Application Note

在 TI 嵌入式平台上使用 Qt 6 构建现代 GUI

Shriya Surti, Krunal Bhargav

摘要

Qt 是一个功能强大的跨平台框架，用于开发图形用户界面 (GUI) 应用程序。本应用手册提供了一个分步示例，演示了如何创建基于 Qt 的简单 GUI、使用 GUI 闪烁 LED、针对 AArch64 架构交叉编译工程以及在 德州仪器 (TI) 平台上部署 GUI。本文档介绍了完整的工作流程，包括设置 Qt 环境、开发基本应用程序、为目标架构构建应用程序以及在 AM62P 平台上运行该应用程序。

内容

商标

1 简介

本演示使用 TI EVM：SK-AM62P-LP 评估板

备注

本教程也适用于任何 AM62x/AM62P 器件。

2 详细说明

先决条件

首先确保正在使用 Ubuntu 22.04。

安装 QEMU 用户模式仿真器，以便在主机系统上启用跨架构执行：

sudo apt-get install qemu-user-static

需要 qtcreator 才能创建 Qt 应用程序。

sudo apt install qtcreator

使用以下示例了解如何在 AM62P/AM62X 上闪烁 LED：闪烁 LED

2.1 启动 Qt Creator

要开始创建应用程序，请运行以下命令以启动 Qt Creator：

qtcreator

图 2-1 显示启动 Qt Creator 后的屏幕。

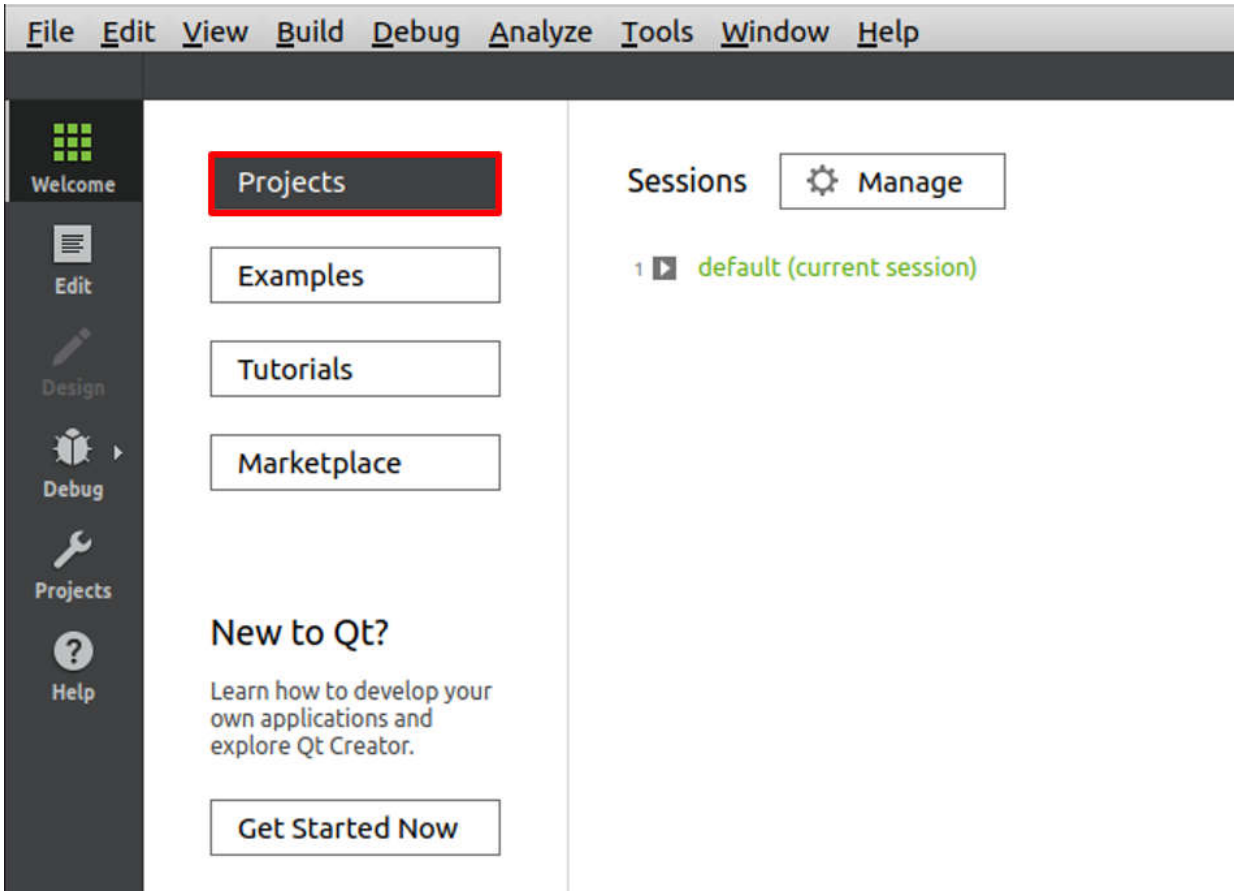

图 2-1. 启动新工程

在此处，单击图 2-1 中所示的*工程*按钮，创建一个新工程。

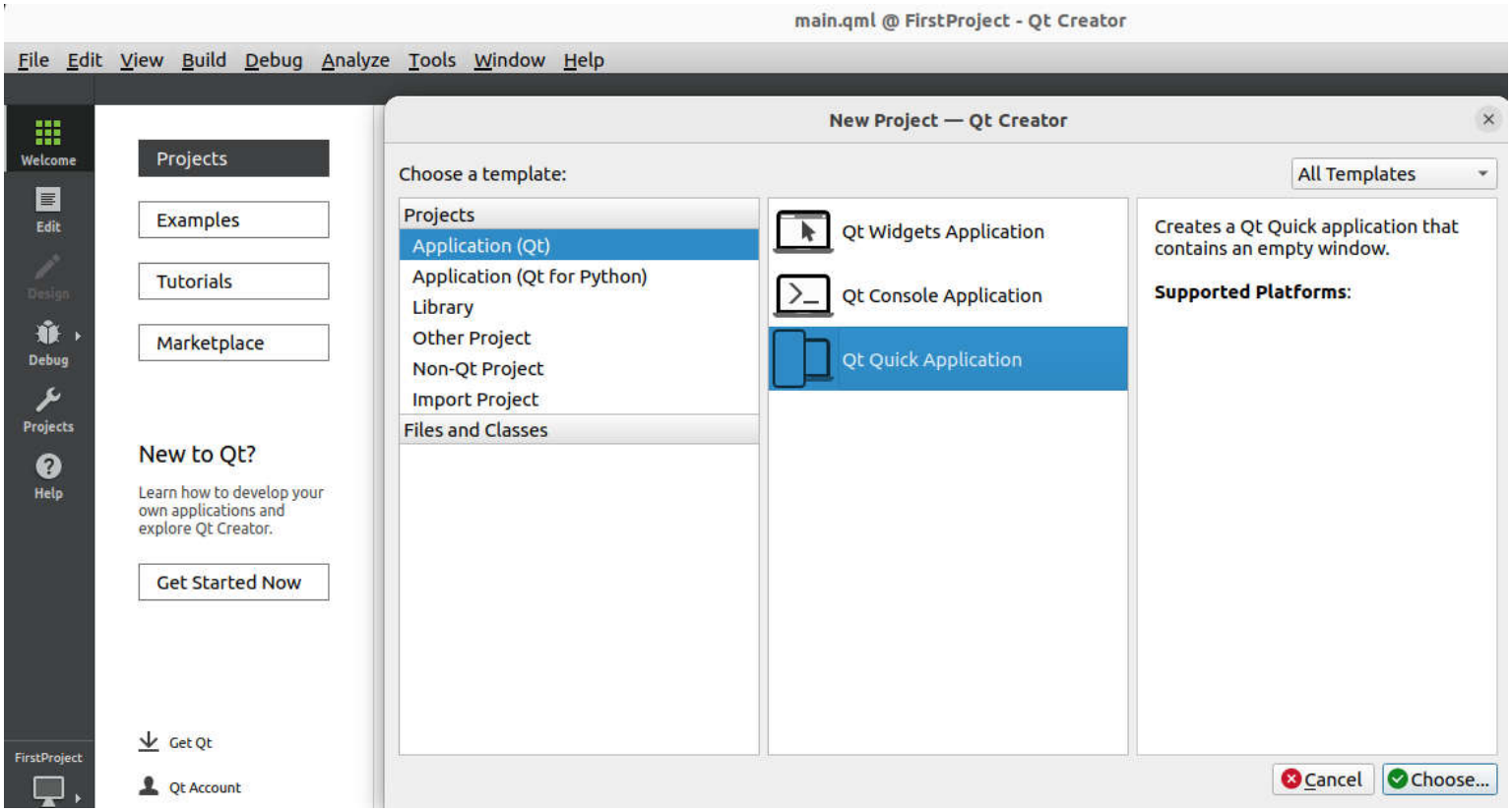

图 2-2. 创建 Qt 应用程序

选择 Qt Quick Application（Qt 快速应用程序）。单击*选择*后，单击*下一步*，直到显示以下屏幕。选择 ***cmake*** 并确保选择 Qt 6.X。在本例中，最低版本为 6.2。

接下来，在 /home/USER（其中 user 是用户名）或用户的选定位置下指定一个位置。

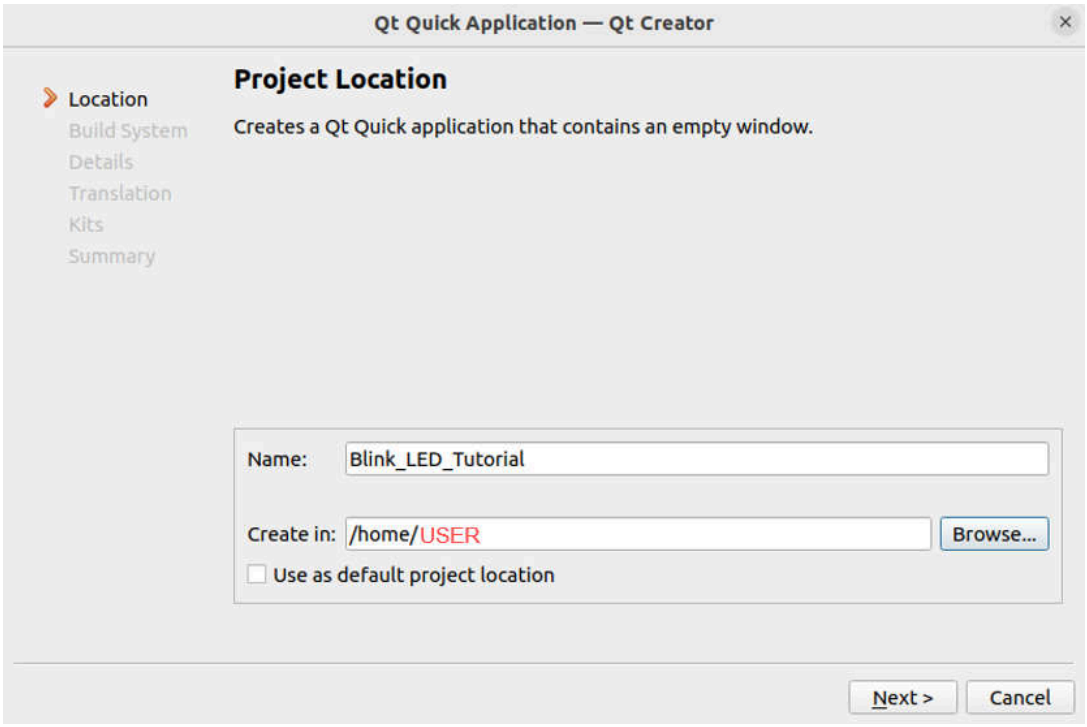

图 2-3. 指定工程位置

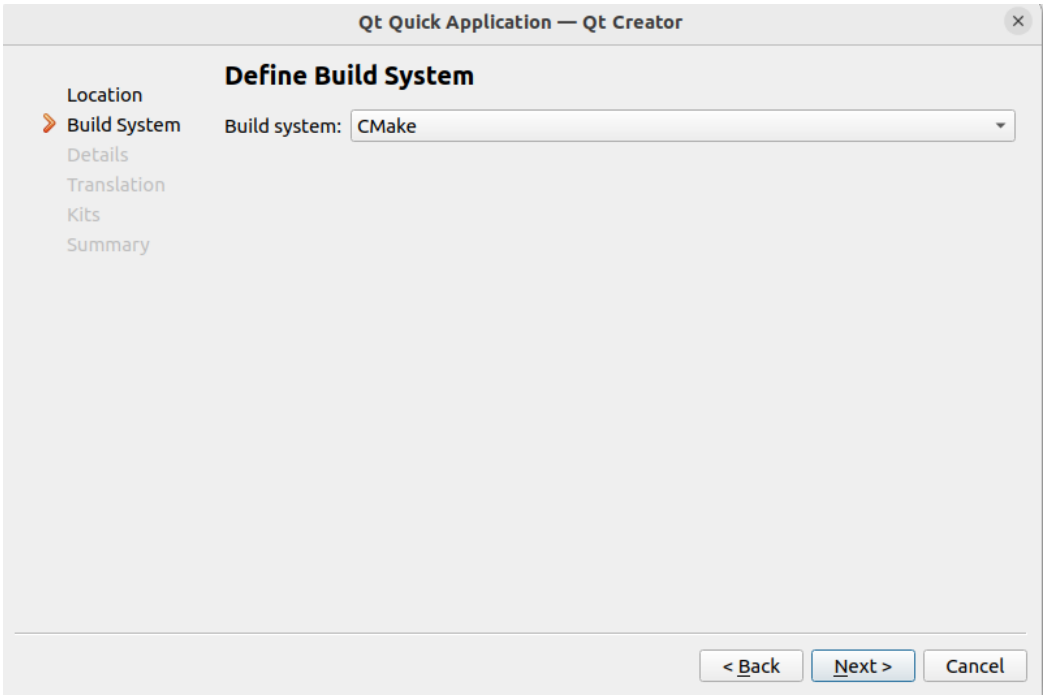

图 2-4. 使用 CMake

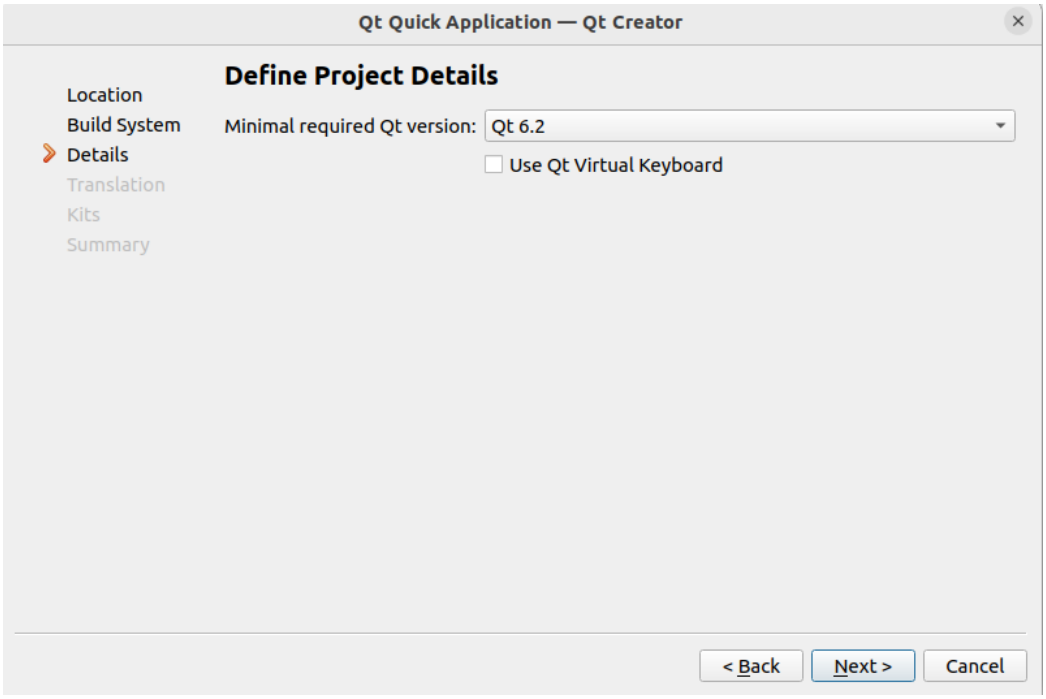

图 2-5. 指定 Qt 版本

然后继续完成设置，直到单击*完成*。

创建工程后，用户就有了一个基本的 Qt 应用程序。

2.2 在 PC 上运行第一个工程

工作区必须类似于图 2-6 所示

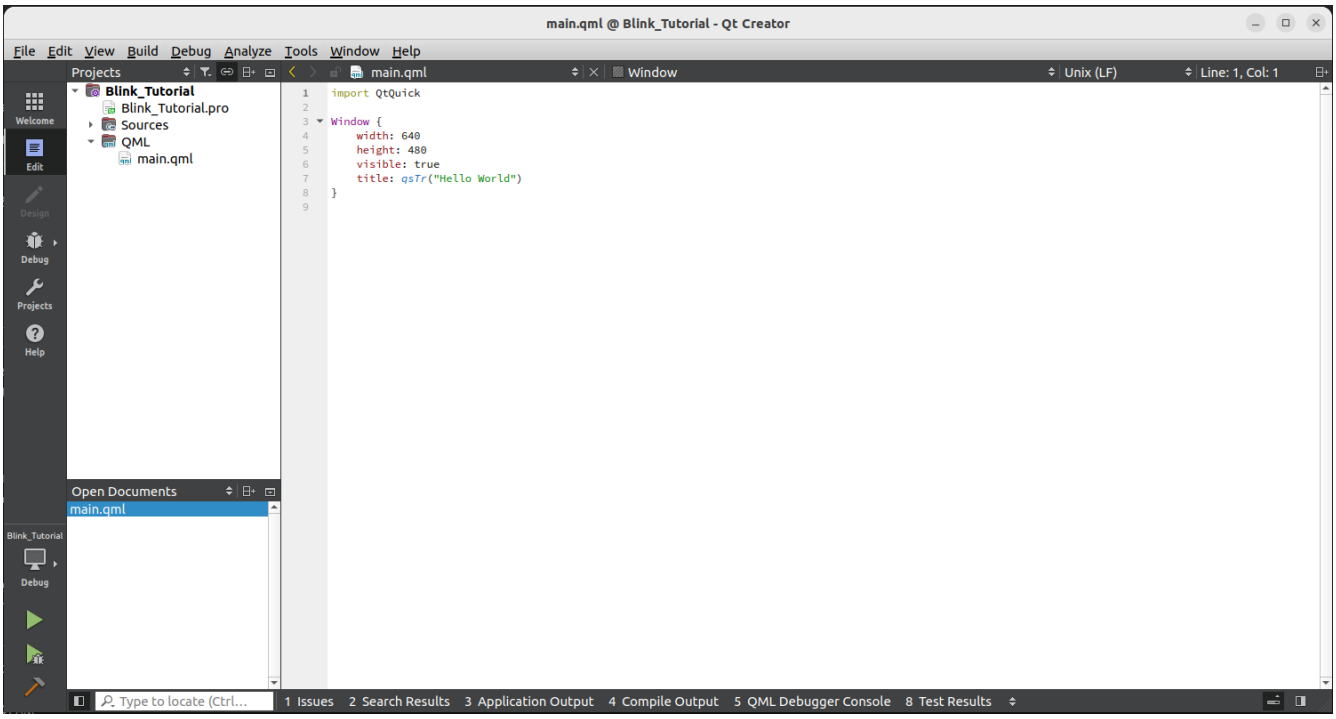

图 2-6. 工程基线

图 2-6 所示为创建了一个窗口。本教程不讨论窗口声明。若要了解有关窗口声明的更多信息，请参阅以下内容：Window QML Type | Qt Quick | Qt 6.10.1

继续操作，然后单击*运行*按钮。

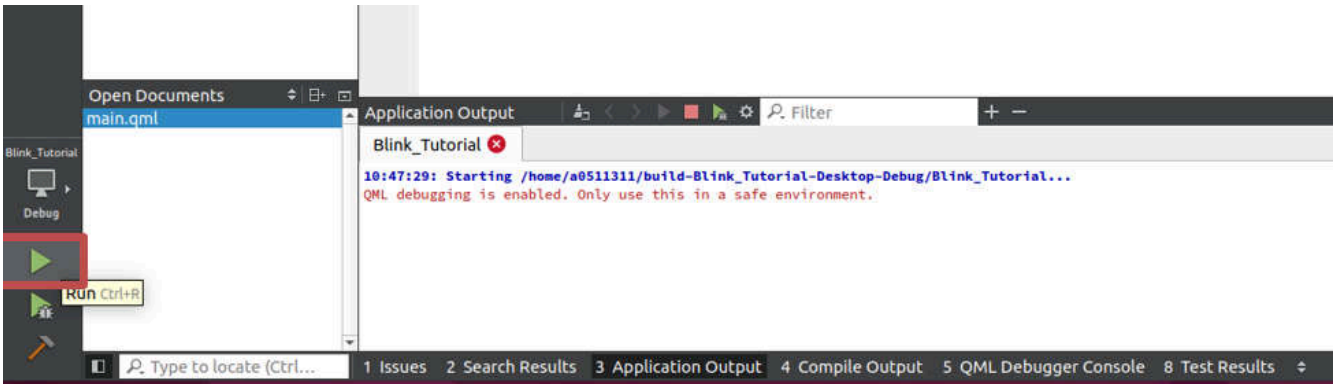

图 2-7. 运行您的应用程序

当应用程序运行时，将显示一个带有标题 *Hello World* 的屏幕，因为已为该窗口定义了标题。

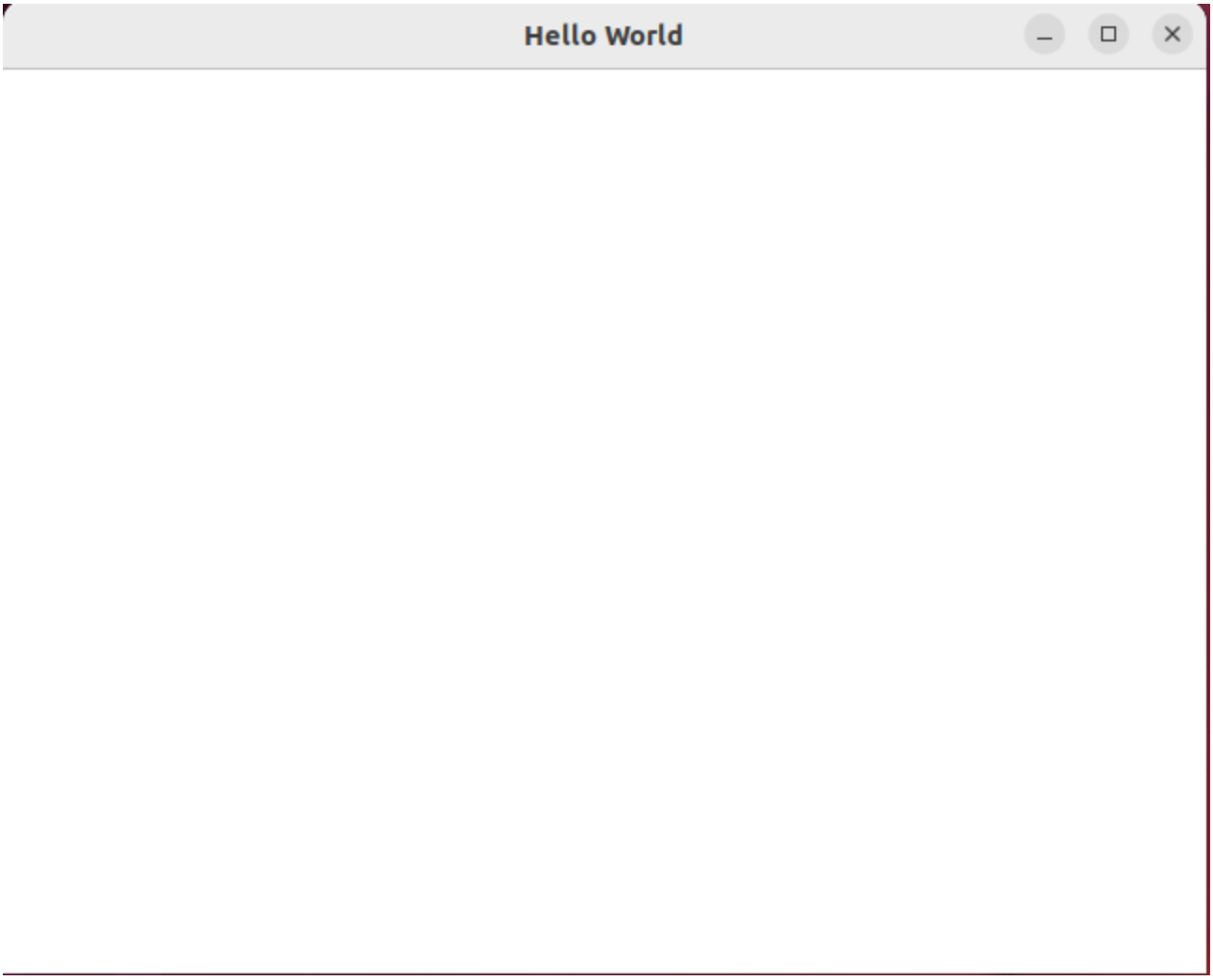

图 2-8. Hello World 窗口

窗口大小显示为 480×640，与窗口元件中指定的尺寸相匹配。

下一步是向应用程序中添加一个名为*闪烁*的按钮。

2.3 创建按钮

在 Qt QML 中，使用*按钮*元件创建按钮，该元件是 *QtQuick.Controls* 模块的一部分。QML 中的元件只是可重复使用的构建块，例如窗口、标签、按钮、图像或已定义的自定义 UI 元素。

元件封装了行为和外观，使元件在 UI 布局中易于使用和整理。对于自定义元件，请参见以下 QT 列表： The QML Reference | Qt Qml | Qt 6.10.2

```
1  import QtQuick
2  import QtQuick.Window 2.15
3  import QtQuick.Controls 2.0
4
5  Window {
6      id: window
7      width: 640
8      height: 480
9      visible: true
10     title: qsTr("Hello World")
11
12     Button {
13         id: blink
14         text : qsTr("Blink");
15     }
16 }
17
```

图 2-9. 添加按钮

该 id 为按钮提供了一个唯一引用标识，以便程序可以调用该按钮。例如，处理信号、检查状态或修改属性。

text 属性设置显示在按钮上的标签。

当用户单击*运行*时，将显示以下内容：

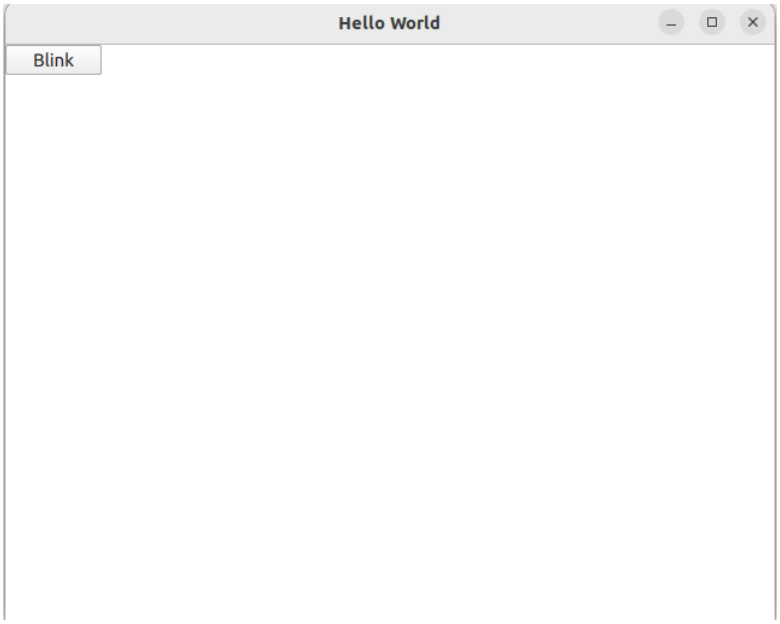

图 2-10. 带有闪烁按钮的修改后窗口

现在在名为 *Blink* 的窗口中有一个按钮。

2.3.1 修改按钮的属性

可以移动屏幕中央的按钮。

```
1  import QtQuick
2  import QtQuick.Window 2.15
3  import QtQuick.Controls 2.0
4
5  Window {
6      id: window
7      width: 640
8      height: 480
9      visible: true
10     title: qsTr("Hello World")
11
12     Button {
13         id: blink
14         text : qsTr("Blink");
15         anchors.verticalCenter: parent.verticalCenter
16         anchors.horizontalCenter: parent.horizontalCenter
17     }
18 }
```

图 2-11. 移动闪烁按钮

使用锚点。锚点是 QML 中的一个属性，允许相对于其他元素对 UI 元素进行定位。父级是无需按像素手动计算坐标像素即可使用的元件。它有助于定义项目（如窗口和按钮）之间的关系。在本例中，*闪烁* 按钮位于窗口的中央。

现在，当用户运行应用程序时，按钮位于屏幕中央。

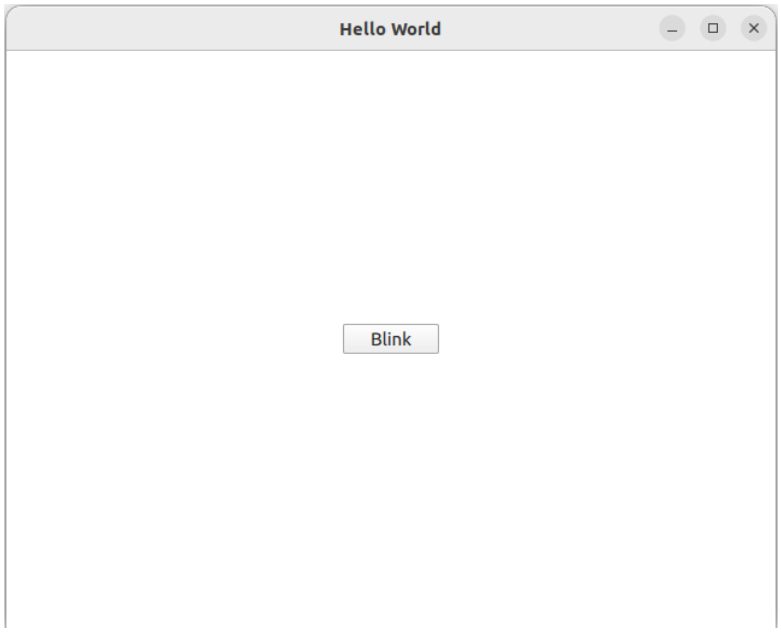

图 2-12. 按钮在窗口中重新定位

同一行中是否可以有多个按钮？

```
import QtQuick
import QtQuick.Window 2.15
import QtQuick.Controls 2.0

Window {
    id: window
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello World")

    Row {
        id: col [1]
        anchors.verticalCenter: parent.verticalCenter [2]
        anchors.horizontalCenter: parent.horizontalCenter [3]
        spacing: 25

        Button {
            id: blink
            text : qsTr("Blink");
        }

        Button {
            id: on
            text : qsTr("Turn On LED");
        }

        Button {
            id: off
            text : qsTr("Turn Off LED");
        }
    }
}
```

图 2-13. 添加行元件

通过添加名为 Row [1] 的元件来修改程序。Row 是一个工具，用于帮助整理一行中的所有元件。在本例中，利用窗口的垂直 [2] 和水平 [3] 中心，沿着中心行整理元件。现在，在此范围内添加的任何元件都位于同一行，并按图 2-14 所示居中。

图 2-14. 多个元件在窗口中居中

2.4 创建 C++ 后端元件

由于该按钮是在 QML 文件中创建的，因此应向其添加功能，以便点击该按钮时能够产生一些结果。为此，编写一个 C++ 应用程序来处理工程的后端元件。

这是 Qt 开发中的一种常用方法，其中用户将 QML 用于前端（用户界面），将 C++ 用于后端（业务逻辑）。通过将前端与后端分开，用户可以组织和维护代码。

2.4.1 定义工程的范围

首先，在工程范围内创建一个新文件。

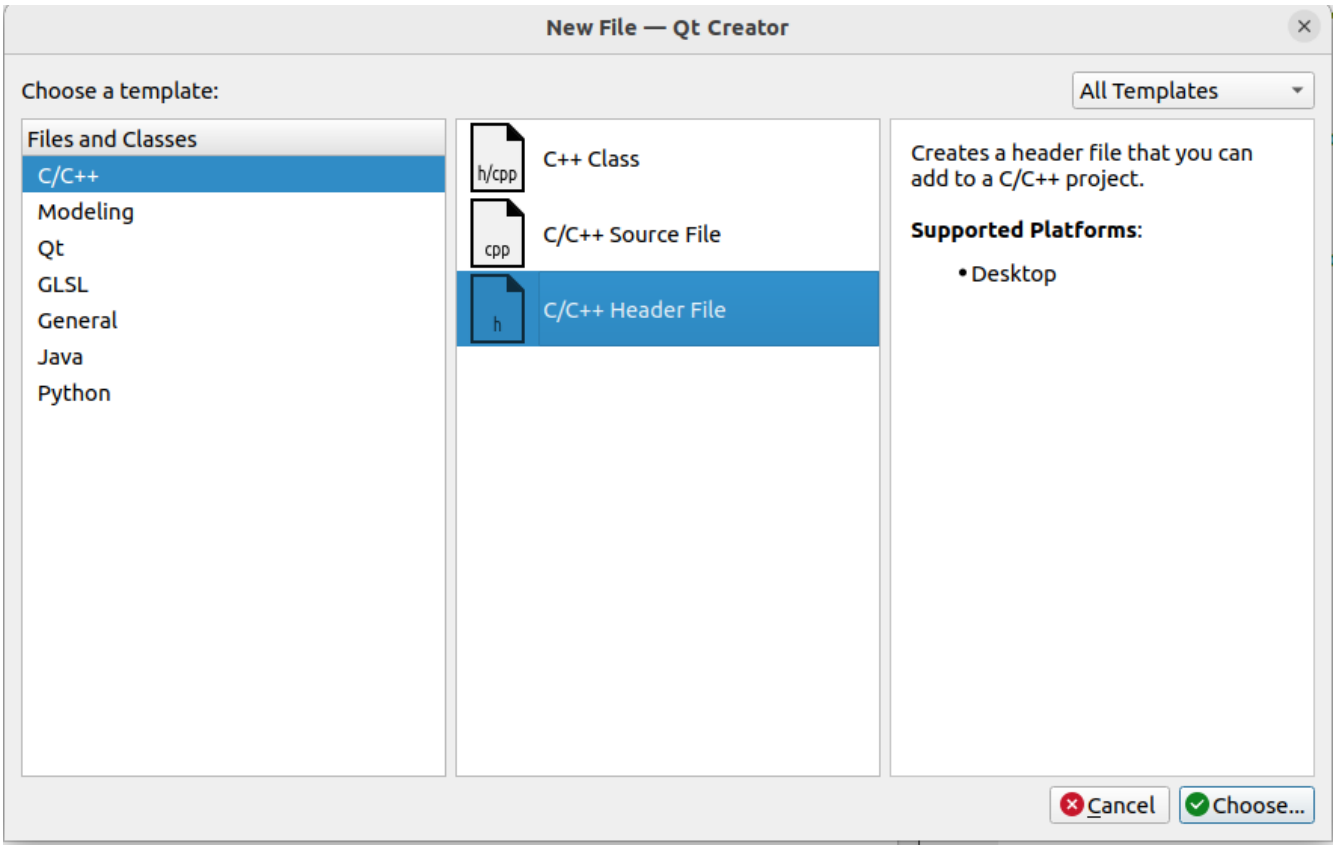

图 2-15. 创建头文件

创建文件后，屏幕如下所示：

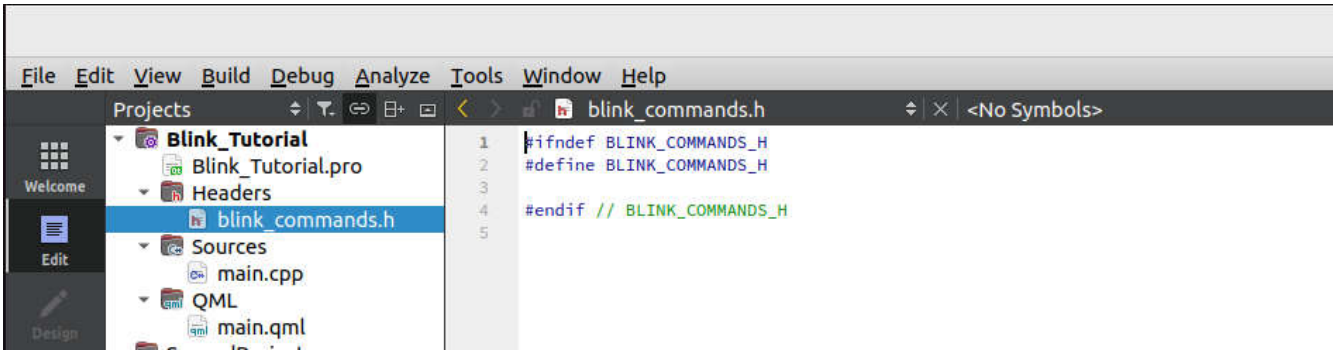

图 2-16. 基线头文件

在 Qt 中，*槽位* 是程序中发生某些事情时运行的函数。槽位与一个 *信号* 一起工作，该信号由一个对象发送，让程序知道发生了一个事件，例如按钮被点击。

将信号视为一条消息，将槽位视为接收和响应该消息的功能。在这个头文件中，定义了点击 *blink[1]*、*on[2]*和 *off[3]* 这三个按钮时发生的情况。每个按钮都连接到槽位，当用户单击按钮时，将调用相应的槽位以执行所需操作。

```
7  class BlinkCommands: public QObject
8  {
9      Q_OBJECT
10
11 public:
12     explicit BlinkCommands(QObject *parent = nullptr);
13
14 signals:
15
16 public slots:
17         void blink();[1]
18         void on(); [2]
19         void off(); [3]
20 };
21
22 #endif //BLINK_COMMANDS_H
23
```

图 2-17. 创建槽位

现在已经创建了头文件并定义了槽位，接下来创建一个 C++ 文件来实现槽位。此 C++ 文件包含触发槽位时执行的实际代码。

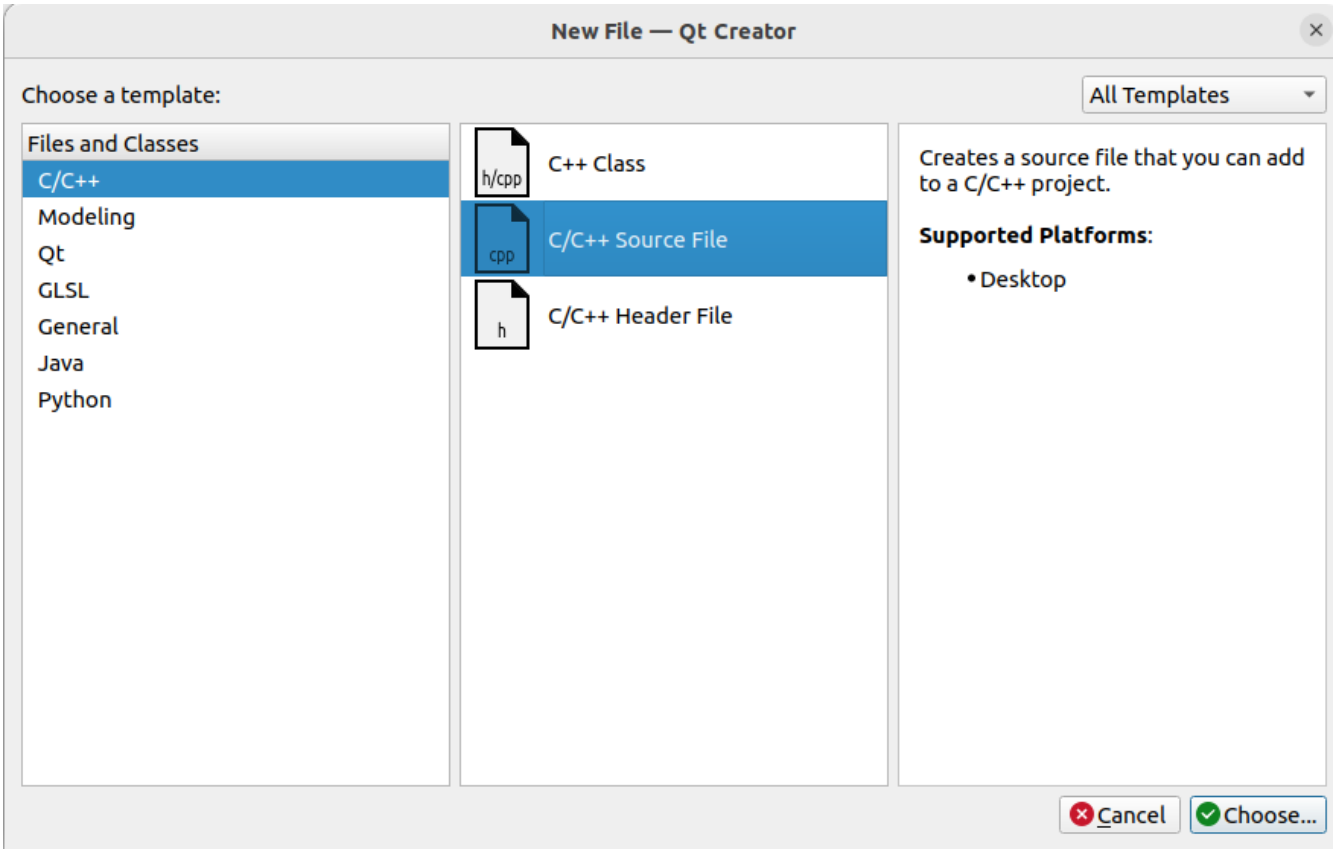

图 2-18. 创建源文件

使用与头文件相同的命名约定。

使用 QDebug，每次单击每个按钮时，都生成简单的打印语句：

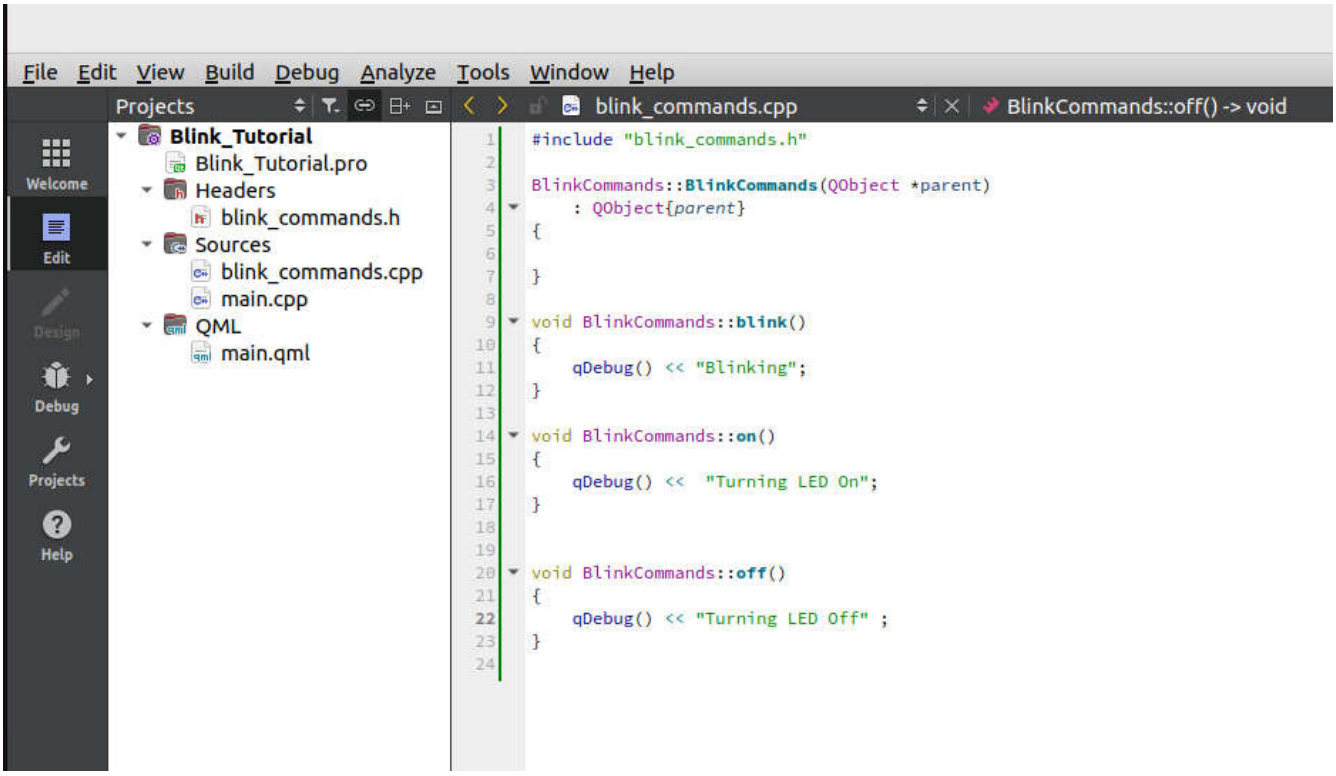

图 2-19. 为槽位创建输出

只要单击每个按钮，就会执行这些打印语句，并将相应的消息打印到控制台。

2.4.2 将 C++ 应用程序连接到 QML 应用程序

要将槽位连接到前端，请在 main.cpp 文件中创建一个对象。此对象将在 QML 文件中用于调用已写入后端中定义的函数。

```
12
13      QGuiApplication app(argc, argv);
14
15      QQmlApplicationEngine engine;
16
17      BlinkCommands test;
18      engine.rootContext()->setContextProperty("test", &test);
19
20      const QUrl url(QStringLiteral("qrc:/Blink_Tutorial/main.qml"));
21      QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
22                       &app, [url](QObject *obj, const QUrl &objUrl) {
23          if (!obj && url == objUrl)
24              QCoreApplication::exit(-1);
25      }, Qt::QueuedConnection);
26      engine.load(url);
27
28      return app.exec();
29  }
30
```

图 2-20. 将槽位连接到对象

在 QML 文件中使用 onClicked() 函数，让 QT 应用程序感知到，当按钮被点击时，应用程序必须执行相应操作。在这种情况下，调用在后端定义的 C++ 函数 blink()、on() 和 off()。

```
12      Row {
13          id: col
14          anchors.verticalCenter: parent.verticalCenter
15          anchors.horizontalCenter: parent.horizontalCenter
16          spacing: 25
17
18          Button {
19              id: blink
20              text : qsTr("Blink");
21
22              onClicked: {
23                  test.blink();
24              }
25          }
26
27          Button {
28              id: on
29              text : qsTr("Turn On LED");
30
31              onClicked: {
32                  test.on();
33              }
34          }
35
36          Button {
37              id: off
38              text : qsTr("Turn Off LED");
39
40              onClicked: {
41                  test.off();
42              }
43          }
44      }
45  }
46
```

图 2-21. 调用 C++ 函数

现在，当用户运行应用程序并点击窗口中三个按钮中的每一个时，终端中将显示以下结果：

```
Blink_Tutorial
14:38:55: Starting /home/a0511311/build-Blink_Tutorial-Desktop-Debug/Blink_Tutorial...
QML debugging is enabled. Only use this in a safe environment.
14:39:16: /home/a0511311/build-Blink_Tutorial-Desktop-Debug/Blink_Tutorial exited with code 0

14:39:46: Starting /home/a0511311/build-Blink_Tutorial-Desktop-Debug/Blink_Tutorial...
QML debugging is enabled. Only use this in a safe environment.
Blinking
Turning LED On
Turning LED Off
```

图 2-22. 每个函数的输出

2.5 连接到 EVM

现在，按照以下步骤使 LED 实际闪烁。在 C++ 应用程序中，返回并修改 blink_command.cpp 文件（或对应命名文件）内的打印语句，以实现对板载 LED 的控制。对于 AM62P，请使用以下命令：

闪烁 LED：

system("echo heartbeat > /sys/class/leds/am62-sk\\:green\\:heartbeat/trigger");

打开 LED：

system("echo 1 > /sys/class/leds/am62-sk\\:green\\:heartbeat/brightness");

关闭 LED：

system("echo none > /sys/class/leds/am62-sk\\:green\\:heartbeat/trigger");

注意：如果用户使用的是不同的电路板，那么上述语句可能会略有不同。

目前无需关注终端输出，因为程序没有在电路板上运行。只有在电路板上安装了程序后，程序才会理解上述命令。

2.6 为您的嵌入式平台构建和部署应用程序

第一个 Qt 应用程序现已完成。下一步是构建工程并生成可在 AM62P（或其他 AM6x）电路板上运行的可执行文件。

2.6.1 构建应用程序

为了简化编译过程，请使用 Docker 映像。以下是通过 Docker 映像构建的步骤。首先，将 CD 放入项目目录。以 Blink_Tutorial 为例。

拉取 TI 的 debian-arm64 Docker 映像并运行映像：

docker pull ghcr.io/texasinstruments/debian-arm64:latest

docker run -it -v ${PWD}:/root/workspace ghcr.io/texasinstruments/debian-arm64:latest bash

在容器内，使用以下命令创建工程：

cd /root/workspace

cmake -B build -S .

make -C build

这会在 Build 文件夹内创建一个可执行文件，然后用户可以将其发送到 AM62P。

2.6.2 将可执行文件传输到电路板

使用以下命令，通过以太网复制可执行文件：

scp file_name root@ip_address:/destination/path

2.7 运行应用程序

启动 Qt 应用程序之前，启用 Weston。要执行此操作，请导出 Wayland 显示：

export WAYLAND_DISPLAY=/run/user/1000/wayland-1

现在运行应用程序： *./ProjectName*

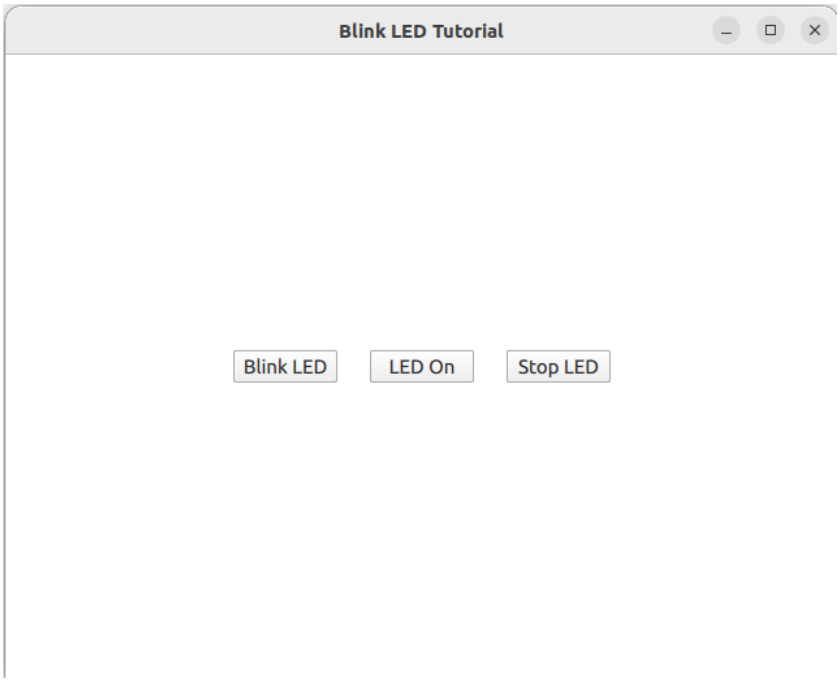

图 2-23. 在电路板上运行应用程序

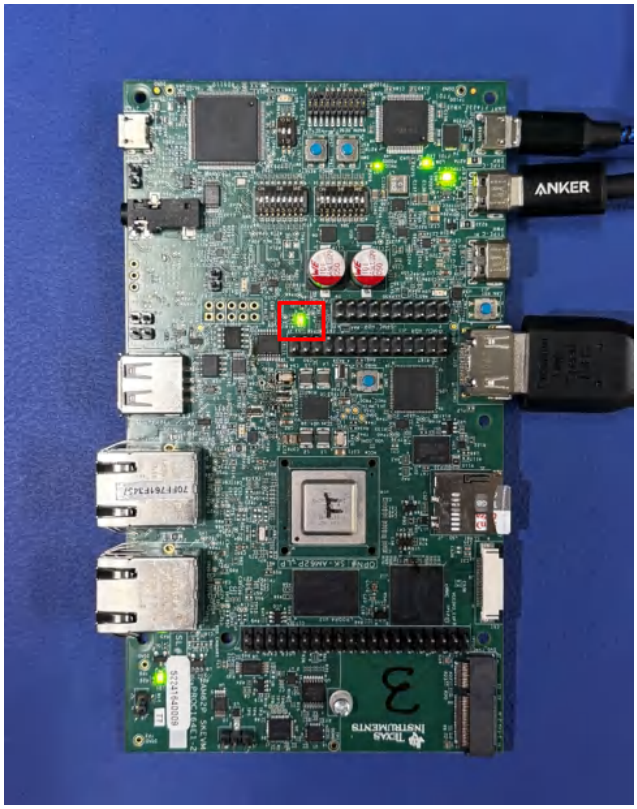

图 2-24. 在板上闪烁 LED

恭喜！Qt 工程现已成功启动并在板上正常显示！

单击每个按钮时，上面显示的绿色 LED 会闪烁、保持亮起或熄灭。

3 总结

完成这些步骤后，用户将能够通过 Qt 开发 GUI，并在嵌入式平台上部署 GUI。在解释了基础知识之后，用户现在可以开发更复杂的 GUI。

4 参考资料

1. 德州仪器 (TI)， *闪烁 LED* 资源管理器。
2. 德州仪器 (TI)，*1.概述 — 处理器 SDK AM62Px 文档* 网页。
3. 德州仪器 (TI)，*Qt 框架 – 构建快速、可扩展的跨平台软件 | Qt* 网页。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月