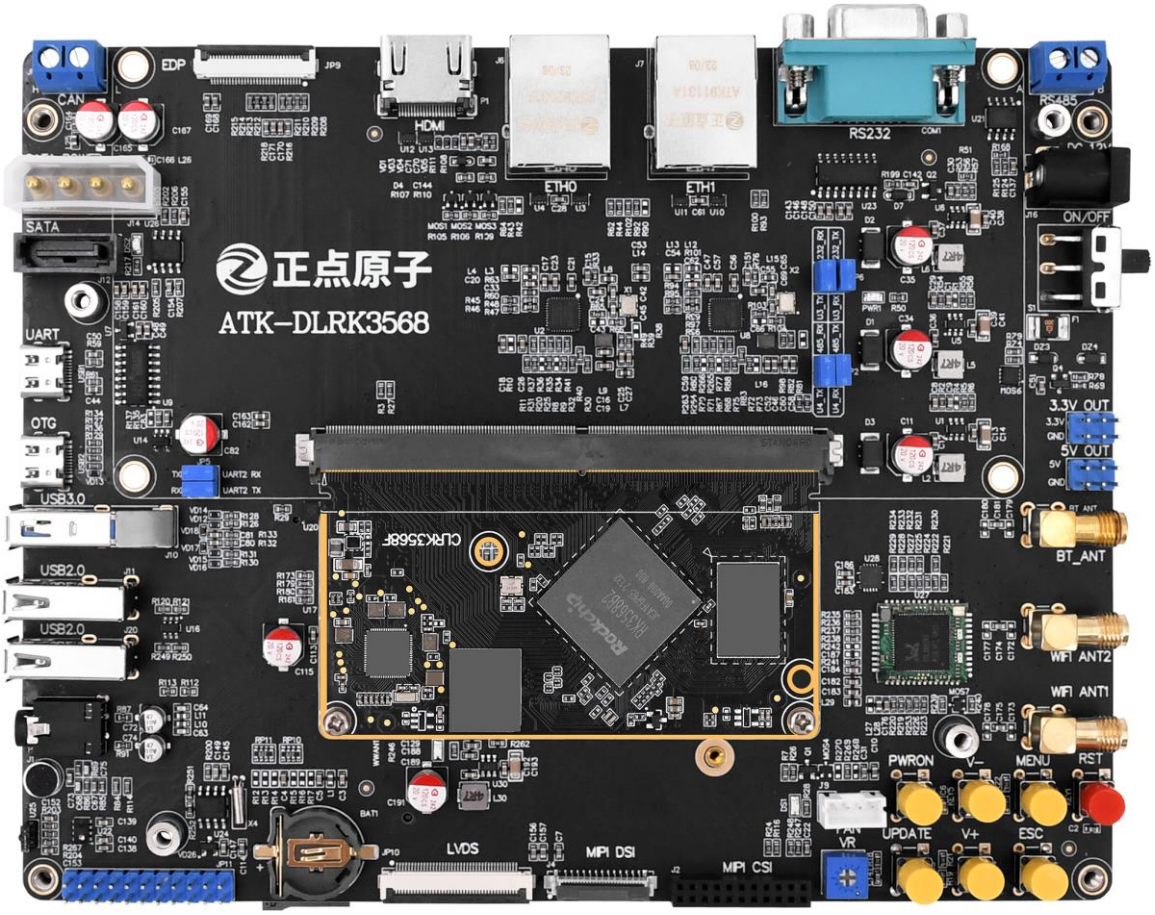
ATK-DLRK3568 嵌入式 Qt 开发实例 V1.0



正点原子

正点原子公司名称 ：广州市星翼电子科技有限公司

原子哥在线教学平台：www.yuanzige.com

开源电子网 / 论坛 ：http://www.openedv.com/forum.php

正点原子淘宝店铺 ：https://openedv.taobao.com

正点原子官方网站 ：www.alientek.com

正点原子 B 站视频 ：

https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请关注正点原子公众号，资料发布更新我们会通知。

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

扫码关注正点原子公众号

扫码下载“原子哥”APP

文档更新说明

版本	版本更新说明	负责人	校审	发布日期
V1.0	初稿：	正点原子 linux 团队	正点原子 linux 团队	2024.07.6

目录

前言

本文档介绍如何使用 Qt 简单操控 Linux 开发板上的硬件及使用对应的硬件接口进行收发数据控制等。

本文档主要以 Ubuntu 下开发为例，Windows 下开发测试需要自行测试。另外本教程不会讲解 Qt 基础知识，读者需具备简单的 Qt 基础。

本文档旨在引领大家在 Linux 环境下进行 Qt 快速开发，内容精简，希望对大家有所帮助，水平有限，如有错误，请联系作者 QQ 2101443104 指正，或者到正点原子论谈 www.openedv.com 发帖讨论，或发邮件到 shihoude@alientek.com，附上截图和文字说明，笔者会及时更新，谢谢大家。

本文档教程用的 Qt 版本为 Qt5.15.2。Ubuntu 版本为 20.04，建议与笔者的版本一样，此 Qt 版本为长期支持版本。

建议读者会使用 FileZilla、WinSCP 及 Windows Git 进行 Ubuntu 与 Windows 间互传文件的方法。还需要读者会使用 adb 指令，熟悉与开发板互传文件。

免责声明

本文档所提及的产品规格和使用说明仅供参考，如有内容更新，恕不另行通知；除非有特殊约定，本文档仅作为产品指导，所作陈述均不构成任何形式的担保。

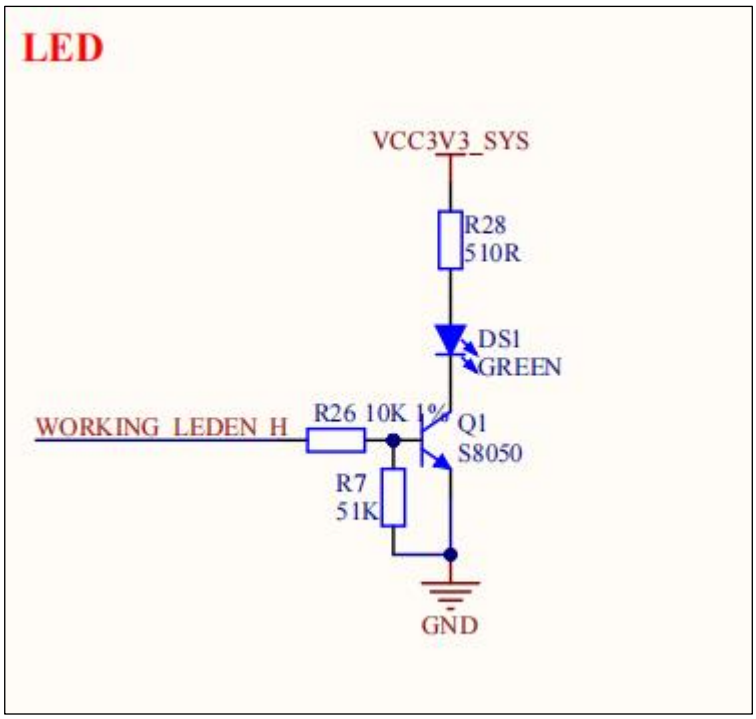
为了得到最新版本的产品信息，请用户定时访问正点原子资料下载中心或者与淘宝正点原子旗舰店客服联系索取。感谢您的包容与支持。

第一章 Qt 如何使用 LED

本章开始使用 Qt 应用到正点原子的嵌入式 ATK-DLRK3568 开发板上，凡事是先易后难，我们也是从最简单的点亮 LED 说起。介绍如何使用 Qt 知识应用到正点原子的嵌入式 ATK-DLRK3568 开发板，亦可参考来修改到其他平台的嵌入式 Linux 开发板上。

1.1 资源简介

在正点原子的 ATK-DLRK3568 开发板板载资源上有一个 LED。如下图原理图。

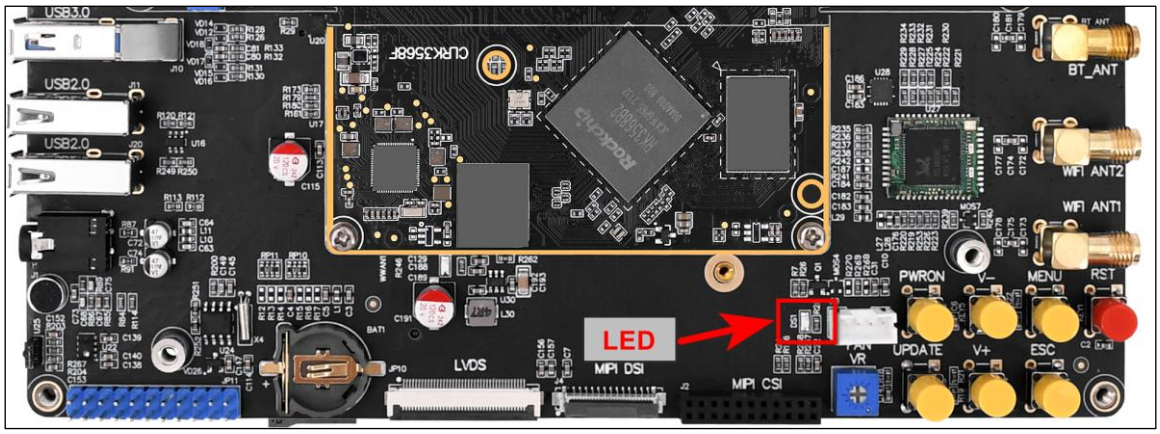


想要控制这个 LED，首先我们需要使用正点原子的出厂系统来实现，其他系统请自行测试解决。

我们可以直接在应用层接口直接可以操作这个 LED 设备。我们在 Qt 里有很多种方法可以控制 Linux 开发板的 LED 设备。还可以用 C 语言的读写函数读写来控制 LED 的状态，或者直接使用 system()函数启动一个进程执行相关指令直接控制 LED 等。

我们介绍最简单的方法控制开发板上的 LED，就是使用 Qt 的操作文件的类直接控制 LED。因为 Linux 上一切皆文件，所有的东西都当作文件来处理。

下面将贴上代码，其中不会再去讲如何搭建工程，不会贴上实验现象图。代码注释详细，不额外说明。实现现象请自行编译到开发板上运行查看。项目虽然简单，但是在嵌入式里基本都是从点亮一个 LED 里开始说起，只要我们会操作一个 IO，剩下的基本都不会难。另外编译运行 LED 程序的时候请注意观察如下开发板上的 LED 灯状态，如下是 ATK-DLRK3568 开发板上面的 LED 位置。如图



1.2 应用实例

项目简介：设置窗口按钮，点击即可控制 LED 状态反转（点亮或者熄灭 LED）。项目看来很起来很简单，实际上有些需要注意的地方，我们在改变 LED 的状态时，需要先去读取 LED 的状态，防止外界（外面应用程序）将 LED 的状态改变了。否则我们反转操作将不成立。

另外，我们这里在 UI 界面里面不需要添加任何控件，但是在创建项目的时候 UI 界面需要带上

例 01_led,控制 LED(难度简单)。项目路径为 01、程序源码\09、Qt 开发实例例程\01_led。在源文件“mainwindow.h”的代码如下

```
/**********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**********************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QVBoxLayout>
#include <QFile>
#include <QDebug>
#include <QLabel>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QMainWindow *parent = nullptr);
    ~MainWindow();

private:
    QPushButton *pushButton;
    /*设置 LED 状态*/
```

```
26      void setLedState();
27
28      /*获取 LED 状态*/
29      bool getLedState();
30
31  private:
32      Ui::MainWindow *ui;
33
34  private slots:
35      void controlLed(bool checked);
36  };
37  #endif // MAINWINDOW_H
```

在头文件“mainwindow.h”里第 24 行声明一个设置 LED 状态方法，另一个是获取状态的方法。另外声明一个槽函数，作用是点击切换 LED 的状态。

在源文件“mainwindow.cpp”的代码如下。

```
/*********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
*********************************************************************/
1   #include "mainwindow.h"
2   #include "ui_mainwindow.h"
3   #include <QDirIterator>
4   #include <QDir>
5   #include <QFileInfoList>
6
7   MainWindow::MainWindow(QMainWindow *parent)
8       : QMainWindow(parent)
9       , ui(new Ui::MainWindow){
10
11      ui->setupUi(this);
12      this->resize(480,232);
13      QString paths = "/sys/class/leds/";
14
15      /*创建一个 QDir 对象，该对象表示一个目录，并提供目录相关的操作*/
16      QDir dir(paths);
17      if (!dir.exists()) {
18          qDebug() << "Directory does not exist:" ;
19          return;
```

```
20      }
21
22      /*定义一个 QStringList 对象 ledpaths，用于存储 LED 的路径*/
23      QStringList ledpaths;
24      ledpaths.clear();
25      QFileInfoList fileInfoList = dir.entryInfoList(QDir::Files |
QDir::Dirs | QDir::NoDotAndDotDot);
26      /*遍历文件和子目录列表*/
27      for (const QFileInfo &fileInfo : fileInfoList) {
28
29          /*把此目录下带 emmc 的文件名过滤掉*/
30          if (!fileInfo.fileName().contains("mmc")) {
31              qDebug() << fileInfo.fileName();
32
33              /*构造 LED 的 brightness 文件路径*/
34              QString str = paths + fileInfo.fileName() + "/brightness";
35              qDebug() << str;
36              ledpaths.append(str);
37          }
38      }
39
40      if (ledpaths.count() == 0)
41      return;
42      QPushButton *buttons[ledpaths.count()];
43
44      /*遍历 ledpaths 列表中的每个 LED 路径*/
45      for (int i = 0; i < ledpaths.count(); i++) {
46          buttons[i] = new QPushButton();
47          buttons[i]->setFixedSize(300, 200);
48          buttons[i]->setText("<br>");
49          ui->centralwidget->layout()->addWidget(buttons[i]);
50          buttons[i]->setText(ledpaths[i]);
51          buttons[i]->setCheckable(true);
52
53          QString ledDir =
ledpaths[i].left(ledpaths[i].lastIndexOf('/'));
54          buttons[i]->setProperty("ledPath", ledDir);
55
56          /*信号槽*/
57          connect(buttons[i], &QPushButton::clicked, this,
&MainWindow::controlLed);
58      }
59  }
```

```
60
61  MainWindow::~MainWindow()
62  {
63      delete ui;
64  }
65
66  void MainWindow::controlLed(bool checked)
67  {
68      QPushButton *button = (QPushButton*)sender();
69      QString ledPath = button->property("ledPath").toString();
70      QString brightnessPath = ledPath + "/brightness";
71
72      QFile file(brightnessPath);
73      if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
74          QTextStream out(&file);
75
76          /*根据按钮的选中状态写入对应的值（"1"表示开，"0"表示关）*/
77          out << (checked ? "1" : "0");
78          file.close();
79          qDebug() << (checked ? "开" : "关");
80      }
81      else {
82          qDebug() << "Failed to open file:" << brightnessPath;
83      }
84  }
```

第 12 行~38 行，设置 UI 界面窗口大小。遍历文件和子目录列表。然后过滤掉其他文件目录，比如我们过滤掉的 emmc 文件。

第 68 行，获取触发事件的按钮来控制 LED 的状态。

第 77~82 行设置 LED 的方法，写入“0”或“1”代表开和关。写入之前先读取 LED 的状 态，预防在用户其他地方有设置过 LED。至此常规的控制一个 IO，大概流程已经完成。

1.3 程序运行效果

下面将程序交叉编译到开发板运行的效果，可以点击下图小窗口进行对 ATK-DLRK3568 开发板上面的 LED 灯进行控制。

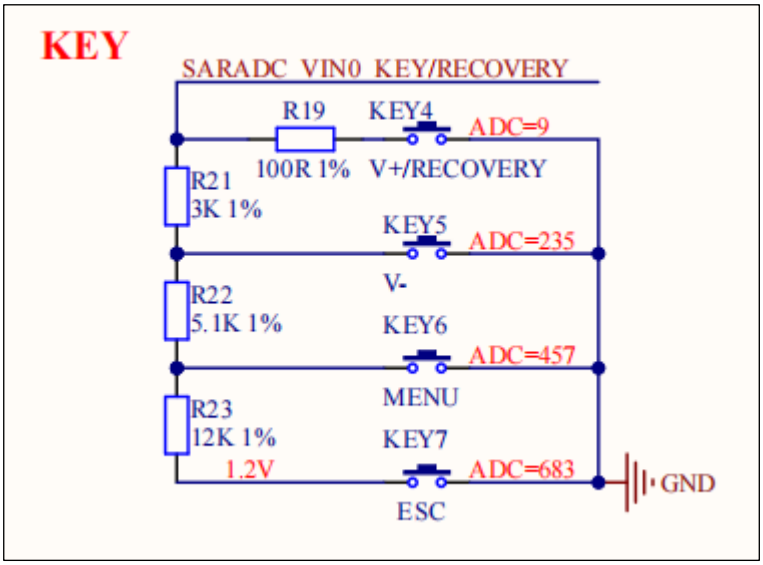

MainWindow
/sys/class/leds/work/brightness

第二章 Qt 如何使用 KEY

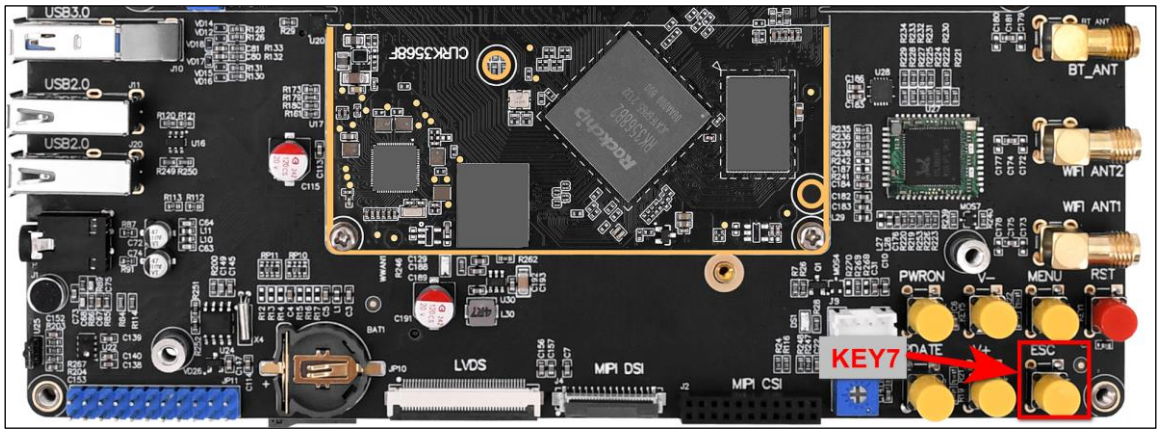
本章是按键实验，介绍如何在 Qt 应用上使用正点原子嵌入式 ATK-DLRK3568 开发板上的按键。

2.1 资源简介

在正点原子的 ATK-DLRK3568 开发板，ATK-DLRK3568 开发板板载资源上有七个可用按键，板子上还有其它的按键为功能按键，这里我们用的是 KEY7 按键，原理图如下所示。



正点原子 ATK-DLRK3568 开发板上面有很多按键，下图是开发板板载的实体按键，也是本项目实验测试的按键。如图。



2.2 应用实例

想要监测这个 KEY7，首先正点原子的出厂内核已经默认将这个按键注册成了 gpio-keys 类型设备，键值为 KEY_ESC 也就是对应 Qt 的 KEY7 键值。

我们本例程可以运行在虚拟机上面然后使用电脑键盘 Esc 按键（键盘方向键 Esc）来检测按键测试，在开发板上使用 KEY7 按键测试。

在开发板监测这个 KEY7 有很多方法。比如使用 C 语言开一个线程监测这个按键，或者按本例重写键盘事件来监测 KEY7 按键按下或者松开。

项目简介：监测 KEY7 按键的按下和松开。使用一个标签文本，通过按键按下来改变标签文本的文字属性。

例 02_key，监测 KEY7（难度简单）。项目路径为 01、程序源码\09、Qt 开发实例例程\02_key。

在源文件“mainwindow.h”的代码如下。

```
/**************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QKeyEvent>
#include <QLabel>
#include <QDebug>
#include <QEvent>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 标签文本 */
    QLabel *label;

    /*label 颜色*/
    QColor backgroundColor;

    /* 重写按键事件 */
    void keyPressEvent(QKeyEvent *event);
    void keyReleaseEvent(QKeyEvent *event);
};

#endif // MAINWINDOW_H
```

第 23~24 行，声明需要重写的按键事件类型。分别是按下事件和松开事件。通过重写这两个事件可以监测到键盘或 KEY7 按下的状态。

在源文件“mainwindow.cpp”的代码如下。

```
/*********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
*********************************************************************/
1   #include "mainwindow.h"
2   #include <QGuiApplication>
3   #include <QScreen>
4   #include <QRect>
5
6   MainWindow::MainWindow(QWidget *parent)
7       : QMainWindow(parent)
8   {
9       /* 获取屏幕的分辨率，Qt 官方建议使用这
10       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
11       * 注意，这是获取整个桌面系统的分辨率
12       */
13      QList <QScreen *> list_screen =   QGuiApplication::screens();
14
15      /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
16  #if __arm__
17      /* 重设大小 */
18      this->resize(list_screen.at(0)->geometry().width(),
19                   list_screen.at(0)->geometry().height());
20  #else
21      /* 否则则设置主窗体大小为 800x480 */
22      this->setGeometry(0, 0, 800, 480);
23  #endif
24
25      /* 标签实例化 */
26      label = new QLabel(this);
27
28      /* 设置默认文本 */
29  #if __arm__
30      label->setText("松开状态");
31  #else
32      label->setText("按键松开");
```

```
33  #endif
34
35      /* 设置对齐方式 */
36      label->setAlignment(Qt::AlignCenter);
37
38      /* 居中显示 */
39      setCentralWidget(label);
40  }
41
42  MainWindow::~MainWindow()
43  {
44  }
45
46  void MainWindow::keyPressEvent(QKeyEvent *event)
47  {
48  #if __arm__
49      /* 判断按下的按键，也就是板子 KEY7 按键 */
50      if(event->key() == Qt::Key_Escape) {
51          /* 设置 label 的文本 */
52          label->setText("Esc 按键按下");
53      }
54
55  #else
56      /* 判断按下的按键，也就是 KEY7 键 */
57      if(event->key() == Qt::Key_Escape) {
58          /* 设置 label 的文本 */
59          label->setText("Esc 按键按下");
60          backgroundColor = Qt::cyan; // 按键按下时背景变为对应的颜色
61          label->setStyleSheet("QLabel { background-color: " + backgroundColor.name() + "; }");
62      }
63
64  #endif
65      /* 保存默认事件 */
66      QWidget::keyPressEvent(event);
67  }
68
69  void MainWindow::keyReleaseEvent(QKeyEvent *event)
70  {
71  #if __arm__
72      /* 判断松开的按键，也就是板子 KEY7 按键 */
73      if(event->key() == Qt::Key_Escape) {
74          /* 设置 label 的文本 */
75          label->setText("Esc 按键松开");
```

```
76      }
77
78  #else
79      /* 判断按下的按键，也就是 KEY7 键 */
80      if(event->key() == Qt::Key_Escape) {
81          /* 设置 label 的文本 */
82          label->setText("Esc 按键松开");
83          backgroundColor = Qt::white; // 假设按键松开时背景恢复为白色
84          label->setStyleSheet("QLabel { background-color: " + backgroundColor.name() + "; }");
85      }
86
87  #endif
88      /* 保存默认事件 */
89      QWidget::keyReleaseEvent(event);
90  }
```

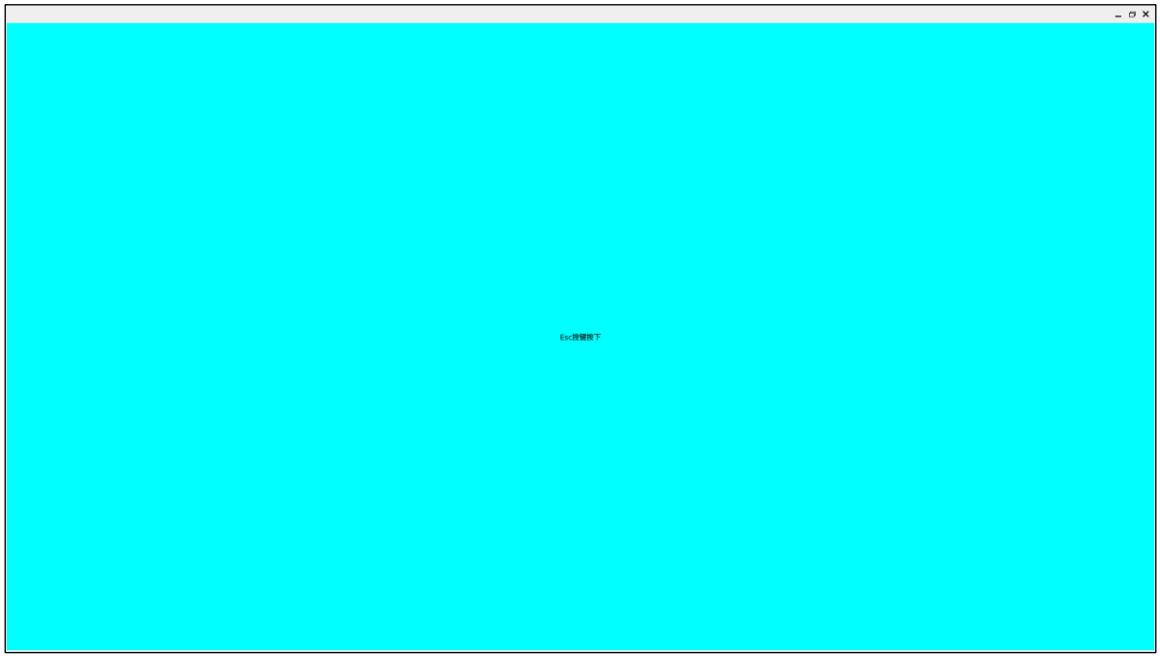
第 9 行~23 行，界面初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示，按钮居中显示。

第 46~83 行，重写按下事件和松开事件，通过判断 event->key()等哪个按键，就可以知道是哪个按键按下或者松开了。并设置了标签文本的属性颜色。

2.3 程序运行效果

下面将程序交叉编译后到开发板运行，按下 ATK-DLRK3568 开发板上面的 KEY7 按钮的时候，标签文本的值会改变为“Esc 按键按下，窗口颜色会改变”，当松开 KEY7 按钮时，标签的文本值会改变为默认状态“Esc 按键松开，背景颜色会变成白色，非常有趣”。

说明：ATK-DLRK3568 开发板上面 KEY7 按钮对应的是 ESC 按键值。



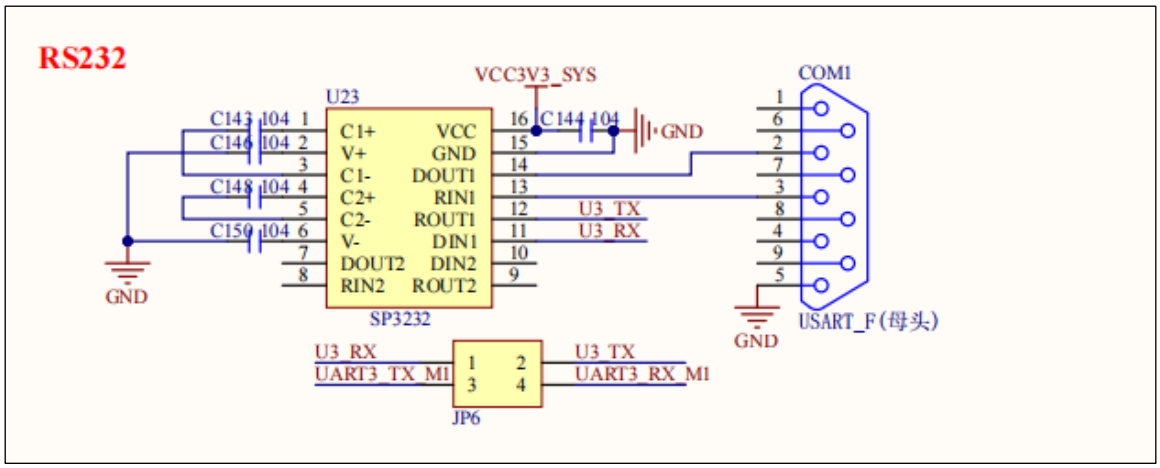
第三章 Qt 如何使用 RS232

Qt 提供了串口类，可以直接对串口访问。我们可以直接使用 Qt 的串口类编程即可，十分方便，Qt 串口类不仅在 Windows 能用，还能在 Linux 下用，不过这里我们只在 Linux 下进行使用。

既然 Qt 提供了这方面的接口，我们就充分利用起来，这将会使我们的开发十分方便！其实 Qt 也提供了相关的 Qt 串口的例子，我们也可以直接参考来编程，不过本小节笔者是通过参考我们经常用的 XCOM 串口工具进行简单的修改实现了一个带 UI 界面的串口工具给大家参考。

3.1 资源简介

在正点原子的 ATK-DLRK3568 开发板的出厂系统里，默认已经配置了两路串口可用。一路是 UART2 一 路是 UART3，我们本例程使用的是 UART3，原理图如下所示。



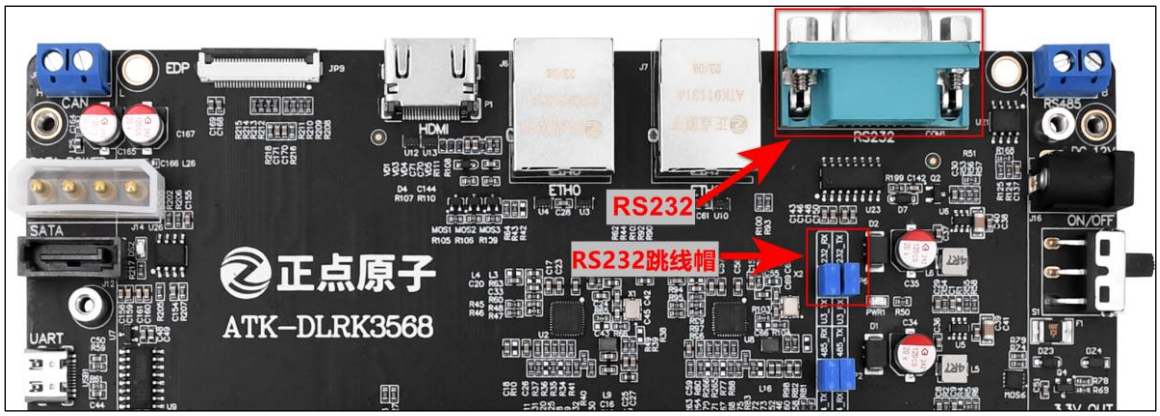
串口 UART2(对应系统里的节点/dev/tty2)，另一路是 UART3(对应系统里的节点/dev/tty3)。

由于 UART2 已经作为调试串口被使用。所以我们只能对 UART3 编程，（如需要使用多路串口，请自行设计底板与系统）。

另外 在正点原子“RK3568 开发板\开发板光盘 A 盘-基础资料\10、用户手册\01、测试文档\01【正点原子】ATK-DLRK3568_Buildroot 系统快速体验手册 V1.2.pdf”文档功能测试小节有 RS232 的测试教程，这里就不再做多的叙述。

说明：开发板上的 UART3 连接的是 RS232 DB9 接口（母头），如果需要测试这个串口，那么就需要有一个 USB 转 RS232 的串口线（USB 转 RS232 的串口线属于非开发板配件，请自行配备），而且需要是公头的（正点原子店铺有售卖），如下图。

注意开发板跳线帽的位置需要接到 UART3 上面，开发板 RS232 接口的位置如下。



3.2 UI 界面设计

我们这里在 UI 界面里面添加相关控件进行部署，通过 UI 进行简单的界面设计，然后在通过程序去实现相关功能即可。如下。

3.3 应用实例

项目简介：Qt 串口的使用示例，应用到正点原子 ATK-DLRK3568 开发板上。例 03_RS232，Qt 串口编程（难度：一般）。项目路径为 01、程序源码\09、Qt 开发实例例程\03_ RS232。

在 03_ RS232.pro 里，我们需要使用串口，需要在 pro 项目文件中添加串口模块的支持，如下。

```
1  QT       += core gui serialport
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++17
```

```
6
7   # You can make your code fail to compile if it uses deprecated APIs.
8   # In order to do so, uncomment the following line.
9   #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
10
11  SOURCES += \
12      main.cpp \
13      mainwindow.cpp
14
15  HEADERS += \
16      mainwindow.h
17
18  FORMS += \
19      mainwindow.ui
20
21  # Default rules for deployment.
22  qnx: target.path = /tmp/$${TARGET}/bin
23  else: unix:!android: target.path = /opt/$${TARGET}/bin
24  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加的 serialport 就是串口模块的支持。

在头文件“mainwindow.h”的代码如下。

```
/**********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**********************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QSerialPort>
6   #include <QMessageBox>
7   #include <QFileDialog>
8   #include <QSerialPortInfo>
9   #include <QStandardPaths>
10  #include <QWidget>
11  #include <QTimer>
12  #include <QDebug>
```

```
13  #include <QFile>
14
15  QT_BEGIN_NAMESPACE
16  namespace Ui { class MainWindow; }
17  QT_END_NAMESPACE
18
19  class MainWindow : public QMainWindow
20  {
21      Q_OBJECT
22
23  public:
24      MainWindow(QWidget *parent = nullptr);
25      ~MainWindow();
26
27      /*初始化*/
28      void init();
29
30  private slots:
31
32      /*打开串口*/
33      void on_openPortBtn_released();
34
35      /*发送数据*/
36      void on_sendBtn_released();
37
38      /*接收数据*/
39      void onReadyRead();
40
41      /*16 进制显示*/
42      void on_hexDisplayChx_toggled(bool checked);
43
44  private:
45
46      /*是否用 16 进制显示接收的数据*/
47      void displayHex();
48
49      /*文本形式显示接收到的数据*/
50      void displayText();
51
52      Ui::MainWindow *ui;
53      QSerialPort serial;
54      QTimer timer;
55  };
```

```
56  #endif // MAINWINDOW_H
```

上面代码是在 mianwindow.h 里声明需要用到的变量，方法及槽函数。

mainwindow.cpp 的代码如下

```
/**************************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************************/
1   #include "mainwindow.h"
2   #include "ui_mainwindow.h"
3
4
5   MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7       , ui(new Ui::MainWindow)
8   {
9       ui->setupUi(this);
10      init();
11  }
12
13  MainWindow::~MainWindow()
14  {
15      delete ui;
16  }
17
18  /*初始化串口*/
19  void MainWindow::init()
20  {
21
22      setWindowTitle("Qt 串口助手");
23
24      /*1.获取可用的串口设备*/
25      auto portsInfo = QSerialPortInfo::availablePorts();
26      for(auto& info : portsInfo)
27      {
28
qInfo()<<info.description()<<info.portName()<<info.systemLocation();
29          ui->protsCmb->addItem(info.portName() + ":" +
info.description(),info.portName());
```

```
30      }
31      /*2.获取标准的波特率*/
32      auto baudRates = QSerialPortInfo::standardBaudRates();
33      for(auto br : baudRates)
34      {
35          ui->baudRateCmb->addItem(QString::number(br),br);
36      }
37
38      ui->baudRateCmb->setCurrentText("9600");
39
40      /*3.设置停止位*/
41      ui->stopBitsCmb->addItem("1",   QSerialPort::OneStop);
42      ui->stopBitsCmb->addItem("1.5", QSerialPort::OneAndHalfStop);
43      ui->stopBitsCmb->addItem("2",   QSerialPort::TwoStop);
44      /*4.设置数据位*/
45      ui->dataBitsCmb->addItem("5", QSerialPort::Data5);
46      ui->dataBitsCmb->addItem("6", QSerialPort::Data6);
47      ui->dataBitsCmb->addItem("7", QSerialPort::Data7);
48      ui->dataBitsCmb->addItem("8", QSerialPort::Data8);
49      ui->dataBitsCmb->setCurrentText("8");
50      /*5.设置校验位*/
51      ui->parityCmb->addItem("NoParity",    QSerialPort::NoParity);
52      ui->parityCmb->addItem("EvenParity",  QSerialPort::EvenParity);
53      ui->parityCmb->addItem("OddParity",   QSerialPort::OddParity);
54      ui->parityCmb->addItem("SpaceParity", QSerialPort::SpaceParity);
55      ui->parityCmb->addItem("MarkParity",  QSerialPort::MarkParity);
56
57
connect(&serial,&QSerialPort::readyRead,this,&MainWindow::onReadyRead);
58
59      ui->sendEdit->setPlainText("www.openedv.com");
60      timer.callOnTimeout([=]
61                          {
62                              this->on_sendBtn_released();
63                          });
64
65      connect(ui->claerRecvBtn,&QPushButton::clicked,
66              ui->recvEdit,&QPlainTextEdit::clear);
67      connect(ui->sendClearBtn,&QPushButton::clicked,
68              ui->sendEdit,&QPlainTextEdit::clear);
69  }
70
71  /*打开串口*/
```

```
72  void MainWindow::on_openPortBtn_released()
73  {
74      /*判断串口是否已经打开*/
75      if(serial.isOpen())
76      {
77          serial.close();
78          ui->openPortBtn->setText("打开串口");
79          if(timer.isActive())
80              timer.stop();
81          return;
82      }
83      /*获取串口名字*/
84      auto portName = ui->protsCmb->currentData().toString();
85      /*获取波特率*/
86      auto baudRate =
ui->baudRateCmb->currentData().value<QSerialPort::BaudRate>();
87      /*获取数据位*/
88      auto dataBits =
ui->dataBitsCmb->currentData().value<QSerialPort::DataBits>();
89      /*获取停止位*/
90      auto stopBits =
ui->stopBitsCmb->currentData().value<QSerialPort::StopBits>();
91      /*获取校验位*/
92      auto parity   =
ui->parityCmb->currentData().value<QSerialPort::Parity>();
93
94      serial.setPortName(portName);   /*设置串口名字*/
95      serial.setBaudRate(baudRate);   /*设置波特率*/
96      serial.setDataBits(dataBits);   /*设置数据位*/
97      serial.setStopBits(stopBits);   /*设置停止位*/
98      serial.setParity(parity);      /*设置校验位*/
99      serial.setPortName(portName);
100
101     /*打开串口*/
102     if(!serial.open(QIODevice::ReadWrite))
103     {
104         QMessageBox::warning(this,"warning",portName+ "open filed:"
+serial.errorString());
105         return;
106     }
107     else
108     {
109         ui->openPortBtn->setText("关闭串口");
```

```
110     }
111 }
112
113 /*发送窗口*/
114 void MainWindow::on_sendBtn_released()
115 {
116     auto dataStr = ui->sendEdit->toPlainText();
117     serial.write(dataStr.toLocal8Bit());
118 }
119
120 /*接收窗口*/
121 void MainWindow::onReadyRead()
122 {
123     auto data = serial.readAll();
124     ui->recvEdit->setPlainText(QString::fromLocal8Bit(data) +
ui->recvEdit->toPlainText());
125 }
126
127 void MainWindow::on_hexDisplayChx_toggled(bool checked)
128 {
129     if(checked)
130         displayHex();
131     else
132         displayText();
133 }
134
135 void MainWindow::displayHex()
136 {
137     /*先读取数据*/
138     auto dataStr = ui->recvEdit->toPlainText();
139     /*转成十六进制*/
140     auto hexData = dataStr.toLocal8Bit().toHex(' ').toUpper();
141     /*再把数据写回去*/
142     ui->recvEdit->setPlainText(hexData);
143 }
144
145 void MainWindow::displayText()
146 {
147     /*先读取数据*/
148     auto dataStr = ui->recvEdit->toPlainText();
149     /*转成文本形式*/
150     auto textData = QString::fromLocal8Bit(dataStr.toLocal8Bit());
151     /*再把数据写回去*/
```

```
152     ui->recvEdit->setPlainText(textData);
153 }
```

第 28~47 行，查找系统可用的串口和设置波特率。
第 50~52 行，设置停止位，默认停止位为 1。
第 54~58 行，设置数据位，默认数据位为 8。
第 60~64 行，设置数据位，默认无校验位。
第 66~76 行，把相关功能通过信号槽关联起来
第 80~109 行，判断串口状态，打开或者关闭串口
第 122~133 行，发送窗口和接收窗口
第 135~161 行，接收是否通过 16 进制进行显示 。

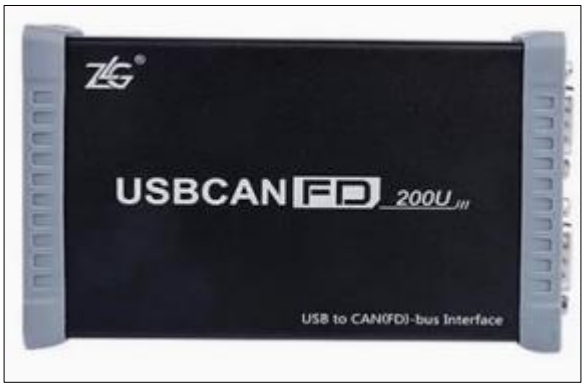
3.4 程序运行效果

下面将程序交叉编译到开发板运行，用前面 3.1 资源简介小节所介绍的 USB 转 RS232 的串口线和开发板的 RS232 连接 ，注意：跳线帽需要跳接到对应的 UART3 上面，这里结合正点原子的 XCOM 上位机工具(或者本程序亦可当上位机软件)测试，其他工具可自行测试，接好后设置相同的串口参数，选择串口号为 tty3，设置波特率为 115200，数据位为 8，校验为 None，停止位为 1，最后点击打开串口就可以进行消息收发了。

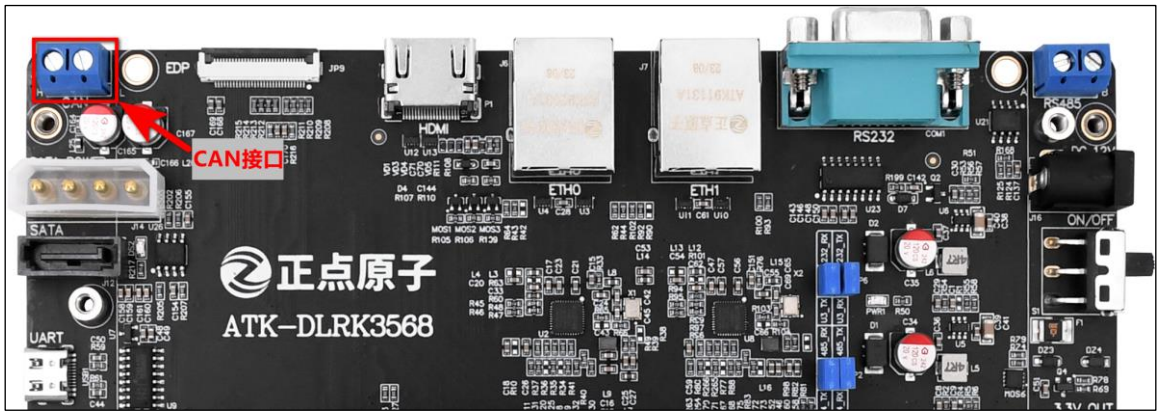
第四章 Qt 如何使用 CAN

从 Qt5.8 开始，提供了 CAN Bus 类，很庆幸，正点原子的 ATK-DLRK3568 出厂系统里 Qt 版本是 QT5.15.2。我们可以直接使用 Qt 的提供的 CAN 相关类编程即可。其实 Qt 也提供了相关的 Qt CAN 的例子，我们也可以直接参考来编程。笔者根据实际情况，化繁为易，直接写了个 简单的例子给大家参考。

本节笔者会带大家如何使用 Qt CAN 进行通信，CAN 测试的仪器需要大家自备！周立功的 USBCANFD 分析仪，如下图。

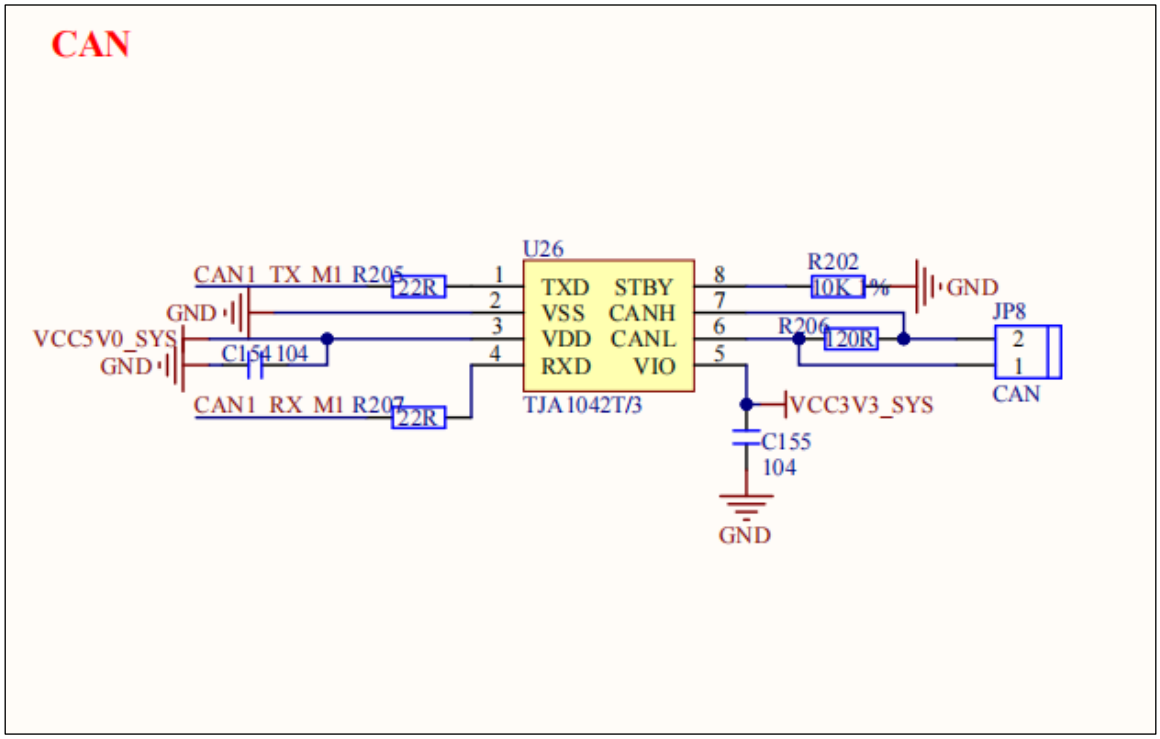


注意，关于 CANFD 分析仪的使用请咨询商家，本实验不讲解 USB CANFD 分析仪的用。开发板对应的 CAN 接口位置如下。



4.1 资源简介

正点原子 ATK-DLRK3568 开发板底板上预留了一路 CAN 接口（ATK-DLRK3568 芯片最大支持三路 CAN）。如下原理图。



4.2 应用实例

项目简介：本例适用于正点原子 ATK-DLRK3568 开发板。不适用于 Windows。因为 Windows 没有 CAN 设备。虽然 Windows 可以外接 USB CAN 模块，但是这些模块都是某些厂商开发的，需要有相应的固件才能驱动 CAN 设备。所以编写的例子不一定适用于 Windows 下的 CAN。笔者写的例子已经在正点原子 ATK-DLRK3568 开发板上验证了，确保正常使用！

在正点原子“RK3568 开发板\开发板光盘 A 盘-基础资料\10、用户手册\01、测试文档\01【正点原子】ATK-DLRK3568_Buildroot 系统快速体验手册 V1.2.pdf” 里也有相关的 CAN 测试方法。这里就不多介绍 CAN 了，笔者默认读者是会使用 CAN 的。同时不对 CAN 总线协议进行讲解，主要是讲解如何在 Qt 里对 CAN 编程。

例 04_can，Qt CAN 编程（难度：较难）。项目路径为 01、程序源码\09、Qt 开发实例例程\03_serialport。

04_can.pro 要想使用 Qt 的 QCanBus，需要在 pro 项目文件里添加相应的模块支持。同时还需要添加对应的头文件，详细请看项目里的代码。

```
1  QT       += core gui serialbus
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5  CONFIG += c++17
```

```
6
7   # You can make your code fail to compile if it uses deprecated APIs.
8   # In order to do so, uncomment the following line.
9   #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
10
11  SOURCES += \
12      main.cpp \
13      mainwindow.cpp
14
15  HEADERS += \
16      mainwindow.h
17
18  # Default rules for deployment.
19  qnx: target.path = /tmp/$${TARGET}/bin
20  else: unix:!android: target.path = /opt/$${TARGET}/bin
21  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加的 serialbus 就是添加串行总线模块的支持。

在头文件“mainwindow.h”的代码如下。一些声明。

```
/*****************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
*****************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include <QCanBusDevice>
6   #include <QCanBus>
7   #include <QPushButton>
8   #include <QTextBrowser>
9   #include <QLineEdit>
10  #include <QVBoxLayout>
11  #include <QLabel>
12  #include <QComboBox>
13  #include <QGridLayout>
14  #include <QMessageBox>
15  #include <QDebug>
```

```
16
17  class MainWindow : public QMainWindow
18  {
19      Q_OBJECT
20
21  public:
22      MainWindow(QWidget *parent = nullptr);
23      ~MainWindow();
24
25  private:
26      /* CAN 设备 */
27      QCanBusDevice *canDevice;
28
29      /* 用作接收数据 */
30      QTextBrowser *textBrowser;
31
32      /* 用作发送数据 */
33      QLineEdit *lineEdit;
34
35      /* 按钮 */
36      QPushButton *pushButton[2];
37
38      /* 下拉选择盒子 */
39      QComboBox *comboBox[3];
40
41      /* 标签 */
42      QLabel *label[4];
43
44      /* 垂直布局 */
45      QVBoxLayout *vboxLayout;
46
47      /* 网络布局 */
48      QGridLayout *gridLayout;
49
50      /* 主布局 */
51      QWidget *mainWidget;
52
53      /* 设置功能区域 */
54      QWidget *funcWidget;
55
56      /* 布局初始化 */
57      void layoutInit();
58
```

```
59      /* 插件类型项初始化 */
60      void pluginItemInit();
61
62      /* 比特率项初始化 */
63      void bitrateItemInit();
64
65  private slots:
66      /* 发送消息 */
67      void sendFrame();
68
69      /* 接收消息 */
70      void receivedFrames();
71
72      /* 插件发生改变 */
73      void pluginChanged(int);
74
75      /* 处理 can 错误 */
76      void canDeviceErrors(QCanBusDevice::CanBusError) const;
77
78      /* 连接或者断开 can */
79      void connectDevice();
80  };
81  #endif // MAINWINDOW_H
```

上面代码是在 mianwindow.h 里声明需要用到的变量，方法及槽函数。

mainwindow.cpp 的代码如下。

```
/*********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
*********************************************************************/
1   #include "mainwindow.h"
2   #include <QGuiApplication>
3   #include <QScreen>
4
5   MainWindow::MainWindow(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       /* 使用系统指令比特率初始化 CAN,默认为 1000000bits/s */
9       system("ifconfig can0 down");
```

```
10      system("ip link set up can0 type can bitrate 1000000 restart-ms
100");
11
12      /* 布局初始化 */
13      layoutInit();
14
15      /* 可用插件初始化 */
16      pluginItemInit();
17
18      /* 可用接口项初始化 */
19      pluginChanged(comboBox[0]->currentIndex());
20
21      /* 比特率项初始化 */
22      bitrateItemInit();
23  }
24
25  MainWindow::~MainWindow()
26  {
27  }
28
29  static QString frameFlags(const QCanBusFrame &frame)
30  {
31      /* 格式化接收到的消息 */
32      QString result = QLatin1String(" --- ");
33
34      if (frame.hasBitrateSwitch())
35          result[1] = QLatin1Char('B');
36      if (frame.hasErrorStateIndicator())
37          result[2] = QLatin1Char('E');
38      if (frame.hasLocalEcho())
39          result[3] = QLatin1Char('L');
40
41      return result;
42  }
43
44  /* 发送消息 */
45  void MainWindow::sendFrame()
46  {
47      if (!canDevice)
48          return;
49      /* 读取 QLineEdit 的文件 */
50      QString str = lineEdit->text();
51      QByteArray data = 0;
```

```
52      QString strTemp = nullptr;
53      /* 以空格分隔 lineEdit 的内容，并存储到字符串链表中 */
54      QStringList strlist = str.split(' ');
55      for (int i = 1; i < strlist.count(); i++) {
56          strTemp = strTemp + strlist[i];
57      }
58      /* 将字符串的内容转为 QByteArray 类型 */
59      data = QByteArray::fromHex(strTemp.toLatin1());
60
61      bool ok;
62      /* 以 16 进制读取要发送的帧内容里第一个数据，并作为帧 ID */
63      int framId = strlist[0].toInt(&ok, 16);
64      QCanBusFrame frame = QCanBusFrame(framId, data);
65      /* 写入帧 */
66      canDevice->writeFrame(frame);
67  }
68
69  /* 接收消息 */
70  void MainWindow::receivedFrames()
71  {
72      if (!canDevice)
73          return;
74
75      /* 读取帧 */
76      while (canDevice->framesAvailable()) {
77          const QCanBusFrame frame = canDevice->readFrame();
78          QString view;
79          if (frame.frameType() == QCanBusFrame::ErrorFrame)
80              view = canDevice->interpretErrorFrame(frame);
81          else
82              view = frame.toString();
83
84          const QString time = QString::fromLatin1("%1.%2  ")
85                               .arg(frame.timeStamp()
86                                       .seconds(), 10, 10, QLatin1Char('
'))
87                               .arg(frame.timeStamp()
88                                           .microSeconds() / 100, 4, 10,
QLatin1Char('0'));
89
90          const QString flags = frameFlags(frame);
91          /* 接收消息框追加接收到的消息 */
92          textBrowser->insertPlainText(time + flags + view + "\n");
```

```
93      }
94  }
95
96  void MainWindow::layoutInit()
97  {
98      /* 获取屏幕的分辨率，Qt 官方建议使用这
99       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
100      * 注意，这是获取整个桌面系统的分辨率
101      */
102     QList <QScreen *> list_screen =  QGuiApplication::screens();
103
104     /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
105 #if __arm__
106     /* 重设大小 */
107     this->resize(list_screen.at(0)->geometry().width(),
108                  list_screen.at(0)->geometry().height());
109 #else
110     /* 否则则设置主窗体大小为 800x480 */
111     this->resize(800, 480);
112 #endif
113     /* 对象初始化 */
114     textBrowser = new QTextBrowser();
115     lineEdit = new QLineEdit();
116     vboxLayout = new QVBoxLayout();
117     funcWidget = new QWidget();
118     mainWidget = new QWidget();
119     gridLayout = new QGridLayout();
120
121     /* QList 链表，字符串类型 */
122     QList <QString> list1;
123     list1<<"插件类型:"<<"可用接口:"<<"比特率 bits/sec:";
124
125     for (int i = 0; i < 3; i++) {
126         label[i] = new QLabel(list1[i]);
127         /* 设置最小宽度与高度 */
128         label[i]->setMinimumSize(120, 30);
129         label[i]->setMaximumHeight(50);
130         /* 自动调整 label 的大小 */
131         label[i]->setSizePolicy(QSizePolicy::Expanding,
132                                 QSizePolicy::Expanding);
133         /* 将 label[i]添加至网格的坐标(0, i) */
134         gridLayout->addWidget(label[i], 0, i);
135     }
```

```
136     label[3] = new QLabel();
137     label[3]->setMaximumHeight(30);
138
139     for (int i = 0; i < 3; i++) {
140         comboBox[i] = new QComboBox();
141         comboBox[i]->setMinimumSize(120, 30);
142         comboBox[i]->setMaximumHeight(50);
143         /* 自动调整 label 的大小 */
144         comboBox[i]->setSizePolicy(QSizePolicy::Expanding,
145                                    QSizePolicy::Expanding);
146         /* 将 comboBox[i]添加至网格的坐标(1, i) */
147         gridLayout->addWidget(comboBox[i], 1, i);
148     }
149
150     /* QList 链表，字符串类型 */
151     QList <QString> list2;
152     list2<<"发送"<<"连接 CAN";
153
154     for (int i = 0; i < 2; i++) {
155         pushButton[i] = new QPushButton(list2[i]);
156         pushButton[i]->setMinimumSize(120, 30);
157         pushButton[i]->setMaximumHeight(50);
158         /* 自动调整 label 的大小 */
159         pushButton[i]->setSizePolicy(QSizePolicy::Expanding,
160                                      QSizePolicy::Expanding);
161         /* 将 pushButton[0]添加至网格的坐标(i, 3) */
162         gridLayout->addWidget(pushButton[i], i, 3);
163     }
164     pushButton[0]->setEnabled(false);
165
166     /* 布局 */
167     vboxLayout->addWidget(textBrowser);
168     vboxLayout->addWidget(lineEdit);
169     funcWidget->setLayout(gridLayout);
170     vboxLayout->addWidget(funcWidget);
171     vboxLayout->addWidget(label[3]);
172     mainWidget->setLayout(vboxLayout);
173     this->setCentralWidget(mainWidget);
174
175     /* 设置文本 */
176     textBrowser->setPlaceholderText("接收的数据：帧 ID 长度 数据");
177     lineEdit->setText("123 aa 77 66 55 44 33 22 11");
178     label[3]->setText(tr("未连接！"));
```

```
179
180     connect(pushButton[1], SIGNAL(clicked()),
181             this, SLOT(connectDevice()));
182     connect(pushButton[0], SIGNAL(clicked()),
183             this, SLOT(sendFrame()));
184 }
185
186 /* 从系统中读取可用的插件，并显示到 comboBox[0] */
187 void MainWindow::pluginItemInit()
188 {
189     comboBox[0]->addItems(QCanBus::instance()->plugins());
190     for (int i = 0; i < QCanBus::instance()->plugins().count(); i++)
{
191         if (QCanBus::instance()->plugins().at(i) == "socketcan")
192             comboBox[0]->setCurrentIndex(i);
193     }
194     connect(comboBox[0], SIGNAL(currentIndexChanged(int)),
195             this, SLOT(pluginChanged(int)));
196 }
197
198 /* 插件类型改变 */
199 void MainWindow::pluginChanged(int)
200 {
201     QList<QCanBusDeviceInfo> interfaces;
202     comboBox[1]->clear();
203     /* 当我们改变插件时，我们同时需要将可用接口，从插件类型中读取出来 */
204     interfaces = QCanBus::instance()
205                     ->availableDevices(comboBox[0]->currentText());
206     for (const QCanBusDeviceInfo &info : qAsConst(interfaces)) {
207         comboBox[1]->addItem(info.name());
208     }
209 }
210
211 /* 初始化一些常用的比特率，can 的比特率不是随便设置的，有相应的计算公式 */
212 void MainWindow::bitrateItemInit()
213 {
214     const QList<int> rates = {
215         10000, 20000, 50000, 100000, 125000,
216         250000, 500000, 800000, 1000000
217     };
218
219     for (int rate : rates)
220         comboBox[2]->addItem(QString::number(rate), rate);
```

```
221
222     /* 默认初始化以 1000000 比特率 */
223     comboBox[2]->setCurrentIndex(8);
224 }
225
226 /* 连接或断开 CAN */
227 void MainWindow::connectDevice()
228 {
229     if (pushButton[1]->text() == "连接 CAN") {
230         /* Qt 中的 QCanBusDevice::BitRateKey 不能设置比特率 */
231         QString cmd1 = tr("ifconfig %1 down")
232                         .arg(comboBox[1]->currentText());
233         QString cmd2 =
234             tr("ip link set up %1 type can bitrate %2 restart-ms
100")
235                 .arg(comboBox[1]->currentText())
236                 .arg(comboBox[2]->currentText());
237         /* 使用系统指令以设置的比特率初始化 CAN */
238         system(cmd1.toStdString().c_str());
239         system(cmd2.toStdString().c_str());
240
241         QString errorString;
242         /* 以设置的插件名与接口实例化 canDevice */
243         canDevice = QCanBus::instance()->
244                     createDevice(comboBox[0]->currentText(),
245                                  comboBox[1]->currentText(),
246                                  &errorString);
247
248         if (!canDevice) {
249             label[3]->setText(
250                 tr("Error creating device '%1', reason: '%2'")
251                     .arg(comboBox[0]->currentText())
252                     .arg(errorString));
253             return;
254         }
255
256         /* 连接 CAN */
257         if (!canDevice->connectDevice()) {
258             label[3]->setText(tr("Connection error: %1")
259                               .arg(canDevice->errorString()));
260             delete canDevice;
261             canDevice = nullptr;
262
```

```
263             return;
264         }
265
266         connect(canDevice, SIGNAL(framesReceived()),
267                 this, SLOT(receivedFrames()));
268         connect(canDevice,
269                 SIGNAL(errorOccurred(QCanBusDevice::CanBusError)),
270                 this,
271                 SLOT(canDeviceErrors(QCanBusDevice::CanBusError)));
272         /* 将连接信息插入到 label */
273         label[3]->setText(
274             tr("插件类型为: %1, 已连接到 %2, 比特率为 %3 kBit/s")
275                 .arg(comboBox[0]->currentText())
276                 .arg(comboBox[1]->currentText())
277                 .arg(comboBox[2]->currentText().toInt() / 1000));
278         pushButton[1]->setText("断开 CAN");
279         /* 使能/失能 */
280         pushButton[0]->setEnabled(true);
281         comboBox[0]->setEnabled(false);
282         comboBox[1]->setEnabled(false);
283         comboBox[2]->setEnabled(false);
284     } else {
285         if (!canDevice)
286             return;
287
288         /* 断开连接 */
289         canDevice->disconnectDevice();
290         delete canDevice;
291         canDevice = nullptr;
292         pushButton[1]->setText("连接 CAN");
293         pushButton[0]->setEnabled(false);
294         label[3]->setText(tr("未连接! "));
295         comboBox[0]->setEnabled(true);
296         comboBox[1]->setEnabled(true);
297         comboBox[2]->setEnabled(true);
298     }
299 }
300
301 void MainWindow::canDeviceErrors(QCanBusDevice::CanBusError error)
const
302 {
303     /* 错误处理 */
304     switch (error) {
```

```
305     case QCanBusDevice::ReadError:
306     case QCanBusDevice::WriteError:
307     case QCanBusDevice::ConnectionError:
308     case QCanBusDevice::ConfigurationError:
309     case QCanBusDevice::UnknownError:
310         label[3]->setText(canDevice->errorString());
311         break;
312     default:
313         break;
314     }
315 }
```

第 9~10 行，使用系统的 CAN 硬件，必须初始化系统的 CAN。在项目里添加相应的开启 CAN 的指令。第一个指令是先关闭本地的 CAN，因为只有关闭 CAN，才能以新的速再开启。

第 12~22 行，构造函数里界面初始化，以及 QComboBox 里的项初始化。

第 29~42 行，格式化帧处理函数。

第 45~67 行，发送消息，将 lineEdit 的文本进行处理后，第一个作为 CAN 的帧 ID，后面 8 个数据作为需要要发送的数据。每帧只能发送 8 个数据。

第 70~94 行，接收消息，读取帧并格式化处理，显示到 textBrowser 里。

第 96~184 行，界面布局初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。

第 187~196 行，可用插件初始化，检查系统 QCanBus 提供的插件。在 Linux 里使用的插件类型是 SocketCAN, SocketCAN 插件支持 Linux 内核和用于所用 CAN 硬件的 SocketCAN 设备驱动程序。下面程序遍历可用的 CAN 插件，并设置 socketcan 为当前插件。注意，只能使用 SocketCAN 访问本地硬件 CAN，其他插件是不同类型的 CAN 驱动程序所使用的请自行测试。

第 199~209 行，当插件类型改变时，我们需要更新可用接口。

第 212~224 行，常用的比特率初始化。

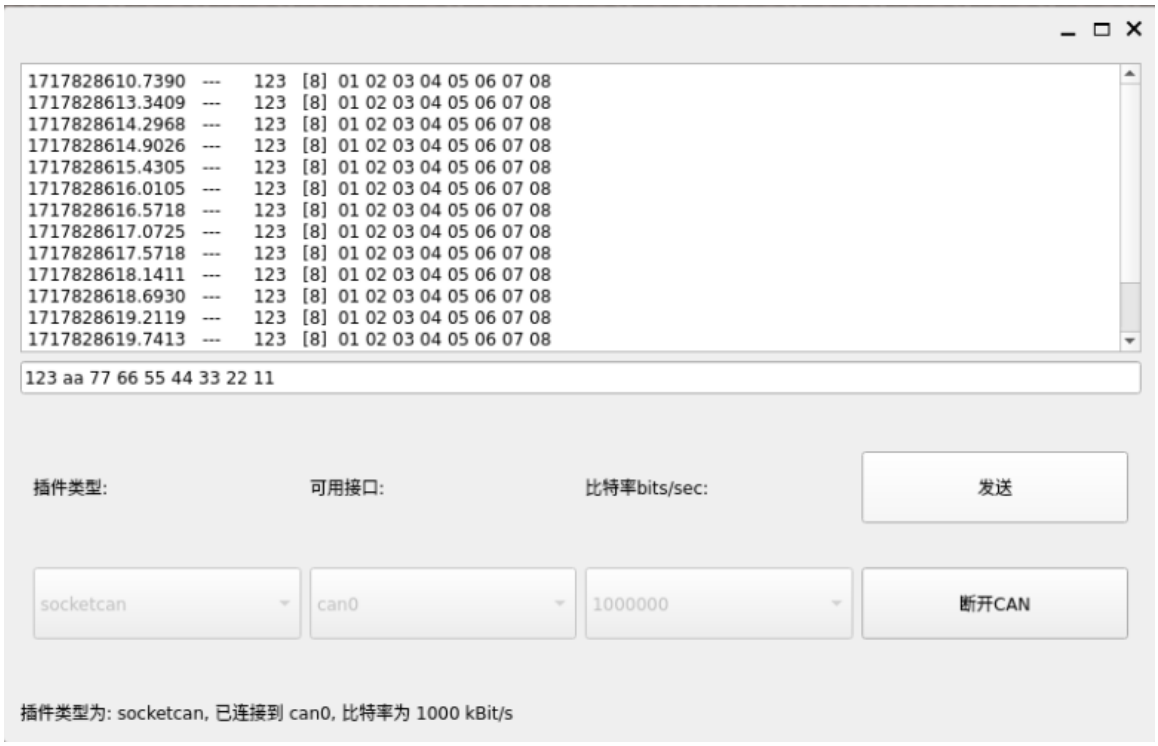
第 227~299 行，连接/断开 CAN，很遗憾 Qt 的 QCanBusDevice::BitRateKey 不能设置比特率，因为系统的 CAN 需要使用 ip 指令以一个比特率才能进行初始化，Qt 需要系统 CAN 起来才能进行操作。所以需要使用系统指令设置 CAN。

第 301~315 行，错误处理，CAN 设备可能遇到错误，打印错误的信息。

4.3 程序运行效果

下面将程序交叉编译到开发板运行，用上面所介绍的 USB CANFD 设备接到开发板的 CAN 上面，将 USB CANFD 分析仪一个通道的 CANH 接开发板的 CAN 接口处的 H，CANL 接开发板的 CAN 接口处的 L，启动 CAN 上位机。设置相同的比特率通信，这里插件类型默认是（必须是）socketcan，可用接口为 can0，即可发送消息与接收消息。

下图最上面的是接收消息框，我们这里接收到的数据是“123 [8] 01 02 03 04 05 06 07 08”另外下面“123 aa 77 66 55 44 33 22 11”这个是需要发送的帧，“123”为帧 ID，后面的为 8 个字节数据，每个字节需要以空格隔开。点击连接后，发送按钮才能使用。


1717828610.7390 --- 123 [8] 01 02 03 04 05 06 07 08
1717828613.3409 --- 123 [8] 01 02 03 04 05 06 07 08
1717828614.2968 --- 123 [8] 01 02 03 04 05 06 07 08
1717828614.9026 --- 123 [8] 01 02 03 04 05 06 07 08
1717828615.4305 --- 123 [8] 01 02 03 04 05 06 07 08
1717828616.0105 --- 123 [8] 01 02 03 04 05 06 07 08
1717828616.5718 --- 123 [8] 01 02 03 04 05 06 07 08
1717828617.0725 --- 123 [8] 01 02 03 04 05 06 07 08
1717828617.5718 --- 123 [8] 01 02 03 04 05 06 07 08
1717828618.1411 --- 123 [8] 01 02 03 04 05 06 07 08
1717828618.6930 --- 123 [8] 01 02 03 04 05 06 07 08
1717828619.2119 --- 123 [8] 01 02 03 04 05 06 07 08
1717828619.7413 --- 123 [8] 01 02 03 04 05 06 07 08
123 aa 77 66 55 44 33 22 11
插件类型:
可用接口:
比特率bits/sec:
发送
socketcan
can0
1000000
断开CAN
插件类型为: socketcan, 已连接到 can0, 比特率为 1000 kBit/s

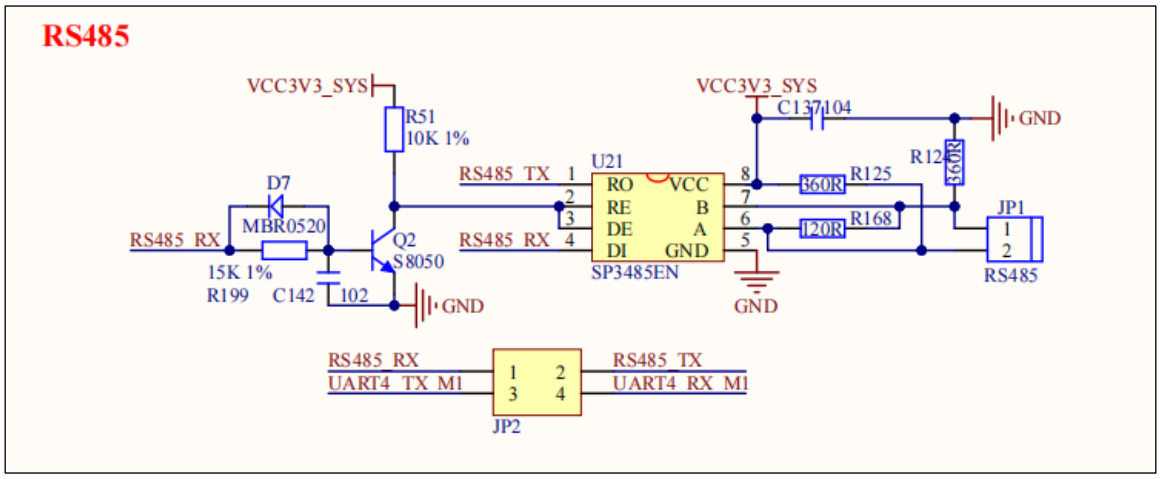
第五章 Qt 如何使用 RS485

Qt 是一个跨平台的 C++图形用户界面应用程序框架，提供了丰富的库和工具用于开发各种类型的应用程 序。其中，Qt 的串口通信库（QtSerialPort 模块）使得与 RS485 等串口设备的通信变得简单而高效。本教程将详细介绍如何在 Qt 中使用 RS485 接口进行串口通信。

5.1 资源简介

在正点原子的 ATK-DLRK3568 开发板的出厂系统里，默认已经配置了两路串口可用。一路是调试 串口 UART2(对应系统里的节点/dev/tty2)，另一路是 UART3(对应系统里的节/dev/tty3)。

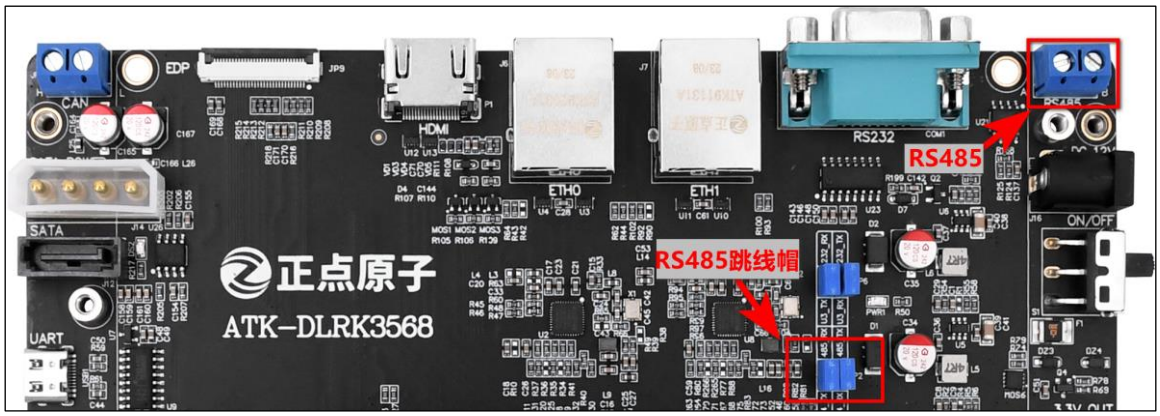
由于 UART2 已经作为调试串口被使用。所以我们只能对 UART3 编程，对应的原理图如下所示。



说明：此时我们需要一个 RS485 转串口模块，此模块正点原子店铺有售卖，当然用户手上有其他可以测试 485 的工具也可以自己测试。注意 485 是半双工的，不能同时收发。

正点原子店铺有售卖 USB 串口转换器三合一（支持 485 测试），接上杜邦线，A 对 A，B 对 B 连接到开发板的 RS485 接口，如下图。

注意，关于 USB 转换器请看正点原子“RK3568 开发板\开发板光盘 A 盘-基础资料\10、用户手册\01、测试文档\01【正点原子】ATK-DLRK3568_Buildroot 系统快速体验手册 V1.2.pdf”文档功能测试章节，本实验不讲解 USB CANFD 分析仪的使用。开发板对应的 CAN 接口位置如下。



5.2 应用实例

项目简介：Qt 串口的使用示例，应用到正点原子 ATK-DLRK3568 开发板上。例 05_rs485，Qt 串口编程（难度：一般）。项目路径为 01、程序源码\09、Qt 开发实例例程\05_rs485。

在 05_rs485.pro 里，我们需要使用串口，需要在 pro 项目文件中添加串口模块的支持。

```
1   QT       += core gui serialport
2   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
3
4   CONFIG += c++17
5
6   # You can make your code fail to compile if it uses deprecated APIs.
7   # In order to do so, uncomment the following line.
8   #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
9
10  SOURCES += \
11      main.cpp \
12      mainwindow.cpp
13
14  HEADERS += \
15      mainwindow.h
16
17  # Default rules for deployment.
18  qnx: target.path = /tmp/$${TARGET}/bin
19  else: unix:!android: target.path = /opt/$${TARGET}/bin
20  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加的 serialport 就是串口模块的支持。

在头文件“mainwindow.h”的代码如下。

```
/*****************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
```

```
* @brief        mainwindow.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
****************************************************************/
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QPushButton>
#include <QTextBrowser>
#include <QTextEdit>
#include <QVBoxLayout>
#include <QLabel>
#include <QComboBox>
#include <QGridLayout>
#include <QMessageBox>
#include <QDebug>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    /* 串口对象 */
    QSerialPort *serialPort;

    /* 用作接收数据 */
    QTextBrowser *textBrowser;

    /* 用作发送数据 */
    QTextEdit *textEdit;

    /* 按钮 */
    QPushButton *pushButton[2];

```

```
38      /* 下拉选择盒子 */
39      QComboBox *comboBox[5];
40
41      /* 标签 */
42      QLabel *label[5];
43
44      /* 垂直布局 */
45      QVBoxLayout *vboxLayout;
46
47      /* 网络布局 */
48      QGridLayout *gridLayout;
49
50      /* 主布局 */
51      QWidget *mainWidget;
52
53      /* 设置功能区域 */
54      QWidget *funcWidget;
55
56      /* 布局初始化 */
57      void layoutInit();
58
59      /* 扫描系统可用串口 */
60      void scanSerialPort();
61
62      /* 波特率项初始化 */
63      void baudRateItemInit();
64
65      /* 数据位项初始化 */
66      void dataBitsItemInit();
67
68      /* 检验位项初始化 */
69      void parityItemInit();
70
71      /* 停止位项初始化 */
72      void stopBitsItemInit();
73
74  private slots:
75      void sendPushButtonClicked();
76      void openSerialPortPushButtonClicked();
77      void serialPortReadyRead();
78  };
79  #endif // MAINWINDOW_H
```

上面代码是在 mianwindow.h 里声明需要用到的变量，方法及槽函数。

mainwindow.cpp 的代码如下

```
/**************************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************************/
#include "mainwindow.h"
#include <QDebug>
#include <QGuiApplication>
#include <QScreen>
#include <QRect>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 布局初始化 */
    layoutInit();

    /* 扫描系统的串口 */
    scanSerialPort();

    /* 波特率项初始化 */
    baudRateItemInit();

    /* 数据位项初始化 */
    dataBitsItemInit();

    /* 检验位项初始化 */
    parityItemInit();

    /* 停止位项初始化 */
    stopBitsItemInit();
}

void MainWindow::layoutInit()
{
    /* 获取屏幕的分辨率，Qt 官方建议使用这
     * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
     * 注意，这是获取整个桌面系统的分辨率
```

```
34      */
35     QList <QScreen *> list_screen =  QGuiApplication::screens();
36
37     /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
38 #if __arm__
39     /* 重设大小 */
40     this->resize(list_screen.at(0)->geometry().width(),
41                  list_screen.at(0)->geometry().height());
42 #else
43     /* 否则则设置主窗体大小为 800x480 */
44     this->resize(800, 480);
45 #endif
46     /* 初始化 */
47     serialPort = new QSerialPort(this);
48     textBrowser = new QTextBrowser();
49     textEdit = new QTextEdit();
50     vboxLayout = new QVBoxLayout();
51     funcWidget = new QWidget();
52     mainWidget = new QWidget();
53     gridLayout = new QGridLayout();
54
55     /* QList 链表，字符串类型 */
56     QList <QString> list1;
57     list1<<"串口号:"<<"波特率:"<<"数据位:"<<"检验位:"<<"停止位:";
58
59     for (int i = 0; i < 5; i++) {
60         label[i] = new QLabel(list1[i]);
61         /* 设置最小宽度与高度 */
62         label[i]->setMinimumSize(80, 30);
63         /* 自动调整 label 的大小 */
64         label[i]->setSizePolicy(
65             QSizePolicy::Expanding,
66             QSizePolicy::Expanding
67             );
68         /* 将 label[i]添加至网格的坐标(0, i) */
69         gridLayout->addWidget(label[i], 0, i);
70     }
71
72     for (int i = 0; i < 5; i++) {
73         comboBox[i] = new QComboBox();
74         comboBox[i]->setMinimumSize(80, 30);
75         /* 自动调整 label 的大小 */
76         comboBox[i]->setSizePolicy(
```

```
77              QSizePolicy::Expanding,
78              QSizePolicy::Expanding
79              );
80          /* 将 comboBox[i]添加至网格的坐标(1, i) */
81          gridLayout->addWidget(comboBox[i], 1, i);
82      }
83
84      /* QList 链表，字符串类型 */
85      QList <QString> list2;
86      list2<<"发送"<<"打开串口";
87
88      for (int i = 0; i < 2; i++) {
89          pushButton[i] = new QPushButton(list2[i]);
90          pushButton[i]->setMinimumSize(80, 30);
91          /* 自动调整 label 的大小 */
92          pushButton[i]->setSizePolicy(
93              QSizePolicy::Expanding,
94              QSizePolicy::Expanding
95              );
96          /* 将 pushButton[0]添加至网格的坐标(i, 5) */
97          gridLayout->addWidget(pushButton[i], i, 5);
98      }
99      pushButton[0]->setEnabled(false);
100
101     /* 布局 */
102     vboxLayout->addWidget(textBrowser);
103     vboxLayout->addWidget(textEdit);
104     funcWidget->setLayout(gridLayout);
105     vboxLayout->addWidget(funcWidget);
106     mainWidget->setLayout(vboxLayout);
107     this->setCentralWidget(mainWidget);
108
109     /* 占位文本 */
110     textBrowser->setPlaceholderText("接收到的消息");
111     textEdit->setText("www.openedv.com");
112
113     /* 信号槽连接 */
114     connect(pushButton[0], SIGNAL(clicked()),
115             this, SLOT(sendPushButtonClicked()));
116     connect(pushButton[1], SIGNAL(clicked()),
117             this, SLOT(openSerialPortPushButtonClicked()));
118
119     connect(serialPort, SIGNAL(readyRead()),
```

```
120             this, SLOT(serialPortReadyRead()));
121 }
122
123 void MainWindow::scanSerialPort()
124 {
125     /* 查找可用串口 */
126     foreach (const QSerialPortInfo &info,
127              QSerialPortInfo::availablePorts()) {
128         comboBox[0]->addItem(info.portName());
129     }
130 }
131
132 void MainWindow::baudRateItemInit()
133 {
134     /* QList 链表，字符串类型 */
135     QList <QString> list;
136     list<<"1200"<<"2400"<<"4800"<<"9600"
137         <<"19200"<<"38400"<<"57600"
138         <<"115200"<<"230400"<<"460800"
139         <<"921600";
140     for (int i = 0; i < 11; i++) {
141         comboBox[1]->addItem(list[i]);
142     }
143     comboBox[1]->setCurrentIndex(7);
144 }
145
146 void MainWindow::dataBitsItemInit()
147 {
148     /* QList 链表，字符串类型 */
149     QList <QString> list;
150     list<<"5"<<"6"<<"7"<<"8";
151     for (int i = 0; i < 4; i++) {
152         comboBox[2]->addItem(list[i]);
153     }
154     comboBox[2]->setCurrentIndex(3);
155 }
156
157 void MainWindow::parityItemInit()
158 {
159     /* QList 链表，字符串类型 */
160     QList <QString> list;
161     list<<"None"<<"Even"<<"Odd"<<"Space"<<"Mark";
162     for (int i = 0; i < 5; i++) {
```

```
163         comboBox[3]->addItem(list[i]);
164     }
165     comboBox[3]->setCurrentIndex(0);
166 }
167
168 void MainWindow::stopBitsItemInit()
169 {
170     /* QList 链表，字符串类型 */
171     QList <QString> list;
172     list<<"1"<<"2";
173     for (int i = 0; i < 2; i++) {
174         comboBox[4]->addItem(list[i]);
175     }
176     comboBox[4]->setCurrentIndex(0);
177 }
178
179 void MainWindow::sendPushButtonClicked()
180 {
181     /* 获取 textEdit 数据,转换成 utf8 格式的字节流 */
182     QByteArray data = textEdit->toPlainText().toUtf8();
183     serialPort->write(data);
184 }
185
186 void MainWindow::openSerialPortPushButtonClicked()
187 {
188     if (pushButton[1]->text() == "打开串口") {
189         /* 设置串口名 */
190         serialPort->setPortName(comboBox[0]->currentText());
191         /* 设置波特率 */
192         serialPort->setBaudRate(comboBox[1]->currentText().toInt());
193         /* 设置数据位数 */
194         switch (comboBox[2]->currentText().toInt()) {
195         case 5:
196             serialPort->setDataBits(QSerialPort::Data5);
197             break;
198         case 6:
199             serialPort->setDataBits(QSerialPort::Data6);
200             break;
201         case 7:
202             serialPort->setDataBits(QSerialPort::Data7);
203             break;
204         case 8:
205             serialPort->setDataBits(QSerialPort::Data8);
```

```
206             break;
207         default: break;
208         }
209         /* 设置奇偶校验 */
210         switch (comboBox[3]->currentIndex()) {
211         case 0:
212             serialPort->setParity(QSerialPort::NoParity);
213             break;
214         case 1:
215             serialPort->setParity(QSerialPort::EvenParity);
216             break;
217         case 2:
218             serialPort->setParity(QSerialPort::OddParity);
219             break;
220         case 3:
221             serialPort->setParity(QSerialPort::SpaceParity);
222             break;
223         case 4:
224             serialPort->setParity(QSerialPort::MarkParity);
225             break;
226         default: break;
227         }
228         /* 设置停止位 */
229         switch (comboBox[4]->currentText().toInt()) {
230         case 1:
231             serialPort->setStopBits(QSerialPort::OneStop);
232             break;
233         case 2:
234             serialPort->setStopBits(QSerialPort::TwoStop);
235             break;
236         default: break;
237         }
238         /* 设置流控制 */
239         serialPort->setFlowControl(QSerialPort::NoFlowControl);
240         if (!serialPort->open(QIODevice::ReadWrite))
241             QMessageBox::about(NULL, "错误",
242                                "串口无法打开！可能串口已经被占用！");
243         else {
244             for (int i = 0; i < 5; i++)
245                 comboBox[i]->setEnabled(false);
246             pushButton[1]->setText("关闭串口");
247             pushButton[0]->setEnabled(true);
248         }
```

```
249     } else {
250         serialPort->close();
251         for (int i = 0; i < 5; i++)
252             comboBox[i]->setEnabled(true);
253         pushButton[1]->setText("打开串口");
254         pushButton[0]->setEnabled(false);
255     }
256 }
257
258 void MainWindow::serialPortReadyRead()
259 {
260     /* 接收缓冲区中读取数据 */
261     QByteArray buf = serialPort->readAll();
262     textBrowser->insertPlainText(QString(buf));
263 }
264
265 MainWindow::~MainWindow()
266 {
267 }
```

第 29~121 行，界面布局初始化设置，在嵌入式里，根据实际的屏的大小，设置全屏显示。

第 123~130 行，查找系统可用的串口，并添加串口名到 comboBox[0]中。

第 132~144 行，波特率初始化，预设常用的波特率，115200 作为默认选项。并添加波特率到 comboBox[1]中。

第 146~155 行，数据位项初始化，设置默认数据位为 8。

第 157~166 行，校验位项初始化，默认无校验位。

第 168~177 行，停止位项初始化，默认停止位为 1。

第 179~184 行，发送数据，点击发送按钮时触发。

第 186~256 行，打开或者关闭串口。以我们设置的项使用 Qt 串口提供的设置串口的方法如 setDataBits(QSerialPort::DataBits)等，按第 188~239 行步骤设置完串口需要配置的参数就可以打开或者关闭串口了。

第 258~263 行，从缓冲区里读出数据，并显示到 textBrowser 里。

5.3 程序运行效果

下面将程序交叉编译到 ATK-DLRK3568 开发板运行，然后用上面所介绍的 USB 串口转换器三合一设备和开发板的 RS485 连接，连接方式：请看前面 5.1 资源简介，这里结合正点原子的 XCOM 上位机工具(或者本程序亦可当上位机软件)测试，其他工具可自行测试，接好后设置相同的串口参数，选择串口号为 tty3（注意 tty2 已经作为调试串口被使用了！），点击打开串口就可以进行消息收发了。默认参数为波特率为 115200，数据位为 8，校验为 None，停止位为 1。


ABCD123ERTYUABCD123ERTYUABCD123ERTYUABCD123ERTYUABCD123ERTYUABCD123ERTYUACFMABCD123ERTYUACFMABCD
123ERTYUACFMABCD123ERTYUACFMABCD123ERTYUACFMABCD123ERTYUACFMABCD123ERTYUACFMABCD123ERTYUACFMABC
D123ERTYUACFM
www.openedv.com
串口号:
波特率:
数据位:
检验位:
停止位:
发送
ttyS3
115200
8
None
1
关闭串口

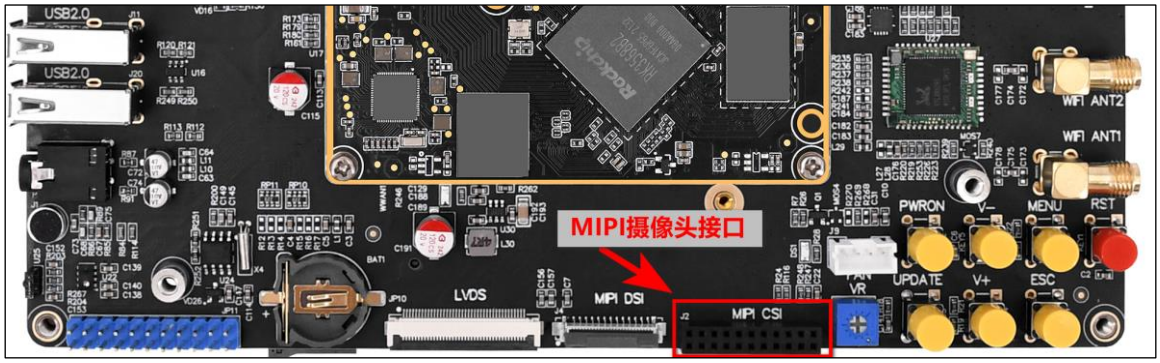
第六章 Qt 如何使用 Camera

此章节例程适用于正点原子 ATK-DLRK3568 开发板，不适用于 **Windows**（需要自行修改才能适用 **Windows**，**Windows** 上的应用不在我们讨论范围）！

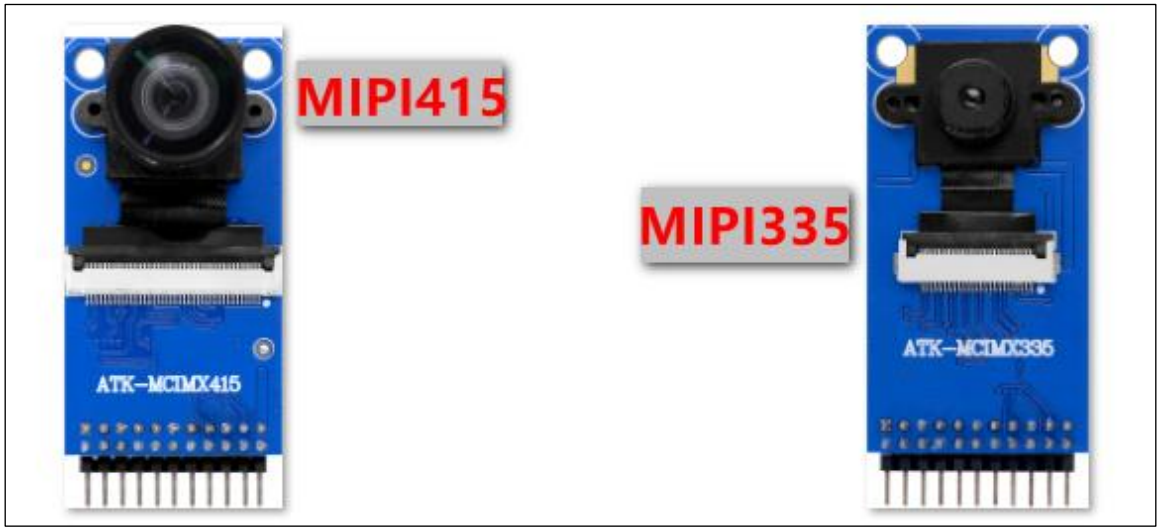
注意：摄像头拍照，会把对应的图片保存在本地。 以正点原子 ATK-DLRK3568 开发板为例，拍照所保存的图片路径为根目录下 /root/ 目录里面

6.1 资源简介

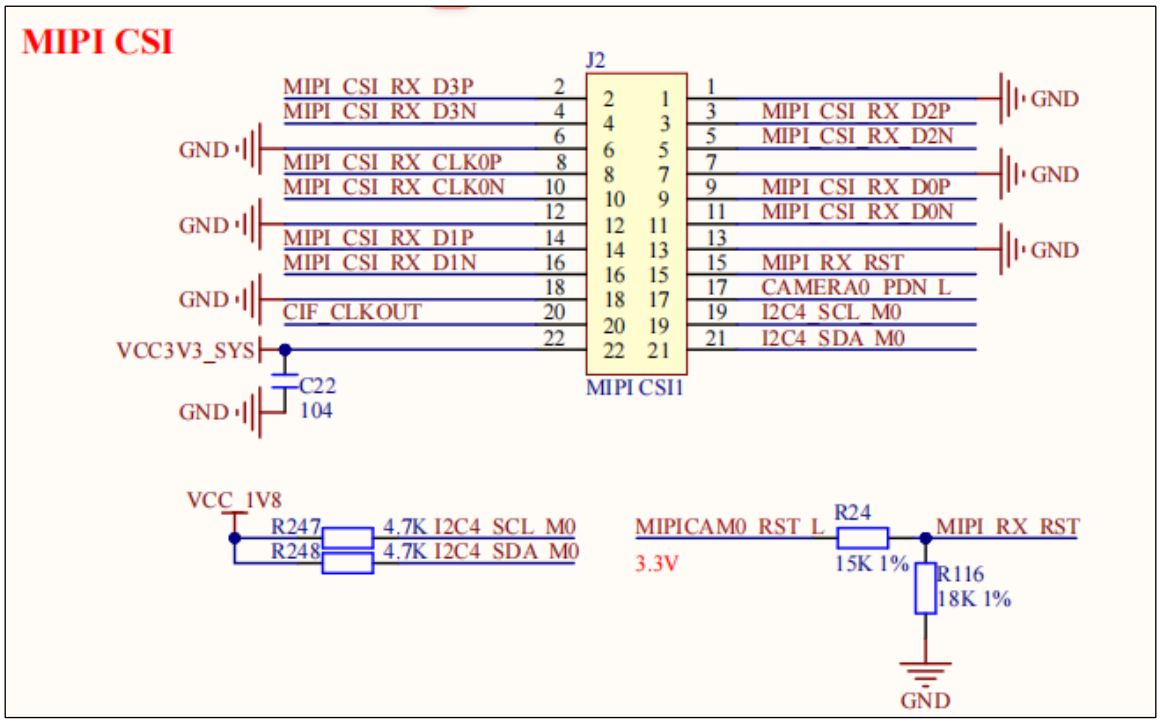
在正点原子 ATK-DLRK3568 Linux 出厂系统默认配置了 **MIPI CSI** 摄像头接口。请你将摄像头模块插在 ATK-DLRK3568 开发板的 **MIPI CSI** 接口上。在 ATK-DLRK3568 开发板上只有一路 mipi csi 接口，如果需要多路 **MIPI CSI** 摄像头接口请选购我们的 ATK-DLRK3588 开发板，如图是 ATK-DLRK3568 开发板上面 **MIPI CSI** 接口位置。



说明： 本章节用的是正点原子的 MIPI415 摄像头和 MIPI335 摄像头都可以 (两款摄像头正点原子店铺有售卖)，对应系统里的节点 /dev/video-camera0。安装摄像头的方法请看 正点原子 “RK3568 开发板\开发板光盘 A 盘-基础资料\10、用户手册\01、测试文档\01【正点原子】ATK-DLRK3568_Buildroot 系统快速体验手册 V1.2.pdf” 文档里的摄像头测试小节，故此这里不在叙述，如下是 MIPI415 摄像头和 MIPI335 摄像头实物图。



如下是 ATK-DLRK3568 开发板上面 mipi csi 接口的原理图。



6.2 应用实例

项目简介：Qt 提供了一个 QCamera 类。我们可以直接使用这个类来开发摄像头应用。例 06_qcamera，Qt 摄像头编程（难度：一般）。项目路径为 01、程序源码\09、Qt 开发实例例程\06_qcamera。

在 06_qcamera 里，要使用 QCamera 类，要加在项目文件中加上 QT += core gui multimedia multimediawidgets。其中 multimediawidgets 为 QCamera 视频输出显示的模块，如下

```
QT       += core gui multimedia multimediawidgets

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    widget.cpp

HEADERS += \
```

```
16      widget.h
17
18  # Default rules for deployment.
19  qnx: target.path = /tmp/$${TARGET}/bin
20  else: unix:!android: target.path = /opt/$${TARGET}/bin
21  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加的 multimedia multimediawidgets 就是摄像头模块的支持。

在头文件“widget.h”的代码如下。

```
/**********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        widget.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**********************************************************************/
1   #ifndef WIDGET_H
2   #define WIDGET_H
3
4   #include <QWidget>
5   #include <QCamera>
6   #include <QCameraImageCapture>
7   #include <QVideoWidget>
8
9   class Widget : public QWidget
10  {
11      Q_OBJECT
12
13  public:
14      explicit Widget(QWidget *parent = nullptr);
15      ~Widget();
16
17  private slots:
18      void onCaptureImage();
19      void onExit();
20
21  private:
22      QCamera *m_qcamera;
23      QCameraImageCapture *m_imageCapture;
24      QVideoWidget *m_videoWidget;
25  };
26
```

```
27  #endif // WIDGET_H
```

上面代码是在 widget.h 里声明需要用到的变量，方法及槽函数。

widget.cpp 的代码如下

```
/**************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        widget.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************/
1   #include "widget.h"
2   #include <QCamera>
3   #include <QCameraImageCapture>
4   #include <QCameraViewfinderSettings>
5   #include <QVideoWidget>
6   #include <QPushButton>
7   #include <QVBoxLayout>
8   #include <QDebug>
9
10  Widget::Widget(QWidget *parent)
11      : QWidget(parent)
12      , m_qcamera(nullptr)
13      , m_imageCapture(nullptr)
14  {
15      this->resize(640, 480);
16
17      // 初始化摄像头
18      m_qcamera = new QCamera("/dev/video-camera0", this);
19      if (!m_qcamera) {
20          qDebug() << "摄像头初始化失败！";
21          return;
22      }
23
24      // 设置摄像头分辨率
25      QCameraViewfinderSettings settings;
26      settings.setResolution(640, 480);
27      m_qcamera->setViewfinderSettings(settings);
28
29      // 创建视频窗口
30      m_videoWidget = new QVideoWidget(this);
31
```

```
32      // 创建拍照和退出按钮
33      QPushButton *captureButton = new QPushButton("拍照", this);
34      QPushButton *exitButton = new QPushButton("退出", this);
35      //设置按钮的宽度和高度
36      captureButton->setStyleSheet("QPushButton { min-width: 80px;
min-height: 50px; }");
37      exitButton->setStyleSheet("QPushButton { min-width: 80px; min-
height: 50px; }");
38
39      // 布局
40      QVBoxLayout *layout = new QVBoxLayout(this);
41      layout->addWidget(m_videoWidget, 1); // 视频窗口占据大部分空间
42
43      // 创建一个水平布局来放置按钮
44      QHBoxLayout *buttonLayout = new QHBoxLayout();
45      buttonLayout->addWidget(captureButton);
46      buttonLayout->addWidget(exitButton);
47
48      // 将按钮水平布局添加到主垂直布局中
49      layout->addLayout(buttonLayout);
50
51      // 设置视频输出
52      m_qcamera->setViewfinder(m_videoWidget);
53
54      // 初始化拍照类
55      m_imageCapture = new QCameraImageCapture(m_qcamera);
56
57      // 连接按钮信号到槽函数
58      connect(captureButton, &QPushButton::clicked, this,
&Widget::onCaptureImage);
59      connect(exitButton, &QPushButton::clicked, this,
&Widget::onExit);
60
61      // 开始摄像头
62      m_qcamera->start();
63  }
64
65  void Widget::onCaptureImage() {
66      // 拍照并保存到文件
67      QString fileName = "capture.jpg"; // 图片文件名
68      m_imageCapture->capture(fileName);
69      qDebug() << "图片已保存到: " << fileName;
70  }
```

```
71
72  void Widget::onExit() {
73      // 停止摄像头并关闭窗口
74      m_qcamera->stop();
75      this->close();
76  }
77
78  Widget::~Widget() {
79      // 确保在析构时释放资源
80      if (m_qcamera) {
81          m_qcamera->stop();
82          delete m_qcamera;
83      }
84      if (m_imageCapture) {
85          delete m_imageCapture;
86      }
87  }
```

第 15~27 行，初始化摄像头，设置摄像头分辨率。

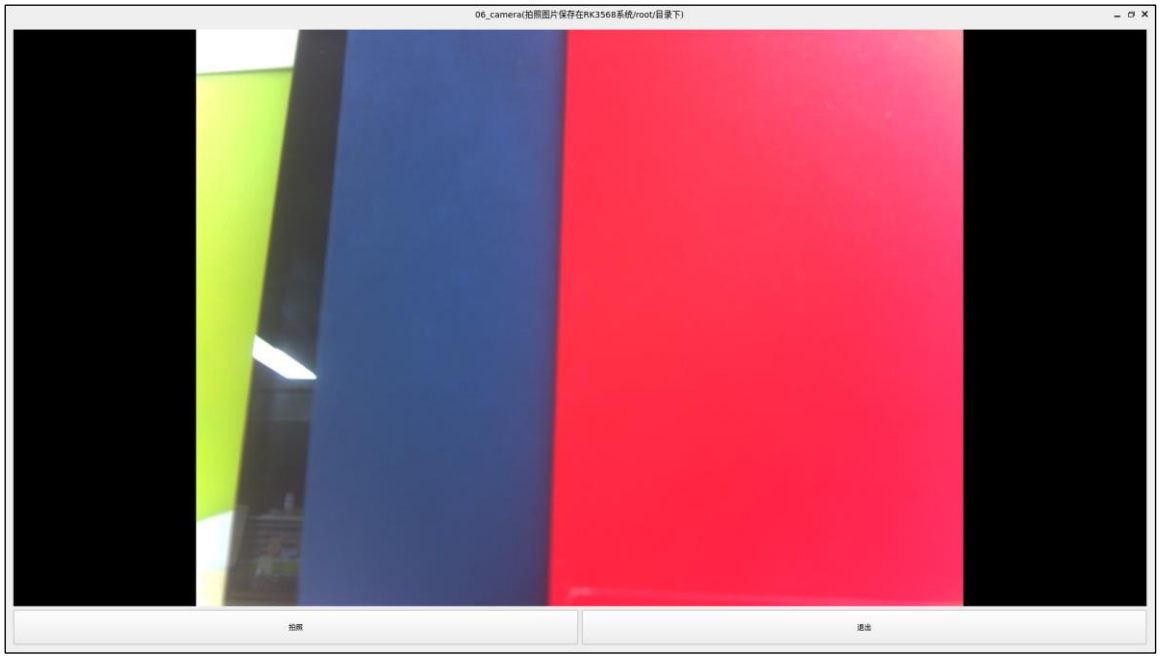
第 30~49 行，创建窗口和按钮并进行布局。

第 55~63 行，初始化一个拍照类，关联信号槽以及开启摄像头。

第 65~70 行，摄像头拍照，把对应的图片保存在本地。 以正点原子 ATK-DLRK3568 开发板为例，拍照所保存的图片地址路径为根目录下 /root/ 目录里面

第 78~87 行，释放资源的目的是为了防止资源泄露

6.3 程序运行效果

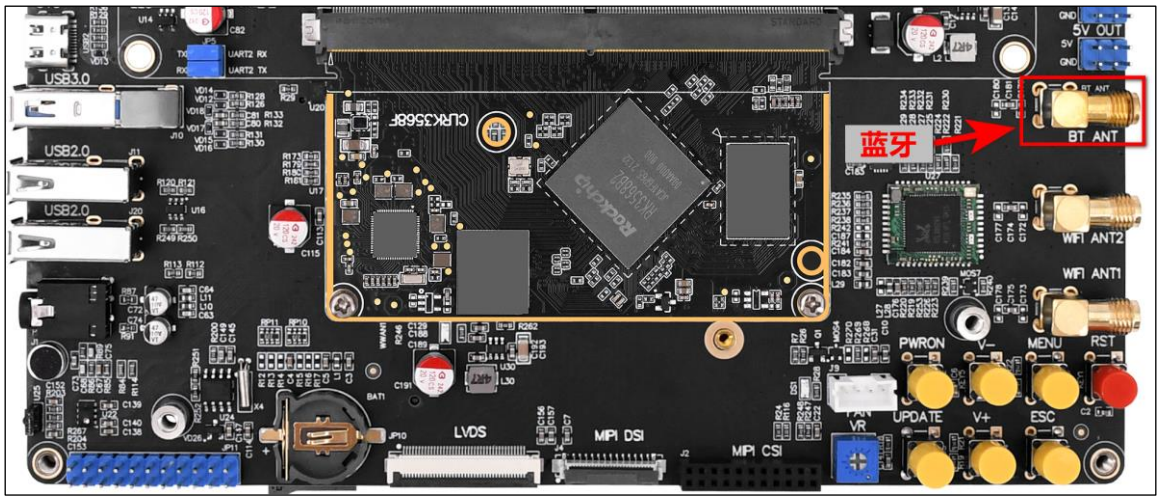


第七章 Qt 如何使用 Bluetooth

Qt 官方提供了蓝牙的相关类和 API 函数，也提供了相关的例程给我们参考。笔者根据 Qt 官方的例程编写出适合我们 ATK-DLRK3568 开发板的例程。注意 Windows 上不能使用 Qt 的蓝牙例程，因为底层需要有 BlueZ 协议栈，而 Windows 没有。

7.1 资源简介

在正点原子 ATK-DLRK3568 发板上有一个带板载蓝牙，本小节对其蓝牙模块就进行本章节的实验。如下图是 ATK-DLRK3568 开发板板载的蓝牙模块。



切记先看正点原子 ATK-DLRK3568 开发板的快速体验文档，路径：“RK3568 开发板\开发板光盘 A 盘-基础资料\10、用户手册\01、测试文档\01【正点原子】ATK-DLRK3568_Buildroot 系统快速体验手册 V1.2.pdf” 的第 **3.11** 小节蓝牙测试，先了解 ATK-DLRK3568 开发板上面的板载蓝牙是如何使用的。

7.2 应用实例

项目简介：Qt 蓝牙聊天。将蓝牙设置成一个服务器，或者用做客户端，连接手机即可通信。

例 07_bluetooth，Qt 蓝牙聊天（难度：**难**）。项目路径为 01、程序源码\09、Qt 开发实例例程\07_bluetooth。

Qt 使用蓝牙，需要在项目文件加上相应的蓝牙模块。添加的代码如下红色加粗部分。

07_bluetooth.pro 文件代码如下。

```
1   QT       += core gui bluetooth
2
3   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5   CONFIG += c++17
6
7   # You can make your code fail to compile if it uses deprecated APIs.
8   # In order to do so, uncomment the following line.
9   #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all
the APIs deprecated before Qt 6.0.0
10
11  SOURCES += \
12      chatclient.cpp \
```

```
13      chatserver.cpp \
14      main.cpp \
15      mainwindow.cpp \
16      remoteselector.cpp
17
18  HEADERS += \
19      chatclient.h \
20      chatserver.h \
21      mainwindow.h \
22      remoteselector.h
23
24  # Default rules for deployment.
25  qnx: target.path = /tmp/$${TARGET}/bin
26  else: unix:!android: target.path = /opt/$${TARGET}/bin
27  !isEmpty(target.path): INSTALLS += target
```

第 1 行，添加的 bluetooth 就是板载蓝牙模组的支持。

第 18~29 行，可以看到我们的项目组成文件。一个客户端，一个服务端，一个主界面和一 个远程选择蓝牙的文件。总的看起来有四大部分，下面就介绍这四大部分的文件。

chatclient.h 的代码如下。

```
/**************************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        chatclient.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************************/
1   #ifndef CHATCLIENT_H
2   #define CHATCLIENT_H
3
4   #include <qbluetoothserviceinfo.h>
5   #include <QBluetoothSocket>
6   #include <QtCore/QObject>
7
8   QT_FORWARD_DECLARE_CLASS(QBluetoothSocket)
9
10  class ChatClient : public QObject
11  {
12      Q_OBJECT
13
14  public:
15      explicit ChatClient(QObject *parent = nullptr);
```

```
16      ~ChatClient();
17
18      /* 开启客户端 */
19      void startClient(const QBluetoothServiceInfo &remoteService);
20
21      /* 停止客户端 */
22      void stopClient();
23
24  public slots:
25      /* 发送消息 */
26      void sendMessage(const QString &message);
27
28      /* 主动断开连接 */
29      void disconnect();
30
31  signals:
32      /* 接收到消息信号 */
33      void messageReceived(const QString &sender, const QString
&message);
34
35      /* 连接信号 */
36      void connected(const QString &name);
37
38      /* 断开连接信号 */
39      void disconnected();
40
41  private slots:
42      /* 从 socket 里读取消息 */
43      void readSocket();
44
45      /* 连接 */
46      void connected();
47
48  private:
49      /* socket 通信 */
50      QBluetoothSocket *socket;
51  };
52
53  #endif // CHATCLIENT_H
```

chatclient.h 文件主要是客户端的头文件，其中写一些接口，比如开启客户端，关闭客户端，接收信号与关闭信号等等。

chatclient.cpp 的代码如下。

```
/****************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        chatclient.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
****************************************************************/
#include "chatclient.h"
#include <qbluetoothsocket.h>

ChatClient::ChatClient(QObject *parent)
    :   QObject(parent), socket(0)
{
}

ChatClient::~ChatClient()
{
    stopClient();
}

/* 开启客户端 */
void ChatClient::startClient(const QBluetoothServiceInfo
&remoteService)
{
    if (socket)
        return;

    // Connect to service
    socket = new
QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol);
    qDebug() << "Create socket";
    socket->connectToService(remoteService);
    qDebug() << "ConnectToService done";

    connect(socket, SIGNAL(readyRead()),
            this, SLOT(readSocket()));
    connect(socket, SIGNAL(connected()),
            this, SLOT(connected()));
    connect(socket, SIGNAL(disconnected()),
            this, SIGNAL(disconnected()));
}

```

```
34  /* 停止客户端 */
35  void ChatClient::stopClient()
36  {
37      delete socket;
38      socket = 0;
39  }
40
41  /* 从 Socket 读取消息 */
42  void ChatClient::readSocket()
43  {
44      if (!socket)
45          return;
46
47      while (socket->canReadLine()) {
48          QByteArray line = socket->readLine();
49          emit messageReceived(socket->peerName(),
50                               QString::fromUtf8(line.constData(),
51                                                 line.length()));
52      }
53  }
54
55  /* 发送的消息 */
56  void ChatClient::sendMessage(const QString &message)
57  {
58      qDebug()<<"Sending data in client: " + message;
59
60      QByteArray text = message.toUtf8() + '\n';
61      socket->write(text);
62  }
63
64  /* 主动连接 */
65  void ChatClient::connected()
66  {
67      emit connected(socket->peerName());
68  }
69
70  /* 主动断开连接*/
71  void ChatClient::disconnect() {
72      qDebug()<<"Going to disconnect in client";
73      if (socket) {
74          qDebug()<<"diconnecting...";
75          socket->close();
76      }
```

```
77  }
```

chatclient.cpp 文件主要是客户端的 chatclient.h 头文件的实现。代码参考 Qt 官方 btchat 例子， 代码比较长，也有相应的注释了，大家自由查看。主要我们关注的是下面的代码。第 15~32 行，我们需要开启客户端模式，那么我们需要将扫描服务器（手机蓝牙）的结果，

实例化一个蓝牙 socket，使用 socket 连接传入来的服务器信息，即可将本地蓝牙当作客户端，实现了客户端创建。

chatserver.h 代码如下

```
/**************************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        chatserver.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************************/
#ifndef CHATSERVER_H
#define CHATSERVER_H

#include <qbluetoothserviceinfo.h>
#include <qbluetoothaddress.h>
#include <QtCore/QObject>
#include <QtCore/QList>
#include <QBluetoothServer>
#include <QBluetoothSocket>


1   class ChatServer : public QObject
2   {
3       Q_OBJECT
4
5   public:
6       explicit ChatServer(QObject *parent = nullptr);
7       ~ChatServer();
8
9       /* 开启服务端 */
10      void startServer(const QBluetoothAddress &localAdapter =
QBluetoothAddress());
11
12      /* 停止服务端 */
13      void stopServer();
14
15  public slots:
```

```
16      /* 发送消息 */
17      void sendMessage(const QString &message);
18
19      /* 服务端主动断开连接 */
20      void disconnect();
21
22  signals:
23      /* 接收到消息信号 */
24      void messageReceived(const QString &sender, const QString
&message);
25
26      /* 客户端连接信号 */
27      void clientConnected(const QString &name);
28
29      /* 客户端断开连接信号 */
30      void clientDisconnected(const QString &name);
31
32  private slots:
33
34      /* 客户端连接 */
35      void clientConnected();
36
37      /* 客户端断开连接 */
38      void clientDisconnected();
39
40      /* 读 socket */
41      void readSocket();
42
43  private:
44      /* 使用 rfcomm 协议 */
45      QBluetoothServer *rfcommServer;
46
47      /* 服务器蓝牙信息 */
48      QBluetoothServiceInfo serviceInfo;
49
50      /* 用于保存客户端 socket */
51      QList<QBluetoothSocket *> clientSockets;
52
53      /* 用于保存客户端的名字 */
54      QList<QString> socketsPeername;
55  };
56
57  #endif // CHATSERVER_H
```

chatserver.h 文件主要是服务端的头文件，其中写一些接口，比如开启服务端，关闭服务端，接收信号与关闭信号等等。

chatserver.cpp 代码如下。

```
/**************************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        chatserver.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************************/
#include "chatserver.h"

#include <qbluetoothserver.h>
#include <qbluetoothsocket.h>
#include <qbluetoothlocaldevice.h>

static const QLatin1String serviceUuid("e8e10f95-1a70-4b27-9ccf-02010264e9c8");
ChatServer::ChatServer(QObject *parent)
    :   QObject(parent), rfcommServer(0)
{
}

ChatServer::~ChatServer()
{
    stopServer();
}

/* 开启服务端，设置服务端使用 rfcomm 协议与 serviceInfo 的一些属性 */
void ChatServer::startServer(const QBluetoothAddress& localAdapter)
{
    if (rfcommServer)
        return;

    rfcommServer = new QBluetoothServer(QBluetoothServiceInfo::RfcommProtocol, this);
    connect(rfcommServer, SIGNAL(newConnection()), this, SLOT(clientConnected()));
    bool result = rfcommServer->listen(localAdapter);
    if (!result) {
```

```
28          qWarning()<<"Cannot bind chat server
to"<<localAdapter.toString();
29          return;
30      }
31
32
serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceRecordHandle,
(uint)0x00010010);
33
34      QBluetoothServiceInfo::Sequence classId;
35
36
classId<<QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::SerialPort)
);
37
serviceInfo.setAttribute(QBluetoothServiceInfo::BluetoothProfileDescrip
torList,
38                           classId);
39
40
classId.prepend(QVariant::fromValue(QBluetoothUuid(serviceUuid)));
41
42      serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceClassIds,
classId);
43
serviceInfo.setAttribute(QBluetoothServiceInfo::BluetoothProfileDescrip
torList,classId);
44
45      serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceName,
tr("Bt Chat Server"));
46
serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceDescription,
47                           tr("Example bluetooth chat server"));
48      serviceInfo.setAttribute(QBluetoothServiceInfo::ServiceProvider,
tr("qt-project.org"));
49
50      serviceInfo.setServiceUuid(QBluetoothUuid(serviceUuid));
51
52      QBluetoothServiceInfo::Sequence publicBrowse;
53      publicBrowse<<
QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::PublicBrowseGroup));
54      serviceInfo.setAttribute(QBluetoothServiceInfo::BrowseGroupList,
55                           publicBrowse);
```

```
56
57      QBluetoothServiceInfo::Sequence protocolDescriptorList;
58      QBluetoothServiceInfo::Sequence protocol;
59      protocol<<
QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::L2cap));
60      protocolDescriptorList.append(QVariant::fromValue(protocol));
61      protocol.clear();
62      protocol<<
QVariant::fromValue(QBluetoothUuid(QBluetoothUuid::Rfcomm))
63              <<
QVariant::fromValue(quint8(rfcommServer->serverPort()));
64      protocolDescriptorList.append(QVariant::fromValue(protocol));
65
serviceInfo.setAttribute(QBluetoothServiceInfo::ProtocolDescriptorList,
66                           protocolDescriptorList);
67
68      serviceInfo.registerService(localAdapter);
69  }
70
71  /* 停止服务端 */
72  void ChatServer::stopServer()
73  {
74      // Unregister service
75      serviceInfo.unregisterService();
76
77      // Close sockets
78      qDeleteAll(clientSockets);
79
80      // Close server
81      delete rfcommServer;
82      rfcommServer = 0;
83  }
84
85  /* 主动断开连接 */
86  void ChatServer::disconnect()
87  {
88      qDebug()<<"Going to disconnect in server";
89
90      foreach (QBluetoothSocket *socket, clientSockets) {
91          qDebug()<<"sending data in server!";
92          socket->close();
93      }
94  }
```

```
95
96  /* 发送消息 */
97  void ChatServer::sendMessage(const QString &message)
98  {
99      qDebug()<<"Going to send message in server: " << message;
100     QByteArray text = message.toUtf8() + '\n';
101
102     foreach (QBluetoothSocket *socket, clientSockets) {
103         qDebug()<<"sending data in server!";
104         socket->write(text);
105     }
106     qDebug()<<"server sending done!";
107 }
108
109 /* 客户端连接 */
110 void ChatServer::clientConnected()
111 {
112     qDebug()<<"clientConnected";
113
114     QBluetoothSocket *socket =
rfcommServer->nextPendingConnection();
115     if (!socket)
116         return;
117
118     connect(socket, SIGNAL(readyRead()), this, SLOT(readSocket()));
119     connect(socket, SIGNAL(disconnected()), this,
SLOT(clientDisconnected()));
120     clientSockets.append(socket);
121     socketsPeername.append(socket->peerName());
122     emit clientConnected(socket->peerName());
123 }
124
125 /* 客户端断开连接 */
126 void ChatServer::clientDisconnected()
127 {
128     QBluetoothSocket *socket = qobject_cast<QBluetoothSocket
*>(sender());
129     if (!socket)
130         return;
131
132     if (clientSockets.count() != 0) {
133         QString peerName;
134
```

```
135         if (socket->peerName().isEmpty())
136             peerName =
socketsPeername.at(clientSockets.indexOf(socket));
137         else
138             peerName = socket->peerName();
139
140         emit clientDisconnected(peerName);
141
142         clientSockets.removeOne(socket);
143         socketsPeername.removeOne(peerName);
144     }
145
146     socket->deleteLater();
147
148 }
149
150 /* 从 Socket 里读取数据 */
151 void ChatServer::readSocket()
152 {
153     QBluetoothSocket *socket = qobject_cast<QBluetoothSocket
*>(sender());
154     if (!socket)
155         return;
156
157     while (socket->bytesAvailable()) {
158         QByteArray line = socket->readLine().trimmed();
159         qDebug()<<QString::fromUtf8(line.constData(),
line.length())<<endl;
160         emit messageReceived(socket->peerName(),
161                              QString::fromUtf8(line.constData(),
line.length()));
162         qDebug()<<QString::fromUtf8(line.constData(),
line.length())<<endl;
163     }
164 }
```

chatserver.cpp 文件主要是服务端的 chatserver.h 头文件的实现。代码也是参考 Qt 官方 btchat 例子，代码比较长，也有相应的注释了，大家自由查看。主要我们关注的是下面的代码。

第 19~69 行，我们需要开启服务端模式，那么我们需要将本地的蓝牙 localAdapter 的地址传入，创建一个 QBluetoothServer 对象 rfcommServer。在 19 至 69 行代码很长，其中使用了 serviceInfo.setAttribute() 设置了许多参数，这个流程是官方给出的流程，我们只需要了解下就可以了。大体流程：使用了 QBluetoothServiceInfo 类允许访问服务端蓝牙服务的属性，

其中有设置蓝牙的 UUID 为文件开头定义的 serviceUuid，设置 serviceUuid 的目的是为了区分其他蓝牙，用于搜索此类型蓝牙，但是作用并不是很大，因为我们的手机并不一定开启了这个 uuid 标识。

最后必须用 registerService()启动蓝牙。

第 36 行，转换串行端口（SerialPort），转换成 classId，然后再设置串行端口服务。通信原理就是串行端口连接到 RFCOMM server channel。（PS：蓝牙使用的协议多且复杂，本教程并不能清晰解释这种原理，如果有错误，欢迎指出）。

remoteselector.h 代码如下

```
/**************************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        remoteselector.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************************/
1   #ifndef REMOTESELECTOR_H
2   #define REMOTESELECTOR_H
3
4   #include <qbluetoothuuid.h>
5   #include <qbluetoothserviceinfo.h>
6   #include <qbluetoothservicediscoveryagent.h>
7   #include <QListWidgetItem>
8
9   /* 声明一个蓝牙适配器类 */
10  class RemoteSelector : public QObject
11  {
12      Q_OBJECT
13
14  public:
15      explicit RemoteSelector(QBluetoothAddress&,
16                              QObject *parent = nullptr);
17      ~RemoteSelector();
18
19      /* 开启发现蓝牙 */
20      void startDiscovery(const QBluetoothUuid &uuid);
21
22      /* 停止发现蓝牙 */
23      void stopDiscovery();
24
25      /* 蓝牙服务 */
26      QBluetoothServiceInfo service() const;
27
```

```
28  signals:
29      /* 找到新服务 */
30      void newServiceFound(QListWidgetItem*);
31
32      /* 完成 */
33      void finished();
34
35  private:
36      /* 蓝牙服务代理，用于发现蓝牙服务 */
37      QBluetoothServiceDiscoveryAgent *m_discoveryAgent;
38
39      /* 服务信息 */
40      QBluetoothServiceInfo m_serviceInfo;
41
42  private slots:
43      /* 服务发现完成 */
44      void serviceDiscovered(const QBluetoothServiceInfo
&serviceInfo);
45
46      /* 蓝牙发现完成 */
47      void discoveryFinished();
48
49  public:
50      /* 键值类容器 */
51      QMap<QString, QBluetoothServiceInfo> m_discoveredServices;
52  };
53
54  #endif // REMOTESELECTOR_H
```

remoteselector.h 翻译成远程选择器，代码也是参考 Qt 官方 btchat 例子，这个头文件定义了开启蓝牙发现模式，蓝牙关闭发现模式，还有服务完成等等，代码有注释，请自由查看。

remoteselector.cpp 代码如下。

```
/*********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        remoteselector.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
*********************************************************************/
1   #include "remoteselector.h"
2
3   /* 初始化本地蓝牙 */
```

```
4   RemoteSelector::RemoteSelector(QBluetoothAddress &localAdapter,
QObject *parent)
5       : QObject(parent)
6   {
7       m_discoveryAgent = new
QBluetoothServiceDiscoveryAgent(localAdapter);
8
9       connect(m_discoveryAgent,
SIGNAL(serviceDiscovered(QBluetoothServiceInfo)),
10              this, SLOT(serviceDiscovered(QBluetoothServiceInfo)));
11      connect(m_discoveryAgent, SIGNAL(finished()), this,
SLOT(discoveryFinished()));
12      connect(m_discoveryAgent, SIGNAL(canceled()), this,
SLOT(discoveryFinished()));
13  }
14
15  RemoteSelector::~RemoteSelector()
16  {
17      delete m_discoveryAgent;
18  }
19
20  /* 开启发现模式，这里无需设置过滤 uuid，否则搜索不到手机
21   * uuid 会过滤符合条件的 uuid 服务都会返回相应的蓝牙设备
22   */
23  void RemoteSelector::startDiscovery(const QBluetoothUuid &uuid)
24  {
25      Q_UNUSED(uuid);
26      qDebug()<<"startDiscovery";
27      if (m_discoveryAgent->isActive()) {
28          qDebug()<<"stop the searching first";
29          m_discoveryAgent->stop();
30      }
31
32      //m_discoveryAgent->setUuidFilter(uuid);
33
m_discoveryAgent->start(QBluetoothServiceDiscoveryAgent::FullDiscovery)
;
34  }
35
36  /* 停止发现 */
37  void RemoteSelector::stopDiscovery()
38  {
39      qDebug()<<"stopDiscovery";
```

```
40      if (m_discoveryAgent){
41          m_discoveryAgent->stop();
42      }
43  }
44
45  QBluetoothServiceInfo RemoteSelector::service() const
46  {
47      return m_serviceInfo;
48  }
49
50  /* 扫描蓝牙服务信息 */
51  void RemoteSelector::serviceDiscovered(const QBluetoothServiceInfo
&serviceInfo)
52  {
53  #if 0
54      qDebug() << "Discovered service on"
55               << serviceInfo.device().name() <<
serviceInfo.device().address().toString();
56      qDebug() << "\tService name:" << serviceInfo.serviceName();
57      qDebug() << "\tDescription:"
58               <<
serviceInfo.attribute(QBluetoothServiceInfo::ServiceDescription).toStri
ng();
59      qDebug() << "\tProvider:"
60               <<
serviceInfo.attribute(QBluetoothServiceInfo::ServiceProvider).toString(
);
61      qDebug() << "\tL2CAP protocol service multiplexer:"
62               << serviceInfo.protocolServiceMultiplexer();
63      qDebug() << "\tRFCOMM server channel:" <<
serviceInfo.serverChannel();
64  #endif
65
66      QMapIterator<QString, QBluetoothServiceInfo>
i(m_discoveredServices);
67      while (i.hasNext()){
68          i.next();
69          if (serviceInfo.device().address() ==
i.value().device().address()){
70              return;
71          }
72      }
73
```

```
74      QString remoteName;
75      if (serviceInfo.device().name().isEmpty())
76          remoteName = serviceInfo.device().address().toString();
77      else
78          remoteName = serviceInfo.device().name();
79
80      qDebug()<<"adding to the list....";
81      qDebug()<<"remoteName: "<< remoteName;
82      QListWidgetItem *item =
83          new QListWidgetItem(QString::fromLatin1("%1%2")
84                              .arg(remoteName,
serviceInfo.serviceName()));
85      m_discoveredServices.insert(remoteName, serviceInfo);
86      emit newServiceFound(item);
87  }
88
89  /* 发现完成 */
90  void RemoteSelector::discoveryFinished()
91  {
92      qDebug()<<"discoveryFinished";
93      emit finished();
94  }
```

remoteselector .cpp 是 remoteselector.h 的实现代码。主要看以下几点。

第 4~13 行，初始化本地蓝牙，实例化对象 discoveryAgent（代理对象），蓝牙主要通过本地代理对象去扫描其他蓝牙。

第 32 行，这里官方 Qt 代码设计是过滤 uuid。只有符合对应的 uuid 的蓝牙，才会返回结果。因为我们要扫描我们的手机，所以这里我们要把它注释掉。手机的 uuid 没有设置成设定的 uuid，如果设置了 uuid 过滤，手机就扫描不出了。（uuid 指的是唯一标识，手机蓝牙有很多 uuid，不同的 uuid 有不同的作用，指示着不同的服务）。

其他代码都是一些逻辑性的代码，比较简单，请自由查看。

mainwindow.h 代码如下。

```
/**************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.h
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**************************************************************/
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
```

```
4   #include <QMainWindow>
5   #include <qbluetoothserviceinfo.h>
6   #include <qbluetoothsocket.h>
7   #include <qbluetoothhostinfo.h>
8   #include <QDebug>
9   #include <QTabWidget>
10  #include <QHBoxLayout>
11  #include <QVBoxLayout>
12  #include <QPushButton>
13  #include <QListWidget>
14  #include <QTextBrowser>
15  #include <QLineEdit>
16
17  class ChatServer;
18  class ChatClient;
19  class RemoteSelector;
20
21  class MainWindow : public QMainWindow
22  {
23      Q_OBJECT
24
25  public:
26      MainWindow(QWidget *parent = nullptr);
27      ~MainWindow();
28
29  public:
30      /* 暴露的接口，主动连接设备*/
31      Q_INVOKABLE void connectToDevice();
32
33  signals:
34      /* 发送消息信号 */
35      void sendMessage(const QString &message);
36
37      /* 连接断开信号 */
38      void disconnect();
39
40      /* 发现完成信号 */
41      void discoveryFinished();
42
43      /* 找到新服务信号 */
44      void newServicesFound(const QStringList &list);
45
46  public slots:
```

```
47      /* 停止搜索 */
48      void searchForDevices();
49
50      /* 开始搜索 */
51      void stopSearch();
52
53      /* 找到新服务 */
54      void newServiceFound(QListWidgetItem*);
55
56      /* 已连接 */
57      void connected(const QString &name);
58
59      /* 显示消息 */
60      void showMessage(const QString &sender, const QString &message);
61
62      /* 发送消息 */
63      void sendMessage();
64
65      /* 作为客户端断开连接 */
66      void clientDisconnected();
67
68      /* 主动断开连接 */
69      void toDisconnected();
70
71      /* 作为服务端时，客户端断开连接 */
72      void disconnected(const QString &name);
73
74  private:
75      /* 选择本地蓝牙，些方法未没使用 */
76      int adapterFromUserSelection() const;
77
78      /* 本地蓝牙的 Index */
79      int currentAdapterIndex;
80
81      /* 蓝牙本地适配器初始化 */
82      void localAdapterInit();
83
84      /* 布局初始化 */
85      void layoutInit();
86
87      /* 服务端*/
88      ChatServer *server;
89
```

```
90      /* 多个客户端 */
91      QList<ChatClient *> clients;
92
93      /* 远程选择器，使用本地蓝牙去搜索蓝牙，可过滤蓝牙等 */
94      RemoteSelector *remoteSelector;
95
96      /* 本地蓝牙 */
97      QList<QBluetoothHostInfo> localAdapters;
98
99      /* 本地蓝牙名称 */
100     QString localName;
101
102     /* tabWidget 视图，用于切换页面 */
103     QTabWidget *tabWidget;
104
105     /* 3 个按钮，扫描按钮，连接按钮，发送按钮 */
106     QPushButton *pushButton[5];
107
108     /* 2 个垂直布局，一个用于页面一，另一个用于页面二 */
109     QVBoxLayout *vBoxLayout[2];
110
111     /* 2 个水平布局，一个用于页面一，另一个用于页面二 */
112     QHBoxLayout *hBoxLayout[2];
113
114     /* 页面一和页面二容器 */
115     QWidget *pageWidget[2];
116
117     /* 用于布局，pageWidget 包含 subWidget */
118     QWidget *subWidget[2];
119
120     /* 蓝牙列表 */
121     QListWidget *listWidget;
122
123     /* 显示对话的内容 */
124     QTextBrowser *textBrowser;
125
126     /* 发送消息输入框 */
127     QLineEdit *lineEdit;
128
129 };
130 #endif // MAINWINDOW_H
```

mainwindow.h 是整个代码重要的文件，这里使用了客户端类，服务端类和远程服务端类。

前面介绍的客户端类，服务端类和远程服务端类都是为 mainwindow.h 服务的。我们在编程的时候可以不用改动客户端类，服务端类和远程服务端类了，直接像 mainwindow.h 一样使用它们的接口就可以编程了。

其中笔者还在 mainwindow.h 使用了很多控件，这些控件都是界面组成的重要元素。如果看不懂界面布局，或者理解不了界面布局，请自行学习熟练掌握 Qt 基础。

mainwindow.cpp 代码如下。

```
/**********************************************************************
Copyright © Shi Houde Co., Ltd. 2024-2060. All rights reserved.
* @brief        mainwindow.cpp
* @author       Shi Houde
* @email        shihoude@alientek.com/2101443104@qq.com
* @link         http://www.openedv.com
* @date         2024-07-05
**********************************************************************/
#include "mainwindow.h"
#include "remoteselector.h"
#include "chatserver.h"
#include "chatclient.h"
#include <qbluetoothuuid.h>
#include <qbluetoothserver.h>
#include <qbluetoothservicediscoveryagent.h>
#include <qbluetoothdeviceinfo.h>
#include <qbluetoothlocaldevice.h>
#include <QGuiApplication>
#include <QScreen>
#include <QRect>
#include <QTimer>
#include <QDebug>
#include <QTabBar>
#include <QHeaderView>
#include <QTableView>


static const QLatin1String
    serviceUuid("e8e10f95-1a70-4b27-9ccf-02010264e9c8");

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    /* 本地蓝牙初始化 */
    localAdapterInit();
```

```
28
29      /* 界面布局初始化 */
30      layoutInit();
31  }
32
33  MainWindow::~MainWindow()
34  {
35      qDeleteAll(clients);
36      delete server;
37  }
38
39  /* 初始化本地蓝牙，作为服务端 */
40  void MainWindow::localAdapterInit()
41  {
42      /* 查找本地蓝牙的个数 */
43      localAdapters = QBluetoothLocalDevice::allDevices();
44      qDebug() << "localAdapter: " << localAdapters.count();
45
46      QBluetoothLocalDevice localDevice;
47
localDevice.setHostMode(QBluetoothLocalDevice::HostDiscoverable);
48
49      QBluetoothAddress adapter = QBluetoothAddress();
50      remoteSelector = new RemoteSelector(adapter, this);
51      connect(remoteSelector,
52              SIGNAL(newServiceFound(QListWidgetItem*)),
53              this, SLOT(newServiceFound(QListWidgetItem*)));
54
55      /* 初始化服务端 */
56      server = new ChatServer(this);
57
58      connect(server, SIGNAL(clientConnected(QString)),
59              this, SLOT(connected(QString)));
60
61      connect(server, SIGNAL(clientDisconnected(QString)),
62              this, SLOT(disconnected(QString)));
63
64      connect(server, SIGNAL(messageReceived(QString, QString)),
65              this, SLOT(showMessage(QString, QString)));
66
67      connect(this, SIGNAL(sendMessage(QString)),
68              server, SLOT(sendMessage(QString)));
69
```

```
70      connect(this, SIGNAL(disconnect()),
71              server, SLOT(disconnect()));
72
73      server->startServer();
74
75      /* 获取本地蓝牙的名称 */
76      localName = QBluetoothLocalDevice().name();
77  }
78
79  void MainWindow::layoutInit()
80  {
81      /* 获取屏幕的分辨率，Qt 官方建议使用这
82       * 种方法获取屏幕分辨率，防上多屏设备导致对应不上
83       * 注意，这是获取整个桌面系统的分辨率
84       */
85      QList <QScreen *> list_screen =  QGuiApplication::screens();
86
87      /* 如果是 ARM 平台，直接设置大小为屏幕的大小 */
88  #if __arm__
89      /* 重设大小 */
90      this->resize(list_screen.at(0)->geometry().width(),
91                   list_screen.at(0)->geometry().height());
92  #else
93      /* 否则则设置主窗体大小为 800x480 */
94      this->resize(800, 480);
95  #endif
96
97      /* 主视图 */
98      tabWidget = new QTabWidget(this);
99
100     /* 设置主窗口居中视图为 tabWidget */
101     setCentralWidget(tabWidget);
102
103     /* 页面一对象实例化 */
104     vBoxLayout[0] = new QVBoxLayout();
105     hBoxLayout[0] = new QHBoxLayout();
106     pageWidget[0] = new QWidget();
107     subWidget[0] = new QWidget();
108     listWidget = new QListWidget();
109     /* 0 为扫描按钮，1 为连接按钮 */
110     pushButton[0] = new QPushButton();
111     pushButton[1] = new QPushButton();
112     pushButton[2] = new QPushButton();
```

```
113     pushButton[3] = new QPushButton();
114     pushButton[4] = new QPushButton();
115
116     /* 页面二对象实例化 */
117     hBoxLayout[1] = new QHBoxLayout();
118     vBoxLayout[1] = new QVBoxLayout();
119     subWidget[1] = new QWidget();
120     textBrowser = new QTextBrowser();
121     lineEdit = new QLineEdit();
122     pushButton[2] = new QPushButton();
123     pageWidget[1] = new QWidget();
124
125
126     tabWidget->addTab(pageWidget[1], "蓝牙聊天");
127     tabWidget->addTab(pageWidget[0], "蓝牙列表");
128
129     /* 页面一 */
130     vBoxLayout[0]->addWidget(pushButton[0]);
131     vBoxLayout[0]->addWidget(pushButton[1]);
132     vBoxLayout[0]->addWidget(pushButton[2]);
133     vBoxLayout[0]->addWidget(pushButton[3]);
134     subWidget[0]->setLayout(vBoxLayout[0]);
135     hBoxLayout[0]->addWidget(listWidget);
136     hBoxLayout[0]->addWidget(subWidget[0]);
137     pageWidget[0]->setLayout(hBoxLayout[0]);
138     pushButton[0]->setMinimumSize(120, 40);
139     pushButton[1]->setMinimumSize(120, 40);
140     pushButton[2]->setMinimumSize(120, 40);
141     pushButton[3]->setMinimumSize(120, 40);
142     pushButton[0]->setText("开始扫描");
143     pushButton[1]->setText("停止扫描");
144     pushButton[2]->setText("连接");
145     pushButton[3]->setText("断开");
146
147     /* 页面二 */
148     hBoxLayout[1]->addWidget(lineEdit);
149     hBoxLayout[1]->addWidget(pushButton[4]);
150     subWidget[1]->setLayout(hBoxLayout[1]);
151     vBoxLayout[1]->addWidget(textBrowser);
152     vBoxLayout[1]->addWidget(subWidget[1]);
153     pageWidget[1]->setLayout(vBoxLayout[1]);
154     pushButton[4]->setMinimumSize(120, 40);
155     pushButton[4]->setText("发送");
```

```
156     lineEdit->setMinimumHeight(40);
157     lineEdit->setText("正点原子论坛网址 www.openedv.com");
158
159     /* 设置表头的大小 */
160     QString str = tr("QTabBar::tab {height:40; width:%1};")
161                     .arg(this->width()/2);
162     tabWidget->setStyleSheet(str);
163
164     /* 开始搜寻蓝牙 */
165     connect(pushButton[0], SIGNAL(clicked()),
166             this, SLOT(searchForDevices()));
167
168     /* 停止搜寻蓝牙 */
169     connect(pushButton[1], SIGNAL(clicked()),
170             this, SLOT(stopSearch()));
171
172     /* 点击连接按钮，本地蓝牙作为客户端去连接外界的服务端 */
173     connect(pushButton[2], SIGNAL(clicked()),
174             this, SLOT(connectToDevice()));
175
176     /* 点击断开连接按钮，断开连接 */
177     connect(pushButton[3], SIGNAL(clicked()),
178             this, SLOT(toDisconnected()));
179
180     /* 发送消息 */
181     connect(pushButton[4], SIGNAL(clicked()),
182             this, SLOT(sendMessage()));
183 }
184
185 /* 作为客户端去连接 */
186 void MainWindow::connectToDevice()
187 {
188     if (listWidget->currentRow() == -1)
189         return;
190
191     QString name = listWidget->currentItem()->text();
192     qDebug() << "Connecting to " << name;
193
194     // Trying to get the service
195     QBluetoothServiceInfo service;
196     QMapIterator<QString,QBluetoothServiceInfo>
197         i(remoteSelector->m_discoveredServices);
198     bool found = false;
```

```
199     while (i.hasNext()){
200         i.next();
201 
202         QString key = i.key();
203 
204         /* 判断连接的蓝牙名称是否在发现的设备里 */
205         if (key == name) {
206             qDebug() << "The device is found";
207             service = i.value();
208             qDebug() << "value: " << i.value().device().address();
209             found = true;
210             break;
211         }
212     }
213 
214     /* 如果找到，则连接设备 */
215     if (found) {
216         qDebug() << "Going to create client";
217         ChatClient *client = new ChatClient(this);
218         qDebug() << "Connecting...";
219 
220         connect(client, SIGNAL(messageReceived(QString,QString)),
221                 this, SLOT(showMessage(QString,QString)));
222         connect(client, SIGNAL(disconnected()),
223                 this, SLOT(clientDisconnected()));;
224         connect(client, SIGNAL(connected(QString)),
225                 this, SLOT(connected(QString)));
226         connect(this, SIGNAL(sendMessage(QString)),
227                 client, SLOT(sendMessage(QString)));
228         connect(this, SIGNAL(disconnect()),
229                 client, SLOT(disconnect()));
230 
231         qDebug() << "Start client";
232         client->startClient(service);
233 
234         clients.append(client);
235     }
236 }
237 
238 /* 本地蓝牙选择，默认使用第一个蓝牙 */
239 int MainWindow::adapterFromUserSelection() const
240 {
241     int result = 0;
```

```
242     QBluetoothAddress newAdapter = localAdapters.at(0).address();
243     return result;
244 }
245
246 /* 开始搜索 */
247 void MainWindow::searchForDevices()
248 {
249     /* 先清空 */
250     listWidget->clear();
251     qDebug() << "search for devices!";
252     if (remoteSelector) {
253         delete remoteSelector;
254         remoteSelector = NULL;
255     }
256
257     QBluetoothAddress adapter = QBluetoothAddress();
258     remoteSelector = new RemoteSelector(adapter, this);
259
260     connect(remoteSelector,
261             SIGNAL(newServiceFound(QListWidgetItem*)),
262             this, SLOT(newServiceFound(QListWidgetItem*)));
263
264     remoteSelector->m_discoveredServices.clear();
265     remoteSelector->startDiscovery(QBluetoothUuid(serviceUuid));
266     connect(remoteSelector, SIGNAL(finished()),
267             this, SIGNAL(discoveryFinished()));
268 }
269
270 /* 停止搜索 */
271 void MainWindow::stopSearch()
272 {
273     qDebug() << "Going to stop discovery...";
274     if (remoteSelector) {
275         remoteSelector->stopDiscovery();
276     }
277 }
278
279 /* 找到蓝牙服务 */
280 void MainWindow::newServiceFound(QListWidgetItem *item)
281 {
282     /* 设置项的大小 */
283     item->setSizeHint(QSize(listWidget->width(), 50));
284
```

```
285     /* 添加项 */
286     listWidget->addItem(item);
287 
288     /* 设置当前项 */
289     listWidget->setCurrentRow(listWidget->count() - 1);
290 
291     qDebug() << "newServiceFound";
292 
293     // get all of the found devices
294     QStringList list;
295 
296     QMapIterator<QString, QBluetoothServiceInfo>
297         i(remoteSelector->m_discoveredServices);
298     while (i.hasNext()){
299         i.next();
300         qDebug() << "key: " << i.key();
301         qDebug() << "value: " << i.value().device().address();
302         list << i.key();
303     }
304 
305     qDebug() << "list count: "  << list.count();
306 
307     emit newServicesFound(list);
308 }
309 
310 /* 已经连接 */
311 void MainWindow::connected(const QString &name)
312 {
313     textBrowser->insertPlainText(tr("%1:已连接\n").arg(name));
314     tabWidget->setCurrentIndex(0);
315     textBrowser->moveCursor(QTextCursor::End);
316 }
317 
318 /* 接收消息 */
319 void MainWindow::showMessage(const QString &sender,
320                              const QString &message)
321 {
322     textBrowser->insertPlainText(QString::fromLatin1("%1: %2\n")
323                                  .arg(sender, message));
324     tabWidget->setCurrentIndex(0);
325     textBrowser->moveCursor(QTextCursor::End);
326 }
327 
```

```
328 /* 发送消息 */
329 void MainWindow::sendMessage()
330 {
331     showMessage(localName, lineEdit->text());
332     emit sendMessage(lineEdit->text());
333 }
334
335 /* 作为客户端断开连接 */
336 void MainWindow::clientDisconnected()
337 {
338     ChatClient *client = qobject_cast<ChatClient *>(sender());
339     if (client) {
340         clients.removeOne(client);
341         client->deleteLater();
342     }
343
344     tabWidget->setCurrentIndex(0);
345     textBrowser->moveCursor(QTextCursor::End);
346 }
347
348 /* 主动断开连接 */
349 void MainWindow::toDisconnected()
350 {
351     emit disconnect();
352     textBrowser->moveCursor(QTextCursor::End);
353     tabWidget->setCurrentIndex(0);
354 }
355
356 /* 作为服务端时，客户端断开连接 */
357 void MainWindow::disconnected(const QString &name)
358 {
359     textBrowser->insertPlainText(tr("%1:已断开\n").arg(name));
360     tabWidget->setCurrentIndex(0);
361     textBrowser->moveCursor(QTextCursor::End);
362 }
```

mainwindow.cpp 则是整个项目的核心文件，包括处理界面点击的事件，客户端连接，服务端连接，扫描蓝牙，断开蓝牙和连接蓝牙等。

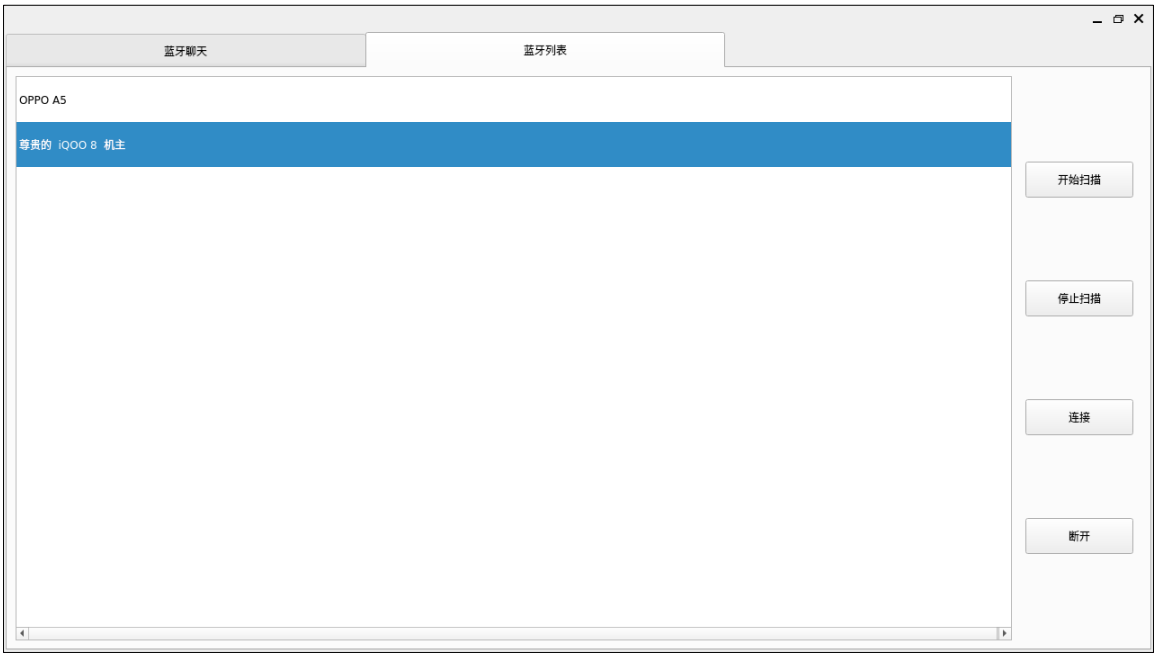
7.3 程序运行效果

本例程运行后，默认开启蓝牙的服务端模式，可以用手机安装蓝牙调试软件（安卓手机如蓝牙调试宝、蓝牙串口助手）。

这里我们用蓝牙调试宝进行测试，当我们点击蓝牙列表页面时，点击扫描后请等待扫描的结果，选中需要连接的蓝牙再点击连接。

下面将程序运行编译部署到正点原子 ATK-DLRK3568 开发板上面进行板载蓝牙测试效果，用手机连接后运行的蓝牙聊天第一页效果图。

下面程序运行的蓝牙聊天第二页效果图。



安卓手机可以用蓝牙调试宝等软件进行配对连接。IOS 手机请下载某些蓝牙调试软件测试即可。手机接收到的消息如下。

详细请看“RK3568 开发板\开发板光盘 A 盘-基础资料\10、用户手册\01、测试文档\01【正点原子】ATK-DLRK3568_Buildroot 系统快速体验手册 V1.2.pdf”**3.11** 小节蓝牙测试开启蓝牙，启用蓝牙被扫描后，**先进行配对**，手机用蓝牙调试软件就可以连接上进行聊天了。