

浙江大学

OOP(object-oriented programming)

Project Final Report



- | | | | |
|-----------|------------|------------------|-------------------|
| 1. Name : | <u>邱明冉</u> | Student Number : | <u>3190104698</u> |
| 2. Name : | <u>孟俊邑</u> | Student Number : | <u>3190106104</u> |
| 3. Name : | <u>郭胜贤</u> | Student Number : | <u>3190106080</u> |
| 4. Name : | <u>罗志凡</u> | Student Number : | <u>3190103256</u> |

2019~2020 秋冬学期 2021 年 1 月 10 日

1、题目描述和题目要求

本项目实现了一个基于绘制原理图的**逻辑电路设计**软件。用户可根据个人需要，在绘制区域进行原理图的设计。该软件有文件操作、元件库、设计原理图等模块。用户可以打开或新建图片文件作为背景，设计好原理图之后也可以将图片文件保存。设计时，用户可以在元件库中选择需要的元件，例如二输入与门、二输入或门以及电路连接线等等。选择元件之后即可在设计原理图区域进行设计，当然也可以进行擦除等基本操作。在文件区域，可以选择对画笔粗细和颜色进行更改。

2、需求分析

本项目实现的是一款简单清晰的逻辑电路设计软件，它有基本的逻辑门，如二输入与门、二输入或门、三输入与门等等。用户可根据个人需要，先导入背景图片，再采用多种基本元件和电路连线构建出庞大的逻辑电路。设计完成后可以保存图像，简单方便。同时，我们还设计了不同颜色的选择，以便不同用户的喜好而做出相应选择，当然在同一个逻辑电路中采用不同颜色的线条也会使得逻辑更加清晰。总的来说，此软件能够满足以原理图实现逻辑电路设计的基本需求，对文件打开和保存等也做得很好。

3、总体设计

3.1 功能模块设计

3.1.1 mainwindow.cpp

此部分包含 `MainWindow` 构造函数和析构函数。

构造函数中，实现了软件主页面，包括建立菜单栏，建立动作，设置动作快捷键，设置状态栏提示信息，创建画布等等。其中大部分利用 `QT` 自带函数进行创建按钮（为按钮设置图表样式、大小和坐标，为按钮命名），链接按钮（为按钮赋值，从 1 至 14 每一个正整数相当于各个按钮的代号，之后与用户交互时将触发相应的函数）以及创建并配置参数停靠窗。其中还包含了 `connect` 函数，它是 `Qt` 信号槽的特有函数。信号槽是 `Qt` 框架引以为豪的机制之一。所谓信号槽，实际就是观察者模式。当某个事件发生之后，比如，按钮检测到自己被点击了一下，它就会发出一个信号（`signal`）。这种发出是没有目的的，类似广播。如果有对象对这个信号感兴趣，它就会使用连接（`connect`）函数，意思是，将想要处理的信号自己的一个函数（称为槽（`slot`））绑定来处理这个信号。也就是说，当信号发出时，被连接的槽函数会自动被回调。这就类似观察者模式：当发生了感兴趣的事件，某一个操作就会被自动触发。不需要用析构函数实现功能，因此在析构函数中不需要写任何内容。

3.1.2 painter.cpp

此部分实现画原理图的细节，包括鼠标的点击、移动、松开等操作和每一个图样在画布上显示的所有细节功能。

3.1.3 slot.cpp

此部分实现类 MainWindow 的所有成员函数，实现对图片文件的操作（新建、打开、保存、另存为）、提取元器件、改变画笔参数（粗细、颜色）等，实现细节在“类主要功能描述”中具体介绍。具体代码如下：

3.2 数据结构设计

此工程有两个类定义，分别是 MainWindow 类和 PaintWidget 类。
MainWindow 类定义如下：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    PaintWidget *painter;
    QMenuBar *menubar; //菜单栏
    QStatusBar *statusbar; //状态栏
    //QToolBar *maintoolbar;

    QDockWidget *canvasDock; //画布停靠窗
    QDockWidget *toolboxDock; //工具箱停靠窗
    QToolBar *penArgDock; //笔尖参数停靠窗
    QButtonGroup *designLib; //工具箱中的设计库
    QButtonGroup *penSizeChg; //修改笔尖尺寸

    QString path_of_curr_file;
    //Status
    QLabel *tempStatus;
    QLabel *permStatus;
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    /* 四个 slot 分别对应菜单上的四个动作 */
    void _NewFile();
    void _OpenFile();
    void _SaveFile();
    void _SaveAsFile();
```

```

    void _ConnectUs();
    void designLibSelect(int id);
    void add_pen_width();
    void minus_pen_width();
    void set_red();
    void set_blue();
    void set_black();
};

```

MainWindow 类是 QMainWindow 的派生类。除了在下文即将提到的所有成员函数之外，此类还定义了很多私有成员变量，包括 `QMenuBar *menubar`（菜单栏），`QStatusBar *statusbar`（状态栏），`QDockWidget *canvasDock`（画布停靠窗），`QDockWidget *toolboxDock`（工具箱停靠窗），`QToolBar *penArgDock`（笔尖参数停靠窗）等等描述该类的参数。此类声明了构造函数和析构函数。

PaintWidget 类定义如下：

```

class PaintWidget : public QLabel
{
    Q_OBJECT
public:
    explicit PaintWidget(QWidget *parent = nullptr);
    void paint(QImage &CurrImg);
    void setImage(QImage img);
    QImage getImage();
    void setPenColor(QColor color);
    void setPenWidth(int width);
    void setShape(int type);
    QColor getPenColor() const;
    int getPenWidth() const;
    QImage image;
private:
    int penWidth;
    int drawtype;
    QColor penColor;
    QImage tmpImg;
    QPoint endPoint;
    QPoint startPoint;
    bool drawing;
protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);
};

```

PaintWidget 类是 QLabel 的派生类。除了在下文即将提到的所有成员函数之外，PaintWidget 类中还定义了很多私有成员变量，包括 `int penWidth`（笔的宽度），`int drawtype`（元件类型），`QColor penColor`（笔的颜色）等等描述该类的参数。

3.3 类主要功能描述

(1) MainWindow 类:

此类的成员函数比较特殊，它是一种“槽”。信号和槽是 Qt 特有的一种机制（上文已经提及）。在这里，我们使用的是 private slots，它是指在这个区内声明的槽意味着只有类自己可以将信号与之相连接。所以本质上还是私有的成员函数，唯一的区别是：槽可以与信号连接在一起，每当和槽连接的信号被发射的时候，就会调用这个槽。

此类中包含如下成员函数：

```
void _NewFile();
```

函数原型：Void _NewFile();

功能描述：新建文件

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：QImage img;

重要局部变量用途描述：

QImage img = QImage(canvas_width*2, canvas_height*2, QImage::Format_RGB32);
新建一个图片用来当画布，数据储存到定义的 QImage 类型的 img 中。

函数算法描述：通过 QT 自带的 QImage 函数新建一个图片，将图片填充为白色，用来当背景。

```
void _OpenFile();
```

函数原型：void _OpenFile();

功能描述：打开文件

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：QString path; QImage *open_file;

重要局部变量用途描述：

QString path = QFileDialog::getOpenFileName(this, "选择电路原理图", ".",
"schematic(*.jpg, *.png, *.bmp)");

定义 QString 类型的局部变量 path 存储用户当前的文件路径。

QImage *open_file = new QImage();

定义 QImage 类型的指针 open_file 指向新开辟的空间。

函数算法描述：通过 QFileDialog::getOpenFileName 获取当前需打开的文件的路

径，若路径为空（无效）则发生错误，打开文件失败；否则文件路径有效，则获取该路径并打开文件，显示该图片。

void _SaveFile();

函数原型：void _SaveFile();

功能描述：保存文件

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：QString path; QImage save_file;

重要局部变量用途描述：

QString path = QFileDialog::getSaveFileName(this, "保存电路原理图", ".", "schematic(*.jpg, *.png, *.bmp)");

定义 QString 类型的局部变量 path 存储用户当前的文件路径。

QImage save_file = painter->getImage();

定义 QImage 类型的变量 Save_file 存储图片文件数据

函数算法描述：如果之前没有经过 open 操作，当前文件路径是空的，需要选择一个路径保存。保存的核心操作是 QT 自带的.save();

void _SaveAsFile();

函数原型：void _SaveAsFile();

功能描述：文件另存为

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：QString path; QImage save_file;

重要局部变量用途描述：

QString path = QFileDialog::getSaveFileName(this, "另存为电路原理图", ".", "schematic(*.jpg, *.png, *.bmp)");

定义 QString 类型的局部变量 path 存储用户当前的文件路径。

QImage save_file = painter->getImage();

定义 QImage 类型的变量 Save_file 存储图片文件数据

函数算法描述：不需要判断之前是否经过 open 操作，直接选择一个路径保存。保存的核心操作是 QT 自带的.save();

void designLibSelect(int id);

函数原型：void designLibSelect(int id);

功能描述：用户选择所需元件（或其他功能如擦除）

参数描述：int id （从 1 至 14，每个值分别对应元件库中的一个元件）

返回值描述：无返回值

重要局部变量定义：无局部变量

函数算法描述：通过一个 switch 函数，将每一个传进的 id 值（1 至 14），分别对应某一种形状（不同的元件类型）。

void add_pen_width();

函数原型：void add_pen_width();

功能描述：使画笔变粗

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：无局部变量

函数算法描述：通过 `getPenWidth` 函数读取笔的粗细，若小于 10，则通过 `setPenWidth` 函数对笔尖宽度增加 1，达到使画笔变粗的目的。

```
void minus_pen_width();
```

函数原型：`void minus_pen_width();`

功能描述：使画笔变细

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：无局部变量

函数算法描述：通过 `getPenWidth` 函数读取笔的粗细，若大于 1，则通过 `setPenWidth` 函数对笔尖宽度减小 1，达到使画笔变细的目的。

```
void set_red();
```

函数原型：`void set_red();`

功能描述：将画笔变为红色

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：无局部变量

函数算法描述：通过 `setPenColor` 函数将笔的颜色置为红色。

```
void set_blue();
```

函数原型：`void set_blue();`

功能描述：将画笔变为蓝色

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：无局部变量

函数算法描述：通过 `setPenColor` 函数将笔的颜色置为蓝色。

```
void set_black();
```

函数原型：`void set_black();`

功能描述：将画笔变为黑色

参数描述：无参数

返回值描述：无返回值

重要局部变量定义：无局部变量

函数算法描述：通过 `setPenColor` 函数将笔的颜色置为黑色。

(2) PaintWidget 类:

此类中包含如下成员函数:

```
void paint(QImage &CurrImg);
```

函数原型: void paint(QImage &CurrImg);

功能描述: 实现 14 种不同的操作

参数描述: QImage &CurrImg (图标地址)

返回值描述: 无返回值

重要局部变量定义: QPen pen;

重要局部变量用途描述: 定义 QPen 型变量 pen, 进而一颗使用 pen.setWidth, pen.setColor 等函数, 对画笔参数进行描述。

函数算法描述: 定义画笔颜色和粗细等相关参数, 利用 switch 函数, 根据选择元件的类型匹配到各个画图的函数, 实现相关功能。

```
void setImage(QImage img);
```

函数原型: void setImage(QImage img);

功能描述: 设置画布参数

参数描述: QImage img (画布类)

返回值描述: 无返回值

重要局部变量定义: QImage img1;

重要局部变量用途描述: 定义 QImage 型变量 img1, 以获取 img 参数

函数算法描述: 为参数赋值

```
void setPenColor(QColor color);
```

函数原型: void setPenColor(QColor color);

功能描述: 设置笔的颜色

参数描述: QColor color (笔的颜色)

返回值描述: 无返回值

重要局部变量定义: 未定义局部变量

```
void setPenWidth(int width);
```

函数原型: void setPenWidth(int width);

功能描述: 设置笔的宽度 (粗细)

参数描述: int width (笔的宽度值)

返回值描述: 无返回值

重要局部变量定义: 未定义局部变量

```
void setShape(int type);
```

函数原型: void setShape(int type);

功能描述: 获得绘制图片的类型

参数描述: int type (类型编号)

返回值描述: 无返回值

重要局部变量定义: 未定义局部变量


```
int getPenWidth() const;
```

函数原型: int getPenWidth() const;

功能描述: 获取画笔宽度 (粗细)

参数描述: 无参数

返回值描述: return penWidth; 返回画笔宽度

重要局部变量定义: 无局部变量

```
void paintEvent(QPaintEvent *);
```

函数原型: void paintEvent(QPaintEvent *);

功能描述: 设置绘画初始状态

参数描述: QPaintEvent * (鼠标的操作)

返回值描述: 无返回值

重要局部变量定义: 无

```
void mousePressEvent(QMouseEvent *);
```

函数原型: void mousePressEvent(QMouseEvent *);

功能描述: 鼠标左键按下时, 获得绘图起始点数据

参数描述: QMouseEvent * (鼠标的操作)

返回值描述: 无返回值

重要局部变量定义: 未定义局部变量

函数算法描述: 判断鼠标操作, 若是左键单击, 则为 drawing 赋值为 true, 即达到“开始画”的目的, 画了一个点。

```
void mouseMoveEvent(QMouseEvent *);
```

函数原型: void mouseMoveEvent(QMouseEvent *);

功能描述: 鼠标拖动时, 当前点为绘图终点数据

参数描述: QMouseEvent * (鼠标的操作)

返回值描述: 无返回值

重要局部变量定义: 未定义局部变量

函数算法描述: 判断鼠标操作, 若是左键拖动, 则将当前点作为终点画出一条线。

```
void mouseReleaseEvent(QMouseEvent *);
```

函数原型: void mouseReleaseEvent(QMouseEvent *);

功能描述: 鼠标 release 时, 在真实画布上绘制

参数描述: QMouseEvent * (鼠标的操作)

返回值描述: 无返回值

重要局部变量定义: 未定义局部变量

函数算法描述: 将 drawing 赋值为 false, 即松开笔, 画的线固定住。

4. 部署与运行

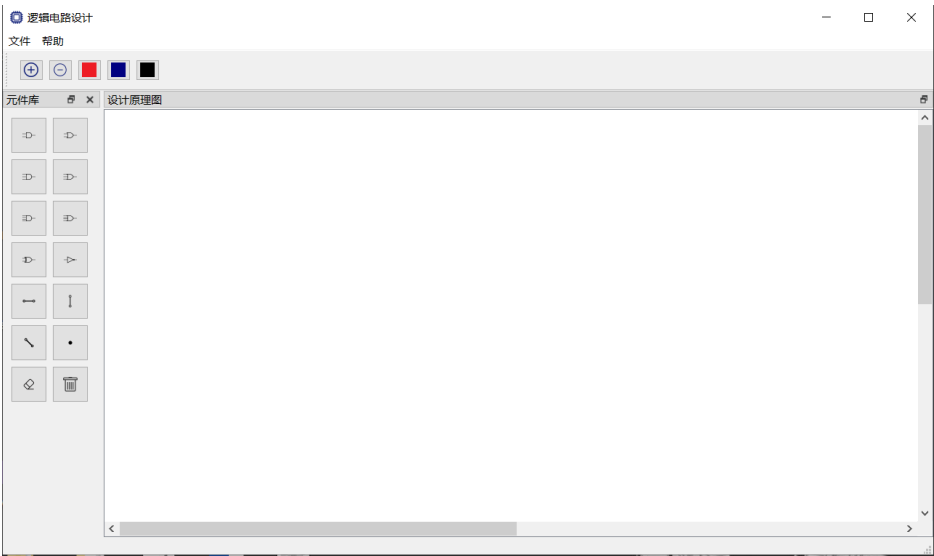
4.1 编译安装运行说明

（此部分介绍如何由提交的源代码包，进行存放、编译生成.exe 文件的过程说明，以及运行.exe 后的用户使用手册。）

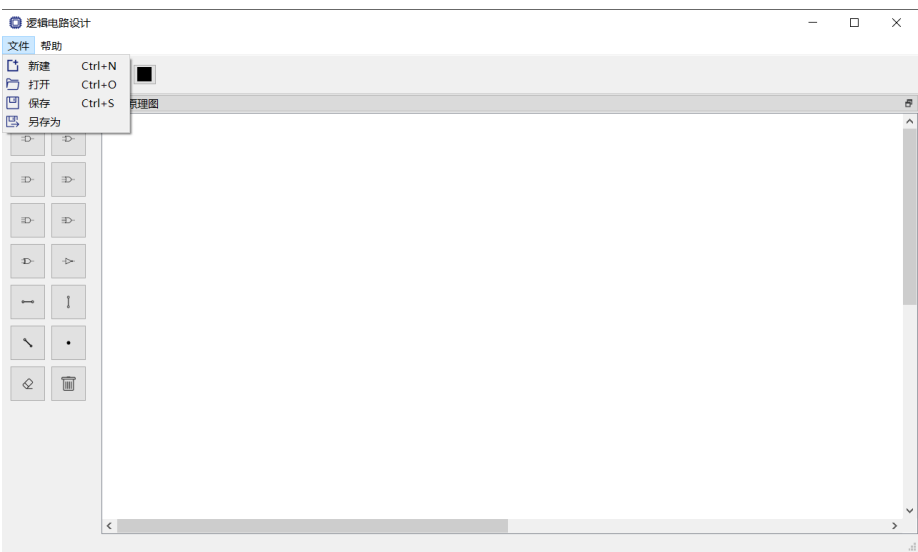
可以直接打开压缩包中的.exe 文件，或者在 Qt creator 中打开压缩包中的.pro 文件，自行编译链接，生成软件主页面。

4.2 典型测试情况

软件初始化界面如下：

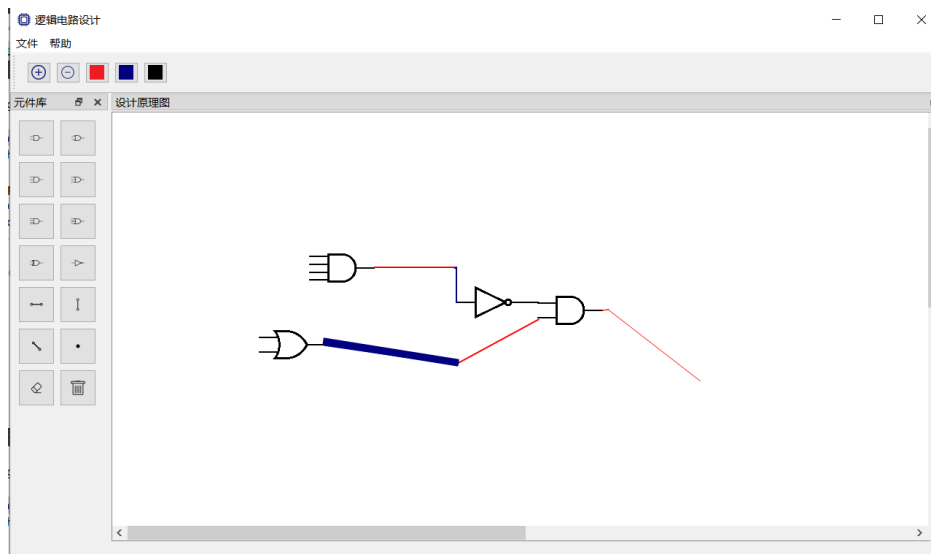


鼠标移动到“文件”处，可以对图片文件进行操作，包括新建、打开、保存和另存为。可以鼠标左键单击实现功能，也可以通过快捷键实现功能。



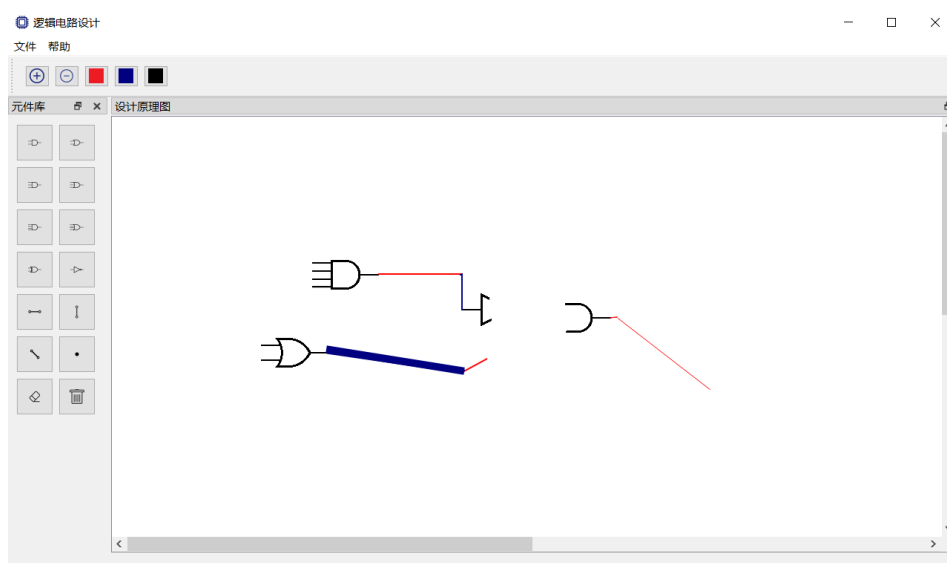
若直接在空白画布上作原理图，效果如下：

（鼠标左键单击左侧元件库选择不同元件、点击上方加号和减号分别可以将线条变粗和变细，点击上方三个颜色可以改变线条颜色……）



若使用擦除按钮，则效果如下：

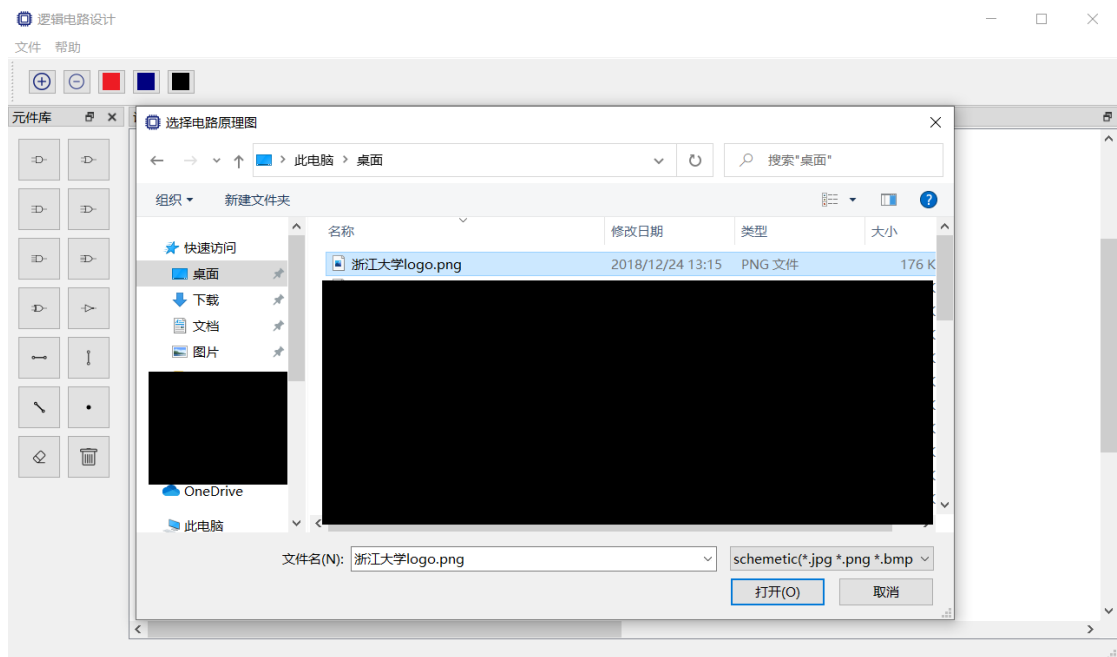
（可以擦除一部分原理图）



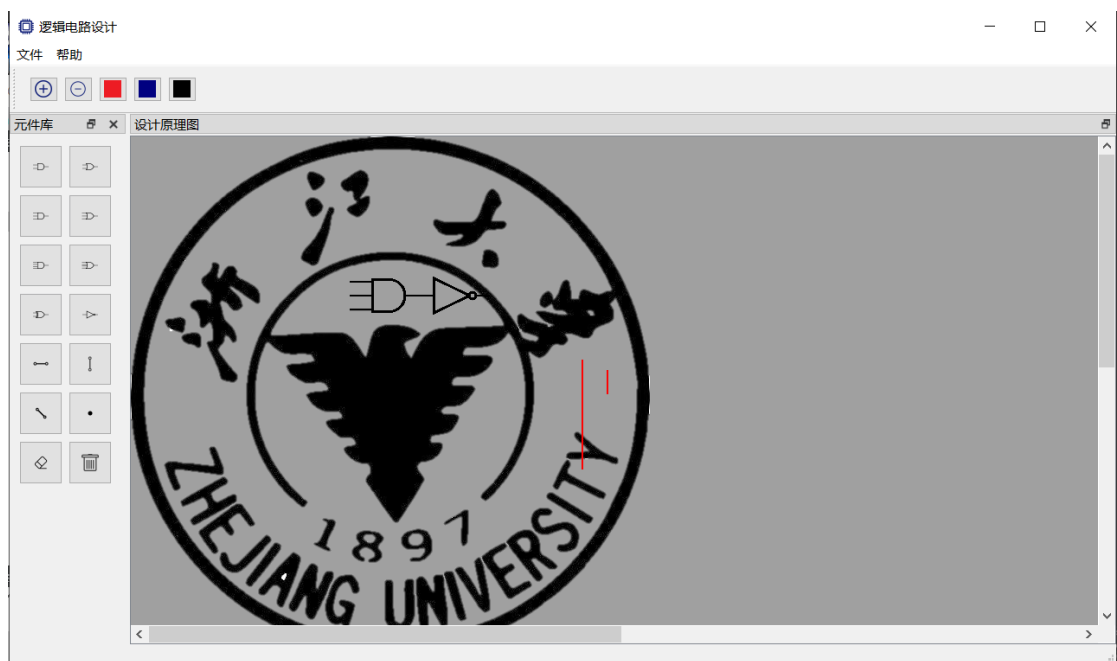
若使用“清除整张画布”按钮，则整张画布内容全部消失。

可以导入图片作为背景，具体方法如下：

在“文件”处选择“打开”/快捷键 ctrl-o，选择背景图片



之后就可以在背景上作图了。



作好图之后可以在“文件”处选择“保存”或快捷键 ctrl-s 保存整个图片文件至本地。

5. 组内分工

5.1 组内分工情况

邱明冉：IDE 和 Compiler 的调试，代码实现与调试，Project 总体测试，组内分工。

孟俊邑：IDE 和 Compiler 调试，培训组内成员第三方库 QT 的下载、安装与使用，Project 总体架构设计。

郭胜贤：最终报告撰写，Project 总体测试。

罗志凡：具体绘图功能的代码实现与调试，Project 总体测试与完善。

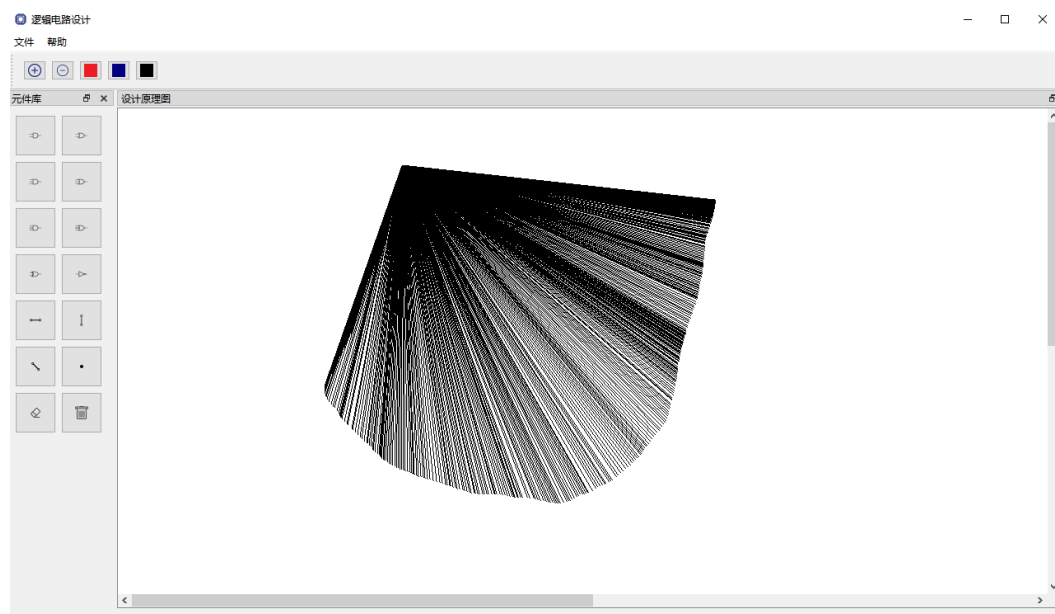
5.2 个人实践过程中遇到的难点及解决方案

罗志凡：

(1) 直线绘制功能的实现

a) 问题：

最初是想绘制一条自由倾斜的直线，但在绘制过程中却出现了如下的问题：



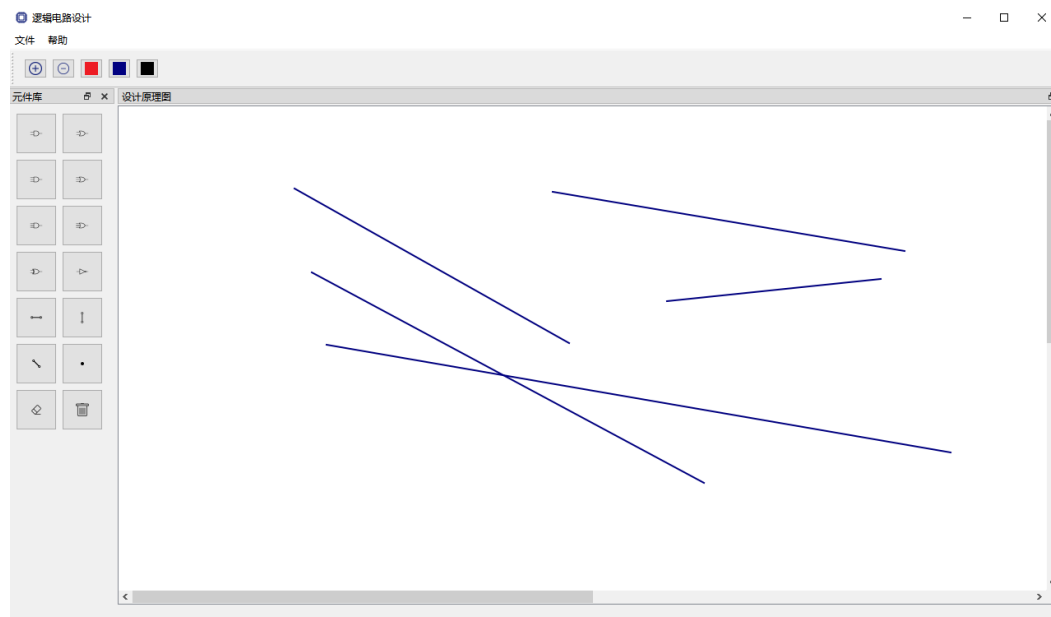
经分析后，发现是由于直接画在真实画布上，而每次移动鼠标都进行了一次绘制，导致了问题的发生。

b) 解决

在代码中加上一个临时画布，在每次绘制时都是在临时画布上绘制，并且每次鼠标移动时都重置临时画布(将其赋值为真实画布)，在鼠标放下时将临时画布的值赋给真实画布完成绘图。由此实现绘制的正常功能，以及绘制时的预览，实现代码如下：

```
void PaintWidget::paintEvent(QPaintEvent *) //画笔的绘画事件
{
    //qDebug() << "paintEvent" << endl;
    QPainter painter(this);
    if (drawing)
        painter.drawImage(0, 0, tmpImg);    //先画在临时画布上
    else
        painter.drawImage(0, 0, image);    //鼠标 release 时，将临时画布内容画到真实画布上
}
void PaintWidget::mouseMoveEvent(QMouseEvent *event)
{
    //qDebug() << startPoint << endPoint << endl;
    if (event->buttons() & Qt::LeftButton){
        endPoint = event->pos();    //鼠标拖动时，当前点为绘图终点数据
        tmpImg = image;
        paint(tmpImg);
    }
}
void PaintWidget::mouseReleaseEvent(QMouseEvent *)
{
    drawing = false;
    paint(image); //鼠标 release 时，在真实画布上绘制
}
```

c) 问题解决



(2) 橡皮擦功能的实现

无法实现拖动擦除，经分析需要橡皮擦直接在真实画布上绘图，如上反向调整后问题解决。

(3) 图片调用时的相对路径问题，通过查阅文档得以解决。

邱明冉：

(1) 按钮的信号和槽未能成功连接。我最开始用 `connect` 将按钮和对应处理的槽函数链接的时候，发现点击按钮后没有反应。同时编译时也提示“某信号没有成功连接”这样的蓝色英文小字提示。于是在槽函数中加入了 `QDebug`，让他输出信息（对应于平时使用的 `printf` 调试法）。果然没有信息的输出。于是我试着切换发出的信号，发现使用 `clicked()` 信号能够正常工作。

(2) 工程素材的导入。最开始我试图使用相对路径，但是没有成功。我也不清楚它的当前工作目录是什么。于是我使用绝对工作路径，一种是“`G:/QtProject/...`”的写法，一种是“`G:\\QtProject\\...`”的写法，虽然网上说采用前一种写法，但是我还是采用后一种写法了，我认为这是 `windows` 风格下的目录。最终我用一个全局字符串常量 `pwd+ “icon\\a.png”` 这样的方法做成了伪相对路径。相对路径的真正实现由组员进行了完善。

6. 合作纪要

2020 年 11 月 25 日：分工完成

2020 年 11 月 30 日：配置好 IDE

2020 年 12 月 2 日：demo 核心的编写

2020 年 12 月 8 日：整体框架的搭建

2020 年 12 月 15 日：开始实现每个模块的功能
















2020 年 12 月 25 日：完善所有基本功能，优化界面显示

2021 年 1 月 2 日：查漏补缺，修改小 bug，增加特殊功能

2021 年 1 月 5 日：报告撰写

2021 年 1 月 10 日：检查整个工程，报告最终定稿

Github 与钉钉群聊天记录截图：

This branch is 7 commits ahead of main.			Pull request	Compare
 SiO-2 add some commits			1993c55 10 hours ago	🕒 11 commits
 icon	Fixed some bugs	16 hours ago		
 .gitignore	test2	28 days ago		
 CircuitDesign.pro	update	2 days ago		
 CircuitDesign.pro.user	add some commits	10 hours ago		
 CircuitDesign.pro.user.59b24eb	update	2 days ago		
 README.md	Initial commit	last month		
 counter.h	完成框架	3 days ago		
 icon.qrc	fixed some bugs	16 hours ago		
 main.cpp	完成框架	3 days ago		
 mainwindow.cpp	update	14 hours ago		
 mainwindow.h	update	14 hours ago		
 painter.cpp	add some commits	10 hours ago		
 painter.h	完成框架	3 days ago		
 slot.cpp	update	14 hours ago		





7. 总结

7.1 程序亮点或创新之处

元件库所含元件类型多样，可以满足用户基本需求，且具有擦除功能，也可以对线条粗细、颜色等进行自定义选择，适应广大不同需求的用户。程序具有新建，打开，保存，另存为图片文件的功能，操作方便，上手简单。整体来看，窗口清爽简洁，构造美观。程序利用 Qt creator，利用了许多自带函数和功能，较好地实现了用户基本需求，实现起来很方便，很有新意。

7.2 应用知识点

7.2.1 基本知识点

- (1) 类和对象的基本概念
- (2) private, public, protected
- (3) 成员函数，构造函数，析构函数
- (4) 继承与派生

7.2.2 补充知识点

(1) QT 信号与槽机制

信号槽是 Qt 框架引以为豪的机制之一。所谓信号槽，实际就是观察者模式。当某个事件发生之后，比如，按钮检测到自己被点击了一下，它就会发出一个信号（**signal**）。这种发出是没有目的的，类似广播。如果有对象对这个信号感兴趣，它就会使用连接（**connect**）函数，意思是，将想要处理的信号自己的一个函数（称为槽（**slot**））绑定来处理这个信号。也就是说，当信号发出时，被连接的槽函数会自动被回调。这就类似观察者模式：当发生了感兴趣的事件，某一个操作就会被自动触发。

信号和槽是 Qt 特有的信息传输机制，是 Qt 设计程序的重要基础，它可以让互不干扰的对象建立一种联系。

槽的本质是类的成员函数，其参数可以是任意类型的。和普通 C++ 成员函数几乎没有区别，它可以是虚函数；也可以被重载；可以是公有的、保护的、私有的、也可以被其他 C++ 成员函数调用。唯一区别的是：槽可以与信号连接在一起，每当和槽连接的信号被发射的时候，就会调用这个槽。

信号与槽的实现：

- **public slots:** 在这个区内声明的槽意味着任何对象都可将信号与之相连接。这对于组件编程非常有用，你可以创建彼此互不了解的对象，将它们的信号与槽进行连接以便信息能够正确的传递。

- **protected slots:** 在这个区内声明的槽意味着当前类及其子类可以将信号与之相连接。这适用于那些槽，它们是类实现的一部分，但是其界面接口却面向外部。
- **private slots:** 在这个区内声明的槽意味着只有类自己可以将信号与之相连接。这适用于联系非常紧密的类。

private slots:

```
void helloslot();

// 对于发出信号的对象的函数，需要使用 connect 方法将其与槽函数连接
connect(button, SIGNAL(clicked()), this, SLOT(helloslot()));
// 连接信号与槽
connect(const QObject *sender, const QMetaMethod &signal,
        const QObject *receiver, const QMetaMethod &method,
        Qt::ConnectionType type = Qt::AutoConnection);

// 可以认为是系统观察着所有对象的状态，侦听他们的变化，当
sender->signal()发出信号时（注意，这里的信号不是指函数的返回值），
receiver->slot()被调用，最后一个为连接模式，使用缺省值即可
```

(2) QWidget Class

所有的自定义窗口类都是由 **QWidget** 继承而来的,基本上我们所需要自己写的类都需要继承 **QWidget** 或者其派生子类。将一个 **QWidget** 组件作为另一个组件的子组件是一个很常见的行为，其表现为共用一个调色板，子组件显示在父组件的区域内，父组件消亡时，子组件也会消失，且子对象的创建必须在与其父对象相关的线程内。

7.3 心得体会

邱明冉：

心得体会：相比于 C 大程使用 **libGraphics** 来进行图形化界面设计，使用第三方库 **Qt** 要方便得多。**Qt** 的教程在网上可以容易地找到，同时也多亏了已经有 **Qt** 使用经验的组员孟俊邑，我们组的 **Qt** 入门相对来说要快很多。经验教训的其中之一是使用 **Linux** 平台进行编译。因为 **Qt** 库文件比较庞大，用 **windows** 编译较慢。

自我评价：作为组长组内沟通方面有做的不到位的地方，组员误以为这次大作业的 DDL 是 1 月 29 号。然后一些进度关注的情况还不够，导致了拖 DDL 的情况发生。在代码方面，我主要负责框架部分，即除了元件的具体绘制以外的部分。

但是因为精力和能力有限，原本打算加进去的大量画布操作，例如画布的旋转、裁剪、整体移动，最后没有实装。不过，我认为自己做的比较好的地方是写了很多的注释，以及代码没有 BUG，对另一位同学在此基础上继续 coding 带来了方便。另外，自己作为组长，在 git 的使用上还有所欠缺，以后还需要进一步的学习。

孟俊邑：

心得体会：这次的大程序设计让我加深了对于 OOP 的基本语法和面向对象设计思想的理解，对已经学到的知识得到了相应的巩固，而对于 QT 这个庞大而强大的框架，如何定位自己的需求，找出解决方案是很重要的，要先设计出整体架构和功能基本实现思路，管理一个庞大项目才能有条不紊。同时我进一步认识到了协作编程和交流合作的重要性，通过 git 私有仓库有效管理团队的代码，高效合作编程，对于错误定位大有帮助。

自我评价：本次大程序设计中，我主要是负责了 QT 环境的搭建踩坑和核心功能 demo 的编写，给出了核心功能的实现方式，便于组员们将其完善和丰满。

郭胜贤：

心得体会：o 对于此次 oop 的大作业，我们小组以 qt 为基础写了一个逻辑电路设计软件。我一开始对 qt 很陌生，但是有一位组员对 qt 很了解，他教我们搭建好了环境和编译器等等，我们也逐渐地掌握了一些 qt 自带的功能和相应的函数实现。总体来过程不算很艰难，有了 c 大程的经验，再加上我们组的齐心协力，分工配合，大家都完成得很不错。尽管中间出了一些小插曲，有位组员因为个人原因退出了大作业的设计，但是这不会影响我们的进程。由于这学期其他几门课压力也非常大，所以最终成果当然不是完美的，也是存在一些不足之处的，不过结果还是非常令人满意。

自我评价：在实现这个程序的过程中，我主要是负责对前几位同学构建好的主要框架和内容进行查漏补缺，测试程序和数据，并进行实验报告的全部撰写。

罗志凡：

心得体会：在这次大作业的过程中，遇到了许多困难，但也有不少收获了。首先是对于 QT 的认识，这是我第一次使用 QT 进行带有图形化界面的软件开发，其功能的强大深深震撼了我。在学习的过程中，也掌握了使用 QT 库函数的技巧，日后有机会一定会再用 QT 做些东西。

自我评价：由于我没有上过 C 大程，这是我第一次正式参与了团队合作开发的整个流程，缺乏经验，在分工阶段出现了磨合不足的问题。在开发过程中，我在组长给出的框架上完成了所有画图功能的实现与调试，并将整体框架进一步完善，并且负责了最终的调试，进行 BUG 的修复。在开发过程中，由于时间不够充裕以及习惯于面向过程的程序设计，所写的程序中带有了过多面向过程的属性，写得差不多时才发现到这个问题，却没有足够的时间修改，算是一个遗憾。

8、参考文献和资料

<https://doc.qt.io/qt-5/qwidget.html>

9、联系方式

电话：17799856065 邮箱：408173145@qq.com

10、附录

本工程所有代码如下：

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtWidgets>
#include "painter.h"

extern const int canvas_width;
extern const int canvas_height;
extern const int btn_size;

class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    PaintWidget *painter;
    QMenuBar *menubar; // 菜单栏
    QStatusBar *statusbar; // 状态栏
    //QToolBar *maintoolbar;

    QDockWidget *canvasDock; // 画布停靠窗
    QDockWidget *toolboxDock; // 工具箱停靠窗
    QToolBar *penArgDock; // 笔尖参数停靠窗
    QButtonGroup *designLib; // 工具箱中的设计库
    QButtonGroup *penSizeChg; // 修改笔尖尺寸
```

```

    QString path_of_curr_file;
    //Statues
    QLabel *tempStatus;
    QLabel *permStatus;
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    /* 四个 slot 分别对应菜单上的四个动作 */
    void _NewFile();
    void _OpenFile();
    void _SaveFile();
    void _SaveAsFile();
    void _ConnectUs();
    void designLibSelect(int id);
    void add_pen_width();
    void minus_pen_width();
    void set_red();
    void set_blue();
    void set_black();
};

#endif // MAINWINDOW_H

```

painter.h

```

#ifndef PAINTER_H
#define PAINTER_H

#include <QMainWindow>
#include <QtWidgets>

class PaintWidget : public QLabel
{
    Q_OBJECT
public:
    explicit PaintWidget(QWidget *parent = nullptr);
    void paint(QImage &CurrImg);
    void setImage(QImage img);
    QImage getImage();
    void setPenColor(QColor color);

```

```

    void setPenWidth(int width);
    void setShape(int type);
    QColor getPenColor() const;
    int getPenWidth() const;
    QImage image;
private:
    int penWidth;
    int drawtype;
    QColor penColor;
    QImage tmpImg;
    QPoint endPoint;
    QPoint startPoint;
    bool drawing;
protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);
};

#endif // PAINTER_H

```

main.cpp

```

#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}

```


mainwindow.cpp

```
#include <QtWidgets>
#include "mainwindow.h"
#include "painter.h"

const int canvas_width = 1100;
const int canvas_height = 600;
const int btn_size = 50;

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    setWindowTitle("逻辑电路设计");
    setWindowIcon(QIcon(":/window.png"));
    resize(1280, 720); //设置窗口大小为 1280*720

    //    //About Menu
    //    aboutMenu = menuBar()->addMenu(tr("关于"));

    /* 建立并链接菜单 */
    /* 建立菜单栏 */
    menubar = new QMenuBar(this); //建立一个以主窗口为 parent 的
menubar
    this->setMenuBar(menubar); //设置窗口的菜单栏为刚创建的 menubar
    QMenu *menu_file = menuBar()->addMenu("文件"); //在菜单栏上添加“文件”菜单
    /* 建立动作 */
    QAction *new_file = new QAction(QIcon(":/new_file.png"), "新建", this);
    QAction *open_file = new QAction(QIcon(":/open_file.png"), "打开", this);
    QAction *save_file = new QAction(QIcon(":/save.png"), "保存", this);
    QAction *save_as_file = new QAction(QIcon(":/save_as.png"), "另存为", this);
    /* 设置动作快捷键 */
    new_file->setShortcuts(QKeySequence::New);
    open_file->setShortcuts(QKeySequence::Open);
    save_file->setShortcuts(QKeySequence::Save);
    save_as_file->setShortcuts(QKeySequence::SaveAs); //另存为的快捷键 ctrl+shift+s 没有实装?
```

```

        /* 将动作添加到“文件”菜单中 */
menu_file->addAction(new_file);
menu_file->addAction(open_file);
menu_file->addAction(save_file);
menu_file->addAction(save_as_file);
        /* 将动作与槽链接 */
connect(new_file, SIGNAL(triggered()), this,
        SLOT(_NewFile()));
connect(open_file, SIGNAL(triggered()), this,
        SLOT(_OpenFile()));
connect(save_file, SIGNAL(triggered()), this,
        SLOT(_SaveFile()));
connect(save_as_file, SIGNAL(triggered()), this,
        SLOT(_SaveAsFile()));
        /* 设置状态栏提示信息 */
new_file->setStatusTip("新建电路原理图");
open_file->setStatusTip("打开电路原理图");
save_file->setStatusTip("保存电路原理图");
save_as_file->setStatusTip("另存为电路原理图");

QMenu *menu_about = menuBar()->addMenu("帮助");//在菜单栏上添加“关于”菜单
QAction *connect_us = new QAction("联系我们",this);
menu_about->addAction(connect_us);
connect(connect_us, SIGNAL(triggered()), this,
        SLOT(_ConnectUs()));

        /* 建立并链接状态栏 */
statusbar = new QStatusBar(this);//建立一个以主窗口为 parent
        的 statusbar
this->setStatusBar(statusbar);//设置窗口的状态栏为刚创建的
        statusbar

        /* 创建并配置画布停靠窗 */
        /* 创建并设置画布停靠窗 */
canvasDock = new QDockWidget("设计原理图", this);
canvasDock->setFeatures(QDockWidget::DockWidgetMovable |
        QDockWidget::DockWidgetFloatable);
canvasDock->setAllowedAreas(Qt::LeftDockWidgetArea |
        Qt::RightDockWidgetArea);
canvasDock->setMinimumSize(canvas_width, canvas_height);
setCentralWidget(canvasDock);

```

```

    /* 在停靠窗中创建画布 */
    painter = new PaintWidget(canvasDock); //画布的 parent 是停靠窗
    painter->setScaledContents(true); //允许自动缩放
    /* 初始化画布 */
    QImage img;
    painter->setImage(img);
    painter->setPenWidth(2);
    painter->setPenColor(Qt::darkBlue);
    painter->setPixmap(QPixmap::fromImage(img)); //??有用, 但还不知道具体什么用
    painter->setShape(0);
    painter->resize(canvas_width*2, canvas_height*2); //设置画布大小

    /* 为画布添加滚动条效果 */
    QScrollArea *scroller = new QScrollArea(this); //为什么不是以 painter 或 canvasDock 为 parent?
    scroller->setBackgroundRole(QPalette::Dark); //?
    scroller->setAlignment(Qt::AlignCenter); //设置中间对齐
    scroller->setWidget(painter); //设置滚动范围为哪个组件
    canvasDock->setWidget(scroller);

    /* 创建并设置 toolbox */
    /* 创建并设置 toolbox 停靠窗 */
    toolboxDock = new QDockWidget("元件库", this);
    addDockWidget(Qt::LeftDockWidgetArea, toolboxDock); //停靠窗配置在左侧
    splitDockWidget(toolboxDock, canvasDock, Qt::Horizontal); //?
    /* 创建按钮 */
    QPushButton *and2_btn = new QPushButton(QIcon(":/and2.png"), "", this); //为什么 parent 是 mainwindow?
    QPushButton *or2_btn = new QPushButton(QIcon(":/or2.png"), "", this);
    QPushButton *and3_btn = new QPushButton(QIcon(":/and3.png"), "", this);
    QPushButton *or3_btn = new QPushButton(QIcon(":/or3.png"), "", this);
    QPushButton *and4_btn = new QPushButton(QIcon(":/and4.png"), "", this);
    QPushButton *or4_btn = new QPushButton(QIcon(":/or4.png"), "", this);
    QPushButton *xor_btn = new QPushButton(QIcon(":/xor.png"), "", this);
    QPushButton *inv_btn = new QPushButton(QIcon(":/inv.png"), "", this);

```

```

    QPushButton *h_line_btn = new
QPushButton(QIcon(":/h_line.png"), "", this);
    QPushButton *v_line_btn = new
QPushButton(QIcon(":/v_line.png"), "", this);
    QPushButton *free_line_btn = new
QPushButton(QIcon(":/free_line.png"), "", this);
    QPushButton *dot_btn = new QPushButton(QIcon(":/dot.png"), "",
this);
    QPushButton *erase_btn = new
QPushButton(QIcon(":/eraser.png"), "", this);
    QPushButton *delete_btn = new
QPushButton(QIcon(":/delete.png"), "", this);
    and2_btn->setFixedSize(btn_size, btn_size);
    or2_btn->setFixedSize(btn_size, btn_size);
    and3_btn->setFixedSize(btn_size, btn_size);
    or3_btn->setFixedSize(btn_size, btn_size);
    and4_btn->setFixedSize(btn_size, btn_size);
    or4_btn->setFixedSize(btn_size, btn_size);
    xor_btn->setFixedSize(btn_size, btn_size);
    inv_btn->setFixedSize(btn_size, btn_size);
    h_line_btn->setFixedSize(btn_size, btn_size);
    v_line_btn->setFixedSize(btn_size, btn_size);
    free_line_btn->setFixedSize(btn_size, btn_size);
    dot_btn->setFixedSize(btn_size, btn_size);
    erase_btn->setFixedSize(btn_size, btn_size);
    delete_btn->setFixedSize(btn_size, btn_size);
    and2_btn->setToolTip("二输入与门");
    or2_btn->setToolTip("二输入或门");
    and3_btn->setToolTip("三输入与门");
    or3_btn->setToolTip("三输入或门");
    and4_btn->setToolTip("四输入与门");
    or4_btn->setToolTip("四输入或门");
    xor_btn->setToolTip("异或门");
    inv_btn->setToolTip("非门");
    h_line_btn->setToolTip("水平连接线");
    v_line_btn->setToolTip("垂直连接线");
    free_line_btn->setToolTip("自由连接线");
    dot_btn->setToolTip("电路连接点");
    erase_btn->setToolTip("区域清除");
    and2_btn->setStatusTip("绘制二输入与门");
    or2_btn->setStatusTip("绘制二输入或门");
    and3_btn->setStatusTip("绘制三输入与门");
    or3_btn->setStatusTip("绘制三输入或门");
    and4_btn->setStatusTip("绘制四输入与门");

```

```

or4_btn->setStatusTip("绘制四输入或门");
xor_btn->setStatusTip("绘制异或门");
inv_btn->setStatusTip("绘制非门");
h_line_btn->setStatusTip("绘制水平连接线");
v_line_btn->setStatusTip("绘制垂直连接线");
free_line_btn->setStatusTip("绘制自由连接线");
dot_btn->setStatusTip("绘制电路连接点");
erase_btn->setStatusTip("清除指定区域");
delete_btn->setStatusTip("清除整张画布");
//and2_btn->setObjectName("custombutton");???
/* 将按钮排列在停靠窗中 */
QGridLayout *toolbox_layout = new QGridLayout();
toolbox_layout->setAlignment(Qt::AlignTop);
toolbox_layout->addWidget(and2_btn, 0, 0);
toolbox_layout->addWidget(or2_btn, 0, 1);
toolbox_layout->addWidget(and3_btn, 1, 0);
toolbox_layout->addWidget(or3_btn, 1, 1);
toolbox_layout->addWidget(and4_btn, 2, 0);
toolbox_layout->addWidget(or4_btn, 2, 1);
toolbox_layout->addWidget(xor_btn, 3, 0);
toolbox_layout->addWidget(inv_btn, 3, 1);
toolbox_layout->addWidget(h_line_btn, 4, 0);
toolbox_layout->addWidget(v_line_btn, 4, 1);
toolbox_layout->addWidget(free_line_btn, 5, 0);
toolbox_layout->addWidget(dot_btn, 5, 1);
toolbox_layout->addWidget(erase_btn, 6, 0);
toolbox_layout->addWidget(delete_btn, 6, 1);
QWidget *toolWidget = new QWidget(toolboxDock);
toolWidget->setLayout(toolbox_layout);
toolboxDock->setWidget(toolWidget);
/* 链接按钮 */
designLib = new QButtonGroup();
designLib->addButton(and2_btn, 1);
designLib->addButton(or2_btn, 2);
designLib->addButton(and3_btn, 3);
designLib->addButton(or3_btn, 4);
designLib->addButton(and4_btn, 5);
designLib->addButton(or4_btn, 6);
designLib->addButton(xor_btn, 7);
designLib->addButton(inv_btn, 8);
designLib->addButton(h_line_btn, 9);
designLib->addButton(v_line_btn, 10);
designLib->addButton(free_line_btn, 11);
designLib->addButton(dot_btn, 12);

```

```

designLib->addButton(erase_btn, 13);
designLib->addButton(delete_btn, 14);
connect(designLib, SIGNAL(buttonClicked(int)), this,
SLOT(designLibSelect(int)));

/* 创建并配置参数停靠窗 */
penArgDock = new QToolBar();
penArgDock = addToolBar("笔尖参数设置");//???
penArgDock->setMovable(true);
QPushButton *add_width_btn = new
QPushButton(QIcon(":/add_width.png"), "", this);
QPushButton *minus_width_btn = new
QPushButton(QIcon(":/minus_width.png"), "", this);
QPushButton *red_btn = new QPushButton(QIcon(":/red.png"), "",
this);
QPushButton *blue_btn = new QPushButton(QIcon(":/blue.png"),
"", this);
QPushButton *black_btn = new
QPushButton(QIcon(":/black.png"), "", this);

add_width_btn->resize(btn_size, btn_size);
minus_width_btn->resize(btn_size, btn_size);
red_btn->resize(btn_size, btn_size);
blue_btn->resize(btn_size, btn_size);
black_btn->resize(btn_size, btn_size);
connect(add_width_btn, SIGNAL(clicked()), this,
SLOT(add_pen_width()));
connect(minus_width_btn, SIGNAL(clicked()), this,
SLOT(minus_pen_width()));
connect(red_btn, SIGNAL(clicked()), this, SLOT(set_red()));
connect(blue_btn, SIGNAL(clicked()), this,
SLOT(set_blue()));
connect(black_btn, SIGNAL(clicked()), this,
SLOT(set_black()));
QHBoxLayout *penArgLayout = new QHBoxLayout();
penArgLayout->setAlignment(Qt::AlignLeft);
penArgLayout->addWidget(add_width_btn);
penArgLayout->addWidget(minus_width_btn);
penArgLayout->addWidget(red_btn);
penArgLayout->addWidget(blue_btn);
penArgLayout->addWidget(black_btn);
QWidget *penArgWidget = new QWidget();
penArgWidget->setLayout(penArgLayout);
penArgDock->addWidget(penArgWidget);

```

```

        /*
        QImage img(nullptr);
        painter->setImage(img);
        painter->resize(1000, 1000);
        painter->setShape(PaintWidget::Pen);
        painter->setPenWidth(2);
        */
    }

MainWindow::~MainWindow()
{

}

```

painter.cpp

```

#include <QtGui>
#include <QtWidgets>
#include <QMainWindow>
#include "painter.h"
#include "mainwindow.h"

#define ITEM_WIDTH 100
#define ITEM_HEIGHT 50

PaintWidget::PaintWidget(QWidget *parent) : QLabel (parent)
{
    drawing = false;
}

void PaintWidget::setImage(QImage img) //设置画布
{
    QImage img1 = QImage(canvas_width*2, canvas_height*2,
QImage::Format_ARGB32); //ARGB 支持透明像素
    img1.fill(qRgb(255, 255, 255)); //白色
    if (img.isNull())
        img = img1;
    image = img; //真实的画布
    tmpImg = img; //临时画布，用于实现绘制（鼠标拖动）时的效果预览
    paint(img);
}

QImage PaintWidget::getImage() //获得画布
{

```



```

    QImage non_image;
    if (!image.isNull())
        return image;
    return non_image;
}

void PaintWidget::setPenColor(QColor color) //设置笔的颜色
{
    penColor = color;
}

QColor PaintWidget::getPenColor() const //获得笔的颜色, const
    保证数据不被修改
{
    return penColor;
}

void PaintWidget::setPenWidth(int width) //设置笔的宽度
{
    penWidth = width;
}

int PaintWidget::getPenWidth() const //获得笔的宽度
{
    return penWidth;
}

void PaintWidget::setShape(int type) //获得绘制图片的类型
{
    drawtype = type;
}

void PaintWidget::paintEvent(QPaintEvent *) //画笔的绘画事件
{
    //QDebug() << "paintEvent" << endl;
    QPainter painter(this);
    if (drawing)
        painter.drawImage(0, 0, tmpImg); //先画在临时画布上
    else
        painter.drawImage(0, 0, image); //鼠标 release 时, 将临
    时画布内容画到真实画布上
}

void PaintWidget::mousePressEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton){
        startPoint = event->pos(); //鼠标左键
        按下时, 获得绘图起始点数据
        drawing = true; //在临时画布
        上绘图
    }
}

```



```

}
void PaintWidget::mouseMoveEvent(QMouseEvent *event)
{
    //QDebug() << startPoint << endPoint << endl;
    if (event->buttons() & Qt::LeftButton){
        endPoint = event->pos(); //鼠标拖动
        时, 当前点为绘图终点数据
        tmpImg = image;
        paint(tmpImg);
    }
}
void PaintWidget::mouseReleaseEvent(QMouseEvent *)
{
    drawing = false;
    paint(image); //鼠标
    release 时, 在真实画布上绘制
}
void PaintWidget::paint(QImage &CurrImg)
{
    QPainter qpainter(&CurrImg); //现在临时
    画布上绘图
    QPainter eraser(&image); //擦除时无
    需预览, 直接在真实画布上进行
    //qpainter 对象初始化
    QPen pen;
    pen.setWidth(penWidth);
    pen.setColor(penColor);
    qpainter.setPen(pen);
    qpainter.setRenderHint(QPainter::Antialiasing, true); //抗锯齿
    齿

    switch (drawtype) {
    case 1:
        //draw and2 调用 drawPixmap 函数, 使用图片实现绘图, 下同
        qpainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
        ,ITEM_HEIGHT,QPixmap(":/and2.png"));
        break;
    case 2:
        //draw or2

        qpainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
        ,ITEM_HEIGHT,QPixmap(":/or2.png"));
    }
}

```

```

        break;
    case 3:
        //draw and3

    QPainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
    ,ITEM_HEIGHT,QPixmap(":/and3.png"));
        break;
    case 4:
        //draw or3

    QPainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
    ,ITEM_HEIGHT,QPixmap(":/or3.png"));
        break;
    case 5:
        //draw and4

    QPainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
    ,ITEM_HEIGHT,QPixmap(":/and4.png"));
        break;
    case 6:
        //draw or4

    QPainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
    ,ITEM_HEIGHT,QPixmap(":/or4.png"));
        break;
    case 7:
        //draw xor

    QPainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
    ,ITEM_HEIGHT,QPixmap(":/xor.png"));
        break;
    case 8:
        //draw inv

    QPainter.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH
    ,ITEM_HEIGHT,QPixmap(":/inv.png"));
        break;
    case 9:
        //draw h_line

    QPainter.drawLine(startPoint.x(),startPoint.y(),endPoint.x(),s
    tartPoint.y());    //y 坐标不变，绘制水平直线
        break;
    case 10:

```

```

        //draw v_line

qpainter.drawLine(startPoint.x(),startPoint.y(),startPoint.x()
,endPoint.y());    //x 坐标不变，绘制垂直直线
        break;
        case 11:
            //draw free_line
            QPainter.drawLine(startPoint, endPoint);    //调用
QPainter 的 drawLine 函数绘制任意倾斜的直线
            break;
        case 12:
            //draw dot

qpainter.drawPixmap(endPoint.x()-49,endPoint.y()-22,ITEM_WIDTH
,ITEM_HEIGHT,QPixmap(":/node.png"));
            break;
        case 13:
            //erase

eraser.drawPixmap(endPoint.x()-30,endPoint.y()-15,ITEM_WIDTH,I
TEM_WIDTH,QPixmap(":/blank.png"));
            break;
        case 14:
            //clear all
            QPainter.drawPixmap(0,0,canvas_width*2,
canvas_height*2,QPixmap(":/blank_all.png"));
            break;
    }
    update();
}

```

slot.cpp

```

#include "mainwindow.h"

void MainWindow::_NewFile()
{
    QImage img = QImage(canvas_width*2, canvas_height*2,
QImage::Format_RGB32);//新建一个图片当画布
}

```

```

        img.fill(qRgb(255, 255, 255)); //这个用来当画布的图片填充为白色
        path_of_curr_file = ""; //不设置当前路径
        painter->setImage(img);
    }
    void MainWindow::_OpenFile()
    {
        /* 选择一个图片路径，打开这个图片，以这时候的路径为当前路径。显示
        这个图片。 */
        QString path = QFileDialog::getOpenFileName(this, "选择电路原
        理图", ".", "schematic(*.jpg *.png *.bmp)");
        if (!path.isEmpty()){
            QImage *open_file = new QImage();
            if (!open_file->load(path)){
                QMessageBox::information(this, "错误", "打开文件失败!
                ");
                delete open_file;
            }
            else{
                path_of_curr_file = path;
                painter->setImage(*open_file);
            }
        }
    }
    void MainWindow::_SaveFile()
    {
        /* 如果之前没有经过 open 操作，当前文件路径是空的，需要选择一个路
        径保存。保存的核心操作是自带的.save() */
        if (path_of_curr_file.isEmpty()){
            QString path = QFileDialog::getSaveFileName(this, "保存电
            路原理图", ".", "schematic(*.jpg *.png *.bmp)");
            if (!path.isEmpty())
                path_of_curr_file = path;
        }
        QImage save_file = painter->getImage();
        save_file.save(path_of_curr_file);
    }
    void MainWindow::_SaveAsFile()
    {
        QString path = QFileDialog::getSaveFileName(this, "另存为电路
        原理图", ".", "schematic(*.jpg *.png *.bmp)");
        if (!path.isEmpty())
            path_of_curr_file = path;
        QImage save_file = painter->getImage();
        save_file.save(path_of_curr_file);
    }

```

```
}  
void MainWindow::_ConnectUs()  
{  
    QMessageBox::information(this, "Email:",  
        "XXXXXXXX@XXXX.XXX");  
}  
  
void MainWindow::designLibSelect(int id)  
{  
    switch (id) {  
        case 1:  
            painter->setShape(1);  
            break;  
        case 2:  
            painter->setShape(2);  
            break;  
        case 3:  
            painter->setShape(3);  
            break;  
        case 4:  
            painter->setShape(4);  
            break;  
        case 5:  
            painter->setShape(5);  
            break;  
        case 6:  
            painter->setShape(6);  
            break;  
        case 7:  
            painter->setShape(7);  
            break;  
        case 8:  
            painter->setShape(8);  
            break;  
        case 9:  
            painter->setShape(9);  
            break;  
        case 10:  
            painter->setShape(10);  
            break;  
        case 11:  
            painter->setShape(11);  
            break;  
        case 12:
```

```

        painter->setShape(12);
        break;
    case 13:
        painter->setShape(13);
        break;
    case 14:
        painter->setShape(14);
        break;
    }
}

void MainWindow::add_pen_width()
{
    qDebug() << "add_pen_width, penWidth: " <<
    painter->getPenWidth() << endl;
    if (painter->getPenWidth() < 10)
        painter->setPenWidth(painter->getPenWidth()+1);
}

void MainWindow::minus_pen_width()
{
    qDebug() << "minus_pen_width, penWidth: " <<
    painter->getPenWidth() << endl;
    if (painter->getPenWidth() > 1)
        painter->setPenWidth(painter->getPenWidth()-1);
}

void MainWindow::set_red()
{
    painter->setPenColor(Qt::red);
}

void MainWindow::set_blue()
{
    painter->setPenColor(Qt::darkBlue);
}

void MainWindow::set_black()
{
    painter->setPenColor(Qt::black);
}

```