were performed over a local area network (LAN) that connected the controller to the display device.

3.9.2 Test Procedures

1. **Display of Prayer Times and Current Time:** Verified that prayer times, current date, and time are accurately displayed on the screen.
2. **Adhan Sound Playback:** Confirmed that the Adhan audio is played at the correct prayer times without delay or interruption, utilizing the Phonon multimedia framework.
3. **Countdown Timer:** Ensured that the countdown timer updates accurately every second, reflecting the remaining time until the next prayer.

4. **Image Reception over TCP:** Tested the reception of images sent from the controller via TCP.

5. **UDP Communication:** Validated the transmission and reception of prayer time updates and control messages over UDP, ensuring real-time synchronization on the display unit.

3.9.3 Issues and Solutions

Several challenges emerged during the development phase, particularly with communication protocols, multimedia handling, real-time interface updates, and Arabic language support.

- **Socket Communication (UDP and TCP):** Occasional packet loss and delays were mitigated by optimizing socket configurations and implementing periodic ping checks to ensure reliable connectivity.
- **Audio Playback (Phonon):** Delays in playback were minimized by preloading audio files and handling playback asynchronously through threading.
- **Real-Time Display Updates:** Utilizing `QTimer` and `QLabel` enabled smooth and reliable updates for prayer times, date, and countdown timers.
- **UI Design Without Qt Designer:** Manual UI construction in C++ was facilitated through code modularization and the use of layout managers.
- **Arabic and RTL Text Support:** Given Qt 4's limited support for Arabic and right-to-left (RTL) scripts, Arabic text was converted into images to guarantee proper visual rendering. Font issues and encoding inconsistencies were resolved by embedding Arabic fonts and using UTF-8 encoding. Cursor misalignment in input fields was addressed by restricting input direction, and static translations were implemented manually.

Testing results confirmed that the application performs efficiently and fulfills all functional requirements. Image transmission via TCP is stable and effective, UDP-based communication between the controller and display functions is reliable, and both prayer time display and Adhan playback are accurate.

3.10 Potential Enhancements

To enhance usability and adaptability, the following enhancements are proposed for future development:

- **Multilingual Support:** Enable language switching between Arabic and French.
- **Touchscreen Interface:** Redesign the user interface to support touchscreen interaction.
- **Automatic Calendar Updates:** Implement online integration to automatically update daily prayer times.
- **Visual Alerts:** Introduce visual indicators such as animations or color changes to alert users of upcoming prayer times.
- **Remote Configuration:** Create a web-based or mobile configuration panel for adjusting settings such as the mosque name and Adhan volume.

These enhancements aim to increase accessibility, improve user experience, and ensure suitability for a wide range of mosque environments.

3.11 Conclusion

This chapter presents the implementation phase of a networked prayer times display system, developed using Qt 4 and Raspberry Pi. The system provides real-time updates of prayer times, dates, and Quranic verses, and plays the Adhan on time. Despite the limitations of Qt 4, particularly in multimedia synchronization and Arabic text rendering, practical workarounds were applied to maintain both functionality and user experience. Communication between components was reliably managed through UDP and TCP sockets. The development process addressed key technical challenges in interface design, network communication, and embedded audio playback, contributing to meaningful technical insights. This work lays a solid foundation for future enhancements such as remote access, multilingual support, and advanced user interaction, demonstrating how modern technology can effectively serve spiritual and community needs.

Conclusion

In this thesis, we presented the design and implementation of a digital prayer times display system that combines modern programming, embedded systems, and networking techniques. Using the Qt 4 framework and a Raspberry Pi board, we developed a fast and efficient application capable of displaying real-time prayer times, Gregorian and Hijri dates, Quranic verses, and playing the Adhan (call to prayer) at appropriate times.

The project was structured to ensure flexibility, organization, and support for network communication between the control unit (computer) and the display unit (Raspberry Pi). UDP and TCP protocols were strategically used to balance performance and reliability in data exchange. Despite some limitations related to the use of Qt 4, particularly regarding multimedia synchronization and Arabic language support, practical solutions were implemented to ensure the stability and effectiveness of the system.

The project also highlighted several technical challenges, such as real-time interface updates, asynchronous audio playback, and interface design without the use of graphic design tools. These challenges provided an opportunity to gain practical experience and employ critical thinking methods to overcome technical obstacles.

In the future, this work will provide a solid foundation for developing more advanced and flexible display systems for mosques. Supporting remote training, multilingualism, and linking to online prayer time sources can enhance its capabilities. This project demonstrates that modern technology can be effectively harnessed to serve religious needs in a practical and sophisticated manner.

Bibliography

[1] Daniel Molkentin. *The book of Qt 4: The art of building Qt applications*. No Starch Press, 2007.

[2] Jasmin Blanchette and Mark Summerfield. *C++ GUI programming with Qt 4*. Prentice Hall Professional, 2006.

[3] The Qt Company. Official qt documentation and history. Available online: `https://doc.qt.io`, 2025. Accessed: 2025-02-16.

[4] Matthias Ettrich. The history of kde and its relationship with qt. *KDE Official Documentation*, 1996.

[5] Open Source Initiative. Qt's transition to open source and the qpl license. *Open Source Initiative*, 1999.

[6] Tech Analysts. Qt's global success in software development. *Software Development Journal*, 2022.

[7] Ray Rischpater. *Application development with qt creator*. Packt Publishing Birmingham, 2013.

[8] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall, 2nd edition, 2008.

[9] Qt Company. Qt networking programming guide, 2021. Accessed: 2025-02-17.

[10] Nokia Corporation. *Qt 5.6 Documentation - Qt Designer*. Nokia, 2016. Accessed: 2025-02-17.

[11] Canonical Ltd. A comprehensive guide to ubuntu on raspberry pi. *Ubuntu Official Documentation*, 2021.

[12] Gareth Halfacree and Eben Upton. *Raspberry Pi Manual: A Practical Guide to the Revolutionary Computer*. Haynes Publishing, 2012.

[13] Sean Everett. *Beginning IoT with Raspberry Pi and Arduino*. Apress, 2016.

[14] Simon Monk. *Programming the Raspberry Pi: Getting Started with Python*. McGraw-Hill Education, 2013.

[15] Eben Upton. *The Official Raspberry Pi Beginner's Guide*. Raspberry Pi Trading Ltd, 2nd edition, 2019.

[16] Eben Upton and Gareth Halfacree. *Raspberry Pi User Guide*. Wiley, 4th edition, 2016.

[17] David Morgan. A comparative study of single-board computers. *Computing Research Journal*, 30(2):120–135, 2021.

[18] Derek Molloy. *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux*. Wiley, 2016.

[19] Aladhan API. Prayer times api, 2023. Accessed: 2025-03-07.

[20] Pygame Documentation. Getting started with pygame, 2023. Accessed: 2025-03-07.

[21] Matt Richardson and Shawn Wallace. *Getting Started with Raspberry Pi*. O'Reilly Media, 2012.

[22] Raspberry Pi Foundation. Education. `https://www.raspberrypi.org/education/`. 2025-05-21.

[23] Simon Monk. *Programming the Raspberry Pi: Getting Started with Python*. McGraw-Hill, 2nd edition, 2018.