

[**OOP**](javascript:;)([**object-oriented**](javascript:;) [**programming**](javascript:;))

# Project Final Report



1. **Name ： 邱明冉 Student Number： 3190104698**
2. **Name ： 孟俊邑 Student Number： 3190106104**
3. **Name ： 郭胜贤 Student Number： 3190106080**
4. **Name ： 罗志凡 Student Number： 3190103256**

**2019~2020春夏学期 2021 年 1 月 10 日**

# 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，可以选择放大或缩小所选元件，也可以对其颜色进行修改。

# 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元件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。

具体代码如下：

|  |
| --- |
| #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***()  {  } |

3.1.4 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。

|  |
| --- |
| #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)  {  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));  if (img.isNull())  img = img1;  image = img;  tmpImg = img;  paint(*img*);  }  QImage PaintWidget::**getImage**()  {  if (!image.isNull())  return image;  }  void PaintWidget::**setPenColor**(QColor color)  {  penColor = color;  }  QColor PaintWidget::**getPenColor**() const  {  return penColor;  }  void PaintWidget::**setPenWidth**(int width)  {  penWidth = width;  }  int PaintWidget::**getPenWidth**() const  {  return penWidth;  }  void PaintWidget::**setShape**(int type)  {  drawtype = type;  }  void PaintWidget::***paintEvent***(QPaintEvent \*)  {  //qDebug() << "paintEvent" << endl;  QPainter painter(this);  if (drawing)  painter.drawImage(0, 0, tmpImg);  else  painter.drawImage(0, 0, image);  }  void PaintWidget::***mousePressEvent***(QMouseEvent \*event)  {  if (event->button() == Qt::LeftButton){  startPoint = event->pos();  drawing = true;  }  }  void PaintWidget::***mouseMoveEvent***(QMouseEvent \*event)  {  //qDebug() << startPoint << endPoint << endl;  if (event->buttons() & Qt::LeftButton){  endPoint = event->pos();  tmpImg = image;  paint(*tmpImg*);  }  }  void PaintWidget::***mouseReleaseEvent***(QMouseEvent \*)  {  drawing = false;  paint(*image*);  }  void PaintWidget::**paint**(QImage &CurrImg)  {  QPainter qpainter(&CurrImg);  QPainter eraser(&image);  QPen pen;  pen.setWidth(penWidth);  pen.setColor(penColor);  qpainter.setPen(pen);  qpainter.setRenderHint(QPainter::Antialiasing, true); //抗锯齿  switch (drawtype) {  case 1:  //draw and2  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(),startPoint.y());  break;  case 10:  //draw v\_line  qpainter.drawLine(startPoint.x(),startPoint.y(),startPoint.x(),endPoint.y());  break;  case 11:  //draw free\_line  qpainter.drawLine(startPoint, endPoint);  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,ITEM\_WIDTH,QPixmap(":/blank.png"));  break;  case 14:  //clear all  qpainter.drawPixmap(0,0,canvas\_width\*2, canvas\_height\*2,QPixmap(":/blank\_all.png"));  break;  }  qpainter.save();  update();  } |

3.1.5 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：

|  |
| --- |
| #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", ".", "schemetic(\*.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", ".", "schemetic(\*.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", ".", "schemetic(\*.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);  } |

## 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;

//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**();

};

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QLable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 ", ".", "schemetic(\*.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", ".", "schemetic(\*.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", ".", "schemetic(\*.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imag1，以获取imag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

* 1. 编译安装运行说明

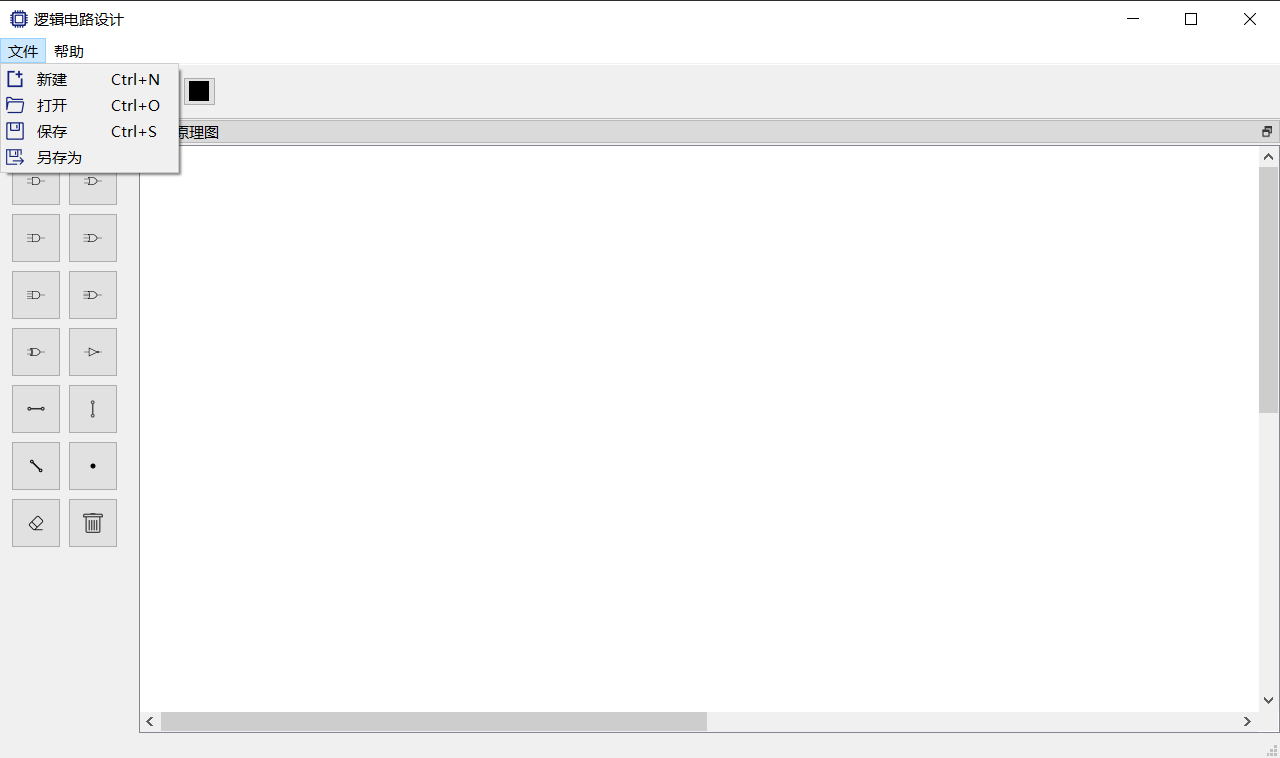
（此部分介绍如何由提交的源代码包，进行存放、编译生成.exe文件的过程说明，以及运行.exe后的用户使用手册。）

* 1. 典型测试情况

软件初始化界面如下：

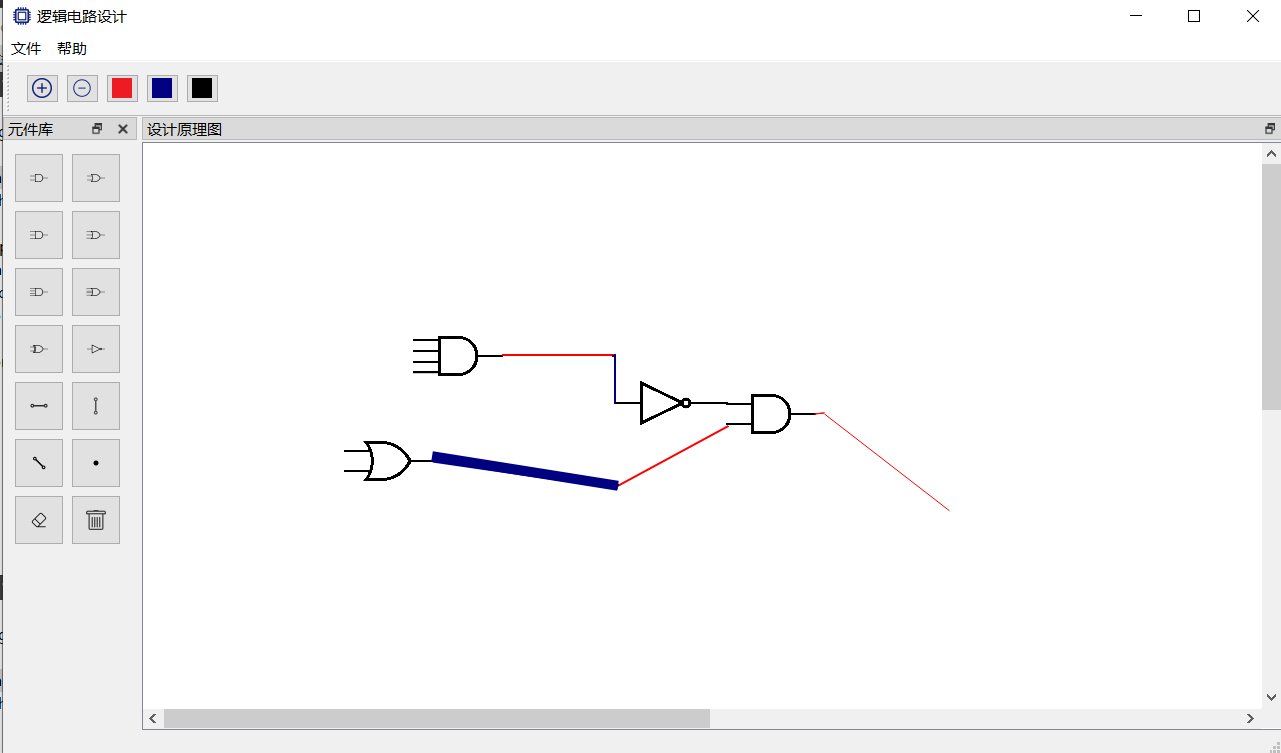


鼠标移动到“文件”处，可以对图片文件进行操作，包括新建、打开、保存和另存为。可以鼠标左键单击实现功能，也可以通过快捷键实现功能。



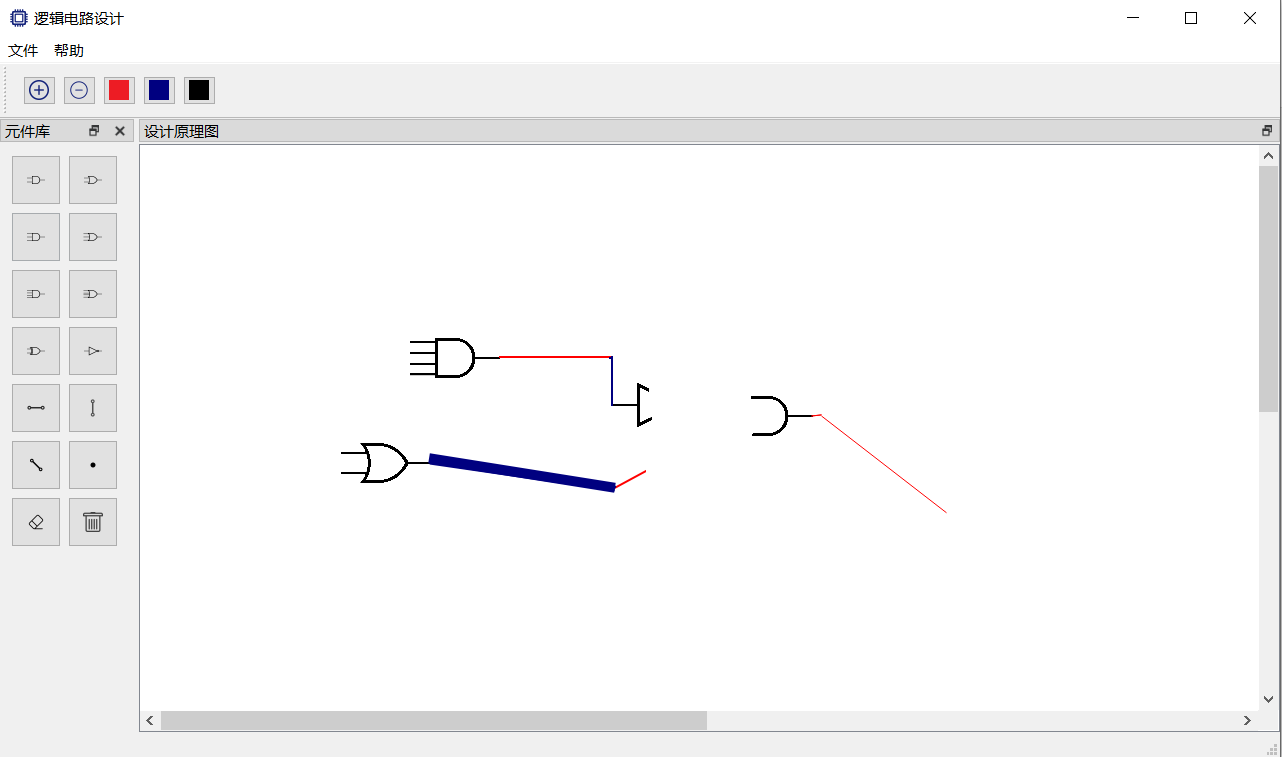
若直接在空白画布上作原理图，效果如下：

（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：

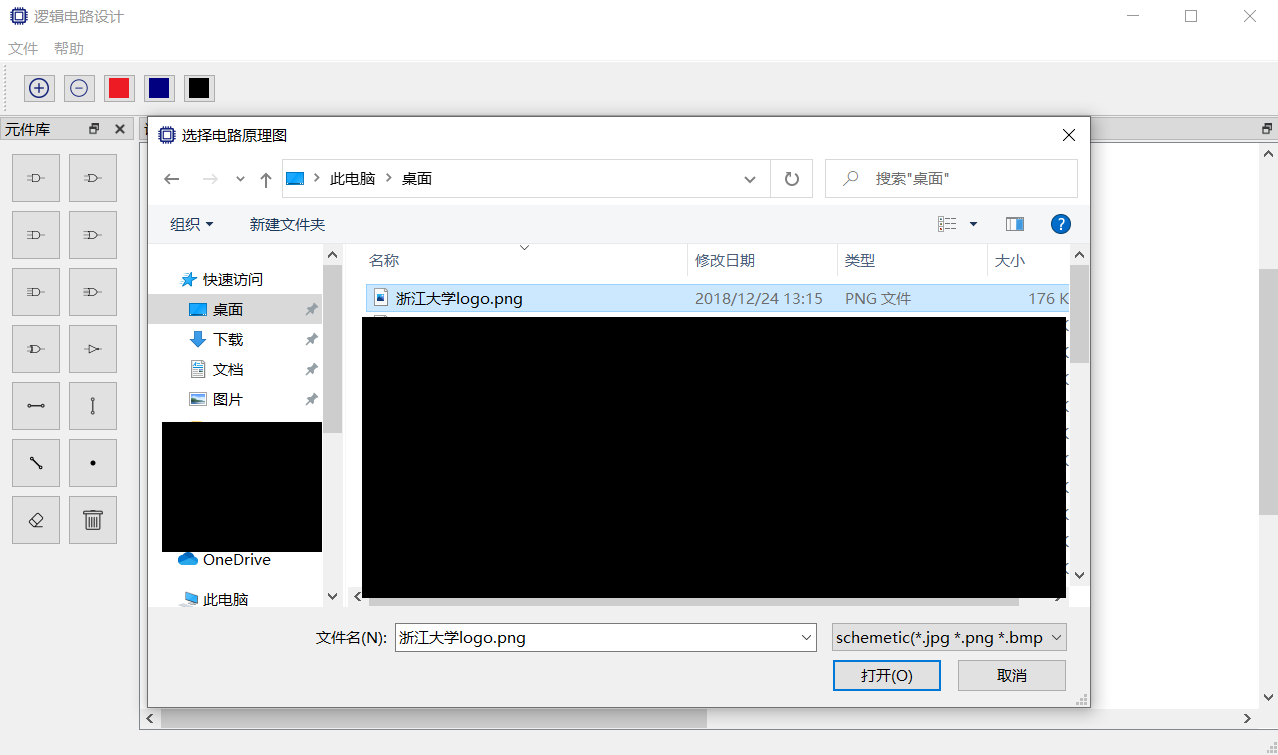
（可以擦除一部分原理图）



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crtl-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。

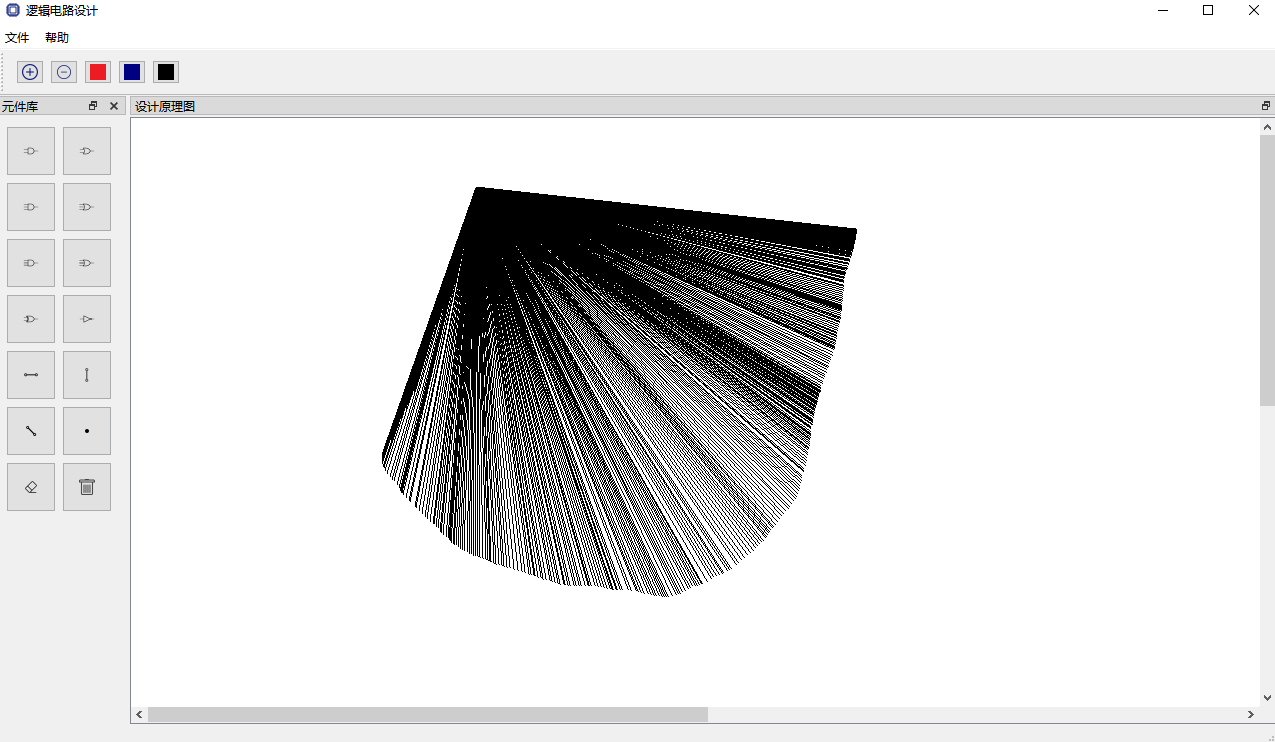
郭胜贤：最终报告撰写，Project 总体测试。

罗志凡：代码实现与调试，Project 总体测试。

5.2个人实践过程中遇到的难点及解决方案

1. 直线绘制功能的实现
   1. 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：



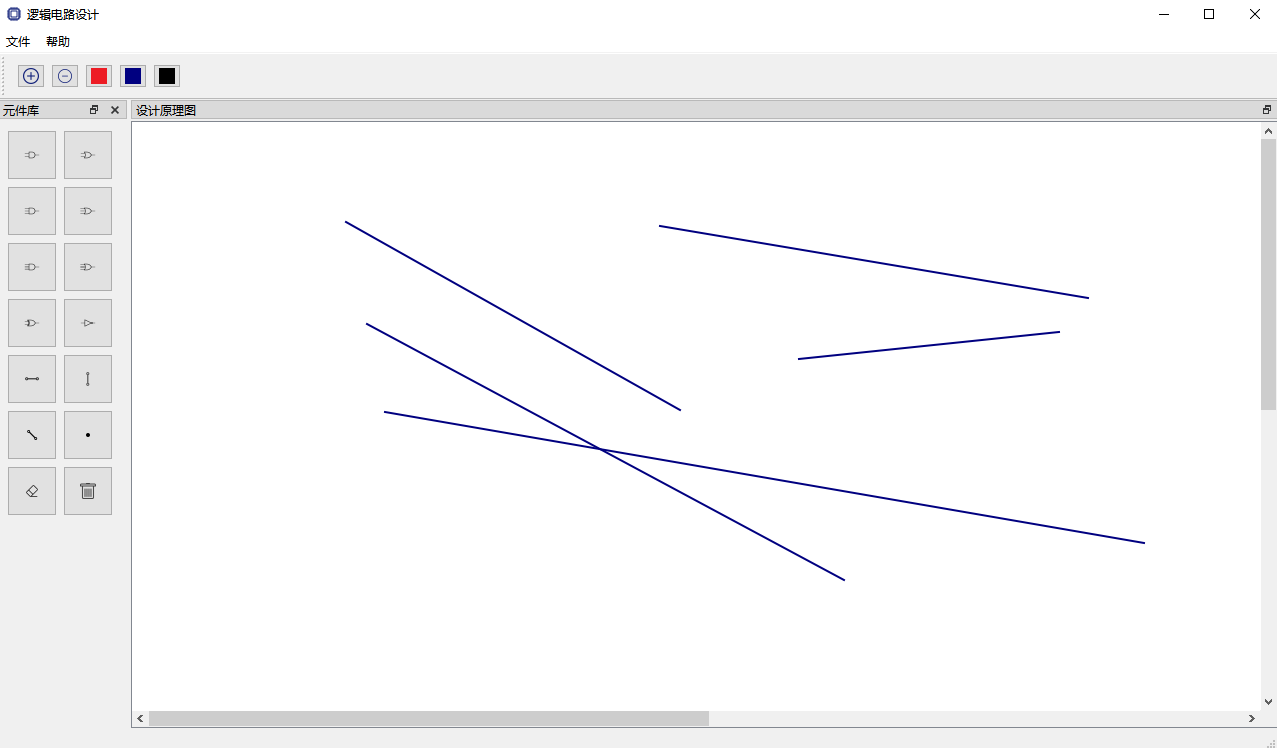
经分析后，发现是由于直接画在真实画布上，而每次移动鼠标都进行了一次绘制，导致了问题的发生。

* 1. 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  } |

* 1. 问题解决



1. 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。

# 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

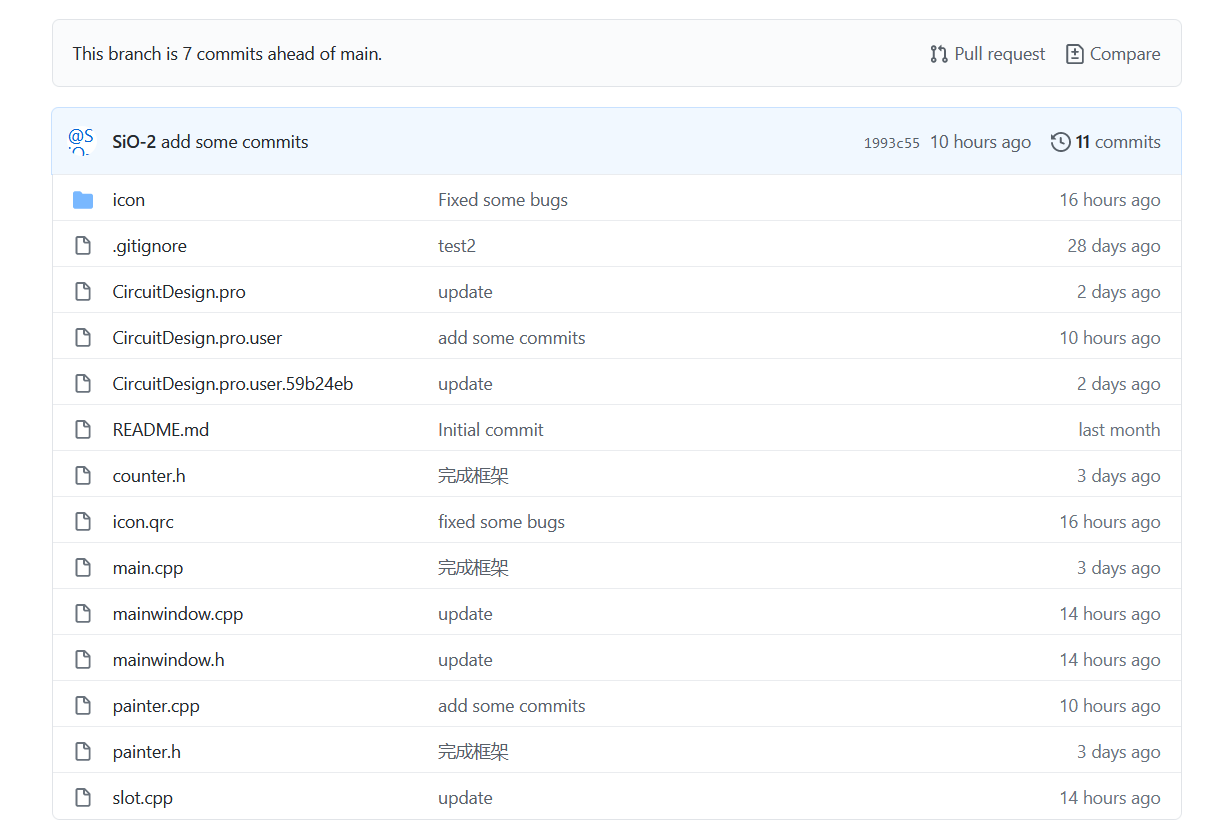
2020年 12月 25日：完善所有基本功能，优化界面显示

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：





# 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QWidgets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，对于错误定位大有助益。

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，他教我们搭载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。总体来说过程不算很艰难，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，所以最终的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：

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，其功能的强大带深深地震撼了我。在学习的过程中，也掌握了使用QT库函数的技巧，按F1就能查阅手册，非常方便，日后有机会一定会再用QT做些东西。另外，由于我没有上过C大程，所以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，我们小组在分工阶段出现了磨合不足的问题。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，不自觉地就把程序写成了过程，而不是采用面向对象的思想，写得差不多时才意识到这个问题，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